ЕСЛИ ВЫ ЖЕЛАЕТЕ РАЗОБРАТЬСЯ В ОСОБЕННОСТЯХ LINUX И ПОЗНАТЬ ЕЕ ВНУТРЕННИЙ МИР, ЭТА КНИГА — ВАШ ЛУЧШИЙ ВЫБОР!

Д. Н. КОЛИСНИЧЕНКО
ПИТЕР В. АЛЛЕН

LINUX

Полное руководство

- Linux для пользователя
- Сетевые технологии Linux или Linux для администратора
- Программирование Linux

ПОЛНОЕ РУКОВОДСТВО

Д.Н. Колисниченко, Питер В. Аллен

Linux

ПОЛНОЕ РУКОВОДСТВО

Наука и Техника, Санкт-Петербург
2006

Колисниченко Д.Н., Аллен Питер В.

LINUX: полное руководство. — СПб: Наука и Техника, 2006. — 784 с.: ил.

Под редакцией М.В. Финкова

ISBN 5-94387-139-X

Серия «Полное руководство»

Данная книга представляет собой великолепное руководство по Linux, позволяющее получить наиболее полное представление об этой операционной системе. Книга состоит из трех частей, каждая из которых раскрывает один из трех основных аспектов работы с Linux: Linux для пользователя, сетевые технологии Linux (и методика настройки Linux-сервера), программирование Linux. В книге охвачен очень широкий круг вопросов, начиная с установки и использования Linux «в обычной жизни» (офисные пакеты, игры, видео, Интернет), и заканчивая описанием внутренних процессов Linux, секретами и трюками настройки, особенностями программирования под Linux, созданием сетевых приложений, оптимизацией ядра и др.

Изложение материала ведется в основном на базе дистрибутивов Fedora Core (Red Hat) и Mandriva (Mandrake). Однако не оставлены без внимания и другие дистрибутивы SuSe, Slackware, Gentoo, Alt Linux, Knoppix. Дается их сравнительное описание, а по ходу изложения всего материала указываются их особенности.

Книга написана известными специалистами и консультантами по использованию Linux, авторами многих статей и книг по Linux, заслуживших свое признание в самых широких Linux-кругах. Если вы желаете разобраться в особенностях Linux и познать ее внутренний мир, эта книга — ваш лучший выбор.

9 785943 871399

ISBN 5-94387-139-X

Контактные телефоны издательства
(812) 567-70-25, 567-70-26
(044) 516-38-66

Официальный сайт www.nit.com.ru

© Колисниченко Д.Н., Аллен Питер В.
© Наука и Техника (оригинал-макет), 2006

ООО «Наука и Техника».
Лицензия №000350 от 23 декабря 1999 года.
198097, г. Санкт-Петербург, ул. Маршала Говорова, д. 29.
Подписано в печать 10.10.05. Формат 70×100 1/16.
Бумага газетная. Печать офсетная. Объем 49 п. л.
Тираж 5000 экз. Заказ № 388 .

Отпечатано с готовых диапозитивов в ОАО «Техническая книга»
190005, Санкт-Петербург, Измайловский пр., 29

Содержание

Введение 15

Об этой книге 15
Почему Linux? 16
История Linux 17
Все началось с игры 17
Становление UNIX как операционной системы 20
Бурное развитие UNIX 22
Какие бывают UNIX'ы? 23
Происхождение Linux 24
Операционная система Minix 24
Линус Торвальдс 24
Линус недоволен Minix 26
При чем здесь пингвины? 28
Путь к первой версии (1.0) 30
Они создавали Linux 31
Первые дистрибутивы 31
«Размножение дистрибутивов» 32
Выбор дистрибутива 36
Fedora Core 3 40
Slackware 47
Linux Mandrake 10.1 51
Разное 54
ASP Linux 54
ALT Linux 55
SuSE Linux 57
Knoppix 57
Debian 58

Глава 1. Установка операционной системы 59

1.1. Подготовка жесткого диска 60
1.1.1. Рекомендуемая схема разбиения диска 61
1.1.2. Имена разделов в ОС Linux 62
1.1.3. Разделы и точки монтирования 62
1.2. Загрузка программы установки 63
1.2.1. С использованием загрузочного компакт-диска 63
1.2.2. С использованием загрузочной дискеты 63
1.2.3. С использованием жесткого диска 64
1.2.4. Установка по сети 64
1.3. Установка Fedora Core 65
1.3.1. Описание дистрибутива 65
Версии Fedora Core — 1, 2, 3 и 4. Системные требования 66
Специальная технология разграничения доступа SELinux 68
1.3.2. Особенности установки различных версий Fedora Core 68
Fedora Core 2: подготовка к совместной жизни с Windows 68
Особенности и преимущества процесса установки Fedora Core 4 70
1.3.3. Установка загрузчика 70
1.3.4. Продолжение установки 71
1.4. Особенности установки Mandrake 10.0 77
1.5. Проблемы при установке 82
1.5.1. Конфликты Fedora Core 1 и 2 с различным оборудованием 82
Материнские платы ASUS 82
LCD-мониторы 82
Ноутбуки SONY 82
Не работает мышь 82

1.5.2. Fedora Core: не удается войти в систему как root в графическом режиме ... 83
1.5.3. Ошибка Signal 11 ... 83
1.5.4. Не определяется мышь ... 83
1.5.5. Проблемы с переходом в графический режим ... 83
1.5.6. Проблемы при загрузке ... 83
1.5.7. Проблемы с графическим экраном загрузчика ... 84
1.5.8. Не загружается система X Window ... 84
1.5.9. Я забыл пароль пользователя root ... 84
1.5.10. У меня больше оперативной памяти! ... 84
1.5.11. Не работают принтер или звуковая плата ... 84
1.6. Как удалить Linux ... 84
1.7. Системы с двойной загрузкой ... 85
1.7.1. Установка Windows 9x и Linux ... 85
1.7.2. Установка Windows 9x, Windows NT/2000/XP и Linux ... 86
Первый способ ... 86
Второй способ ... 87
Использование loadlin ... 87
1.8. Первый запуск Linux ... 89
1.9. Стандартные сервисы Linux ... 92
1.10. Справочная система ... 94

Глава 2. Файловая система Linux ... 96

2.1. Видимая сторона файловой системы ... 97
2.1.1. Имена файлов и каталогов ... 97
2.1.2. Назначение основных системных каталогов ... 99
2.1.3. Типы файлов ... 100
Обычные файлы и каталоги ... 100
Файлы физических устройств ... 101
Жесткие и символические ссылки ... 102
Именованные каналы ... 104
Гнезда ... 104
2.1.4. Команды работы с файлами и каталогами ... 105
Текущий каталог ... 105
Просмотр содержимого каталога ... 105
Создание и удаление файла ... 106
Копирование и перемещение файла ... 107
Просмотр текстовых файлов ... 108
Редактирование текстовых файлов ... 109
Поиск файлов ... 111
Изменение прав доступа к файлу ... 112
2.2. Изнанка файловой системы ... 115
2.2.1. Файловая система ext2fs — предшественница ext3fs ... 116
2.2.2. Журналируемые файловые системы ... 119
ReiserFS ... 121
XFS ... 121
JFS ... 122
Ext3fs ... 122
2.3. Создание и монтирование файловых систем ... 123
Создание файловой системы «вручную». Команда mkfs ... 123
Настройка автоматического монтирования при загрузке компьютера. Команда mount ... 125

Глава 3. Работаем в командной строке ... 127

3.1. Как устроен Linux: ядро и процессы ... 128
3.2. Жизнь процесса ... 130
Таблица процессов ... 130
Системные вызовы fork() и exec() или как размножаются процессы ... 131
Снимок протекающих в системе процессов — команда ps ... 132

Динамика процессов — команда top . . . 133
Категории процессов . . . 133
3.3. Взаимодействие процессов . . . 134
3.3.1. Конвейер (pipe) . . . 134
3.3.2. Сигналы . . . 135
3.4. Командная оболочка. Bash . . . 136
3.4.1. Встроенные команды . . . 138
3.4.2. История команд . . . 138
3.4.3. Переменные . . . 140
Описание и использование переменных . . . 140
Переменные окружения . . . 141
Быстрая смена каталога . . . 142
Настройка командной строки. Утилита tput . . . 142
3.4.4. Подстановка переменных и команд . . . 144
3.4.5. Шаблоны имен файлов . . . 144
3.4.6. Потоки ввода-вывода . . . 145
3.4.7. Группировка команд . . . 147
3.4.8. Инициализационные файлы bash . . . 149

Глава 4. Рабочее место пользователя . . . 150

4.1. Графическая система X Window . . . 152
4.2. Оконная среда KDE . . . 154
4.2.1. Рабочий стол KDE . . . 154
4.2.2. Запуск программ и переключение между ними . . . 155
4.2.3. Файловый менеджер Konqueror . . . 156
Основные возможности . . . 156
Konqueror — не только файловый менеджер . . . 156
4.2.4. Центр управления KDE . . . 157
4.2.5. Работа со сьемными носителями в KDE . . . 158
4.2.6. Добавление собственных команд в контекстное меню KDE . . . 159
4.2.7. Новое в KDE 3.4 . . . 161
4.3. Оконная среда GNOME . . . 162
4.3.1. Общее описание и методика работы . . . 162
Что за зверь эта GNOME . . . 162
Файловый менеджер Nautilus . . . 164
Расширенная система управления MIME-типами . . . 165
Поддержка службы DNS-Based Service Discovery . . . 166
Особенности последних версий . . . 166
4.3.2. Добавление собственных команд в контекстное меню GNOME . . . 169
4.3.3. Автоматическая смена обоев . . . 169
4.4. Офисные пакеты . . . 170
Open Office и K Office . . . 170
Уменьшение времени запуска Open Office . . . 172
Новое в Open Office 2.0 . . . 172
4.5. Издательские системы . . . 173
Язык разметки TeX . . . 173
Текстовый процессор LyX . . . 175
Издательская система Scribus . . . 176
4.6. Графика в Linux . . . 177
4.6.1. Графические редакторы . . . 177
Редактор растровой графики GIMP — обрабатываем фото . . . 177
KPaint . . . 178
KIconEdit — редактор иконок . . . 178
4.6.2. Программы — просмотрщики изображений . . . 178
GQView . . . 178
Electric Eyes . . . 179
KuickShow . . . 179
KView . . . 180
Kooka . . . 180
XSane . . . 180
4.6.3. Как сделать снимок экрана . . . 180

KSnapShot . . . 180
ImageMagick . . . 181
4.7. Полезные трюки . . . 182
Просмотр в консоли документов MS Word и PDF . . . 182

Глава 5. Звук и видео в Linux . . . 184

5.1. Почему воспроизведение аудио в Linux лучше, чем в Windows . . . 185
5.2. Прослушивание музыки . . . 186
mpg123 . . . 186
Xmms . . . 187
Другие программы . . . 187
Сравнение Xmms и NoAtun . . . 189
5.3. «Ограбление» Audio-CD . . . 192
5.4. Программы для просмотра видео . . . 194
Обзор программ . . . 194
Программа Xine . . . 194
Просмотр DVD . . . 196
Смотрим телепередачи под Linux . . . 197
5.5. Воспроизведение неподдерживаемых форматов . . . 198

Глава 6. Сеть и Интернет . . . 200

6.1. Основные сетевые понятия . . . 201
6.1.1. Протокол и интерфейс . . . 201
6.1.2. Уровни взаимодействия OSI . . . 204
Физический уровень (Physical Layer) . . . 204
Канальный уровень (Data link Layer) . . . 204
Сетевой уровень (Network Layer) . . . 205
Транспортный уровень (Transport Layer) . . . 205
Сеансовый уровень (Session Layer) . . . 206
Представительный уровень (Presentation Layer) . . . 206
Прикладной уровень (Application Layer) . . . 206
Интернет и модель OSI . . . 207
Основные протоколы . . . 207
6.1.3. Протокол TCP/IP и IP-адресация . . . 208
6.1.4. DNS — система доменных имен . . . 209
6.1.5. Порты . . . 211
6.1.6. Динамическое выделение адреса . . . 211
6.2. Подключение к локальной сети . . . 211
6.2.1. Настройка сети в Linux Mandrake . . . 212
6.2.2. Настройка сети в Linux Red Hat . . . 214
6.2.3. Настройка сети в Fedora Core . . . 215
6.2.4. Проверка работы сетевого интерфейса . . . 215
6.2.5. Настройка сети в старых дистрибутивах . . . 218
6.3. Подключение к Windows-сети . . . 218
6.4. Подключение к Интернету . . . 221
6.4.1. Настройка модема . . . 221
6.4.2. Подготовка к выходу в Интернет . . . 223
6.4.3. Специальные возможности браузеров . . . 227
Konqueror . . . 227
Galeon . . . 228
Epiphany . . . 230
Netscape . . . 230
FireFox . . . 231
6.4.4. Текстовые браузеры . . . 231
6.4.5. Полезный трюк: Что делать, если браузер «подвисает» на какой-то странице . . . 232
6.4.6. Набор программ для работы в Интернете . . . 233
Болталка ICQ . . . 233
Менеджер закачки файлов Downloader for X . . . 233
FTP-клиенты . . . 237

Глава 7. Основы администрирования системы 240

7.1. Что понимается под администрированием системы 241
7.2. Конфигураторы Linux 242
7.3. Пользователи и квоты 244
7.3.1. Учетные записи пользователей 244
Файл /etc/passwd — информация о пользователях 245
Информация о группах пользователей. Файл /etc/group 246
7.3.2. Создание и удаление пользователей и групп 247
7.3.3. Квотирование 248
Что такое квотирование. Особенности квотирования в Linux 248
Ядро и поддержка квотирования 249
Назначение и активация квот 251
7.4. Подключение и конфигурирование аппаратных устройств 255
7.4.1. Ядро и поддержка устройств 255
7.4.2. Утилиты для работы с модулями 256
7.4.3. Kudzu — утилита для автоматического определения устройств 257
7.4.4. Настройка установленных устройств 258
7.5. Установка программного обеспечения 258
7.5.1. Установка из исходных текстов 258
7.5.2. Установка из бинарных пакетов 260
Как это делается и что для этого нужно 260
Менеджер пакетов rpm 261
Графические менеджеры пакетов 263
Apt: Debian-совместимый менеджер пакетов 264
7.5.3. Установка из пакетов, содержащих исходный код 267
7.6. Клонирование и восстановление системы 268

Глава 8. Язык командного интерпретатора 270

8.1. Параметры 272
8.2. Подстановки 273
8.3. Массивы 274
8.4. Управляющие структуры 275
8.4.1. Условные операторы 275
Условный оператор if 276
Оператор test и условные выражения 276
8.4.2. Операторы цикла 278
Оператор цикла с перечислением for 278
Оператор цикла с условием while 279
Оператор цикла с инверсным условием until 280
Оператор цикла с выбором select 280
8.5. Условная подстановка параметров 281
8.6. Функции 282
8.7. Обработка сигналов и протоколирование 283

Глава 9. Управление процессами 285

9.1. Как загружается Linux 286
9.1.1. Начальная загрузка: LILO и GRUB 286
Общие механизмы 286
Загрузчик LILO 287
Загрузчик GRUB 291
Как установить графический фон загрузчика GRUB 293
9.1.2. Продолжение загрузки 293
Демон init 293
Уровни выполнения 294
Конфигурационный файл /etc/inittab 296
Инициализация в стиле BSD 298
Инициализация в стиле System V 299
9.2. Команды управления процессами 302

9.2.1. Иерархия процессов: ps и pstree . . . 302
9.2.2. Информация о ресурсах системы: команды free, df, du . . . 303
9.2.3. Процессы в реальном времени: команда top . . . 304
9.2.4. Приоритет процесса: команды nice и renice . . . 306
9.2.5. Фоновый режим: команды jobs, fg, bg . . . 307
9.3. Протоколирование системы . . . 307
9.3.1. Конфигурационный файл /etc/syslog.conf . . . 309
9.3.2. Сетевое протоколирование . . . 312
9.3.3. Протоколирование ядра. Демон klogd и команда dmesg . . . 313
9.3.4. Что делать с протоколами дальше? Утилита logrotate . . . 313
9.4. Выполнение заданий по расписанию . . . 315
9.4.1. Запуск задания в назначенное время: команда at . . . 315
9.4.2. Диспетчер расписаний — демон cron . . . 316
Управление файлами расписаний . . . 316
Формат файла расписания . . . 317

Глава 10. Резервное копирование и восстановление данных . . . 318

10.1. Восстановление удаленного файла . . . 319
10.1.1. Midnight Commander . . . 320
10.1.2. Утилита debugfs . . . 320
10.2. Стратегия резервного копирования . . . 321
10.3. Оборудование для резервного копирования . . . 322
10.3.1. Стример . . . 322
Подключение стримера с интерфейсом SCSI . . . 323
Подключение стримера с интерфейсом FDC . . . 323
10.3.2. Магнитооптический диск . . . 323
10.4. Программное обеспечение для резервного копирования . . . 324
10.4.1. Простое резервное копирование по сети . . . 324
10.4.2. Управление стримером . . . 325
10.4.3. Команды dump и restore . . . 326
10.4.4. Архиватор cpio . . . 327
10.4.5. Программа AMANDA . . . 329
10.5. Дублирование данных: введение в RAID . . . 332
10.6. Как найти, спрятать и безвозвратно уничтожить данные . . . 335

Глава 11. Базовое конфигурирование сервера . . . 338

11.1. Серверные технологии Linux . . . 339
11.2. Организация и состав Linux-сервера . . . 340
11.3. Суперсервер xinetd . . . 344
11.3.1. Установка суперсервера xinetd . . . 344
11.3.2. Настройка суперсервера xinetd . . . 345
11.3.3. Запуск xinetd . . . 349
11.3.3.1. Защита xinetd . . . 350
11.3.3.2. Пример файла конфигурации /etc/xinetd . . . 350
11.4. Удаленный доступ: ssh и telnet . . . 354
11.4.1. Использование telnet . . . 354
11.4.2. Настройка и использование SSH . . . 355
Что такое SSH . . . 355
Настройка SSH на сервере . . . 357
Запуск демона sshd . . . 359
Использование SSH-клиента . . . 360
Аутентификация средствами SSH . . . 361

Глава 12. Разделение ресурсов: NFS и Samba . . . 363

12.1. NFS — сетевая файловая система . . . 364
12.1.1. Клиент NFS: монтирование сетевого каталога . . . 364
12.1.2. Настройка сервера NFS . . . 365

12.2. Samba:Linux-сервер для Windows-клиентов ... 367
12.2.1. Samba на сервере ... 367
12.2.2. Настройка Samba ... 368
Секция [global] ... 369
Секция [homes] ... 371
Секция [public] ... 371
12.2.3. Практические примеры настройки ... 371
12.2.4. Доступ к принтеру Linux для Windows-машин ... 372
12.2.5. Доступ к Windows-принтеру с компьютера, работающего под Linux ... 374
12.2.6. Конфигуратор SWAT ... 376
12.2.7. Samba и безопасность ... 378
12.2.8. Оптимизация Samba ... 379
12.3. Совместное использование каталогов в Linux Mandrake ... 380
12.4. Программа LinNeighborhood — правильный выбор ... 381

Глава 13. DNS — служба имен ... 383

13.1. Введение в DNS ... 384
13.2. Настройка клиента DNS ... 385
13.3. Настройка сервера DNS ... 386
13.3.1. Обновление корневого кэша ... 390
13.4. Кэширующий сервер DNS ... 392
13.4.1. Настройка кэширования на DNS-сервере ... 392
13.4.2. Возможные проблемы и их решение ... 393
13.5. Вторичный сервер DNS ... 394
13.6. Просмотр DNS-зоны. Утилита nslookup ... 395
13.7. Оптимизация настроек сервера DNS ... 397
13.8. Защита сервера DNS ... 398
13.8.1. Настройка и запуск DNS-сервера в chroot-окружении ... 398
13.9. Использование подписей транзакций. Механизм TSIG ... 399

Глава 14. Почтовый сервер ... 401

14.1. Установка и настройка sendmail ... 404
14.1.1. Базовая настройка sendmail ... 405
14.1.2. Редактирование конфигурационных файлов ... 408
14.2. Аутентификация в sendmail ... 412
14.2.1. Установка и настройка SASL ... 412
14.2.2. Настройка sendmail+SASL ... 413
14.2.3. Настройка почтовых клиентов с использованием аутентификации ... 414
14.3. Агент доступа — fetchmail ... 416
14.4. Автоматическая сортировка входящей почты — программа procmail ... 416
14.5. Создание списка рассылки ... 420
14.6. Защита программы sendmail. Программа smrsh ... 421

Глава 15. Настройка сервера FTP ... 423

15.1. Сервер WU-FTPD ... 424
15.1.1. Настройка WU-FTPD. Конфигурационные файлы ... 426
Основной файл ftpaccess. Директивы сервера WU-FTPD ... 426
Файл ftphosts — параметры доступа для пользователей с указанных узлов ... 430
Файл ftpusers — список локальных пользователей, которым запрещено пользоваться WU-FTPD ... 430
Файл ftpservers — разные настройки ftp-сервера для различных узлов ... 431
Файл ftpconversions — форматы сжатия ... 431
Файл xferlog — журнал FTP-сервера ... 432
15.2. Сервер ProFTP ... 433
15.2.1. Установка и запуск ProFTPD ... 433

15.2.2. Настройка ProFTPD. Файл /etc/proftpd.conf . . . 433
15.2.3. Разграничение доступа к серверу ProFTP . . . 436
Организация анонимного FTP-сервера . . . 437
15.3. Утилиты обслуживания FTP-сервера . . . 438
15.4. Виртуальный узел FTP . . . 439
15.5. Защита FTP . . . 440

Глава 16. HTTP-сервер Apache . . . 442

16.1. Установка Apache . . . 443
16.2. Настройка Apache. Файлы конфигурации . . . 445
16.3. Основные настройки. Файл httpd.conf (httpd2.conf) . . . 445
16.3.1. Общие директивы . . . 446
16.3.2. Директивы протоколирования . . . 448
16.3.3. Директивы управления производительностью . . . 448
16.3.4. Директивы обеспечения постоянного соединения с клиентом . . . 449
16.3.5. Директивы создания виртуальных узлов . . . 450
16.3.6. Директивы настройки отображения каталогов . . . 450
16.3.7. Директивы обработки MIME-типов . . . 451
16.3.8. Директивы для работы с многоязычными документами . . . 452
16.3.9. Директивы перенаправления . . . 453
16.3.10. Директивы обработки ошибок . . . 453
16.3.11. Директивы управления доступом к отдельным каталогам . . . 454
Блок директив Limit . . . 455
Блок директив Location . . . 457
16.4. Файл ротации журналов /etc/logrotate.d/httpd . . . 457
16.5. Системный файл конфигурации /etc/sysconfig/httpd . . . 458
16.6. Сценарий запуска сервера Apache /etc/init.d/httpd . . . 458
16.7. Графические конфигураторы Apache . . . 459
16.8. Каталоги пользователей . . . 460
16.9. Виртуальный HTTP-сервер . . . 461
16.9.1. Виртуальные серверы с идентификацией по имени . . . 461
16.9.2. Виртуальные серверы с идентификацией по IP-адресу . . . 463
16.10. SSL и Apache . . . 464
16.10.1. Установка SSL . . . 464
16.10.2. Подключение SSL к Apache . . . 465
16.10.3. Генерирование сертификатов . . . 467
16.11. Пример файла httpd.conf . . . 468
16.12. Перекодирование русскоязычных документов «на лету» . . . 483
16.12.1. Russian Apache: установка, настройка, использование . . . 484
16.12.2. Настройка перекодировки русскоязычных документов . . . 484
16.13. Защита сервера Apache . . . 486
16.14. Сервер kHTTPd — веб-сервер уровня ядра . . . 487
Настройка kHTTPd . . . 487

Глава 17. Установка и настройка MySQL . Связка Apache+PHP+MySQL . . . 489

17.1. Установка MySQL . . . 490
17.1.1. Назначение пароля суперпользователя . . . 490
17.1.2. Автозапуск сервера MySQL . . . 492
17.1.3. Пользователи сервера MySQL и их права . . . 492
17.2. Клиентская часть MySQL . . . 494
17.3. Установка PHP и настройка связки Apache+PHP+MySQL . . . 494
17.3.1. Первый способ: из пакетов RPM . . . 495
17.3.2. Тестируем созданную конфигурацию . . . 497
17.3.3. Второй способ: из исходных текстов . . . 498
17.4. Защита сервера MySQL . . . 499
17.5. Введение в язык SQL . . . 500
17.5.1. Общие понятия . . . 500
17.5.2. Краткий практический курс SQL . . . 500

Глава 18. Прокси-серверы. SQUID и SOCKS 510

18.1. Что такое прокси-сервер? 511
18.2. Установка SQUID 512
18.3. Настройка SQUID. 512
18.4. Запуск SQUID 513
18.5. Расширенные настройки SQUID. Конфигурационный файл squid.conf 514
18.5.1. Параметры сети 514
18.5.2. Параметры соседей 515
18.5.3. Управление кэшем 515
18.5.4. Протоколирование. 516
18.5.5. Параметры внешних программ 516
18.5.6. Параметры администрирования 517
18.6. Списки ACL. 517
18.6.1. Параметры доступа 518
18.7. Отказ от рекламы. Баннерный фильтр. 519
18.8. Разделение канала с помощью SQUID 519
18.9. Настройка поддержки прокси у клиентов 521
18.10. Технология SOCKS5, или как использовать аську из локальной сети 522
18.10.1. Введение в SOCKS. Прокси-сервер SOCKS5 522
18.10.2. Настройка сервера SOCKS5 524
18.10.3. Запуск сервера socks5 525
18.10.4. Dante — еще один сервер SOCKS5. 526
18.10.5. Настройка клиентов SOCKS5 (ICQ и licq) 526

Глава 19. Маршрутизация и межсетевые экраны 528

19.1. Введение в маршрутизацию 529
19.2. Программы маршрутизации в Linux. 530
19.2.1. Демон routed. 530
19.2.2. Демон gated — правильный выбор 532
19.3. Расширенные средства маршрутизации. Комплекс iproute2. 535
19.3.1. Пакет iproute2 535
19.3.2. Утилита ip 535
19.3.3. Просмотр параметров сетевого устройства 536
19.3.4. Операции над адресами: команда ip address 536
19.3.5. Управление таблицей маршрутизации. 537
19.3.6. Динамическая маршрутизация 537
19.3.7. Управление правилами маршрутизации 537
19.4. Что такое брандмауэр 539
19.5. Цепочки правил 539
19.6. IPTables — пакетный фильтр для ядер 2.4.x. и 2.6.x 541
19.6.1. Что изменилось в IPTables по сравнению с IPChains 541
19.6.2. Настройка ядра Linux для поддержки IPTables 541
19.6.3. Первичная настройка IPTables. Задание политики по умолчанию 542
19.6.4. Действия над цепочками 543
19.6.5. Правила фильтрации 543
19.6.6. Фильтрация по отдельным пользователям 545

Глава 20. Настройка ядра 546

20.1. Многообразие ядер Linux 547
2.6.x 547
2.4.x 547
2.6.x.y. 548
2.6.x-mm 548
2.6.x-mm-jedi 548
2.6.x-pre и 2.6.x-rc 549
2.6.x-tiny 549
2.6.x-ac 549
Прочие 549

20.2. Зачем настраивать ядро? ... 549
20.3. Динамические параметры ядра ... 550
20.4. Загрузочные параметры ядра ... 551
20.4.1. Параметры корневой файловой системы ... 552
20.4.2. Объем памяти ... 552
20.4.3. Управление RAMDISK ... 553
20.4.4. Управление планировщиком ввода/вывода ... 554
20.4.5. Другие параметры ядра ... 555
20.5. Компиляция ядра ... 555
20.5.1. Зачем обновлять ядро? ... 555
20.5.2. Конфигурирование ядра ... 556
20.5.2.1. Code maturity level options ... 556
20.5.2.2. General setup ... 557
20.5.2.3. Loadable module support ... 558
20.5.2.4. Processor type and features ... 559
20.5.2.5. Power Management Options ... 562
20.5.2.6. Bus Options ... 564
20.5.2.7. Executable file formats ... 564
20.5.2.8. Device drivers ... 564
20.5.2.9. Filesystems ... 571
20.5.2.10. Kernel hacking ... 572
20.5.2.11. Cryptographic options ... 572
20.5.3. Сборка ядра ... 572

Глава 21. Создаем консольное приложение ... 575

21.1. Компилятор gcc ... 576
21.1.1. Вызов gcc ... 577
21.1.2. Общие опции ... 578
21.1.3. Опции языка ... 579
21.1.4. Опции препроцессора ... 579
21.1.5. Опции компоновщика ... 579
21.1.6. Опции каталогов ... 579
21.1.7. Опции отладки ... 580
21.1.8. Опции оптимизации ... 580
21.2. Сборочная утилита make ... 580
21.3. Пакет binutils и другие полезные программы ... 583
21.3.1. ansi2knr ... 584
21.3.2. as ... 584
21.3.3. bison ... 584
21.3.4. flex ... 585
21.3.5. gprof ... 585
21.3.6. strip ... 585
21.4. Пример программы на C ... 585

Глава 22. Отладка, трассировка и оптимизация программ ... 588

22.1. Ошибки и отладка ... 589
22.2. Отладчик gdb ... 591
22.3. Пример отладки программы ... 593
22.4. Трассировка системных вызовов ... 598
22.5. Оптимизация программ.Профайлер gprof ... 601
22.5.1. Использование профайлера ... 602
22.5.2. Как оптимизировать программу ... 604

Глава 23. Разработка графического приложения: библиотека GTK+ ... 605

23.1. Введение в GTK+ ... 606
23.2. Библиотека Glib ... 607

23.2.1. Стандартные типы данных библиотеки Glib . . . 607
23.2.2. Функции для работы с памятью . . . 608
23.2.3. Строки и Glib . . . 608
23.4.4. Списки . . . 608
23.2.4. Таймеры в Glib . . . 610
23.3. Первая программа на GTK+ . . . 612
23.3.1. Виджиты . . . 612
23.3.2. Окна . . . 613
23.3.3. Изменение размеров окна . . . 616
23.3.4. Обработка сигналов . . . 617
23.3.5. Виджит событий — EventBox . . . 619
23.4. Виджиты . . . 621
23.4.1. Рождение, смерть и состояния виджита . . . 621
23.4.2. Упаковка виджитов, поля ввода и кнопки . . . 622
Нет файла . . . 623
23.4.3. Переключатели . . . 630
23.4.4. Список . . . 634
23.4.5. Выбор файлов . . . 639
23.4.6. Диалог завершения работы . . . 640
23.4.7. Меню . . . 645
23.4.8. Иерархия виджитов . . . 645

Глава 24. Студия Glade . . . 648

24.1. Что такое Glade? . . . 649
24.2. Знакомство с Glade . . . 649
24.3. Работа с проектом . . . 659
24.4. Создание меню . . . 663
24.5. Интересные виджиты . . . 664

Глава 25. Пакет Dialog . . . 667

25.1. Что такое Dialog? . . . 668
25.2. Сообщения . . . 668
25.3. Виджит Yes-no . . . 670
25.4. Окно ввода текста . . . 671
25.5. Зависимые и независимые переключатели . . . 673
25.6. Организация меню . . . 675
25.7. Календарь . . . 677
25.8. Шкала прогресса . . . 678

Глава 26. Взаимодействие процессов в Linux . . . 679

26.1. Способы взаимодействия . . . 680
26.2. Полудуплексные каналы . . . 680
26.3. Каналы типа FIFO . . . 683
26.4. Основные принципы System V IPC . . . 686
26.5. Очереди сообщений . . . 687
26.5.1. Основные структуры ядра для работы с очередями . . . 687
26.5.2. Создание очереди сообщений . . . 690
26.5.3. Постановка сообщения в очередь . . . 691
26.5.4. Получение сообщений очереди . . . 694
26.5.5. Проверка наличия сообщения в очереди . . . 695
26.5.6. Тотальный контроль . . . 696
26.6. Семафоры . . . 697
26.6.1. Создание множества семафоров . . . 699
26.6.2. Выполнение операций над семафорами . . . 700
26.6.3. Контроль семафора . . . 702
26.7. Разделяемые сегменты памяти . . . 705

Глава 27. Создание сетевого приложения в Linux 712

27.1. Протокол TCP/IP 713
27.1.1. Многоуровневая архитектура стека TCP/IP 713
27.1.1.1. Уровень сетевого интерфейса 714
27.1.1.2. Межсетевой уровень 714
27.1.1.3. Транспортный (основной) уровень 715
27.1.1.4. Уровень приложений 716
27.1.2. Структура пакетов IP и TCP 716
27.2. Протокол ICMP 717
27.2.1. Для чего используется протокол ICMP 717
27.2.2. Структура ICMP-пакета 718
27.2.3. Тип и код ICMP-сообщения 720
27.2.4. Функции для работы с протоколом ICMP 721
Технические подробности 722
27.3. Программирование сокетов 726
27.3.1. Что такое сокет? 726
27.3.2. Создание и связывание сокета 727
27.3.3. Установление связи с удаленным компьютером 730
Функция listen() 730
Функция connect() 731
Функция accept() 731
27.3.4. Функция gethostbyname() 732
27.3.5. Функции сетевого ввода/вывода 733
Обмен данными в режиме SOCK_STREAM 733
Обмен данными в режиме SOCK_DGRAM 734
27.3.6. Завершение сеанса связи 735
27.3.7. Программа-сервер 736
27.3.8. Программа-клиент 740
27.3.9. Установка опций сокета 742
27.3.10. Сигналы и сокеты 744
27.3.11. Мультиплексирование 745
27.3.12. Неблокирующие операции 749

Глава 28. Программирование ядра 750

28.1. Каркас модуля 751
28.2. Компиляция модуля 754
28.3. Работа с устройствами 755
28.4. Операции над устройством. Поиск устройств 760

Приложение. Таблицы соответствия Windows- и Linux-программ 768

Работа в Интернет 769
Работа с файлами 771
Прикладные и системные программы 771
Офисные приложения 773
Мультимедиа 774
Разработка программного обеспечения 775
СУБД 776
Математические пакеты 776
Игры 777

Введение

Об этой книге

Все большую и большую популярность в России и во всем мире завоевывает Linux — UNIX-подобная операционная система для IBM-совместимых персональных компьютеров. Растет количество приложений, разработанных для нее. Постоянно расширяется круг задач, для которых она используется. Из операционной системы для программиста и администратора интернет-сервера она уже превратилась в реальную альтернативу ОС Windows в сфере домашнего и офисного применения.

Эта книга призвана помочь читателю освоить ОС Linux и научиться полноценно работать в ней. Я предполагаю, что читатель знает, с какой стороны подойти к компьютеру, умеет решать свои прикладные задачи, работая в среде Windows или хотя бы MS DOS, и хочет попробовать на практике операционную систему совсем другого класса, чтобы в дальнейшем, возможно, полностью «пересесть» на нее. Изложение материала происходит последовательно, начиная с этапа установки ОС и заканчивая советами по разработке собственных приложений. Знакомство с операционной системой — соперницей Windows начинается с описания прикладных программ, работающих под ее управлением и предназначенных для решения повседневных задач, стоящих перед пользователем домашнего компьютера: редактирование текста и изображений, работа в Интернете, проигрывание музыки и видео и т.п. Для облегчения перехода с Windows на Linux в конце книги приведена таблица соответствия популярных Windows-приложений приложениям Linux, разработанным для решения тех же задач. Далее в разделе для пользователя изложены основные сведения о внутренней организации Linux — ядре, файловой системе, процессах и демонах, их взаимодействии, — необходимые для настройки и базового администрирования системы, то есть обслуживания многопользовательской среды, установки программного обеспечения, обеспечения сохранности данных. Собственно сетевому администрированию, то есть настройке различных сетевых служб, посвящен второй раздел книги. И для читателей, знакомых с программированием в среде

Windows, предназначен последний раздел, освещающий особенности разработки приложений в ОС Linux и некоторые полезные инструменты разработчика.

Почему Linux?

Устанавливая Linux, вы получаете множество преимуществ:

- **Гибкость**. Мало того, что практически все в Linux поддается настройке в соответствии с именно вашими задачами и оборудованием, так вам еще и становятся доступны исходные тексты ядра и приложений, и вы можете модифицировать систему так, как вам нужно. Такое можно встретить далеко не в каждой операционной системе, особенно семейства Windows. Вы видели где-нибудь исходные тексты хотя бы Блокнота Windows? Мне, например, очень не хватает функции замены текста в этом редакторе. Для решения этой проблемы я написал собственный редактор, в котором и реализовал эту функцию. А если мне нужно сделать небольшое изменение в ядре? Не буду же я полностью переписывать Windows? Остается только надеяться, что новая функция будет реализована в следующей версии, и ради этой единственной функции устанавливать «монстра», пожирающего еще больше системных ресурсов.
- **Дешевизна**. ОС Linux абсолютно бесплатна. Конечно, компакт-диски с дистрибутивами продаются за деньги, но эти деньги вы платите не за лицензию, а за сам носитель, подбор программного обеспечения на нем и программу-инсталлятор — все, как у пиратов, с той лишь разницей, что это: а) полностью легально, б) гарантированно работает и пользуется технической поддержкой. Вы можете и не покупать дистрибутив, а анонимно и бесплатно выкачать исходные тексты или уже собранные программы из Интернета, установив их самостоятельно и заплатив только за трафик — и это тоже полностью легально. Вам не придется ничего доплачивать, устанавливая Linux на каждый следующий компьютер, не нужно покупать отдельную лицензию на использование Linux на сервере. В любом случае стоимость всего программного обеспечения составит всего несколько долларов. Я не буду сравнивать стоимость построения Linux-сервера со стоимостью аналогичного сервера на платформе Microsoft, вы сами можете это сделать на сайте компании Microsoft.
- **Простота обслуживания**. Сама система и все службы настраиваются путем редактирования конфигурационных файлов. Это обычные текстовые файлы; зная их расположение и формат, вы сможете настроить любой дистрибутив, даже если у вас под рукой нет никаких инструментов, кроме текстового редактора. Кроме того, для облегчения перехода с ОС Windows NT/2000/2003 Server, где сервисы настраиваются в основном через графический интерфейс, создано

множество графических конфигураторов, работа с которыми интуитивно понятна и позволяет сосредоточиться на сути выполняемых действий, а не способе их выполнить.

- **Нетребовательность к ресурсам**. Системные требования зависят от дистрибутива (конкретной реализации Linux) и версии ядра. Существуют дистрибутивы, специально созданные для корректной работы на старых и «бедных» машинах. Например, для организации интернет-сервера на базе дистрибутива Red Hat версии 5.2 вам вполне хватит компьютера с процессором Intel 80486DX и 32 мегабайтами ОЗУ. А окончательно устаревшую 386 машину, на которой никакая современная ОС Windows не запустится, под управлением Linux можно вернуть в строй в качестве маршрутизатора или брандмауэра.

В чем же причина того, что большинство пользователей до сих пор не рассматривают Linux как полноценную настольную операционную систему? Нет, не в том, что Linux так недружелюбен, как его малюют. И не в том, что под Linux нет ни офисных приложений, ни игр. И не в том, что Linux довольно долго имел проблемы с русским языком. Причина заключается в правильной маркетинговой политике Microsoft — нужно отдать ей должное. Совсем не обязательно создать лучший программный продукт, нужно убедить в том, что он лучше. Это я о Windows (про DOS я вообще молчу — урезанная версия XENIX, которая, в свою очередь является урезанной версией UNIX) — эта операционная система использовала UNIX-решения 20 летней давности, в то время как о существовании самой UNIX некоторые пользователи Windows 95 и не догадывались.

Задача этой книги — не только научить пользователей работать с Linux, но и развеять мифы о том, что это сложная и неподъемная система, предназначенная только для профессионалов. Если вы читаете эту книгу, то для себя данный миф вы уже почти разрушили. Осталось только чуть-чуть. Вам осталось узнать об истории создания и развития Linux. Ведь о развитии DOS и Windows знают все — об этом написано даже в школьных учебниках по информатике. А про Linux и UNIX в лучшем случае говорят: «Есть такая система».

История Linux

Все началось с игры

Не поверите, но все началось с небольшой игры, написанной Кеном Томпсоном. Но чтобы вам было понятно, при чем здесь игра, нужно сделать небольшой экскурс во времена «до игры».

В далеком 1965 году начались работы над созданием операционной системой MULTICS (MULTiplexed Information and Computing System)

для компьютера GE-645 (об этом компьютере вы точно не прочитаете в школьном учебнике по информатике). Инициатором была компания Bell Laboratories (подразделение компании AT&T), а ее помощниками выступали не менее известные организации — General Electric и Массачусетский технологический институт (МТИ). Что же ожидали от MULTICS? Процессорное время GE-645 стоило очень дорого, поэтому нужна была система разделения процессорного времени, обеспечивающая высокую скорость обработки данных и их совместное использование. Посмотрите на рис. 1 — это система GE-645. Обратите внимание, какое помещение она занимает. А теперь вспомните процессор Intel 80386. К чему это я? Производительность GE-645 чуть выше, чем у процессора 80386. До чего техника дошла (кстати, «шла» она к этому лет эдак 20)!

Операционная система Multics была запущена в 1969 году, но она не оправдала возложенных на нее надежд. Поэтому вскоре компания Bell Laboratories прекратила работу над этой системой. Коммерческого успеха эта система не принесла. Правда, для справедливости нужно отметить, что МТИ умудрился продать около 80 инсталляций этой системы, причем некоторые из них «дожили» до начала 90-х годов.

Но не бывает худа без добра. Multics послужила своеобразным толчком для создания новой операционной системы — UNIX.

Что же произошло в 1969 году, который считают годом рождения UNIX? Четыре сотрудника Bell Labs, а именно Руд Кенедей (Rudd Canaday), Дуг Мак-Илрой (Doug McIlroy), Дэннис Ричи (Dennis Ritchie) и Кен

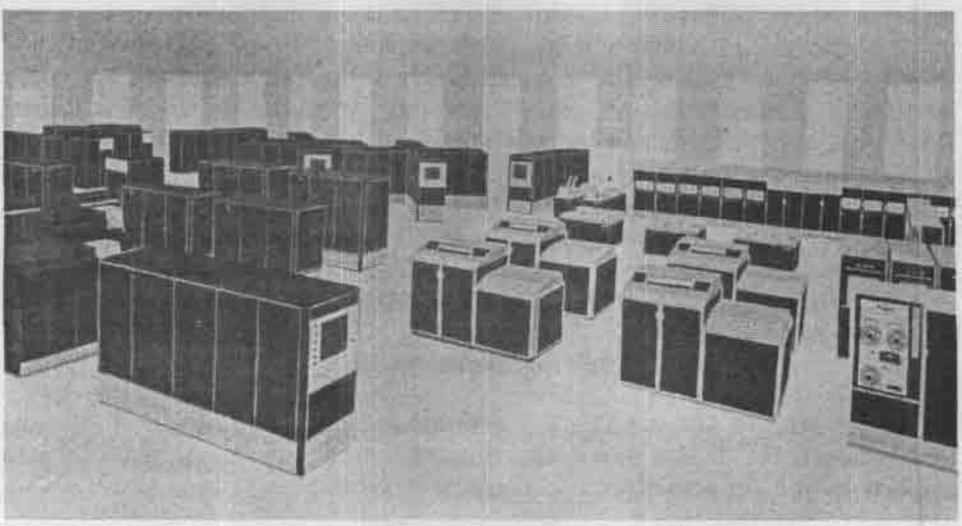


Рис. 1. *GE-645 © General Electric Information Systems Equipment Division, 1968*

Томпсон (Ken Thompson) попытались возродить Multics. Все они ранее работали над проектом Multics и не смогли смириться с таким поворотом событий.

Прежде всего был разработан проект файловой системы — потом эта файловая система стала файловой системой UNIX. Кен Томпсон написал программы, эмулирующие работу файловой системы и режима подкачки. Получилось что-то вроде прототипа ядра новой системы. Это ядро было предназначено для GE-645.

А сейчас на арене появляется та самая игра — «Космическое путешествие», симулятор полета, написанный Кеном Томпсоном под Multics. Но после прекращения работы над Multics эту игру оказалось не на чем запускать. Обнаружив в углу лаборатории редко используемую машину PDP-7, Томпсон с Ричи решили переписать игру для нее. Переносить программу на перфоленте оказалось неудобно, и Томпсон разработал для PDP-7 файловую систему, утилиты для работы с файлами и командную оболочку. Все это писалось на машине GE и переносилось на PDP-7 на перфоленте, но, как только был закончен ассемблер для PDP-7, система встала на собственные ноги. Получилась новая операционная система для компьютера PDP-7, которую назвали, по аналогии с Multics, "UNICS" (UNiplexed Information and Computing Service). Такое название ей дал Питер Нейман (Peter Neumann). Кто же переименовал ее в UNIX, до сих пор не известно.

Рис. 2. *PDP-7*

В то время все программное обеспечение, том числе и операционные системы, писалось на машинно-зависимом языке (ассемблере), то есть если программа, например, написана для компьютера PDP-7, ее нельзя запустить на компьютере с другой архитектурой, например, на GE. Кен Томпсон задался целью сделать универсальную систему, которая могла бы работать на нескольких платформах, то есть создать переносимую, независимую от «железа» операционную систему. Для своей системы он начал разрабатывать язык B, на котором он хотел переписать всю систему. Но «до ума» этот язык довел Дэннис Ричи, обновленная версия языка стала называться C. Да, этот тот самый язык, на котором каждый из нас написал не одну программу. Сейчас мы подходим к созданию той идеологии UNIX'а, которая используется по сей день.

О работе над новой операционной системы руководство компании Bell Labs ничего не знало. Они думали, что ведущие программисты работают над системой обработки текстов для AT&T. Действительно, Томпсон написал строковый (не текстовый!) редактор ed, который присутствует до сих пор во всех современных дистрибутивах, а Ричи создал программу roff. Новые разработки понравились компании AT&T, которая была заинтересована в дальнейшем развитии проекта и приобрела новый компьютер — PDP-11. Для этого компьютера UNIX был полностью переписан: вся система, включая ядро, была написана на языке C, что позволило переносить систему на другие платформы. На ассемблере была написана только та часть ядра, которая была непосредственно связана с «железом». Это было в 1971 году.

Становление UNIX как операционной системы

Очень много новых функций в ядро системы было добавлено всего за один год — с 1971 по 1972. Система заметно «возмужала» по сравнению с 1969 годом. Теперь она действительно стала похожа на операционную систему. Но система не распространялась — ее использовало только одно подразделение AT&T. Первым шагом системы за пределы компании AT&T стала установка системы в компании New York Telephone. Система была установлена на компьютере PDP-11/20, обладавшем аж 56 килобайтами оперативной памяти и двумя жесткими дисками по 2,4 Мб каждый. С этого момента началось распространение системы.

Уже в начале 1973 года насчитывалось 16 инсталляций системы. Это довольно неплохо для времен, когда такую роскошь, как компьютер, могла позволить себе только крупная организация, и то не каждая. Компьютер тогда покупался для решения промышленных задач, а не для набора текста и прослушивания музыки.

Вот что писал Ричи о UNIX'е: "*Мы хотели сохранить не только хорошую среду программирования, в которой можно было решать задачи, но и*

систему, вокруг которой могло сформироваться товарищество. Мы по опыту знали, что сущностью совместного использования компьютера (the essence of communal computing) является не только замена перфокарточного ввода на терминал, но и предоставление средств и стимулов для более тесного общения, реализуемого средствами удаленного доступа к машине, работающей в режиме разделения времени."

Летом 1974 года в журнале «Communications of the ACM» появилась первая статья о UNIX. Ее авторами были, как и следовало ожидать, Томпсон и Ричи. Они описывали общее устройство UNIX. В той же статье говорилось, что по состоянию на июль 1974 года система была установлена уже на 600 (!) компьютерах. Вы только вдумайтесь в цифру: количество инсталляций за год возросло более чем в 37 раз. Самое интересно, что система развивалась почти при полном отсутствии поддержки со стороны AT&T.

После этой статьи операционной системой UNIX заинтересовались многие научно-исследовательские организации. Но AT&T не имела права заниматься бизнесом, следовательно, система не могла продаваться. Чтобы система все-таки развивалась, Томпсон бесплатно рассылал всем желающим наборы дисков с операционной системой. Такой поворот событий повлек следующую волну развития ОС. Например, студенты Йельского университета разработали командную оболочку (shell), а студенты другого университета создали первую сеть на основе UNIX. В университете Беркли (Калифорния) опять-таки студенты создали огромное количество различных утилит и новую оболочку.

Если вы читали какую-то литературу по UNIX, то, наверное, знаете, как нумеровались версии UNIX — V1, V2 и т.д. Откуда взялась именно такая нумерация версий? В 1971 году Ричи и Томпсон написали руководство по программированию в UNIX. Версия, которая описывалась в первом издании этого руководства, стала называться V1, далее — V2 и т.д. Если быть предельно точными, то вот даты выхода изданий руководства:

- 3 ноября 1971 г. (описывалась версия V1)
- 12 июня 1972 г. (описывалась версия V2)
- Февраль 1973 г. (V3)
- Ноябрь 1973 г. (V4)
- Июнь 1974 г. (V5)
- Май 1975 г. (V6)
- Январь 1979 (V7)
- Февраль 1985 г.
- Сентябрь 1986 г.
- Октябрь 1989 г.

После 1979 года нумерация Vn больше не использовалась — чуть позже мы с вами узнаем, почему именно.

Важную роль в развитии UNIX сыграл Калифорнийский университет в Беркли. Как уже было сказано, студенты (точнее, аспиранты) этого университета разработали новую командную оболочку и несколько очень важных утилит. Одной из этих утилит мы пользуемся до сих пор — это текстовый редактор vi (правда, в 1975 году он назывался ex). В том же 1975 году Томпсон написал версию Pascal для UNIX.

Модифицированная в университете Беркли система получила название BSD (Berkeley Software Distribution). Все мы слышали название FreeBSD, некоторые из нас даже работали с этой системой. Так вот, сейчас мы знаем, откуда оно произошло. Данная система содержала Pascal и редактор ex (vi). Университет Беркли имел право заниматься коммерческой деятельностью, поэтому первая версия BSD сначала продавалась по цене $50. Всего было продано 30 копий системы. В 1978 году вышла вторая версия BSD — 2BSD. Вторая версия продавалась активнее — было продано 75 копий. Идеи, разработанные в Беркли, позже были использованы в следующей версии UNIX от Bell Labs.

Два года спустя Питер Вэйнер (Peter Weiner) и Гейнц Ликлама (Hienz Lycklama) создали компанию Interactive Systems, которая тоже принялась продавать UNIX. Правда, продавалась она под другим именем — Irdis (это первый клон UNIX).

Вернемся к нумерации Vn. Прекратилась она в 1979 году — в этом году вышла последняя «настоящая» UNIX — UNIX V7. В ее состав входили компилятор C, программы awk, make, uucp, find, cpio и командная оболочка Bourne (и, разумеется, много других программ, которых здесь перечислять нет смысла). Именно эта система была перенесена Дэннисом Ричи и Стивом Джонсоном (оба — сотрудники AT&T) на платформу Interdata, а чуть позже несколько программистов из австралийского университета перенесли ее на платформу Interdata 8. Правда, после переноса на другую платформу немного пострадала производительность системы — за все нужно платить. Университет Беркли занялся решением этой проблемы и решил ее. В 1982 году вышла версия 2.8.1BSD, которая работала значительно быстрее предшествующей.

Бурное развитие UNIX

В 1980 году агентству DARPA (the Defense Advanced Research Projects Agency) потребовалось увеличить мощность компьютеров, на которых базировалась сеть ARPANET (предок сети Интернет). Ресурсы стареньких PDP-10 были уже полностью исчерпаны, поэтому решено было перейти на более мощные компьютеры — VAX. Но для этих компьютеров нужен был стек протоколов TCP/IP. Реализация TCP/IP от DEC отпала по ряду причин, поэтому агентству DARPA ничего другого не оставалось, как вы-

Рис. 3. PDP-10

Рис. 4. VAX

брать систему BSD. Данный факт положительно отразился на развитии UNIX — ее выбрали основной системой для интернета.

Благодаря тому, что UNIX сделалась основой интернета, темпы ее развития возросли. В начале 80-х вышло много различных версий UNIX. Выпускали их разные компании, но основными конкурентами были компания AT&T и университет Беркли. Первая выпустила системы Programmer's Workbench (рабочее место программиста) и UNIX System III. Университет Беркли тоже не отставал — с октября 1980 года по сентябрь 1983 года было выпущено 6 версий системы — 4, 4.1, 4.1a, 4.1b, 4.1c, 4.2. Последняя версия отличалась высокопроизводительной файловой системой и встроенной поддержкой сети (TCP/IP). До этого сеть в UNIX поддерживалась, мягко говоря, весьма слабо. Но после реализации стека TCP/IP для UNIX все изменилось. В 1983 году большинство подключенных к интернету компьютеров составляли компьютеры VAX с ОС UNIX.

Какие бывают UNIX'ы?

Как только компании осознали, что на UNIX можно заработать, причем заработать неплохо, на рынке появилось множество клонов UNIX под разными названиями. Например, компания Sun Microsystems (основана в 1982 году, а одним из основателей был автор редактора ex — Билл Джой) перенесла BSD на платформу, разработанную в Стенфорде, чем положила начало новому типу рабочих станций.

Наверное, вы когда-нибудь слышали об операционных системах IRIX, XENIX, HP-UX — все эти системы UNIX-подобные, их создали, соответственно, компании SGI, SCO, Hewlett-Packard. О XENIX можно вообще

долго говорить, но все, что вам нужно знать об этой системе — это то, что она является первым коммерческим UNIX'ом для платформы Intel. ОС IBM HP-UX 1.0 основана на системе System III от AT&T. Кстати, в 1983 году с компании AT&T был снят запрет заниматься компьютерным бизнесом, после чего компания сразу же принялась продавать свою систему System V от Bell Labs (не забываем, что это подразделение AT&T). Выпускала UNIX и компания IBM. Правда, она немного опоздала: продажи ее UNIX — системы AIX — начались только в 1990 году.

Обилие различных версий UNIX стало причиной так называемых UNIX-войн, когда конкурирующие компании выпускали все новые и новые версии UNIX'а.

Это далеко не вся история UNIX, но рассмотренного материала вполне хватит, чтобы заполнить пробел в ваших знаниях относительно истории развития этой операционной системы. Если вы заинтересовались, в Интернете есть множество статей, посвященных истории UNIX. Правда, большая их часть написана на английском языке.

Происхождение Linux

Операционная система Minix

В 80-х годах мощности персональных компьютеров не хватало для запуска на них ОС UNIX, но к началу 90-х ситуация радикально изменилась. Вычислительные мощности персональных компьютеров достигли нужного уровня, что позволило запускать UNIX на обычном PC. К этому же времени начали появляться версии UNIX для PC. Одной из таких систем была система Minix. Система Minix была разработана в 1987 году Эндрю Таненбаумом (Andrew S. Tannenbaum) как учебная программа: ее назначением была демонстрация устройства и принципа работы реальных операционных систем.

Поскольку Minix была больше демонстрационной, нежели реальной системой, она была далека от совершенства. К тому же она была ориентирована на процессор 80286, который был более доступным в то время. Исходные коды (12 000 строк) этой операционной системы были опубликованы в книге А.Таненбаума «Операционные системы», которую мог прочитать любой желающий. Одним из читателей этой книги оказался Линус Торвальдс — будущий создатель Linux.

Линус Торвальдс

Линус Бенедикт Торвальдс (Linus Benedict Torvalds) родился 28 декабря 1969 года в г. Хельсинки, Финляндия. Первым компьютером Линуса был

Commodore VIC-20, который купил его дед Лео Вальдемар Тёрнквист. Лео Вальдемар был профессором в Университете Хельсинки.

На «Коммодоре» Линус написал первую программу «Hello, World!". Наверно, все мы начинали именно с этой программы. За 4 года (компьютер был куплен в 1981 году) Линус выжал из него все, что мог, поэтому он стал копить деньги на новый компьютер. Основным источником его доходов в то время были школьные стипендии. В 1987 году он купил компьютер Sinclair QL. Это был очень мощный на то время компьютер. На Sinclair QL был установлен 32-разрядный процессор Motorola 68008 и 128 Кб оперативной памяти. Не удивляйтесь: именно 128 Кб. Это было очень много: для сравнения, на «Коммодоре» было установлено всего 3,5 Кб. На новом компьютере была установлена операционная система Q-DOS.

За четыре года Линус основательно освоил компьютер и заинтересовался работой операционной системы. В своей книге «Just for fun" он пишет: *«Операционками я заинтересовался так: купил флоппи-контроллер, но к нему прилагался такой поганый драйвер, что пришлось написать новый. Пока писал — обнаружил проколы в самой операционной системе или, по крайней мере, несоответствие между тем, что обещала документация, и тем, что реально происходило. Я с этим столкнулся, когда моя программа отказалась работать.»*

Sinclair QL «прожил» у Линуса 3 года — на год меньше, чем его предшественник. За эти три года Линус написал собственный ассемблер, текстовый редактор и несколько компьютерных игр.

После школы Линус поступил в Хельсинский университет на факультет компьютерных наук. На первом курсе ему немного не повезло — он вынужден был пойти служить в армию и прослужил там 11 месяцев. После возвращения из армии он продолжил учебу в университете. На втором курсе он прочитал книгу Таненбаума, посвященную операционным системам. Потом Линус писал: *«Как только я прочел предисловие, познакомился с концепцией UNIX и узнал, на что способна эта мощная, строгая и красивая операционная система, я захотел купить такой компьютер, на котором сможет работать UNIX. Я решил, что поставлю себе Minix — единственную по-настоящему полезную из известных мне версий.»*

Рис. 5. *Линус Торвальдс*

С этого все и началось...

Линус недоволен Minix

В январе 1991 года Линус приобрел в кредит компьютер на базе процессора Intel 80386@33Mhz с 4 мегабайтами оперативной памяти. На компьютер была установлена операционная система Minix. Но установленная система не совсем оправдала его ожидания. Больше всего ему не нравилась программа эмуляции терминала, которая была для него просто необходима — ведь с ее помощью он подключался к университетскому MicroVAX'у. Линус писал: *«Беда была в том, что я хотел скачивать и закачивать файлы. То есть мне нужно было уметь писать на диск. Для этого моей программе эмуляции нужен был драйвер дисковода. А еще ей был нужен драйвер файловой системы, чтобы она могла вникать в организацию диска и записывать скачиваемые файлы. ... разработка драйверов для дисковода и файловой системы казалась интересным делом. И я решил им заняться. Написал драйвер дисковода. А поскольку я хотел записывать файлы в файловую систему Minix, к тому же эта система была хорошо документирована, я сделал свою файловую систему совместимой с системой Minix. Таким образом, я мог читать файлы, созданные в Minix, и писать файлы на тот же диск, так что Minix могла читать файлы, созданные моей программой эмуляции терминала»*

Линусу ничего другого не оставалось, как написать собственную программу эмуляции терминала, что он и сделал, как видно из приведенной выше цитаты.

Кроме отвратительной терминальной программы Линусу не нравилось то, что в Minix нельзя было перевести текущую программу в фоновый режим. После написания своей программы эмуляции терминала Линус фактически взялся за написание своей операционной системы.

Сначала он принялся писать различные системные вызовы, но эта работа ему быстро надоела — ведь заранее нельзя знать ,что именно тебе понадобится. Поэтому он взял оболочку Bourne Again Shell (bash) и попытался запустить ее на своей системе методом научного «тыка» (в математике этот метод называется методом Коши). Он запускал оболочку, она требовала какой-то системный вызов и завершала свою работу с ошибкой. Линус анализировал, какой вызов был нужен оболочке, и реализовывал его. В 1991 году оболочка все-таки запустилась. Это был очень важный момент в развитии новой ОС: был заложен ее фундамент.

25 августа Линус написал в группу новостей comp.os.minix первое сообщение о создании новой операционной системы. Сохранился даже оригинал этого сообщения:

```
From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Newsgroups: comp.os.minix
Subject: What would you like to see most in minix?
```

```
Summary: small poll for my new operating system
Message-ID: <1991Aug25.205708.9541@klaava.Helsinki.FI>
Date: 25 Aug 91 20:57:08 GMT
Organization: University of Helsinki

Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't
be big and professional like gnu) for 386(486) AT clones.
This has been brewing since april, and is starting to get
ready. I'd like any feedback on things people like/dislike
in minix, as my OS resembles it somewhat (same physical
layout of the file-system (due to practical reasons) among
other things).

I've currently ported bash(1.08) and gcc(1.40), and things
seem to work. This implies that I'll get something prac-
tical within a few months, and I'd like to know what fea-
tures most people would want. Any suggestions are welcome,
but I won't promise I'll implement them :-)

Linus (torvalds@kruuna.helsinki.fi)

PS. Yes - it's free of any minix code, and it has a multi-
threaded fs. It is NOT protable (uses 386 task switching
etc), and it probably never will support anything other
than AT-harddisks, as that's all I have :-(.
```

Вот перевод этого сообщения:

```
From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Newsgroups: comp.os.minix
Subject: Что вам не хватает в minix?
Summary: небольшой опрос для новой операционной системы
Message-ID: <1991Aug25.205708.9541@klaava.Helsinki.FI>
Date: 25 Aug 91 20:57:08 GMT
Organization: Университет Хельсинки

Привет всем пользователям minix

Я разрабатываю (бесплатную) операционную систему (это
просто хобби, ничего «огромного» и профессионального вроде
GNU) для чипов 386(486) AT. Я пишу ее начиная с апреля
и похоже, что скоро она будет готова. Мне очень важно
узнать, что вам нравится/не нравится в minix, потому что
моя ОС напоминает minix (та же самая файловая система (из
практических соображений) и много еще чего).
На данный момент я перенес в нее bash(1.08) и gcc(1.40),
```

```
вроде бы они работают. Похоже, что через несколько месяцев
все будет готово и я хотел бы знать, какие функции вам
нужны. Принимаются любые заявки, но их выполнение я не
гарантирую :-)
Linus (torvalds@kruuna.helsinki.fi)

PS. Система свободна от кода minix и обладает много-
потоковой файловой системой. Она не переносима (поскольку
использует переключение задач 386 и др.) и возможно
никогда не будет поддерживать ничего, кроме AT-
винчестеров, поскольку кроме них у меня ничего нет :-(.
```

Версия Linux 0.01 была выпущена 17 сентября 1991 года. В своей книге «Just for fun» Линус пишет:

«И вот я решился ее выложить. Я не делал публичных объявлений, а просто написал пятерым-десятерым хакерам на личные адреса, что она лежит на FTP-сайте. В числе прочих я написал знаменитому среди фанатов Minix Брюсу Эвансу и Ари Лемке. Я выложил исходники самой Linux и еще несколько бинарников, чтобы можно было хоть что-то делать. Я сказал, что нужно, чтобы запустить все это хозяйство. На машине должна была стоять Minix (версия 386) и нужен был компилятор GCC. Причем на самом деле нужна была моя версия GCC, поэтому ее я тоже выложил.

Не думаю, чтобы ту версию проверяло больше одного-двух человек. Для этого нужно было возиться с установкой специального компилятора, выделить пустой раздел, чтобы использовать его для загрузки, откомпилировать мое ядро и запустить оболочку. А кроме запуска оболочки, делать было особенно нечего. Можно было распечатать исходники — всего 10 000 строк.

Я стал распространять свою операционку прежде всего, чтобы доказать, что все это не пустая болтовня — я действительно что-то сделал. В Интернете много болтают. О чем бы ни шла речь — об операционке или о сексе — многие в киберпространстве просто вешают лапшу на уши. Поэтому важно после того как ты растрезвонил, что пишешь операционку, иметь возможность сказать: «Вот — я ее и правда сделал. Я не трепло — можете сами посмотреть».

При чем здесь пингвины?

Кто и когда придумал для новой операционной системы название и логотип? Линус с самого начала хотел назвать свою операционную систему FREAX. Вот и работали бы мы сейчас на FREAX'ах. Если не верите — найдите в Интернете файл kernl/Makefile ядра версии 0.11.

А название "Linux" получилось следующим образом. Ари Лемке отвел место на FTP-сервере под исходные коды новой системы. Он назвал каталог с новой операционной системой pub/OS/Linux. Впоследствии именно это название за ней и закрепилось.

А логотип Linux придумали в 1996 году. Тогда в рассылке linux-kernel mailing list прозвучала идея о выборе логотипа для Linux. Сами понимаете, желающих заняться разработкой логотипа — предостаточно, поэтому различных вариантов было много. В процессе обсуждения логотипа Линус сказал, что ему нравятся пингвины. В ответ на это было предложено множество пингвинов в разных позах. Был также вариант пингвина, держащего на руках земной шар. На это Линус ответил: «*бедный пингвин не так силен, чтобы удержать земной шар, он, пожалуй, будет раздавлен... Так что если вы думаете о «пингвине», вы должны представлять себе слегка растолстевшего сидящего пингвина, хорошо поевшего и отрыгнувшего. Он сидит с довольной улыбкой — мир кажется прекрасным, если вы только что съели несколько галлонов свежей рыбы...*».

Логотип Linux — полностью соответствует пожеланию Линуса. Логотип создал Ларри Ивинг (Larry Ewing), причем он нарисовал его прямо в Linux с помощью программы GIMP (The GNU Image Manipulation Program). Если вы хорошо знаете зоологию (или хотя бы видели пингвинов по телевизору), то, наверное, заметили, что у настоящих пингвинов клюв и ласты — черные, а у нашего пингвиненка — оранжевые. Это сделано специально, чтобы пингвин был узнаваем.

Рис. 6. *Символ Linux*

Путь к первой версии (1.0)

Нумерация версий Linux началась с 0.01, и путь к стабильной версии 1.0 занял почти три года.

Версии ядра Linux до версии 1.0 *Таблица 1*

Версия	Дата выхода	Особенности версии
0.01	17.09.1991	Только ядро, включающее драйвер диска и несколько драйверов устройств. Программ не было. Практически система использоваться не могла
0.03	26.10.1991	В системе уже запускалась командная оболочка и компилятор C. Систему уже можно было использовать и писать в ней новые программы
0.10	12.1991	Система поддерживала AT-жесткий диск. Программы login не было — после загрузки ядра сразу загружалась оболочка bash.
0.11	12.1991	Уже что-то. Эта версия поддерживала FDD, многонациональные клавиатуры, мониторы VGA, EGA, Hercules. Самое главное, что системой начали пользоваться — ее загрузили и установили несколько человек
0.12	05.01.1992	Появилась подкачка — свопинг оперативной памяти на диск. Систему скачало несколько сотен человек. Это первая версия системы, которая распространялась по лицензии GPL
0.96	04.1992	Нет, я не пропустил несколько версий. Просто нумерация была изменена в связи с приближением к стабильной версии. Данная версия позволяла запускать систему X Window — у Linux появился графический интерфейс
0.99.14	12.1993	А вот здесь я пропустил определенное количество подверсий версии 0.99 — их описывать просто нет смысла
1.0	16.04.1994	Первая версия! С момента выхода версии 0.01 прошло 2 года и 7 месяцев

В апреле 1992 года Орест Збровски перенес на Linux систему X Window. Это был настоящий прорыв — но не из-за того, что у Linux появился графический интерфейс, а потому что обмен между X-клиентом и X-сервером осуществлялся с помощью UDS — UNIX Domain Sockets. Сокеты позволили использовать сеть в Linux. Система «повзрослела», что вдохновило Торвальдса на изменение нумерации версий. Вот что он сам пишет по этому поводу: «*мы с большим воодушевлением принялись разрабатывать сети для Linux поверх этих самых Domain Sockets, хотя они вовсе не были для этого предназначены. Я был настолько уверен, что все получится, что даже сделал скачок в нумерации версий. В марте 1992 года я планировал выпустить версию 0.13. А вместо этого, получив графический интерфейс пользователя, уверился, что мы на 95% достигли цели — выпуска полноценной, надежной операционной системы, пригодной к тому же для работы в сети. И поэтому выпустил версию 0.95.*»

Разработка сетевой подсистемы — дело довольно сложное. Именно поэтому у версии 0.99 так много подверсий. Версия 1.0 уже могла использоваться в сети.

Они создавали Linux

Линусу помогало много энтузистов, но на первых порах особенно важен вклад следующих добровольцев:

- Вернер Альмесбергер (Werner Almesberger) — разработал драйверы FDD и загрузчик Linux — LILO (LInux LOader);
- Теодор Тсо (Theodore Ts'o) — создал файловую систему ext2 (у меня эта файловая система использовалась до конца прошлого года!), библиотеки, распределитель памяти ядра;
- Дональд Беккер (Donald Becker) — занимался разработкой сетевых драйверов;
- Олаф Кирч (Olaf Kirch) — написал руководство по сетевому администрированию Linux;
- Питер Мак-Дональд (Peter McDonald) — разработал поддержку загружаемых модулей в версии 0.99. Правда, в современных версиях ядра этот механизм полностью переделан;
- Пол Гортмейкер (Paul Gortmaker) — разработал драйвер RTC (Real Time Clock), несколько сетевых драйверов (в т.ч. NE-2000), написал документы «Linux Ethernet HOWTO» и «Boot-Prompt HOWTO».

Первые дистрибутивы

Если системой заинтересовались, значит, ее нужно распространять. Сейчас Linux распространяется в виде так называемых дистрибутивов. Но первые версии распространялись по-другому. Версии 1991 года помещались на двух дискетах, копии которых можно было скачать с сервера университета в Хельсинки. Первая дискета была загрузочной — на ней было ядро. Вторая содержала корневую файловую систему и основные утилиты. Установка и конфигурирование первых версий системы было очень сложным занятием. Установить систему мог только эксперт в UNIX.

Чтобы упростить процесс установки и настройки системы, были созданы так называемые дистрибутивы. Попробую объяснить «на пальцах», что такое дистрибутив. Предположим, что у нас есть те две дискеты, содержащие ядро и файловую систему. Чтобы превратить «это» в дистрибутив, достаточно еще одной дискеты, на которой будет программа установки системы на компьютер пользователя. Программа установки поможет не только установить, но и настроить основные параметры системы. Современные дистрибутивы кроме самой системы и программы установки содержат еще и набор различных пользовательских программ.

Первые дистрибутивы появились в 1992 году, когда Линус выпустил ядро Linux по Стандартной Общественной лицензии GNU (GPL). Независимые разработчики (группы программистов) начали выпускать свои дистрибутивы Linux: они разрабатывали программы установки, программы

управления пакетами, прикладные программы. Ясно, что каждый дистрибутив выпускался под своим именем.

Первый дистрибутив, созданный в феврале 1992 года Оуэном Ле Бланк (Owen Le Blanc), назывался MCC Interim Linux (Manchester Computing Centre — Манчестерский компьютерный центр). Любой желающий мог загрузить этот дистрибутив с FTP-сервера MCC. Чуть позже сотрудники университета Техаса разработали дистрибутив TAMU.

А в октябре 1992 года появился дистрибутив Softlanding Linux System (SLS), созданный Питером Мак-Дональдом (Peter McDonald). Важность этого дистрибутива заключается даже не в том, что это первый дистрибутив, содержавший систему X Window и поддержку TCP/IP. SLS — это прямой предок всемирно известного дистрибутива Slackware.

«Размножение дистрибутивов»

Можно сказать, что современные дистрибутивы распространяются так же, как и первые дистрибутивы:

- На дискетах (только сейчас вместо дискет используются CD и DVD);
- На FTP-сайтах и досках BBS (правда, сейчас из-за большого размера на BBS дистрибутивы не выкладываются).

С удешевлением CD-ROM дистрибутивы начали распространяться на CD. Первыми компаниями, распространявшими дистрибутивы на CD, были: InfoMagic, Morse Telecommunication, Nascent, **Red Hat Software**, Trans-Ameritech, Walnut Creek и Yggdrasil Computing Inc. Один CD — намного удобнее двух-трех десятков дискет (средний размер первых дистрибутивов был около 50 Мб). К тому же на CD можно было записать много дополнительной информации: дополнительные программы, коды ядра и приложений, систему X Window, документацию и пр. Диск с дистрибутивом стоит от 20 до 40 долларов, а дистрибутив на дискетах — 20 долларов. Как видите, разница небольшая (если не считать стоимости односкоростного CD-ROM, который тогда стоил 100 долларов).

Первое, что вы должны помнить о версии дистрибутива — то, что версия дистрибутива почти никогда не совпадает с версией ядра. Например, взять версию дистрибутива Mandrake 10.1 — она содержит версию ядра 2.6. Точно также было и с первыми дистрибутивами: дистрибутив мог называться XXX 2.0 release, но содержать версию ядра 1.0, что создавало определенную путаницу.

О дистрибутивах можно говорить очень долго. Сегодня существует три основных дистрибутива: Red Hat, Slackware и Debian. Все остальные дистрибутивы являются производными от этих трех дистрибутивов. На наших

просторах более популярны RH-совместимые дистрибутивы (Mandrake, ALT Linux, ASP Linux), поэтому мы остановимся только на этих дистрибутивах, точнее только на дистрибутиве Red Hat, поскольку, если описывать историю каждого (или более или менее известного) дистрибутива, можно написать книгу толще, чем та, которую вы держите в руках.

Компания Red Hat была основана Марком Ивингом (Marc Ewing) в 1994 году, а в 1995 году была куплена компанией ACC Bookstores, принадлежавшей Бобу Янгу (Bob Young). Дистрибутивы от Red Hat пользовались большой популярностью, поскольку они были достаточно надежны (их смело можно было устанавливать даже на серверы), обладали удобной программой установки и удобными средствами конфигурирования системы. Red Hat — это самая большая коммерческая компания, которая занимается только ОС Linux.

Давайте рассмотрим этапы развития дистрибутивов Red Hat (источником информации послужил сайт компании Red Hat — http://fedora.redhat.com/about/history/).

История дистрибутивов Red Hat *Таблица 2*

Дата	Версия	Описание
29 июля 1994	Просто бета-версия	Первая бета-версия, основанная на ядре версии 1.1.18 и системе управления пакетами RPP (разработка Red hat). Данная версия поставлялась на CD, на котором кроме всего прочего поставлялась полная документация к этой системе. Не получила широкого распространения. Данная версия называлась «Red Hat Software Linux»
31 октября 1994	RHL 0.9 (Halloween)	Первая бета-версия, получившая широкое распространение. Пользователи могли выбрать два ядра — стабильное (1.0.9) или экспериментальное (1.1.54). В этой версии появился графический интерфейс к программе rpp — LIM (Linux Installation Manager). Кроме LIM в этой версии появились графические программы настройки ОС: управления пользователями и группами, файлом fstab и сетью
май 1995	RHL 1.0 (Mother's Day)	Первый релиз (не бета-версия). Построена на ядре 1.2.8. В документации она называлась не «Red Hat Software Linux», а «Red Hat Commercial Linux». Очевидно, изменение название произошло после покупки компании Red Hat фирмой ACC Bookstores. Логотип этой версии содержал «красную шляпу» — на логотипе был изображен мужчина, несущий в одной руке красную шляпу, а в другой — портфель
Август 1995	RHL 2.0 beta	Отличительная особенность этой версии — поддержка формата исполнимых файлов ELF (Executable Linkage Format), до этого использовался формат "a.out". Система управления пакетами RPP была заменена системой RPM, в результате чего была полностью не совместима с предыдущими версиями
20 сентября 1995	RHL 2.0	То же самое, но не бета-версия. Дистрибутив использует систему RPM

Продолжение табл. 2

Дата	Версия	Описание
23 ноября 1995	RHL 2.1	В этой версии были исправлены некоторые ошибки предыдущей. На основе этой версии компания Digital создала диск «Red Hat 2.1 LINUX» (для x86-компьютеров), который послужил основой для создания RH для платформы Alpha («Red Hat Linux/Alpha 2.1»)
Март 15 1996	RHL 3.0.3 (Picasso)	Первая многоплатформенная версия. Поддерживались архитектуры x86 и Alpha. Для Альфы использовался формат исполнимых файлов a.out, а для x86 — ELF. В этой версии появился X-сервер Metro-X, утилита настройки принтера и оболочка glint для программы rpm.
июль-август 1996	RHL 3.0.4/3.95 (Rembrandt)	Система RPM переписана на C (до этого она была написана на Perl). Появились новые средства конфигурирования, модуль PAM (Pluggable Authentication Modules). Благодаря переходу на ядро версии 2.0 появилась возможность использовать модули ядра, а до этого на дисках дистрибутива RH поставлялось 72 варианта ядер, из которых пользователь должен был выбрать наиболее подходящее его «железу»
3 октября 1996	RHL 4.0 (Colgate)	Ядро — 2.0.18. Поддерживаются уже три архитектуры: x86, Alpha и SPARC. Впервые на Alpha используется формат ELF. Изменен логотип дистрибутива, который используется до сих пор — $$$intro.tif
3 февраля 1997	RHL 4.1 (Vanderbilt)	Исправляла ошибки предыдущей версии
19 мая 1997	RHL 4.2 (Biltmore)	Отличалась использованием стабильной, хотя и устаревшей, версии libc 5.3. Использование устаревшей версии себя оправдало: в версии 5.4 обнаружилось очень много ошибок
27 августа — 16 сентября 1997	RHL 4.8/4.8.1/4.95 (Thunderbird)	Использовалась библиотека glibc 2.0.
7 — 16 октября 1997	RHL 4.9/4.9.1/4.96 (Mustang)	Исправлены ошибки, связанные с переходом на другую версию C-библиотек (с libc на glibc)
1 декабря 1997	RHL 5.0 (Hurricane)	Этот дистрибутив включал программу резервного копирования BRU2000-PE™ и клиент для Real Audio ™
1 июня 1998	RHL 5.1 (Manhattan)	Особенности этого дистрибутива: графическая среда GNOME (но не устанавливалась по умолчанию), единая утилита конфигурации — linuxconf, браузер Netscape (до этого использовался браузер Red Baron, содержавший много ошибок).
12 октября 1998	RHL 5.2 (Apollo)	–
17 марта 1999	RHL 5.9 (Starbuck)	–
19 апреля 1999	RHL 6.0 (Hedwig)	Среда GNOME наконец-то интегрирована в дистрибутив. Ядро 2.2, glibs 2.1. *Это первый мой RH-дистрибутив. Запомнился как отличный стабильный дистрибутив. У меня до сих пор есть компакт-диск с этим дистрибутивом.*
6 сентября 1999	RHL 6.0.50 (Lorax)	Новый инсталлятор системы anaconda мог работать как в текстовом, так и графическом режимах.
4 октября 1999	RHL 6.1 (Cartman)	–
27 марта 2000	RHL 6.2 (Zoot)	ISO-образы этой версии были доступны на FTP-сервере Red Hat
25 сентября 2000	RHL 7.0 (Guinness)	В составе дистрибутива была новая версия библиотеки glibc 2.2 и последняя версия gcc 2.96. Включение версии gcc 2.96 не было согласовано с разработчиками gcc, что вызвало небольшой конфликт между ними и компанией Red Hat

Продолжение табл. 2

Дата	Версия	Описание
31 января 2001	RHL 7.0.90 (Fisher)	Первая версия, использующая ядро 2.4
16 апреля 2001	RHL 7.1 (Seawolf)	Впервые появилась поддержка большого количества языков (дистрибутив поддерживал даже китайский и японский языки). Вместе с Netscape поставлялся браузер Mozilla
2-21 августа 2001	RHL 7.1.93, 7.1.94 (Roswell)	По умолчанию использовалась новая версия файловой системы — ext3 вместо ext2. А вместо LILO начал использоваться загрузчик GRUB. Хотя при установке можно было выбрать LILO, если GRUB пользователю не нравился
22 октября 2001	RHL 7.2 (Enigma)	Довольно неплохая версия, она «прожила» на моем компьютере до версии 7.3. Правда, в ней были некоторые ошибки, которые были исправлены в версии 7.3. Особенности: GNOME 1.4, KDE 2.2. Послужила основой для дистрибутива Red Hat Enterprise Linux 2.1 AS (Advanced Server).
22 марта 2002	RHL 7.2.91 (Skipjack)	-
6 мая 2002	RHL 7.3 (Valhalla)	Выход версии 7.3 не планировался — сразу должна была выйти версия 8.0. Поэтому версию 7.3 нужно рассматривать как «переходную» версию. В версии 8.0 должны быть gcc3, GTK+ 2, Python 2 и другие новые версии инструментальных средств. Но они «опаздывали» к запланированному сроку выпуска версии 8.0, поэтому было решено выпуск версии 8.0 немного отложить, а вместо нее выпустить версию 7.3 без всех этих нововведений. Но в версии 7.3 все же были исправлены некоторые ошибки версии 7.2. Я лично работал с этим дистрибутивом, и он мне понравился даже больше, чем 8.0. Это последний дистрибутив, содержавший браузер Netscape
6 мая 2002	RHEL 2.1 AS (Pensacola)	Red Hat Enterprise Linux 2.1 AS — дистрибутив, предназначенный для корпоративного применения. Его основа — дистрибутив RH 7.2
30 сентября 2002	RHL 8.0 (Psyche)	В этой версии были gcc 3.2, glibc 2.3, GNOME 2, KDE 3.0.3 и OpenOffice.org 1.0.1
31 марта 2003	RHL 9 (Shrike)	Основан на ядре 2.4.20 с поддержкой NPTL (Native POSIX Thread Library). Послужил основой для Red Hat Enterprise Linux 3
21 июля 2003	RHL 9.0.93 (Severn)	Это последняя версия «Красной Шапочки» — после этого дистрибутивы стали называться Fedora Core (для настольных систем) и RHEL (для корпоративного применения). Fedora Core — это открытый (читайте — бесплатный) проект, а RHEL — коммерческий проект от RH
25 сентября 2003	FC 0.94 (Severn)	Пробная версия Fedora Core
13 октября 2003	FC 0.95 (Severn)	Первый дистрибутив от RH, использующий репозиторий yum для обновления системы
22 октября 2003	RHEL 3 (Taroon)	Red Hat Enterprise Linux 3 поддерживал одновременно 7 архитектур: Intel X86, Intel Itanium, AMD AMD64, IBM zSeries, IBM iSeries, IBM pSeries и IBM S/390. Основан на ядре 2.4.21
5 ноября 2003	FC 1 (Yarrow)	Первый релиз Fedora Core, используется пока что ядро 2.4 — в последний раз
12 февраля; 29 марта; 27 апреля 2004	FC 1.90, 1.91, 1.92	Первые версии, использующее ядро 2.6. Поддерживаются архитектуры x86 и x86-64
5 марта 2004	FC 1 (Yarrow)	Fedora Core 1 для x86-64

Продолжение табл. 2

Дата	Версия	Описание
18 мая 2004	FC 2 (Tettnang)	Второй релиз Fedora Core; KDE 3.2, GNOME 2.6
8 ноября 2004	FC 3 (Heidelberg)	Третий релиз Fedora Core. GNOME 2.8 и KDE 3.3. Довольно «глючное» создание, до недавнего времени обитавшее на моем компьютере
13 июня 2005	FC 4 (Stentz)	Четвертый релиз Fedora Core. GNOME 2.10 и KDE 3.4. Довольно много нововведений

Дистрибутив Red Hat Linux имеет множество потомков — производных дистрибутивов, которые основаны на RH и практически полностью с ним совместимы. Самыми известными потомками RH являются:

- Linux Mandrake (версия MDK 5.1 была основана на RH 5.1) — а на MDK основан ALT Linux;
- Black Cat (Версия BC 5.2 основана на RH 5.2) — впоследствии BC перерос в ASP Linux.

Выбор дистрибутива

Итак, дистрибутив — это ядро + системные утилиты + заранее подготовленный пакет программного обеспечения, снабженный удобной программой-инсталлятором.

Строго говоря, название Linux принадлежит только ядру. Другие компоненты ОС Linux и прикладные программы для нее разрабатываются не какой-то одной компанией, а независимыми группами программистов, работающих на условиях Стандартной Общественной Лицензии (General Public License; ее русский перевод можно прочитать по адресу `http://rus-linux.net/MyLDP/histori/gpl/gplrus.html`). Эта лицензия обязывает ее держателя бесплатно предоставлять исходный код распространяемого им программного обеспечения, так что теоретически каждый может легально скачать с публичного сервера тексты программ, скомпилировать и собрать их на своем компьютере и получить готовую к установке операционную систему.

Если вы решитесь собрать Linux с нуля, то прочитать руководство можно по адресу `www.linuxfromscratch.org` (русский перевод: `multilinux.sakh.com`). Если нет — то покупайте дистрибутив. Необходимость платить деньги вовсе не противоречит некоммерческой идеологии Linux: поставщик дистрибутива продает не код, а только услуги по его записи на носитель, доставке, установке, технической поддержке и т.п.

На сегодня существуют и поддерживаются разработчиками сотни разных дистрибутивов, различающихся по области применения, версии

ядра, составу включенных в них прикладных программ, требованиям к аппаратному обеспечению и другим признакам. Классификации этого многообразия посвящены статьи В.Костромина и А.Федорчука в Библиотеке ЛинуксЦентра по адресу `www.linuxcenter.ru/lib/articles/distrib`.

Можно с уверенностью утверждать, что каждый начинающий пользователь Linux столкнется или уже столкнулся с проблемой выбора среди такого изобилия. Какой дистрибутив лучше? Чем отличается Red Hat от Mandrake? Некоторых начинающих пользователей этой замечательной ОС вводят в заблуждение имена дистрибутивов, например, LinuxXP. Вот они его и покупают, а потом полностью разочаровываются в Linux. Не скажу, что LinuxXP очень плохой дистрибутив, но у начинающего пользователя будет меньше проблем с полной, а не облегченной, версией какого-нибудь «серьезного» дистрибутива, например, Linux Mandrake или Fedora Core.

Сейчас я попытаюсь помочь читателю выбрать дистрибутив, с которого можно начать свое знакомство с Linux.

Прежде всего нужно определиться, для чего вам нужен Linux. Хотите ли вы глубоко изучать эту операционную систему? Или, быстро освоив ее, заменить ею Windows в своем офисе, чтобы избежать визитов «борцов с компьютерным пиратством»? Или, может быть, вам нужно настроить сервер для выхода вашей локальной сети в Интернет? Многие компании выпускают дистрибутивы, адаптированные под любое из указанных применений.

Следующий вопрос — совместимость с имеющимся у вас оборудованием: установится ли выбранный дистрибутив на ваш компьютер? А если установится, то не откажется ли работать, например, с модемом? Не секрет, что долгое время разрыв по количеству поддерживаемых устройств между Windows и Linux был попросту вопиющим. Сегодня ситуация значительно выровнялась. Нынешние дистрибутивы неплохо распознают современное оборудование — процессоры, чипсеты, IDE-, SCSI- и USB-устройства. Практически наверняка не возникнет проблем даже с TV-тюнерами и приводами CD-RW.

Правда, почти у всех дистрибутивов не сложились отношения с Windows-модемами. Ситуация обусловлена тем, что производители не спешат предоставить информацию по своим продуктам для создания Linux-драйверов. Первая попытка поддержки Windows-модема была сделана компанией ALT Linux, на сайте которой есть пакет hsf, обеспечивающий работу устройств на чипе Connexant. В Интернете можно также поискать драйверы для других модемов, скажем, Motorola.

Примерно то же самое относится и к GDI-принтерам. Их производители неохотно поддерживают Linux, во многих случаях приходится надеяться

исключительно на энтузиастов. Относительно благополучно дела обстоят разве что с оборудованием компании Oki.

В целом же ситуацию с аппаратным обеспечением можно подытожить таким образом: со «средним» оборудованием, не слишком старым и не самым новым, существенных проблем возникнуть не должно. Для новейших, дорогих и «навороченных» устройств могут поддерживаться лишь их базовые функции, то есть деньги, заплаченные вами за усовершенствования, окажутся потрачены впустую. Поэтому стоит предварительно изучить список совместимых устройств и базовую комплектацию (как минимум версию ядра и графической системы XFree86) каждого конкретного дистрибутива. Обычно эту информацию легко найти на сайте разработчика: например, для Red Hat это `hardware.redhat.com/hcl`.

Что касается минимальных системных требований, то малыми ресурсами могут обойтись лишь дистрибутивы, предназначенные к установке на компьютеры, обслуживающие сеть (шлюзы, интернет-серверы), которые в графическом режиме работать не должны. Для дистрибутивов же, ориентированных на домашнее или офисное применение, официальные данные (как и в случае с Windows) занижаются беззастенчиво. Да, на машине с Pentium 133 и 32 MB RAM можно запустить Linux, но не работать. Pentium 200 MMX и 64 MB больше похожи на правду, но если вы планируете использовать оконную среду KDE 3, то и 128 MB памяти не окажутся лишними. В противном случае применяйте менее ресурсоемкую среду GNOME. Дополнительно нужно учитывать потребности прикладных программ, которые сами по себе могут быть весьма немаленькими.

Одно из правил, которое следует всегда помнить: ОС Linux гораздо более требовательна к объему оперативной памяти, чем к частоте процессора: Pentium III 600 МГц/32 Мб будет работать гораздо медленнее, чем Celeron 400 МГц/64 Мб.

Следует позаботиться и о видеосистеме — она должна обеспечивать (как минимум) комфортную работу с разрешением 1024x768. Дело в том, многие Linux-приложения проектировались исключительно для данного режима, поэтому при использовании 800x600 могут возникнуть затруднения (часть окна просто не будет видна).

Потребности в дисковом пространстве вполне стандартны по нынешним временам. Минимальная конфигурация требует 300–500 Мбайт, однако новичкам рекомендуется сразу устанавливать определенный набор прикладных программ, для которого требуется около 1,5 Гбайт. К тому же нужно оставить место для документов и рабочих файлов, так что в качестве отправной точки вполне подойдут два гигабайта.

Я перечислю несколько дистрибутивов, которые купить можно, но сначала скажу о тех, которые покупать категорически не советую. Не нужно

покупать экзотические дистрибутивы вроде LinuxXP и Lindows (он же Linspire). Правда, последний дистрибутив — редкость в интернет-магазине, но если вы его все-таки найдете, то не вздумайте покупать. Не верите мне? Тогда купите его. Вы поймете, что даром потратили свои деньги и время.

В некоторых интернет-магазинах дистрибутивы Linux ставятся в один ряд с дистрибутивами FreeBSD. Так вот, FreeBSD покупать не стоит — во всяком случае, если вы не хотите купить еще одну книгу, но только по FreeBSD. Да, FreeBSD является дальним родственником Linux, но начинающему Linux-пользователю лучше с FreeBSD не связываться.

Вы смело можете покупать Linux Mandriva (в прошлом — Linux Mandrake) и Fedora Core 4 (в прошлом — Linux Red Hat), а также дистрибутивы, так или иначе основанные на них — это ALT Linux 2.4, ASP Linux 10. Также смело можете купить не очень новую, но проверенную версию Linux Mandrake 10 — очень хороший дистрибутив.

Вы можете купить дистрибутив Knoppix, но не как основной дистрибутив, а как вспомогательный, который вы сможете использовать как средство восстановления системы, когда в очередной раз вы переустановите Windows или же после неудачного эксперимента с загрузчиком Linux перестанет загружаться. Особенность Knoppix в том, что он может загружаться и нормально работать с компакт-диска.

Дистрибутивы Debian и Slackware Linux — очень хорошие, стабильные и проверенные временем, но вам их лучше не покупать, поскольку они основаны на BSD-системе инициализации, которая ближе к миру BSD и в этой книге подробно не рассматривается. Конечно, если вы можете себе позволить еще одну книгу — по Slackware — то можете его купить. Но все-таки Slackware несколько сложнее в освоении, поэтому я рекомендую попробовать этот дистрибутив после того, как вы научитесь работать с Mandrake или Fedora Core. То же самое относится и к SuSe — немецкому дистрибутиву со своими, немецкими, особенностями. Лично мне больше нравятся французский дистрибутив Mandrake.

Дистрибутив Gentoo, о котором много говорили определенное время назад, хорош тем, что позволяет настроить систему «под себя» в прямом смысле этого слова. Все устанавливаемые программы компилируются на вашем компьютере, а не устанавливаются из RPM-пакетов, содержащих уже откомпилированные на *чужой* машине двоичные файлы. Это главное достоинство этого дистрибутива, но оно же является и его главным недостатком — Gentoo начинающим пользователям противопоказан.

Покупать остальные дистрибутивы нужно только в том случае, когда вы точно знаете, что это за дистрибутив и зачем он вам нужен.

Теперь давайте рассмотрим дистрибутивы Linux Mandrake, Fedora Core 3 и Slackware Linux поближе. Первые два рассматриваются, чтобы у вас была возможность сравнить, а Slackware — чтобы вы поняли, почему его вам не нужно покупать (во всяком случае — пока).

Смотреть на дистрибутивы я буду глазами начинающего пользователя, поэтому не удивляйтесь, что я не буду приводить рекомендации по исправлению той или иной ошибки — начинающий пользователь не знает, как ее исправить. Во время установки и работы с системой буду проводить небольшие тесты: а именно, сколько времени занимает процесс установки, сколько — загрузка системы, копирование файла и т.д. Время буду измерять не какой-нибудь программой, а обычным секундомером, встроенным в мобильник — это же домашние условия. Все дистрибутивы тестировались на следующей рабочей лошадке: Duron 1,6 GHz/256MB/HDD 40 GB 7200 rpm Maxtor/128 MB GeForceFX.

Итак, начнем наш обзор. И начнем его с классики, то есть с дистрибутива Fedora Core.

Fedora Core 3

Установка

Дистрибутив Fedora Core 3 поставляется на четырех компакт-дисках, но программа установки, если не отмечать какие-нибудь дополнительные пакеты, а оставить все как есть, требует только первые три компакт-диска.

Как обычно, загружаюсь с компакт диска, нажимаю Enter для начала установки и одновременно (практически одновременно, сотые учитывать не буду) запускаю секундомер.

Программа установки спросила меня, хочу ли я протестировать установочные диски. Как начинающий и ничего не понимающий в установке Linux пользователь, я согласился. Да мне и не хотелось «убивать» нормально работающий и уже установленный Linux Mandrake в случае, если что-то с компакта не скопируется, ведь FC3 должен быть установлен на его место.

Во время установки я не отхожу от компьютера, чтобы оперативно отвечать на запросы программы-инсталлятора. Проверка первого диска не заняла много времени, а от проверки остальных дисков я отказался, нажав кнопку Continue.

Программа установки очень удобна и проста — даже у начинающего пользователя не будет с ней проблем. Выбираю Выборочную установку и ручное разбиение дисков. Даже если вы — начинающий пользователь, я настоятельно рекомендую сделать то же. Ведь если выбрать другой тип

установки, ваш винчестер будет переразбит, что приведет к потере всех ваших данных и установленной Windows вместе с ними. Здесь у меня будет небольшая погрешность в измерениях — ведь у меня уже есть созданные Linux-разделы, я только изменил тип файловой системы с ext2 на ext3 и выбрал форматирование этих разделов. Если у вас никогда еще не был установлен Linux, вам понадобится дополнительное время на создание разделов и изменения границ уже имеющихся разделов.

Параметры загрузки — по умолчанию, чтобы не тратить дополнительное время на их установку.

Следующий шаг — выбор программного обеспечения. Ведь я — начинающий пользователь, поэтому не знаю, что и для чего используется. Просто оставляю все как есть. Как пользователь понимающий, нужно заметить, что FC3 по умолчанию использует среду GNOME, а KDE вообще даже не предлагается устанавливать.

Появляется окно «Запуск процесса установки ...» и я перезапускаю секундомер. До этого весь подготовительный этап занял 7 минут и 31 секунду (7:31).

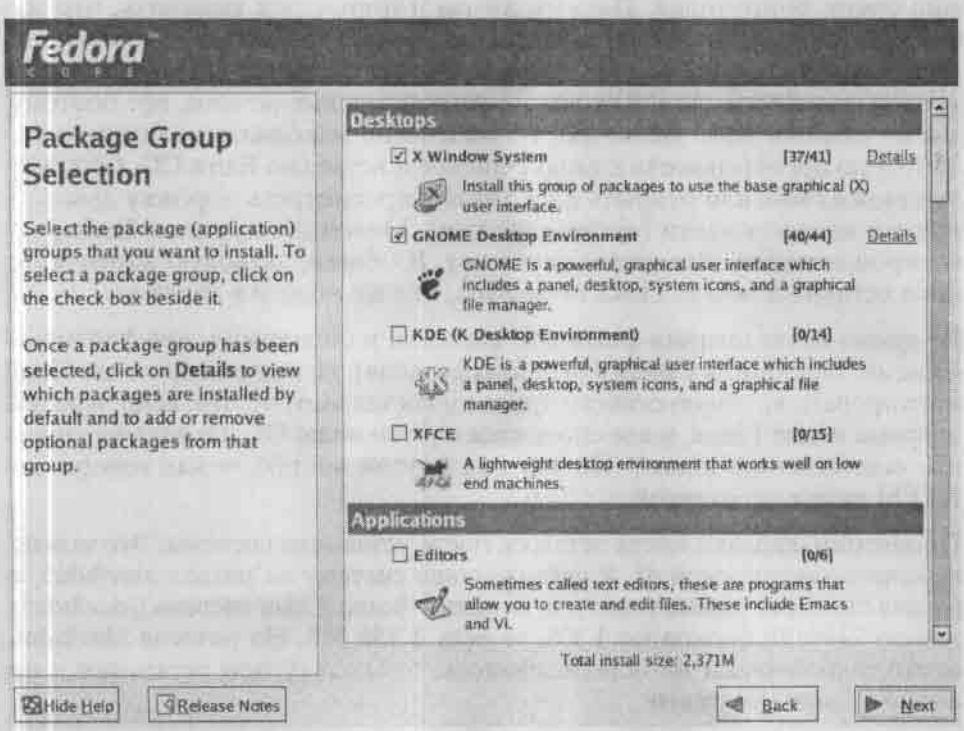


Рис. 7. *Установка Fedora Core 3*

Началось копирование пакетов на мой жесткий диск. Таймер программы-инсталлятора при копировании бессовестно врал: показывал 15 минут до вставки второго диска, а после этого момента стал показывать 10 минут, хотя прошло больше чем 5 минут. Четвертый диск в моем случае (все пакеты — по умолчанию) не понадобился. Установка (до появления окна перезагрузки компьютера) заняла 15 минут и 38 секунд (15:38).

После первой перезагрузки система попросила указать некоторые параметры (дату/время, параметры дисплея), добавить новых пользователей, настроить звуковую плату и т.д. Сразу скажу, моя встроенная звуковая плата отказалась работать. Плата была опознана, но звука при воспроизведении семпла я так и не услышал. Пока оставил все как есть.

Вторая перезагрузка

Сразу перезагружаю систему. Запускаю секундомер (с момента начала загрузки ядра). Запуск системы (до окна ввода имени пользователя и пароля) занял на моей машинке всего 1 минуты и 8 секунд. Запуск GNOME занял еще 7 секунд.

Начинаю работать с системой. Обновленный GNOME (версия 2.8.0) мне очень понравился. Первым делом я попытался выяснить, что же все-таки случилось с моей звуковой платой. Поскольку я начинающий пользователь, на ум мне не пришло ничего другого, кроме проверки уровня громкости. Ааа! Так все же устройства выключены, вот поэтому я и не слышал звук! Включаю, устанавливаю максимальную громкость. Запускаю проигрыватель компакт-дисков и вставляю Extra CD. GNOME предложил мне или открыть диск (чтобы просмотреть дорожку данных) или же воспроизвести (аудио-дорожки). Мелочь, а приятно. Выбираю воспроизведение. Все равно звука нет. В общем, звуковую карточку я пока оставил в покое. Пока посмотрю, что же нового в системе.

Во время исследования файловой системы, я обнаружил, что файловые системы Windows (а у меня 4 Windows-раздела), не подмонтированы. Подмонтировать вручную особого труда не составляет, но что, если человек впервые видит Linux, и еле справился с установкой ОС, он ведь не знает, что есть команда mount! Но это уже другой вопрос — как говорится, RTFM перед установкой.

Посмотрим, сколько места осталось после установки системы. Это можно выяснить программой df. Я устанавливал систему на раздел /dev/hda5, а раздел /dev/hda6 использовал для каталога /home. Сама система (/dev/hda5) заняла 2446188 блоков по 1 Кб, то есть **2 388** Мб. На разделе /dev/hda6 место практически не использовалось, поскольку при установке я не создавал пользователей.

Программное обеспечение

По умолчанию устанавливается не все программное обеспечение, а только самое необходимое. Очень понравился удобный просмотрщик изображений Photo Tool (рис. 8).

Также понравился обновленный GIMP 2.0.5, хотя с ним делал только скриншоты, его интерфейс стал более красивым и удобным (рис. 9).

Хочу просмотреть видео. Пусть и без звука. Запускаю Helix Player — именно этот проигрыватель используется в FC3 для просмотра видео. Открываю файл, который без проблем воспроизводился в MDK, и вижу картину "The player does not have the capabilities to play back this content" Больше комментариев нет.

Далее все как обычно — Open Office, Project Planner, Диаграммы DIA, утилиты настройки системы. Стоп! Кажется, в группе Интернет есть что-то новенькое — Firefox Web Browser. Запускаю. Как следует из названия, данный браузер основан на Mozilla. Только почему-то на английском.

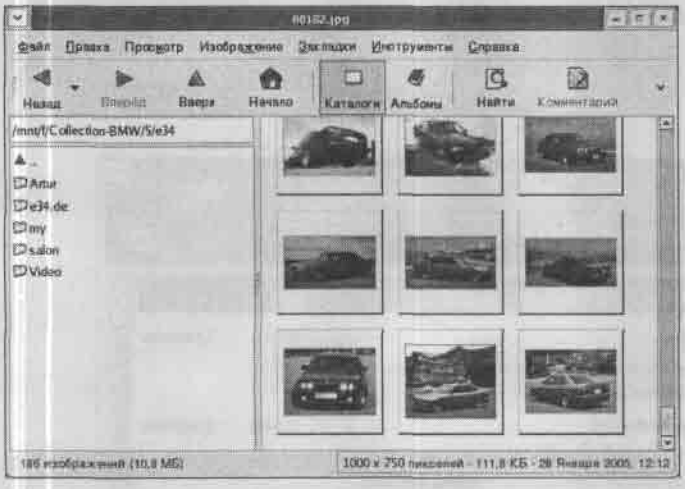


Рис. 8. *Photo Tool*

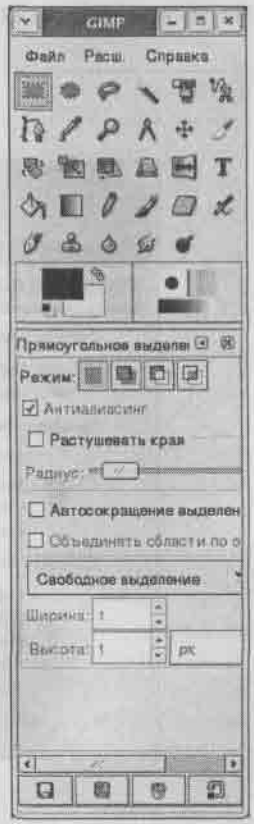


Рис. 9. *Панель инструментов The GIMP 2.0.5*

Исследовав опции, я так и не нашел выбора языка интерфейса, скорее всего, для его русификации нужно доустановить какой-то пакет.

Тесты

Переходим к самому интересному — тестам. Тесты будут следующими: тест чтения блока данных с помощью hdparm, тест на время копирования большого файла с раздела FAT32 в раздел ext3 и копирование этого же файла из одного раздела ext3 в другой раздел ext3. Хотел запустить привычный мне Midnight Commander, но его я не нашел и попытался установить с помощью утилиты «Управление пакетами». Данная утилита мне не понравилась. Она отлично подходит для начинающего пользователя, который и сам не знает, что он хочет установить. Мне же лучше было ввести имя пакета и установить его. К сожалению, в этой программе данной возможности не было.

Тратить время на установку MC я не стал, а решил воспользоваться Браузером файлов. Может это и правильнее, ведь начинающий пользователь будет использовать именно его, а не какой-то MC, который даже не устанавливается по умолчанию.

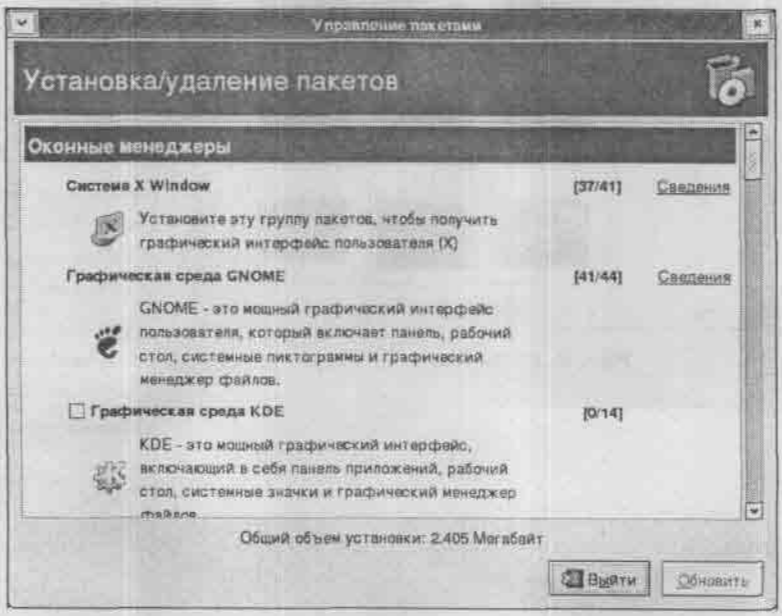


Рис. 10. *Управление пакетами*

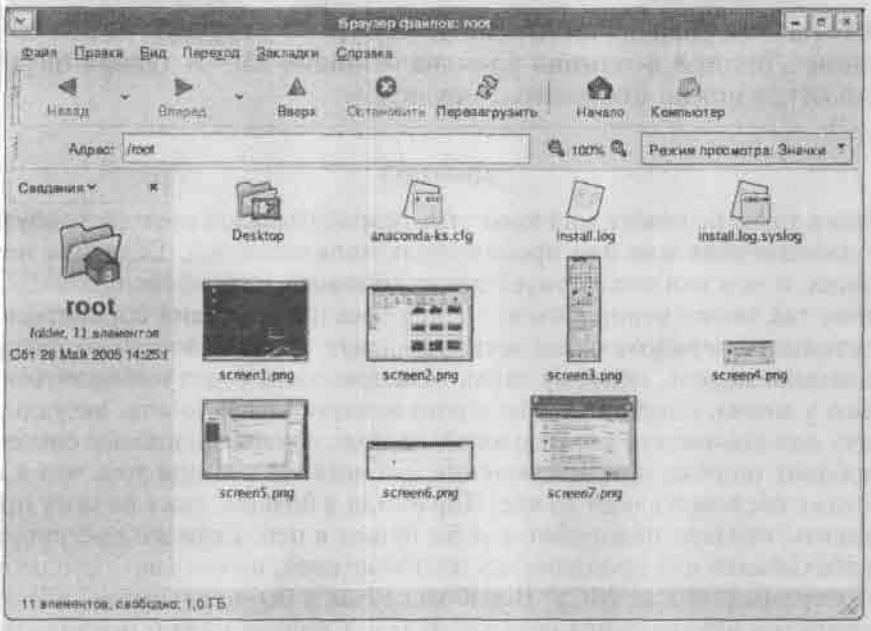


Рис. 11. *Браузер файлов*

У нас есть уже все необходимые программы для проведения тестов. Начнем по порядку — с hdparm. Запускаю Терминал (находится в меню Приложения→Система). Результат чтения блока размером в 174 Мб за 3,02 сек довольно неплохой — это около 57,68 Мб/с.

Теперь буду копировать файл размером ровно 700 Мб. Это фильм, который мне так и не удалось просмотреть. Получил такие результаты:

- Копирование с раздела FAT32 (/dev/hda10) на раздел ext3 (/dev/hda6): 2 мин 32 сек.
- Копирование с раздела ext3 (/dev/hda6) на раздел ext3 (/dev/hda5): 1 мин 04 сек.

Как видите, с «родными» разделами FC3 работает в два раза быстрее. Файловая система ext3 использовалась только потому, что при нормальной установке она предлагается по умолчанию — ее и выберет начинающий пользователь. Он же и не подозревает о наличии других файловых систем.

Все, обзор FC3 на этом заканчивается, и я решил выключить компьютер. При выключении вижу картину из серии «Не ждали»:

```
Power down
acpi_power_off called
```

И все. Так машинка простояла минут пять, пока я вручную не выключил питание. Лично я вспомнил времена Windows 95: «А теперь питание компьютера можно отключить... вручную».

Диагноз

Лично я так и не понял, для кого этот разрабатывался этот дистрибутив: для начинающих или для продвинутых пользователей. Если для начинающих, о чем свидетельствует дружественный интерфейс GNOME, то почему так много недоработок, с которыми начинающий справиться не в состоянии: неработающая звуковая плата (у меня довольно распространенная модель, поэтому такая неисправность будет наблюдаться не только у меня), ошибки при воспроизведении видео (очень актуальная задача для домашнего компьютера), не подключены файловые системы, а вдобавок ошибка при выключении питания? И это при том, что я поработал с системой какой-то час. Поработай я больше, даже не могу предположить, сколько недоработок я бы нашел в ней. Если же дистрибутив разрабатывался для продвинутых пользователей, почему нет привычных им программ (того же MC)? В любом случае я бы не рекомендовал этот дистрибутив начинающим пользователям. Сначала нужно поучиться на более легком дистрибутиве, у которого с самого начала все работает, а затем переходиться на Fedora Core 3, если, конечно, захочется.

Fedora Core 4, к сожалению, попробовать не получилось, привожу только версии программного обеспечения, входящего в этот дистрибутив.

Преимущества	Недостатки
Дружественный интерфейс GNOME 2.8.0	Проблемы с определением звуковой карты
Хороший выбор программного обеспечения	Не хватает видео-кодеков
Удобная программа установки	Проблемы с выключением питания
Новые версии GNOME и KDE	По умолчанию не устанавливаются некоторые привычные программы
	По умолчанию не подмонтируются файловые системы Windows
	Требует самостоятельной донастройки после установки
Программное обеспечение	
Fedora Core 3 Ядро версия 2.6.9 GNOME 2.8.0 KDE 3.3.0 Open Office 1.1.2 GIMP 2.0.5 Загрузчик GRUB	Fedora Core 4 (**) Ядро версия 2.6.11 GNOME 2.8.3 KDE 3.4.0 Open Office 1.9.104 GIMP 2.2.7 Загрузчик GRUB

Slackware

Вторым дистрибутивом я решил выбрать Slackware. Просто хотелось на него посмотреть: ведь до этого я видел его в работе только несколько раз. А тут еще и последняя на данный момент версия — 10.1. Дистрибутив поставляется на двух дисках.

Установка

Как обычно, загружаюсь с первого компакт-диска. Вижу экран приветствия:

```
Welcome to Slackware version 10.1 (Linux kernel 2.4.29)
```

Что? Неужели последний дистрибутив от Slackware основан на старом ядре? Возможно, разработчики посчитали, что новое им не нужно.

Так и есть: в состав Slackware и Debian включаются только старые добрые версии программного обеспечения, в том числе и ядра. Так разработчики пытаются создать стабильный дистрибутив, чтобы в нем не было различного рода недоработок.

Знакомая загрузка ядра предыдущей версии. Программа установки предложила выбрать раскладку клавиатуры, отличную от US:

```
Enter 1 to select a keyboard map
```

Неужели установка системы будет в текстовом режиме? Да еще и без меню. Ага, нет, меню все-таки появилось, но текстовое. Меню выбора раскладки клавиатуры содержит список файлов (!) раскладок безо всякого описания. Догадайся, мол, сам. Выбираю qwerty/ru.map и нажимаю Enter. Честно говоря, текстовая программа установки выглядит как-то архаично. Такое впечатление, что устанавливается одна из первых версий Red Hat. Уже в 6.0 версии Red Hat программа установки была намного удобнее, хотя она тоже работала в текстовом режиме. Затем система предложила ввести пароль пользователя root. На данный момент пароля нет, поэтому можно просто нажать Enter, что я и сделал. Система сообщила, что для изменения таблицы разделов можно ввести команду fdisk. Мне повезло — Linux-разделы у меня уже есть, а начинающему пользователю лучше вообще отложить эту затею и попросить кого-нибудь более знающего переразбить жесткий диск за него. Только на использование текстовой версии fdisk может уйти минут пять — это при условии, если знаешь, что делаешь. К тому же fdisk не умеет изменять размеры разделов без уничтожения всей информации. Поэтому лучше использовать для этой цели какую-нибудь другую программу, например, тот же Partition Magic для Windows. Да, есть в Slackware более удобная программа — cfdisk, но в программе установке о ней ни слова...

Для начала установки нужно ввести команду setup. Ввожу. Появляется меню программы установки. KEYMAP я уже выбрал, поэтому выбираю второй пункт ADDSWAP. Он позволяет установить раздел подкачки для вашей системы. У меня уже создан, поэтому программа запросила разрешения только активизировать этот раздел. После этого появилось текстовое окошко FORMAT, в котором нужно выбрать, какие Linux-разделы нужно форматировать, а также файловую систему. По умолчанию предлагается Reiser FS, но для равенства условий я решил выбрать ext3, чтобы результаты файлового тестирования были более объективными. Программа установки работает только на английском языке, поэтому если не знаете его, лучше вам попытаться установить другой дистрибутив.

Затем программа спросила, хочу ли я добавить найденные Windows-разделы в /etc/fstab. Соглашаюсь и добавляю 2 из четырех разделов — для экономии времени. После этого нужно выбрать источник установки, выбираю Slackware CD or DVD, а также опцию auto для автоматического определения диска.

Какие пакеты устанавливать? Выбираю full, то есть установку всех пакетов, чтобы не терять время на их выбор. До этого весь подготовительный процесс занял 6 минут и 6 секунд. У вас, скорее всего, этот показатель будет другим. А если у вас еще нет Linux-разделов, то можно смело умножить это время на 2.

Время копирования пакетов с двух компактов заняло 13 минут и 10 секунд. После копирования пакетов программа установки настраивает шрифты, документацию и выполняет другие действия. На все про все ушло 53 секунды.

Вот мы и добрались до самого интересного. Программа установки не знает, откуда брать ядро. Она предлагает несколько вариантов: bootdisk, Slackware CD и др. Причем первый — по умолчанию. Я его и выбрал, просто механически так получилось. Программа попросила вставить загрузочную дискету в дисковод /dev/fd0. У меня не только нет этой дискеты, но и самого дисковода. Отказаться — никак. Пришлось, не долго думая, нажать Reset и повторить весь процесс заново. Правильный вариант — Slackware CD. Затем — список ядер. Какое устанавливать? Я выбрал ядро, которое использовалось при установке:

```
/cdrom/kernel/bare.i/bzImage
```

Следующий этап — просто анекдотический. Программа хочет создать загрузочный диск, который она запрашивала на предыдущем этапе. Во как. Я отказался от этой возможности. После этого я также отказался от использования HOTPLUG-устройств. Для экономии времени.

Настройку загрузчика LILO я производил вручную. Не полагаясь на интеллектуальные возможности программы, я выбрал expert и ввел все

параметры вручную. LILO установил в MBR. Затем добавил две записи: Linux (/dev/hda5) и Win (/dev/hda1). Обычно эти параметры указываются автоматически — но это в нормальных программах установки... Для инсталляции LILO выбрал команду меню Install LILO.

После этого нужно указать различные параметры системы — тип мыши, параметры сети (я их не устанавливал), выбор запускаемых сервисов (оставил как есть). Потом программа спросила, хочу ли я добавить свои собственные консольные шрифты. А откуда ж они у меня? Ясно, что я отказался от этого. Затем — параметры времени (UTC или не UTC, выбор региона, я выбрал Europe/Kiev).

После установки времени нужно выбрать оконный менеджер по умолчанию. Программа предложила KDE, я согласился.

Последний этап — ввод пароля root. Все. Перезагружаться сами не захотели — мол, уже не маленькие, знаете, как Ctrl + Alt + Del нажимать. Нажимаю заветную комбинацию, и система перезагружается. С момента выбора ядра до перезагрузки прошло 4 минуты и 51 секунда.

Загружаюсь. Система встретила меня голой консолью. Ввожу имя пользователя root и свой пароль. Все. Для запуска X Window (ну и KDE) нужно ввести startx. Некоторые пользователи Linux, даже которые уже использовали эту ОС, и не подозревают о существовании такой команды.

Вторая перезагрузка

Я перезагружаю систему, чтобы вычислить время второй загрузки. Оно составило 22 секунды до появления приглашения ввести имя пользователя и пароль и еще 22 секунды на запуск XWindow и KDE. Итого 44 секунды. После установки на разделе размером 3 Гб осталось 473 Мб свободного места.

Программное обеспечение

В этом дистрибутиве используется новая версия KDE — 3.3.3 и еще более новая, чем в FC3, версия GIMP — 2.2.3.

KDE русифицировать никак не удалось. Я установил все пакеты, а из доступных языков был только английский. И кому нужен такой дистрибутив?

Существенным недостатком, на мой взгляд, является отсутствие пакета OpenOffice. Вместо него установлен K Office 1.3.4.

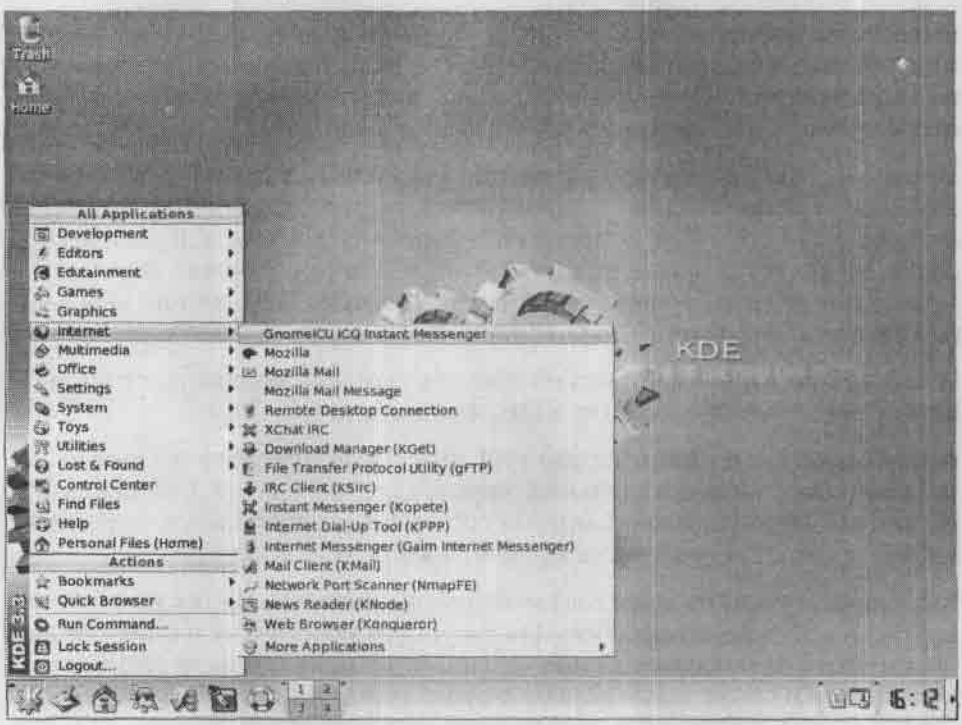


Рис. 12. *Рабочий стол KDE 3.3 (Slackware)*

Тесты

А вот тут началось самое интересное. Результат hdparm такой же, как и у FC3. 174 Мб скопировалось за 3,01 секунды. При копировании того же самого 700-Мбайтного фильма я получил следующие результаты:

- Копирование с FAT32-раздела на ext3-раздел: 1 мин 12 сек (!)
- Копирование с ext3-раздела на ext3-раздел: 0 мин 31 сек

Вот вам и ядро 2.4. Разница с FC почти в два раза: Slackware в два раза быстрее работает с файлами, чем Fedora Core. Файловая система одна и та же — ext3. Уже и не знаешь, куда отнести версию ядра 2.4 — к преимуществам или недостаткам. Никуда не буду относить. Будем считать это особенностью данного дистрибутива.

Диагноз

Неудобная текстовая англоязычная программа установки сразу отпугнет начинающих пользователей. Отсутствие Open Office и поддержки русского языка тоже не в сторону выбора дистрибутива. Зато это компенсируется

быстрой работой файловой системы и более быстрым запуском системы: 44 секунды против 1 минуты и 15 секунд у FC3. Этот дистрибутив можно порекомендовать квалифицированным пользователям, знающим толк в Linux — уж они-то смогут довести его «до ума». И вообще, учитывая особенности этого дистрибутива, он больше подходит для установки на сервере, чем на рабочей станции.

Преимущества	Недостатки
Дружественный интерфейс KDE 3.3	Неудобная, текстовая и англоязычная программа установки
Высокая производительность файловой системы	Нет поддержки русского языка. Ее обеспечить можно, но начинающему пользователю придется потратить на это уйму времени
Быстрая загрузка системы	Отсутствует пакет Open Office
Программное обеспечение	
Ядро версия 2.4.29 GNOME 2.6.2 KDE 3.3 Open Office — отсутствует GIMP 2.2.3 Загрузчик LILO	

Linux Mandrake 10.1

Этот дистрибутив я устанавливал последним специально: до этого на моем домашнем компьютере был установлен именно он. Как обычно, начнем с установки дистрибутива.

Установка

Как и Fedora Core 3, Mandrake поставляется на четырех компакт-дисках. Все четыре диска нужны при установке — даже если не отмечать дополнительные пакеты.

Программа установки понятна и новичку, с ней практически не бывает проблем — это я уж знаю точно, потому что данный дистрибутив я устанавливал не один раз, причем на разные компьютеры.

Весь подготовительный процесс в моем случае (напомню, что я не создавал разделы — они уже были созданы) занял всего 2 минуты и 1 секунду. На копирование пакетов, выбранных по умолчанию, понадобилось всего 7 минут и 2 секунды. После копирования пакетов на указание дополнительных параметров (я оставил все по умолчанию, только добавил одно пользователя) понадобилось 1 минута и 12 секунд. Итого на установку системы понадобилось 10 минут и 15 секунд (!)

После перезагрузки приятно обнаружить, что твоя звуковая плата работает, консоль полностью русифицирована (а не как в случае с FC3), подмонтированы все Windows-разделы (причем подмонтированы правильно — с русскими буквами проблем нет), да и выключается компьютер без всяких глюков.

Вторая перезагрузка

Вторая загрузка (с запуском X Windows и автоматическим входом пользователя) заняла 42 секунды. После установки осталось свободным 2,1 Гб дискового пространства.

Программное обеспечение

На четырех компакт-дисках Linux Mandrake вы найдете все необходимое программное обеспечение — от офисных приложений до простых игрушек. Для меня наиболее важны офисный пакет OpenOffice 1.1.0 и графический редактор GIMP 1.2.5. Версия GIMP не очень новая, но это поправимо — всегда можно загрузить новейшую.

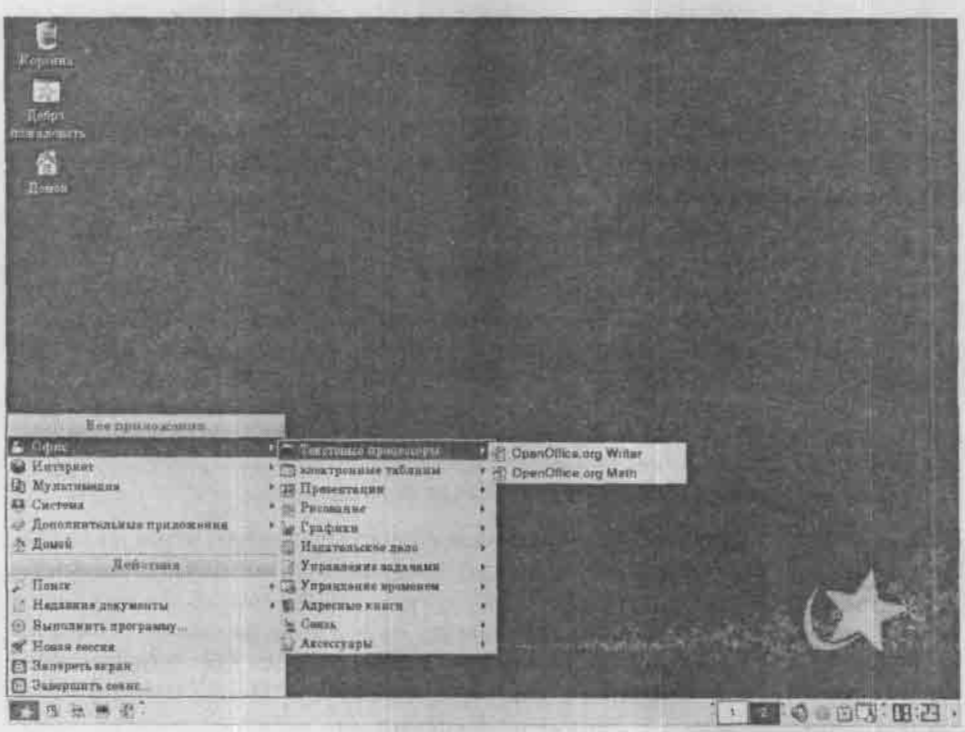


Рис. 13. Рабочий стол

Рис. 14. *Windows-разделы сразу правильно подмонтированы*

Рис. 15. *Воспроизведение фильма*

В качестве оконной среды по умолчанию используется KDE версии 3.2 — довольно удачная и быстрая версия.

С воспроизведением видеофайлов нет никаких проблем — все фильмы, имеющиеся у меня на Windows-разделах, были воспроизведены проигрывателем Totem — даже те, для которых я в Windows выкачивал из Интернета дополнительные кодеки.

Тесты

Дополнительно описывать тесты я не буду, а приведу только уже готовые результаты:

- Hdparm: 174 Мб прочитано за 3,02 секунды, скорость чтения 57,55 Мб/с;
- Копирование 700-Мбайтного файла с FAT32-раздела на ext3-раздел: 01:26;
- Копирование 700-Мбайтного файла с ext3-раздела на ext3-раздел: 0:36

Диагноз

Linux Mandrake — отличный дистрибутив для начинающих пользователей. После установки системы пользователь получает «готовую к употреблению» рабочую среду. Все, что ему нужно настроить (и то, только если он это не сделал при установке системы) — это настроить сеть и/или подключение к Интернету. Никаких особых проблем ни при установке, ни при работе с дистрибутивом не возникло.

Преимущества	Недостатки
Простая и быстрая установка	Особых недостатков нет, если не считать немного устаревших версий GNOME и GIMP
Хороший выбор программного обеспечения	
После установки система полностью готова к работе и не требует донастройки	
Нет проблем ни со звуком, ни с видео-кодеками	
Программное обеспечение	
Linux Mandrake 10 Ядро версия 2.6.3 GNOME 2.4.1 KDE 3.2 Open Office 1.1.0 GIMP 1.2.5 Загрузчик GRUB	Linux Mandriva 2005 (*) Ядро версия 2.6.11.6 GNOME 2.8.3 KDE 3.3.2 с поддержкой функций 3.4.0 Open Office 1.1.4 GIMP 2.2.5 Загрузчик GRUB

Примечание

Linux Mandriva Limited Edition 2005 — это первый выпуск популярного дистрибутива после слияния компаний MandrakeSoft и Conectiva. В этом дистрибутиве есть очень много интересного — от полной поддержки контроллеров Adaptec до поддержки игровых консолей Xbox.

Разное

ASP Linux

Если вы не достанете Mandrake (что маловероятно), установите ASP Linux (`www.asplinux.ru`) — вам понравится. Своей отлаженностью и стабильностью он произвел на меня очень приятное впечатление.

Дистрибутив отличается хорошей совместимостью с видеоакселераторами NVidia, ATI, а также чипсетом SiS630, который часто используется в ноутбуках.

ASP Linux также выделяется улучшенной поддержкой украинского языка и возможностью легкой смены кодировки (koi8-r, koi8-u, cp1251, iso8859-5). Естественно, в состав дистрибутива входят OpenOffice и другие традиционные Linux-приложения. Неожиданностью для меня стало наличие антивируса (eSafe от компании Aladdin), а включение в дистрибутив бухгалтерского программного обеспечения (от компании Hansa) — явление просто уникальное. Имеется также масса вспомогательных средств, например, система автоматического обновления, автоопределение приводов CD-RW и DVD и т.п.

Наиболее полный дистрибутив версии 7.3 «Восток» распространяется на десяти (!) компакт-дисках, среди которых три установочных, два с исходными текстами, по одному с документацией, играми и пользовательскими приложениями. На отдельном CD-ROM поставляется полная версия Acronis OS Selector 8.0, еще один диск содержит демонстрационные версии серверных приложений.

В коробочную поставку ASP Linux входят печатные руководства «Быстрый старт», «Руководство по инсталляции», «Руководство пользователя», «Руководство администратора», которые фактически освобождают пользователя от необходимости приобретения дополнительной литературы по Linux.

К тому же на Documentation CD, кроме традиционных FAQ и HOWTO от группы Linux Documentation Project, содержатся и их переводы, выполненные командой ASP Linux и сторонними переводчиками. Просмотр электронной документации обеспечивается через удобный гипертекстовый интерфейс.

На момент написания книги уже вышла десятая версия этого отличного дистрибутива.

ALT Linux

Использовать для обучения дистрибутив ALT Junior 2 я не рекомендую — уж слишком все там просто. Так вы уподобитесь иным пользователям Windows, которые привычно переустанавливают ОС при малейших затруднениях. Зато этот дистрибутив прекрасно подойдет, если вы хотите заменить Windows в офисе или дома — поставьте и не мучайтесь. В нем имеется полный комплект «стандартных» офисных приложений, приложения для работы в Интернете, проигрыватели MP3 и MP4, поддерживаются цифровые камеры, USB-принтеры и много чего еще. Сразу после установки практически все это будет уже настроено, останется лишь

изменить имя компьютера и параметры PPP-соединения. Интересно, что при этом ALT Junior 2 очень компактен, все необходимое ПО уместилось фактически на одном компакт-диске, на втором размещены исходные тексты и дополнительные средства разработки.

Что еще хорошо в этом дистрибутиве, а точнее, в политике компании ALT Linux (`www.altlinux.ru`), так это сопровождение — обновления появляются регулярно и оперативно. Скажем, версия 2.0 отличалась некоторой нестабильностью работы, однако вскоре вышел доработанный Junior 2.1.

Дистрибутив ALT Master предназначен для пользователей, уже знакомых с Linux. Он также создает впечатление более стабильного и надежного, чем ALT Junior. К тому же в коробочной версии вы найдете целых шесть компакт-дисков с различным программным обеспечением для Linux, в том числе — популярный эмулятор VMWare, который позволяет запускать одну ОС внутри другой.

Говоря о локализации, зачастую подразумевают только перевод интерфейса и возможность ввода символов национального алфавита и напрочь забывают о документации. Компания ALT Linux демонстрирует более цельный подход и в этом вопросе: в комплект стандартной поставки ALT Junior 2 входит печатное руководство по установке операционной системы. Хотя, на мой взгляд, в нем не хватает иллюстраций, с его помощью новичку будет гораздо проще инсталлировать ОС. Кроме того, в нем содержится краткая информация по установке дополнительных программ и обновлений.

В состав коробочной версии дистрибутива ALT Master входят четыре (!) прекрасных печатных руководства: по инсталляции ОС, администратора и пользователя, а также отдельное по OpenOffice. Можно с уверенностью сказать, что на первое время этой литературы более чем достаточно. А если учитывать, что руководства составлены разработчиками ALT Linux и все описанные в них примеры ориентированы на «родной» дистрибутив, то им и вовсе цены нет.

Говоря о политике компании ALT, невозможно не упомянуть Sisyphus — ежедневно обновляемый репозиторий пакетов программного обеспечения. Все дистрибутивы ALT Linux созданы на его основе. Для чего это нужно? Если у вас есть постоянный доступ к Интернету, вы можете настроить ваш дистрибутив на ежедневное обновление пакетов — тогда ваша система всегда будет поддерживаться в «актуальном состоянии» и вам не нужно будет периодически устанавливать новые версии — все будет происходить автоматически.

Конечно, автоматическое обновление имеет свои преимущества и недостатки — вы же обновляете программное обеспечение, а не простую

базу данных. Новое программное обеспечение может содержать новые функции, что является положительной стороной, но и «подводные камни»: никто не знает, сколько ошибок было внесено в программу при исправлении уже известных ошибок и добавлении новых функций. Лично я никому не доверяю обновлять свою систему (это даже недопустимо с точки зрения безопасности), поэтому предпочитаю покупать и устанавливать новые версии дистрибутивов. Впрочем, если в ваш дистрибутив какой-то пакет не включен, в репозитории Sisyphus вы, скорее всего, его найдете.

Подробнее о Sisyphus вы сможете прочитать по адресу `www.altlinux.ru`.

SuSE Linux

SuSE Linux — немецкий дистрибутив компании Novell. Это очень хороший, надежный дистрибутив с «немецким качеством». Чего стоит только тот факт, что разработчик этого дистрибутива — компания Novell — разработала одну из самых надежных сетевых операционных систем Novell Netware. Ранние версии SuSE грешили недружелюбием по отношению к русскому языку, но в последних версиях этот недостаток искоренен. Последние версии SuSE можно порекомендовать не только опытным (как это было с предыдущими версиями), но и начинающим пользователям Linux. Последняя на момент написания этих строк (9.2) версия SuSE содержит:

- ядро версии 2.6.8;
- KDE версии 3.3.0;
- Open Office 1.1.3 — популярный офисный пакет, аналог MS Office;
- KOffice 1.3.3 — еще один офисный пакет;
- The GIMP 2.0.4 — мощнейший графический редактор, аналог PhotoShop;
- много другого полезного программного обеспечения.

Версия 9.2 поставляется на пяти компакт-дисках — как видите, дистрибутив не маленький (для сравнения: Mandrake 10 поставляется на четырех CD).

Подробно об этом дистрибутиве вы сможете узнать по адресу `www.suse.com`, а если заинтересуетесь, то можете купить его в любом интернет-магазине.

Knoppix

Эта разработка Клауса Кноппера (`www.knopper.net/knoppix`) — самый лёгкий и быстрый способ познакомиться с Linux. Дистрибутив поставляется на одном-единственном компакт-диске и может работать прямо с него, без установки на компьютер. Хотя при желании вы можете

и установить его на жесткий диск, а дополнительные программные пакеты скачивать из архива на www.debian.org. Да, этот дистрибутив основан на Debian, т.е. не родствен Red Hat.

При таком детском размере Knoppix содержит недетское количество программного обеспечения, включая офисные пакеты, медиа-плейеры и средства работы в Интернете. Дело в том, что программы уложены на диск в сжатом виде и в процессе запуска распаковываются из архивов на лету.

Самая свежая на момент написания этой книги версия дистрибутива — Knoppix 3.8.1 — продается в интернет-магазине knoppix.ru за 99 рублей.

Debian

Debian — это целая линия развития Linux, альтернативная Red Hat и не слишком совместимая с ней. Среди лидирующих дистрибутивов Debian — единственная полностью некоммерческая система. Если разработкой Red Hat занимается Red Hat Software, а за Slackware стоит Walnut Creek, то дистрибутив Debian/GNU Linux создает и поддерживает группа энтузиастов — строго в духе движения за открытые исходники. Не зря его предпочитает Ричард Столлмен, лидер Фонда Свободного Программного Обеспечения.

Debian — настоящая энциклопедия свободных программ: в его состав входит более 15490 пакетов заранее скомпилированного программного обеспечения, которые можно бесплатно скачать с www.debian.org/distrib/packages. Там же предлагаются дополнительные пакеты, которые нельзя включать в основной дистрибутив, потому что их лицензии не соответствуют принятому в этой группе определению бесплатного программного обеспечения (с правом дальнейшего распространения, наличием исходных текстов, разрешением их модификации и использования в качестве основы новых разработок). Система управления пакетами у Debian собственная, похожая на Red Hat'овскую только внешне.

Debian пользуется также репутацией самого надежного дистрибутива благодаря тому, что его разработчики имеют опубликованные точные критерии качества программного обеспечения и отлаживают его исключительно тщательно. Вместе с тем поддерживается так называемая «нестабильная» ветвь — экспериментальное направление, на котором разрабатываются новейшие идеи.

Годы работы многих сотен разработчиков сделали Debian самым понятным, логичным в настройке и администрировании дистрибутивом. Но для быстрого старта он не подходит: разработчики не приняли во внимание интересы желающих щелкать мышью и не лезть в основы системы.

Глава 1

УСТАНОВКА ОПЕРАЦИОННОЙ СИСТЕМЫ

- ПОДГОТОВКА ЖЕСТКОГО ДИСКА
- ЗАГРУЗКА ПРОГРАММЫ УСТАНОВКИ
- УСТАНОВКА FEDORA CORE
- ОСОБЕННОСТИ УСТАНОВКИ MANDRAKE 10.0
- ПРОБЛЕМЫ ПРИ УСТАНОВКЕ
- КАК УДАЛИТЬ LINUX
- СИСТЕМЫ С ДВОЙНОЙ ЗАГРУЗКОЙ
- ПЕРВЫЙ ЗАПУСК LINUX
- СТАНДАРТНЫЕ СЕРВИСЫ LINUX
- СПРАВОЧНАЯ СИСТЕМА

1.1. Подготовка жесткого диска

Сейчас на вашем компьютере установлена, скорее всего, одна из ОС семейства Windows со своей файловой системой. ОС Linux использует другой тип файловой системы, поэтому для ее установки вы должны освободить место на диске и отформатировать его (т.е. создать на нем новую файловую систему). Если вы не намерены окончательно избавляться от Windows, то вам придется создать на диске несколько разделов, каждый для своей операционной системы.

Примечание

Существует несколько определений файловой системы. Для себя вы можете выбрать одно из нижеприведенных — они оба вполне корректны.

Файловая система — часть операционной системы, обеспечивающая выполнение операций над файлами.

Файловая система — способ организации и представления битов на жестком диске.

Разделом называется участок жесткого диска, к которому можно обратиться как к отдельному диску, что достигается использованием таблицы разбиения жесткого диска *(partition table)*. Эта таблица записана в самом начале диска и подразделяется на 4 секции, содержащие характеристики 4 возможных разделов: их расположение, тип и отметку об активности. Отметка об активности используется загрузчиками некоторых ОС, в частности, Windows может быть загружена только из «активного» раздела.

Эти 4 раздела называются первичными (*primary*). Один из разделов может иметь тип «расширенный» (*extended*). Такой раздел содержит собственную таблицу разбиения на один или несколько разделов, называемых логическими (*logical*). Таким образом, пространство жесткого диска может быть структурировано удобным вам способом.

Итак, на вашем диске достаточно места для установки Linux, но все это место находится в активно используемом разделе, содержащем ОС Windows и все данные. Вы можете:

- Полностью переразметить диск.
  При этом вы уничтожаете один большой раздел и создаете на его месте несколько маленьких. Все данные, конечно, будут потеряны, и Windows тоже придется переустановить. Ваши действия:
  1. Убедитесь, что у вас есть компакт-диск, с которого можно переустановить исходную ОС;
  2. Создайте резервную копию всех своих данных;
  3. Разбейте диск на разделы с помощью утилиты `fdisk` от Windows;
  4. Восстановите свои данные и возвращайтесь к установке Linux.
- Уменьшить имеющийся большой раздел с сохранением данных.
  Существует несколько программ, позволяющих это сделать. В предыдущие дистрибутивы Red Hat входила утилита **fips**, умевшая работать с разделами FAT и FAT32, но не NTFS. Она все еще доступна по адресу `http://www.igd.fhg.de/~aschaefe/fips`, но в современные дистрибутивы обычно не включена. Вместо нее лучше воспользоваться программой **PartitionMagic** от Symantec Norton, понимающей все типы файловых систем. Ваши действия:
  1. Уплотните существующие данные (дефрагментируйте ваш диск), чтобы увеличить свободное место в «конце» раздела.
  2. Уменьшите размер раздела, создав на диске неразмеченное пространство.
  3. Создайте новые разделы. Можете отложить этот шаг до этапа установки Linux: инсталляторы современных дистрибутивов (в частности, Fedora Core и Mandrake) вызывают специальную программу для работы с разделами диска.

1.1.1. Рекомендуемая схема разбиения диска

- Раздел подкачки или свопинга (*swapping*) — используется виртуальной памятью. Иными словами, данные, не умещающиеся в оперативной памяти, перемещаются на жесткий диск. Например, у вас в данный момент свободно всего 8 Мбайт оперативной памяти, а вы пытаетесь открыть документ размером в 16 Мбайт. В оперативную память при этом будут загружены первые несколько мегабайт, а все остальное будет находиться во внешней памяти. Когда вам потребуется перейти в конец документа, операционная система подкачает в память нужные данные, а неиспользуемые будут помещены во внешнюю память. Размер раздела подкачки должен быть не меньше 32 Мбайт. Обычно его размер равен удвоенному объему оперативной памяти.

- Раздел /boot — содержит ядро операционной системы и несколько файлов, используемых при загрузке. Необходимость этого раздела вызвана «барьером 1024 цилиндра», то есть тем, что BIOS большинства персональных компьютеров «не видит» цилиндров с номерами больше 1024. Рекомендуемый размер этого раздела — 100 Мбайт.
- Корневой раздел, содержащий файлы, необходимые для работы системы и прикладных программ;
- Раздел /home для пользовательских данных.

1.1.2. Имена разделов в ОС Linux

Linux представляет наименования разделов как имена файлов, в виде /dev/xxyN, где:

- /dev — это каталог, в котором расположены все файлы, связанные с устройствами;
- xx — две буквы названия раздела, указывающие тип устройства, на котором размещается раздел. Как правило, это либо hd (если это IDE диск), либо sd (для SCSI дисков);
- y — буква, описывающая само устройство, на котором находится раздел. Например, /dev/hda (первый IDE жесткий диск) или /dev/sdb (второй SCSI диск);
- N — число, обозначающее непосредственно раздел. Первичные разделы нумеруются числами с 1 по 4. Нумерация логических разделов начинается с 5, даже если первичных разделов меньше четырех.

Таким образом, /dev/hda2 — это второй первичный раздел на первом диске IDE, а /dev/sdb5 — это первый логический раздел на втором диске SCSI.

1.1.3. Разделы и точки монтирования

С логической точки зрения файловая система каждого раздела представляет собой отдельное дерево каталогов. Объединение их в общую иерархическую структуру с одним корнем достигается путем ассоциирования раздела с одним из каталогов, содержащихся в корневом каталоге. Эта операция называется монтированием (*mounting*). Монтирование раздела дает доступ к нему через указанный каталог, называемый точкой монтирования (*mounting point*).

Например, если раздел /dev/sdb5 был смонтирован как /home/ivan/doc, то все файлы и каталоги, находящиеся в /home/ivan/doc, физически будут размещаться на /dev/sdb5. А файлы, находящиеся в /home/ivan/doc/Russian, — на /dev/sdb6, если назначить каталог Russian точкой монтирования для раздела /dev/sdb6.

1.2. Загрузка программы установки

1.2.1. С использованием загрузочного компакт-диска

Самый удобный и уже самый распространенный способ. Первый диск любого дистрибутива — загрузочный. Вставьте его в дисковод, перезагрузите компьютер, войдите в BIOS Setup и сделайте CD-привод первым в последовательности загрузки.

Некоторые дистрибутивы (например, Mandrake 10.0) предлагаются на DVD, с которыми нужно поступать точно так же (если, конечно, у вас есть соответствующий привод).

1.2.2. С использованием загрузочной дискеты

Способ настолько устаревший, что в современные дистрибутивы загрузочный образ для дискеты и не включен. Опишу его, опираясь на дистрибутив RedHat 7.1.

1. Скопируйте каталоги `dosutils` и `images` на жесткий диск (желательно на C:).
2. Перезагрузите компьютер в режиме MS DOS.
3. Введите команду `rawrite` (если вы — на свой страх и риск — предпочитаете работать в Windows, то пользуйтесь командой `rawritewin`):

   ```
   C:\>c:\dosutils\rawrite
   ```

   На запрос программы

   ```
   Enter disk image source file name:
   ```

   введите:

   `c:\images\boot.img` — для обычной установки или

   `c:\images\bootnet.img` — для установки по сети.

   А затем на предложение программы ввести диск назначения введите имя дискеты:

   ```
   Enter destination drive:
    a:
   ```

4. Вы получили загрузочную дискету. Вставьте ее в дисковод и перезагрузите компьютер.

Внимательно прочитайте INSTALL.TXT в корневом каталоге CD, загляните также в каталог `images` и прочитайте файл README. Инструкции по установке обычно находятся там.

1.2.3. С использованием жесткого диска

Если вы скачали образы компакт-дисков дистрибутива, то можете прожечь их на CD и устанавливать так, как сказано в п.1.2.1. Когда скорость вашего CD-привода слишком мала (например, 4x), имеет смысл разместить ISO-образы на жестком диске, а с CD только загружаться. Загрузочный образ находится в каталоге `images` на первом диске дистрибутива и называется `boot.iso`.

Если какой-то вариант Linux у вас уже установлен, то вы можете извлечь этот файл из файла образа, смонтировав этот образ:

```
$ mount -o loop -t iso9660 <файл_iso> <точка_монтирования>
```

Прожгите файл `boot.iso` на компакт-диск и загрузитесь с этого компакт-диска.

Если ваш BIOS поддерживает загрузку с USB-устройств, то вы можете скопировать на USB-диск загрузочный образ `images/diskboot.img` с первого диска дистрибутива. Из-под Linux это можно сделать так:

```
$ dd if=diskboot.img of=/dev/sda1
```

После чего загрузитесь с USB-диска и следуйте указаниям программы установки.

Если вы загружаетесь с дистрибутивного CD, но устанавливать собираетесь с жесткого диска, то в ответ на приглашение загрузчика

```
boot:
```

введите тот вариант загрузки, который позволяет выбрать устройство (в дистрибутиве Fedora Core эта директива называется askmethod). Укажите инсталлятору раздел диска и каталог, в котором находятся образы дисков дистрибутива.

1.2.4. Установка по сети

Для этого варианта установки вам нужно иметь доступ к FTP-серверу, где хранится каталог с избранным вами дистрибутивом. Загрузитесь с того носителя, на который вы скопировали загрузочный образ, ответьте на вопросы инсталлятора и выберите в качестве устройства для установки FTP-сервер.

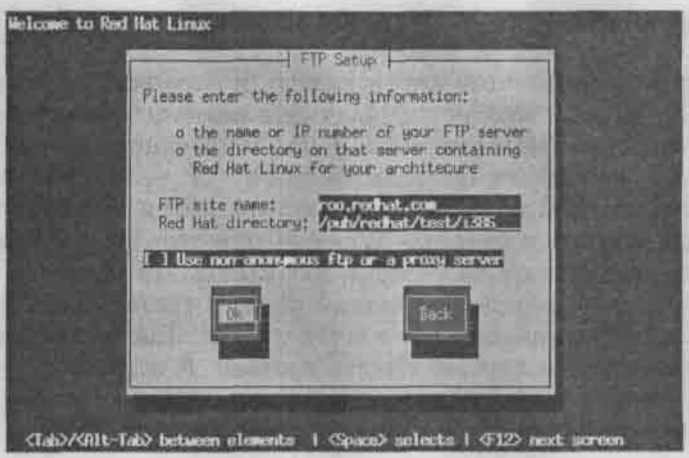


Рис. 1.1. *Установка RedHat по протоколу FTP*

1.3. Установка Fedora Core

1.3.1. Описание дистрибутива

Название проекта Fedora Core не имеет никакого отношения к знакомому нам с детских лет произведению К.И. Чуковского «Федорино горе». Fedora Core является наследницей линейки RedHat (бесплатной), а Fedore означает фетровая шляпа (против Красной Шапочки RedHat)

Fedora Core Linux — это наследник культового дистрибутива Red Hat Linux, созданный сообществом добровольцев Fedora Project (`http://www.redhat.com/fedora`) при участии и спонсорской поддержке компании Red Hat. Дистрибутивы Fedora Core содержат только новейшие версии программного обеспечения. Лучшие решения, найденные командой разработчиков Fedora, Red Hat включает в программное обеспечение, поставляемое в ее коммерческих дистрибутивах Enterprise Linux.

Несмотря на это, Fedora Core — не тестовая площадка для энтузиастов, а полноценный дистрибутив, пригодный для домашнего, офисного или серверного применения. Дистрибутив включает удобную программу для установки и полный набор приложений — офисные пакеты, браузеры и web-сервера, мультимедийные средства и инструменты разработчика ПО. Fedora Core Linux может легко соседствовать с MS Windows как на одном компьютере, так и в локальной сети, а офисный пакет OpenOffice позволяет без проблем открывать, редактировать и сохранять документы в форматах MS Word и MS Excel.

Версии Fedora Core — 1, 2, 3 и 4. Системные требования

На момент написания этой книги вышло четыре версии дистрибутива Fedora Core. Первая версия работала еще недостаточно стабильно. Во второй замечен довольно неприятный баг, мешающий сосуществованию Fedora Core 2 и Windows, особенно Windows XP, на одном компьютере: после установки FC2 на некоторые жесткие диски Windows перестает загружаться. Данные в разделах Windows при этом не портятся, но для того, чтобы Windows снова стала распознавать эти разделы, требуется вручную реконфигурировать жесткий диск. В третьей версии этот баг устранен, однако появилось множество других. Так что рекомендуется использовать вторую версию вместо третьей. В последней, четвертой версии FC ошибки предыдущих версий были исправлены. Так что что в качестве рекомендации могу посоветовать именно ее.

Третью версию устанавливать не рекомендуется ввиду ее нестабильности и большого количества возникающих коллизий (типа неработающих драйверов и т.п.), разрешить которые может только опытный пользователь, но и ему зачем с этим всем возиться — тоже не совсем понятно. Лучше изначально выбрать стабильный дистрибутив.

Команда разработчиков Fedora Project обещает выпускать новые версии дистрибутива 2-3 раза в год. Вы можете следить за новостями на странице `http://fedora.redhat.com`.

Системные требования всех версий обычны для современных дистрибутивов:

- процессор класса не ниже Pentium (Pentium 200 МГц для работы в текстовом режиме, Pentium II 400 МГц или лучше — для работы с графикой);
- память: не менее 64 Мб для текстового режима, не менее 192 Мб (рекомендуется 256 Мб) для графического;
- пространство на диске зависит от выбранного типа установки:
- 620 Мб — минимальная установка;
- 1.1 Гб — сервер;
- 2.3 Гб — настольный компьютер;
- 3.0 Гб — рабочая станция;
- 6.9 Гб — полная установка.

Кроме того, сама программа установки требует еще от 90 Мб (минимальная установка) до 175 Мб (полная установка) дополнительного пространства на жестком диске, которое после завершения установки будет освобождено. И, конечно, нужно место для пользовательских данных и не менее 5% дискового пространства для работы самой системы.

Вторая версия Fedora Core отличается от первой следующими новинками:

- Ядро версии 2.6 со всеми вытекающими отсюда последствиями (лучшая масштабируемость, производительность и поддержка новых устройств).
- Новая версия системы X Window X11R6.7.0.
- Новая версия оконной среды GNOME (2.6), которая работает значительно быстрее предыдущей.
- Новая версия оконной среды KDE (3.2.2), в которой исправлены некоторые проблемы, включая проблемы с переводом интерфейса.
- Xfce 4: легкое и быстрое окружение рабочего стола.
- Поддержка технологии Subversion 1.0, которая должна заменить технологию CVS.
- Реализация системы безопасности SELinux.

В третьей версии добавлены или обновлены:

- Новая версия оконной среды GNOME 2.8.0.
- Новая версия оконной среды KDE 3.3.0 (к сожалению, пока только на английском).
- Клиент электронной почты Evolution 2.0.
- Дополнительная «целевая» политика безопасности SELinux, следящая не за всеми операциями, а только за определенными демонами.
- Динамическое управление каталогом устройств (программа **udev**), позволяющее создавать файлы устройств по мере подключения драйверов.

В четвертой версии Fedora Core, появившейся 13 июня 2005 г, можно отметить следующие особенности:

- Ядро версии 2.6.11.
- Fedora Core 4 стала первым дистрибутивом, скомпилированным с помощью четвертого компилятора GNU C (GCC).
- Новая версия системы X Window X11R6.7.0.
- Новые версия оконных сред: GNOME 2.10 и KDE 3.4.0.
- Кластерная файловая система GFS 6.1.-0.pre22.
- Универсальный просмотрщик документов Evince 0.2.1.
- Существенно улучшенная и расширенная целевая политика безопасности SELinux (дополнительно стали покрываться еще 80 демонов).
- Eclipse 3.1M6. — полнофункциональная среда Java-разработок.
- Некоторые приложения оказались из дистрибутива изъяты. В четвертой версии FC вы не найдете exim, XFce 4, медиаплеера xmms и даже KOffice. Разработчики пошли, на мой взгляд, вполне правильным путем и стали помещать в дистрибутивы только приложения, лучшие в своей функциональной категории. Например, возможности XFce 4 с лихвой покрываются возможностями KDE и GNOME, так зачем же из дистрибутива делать свалку программ. При этом, если вам нужна какая-то программа, отсутствующая в дистрибутиве, вы всегда сможете ее бесплатно скачать.

Детальную информацию о новинках и нововведениях в четвертой версии можно узнать по адресу `www.rhd.ru/docs/manuals/fedora/4/RELEASE-NOTES-ru.html`.

Специальная технология разграничения доступа SELinux

SELinux (Security Enhanced Linux) — это технология, позволяющая лучше защитить ваш компьютер от взлома и непродуманных действий пользователей. Технология SELinux работает совместно с механизмом прав доступа в Linux, позволяя администратору ввести дополнительные правила доступа к файлам.

Работает это так: если права доступа запрещают доступ к файлу, запрос отклоняется сразу. Если же права доступа разрешают доступ к файлу, настает черед SELinux. Система SELinux проверяет текущие правила работы с данным файлом для запросившего его процесса, запустившего процесс пользователя и выполняемой им «роли» (например, сам администратор может запретить себе доступ к критически важным ресурсам, когда он не в административной роли. Тогда даже украденный пароль root не слишком поможет злоумышленнику). Если правила SELinux не разрешают доступ к этому файлу, операция отклоняется. Как видите, SELinux — это еще один барьер в системе безопасности вашего сервера (или просто домашнего компьютера).

Кроме ограничения доступа к файлам, с помощью SELinux можно ограничивать действия отдельных пользователей, групп пользователей, процессов. Подробно о SELinux на русском языке вы сможете прочитать по адресу `http://www.opennet.ru/base/sec/intro_selinux.txt.html`.

1.3.2. Особенности установки различных версий Fedora Core

Fedora Core 2: подготовка к совместной жизни с Windows

Если вы собираетесь держать на одном компьютере ОС Windows и Fedora Core 2, то не торопитесь устанавливать FC 2. Дело в том, что программа-инсталлятор этой версии дистрибутива по-своему определяет параметры физической разметки жесткого диска. После того, как эти параметры будут занесены в таблицу разделов, Windows перестанет распознавать диск. Чтобы обойти эту ошибку, нужно помешать инсталлятору автоматически определять параметры разметки диска, указав их вручную.

Для определения геометрии диска (цилиндры, головки и секторы, CHS) служит утилита **fdisk**, входящая в любой дистрибутив Linux. Если никакого Linux у вас еще не установлено, то загрузитесь с первого компакт-диска устанавливаемого дистрибутива и в ответ на приглашение загрузчика введите параметр ядра **rescue**:

```
boot: linux rescue
```

Будет загружено средство восстановления операционной системы. Выберите язык, раскладку клавиатуры и нажмите кнопку **Continue**. В командной строке введите команду:

```
fdisk -l /dev/hda
```

(/dev/hda — это ваш загрузочный жесткий диск, на который вы собираетесь устанавливать Linux).

Программа **fdisk** выведет среди прочей информации сведения о количестве цилиндров, головок и секторов (CHS) вашего диска, как показано на рис. 1.2.

Затем перезагрузитесь с первого компакт-диска (Ctrl+Alt+Del) и в ответ на приглашение загрузчика введите геометрию диска:

```
boot: linux hda=C,H,S
```

где C — количество цилиндров, H — количество головок и S — количество секторов.

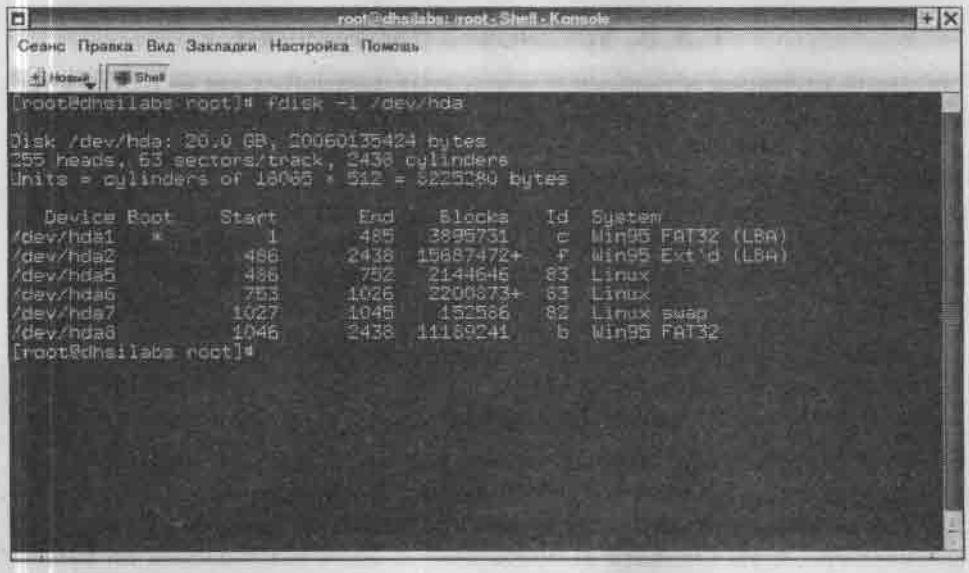


Рис. 1.2. *Информация о геометрии диска*

В нашем случае строка загрузки Linux будет выглядеть так:

```
linux hda=2438,255,63
```

Теперь инсталлятор будет руководствоваться этими параметрами вместо того, чтобы пытаться определять геометрию диска самостоятельно.

Примечание

Не следует пытаться определять геометрию диска с помощью BIOS, так как способы определения CHS у ОС Linux и BIOS отличаются, и вы можете указать неправильную информацию. Для определения CHS нужно использовать только **fdisk**!

Если вы поспешили и уже установили Fedora Core 2, в результате чего ваша Windows XP перестала загружаться, обратитесь к сайту Red Hat. По адресу `http://www.redhat.com/archives/fedora-devel-list/2004-May/msg00908.html` вы сможете прочитать рекомендации по решению проблемы загрузки Fedora + XP.

Особенности и преимущества процесса установки Fedora Core 4

Собственно особенностей то никаких и нет — все стандартно. А среди преимуществ, на которые упирают даже сами разработчики — это наличие подробнейшего руководства по установке Fedora Core 4.

1.3.3. Установка загрузчика

На одном из этапов установки (в некоторых дистрибутивах — ближе к началу, в других — к концу) программа-инсталлятор позволит вам выбрать и установить загрузчик. Стандартными загрузчиками Linux служат LILO (*LInux LOader*) или GRUB (*GRand Unified Bootloader*). Сейчас я не буду описывать достоинства и недостатки этих загрузчиков, а только скажу, что я бы выбрал LILO: он мне больше нравится. Потом вы сможете сменить загрузчик в любой удобный для вас момент. Программа установки спросит вас, куда установить загрузчик:

- В MBR (*Master Boot Record* — Главная загрузочная запись);
- В первый сектор раздела Linux;
- Не устанавливать загрузчик вообще.

Если вы не планируете загружать Linux посторонним загрузчиком, например, загрузчиком Window NT — **NTLoader**, то устанавливайте загрузчик в MBR. В этом случае загрузчик получит управление сразу после загрузки компьютера, и вы сможете выбрать любую из установленных на вашем компьютере операционных систем.

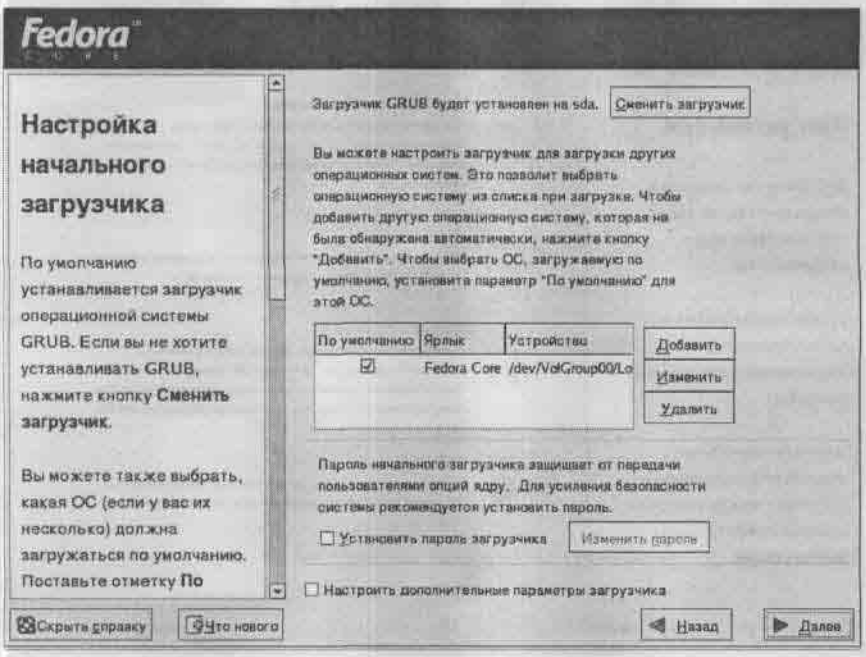


Рис. 1.3. *Выбор загрузчика*

Если вы планируете загружать Linux другим загрузчиком, выберите установку загрузчика в раздел жесткого диска.

Последний вариант — Не устанавливать загрузчик — следует использовать, если вы хотите загружать Linux с дискеты.

1.3.4. Продолжение установки

Выберите класс установки (рис. 1.4).

Класс «Персональный компьютер» подойдет для начинающих пользователей. Будут установлены: графический интерфейс, очень похожий на привычный рабочий стол ОС Windows, и программы, необходимые для домашнего использования компьютера или для организации домашнего офиса. Этот набор программного обеспечения будет занимать около 1.5 Гбайт на жестком диске, а если вы установите сразу две оконных среды (GNOME и KDE), то такой вариант займет около 1.8 Гбайт.

Класс «Рабочая станция» подойдет, если вы планируете использовать свой компьютер в локальной сети или для разработки программного обеспечения. Класс «Рабочая станция» занимает около 2 Гбайт на жестком

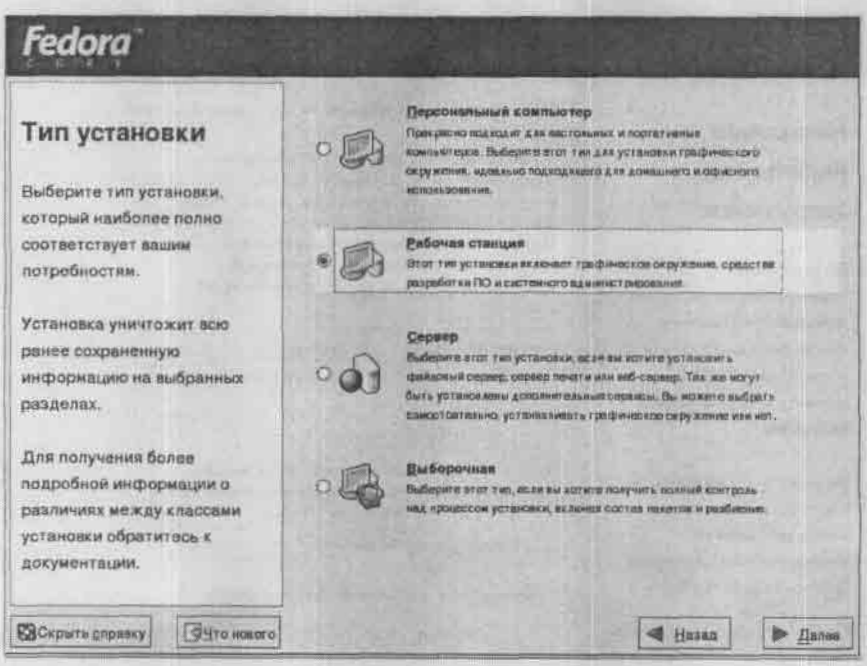


Рис. 1.4. Выбор класса установки

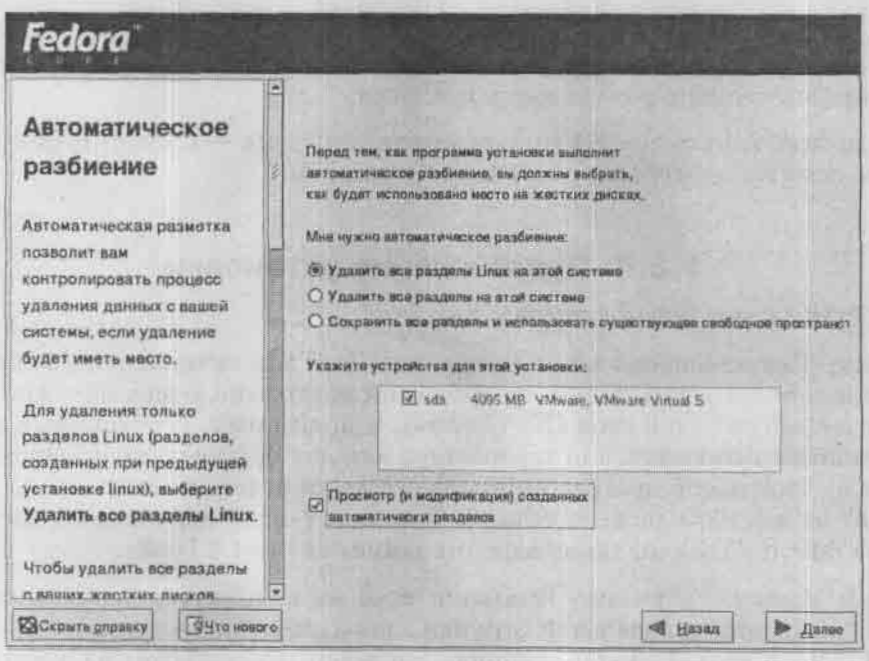


Рис. 1.5. Уточнение параметров автоматического разбиения

диске, а если вы хотите установить данный класс с возможностью выбора графической среды (KDE или GNOME), то такой вариант займет около 2.3 Гбайт.

Класс «Сервер» нужно использовать, если вы хотите установить и настроить сервер на основе ОС Linux. Будет установлено все необходимое для организации сервера программное обеспечение. Графический интерфейс по умолчанию не устанавливается. Минимальный размер установки — 1.3 Гбайт и еще около 900 Мбайт, если графический интерфейс вам все-таки нужен.

Я рекомендую вам выбрать класс установки «Выборочная», даже если вы — начинающий пользователь. Этот класс позволяет более гибко настроить вашу систему еще на стадии установки, точно определить устанавливаемые пакеты и их размер.

После выбора класса установки программа поинтересуется, как вы будете разбивать жесткий диск: автоматически или с использованием программы Disk Druid. Если вы сомневаетесь — выберите автоматическое создание разделов.

При выборе автоматического разбиения программа попросит вас уточнить параметры разбиения:

- Удалить все Linux-разделы (Remove all Linux Partitions).
- Удалить все разделы (Remove all Partitions).
- Оставить все разделы и использовать свободное место (Keep all partitions).

В первом случае инсталлятор удалит все Linux-разделы, если такие имеются на вашем диске. Во втором — будут удалены абсолютно все разделы, включая Windows-разделы. Последний вариант предполагает, что на жестком диске имеется неиспользуемое пространство (не относящееся ни к одному из разделов), которое будет использоваться для установки.

Если вы выберете возможность просмотра автоматически созданных разделов, то следующий шаг установки вы сделаете с программой разметки диска Disk Druid (рис. 1.6).

Кнопка **Создать** создает новый раздел. Для этого нужно освободить пространство на жестком диске (именно пространство, т.е. область жесткого диска, не принадлежащую ни одному из разделов, а не свободное место).

Для редактирования параметров раздела предназначена функция **Изменить**. Она позволяет изменить точку монтирования раздела, его тип и размер. Если раздел уже был создан на диске, то программа Disk Druid не в состоянии изменить его размер. Для изменения размера нужно удалить этот раздел и создать заново, но с другими параметрами.

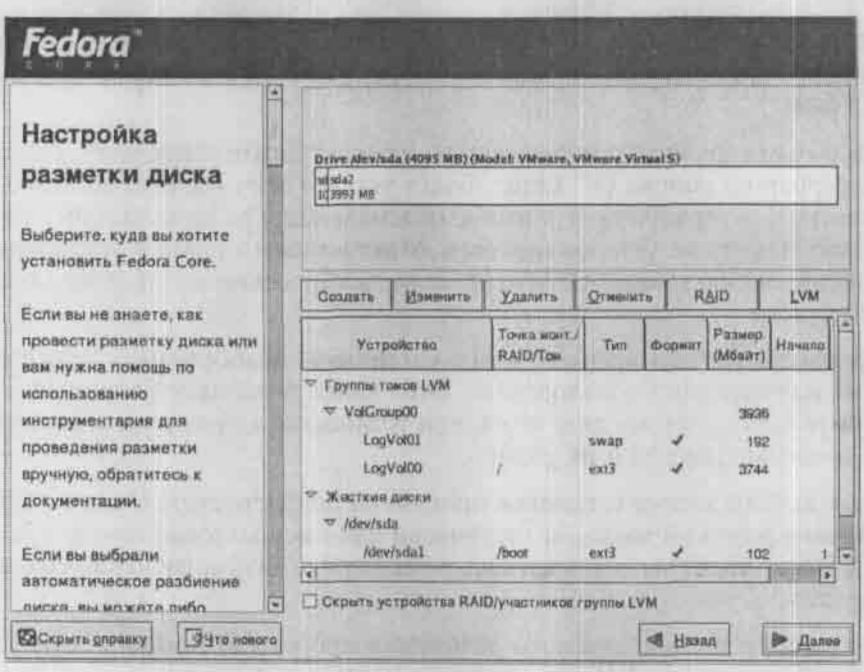


Рис. 1.6. *Программа Disk Druid*

Для создания Linux-раздела нам нужно удалить один из Windows-разделов. Желательно удалять тот, который вы создали с помощью программы **Partition Magic**, поскольку он не содержит данных. После удаления раздела на его месте нужно создать три раздела:

1. **Linux Native** (файловая система ext2 или ext3, точка монтирования /);
2. **Linux Swap** (точка монтирования не нужна);
3. **Linux Native** (файловая система ext2 или ext3, точка монтирования `/home`).

В таблице 1.1 представлена рекомендуемая схема разбиения единственного жесткого диска (`/dev/hda`) для установки на нем двух ОС: Linux и Windows.

Далее программа-инсталлятор предложит вам настроить сеть и брандмауэр (рис. 1.7) — эти шаги можно пропустить, оставив их настройку на потом. Систему безопасности SELinux обязательно выключите или, в крайнем случае, оставьте в режиме предупреждений — в этом режиме она не запрещает недопустимые с ее точки зрения действия пользователя, а протоколирует их.

Схема разбиения диска /dev/hda *Таблица 1.1*

Раздел	Активный	Тип/ФС	Размер, Гб	Комментарий
hda1 (диск C:)	Да	Первичный/ vfat	1.5-2	Для самой Windows больше и не нужно, приложения можно поместить на диск D:
hda2		Расширенный	Все оставшееся место	На жестком диске может быть четыре первичных раздела. Дополнительные разделы следует помещать в расширенный раздел. Некоторые ОС (в частности, Windows) требуют, чтобы их раздел был обязательно первичным и активным. Linux может загружаться как с активных, так и с неактивных разделов
hda5		Логический/ ext2 или ext3	2.5-3	Для корневой файловой системы Linux. Нумерация логических разделов начинается с 5!
hda6		Linux swap	128-256 Мб	Не следует этот раздел помещать в конец диска, поскольку операционная система использует его в качестве раздела подкачки. Минимальный размер данного раздела — 32 Мб
hda7		Логический/ ext2 или ext3	5-10	Для пользовательских данных — файловой системы /home
hda8 (диск D:)		Логический/ vfat	Все оставшееся место	Это диск D: — на него следует устанавливать программы Windows

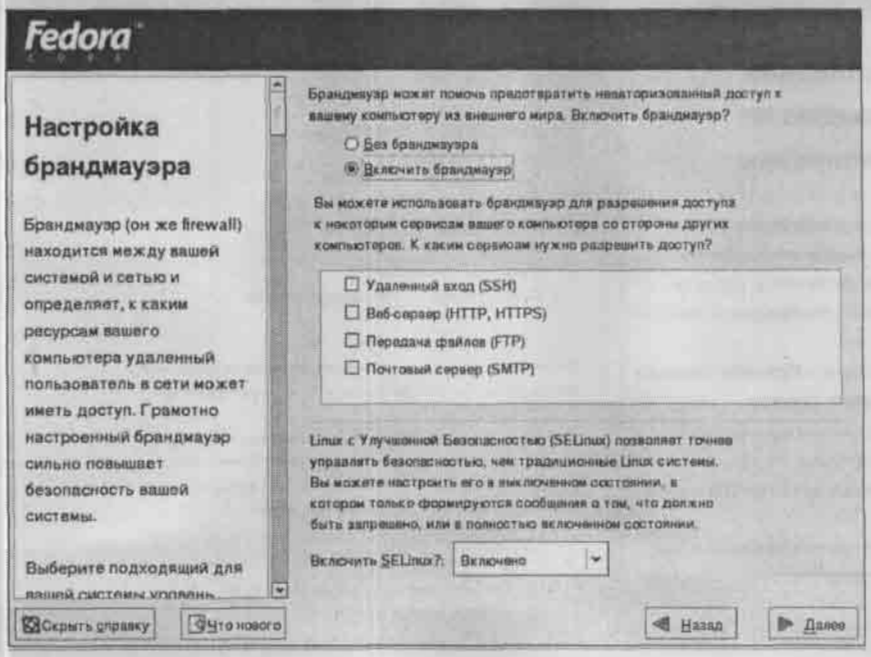


Рис. 1.7. *Настройка безопасности*

Вы также можете выбрать дополнительные языки, установить дату и время и ввести пароль суперпользователя (пользователя root). Задаваемый пароль должен быть не короче 8 символов. При вводе символы не будут отображаться на экране. Категорически не рекомендуется использовать в качестве пароля что-то вроде 123456, qwerty, password и тому подобное. Подумайте о выборе пароля: он должен быть одновременно легким для запоминания и трудным для подбора. Осторожно! Забыв этот пароль, вы не сможете настраивать систему.

Постоянно работать под учетной записью root категорически не рекомендуется из соображений безопасности — вы можете нечаянно разнести всю систему. Поэтому нужно добавить хотя бы одного непривилегированного пользователя, который будет выполнять повседневные задачи (набор текста, просмотр видео, программирование), даже если этим единственным пользователем будете вы сами. Учетную запись root нужно использовать только для настройки системы.

Теперь нужно выбрать пакеты программ для установки (рис. 1.8). Если вы хотите выбрать пакеты самостоятельно, установите переключатель «Уточнить наборы».

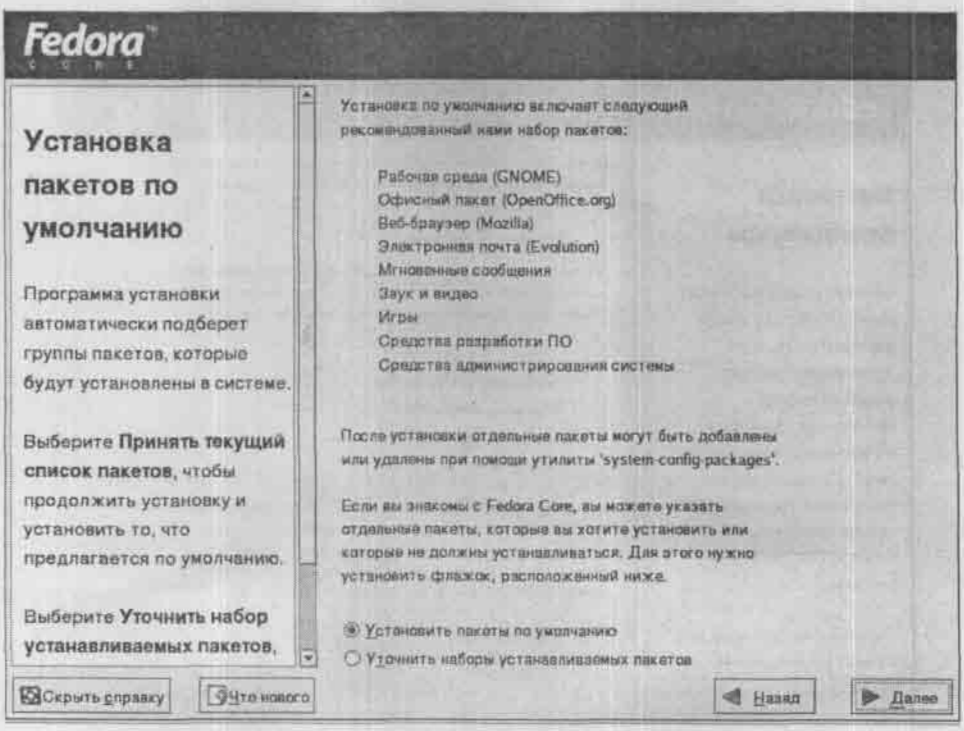


Рис. 1.8. *Выбор групп пакетов*

При индивидуальном выборе пакетов будьте внимательны: некоторые из них для своей работы требуют наличия других пакетов — это называется зависимостью пакетов. Если это первая в вашей жизни установка Linux, не используйте возможность индивидуального выбора пакетов: просто выберите из списка категории программ, которые вас интересуют. После этого останется немного подождать, пока инсталлируются выбранные вами пакеты.

Обязательно создайте загрузочную дискету. Если вы в очередной раз переустановите ОС Windows, она поместит в MBR собственный загрузчик, который и не подумает загружать вашу ОС Linux. В таком случае загрузитесь с загрузочной дискеты, зарегистрируйтесь как пользователь root и введите команду `grub` (или `lilo`, если вы используете LILO).

Для создания загрузочного диска вам понадобится одна отформатированная дискета.

Я рекомендую использовать графический вход в систему: при этом система X Window (графический интерфейс) будет загружаться автоматически, и вам не нужно будет запускать ее командой `startx`.

Все, установка завершена. Извлеките компакт-диск из привода и перезагрузите компьютер.

Загрузчик предложит вам список установленных у вас операционных систем. Выберите из него Linux. Если у вас двухпроцессорная машина, то следует выбрать пункт не Linux, а Linux-smp — это SMP-ядро, поддерживающее несколько процессоров.

1.4. Особенности установки Mandrake 10.0

Относительно недавно вышла версия 10.1 этого замечательного дистрибутива. В ее состав вошли средства для работы с WiFi и Bluetooth-устройствами, драйверы для ноутбуков Intel Centrino (на этой платформе производится большая часть ноутбуков). Что же касается программного обеспечения, входящего в дистрибутив, то в нем вы найдете:

- Ядро версии 2.6.8.
- Новые версии графических интерфейсов KDE 3.2.3 и GNOME 2.6.
- Компилятор gcc версии 3.4.1.
- Web-сервер Apache версии 2.0.50 и интерпретатор PHP 4.3.8.
- СУБД MySQL версии 4.0.18.
- Новую версию популярного офисного пакета OpenOffice — 1.1.3.

Загрузите с компакт-диска программу-инсталлятор. Когда появится экран приветствия, вы можете нажать <Ввод> для начала установки в обычном режиме или <F1> для выбора из списка дополнительных режимов (рис. 1.9).

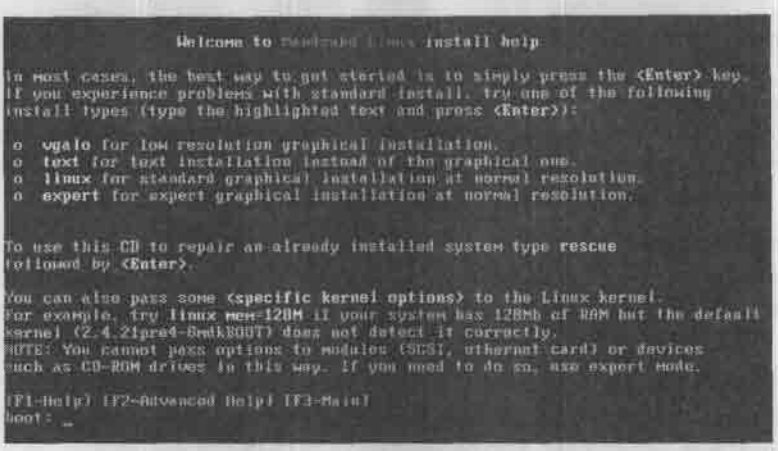


Рис. 1.9. *Дополнительные режимы установки*

Эти режимы включают:

- **vgalo** — если у вас слабенькая видеоплата и вам нужно произвести установку системы при разрешении 640x480 (16 цветов);
- **text** — если вы хотите, чтобы программа установки работала в текстовом режиме;
- **linux** — самая обыкновенная установка;
- **expert** — режим эксперта (вы должны хорошо разбираться в «железе», чтобы устанавливать систему в этом режиме);
- **rescue** — если вы хотите восстановить уже установленную систему.

Вы также можете ввести дополнительные параметры, например,

```
linux mem=512M
```

После нажатия Enter будет запущена программа установки в нормальном графическом режиме, и система будет знать, что у вас установлено 512 Мб оперативной памяти.

В отличие от предыдущей версии Linux Mandrake, выбор класса установки ограничен двумя пунктами:

- Установка (Install)
- Обновление (Upgrade).

Если у вас установлен Linux Mandrake версии 8.1 или выше, вы можете его обновить до версии 10.0. Если же у вас установлена версия Mandrake древнее 8.1 или вообще другой дистрибутив, нужно выбрать пункт «Установка».

При выборе пункта «Установка» старая версия Linux Mandrake, если она была у вас установлена, будет уничтожена. В то же время, если вы хотите

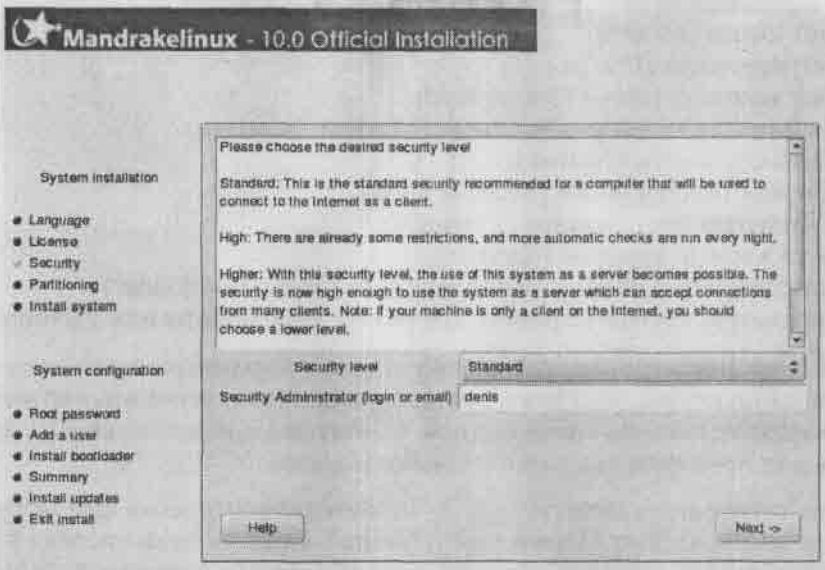


Рис. 1.10. *Выбор уровня безопасности*

изменить разбиение диска или тип файловой системы Linux, вам нужно выбрать именно этот пункт.

После выбора раскладки клавиатуры инсталлятор Linux Mandrake (который, кстати, называется DrakeX) предложит вам установить уровень безопасности. Помните, чем выше уровень безопасности, тем неудобнее работать в системе пользователю. Для домашней машины (на которой вы будете экспериментировать) подойдет средний уровень, а для сервера сети не лишним будет и повышенный. Не устанавливайте параноидальный уровень безопасности при установке системы: вам потом будет трудно ее настроить. Данный уровень подойдет для систем, относящихся к классу «поставил и забыл»: программных маршрутизаторов или шлюзов, администрирование которых не выполняется или выполняется очень редко. Обычно такие системы таятся где-нибудь в углу серверной комнаты и работают круглосуточно, например, передавая пакеты из одной локальной сети в другую.

Вы сможете полностью настроить систему, не дожидаясь для этого перезагрузки (рис. 1.9). Настройке поддаются:

- Раскладка клавиатуры (Keyboard)
- Страна (Country/Region)
- Часовой пояс (Timezone)

- Тип мыши (Mouse)
- Тип принтера (Printer)
- Тип звуковой платы (Sound card)
- Параметры видео подсистемы (Graphical interface)
- Параметры сети (Network)
- Уровень безопасности (Security level)
- Параметры брандмауэра (Firewall)
- Загружаемые сервисы (Services)
- Конфигурация начального загрузчика Linux (Bootloader)
- Параметры других устройств, например, ISDN-платы или TV-тюнера.

Обычно инсталлятор правильно определяет параметры видеосистемы вашего компьютера, но в некоторых случаях нужно уточнить тип вашей видеокарты, ее модель, тип монитора. Обязательно нажмите кнопку **Проверить** для проверки выбранного видеорежима.

Если вы хотите запускать систему X Window автоматически при загрузке Linux, нажмите кнопку **Параметры** (Options). Если же ваша система будет использоваться в качестве сервера, автоматическая загрузка X Window не нужна.

В параметрах сети вы можете попросить инсталлятор автоматически распознать ваши сетевые параметры, если это возможно. Если же он не

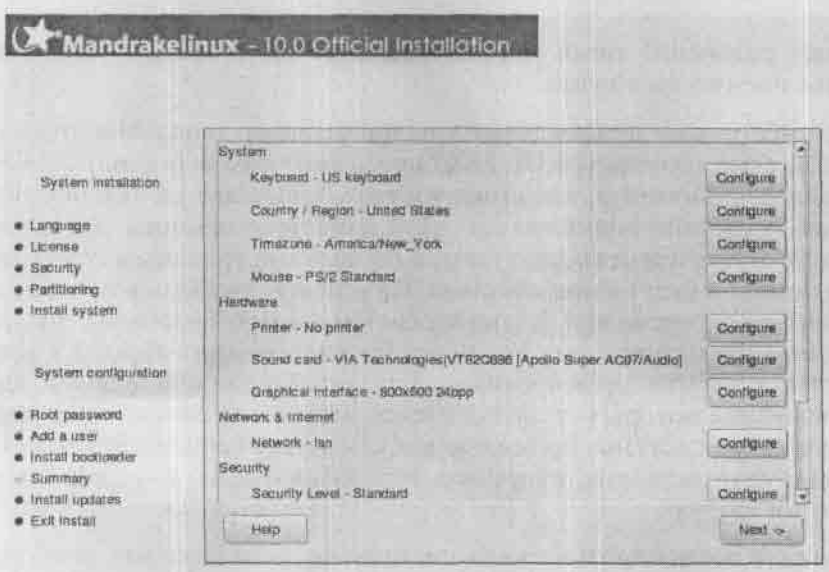


Рис. 1.11. Настройка параметров системы

определил параметры вашей сетевой платы или модема автоматически, попробуйте настроить эти сетевые устройства в режиме эксперта. При этом вам нужно знать следующие параметры:

- IP-адрес этого компьютера и его имя, которое должно быть прописано на сервере DNS — эту информацию вы можете уточнить у администратора вашей сети;
- IP-адрес шлюза;
- IP-адрес сервера DNS.

Инсталлятор позволяет настроить традиционное соединение по модему, по локальной сети, ISDN/ADSL-соединения и даже выделенное соединение.

Наконец, вы сможете даже заняться администрированием системы прямо в процессе установки (рис. 1.12), отключив не нужные пока сервисы, чтобы система загружалась быстрее.

Вы не знаете, что именно стоит отключить? Думаю, в этом вам поможет таблица 1.3 (см. п. 1.9).

Откажитесь от автоматического обновления системы, и установка завершена. Нажав кнопку **Дополнительно**, вы можете создать дискету для клонирования Linux. Эта дискета может понадобиться, если у вас есть

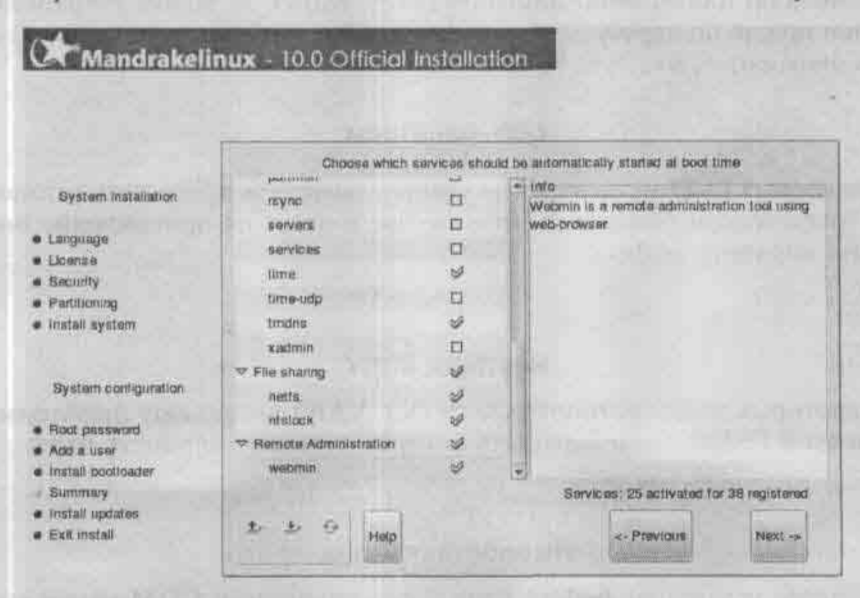


Рис. 1.12. *Выбор служб, запускаемых при загрузке системы*

несколько одинаковых компьютеров, на которые нужно установить Linux. Я же рекомендую создать эту дискету, даже если у вас всего один компьютер — дискета для клонирования существенно облегчит переустановку системы на этом компьютере. Конечно, вы не будете переустанавливать Linux так же часто, как Windows, но кто его знает?

Чтобы использовать дискету клонирования, загрузитесь с первого CD, когда увидите приветствие программы установки, нажмите <F1> и введите `defcfg="floppy"`.

1.5. Проблемы при установке

1.5.1. Конфликты Fedora Core 1 и 2 с различным оборудованием

Материнские платы ASUS

На материнскую плату ASUS серии P4P800 Fedora Core 2 не устанавливается. Пока данная проблема не решена. Остается сменить либо материнскую плату, либо дистрибутив. Следить за ходом устранения ошибки можно по адресу `https://bugzilla.redhat.com/bugzilla/show_bug.cgi?id=121819`.

LCD-мониторы

На некоторых LCD-мониторах не удается запустить программу установки в графическом режиме. В этом случае в ответ на приглашение boot введите параметр nofb:

```
linux nofb
```

Ноутбуки SONY

На некоторых моделях ноутбуков SONY VAIO возникают проблемы с установкой Fedora Core 2, решить которые поможет параметр ядра:

```
linux pci=off ide1=0x180,0x386
```

Не работает мышь

В процессе установки Fedora Core 2 обыкновенные COM-мыши (последовательные) не работают. Они начнут работать после завершения установки, а пока обходитесь клавиатурой или подключите PS/2-мышь.

1.5.2. Fedora Core: не удается войти в систему как root в графическом режиме

Включена система безопасности SELinux. При установке некоторым файлам в домашнем каталоге root назначается ошибочный контекст безопасности. Для исправления этой ошибочки зарегистрируйтесь в консоли как root и введите команду:

```
setfiles /etc/security/selinux/file_contexts /root
```

1.5.3. Ошибка Signal 11

Возникает при сбое шины памяти вашего компьютера. Это может произойти, если ваше аппаратное обеспечение не поддерживается Linux или драйвер для этого аппаратного обеспечения работает некорректно. Список поддерживаемых аппаратных средств находится по адресу `http://bugzilla.redhat.com/hwcert`, а по адресу `http://www.bitwizard.nl/sig11` вы сможете более подробно прочитать об ошибке Signal 11.

1.5.4. Не определяется мышь

Если программа установки не смогла определить мышь, вы увидите соответствующее сообщение. При этом вам будет предложено продолжить установку в текстовом режиме, который не требует мыши. Проверьте, работает ли мышь вообще (на другом компьютере).

1.5.5. Проблемы с переходом в графический режим

Программа установки Linux работает в графическом режиме. Если у вас старая видеокарта, не поддерживающая разрешения 800x600, или старый монитор, вам придется запускать инсталлятор в режиме с низким разрешением. В ответ на приглашение `boot:` введите `lowres` — программа установки будет использовать разрешение 640x480.

1.5.6. Проблемы при загрузке

ОС Linux может использовать разделы, которые расположены дальше 1024-го цилиндра, но не может загружаться с таких разделов. Поэтому, если вы не можете создать Linux-раздел «ближе» к началу диска, создайте небольшой раздел для точки монтирования `/boot` (не более 100 МБ), который нужно разместить до 1024-го цилиндра. Второй вариант — загрузка с дискеты.

1.5.7. Проблемы с графическим экраном загрузчика

1. GRUB: отредактируйте файл `/boot/grub/grub.conf`, закомментировав строку splashimage.
2. LILO: отредактируйте файл `/etc/lilo.conf`, удалив строку message=/boot/message. Изменения вступят в силу после введения команды `lilo`.

1.5.8. Не загружается система X Window

Зарегистрируйтесь в системе и введите команду `startx`. Если при запуске X Window возникла ошибка, запустите конфигуратор системы X Window.

1.5.9. Я забыл пароль пользователя root

Бывает... Перезагрузите компьютер и в строке приглашения загрузчика lilo введите linux single. Будет запущен однопользовательский режим. Для изменения пароля введите команду `passwd root`.

1.5.10. У меня больше оперативной памяти!

Иногда Linux не может точно определить объем оперативной памяти. Например, у вас установлено 128 Мбайт, а Linux видит только 64. Для исправления этого в файл `/etc/lilo.conf` допишите строку:

```
append="mem=128M"
```

Для того, чтобы изменения вступили в силу, введите команду `lilo`.

1.5.11. Не работают принтер или звуковая плата

В более ранних версиях Red Hat Linux настройка принтера происходила при установке операционной системы. Теперь принтер и звуковую плату нужно конфигурировать отдельно. В Red Hat Linux для этого служат графические конфигураторы **redhat-config-printer** и **sndconfig** соответственно. В Fedora Core — **system-config-printer** и **system-config-soundcard**.

1.6. Как удалить Linux

Любой дистрибутив Linux удаляется в два этапа:

1. Удаление загрузчика Linux. Проще всего восстановить оригинальный загрузчик Windows 98, введя команду **fdisk /mbr** в командной строке

Windows. Если у вас Windows 2000/XP, восстановить оригинальный загрузчик помогут команды **fixboot** и **fixmbr** — именно в таком порядке. Удалить загрузчик LILO можно и с помощью самого LILO: зарегистрируйтесь как пользователь root и введите команду `lilo -U`.

2. С помощью программы установки Linux удалите разделы Linux. Для этого выберите тип установки «Установка» (Install), затем выберите ручное разбиение диска, удалите Linux-разделы, сохраните таблицу разделов и перезагрузите машину, нажав Reset.

1.7. Системы с двойной загрузкой

Прежде чем устанавливать на своем компьютере вторую операционную систему, обязательно сделайте резервное копирование важных данных, потому что процесс установки операционной системы включает в себя переразбиение жесткого диска на разделы и форматирование созданных разделов. Переразбиение жесткого диска необходимо, потому что Linux и Windows используют разные файловые системы. Хотя существуют средства установки Linux в раздел FAT/FAT32 — Lin4Win, но я не рекомендую их использовать, т.к. в этом случае Linux работает крайне нестабильно и медленно.

Рассмотрим два наиболее распространенных варианта установки нескольких операционных систем:

1. Вы устанавливаете Windows 9x и Linux.
2. Вы устанавливаете Windows NT, Windows 9x и Linux.

1.7.1. Установка Windows 9x и Linux

Главное правило при такой установке заключается в том, что сначала следует устанавливать Windows 9x, и только после этого Linux. Дело в том, что при установке Windows 9x перезаписывает главную загрузочную запись MBR (*Master Boot Record*) и загрузить Linux с диска вы уже не сможете. При этом очень напрашивается следующий совет: создайте загрузочную дискету для Linux! Если вы в очередной раз переустановите Windows, без этой дискеты загрузить Linux не удастся.

Что делать, если вы переустановили Windows и Linux больше не загружается:

- У вас есть загрузочная дискета для Linux. Загрузитесь, используя эту дискету. Затем зарегистрируйтесь в системе как пользователь root и введите команду `lilo`. Затем перезагрузите машину (команда `reboot`).

- У вас нет загрузочной дискеты: используйте программу **loadlin** — ее описание будет приведено ниже.
- Нет загрузочного диска и нет программы **loadlin**: переустановите Linux — после установки Windows, естественно. При установке Linux не форматируйте разделы, и ваши данные останутся неповрежденными.

Оптимальным загрузчиком при данной схеме установки (Windows 9x + Linux) будет LILO. Я бы не рекомендовал использование каких-либо посторонних загрузчиков, как бы стабильно они ни работали. Во время инсталляции Linux программа установки спросит, куда устанавливать LILO — выберите MBR.

В случае деинсталляции Linux (после того, как вы уже удалили разделы Linux) восстановить MBR, то есть удалить LILO, поможет команда `fdisk /mbr`. При этом нужно использовать программу **fdisk** из комплекта загрузочного диска Windows.

1.7.2. Установка Windows 9x, Windows NT/2000/XP и Linux

Первый способ

В этом способе используется загрузчик **NTLoader**. Выполните установку Windows 2000 в раздел NTFS, а Windows 9x — в раздел FAT/FAT32. Не забудьте приготовить четыре системных дискеты для восстановления! Напомню, что если вы хотите установить Windows 95, то ее нужно устанавливать первой, а потом Windows 2000. При установке Windows 98 и Windows 2000 порядок установки не имеет значения.

Устанавливать Linux нужно после установки Windows 2000. При этом необходимо учесть, что раздел Linux должен находиться до 1024 цилиндра! Это связано с «барьером 1024 цилиндра» — Linux может использовать разделы, расположенные после цилиндра 1024, но не может загружаться с таких разделов. В последних версиях Linux данная проблема устранена, но рассматриваемый способ установки требует, чтобы вы создали раздел Linux до цилиндра 1024 — иначе вам придется загружать Linux с дискеты.

Еще раз напоминаю: обязательно создайте загрузочную дискету для Linux. При установке LILO выберите MBR — Linux перезапишет главную загрузочную запись. Далее загрузите с четырех загрузочных дискет Windows 2000 и выберите пункт «Recover» в меню загрузчика и режим «Command mode». Затем зарегистрируйтесь в системе как Administrator. Выполните команды **fixboot** и **fixmbr** — теперь Windows 2000 будет нормально загружаться.

Примечание

Утилиты **fixboot** и **fixmbr** используются в Windows 2000 для восстановления главной загрузочной записи (MBR). Команда fixmbr практически аналогична команде **fdisk /mbr** в Windows 9x.

Затем загрузитесь с системной дискеты Linux и войдите в систему как root. Откройте в любом текстовом редакторе файл `/etc/lilo.conf`. В начале файла есть ссылка на загрузочный раздел по умолчанию, например, `/dev/hda`. Вам нужно изменить это значение на диск и раздел, в который была установлена ОС Linux, например `/dev/hdb1`.

Введите команду `lilo` и увидите, что загрузочный раздел не является первым на диске — именно это вам и нужно. В этом случае загрузочная запись Windows 2000 не будет повреждена. Запишите загрузчик Linux в файл `/bootsect.lnx`:

```
# dd if=/dev/hdb1 bs=512 count=1 of=/bootsect.lnx
```

Теперь этот файл нужно скопировать на дискету:

```
# mount -t msdos /dev/fd0 /mnt
# copy /bootsect.lnx /mnt
# umount /mnt
```

Перезагрузите Linux командой `reboot` и загрузите Windows 2000. Скопируйте файл `bootsect.lnx` в корневой каталог диска C: и присвойте ему атрибут *read-only*. Добавьте строку в файл `boot.ini`:

```
C:\ bootsect.lnx ="Linux"
```

В результате при перезагрузке компьютера вы сможете загрузить Linux с помощью **NTLoader**.

Второй способ

Существует более простой способ установки Linux и любой операционной системы семейства Microsoft Windows — Windows 9x, NT, 2000. Сначала нужно установить все ОС Windows, а потом установить Linux. При этом вместо начального загрузчика будет использоваться не **NTLoader**, а LILO. В этом случае вы получите двойное меню: сначала нужно выбрать между загрузкой Linux и Windows, а потом выбрать нужную вам ОС Windows — Windows 9x или NT/2000 — в зависимости от того, какую систему вы установили, кроме Windows NT. Второе меню — это как раз меню загрузчика **NTLoader**.

Использование loadlin

В этой книге я просто не мог не упомянуть о компактном загрузчике **loadlin**, который позволяет загрузить Linux из-под DOS или Windows 95.

Если вы используете Windows 98, **loadlin** работать у вас не будет — нужно перезагрузиться в режиме MS DOS. Кроме этого, следует учесть, что при использовании **loadlin** могут возникнуть проблемы с разделами FAT32. В этом случае создайте загрузочную дискету DOS (`format a: /s`) и в autoexec.bat этой дискеты пропишите **loadlin**. Загрузить Linux из-под операционных систем Windows 98, Me или Windows 2000/NT вам не удастся в любом случае.

Использовать **loadlin** я рекомендую лишь в том случае, если при очередной переустановке Windows оказался «затертым» MBR (а вместе с ним и LILO), а системную дискету Linux вы не создали. Вызов программы **loadlin** имеет следующий формат (описание параметров приведено в табл. 1.2):

```
loadlin <ядро> <root=корневая_ФС> <опции>
```

или

```
loadlin @файл_с_параметрами
```

Параметры программы **loadlin** *Таблица 1.2*

Параметр	Описание
Ядро	Ядро, которое вы используете. Если у вас на данный момент нет именно того ядра, которое было установлено, можно взять практически любое (лучше не самое древнее). Обычно ядро можно взять на компакт-диске с дистрибутивом Linux. Оно вам потребуется всего один раз — нужно только загрузиться и перезаписать lilo (команда lilo)
root= корневая_ФС	Корневая файловая система, например root=/dev/hda3
опции	Опции, которые будут переданы ядру во время загрузки. Обычно используется ro vga=normal
@файл_с_параметрами	Если параметры, которые вы передаете loadlin (а не ядру!) окажутся слишком длинными, то их можно записать в отдельный файл, а затем указать его имя в качестве параметра

Совет: можно включить загрузку Linux в стартовое меню DOS. Для этого отредактируйте свой `config.sys` следующим образом (см. листинг 1.1).

Листинг 1.1. Фрагмент файла config.sys

```
MENU]
MENUITEM=DOS, Load DOS
MENUITEM=LINUX, Load Linux
MENUDEFAULT=DOS,5

[LINUX]
install=c:\loadlin\loadlin.exe c:\loadlin\kernel\vmlinuz
root=/dev/hda3 ro vga=normal

[DOS]
; ваш предыдущий config.sys
```

1.8. Первый запуск Linux

Если вы выбрали для входа в систему графический режим (или инсталлятор выбрал его сам, ни о чем не спросив), то вы увидите графический экран с полем ввода имени и пароля. Зарегистрируйтесь в системе (лучше под тем именем, которое завели для обычного пользователя; используйте root, только если обычных учетных записей у вас еще нет), и вы увидите рабочий стол той оконной среды, которую установили в качестве среды по умолчанию, внешне довольно похожий на рабочий стол Windows.

Как же так, вы ведь слышали, что настоящие линуксоиды работают в среде командной строки? Сейчас найдем и командную строку.

Вы знаете, что консолью, или терминалом (для персонального компьютера эти понятия — синонимы) называются устройства ввода-вывода, предназначенные для общения системы с пользователем, то есть клавиатура и монитор. В UNIX-подобных системах существует понятие виртуальных консолей — консолей, по очереди занимающих физически одни и те же монитор и клавиатуру. На каждой из них может быть открыт отдельный сеанс работы пользователя, запущены свои приложения, в общем, они представляют собой почти независимые друг от друга вычислительные системы.

В большинстве дистрибутивов Linux по умолчанию обслуживается шесть текстовых виртуальных консолей, седьмая — графическая. На ней вы и находитесь. Чтобы переключиться с нее на первую текстовую консоль, нажмите комбинацию клавиш Ctrl+Alt+F1 (на шестую — Ctrl+Alt+F6).

В ответ на приглашение программы **login**: введите root и нажмите <Ввод>. Потом введите пароль, и ваш сеанс работы в текстовой консоли начат.

Информацию о сетевом имени машины, версии ОС, архитектуре можно получить по команде `uname -a`.

Если учетной записи для обычного пользователя у вас еще нет, самое время ее завести. Введите команду

```
#useradd < имя >
```

Если вам ответили «command not found», то, значит, в вашей системе эта команда называется `adduser`.

Вы добавили нового пользователя. Однако это еще не все, нужно указать его пароль:

```
#passwd < имя >
```

Теперь можно регистрироваться под новым именем на другой виртуальной консоли. Чтобы с текстовой консоли переключиться на другую

текстовую, нажмите комбинацию Alt+Fn, где n — число от 1 до 6. Вы снова увидите приглашение login:.

Зарегистрировались? Обратите внимание на строку приглашения. На той консоли, где вы зарегистрировались как root, она оканчивается символом #, а для любого обычного пользователя — символом $. Кроме этого символа, приглашение обычно состоит из имени пользователя, имени системы и текущего каталога, причем вид его можно изменить, как только вы узнаете, как это делать. В дальнейших примерах строки, начинающиеся с # или $, будут обозначать вводимую команду, а строки без такого символа — ее сообщения.

Теперь убедитесь сами, что Linux — действительно многозадачная и многопользовательская система, то есть, в отличие от Windows, несколько пользователей могут работать одновременно. Спросите, кто сейчас работает в системе, введя команду who.

Вы увидите что-то вроде:

```
root    tty1    <дата и время начала сеанса root>
ivan    tty2    <дата и время начала сеанса ivan>
root    :0      <дата и время начала сеанса root>
                <на графической консоли>
```

ttyN — это номер виртуальной текстовой консоли.

Если вы потерялись и хотите узнать, на какой консоли находитесь сейчас, введите команду `tty`. Если вы забыли, под каким именем зарегистрировались на текущей консоли, введите команду `whoami`. Команда `w` покажет не только работающих сейчас пользователей, но и запущенные ими задачи.

Листать экран можно комбинациями клавиш Shift+PgUp и Shift+PgDn.

Для копирования текста в командную строку используйте мышь: протаскивание мыши при нажатой левой кнопке выделяет фрагмент, щелчок правой кнопки вставляет его в текущую позицию курсора на любой виртуальной текстовой консоли.

Если вы хотите, не сходя с этой консоли, поработать под именем другого пользователя, введите `su <имя>`. По умолчанию в качестве имени подставляется root. Обычно это нужно для того, чтобы быстро выполнить какое-то администраторское действие. Возвращайтесь к работе под своим именем по команде `exit`.

Чтобы переключиться обратно на графическую консоль, нажмите Alt+F7.

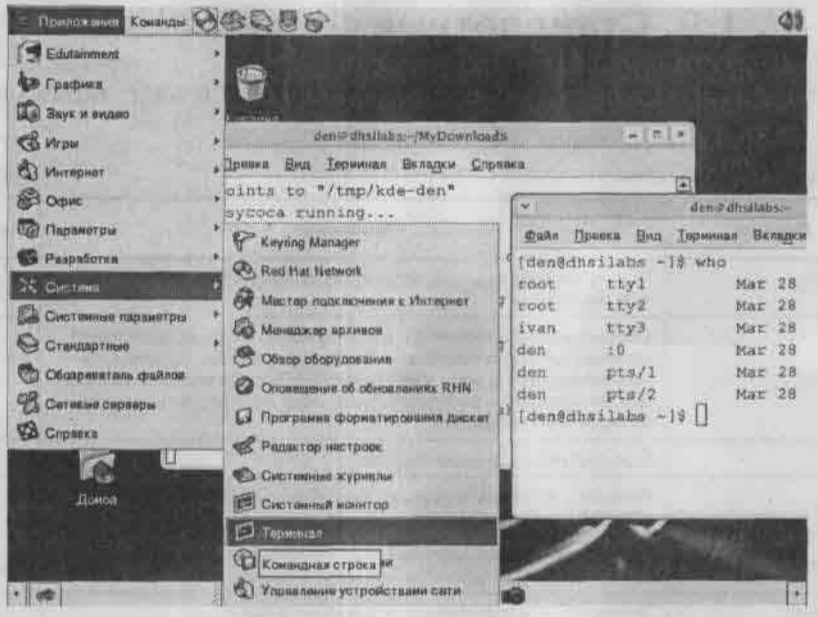


Рис. 1.13. *Окно виртуального терминала*

В графическом режиме тоже можно не только щелкать мышью по значкам, но и вводить команды. Для этого запустите виртуальный терминал (рис. 1.13) — графическое приложение, в окне которого можно работать в режиме командной строки.

Количество виртуальных терминалов, в отличие от количества виртуальных консолей, не ограничено ничем, даже традицией.

Чтобы завершить сеанс работы на виртуальном терминале или на виртуальной консоли, введите команду `exit` (на виртуальной консоли можно также `logout`) или нажмите комбинацию клавиш Ctrl+D.

Завершение работы одного пользователя не приводит к остановке всей системы. Чтобы выключить машину, нужно с правами суперпользователя отдать команду

```
# shutdown -h 19:00 [Конец рабочего дня]
```

При этом за несколько минут до указанного времени всем, кто работает в системе, будет разослано предупреждающее сообщение «Конец рабочего дня», после чего система будет корректно остановлена. Краткой формой этой команды служит `halt`, останавливающая систему сейчас же и без предупреждения. Перезагрузиться можно командой `reboot`.

1.9. Стандартные сервисы Linux

Описание стандартных сервисов Linux приведу в виде таблицы (см. табл. 1.3).

Стандартные сервисы Linux *Таблица 1.3*

Сервис	Описание	Нужен?
anacron, crond	Планировщики заданий, запускающие по графику определенные программы	Да
apmd	Отслеживает состояние батарей ноутбука. У вас стационарный компьютер? Тогда смело отключайте этот сервис. Конечно, он вам понадобится, если у вас умный ИБП: тогда apmd может корректно завершить работу системы, если пропало напряжение, а батарея ИБП разряжена	
atd	Еще один планировщик заданий	
autofs	Средство автоматического монтирования сменных носителей (CDROM, дискета)	Да
chargen, chargen-up	Внутренний сервис демона xinetd, генерирующий символы с определенным интервалом времени и посылающий его по сети	
cpuspeed	Динамическое управление скоростью процессора	
cups	Система печати в Unix-подобных ОС (Common Unix Print System). Принтер у вас есть? Тогда она нужна	
cups-config-deamon	Демон для конфигурирования cups	
diald	Автоматический дозвон к провайдеру	
gpm	Поддержка мыши в консоли Linux	Да
gssftp	FRP-сервис, входящий в состав суперсервера xinetd (гл.11) и использующий авторизацию на Kerberos-сервере	
irda	Поддержка работы с инфракрасным портом	
irqbalance	Осуществляет балансировку прерываний на многопроцессорных системах	
internet	Устанавливает соединение с провайдером при загрузке системы и обрывает его при завершении работы	
iptables	Межсетевой экран	
haldaemon	Собирает информацию об оборудовании	
keytable	Загружает раскладку клавиатуры, указанную в файле /etc/sysconfig/keyboard	Да
klogd	Протоколирует сообщения ядра в файле /var/log/kernel	Да
kudzu	Ищет изменения в аппаратной конфигурации компьютера. Вполне может запускаться вручную. Обычно запуск kudzu необходим один раз — после установки системы, потом его можно отключить и запускать только при установке новых устройств в компьютер (или удалении ранее установленных)	
lisa	Демон, собирающий информацию о ресурсах сети Microsoft	
linuxconf	Организовывает выполнение различных задач конфигурирования при загрузке системы	

Продолжение табл. 1.3

Сервис	Описание	Нужен?
mdmonitor	Мониторинг и управление программным RAID	
named	Сервер DNS	
netfs	Обеспечивает монтирование удаленных сетевых файловых систем (NFS, SMB, NCP (NetWare))	
network	Поддержка сети. Этот сервис должен быть включен всегда, поскольку даже графическая система и система в печати в Linux нуждаются в нем	Да
nfslock	Механизм блокирования файлов NFS	
numlock	Включение режима клавиатуры Num Lock	
oki4daemon	Поддержка OKI 4w-принтеров	
pcmcia	Поддержка устройств PCMCIA	
portmap	Необходим для приложений, использующих удаленный вызов процедур RPC. Нужен для NFS и NIS (Network Information Service)	
psacct	Служба учета процессов	
httpd, proftpd, wu-ftpd, sendmail, postfix	Web-, FTP- и почтовые серверы. Пока включать их не нужно, поскольку они не настроены и будут дополнительной «дырой» в безопасности вашей системы	
random	Улучшает качество генерации случайных чисел	Да
rawdevices	Не вдаваясь в технические подробности, просто отключите этот сервис	
readahead, readahead_early	Сервисы, позволяющие выполнять предварительную загрузку программ в оперативную память	
rhncd	В дистрибутивах Red Hat и Fedora — демон, который периодически подключается к серверу Red Hat на предмет обновлений	
smartd	Мониторинг жеских дисков, с использованием протокола S.M.A.R.T	
saslauthd	Обработка запросов аутентификации	
spamassassin	Сервис, предназначенный для борьбы со спамом	
sshd	Сервер, предназначенный для безопасного удаленного администрирования системы	
syslog	Служба ведения системных журналов	
smb	Если вы планируете работать в сети Microsoft, включите этот сервис, но только после его настройки, описанной в административном разделе. А пока отключите	Да
winbind	Используется для аутентификации Samba-пользователей в домене NT	
xfs	Сервер шрифтов, нужен для системы X Window	Да
named	Сервер DNS	
ypbind	Служба, используемая для обеспечения доступа к NIS-серверу	
yum	Сервис, осуществляющий обновление системы в автоматическом режиме	

Узнать, какие сервисы установлены у вас в системе, можно воспользовавшись системной утилитой **system-config-services** (пункт меню **Системные параметры → Настройка сервера → Службы**) — это в Fedora Core. Кроме того, можно воспользоваться консольной утилитой **ntsysv**.

1.10. Справочная система

«Родной» для UNIX справочной системой, аналогом Справки Windows, служат страницы интерактивного руководства (*manual pages*, man-страницы). Ими снабжена практически каждая программа, важный конфигурационный файл или системный вызов. Это обычные текстовые файлы, содержащие сведения о назначении, синтаксисе, опциях команды, формате файла, примеры их применения, имена и URL разработчиков. Обычный дистрибутив содержит тысячи страниц такой документации. Просматривают ее командой **man**. Например, `man man` — это вызов справки о самой команде `man`.

При установке каждого программного пакета включенные в его состав man-страницы копируются в стандартные каталоги, так что программа **man** легко их находит. В этих каталогах страницы сгруппированы в секции по темам. Перечислим эти секции в порядке просмотра программой **man**:

1	Команды пользователя
8	Системные команды
2	Системные вызовы
3	Библиотечные вызовы (подпрограммы)
4	Устройства
5	Форматы файлов
6	Игры
7	Разное
9	Ядро (kernel internals)

Так что, если вас интересует справка не о консольной команде `exit`, а о системном вызове *exit*(), запрашивайте ее командой `man 2 exit`.

Чтобы ознакомиться с содержанием раздела, введите команду

```
man <номер_раздела> intro.
```

Вот несколько самых употребительных команд управления выводом man-справки на экран:

<Ввод>	вывод следующей строки текста
<Пробел>	вывод следующего экрана текста
<Q>	выход из программы
</>образец<Ввод>	поиск образца
<N>	повторение поиска.

Альтернативой команде man служит гипертекстовая справочная система info. Ссылки обозначены звездочками (*), и перемещает по ним клавиша <Tab>. Чтобы пройти по ссылке, нажмите <Ввод>. Чтобы перейти на следующую или предыдущую страницу, нажмите <N> или <P> соответственно. Чтобы попасть на один уровень иерархии страниц вверх, нажмите <U>. И клавиша <Q> завершает работу с программой info.

Сверхкраткую (одна строка) справку о командах дает команда

```
whatis <ключевое_слово>
```

Для работы ей нужна база данных о системных командах, создаваемая программой **makewhatis**.

Если точного имени команды вы не знаете, но примерно догадываетесь, какие фрагменты слов могут встретиться в ее имени или кратком описании, вам поможет команда **apropos**, производящая поиск в той же базе по фрагментам слов. Например, отдав команду

```
$ apropos config
```

вы получите список команд, имеющих отношение к конфигурированию системы и различных служб.

Команда **man** с ключом **–k** тоже умеет выполнять поиск по ключевым словам. Если вы хотите узнать, какие программы имеют отношение к некоторому системному файлу (конфигурационному или журнальному, например, `/etc/passwd`), введите команду

```
$ man -k passwd
```

Вы получите список man-страниц по командам, функциям, формата файлов и т.п., на которых встречается слово «passwd». В общем, поиск информации в справочной системе UNIX очень похож на поиск в Интернете: находите что-то похожее и переходите по ссылкам все ближе к истине.

Есть и команда **help**: она выводит справку о встроенных командах командной оболочки (о командных оболочках сказано в гл.3).

Наконец, правильно написанные команды понимают ключи **--help** и **--usage**, с которыми выдают краткую справку о себе.

К сожалению, многие страницы документации на русский язык пока не переведены. Linux — ОС некоммерческая, и переводом занимаются добровольцы на общественных началах. Может быть, со временем к ним присоединитесь и вы? А пока загляните на

`http://www.linuxshare.ru/projects/trans/index.html`.

Глава 2

ФАЙЛОВАЯ СИСТЕМА LINUX

ВИДИМАЯ СТОРОНА ФАЙЛОВОЙ СИСТЕМЫ

ИЗНАНКА ФАЙЛОВОЙ СИСТЕМЫ

СОЗДАНИЕ И МОНТИРОВАНИЕ ФАЙЛОВЫХ СИСТЕМ

2.1. Видимая сторона файловой системы

С точки зрения пользователя, файловая система — это логическая структура каталогов и файлов. В отличие от Windows, где каждый логический диск хранит отдельное дерево каталогов, во всех UNIX-подобных системах эта древовидная структура растет из одного корня: она начинается с корневого каталога, родительского по отношению ко всем остальным, а физические файловые системы разного типа, находящиеся на разных разделах и даже на удаленных машинах, представляются как ветви этого дерева.

2.1.1. Имена файлов и каталогов

Имена файлов и каталогов могут иметь длину до 255 символов. Символы «/» (слэш) и символ с кодом 0 запрещены. Кроме того, ряд символов имеет специальное значение для командного интерпретатора, и их использование не рекомендуется. Это символы:

```
~ ! @ # $ & % * ( ) [ ] { } ' " \ : ; > < ` пробел
```

Если вам все-таки нужно употребить один их этих символов в имени файла, то при указании его имени в команде этот символ нужно экранировать знаком «\» (обратный слэш) или заключать все имя в двойные кавычки. Например, вы хотите вывести на экран командой `cat` содержимое текстового файла с именем `my file`, содержащим символ пробела:

```
$ cat my file # неправильно
cat: my: No such file or directory
cat: file: No such file or directory

$cat "my file" # правильно
Привет!
$cat my\ file # правильно
Привет!
```

Знак # — это символ комментария для командного интерпретатора **bash**, о котором подробнее рассказано в главе 8. Командный интерпретатор

игнорирует все символы от этого знака до конца введенной строки. Возможность комментировать не слишком полезна при вводе команд с консоли, но неоценима при написании командных сценариев, о которых мы еще будем говорить.

Заметьте, что точки среди специальных символов нет, и имена вроде `this.is.a.text.file.containing.the.famous.string.hello.world` допустимы и широко распространены. Часто последняя отделенная точкой часть имени используется подобно «расширению имени» в Windows, обозначая файл определенного типа, но это обозначение несет смысл только для человека. Так, человеку имя файла `ivan_home.tar.gz` подсказывает, что это домашний каталог пользователя ivan, упакованный архиватором **tar** и сжатый компрессором **gzip**.

Если имя файла начинается с точки, то этот файл считается скрытым: некоторые команды его «не видят». Например, введя в своем домашнем каталоге команду просмотра содержимого каталога **ls** с ключом `-a`, означающим «показывать скрытые файлы», вы увидите больше файлов, чем введя ту же команду без ключей.

Linux различает регистр символов в именах файлов: так, в одном каталоге могут находиться два разных файла `README` и `Readme`. Кстати, файлы с подобными именами обычно содержат информацию, действительно достойную прочтения.

Имена каталогов строятся по точно тем же правилам, что и имена файлов.

Полным именем файла (или путем к файлу) называется список вложенных друг в друга каталогов, заканчивающийся собственно именем файла. Начинаться он может с любого каталога, потому что в древовидной структуре между любыми двумя узлами существует путь. Если этот список начинается с корневого каталога, то путь называется абсолютным. Если с любого другого — то относительным (по отношению к этому каталогу).

Корневой каталог обозначается символом «/» (слэш), и этим же символом разделяются имена каталогов в списке. Таким образом, абсолютным именем файла `README` в домашнем каталоге пользователя den будет `/home/den/README`.

В каждом каталоге существуют два особых «подкаталога» с именами «две точки» и «точка». Первый из них служит указанием на однозначно определенный родительский каталог, а второй — на сам данный каталог. Для корневого каталога, у которого нет родителя, оба эти «подкаталога» указывают на корневой каталог. С помощью этих имен образуются относительные имена файлов. Так, именем вышеупомянутого файла `README` относительно домашнего каталога `/home/ivan` пользователя `ivan` будет `../den/README`.

2.1.2. Назначение основных системных каталогов

В системных каталогах находятся файлы, необходимые для управления и сопровождения системы, а также стандартные программы. Их имена, расположение и содержание почти одинаковы почти во всех ОС Linux, поэтому эти каталоги называют также стандартными. Впрочем, на данный момент эпитет «стандартные» отражает скорее благие пожелания, чем действительность: иерархия каталогов одинакова только для дистрибутивов, связанных единством происхождения, а исторически сложившиеся различия создают опасность несовместимости разных дистрибутивов. Стандарт файловой иерархии для UNIX-подобных систем разрабатывает группа добровольцев, и с его текущей версией можно ознакомиться по адресу `http://www.pathname.com/fhs`.

Краткое описание основных каталогов линии Red Hat и Fedora Core сведено в табл. 2.1.

Стандартные каталоги *Таблица 2.1*

Каталог	Назначение
/bin	Основные программы, необходимые для работы в системе: командные оболочки, файловые утилиты и т.п.
/sbin	Команды для системного администрирования, а также программы, выполняемые в ходе загрузки
/boot	Файлы, необходимые для загрузки системы (образ ядра)
/home	Домашние каталоги пользователей, кроме root
/dev	Файлы устройств
/etc	Файлы настроек: стартовые сценарии, конфигурационные файлы графической системы и различных приложений
/lib	Системные библиотеки, необходимые для основных программ, и модули ядра
/lost+found	Восстановленные после аварийного размонтирования части файловой системы
/media	Сюда обычно монтируются съемные носители: компакт-диски, flash-накопители
/mnt	Временные точки монтирования жестких дисков. Использовать этот каталог необязательно: подмонтировать файловую систему можно к любому другому каталогу
/opt	Дополнительные пакеты программ. Если программа, установленная сюда, больше не нужна, то достаточно удалить ее каталог без обычной процедуры деинсталляции
/proc	Виртуальная файловая система, дающая доступ к информации ядра (например, выведите на экран файл /proc/cpuinfo). Другие файлы в этом каталоге в каждый момент времени содержат информацию о выполняющихся в этот момент программах
/root	Домашний каталог суперпользователя. Домашние каталоги всех остальных могут находиться на отдельном разделе, но /root должен быть в корневой файловой системе, чтобы администратор всегда мог войти в систему для ремонтных работ
/tmp	Временные файлы
/var	Часто меняющиеся данные: системные журналы и протоколы приложений, замки, почтовые ящики, очереди печати и т.п.
/usr	Практически все остальное: программы, исходные коды, документация. Сюда по умолчанию устанавливаются новые программы

Скорее всего, в ваш дистрибутив включена man-страница `hier`, подробно описывающая назначение этих каталогов и их основных подкаталогов.

2.1.3. Типы файлов

С точки зрения UNIX-подобных ОС, файл представляет собой поток или последовательность байтов. Такой подход позволяет распространить понятие файла на множество ресурсов не только локального компьютера, но и удаленного, связанного с локальным сетью любого рода. Доступ к любому такому ресурсу осуществляется через универсальный интерфейс, благодаря чему запись данных в файл, отправка их на физическое устройство или обмен ими с другой работающей программой происходит аналогично. Это очень упрощает организацию данных и обмен ими.

В ОС Linux можно выделить следующие типы файлов:

- обычные файлы — последовательность байтов (текстовые документы, исполняемые программы, библиотеки и т.п.);
- каталоги — именованные наборы ссылок на другие файлы;
- файлы физических устройств, подразделяющихся на:
  - файлы блочных устройств, драйверы которых буферизуют ввод-вывод с помощью ядра и
  - файлы байт-ориентированных, или символьных, устройств, позволяющих связанным с ними драйверам выполнять буферизацию собственными средствами;
- символические ссылки (symlink, symbolic link);
- именованные каналы (named pipes);
- гнезда (sockets).

Обычные файлы и каталоги

Свойства (атрибуты) файлов и каталогов можно вывести на терминал с помощью команды **ls** с ключом `-l`:

```
$ls -l /home/den/README
-rwxr-xr-- 1 den  users  0 Feb 14 19:08 /home/den/README
```

Что это за свойства?

Первый символ выведенной строки, в данном случае дефис, обозначает тип файла. Другие значения этого свойства: d — каталог, b — блочное устройство, c — символьное устройство, l — символическая ссылка, p — именованный канал и s — гнездо.

Следующие 9 символов означают права доступа к файлу. Они делятся на три тройки, обозначающие права: владельца, членов его группы и всех остальных. Внутри каждой тройки может присутствовать или от-

сутствовать: право чтения (r), записи (w) и исполнения (x, от *execute*). Отсутствие права обозначается символом дефиса. С файлом README из нашего примера владелец (в общем случае, пользователь, создавший его) имеет право делать все, что угодно; члены его группы — только читать и запускать файл на выполнение; все остальные — только читать.

О следующем свойстве, количестве ссылок на файл, будет сказано в параграфе о символических ссылках.

Далее указаны имя владельца файла и имя его группы; размер файла в байтах; дата и время последней модификации и имя файла.

Для каталога вывод команды **ls** выглядит так же, но значение некоторых свойств отличается.

```
$ls -l -a /home/den
drwx------ 3 den  users  4096Feb 14 19:02 .
drwxr-xr-x 4 root root   4096Feb 02 11:32 ..
[...]
```

Ключ -a нужен, чтобы увидеть псевдоподкаталоги «.» и «..» (их имена начинаются с точки, поэтому эти файлы скрыты).

Бит чтения в правах доступа означает право просматривать содержание каталога, записи — право создавать и удалять файлы в каталоге, исполнения — право переходить в этот каталог (делать его своим текущим каталогом).

Файлы физических устройств

Все подключенные к компьютеру устройства понимаются операционной системой как файлы: вывод информации на терминал, печать на принтере, отправка почты — все это, с точки зрения ОС, есть запись в файл. Технически файл устройства — это коммуникационный интерфейс драйвера, ведающего взаимодействием с этим устройством. Большинство таких файлов располагается в каталоге `/dev`.

Загляните в этот каталог, введя команду ls -l /dev. Вывод этой команды займет несколько экранов, что создает повод для знакомства с командой-фильтром more, выводящей поступающие к ней на вход данные по одному экрану за раз:

```
$ ls -l /dev | more
```

Чтобы увидеть следующий экран, нажмите пробел; чтобы прервать работу команды — Ctrl+C. О символе вертикальной черты, разделяющей команды **ls** и **more**, подробнее сказано в п. 3.3.1.

Об именах устройств, соответствующих разделам жесткого диска, сказано в главе 1. В следующей таблице приведена краткая справка по именам

других часто используемых устройств. Символ N означает номер устройства в группе однотипных с ним устройств.

Наиболее часто используемые стандартные имена устройств *Таблица 2.2*

Файл	Устройство
null	Пустое устройство. Все данные, выводимые в него, просто исчезают. Удобно использовать его для вывода ненужных сообщений
console	Системная консоль, т.е. физически подключенные клавиатура и монитор
ttyN	Пользовательская (виртуальная) консоль. Linux поддерживает до 6 таких консолей, для переключения между которыми служит комбинация клавиш Alt+Fn, где n — число от 1 до 6
pts/N	Виртуальный терминал. Программа графического режима, в окне которой можно работать как в консоли
mouseN	Мышь
audioN	Звуковая карта
ttySN	Последовательный порт. Файл /dev/ttyS0 соответствует порту COM1 в MS-DOS, /dev/ttyS1 — порту COM2
lpN	Параллельный порт
cuaN	Специальное устройство для работы с модемом
ethN	Сетевая плата
fdN	Дисковод для гибких дисков. Первому, то есть A:, соответствует /dev/fd0, для B: используется имя /dev/fd1
hdxN	Жесткий диск с интерфейсом IDE, где x — буква, обозначающая номер такого диска (начиная с a), N — номер раздела на диске
sdxN	Жесткий диск с интерфейсом SCSI

Вместо размера файла команда **ls** выдает два числа. Это так называемые старший и младший номера устройства. Старший номер несет информацию о драйвере, к которому относится данный файл, а младший номер указывает, к какому именно из однотипных устройств следует обращаться.

Жесткие и символические ссылки

Жесткая ссылка является просто другим именем для исходного файла. После создания такой ссылки ее невозможно отличить от исходного имени файла. «Настоящего» имени у файла нет, точнее, все такие имена будут настоящими. Команда **ls** показывает количество именно таких жестких ссылок. Удаление файла по любому из его имен уменьшает на единицу количество ссылок, и окончательно файл будет удален только тогда, когда это количество станет равным нулю. Поэтому удобно использовать жесткие ссылки для того, чтобы предотвратить случайное удаление важного файла.

Создадим жесткую ссылку на файл README и посмотрим, что изменилось в его свойствах:

```
$ln /home/den/README /home/den/readme_too
$ls -l /home/den/README
-rwxr-xr-- 2 den  users  0 Feb 14 19:08 /home/den/README
```

Жесткую ссылку можно создавать в любом каталоге, но обязательно на том же физическом носителе (то есть в той же файловой системе), что и исходный файл. О причине этого будет сказано в п. 2.2.1.

Другой тип ссылок представляют собой символические ссылки. По назначению они аналогичны ярлыкам в ОС Windows: указывают на файл, расположенный где угодно (например, на съемном носителе), и после удаления такого файла или размонтирования съемного носителя становятся бесполезны.

Символическая ссылка создается той же командой **ln** с ключом **-s**:

```
$ln -s /home/den/README /home/den/do.not.readme
$ls -l /home/den/do.not.readme
lrwxrwxrwx 1 den users 16 Feb 14 19:17 /home/den/do.not.
readme -> /home/den/README
```

В поле имени файла после стрелки указано его настоящее имя. Права доступа у всех символических ссылок одинаковы и не значат ничего: возможность доступа к файлу определяется правами исходного файла. Заметьте, что в отличие от файла-оригинала файл-ссылка имеет ненулевую длину: в нем хранится абсолютное имя исходного файла. Попытайтесь вывести файл-ссылку на экран с помощью команды `cat`, и вы увидите содержание исходного, пустого, файла:

```
$ cat /home/den/do.not.readme
$
```

Значение самой ссылки, то есть имя файла, на который она ссылается, можно узнать с помощью команды **readlink**.

Символические ссылки на каталог создаются и выглядят точно так же, как символические ссылки на обычный файл. Можно создать и цепочку ссылок на ссылки: ядро ОС проследует по всей цепочке и в итоге подставит вместо ссылки имя исходного файла.

Можно даже закольцевать такую цепочку:

```
$ touch a1
$ ln -s a1 a2; rm a1
$ ln -s a2 a1
$ ls -l a? # это шаблон имени. Ему соответствуют все имена
из двух букв, первая «a»
```

```
a1 -> a2
a2 -> a1
$ cat a1
cat: a1: Too many levels of symbolic links
```

Практический смысл этого упражнения — убедить вас в том, что Linux корректно справляется с разрешением ссылок даже в намеренно некорректной ситуации.

Символическая ссылка на каталог может участвовать в образовании полного имени файла, но есть одна тонкость: по ссылке нельзя проследовать обратно в направлении корня дерева каталогов. Вместо псевдоподкаталога «..» подставляется родительский каталог каталога-оригинала. Так, если в домашнем каталоге пользователя ivan есть ссылка на домашний каталог пользователя den, то путь /home/ivan/link_to_den_home/.. эквивалентен не /home/ivan, а /home/den/.., то есть /home.

Именованные каналы

Этот тип файла еще называется буфером FIFO (First In — First Out). Через файлы такого типа два независимых процесса (две работающих программы) могут обмениваться данными: все, что записано в файл одним процессом, может быть прочитано оттуда другим. Именованный канал создается командой **mkfifo**.

Гнезда

Механизм гнезд (сокетов, sockets) впервые появился в версии 4.3 BSD UNIX (ветвь UNIX, начавшая развиваться в калифорнийском университете Беркли). Позже он превратился в одну из самых популярных систем сетевого обмена сообщениями, реализованную во многих, не только UNIX-подобных, операционных системах. В честь создателей этот механизм до сих пор называют Berkeley Sockets.

Собственно гнездо — это абстрактная конечная точка сетевого соединения. Процесс отправляет данные в сеть, записывая их в файл гнезда. При этом процессы, установившие связь через пару гнезд, могут быть запущены как на разных компьютерах, так и на одном.

Межпроцессный обмен через гнезда используется такими стандартными компонентами Linux, как служба учета syslog и оконная система X Window.

2.1.4. Команды работы с файлами и каталогами

Предварительно замечу, что командой в ОС Linux называется все, что может быть выполнено: исполняемый файл, встроенная команда текущей программы и даже формируемая на ходу последовательность символов. Синтаксис обычной команды:

```
имя_команды [короткие_ключи] [длинные_ключи] [аргументы],
```

где в квадратные скобки взяты необязательные данные. Ключи можно указывать в любом порядке, разделяя их пробелами. Аргументы тоже отделяются друг от друга пробелами.

Имена коротких ключей, или опций, состоят из одной буквы, перед которой стоит символ «-» (дефис). У ключа может быть свой аргумент. Короткие ключи, у которых нет аргументов, можно соединять под одним дефисом: так, команда `ls -l -a -d` эквивалентна команде `ls -lad`.

Имена длинных ключей — это осмысленные слова, перед которыми стоят два символа «-» (дефис). Большинство команд понимают ключи **--help** и **--usage**, требующие вывести краткую справку об использовании команды. Необязательный ключ «--» сигнализирует об окончании списка ключей и начале аргументов.

Справку о ключах и аргументах команды можно получить по команде **man**.

Текущий каталог

Текущий каталог — это каталог, от которого отсчитываются относительные пути. В каждый момент времени с каждой работающей программой, в том числе с командной оболочкой, связан единственный такой каталог. Узнать, какой каталог сейчас является текущим, можно с помощью команды `pwd` без аргументов.

Сменить текущий каталог можно командой

```
cd [новый_каталог]
```

Для смены текущего каталога на домашний каталог пользователя можно вместо имени нового каталога указать специальный символ ~: `cd ~`. Разрешается переходить также в псевдоподкаталоги «.» и «..». В первом случае ничего не изменится, а во втором текущим каталогом станет родительский.

Просмотр содержимого каталога

Уже знакомая команда `ls [имя_каталога]` выводит его содержимое на экран. Если не указывать имя каталога, команда выведет содержание текущего каталога.

Вместо имени каталога можно указать шаблон имен файлов: например, `ls my*` покажет все файлы и подкаталоги, имена которых начинаются с «my».

Подробнее об аргументах и ключах команды **ls** (как, впрочем, и любой другой команды) можно узнать у справочной системы, набрав команду `man ls`.

Создание и удаление файла

Создать пустой файл можно командой

```
touch <имя_файла>
```

Вообще-то она предназначена для того, чтобы для всех заинтересованных программ (например, утилиты сборки проекта **make**) файл выглядел новее, чем на самом деле: она меняет время последнего изменения файла на текущее время. Но если файла с таким именем не существует, то она его создаст.

Текстовые файлы можно создавать, вводя текст с консоли:

```
$cat > hello.world
Привет!
```

Оказывается, команду **cat** можно заставить не только выводить файл на консоль, но и вводить с нее. Это достигается перенаправлением ввода-вывода, о котором подробнее будет сказано в главе 8. В таком режиме команда **cat** считает своими входными данными поток байтов, поступающий с клавиатуры, и выводит его в указанный файл. Иными словами, в файл записывается все, что вы после этой команды введете с клавиатуры. Чтобы закончить ввод, нажмите Ctrl+D.

Если файл с указанным именем существует, то команда `cat` перепишет его. Чтобы вместо этого добавить данные в конец файла, перенаправьте ее вывод с помощью символов `>>`:

```
$cat >> hello.world
```

Каталог создается командой

```
mkdir <имя_каталога>
```

Чтобы операция создания файла или подкаталога прошла успешно, вы должны иметь право записи в каталог, в котором вы его создаете.

Удалить пустой каталог можно командой

```
rmdir <имя_каталога>
```

Удаляется файл командой

```
rm <имя_файла>
```

При этом наличие прав на запись и даже чтение этого файла не требуется: достаточно иметь право на запись в каталог, где находится этот файл. Ключи команды **rm**:

- **-i** : требует подтверждения удаления для каждого удаляемого файла. Если вы заказали удаление группы файлов (например, по шаблону имени `rm chernovik*`), то среди них может оказаться файл, который вам еще нужен: безопаснее применить ключ -i. Подтвердите удаление каждого файла или откажитесь от него, введя символ Y или N соответственно;
- **-f** : не запрашивать подтверждения, не выводить сообщений об ошибках. Если указаны оба ключа **–i** и **–f**, то срабатывает последний указанный;
- **-r** : рекурсивное удаление каталога со всеми его подкаталогами. Непустой каталог можно удалить только так.

Копирование и перемещение файла

Файл копируется командой **cp**. Формат этой команды:

```
cp [ключи] <исходный_файл> {<файл_назначения> | <каталог_
назначения>}
```

Полезные ключи команды **cp**:

- **-i** : требовать подтверждения перед перезаписью существующего файла;
- **-f** : не требовать подтверждения;
- **-r** : рекурсивно копировать каталог со всеми подкаталогами;
- **-a** : сохранять атрибуты файла;
- **-d** : копировать символические ссылки вместо файлов, на которые они указывают;
- **-s** : создавать символические ссылки вместо копирования (-l — жесткие);
- **-u** : не переписывать существующий файл, если он модифицирован позже;
- **-x** : игнорировать подкаталоги, расположенные в других файловых системах.

Кстати, команда ***cat*** с перенаправленным выводом тоже может скопировать файл:

```
$cat hello.world > copy.of.hello.world
```

Команда **mv** перемещает или переименовывает файлы. Формат команды:

```
mv [ключи] <исходный_файл> {<файл_назначения> |
<каталог_назначения>}
```

Ключи -i, -f имеют тот же смысл, что для команд cp и rm.

Просмотр текстовых файлов

Перед тем, как просматривать файл, неплохо было бы убедиться, что он действительно является текстовым, то есть содержит только печатные ASCII-символы. Для проверки типа файла служит команда

```
file <имя_файла>
```

Если ее ответ содержит слово «text», то файл можно безопасно вывести на терминал. Вывод двоичного файла может сбить кодировку так, что вместо набираемых на клавиатуре символов вы увидите черт знает что. Если это все же случилось, введите вслепую команду

```
$ tput reset
```

(или, если вы работаете в графическом режиме в окне виртуального терминала, выберите в его меню команду **Терминал | Сброс и очистка** для среды GNOME или **Edit | Reset & Clear Terminal** для среды KDE).

Кроме уже упомянутой команды **cat**, для вывода файла на терминал служат команда

```
more <имя_файла>
```

и ее улучшенный вариант **less**.

Команда-фильтр **more** разбивает поток своего вывода на порции размером в экран и ожидает ввода пользователя для отображения следующей порции. Чтобы увидеть следующую строку, нажмите <Ввод>; чтобы увидеть следующий экран, нажмите <пробел>; чтобы прервать работу команды, нажмите <Q> или Ctrl+C.

Утилита less позволяет листать выводимый файл не только вперед, но и назад (клавишами PgDn и PgUp), перемещаться к указанному месту в файле, искать по образцу и дает еще много полезных возможностей. В общем, man less.

Если интересующая вас информация находится в конце файла (например, вы хотите просмотреть журнал системных сообщений `messages` в каталоге `/var/log`, чтобы узнать, что именно только что пошло наперекосяк), то вам поможет команда

```
tail [-N] [имя_файла],
```

где N — число выводимых строк файла, считая от последней. Командой

```
head [-N] [имя_файла]
```

можно просмотреть, наоборот, только первые N его строк. Значение N по умолчанию равно 10.

Если вы хотите просмотреть не весь файл, а только те его строки, которые содержат заданный фрагмент текста, используйте команду-фильтр **grep**. Например, я хочу сменить клавишу переключения раскладки клавиатуры в графическом режиме. Я знаю, что строки конфигурационного файла `/etc/X11/xorg.conf`, имеющие отношение к клавиатуре, содержат фрагмент XKB... а может, Xkb или xkb? Неважно:

```
$ grep -in xkb /etc/X11/xorg.conf
[...]
65:Option "XkbLayout" "us,ru"
66:Option "XkbOptions" "grp:shift_toggle,grp_led:scroll"
```

Я указал ключи: -i, требующий игнорировать различия регистра в образце для поиска и файле, и -n, требующий выводить номера строк, в которых встречается образец xkb.

Для всех команд ключи без аргументов можно соединять: запись -in эквивалентна записи –i –n.

Редактирование текстовых файлов

Вышеприведенный пример показывает, что переключением раскладки клавиатуры в графическом режиме управляет комбинация клавиш Shift+Shift. Неудобно: в консольном режиме я привык к Ctrl+Shift. Надо заменить значение shift_toggle на ctrl_shift_toggle. Такие мелкие правки конфигурационных файлов — обычное дело для администратора, поэтому средство их внесения присутствовало в UNIX-системах всегда. Это консольный редактор **vi**, входящий в каждый дистрибутив Linux (в дистрибутив Fedora Core 3 включен его улучшенный вариант **vim**, но команда **vi** тоже доступна: она стала псевдонимом для команды **vim**).

Итак, я делаю на всякий случай резервную копию конфигурационного файла `/etc/X11/xorg.conf` и приступаю к его редактированию:

```
$ cp /etc/X11/xorg.conf /etc/X11/xorg.conf.sav
$ vi /etc/X11/xorg.conf
```

Перемещаюсь к строке 66 командой **66G** (буква G заглавная: редактор **vi** различает регистры). Клавишами управления курсором перемещаюсь к началу слова shift.

Включаю режим вставки командой **i** (строчная буква). Набираю ctrl_. Выключаю режим вставки клавишей Esc. Сохраняю изменения командой **:w**. Выхожу по команде **:q**.

Сложно и на вид бессистемно? Да. Зато есть команда **:help**.

Как полноэкранный редактор, **vi** может находиться в одном из двух режимов. В режиме вставки вводимые символы поступают в редактируемый

файл, в командном режиме они воспринимаются как команды. Перечислю коротко самые употребительные команды редактора **vi**:

РЕЖИМ ВСТАВКИ.

Включение режима вставки:

- i — в текущей позиции курсора;
- I — перед первым непробельным символом в текущей строке;
- w — в новой строке, добавленной после текущей;
- W — в новой строке, добавленной перед текущей.

Выключение режима вставки:

- <Esc>

Команды режима вставки:

- Ctrl+a — повторить предыдущую вставку;
- Ctrl+y — вставить символ, находящийся над курсором (в предыдущей строке);
- Ctrl+e — вставить символ, находящийся под курсором (в следующей строке).

КОМАНДНЫЙ РЕЖИМ.

Удаление (здесь и далее N — это число):

- N x — N символов под курсором и справа от него;
- N X — N символов слева от курсора;
- N dd — N строк;
- D — до конца текущей строки;
- N D — до конца текущей строки и еще N-1 строку.

Копирование и вставка строк:

- N yy — взять в буфер N строк от текущей и ниже;
- p — вставить содержимое буфера после текущей строки;
- P — вставить содержимое буфера перед текущей строкой.

Поиск и переход:

- N G — перейти к строке с номером N;
- $ G — перейти к последней строке файла;
- /<образец> — искать образец вниз от курсора;
- ?<образец> — искать образец вверх от курсора;
- n — повторить поиск в том же направлении;
- N — (буквально «N»): повторить поиск в обратном направлении.

Сохранение и выход:

- :w — сохранить текущий файл;
- :w <имя> — сохранить под новым именем, если файл <имя> еще не существует;

- :w! <имя> — сохранить под новым именем, переписав существующий файл;
- :q — выйти;
- :q! — принудительно выйти без сохранения;
- :wq — сохранить и выйти.

Разное полезное:

- N u — отменить последние N изменений;
- N Ctrl+r — вернуть последние N отмененных изменений;
- U — отменить изменения в последней строке;
- N r <символ> — заменить N следующих символов на <символ>;
- N >> — добавить отступ (Tab) в N следующих строк;
- N << — удалить один отступ (Tab) из N следующих строк;
- :sh — временно выйти в оболочку (вернуться — exit);
- :!<команда> — выполнить команду оболочки.

Поиск файлов

Быстрый поиск имени программы можно выполнить прямо из командной строки: для этого введите первые буквы нужной вам команды и нажмите <Tab>. Если введенные вами буквы однозначно определяют команду или исполняемый файл, то ее имя появится в командной строке. Эта функция называется *автозаполнением командной строки*. Если не появилось ничего, нажмите <Tab> еще раз для вывода списка всех доступных команд, начинающихся со введенных букв. Если таких команд окажется больше сотни, у вас попросят подтверждения того, что вы действительно хотите увидеть их все.

В разных каталогах может оказаться несколько исполняемых файлов с одинаковыми именами. Какой из них будет исполнен? На этот вопрос отвечает команда **which**. Она просматривает каталоги, перечисленные в переменной окружения PATH, в поисках исполняемого файла с указанным именем, и выводит абсолютное имя первого встреченного из них.

Команда **whereis** ищет не только исполняемый файл, но и его справочные страницы и исходный код.

Команда **locate** ищет файл по образцу имени, опираясь на свою базу данных о файловой системе. Ее вариант с повышенной безопасностью **slocate** сохраняет данные о правах доступа к файлам, так что пользователь не увидит тех файлов, на которые у него нет прав. В дистрибутиве Fedora Core 3 команда **locate** представляет собой символическую ссылку на утилиту **slocate**.

Команда **find** ищет файл по его атрибутам в указанном каталоге и его подкаталогах на заданную глубину. Например, при установке операционной системы я отказался устанавливать файловый менеджер Midnight

Commander, а теперь он мне понадобился. Для каждого из 4 компакт-дисков дистрибутива запускаю команду поиска по шаблону имени «mc*», то есть всех файлов, имена которых начинаются с mc:

```
$find /media -name mc*
/media/cdrecorder/Fedora/RPMS/mc-4.6.1-0.8.i386.rpm
```

В итоге на третьем диске найден пакет RPM. Об установке программного обеспечения из пакетов RPM будет сказано в главе 7.

Изменение прав доступа к файлу

Как многопользовательская операционная система, ОС Linux содержит механизм разграничения доступа к данным, позволяющий как защитить данные одного пользователя от нежелательного вмешательства других, так и разрешить другим доступ к этим данным для совместной работы.

Как уже сказано, любой ресурс компьютера под управлением ОС Linux представляется как файл, поэтому мы будем говорить только о правах доступа к файлу.

По отношению к файлу пользователь может входить в одну из трех категорий: владелец, член группы владельца, все остальные. Для каждой из этих категорий есть свой набор прав доступа.

Первым владельцем файла становится его создатель. Дальше файл можно передать другому владельцу или в другую группу командой

```
chown [ключи] <новый_пользователь>[:новая_группа] <файл>
```

или

```
chgrp [ключи] < новая_группа > <файл>
```

В некоторых реализациях Linux передать файл другому владельцу имеет право только суперпользователь, а в других — также его текущий владелец.

Набор прав доступа состоит из прав на чтение, запись и исполнение файла. В символьном представлении он выглядит как строка «rwx», где вместо любого символа может стоять дефис. Буква означает наличие права (r — чтение, w — запись, x — исполнение), дефис — его отсутствие.

Очевидно, что эти три бита могут быть записаны еще и как восьмеричное число. Так, права доступа r-x (чтение и исполнение без записи) понимаются как три двоичные цифры 101 или как восьмеричная цифра 5. Численное представление прав доступа называется абсолютным, или двоичной маской.

Полная строка прав доступа в символьном представлении устроена так:

```
<права_владельца><права_группы><права_остальных>
```

В абсолютном представлении права владельца являются старшим разрядом восьмеричного числа, права группы — вторым и права остальных — третьим. Так, права rwxr-x--x выглядят как число 111 101 001, или 751.

Команда изменения прав доступа **chmod** понимает как абсолютное, так и символьное указание прав.

Назначим файлу `/home/den/README` права rw-r-----, то есть разрешим себе чтение и запись, группе только чтение, остальным пользователям — ничего:

```
$cd ~ # переход в свой домашний каталог
$chmod 640 README # 110 100 000 == 640
$ls -l README
-rw-r----- 1 den  users  0 Feb 14 19:08 /home/den/README
```

В символьном представлении можно явно указывать, кому какое право мы хотим добавить, отнять или присвоить. Добавим право на исполнение файла README группе и всем остальным:

```
$chmod go+x README
$ls -l README
-rw-r-x--x 1 den  users  0 Feb 14 19:08 /home/den/README
```

Формат символьного режима:

```
chmod <категория><действие><набор_прав> < файл >
```

Возможные значения аргументов команды представлены в таблице 2.3.

Аргументы команды chmod в символьном режиме — *Таблица 2.3*

Аргумент		Значение
Категория	u	Владелец
	g	Группа владельца
	o	Прочие
	a	Все пользователи, то есть «a» эквивалентно «ugo»
Действие	+	Добавить набор прав
	-	Отменить набор прав
	=	Назначить набор прав
Право	r	Право на чтение
	w	Право на запись
	x	Право на исполнение
	s	Право смены идентификатора пользователя или группы
	t	Бит прилипчивости (sticky-бит)
	u	Такие же права, как у владельца
	g	Такие же права, как у группы
	o	Такие же права, как у прочих

Название бита прилипчивости унаследовано от тех времен, когда объем оперативной памяти был маленьким, а процесс подкачки медленным. Этот бит позволял оставлять небольшие часто используемые программы в памяти для ускорения их запуска. Сейчас его значение переосмыслено: этот бит, установленный для каталога, приводит к тому, что удалять файлы из этого каталога могут только владелец файла и владелец каталога. Обычно это используется в каталогах, открытых для записи всем (например, `/tmp`).

Права смены пользователя и группы (SUID-бит и SGID-бит) означают следующее. Обычно исполняемый файл (программа или командный сценарий) получает те же права на доступ к файлам, что и пользователь, который запустил его на выполнение. Но у этого файла есть еще и владелец, полномочия которого могут быть совсем другими. Наличие одного из этих битов позволяет выполняющейся программе пользоваться полномочиями владельца программного файла или члена его группы.

Так, команда **su** (*substitute user*), позволяющая «стать» другим пользователем без завершения своего сеанса и входа под новым именем (это нужно, например, чтобы быстро выполнить административную задачу от имени суперпользователя), имеет следующие атрибуты:

```
$ls -l `which su`
-rwsr-xr-x 1 root root [размер, дата, время] /bin/su
```

Биты «x» сообщают, что любой пользователь может запустить эту программу, а бит «s» — о том, что во время ее выполнения он будет пользоваться правами суперпользователя root (если, конечно, знает пароль).

Обратите внимание на применение обратных апострофов: они нужны для того, чтобы направить вывод команды **which** на вход команды **ls**.

Следует учитывать, что программы, требующие установления SUID (или SGID) для своей работы, являются потенциальными дырами в системе безопасности. Представьте такую ситуацию: у вас в системе установлена программа **superformat**, которая предназначена для форматирования дисков. Создание файловой системы, пусть даже на дискете, — это привилегированная операция, требующая полномочий суперпользователя.

При установке этой программы для нее сразу устанавливается право SUID, чтобы разрешить пользователям форматировать дискеты. Пользователь запускает ее для форматирования дискеты. Программа запускается, получает права root, форматирует дискету и нормально завершает работу.

А если она завершает работу аварийно, например, по ошибке переполнения стека (такие случаи отмечались)? Тогда запустивший ее пользователь получит права root! Неквалифицированный пользователь с правами root —

это намного хуже, чем просто крах системы. Помните о потенциальной опасности при работе с такими программами и по возможности избегайте использования прав SUID и SGID.

Справедливости ради нужно заметить, что ряд системных программ (в частности, демон установления интернет-соединения **pppd**) разрабатывался с учетом прав SUID и SGID, и эти программы являются максимально защищенными, хотя полной уверенности в этом нет. Поэтому использовать право SUID нужно только в самых крайних случаях.

Я позволю себе сделать еще несколько замечаний относительно прав доступа SUID и SGID:

1. Лучше не использовать программы, требующие привилегий, на сервере, точнее, не разрешать обыкновенным пользователям их использовать. Использование права доступа SUID вы можете себе позволить только на своей домашней машине, например, для установления того же коммутируемого соединения, чтобы каждый раз при подключении к Интернету не вводить команду su.
2. Перед использованием программ, требующих привилегий root, убедитесь в их надежности. Если программа получена из ненадежного источника, лучше ее не использовать. Надежными источниками считаются сайты или FTP-серверы разработчиков дистрибутивов Linux. Желательно получить исходный код такой программы, чтобы убедиться, что она не производит каких-либо несанкционированных действий.
3. Нет ни одной причины, по какой нужно было бы разрешить использование SUID-программ в домашних каталогах пользователей. Для разделов, в которые разрешена запись обыкновенным пользователям, установите опцию nosuid в файле `/etc/fstab`.

2.2. Изнанка файловой системы

С точки зрения операционной системы, под файловой системой понимается внутренняя управляющая структура, заведующая хранением данных на физическом носителе, их поиском, извлечением и записью по запросам программ. Такие управляющие структуры в каждом семействе операционных систем строятся по схожим принципам. Так, DOS/Windows используют файловую систему FAT с вариантами FAT32 и VFAT. Файловые системы UNIX-подобных ОС разнообразнее, но тоже могут быть объединены в одно семейство. Linux умеет работать со множеством файловых систем, как с родными, и с еще большим их количеством обмениваться данными.

Примечание

Хотя существуют средства установки Linux в раздел FAT/FAT32 — **Lin4Win**, я не рекомендую их использовать, т.к. в этом случае Linux работает крайне нестабильно и медленно.

Типичным представителем файловых систем UNIX является «вторая расширенная файловая система» ext2fs, основная до недавнего времени файловая система Linux. С момента выхода ядра версии 2.4.16 она начала уступать место «файловой системы по умолчанию» полностью совместимой с ней системе ext3fs. Рекомендуется использовать именно ext3fs, и именно она устанавливается по умолчанию инсталляторами большинства современных дистрибутивов.

2.2.1. Файловая система ext2fs — предшественница ext3fs

Рассмотрим логическую структуру файловой системы ext2fs.

Физически жесткий диск разбит на сектора размером 512 байт. Первый сектор дискового раздела в любой файловой системе считается загрузочной областью. В первичном разделе эта область содержит загрузочную запись — фрагмент кода, который инициирует процесс загрузки операционной системы при запуске. На других разделах эта область не используется. Остальные сектора объединены в логические блоки размером 1, 2 или 4 килобайта. Логический блок есть наименьшая адресуемая порция данных: данные каждого файла занимают целое число блоков. Блоки, в свою очередь, объединяются в группы блоков. Группы блоков и блоки внутри группы нумеруются последовательно, начиная с 1.

Раздел диска, на котором сформирована файловая система ext2fs, может быть представлен такой схемой:

Структуры данных, применяемые при работе с файловой системой ext2fs, описаны в заголовочном файле `/usr/include/linux/ext2fs.h`.

Суперблок служит начальной точкой файловой системы и хранит всю информацию о ней. Он имеет размер 1024 байта и располагается по смещению 1024 байта от начала файловой системы. В каждой группе блоков он дублируется, что позволяет быстро восстановить его после сбоев.

В суперблоке определяется размер файловой системы, максимальное число файлов в разделе, объем свободного пространства и содержится информация о том, где искать незанятые участки. При запуске ОС суперблок считывается в память и все изменения файловой системы вначале находят отображение в копии суперблока, находящейся в ОП, и записы-

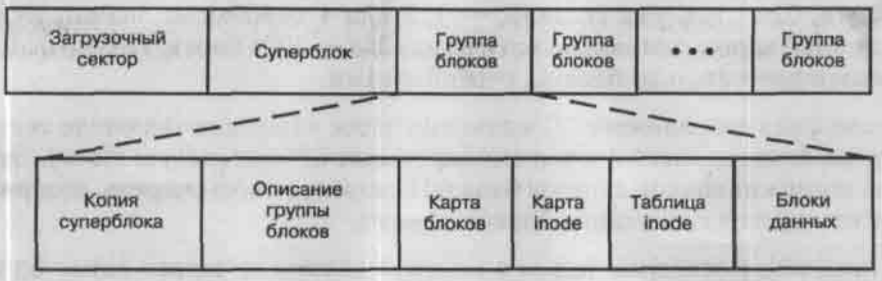


Рис. 2.1. *Структура файловой системы*

ваются на диск только периодически. Это позволяет повысить производительность системы, так как многие пользователи и процессы постоянно обновляют файлы. С другой стороны, при останове системы суперблок обязательно должен быть записан на диск, что не позволяет выключать компьютер простым выключением питания. В противном случае, при следующей загрузке информация, записанная в суперблоке, окажется не соответствующей реальному состоянию файловой системы.

После суперблока следует описание (дескриптор) группы блоков. Хранящаяся в нем информация позволяет найти битовые карты блоков и индексных дескрипторов, а также таблицу индексных дескрипторов.

Битовой картой блоков (block bitmap) называется структура, каждый бит которой показывает, отведен ли такой же по счету блок какому-либо файлу. Значение 1 показывает, что блок занят. Эта карта служит для поиска свободных блоков в тех случаях, когда надо выделить место под файл.

Битовая карта индексных дескрипторов выполняет аналогичную функцию по отношению к таблице индексных дескрипторов: показывает, какие именно дескрипторы заняты.

Каждому файлу соответствует один и только один индексный дескриптор (*inode*, i-узел, информационный узел), который идентифицируется своим порядковым номером — индексом файла. В индексном дескрипторе хранятся метаданные файла. Среди них — все атрибуты файла, кроме его имени, и указатель на данные файла.

Для обычного файла или каталога этот указатель представляет собой массив из 15 адресов блоков. Первые 12 адресов в этом массиве являются прямыми ссылками на номера блоков, в которых хранятся данные файла. Если данные не помещаются в 12 блоков, то включается механизм косвенной адресации. Следующий адрес в этом массиве является косвенной ссылкой, то есть адресом блока, в котором хранится список адресов следующих блоков с данными из этого файла.

Сколько блоков с данными можно так адресовать? Адрес блока занимает 4 байта, блок, как уже сказано, — 1, 2 или 4 килобайта. Значит, путем косвенной адресации можно разместить 256 — 1024 блока. Размер файла, занимающего столько блоков, считайте сами.

А если файл еще длиннее? Следующий адрес в массиве-указателе указывает на блок двойной косвенной адресации (*double indirect block*). Этот блок содержит список адресов блоков, которые, в свою очередь, содержат списки адресов следующих блоков данных.

И, наконец, последний адрес в массиве-указателе задает адрес блока тройной косвенной адресации, то есть блока со списком адресов блоков, которые являются блоками двойной косвенной адресации.

Пока остается непонятным, где находится имя файла, если его нет ни среди данных файла, ни среди его метаданных. В UNIX-подобных системах имя файла есть атрибут не самого файла, а файловой системы, понимаемой как логическая структура каталогов. Имя файла хранится только в каталоге, к которому файл приписан, и больше нигде. Из этого вытекают любопытные следствия.

Во-первых, одному индексному дескриптору может соответствовать любое количество имен, приписанных к разным каталогам, и все они являются настоящими. Количество имен (жестких ссылок) учитывается в индексном дескрипторе. Именно это количество вы можете увидеть по команде ls –l.

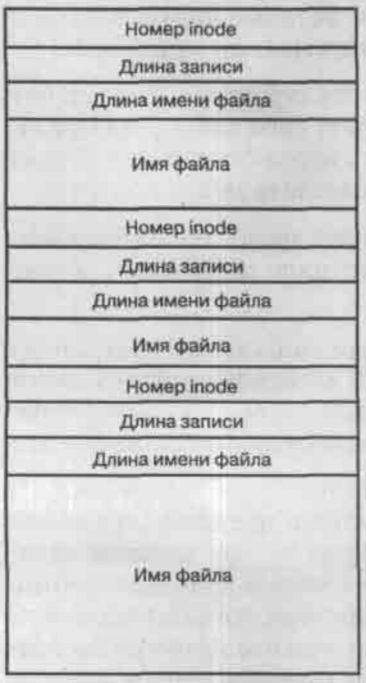


Рис. 2.2. *Строение каталога в ext2fs*

Во-вторых, удаление файла означает просто удаление записи о нем из данных каталога и уменьшение на 1 счетчика ссылок.

В-третьих, сопоставить имя можно только номеру индексного дескриптора внутри одной и той же файловой системы, именно поэтому нельзя создать жесткую ссылку в другую файловую систему (символическую — можно, у нее другой механизм хранения).

Сам каталог таким же образом приписан к своему родительскому каталогу. Корневой каталог всегда записан в индексный дескриптор с номером 2 (номер 1 отведен для списка адресов дефектных блоков). В каждом каталоге хранится ссылка на него самого и на его родительский каталог — это и есть псевдоподкаталоги «.» и «..».

Таким образом, количество ссылок на каталог равно количеству его подкаталогов плюс два.

Данные каталога представляют собой связный список с записями переменной длины и выглядят примерно так, как на рис. 2.2.

А как же файлы физических устройств? Они могут находиться в тех же каталогах, что и обычные файлы: в каталоге нет никаких данных, говорящих о принадлежности имени файлу на диске или устройству. Разница находится на уровне индексного дескриптора. Если i-узел обычного файла указывает на дисковые блоки, где хранятся его данные, то в i-узле файла устройства содержится указатель на список драйверов устройств в ядре — тот элемент списка, который соответствует старшему номеру устройства (рис. 2.3).

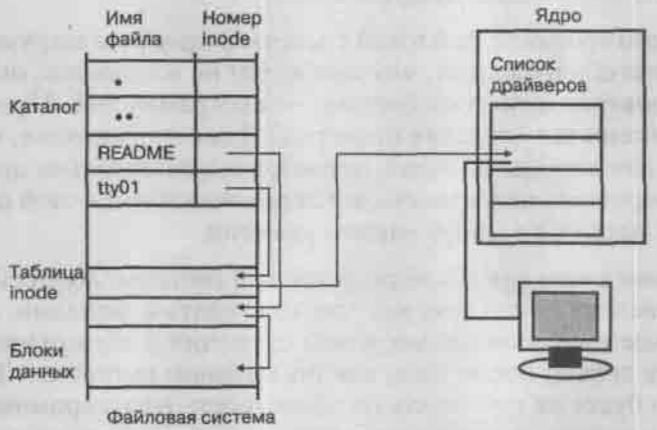


Рис. 2.3. *Разница между обычным файлом и файлом устройства*

Свойства файловой системы ext2fs:

- Максимальный размер файловой системы — 4 Тбайт.
- Максимальный размер файла — 2 Гбайт.
- Максимальная длина имени файла — 255 символов.
- Минимальный размер блока — 1024 байт.
- Количество выделяемых индексных дескрипторов — 1 на 4096 байт раздела.

2.2.2. Журналируемые файловые системы

Представим такую ситуацию. У вас есть жесткий диск, скажем, на 80 Гб. Сегодня таким объемом никого не удивишь, не так ли? Вы поленились разбить его на разделы, и у вас есть один большой раздел, занимающий

все ваши 80 Гб. И вот в момент записи на диск произошло отключение питания. Хорошо, если это случилось во время записи данных какого-то файла, пусть и очень важного: файл можно восстановить хотя бы частично. А вот если свет погас, когда операционная система записывала метаданные, то расположение файла на диске перестанет соответствовать списку принадлежащих ему блоков в индексном дескрипторе. Файловая система может утратить целостность, то есть такое состояние, когда каждый блок принадлежит не более чем одному файлу (*inode*). В результате вы можете не досчитаться не одного, а сотни файлов.

Признаком потери целостности служит бит чистого размонтирования (*clean bit*), точнее, его отсутствие. Этот бит сбрасывается при подключении (монтировании) файловой системы в знак того, что файловая система сейчас используется. После успешного размонтирования файловой системы этот бит устанавливается снова.

Если при монтировании файловой системы в процессе загрузки операционная система обнаруживает, что чистый бит не установлен, она запускает средство проверки файловой системы — программу **fsck**. Представляете, сколько времени займет такая проверка? Даже при условии, что ошибок будет мало или вообще не будет, придется ждать довольно долго. А если еще будет нарушена целостность, тогда восстановление этой целостности займет еще несколько минут вашего времени.

Все это справедливо для обычной файловой системы. Журналируемая же файловая система перед тем, как что-то сделать с файлами, записывает на диск некое описание планируемой операции и вычеркивает каждый пункт плана только после того, как он успешно выполнен. Тогда после сбоя можно будет не проверять на целостность весь огромный раздел, а только просмотреть журнал и откатить незаконченные операции.

Имейте в виду, что целью журналирования является обеспечение целостности файловой системы, а не сохранность пользовательских данных как таковых.

Журналировать операции записи самих данных тоже можно: в этом случае есть вероятность, что данные после сбоя будут восстановлены. Правда, согласно золотому правилу механики, за все нужно платить, и платить приходится быстродействием.

Решают вопрос разными ухищрениями: например, запись происходит в момент наименьшей активности, некоторые журналируемые файловые системы позволяют разместить журнал на другом физическом диске. Да и фактически время работы с журналом намного меньше, чем работа непосредственно с данными. И, естественно, некоторый полезный объем теперь приходится отводить под сам журнал, но его размеры обычно не превышают 32 Мбайт, что по нынешним временам не так уж и много.

И все же лучшим средством от неожиданного отключения до сих пор являются источники бесперебойного питания...

Современные версии ядра Linux (2.6.x) поддерживают в качестве родных четыре журналируемые файловые системы: ReiserFS, ext3fs, XFS и JFS. Из них журналирование данных поддерживает только ext3fs. Список файловых систем, которые поддерживаются вашим ядром, содержится в файле `/proc/filesystems`.

ReiserFS

Разработана Хансом Райзером (Hans Reiser) и его компанией Namesys (`http://www.namesys.com`) и официально включена в ядро 2.4.4. Преимущества данной ФС в основном проявляются в работе с мелкими файлами: они целиком хранятся в своих i-узлах (*inode*), без выделения блоков в области данных. Вместе с экономией места это способствует и росту производительности, так как данные и метаданные хранятся в непосредственной близости и могут быть считаны одной операцией ввода/вывода.

Другая особенность ReiserFS состоит в том, что хвосты файлов длиной меньше чем в один блок могут быть упакованы в один дисковый блок (режим тайлинга). Это обеспечивает около 5% экономии дискового пространства. Именно работа с маленькими файлами (меньше килобайта) и обслуживание большого их количества выделяет данную ФС среди прочих.

ReiserFS несовместима с ext2fs на уровне утилит обслуживания файловой системы, однако соответствующий инструментарий, объединенный в пакет reiserfsprogs, уже давно включается в стандартную поставку современных дистрибутивов. Если у вас его нет, то скачать можно по адресу `ftp://ftp.namesys.com/pub/reiserfsprogs/reiserfsprogs-3.6.19.tar.gz`.

Там же можно взять патчи для ядра 2.4.x.

К сожалению, загрузчики Linux (LILO и GRUB) не способны загрузить ядро Linux с раздела ReiserFS, оптимизированного в режиме тайлинга. Поэтому под каталог /boot лучше отводить отдельный раздел с файловой системой, совместимой с ext2fs.

XFS

При работе с огромными (терабайтными) файлами вне конкуренции остается файловая система XFS, разработанная компанией Silicon Graphics (сейчас SGI) специально для операций с мультимедийными данными и впервые появившаяся в 1994 г. в версии ОС Irix 5.3. Она использует

64-битную адресацию, что позволяет увеличить максимальный размер файловой системы до 18 тысяч петабайт (при этом предельный размер файла составляет 9 тысяч петабайт).

Особенностью этой файловой системы является устройство журнала: в журнал пишется часть метаданных самой файловой системы таким образом, что весь процесс восстановления после сбоя сводится к копированию этих данных из журнала в файловую систему. Размер журнала задается при создании системы, он должен быть не меньше 32 мегабайт.

XFS эффективно распараллеливает операции ввода-вывода: она делит все пространство раздела на несколько равных областей (*allocation group*), служащих своего рода автономными файловыми системами в рамках единой XFS.

Пакет утилит обслуживания xfsprogs можно скачать с `http://oss.sgi.com/projects/xfs/download.html` (содержит ссылку на российское зеркало).

JFS

Разработана IBM для рабочих станций под управлением ОС AIX, затем портирована для Linux и выпущена по Стандартной Общественной лицензии. Всю необходимую информацию о ней можно найти по адресу `http://jfs.sourceforge.net`.

Размер журнала составляет примерно 40% от размера файловой системы. Эта файловая система может содержать несколько сегментов, содержащих журнал и данные. Такие сегменты называются агрегатами и могут монтироваться отдельно. Умеет она также хранить маленькие файлы и каталоги, содержащие не больше 8 файлов, в пределах индексного дескриптора.

Широкого признания пока не получила.

Ext3fs

Файловая система ext3fs официально включена в ядро Linux с версии 2.4.16. Впервые она появилась в дистрибутивах RedHat и SuSE. Современные дистрибутивы, основанные на ядре 2.6.x, предлагают установить ext3fs по умолчанию.

Некоторые источники утверждают, что ext3fs — это всего лишь «надстройка» над файловой системой ext2fs, а не самостоятельная файловая система. Благодаря такому происхождению ext3fs совместима со всеми программами для обслуживания и настройки файловой системы ext2fs. И перейти на ext3fs можно простым добавлением файла журнала к ext2fs, не только без переформатирования раздела, но даже и без перезагрузки

машины. Более того, ОС Linux на старых ядрах, не поддерживающих ext3fs, могут работать с разделами, на которых сформирована эта файловая система, просто подключая их как разделы ext2fs.

Кроме того, ext3fs — самая надежная из рассмотренных в этом параграфе новых файловых систем: в ней предусмотрено журналирование операций не только с метаданными, но и с данными файлов.

Журнал может быть включен в одном из следующих режимов:

- полного журналирования (journal);
- последовательном (ordered, применяется по умолчанию);
- обратной записи (writeback).

Режим полного журналирования позволяет минимизировать ваши потери при отключении питания, но является наиболее медленным из всех трех режимов. Этот режим и подразумевает журналирование записи пользовательских данных.

Самый быстрый режим — это «обратная запись». Это обыкновенное журналирование только метаданных.

Режим «последовательный» представляет собой компромисс между скоростью и полнотой. Официально журналируются только метаданные, но блоки соответствующих им данных записываются первыми. В большинстве случаев такой режим гарантирует сохранность данных, особенно если данные дописывались в конец файла, как обычно и бывает.

Какой режим выбрать? Если ваш сервер является файловым (FTP, WWW-сервер), то есть таким, который используется пользователями для хранения файлов, выберите режим полного журналирования — пользователи будут вам благодарны. Пусть в этом режиме сервер будет работать чуть медленнее, зато в случае ЧП можно минимизировать потери информации. Во всех остальных случаях нужно установить режим «Последовательный», точнее вообще не нужно ничего устанавливать — он используется по умолчанию.

2.3. Создание и монтирование файловых систем

Создание файловой системы «вручную». Команда mkfs

Из предыдущего параграфа следует, что создать файловую систему на разделе жесткого диска или внешнем носителе — это значит разметить его сектора на структуры данных, специфические для этой файловой системы (суперблок, список i-узлов, блоки данных). В DOS/Windows этот

процесс называется форматированием. В UNIX-системах понятие форматирования не используется, а процедура и соответствующая команда так и называются — создание файловой системы.

В ходе установки Linux файловые системы на разделах жесткого диска создал для вас инсталлятор, и переделывать за ним ничего не нужно. Ручное создание ФС может понадобиться при подключении нового диска или, что гораздо чаще, дискеты. Выполняет его команда **mkfs**:

```
mkfs [-t <тип>] [опции_ФС] ФС [блоков]
```

Типы и описание файловых систем, поддерживаемых Linux, вы найдете в справочной системе по команде `man fs`. Те типы, которые чаще всего используются на съемных носителях, перечислены в таблице 4.3. Ясно, что если вы «отформатируете» дискету с типом **vfat**, то она будет читаться и из-под Windows, а если с типом ext2/ext3 — только из-под Linux.

Если не указывать тип, то будет создана ФС с типом по умолчанию — в настоящий момент это ext2.

В качестве аргумента ФС можно указывать либо имя устройства (`/dev/fd0`), либо точку монтирования (`/media/usbdisk`). Последним аргументом можно указать количество блоков, которые нужно отвести под новую файловую систему.

Опции_ФС — это параметры, специфические для файловой системы определенного типа. Например, для ext2/ext3 можно указать:

- `-b <размер_блока>` (по умолчанию 4096 байт, но может быть и 1024 или 2048);
- `-N <количество_i-узлов>`,
- `-i <байт_на_i-узел>`.

Утилита **mkfs** передает эти опции настоящему конструктору ФС, которого вызывает в зависимости от указанного типа. Установленные у вас конструкторы можно найти по команде `ls /sbin/mkfs*`, а список опций конкретной ФС посмотреть на man-странице соответствующего конструктора (например, `man mkfs.vfat`).

Замечу только, что для ext3fs после ключа **-J** можно указать опции журналирования: **size=<размер>** — размер журнального файла в мегабайтах, и device=<устройство> — внешний журнал, заранее созданный на другом разделе.

Как я уже сказал, преобразовать существующий раздел ext2fs в ext3fs можно без перезагрузки системы, простым добавлением журнала командой **tune2fs**:

```
# tune2fs -j /dev/hd5
```

Дополнительные конструкторы (для ReiserFS, XFS и т.п.) устанавливаются из пакетов **reiserfsprogs** и **xfsprogs** соответственно.

В результате на новом разделе образуется корневой каталог новой файловой системы и в нем — подкаталог `/lost+found`. Не удаляйте этот подкаталог: утилита **fsck**, предназначенная для проверки целостности ФС, помещает в него найденные куски нарушенных файлов.

Настройка автоматического монтирования при загрузке компьютера. Команда mount

Чтобы с новой файловой системой можно было работать, она должна быть при помощи операции монтирования включена в общее дерево каталогов (п.1.1.3). О ручном монтировании/размонтировании применительно к съемным носителям сказано в п.4.2.5, а здесь я скажу, как сделать так, чтобы разделы жесткого диска автоматически монтировались при загрузке системы и демонтировались при останове. Для этого их нужно прописать в файл `/etc/fstab`, который читает команда **mount** в ходе начальной загрузки (об этапах загрузки и загрузочных сценариях см. п.9.1).

Каждая строка этого файла соответствует одной файловой системе и состоит из шести полей, разделенных пробельными символами:

```
<устройство> <точка_монтирования> <тип> <опции> <дамп>
<номер_fsck>
```

- **Устройство** — это файл устройства, к которому подключен раздел (например, `/dev/hda5`). Для сетевой файловой системы здесь должно быть указано имя сервера и каталог на нем.
- **Точка_монтирования** — это имя каталога, к которому файловая система будет подключена. Он должен существовать и (желательно) быть пустым. Для раздела подкачки (swap) значение этого поля не используется, но в файле `/etc/fstab` присутствовать все равно должно.
- Вместо **типа ФС** можно указать значение auto: в этом случае команда **mount** попытается определить тип самостоятельно.
- **Дамп** — это отметка о необходимости резервного копирования данной ФС программой **dump** (п.10.4.3). Значение 1 говорит о том, что резервировать нужно, значение 0 — нет.
- **Номер_fsck**: утилита **fsck** обычно запускается перед автоматическим монтированием ФС, проверяет ее на целостность и пытается исправить найденные ошибки. Это процедура долгая, и для ускорения загрузки можно либо отключить проверку для некоторых ФС (значение 0), либо для некоторых разделов запускать ее параллельно. Значение этого поля задает порядок проверки разных ФС: если номера одинаковые, то системы будут проверяться параллельно. Ясно, что ускорение может получиться только в том случае, когда параллельно проверяемые разделы находятся на разных физических дисках.

Справку об опциях монтирования команды **mount** можно получить по команде `man mount`. В таблице 2.4 перечислены самые употребительные из них.

Основные опции монтирования *Таблица 2.4*

Опция	Назначение
auto	ФС может быть смонтирована автоматически
ro	Смонтировать файловую систему в режиме «только чтение»
rw	Смонтировать файловую систему для чтения и для записи (по умолчанию)
dev	ФС может содержать файлы блочных и символьных устройств. Они интерпретируются как специальные файлы
exec	Файлы на ФС могут быть исполняемыми
suid	Разрешить использование битов SUID и SGID
user	Разрешить непривилегированному пользователю монтировать и размонтировать данную файловую систему. Это значение влечет за собой noexec, nosuid и nodev, если после него явно не указано exec, suid или dev
noauto nodev noexec nosuid nouser	Значения, противоположные соответствующим без «но»
defaults	Установки по умолчанию: rw,suid,dev,exec,auto,nouser,async
codepage= <значение>	Интерпретировать символы в именах файлов согласно кодовой странице
iocharset= <значение>	Выводить символы в именах файлов согласно набору символов

Программа установки создала файл `/etc/fstab`, в котором перечислены все ваши Linux-разделы (корневой, swap и, если вы послушались п.1.3.4, `/home`). Теперь нужно сделать так, чтобы из-под Linux были видны данные на ваших Windows-разделах. Проверьте, поддерживает ли ваше ядро типы ФС на Windows-разделах (`cat /proc/filesystems`, в выводе команды должны присутствовать слова `vfat` и/или `ntfs`). Обычно ядро, поставляемое с дистрибутивом, собрано без поддержки NTFS — если ваш Windows-раздел отформатирован с этим типом, то вам придется либо пересобрать ядро, либо, что гораздо проще, конвертировать раздел в тип FAT32.

Итак, после всех этих проверок я вписал в файл `/etc/fstab` строку:

```
/dev/hda6 /mnt/disk_e vfat rw,codepage=866,iocharset=utf8
```

...и получил каталог `/mnt/disk_e`, всем файлам в котором приписан в качестве владельца root, а в качестве прав доступа — rwx-rxr-x. Если вас такой режим доступа не устраивает и хочется иметь право писать в этот каталог от имени непривилегированного пользователя, прочитайте на man-странице команды **mount** об опциях **uid**, **gid** и **umask**.

Глава 3

РАБОТАЕМ В КОМАНДНОЙ СТРОКЕ

- КАК УСТРОЕН LINUX: ЯДРО И ПРОЦЕССЫ
- ЖИЗНЬ ПРОЦЕССА
- ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ
- КОМАНДНАЯ ОБОЛОЧКА. BASH

3.1. Как устроен Linux: ядро и процессы

Главная, постоянно находящаяся в оперативной памяти, часть ОС Linux называется ядром (Kernel). Ядро ОС обрабатывает прерывания от устройств, выполняет запросы системных процессов и пользовательских приложений, распределяет виртуальную память, создает и уничтожает процессы, обеспечивает многозадачность посредством переключения между ними, содержит драйверы устройств, обслуживает файловую систему (см. рис. 3.1).

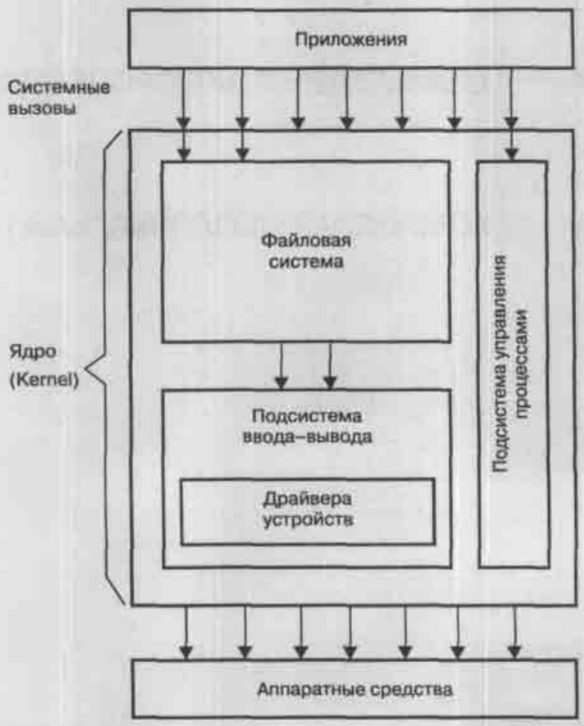


Рис. 3.1. *Устройство ОС Linux*

Пользовательские процессы не могут непосредственно, например, порождать другие процессы, производить чтение или запись на диск, выводить данные на экран или создавать гнездо (*socket*) для обмена по сети. Для выполнения этих действий они должны воспользоваться сервисами ядра. Обращения за такими услугами называются системными вызовами.

Начальная загрузка системы состоит в том, что файл с образом ядра считывается в оперативную память, начиная с нулевого адреса. Этот файл находится в каталоге `/boot` и называется vmlinuz-x.y.z, где x.y.z — это номер версии ядра. На текущий момент большинство дистрибутивов основано на ядре версии 2.4, хотя уже вышло ядро 2.6 (Fedora Core 3) и кое-где еще встречается версия 2.2.

Примечание

По соглашению разработчиков ядра, все ветви с четным номером (2.2, 2.4 и т.д.) считаются стабильными и рекомендуются для широкого использования, а на ветвях с нечетным номером испытываются новые идеи. Линус Торвальдс предложил распространить эту схему нумерации и на третью цифру версии: в ядра с нечетными номерами добавлять новые функции, а в четных — только исправлять обнаруженные ошибки.

В UNIX-подобных системах в отличие от других ОС ядро минимизировано и не выполняет ни одной функции, служащей непосредственно пользователю. Для этой цели применяются многочисленные утилиты, выступающие в качестве посредников между пользователем и ядром. Только в комплекте с ними ядро образует полноценную операционную систему.

Этих компонент ОС Линус Торвальдс не создавал: они поступили из проекта GNU (`http://www.gnu.org`), участники которого с 1984 года работают над созданием полноценной UNIX-подобной ОС, целиком состоящей из свободно распространяемого программного обеспечения. К 1991 году им не хватало только ядра, и эту-то прореху и заполнил Торвальдс. Так что ОС, которой посвящена эта книга, правильнее называть не Linux, а «операционной системой GNU, основанной на ядре Linux», или просто GNU/Linux.

Итак, ядро обслуживает запросы процессов. Что же такое процесс? Это понятие является базовым в UNIX-подобных системах. Процесс можно представить себе как виртуальную машину, отданную в распоряжение одной задачи. Каждый процесс считает, что он на машине один и может распоряжаться всеми ее ресурсами. На самом же деле процессы надежно изолированы друг от друга, так что крушение одного не может повредить всей системе (сколько раз вы наблюдали в Windows, как сбой одной программы приводил к общему зависанию?).

Каждый процесс выполняется в собственной виртуальной памяти (см. рис. 3.2), в которую никакой другой процесс вмешаться не может. Этим и обеспечивается устойчивость всей системы.

Рис. 3.2. *Виртуальная память процесса*

Напоминаю, что такое виртуальная память. Каждому процессу разрешено считать, что его адреса начинаются с нулевого адреса и от него наращиваются. Таким образом, в 32-разрядной ОС процесс может адресовать 4 гигабайта оперативной памяти. Механизм виртуальной памяти позволяет процессу думать, что именно столько ему и выделено, хотя физически объем ОЗУ вашей машины — какие-то жалкие 256 Мбайт. Недостающую память заменяет жесткий диск путем записи временно не используемых страниц памяти в раздел подкачки (свопинга).

Разделяемость библиотек между процессами обеспечивается тем, что их код и статические данные отображаются на один и тот же участок физической оперативной памяти.

3.2. Жизнь процесса

Таблица процессов

С точки зрения ядра процесс представляет собой запись в таблице процессов. Эта запись содержит данные, существующие в течение всего времени жизни процесса, и сведения о его состоянии. Размер таблицы процессов позволяет запускать несколько сотен процессов одновремен-

но. Другая важная информация о процессе — например, таблица всех открытых процессом файлов — хранится в его адресном пространстве. Запись в таблице процессов и пространство процесса вместе составляют контекст, или окружение, процесса. В него входят:

- **PID** — идентификатор процесса. Он принудительно назначается планировщиком при запуске процесса.
- **PPID** — идентификатор родительского процесса (о порождении процессов — дальше в этом же параграфе).
- **TTY** — имя управляющего терминала (терминал, с которого запущен процесс).
- **WD** — текущий каталог процесса, от которого отсчитываются относительные пути.
- **RID, RGID** — реальные ID и групповой ID пользователя, запустившего процесс.
- **EUID, EGID** — эффективные ID и GID: см. п.2.1.4.8.
- **NICE** — показатель уступчивости. Процессы выполняются в режиме разделения времени, то есть время центрального процессора делится между готовыми к выполнению процессами с учетом их приоритета. Чем выше показатель уступчивости, тем ниже приоритет.
- **Переменные окружения**.

Системные вызовы fork() и exec() или как размножаются процессы

Каждый процесс порождается другим процессом, использующим для этого системный вызов *fork*(). Таким образом, структура процессов, подобно файловой системе, древовидна. Корнем этого дерева служит ***init*** — процесс инициализации системы. Он запускается ядром первым, получает идентификатор 1 и порождает еще несколько процессов (сколько и каких, можно узнать из его конфигурационного файла `/etc/inittab`), которые, в свою очередь, при участии пользователя порождают другие процессы.

В результате системного вызова *fork*() родительский процесс полностью копирует свое окружение, включая адресное пространство, в дочерний, так что в момент рождения дочерний процесс отличается только своим ID. Потом дочерний процесс с помощью вызова *exec*() загружает в свое адресное пространство какой-нибудь исполняемый файл и начинает исполнять содержащуюся в нем программу.

Может случиться и так, что процесс выполняет вызов *exec()* без *fork()*: тогда не возникает нового процесса, но в старом начинает выполняться другая программа. Например, программа **login** выполняется с привилегиями суперпользователя, поскольку ей нужен доступ к файлу паролей. Проверив пароль, она устанавливает себе права зарегистрировавшегося пользователя и выполняет вызов *exec*(), замещая свой код кодом ко-

Рис. 3.3. *Как размножаются процессы*

мандной оболочки. После этого из командной оболочки изменить свои привилегии обратно на root нельзя, потому что кода программы **login** в текущем процессе уже нет.

Каждый процесс, завершившись, возвращает родительскому процессу какое-то значение, называемое кодом завершения или кодом возврата. По соглашению разработчиков, нулевой код возврата означает успешное завершение, а ненулевые — разнообразные ошибки. Процесс-родитель может приостановить свое выполнение до завершения потомка и выполнить разные действия в зависимости от возвращенного дочерним процессом значения, а может и не делать этого.

Снимок протекающих в системе процессов — команда ps

Моментальный снимок протекающих в системе процессов можно посмотреть с помощью команды **ps** (*process status*). Без аргументов она покажет список процессов, связанных с текущей консолью (или виртуальным терминалом). Список возможных ключей команды можно, как обычно, получить по команде `ps --help`. Вот некоторые полезные из них:

- **-p < список_PID>** : только процессы с указанными ID;
- **-u < список_USERID >** : только запущенные указанными пользователями;
- **-e** : все процессы в системе;
- **-f** : полная форма вывода;
- **-H** : вывод иерархии процессов в форме дерева.

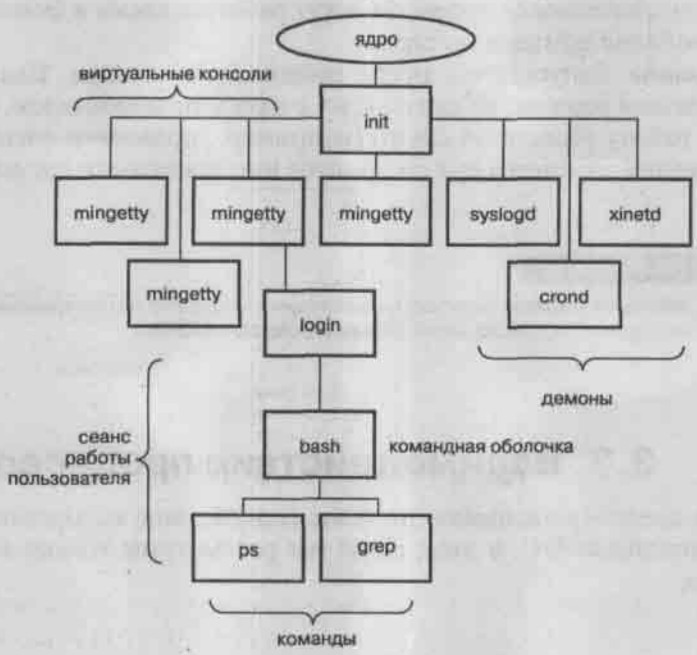


Рис. 3.4. *Фрагмент иерархии процессов*

Динамика процессов — команда top

Представление о динамике процессов дает команда **top**. Она выводит список процессов, отсортированный по количеству занятой памяти или использованного процессорного времени, и обновляет его через указанные промежутки времени (по умолчанию через каждые 3 секунды).

Категории процессов

Процессы делятся на три категории:

- **Системные**. Они порождаются ядром особым образом в процессе загрузки и выполняют системные функции (например, планирование процессов или смену страниц виртуальной памяти). Выполняемая ими программа берется не из исполняемого файла, а является частью ядра.
- **Пользовательские**. Как правило, они порождаются во время сеанса работы пользователя и связаны с терминалом. Если пользовательский процесс работает в интерактивном режиме, то он захватывает терминал в монопольное владение и, пока он не завершится, пользователь не имеет доступа к командной строке на этом терминале.

Пользовательские процессы могут работать также в фоновом режиме, освободив командную строку.

- **Демоны**. Запускаются после инициализации ядра. Выполняются в фоновом режиме, не связаны ни с одним пользователем, обеспечивают работу различных служб (например, управление сетью). Главным демоном считается ***init*** — процесс инициализации системы.

Примечание

Название «демон» (*daemon*) не имеет ничего общего с потусторонними существами: это просто сокращение от Disk And Execution MONitor.

3.3. Взаимодействие процессов

Из всех средств межпроцессного взаимодействия, которыми так богаты UNIX-подобные ОС, в этой главе мы рассмотрим только конвейеры и сигналы.

3.3.1. Конвейер (pipe)

В главе 2 вы познакомились с командой-фильтром ***more***, вызываемой так:

```
<команда_выводящая_много_строк> | more
```

Символ «|» — это и есть конвейер. Его можно понимать как канал, в который один процесс может только писать, а другой — только читать из него. Выборка и помещение информации в такой канал происходит в порядке FIFO (*First In/First Out*).

Посредством конвейера вывод одной команды подается на вход другой. Это одно из мощнейших средств UNIX, позволяющее комбинировать из простых команд длинные и изощренные цепочки обработки данных.

Например, я хочу узнать, осталась ли у меня еще свободная виртуальная консоль, чтобы зарегистрироваться там и спросить справку по какой-то команде, не прерывая процессов, протекающих на других консолях. Я знаю, что виртуальную консоль обслуживает программа **mingetty**, которая после регистрации на этой консоли замещает свой код на код командной оболочки. Значит, мне нужно подсчитать количество процессов **mingetty**. Есть команда **wc** (*word count*), умеющая подсчитывать число строк, слов или байтов в файле. Есть команда **grep**, умеющая выбирать из файла строки, содержащие указанный фрагмент текста. Соединяю их конвейером:

```
$ ps -e | grep mingetty | wc -l
```

3.3.2. Сигналы

Механизм сигналов — это средство, позволяющее сообщать процессам о некоторых событиях в системе, а процессу-получателю — должным образом на эти сообщения реагировать. Послать сигнал может сам процесс (например, при попытке деления на ноль), ядро (при сбое оборудования), пользователь или другой процесс (требуя прервать выполнение задачи).

Всего в Linux 63 сигнала, обозначаемых своими номерами или символическими именами. Имена всех сигналов начинаются с SIG, и эту приставку часто опускают: так, сигнал, требующий прекратить выполнение процесса, называется SIGKILL, или KILL, или сигнал 9.

Получив сигнал, процесс может: игнорировать его; вызвать для обработки установленную по умолчанию функцию; вызвать собственный обработчик (перехватить сигнал). Некоторые сигналы (например, KILL) перехватить или игнорировать невозможно.

Пользователь может послать сигнал процессу с идентификатором PID командой

```
$ kill [-s <сигнал>] <PID>
```

где <сигнал> — это номер или символическое имя.

Несколько часто встречающихся сигналов перечислены в таблице 3.1. Полный список можно получить по команде `kill -l` (*list*).

Сигналы Linux *Таблица 3.1*

№	Имя	Назначение	Реакция процесса-получателя
1	HUP	Hangup — отбой	Демоны перечитывают свои конфигурационные файлы
2	INT	Interrupt	Прекратить выполнение (перехватывается)
3	QUIT	Сильнее, чем INT	то же
4	ILL	Illegal instruction. Программная ошибка	Обработать ошибку. По умолчанию — прекратить выполнение
8	FPE	Floating point exception. Вычислительная ошибка (деление на ноль)	Обработать ошибку. По умолчанию — прекратить выполнение
9	KILL	Убить процесс	Немедленно прекратить выполнение. Не перехватывается
11	SEGV	Segmentation violation. Попытка доступа к чужой области памяти	Обработать ошибку. По умолчанию — прекратить выполнение
13	PIPE	Нет процесса, читающего из конвейера	Обработать ошибку
15	TERM	Termination. Завершить процесс	Корректно завершить выполнение. Перехватывается
17	CHLD	Завершился дочерний процесс	Принять возвращенное им значение

Некоторые сигналы посылаются по нажатии комбинации клавиш. Так, Ctrl+C посылает сигнал INT, а Ctrl+\ (обратный слэш) — сигнал QUIT. Получает эти сигналы тот процесс, который сейчас занимает консоль — например, ожидает вашего ввода.

Команда **kill** носит такое убийственное название потому, что чаще всего используется для принудительного завершения процессов, вышедших из-под контроля, забирающих много ресурсов или просто повисших. По умолчанию она посылает сигнал TERM. Он отличается от сигнала KILL тем, что приказывает процессу завершиться аккуратно, закрыв открытые им файлы, удалив временные и т.п. Сигнал же KILL действует на процесс как выстрел в голову.

Понятно, что для того, чтобы прервать выполнение процесса, нужно быть его хозяином или иметь привилегии суперпользователя.

3.4. Командная оболочка. Bash

Важнейшим из пользовательских процессов является командная оболочка (она же командный интерпретатор, или просто shell). Именно она обеспечивает взаимодействие пользователя с системой в текстовом режиме, позволяя вводить команды. Именно она запускается, когда вы регистрируетесь на текстовой консоли, и предоставляет вам интерфейс командной строки.

Не нужно, увлекшись удобствами графического интерфейса, недооценивать командную строку. Во-первых, многие административные задачи могут быть выполнены только оттуда; во-вторых, командная строка — самое удобное средство автоматизации рутинных процедур.

Командой в Linux считается все, что может быть исполнено: исполняемые файлы, встроенные команды оболочки, псевдонимы команд, пользовательские функции, файлы сценариев (скрипты) — заранее подготовленные последовательности команд в текстовом виде. До сих пор, приводя примеры команд, я не различал их по происхождению, и дальше не собираюсь делать этого, кроме особых случаев.

Оболочка принимает вводимые пользователем команды, обрабатывает, если нужно, их аргументы, отправляет команды на выполнение, принимает возвращаемые ими значения и выполняет определенные действия в зависимости от этих значений. Кроме того, в оболочку встроен язык программирования (командный язык), позволяющий писать сложные разветвленные командные сценарии. Именно командный язык отличает разные оболочки друг от друга, и именно из него исходят пользователи, выбирая любимую и нелюбимую оболочки.

Для Linux разработано много командных интерпретаторов. Вот несколько из них:

sh ..... *Bourne shell*, оболочка Борна, стандарт для многих UNIX-подобных систем;
bash ... *Bourne Again shell*, «еще одна оболочка Борна»;
csh..... *C shell*, оболочка Си: синтаксис ее командного языка похож на синтаксис языка С;
tcsh .... *tiny C shell*, минимальная оболочка Си;
pdksh .. *public domain Korn shell*, общедоступная оболочка Корна;
sash.... *stand-alone shell*, автономная оболочка, может быть использована в случае, когда программные библиотеки недоступны.

Список всех установленных в системе программ-оболочек находится в файле `/etc/shells`. У меня он выглядит так:

```
/bin/sh
/bin/bash
/sbin/nologin   # это "оболочка" для тех,
                # кому запрещен вход в систему
/bin/ash
/bin/bsh
/bin/ksh
/usr/bin/ksh
/usr/bin/pdksh
/bin/tcsh
/bin/csh
```

Начальная оболочка для каждого пользователя, запускаемая для него при регистрации в системе, указывается в файле `/etc/passwd`:

```
$ grep den /etc/passwd          # выбрать из файла строки,
                                # содержащие подстроку den
den:x:501:501:Denis:/home/den:/bin/bash
```

В дальнейшем вы можете сменить текущую оболочку на любую из установленных (точнее, войти в подоболочку). Чтобы выйти из нее и вернуться в родительскую оболочку, введите команду `exit`. В начальной оболочке эта команда завершает сеанс работы.

В любой оболочке можно запускать командные сценарии, состоящие из команд другой оболочки: первая строка каждого сценария содержит указание на то, в какой оболочке его следует выполнять, и текущая оболочка запускает для него указанную как дочерний процесс.

По умолчанию новому пользователю назначается оболочка **bash**. Это прекрасная оболочка, включающая много усовершенствований и лучших свойств других оболочек, и менять ее я не рекомендую. В дальнейшем, говоря «оболочка», я буду иметь в виду именно **bash**.

3.4.1. Встроенные команды

Список встроенных команд оболочки **bash** можно получить по команде **help** или найти на man-странице в секции SHELL BUILTIN COMMANDS. Напоминаю, что поиск в выводе команды **man** выполняется командой /<образец><Ввод>, а поиск следующего вхождения образца — по нажатии клавиши <n>.

Справку по команде, имя которой вы знаете, можно получить командой help <имя>.

Вот несколько полезных встроенных команд:

- alias <псевдоним> <длинная команда с аргументами> — назначение псевдонима. Без аргументов выводит список всех имеющихся псевдонимов. Обратите внимание, что у пользователя root команда **rm** сделана псевдонимом для «rm -i», чтобы он не забыл воспользоваться ключом –i (см.п.2.1.4.3). Вы тоже можете назначить псевдоним для опасной команды **risk**, создав сценарий, который сначала будет спрашивать «а вы уверены?», и только при положительном ответе запускать ***risk*** на выполнение. Дайте этому сценарию имя **risk**, а внутри него ссылайтесь на настоящую команду **risk** по ее полному пути. Удалить псевдоним из списка можно командой unalias.
- echo [аргументы] — вывод аргументов на экран. Полезно, если нужно выполнить подстановку (п.3.4.4) и посмотреть, что получится.
- enable < имя_команды > — заставляет оболочку вместо встроенной команды выполнить исполняемый файл с таким же именем. Полезно, если у вас есть собственный сценарий по имени, например, echo.
- eval [аргументы] — конструирование команды на лету, из указанных аргументов, и отправка ее на выполнение.
- let <переменная>=<арифметическое выражение> — вычисление выражений. Так, команда var=1+2 присвоит переменной var (см.п.3.4.3) значение «1+2», а команда let var=1+2 — значение «3».
- source < файл > — прочитать и выполнить команды, содержащиеся в файле. Применяется для определения пользовательских переменных и функций (п.3.4.3).

Другие встроенные команды служат инструкциями командного языка **bash**.

3.4.2. История команд

Оболочка предлагает вам много возможностей для облегчения ввода команд и редактирования командной строки. Помимо функции автозаполнения, с которой вы познакомились в п.1.1.4.7, **bash** содержит механизм командной истории. Суть его в том, что вводимые вами команды (по

умолчанию 1000) запоминаются и доступны для повторного вызова — без изменений или с ними.

Команда `history` без аргументов выводит всю историю, нумеруя при этом команды в порядке их ввода.

Если вас интересуют только последние несколько команд:

```
$ history 23              # показать последние 23 команды
```

Если вас интересуют все команды, имеющие отношение к монтированию каталога `public`:

```
$ history | grep mount | grep public
                          # еще один пример конвейера
```

Номера команд выводятся для того, чтобы вы могли снова ввести эту команду, набрав

```
$ ! <номер>
```

или

```
$ !! # запускает последнюю из введенных команд
```

или

```
$ !<первые_буквы>          # запускает последнюю из команд,
                           # начинающихся с этих букв.
```

Стрелки «вверх» и «вниз» перемещают по командной истории, не отправляя команду на выполнение, а вводя ее в командную строку для редактирования.

Последнюю команду может для вас отредактировать сама оболочка. Для этого вместо команды введите:

```
$ ^что_заменить^чем_заменить
```

Например, вы запросили справку по команде оболочке **bash**: `man bash`. Если сразу после этого вы хотите посмотреть справку по оболочке **csh**, можете вместо `man csh` набрать

```
$ ^ba^c.
```

Помните, что замене подлежит первое вхождение подстроки «что_заменить».

Если вы хотите не изменить, а дополнить последнюю команду (например, пропустить ее вывод через фильтр **more**), введите

```
$ !! | more.
```

3.4.3. Переменные

Описание и использование переменных

Как любой язык программирования, командный язык **bash** поддерживает переменные. Тип их — строковый. Оператор присваивания выглядит так:

```
$ <имя_переменной>=<значение>
```

Имя должно начинаться с буквы и может состоять из латинских букв, цифр, знака подчеркивания. Если значение переменной содержит специальные символы, их нужно экранировать кавычками или обратным слэшем (см. п. 2.1.1).

Операция подстановки значения переменной обозначается символом $ (не путайте с приглашением **bash**). Вывести значение переменной можно командой **echo**:

```
$ cwd=/home/den/MyDownloads/packages
$ echo cwd # выводит имя переменной
cwd
$ echo $cwd # выводит значение переменной
/home/den/MyDownloads/packages
```

Установленные таким образом переменные доступны только встроенным командам **bash**. Чтобы они стали доступны дочерним процессам (программам и командным сценариям, запускаемым из-под **bash**), их нужно поместить в окружение **bash**. Делается это командой **export**:

```
$ export HELLO="Hello from environment!" # пробел нужно
экранировать
```

Чтобы почувствовать разницу, создайте простейший командный сценарий, выводящий значения двух переменных. Для записи сценария можно создать пустой файл и открыть его в каком-нибудь ASCII-редакторе, а можно вспомнить п.2.1.4 и воспользоваться командой `cat`:

```
$ cat > myscript
echo Env variable: $HELLO
echo Local variable: $hello  # помните о разнице в регистре?
                             # Это другая переменная.
^D
$
```

Комбинация клавиш Ctrl+D завершает ввод и закрывает файл, и вы снова видите приглашение оболочки.

Получившийся файл сценария нужно сделать исполняемым (п.2.1.4):

```
$ chmod a+x myscript
```

Теперь осталось определить переменную hello и запустить сценарий:

```
$ hello="Hello from local"
$ echo $hello
Hello from local
$ ./myscript
Env variable: Hello from environment!
Local variable:
$
```

Переменные окружения

Когда оболочка начинает работу, она устанавливает для себя несколько переменных окружения. Имена их стандартны. Программы и сценарии могут запросить их значения вместо того, чтобы пытаться выяснить нужную им информацию самостоятельно.

Несколько таких переменных перечислены в таблице 3.2.

Переменные окружения **bash** *Таблица 3.2*

Имя	Назначение
BASH_VERSION	Версия оболочки
USER	Имя, под которым вы зарегистрировались
UID, EUID	Реальный и эффективный userID
HOME	Путь к вашему домашнему каталогу
HOSTNAME	Имя вашего компьютера
HOSTTYPE	Тип процессора (i386 или другой)
OSTYPE	Операционная система (linux-gnu)
HISTFILE, HISTSIZE	Расположение и размер файла истории команд
LANG	Язык текущего сеанса
LINES, COLUMNS	Число строк и столбцов на экране текстовой консоли
PS1, PS2, PS3, PS4	Переменные, определяющие вид приглашения оболочки
PATH	Порядок просмотра каталогов в поисках исполняемого файла с заданным вами именем, когда полный путь к нему не указан

Чтобы просмотреть значения всех переменных текущего сеанса, как определенных вами, так и переменных окружения, введите команду **set**.

Обратите внимание на переменную PATH: среди каталогов, в которых **bash** ищет исполняемый файл, нет текущего. Поэтому в предыдущем примере, если бы вы попытались исполнить сценарий командой **myscript**, оболочка ответила бы «Command not found». Нужно было указать путь к исполняемому файлу, и мы указали его относительным способом, считая от текущего каталога: `./myscript`.

Переменная PS1 у меня выглядит так: `[\u@\h  \W]\$`. Это значит, что приглашение оболочки у меня формируется из регистрационного имени (*username*), имени машины (*hostname*), текущего каталога (*working dir*) и символа $. Я могу и изменить его:

```
[den@dhsilabs ~]$ pssave=$PS1     # сохраняю старое значение
[den@dhsilabs ~]$ PS1="\W>"
~>                                # новое приглашение состоит
                                  из имени текущего каталога
                                  и символа >
~> cd My*
MyDownloads>
MyDownloads > PS1=$pssave         # поиграли, и хватит
[den@dhsilabs MyDownloads]$
```

Удалить переменную можно командой unset <имя>.

Быстрая смена каталога

Переменная CDPATH задает список каталогов, в которых будет происходить поиск нужного подкаталога при смене каталога (использовании команды **cd**). Проще всего пояснить, как работает CDPATH, на примере. Пусть в моем каталоге `/home/denis/books/linux-server` есть подкаталоги `chapter1` … `chapter20`. Если мне нужно перейти в подкаталог `chapter2`, то я могу сэкономить на наборе имени его родительского каталога, внеся это имя в переменную CDPATH:

```
$ export CDPATH=:.:/home/denis/books/linux-server
```

Теперь по команде `cd  chapter2` я попаду в каталог `/home/denis/books/linux-server/chapter2` из любого места файловой системы, если подкаталога `chapter2` нет в текущем каталоге.

Настройка командной строки. Утилита tput

Наверное, многим хочется, чтобы их компьютер не был похож на компьютер коллеги за соседним столом. Кто-то меняет темы, кто-то — обои. Мы попробуем изменить командную строку текстовой консоли. Рассмотрим сценарий, выводящий текущий каталог в правом верхнем углу — обычно этот угол при выводе текста остается свободным. Для чего? А просто так — чтобы было не как у всех.

Для манипуляции с курсором и цветом букв и фона используется утилита **tput**. В п.2.1.4 вы узнали, как применить ее для восстановления «сбитой» консоли, а сейчас посмотрите на то, что она умеет еще. А потом прочитайте man-страницу.

Листинг 3.1. Демонстрация возможностей утилиты tput

```
#!/bin/bash

function prompt_command {
# сохраняем текущую позицию курсора
tput sc
# вычисляем длину, необходимую для вывода текущего каталога
# текущий каталог можно узнать с помощью команды pwd
let backwash=$(tput cols)-$(echo $(pwd) | wc -m)-2
# позиционируем курсор Y=0, X=длина
tput cup 0 ${backwash}
# установка цвета букв, начертание - жирное
tput setaf 4 ; tput bold
# выводим полный путь в квадратных скобках
echo -n "["
# устанавливаем цвет
tput setaf 6
# отображаем путь
echo -n "$(pwd)"
# устанавливаем цвет для закрывающей скобки"
tput setaf 4 ; tput bold
# отображаем ]
echo -n "]"
# возвращаем курсор в исходную позицию
tput rc
}

PROMPT_COMMAND=prompt_command

GREEN="\[$(tput setaf 2 ; tput bold)\]"
WHITE="\[$(tput setaf 7 ; tput bold)\]"
NO_COLOUR="\[$(tput sgr0)\]"

case $TERM in
    xterm*|rxvt*)
        TITLEBAR='\[\033]0;\u@\h \007\]'
        ;;
    *)
        TITLEBAR=""
        ;;
esac

PS1="${TITLEBAR}\
$GREEN\u@\h \
$WHITE\$$NO_COLOUR "
PS2='> '
PS4='+ '
```

Команды утилиты **tput**:

tput setaf [1-7] .....установка цвета символов с использованием ANSI ESC- последовательности;
tput setab [1-7] .....установка цвета фона;
tput rev .....обратить цвета фона и переднего плана;
tput bold .....установка жирного начертания;
tput dim .....отключение жирного начертания;
tput smul .....установка подчеркнутого начертания;
tput rmul .....отключение подчеркнутого начертания;

3.4.4. Подстановка переменных и команд

Переменные можно использовать как имена, части имен или аргументы команд. Перед выполнением команды оболочка заменит имена переменных их значениями. Например, после того, как мы присвоили переменной cwd значение пути к каталогу, можно перейти в этот каталог по команде

```
$ cd $cwd
```

Вывести на экран файл README из этого каталога можно, введя команду

```
$ more ${cwd}/README
```

Брать имя переменной в скобки необязательно, но удобно, если нужно отделить имя переменной от предшествующих ему или следующих за ним символов.

Мощным инструментом оболочки **bash** является подстановка команд, то есть замена имени команды на результат ее выполнения. Так, считая `/home/den/MyDownloads/packages` текущим каталогом, мы могли бы присвоить переменной cwd то же самое значение проще:

```
$ cwd=`pwd`      # напоминаю, что команда pwd возвращает
                 # путь к текущему каталогу
```

Можно подставлять значения не только определенных вами переменных, но и переменных окружения. Так, чтобы поэкранно вывести список всех процессов, запущенных от вашего имени, введите:

```
$ ps -ef | grep $USER | less
```

3.4.5. Шаблоны имен файлов

Этот механизм позволяет не перечислять похоже выглядящие имена файлов и каталогов, а указать на целую группу имен, задав краткий об-

разец. Перед отправкой команды на выполнение оболочка раскрывает шаблон, то есть заменяет образец всеми именами, подходящими под этот образец, и выполняет команду для каждого файла или каталога из этой группы. Шаблоны указываются с помощью специальных символов, перечисленных в таблице 3.3. Символы шаблона можно комбинировать в одной команде.

Символы шаблонов *Таблица 3.3*

Символ	Значение	Пример
*	Произвольная строка символов, в том числе пустая	~/*.png — все файлы в домашнем каталоге с расширением png; Glava* — файлы Glava, Glava03 и Glava.old
?	Любой одиночный символ	Glava?? — файлы Glava03 и GlavaXZ, но не Glava и не Glava.old
[m,M,x]	Любой символ из перечисленных в скобках	Glava0[3,8] — файлы Glava03 и Glava08; Glava?[3,8] — файлы Glava03, Glava08, Glava13, Glava18, Glava23
[a-nA-N]	Любой символ из указанных интервалов	Glava0[2-4,9] — Glava02, Glava03, Glava04, Glava09
[^a-n,x,y]	Любой символ, не указанный в скобках	Glava[^0]* — все главы, начиная с 11

Символы шаблона можно использовать и как обычные символы в именах файлов. Тогда их нужно экранировать, чтобы оболочка не приступила к их раскрытию:

```
$ touch \*            # создаст файл с именем "*". Только не
                      # удаляйте его потом командой rm * !
$ rm Glava0\[3\,8\]   # удалит файл с именем Glava0[3,8],
                      # а не Glava03 и Glava08.
```

3.4.6. Потоки ввода-вывода

Как я уже сказал, каждому процессу сопоставлена таблица открытых им файлов. Три первых позиции в этой таблице заняты всегда: каждый процесс открывает потоки (помните, что в UNIX файл — это и есть поток данных?) для ввода и вывода данных, а также вывода сообщений об ошибках и другой диагностической информации. Эти потоки:

- 0 — стандартный ввод (**stdin**),
- 1 — стандартный вывод (**stdout**),
- 2 — стандартный поток сообщений об ошибках (**stderr**).

Ссылаться на эти потоки можно по их *файловым дескрипторам*. 0, 1 и 2 — это и есть такие дескрипторы.

По умолчанию потоки ввода-вывода связываются с консолью, с которой запущен процесс: стандартный ввод — с клавиатурой, другие два потока — с экраном (рис. 3.5, потоки `cmd1`).

Все потоки можно перенаправить в другой файл. Это может быть файл на диске, файл устройства (например, принтер или `/dev/null`) или стандартный поток другого процесса.

Для перенаправления стандартного вывода команды используется символ > («больше»). Если местом назначения служит файл, то можно его не перезаписывать, а присоединить (*append*) выводимые данные в его конец. Для такого перенаправления применяется символ >>.

Стандартный ввод перенаправляется символом < («меньше»).

Для перенаправления стандартного потока ошибок используется конструкция `2>`. Чтобы присоединить **stderr** к **stdout** и перенаправить их вместе, пользуйтесь переадресацией `2>&1`.

Для направления стандартного вывода одной команды на стандартный ввод другой применяется символ | — уже знакомый вам конвейер.

Ситуация, изображенная на рис. 3.5, могла бы сложиться после выполнения следующих команд:

```
cmd2 < file1.txt | cmd3 2>&1 > file2.txt
cmd4 2> /dev/null | tee file3.txt file4.txt file5.txt
```

Команда-фильтр **tee** копирует данные со своего стандартного ввода в стандартный вывод, дублируя их при этом в указанные ей файлы.

Рис. 3.5. *Перенаправление потоков ввода-вывода*

3.4.7. Группировка команд

Кроме конвейеров, команды можно соединять в списки. Простейший вид списка — несколько команд, разделенных точкой с запятой:

```
$ lpr myfile.txt ; lpq
```

Команды в таком списке выполняются последовательно: сначала будет выполнена команда постановки задания в очередь печати, а потом проверено состояние принтера. Оболочка ждет завершения каждой команды, чтобы отправить на выполнение следующую (синхронный режим).

В списки можно группировать не только одиночные команды, но и конвейеры:

```
$ ps -ef | head -n 1; ps -ef | grep httpd

UID    PID   PPID  C  STIME TTY TIME     CMD
root   13565 1     0  13:11 ?   00:00:00 /usr/local/sbin/httpd
nobody 13566 13565 0  13:11 ?   00:00:00 /usr/local/sbin/httpd
nobody 13567 13565 0  13:11 ?   00:00:00 /usr/local/sbin/httpd
nobody 13642 13565 0  13:16 ?   00:00:00 /usr/local/sbin/httpd
```

Первый конвейер вырезает из списка процессов заголовок, а второй — информацию о демоне **httpd**.

Если требуется запустить команду на заднем плане и не ждать ее выполнения (асинхронный режим), то нужно завершить ее управляющим оператором **&**:

```
$ du --human-readable --total /home > diskusage.txt &
```

Команда **du** сообщает, сколько места на диске занято файлами в указанном каталоге. Обратите внимание на ключи: их имена предваряются не одним минусом, а двумя. Большинство команд поддерживает как короткие однобуквенные ключи, так и длинные — с осмысленными, легко запоминаемыми именами. Напоминаю, что список и значения ключей команды **cmd** можно получить, выполнив ее с ключом **--help** или **--usage**.

Конвейеры (или одиночные команды) в списке можно соединять управляющими операторами **&&** и **||**, получая «список И» или «список ИЛИ» соответственно:

```
cmd1 && cmd2
cmd3 || cmd4
```

Вторая команда в «списке И» будет выполнена только в случае успешного завершения первой. Типичный пример — создание каталога и переход в него:

```
$ mkdir mydir && cd mydir
```

Вторая команда в «списке ИЛИ» будет выполнена только в случае неуспешного завершения первой (возврата ею ненулевого значения):

```
$ my_command || echo "Команда my_command не найдена" | mail den
```

Вторая команда (конвейер) в этом примере формирует и посылает письмо с отчетом о неуспехе пользователю den.

Чтобы перенаправить в файл вывод всех команд из списка, нужно взять весь список в круглые скобки:

```
$ ( date; free; who; ) > logfile
```

Список, взятый в круглые скобки, выполняется в дочерней оболочке, имеющей собственные локальные переменные и текущий каталог:

```
$ pwd; ( cd /tmp ; pwd ) ; pwd;
/home/den
/tmp
/home/den
$
```

Если нужно часто выполнять одну и ту же последовательность команд, можно оформить ее как функцию:

```
$ function morning_report {
> date;
> free;
> w;
> }
$ morning_report | mail root
```

Имена и область видимости функций подчиняются тем же правилам, что и для переменных. Нельзя определять функцию и переменную с одинаковыми именами.

Определенные вами переменные и функции действительны только для текущего сеанса работы в оболочке **bash**. Чтобы воспользоваться ими в следующем сеансе, запишите их в текстовый файл, а когда они понадобятся, загрузите этот файл в память командного интерпретатора встроенной командой `source`:

```
$ cat > foo
myvar="Моя переменная"
function myfun {
echo $myvar
}
^D
$ source foo
$ myfun
Моя переменная
$
```

Команда `source` выполняет инструкции, содержащиеся в файле, в текущей оболочке в отличие от исполнения файла, содержащего сценарий: тот выполняется в дочерней оболочке, и определенные в ней переменные и функции для родительской оболочки невидимы. Чтобы заметить разницу, удалите переменную `myvar` и функцию `myfun` из памяти оболочки командой `unset`, сделайте файл `foo` исполняемым командой **chmod** (п.3.4.3) и исполните его. Убедитесь, что после его выполнения переменная `myvar` и функция `myfun` остались не определены.

3.4.8. Инициализационные файлы bash

Начальные значения переменных окружения становятся известны командному интерпретатору **bash** из инициализационных файлов, которые он прочитывает сразу после своего запуска. Эти файлы называются `.bash_profile` и `.bashrc` (в порядке чтения оболочкой) и берутся из домашнего каталога запустившего оболочку пользователя.

Команды, присутствующие в этих файлах по умолчанию, только прочитывают общесистемный файл настроек `/etc/bashrc`. Вы можете добавить к ним свои личные настройки, определив нужные вам переменные, функции и псевдонимы.

При завершении сеанса работы с оболочкой выполняются команды в файле `~/.bash_logout`. Туда вы тоже можете добавить свои команды: например, зафиксировать время окончания своего сеанса в файле или в письме другому пользователю.

Команду установки переменной CDPATH, рассмотренной в п.3.4.3, имеет смысл добавить в `.bash_profile`, чтобы не пришлось вводить ее вручную в начале каждого сеанса работы в **bash**.

Глава 4

РАБОЧЕЕ МЕСТО ПОЛЬЗОВАТЕЛЯ

- ГРАФИЧЕСКАЯ СИСТЕМА X WINDOW
- ОКОННАЯ СРЕДА KDE
- ОКОННАЯ СРЕДА GNOME
- ОФИСНЫЕ ПАКЕТЫ
- ИЗДАТЕЛЬСКИЕ СИСТЕМЫ
- ГРАФИКА В LINUX
- ПОЛЕЗНЫЕ ТРЮКИ

Если вы собираетесь только изучать, настраивать и администрировать свою ОС Linux, то вашим рабочим местом станет консоль, а средой обитания — командная строка. Для нормальных же людей компьютер — не хобби, а инструмент для решения определенного круга задач. Задачи эти решаются не средствами операционной системы, а прикладными программами, и большинство людей привыкло решать их с помощью приложений, работающих в графическом режиме под управлением ОС семейства Windows. Среди таких приложений:

- офисный набор: текстовый процессор, редактор электронных таблиц, система управления базами данных;
- средства просмотра и редактирования графической информации;
- средства общения с коллегами (электронная почта, интернет-пейджер);
- средства получения информации из Интернета: веб-браузер, ftp- клиент;
- программы для воспроизведения аудио- и видеозаписей;
- узкопрофессиональные программные пакеты: математические, инженерные, бухгалтерские, разработчика программного обеспечения и т.п.

Для подавляющего большинства таких Windows-приложений существуют достойные (и, что немаловажно, бесплатные) Linux-аналоги, совокупность которых и создает удобную среду рабочего места.

4.1. Графическая система X Window

Работа в графическом режиме под Linux возможна благодаря системе, именуемой X Window (или просто Иксы; только не называйте ее X Windows), разработанной в Массачусетском технологическом институте (MIT) и ставшей стандартом для всех UNIX-подобных систем. Сами по себе Иксы — это не графический интерфейс как таковой, а лишь набор спецификаций, которым этот самый графический интерфейс должен соответствовать. В настоящее время действует версия 11 выпуск 6 стандарта на графическую подсистему для UNIX-систем, которая кратко обозначается как X11R6.

Группа программистов, возглавляемая Дэвидом Вексельблатом (David Wexelblat) создала свободно распространяемую реализацию X Window для процессоров i80386-Pentium IV и совместимых с ними. Эта версия получила название XFree86, поскольку могла выполняться в операционных системах, предназначенных для процессоров, использующих систему команд x86 — Linux, FreeBSD и других. Однако с версии 4.4 команда разработчиков XFree86 перешла на новую лицензию, которую общественность сочла несовместимой со Стандартной Общественной лицензией и поэтому недостойной включения в некоммерческую ОС.

Блюдя чистоту идеи открытого кода, другая команда запустила проект X.Org (`http://www.x.org`), представляющий собой развитие XFree86 от версии 4.3, еще имевшей «правильную» лицензию. В результате одни дистрибутивы (например, Fedora Core) содержат реализацию стандарта X11 от X.org, а другие (например, предыдущие ОС Red Hat) — от XFree86.org. Различия коснулись имен и расположения некоторых файлов.

Имена файлов в разных реализациях X11 — *Таблица 4.1*

	XFree86.org	X.org
Исполняемые файлы X Server	XFree86	Xorg
Файл настроек X Server	/etc/X11/XF86Config	/etc/X11/xorg.conf
Файл журнала X Server	/var/log/XFree86.$DISPLAY.log	/var/log/Xorg.$DISPLAY.log

Сама по себе система X Window не предоставляет никакого пользовательского интерфейса. Она только предоставляет другим программам средства для работы с видеосистемой компьютера, то есть видеокартой и монитором, и устройствами ввода: клавиатурой и мышью.

Организована она по модели клиент-сервер, хотя эти термины понимаются не совсем обычно. Обычно программа-клиент работает у вас дома, а сервер — в Калифорнии. X-сервер же — это аппаратно-зависимая система ввода-вывода, работающая там, где находятся устройства ввода-вывода, то есть на вашем компьютере. А X-клиент, выводящий данные в видеосистему, может быть запущен и на удаленной машине.

Один из X-клиентов — это оконный менеджер (или диспетчер окон). Он управляет размещением окон на экране, определяет их вид и характер управляющих элементов. То есть именно он и представляет собой графический интерфейс пользователя (GUI) в собственном смысле. Таким образом, пользователь ОС Linux, в отличие от пользователя Windows, не привязан к одному графическому интерфейсу: таковых, определяемых оконным менеджером, теоретически может быть невообразимое множество.

От оконных менеджеров отличаются так называемые интегрированные графические среды, или просто оконные среды. Их отличие в том, что наряду с функциями управления окнами они предоставляют доступ к некоторым наборам утилит и приложений, написанных специально для конкретной среды, более или менее тесно в нее встроенных и легко обменивающихся данными. Со многими популярными диспетчерами окон и оконными средами можно познакомиться по адресу `http://xwinman.org`.

Почти в любой дистрибутив включены хотя бы две интегрированные графические среды: KDE и GNOME. Я рекомендую использовать среду KDE — это очень мощная и одновременно простая в освоении и использовании оконная среда.

Не зря ее чаще, чем GNOME, назначают средой по умолчанию. По внешнему виду и функциональности вчерашним пользователям Windows она наиболее симпатична.

Обычно система X Window запускается автоматически при загрузке системы. Вы увидите графический менеджер регистрации (в зависимости от настроек вашей системы он может выглядеть по-разному), приглашающий вас ввести имя пользователя и пароль. Менеджер регистрации позволяет также выбрать сеанс, то есть оконную среду (KDE, GNOME или другую), которую вы будете использовать. Одна из установленных оконных сред запускается по умолчанию.

Из оконной среды вы можете переключиться на текстовую виртуальную консоль, нажав комбинацию клавиш Ctrl+Alt+F<n>, где n — число от 1 до количества запущенных виртуальных консолей (по умолчанию их 6, но несколько можно отключить. Как это сделать, вы узнаете в п.9.1.2). Чтобы вернуться на графическую консоль, нажмите Alt+F7.

4.2. Оконная среда KDE

По своей простоте и интуитивности среда KDE (K Desktop Environment, `http://www.kde.org`) подобна графическим интерфейсам MacOS или Windows. KDE предоставляет богатые возможности взаимодействия программ, обладает встроенным механизмом drag-and-drop, полностью реализует вид и ощущение (look-and-feel) рабочего стола.

4.2.1. Рабочий стол KDE

Рабочий стол KDE состоит из трех частей:

1. самого рабочего стола, на котором могут размещаться значки файлов, каталогов и устройств;
2. управляющей панели, которая используется для запуска программ;
3. панели задач, которая предназначена для переключения между работающими программами.

Привычный пользователь Windows сразу же ищет кнопку «Пуск». Есть в KDE такая кнопка. Она называется кнопкой «K» и находится в начале

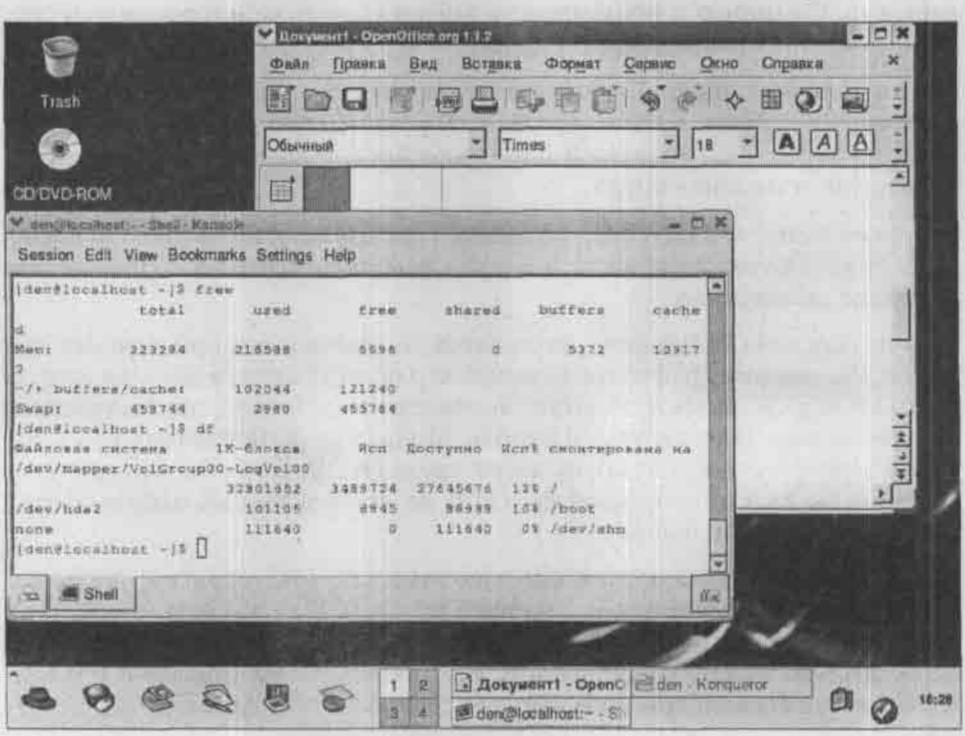


Рис. 4.1. Рабочий стол KDE

управляющей панели (по умолчанию — в левом нижнем углу). Под ней, как и в Windows, скрывается многоуровневое иерархическое меню. В вашем дистрибутиве оно, как и встроенная гипертекстовая справка, может быть даже русифицировано.

Для быстрого вызова меню «K» можно использовать комбинацию клавиш Alt+F1.

4.2.2. Запуск программ и переключение между ними

В общем случае запустить программу или приложение на выполнение можно несколькими способами:

- Щелкнуть мышкой по значку программы на панели (если таковой имеется).
- Щелкнуть мышкой по соответствующему значку рабочего стола (если такой есть).
- Выбрать программу из меню «K».
- Использовать файловый менеджер, например, **Konqueror**.
- Выбрать в меню «K» команду «Выполнить программу» и ввести имя запускаемой программы в строку ввода появившегося диалога. Быстро вызвать этот диалог можно комбинацией клавиш Alt+F2.
- Запустить программу с виртуального терминала.

Виртуальный терминал — это программа (**Kterm** или **Konsole**), в окне которой вы можете работать в режиме командной строки, как на обычной виртуальной консоли.

Примечание

Для знающих английский язык: названия приложений вроде **Konqueror**, **Konsole**, **Kontact** — это не опечатки, а знак того, что приложения разрабатывались для Оконной Среды K. Название «K» не расшифровывается никак. Впрочем, есть версия, что это название было выбрано в пику CDE (Common Desktop Environment), коммерческой оконной среде, разработанной для коммерческой UNIX.

Переключаться между запущенными приложениями можно комбинацией клавиш Alt+Tab. Вызвать контекстное меню любого объекта можно правой кнопкой мыши. В общем, все как в Windows... и еще несколько виртуальных рабочих столов. Они обозначены кнопками с цифрами на панели задач. Окна приложений открываются на текущий рабочий стол, и, когда вы наведете там достаточный беспорядок, вы сможете переключиться на чистый стол щелчком по такой кнопке или комбинацией клавиш Ctrl+Tab. Обратите внимание, что комбинация Alt+Tab переключает только между приложениями текущего стола.

4.2.3. Файловый менеджер Konqueror

Основные возможности

Среда KDE обладает собственными файловым менеджером и веб-браузером. Обе эти функции выполняет программа Konqueror. Менеджер Konqueror может работать не только с локальными файлами, а и с файлами, расположенными на других компьютерах вашей сети или на серверах FTP.

Как и любой другой файловый менеджер, Konqueror умеет:

- Копировать, перемещать, переименовывать и удалять файлы и каталоги;
- Просматривать файлы различных форматов;
- Создавать ссылки на файлы;
- Изменять свойства файла.

Команды меню и инструментальные кнопки представляют собой объединение команд, знакомых вам по Проводнику Windows и по привычному веб-браузеру. Кнопка «Домой» в режиме файлового менеджера выполняет переход в домашний каталог. Часть рабочей области можно отвести под виртуальный терминал (меню **Окно | Эмуляция терминала**).

Можно переключить Konqueror и в режим Midnight Commander. Все режимы на самом деле задаются настроечными файлами в каталоге `~/.kde/share/apps/konqueror/profiles` (помните, что «~» — это ваш домашний каталог).

Программа, пользующаяся этими настройками, называется **kfmclient**.

Konqueror — не только файловый менеджер

При работе с FTP-сервером удобно видеть сразу два каталога — локальный и удаленный. Если вы используете Konqueror для работы с FTP, вы можете разделить окно «Завоевателя» на две части. Для этого нажмите Ctrl + Shift + L. Перед этим весьма желательно развернуть окно на весь экран. Выберите одну из частей (какую вам удобнее) и в строке Адреса (Location) введите адрес вашего FTP-сервера: `ftp://ftp.server.ru`. «Завоеватель» попросит вас ввести имя пользователя и пароль, а после этого вы сможете удобно работать с FTP-сервером.

Если в строке Адреса ввести smb:/, вы увидите рабочие группы и домены Windows, если таковые есть в вашей сети. Для работы этой функции должен быть установлен и запущен пакет Samba (подробнее см. п.6.3).

А теперь попробуйте ввести в строке адреса `rlan:/`. Думаю, вы найдете в своей сети много интересного.

Вставьте Audio CD в ваш привод CD-ROM. Запустите Konqueror и введите `audiocd:/`. Вы увидите список дорожек вашего Audio CD, которые будут представлены в виде отдельных WAV-файлов. В зависимости от установленных плагинов вы можете преобразовать дорожки в форматы MP3 или OGG или же просто сохранить на диске как WAV-файлы.

4.2.4. Центр управления KDE

Центр управления играет ту же роль, что панель управления Windows, с той лишь разницей, что Панель управления настраивает всю систему в целом, а Центр управления — только KDE. Зато в пределах KDE настройке поддается практически все. В разных дистрибутивах и разных версиях KDE может отличаться не только организация и состав разделов Центра управления, но и способ его вызова. Так, в KDE версии 3.2 он называется **Настройка своего рабочего стола** и вызывается через **K | Система**, а в KDE 3.3 он вернулся в меню K. Следующая таблица (табл. 4.2) пригодна для KDE 3.2. Последней вышла версия KDE 3.4. Ее особенности мы рассмотрим в отдельном разделе (см. п.4.2.7).

Разделы Центра управления *Таблица 4.2*

Раздел	Описание
LookNFeel	В этом разделе находятся настройки виртуальных рабочих столов, декорации окон, панелей (в том числе и панели задач), пиктограмм, стиля и оформления окон, фона рабочего стола и хранителя экрана
Безопасность	Параметры шифрования, в том числе поддерживаемые алгоритмы шифрования
Звук	Параметры сервера звука aRts и системных звуков
Информация	Сведения о системе. Этот раздел — своеобразный аналог программы SysInfo из пакета Norton Utilites
Периферия	Настройки периферийных устройств: цифровой камеры, клавиатуры и мыши
Компоненты	Опции различных компонентов и служб KDE
Просмотр Web	Параметры браузеров Konqueror и Netscape
Сеть	Настройки для работы с сетью. В этом разделе можно указать, как с сетью будете работать вы. Настраивать же собственно сеть может только администратор
Система	Общесистемные параметры, например, параметры менеджера регистраций. Изменить их вправе только суперпользователь (администратор)
Специальные возможности	В этом разделе вы сможете определить привязки клавиш, раскладки клавиатуры, выбрать страну и язык
Управление питанием	Загляните в этот раздел, если вы используете ноутбук

В разделе **Специальные возможности → Страна и язык** обязательно установите свою страну (Россия или Украина), выберите язык и установите кодировку (koi8-r или koi8-u). Все эти параметры необходимы для правильного отображения русскоязычных символов в окнах приложений KDE.

4.2.5. Работа со съемными носителями в KDE

Чтобы ОС Linux получила доступ к данным на вашей дискете, компакт-диске или flash-накопителе, файловая система съемного носителя должна быть включена в общее дерево каталогов как одна из его ветвей. Этот процесс называется монтированием. Логически он представляет собой сопоставление корневому каталогу подключаемой файловой системы какого-то из уже существующих каталогов. Этот каталог называется точкой монтирования. После того, как к каталогу примонтирована другая файловая система, все бывшие в нем файлы временно — до размонтирования — становятся недоступны, а вместо них видны файлы съемного носителя. Отсюда следует, что в качестве точки монтирования лучше выбирать пустой каталог.

Оконная среда KDE до версии 3.3 не выполняла монтирование автоматически, и вы должны знать, как сделать это вручную.

Обычно команда `mount` применяется в формате:

```
mount -t <тип_ФС> <устройство> <точка_монтирования>
```

Устройство — это имя файла этого устройства. Большинство современных CD-ROM- приводов подключаются к IDE контроллеру, поэтому для ссылки на компакт-диск используется имя IDE-устройства — чаще всего `/dev/hdc`. Инсталлятор обычно распознает CD-ROM и создает символическую ссылку `/dev/cdrom->/dev/hdc`.

Типы файловой системы на съемных носителях чаще всего такие:

Таблица 4.3

Название	Система
ext2	Основная файловая система Linux
ext3	Журналируемая надстройка, полностью совместимая с ext2
vfat	Файловая система Windows: FAT, VFAT, FAT32
iso9660	Файловые системы для компакт-дисков

Как правило, монтировать постоянные носители информации (разделы жесткого диска) имеет право только администратор. Съемные же носители разрешается монтировать обыкновенным пользователям.

Специально для них у команды `mount` есть сокращенная форма, при которой достаточно указать либо устройство, либо точку монтирования. Недостающую информацию программа монтирования получает из файла `/etc/fstab`.

Так, для подключения flash-накопителя я:

1. включаю его в USB-порт;
2. перехожу в текстовую консоль или открываю окно виртуального терминала;
3. `mount /media/usbdisk # или mount /dev/sda1`.
   Перед извлечением съемный носитель необходимо размонтировать командой
   `umount [устройство | точка_монтирования]`.

Не сделав этого, вы рискуете потерять все данные на нем. Размонтировать устройство можно только тогда, когда к данным на нем не обращается ни одна программа (нет открытых файлов, ни один каталог не является текущим ни для одной программы). Узнать, какие процессы обращаются к файловой системе, можно с помощью утилиты **fuser**.

При работе с компакт-дисками обычно не возникает сбоев, потому что лоток привода CD-ROM контролируется системой. Вы просто не сможете извлечь компакт-диск до тех пор, пока устройство не будет размонтировано. А вот при работе с дискетами и flash-накопителями проконтролировать вас некому. Помните, что, пока вы не размонтировали дискету, физическая запись на нее не будет произведена. Это значит, что, если вы сохранили свой документ на дискету и вытащили ее из дисковода, файла документа на ней не будет, несмотря на то, что вы вышли из текстового редактора. Вы забыли размонтировать файловую систему!

4.2.6. Добавление собственных команд в контекстное меню KDE

Запустите Konqueror и щелкните правой кнопкой мыши на каком-нибудь файле. Вам чего-то не хватает? Хочется добавить какую-то команду? Или есть команда, которую вы выполняете очень часто? KDE позволяет создавать собственные команды меню, чем мы и займемся в этом пункте.

Давайте создадим дополнительную команду, которая делает файл исполняемым. Это действие очень полезно, если вы часто пишете сценарии **bash** (глава 8). Ведь сценарии **bash** — это обычные файлы, созданные в текстовом редакторе. Чтобы сценарий запускался, нужно с помощью команды `chmod +x имя_файла` сделать его исполняемым — но команду вводить лень, хочется все сделать мышкой...

В любимом текстовом редакторе создайте файл следующего содержания:

Листинг 4.1. Файл make_exe.desktop

```
[Desktop Entry]
ServiceTypes=all/allfiles
ServiceType=application/x-shellscript
Actions=MakeExe

[Desktop Action MakeExe]
Name=Make executable
Name[ru]=Сделать файл исполняемым
Exec=chmod +x %f
Icon=kfm
```

Рассмотрим первую секцию. Первая директива задает тип файлов, для которых можно выполнить указанное действие. В данном случае действие доступно для всех файлов (allfiles). Если вам нужно выполнить какое-то действие для каталога, то значением директивы **ServiceTypes** должно быть inode/directory.

Вообще в качестве значения этой директивы можно указать любой MIME-тип, например:

```
ServiceTypes=audio/x-mp3
```

Если действие должно быть выполнено для всех типов файлов, кроме некоторых, используйте директиву **ExcludeServiceTypes**. Например, действие архивирования не имеет смысла производить над архивами:

```
ServiceTypes=all/allfiles
ExcludeServiceTypes=application/x-zip,kdedevice/*
```

Директива **Actions** определяет действия, описанные в файле. В данном случае описано только одно действие **MakeExe**, которое определено в секции [Desktop Action MakeExe]. Директива **Name** — это надпись, которую вы увидите в контекстном меню KDE. Желательно писать ее на английском языке. Директива **Name[ru]** — это надпись, которую увидит пользователь локализованного KDE.

Директива **Exec** — это команда, которая будет выполнена. %f — параметр, определяющий имя файла; то есть имя файла, на котором вы щелкнули правой кнопкой, будет подставлено вместо %f.

Созданный файл сохраните под именем `make_exe.desktop`.

Как видите, в этом нет ничего сложного. Осталось только сохранить файл в нужном каталоге — `/usr/share/apps/konqueror/servicemenus`.

Для записи в этот каталог нужны права суперпользователя. Если же администратор системы не вы, то скопируйте файл в каталог `~/.kde/share/apps/konqueror/servicemenus`, и новая команда будет доступна только вам.

4.2.7. Новое в KDE 3.4

В состав дистрибутива Fedora Core 4 вошла новая версия KDE 3.4, в которой, как сообщают (`www.kde.org`), исправлено более 6500 ошибок, учтено более 1700 пожеланий, включено 80000 пополнений от различных разработчиков. Среди главных новшеств:

- Полностью обновлена система корзины.
- Новая версия KPDF с возможностью выбора, копирования и вставки текста и изображений из PDF, а также с другими улучшениями.
- Добавлен синтезатор речи с возможностью интеграции в браузер Konqueror, текстовый редактор Kate, PDF-просмотрщик KPDF.
- Приложение Kontact поддерживает больше серверов совместной работы (groupware).

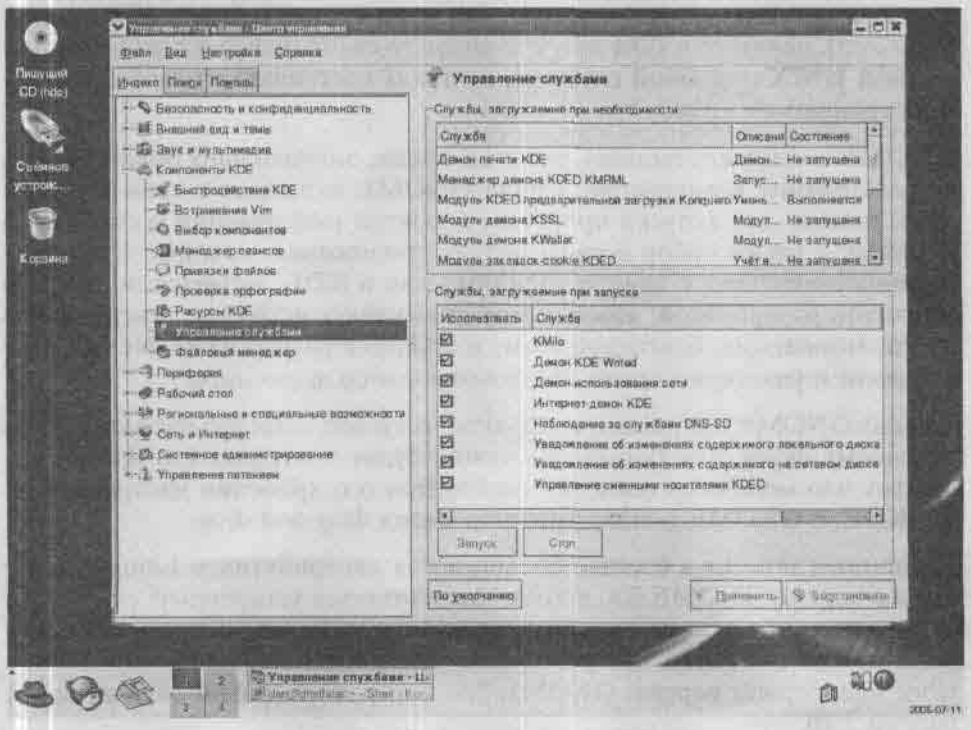


Рис. 4.2. Центр управления KDE 3.4

- Улучшен механизм обнаружения подключенных устройств через HAL. Теперь KDE монтирует съемные носители автоматически, но не спешите радоваться: кириллицу в именах файлов на USB-диске она автоматически не распознает, так что п.2.3 вам еще пригодится.
- Почтовый клиент KMail теперь может использовать KWallet (бумажник KDE) для хранения паролей, а доступ к самому бумажнику можно сделать беспарольным.
- Появилась возможность использования файлов в формате SVG (*Scalable Vector Graphics*) в качестве обоев для рабочего стола.

4.3. Оконная среда GNOME

4.3.1. Общее описание и методика работы

Что за зверь эта GNOME

Среда GNOME (GNU Network Object Model Environment — Сетевая Объектная Среда GNU) — один из основных конкурентов среде KDE. Среда GNOME (`www.gnome.org`) является частью проекта GNU (`www.gnu.org`), начатого в 1984 году и ставящего своей целью создание полноценной UNIX-подобной системы, целиком состоящей из свободно распространяемого кода.

GNOME — дружественная рабочая среда, значительно облегчающая использование компьютера. Среда GNOME включает в себя рабочий стол, панель для запуска программ и показа информации о состоянии системы, а также набор всевозможных приложений, которые тесно взаимодействуют друг с другом. GNOME, как и KDE, является полностью открытой разработкой: каждый может выкачать исходные тексты среды и использовать их. Благодаря этому в процессе разработки GNOME участвовали и участвуют сотни программистов со всего мира.

В среде GNOME настраивается практически все: один раз настроив сеанс по своему вкусу, вам больше не нужно будет повторять его настройку, потому что менеджер сеансов позаботится о сохранении настроек. Как и в KDE, в GNOME поддерживается метод drag-and-drop.

На данный момент в составе большинства дистрибутивов Linux поставляется версия GNOME 2.6, но она уже считается устаревшей, поскольку относительно недавно вышла версия GNOME 2.10. Справедливости ради нужно отметить, что некоторые дистрибутивы (например, Fedora Core 3) содержат версию GNOME 2.8, которая не сильно отличается от версии 2.10.

Основные элементы среды — это рабочий стол, панель GNOME и панель задач. На панели GNOME (узкая полоска вверху экрана) расположены кнопки главного меню **Приложения** (выбор приложения) и **Команды** (различные команды, например, **Выполнить** или **Завершить сеанс**), а также апплеты. Все остальное пространство называется рабочим столом (см. рис. 4.3). Нижняя полоска внизу экрана — это панель задач. Ее можно временно убрать с экрана, щелкнув по стрелке в углу. Апплеты — небольшие программы, которые работают внутри панели и запускаются щелчком мыши по значку на панели. Сразу после установки вы можете увидеть там, например, календарь.

Один из базовых принципов GNOME 2 состоит в том, чтобы упрощать все, что только можно: элементов управления под рукой у пользователя должно быть как можно меньше. Девизом GNOME 2 стала фраза «just works» — «работает, и все», без дополнительных настроек.

Так, монтирование съемных носителей выполняется автоматически: вставьте компакт-диск или flash-накопитель, и на рабочем столе сразу же появится соответствующий значок. Размонтируйте носитель командой **Отсоединить (Eject)** из контекстного меню этого значка. Следует отметить

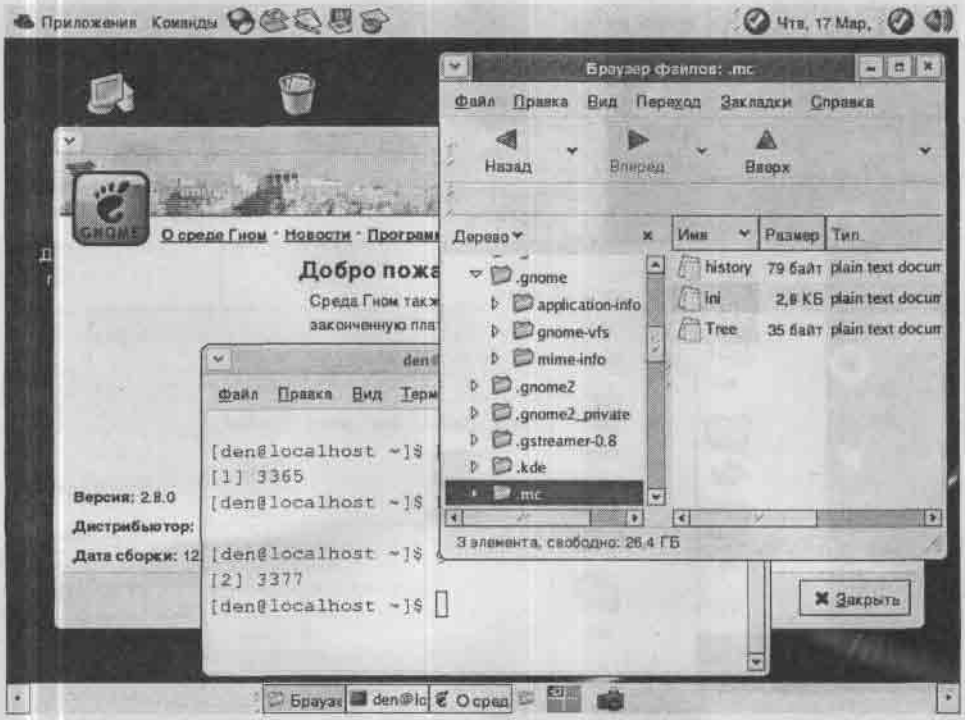


Рис. 4.3. *Рабочий стол GNOME*

также, что GNOME распознает тип съемного носителя и в зависимости от типа запускает то или иное приложение. Например, когда вы вставите DVD-диск, будет запущен DVD-проигрыватель.

Файловый менеджер Nautilus

Начиная с версии 1.4, в пакет GNOME входит файловый менеджер **Nautilus**, в дальнейших версиях ставший файловым менеджером по умолчанию. Это разработка компании Eazel, собравшей у себя ветеранов Apple — создателей интерфейса Macintosh, до сих пор никем не превзойденного по простоте и удобству. В нем нет ничего лишнего: ни панели инструментов, ни адресной строки — только окно с файлами и каталогами.

Каждый каталог открывается в новом окне, а размер, положение окна и представление файлов (значками или списком) привязаны к этому каталогу: в следующем сеансе окно этого каталога откроется в том же месте рабочего стола. Такой интерфейс называется пространственно ориентированным. Идея состоит в том, что на настоящем рабочем столе бумажные папки не замещают друг друга, а всегда лежат там, куда вы их последний раз переложили. Считается, что так новичку легче приме-

Рис. 4.4. Файловый менеджер Nautilus

нить навыки, приобретенные в реальном мире. Если же вы не новичок и привыкли к нормальному интерфейсу браузера, то пользуйтесь правой кнопкой мыши или смените настройки через конфигурационный файл в `~/.gconf/apps/nautilus/preferences`.

Кнопка «Назад», привычная по Проводнику Windows, выглядит как галочка в левом углу строки состояния. В диалоге открытия файла отсутствует строка ввода имени, но ее можно вызвать комбинацией клавиш Ctrl+L.

В Nautilus встроена возможность записи компакт-дисков: программа **nautilus-cd-burner**.

С версии 2.6 основным веб-браузером стал **Epiphany** (`http://www.gnome.org/projects/epiphany`), облегченная версия браузера Galeon на движке Mozilla. Роль почтового клиента исполняет органайзер Evolution.

Настроить среду GNOME можно как из меню **Приложения → Параметры (Applications → Desktop Preferences)**, так и из командной строки. Графический конфигуратор называется **gconf-editor**.

Расширенная система управления MIME-типами

Следующая особенность GNOME 2.8-2.10 — расширенная система управления MIME-типами. Если вы пытаетесь открыть не зарегистрированный в системе MIME-тип (например, файл с расширением .doc, а система не знает, какое приложение обрабатывает файлы с таким расширением), то будет отображено окошко, в котором можно будет связать данный тип с конкретным приложением для его обработки.

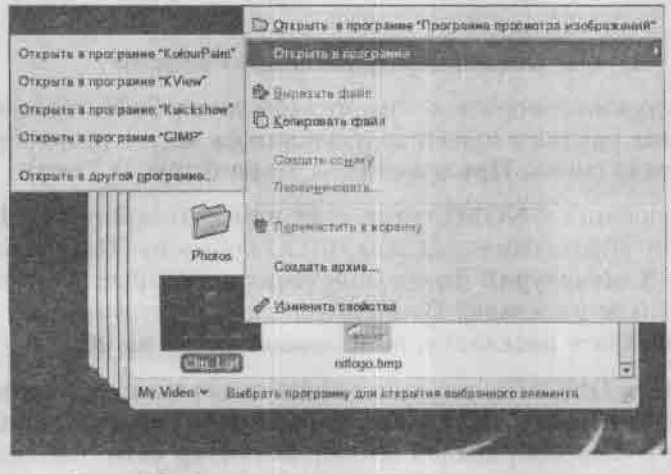


Рис. 4.5. Аналог команды «Открыть с помощью»

Если для данного типа установлено несколько программ (например, для HTML-файлов установлено несколько браузеров), то, щелкнув по файлу правой кнопкой мыши, вы сможете выбрать предпочитаемое приложение — прямо как команда «Открыть с помощью» в Windows. Такая организация интерфейса упрощает работу с Linux начинающим пользователям.

Поддержка службы DNS-Based Service Discovery

Теперь GNOME поддерживает службу DNS-Based Service Discovery, которая позволяет организовать привычное для пользователей Windows сетевое окружение — вам не нужно ни ломать голову над настройкой Samba, ни настраивать посторонние программы для просмотра Windows-сети.

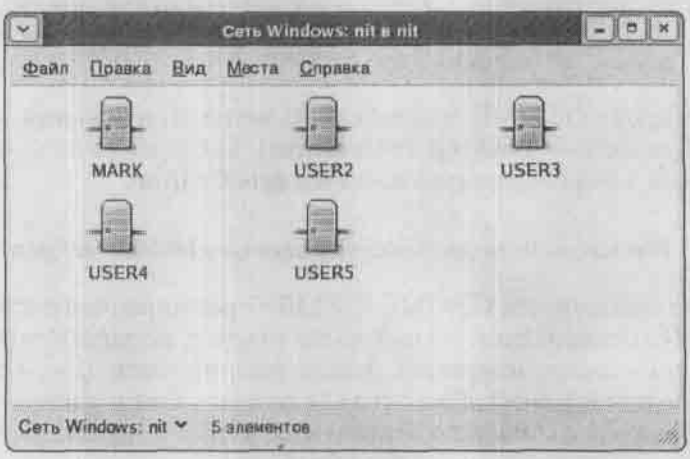


Рис. 4.6. *Самое обычное сетевое окружение*

Особенности последних версий

Думаю, не нужно говорить о том, что сам интерфейс последней версии GNOME стал намного приятнее и появилось много разнообразных тем рабочего стола (меню **Приложения** → **Параметры** → **Тема**):

Центр управления GNOME тоже стал намного приятнее. Чего только стоит апплет управления раскладками клавиатуры (**Приложения** → **Параметры** → **Клавиатура**)! До его появления некоторые пользователи и не знали, что такое раскладка Dvorak (или любая другая) — теперь можно не просто выбрать раскладку, но и «посмотреть» на нее.

Как на панель GNOME, так и на панель задач можно добавить еще несколько полезных апплетов, щелкнув по панели правой кнопкой мыши. Лично мне очень понравился апплет **Монитор сети**, наблюдающий за состоянием сетевого соединения и показывающий, кроме всего прочего,

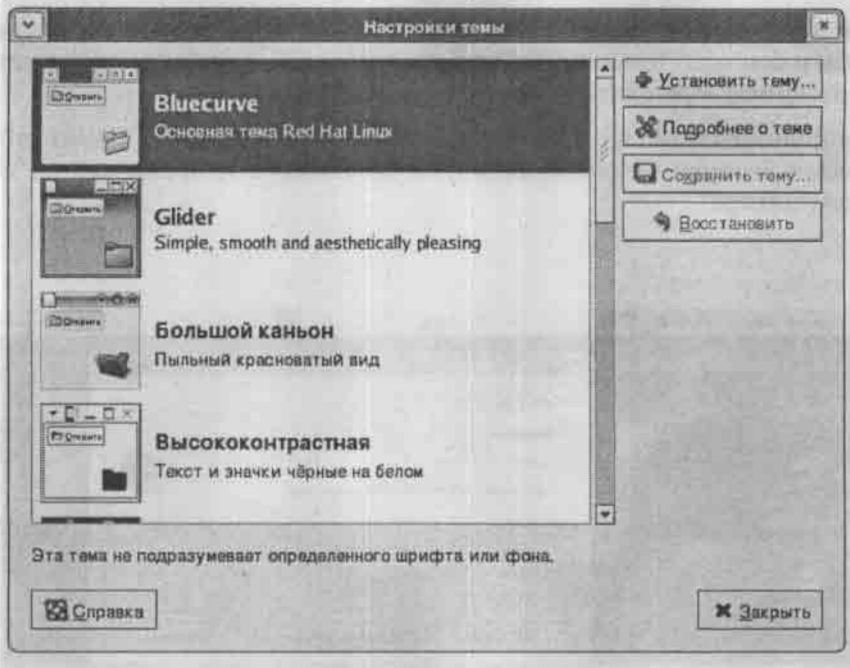


Рис. 4.7. *Выбор темы рабочего стола*

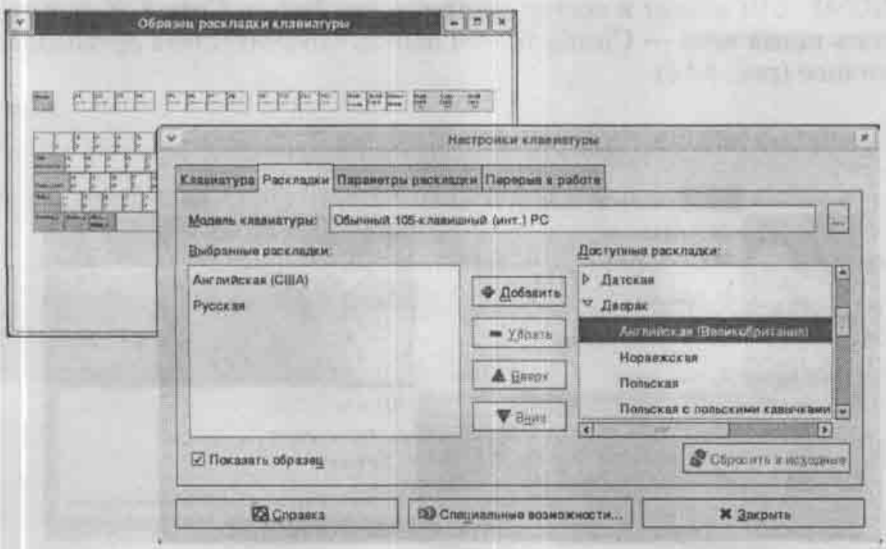


Рис. 4.8. *Выбор раскладки клавиатуры*

еще и силу сигнала, что особенно актуально для беспроводных средств связи и dial-up соединений. Для более подробной информации о сетевых соединениях используйте новую утилиту **nettool**.

Для пользователей ноутбуков особенно актуален апплет **Индикатор состояния батареи**, показывающий, на сколько времени еще хватит заряда аккумулятора.

Рис. 4.9. *Сетевой монитор*

GNOME 2.10 входит в состав дистрибутива Fedora Core 4. К ней добавилась новая тема — Clearlook — и панель GNOME стала организована логичнее (рис. 4.10).

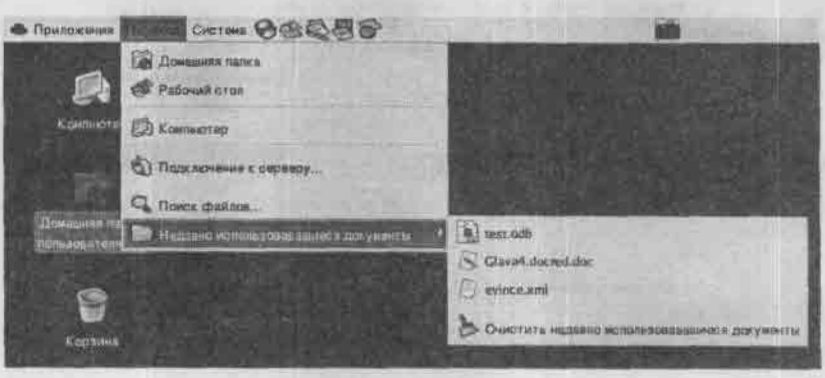


Рис. 4.10. *Панель GNOME 2.10*

4.3.2. Добавление собственных команд в контекстное меню GNOME

Меню **Сценарии** в GNOME устроено и действует аналогично меню **Действия** в KDE. Только если в случае KDE файл контекстного меню имел определенный формат, то в GNOME это — обычный **bash**-сценарий. Хорошо это или плохо, зависит от того, хорошо ли вы знаете **bash**. Хорошее знание языка командного интерпретатора **bash** поможет вам создать очень сложные сценарии, автоматизирующие огромное количество рутинной работы.

Идея достаточно проста. Вы создаете **bash**-сценарий и помещаете его в каталог `~/.gnome2/nautilus-scripts/`. В этом каталоге можно создавать подкаталоги — они будут отображаться как дополнительные меню.

Теперь рассмотрим пример сценария, конвертирующего изображение в формат GIF при помощи программы-конвертера **convert** (листинг 4.2). Файл, по которому пользователь щелкнул правой кнопкой мыши, передается нашему сценарию как первый аргумент (о передаче аргументов сценариям говорится в п.8.1).

Листинг 4.2. Файл x2gif

```
#!/bin/bash
convertprg=`which convert`

while [ $# -gt 0 ]; do
    picture=$1
    filetype=`file $picture | cut -d ' ' -f 3`
    if [ $filetype = "image" ]
    then
        newfile=`echo "$picture" | cut -d . -f 1`
        $convertprg "$picture" "$newfile".gif
    fi
    shift
done
```

4.3.3. Автоматическая смена обоев

Обои на рабочем столе рано или поздно надоедают, и хочется их сменить. Делать это вручную — лень: это ж сколько придется щелкать мышкой. Автоматизируем эту рутинную процедуру, написав сценарий, выбирающий обои из каталога `/home/den/Wallpapers` (где у меня хранятся мои любимые обои) случайным образом. Собственно установку обоев выполняет утилита GNOME **gconftool-2**.

Листинг 4.3. Файл change_wallpaper

```
#!/bin/bash
export DIR='/home/den/Wallpapers/'
export NUMBER=$RANDOM
export TOTAL=0
for f in `ls $DIR`
do
    let "TOTAL += 1"
done
let "NUMBER %= TOTAL"
export CURRENT=0
for f in `ls $DIR`
do
    if [ $CURRENT = $NUMBER ]
    then
        /usr/bin/gconftool-2 -t string
-s /desktop/gnome/background/picture_filename $DIR/$f
        break
    fi
    let "CURRENT += 1"
done
```

Но это еще не все. Можно сэкономить и на запуске сценария change_wallpaper, заставив его запускаться автоматически. Во-первых, вы можете добавить его в сценарии загрузки системы (п. 9.1.2). Для домашнего компьютера, который на ночь выключают, это неплохое решение: вы будете каждый день сидеть за столом с новыми обоями. Во-вторых, вызов сценария change_wallpaper можно поместить в сценарий запуска GNOME. И в-третьих, change_wallpaper можно вручить диспетчеру расписаний ***cron*** (п.9.4) для выполнения по назначенному вами расписанию — хоть каждую минуту.

4.4. Офисные пакеты

Open Office и K Office

В среде Linux наибольшее распространение получили два офисных пакета: K Office и Open Office, оба — свободно распространяемые. В большинство дистрибутивов включены они оба, так что вы можете выбирать инструмент, исходя из конкретной задачи.

Пакет Open Office обладает гораздо большим количеством возможностей, чем K Office, и наиболее приближен по своей функциональности к MS Office.

K Office предназначен для среды KDE, которую чаще назначают средой по умолчанию, и чувствует себя в ней намного лучше, чем Open Office. Например, некоторые комбинации клавиш Open Office не работают из-за того, что они используются самой средой KDE. Разумеется, можно их переназначить, но тогда использовать KDE будет не так удобно.

С другой стороны, K Office хуже справляется с форматами MS Office. При попытке открыть большие документы, например, документ MS Word 97/2000/XP размером 1.2 Мб, программа K Word просто закрылась. Точно так же себя ведет и программа K Spread — прайс-лист одной фирмы (около 1М) загружался около 30 секунд, а когда индикатор достиг отметки 100%, окно программы просто закрылось. K Presenter кое-как открыл презентацию PowerPoint, но... Лучше я не буду говорить о том, что я увидел.

Состав пакета Open Office *Таблица 4.4*

Приложение	Назначение
Writer	Текстовый процессор, предлагающий мощные функции для создания и редактирования документов разных форматов
Calc	Редактор электронных таблиц
Draw	Программа Draw ориентирована на векторную графику и служит для создания рисунков, эмблем и плакатов
Impress	Программа для создания презентаций
Math	Благодаря этому редактору, вы можете вставлять математические формулы в различные документы пакета
Image	Программа предназначена для сканирования и обработки фотографий
Schedule	Планировщик, который поможет вам организовать свой рабочий день

Примечание

Open Office — это бесплатный вариант пакета Star Office от Sun Microsystems. Названия программ, входящих в состав обоих пакетов, отличаются только префиксом, OO или SO соответственно. Например, SO Writer и OO Writer — это одна и та же программа.

Состав пакета K Office *Таблица 4.5*

Приложение	Назначение
K Word	Текстовый процессор, аналог программы MS Word
K Spread	Редактор электронных таблиц. Обладает несколькими оригинальными функциями, которые вы не найдете ни в OO SpreadSheet, ни в MS Excel
K Chart	Приложение для построения диаграмм
K Illustrator, Kontour	Редактор векторной графики
K Presenter	Программа для создания презентаций
K Formula	Редактор математических формул

При помощи K Word можно сохранять текстовые файлы в формате PDF. Выполните команду **Файл → Печать** и вместо принтера выберите опцию **Печать в файл PDF**.

K Spread не поддерживает формат MS Excel, но совместима с форматом GNUmeric — редактора электронных таблиц, входящего в состав большинства Red Hat-подобных дистрибутивов; а уж из GNUmeric можно экспортировать таблицу в MS Excel. Если в вашем дистрибутиве этого пакета нет, то скачать его можно с `http://www.gnome.org/projects/gnumeric`.

Возможностей K Presenter хватит даже для самой сложной презентации. Программа K Presenter, как и все программы K Office, довольно быстро работает, что немаловажно, если вам нужно показывать презентацию на стареньком 486-ом ноутбуке. К тому же, если у вас установлена KDE, скорее всего, будет установлен и K Office, чего нельзя сказать об Open Office.

Уменьшение времени запуска Open Office

Наверное, вы заметили, что любое приложение из пакета Open Office запускается, мягко говоря, медленно. Чтобы ускорить этот процесс, нужно установить утилиту быстрого запуска Open Office. Помните, что-то подобное было в Microsoft Office, только в случае с Open Office данная утилита действительно работает.

Утилита называется Open Office Quick Starter. Пакет, содержащий эту утилиту, называется **ooqs** или **ooqs-kde** — в зависимости от дистрибутива. В случае, если вы используете GNOME, вам нужен пакет `ooqstart-gnome`. В большинстве дистрибутивов эти пакеты имеются. Все, что вам нужно сделать, — установить нужный пакет и наслаждаться быстрым запуском Open Office.

Новое в Open Office 2.0

В состав дистрибутива Fedora Core 4 входит пакет Open Office версии 2.0. По сравнению с версией 1.1 в нем улучшена совместимость с форматами документов MS Office. Текстовый редактор OO Writer теперь поддерживает вложенные таблицы, вертикальную ориентацию текста в ячейке, появилась возможность импорта документов в формате WordPerfect. OO Calc теперь разрешает таблицы того же размера, что и MS Excel, то есть 65536 строк, поддерживает диагональные линии в ячейках и колонтитулы в стиле MS Excel; добавилась также возможность импорта таблиц Lotus 1-2-3 до версии 9.7. Появилась возможность открывать защищенные паролем документы MS Word и MS Excel (если пользователь, конечно, знает пароль).

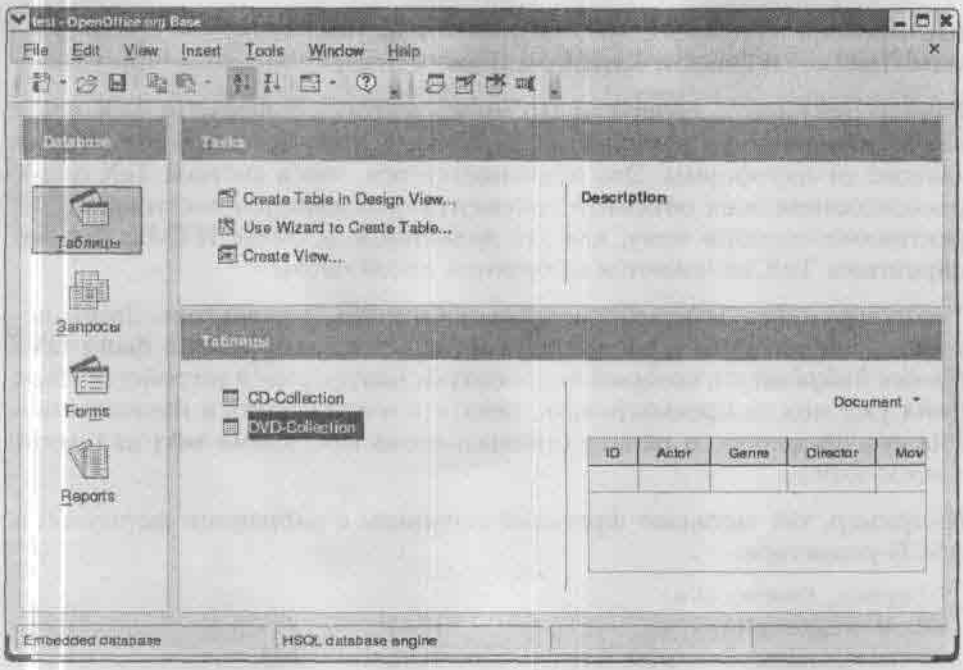


Рис. 4.11. *СУБД OO Base*

На панель рисования OO Draw добавлены новые группы готовых объектов: основные формы, стрелки, элементы блок-схем, звездочки. Полностью переписан движок презентаций, что позволяет OO Impress показывать почти все анимационные эффекты MS PowerPoint.

В составе пакета появилось новое приложение — OO Base. Как нетрудно догадаться по названию, это СУБД, позволяющая создавать и управлять таблицами, запросами, формами и отчетами. Мастер таблиц предлагает множество заготовок таблиц делового и личного (например, фитнесс-дневник) назначения. OO Base понимает базы данных dBase, MS Access, ODBC, JDBC, MySQL, импортирует электронные таблицы и текстовые файлы, может использовать LDAP сервер как источник данных.

4.5. Издательские системы

Язык разметки TeX

TeX (произносится «тех») — это система подготовки документов, содержащих большое количество формул и таблиц. Она была разработана в конце 70-х гг. американским математиком Дональдом Кнутом, автором знаменитой книги «Искусство программирования». Система TeX была

портирована на платформу Unix программистами Говардом Трики (Howard Trickey) и Павлом Куртисом (Pavel Curtis).

Размер документа, содержащего много формул, в формате TeX раз в десять меньше, чем в формате MS Word. К тому же документ TeX не зависит от платформы. Это объясняется тем, что в системе TeX вид и расположение всех объектов документа описываются текстовыми директивами подобно тому, как это делается в формате HTML. Только директивы TeX начинаются с обратной косой черты.

Текст документа TeX набирается в любом ASCII-редакторе. Затем исходный текст (файл с расширением .tex) компилируется в файл DVI (*Device Independent*), который не зависит от платформы и устройства. Этот файл уже можно просматривать, печатать и отправлять в издательство. Для его просмотра и печати предназначена программа **xdvi** из пакета `tetex-xdvi`.

Например, так выглядит фрагмент страницы с набранной формулой в ASCII-редакторе:

```
\layout Enumerate
Математических:
\begin_inset Formula $\sin\frac{\alpha}{2}$
```

А на следующем рисунке показан фрагмент той же страницы, подготовленной к печати.

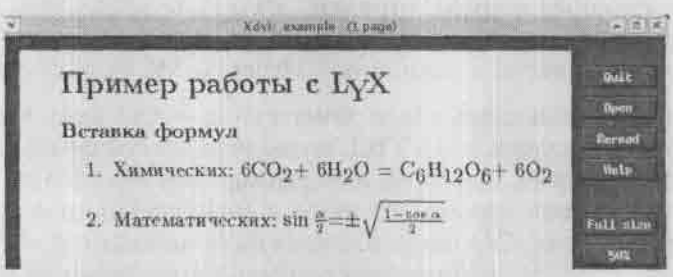


Рис. 4.12. *Фрагмент документа TeX*

В настоящее время оригинальная система TeX почти не используется, а применяются ее модификации, например, система LaTeX, представляющая собой пакет макрокоманд, упрощающий работу с TeX.

TeTeX — система подготовки документов для UNIX-подобных ОС. В состав дистрибутива Fedora Core 3 входит пакет tetex-latex версии 2.0.2. Более ранние версии TeTeX (до версии 0.9) не умели конвертировать TeX-документы в формат PDF. Начиная с версии 0.9, в состав TeTeX входит утилита **pdftex**, которая предназначена для прямого конвертирования tex-файла в формат PDF.

Коллекцию ссылок на справочные материалы по языку TeX можно найти по адресу `http://www.tex.uniyar.ac.ru/doc.htm`.

Текстовый процессор LyX

Работа над любым документом распадается на две части: разработку содержания документа и представление его в удобном для читателя виде. Принцип WYSIWYG («видишь то, что получишь на печати») облегчает только вторую часть работы, и обычный автор, не будучи одновременно дизайнером, не умеет и не хочет использовать все возможности какого-нибудь MS Word для грамотного оформления своей статьи. Автор предпочел бы описать структуру своего документа так, чтобы легко было применить к нему стиль, разработанный профессиональным дизайнером, и сосредоточиться на своей работе, а не на подробностях оформления.

Этот принцип называется WYSIWYM (*What You See Is What You Mean*, «видишь то, что подразумеваешь»), и реализован в текстовом процессоре LyX (`http://www.lyx.org`). LyX — это визуальный редактор TeX, отображающий документ не в точности так, как он будет выглядеть на бумаге, но достаточно похоже, и вместе с тем не затемняющий его структуры. На рисунке 4.13 показано продолжение работы над документом example: вставка сноски.

LyX не создает DVI-файлы в домашнем каталоге пользователя. Найти их можно во временном каталоге. Обычно это `/tmp/lyx_tmpdir??????`, где ?????? — это произвольная последовательность букв и цифр. Если вы запускали просмотр DVI несколько раз, у вас будет несколько таких каталогов. Поищите DVI-файлы в этих каталогах — вы обязательно найдете нужный вам файл. Для автоматизации поиска этих файлов откройте окно виртуального терминала и введите команду:

```
find /tmp/lyx_tmpdir* -name *.dvi
```

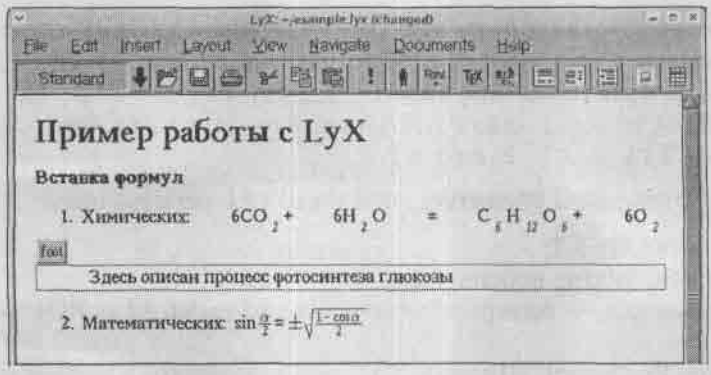


Рис. 4.13. *Вставка сноски*

Издательская система Scribus

TeX/LaTeX — удачная система для верстки научных работ, но в мире Windows есть много издательских систем, куда лучше приспособленных для создания макетов: Corel Ventura®, Quark Xpress®, Adobe Pagemaker®, InDesign®... Что же делать пользователю Linux? Не устанавливать же Windows из-за PageMaker.

Положение сторонников свободного программного обеспечения изменилось в 2003 г., когда вышел финальный релиз пакета компьютерной верстки Scribus 1.0 (`http://www.scribus.org.uk`, автор ядра программы — Франц Шмид).

Scribus — полноценная издательская система, в которой реализованы все необходимые для предпечатной подготовки документов функции, в том числе цветоделение по каналам CMYK, система управления цветом, поддержка Unicode для вывода на нетрадиционных языках, экспорт результатов в форматы PDF и PostScript. В качестве внутреннего формата хранения данных программа использует язык описания XML, что облегчает экспорт данных в другие пакеты.

Программа Scribus поддерживает такие распространенные графические форматы, как Encapsulated PostScript (EPS), JPEG, TIFF, Portable Network Graphics (PNG), XpixMap (xpm). Начиная с версии 1.0, Scribus поддерживает русский язык.

Scribus был включен в состав следующих дистрибутивов:

- Debian (нестабильный релиз)
- Gentoo Linux
- Mandrake 8.2
- Lindows
- Lycoris
- Fink 0.4

Если вы используете один из этих дистрибутивов, то, возможно, Scribus уже установлен у вас. А если нет, то пакет `scribus` версии 1.2 можно скачать из репозитария Sisyphus по адресу `ftp://ftp.altlinux.ru/pub/distributions/ALTLinux/Sisyphus/SRPMS.classic/scribus-1.2.1-alt1.1.src.rpm`.

Перед его установкой убедитесь, что у вас уже установлены:

- библиотека Qt-3.3;
- библиотека libpng версии 1.2.x;
- `ghostscript` — интерпретатор языков PostScript и PDF — версии 7.x;
- шрифты PostScript. Шрифты TTF поддерживаются Scribus начиная с версии 0.4.7.

- среда KDE: для работы самой Scribus она не нужна, но понадобится, если вы хотите использовать метод drag-and-drop.

Дополнительно можно установить Little CMS — систему управления цветом и цветоделения с поддержкой ICC-профилей, а взять ее можно с `http://www.littlecms.com`. В меню **Edit** появится команда **Color Management System.**

4.6. Графика в Linux

Несколько графических программ входят в состав большинства современных дистрибутивов. Далее я кратко опишу их и дам им оценку.

4.6.1. Графические редакторы

Редактор растровой графики GIMP — обрабатываем фото

The GIMP (*The GNU Image Manipulation Program*) — свободно распространяемый редактор растровой графики, по своим функциям ставший достойной заменой программе Photoshop. Вот некоторые возможности GIMP (далеко не все!):

- поддерживает практически все форматы растровой графики: GIF, JPEG, PNG, TIFF, BMP, AVI, MPEG, PS, PCX, XPM, TGA и другие;
- содержит множество инструментов для обработки изображений;
- поддерживает работу со слоями;
- полный Альфа-канал;
- может использоваться для создания анимации — Gif, AVI;
- содержит средства для разложения видео в форматах MPEG1, XANIM на кадры;
- поддерживает сценарии и расширения, благодаря которым можно добавить в программу новые функции. Поддерживаются сценарии Script-Fu, Perl-Fu, Python-Fu;
- обладает улучшенным управлением памятью: в результате размер изображения ограничен только объемом жесткого диска;
- операции Отмена (Undo) и Повтор (Redo) ограничены только дисковым пространством;
- имеет многочисленные инструменты выделения областей: выделение прямоугольной, произвольной, эллиптической и других областей;
- снабжен огромным набором графических фильтров, расширяемым с помощью плагинов.

Программа The GIMP доступна по адресу `http://www.gimp.org`.

Оценка: *отлично.*

KPaint

Программа KPaint, входящая в состав KDE, представляет собой полную противоположность программе The GIMP . KPaint обладает небольшим числом функций и инструментов, в ней нет системы плагинов, а о каналах и слоях эта программа вообще ничего не знает. Даже для создания самых простых изображений приятнее и удобнее использовать программу GIMP. KPaint поддерживает форматы BMP, EPS (встраиваемый PostScript), TIFF, GIF, JPEG, PNG, ICO, PGN, XPM.

Оценка: *удовлетворительно*.

KIconEdit — редактор иконок

Программа KIconEdit — редактор пиктограмм KDE. Используется для создания пиктограмм для среды KDE. Довольно простой редактор, позволяющий быстро создать пиктограмму для вашей программы.

Оценка: *хорошо*.

4.6.2. Программы — просмотрщики изображений

Следующая группа программ — это программы просмотра изображений: GQView, KView, KuickShow, ImageMagick, Electric Eyes (ee). Я предпочитаю программу GQView — она больше напоминает мне популярную программу ACDSee.

GQView

GQView можно использовать вместе с программами The GIMP, Electric Eyes и XPaint. Для редактирования загруженного изображения в программе The GIMP нажмите Ctrl + 1, в Electric Eyes — Ctrl + 2, в XPaint — Ctrl + 3. Разумеется, указанные программы должны быть установлены. Вы хотите использовать другой редактор? Нажмите кнопку **Конфигурация** на панели инструментов программы, в открывшемся окне перейдите на закладку **Редакторы** и пропишите свой редактор (нужно указать название редактора и командную строку).

Программа может создавать коллекции рисунков, находить похожие файлы (по имени, размеру и другим параметрам).

Программа поддерживает форматы файлов: JPEG, TIFF, GIF, PNG, XPM, BMP, PCX, PGM, PPM. Могу порекомендовать для использования в качестве основного просмотрщика.

Оценка: *отлично*.

Electric Eyes

Следующая программа — Electric Eyes. Эта программа в некоторых дистрибутивах не устанавливается по умолчанию или вообще отсутствует. Для ее установки нужно установить пакет «ee». Взять его можно с `http://ftp.vn.ua/pub/unix/redhat-7.2/RedHat/RPMS/ee-0.3.12-5.i386.rpm`.

Electric Eyes — не только просмотрщик: с его помощью можно редактировать изображения, сохранять файлы в других форматах.

Для перехода в режим редактирования щелкните правой кнопкой мыши на изображении и выберите команду **Просмотр → Показать/убрать панель редактирования**.

Используя панель редактирования, очень легко отредактировать цветовую гамму рисунка, изменить размер, повернуть изображение, сделать скриншот экрана или выбранного окна. Для «фотографирования» экрана нажмите кнопку **Снять содержимое экрана целиком и сейчас**. Вы также можете выбрать окно для снятия — нажмите кнопку **Выбрать окно для снятия**.

Интерфейс программы немного необычен, но со временем к нему можно привыкнуть. Как и в программе The GIMP, любое действие здесь производится через контекстное меню, то есть нужно активно использовать правую кнопку мыши.

Программа поддерживает форматы: JPEG, GIF, PNG, TIFF, PS, BMP и другие.

Оценка: *отлично*.

KuickShow

Программа быстрого просмотра KuickShow соединяет в себе функции файлового браузера с программой просмотра. Программа относится к среде KDE, чем и объясняется первая буква в ее названии.

При щелчке по имени файла этот файл будет загружен в новое окно программы. Файлы можно открывать и в том же окне, и в новом. При желании можно установить полноэкранный режим просмотра. Программа в состоянии организовать даже слайд-шоу. Поддерживаются все графические форматы, известные KDE (*.jpeg, *.jpg, *.gif, *.xpm, *.ppm, *.pgm, *.pbm, *.pnm, *.png, *.bmp, *.psd, *.eim, *.tiff, *.xcf).

В программу встроены функции предварительного просмотра и печати. KuickShow обладает всеми необходимыми для просмотрщика функциями — ни больше, ни меньше. Конечно, в отличие от GQView, она не

умеет составлять списки коллекций, но я сомневаюсь, что вам когда-нибудь понадобится эта функция.

Существенным достоинством программы является то, что для просмотра каталогов используется Konqueror, поэтому вам доступны все операции над файлами, которые вы можете использовать в файловом менеджере. Можно настроить комбинации клавиш управления просмотром.

Оценка: *отлично*.

KView

Программа KView является достойной альтернативой программе KuickShow. KView не умеет показывать слайд-шоу в автоматическом режиме, зато она в состоянии повернуть изображение на заданный угол, отразить и сохранить результат, а также работать в полноэкранном режиме. Для перехода в этот режим нажмите комбинацию клавиш Ctrl + Shift + F. Кроме того, KView умеет еще и сканировать изображения.

Оценка: *отлично*.

Kooka

Следующая группа программ предназначена для сканирования изображений: это Kooka и XSane. Первая программа, как указывается в документации KDE, предназначена для сканирования и распознавания текста. Коротко о ней можно сказать так: до FineReader ей еще очень далеко.

Оценка: *удовлетворительно*.

XSane

Для сканирования изображений я предпочитаю использовать программу XSane (`http://www.xsane.org`). Правда, она работает не очень стабильно, но результаты сканирования лучше, чем у программы Kooka.

Оценка: *удовлетворительно*.

4.6.3. Как сделать снимок экрана

KSnapShot

Эта программа предназначена для захвата экрана или отдельного окна и сохранения результата в PNG-файл. Скриншоты для этой книги я делал именно с помощью KSnapShot. Вообще не представляю, как бы я обошелся без этой программы. Можно, конечно, использовать Electric

Eyes, но запускать редактор изображений для того, чтобы сделать пару скриншотов, не очень рационально.

Можно воспользоваться комбинациями клавиш KDE — Ctrl + Print и Alt + Print. Первая делает снимок всего экрана, а вторая — только активного окна. Но куда потом вставить этот снимок? В KPaint? Попробуйте. В нормальный редактор, например, The GIMP, копию экрана вставить невозможно, а использовать KPaint очень неудобно.

KSnapShot позволяет установить задержку, за время которой можно подготовить экран к съемке: открыть нужные окна, меню приложения или меню KDE. Через установленное время программа сфотографирует весь экран или текущее окно, оповестит об этом звуковым сигналом и предложит сохранить снимок.

ImageMagick

Пакет ImageMagick обычно используется для просмотра изображений и преобразования их в другой формат. Честно говоря, мне не понравился интерфейс и набор функций этого просмотрщика. Но в пакете ImageMagick есть полезная утилита **import**, позволяющая легко и быстро сделать снимок экрана. Может быть, среда KDE у вас вообще не установлена, поэтому программой **ksnapshot** воспользоваться нельзя? Тогда введите (в X-терминале или окне запуска приложений вашей оконной среды) команду:

```
$ sleep 5; import -window root screen.png
```

Команда sleep генерирует необходимую задержку в секундах. Думаю, 5 секунд хватит, чтобы вы смогли привести экран в нужный вид: вывести на передний план нужное окно, выбрать нужный пункт меню и т.п. Через 5 секунд будет выполнена команда **import**. Она сделает снимок корневого окна, то есть всего экрана, и запишет его в файл `screen.png`. Формат PNG для снимков экрана оптимален. Если вам нужен другой формат, например, JPG, просто измените расширение выходного файла:

```
$ sleep 5; import -window root screen.jpg
```

А как сделать снимок конкретного окна? Первый способ — указать координаты или геометрию окна, предварительно узнав их по команде **xwininfo**:

```
$ sleep 5; import -crop 400x300 screen1.png
$ sleep 5; import -geometry геометрия screen2.png
```

После запуска программы **xwininfo** указатель мыши изменит свой вид со стрелки на крестик. Щелкните по окну, и программа предоставит о нем подробную информацию.

И второй способ:

```
$ import window.png
```

Команда **sleep** не нужна, поскольку программа **import** предоставит вам возможность выбрать нужное окно — указатель мыши тоже изменится на крестик. Щелкните на нужном окне, и буквально через полсекунды в файл `window.png` будет записан образ выбранного окна.

Программа **import** довольно гибкая, рекомендую прочитать man этой программы — вы найдете там много интересного.

А что если вам нужно сделать снимок текстовой программы, а не графического окна? Проще всего запустить эту программу в X-терминале, сделать снимок окна X-терминала и отрезать обрамление окна с помощью GIMP.

4.7. Полезные трюки

Просмотр в консоли документов MS Word и PDF

Для просмотра и редактирования документов MS Word с успехом используется Open Office Writer, но иногда нам нужно быстро просмотреть документ (например, вспомнить номер телефона, указанный в конце документа), и запускать тяжеловесный OO Writer не хочется.

Для просмотра документов MS Word нам понадобятся две утилиты — **wvWare** и **w3m**. Первая — это конвертер документов Word в формат HTML (`wvware.sourceforge.net`), а вторая — универсальный браузер, которым мы будем просматривать полученный HTML-файл. Обе утилиты входят в состав современных дистрибутивов, правда, могут не устанавливаться по умолчанию.

Итак, для просмотра документа `document.doc` введите команду (конвейер):

```
$ wvWare -x /usr/lib/wv/wvHtml.xml document.doc | w3m -T text/html
```

Если вы собираетесь пользоваться этой длиннющей командой часто, оформите ее как bash-сценарий `viewdoc`, не забыв сделать файл `viewdoc` исполняемым. Поместите сценарий в каталог `/usr/local/bin`, чтобы он был доступен всем пользователям системы.

```
#!/bin/bash
wvWare -x /usr/lib/wv/wvHtml.xml $1 2>/dev/null | w3m -T text/html
```

Обратите внимание на перенаправление «`2>/dev/null`». Таким образом мы подавляем сообщения об ошибках, чтобы они не путались с выводом конвертера.

Теперь для просмотра документа document.doc можете воспользоваться командой:

```
$ viewdoc document.doc
```

Такой же сценарий можно написать и для просмотра PDF-документов. Конвертером в этом случае будет утилита **pdftohtml** (`http://pdftohtml.sourceforge.net`). Для просмотра созданного программой HTML-кода мы будем применять текстовый браузер **elinks**.

Итак, для просмотра файла `file.pdf` будем используем команду:

```
$ pdftohtml -q -noframes -stdout  file.pdf | elinks
```

Опять создадим сценарий `/usr/local/bin/viewpdf`, автоматизирующий работу:

```
#!/bin/bash
pdftohtml -q $1 ~/temp.html
elinks ~/temp.html
```

А вот в дистрибутиве Fedora Core 4 таких фокусов проделывать не нужно: в него включен **Evince** — быстрый просмотрщик документов в формате PDF и PostScript.

Глава 5

ЗВУК И ВИДЕО В LINUX

- ПОЧЕМУ ВОСПРОИЗВЕДЕНИЕ АУДИО В LINUX ЛУЧШЕ, ЧЕМ В WINDOWS
- ПРОСЛУШИВАНИЕ МУЗЫКИ
- «ОГРАБЛЕНИЕ» AUDIO-CD
- ПРОГРАММЫ ДЛЯ ПРОСМОТРА ВИДЕО
- ВОСПРОИЗВЕДЕНИЕ НЕПОДДЕРЖИВАЕМЫХ ФОРМАТОВ

5.1. Почему воспроизведение аудио в Linux лучше, чем в Windows

Самыми распространенными аудиоподсистемами для Linux являются OSS (Open Sound System) и ALSA (Advanced Linux Sound Architecture). Обе системы поддерживают большое количество ISA- и PCI-звуковых карт, поэтому, скорее всего, вам не придется разыскивать в Интернете драйвер для своей звуковой карты. В последнее время чаще используется система ALSA, которая полностью совместима с системой OSS, но содержит много дополнительных функций.

Сейчас Linux, если не идеально, то, во всяком случае, подходит для работы со звуком. Когда Линусу Торвальдсу прислали первые патчи, оптимизирующие Linux для работы с аудиоинформацией (так называемые low latency-патчи), он не одобрил эту идею. В результате — отставание от Windows по работе со звуком как минимум на три года. В 1995 году насчитывалось 30-35 (сейчас насчитывается около 800) приложений для Linux, способных работать со звуком. Работать-то они работали, но довольно криво. Сейчас объясню, почему.

Для работы со звуком в реальном времени нужно минимизировать задержки (англ. latency — время ожидания). Latency в 100 мс вы уж точно услышите невооруженным ухом, задержку в 10 мс можно услышать в виде небольшого шума на фоне — тумана. Идеальное время задержки — 3 мс для аудио (WAV) и 1 мс для MIDI. Кстати, проблема latency — это проблема не только Linux, а всех операционных систем, не являющихся системами реального времени (RTOS — Real Time Operation System) — Windows, MacOS. До появления системы ALSA время задержки при работе с аудиоинформацией в Linux (использовалась система OSS/Free) составляло около 150 мс. Система ALSA снизила время задержки до 6 мс — результат лучше, чем у Windows 2000.

В настоящее время задержки (последняя версия ALSA) составляет 4.3 мс. Это довольно неплохо, что **позволило Linux вырваться на второе место по обработке аудиоинформации среди не-RT операционных систем.** На первом месте — MacOS X (CoreAudio API), на третьем — Windows 2000 (ASIO) и Mac OS 9.

В пользу ОС Linux говорит также ее надежность и стабильность при работе с любыми данными. Даже если взять непрофессиональную работу с мультимедиа-данными — прослушивание MP3: в Linux у меня еще ни разу не заикался Xmms, что бы я ни делал. В Windows же при открытии больших документов в том же Word наблюдаются искажения при проигрывании музыки с помощью WinAmp.

5.2. Прослушивание музыки

mpg123

Это самая простая утилита, позволяющая слушать музыку в консольном режиме. Скачать ее можно с домашней страницы разработчика `www.mpg123.de`.

В графическом режиме запускайте ее через виртуальный терминал:

```
$ mpg123 file.mp3
```

Программа позволяет прослушивать музыку, записанную в форматах MPEG 1.0/2.0 (уровни 1, 2 или 3). С помощью **mpg123** вы можете как проигрывать отдельные песни, так и создавать списки песен:

```
$ mpg123 -@ file-list.txt
```

В файле `file-list.txt` перечислите имена файлов (по одному в каждой строке), которые вы хотите прослушать. Можно указывать файлы, расположенные как на локальной машине, так и на удаленной, например:

```
/home/den/mp3/track01.mp3
ftp://ftp.server.ru/pub/song.mp3
http://www.server.ru/audio/track8.mp3
```

Еще одна полезная консольная программа — **cplay** (`http://www.tf.hut.fi/~flu/cplay`). Это оболочка для различных аудиоплейеров, позволяющая удобно создавать списки песен для проигрывания. Вы переходите в нужный вам каталог, выбираете песню, нажимаете Enter, и **cplay** запускает **mpg123** для проигрывания выбранного файла. Использовать эту программу намного удобнее, чем **mpg123**, поскольку вам не нужно вручную создавать списки песен, к тому же **cplay** отображает индикатор проигрывания и время, прошедшее с начала проигрывания. Для выхода из программы нажмите клавишу <Q>.

Xmms

Наверное, самым знаменитым проигрывателем MPEG-файлов является **Xmms** — полный аналог популярной программы WinAmp (`http://www.xmms.org`). К сожалению, в дистрибутивах, основанных на Red Hat начиная с версии 8.0, из него исключена возможность проигрывания MP3. Программы с открытым исходным кодом оказались законодательно несовместимы с этим коммерческим форматом, и компания Red Hat переработала все мультимедийные приложения, удалив из них весь код, связанный с MP3.

Пользователям таких дистрибутивов можно посоветовать скачать раннюю (1.2.7) версию пакета **xmms** с `ftp://ftp.sunet.se/pub/multimedia/xmms/1.2.x` и дополнить ее из `http://mcmcc.bat.ru/RPMS/mpg123-xmms-1.2.7-13.p.i386.rpm`.

Другие программы

Для проигрывания MIDI-файлов в состав Linux (не во всех дистрибутивах) входят сразу две программы: **KMid** и **AWE32** (полное название TkAWEMidi). Первая программа входит в состав KDE (пакет `kdemultimedia`), поэтому, если среда KDE у вас установлена, она всегда будет вам доступна. Кроме MIDI-файлов, **KMid** позволяет проигрывать и караоке (KAR-файлы).

Также две программы предназначены для регулирования громкости — **Aumix** и **KMix**. Обе программы позволяют установить уровень громкости для всех аудиоустройств, установленных в системе.

В состав KDE входит аудиосервер aRts (демон **artsd**; в среде GNOME ту же роль играет eSound, **esd**). Как написано в документации по KDE, «звуковой сервер позволяет вам слышать системные звуки и при этом одновременно проигрывать mp3-файл или играть в игру с фоновой музыкой. С ее помощью системные звуки обогащаются различными эффектами, а программисты имеют возможность легко включать в программы поддержку звука».

Но не все так красиво, как кажется на первый взгляд. Если вы запустите сервер aRts, у вас не будет работать ни одна программа-проигрыватель, которая его не поддерживает. Любая программа, которая попытается обратиться к звуковой плате в обход aRts, получит сообщение, что устройство занято другой программой. Не поддерживают aRts уже рассмотренные проигрыватели **mpg123** и **Xmms**.

Примечание

В программу Xmms можно включить поддержку сервера aRts, добавив модуль aRts Driver 0.4 (libartsout.so). Этот модуль не устанавливается по умолчанию в некоторых дистрибутивах.

Поддерживает звуковой сервер MP-проигрыватель **NoAtun**, входящий в состав KDE. Поэтому вам придется решить, какой проигрыватель использовать. Если вы хотите использовать Xmms, тогда вы не будете слышать системные звуки, или же вы будете использовать **NoAtun**, но будете слышать звуки KDE. Лично я предпочитаю первое: дело в том, что любая программа, поддерживающая aRts, «съедает» больше системных ресурсов, чем Xmms.

Рассмотрим пару полезных трюков, которые можно проделать с помощью **NoAtun**. Допустим, вы хотите слышать одну музыку, то есть вам нужно убрать голос исполнителя. Выполните команду меню **Параметры → Эффекты** и установите эффект VoiceRemoval.

Конечно, голос удаляется не всегда: иногда он все еще слышен, хотя и слабо. В этом случае нужно применить эффект **VoiceRemoval** несколько раз (два и более).

Вам также доступны эффекты реверберации, расширения стереобазы (ExtraStereo), питч (замедление и ускорение воспроизведения).

При запуске любой программы, использующей aRts, будет запущен сервер aRts, если он не был до этого запущен. Сервер aRts можно запускать и при загрузке KDE. Для этого запустите Центр управления KDE и перейдите в раздел **Звук → Звуковой сервер**. В этом разделе вы можете установить различные параметры звукового сервера — от метода ввода/вывода звука до частоты дискретизации. В качестве метода ввода/вывода звука вы можете выбрать один из методов: Open Sound System, Advanced Linux Sound Architecture, Threaded (многопотоковая) OSS или Автоопределение.

Вам уже надоело слушать MP3-файлы и теперь хочется прослушать компакт-диск? Проигрывать компакт-диски можно или с помощью специальной программы **KsCD** (входящей в пакет kdemultimedia), или используя какой-либо другой проигрыватель, который поддерживает компакт-диски (например, тот же **Xmms**).

Вы можете редактировать названия песен с помощью редактора базы данных CD (FreeDB). По окончании редактирования вы можете сохранить названия песен локально или же на сервер FreeDB, чтобы другие пользователи не заполняли названия альбома и дорожек вручную. Впоследствии вы сами можете получить эти данные от сервера FreeDB. Однако эту возможность вы вряд ли будете использовать, разве что сами организуете свой сервер FreeDB и будете использовать его вместе со своими знакомыми.

5.2.1. Сравнение Xmms и NoAtun

Сегодня самыми популярными проигрывателями для Linux являются программы **Xmms** и **NoAtun**. Первая программа поставляется с большинством дистрибутивов, а вторая входит в состав KDE. Как вы используете программу **Xmms**? Скорее всего, просто для прослушивания музыки, а она много чего умеет. В этом параграфе мы поговорим о нестандартных возможностях программы **Xmms** — плагинах, скинах. Параллельно будем сравнивать возможности **Xmms** с возможностями программы **NoAtun**.

Полное название проигрывателя **Xmms** — X Multimedia System. Программа **Xmms** использует интерфейс WinAMP и поддерживает скины программы WinAMP. **Xmms** может воспроизводить звук, записанный в форматах MP3, WAVE, MOD, S3M и других. Версия 1.2.7 поддерживает плагины input, output, general и visualization.

Окно программы **Xmms** состоит из трех частей: главное окно, эквалайзер и редактор песен. На рис. 5.1 изображена программа **Xmms** в «полной комплектации».

Рис. 5.1. *Проигрыватель Xmms*

Как я уже отмечал, программа поддерживает форматы MPEG 1/2/3, WAVE, MOD, S3M, а также формат AudioCD. Таким обилием форматов не может похвастаться программа **NoAtun**. Честно говоря, меня не интересуют форматы MOD, S3M, CIN (ID Software), мне нужна поддержка только форматов MP1/2/3, WAVE и AudioCD.

Программе **NoAtun** очень не хватает поддержки AudioCD. Конечно, можно запустить проигрыватель AudioCD **KsCD**, но зачем — ведь есть **Xmms**. Если вы хотите узнать, какие еще форматы поддерживает ваш **Xmms**, откройте окно опций (нажмите Ctrl + P). На странице **Звуковые модули Ввода/Вывода** представлены все модули, отвечающие за поддержку тех или иных форматов.

В этом же окне вы можете выбрать модуль вывода: драйвер OSS, модуль вывода звука eSound или модуль записи на диск. Для нормального воспроизведения нужно использовать драйвер OSS. Нажав кнопку **Настройка**, вы можете настроить выбранный драйвер. Если у вас установлена звуковая карта Sound Blaster 16, ViBRA или AWE32/64, в настройках драйвера OSS вы можете установить трехмерное стереорасширение (3D Stereo Enhancement).

Теперь немного поговорим о модуле записи на диск. Если установить этот модуль в качестве основного модуля вывода, музыка не будет воспроизводиться с помощью драйвера OSS, а будет записываться на диск в виде WAVE-файла. А это значит, что любой формат, который поддерживается модулями ввода программы ***Xmms***, вы можете конвертировать в WAVE.

Скорее всего, MP3 конвертировать в WAVE вам не придется: это просто неразумно. А вот «сграбить» пару компактов, не выходя из любимой программы — это можно. Откройте компакт-диск (для этого нажмите Shift + L, чтобы загрузить каталог, и выберите каталог `/mnt/cdrom`) и включите модуль записи на диск. Теперь нажмите кнопку **Play**. Процесс записи можно остановить в любой момент. Все! Все дорожки будут записаны на диск (в ваш домашний каталог) в виде WAVE-файлов. Имена файлов будут типа `TrackN.wav`.

Естественно, во время записи музыки вы не услышите, потому что модуль OSS не используется. Стоит заметить, что **Xmms** «грабит» компакты значительно быстрее, чем программа **Grip** (по крайней мере у меня). Потом эти файлы можно будет преобразовать в MP3 с помощью любого конвертера, например, **lame** или **l3end**. Преобразовать WAVE в MP3 можно так:

```
$ lame -b 128 input.wav output.mp3
```

Опция **-b** устанавливает скорость потока (bitrate) 128 Кбит/с. Более подробно о преобразовании WAVE в MP3 вы можете прочитать на страницах помощи программы **lame**, выполнив следующую команду:

```
$ man lame
```

Программа **NoAtun** также обладает модулем записи на диск. Его можно выбрать в окне **Эффекты**. Должен заметить, что в программе **NoAtun** этот модуль не работает. Впрочем, он и не нужен — все равно программа **NoAtun** не поддерживает AudioCD.

На странице **Эффекты → Общие модули** вы можете выбрать нужный вам эффект или дополнительный модуль. Например, модуль **Смена песни** позволяет установить команду, которая будет выполнена после воспроизведения песни, и команду, которая будет выполнена после воспроизведения всех песен. Первая команда может включать параметр %s, который будет заменен названием только что воспроизведенной песни. Эту возможность можно использовать для автоматического конвертирования WAVE в MP3. Вы будете слушать музыку, а **Xmms** будет конвертировать ее в формат MP3.

При воспроизведении музыки вы можете использовать один из эффектов:

- LADSPA (по умолчанию)
- SOX
- Модуль удаления голоса
- Улучшенное стерео (расширение стереобазы)
- Эхо.

Включив модуль удаления голоса, вы будете слышать только одну музыку. Конечно, качество работы этого модуля очень сильно зависит от качества записи вашего MP3. Этот модуль намного лучше работает с AudioCD — голос полностью удаляется.

Аналогичные эффекты поддерживает и программа **NoAtun**. К тому же в программе **NoAtun** имеется возможность применить один и тот же эффект несколько раз.

Теперь рассмотрим модули визуализации. Лично мне больше всех нравится модуль **G-Force**. Кроме обыкновенной визуализации, он обладает множеством дополнительных функций, например, функцией измерения скорости кадров (fps). Нажмите <?> в окне модуля, и вы увидите список функций модуля.

Модули визуализации поддерживаются и программой **NoAtun**, но почему-то они очень нестабильны, и их использование может вызвать сбой **NoAtun**.

Нам осталось рассмотреть только скины. Нажмите Alt + S, чтобы открыть окно просмотрщика «шкур» — так официально называется окно выбора скина. У вас нет «шкур»? Где их взять? На официальном сайте `http://xmms.org` или на `http://www.skinbase.org`. Подойдут также ваши «шкуры» от WinAmp'a. Zip-файлы скинов нужно скопировать в

каталог /usr/share/xmms/Skins. После этого установленные «шкуры» будут доступны всем пользователям системы. Если вы хотите, чтобы с трудом добытая «шкура» была доступна только вам или же если у вас нет прав на запись в каталог /usr/share, установите «шкуру» в каталог ~/.xmms/Skins.

Как видите, программа **NoAtun** уступает программе **Xmms** практически по всем параметрам. Кроме того, **NoAtun**, работая через звуковой сервер aRts, требует больше памяти, чем **Xmms**, воспроизводящий звук непосредственно через /dev/dsp.

Чтобы запустить **Xmms** сразу после окончания работы с **NoAtun**, остановите демон аудиосервера **artsd**, иначе вы получите сообщение, что звуковая плата занята другим приложением — а именно этим демоном.

5.3. «Ограбление» Audio-CD

Вот мы и подошли к самому интересному моменту в этой главе. Допустим, вы одолжили у кого-то новый компакт-диск и захотели сохранить его содержимое в формате MP3 на своем жестком диске. Под Windows написано множество программ, позволяющих «грабить» музыку с компакт-дисков, самые известные — AudioGrabber, EAC и CDEX. Но не устанавливать же Windows только для того, чтобы «сграбить» пару компактов? Есть программа-аудиограббер и для Linux. Это консольная программа **cdparanoia** (http://www.xiph.org/paranoia) и графическая оболочка для нее **Grip** (http://www.nostatic.org/grip), работающая в среде GNOME.

Запустите программу **Grip**, затем нажмите кнопку «Eject», вставьте компакт-диск и снова нажмите кнопку «Eject».

Вы видите, что вместо названия песен отображаются только надписи Track1..TrackN. Вряд ли нас устраивают такие названия, поэтому нажмите кнопку **Toggle Disc Editor**. Окно программы **Grip** расширится, и вы увидите редактор названий песен. После того, как вы введете названия песен, нажмите кнопку **Save disc info**.

Перед началом «ограбления» компакт-диска перейдите на вкладку **Config** программы **Grip**. Нас интересуют опции на вкладках CD, Rip, MP3, ID3.

На вкладке **CD** можно указать имя и параметры CD-привода. По умолчанию используется устройство /dev/cdrom. Вы же можете указать другое устройство CD-ROM, например /dev/hdd (если CDROM подключен как Slave ко второму контроллеру).

Теперь откройте вкладку **Rip → Options** и отключите надоедающий режим **Auto-eject after rip**. В этом режиме после того, как Grip скопировал на

винчестер содержимое аудиодорожек, компакт-диск автоматически извлекается из привода. На вкладке **MP3 → Options** вы можете установить скорость потока MP3 (по умолчанию используется 128 Кбит/с). Здесь же можно установить количество процессоров, которые будут использованы для сжатия музыки (параметр **Number of CPUs to use**).

Как работает аудиограббер? Сначала он читает дорожку компакт-диска и записывает ее в WAV-файл. Затем он запускает MP3-компрессор и сжимает WAV-файл. После сжатия исходный WAV-файл удаляется. Вы можете отключить параметр **Delete .wav after encoding**, чтобы WAV-файл не удалялся после сжатия, но помните, что WAV-файлы занимают очень много дискового пространства. Например, обыкновенный аудио компакт-диск в формате WAV будет занимать около 650 МБ на жестком диске, а этот же компакт-диск в формате MP3 — всего около 65Мб (при битрейте 128 Кбит/с).

Примечание

Битрейт (bitrate) — количество информации, описывающей одну секунду звука. Опыт показывает, что битрейт 128 Кбит/с для большинства людей достаточен, чтобы считать звучание идеальным. Значение 128 Кбит/с является пороговым значением, выше которого качество не так существенно увеличивается при увеличении битрейта, чем до него. Но тем не менее чем битрейт больше, тем лучше. Сейчас стараются использовать битрейт 160-192 Кбит/с.

Теперь мы готовы приступить к непосредственному преобразованию аудиодорожек в MP3-файлы. Выберите нужные вам дорожки с помощью правой кнопки мыши и перейдите на вкладку **Rip**. Вам доступны два режима: **Rip+Encode** и просто **Rip**. В первом случае программа **Grip** создаст WAV-файлы и преобразует их в формат MP3. Во втором случае преобразование в формат MP3 произведено не будет. Нажмите кнопку **Rip+Encode**. Если вы не выбрали ни одной дорожки, **Grip** спросит вас, хотите ли вы записать сразу весь компакт-диск.

Программа **Grip** работает очень быстро, поскольку используются сразу два потока — один для чтения дорожек CD и записи их в WAV-файл, а другой — для преобразования WAV-файла в формат MP3. Например, вы выбрали две дорожки. Сначала ***Grip*** прочитает первую и сохранит ее на диск в формате WAV. Затем, пока будет читаться вторая дорожка, **Grip** параллельно будет сжимать первый WAV-файл. Качество сжатия тоже вполне приемлемое: песня продолжительностью 3 минуты 30 секунд заняла 3.36 Мб в формате MP3 (при битрейте 128 Кбит/с).

5.4. Программы для просмотра видео

Обзор программ

Как вы знаете, видео может быть записано в форматах AVI, VCD, DVD, MPEG-1, MPEG-2, MPEG-4. Больше всего нас (во всяком случае меня) интересует самый распространенный формат — последний. Своей популярности формат MPEG-4 добился благодаря тому, что он не требует никакой дополнительной аппаратуры, как, например, DVD. В зависимости от дистрибутива у вас могут быть установлены разные программы для просмотра видео. Самые распространенные: **XMovie** (`http://heroines.sourceforge.net/xmovie-1.8.tar.gz`), **Gtv** и **Xine**.

Программа **XMovie** поддерживает видео, записанное в форматах MPEG-1/2, DVD и QuickTime. Программа **Gtv** — поддерживает AVI, MPEG-1/2, VCD. Как видите, эти программы не поддерживают нужный нам формат. Если вам нужен MPEG-4, то вам прямая дорога к Xine

В состав KDE входит простенький видео проигрыватель **aKtion**. Возможности его довольно скудны: он поддерживает только MPEG1, QuickTime, AVI и анимированный GIF. К тому же MPEG полностью не поддерживается — воспроизведение без звука. Почему так слабо? Да потому, что **aKtion** — это всего лишь оболочка для старого проигрывателя **xanim**.

Единственная функция, которая понравилась мне в **aKtion**, — это захват экрана. Для захвата текущего кадра нажмите клавишу <C>. По умолчанию используется формат PNG, но в параметрах программы вы можете установить любой другой формат.

Программа Xine

Программа **Xine** (`http://xinehq.de`) — свободно распространяемый видеопроигрыватель для UNIX-систем. Поддерживает форматы VCD, DVD, MPEG-1/2 и дополнительно MPEG-4, а также другие форматы. Это значит, что программу можно расширить, добавив в нее поддержку нужных форматов. Однако в некоторых дистрибутивах поддержка MPEG-4 добавлена, а в некоторых нет. Например, в дистрибутиве ALT Junior 1.1 я не смог просмотреть фильм в формате MPEG-4 с помощью **Xine**. Пришлось обновить систему. Во второй версии дистрибутива ALT Junior с поддержкой MPEG-4 все было нормально.

Кроме пакета самого проигрывателя `xine-ui`, необходимо скачать и библиотеку `xine-lib` с тем же номером версии, а если смотрите видеоDVD, то установите дополнительно и `xine-vcdx` — плагин, обеспечивающий удобство управления. Пакеты RPM версии 0.99.3 можно взять с `http://people.linux-online.ru/xpdev/bloody/rpms`.

Рис. 5.2. *Проигрыватель Xine*

Когда программа будет установлена, введите команду **xine-check** для проверки конфигурации вашей системы: программа определит используемый вами сервер звука, видеодрайвер и другие параметры.

Программа **Xine** по умолчанию использует видеодрайвер XShm и аудиодрайвер alsa09, но можно их сменить, указав новые драйвера как аргументы опций –V и –A соответственно. Проигрыватель поддерживает аудиосистемы OSS, ALSA, aRts, ESD, Irix и Sun Audio и видеодрайверы Xvideo, XShm, OpenGL, SDL, ASCII Art library, Syncfb и framebuffer.

Если вы установили какой-нибудь драйвер (аудио или видео), а **xine** перестал запускаться, запустите **xine** с параметрами **–A null** и **–V null** или заново введите команду **xine-check**.

Список опций команды **xine** можно посмотреть, как обычно, введя `xine --help`.

Остановлюсь только на опции **–n**, разрешающей управлять проигрывателем по сети.

Для управления по сети создайте файл `~/.xine/passwd` и добавьте в него строку ALL:ALLOW. Затем добавьте строку

```
xinectl 6789/tcp # управление через порт 6789
```

в файл `/etc/services`. Запустите **xine** с параметром **–n**, а на том компьютере, с которого собираетесь управлять, выполните команду `telnet`

`<имя_узла> 6789`. Теперь можно управлять проигрывателем по сети. Введите команду `help` для получения краткой помощи по командам для сетевого управления.

Просмотр DVD

Для просмотра DVD в Linux используется проигрыватель Totem (меню **K → Мультимедиа → Видео → Проигрыватель Totem**, если вы работаете в среде KDE, или **Приложения → Звук и видео → Видеопроигрыватель Totem**, если в GNOME).

Для открытия DVD-диска выберите в меню команду **Фильм → DVD**. Сначала Totem отобразит меню DVD-диска (файл VIDEO_TS.VOB), в котором вы сможете выбрать звуковой поток (язык фильма), титры и др.

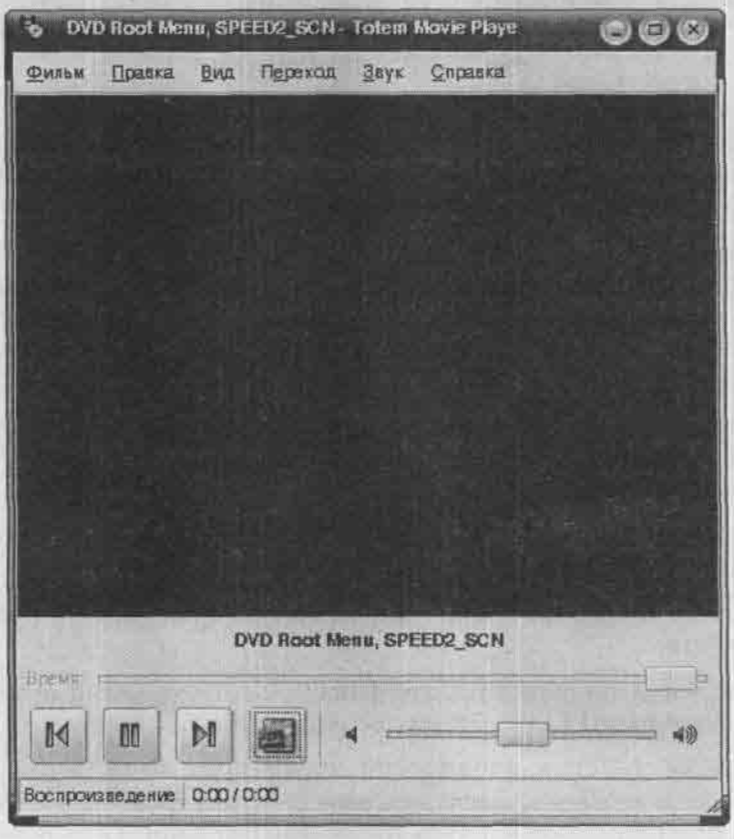


Рис. 5.3. *Totem*

Смотрим телепередачи под Linux

Чтобы смотреть телевизор без отрыва от любимого компьютера, придется купить TV-тюнер. Я рекомендую приобретать тюнеры производства компании AverMedia. Не пожалейте денег на тюнер этой компании. А внешние USB-тюнеры, наоборот, не покупайте: для них еще не создано стабильных драйверов под ОС Linux. К тому же пропускная способность шины USB 1.1 недостаточна для качественного просмотра телепередач.

Следующий момент, который важно учесть при выборе TV-тюнера, это поддерживаемые им стандарты. Телевещание в странах СНГ производится в системе SECAM, а звук передается в системе D/K. Поэтому при покупке тюнера, убедитесь, что он поддерживает оба эти стандарта. Если ваш тюнер поддерживает только PAL, изображение будет черно-белым, а если отсутствует поддержка D/K, то... Я умолчу об этом (не хочется вспоминать издаваемый шум).

Наконец, некоторые тюнеры могут не поддерживаться вашим ядром. Чтобы получить список поддерживаемых плат,

1. установите из вашего дистрибутива пакет kernel-source, содержащий исходные тексты ядра и модулей;
2. загляните в появившийся каталог `/usr/src/linux-2.xx.xx/Documentation/video4linux/bttv`, где 2.xx.xx — версия ядра;
3. читайте README и CARDLIST (в некоторых версиях ядра он называется CARDS).

После подключения TV-тюнер придется настраивать. Здесь вам поможет bttv mini-HOWTO на русском языке по адресу

`http://linux.yaroslavl.ru/docs/howto/mini/BTTV-Mini-HOWTO-0.3`

Облегчить вам работу по настройке тюнера сможет конфигурационная утилита с графическим интерфейсом **GCbttv** (`http://freshmeat.net/projects/gcbttv`).

Наконец, TV-тюнер подключен и настроен, и пора смотреть передачи.

Лучше всего использовать программы, поддерживающие библиотеку **libXaw**. Во многие дистрибутивы включен **XawTV** (домашняя страница `http://linux.bytesex.org/xawtv`) — набор программ для управления видеоустройством `video4linux`, захвата видеопотока в файл в различных форматах или вывода его на экран. Настройки главной программы **xawtv**, собственно просмотрщика TV, хранятся в файле `~/.xawtv`. Создайте вручную файл примерно такого содержания:

```
[global]
fullscreen = 800 x 600
freqtab = europe-east
```

```
pixsize = 128 x 96
pixcols = 1
jpeg-quality = 75

[defaults]
norm = SECAM
capture = over
source = Television
```

Если вы чувствуете, что ваш старенький компьютер не «вытянет» полноэкранного режима 800x600, установите разрешение экрана 640x480. В зависимости от поворотливости вашего компьютера установите уровень JPEG-компрессии. Чем выше качество, тем больше нагрузка на систему. Обычно для максимального качества хватает значения **jpeg-quality = 90**. Но самым оптимальным значением будет все-таки 75.

Затем указываем программе, что мы будем принимать передачи в формате SECAM. Частотная таблица каналов — **europe-east**. Если вы живете в западной Европе, укажите **europe-west**.

Другие программы просмотра телепередач, которые вам могут пригодиться:

- **tvtime** — программа, имитирующая телевизор на компьютере, выводя изображение на экран с частотой 50/60 Гц, снимая тем самым проблему с чересстрочным изображением;
- **GnomeTV** — простая программа для просмотра телепередач в оконной среде GNOME, перестала развиваться в 2001 году;
- **KwinTV** — интегрирована в среду KDE;
- **bttvgrab** — мощная программа, позволяющая записывать телепередачи на диск.

5.5. Воспроизведение неподдерживаемых форматов

Во многих дистрибутивах Linux мы не можем воспроизвести, казалось бы, привычные форматы — DivX, Windows Media (WMV), QuickTime, DVD и в некоторых случаях даже MP3. Причина всему этому — всевозможные лицензионные соглашения, нарушения которых не допускается. Но выход из ситуации есть: распространять тот или иной кодек в составе дистрибутива запрещено, но никто не мешает вам загрузить его и использовать в свое удовольствие.

В большинстве случаев кодеки выполнены в виде динамических библиотек Windows (dll-файлов). Чтобы их использовать в Linux, вам понадобится **mplayer** — это кроссплатформенное приложение, позволяющее

воспроизводить различные форматы мультимедиа. Найти программу **mplayer** в Интернете не проблема. А вот кодеки найти намного сложнее. Я рекомендую загрузить пакет `essential`, который доступен по адресу: `http://www.mplayerhq.hu/homepage/dload.html`.

Установим этот пакет:

```
$ tar -jxvf  essential-20050216.tar.bz2
$ cp  essential-20050216/* /usr/lib/win32/
```

Чтобы кодеки стали доступны, вам нужно перезапустить **mplayer**.

С воспроизведением DVD ситуация тоже неоднозначна. Некоторые дистрибутивы, особенно новые (но не все), воспроизводят DVD без проблем. А проигрыватели других дистрибутивов вообще отказываются работать с DVD. Дело в том, что в состав проигрывателей не включена поддержка DVD. Для ее включения нужно перекомпилировать проигрыватели из исходных кодов. Прочитать обо всем этом можно по адресу

`http://www.geniusweb.com/LDP/HOWTO/html_single/DVD-Playback-HOWTO`.

И еще: если у вас не воспроизводятся DVD, то, возможно, с вашим дистрибутивом все нормально, просто у вас слишком слабый компьютер. Для воспроизведения DVD нужен компьютер не слабее Pentium III 500 Mhz и 256 Мб оперативной памяти. Также убедитесь, что включен DMA (прямой доступ к памяти) для вашего DVD-привода.

Глава 6

СЕТЬ И ИНТЕРНЕТ

- ОСНОВНЫЕ СЕТЕВЫЕ ПОНЯТИЯ
- ПОДКЛЮЧЕНИЕ К ЛОКАЛЬНОЙ СЕТИ
- ПОДКЛЮЧЕНИЕ К WINDOWS-СЕТИ
- ПОДКЛЮЧЕНИЕ К ИНТЕРНЕТУ

6.1. Основные сетевые понятия

6.1.1. Протокол и интерфейс

В основе сети Интернет лежит протокол TCP/IP, поэтому знакомство с понятием протокола необходимо любому пользователю.

Протокол — это совокупность правил, определяющая взаимодействие абонентов вычислительной системы (в нашем случае — сети) и описывающая способ выполнения определенного класса функций. Еще один термин, который мы будем часто употреблять, интерфейс. *Интерфейс* — это средства и правила взаимодействия компонент системы между собой. Чтобы лучше понять значения этих терминов, обратите внимание на рис. 6.1. На этом рисунке изображены две системы (компьютера) — А и В.

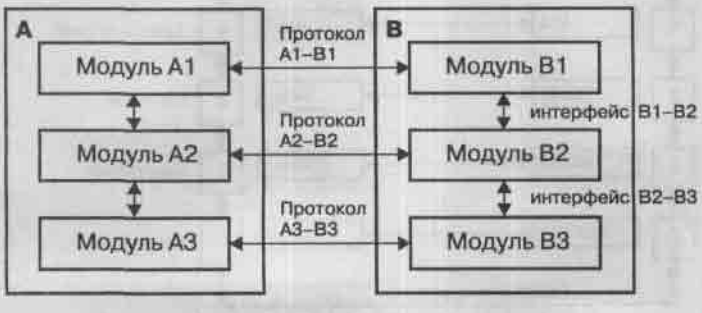


Рис. 6.1. *Протоколы и интерфейсы*

Таким образом, средства, которые обеспечивают взаимодействие модулей разных уровней в рамках одной системы, называются интерфейсом, а средства, обеспечивающие взаимодействие компонент одного уровня разных систем, — протоколом.

Прежде чем перейти к протоколу TCP/IP, рассмотрим семиуровневую модель взаимодействия открытых систем (сокращенное название — модель OSI), разработанную в начале 80-х годов международной организацией по стандартизации ISO.

Средства взаимодействия (см. рис. 6.2) в модели OSI делятся на семь уровней:

1. Физический.
2. Канальный.
3. Сетевой.
4. Транспортный.
5. Сеансовый.
6. Представительный.
7. Прикладной.

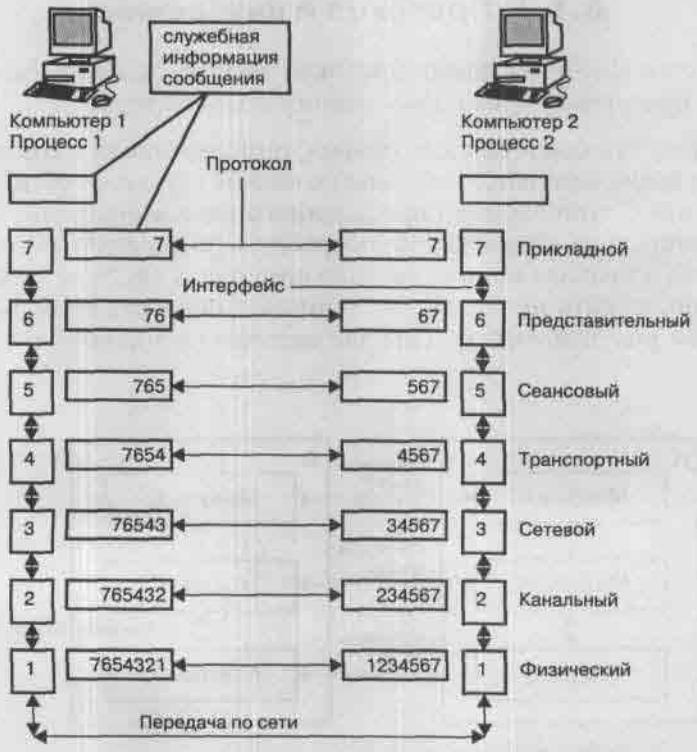


Рис. 6.2. *Модель OSI*

Благодаря этому задача сетевого взаимодействия разбивается на несколько более мелких задач. Это позволяет при разработке новых способов и инструментов сетевого взаимодействия не создавать их с нуля, а использовать уже готовые решения. Непосредственно друг с другом взаимодействуют только физические уровни. Все остальные уровни напрямую взаимодействуют только с выше- и нижележащими уровнями: пользуются услугами нижележащего и предоставляют услуги вышележащему. Друг с другом такие уровни контактируют косвенным образом, через посредство нижележащих уровней.

Примечание

В некоторых случаях сетевого взаимодействия физический уровень как таковой отсутствует, при этом его функции выполняет самый низкий из присутствующих уровень.

Из рис. 6.2 видно, что по мере прохождения сообщения через уровни модели OSI к пересылаемым данным добавляется служебная информация, свидетельствующая о прохождении данных через определенный уровень.

Рассмотрим взаимодействие двух компьютеров более подробно на примере файловой службы. Допустим, нам (компьютер 1) нужно записать какую-нибудь информацию в файл на удаленном компьютере 2. Обычное сообщение состоит из заголовка и поля данных. В заголовке содержится различная служебная информация: например, адреса нашего компьютера и компьютера-получателя, имя и расположение файла. В поле данных в нашем случае находится содержимое файла.

Приложение (процесс 1) формирует стандартное сообщение, которое передается прикладному уровню. Точнее, процесс 1 работает на прикладном уровне.

После формирования сообщения прикладной уровень передает его представительному уровню. На этом уровне в заголовок добавляются указания для представительного уровня компьютера-адресата. Потом сообщение передается сеансовому уровню, который добавляет свою информацию и т.д. Процесс вложения одного протокола в другой называется *инкапсуляцией*.

Когда сообщение поступает на компьютер-адресат, оно принимается физическим уровнем и передается вверх с уровня на уровень. Каждый уровень анализирует содержимое заголовка своего уровня, выполняет содержащиеся в нем указания, затем удаляет относящуюся к себе информацию из заголовка и передает сообщение далее вышележащему уровню. Этот процесс называется *декапсуляцией*. Далее мы рассмотрим каждый из уровней взаимодействия отдельно.

6.1.2. Уровни взаимодействия OSI

Физический уровень (Physical Layer)

Физический уровень передает биты по физическим каналам связи, например, коаксиальному кабелю или витой паре. На этом уровне определяются характеристики электрических сигналов, которые передают дискретную информацию, например: тип кодирования, скорость передачи сигналов. К этому уровню также относятся характеристики физических сред передачи данных: полоса пропускания, волновое сопротивление, помехозащищенность.

Функции физического уровня реализуются сетевым адаптером или последовательным портом. Примером протокола физического уровня может послужить спецификация 100Base-TX (технология Ethernet).

Канальный уровень (Data link Layer)

Канальный уровень отвечает за передачу данных между узлами в рамках одной локальной сети. Узлом будем считать любое устройство, подключенное к сети.

Этот уровень выполняет адресацию по физическим адресам (MAC-адресам), «вшитым» в сетевые адаптеры предприятием-изготовителем. Каждый сетевой адаптер имеет свой уникальный MAC-адрес.

Канальный уровень переводит поступившую с верхнего уровня информацию в биты, которые потом будут переданы физическим уровнем по сети. Он разбивает пересылаемую информацию на фрагменты данных — кадры (*frames*).

На этом уровне открытые системы обмениваются именно кадрами. Процесс пересылки выглядит примерно так: канальный уровень отправляет кадр физическому уровню, который отправляет кадр в сеть. Этот кадр получает каждый узел сети и проверяет, соответствует ли адрес пункта назначения адресу данного узла. Если адреса совпадают, канальный уровень принимает кадр и передает наверх вышележащим уровням. Если же адреса не совпадают, то он просто игнорирует кадр.

В используемых в локальных сетях протоколах канального уровня заложена определенная топология, то есть способ организации и адресации физических связей. Канальный уровень обеспечивает доставку данных между узлами в сети с определенной топологией, а именно той, для которой он разработан. Основные топологии показаны на рисунке 6.3.

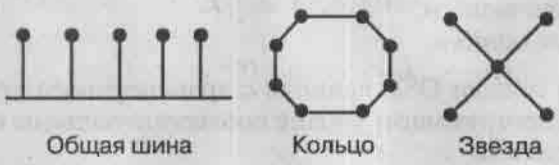


Рис. 6.3. *Основные топологии локальных компьютерных сетей*

Сетевой уровень (Network Layer)

Канальный уровень обеспечивает связь только между компьютерами, соединенными индивидуальной линией связи, то есть в рамках локальной сети. Межсетевое взаимодействие обеспечивает сетевой уровень.

Локальные сети соединяются специальными устройствами — *маршрутизаторами*. Маршрутизатор собирает информацию о топологии межсетевых соединений и на основании этой информации пересылает сообщения сетевого уровня в сеть назначения. Сообщения на сетевом уровне называются *пакетами*. Чтобы передать пакет от компьютера-отправителя компьютеру-адресату, который находится в другой локальной сети, нужно совершить некоторое количество *транзитных передач между сетями*. Иногда их еще называют хопами (от англ. hop — прыжок). При этом каждый раз выбирается подходящий маршрут.

На сетевом уровне работают несколько видов протоколов: сетевые протоколы, которые обеспечивают передвижение пакетов по сети; протоколы маршрутизации RIP и OSPF; протоколы разрешения адреса ARP (*Address Resolution Protocol*).

Классические примеры протоколов сетевого уровня: IP (стек TCP/IP), IPX (стек Novell).

Транспортный уровень (Transport Layer)

На пути от отправителя к получателю пакеты могут быть искажены или утеряны. Некоторые приложения самостоятельно выполняют обработку ошибок при передаче данных, но большинство все же предпочитают иметь дело с надежным соединением, которое как раз и призван обеспечить транспортный уровень. Этот уровень обеспечивает требуемую приложению или верхнему уровню (сеансовому или прикладному) надежность доставки пакетов. На транспортном уровне определены пять классов сервиса:

1. Срочность.
2. Восстановление прерванной связи.
3. Наличие средств мультиплексирования нескольких соединений.

4. Обнаружение ошибок.
5. Исправление ошибок.

Обычно уровни модели OSI, начиная с транспортного уровня и выше, реализуются на программном уровне соответствующими компонентами операционных систем.

Примеры протоколов транспортного уровня: TCP и UDP (стек TCP/IP), SPX (стек Novell).

Сеансовый уровень (Session Layer)

Сеансовый уровень устанавливает и разрывает соединения между компьютерами, управляет диалогом между ними, а также предоставляет средства синхронизации. Средства синхронизации позволяют вставлять определенную контрольную информацию в длинные передачи (точки), чтобы в случае обрыва связи можно было вернуться назад (к последней точке) и продолжить передачу.

Сеанс — это логическое соединение между компьютерами. Каждый сеанс имеет три фазы:

1. Установление соединения. Здесь узлы «договариваются» между собой о протоколах и параметрах связи.
2. Передача информации.
3. Разрыв связи.

Не нужно путать сеанс сетевого уровня с сеансом связи. Пользователь может установить соединение с Интернетом, но не устанавливать ни с кем логического соединения, то есть не принимать и не передавать данные.

Представительный уровень (Presentation Layer)

Представительный уровень изменяет форму передаваемой информации, но не изменяет ее содержания. Например, средствами этого уровня может быть выполнено преобразование информации из одной кодировки в другую, шифрование и дешифрование данных.

Пример протокола представительного уровня: SSL (*Secure Socket Layer*), обеспечивающий секретный обмен данными.

Прикладной уровень (Application Layer)

Это набор разнообразных протоколов, с помощью которых пользователи сети получают доступ к совместно используемым ресурсам. Единица данных называется сообщением.

Примеры протоколов: HTTP, FTP, TFTP, SMTP, POP, SMB, NFS.

Интернет и модель OSI

При взаимодействии открытой системы и Интернета модель OSI упрощается, так как некоторые протоколы Интернета включают в себя функции нескольких уровней. Если к сети Интернет подключается один пользователь, а не вся сеть, то автоматически исчезают канальный и физический уровни, потому что нет сетевых адаптеров, а значит, нет и физических адресов. В данном случае конечным протоколом будет протокол типа «точка-точка», например, PPP. В этот протокол будут вложены все остальные.

Основные протоколы

Среди сетевых протоколов выделяются следующие основные:

- **TCP/IP** (*Transmission Control Protocol / Internet Protocol*) — святыня всех святынь. Это базовый транспортный сетевой протокол. На этом протоколе основана вся сеть Интернет.
- **RIP** (*Routing Information Protocol*) используется для маршрутизации пакетов в компьютерных сетях.
- **ICMP** (*Internet Control Message Protocol*) — протокол межсетевых управляющих сообщений. Он применяется для проверки доступности узла, установления соединения и т.п.
- **FTP** (*File Transfer Protocol*) — протокол передачи файлов. Служит для обмена файлами между системами. Например, вам нужно передать файл на сервер или, наоборот, скачать файл с сервера. Для этого вам нужно подключиться к файловому серверу (он же FTP-сервер) и выполнить необходимую вам операцию. Подключение осуществляется с помощью FTP-клиента. Простейший FTP-клиент входит в состав практически любой операционной системы. Обычно для запуска FTP-клиента нужно ввести команду ftp.
- **HTTP** (*Hyper Text Transfer Protocol*) — протокол обмена гипертекстовой информацией, то есть документами HTML. Протокол HTTP используется веб-серверами. HTTP-клиенты называются браузерами.
- **POP** (*Post Office Protocol*) — протокол почтового отделения. Этот протокол используется для получения электронной почты с почтовых серверов. А для передачи почтовых сообщений на сервер служит протокол **SMTP** (*Simple Mail Transfer Protocol*).
- **SLIP** (*Serial Line Internet Protocol*) — протокол подключения к сети Интернет по последовательной линии. Используется для установления связи с удаленными узлами через низкоскоростные последовательные интерфейсы. В настоящее время вытеснен протоколом PPP и практически не используется.
- **PPP** (*Point-to-Point Protocol*) обеспечивает управление конфигурацией, обнаружение ошибок и повышенную безопасность при передаче данных на более высоком уровне, чем протокол SLIP.

6.1.3. Протокол TCP/IP и IP-адресация

Любому компьютеру в IP-сети (TCP/IP-сети) назначен уникальный адрес, который называется IP-адресом. IP-адрес — это 32-разрядное двоичное число, которое принято записывать в виде четырех десятичных чисел, разделенных точками, например, 111.111.213.232 или 127.0.0.1.

Уникальность IP-адреса достигается достаточно просто: IP-адреса назначаются централизованно Сетевым Информационным Центром (NIC, Network Information Center). Если ваша локальная (или даже региональная) сеть не соединена с Интернет, то внутри сети вы можете использовать любые IP-адреса без согласования с NIC. Если же ваша сеть подключена к Интернету, протокол TCP/IP обеспечивает работу вашей сетевой программы с любым компьютером в мире, как будто тот находится в локальной сети.

Любая сеть, в том числе локальная, может быть разделена на подсети. О причинах и целях такого разбиения вы можете прочитать в руководстве IP Sub-networking-HOWTO, которое вы найдете на сайте `http://dkws.narod.ru` или на `http://www.dhsilabs.com.ua`. Подсети связывает маршрутизатор, в роли которого может выступать любой компьютер с двумя или более сетевыми интерфейсами, например, двумя сетевыми платами.

Каждая сеть (подсеть) также имеет свой уникальный адрес. Под сетью можно понимать «пачку» IP-адресов, идущих подряд, то есть 192.168.1.0 — 192.168.1.255. Самый младший и самый старший адреса резервируются. Младший (192.168.1.0) служит адресом сети и используется, когда нужно указать всю сеть (подсеть), например, при задании маршрутизации для нее. Старший служит широковещательным (*broadcast*) адресом: в этот адрес направляются сообщения, которые нужно передать сразу всем компьютерам сети. Широковещательные запросы очень часто используются, например, для построения ARP-таблиц.

Для каждой сети (подсети) определена ее маска. Фактически, маска — это размер сети, то есть число адресов в сети. Маску принято записывать в десятично-побайтном виде:

- 255.255.255.0 — маска на 256 адресов (0 — 255)
- 255.255.255.192 — маска на 64 адреса (192 — 255)
- 255.255.0.0 — маска на 65536 адресов (256*256)

IP-сети делятся по размеру на классы, каждому из которых соответствует свой диапазон адресов. Запишите первое число адреса в виде восьмизначного двоичного числа, и количество идущих подряд единиц укажет на класс сети. В таблице 6.1 приведены характеристики классов сетей.

Классы IP-сетей *Таблица 6.1*

Класс	Первые биты	Диапазон адресов	Количество узлов	Маска
A	0	1.0.0.0 — 126.0.0.0	16777216 (2^{24})	255.0.0.0
B	10	128.0.0.0 — 191.255.0.0	65536 (216)	255.255.0.0
C	110	192.0.1.0 — 223.255.255.0	256 (2^8)	255.255.255.0
D	1110	224.0.0.0 — 239.255.255.255	Multicast	
E	11110	240.0.0.0 — 247.255.255.255	Зарезервирован	

Адреса узлов (компьютеров) в сети класса A выглядят как 125.*.*.*, класса B — как 136.12.*.*, класса C — как 195.136.12.*.

Если адрес начинается с последовательности 1110, то сеть является сетью класса D, а сам адрес является особым — групповым (*multicast*). Если в пакете указан адрес сети класса D, то этот пакет должны получить все хосты, которым присвоен данный адрес.

Адреса класса E зарезервированы для будущего применения.

За некоторыми адресами закреплены особые значения, приведенные в таблице 6.2.

Специальные IP- адреса *Таблица 6.2*

Адрес	Назначение
0.0.0.0	Адрес узла, сгенерировавшего этот пакет
255.255.255.255	Широковещательный адрес (ограниченное широковещание). Пакет с таким адресом будет рассылаться всем узлам, которые аходятся в той же сети, что и источник пакета
127.0.0.1	Loopback — адрес локального компьютера. Используется для тестирования сетевых программ и взаимодействия сетевых процессов. При попытке отправить пакет по этому адресу данные не передаются по сети, а возвращаются протоколам верхних уровней как только что принятые. Любой адрес подсети 127.0.0.0 относится к локальному компьютеру
10.0.0.0	Изолированная сеть класса A, которая использует протокол IP, но не подключена к Интернету. Маска сети 255.0.0.0
172.16.0.0 — 172.31.0.0	16 изолированных IP-сетей класса B. Маска каждой сети 255.255.0.0
192.168.0.0 — 192.168.255.0	256 сетей класса C, маска каждой сети 255.255.255.0

6.1.4. DNS — система доменных имен

Человеку обычно легче запомнить символьное имя (www.dhsilabs.com.ua), чем последовательность чисел (217.20.163.34). Компьютеру же, наоборот, проще обрабатывать числа, а не символьную информацию. Для преоб-

разования IP-адреса в символьное имя и обратно используется служба доменных имен — DNS (*Domain Name System*).

Домены объединены в иерархическую структуру. Корневой домен управляется центром InterNIC, который назначает домены верхнего (первого) уровня для каждой страны и регистрирует национальных координаторов. Национальные координаторы (в России это — RU-CENTER, `http://www.nic.ru`) повторяют эту процедуру в своем домене и так далее, в результате типичное доменное имя подразделения компании выглядит как на рисунке 6.4.

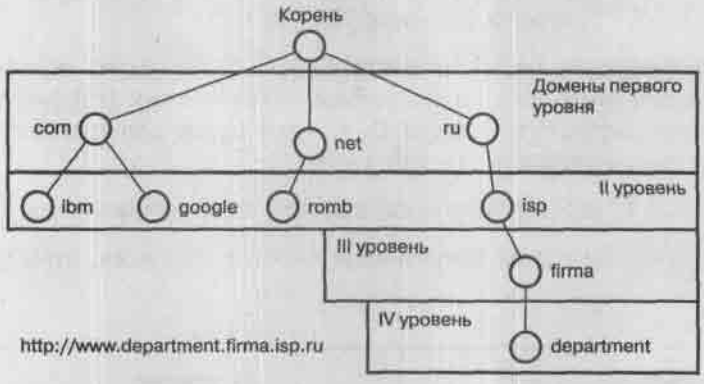


Рис. 6.4. *Иерархическая структура системы доменных имен*

DNS можно назвать гигантской распределенной базой данных. Ее поддерживают серверы имен (*name server, DNS-сервер*), которые снабжают всех информацией о данном домене или нескольких доменах сразу. Для каждой зоны (группы узлов, приписанных к этому домену, но не к его поддоменам) есть по крайней мере два сервера имен, которые содержат всю информацию относительно хостов (узлов) в этой зоне.

Что происходит, когда пользователь вводит в окне браузера адрес department.firma.isp.ru? Запрос на разрешение (преобразование) имени в IP-адрес сначала отправляется серверу имен, принадлежащему провайдеру пользователя. Если этот сервер знает такое имя, он возвращает IP-адрес, и браузер устанавливает соединение с нужным компьютером. Если же нет, то сервер имен провайдера обращается к корневому серверу, обслуживающему домен наивысшего уровня, тот перенаправляет запрос DNS-серверу домена ru, тот обращается к DNS-серверу домена isp, тот — к DNS-серверу домена firma, а этот последний возвращает IP-адрес зарегистрированного в нем хоста department. Если эта цепочка запросов оборвется на каком-либо звене, то пользователю будет сообщено о невозможности разрешения имени компьютера в IP-адрес.

6.1.5. Порты

На одном компьютере могут одновременно работать несколько приложений, обменивающихся данными через сеть. Если в качестве адресата сообщения указывать только IP-адрес получателя, то приложения не смогут разобраться, которому из них предназначены присланные данные. Для решения этой проблемы используется механизм портов. Номер порта — это просто номер программы, которая будет обрабатывать переданные данные.

Этот номер может быть любым, но за наиболее популярными службами закреплены стандартные номера, чтобы клиенты были твердо уверены, что обращаются к нужному серверу. Так, стандартный номер порта электронной почты 25, пересылки файлов — 21, веб-сервера — 80, службы telnet — 23, сервера имен — 53 . Стандартные номера выбираются из промежутка от 0 до 1023. Числа начиная с 1024 и до 65 535 можно использовать для своих собственных номеров портов.

6.1.6. Динамическое выделение адреса

Как я уже сказал, IP-адрес любого устройства, подключенного к Интернету, должен быть уникальным. Это означает, что статически назначенный вам адрес не сможет использовать никто другой, даже тогда, когда вы отключитесь от сети. Избавиться от такого расточительного расходования адресов помогает механизм DHCP (*Dynamic Host Configuration Protocol*), позволяющий множеству временно подключающихся к сети клиентов совместно использовать один и тот же небольшой набор адресов.

Если в сети присутствует DHCP-сервер, то клиент, подключаясь к сети, может получить у него временный IP-адрес, по которому и будет доступен всем пользователям Интернет на протяжении всего сеанса подключения. По завершении сеанса адрес освобождается и может быть выделен кому-то другому.

6.2. Подключение к локальной сети

Если среди вашего оборудования есть сетевая плата, то она будет обнаружена при установке системы. Если же вы установили сетевую плату после установки системы, то запустите средство поиска нового оборудования. Обычно для этого применяется утилита **kudzu** (в дистрибутиве Linux Mandrake 10 есть своя — **harddrake**). Запускать ее нужно от имени суперпользователя.

Kudzu самостоятельно определит новое устройство и установит его: пропишет устройство в файл установленного оборудования (имя его зависит

от дистрибутива: попробуйте `/etc/modules.conf`, `/etc/modprobe.conf`, `/etc/sysconfig/hwconf`) и добавит его модуль (драйвер) в состав ядра.

Нужно напомнить, что `eth0` — это первая сетевая плата, `eth1` — вторая и т.д. Скорее всего, у вас есть всего одна — `eth0`. Ее мы и будем настраивать.

В большинство современных дистрибутивов включены графические программы-конфигураторы, позволяющие настроить всю систему. Первые конфигураторы были не очень удобными, и многие администраторы предпочитали редактировать конфигурационные файлы вручную, тем более что ни один конфигуратор не позволял настроить ту или иную часть системы полностью, а выполнял лишь базовую настройку.

Сейчас же все в корне изменилось: конфигураторы Linux стали такими ми же удобными, как и апплеты Панели управления Windows. Тем не менее, мы рассмотрим не только настройку с помощью конфигураторов, но те самые команды, с помощью которых администраторы настраивали сеть лет десять назад. В этой книге будем ориентироваться на современные дистрибутивы Linux Mandrake и Red Hat Linux (Fedora Core) — совсем старые их версии рассматривать нет смысла — я не встречал ни одного сервера, на котором был бы установлен Red Hat версии ниже 7.x.

6.2.1. Настройка сети в Linux Mandrake

Начнем с моего любимого дистрибутива — Linux Mandrake. Запустите конфигуратор **drakconf**, в нем выберите апплет **«Сеть и Интернет»**, далее **«Новое соединение»**, а в появившемся окне — **«Соединение по локальной сети»**.

Если у вас в сети есть DHCP-сервер, рекомендую выбрать автоматическую настройку. Если же такого сервера нет или у вас другие планы относительно этого компьютера, выбираем ручную настройку.

Кроме IP-адреса сетевого интерфейса и маски сети обязательно укажите опцию «Запускать при запуске», в противном случае вам придется поднимать интерфейс eth0 каждый раз при запуске/перезапуске Linux. Опция **Network Hotplugging** в официальной документации не описана, но, насколько я понял, она используется для определения физического подключения к сети. В режиме **Network Hotplugging**, если ваш компьютер не подключен физически (с помощью кабеля) к хабу или другому компьютеру, то при попытке обращения к интерфейсу вы получите сообщение «Сеть недоступна». Поэтому, если физической сети у вас нет, а сеть как таковая вам нужна, например, для тестирования сетевых приложений, рекомендуется этот режим не включать.

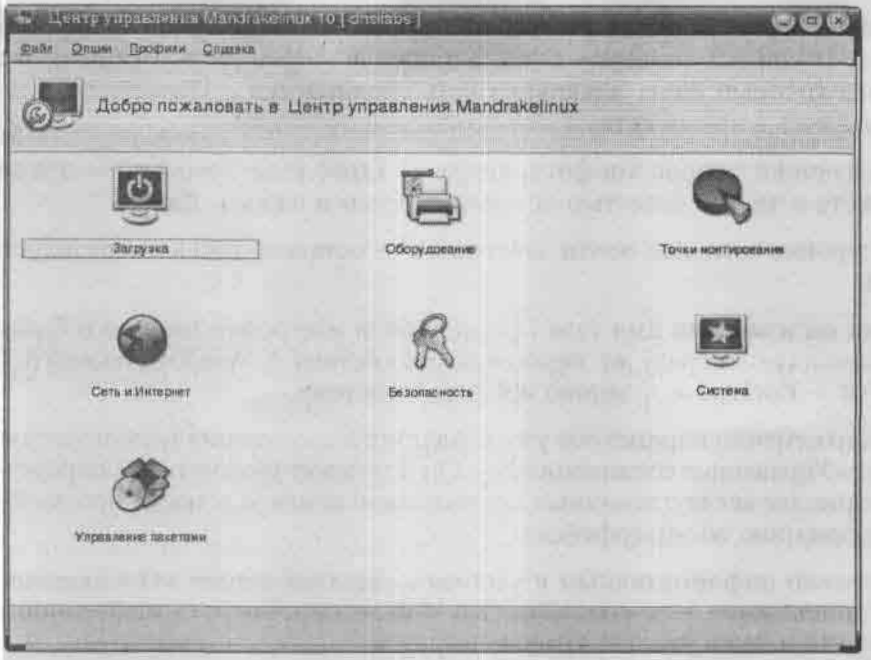


Рис. 6.5. *Конфигуратор DrakConf*

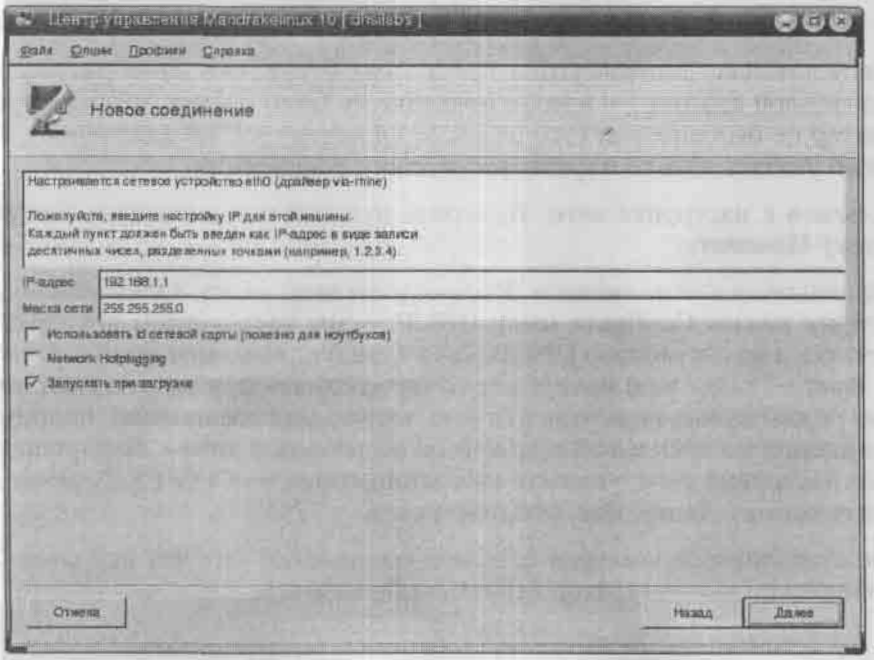


Рис. 6.6. *Указание параметров сетевого интерфейса при ручной настройке*

После этого вам нужно указать общие сетевые параметры: имя компьютера, IP-адреса серверов DNS, а также IP-адрес шлюза (компьютера, через который ваша локальная сеть соединяется с Интернетом), если такой есть в вашей сети.

Следующий вопрос конфигуратора — адрес узла Zeroconf — это поле можете с чистой совестью оставить пустым и нажать **Далее**.

Настройка сети уже почти завершена — осталось только перезапустить сеть.

Если вы изменяли имя узла (а при первой настройке так оно и бывает), рекомендуется сразу же перезапустить систему X Window: нажмите Ctrl + Alt + Backspace и заново войдите в систему.

Для изменения параметров уже созданного соединения используется апплет «Управление соединениями». Он позволяет изменить все параметры, которые вы ввели с помощью предыдущего апплета, а также просмотреть информацию об интерфейсах.

Довольно информативным и удобным оказался апплет «Наблюдение за соединениями» — с его помощью можно просмотреть информацию о трафике и даже увидеть график загрузки.

6.2.2. Настройка сети в Linux Red Hat

При установке дистрибутива программа установки сама распознает ваш сетевой адаптер — в моей практике не было случая, чтобы сетевой адаптер не был опознан (это не касается внешних USB-адаптеров). Вам нужно указать только параметры сетевого соединения.

Вернемся к настройке сети. Выберите нужный вам адаптер и нажмите кнопку **Изменить**.

В появившемся окне введите IP-адрес и сетевую маску. Обязательно выключите режим **Configure using DHCP** — мы настраиваем интерфейсы вручную, а не с помощью DHCP. Также следует включить режим **Activate On Boot** — тогда наш интерфейс будет «поднят» при загрузке системы. Этот режим нужно включать для всех постоянных соединений, например, соединения по локальной сети или по выделенной линии. Вернувшись в окно настройки сети, укажите имя компьютера — COMP5. А затем нажмите кнопку **Далее**. Все, сеть настроена.

Для изменения параметров сетевого соединения удобнее использовать графический конфигуратор **redhat-config-network**.

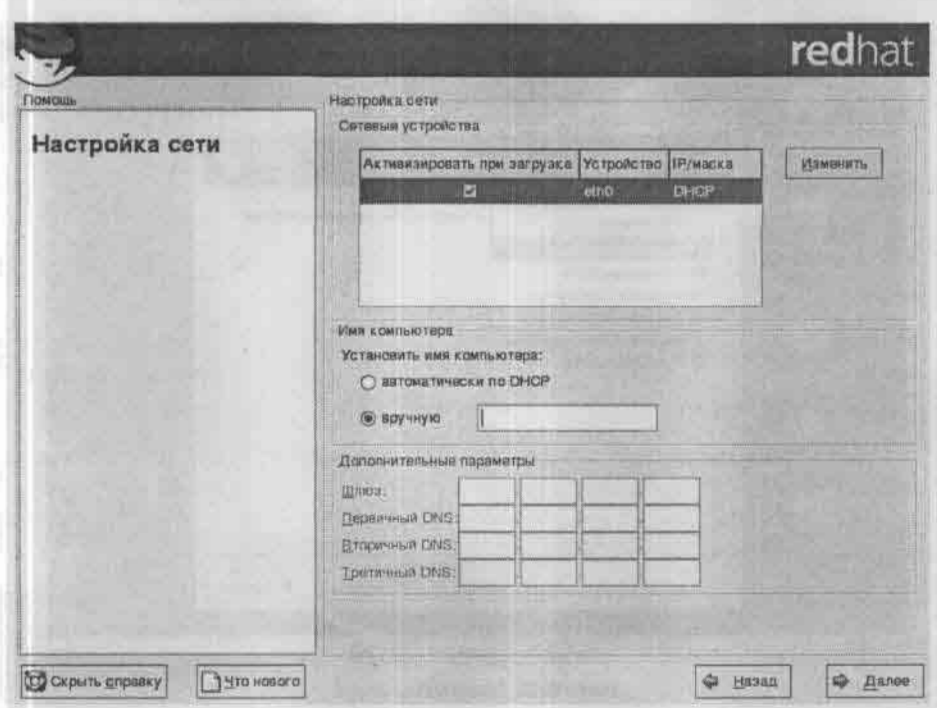


Рис. 6.7. *Настройка сети*

6.2.3. Настройка сети в Fedora Core

Графический конфигуратор в Fedora Core называется **internet-druid** (рис. 6.8). Можно запустить его и из меню оконной среды (в GNOME это **Система → Мастер подключения к Интернет**).

Выберите «**Соединение Ethernet**», а дальше следуйте инструкциям. После создания соединения для вас будет запущен конфигуратор **system-config-network**, позволяющий указать дополнительные параметры сетевого интерфейса или изменить только что введенные. В дальнейшем его можно вызвать из меню оконной среды (в GNOME это **Система → Управление устройствами сети**).

6.2.4. Проверка работы сетевого интерфейса

Если вы не подняли (активировали) интерфейс в процессе графического конфигурирования, сделайте это сейчас. Перейдите на текстовую консоль или откройте окно терминала и выполните команду `ifup eth0` (деактивировать интерфейс можно командой `ifdown eth0`).

Рис. 6.8. *Настройка сети*

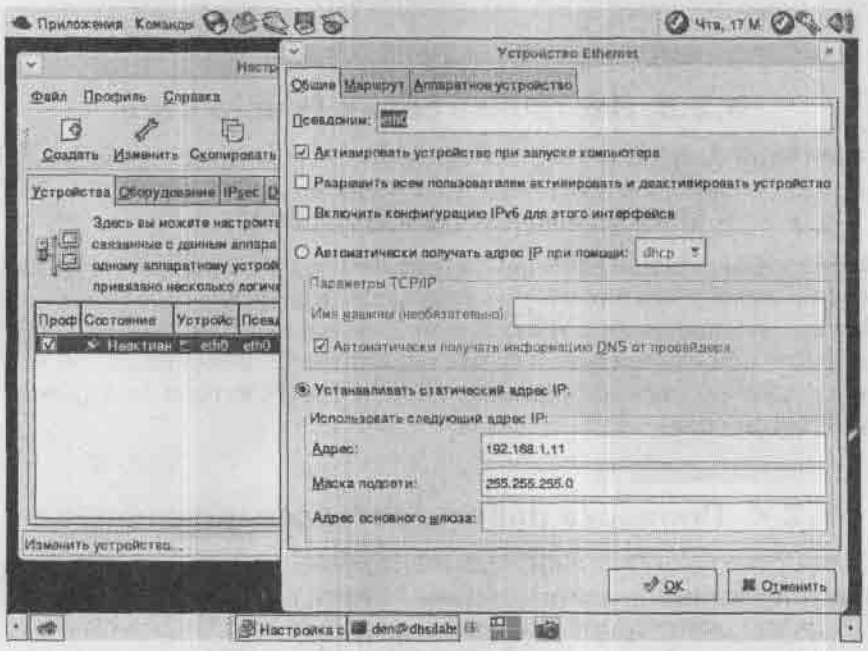


Рис. 6.9. *Указание параметров сетевого интерфейса при ручной настройке*

Для получения сведений об активных интерфейсах выполните команду **ifconfig**. Она покажет примерно следующее:

```
eth0  Link encap:Ethernet  HWaddr 00:02:F0:73:B0:85
inet addr:192.168.1.11 Bcast:192.168.1.255
Mask:255.255.255.0
UP BROADCAST MULTICAST  MTU:1500  Metric:1
[...]

lo Link encap:Local Loopback
inet addr:127.0.0.1  Mask:255.0.0.0
UP LOOPBACK RUNNING  MTU:16436  Metric:1
[...]
```

Интерфейс `lo`, которого вы не настраивали, — это интерфейс обратной петли. Не отключайте его, он необходим для работы некоторых приложений.

В первых двух строчках утилита **ifconfig** выводит тип (Ethernet) адаптера, его физический адрес (MAC-адрес) и присвоенный ему IP-адрес. Дальше — параметры интерфейса, указывающие, что он запущен и используется.

MTU (*Maximum Transfer Unit*) — максимальный размер единицы передачи данных. Практически все протоколы позволяют использовать в кадре поля переменной длины, это касается даже заголовка кадра. Максимально допустимое значение длины поля — это как раз и есть MTU.

Далее следует статистика — сколько пакетов принято/передано, сколько байтов принято/передано, сколько коллизий было с участием этого интерфейса.

Теперь проверим, как работает соединение. Это делают командой **ping** (пингуют нужный адрес).

```
# ping 192.168.1.11
```

Эта команда посылает на указанный адрес по протоколу ICMP маленький пакет, требующий эхо-ответа, раз за разом, пока не будет остановлена (например, нажатием комбинации клавиш Ctrl +C). Обычно ею пользуются для проверки доступности узлов.

Потом пропингуйте свою машину по имени, которое вы ей дали: `ping dhsilabs`.

Убедившись, что проблем с локальными настройками не возникает, можно пропинговать какую-нибудь удаленную машину из вашей локальной сети по ее IP-адресу.

Теперь попробуйте обратиться к удаленной машине по имени. Помните, что символьное имя должно быть разрешено в IP-адрес? В вашей не-

большой сети сервера имен, скорее всего, нет. В этом случае для преобразования IP-адресов в имена и обратно служит файл /etc/hosts. Это обычный текстовый файл, каждая строка которого содержит

```
<IP-адрес> <полное_имя_узла> <псевдонимы>
```

разделенные пробелами. Откройте этот файл в любом текстовом редакторе и добавьте туда сведения о машинах своей локальной сети. Символ # в этом файле понимается как знак комментария.

6.2.5. Настройка сети в старых дистрибутивах

Если вам недоступны графические конфигураторы, то можно настроить сетевой интерфейс и из командной строки.

Добавьте в файл установленного оборудования (скорее всего, он будет называться /etc/conf.modules) сведения о своей сетевой плате.

Добавьте в ядро модуль сетевой платы:

```
# insmod rtl8139.o (для Realtek 8139)
# insmod ne2k-pci.o (для NE2000 PCI)
```

Назначьте интерфейсу IP-адрес:

```
# ifconfig eth0 <IP_адрес > broadcast <BROADCAST_адрес>
netmask <маска>
```

Указать шлюз для выхода в Интернет можно командой route:

```
#route add default gateway <IP_адрес_шлюза>
```

Если в Интернет выходить вы пока не собираетесь, то удалить установленный маршрут можно командой route del default. Команда route без аргументов выводит текущую таблицу маршрутизации пакетов.

Перезапустите сетевую службу, и можно пинговать только что настроенный интерфейс.

6.3. Подключение к Windows-сети

Вам удалось достучаться (**ping**) до всех компьютеров своей локальной сети, работающих под управлением ОС Windows, но хотелось бы большего? Например, обмениваться с ними файлами? Если в вашей сети есть папки, к которым открыт общий сетевой доступ, то это возможно. Поможет вам пакет Samba.

Название Samba происходит от названия протокола SMB (*Server Message Block*), он же NetBIOS, на котором основана работа Windows-сети. Пакет

Samba содержит набор приложений, позволяющих Linux-машине включиться в работу такой сети — как в роли клиента, так и в роли сервера.

В роли обычного пользователя вам достаточно уметь подключаться к Windows-сети как клиенту. Для этого вам понадобятся пакеты samba-common и samba-client, входящие в большинство современных дистрибутивов. Если их у вас нет, то загляните на `http://www.samba.org`.

Пусть сетевой доступ к папке открыт на компьютере, имя которого в сети Windows USER5. Посмотрим сначала на список всех доступных ресурсов на этом компьютере:

```
$ smbclient -L user5 -U < имя_пользователя >
```

Имя пользователя — это не ваше регистрационное имя на Linux-машине, а имя того пользователя машины Windows, кто имеет доступ к ее ресурсам. После ввода пароля вы увидите что-то вроде:

```
Domain=[USER5] OS=[Windows 5.1] Server=[Windows 2000 LAN
Manager]
 Sharename   Type    Comment
 ---------   ----    -------
 SharedDocs  Disk
 WIN (C)     Disk
 ADMIN$      Disk    Remote Admin
 Public      Disk
[...]
```

Public — это та папка, которая вас интересует. Для приема-передачи файлов предназначена программа **smbclient**. Чтобы ускорить (или автоматизировать) ввод пароля, введите ее в таком виде:

```
$ smbclient //user5/public -U < имя_пользователя >%< пароль >
```

Вы увидите приглашение программы:

```
smb: >
```

и сможете вводить ее команды. Первым делом введите команду help для получения списка всех доступных команд. В таблице 6.3 перечислено несколько самых полезных из них.

Использовать программу **smbclient** не очень удобно. Если вы собираетесь обмениваться файлами часто, то лучше использовать программу **smbmount**, которая умеет монтировать удаленный общий ресурс как обычную файловую систему. Ниже приведен пример команды, которая монтирует папку `customers` компьютера USER5, используя имя пользователя den. Точка монтирования — каталог `/mnt/customers`:

```
$ smbmount //user5/customers -U den /mnt/customers -U 500 -G 100
```

Команды программы smbclient *Таблица 6.3*

Команда	Описание
ls	Выводит список файлов в папке
cd [папка]	Выполняет переход в заданный каталог на сервере (учтите, что именно на сервере, а не на клиентском компьютере). В том случае, если каталог не указан, то smbclient просто выдаст имя текущего каталога
get [файл] [локальное имя]	Получает указанный файл из общего ресурса и сохраняет его на локальном компьютере. Если указано локальное имя, то полученный с сервера файл будет сохранен на клиентском компьютере под этим именем
put [файл] [удаленное имя]	Копирует файл на сервер и сохраняет его там под указанным именем. Если это имя не указано, то файл при сохранении переименовываться не будет
mget [файлы]	Получает все указанные файлы с сервера
mput [файлы]	Копирует все указанные файлы на сервер
del [файлы]	Удаляет на сервере указанные файлы, если, конечно, пользователь обладает на это правами
!	Позволяет временно выйти из smbclient, чтобы выполнить команду на локальном компьютере. Например, ! ls — это просмотр текущего каталога на локальной машине
exit или quit	Завершение сеанса работы программы smbclient

В этом примере владельцем смонтированного каталога объявляется пользователь с идентификатором (UID) 500 и идентификатором группы 100.

Закончив работу с общей папкой, размонтируйте ее командой

```
$ smbumount /mnt/customers
```

Если вместо русских имен файлов вы видите непонятные символы, значит, кодировки кириллицы на вашем компьютере и на Windows-машине различны и вам нужно указать дополнительные опции монтирования: `codepage=<arg>` — для кодовой страницы, применяемой на удаленной машине (обычно для кириллицы это cp866), и `iocharset=<arg>` — для кодовой страницы на стороне Linux. Значение **iocharset** зависит от установленной локали, и узнать его можно по команде **locale**:

```
$ locale
LANG=ru_RU.UTF-8
[...]
```

В моем примере на стороне Linux используется Юникод (а могло бы быть, например, KOI8-R), и команда монтирования будет выглядеть так:

```
$ smbmount //user5/customers /mnt/customers \
>-o username=den,password="12345",\
>codepage=cp866,iocharset=utf8
```

Параметры, перечисленные после ключа –o, нужно разделять запятыми, но не пробелами.

Обратный слэш — это конструкция, позволяющая записать одну длинную команду на нескольких строках. Я использовал ее для наглядности, а вы пишите в одну строку.

Обратите внимание, что при монтировании внешних файловых систем значение кодовой страницы пишется как «cp866», а для локальных файловых систем (раздела FAT32 на вашем жестком диске, где установлена Windows) — просто «866».

Запускать программу **smbmount** имеет право только суперпользователь. Чтобы обычный пользователь мог ее запустить, следует установить для нее атрибут SUID, однако такое решение является небезопасным. Выходом из этого положения может послужить запуск программы **smbmount** при загрузке системы. Добавьте в сценарии автозагрузки (советую — в `/etc/rc.local`, см. п.9.1.2) вызов программы **smbmount** для монтирования файловых систем совместного использования, с которыми вы работаете чаще всего. После этого обычные пользователи смогут работать с удаленными ресурсами как с обычной локальной файловой системой.

6.4. Подключение к Интернету

Если вы собираетесь входить в Интернет через локальную сеть, то ничего дополнительно настраивать не нужно. Надо только в настройках `eth0` указать IP-адрес шлюза, через который вы собираетесь выходить, а остальное — забота администратора локальной сети.

Если же вы планируете выходить в сеть из дома через модем, то следующий параграф для вас.

6.4.1. Настройка модема

Модем подключается очень просто — вам достаточно вставить плату модема в корпус компьютера или подключить внешний модем, и система автоматически определит и установит его. В случае, если у вас старый дистрибутив, например, Red Hat Linux версии 6 и ниже, то вам самим придется создать ссылку `/dev/modem` на устройство `/dev/ttySn`, где **n** — это номер последовательного порта. Напомню, что устройство `/dev/ttyS0` соответствует порту COM1 в DOS. В принципе, создавать ссылку даже не обязательно, потому что в любой коммуникационной программе можно указать устройство, с которым она будет работать. Устройство `/dev/modem` используется большинством программ по умолчанию.

Для проверки работоспособности модема можно использовать программу **minicom**. Это обычная терминальная программа. Первый раз ее нужно запускать от имени суперпользователя с опцией **–s** (*setup*): в этом режиме программа выводит конфигурационное меню, позволяющее создать (или отредактировать) настроечный файл `/etc/minirc.dfl` (рис. 6.10). Нужно изменить только устройство, которое будет использоваться в качестве модема.

Рис. 6.10. *Программа настройки minicom*

Для тестирования модема обычно используются стандартные AT-команды. Инициализировать модем можно командой **ATZ**, поднять и положить трубку — **ATH1** и **ATH0** соответственно, а набрать номер — **ATDPномер**, используя импульсную систему набора номера, и **ATDTномер**, используя тональную систему.

Ссылку можно также создать программой **modemtool** в RedHat, а в Linux Mandrake нужно воспользоваться все тем же конфигуратором **DrakConf**.

Будет справедливо отметить, что Linux не работает с программными модемами для Windows (win-модемы). А вот модемы, которые подключаются к шине USB, в ОС Linux использовать можно. Только для этого нужно включить в ядро поддержку шины USB и USB-модемов.

В последнее время некоторые производители программных модемов (например, Lucent) обратили внимание и на Linux-пользователей. Компания Lucent выпустила драйверы под Linux для своего модема — их вы можете найти в Интернете.

Также следует учитывать появившиеся относительно недавно внутренние аппаратные PCI-модемы. С ними Linux работать может, но не со всеми и «через не хочу». Что можно порекомендовать пользователям, у которых установлен win-модем? Просто купить аппаратный COM-модем (не USB!). Этим вы сэкономите себе и нервы, и время. Внешний аппаратный модем, пригодный для домашнего использования, стоит от 20 до 40 долларов — не так уж и много. На сервере же внутренний модем вообще непригоден: тут нужен внешний и только внешний. Желательно ZyXEL. Цены на него начинаются от 100 долларов, но он того стоит, уж поверьте.

Если вами движет только спортивный интерес, мол, ни у кого не работает, а я настрою, что ж — дерзайте. Я могу помочь только тем, что дам ссылочки на сайты, где можно прочитать про настройку win-модемов:

- `http://www.linmodems.org/` — здесь можно найти ссылки на драйверы для Win-модемов;
- `http://linux-forever.narod.ru/writes/zyhel.htm` — настройка модема ZyXEL OMNI 56K PCI;
- `http://www.loe.lg.ua/help/new1/solaris/Untitled/index-106.html` — HOWTO по настройке Win-модемов.

Примечание

О том, как настроить ADSL-соединение и вообще выделенную линию, можно прочитать в моей книге «Linux-сервер своими руками». Также в этой книге описано подключение и настройка сетевой карты.

6.4.2. Подготовка к выходу в Интернет

Рассмотрим, как можно подключить Linux к Интернету. Отмечу, что это подключение имеет больший смысл, нежели подключение Windows, так как Linux намного лучше и, что самое главное, быстрее работает с сетевыми устройствами. Лично у меня соединение с моим провайдером при использовании Linux работает где-то в два раза быстрее и не простаивает, как при работе в Windows.

У вас, конечно же, есть модем, выбран и оплачен провайдер, от него получены логин-имя, пароль и все остальные сведения, которые могут потребоваться для доступа к его службам. Осталось убедиться в том, что ваша конфигурация Linux поддерживает протокол PPP, по которому происходит соединение с Интернетом, и настроить его. Проверьте, установлен ли у вас пакет `ppp`, и если нет, то доустановите его из вашего дистрибутива (п. 7.4).

В современных дистрибутивах настроить подключение по протоколу PPP можно по-разному:

- Используя конфигуратор (в Linux Mandrake это программа ***DrakConf***). Если операционная система предоставляет средства для удобной настройки соединения, так почему бы ими не воспользоваться? Зачем действовать в обход? Не модифицируем же мы реестр Windows вручную с помощью regedit для создания нового соединения.
- Используя программу **kppp**. В предыдущих версиях Linux Mandrake конфигуратор **DrakConf** был менее удобен, поэтому многие пользователи предпочитали использовать программу **kppp**. Она очень напоминает стандартный Windows-дайлер, в котором интегрированы все функции по установке и управлению соединениями.
- Редактируя конфигурационные файлы вручную (в этой главе этот способ не рассматривается. Если вы заинтересовались им, рекомендую прочитать мою книгу «Linux-сервер своими руками»).

Лучше всего использовать конфигуратор **DrakConf**, во всяком случае, при настройке первого соединения. Он автоматически установит все необходимые пакеты и создаст файл соединения для программы **kppp**. Устанавливать же настроенное соединение вы будете с помощью программы ***kppp***. Создавать следующие модемные соединения уже удобнее с помощью **kppp** — не нужно вызывать **DrakConf** и вводить пароль суперпользователя.

DrakConf (точнее, его модуль — **DrakConnect**) — отличная и удобная программа, но при работе с ней нужно учитывать некоторые нюансы. Теперь обо всем по порядку. Выполняем команду `drakconf`, выбираем апплет **Сеть и Интернет → Новое соединение → Модемное соединение**. Можно сразу запустить апплет **Сеть и Интернет** — командой **drakconnect**.

Сразу после этого программа предложит выбрать один из подключенных к вашей системе модемов. Затем все как будто просто. Программа поочередно будет запрашивать следующую информацию:

- Название соединения, номер телефона, имя пользователя и пароль.
- Назначение IP-адреса: автоматическое (с помощью DHCP) или вручную. Рекомендуется использовать автоматическое назначение IP-адреса, поскольку практически все провайдеры настраивают DHCP-сервер для назначения IP-адресов клиентов.
- Назначение IP-адреса DNS-сервера: автоматическое или ручное. Выбираем автоматическое и нажимаем **Далее**.
- Назначение IP-адреса шлюза: автоматическое.

Следующий вопрос конфигуратора касается запуска соединения при загрузке системы. Поскольку это модемное соединение, наверное, стоит отказаться от автоматического запуска соединения при загрузке системы.

Последний вопрос конфигуратора — стоит ли подключаться к Интернету прямо сейчас. Я согласился. Несколько минут **DrakConnect** тщетно

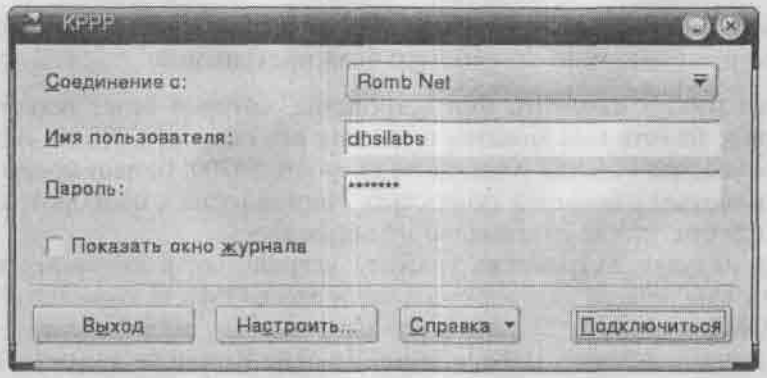


Рис. 6.12. *Основные параметры соединения*

пытался набрать номер, но так и не набрал его. А причина была в том, что он использовал тоновый набор номера, а моя АТС — импульсный. Да, вы правильно меня поняли, от проверки только что созданного соединения стоит отказаться.

Дальнейшую настройку соединения будем производить в программе **kppp**, потому что «умений и навыков» конфигуратора **DrakConf** для этого явно недостаточно.

Программа **kppp**, входящая в состав KDE (пакет kdenetwork), предназначена для того, чтобы совместно с демоном **pppd** (Point-to-Point Protocol Daemon) устанавливать и поддерживать соединение с провайдером. Запустите ее в окне терминала или выберите из меню **K** → **Интернет** → **Kppp** и сразу же нажмите кнопку **Настроить**.

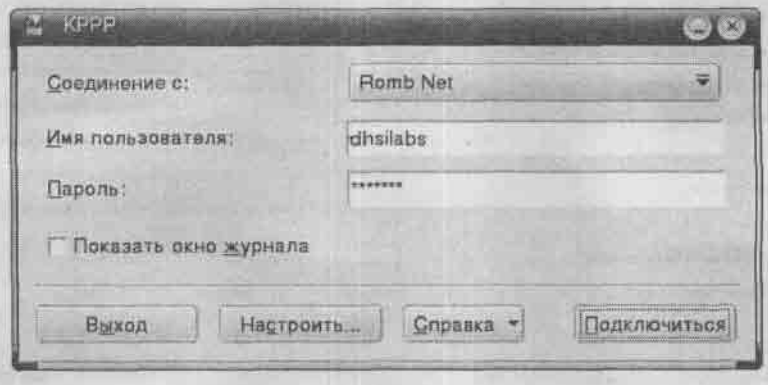


Рис. 6.12. *Программа kppp*

Интерфейс программы предельно прост. Однако хотелось бы дать несколько рекомендаций по работе с этой программой:

1. Вам нужно изменить имя устройства, которое будет использовано **kppp**, то есть имя модема, и указать его скорость. Но не подумайте только, что если вы установите скорость 56700, то ваш модем на 9600 заработает с большей скоростью. Иногда нужно, наоборот, занизить скорость, чтобы соединение не обрывалось.
2. На вкладке **Устройство** укажите устройство, к которому подключен ваш модем: `/dev/ttyS0`, если это COM1, и `/dev/ttyS1`, если COM2.
3. Измените команду набора номера в окне **Команды модема**. По умолчанию используется тональный набор (команда ATDT). Для работы с импульсной системой набора номера используйте команду ATDP.

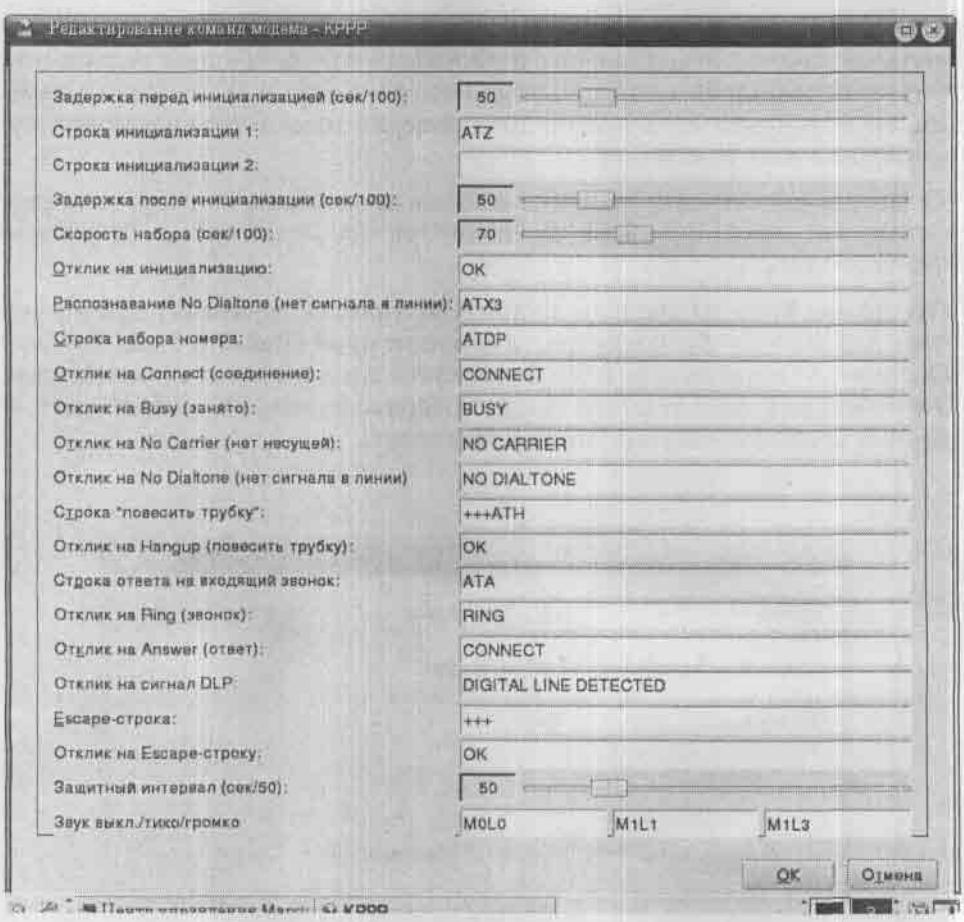


Рис. 6.13. Изменение команды набора номера

4. Нужно изменить права доступа к демону `/usr/sbin/pppd`, позволяющие обычному пользователю его запускать. В противном случае запускать программу **kppp** вам придется от имени суперпользователя.

Теперь можно попытаться набрать номер. Для этого нажмите OK и в окне KPPP нажмите кнопку **Подключиться**.

После установки соединения на панели задач KDE вы увидите статистику подключения, а именно, сколько времени вы провели в сети. Открыв окно с названием соединения, вы увидите скорость подключения и сможете прервать сеанс связи нажатием кнопки **Разъединить**.

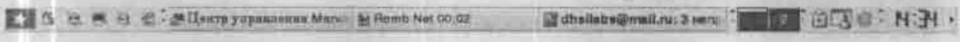


Рис. 6.14. *Панель задач KDE с кнопкой работающего соединения*

6.4.3. Специальные возможности браузеров

Неужели в этом параграфе будет рассказываться, как работать с браузерами? Тогда его можно вообще пропустить или прочитать по диагонали. Нет, уважаемый читатель, я и не сомневался, что вы умеете работать с браузерами. Сейчас мы поговорим об особенностях того или иного браузера, а также о некоторых специальных параметрах популярных браузеров.

Konqueror

Каковы же возможности Konqueror? Я и не сомневался, что он поддерживает последние стандарты HTML (4.0) и CSS. А вот как у него обстоят дела с Java и JavaScript? Очень много сайтов написано с использованием JavaScript и Java-апплетов.

Я попробовал загрузить документ, содержащий Java-скрипт. Визуальные эффекты отображались так же, как в старом недобром Internet Explorer. Это меня обрадовало. Но я решил копнуть глубже.

Вас не раздражают всплывающие окна? Этим особенно знаменита Система Лидеров. Например, когда вы заходите на сайт одного из участников системы, браузер открывает еще несколько окон и загружает в них сайты других участников. Кроме своих денег, мы еще теряем и часть нервных клеток — меня ничто так не раздражает, как вываливающиеся окна, а также приглашения установить Flash. К счастью, с окнами мы сейчас справимся. Выполните команду меню **Настройка → Настроить Konqueror**. Перейдите в раздел **Java & JavaScript** (рис. 6.15).

Вы можете вообще отключить JavaScript, но я предпочитаю создать «черный список». Нажмите кнопку **Добавить**, затем введите имя сайта или

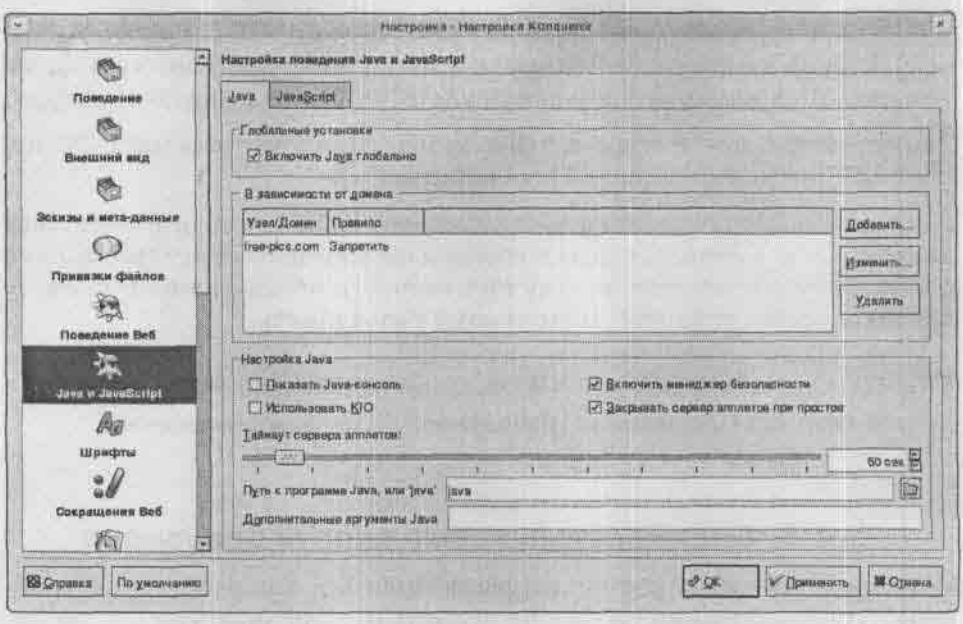


Рис. 6.15. *Настройка политики JavaScript*

домена, например, free-pics.com и правило **Запретить**. Обратите внимание: вы можете указать как имя узла, так и имя всего домена.

После настройки JavaScript я решил посмотреть, как же **Konqueror** работает с Java-апплетами. По умолчанию поддержка Java отключена, поэтому на вкладке Java ее нужно включить и указать путь к программе Java: `/usr/bin/java` или просто `java`.

С флешем (Flash) дела обстоят лучше. Браузер **Konqueror** использует плагин Flash Player программы **Netscape**. Поэтому, если в вашей системе установлен браузер **Netscape** или **Mozilla**, ваш Завоеватель будет отображать «флешки».

Но что больше всего мне понравилось в этом браузере, так это то, что я могу изменить строку идентификации агента. Например, я могу «замаскироваться» под Internet Explorer 5.5 on Windows 2000. Пусть думают, что у меня Windows стоит. Для чего это нужно? Когда вы посещаете узел, в протоколах веб-сервера отображается IP-адрес и строка идентификации агента, например, Mozilla/5.0 (compatible; Konqueror/3; Linux).

Galeon

Он был «придворным» браузером среды GNOME до версии 2.6. Начиная с этой версии вместо **Galeon** устанавливается его облегченная версия

Epiphany, а пакета Galeon может даже не оказаться в вашем дистрибутиве. Достать его можно из репозитория ALT Linux Sisyphus (`http://alt.linux.kiev.ua/srpm/galeon/get`). Основан **Galeon** на движке Mozilla (можно читать Netscape), поэтому он умеет все, что умеет **Netscape** (Mozilla). Это означает, что веб-страницы в браузере **Galeon** будут отображаться точно так же, как в браузере **Netscape** и наоборот.

Galeon без проблем справился со всеми приготовленными мною тестами. Самое главное, что мне не пришлось вокруг него плясать с бубном и песнями, чтобы запустить апплеты Java.

Что мне еще понравилось в этом браузере, так это работающая консоль Java. В отличие от **Konqueror**, в котором я не смог добиться появления этой консоли, в **Galeon** консоль Java еще и нормально функционировала. Для появления консоли выполните команду меню **Инструменты → Консоль Java**.

Диспетчер постоянных данных **Galeon** позволяет централизованно управлять Cookie и паролями. Для его вызова выполните команду меню **Инструменты → Cookie → Просмотр Cookie**. Вы можете удалить выбранные или все Cookie и одновременно запретить их прием.

Что такое Cookie? Некоторые сервера, когда вы их посещаете, записывают в определенный файл на диске какую-то информацию. Обычно это время последнего посещения сайта, ваше имя (если вы его ввели в анкете), параметры сайта и другие сведения. Например, для почтового сервера Mail.ru это дата последней регистрации и применяемая схема дизайна. Хранить эти несколько байтов, умноженных на огромное количество пользователей, на самом сервере невыгодно, поэтому сервер предписывает каждому клиенту хранить свои данные у себя. Это и есть Cookie.

Вы можете указать, каким сайтам вы разрешаете устанавливать Cookie, а каким — нет.

Не разрешайте устанавливать Cookie интернет-магазинам, и вот почему. Один клиент заходил в интернет-магазин, покупал товары, которые потом доставлялись ему домой. Он уже 2-3 месяца пользовался услугами этого магазина и не замечал ничего странного в поведении цен: рынок — дело непредсказуемое, цены могут постоянно меняться, на некоторые товары — в течение дня. Но однажды ему пришлось работать в интернет-кафе. Знаете, что он обнаружил, когда вошел на сайт магазина с другого компьютера? Что цены для новых пользователей на 20% ниже. Магазин привлекал покупателей низкими ценами, постепенно увеличивая цену (или делая формальные скидки) для постоянных клиентов. Старых клиентов магазин узнавал именно по Cookie, после чего предлагал им соответствующий прейскурант.

Epiphany

Прямо-таки браузер для настоящего аскета: минимум функций и жиденький интерфейс пользователя. У текстового браузера **lynx**, наверное, больше возможностей, чем у **Epiphany**. Единственная функция, оставшаяся от **Galeon** — это возможность управления Cookies. Вызвать ее можно из меню браузера **Правка → Изменить персональные данные**. Хотя, опять-таки, эту функцию «урезали» — если раньше можно было определить, какие сайты могут устанавливать Cookies, а какие — нет, то в **Epiphany** этого сделать нельзя.

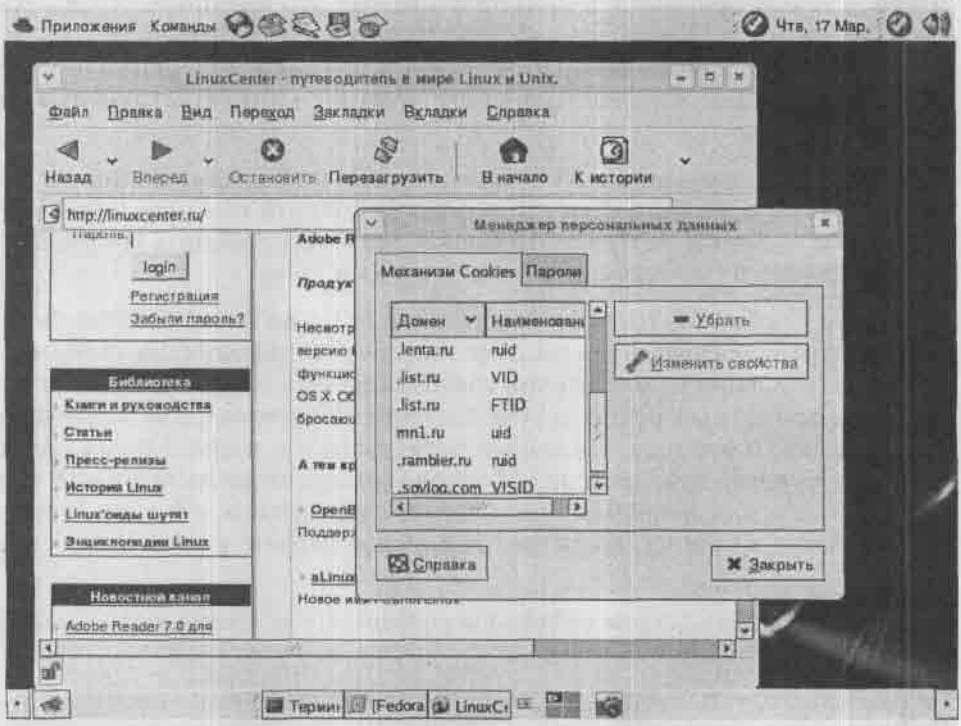


Рис. 6.16. *Менеджер персональных данных*

Netscape

Что я вам могу сказать относительно **Netscape**? Вы и так все знаете. Единственное правильное решение, которое вы можете принять при выборе браузера, это **Netscape**. Большинство страниц будут корректно отображаться в этом браузере. У вас никогда не будет проблем ни с апплетами Java, ни с Flash или JavaScript. То, что некоторые страницы будут выглядеть не так, как в Internet Explorer, так это уже не глюк На-

вигатора, а непрофессионализм веб-дизайнеров. Некоторые горе-мастера сознательно создают сайты, которые эффектно выглядят в IE, но куда хуже отображаются в **Netscape**. Это и есть непрофессионализм.

Легкие и быстрые браузеры **Konqueror** и **Epiphany** можно использовать для просмотра несложных страниц, например, чтения HTML-документации, а все остальные открывать с помощью **Netscape**. Ну и что, что время его запуска на несколько секунд больше?

FireFox

Браузер FireFox, основанный на движке Mozilla, появился сравнительно недавно. Его можно найти в последних версиях дистрибутивов (уже в Fedora Core 3 он есть). Большинство веб-страниц просматриваются в браузере без особых нареканий, кроме страниц, содержащих Flash-анимацию.

Для просмотра Flash-анимации вам нужно загрузить Macromedia FlashPlayer. Сделать это можно на сайте `http://www.macromedia.com`. Распакуйте его в какой-нибудь каталог, перейдите в этот каталог и введите команды:

```
$ chmod a+x flashplayer-installer
$ ./flashplayer-installer
```

После этого проверить корректность установки проигрывателя можно с помощью команды **about:plugins**, введенной в поле адреса браузера. В выводе браузера вы должны увидить что-то вроде этого:

```
MIME                          Type                 Suffixes   Enabled
application/x-shockwave-flash Shockwave Flash      swf        Yes
application/futuresplash      FutureSplash Player  spl        Yes
```

Если вы увидите эти строки, значит, плагин корректно установлен.

6.4.4. Текстовые браузеры

Текстовые браузеры не показывают рисунков, различных шрифтов — только текст. Зато при этом значительно повышается скорость доступа к WWW.

Популярны такие представители этого семейства, как **Lynx** и похожий на него **Links**.

Формат команды **lynx** таков:

```
$ lynx [опции] URL
```

Стрелки управления курсором «вверх» и «вниз» перемещают по ссылкам, стрелка «вправо» открывает ссылку, стрелка «влево» возвращает на предыдущую страницу. Справку можно получить, нажав клавишу <H>.

Некоторые полезные опции перечислены в таблице 6.4.

Опции программы lynx *Таблица 6.4*

Параметр	Описание
-anonymous	Использовать анонимную регистрацию
-auth=имя пароль	Использовать указанную информацию для аутентификации
-book	Загрузить страницу с избранными ссылками
-cache n	Установить количество кэшируемых документов
-case	Учитывать регистр букв при поиске
-dump	Преобразовать страницу в текстовый вид и отправить результат на стандартный вывод
-get_data	Использование метода GET для отправления данных
-nolist	Не выводить список ссылок при указании параметра –dump
-post_data	Отправка данных с помощью метода POST
-realm	Запрещает указывать URL при запуске
-reload	Сброс кэша прокси-сервера
-source	Отображает страницы Web в виде исходного текста HTML

6.4.5. Полезный трюк: Что делать, если браузер «подвисает» на какой-то странице

Иногда попадаются веб-страницы, которым требуется целая вечность, чтобы загрузиться. Если с одной и той же страницей это повторяется изо дня в день, то причина, скорее всего, в следующем: код загружаемой страницы содержит ссылки на какой-то объект (картинку или, еще хуже, рекламный баннер), расположенный на другом сервере, причем имя этого сервера невероятно трудно разрешить в IP-адрес. Вы думаете, что браузер «висит», а на самом деле он ждет ответа от модуля распознавания имен (резолвера), который в свою очередь ждет ответа от сервера DNS. Службе DNS посвящена 13 глава этой книги, и для лучшего понимания этого совета просмотрите хотя бы бегло эту главу.

Путей решения проблемы два (не считая тривиального — прекратить посещать «тормозную» страницу). Во-первых, можно установить кэширующий сервер DNS. Вы сэкономите не только время, но и деньги, ведь все запросы и ответы на них будут храниться в кэше вашего собственного сервера и, значит, завтра не понадобится обращаться к внешнему серверу DNS за разрешением того же самого имени.

Во-вторых, можно явно отказаться от попыток открыть элемент страницы по этой «проблемной» ссылке. В файле `/etc/nsswitch.conf` (п. 13.2) определен порядок просмотра источников модулем распознавания имен. Убедитесь, что первым опрашивается файл:

```
hosts: files dns
```

Этот файл — `/etc/hosts`. Впишите в него строку, «разрешающую» имя проблемного сервера в ваш собственный (localhost) IP-адрес:

```
127.0.0.1 www.server.nedostupen.com
```

Когда вы в следующий раз заглянете на эту страницу, браузер быстро получит ошибку, успокоится и продолжит грузить страницу.

6.4.6. Набор программ для работы в Интернете

Болталка ICQ

Разве может популярная операционная система не включить в свой состав популярную программу для общения? Нельзя сказать однозначно, что Linux является самой популярной операционной системой, первенство все-таки принадлежит Windows, но то, что ICQ — это самая популярная программа для общения, сомнению не подлежит. Посмотрите на свой UIN: вот столько пользователей было зарегистрировано до вас.

Количество пользователей ICQ растет не по дням и даже не по часам, а по минутам. Наверное, компания Mirabilis, когда разрабатывала первую версию ICQ, и не рассчитывала, что программа станет настолько популярной.

Клиент ICQ для Linux так и называется — Linux ICQ (`http://licq.org`). Запустить программу можно с помощью команды **licq**. Программа может даже работать в терминале (консоли), для этого введите команду `licq -p console`.

Менеджер закачки файлов Downloader for X

Штатной утилитой для скачивания файлов в Linux служит **wget**, поддерживающая протоколы FTP, HTTP и HTTPS. Она работает в консольном режиме и не требует взаимодействия с пользователем, то есть можно запустить процесс скачивания в фоновом режиме и завершить свой сеанс работы с Linux. Разработанная специально для медленных и ненадежных линий связи, она умеет возобновлять процесс закачки с того места, на котором он был прерван.

Рис. 6.17. *Сеанс ICQ для Linux*

Воспользуемся ею, чтобы скачать последнюю версию программы X-Downloader, разработанной Максимом Кошелевым (`www.krasu.ru/soft/chuchelo`):

```
$ wget http://www krasu.ru/soft/chuchelo/files/
d4x-2.5.0final-1.i386.rpm
```

Как видно из названия X-Downloader, этот менеджер закачек предназначен для работы в графической среде X Window.

Возможности программы таковы:

- Загрузка файлов по протоколам FTP и HTTP.
- Организация очередей и подочередей закачки.
- Многопоточный дизайн, благодаря которому файлы загружаются намного быстрее, потому что каждый файл закачивается одновременно несколькими потоками. Количество потоков можно настроить. По умолчанию оно равно пяти.
- Множественные параллельные закачки.
- Рекурсивное копирование по FTP и HTTP.
- Поддержка шаблонов при рекурсивном копировании по FTP.
- Поддержка прокси (FTP и HTTP).
- Принудительное ограничение скорости загрузки.
- Организация расписания загрузки.

- Приостановка закачки файлов и докачка в случае разрыва соединения.
- Поддержка метода drag-and-drop для добавления закачек.
- Прекрасная настраиваемость программы.
- Дополнительные возможности: шкуры (skins) и т.п.

Программа X-Downloader входит в состав некоторых дистрибутивов. Точно знаю, что она входит в состав дистрибутивов группы ALT.

Я с программой знаком достаточно давно. С тех времен программа заметно выросла, появились новые функции и усовершенствовались старые.

Запустите программу `d4x` в окне терминала или из меню **Команды → Запустить приложение**.

В левой части окна ***X*-Downloader** (рис. 6.18) отображаются активные очереди закачки. Каждая очередь содержит список файлов, которые нужно закачать. Работа с очередями (создание, удаление) осуществляется с помощью меню **Очередь**. Если щелкнуть правой кнопкой мыши на названии очереди и выбрать команду **Свойства**, мы сможем изменить параметры очереди.

Рис. 6.18. *Очередь закачек*

Сейчас в списке файлов очереди **Main** находятся три задания. Для каждого задания отображаются:

- Его состояние (иконка с изображением бутылки): загружается, загружен, приостановлен, неудачное завершение;
- Имя, тип, размер файла;
- Сколько загружено, сколько осталось, отношение размера загруженной части к размеру файла в процентах;

- Скорость соединения, время загрузки;
- Количество попыток загрузки.
- Адрес (полный URL файла).

На панели состояния отображается информация о загружаемом файле.

Чтобы добавить новую закачку, нажмите кнопку **Добавить закачку** на панели инструментов. Если вы предварительно поместили URL загружаемого файла в буфер обмена, то можете нажать кнопку **Добавить закачку из буфера**. А можно воспользоваться, наверное, самым удобным способом — просто перетащить ссылку из окна **Konqueror** в окно программы **X-Downloader**. Появится окно **Добавить закачку**, на вкладках которого можно дополнительно настроить параметры загрузки. Если ваш сервер требует авторизации, укажите имя пользователя и пароль. Обычно это нужно для неанонимных FTP-серверов.

На вкладке **Время** вы можете определить время и дату начала закачки. Это очень удобно, если вам нужно закачать какой-нибудь большой файл, например, то же ядро Linux: вы можете поставить закачку на ночь. Только не забудьте об этом и не выключите компьютер!

Дважды щелкнув на любой закачке, вы увидите протокол загрузки файла. В протоколе отображаются все ответы сервера и сообщения самой программы X-Downloader. Протокол помогает понять, почему же вы никак не можете закачать тот или иной файл.

Обратите внимание на горизонтальный «светофор» на панели инструментов. Это ограничитель скорости: красный свет означает, что будет использоваться самая низкая скорость, желтый — средняя, а зеленый — максимальная, то есть без ограничения.

Самые главные параметры находятся в группе **Закачка** (**Настройка → Свойства программы → Закачка → Ограничения**). Здесь вы можете определить:

- Таймаут чтения из сокета (по умолчанию 300 сек);
- Таймаут до повторного соединения (5 сек.);
- Количество попыток (0 — не ограничено);
- Откат после обрыва (0 байт, то есть закачка будет продолжена с того же самого места). Эта возможность позволяет заново передать указанное число байт, которые, возможно, были искажены перед обрывом соединения;
- Ограничение максимальной скорости (0 — без ограничения);
- Максимум строк в логах (100).

Еще одна полезная функция программы — Планировщик (**Настройки → Планировщик**). С его помощью можно создавать расписания закачек и выполнять другие действия, перечисленные в таблице 6.5.

*Допустимые действия Планировщика программы **X-Downloader*** *Таблица 6.5*

Действие	Описание
Ограничить скорость	С помощью этого действия вы можете как снизить, так и повысить скорость, поскольку вам по-прежнему доступны три режима: Минимум, Среднее и Неограниченное. Зачем это нужно? Например, вы целый день загружаете файлы параллельно с другой работой в сети. Тогда на время обеденного перерыва можно повысить скорость, а в остальное время снизить, чтобы закачка не загружала канал и тем самым не мешала вашей основной работе
Поднять главное окно	Отображает главное окно программы в указанное время
Выйти	Завершает работу программы
Удалить завершенные	Удаляет завершенные закачки из очереди
Удалить неудачные	Удаляет неудачные закачки из списка
Перезапустить закачку	Вы можете указать закачку, которую нужно перезапустить. Тут, наверное, нужно быть волшебником, чтобы угадать точное время перезапуска закачки
Остановить закачку	Останавливает закачку
Удалить закачку	Удаляет закачку. Закачка будет удалена в любом случае, даже если она еще не завершена
Удалить, если завершена	Удаляет закачку только в том случае, если она завершена
Добавить закачку	Добавляет закачку в список
Сохранить список	Сохраняет список
Выполнить команду	Позволяет выполнить внешнюю команду, то есть запустить программу в указанное время

FTP-клиенты

В любой сетевой операционной системе (а ОС Linux — именно такая) есть простейший FTP-клиент — программа **ftp**. Она предлагает интерфейс командной строки. В ответ на приглашение `ftp>` введите команду `help`, чтобы увидеть список встроенных команд **ftp**.

Эти команды очень похожи на те, с которыми вы уже встречались, изучая программу **smbclient**. Важнейшие из них для вас — команды **get** и **put**.

Удобный FTP-клиент с графическим интерфейсом — программа **gFTP**. Я рекомендую устанавливать самую новую версию программы, не ниже 2.0. Запустите программу командой **gftp** (см. рис. 6.15).

На рисунке видно, что я подключился к серверу ftp.yars.free.net под именем anonymous. Если ваш сервер не разрешает анонимный доступ, придется ввести имя и пароль. Порт в большинстве случаев — 21. После ввода всех необходимых параметров соединения нажмите кнопку **Соединить** (с изображением компьютеров). В правом углу находится кнопка **Отключиться**.

Рис. 6.19. *Окно программы gFTP*

С помощью меню FTP можно установить один из режимов работы FTP-сервера: ASCII (текстовые данные) или Binary (двоичные). Режим Binary устанавливается по умолчанию, и лучше эту установку не изменять: текстовый файл в этом режиме поврежден не будет, а вот наоборот... Дело в том, что текстовый режим заменяет символы конца строки на те, что используются на принимающей стороне, так что разбиение текста на строки сохраняется; понятно, что двоичному файлу от такой замены не поздоровится.

Чтобы принять или передать файл, пользуйтесь командами меню **Передача** или инструментальными кнопками, расположенными между областями локального и удаленного компьютеров.

Мне очень понравился инструмент **Сравнить окна**, который сравнивает окна **Локальное** и **Удаленное**. Эта функция незаменима, если вы обновляете свой веб-сайт по FTP.

Встроенными FTP-клиентами снабжены также браузеры **Netscape/Mozilla**, **Konqueror** и файловый менеджер **Midnight Commander**.

Однако все эти программы обладают одним небольшим недостатком: они не могут докачать файл в случае разрыва соединения (операция REGET). От этого неудобства избавлена программа **ncftp**.

Она не обладает удобным графическим интерфейсом, а больше похожа на стандартный FTP-клиент **ftp** и понимает его команды. Это значит, что

для соединения с узлом вам нужно ввести команду **open**, а для закачки файла — команду **get** и т.д.

Предположим, что когда мы выкачивали все rpm-пакеты, соединение было разорвано (ох уж эти линии!). Для докачки файлов установите соединение заново, откройте нужный вам сервер (например, `open ftp.redhat.com`) и введите команду:

```
ncftp> get -C *.rpm
```

Опция **–C** активизирует функцию докачки.

При работе с этой программой меня очень обрадовала возможность создания закладок «на лету». Например, если вы работали с сервером `ftp.redhat.com` и последним каталогом был каталог `/pub`, то при вводе команды `open redhat` программа установит связь с сервером `ftp.redhat.com` и перейдет в каталог `/pub`. Просмотреть и отредактировать закладки позволяет команда **bookmarks**.

Глава 7

ОСНОВЫ АДМИНИСТРИРОВАНИЯ СИСТЕМЫ

- ЧТО ПОНИМАЕТСЯ ПОД АДМИНИСТРИРОВАНИЕМ СИСТЕМЫ
- КОНФИГУРАТОРЫ LINUX
- ПОЛЬЗОВАТЕЛИ И КВОТЫ
- ПОДКЛЮЧЕНИЕ И КОНФИГУРИРОВАНИЕ АППАРАТНЫХ УСТРОЙСТВ
- УСТАНОВКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ
- КЛОНИРОВАНИЕ И ВОССТАНОВЛЕНИЕ СИСТЕМЫ

7.1. Что понимается под администрированием системы

В ОС Linux существует корневая учетная запись root для привилегированного пользователя, которому разрешено делать все: читать, изменять и удалять любые файлы, создавать и разрушать файловые системы, запускать и прекращать выполнение любых программ. Эти привилегии нужны для администрирования системы.

В обязанности системного администратора обычно входит:

- управление пользователями: ведение учетных записей пользователей и групп, квотирование дискового пространства;
- управление процессами: перераспределение ресурсов с целью повышения производительности системы;
- установка и модернизация программного обеспечения;
- подключение и конфигурирование аппаратных устройств;
- настройка системы: управление общесистемными сервисами, настройка сетевых служб с точки зрения безопасности и производительности, конфигурирование ядра;
- резервное копирование и восстановление данных.

Я включил главы, посвященные основам администрирования, в пользовательский раздел, потому что пользователю домашнего компьютера придется большинство администраторских обязанностей взвалить на себя. Помощи ему ждать неоткуда.

В следующем разделе я подробно остановлюсь на настройке различных сетевых служб.

Несколько советов начинающему администратору:

- Входить в систему под именем root только тогда, когда это абсолютно необходимо, и завершать сеанс привилегированной работы сразу же

после выполнения административной задачи. Чем реже вы работаете под именем root, тем меньше шансов случайно разрушить систему, спутав права пользователя root с правами других пользователей.

- Настроить строку приглашения командной оболочки так, чтобы для пользователя root она отличалась от приглашения для обычных пользователей. Как правило, строка приглашения root оканчивается символом #, а для других пользователей — символом $. Если все-таки есть вероятность забыть, под каким именем вы сейчас работаете, пользуйтесь командой **whoami**.
- Для часто выполняемых работ по обслуживанию системы сохранять нужные последовательности команд в файлах сценариев (скриптах), чтобы автоматизировать выполнение этих работ.

7.2. Конфигураторы Linux

Настройка всего программного обеспечения в UNIX-подобных системах сводится к редактированию текстовых файлов, которые программа прочитывает при запуске и которыми руководствуется при выборе режима работы. Каждая прикладная программа или демон, включая демон **init**, поддерживает отдельный набор этих, называемых конфигурационными, файлов (в отличие, скажем, от MS DOS, где в один файл могут быть объединены секции, управляющие различными подсистемами). Общесистемные службы держат их, как правило, в каталоге `/etc`, а пользовательские приложения — в домашних каталогах пользователей: такое размещение позволяет каждому пользователю настроить приложение в соответствии со своими нуждами.

Конфигурационные файлы самодокументированны: любая строка, начинающаяся с символа #, считается комментарием и может содержать любые справочные сведения. Кроме того, если вы вносите мелкую правку — скажем, изменяете значение одного параметра, — вам необязательно сохранять резервную копию старого конфигурационного файла: достаточно закомментировать строку с этим параметром.

Несмотря на удобство настройки путем редактирования текстовых файлов, все еще находятся администраторы, привыкшие к настройке в стиле Windows: через многоуровневые меню и диалоговые окна. Для таких администраторов предусмотрены графические конфигураторы, в разных дистрибутивах называющиеся по-разному. Запускать их нужно с графической консоли, потому что многие из них нуждаются в графическом режиме. В таблицах 7.1, 7.2 и 7.3 перечислены основные конфигураторы, включенные в дистрибутивы Mandrake, RedHat и Fedora Core.

Основные конфигураторы Linux Mandrake *Таблица 7.1*

Программа	Назначение
drakconf	Основной конфигуратор
drakboot	Конфигуратор загрузчика LILO/GRUB
drakgw	Совместное использование интернет-соединения
draknet	Настройка сети
drakfloppy	Создание загрузочного диска
draksec	Определение уровня безопасности
drakxservices	Автозапуск сервисов
diskdrake	Программа для работы с разделами диска
drakconcole	Доступ к консоли
draktime	Настройка даты и времени
diskdrake-fileshare	Разрешение совместного использования каталогов
adduserdrake	Управление учетными записями
harddrake2	Средство для настройки оборудования (в Linux Mandrake до версии 9 называется harddrake)
keyboarddrake	Настройка клавиатуры
localedrake	Изменение параметров локализации
mousedrake	Настройка мыши
menudrake	Настройка системного меню GNOME и KDE
printerdrake	Настройка принтера
netconf	Настройка сети
Logdrake	Поиск в файлах протоколов
Modemconf	Конфигурирование модема
Tinyfirewall	Параметры брандмауэра
XFdrake	Настройка сервера X
Xdrakres	Установка разрешения монитора
Xconfigurator	Настройка графической системы X Window

Основные конфигураторы Linux Red Hat *Таблица 7.2*

Программа	Назначение
Setup	Основной конфигуратор
control-panel	Вспомогательный конфигуратор
Modemtool	Конфигурирование модема
Printertool	Настройка принтера
Netconf	Настройка сети
Xconfigurator	Настройка X Window
Authconfig	Параметры аутентификации
redhat-config-securitylevel	Установка уровня безопасности (*)
redhat-config-language	Выбор языка (*)
redhat-config-date	Установка даты (*)
redhat-config-users	Управление пользователями (*)
redhat-config-packages	Работа с пакетами (*)

Продолжениет табл. 7.2

Программа	Назначение
redhat-config-xfree86	Настройка системы X Window (*)
redhat-config-printer	Конфигуратор принтера (*)
Sndconfig	Настройка звуковой платы

(*) Данные конфигураторы доступны в версии Red Hat 8.0 и выше

Основные конфигураторы Linux Fedora Core *Таблица 7.3*

Программа	Назначение
setup	Основной конфигуратор
system-config-packages	Установка и удаление пакетов
system-config-authentication	Параметры аутентификации
system-config-securitylevel	Установка уровня безопасности
system-config-securitylevel-tui	Настройка брандмауэра без графического режима
system-config-users	Управление пользователями
system-config-network	Настройка сети
system-control-network	Настройка сетевых интерфейсов
system-config-network-druid	Мастер установки сетевого соединения
system-config-services	Настройка служб
system-config-display	Настройка монитора и видеокарты
system-config-keyboard	Выбор раскладки клавиатуры
system-config-mouse	Настройка мыши
system-config-printer	Настройка принтера
system-config-soundcard	Настройка зауковой платы

Основной конфигуратор (**setup** или **drakconf**) обычно используют для первоначальной настройки системы сразу после установки.

7.3. Пользователи и квоты

7.3.1. Учетные записи пользователей

Система учета пользователей опирается на следующие конфигурационные файлы:

- `/etc/passwd` — учетная информация о пользователе;
- `/etc/shadow` — скрытая информация о пользователях: пароли в зашифрованном виде;
- `/etc/group` — информация о группах;
- `/etc/gshadow` — скрытая информация о группах;

- /etc/default/useradd — свойства, назначаемые по умолчанию новым учетным записям;
- /etc/login.defs — настройки безопасности пароля (время истечения, минимальная длина);
- /etc/skel — каталог, содержащий личные файлы настроек по умолчанию (когда для нового пользователя создается домашний каталог, в него помещаются эти файлы).

Файл /etc/passwd — информация о пользователях

Учетная информация о пользователях системы хранится в файле /etc/passwd в следующем виде:

```
username:password:UID:GID:full_name:home_dir:login_shell
```

Где:

- Username — это регистрационное имя пользователя, часто называемое логином. Обычно администраторы присваивают пользователям логины, образованные от их реальных имен. Имя root закреплено за суперпользователем. Кроме реальных пользователей, в системе существуют еще фиктивные (bin, daemon и т.п.), от имени которых запускаются общесистемные службы.
- Password — пароль. Обычно используются теневые пароли, и вместо пароля в файле /etc/passwd стоит знак *, а сам пароль в зашифрованном виде хранится в файле /etc/shadow.
- UID — уникальный числовой идентификатор пользователя из диапазона от 0 до 65534. Суперпользователю присвоен UID, равный 0.
- GID — числовой идентификатор первичной группы пользователя. Помимо первичной группы, пользователь может входить или не входить в состав разных групп, но в первичную группу (*native group*) он входит всегда. Пользователи, входящие в одну группу, могут работать с общими файлами. Первичная группа суперпользователя называется root. Ей присвоен GID, равный 0.
- Full_name — обычно представляет собой реальное имя пользователя, например Ivan Ivanov. Может содержать и другие данные: номер телефона и т.п. Эти сведения системой не используются, но доступны по команде finger <username>.
- Home_dir — домашний каталог пользователя. Стандартное место для него — это каталог /home/<username> (например, /home/den), и без особых причин изменять такую организацию домашних каталогов не рекомендуется.
- Login_shell — командный интерпретатор, запускаемый по умолчанию при входе пользователя в систему.

Пример фрагмента файла /etc/passwd:

```
root:x:0:0:root:/root:/bin/bash
```

```
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
den:x:500:500:den:/home/den:/bin/bash
evg:x:501:501::/home/evg:/bin/bash
```

Применение теневых паролей оправдывает себя с точки зрения безопасности. Обычно к файлу /etc/passwd разрешен доступ в режиме «только чтение» всем пользователям. К файлу /etc/shadow обычный пользователь не имеет даже такого доступа.

Системы, использующие TCB (*Trusted Computing Base*), хранят пароли не в файле /etc/shadow, а в файлах /etc/tcb/<имя_пользователя>/shadow. Считается, что технология TCB более практична с точки зрения безопасности, поскольку доступ к паролям отдельного пользователя не позволит злоумышленнику скомпрометировать сразу всю систему. Для включения поддержки TCB нужно установить пакет tcb. Многие современные дистрибутивы (например, дистрибутивы ALT Linux) поддерживают TCB по умолчанию.

В качестве основного алгоритма шифрования используется MD5, превращающий пароль в 32-значное шестнадцатеричное число. Этот алгоритм является самым надежным. Раньше использовались алгоритмы DES и 3DES, но здесь я не буду подробно останавливаться ни на одном из них. При установке системы обычно спрашивается, хотите ли вы использовать теневые пароли (Shadow Passwords) и MD5. Я очень рекомендую вам использовать обе эти возможности.

Информация о группах пользователей. Файл /etc/group

Информация о группах пользователей хранится в файле /etc/group в следующем формате:

```
имя_группы:пароль:GID:члены_группы
```

Пароль используется крайне редко. Пример фрагмента файла /etc/group:

```
root:*:0:root
bin:*:1:root,bin,daemon
local:*:100:den,operator,ivan
guest:*:200:
dialup:*:250:victor,evg
```

В этом примере группа root зарезервирована для пользователя root. Группа с идентификатором 100 используется для локальных пользователей. В ее состав входят пользователи den, operator, ivan. Группа guest предназначена для гостевого входа и пользователя guest. В состав группы dialup входят пользователи victor и evg.

Системой определяются группы для фиктивных пользователей: `bin`, `sys`, `adm` и т.п. Реальные пользователи входить в них не могут. Эти группы используются для системных файлов.

Добавить группу вы можете с помощью команды **groupadd**. Я, как правило, просто добавляю запись в файл `/etc/group`, а если мне нужно удалить группу, то удаляю соответствующую строку.

7.3.2. Создание и удаление пользователей и групп

Большинство утилит административного назначения хранится в каталогах /sbin и /usr/sbin. Полный набор утилит для управления пользователями и группами, установленных в вашей системе, вы можете увидеть по команде

```
# ls {/sbin,/usr/sbin}/*{user,group}*
```

Выполнив подстановки аргументов, командная оболочка ***bash*** превратит эту команду в четыре:

```
ls /sbin/*user*
ls /sbin/*group*
ls /usr/sbin/*user*
ls /usr/sbin/*group*.
```

Чтобы добавить пользователя, выполните команду

```
# useradd <имя> ; passwd < имя >
```

Для удаления учетной записи пользователя служит утилита **userdel**, а для модификации существующей учетной записи — утилита **usermod**. Все они снабжены man-страницами.

Чтобы добавить или изменить несколько учетных записей сразу (на больших системах это приходится делать чаще, чем на персональном компьютере), можно воспользоваться утилитой **newusers**.

Для групп те же функции создания, удаления и модификации выполняют утилиты **groupadd**, **groupdel** и **groupmod** соответственно.

Если вам почему-либо неудобно управлять пользователями из командной строки, можно делать это с помощью графического конфигуратора. На рис. 7.1 показан пример работы **system-config-users** из дистрибутива Fedora Core 3.

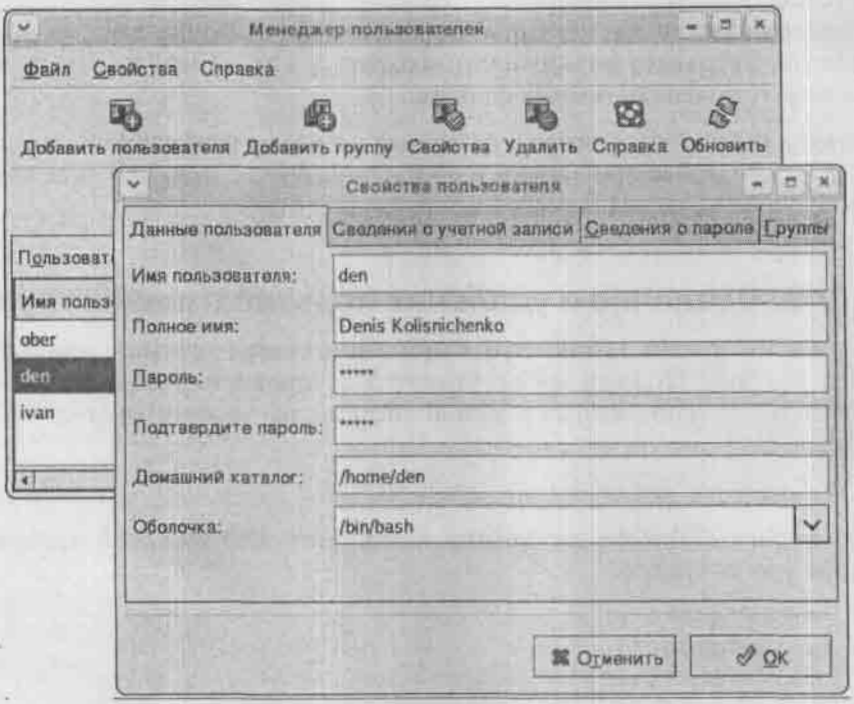


Рис. 7.1. *Добавление учетной записи*

7.3.3. Квотирование

Что такое квотирование. Особенности квотирования в Linux

Квотирование — мощный механизм ограничения использования дискового пространства, применявшийся еще в самых ранних версиях UNIX, тогда как в состав ОС семейства Windows компания Microsoft включила их только в Windows 2000, естественно, гордо заявив об этом. Этот материал настолько объемен, что ему можно было бы смело посвятить целую главу.

При помощи квот системный администратор принуждает пользователя не расходовать неограниченный объем дискового пространства. Существует два типа ограничений: ограничение на количество файлов (*inodes*) и ограничение на размер дискового пространства в килобайтах (*blocks*). Если установлены оба ограничения, то они будут применяться одновременно.

Ограничения на inodes и на blocks могут быть установлены как для пользователя, так и для группы. Если вы входите в группу, которая превысила

наложенное на нее ограничение, то вы не сможете использовать дисковое пространство, даже если вы не превысили квоту как пользователь.

Ограничения устанавливаются отдельно для каждого пользователя на каждой файловой системе. Ограничение определяется четырьмя числами:

1. Текущее значение ограничения;
2. «Мягкое» ограничение (*softlimit*);
3. «Жесткое» ограничение (*hardlimit*);
4. Период отсрочки: время, по истечении которого «мягкое» ограничение будет интерпретироваться как «жесткое».

«Мягкое» ограничение определяет число блоков, которые пользователь все еще может превысить, «жесткое» ограничение превысить невозможно. При попытке сделать это пользователь получит сообщение об ошибке. По истечении определенного времени (обычно 7 дней) «мягкое» ограничение переходит в «жесткое». За это время пользователь должен удалить ненужные ему файлы.

Ядро и поддержка квотирования

В ядро ОС Linux поддержка квотирования встроена (по некоторым сведениям, она появилась в ядре, начиная с версии 1.3.8), осталось убедиться, что она у вас включена. Если это не так, то ядро придется перекомпилировать.

После установки дистрибутива исходные тексты модулей ядра обычно находятся в каталоге `/lib/modules/<версия_ядра>/build`.

Примечание

Привыкайте к возможностям подстановки аргументов командной оболочкой **bash**: перейти в этот каталог можно командой

```
$ cd /lib/mod*/`uname -r`/build
```

Там лежит файл `.config`, хранящий текущие настройки ядра (имя файла начинается с точки, поэтому он считается скрытым: чтобы увидеть его по команде **ls**, пользуйтесь ключом **-a**). Найдите в нем строку CONFIG_QUOTA и убедитесь, что значение этого параметра равно «y». Далее обратите внимание на формат квот, строки CONFIG_QFMT_V<x>, где x = 1 или 2. Версия 1 — это старый формат, использовавшийся в ядрах до версии 2.6. Если ваш дистрибутив основан на ядре 2.4, должен быть включен старый формат. Он отличается именами файлов квот.

Ту же самую проверку (при необходимости — настройку) можно выполнить и с помощью диалогового конфигуратора. Чтобы запустить его, выполните одну из следующих команд:

- **make menuconfig**: работает в консоли, предлагает текстовые меню и кнопки;
- **make config**: работает в консоли, задает вопросы;
- **make xconfig**: работает в графическом режиме при запущенной системе X Window;
- **make gconfig**: то же самое, только для оконной среды GNOME.

На рис. 7.2 показано окно конфигуратора ядра для оконной среды GNOME. Обратите внимание, что квоты поддерживаются только для файловых систем ext2, ext3 и ReiserFS.

Проверьте, установлены ли у вас утилиты квотирования, командой

```
# ls {/sbin,/usr/sbin,/usr/bin}/*quota*
```

Скачать пакет этих утилит можно из репозитория Сизиф, поддерживаемого командой разработчиков ALT Linux: `http://alt.linux.kiev.ua/srpm/quota/get`.

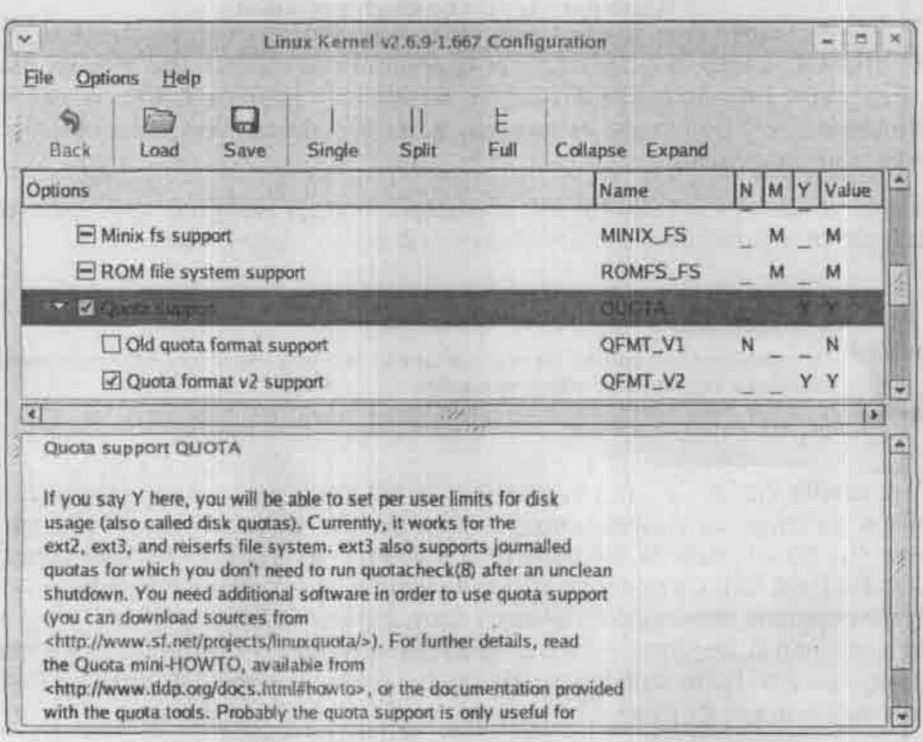


Рис. 7.2. *Графический конфигуратор ядра, меню Filesystems*

Назначение и активация квот

Теперь нужно определить, использование какой файловой системы вы хотите ограничить. Обычно это `/home` (домашние каталоги пользователей), `/usr` (пользователи имеют право записывать информацию в этот каталог) и, возможно, `/var`. Чтобы включить квотирование файловой системы, отредактируйте файл `/etc/fstab`, добавив ключ **usrquota** (и/или **grpquota** для групп) в поле, содержащее слово **defaults**

```
/dev/hda1  /      ext2  defaults
/dev/hda4  /home  ext3  defaults,usrquota=/путь/к/файлу/квот
/dev/hda5  /usr   ext3  defaults,usrquota,grpquota
```

Если путь к файлу квот не указан, то используется файл в корневом каталоге квотируемой файловой системы. В версии 2 (ядро 2.6) он называется `aquota.user` (`aquota.group` для групп), в версии 1 — `quota.user` и `quota.group`.

Периодически необходимо проверять файлы ограничений и само дисковое пространство, выделенное пользователям, на целостность, особенно после аварийного завершения работы системы. Для этого используется команда **quotacheck**. Не рекомендуется применять ее к используемой в данный момент, а поэтому подверженной изменениям, файловой системе: сама программа пытается перемонтировать проверяемую файловую систему в режиме «только чтение», но на занятой ФС это невозможно. Рекомендуется также на время проверки отключать квотирование.

Если программа **quotacheck** не находит файлов ограничений, то создает их.

Для проверки файловых систем на число блоков, которые используются пользователем или группой, выполните команду:

```
# quotacheck -avug
```

Ключ **–a** требует проверки всех смонтированных файловых систем (если его не указывать, то нужно задать точку монтирования), ключ **–v** требует подробного отчета о работе команды, ключи **–ug** требуют проверки квот как для пользователей, так и для групп.

Команда `quota <имя> | <имя_группы>` позволяет просмотреть ограничения дискового пространства, наложенные системным администратором на данного пользователя или группу, а команда **repquota** — сводку ограничений и фактически занятого дискового пространства для всех пользователей данной файловой системы (с ключом **–a** — всех файловых систем).

Только суперпользователь может просматривать квоты других пользователей. Обычный пользователь может просматривать только свои квоты и квоты группы, к которой он принадлежит.

```
Terminal - Terminal
Файл Сессии Настройки Помощь
[root@dhsilabs /root]# quotacheck -avug
Scanning /dev/hda5 [/] done
Checked 6730 directories and 109842 files
Using quotafile /quota.user
Updating in-core user quotas
Using quotafile /quota.group
Updating in-core group quotas
Scanning /dev/hda8 [/mnt/ext2] done
Checked 7 directories and 11 files
Using quotafile /mnt/ext2/quota.user
Updating in-core user quotas
[root@dhsilabs /root]#
Новый   Terminal No 1
```

Рис. 7.3. *Результат выполнения команды quotacheck –avug*

```
Terminal - Terminal
Файл Сессии Настройки Помощь
[root@dhsilabs /root]# repquota -ua
                        Block limits                    File limits
User            used    soft    hard  grace    used  soft  hard  grace
root      --  1682876       0       0          113629     0     0
bin       --      348       0       0               4     0     0
daemon    --        8       0       0               3     0     0
lp        --      276       0       0               9     0     0
news      --       56       0       0               2     0     0
uucp      --        0       0       0               1     0     0
ftp       --        8       0       0               2     0     0
named     --        4       0       0               1     0     0
apache    --       24       0       0               4     0     0
xfs       --       12       0       0               4     0     0
htdig     --     6184       0       0              21     0     0
mysql     --      316       0       0              53     0     0
postfix   --       60       0       0              32     0     0
195       --     4428       0       0             570     0     0
den       --   167320       0       0            1720     0     0
synthetic --      808       0       0             137     0     0
1000      --     6240       0       0             370     0     0
1001      --       72       0       0              11     0     0

                        Block limits                    File limits
User            used    soft    hard  grace    used  soft  hard  grace
root      --   429980       0       0              16     0     0
den       --        4       0       0               1     0     0
[root@dhsilabs /root]#
Новый   Terminal No 1
```

Рис. 7.4. *Результат выполнения команды repquota –ua*

Для задания ограничений предназначена команда **edquota**. Файлы ограничений при этом должны уже существовать. Создайте их и включите режим подсчета квот командами:

```
# quotacheck -u <точка_монтирования> # для групп нужно
                                     # выполнить то же
# quotaon -u <точка_монтирования>    # самое с ключом -g
```

Команда `edquota -u <имя>` создает временный текстовый файл, представляющий собой выдержку из двоичного файла квот, запускает ASCII-редактор, указанный в переменной окружения $EDITOR (по умолчанию это редактор **vi**), в котором вы можете отредактировать ограничения для данного пользователя, и записывает сохраненный текстовый файл обратно в файл квот.

Текущие показатели занятого дискового пространства приводятся только для справки, редактировать нужно только число, которое следует за словом `hard` или `soft` (рис. 7.5). Значение «0» указывает на отсутствие ограничений.

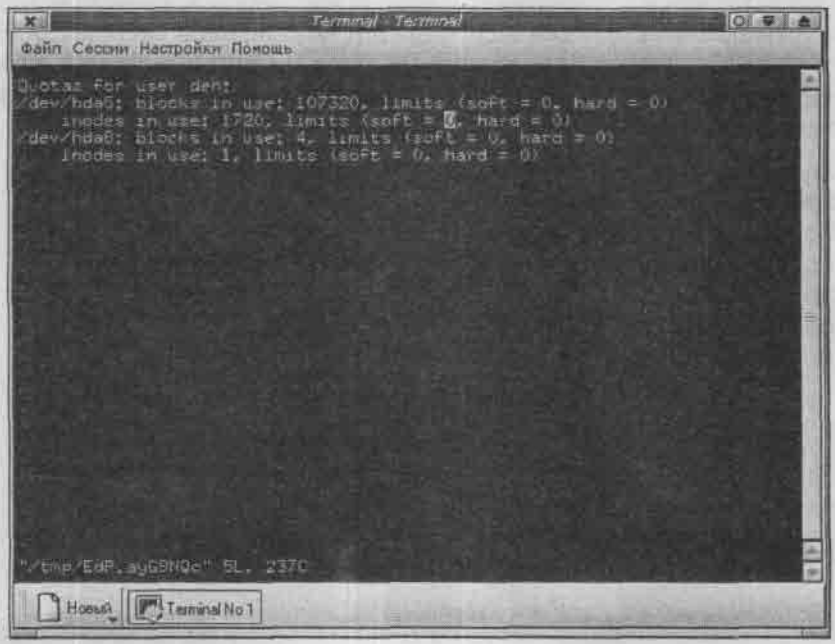


Рис. 7.5. *Результат выполнения команды edquota -u den*

Чтобы изменить период отсрочки, пользуйтесь ключом **-t**:

```
# edquota -t
Time units may be: days, hours, minutes, or seconds
Grace period before enforcing soft limits for users:
/dev/hda4: block grace period: 50 minutes,
file grace period: 50 minutes
```

В старой версии формата квот существовало значение периода отсрочки по умолчанию, вкомпилированное в ядро. В версии 2 значения по умолчанию нет, и период отсрочки для каждой файловой системы нужно указывать явно.

В большинстве случаев у вас есть несколько пользователей, на которых нужно наложить одинаковые ограничения. Самым быстрым способом редактирования ограничений в этом случае будет использование прототипа. С помощью команды:

```
# edquota -u <пользователь_который_станет_прототипом>
```

можно определить ограничения прототипа, а затем с помощью команды:

```
# edqouta -p <прототип> <пользователь>
```

создать квоты для всех оставшихся пользователей, применив к ним ограничения прототипа.

Для активации/деактивации подсчета квот на файловой системе, для которой заказано квотирование, предназначены команды **quotaon** и **quotaoff**. Команду **quotaon** нужно включить в сценарий загрузки системы, как и команду **quotacheck**:

```
# Смонтированы все файловые системы из /etc/fstab
# Проверка и активация квот
echo "Checking quotas. This may take some time."
/usr/sbin/quotacheck -avug
echo " Done."
echo "Turning on quota."
/usr/sbin/quotaon -avugfi
```

7.4. Подключение и конфигурирование аппаратных устройств

7.4.1. Ядро и поддержка устройств

Ядро ОС Linux может быть собрано как монолитное или модульное. Монолитное ядро — это один большой файл, в который включены сразу все возможности, заложенные в данную версию ядра. Оно без всяких изменений находится в оперативной памяти от запуска до остановки системы.

В модульном же варианте сборки в ядро включают только самый необходимый код, обеспечивающий загрузку системы. Все возможности, которые могут быть вынесены в отдельный файл (модуль), туда и выносятся, чтобы при необходимости динамически подключать их к ядру и отключать без перезагрузки компьютера. В результате ядро получается небольшим, быстрым и гибким.

Раньше, в первых версиях ядра Linux, механизм работы с модулями не был предусмотрен, и ядра тех времен содержали в себе код драйверов для всех поддерживаемых устройств. Такое решение нельзя было назвать рациональным: невозможно предусмотреть, какие устройства будут установлены у конечного пользователя, даже если включить в состав ядра драйверы всех известных устройств. Кроме того, даже если нужное устройство (скажем, звуковая плата Yamaha) ядром распознается, то драйверы остальных устройств того же назначения будут впустую занимать оперативную память. Поэтому, начиная с версии 2.0, ядро Linux поддерживает модульную организацию и из дистрибутива, основанного на ядре 2.x, ставится в модульном виде.

При этом, кроме файла образа ядра `/boot/vmlinuz-<версия_ядра>`, ставится каталог с модулями `/lib/modules/<версия_ядра>` и файл образа загрузки `/boot/initrd-<версия_ядра>`. Образ загрузки содержит все модули, необходимые для того, чтобы ядро загрузило систему. Без этих модулей оно неспособно подключить системный раздел жесткого диска и прочитать файлы. Другие модули подключаются к ядру сценариями загрузки при старте системы.

Перед тем, как устанавливать новое оборудование, нужно убедиться, что ядро поддерживает ваше устройство. Если это не так, нужно пересобрать ядро, включив поддержку нового устройства. Можно со стопроцентной уверенностью сказать, что ваше ядро будет поддерживать вашу сетевую плату RTL8139 или любую другую, совместимую с NE2K PCI. А вот о поддержке USB-модема или принтера заранее ничего сказать нельзя: пользуйтесь диалоговым конфигуратором ядра (п.7.2.3.1) или загляните

в базу поддерживаемого оборудования по адресу `http://www.mandrakelinux.com/en/hardware.php3`. Ничего, если у вас другой дистрибутив, например, основанный на Red Hat: основные устройства те же. Современное ядро версии 2.6 поддерживает очень много устройств, и проблемы могут возникнуть только со следующими их типами:

1. Win-модемы, то есть программные модемы, часть функций которых выполняет сама ОС Windows. Я не говорю, что под Linux они вообще не работают, но, потратив уйму времени, даже если вы и настроите этот модем, удовольствия от его работы вы не получите.
2. Win-принтеры — комментарии те же, что и для Win-модемов. Разработчики дешевых устройств, как правило, стараются сэкономить на драйверах для менее распространенных ОС.
3. Экзотические TV- и FM-тюнеры.

7.4.2. Утилиты для работы с модулями

Основу модульной организации ядра составляет возможность динамической загрузки и выгрузки модулей. Обеспечивается эта возможность тем, что, в отличие от обычного приложения, модуль имеет несколько точек входа, исполняемых при установке и удалении модуля из ядра, а также при обработке поступающих от пользователя запросов (рис. 7.6).

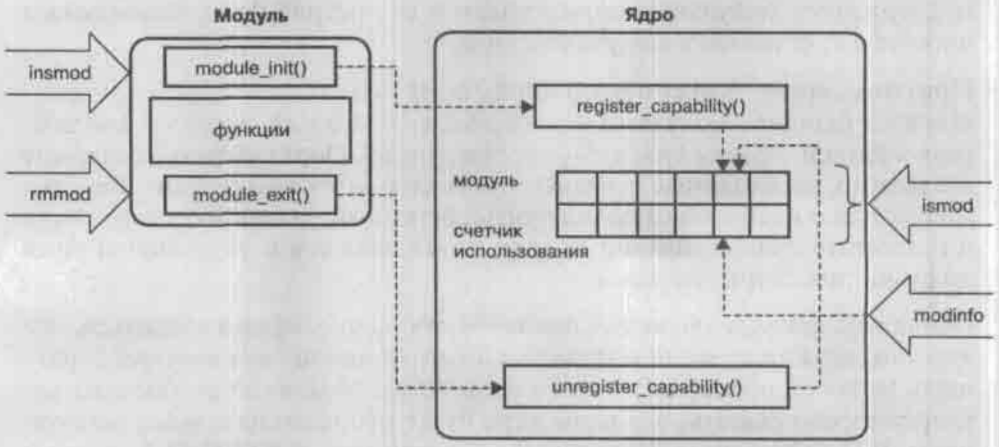


Рис. 7.6. *Связь модуля с ядром*

Утилиты, обеспечивающие загрузку, выгрузку и просмотр загруженных модулей, собраны в пакет, который для ядер 2.4.x называется **modutils**, а для ядер 2.6.x — **module-init-tools**. Не смешивайте эти пакеты: одноименные утилиты из них конфликтуют друг с другом.

Выполнять эти утилиты может только суперпользователь. В состав обоих пакетов входят:

- **lsmod** — просмотр списка загруженных модулей;
- **modinfo <имя_модуля>** — получение информации о загруженном модуле;
- **insmod < имя_модуля >** — загрузка модуля;
- **rmmod < имя_модуля >** — выгрузка модуля;
- **depmod** — нахождение зависимостей между модулями;
- **modprobe** — загрузка модуля с аргументами и теми модулями, от которых он зависит.

Из сценария инициализации системы вызывается именно **modprobe**. Эта команда руководствуется конфигурационным файлом `/etc/modprobe.conf`, в котором могут быть записаны аргументы, передаваемые загружаемым модулям, определены псевдонимы модулей и указаны команды, которые нужно выполнить перед стандартной процедурой загрузки модуля или вместо нее. В дистрибутивах, основанных на ядре 2.4, этот файл называется `/etc/modules.conf` и имеет несколько более сложный синтаксис. В совсем старых версиях Linux (до дистрибутива Red Hat Linux 7.0) вместо этого файла использовался `/etc/conf.modules`.

7.4.3. Kudzu — утилита для автоматического определения устройств

В Linux для автоматического определения устройств используется специальная утилита **kudzu**, названная в честь китайской лианы — злостного сорняка. В дистрибутивы, основанные на Linux Mandrake, вместо нее может входить утилита **harddrake**. Задача этой утилиты состоит в том, чтобы определить, какие устройства установлены, и добавить в файл `/etc/modprobe.conf` (как бы он ни назывался в вашем дистрибутиве) команды загрузки модулей ядра с драйверами для этих устройств.

Обычно **kudzu** запускается при каждом запуске системы из сценария загрузки. Ее работа занимает довольно заметное время, поэтому я рекомендую сразу после установки дистрибутива, когда все устройства уже определены и настроены, отключить ее автоматический запуск. Если вы установите новое устройство, что случается не каждый день, запустите **kudzu** вручную от имени суперпользователя.

Напоминаю, что отключить автоматический запуск служб можно с помощью диалогового конфигуратора (см.п.7.1) **system-config-services** или **drakxservices**, в зависимости от дистрибутива.

Если нужно передать драйверу нового устройства дополнительные параметры, отредактируйте вручную файл `/etc/modprobe.conf`.

7.4.4. Настройка установленных устройств

Настройка устройства обычно выполняется с помощью диалогового конфигуратора устройств соответствующего типа. Например, для настройки принтера можно запустить **system-config-printer** в дистрибутивах Fedora, **redhat-config-printer** в дистрибутивах Red Hat, **printerdrake** в дистрибутиве Mandrake.

Если вы не знаете точного названия нужного конфигуратора, вам поможет функция автозаполнения командной строки (п.2.1.4.7): имена многих конфигураторов начинаются одинаково, с «system-config» («redhat-config») или «drak». Если подходящий конфигуратор таким способом не нашелся, командой **which** определите каталог, в котором находятся найденные конфигураторы, поройтесь в нем командой `ls *conf*` и `man <утилита_похожая_на_нужную>`.

И, наконец, пользуйтесь командой **apropos** с аргументом «conf», которая в числе прочего сообщит вам об установленных в вашей системе конфигураторах и конфигурационных файлах, для которых предусмотрена справка.

7.5. Установка программного обеспечения

В ОС Windows установка новых программ происходит просто: достаточно запустить `setup.exe`, ввести серийный номер, каталог для установки и нажать на кнопку «Далее». После этого вы можете поступить так, как рекомендует Microsoft: «откиньтесь на спинку стула и подождите, пока программа установки все сделает за вас».

В Linux же установить программное обеспечение можно одним из трех способов: из исходного кода, из бинарного пакета и из пакета, содержащего исходный код. Рассмотрим по порядку все три способа.

7.5.1. Установка из исходных текстов

Бесплатное распространение исходных текстов программ — именно то, что делает Linux уникальной операционной системой и составляет одно из

Величайших Достижений Человечества. Поэтому традиционный способ распространения приложений под Linux — это архивы исходных текстов (в просторечии — тарболлы).

Обычно имя файла, содержащего такой архив, имеет двойное расширение: например, tar.gz или tar.bz2. Это означает, что данный файл получился в результате работы сначала архиватора tar (*Tape Archive*, по первоначальному назначению — работе с ленточными накопителями), а потом компрессора **gzip** или **bzip2**. Чтобы распаковать архив, нужно применить сначала декомпрессор **gunzip** или **bunzip2**, после чего разархивировать его командой **tar**.

Иногда расширение только одно: **tgz**. В этом случае нужно запускать разархиватор **tar** с ключом, указывающим ему на необходимость применить фильтр-декомпрессор **gunzip**.

Формат команды **tar**:

```
tar [ключи] [файл_архива] [архивируемые файлы и/или каталоги]
```

Подробные сведения о ключах команды **tar** ищите на man-странице, я перечислю только самые употребительные:

- **c** (*create*) — создать архив;
- **x** (*eXtract*) — извлечь файлы из архива;
- **t** (*lisT*) — показать содержимое архива;
- **v** (*verbose*) — выводить на консоль подробный отчет о своей работе;
- **f** — работать с файлом, а не ленточным накопителем;
- **z** — применить фильтр-компрессор при создании архива или декомпрессор при распаковке.

Архивный файл обычно содержит дерево каталогов, которое после распаковки будет создано в вашем текущем каталоге, или, по случайной небрежности создателя архива, — несколько деревьев и файлов россыпью: в этом случае каталог для распаковки вам нужно создать вручную. Таким образом, первый шаг при установке из исходных текстов пакета `program 3.14.tar.bz2` — распаковка — выглядит так:

```
$ bunzip2 program 3.14.tar.bz2
$ tar tvf program 3.14.tar # проверьте, есть ли объемлющий каталог
$ ^tv^xv # оболочка bash превратит это в команду
tar xvf program 3.14.tar
```

Следующий шаг — собственно установка. Перейдите в распакованный каталог (обычно он называется <имя_пакета-версия>) и прочитайте все README-подобные файлы, которые там найдете. Обычная процедура установки состоит из трех этапов:

1. `$ ./configure # помните, что текущего каталога в $PATH нет?`

   Сценарий configure, приложенный к архиву, опрашивает компоненты вашей системы с целью определить, сможет ли устанавливаемый пакет собраться и заработать именно у вас и что в нем для этого надо «подкрутить». При успешном завершении он создает файл Makefile — основной документ для сборочной утилиты **make**, содержащий инструкции и необходимые параметры (пути к заголовочным файлам, библиотекам и т.п.) для компиляции и сборки программ пакета.
2. `$ make`

   Собственно компиляция и сборка.
3. `$ make install`

   Установка собранных программ пакета, конфигурационных файлов и справочных страниц в каталоги, указанные в Makefile. Обычно исполняемые файлы помещаются в каталог `/usr/bin`, а man-страницы — в `/usr/man`, но после этапа конфигурирования ничто не мешает отредактировать Makefile вручную.

После этого можно, прочитав приложенную к пакету документацию, запустить программу.

Часто этими тремя этапами процедура установки и исчерпывается, но не менее часто неприятности начинаются уже на этапе конфигурирования: сценарий configure обнаруживает, что необходимая для этого пакета библиотека у вас не установлена. Что ж, найдите и установите ее и снова запустите сценарий configure. Он сообщит о нехватке чего-нибудь другого... но при достаточном терпении, времени и дешевом Интернете эти проблемы решаются.

На этапе компиляции и сборки можно столкнуться с тем, что нужные заголовочные файлы и библиотеки называются по-другому или расположены в другом месте, чем ожидал разработчик. Придется разбираться в сообщениях компилятора и утилиты ***make***, подсовывать вместо недостающих файлов символические ссылки на имеющиеся и выполнять другие нетривиальные действия, помогающие короче познакомиться с вашей операционной системой.

7.5.2. Установка из бинарных пакетов

Как это делается и что для этого нужно

Как ни гибок способ установки приложений из исходных текстов, позволяющий установить программу, созданную для другого дистрибутива, и настроить ее под конкретную системную среду, вчерашние пользователи Windows тоскуют по простоте setup.exe. Хорошим компромиссом считается распространение приложений в виде заранее собранных на определенной платформе бинарных пакетов.

Пакет содержит исполняемые файлы и библиотеки, подлежащие установке, а также разную служебную информацию об этом пакете: какие пакеты необходимы для его работы (зависимости), с какими пакетами он конфликтует, какие действия следует выполнить при его установке, список файлов, сведения о разработчике.

В мире Linux известны два формата бинарных пакетов: RPM от компании Red Hat, используемый не только в клонах Red Hat, но и в других популярных дистрибутивах: Mandrake, SuSE, ASPLinux, ALTLinux, Black Cat, и DEB, разработанный для Debian Linux и применяемый в его потомках: Knoppix, Corel Linux, Lindows. Пакеты дистрибутива Slackware — это просто сжатые архивы .tgz, не поддерживающие зависимостей. От архивов исходных текстов они отличаются только тем, что в них находятся заранее скомпилированные программы.

Набор утилит для установки, конфигурирования, удаления и ведения базы пакетов определенного формата называется системой управления пакетами. Наиболее распространены системы:

- RPM — менеджер пакетов формата RPM;
- DPKG — система управления пакетами DEB;
- APT — менеджер пакетов, поддерживающий автоматическое разрешение зависимостей, разработанный для Debian и заимствованный RPM-дистрибутивами.

На сегодняшний день самым распространенным в бывшем СССР средством управления пакетами является RPM, на котором я остановлюсь подробнее.

Менеджер пакетов RPM

Первоначально название программы RPM расшифровывалось как Red Hat Package Manager, но в соответствии с соглашением GNU о рекурсивном именовании (GNU's Not Unix) сейчас оно читается как RPM Package Manager. Это открытая пакетная система, позволяющая создавать пакеты из исходного и двоичного кода, так что двоичные файлы легко установить и сопровождать, а исходный код легко собрать. Система также поддерживает базу данных обо всех установленных пакетах и входящих в них файлах.

Пакеты формата RPM — это файлы с расширением .rpm. В имени файла обычно присутствуют название и версия приложения, выпуск пакета и платформа, для которой он собран. Например, для пакета `software-3.0-2.i386.rpm`: 3.0 — это версия программы software, 2 — выпуск пакета, i386 — платформа Intel 386. Обратите внимание на разницу между версией программы и выпуском пакета: номер версии назначается автором программы и характеризует ее саму, а номер выпуска — сборщиком пакета и характеризует этот пакет. В некоторых случаях,

даже если программное обеспечение не изменилось, бывает необходимо его переупаковать.

Самыми «универсальными» пакетами являются пакеты, рассчитанные на архитектуру Intel 386. Собранная таким образом программа должна работать на любом процессоре Intel, начиная с 80386DX (или совместимого с ним). А вот если у вас процессор 80486, пакет, рассчитанный для работы с архитектурой 80586 (Pentium), скорее всего, не установится в вашей системе. Обычно для процессоров архитектуры CISC (с набором команд x86) используются следующие обозначения:

- i386 — Intel 80386DX;
- i586 — Intel Pentium (MMX), AMD K5 (K6);
- i686 — Intel PPro, Celeron, PII, PIII, PIV.

Узнать о своем процессоре можно по команде `uname -a`.

В простейшем случае команда установки пакета выглядит так:

```
$ rpm -i <пакет>.rpm
```

Установку можно производить не только с локального диска, но и по протоколу FTP:

```
$ rpm -i ftp://somehost.domain/pub/package.rpm
```

Перед установкой пакета менеджер **rpm** проверит его зависимости, то есть другие пакеты, которые необходимы новой программе или, наоборот, конфликтуют с ней. Если установлены все нужные программе пакеты и ни с одним из установленных она не конфликтует, менеджер **rpm** установит программу, в противном случае сообщит вам о проблеме. Если нужен дополнительный пакет, просто установите его. А вот если программа конфликтует с уже установленным пакетом, то вам нужно будет выбрать, какой пакет вам больше нужен: уже установленный или новый.

Для удаления пакета используется команда:

```
$ rpm -e <пакет>
```

При удалении программы менеджер пакетов тоже проверяет зависимости между пакетами. Если удаляемый пакет нужен каким-нибудь другим пакетам, удалить его вы не сможете.

При установке программы я рекомендую указывать два дополнительных ключа: **-h** и **-v**. Первый требует показывать индикатор выполнения в виде строки, заполняющейся символами #, а второй выводит дополнительные сообщения.

Для пропуска проверки зависимостей нужно использовать ключ **--nodeps**. Например, у вас установлен агент отправки почты (MTA — *Mail Transfer Agent*) **postfix**, а вы хотите установить программу того же назначения

sendmail, которая с ним конфликтует. Просто так удалить пакет **postfix** вы не сможете, потому что он нужен многим почтовым клиентам. Удаляйте его командой:

```
$ rpm -e --nodeps postfix
```

После такого удаления нормальная работа других программ, использующих MTA, невозможна, поэтому вам сразу же нужно установить программу **sendmail** или другой агент MTA.

Ключ **–U** служит для обновления программ. Я рекомендую использовать его и при установке программ, потому что если устанавливаемый пакет у вас уже стоял, то будет произведено его обновление, а если нет, то будет просто установлен новый пакет:

```
$ rpm -Uhv <пакет>
```

Просмотреть все установленные пакеты можно с помощью команды:

```
$ rpm -qa | less
```

Если вам требуется узнать, установлен ли определенный пакет, выполните команду:

```
$ rpm -qa | grep <пакет>
```

Просмотреть общую информацию о пакете можно с помощью команды:

```
$ rpm -qi <пакет>
```

а информацию о файлах, которые установит этот пакет:

```
$ rpm -ql <пакет>
```

Чтобы узнать, какому пакету принадлежит некоторый файл, выполните команду:

```
$ rpm -qf <файл>
```

Графические менеджеры пакетов

Менеджер пакетов **rpm** является мощным средством для произведения операций над пакетами — создание, установка, обновление, удаление. Однако интерфейс командной строки нравится далеко не всякому начинающему администратору.

Существуют и графические (под X Window) реализации менеджера пакетов — например, для оконной среды KDE разработан **kpackage**, для GNOME — **gnorpm**, аналогичный по своим функциям. Какую из этих программ использовать — дело вкуса и привычки (я вообще обхожусь одним **rpm**).

Скажу несколько слов о программе **kpackage** (рис. 7.7).

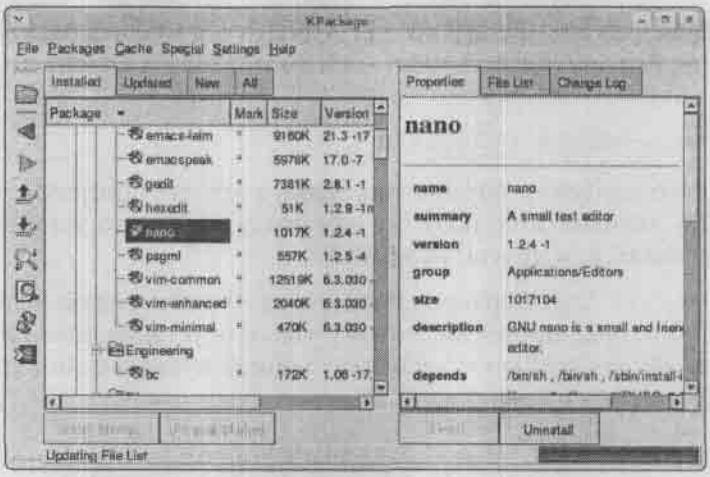


Рис. 7.7. *Просмотр установленных пакетов*

В функции программы **kpackage** входит:

1. Установка и удаление пакетов;
2. Получение сведений о пакете;
3. Проверка зависимостей пакета;
4. Поиск файлов и пакетов в базе RPM.

Вы можете установить пакет со своего жесткого диска, с инсталляционного компакт-диска или по протоколу FTP. Для установки пакета выберите в меню команду **File → Open** и введите путь (или URL) к каталогу с пакетами. Открывать подкаталоги можно и в окне выбора пакетов (рис. 7.8). Выбрав пакет, нажмите OK и в появившемся окне установки (рис. 7.9) закажите режим установки. Поставьте флажок **Test**, если вы хотите только проверить зависимости пакета, не устанавливая его.

Для поиска установленных пакетов и входящих в них файлов служат команды **File → Find Package** и **File → Find File**.

Apt: Debian-совместимый менеджер пакетов

Система управления пакетами программного обеспечения APT была разработана для Debian Linux, но впоследствии заимствована многими Red Hat-совместимыми дистрибутивами. В сам Red Hat и его потомки (Fedora Core) эта система не включена, но включена, например, в состав ALT Linux, и ее можно скачать из репозитория Сизиф `http://sisyphus.ru/srpm/apt/get`.

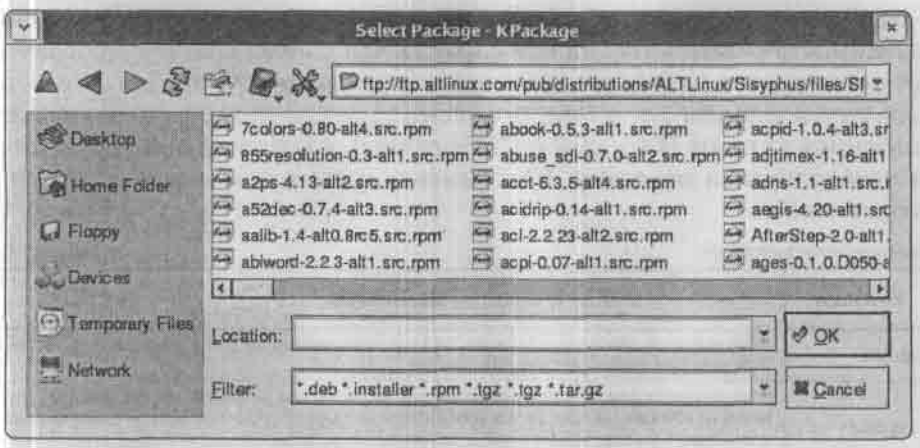


Рис. 7.8. *Выбор пакета для установки*

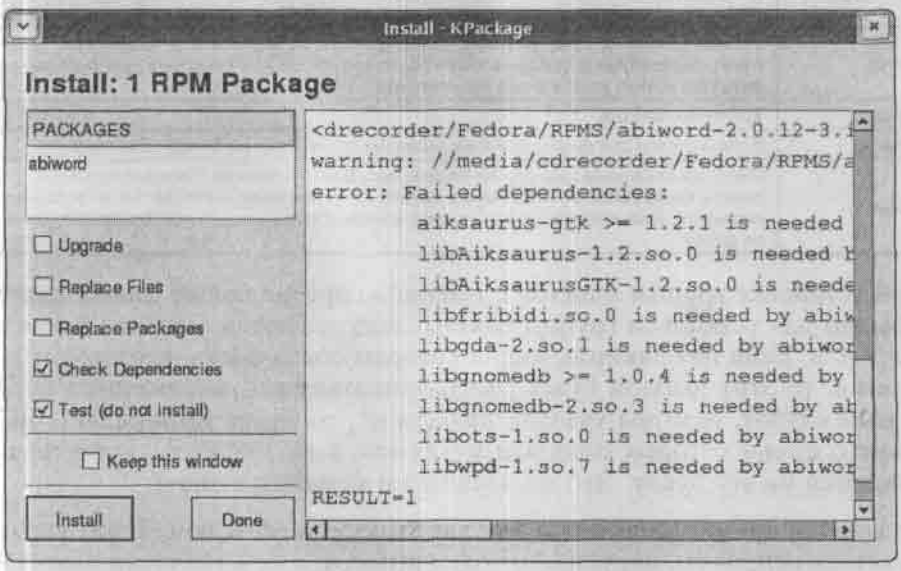


Рис. 7.9. *Проверка зависимостей пакета*

Для управления пакетами используется программа **apt-get**. Формат ее вызова:

```
$ apt-get [ключи] [команды] [пакеты]
```

Самые полезные команды перечислены в таблице 7.4.

В отличие от системы **rpm**, которая только докладывала о неразрешенных зависимостях, предоставляя вам справляться с ними самому, программа **apt-get** пытается разрешить зависимости самостоятельно. Для этого она пользуется файлом `/etc/apt/sources.list`, в котором перечислены источники пакетов (каталоги и FTP-архивы), к которым она обращается за необходимыми пакетами. Раскомментируйте в нем нужные строки и добавьте свои.

Команды программы apt *Таблица 7.4*

Команда	Назначение
update	Используется для синхронизации файлов описаний пакетов с их источником, который указан в файле /etc/apt/sources.list
upgrade	Используется для обновления установленного пакета до новейшей версии, доступной в источнике. Может также использоваться для обновления всех установленных в системе пакетов. Новые пакеты при этом не устанавливаются. Перед этой командой обязательно должна быть выполнена команда update
dist-upgrade	Более «интеллектуальная» версия команды upgrade. Кроме установки новых версий пакетов, она также проверит изменившиеся зависимости между новыми версиями пакетов и попытается разрешить конфликты в пользу более важных пакетов
install	Установка пакета. Если источник, из которого вы собираетесь устанавливать этот пакет, перечислен в файле источников, то в качестве имени пакета нужно указывать только имя упакованной программы
remove	Удаление пакетов
check	Используется для диагностики нарушенных зависимостей между пакетами
clean	Очищает локальное хранилище полученных файлов пакетов. Перед установкой пакеты копируются из источника в локальное хранилище, а оттуда потом устанавливаются. Пользуйтесь этой командой время от времени для освобождения места на диске

При установке группы пакетов с помощью **apt-get** будьте внимательны. Обычно для установки группы пакетов используются символы шаблона «?» и «*». Если нет пакетов, имена которых совпадают с указанным шаблоном, то этот шаблон будет рассматриваться как выражение POSIX. В этом случае, если вы указали шаблон a*, то будут установлены ВСЕ пакеты, имена которых содержат букву «а», а не только те, которые начинаются на эту букву. Это же касается и команды **remove**.

Из ключей **apt-get** полезными для вас будут **-f** и **-d.** Ключ **–f** требует попытаться исправить нарушенные зависимости, а при указании ключа **–d** пакеты только скачиваются из источника, но не устанавливаются. Ключ **--no-upgrade**, указанный при установке группы пакетов, запрещает обновлять те из них, что уже установлены. Еще одна полезная, но опасная возможность — ключ **--force-yes**, принуждающий программу не задавать вопросов, выполняя потенциально разрушительные действия. Иногда она действительно необходима.

Для установки пакета не по сети, а с дистрибутивного компакт-диска предназначена команда **apt-cdrom**.

7.5.3. Установка из пакетов, содержащих исходный код

Иногда в пакетах RPM находятся не откомпилированные версии программ, а их исходный код. Признаком этого является слово «src» вместо названия архитектуры. Для установки такого пакета введите:

```
$ rpm -iv <пакет>.src.rpm
```

Менеджер пакетов распакует исходные тексты в каталог Red Hat: по умолчанию это `/usr/src/redhat`, но вы можете установить другой каталог директивой topdir в конфигурационном файле `/etc/rpmrc`. В подкаталог SOURCES будут распакованы исходные тексты и заплатки (патчи) к ним, в подкаталог SPECS — spec-файлы, содержащие инструкции по прикладыванию заплаток и последующей сборке. Чтобы собрать программу, выполните команды:

```
$ cd /usr/src/redhat/SPECS
$ rpm -bp <пакет>.spec
```

Явление патча (заплатки — мы же на русском языке говорим) очень распространено в мире открытого кода, поэтому я скажу о нем здесь. Допустим, кто-то нашел и исправил ошибку в каком-нибудь известном пакете. Исправление может заключаться в двух строках кода, так что же — выкладывать в общий доступ исправленные исходники целиком? Нет, он распространяет заплатку, которую желающие могут приложить сами. Заплатка представляет собой текстовый файл, содержащий список отличий исправленного кода от исходного.

Такой список в стандартном виде генерирует утилита **diff** с ключами **-uNr**. Чтобы его приложить, нужно перейти в корневой каталог дерева исходного кода пакета и выполнить команду

```
$ patch -p0 < <файл_заплатка>
                    # < – это знак перенаправления ввода
```

Ключ –p указывает, сколько отделенных «/» частей нужно отщипнуть от путей к файлам, подлежащим исправлению (вдруг каталог пакета у автора заплатки назывался иначе?).

Приложив заплатку, выполните обычные команды сборки:

```
$ ./configure
$ make
$ make install
```

7.6. Клонирование и восстановление системы

Клонирование — это создание точной (побитной) копии исходного носителя. Носителем в нашем случае будет корневая файловая система Linux. Клонированная копия называется образом.

Если вам нужно установить дистрибутив на несколько компьютеров одинаковой конфигурации (например, вы администрируете интернет-зал), то целесообразно установить и настроить его на одной машине, а на другие — склонировать. Для домашней системы из одного компьютера клонирование тоже имеет смысл: если что-то вдруг «слетит», то вы сможете быстро восстановить исходное состояние системы простым развертыванием образа. На развертывание образа нужно намного меньше времени, чем на установку и настройку системы.

Перезагрузитесь в однопользовательском режиме (о режимах и вариантах загрузки сказано в п.9.1.1). Введите команду **mount**, чтобы узнать, какой раздел содержит корневую файловую систему:

```
/dev/hda1 on / type ext3 (rw,noatime)
none on /proc type proc (rw)
none on /sys type sysfs (rw)
```

Корневая файловая система расположена на разделе `/dev/hda1`. Образ именно этого раздела мы будем сохранять на другом носителе. Носителем может быть другой жесткий диск — я буду использовать внешний USB-винчестер. Для его подключения должен быть загружен модуль `usb_storage`. Создадим каталог `/mnt/image` и примонтируем к нему USB-винчестер:

```
# modprobe usb_storage
# mkdir /mnt/image
# mount /dev/sda1 /mnt/image
```

Все, что осталось сделать, — это перемонтировать корневую файловую систему в режиме «только чтение» и создать образ:

```
# sync
# mount -o remount,ro /
# dd if=/dev/hda1 of=/mnt/image/image.bin
```

Подробно описывать утилиту **dd** не стану — это очень хорошо сделано в справочной системе. Скажу только, что кроме **dd** существует еще и утилита **dd_rescue**, которая при клонировании пропускает плохие секторы и делает максимально возможную копию файловой системы. Для меня, да и для вас использование этой утилиты неактуально — ведь винчестеры меняют минимум один раз в год, поэтому не думаю, что у вас будет «битый» винчестер.

Теперь рассмотрим, как можно восстановить систему. Для восстановления системы вам понадобится загрузочный компакт-диск Linux. Подойдет первый диск любого дистрибутива, даже несовместимого с вашим. Если вы используете не LiveCD, а простой загрузочный диск, для перехода на текстовую консоль нажмите Ctrl + Alt + F2. Подмонтируем наш внешний винчестер:

```
# mkdir /image
# modprobe usb_storage
# mount /dev/sda1 /image
```

Теперь на новом винчестере нужно создать разделы. Это можно сделать с помощью утилиты **fdisk** или с помощью программы установки — как вам удобнее. Если же вы восстанавливаете систему после сбоя, ничего создавать не нужно — все уже создано. Не забудьте только создать раздел подкачки (тип раздела `/dev/hda2` должен быть Linux swap):

```
# mkswap /dev/hda2
```

Теперь развернем образ (обратите внимание на параметры **if** и **of** программы **dd** — на этот раз их аргументы поменялись местами):

```
# dd if=/image/image.bin of=/dev/hda1
```

Сейчас нам нужно изменить корневую файловую систему, чтобы попасть «внутрь» развернутого образа:

```
# mkdir /install
# mount /dev/hda1 /install
# chroot /install /bin/bash
```

Теперь корнем стал каталог `/install`, к которому примонтирован новый винчестер, а в качестве командной оболочки используется `/bin/bash`. Все, что нам осталось сделать — это перезаписать загрузчик. Если у вас LILO, введите команду:

```
# lilo
```

А если GRUB:

```
# grub-install /dev/hda
```

Теперь перезагрузите компьютер (reboot) — ваша система успешно восстановлена после клонирования.

Глава 8

ЯЗЫК КОМАНДНОГО ИНТЕРПРЕТАТОРА

- ПАРАМЕТРЫ
- ПОДСТАНОВКИ
- МАССИВЫ
- УПРАВЛЯЮЩИЕ СТРУКТУРЫ
- УСЛОВНАЯ ПОДСТАНОВКА ПАРАМЕТРОВ
- ФУНКЦИИ
- ОБРАБОТКА СИГНАЛОВ И ПРОТОКОЛИРОВАНИЕ

Для автоматизации часто выполняемых работ по обслуживанию системы вам понадобится объединять команды в сценарии. С простейшими сценариями, обеспечивающими последовательное выполнение перечисленных команд, вы уже познакомились в гл. 3. Уточню только, как система обрабатывает сценарии. Она ищет в первой строке файла сценария выражение

```
#!/абсолютный/путь/к/программе/интерпретатору/сценария
```

и передает сценарий ей на обработку. Как правило, эта программа — либо одна из установленных в системе оболочек (например, `#!/bin/tcsh`), либо один из интерпретирующих языков (`#!/usr/bin/perl`), либо ваш собственный интерпретатор: `#!/usr/bin/my_program`. Если сценарий предназначен для обработки оболочкой **sh**, то ее имя указывать необязательно. Между символами # и ! не должно быть пробелов.

Чтобы непосредственно запустить файл сценария на выполнение, вы должны иметь полномочия на исполнение этого файла. Если он доступен вам только для чтения, выполнить его можно командой

```
$ /программа/обработчик -f <имя_файла>
```

Эта глава посвящена языку программирования, встроенному в оболочку **bash**.

Базовыми операциями этого языка служат все установленные в системе программы, утилиты и сценарии. Оболочка находит их, запускает, обеспечивает передачу им — аргументов, а результатов их работы — другим программам и пользователю, выполняет подстановку переменных и раскрытие шаблонов. Кроме того, оболочка содержит операторы цикла и условные операторы, в результате чего получается мощный язык программирования.

8.1. Параметры

Вместо переменных, как в обычных языках программирования, в **bash** используется понятие параметра. Именем параметра может быть: слово, состоящее из латинских букв, цифр и знаков подчеркивания, начинающееся с буквы; число; один из специальных символов: *, @, #, ?, $, !, 0, _ .

Тип всех параметров — строковый. Если параметру присвоено значение (хотя бы пустая строка), то говорят, что он задан или установлен. Чтобы сослаться на значение параметра, нужно поставить символ $ перед его именем:

```
$ echo $0
bash
```

Параметры делятся на:

- **переменные оболочки** — о них рассказано в п. 3.4.3;
- **позиционные параметры** — их имена представляют собой натуральное число, а значениями служат аргументы с соответствующими номерами, начиная с 1, переданные сценарию или самой оболочке при их запуске;
- **специальные параметры** — их именами служат перечисленные специальные символы, а назначение сведено в таблицу 8.1. Эти параметры — шаблоны, подстановку которых производит оболочка.

Специальные переменные *Таблица 8.1*

Название	Подстановка
$0	Имя выполняемого сценария (или, в интерактивном режиме, — самой оболочки)
$#	Количество позиционных параметров, переданных сценарию
$_	Последний аргумент предыдущей из выполнявшихся команд
$?	Код завершения последней выполненной команды (напоминаю, что значение 0 говорит об успешном завершении, другое — об ошибочном)
$$	Номер текущего процесса (PID)
$!	Номер (PID) последнего асинхронного процесса (команды, выполненной в фоновом режиме)
$*	Все позиционные параметры, слитые в одну строку: «$*» эквивалентно «$1x$2x...$n», где x — первый символ значения переменной IFS (internal field separator), по умолчанию — пробел
$@	Все позиционные параметры, подлежащие дальнейшему разбору: «$@» эквивалентно «$1» «$2» ...«$n»

Напоминаю, что специальные символы в значениях параметров нужно экранировать. Экранировать одиночный символ можно символом «\» (обратный слэш), несколько — апострофами или двойными кавычками. Разница в том, что внутри двойных кавычек выполняются подстановки значений: сравните вывод команд `echo '$0'` и `echo "$0"`.

Организовать диалоговый ввод значения переменной можно с помощью встроенной команды `read`:

```
echo -n "Продолжать? (y/n): "
read yesno
echo $yesno
```

Ключ **-n** команды `echo` не выводит символ новой строки в конце сообщения, то есть не переводит строку. Команда `read` читает значение, введенное пользователем с клавиатуры, и записывает его в переменную `yesno`. Последняя команда выводит только что введенное значение.

Одной командой `read` можно прочитать несколько переменных:

```
read name middle lastname
```

Пользователь должен ввести значения переменных, разделяя их пробелами, и для окончания ввода нажать <Ввод>. Если введено меньше значений, чем нужно, оставшимся переменным будет присвоена пустая строка. Если больше, то весь остаток строки будет присвоен последней из перечисленных переменных.

8.2. Подстановки

Кроме подстановки обычных переменных (п.3.4.4) и раскрытия шаблонов имен файлов (п.3.4.5), оболочка **bash** умеет выполнять еще и такие подстановки:

- **Подстановка тильды**:
  - ~ заменяется на имя домашнего каталога пользователя, запустившего сценарий;
  - ~+ заменяется на путь к текущему каталогу;
  - ~- заменяется на путь к предыдущему каталогу.
- **Раскрытие скобок**:
  `строка1{строка2,строка3…}строкаN` заменяется на: `строка1строка2строкаN строка1строка3строкаN …`
  Пробелов внутри скобок быть не должно. Эта функция полезна, когда нужно применить одну команду к нескольким файлам, не подходящим под общий шаблон:
  ```
  $ cat /home/den/linuxbook/{intro,param,subst} > glava8
  ```
- **Подстановка арифметических выражений**:
  `$((выражение))` или `$[выражение]` — эквивалентные формы записи. Внутри выражения выполняются подстановки параметров. Приоритет арифметических операций — обычный, подробнее см. `man bash`, секция ARITHMETIC EVALUATION.

Например, количество часов, прошедшее с момента запуски оболочки, можно подсчитать командой:

```
$ echo $(( $SECONDS/3600 ))
```

8.3. Массивы

Интерпретатор **bash** поддерживает одномерные массивы с неограниченным числом элементов. В других оболочках существуют определенные ограничения на массивы, например, в **ksh** максимальное число элементов массива ограничено 1024 элементами.

Нумерация элементов начинается с нуля. Тип элементов массива, как и тип параметров, строковый. Присвоить значение элементу массива можно с помощью такой конструкции:

`Имя_массива[индекс]=значение`, например:

```
$ weekday[0]=Понедельник
$ weekday[4]=Пятница
```

Обратиться к значению элемента массива можно следующим образом:

```
${имя_массива[индекс]}
```

Например, вывести значение первого элемента массива можно так:

```
$ echo ${weekday[0]}
```

Обратиться ко всем элементам массива сразу можно с помощью одного из выражений:

`${имя_массива[*]}` или `${имя_массива[@]}`, например:

```
$echo ${weekday[*]}
Понедельник Пятница
```

Второе выражение нужно использовать, если значение хотя бы одного элемента массива может содержать пробелы.

Можно инициализировать массив целиком: команда

```
$ weekday=(Пн Вт Ср Чт Пт Сб Вс)
```

эквивалентна списку

```
$ weekday[0]=Пн; weekday[1]=Вт; … weekday[6]=Вс
```

Эти способы инициализации массивов могут применяться как в **bash**, так и в **ksh** и в других командных оболочках. А следующий способ работает только в **bash**:

```
$ holidays= ([0]=Sunday [6]=Saturday)
```

8.4. Управляющие структуры

Напоминаю (п.3.4.7), что список команд — это одиночная команда, конвейер или последовательность команд/конвейеров, разделенных одним из операторов: ; && ||, завершенная точкой с запятой. Не забывайте ставить точку с запятой даже после одиночной команды, терять ее — типичная ошибка начинающих программистов.

В синтаксис следующих команд квадратные скобки не входят: они ограничивают необязательные выражения.

8.4.1. Условные операторы

Оператор варианта case

Синтаксис:

```
case значение in
   [шаблон1) список1;;
   шаблон2 | шаблон3) список2;; ]
esac
```

Ищется первый шаблон, совпадающий со значением. Если он найден, то выполняется соответствующий ему список команд, завершенный двумя символами «;». Шаблон и список разделяются символом «)». Одному списку команд может соответствовать несколько шаблонов, тогда они разделяются символом «|».

В шаблонах могут использоваться метасимволы *, ? и [ ] (о подстановке метасимволов сказано в п.3.4.5). С их помощью можно организовать инструкцию, действующую как **default** в операторе **switch** языка С.

Порядок сравнения шаблонов со значением не определен. Первое совпадение прекращает дальнейшее сравнение и приводит к выполнению соответствующего списка команд и выходу из структуры **case** — поведение, больше похожее на поведение оператора case в языке Паскаль, чем на switch в языке С.

Листинг 8.1. Пример использования оператора case

```
echo "Ошибка. Кому переслать протокол?"
echo "Начальнику:   b"
echo "Коллегам:   c"
echo "Сам разберусь:   any key"
read answer
case $answer in
 b|B) mail -s "error log" boss < error.log;;
```

```
  c|C) mail -s 'Help!!! error log' -c ivan den < error.log;;
  *) echo "OK. Exiting"; exit;;
esac
```

Обратите внимание на апострофы вместо кавычек во втором списке: они экранируют подстроку «!!» от подстановки предыдущей команды.

Условный оператор if

Синтаксис:

```
if список1 then
 список2
[elif список3 then
 список4]
[else
 список5]
fi
```

Эта конструкция работает так же, как в других языках программирования. Сначала выполняются команды из `списка1`. Если этот список выполнен успешно, то есть с кодом завершения 0, то выполняется `список2`, если нет — то список, стоящий после очередного **elif**. При невозможности выполнить список команд, стоящий после очередного **then**, выполняется список, стоящий после **else**.

Можно использовать сокращенный вариант, только **if-then-fi**:

```
$ if [ $? -ne 0 ]; then echo "Ошибка. Смотри протокол"; fi;
```

Оператор test и условные выражения

В вышеприведенной команде вместо анализа кода завершения списка использована проверка условия. Две формы такой проверки эквивалентны: встроенная команда **test** и [условие]. Например, для проверки существования файла можно написать

```
test -e <файл>
```

или

```
[ -e <файл> ]
```

Если вместо слова **test** используются квадратные скобки, они обязательно должны быть отделены от аргументов пробелом, потому что на самом деле «[» — это название команды, а «]» — обязательный последний аргумент этой команды.

В случае истинности условия команда **test** возвращает код успешного завершения, то есть 0; в случае ложности — код ошибки 1 (не спутайте с обычными языками программирования, где 1 — другое имя для true !).

Команда **test** может проверять и строку на пустоту: непустая строка считается выполнением условия и приводит к коду завершения 0. Пример:

```
$ test $USER; echo $?
0
$ test $VAR_not_set_yet; echo $?
1
```

Условные выражения можно комбинировать с помощью обычных логических операций:

- `!(выражение)` — отрицание;
- `выражение1 -a выражение2` — логическое И (*and*);
- `выражение1 -o выражение2` — логическое ИЛИ (*or*).

Элементарные условные выражения перечислены в таблицах 8.2 и 8.3. Полный список их можно получить по команде `help test`.

Основные условные выражения для файлов *Таблица 8.2*

Выражение	Истинно, если
-d файл	файл существует и является каталогом
-e файл	файл существует
-f файл	файл существует и является обычным файлом
-L файл	файл существует и является символической ссылкой
-r файл	файл существует и доступен для чтения
-w файл	файл существует и доступен для записи
-x файл	файл существует и является исполняемым
-s файл	файл существует и его размер больше 0
-N файл	файл существует и изменился со времени последнего чтения
файл1 -nt файл2	время модификации файла1 позже (newer than), чем файла2
файл1 -ot файл2	время модификации файла1 раньше (older than), чем файла2
файл1 -ef файл2	файл1 — это жесткая ссылка на файл2

Элементарные условные выражения для сравнения строк *Таблица 8.3*

Выражение	Истинно, если
-z строка	длина строки равна 0
-n строка	длина строки не равна 0
стр1 == стр2	строки совпадают
стр1 !== стр2	строки не совпадают
стр1 < стр2	строка1 предшествует строке2 в лексикографическом порядке. Алфавит соответствует текущей локали
стр1 > стр2	строка1 следует за строкой2 в лексикографическом порядке

Арифметическое условное выражение имеет формат `arg1 OP arg2`, где `arg1` и `arg2` — целые числа, а `OP` — одна из операций:

- **-eq** — равно;
- **-ne** — не равно;
- **-lt** — меньше;
- **-le** — меньше или равно;
- **-gt** — больше;
- **-ge** — больше или равно.

Таким образом, можно переписать предыдущий пример с использованием оператора **if**:

Листинг 8.2. Пример использования оператора if

```
echo "Ошибка. Кому переслать протокол?"
echo "Начальнику:   b"
echo "Коллегам:   c"
echo "Сам разберусь:  any key"
read answer
if [ "$answer" == "b" -o "$answer" == "B" ]; then
 mail -s "error log" boss < error.log;
elif [ "$answer" == "c" -o "$answer" == "C" ]; then
 mail -s 'Help!!! error log' -c ivan den < error.log;
else
 echo "OK. Exiting"; exit;
fi
```

8.4.2. Операторы цикла

Командные интерпретаторы **bash** и **ksh** поддерживают циклы типа for, while, until и select, а интерпретатор **sh** — только циклы for и while.

Оператор цикла с перечислением for

Синтаксис:

```
for переменная [in шаблон]
do
 список
done
```

В результате подстановки шаблона получается список слов. Переменная получает значение первого слова из этого списка, и выполняется список команд, стоящий между **do** и **done**. Затем переменная получает значение

очередного слова из списка слов, и снова выполняется список команд. Повторение прекращается по исчерпании слов в списке. Отсутствие конструкции [**in** шаблон] эквивалентно записи **in $@**.

Список слов можно сформировать и вручную:

```
$ for day in Mon Tue Wed Thu Fri; do echo "План работы на
$day:"; cat $day.plan; done
```

Еще раз напомню, что любой список в **bash** нужно заканчивать точкой с запятой.

Пример использования цикла с перечислением: допустим, у вас не хочет собираться некий программный пакет — он рассчитывал, что имена заголовочных файлов в некотором каталоге имеют расширение **.h**, а у вас они такого расширения не имеют (установлена другая версия библиотеки). При этом содержание этих файлов его устраивает. Так создадим символические ссылки, чтобы он нашел заголовки по знакомому имени:

```
$ cd /путь/к/каталогу/include
$ for name in *; do ln -s $name $name.h; done
```

Оператор цикла с условием while

Синтаксис:

```
while список1
do
 список2
done
```

Оператор выполняет `список1` и в случае его успешного завершения (нулевого кода возврата) — `список2`. Процедура повторяется до тех пор, пока результат выполнения `списка1` не станет ненулевым. Например:

```
$ i=10
$ while [ $i -gt 0 ]; do
> echo $i...
> i=$(($i-1))
>done; echo 'BANG!!!'
$
```

Примечание

Тот же самый обратный отсчет можно реализовать и с помощью цикла for, если у вас установлена утилита **seq**, печатающая последовательность (*sequence*) чисел с заданным шагом:

```
$ for i in `seq 10 -1 0`; do echo $i...; done; echo 'BANG!!!'
```

Оператор цикла с инверсным условием until

Синтаксис:

```
until список1
do
 список2
done
```

Оператор выполняет `список1` и, если он выполнен неуспешно (код возврата ненулевой), то выполняет `список2`. Процедура повторяется до тех пор, пока результат выполнения `списка1` не станет нулевым.

Оператор цикла с выбором select

Синтаксис:

```
select переменная [in шаблон]
do
 список
done
```

В результате подстановки шаблона получается список слов. К этим словам оператор добавляет порядковые номера и выводит весь набор в стандартный поток ошибок (*stderr*). Если шаблон опущен, то вместо него используется список позиционных параметров $@. После этого оболочка выводит приглашение и считывает строку из стандартного потока ввода (*stdin*). Если строка содержит номер, соответствующий какому-либо слову из списка, то переменная получает это слово в качестве значения. Если в строке подходящего номера нет, то значением переменной становится пустая строка. После этого выполняется список команд, и процедура повторяется до тех пор, пока в строке ввода не встретится символ конца файла (введите Ctrl+D) или пока в списке команд не встретится команда **break** или **return**.

Этот оператор полезен для создания нумерованных пунктов меню. Например, у меня в каталоге `~/temp` есть три файла: `proto.txt`, `file.txt` и `README`. В листинге 8.3. приведен фрагмент сценария, позволяющего быстро просмотреть любой из них.

Листинг 8.3. Пример использования оператора select

```
echo "Выберите файл для просмотра:"
select file in ~/temp/* Quit;
do
if [ -f $file ]; then cat $file;
else break;
fi
done
```

Запустив этот сценарий, я увижу на экране:

```
Выберите файл для просмотра:
1) /home/den/temp/file.txt
2) /home/den/temp/proto.txt
2) /home/den/temp/README
4) Quit
#?
```

Последняя строка — это приглашение, устанавливаемое переменной окружения PS3.

8.5. Условная подстановка параметров

Условная подстановка позволяет проверить, установлен ли определенный параметр, или использовать вместо его значения другое. Значение самого параметра при этом не изменяется. Допустимые виды условных подстановок перечислены в таблице 8.4.

Условная подстановка *Таблица 8.4*

Конструкция	Выполняет подстановку
${параметр:-строка}	Значение по умолчанию. Если параметр имеет непустое значение, то подставляется оно, иначе — указанная строка
${параметр:=строка}	Присваивание значения по умолчанию. Если параметр не имеет непустого значения, то ему присваивается «строка», после чего значение подставляется. Конструкция допустима только для переменных оболочки
${параметр:?сообщение}	Ошибка, если пусто. Если параметр не имеет непустого значения, то выводится указанное сообщение. Сообщение можно опустить, тогда будет выведено стандартное сообщение
${параметр:+строка}	Дополнительное значение. Если параметр имеет непустое значение, подставляется «строка», иначе — пустая строка
${параметр#шаблон}	Подставляется значение параметра, в котором из головной части удален наименьший фрагмент, удовлетворяющий шаблону
${параметр##шаблон}	Подставляется значение параметра, в котором из головной части удален наибольший фрагмент, удовлетворяющий шаблону
${параметр%шаблон}	Подставляется значение параметра, в котором из хвостовой части удален наименьший фрагмент, удовлетворяющий шаблону
${параметр%%шаблон}	Подставляется значение параметра, в котором из хвостовой части удален наибольший фрагмент, удовлетворяющий шаблону
${#параметр}	Если параметр есть * или @, подставляется количество позиционных параметров, иначе — длина значения параметра в байтах

Например, команда `echo ${0:+"Моя любимая оболочка"}` заменит непустое значение параметра $0, равное «bash», на указанное дополнительное значение. Команда `${1:?"Не хватает параметра"}` выведет

сообщение, если сценарий, в котором она встречается, будет случайно запущен без аргументов. Правильным подходом к написанию сценариев было бы выводить не такое малоинформативное сообщение, а краткую справку об использовании этого сценария, подобную тому, что можно увидеть, запустив почти любую команду с ключом --usage.

Подстановки # и % полезны, например, тогда, когда нужно «выкусить» из полного пути к файлу собственно его имя или, наоборот, родительский каталог:

```
$ path=`which twm`; echo $path
/usr/X11R6/bin/twm
$ echo ${path##*/}
twm
$ echo ${path%/*}
/usr/X11R6/bin
$
```

8.6. Функции

Оператор определения функции имеет следующий синтаксис:

```
[function] имя()
{
 список
}
```

Определять функцию можно в любом месте сценария, но вызов ее должен осуществляться строго после описания. Вызывается функция подобно любой команде — по имени. Переданные ей аргументы в теле функции рассматриваются как позиционные параметры, причем в вызывающем сценарии значения позиционных параметров не меняются. Значение позиционного параметра 0 — это имя функции.

Вызов функции не порождает нового процесса, поэтому ей видны локальные переменные, установленные вызывающим сценарием или оболочкой до ее вызова.

Ошибка при выполнении функции приводит к немедленному ее завершению с ненулевым кодом возврата. Если вы хотите передать в коде возврата собственное значение, пользуйтесь оператором `return <число>`. Отсутствие числа или всего оператора **return** означает возврат нулевого значения. Код возврата функции помещается в переменную $? и доступен до выполнения следующей команды.

Если вы задумали функцию как «библиотечную» (вызываемую из разных сценариев и в связи с этим определенную в отдельном файле), то для того, чтобы определить ее в текущем процессе, нужно не запускать ее файл на выполнение, а прочитать его встроенной командой **source**.

8.7. Обработка сигналов и протоколирование

Обычно при завершении сеанса работы пользователя система посылает всем запущенным им процессам сигналы (п.3.3.2), которые приводят к прекращению этих процессов. Возможно, вам понадобится обеспечить своему сценарию возможность продолжать выполнение даже после отключения запустившего его пользователя. Тогда посланный сигнал придется перехватывать и обрабатывать собственными средствами сценария.

Перехватить сигнал можно с помощью встроенной команды **trap**. Формат ее следующий:

```
trap [-lp] [команда сигнал сигнал...]
```

Ключ **–l** выводит список имен и номеров сигналов, известных в ОС Linux. Ключ **–p** выводит список команд, связанных с каждым сигналом. Сигналы указываются по номерам или именам, приставку SIG можно опускать.

Команда — это та команда, которая будет выполнена оболочкой при получении сигнала (ваш собственный обработчик). Если вместо нее указать пустую строку, то перечисленные сигналы будут проигнорированы. Если вместо сигналов указать EXIT или 0 (фиктивный номер), то команда будет выполнена при завершении сеанса работы с оболочкой.

Чаще всего перехватываются сигналы:

01	SIGHUP	hangup, освобождение линии связи;
02	SIGINT	interrupt, прерывание;
03	SIGQUIT	quit, выход;
15	SIGTERM	terminate, программный сигнал завершения.

Чтобы игнорировать все эти сигналы, введите команду:

```
$ trap "" 1 2 3 15
```

Протоколировать работу собственного сценария можно двумя способами.

Первый состоит в использовании команды-фильтра **tee** (п.3.4.6). Название этой команды происходит от английского названия буквы T, и действие ее похоже на эту букву: она копирует данные из своего стандартного потока ввода и раздваивает их на стандартный поток вывода и поток в указанный файл:

```
$ LOGFILE=my_log
$ if [ "$LOGGING" == "true" ]; then
> my_script | tee $LOGFILE; else
> my_script;
> fi
$
```

Если вы собираетесь не вводить эти команды из командной строки, а включить их в сценарий my_script, то вызов сценария изнутри него самого должен выглядеть так:

```
exec $0
```

Встроенная команда **exec** заменяет текущий процесс (то есть ту дочернюю оболочку, в которой запущен сценарий) на выполняемую команду, и сценарий, завершившись, возвращает управление прямо родительской оболочке. Интерактивную оболочку (ту, с которой вы начинаете сессию) подменить таким образом нельзя.

Если команде **exec** не указан аргумент, но указано перенаправление ввода-вывода, то **exec** совершает это перенаправление, продолжая выполнение текущего сценария. Таким способом можно получить динамическое перенаправление:

```
$ tty
/dev/pts/2
$ echo "Вывожу строку на терминал"
Вывожу строку на терминал
$ exec > log
$ echo "Вывожу строку в файл"
$ echo "И эту в файл"
$ exec > /dev/pts/2
$ echo "А эту снова на терминал "
А эту снова на терминал
$
```

Второй способ заключается в использовании команды **script**, которая копирует в файл весь сеанс работы в текстовой консоли: ввод пользователя и вывод команд. Это должен быть в полном смысле слова сеанс работы в командной строке: полноэкранные команды, такие, как редактор **vi** и даже **man**, оставляют в файле протокола мусор. Если вы запускаете команду **script** вручную, то остановить протоколирование можно командой **exit**.

```
$ LOGFILE=my_log
$ if [ "$LOGGING" == "true" ]; then
> script my_script $LOGFILE; else
> my_script;
> fi
$
```

Вызов сценария изнутри него самого должен выглядеть так:

```
exec script $0 $LOGFILE
```

Начать знакомство с чужими сценариями вы можете с инициализационных файлов **bash** `/etc/bashrc` и `/etc/profile`. Команда «.» (точка) в оболочке ***sh*** и ее производных (**bash**, **ksh**), подобно команде **source**, означает чтение и выполнение команд из файла-аргумента этой команды в текущем процессе.

Глава 9

УПРАВЛЕНИЕ ПРОЦЕССАМИ

- КАК ЗАГРУЖАЕТСЯ LINUX
- КОМАНДЫ УПРАВЛЕНИЯ ПРОЦЕССАМИ
- ПРОТОКОЛИРОВАНИЕ СИСТЕМЫ
- ВЫПОЛНЕНИЕ ЗАДАНИЙ ПО РАСПИСАНИЮ

9.1. Как загружается Linux

9.1.1. Начальная загрузка: LILO и GRUB

Общие механизмы

Как известно, первая программа, которая выполняется после включения компьютера, — это BIOS. Она находит загрузочное устройство, считывает в память его первый (нулевой) сектор и передает на него управление. В этом секторе находится MBR (*Master Boot Record*) — главная загрузочная запись размером в 512 байт, в которой помещаются:

- первичный загрузчик;
- таблица разделов диска (*Partition Table*) размером в 64 байта, описывающая четыре первичных раздела: номера их первого и последнего цилиндров, тип файловой системы и признак активности раздела;
- «волшебное число» (0xAA55), предназначенное для проверки, служит ли данный сектор загрузочным.

Формат MBR стандартен для всех операционных систем, а содержание области, отведенной под первичный загрузчик, может различаться. Этот загрузчик очень мал, поэтому перед ним стоит всего одна задача: найти на диске и считать в память код загрузчика следующего этапа, разворачивающего уже саму операционную систему, и передать ему управление.

В ОС Windows 9x первичный загрузчик передает управление на *Boot Record* — первый сектор того первичного раздела, который отмечен как активный (*bootable* — такой может быть только один). В более сложных системах из MBR запускается диспетчер загрузки (**NTLoader** для Windows NT, **LILO** и **GRUB** — для Linux), позволяющий выбрать вариант загрузки и даже загружаемую ОС. Такую гибкость диспетчеру обеспечивает возможность не ограничиваться тем объемом данных, который помещается в MBR, а читать необходимые данные из файлов на диске. Достигается эта гибкость ценой зависимости от файловой системы: существуют файловые системы (например, XFS и ReiserFS с включенным режимом оптимизации дискового пространства), с которыми ОС Linux

может работать, но загружаться с них не может. Отдельный раздел /boot, о котором говорилось в п.1.2.2 в связи с «барьером 1024 цилиндра», необходим еще и поэтому: на нем должна быть создана файловая система ext2fs или ext3fs, а для всех остальных разделов файловые системы можно выбирать произвольно.

Загрузчик LILO

Стандартный загрузчик Linux — LILO (*LInux LOader*) — состоит из двух частей: первичного загрузчика LI и вторичного LO. LI располагается в MBR и только и умеет, что загружать LO, а тот уже передает управление ядру или вызывает другой первичный загрузчик (например, Windows 9x). LO находится в файле на диске (по умолчанию /boot/boot.b). О файловых системах LI не знает, поэтому карта размещения этого файла хранится в нем в виде «цилиндр/головка/сектор». Помещает ее туда утилита **/sbin/lilo**, которую нужно запускать после любого изменения LO или его конфигурационного файла /etc/lilo.conf.

У вторичного загрузчика LO есть собственная карта размещения файлов (по умолчанию /boot/map). По ней он ищет загружаемое ядро и образ виртуального диска, поэтому после любого изменения ядра или загружаемых модулей тоже обязательно запускать утилиту **lilo**.

Что такое виртуальный диск? Представьте себе загрузку Linux со SCSI-диска или другого устройства, драйвер которого не вкомпилирован в ядро, а подгружается в виде модуля. LILO сможет найти и прочитать с него файл образа ядра. Теперь ядру предстоит смонтировать корневую файловую систему. Чтобы сделать это, нужно подключить драйвер SCSI, а чтобы найти драйвер в /lib/modules, нужно смонтировать корневую файловую систему.

Похожая проблема возникает при первоначальной установке ОС Linux: для работы инсталлятора нужна файловая система со стандартными утилитами, а на диске ее еще нет. Обе проблемы решаются в Linux с помощью технологии initrd (*INITial Ram Disk*): вместе с ядром LILO загружает в память образ стартового диска, и ядро монтирует его как обычную файловую систему. В этой файловой системе находятся модули, необходимые для работы с нестандартными внешними устройствами и сетью, и утилиты для их подгрузки. Подключив модули, ядро отсоединяет виртуальный диск и монтирует настоящую корневую файловую систему.

Файл образа виртуального диска обычно называется /boot/initrd-<версия_ядра>. Если нестандартных устройств у вас нет или их драйверы встроены в ядро, то этот файл для загрузки не нужен.

Поведение LILO зависит от настроек в его конфигурационном файле /etc/lilo.conf. Ниже приведен пример такого файла. Символ #, как обычно, служит для комментариев.

Листинг 9.1. Примерный файл /etc/lilo.conf

```
# LILO version: 21.5
# Общий раздел
#
# использовать MBR первого жесткого диска первого
# контроллера IDE
boot=/dev/hda
#
# Карта LO
map=/boot/map
#
# Файл вторичного загрузчика
install=/boot/boot.b
#
# Режим для загрузочных дискет. У меня закомментирован.
# compact
#
# Режим VGA: normal - 80x25, ext - 80x50
vga=normal
#
# Раскладка клавиатуры
keytable=/boot/ru4.klt

# Диск поддерживает режим LBA (Large Block Access) —
# трансляцию
# физических адресов в логические так,
# чтобы число цилиндров не превышало
# понятных BIOS 1024. Другое значение этого
# параметра — linear. Не меняйте значения, выставленного
#инсталлятором, если вы не знаете точно, что делаете.
lba32
#
# Сообщение, которое выдается при загрузке
message=/boot/message
#
# Задержка 5 секунд (в других версиях LILO этот параметр
# называется delay)
timeout=50
#
# Вывести message и приглашение
# к выбору загружаемого ядра на
# timeout/10 секунд, после которых загружается ядро,
# выбранное по умолчанию.
```

```
# Если prompt не установлен, добиться приглашения
# можно, удерживая <Shift> во время загрузки.
prompt
#
# Цветовая схема
menu-scheme=wb:bw:wb:bw
#
# Ядро, загружаемое по умолчанию. Если не указано, то
# загружается первое в списке
default=Fedora
#
# Список вариантов загрузки, не более 16.
# В каждой секции варианта должна быть строка
# label. Это имя, которое вводится в ответ на приглашение
# LILO или является командой меню и служит для выбора
# загружаемого ядра или ОС.
#
image=/boot/vmlinuz-2.4.20   # ядро
 label=linux-initrd    # метка
 initrd=/boot/initrd-2.4.20.img
 root=/dev/hda6   # монтировать этот раздел как корневой
 read-only      # режим монтирования / на время загрузки
#
image=/boot/vmlinuz-2.6.9-1.667
 label=Fedora
 root=/dev/hda2
 read-only
#
image=/boot/vmlinuz
 label=failsafe
 root=/dev/hda6
 append=" mem=64M failsafe" # параметры, передаваемые ядру
read-only
#
other=/dev/hda1   # ОС – не Linux
 label=WindowsXP  # root не указывается
 table=/dev/hda   # где находится таблица разделов
```

Если вы определили секцию other=/dev/hda1, то в корневом каталоге раздела /dev/hda1 (диска C:) должен находиться вторичный загрузчик. У меня, например, на одном из компьютеров с многовариантной загрузкой там находится NTLoader (поскольку WindowsXP была установлена до Linux), и LILO успешно загружает WindowsXP.

Команда append позволяет передать ядру необходимые параметры. Ее формат:

```
append=" параметр1[=значение1][,значение2…]
[параметр2[=значение3][,значение4…]...]"
```

Значения разделяются запятой без пробелов, параметры разделяются пробелами. Например, у вас есть устройство CD-RW, которое вы до сих пор использовали как обыкновенный CD-ROM, подключив его как Secondary Slave, то есть `/dev/hdd`. Чтобы записывать компакт-диски под Linux, CD-RW должен быть устройством SCSI, значит, это устройство нужно эмулировать. Команда выглядит так:

```
append=" hdd=ide-scsi "
```

Не забудьте после каждого изменения конфигурационного файла запустить утилиту ***lilo***. Некоторые ее полезные ключи:

- **-V** — показать версию LILO;
- **-q** — показать текущую карту загрузки;
- **-t** — проверить `lilo.conf` на ошибки;
- **-u** — удалить LILO. После удаления вы сможете загрузить Linux только с внешнего носителя (дискеты).

Если вы решили пока не удалять LILO, то при очередной загрузке системы вас встретит приглашение LILO. Нажмите <Ввод>, чтобы загрузить вариант по умолчанию. Нажмите <Tab> для просмотра вариантов. Иногда на экран выдается только подсказка:

```
LILO
```

Чтобы выбрать ядро, нужно нажать клавишу «Shift», после чего появится подсказка:

```
LILO boot:
```

и только теперь можно нажать <Tab>. Если вы введете команду **help**, то получите список всех команд LILO.

Чтобы загрузиться в однопользовательском режиме (например, для восстановления системы после аварии), введите в строке приглашения:

```
<метка_варианта> single
```

Если вы переустановили Windows, а она заменила вам содержимое MBR, то восстановить его можно, загрузившись с дискеты или компакт-диска и введя команду `lilo`. Таким же образом можно установить LILO, если при инсталляции системы вы выбрали другой диспетчер загрузки.

Любопытно, что четыре буквы приглашения LILO — это отметки об успешном окончании четырех шагов загрузки. Если вы видите меньше четырех букв, то это значит:

- **L<xx>** : первичный загрузчик запустился, но не смог загрузить вторичный. Двузначное число xx — это код ошибки. Обычно причина ошибки — аппаратный сбой.
- **LI** : LI смог загрузить вторичный загрузчик, но не смог передать ему управление. Причина: перемещение `/boot/boot.b` без запуска утилиты ***lilo***. Чаще всего это значит, что вы удалили или добавили раздел.
- **LIL** : LO запустился, но не смог прочитать карту размещения файлов. Аппаратный сбой.
- **LIL?** : LO загрузился по неправильному адресу. Причина — перемещение `/boot/boot.b` без запуска утилиты **lilo**.
- **LIL-** : испорчена карта размещения файлов. Причина — перемещение `/boot/map` без запуска утилиты **lilo**.
- Еще одной причиной любой из этих ошибок может быть неправильное указание геометрии диска (несовпадение ее с фактической).

Загрузчик GRUB

Другой известный диспетчер загрузки — это набирающий все большую популярность GNU GRUB (*GRand Unified Bootloader*). В дистрибутивах Mandrake и Fedora Core этот загрузчик используется по умолчанию, хотя я рекомендую вам сменить его на проверенный временем LILO.

GRUB использует несколько отличную от LILO схему загрузки. Вторичный загрузчик хранится не в каком-то файле, а в не используемом системой пространстве. Обычно это вся первая дорожка диска. Если места для полноценного вторичного загрузчика там недостаточно, туда помещается маленький загрузчик промежуточного этапа, «полуторный», состоящий из драйвера файловой системы и инструкций для вызова настоящего, большого, вторичного загрузчика. Благодаря такой организации GRUB поддерживает большинство файловых систем (FAT и FAT32, ext2fs и ext3fs, ReiserFS, XFS, BSD FFS), понимает большинство форматов исполняемых файлов и — не главное, но самое заметное для начинающего системного администратора, — изменения в конфигурационном файле вступают в силу сразу же, без прописывания их в специальном месте специальной утилитой.

Конфигурационный файл GRUB называется `/boot/grub/grub.conf`. В мгновенно устаревающих руководствах вы можете встретить другое название — `/boot/grub/menu.lst` — но это не страшно: этот файл является символической ссылкой на первый. Пример такого файла приведен ниже.

Листинг 9.2. Примерный файл /boot/grub/grub.conf

```
#boot=/dev/hda
default=0
fallback=1
timeout=5
splashimage=(hd0,1)/grub/splash.xpm.gz
hiddenmenu
title Fedora Core (2.6.9-1.667)
 root (hd0,1)
 kernel /vmlinuz-2.6.9-1.667 ro root=/dev/hda2
 initrd /initrd-2.6.9-1.667.img
title WindowsXP
 rootnoverify (hd0,0)
 chainloader +1
```

Закомментированная команда boot указывает загрузочный диск. Команда default говорит, какая метка (title) будет загружена по умолчанию, а fallback — какая система будет загружена в случае неудачи. Команда timeout указывает время (в секундах) ожидания ввода команды или выбора другой операционной системы.

Команда splashimage указывает, какой рисунок будет использован в качестве фона. Если у вас что-то не ладится с видеорежимом, закомментируйте это строку.

Труднее всего смириться с тем, как GRUB именует жесткие диски и их разделы. Диски отсчитываются не с буквы «а», а с нуля, а разделы — не с единицы, а тоже с нуля. Таким образом, раздел, который Linux именует `/dev/hda1`, а Windows — C:, GRUB называет (hd0,0). Круглые скобки обязательны.

Команда **rootnoverify** нужна для не-Linux систем, а **chainloader +1** — для ОС, понимающих только «цепочечную» загрузку (MBR → загрузочная запись активного раздела). Если вы зачем-то разместили Windows на неактивном разделе, откуда она сама загрузиться не сможет, то команде **chainloader** должна предшествовать команда **makeactive**.

GRUB имеет графический интерфейс, но попасть в его командную строку все-таки можно: увидев меню вторичного загрузчика, выберите курсором нужную позицию и нажмите <c> (нажав <Ввод>, вы загрузите выбранную систему, нажав <a>, вы сможете передать ядру дополнительные параметры, а нажав <e> — отредактировать последовательность команд, выполняемых при загрузке выбранной системы).

Итак, вы в командной строке. Редактировать вводимые команды можно в стиле **bash**: поддерживается автодополнение команд и путей к файлам, пролистывание истории команд стрелками «вверх» и «вниз». Список всех

доступных команд можно получить по команде **help**, а получить краткую справку о команде — введя `help <имя_команды>`.

Установить диспетчер загрузки GRUB (если вы не сделали этого при инсталляции системы) можно командой `/sbin/grub-install /dev/hda`. Утилита ***grub-install***, как и все остальные компоненты GRUB, содержится в пакете grub, который можно скачать по адресу `http://www.sisyphus.ru/srpm/grub`.

Как установить графический фон загрузчика GRUB

Много интересных картинок, пригодных для установки в качестве фона загрузчика GRUB, можно найти по адресу `http://ruslug.rutgers.edu/~mcgrof/grub-images/images/working-splashimages`. Чтобы не загружать все эти картинки, можно просмотреть их уменьшенные изображения по адресу `http://ruslug.rutgers.edu/~mcgrof/grub-images/images`, а потом уже загрузить понравившуюся картинку.

Выбранную картинку поместите в каталог `/boot/grub` и укажите ее в директиве splashimage конфигурационного файла:

```
splashimage=(hd0,0)/boot/grub/image.xpm.gz
```

Собственную картинку можно использовать как фон для загрузчика, если преобразовать ее в формат, поддерживаемый GRUB. Преобразование выполняется утилитой **convert**, после чего картинку нужно сжать компрессором **gzip**:

```
# convert myimage.png -colors 14 -resize 640x480 myimage.xpm
# gzip myimage.xpm
```

9.1.2. Продолжение загрузки.

Демон init

С момента загрузки ядра процесс начальной загрузки системы идет под управлением самой системы. Первой получает управление процедура автозапуска ядра. Она определяет объем доступной оперативной памяти, тип и быстродействие процессора, тип видеоадаптера, переинициализирует жесткие диски, не полагаясь на инициализацию, выполненную BIOS.

Это делается для того, чтобы выбрать из возможных вариантов выполнения ядром основных функций оптимизированные именно для данной архитектуры компьютера, повысив тем самым быстродействие всей системы. Здесь же определяется системная консоль, на которую выводятся диагностические сообщения.

Наконец, процедура автозапуска распаковывает загруженный в память образ ядра (на этом этапе вы видите сообщение: «Uncompressing Linux...») и передает ему управление («OK, booting the kernel»). Теперь ядро инициализирует таблицу страниц виртуальной памяти, устанавливает обработчики прерываний, разбирает параметры, переданные ему диспетчером загрузки, и настраивается в соответствии с ними.

Завершив самонастройку, ядро создает несколько системных «процессов», фактически представляющих собой части самого ядра: планировщик процессов, диспетчер виртуальной памяти, различные обработчики сигналов ядра.

В списке текущих процессов, который вы видели по команде `ps -e`, эти системные процессы взяты в квадратные скобки. Один из них получает идентификатор 1, он-то и станет полноценным пользовательским процессом, в котором будет выполняться код демона **init**. Этот демон запустит все остальные службы и процессы, управляющие базовыми операциями: например, входом пользователей в систему.

Ядро монтирует корневую файловую систему в режиме «только чтение», находит исполняемый файл демона **init** (в каталоге `/bin`, `/sbin` или там, где вы укажете, передав ядру параметр `init=/путь_к_init`) и посредством системного вызова *exec()* загружает его код в процесс номер 1. Все остальные процессы порождает **init** и его потомки путем деления с помощью системного вызова *fork()*.

Примечание

Ядро покорно запустит в качестве первопроцесса любую программу, которую вы укажете ему как **init**:

```
LILO: my_linux init=/bin/sh
```

Конечно, оболочка **sh** не запустит других процессов, но она предоставит вам интерфейс командной строки, в которой вы сможете выполнить необходимые ремонтные работы.

Процесс **init** прочитывает свой конфигурационный файл `/etc/inittab` и запускает другие процессы согласно указанным в нем инструкциям. В этот момент выводится приглашение нажать определенную клавишу (обычно <I>), чтобы войти в интерактивный режим, позволяющий запускать каждый процесс вручную или отказываться от такого запуска.

Уровни выполнения

Уровень выполнения (*runlevel*) — это такой режим работы системы, в котором разрешается существование только определенной группы про-

цессов. В каждый момент времени система находится на одном из уровней выполнения (на каком именно, можно узнать по команде `who -r`. Она покажет также значение предыдущего уровня).

Разрешенные на каждом уровне процессы указаны в файле `/etc/inittab`. Демон ***init*** заведует переключением уровней, остановкой запрещенных на новом уровне процессов и запуском предписанных. В ОС Linux определено:

- семь уровней выполнения, обозначаемых номерами с 0 до 6;
- особый уровень S или s — однопользовательский;
- уровни по требованию (*ondemand*) A, B и C — фиктивные: при переходе на эти уровни запускаются приписанные к ним процессы, но текущий уровень выполнения не меняется.

Термин «уровень выполнения» унаследован от тех времен, когда система была обязана проходить уровни последовательно, от низшего к высшему при загрузке и обратно при выключении. Сейчас их можно переключать в любом порядке. Для переключения на уровень n нужно от имени суперпользователя ввести команду

```
# telinit n
```

Эта команда посылает соответствующий сигнал процессу **init** (*tell init*). Ее исполняемый файл представляет собой символическую ссылку на `/sbin/init`, так что вместо нее можно отдавать просто команду `init`. При этом не будет запущена копия процесса **init**: стартующий процесс первым делом проверяет свой PID и, если тот не равен 1, просто передает сообщение настоящему процессу **init**.

Запустив все процессы, приписанные к текущему уровню выполнения, **init** засыпает до получения сигнала о завершении дочернего процесса, отключении питания или требовании переключить уровень. Тогда он просыпается, перечитывает свой конфигурационный файл и, если нужно, выполняет записанные в нем инструкции. Чтобы заставить его перечитать измененный вами `/etc/inittab`, не дожидаясь трех вышеуказанных событий, введите команду

```
# telinit q
```

Важно понять, что уровень выполнения — это программная абстракция, аппаратура ни о каких уровнях не знает. Поэтому в разных реализациях Linux (разных дистрибутивах) одному уровню могут соответствовать разные конфигурации системы. Следующие уровни используются в дистрибутивах, основанных на Red Hat:

- 0: Останов системы.
- 1: Однопользовательский режим. То же, что уровень S.
- 2: Многопользовательский режим без поддержки сети.

- 3: Полный многопользовательский режим.
- 4: Не используется.
- 5: Графический режим с X11.
- 6: Перезагрузка.

В однопользовательском режиме никакие службы не стартуют: только грузится ядро, монтируется корневая файловая система и запускается командный интерпретатор. На этом уровне не нужен даже файл `/etc/inittab`, повреждение которого означает невозможность загрузиться на любом другом рабочем уровне. Этот уровень обычно использует администратор для аварийно-восстановительных работ.

Неиспользуемые уровни можно занять под свою собственную конфигурацию служб, собранных для конкретной задачи.

Конфигурационный файл /etc/inittab

Первая незакомментированная строка этого файла определяет уровень выполнения по умолчанию, то есть тот, в котором стартует система, если в процессе загрузки ядру не указано иначе. Эта строка выглядит как

```
id:3:initdefault
```

Обычно в качестве уровня по умолчанию выбирают 3 (полнофункциональный многопользовательский текстовый режим) или графический 5 (запускается X Window и выдается графическое приглашение для входа в систему). Если оставить поле уровня пустым, то ***init*** переспросит значение в процессе загрузки. Если указать в поле уровня несколько значений, то сработает наибольшее. Уровни 0 (останов) и 6 (перезагрузка) указывать нельзя.

Следующие строки имеют формат:

```
идентификатор:уровни_выполнения:действие:запускаемый_процесс
```

- **Идентификатор** — это уникальная последовательность из четырех символов (в старых дистрибутивах — двух).
- **Уровни_выполнения**: перечень уровней выполнения (номера без каких-либо разделителей), для которых будет выполнено указанное действие. Например, значение 2345 требует выполнить действие на уровнях 2, 3, 4 и 5. Здесь можно указывать также уровни по требованию (*ondemand*) A, B и C.
- **Действие**: одно из действий, перечисленных в таблице 9.1.
- **Запускаемый_процесс**: процесс, над которым производится действие. Это может быть исполняемый файл или сценарий.

Действия над процессами, задаваемые в файле /etc/inittab *Таблица 9.1*

Действие	Описание
respawn	В случае завершения процесс будет перезапущен
wait	Процесс будет запущен при переключении на любой из указанных уровней, и ***init*** будет ждать его завершения
once	При переключении на любой из указанных уровней процесс будет запущен только однажды
boot	Процесс будет запущен во время загрузки системы. Поле «уровни_выполнения» игнорируется
bootwait	То же, что и boot, но ***init*** ждет завершения процесса
off	Не выполняет никаких действий
ondemand	Процесс выполняется в режиме по требованию, то есть он будет запущен при переключении на уровни a, b или c
initdefault	Определяет уровень выполнения по умолчанию
sysinit	Процесс запускается во время загрузки системы до любых процессов, стартующих через boot или bootwait
powerwait	Процесс будет запущен, когда исчезнет напряжение в сети. Естественно, для корректной работы этой записи нужен источник бесперебойного питания, от которого система и получит уведомление об исчезновении напряжения; ***init*** будет ждать завершения этого процесса
powerfail	То же, что и powerwait, но ***init*** не будет ждать завершения процесса
powerokwait	Процесс запускается по получении ***init*** сигнала о восстановлении питания
powerfailnow	Процесс запускается, когда источник бесперебойного питания подает сигнал о том, что его батареи почти разряжены
ctrlaltdel	Процесс запускается при получении ***init*** сигнала INT, посланного нажатием комбинации клавиш Ctrl+Alt+Del. Обычно это процесс перезагрузки, выключения или перехода в однопользовательский режим.
kbrequest	Процесс запускается при получении ***init*** сигнала, посланного нажатием «специальной» комбинации клавиш. Назначить клавишам специальное значение можно с помощью утилит из пакета kbd

Сценарий, помеченный действием **sysinit**, выполняется во время запуска системы, однократно, вне зависимости от запрошенного уровня. Затем запускаются процессы, помеченные действиями **boot** и **bootwait**.

На данном этапе загрузки системы происходят следующие действия по инициализации, результат которых необходим на любом уровне выполнения:

1. Устанавливается имя машины (hostname).
2. Конфигурируются параметры ядра.
3. Устанавливаются раскладка клавиш и системный шрифт.
4. Активируются разделы подкачки.
5. Корневая система проверяется программой **fsck**. Если будут найдены ошибки, которые невозможно исправить автоматически, будет запрошен пароль администратора для входа в административный режим, что равноценно переходу на уровень выполнения 1. В этом режиме вы запустите программу **fsck** с аргументом «/», который означает проверку корневой файловой системы. После исправления всех ошибок введите команду `exit` для перезагрузки системы. Если программа

fsck ошибок не обнаружила, файловая система монтируется в режиме чтение/запись.
6. Проверяются зависимости модулей ядра.
7. Выполняется проверка других файловых систем.
8. Монтируются локальные файловые системы.
9. Включаются квоты.
10. Подключается (не активизируется!) раздел подкачки. С этого момента система начинает использовать раздел подкачки.

Дальше включается сценарий загрузки, специфичный для запрошенного уровня, и начинается разница между дистрибутивами.

В процессе развития UNIX-подобных ОС выделились две основных системы инициализации (набора сценариев загрузки). Одна была разработана в рамках ОС BSD (Berkeley Software Distribution) Калифорнийского университета, от которой произошли бесплатная FreeBSD и коммерческая SunOS. Другая применяется в классической System V от AT&T и ее потомках, среди которых UnixWare, IRIX, HP-UX и Solaris. ОС Linux заимствовала удачные решения с обеих эволюционных ветвей, и в результате часть дистрибутивов следует в инициализации стилю BSD (Slackware, CRUX, Gentoo), а часть (Red Hat-подобные) — стилю System V.

Инициализация в стиле BSD

Для этого стиля характерно наличие конфигурационного файла стартовых сценариев `/etc/rc.conf`. Уровней выполнения как таковых в BSD и ориентированных на нее реализациях Linux нет, вместо них вводится понятие режима — группы процессов, объединенных общей функциональностью. Режимов обычно два: однопользовательский и многопользовательский. Каждый режим запускается своим сценарием. Файлы этих сценариев обычно находятся в `/etc` и называются `rc.*` (рис. 9.1).

В дистрибутиве Slackware этим сценариям присвоены имена:

- `rc.S` — сценарий запуска (действие sysinit);
- `rc.0` — останов системы;
- `rc.6` — перезагрузка;
- `rc.K` — однопользовательский режим;
- `rc.M` — многопользовательский текстовый режим;
- `rc.4` — многопользовательский режим с графическим входом в систему.

В `/etc/default/rc.conf` хранятся в неизменном хорошо откомментированном виде все системные настройки в количестве нескольких сотен, а редактируемый администратором `/etc/rc.conf` содержит только отличия желаемой конфигурации системы от `/etc/defaults/rc.conf`, которых раз в десять меньше. Загрузочные сценарии режимов прочиты-

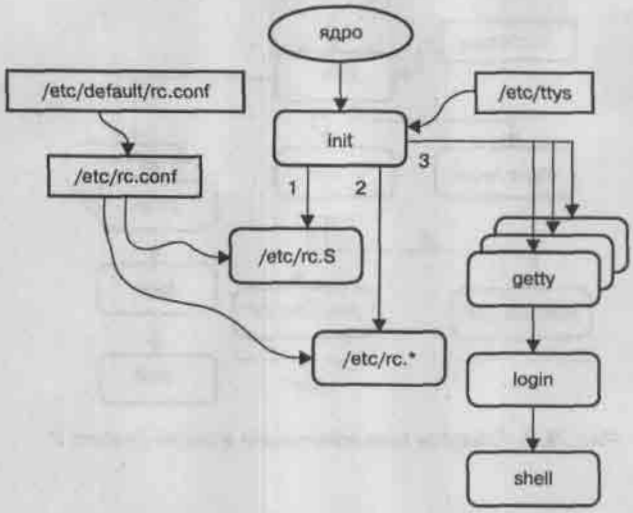


Рис. 9.1. *Порядок инициализации в стиле BSD*

вают оба эти файла и в зависимости от требуемой конфигурации могут запускать из-под себя дополнительные сценарии инициализации различных служб: `rc.inetd*`, `rc.cdrom` и т.п. Последним при загрузке выполняется сценарий rc.local, содержание которого определяется администратором конкретной системы.

Инициализация в стиле System V

В этом стиле каждому уровню выполнения соответствует целый каталог, все сценарии в котором выполняются при переключении на этот уровень. Это подкаталоги `/etc` с именами `rc0.d, rc1.d, ..., rc6.d`. Сценарии в этих каталогах — файлы с именами вроде S12syslog или K95kudzu — только символические ссылки на настоящие сценарии, находящиеся в `/etc/init.d`. Каждый из настоящих сценариев, будучи вызван с аргументом start, запускает свою службу, а с аргументом stop — останавливает ее. Какой аргумент будет ему передан, зависит от первой буквы имени символической ссылки: S означает start, K (*kill*) — stop. Следующее за этой буквой число определяет порядок вызова настоящих сценариев: чем оно больше, тем позже срабатывает данная ссылка при включении текущего уровня. Сначала выполняются все сценарии останова процессов, не разрешенных на данном уровне, потом — все сценарии запуска (рис. 9.2).

Переключением уровней выполнения занимается центральный сценарий `/etc/rc`. Вызванный с аргументом N, где N — это номер включаемого

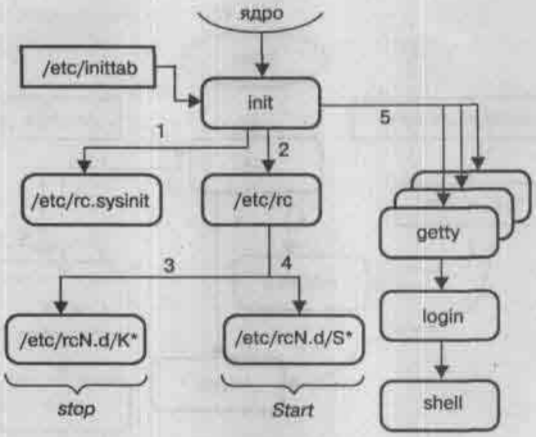


Рис. 9.2. *Порядок инициализации в стиле System V*

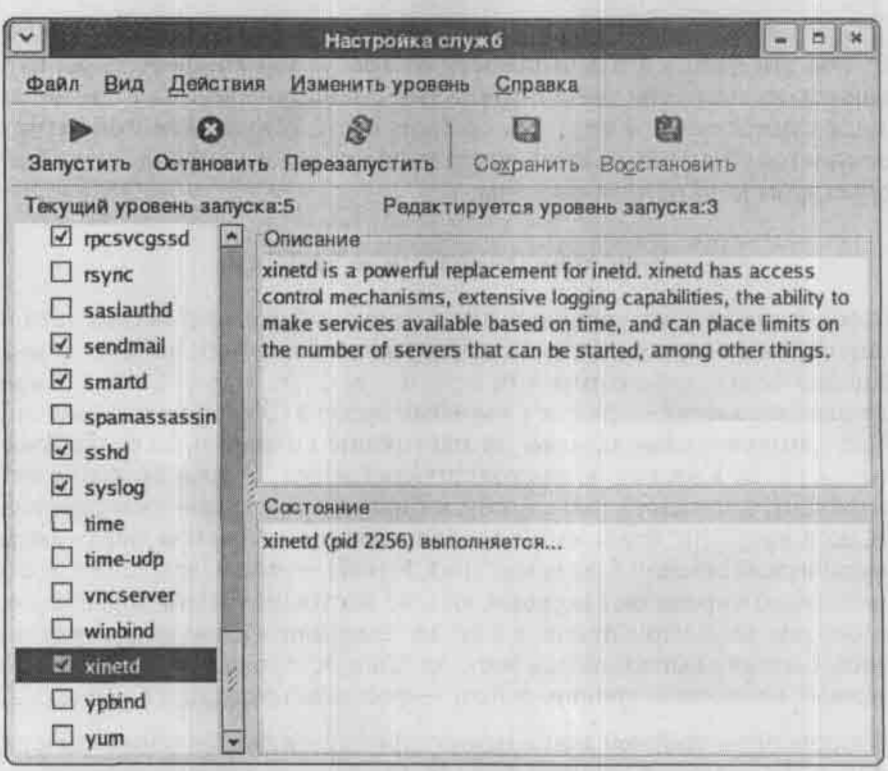


Рис. 9.3. *Конфигуратор служб system-config-services*

уровня, он ищет каталог /etc/rc.N и выполняет в нем сначала все стоп-сценарии, потом все старт-сценарии.

Для выбора демонов, которые будут запускаться автоматически при загрузке системы, обычно используют конфигуратор **drakconf** в операционной системе Linux Mandrake, **system-config-services** в Fedora Core (рис.9.3) или **setup** в других Red Hat-подобных дистрибутивах.

Чтобы обеспечить автоматический запуск какого-нибудь сервиса, нужно создать сценарий для его запуска и поместить его в каталоге /etc/init.d. Затем, в зависимости от уровня выполнения, в каталоге /etc/rcN.d нужно создать символические ссылки на этот сценарий для его запуска и останова.

Если вы хотите сами создать сценарий для запуска своего демона, можете воспользоваться шаблоном, приведенным в листинге 9.3.

Листинг 9.3. Шаблон для запуска демона

```
#!/bin/bash
#
# Подключаем библиотеку функций
. /etc/init.d/functions
#
# Определяем параметры
case "$1" in
  start)
 # Запуск демона
 echo "Starting my_daemon..."
 daemon my_daemon
 touch /var/lock/subsys/my_daemon
 ;;
  stop)
 # Останов демона
 killproc my_daemon
 rm -f /var/lock/subsys/my_daemon
 rm -f /var/run/my_daemon.pid
 ;;
  status)
 # Выводим статистику работы
 ;;
  restart | reload)
 # действия, выполняемые при перезагрузке демона
 ;;
  *)
 # Произошел вызов без параметров
```

```
  echo "Usage: my_daemon {start|stop|status|restart|reload}
"
  exit 1
 esac
 exit 0
```

После того, как в процессе инициализации системы будет выполнен загрузочный сценарий уровня по умолчанию, последним выполняется сценарий `/etc/rc.local`. Выполнив все сценарии, **init** переходит к другим записям в `/etc/inittab`, относящимся к текущему уровню. Обычно там остаются только перезапускаемые (respawn) действия: процессы, которые **init** запускает в фоне, а когда какой-нибудь из них завершается, запускает вновь. Так ведут себя процессы *getty, обслуживающие виртуальные консоли, и менеджер дисплеев системы X Window. Инициализация системы считается законченной, когда запущены все перезапускаемые процессы и **init** остается только следить за ними.

9.2. Команды управления процессами

9.2.1. Иерархия процессов: ps и pstree

О том, что команда **ps** позволяет просмотреть сведения обо всех процессах, протекающих в системе в данный момент, вы уже знаете (п.3.2). С ключом –f эта команда выводит как PID самого процесса, так и PPID его родителя, то есть по ее выводу можно восстановить всю структуру дерева процессов до их общего предка — процесса **init**. «Древовидный» взгляд может понадобиться, например, если вам нужно уничтожить целую группу процессов, происходящих от общего предка: в этом случае вы можете не убивать их по очереди, а просто послать сигнал TERM их родительскому процессу.

Команда **pstree** представляет дерево процессов сразу в наглядном виде:

```
$ pstree
init─┬─acpid
     ├─atd
     ├─bonobo-activati
    ...
     ├─crond
     ├─gconfd-2
     ├─gdm-binary──gdm-binary─┬─X
     │                        └─gnome-session
     ├─gnome-panel
     ├─gnome-settings-
```

```
  ├─gnome-terminal─┬─bash──pstree
  │                ├─bash──su──bash──man──sh──sh─┬─less
  │                │                             └─nroff──iconv
  │                ├─bash──su──bash
  │                └─gnome-pty-helpe
  │
...
  ├─syslogd
  ├─udevd
  └─xinetd
```

Ключ **-p** выводит вместе с именем процесса его PID, а ключ **-u** — имя пользователя, запустившего процесс. Если в качестве аргумента указать PID, то команда выведет не все дерево, а только ветку потомков процесса с этим PID. Имя пользователя в качестве аргумента требует вывести все ветки процессов, запущенных этим пользователем: у них общего предка может и не быть.

9.2.2. Информация о ресурсах системы: команды free, df, du

Команда **free** показывает общее количество занятой и свободной памяти: физической, в разделе подкачки и в буферах ядра. По умолчанию объем памяти выводится в килобайтах, а ключи **-b** и **-m** позволяют измерять его в байтах и мегабайтах соответственно.

Ключ **-s <число>** требует опрашивать систему непрерывно, через каждые <число> секунд, пока вы не прервете выполнение команды, нажав Ctrl+C.

Команды **df** (*disk free*) и **du** *(disk usage)* показывают, сколько места доступно и занято на жестком диске.

Утилита **df** выводит сведения о дисковом пространстве на всех смонтированных в данный момент файловых системах — как локальных, так и сетевых:

```
$ df

Файловая система 1К-блоков Исп      Доступно Исп% смонтирована на
/dev/hda4        22200824  4630864  16442200 22%  /
/dev/hda2        101105    8383     87501    9%   /boot
/dev/hda6        10221420  1737440  8483980  17%  /mnt/win_disk_e
//user5/share    7543680   4623488  2920192  62%  /mnt/win_user5
```

С ключом **–i** команда `df` измеряет дисковое пространство не в блоках, а в индексных дескрипторах (п.2.2.1), а с ключом **–T** показывает тип файловой системы. О других ключах можно узнать, как обычно, на man-странице.

Команда **du** показывает, сколько места занимает каждый подкаталог каталогов, указанных в качестве аргумента (если аргументы не указаны, то это будет текущий каталог), и подсчитывает общий итог. Если у вас недостаточно прав на просмотр некоторых подкаталогов, то итог будет отличаться от правильного на размер запрещенных каталогов.

9.2.3. Процессы в реальном времени: команда top

Команда **top** предназначена для наблюдения за процессами в реальном времени и интерактивного управления ими. Терминал, на котором запущена программа **top**, превращается в окно, разделенное на области (рис. 9.4, сверху вниз):

- сведения о системе — продолжительность текущего сеанса, количество задач, использование памяти и процессора, средняя длина очереди задач, ожидающих выполнения (load average) и т.п.;
- командная строка;
- заголовки столбцов;
- область задач. Задачи в терминологии **top** — это не только пользовательские процессы — потомки **init**, но и системные процессы — части ядра.

```
den@dhsilab:~
Файл  Правка  Вид  Терминал  Вкладки  Справка
top - 20:16:38 up  9:26,  4 users,  load average: 0.30, 0.14, 0.07
Tasks: 100 total,   1 running,  99 sleeping,   0 stopped,   0 zombie
Cpu(s):  2.3% us,  1.0% sy,  0.0% ni, 96.3% id,  0.0% wa,  0.3% hi,  0.0% si
Mem:    223284k total,   217016k used,     6268k free,     7320k buffers
Swap:  1044188k total,     8512k used,  1035676k free,    72372k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 3294 den       16   0 41684  16m  20m S  2.0  7.4   0:17.04 gnome-terminal
 2637 root      15   0 68832  20m  52m S  1.3  9.2   2:04.11 X
    1 root      16   0  2876  564 1408 S  0.0  0.3   0:00.51 init
    2 root      34  19     0    0    0 S  0.0  0.0   0:00.04 ksoftirqd/0
    3 root       5 -10     0    0    0 S  0.0  0.0   0:00.02 events/0
    4 root       6 -10     0    0    0 S  0.0  0.0   0:00.00 khelper
    5 root      15 -10     0    0    0 S  0.0  0.0   0:00.00 kacpid
   24 root       5 -10     0    0    0 S  0.0  0.0   0:00.00 kblockd/0
   34 root      20   0     0    0    0 S  0.0  0.0   0:00.00 pdflush
   35 root      15   0     0    0    0 S  0.0  0.0   0:00.04 pdflush
```

Рис. 9.4. *Программа top запущена на виртуальном терминале*

Каждые несколько секунд (по умолчанию 3 секунды) окно обновляется.

В командную строку можно вводить команды управления самими задачами или порядком отображения сведений в окне программы **top**. Вот некоторые из таких команд:

- **h** — вызов справки;
- **q** — завершение работы программы (для выхода можно нажать и комбинацию Ctrl+C);
- **u** — показывать только процессы, запущенные определенным пользователем (следует указывать его регистрационное имя, а не UID);
- **d** — изменить интервал обновления;
- **f** — добавить или удалить столбцы;
- **F** — изменить столбец, по которому сортируются задачи. По умолчанию это %CPU — доля использованного процессорного времени, но можно сортировать, например, по убыванию объема занятой памяти;
- **i** — переключатель отображения: либо все задачи, либо только активные (в состоянии Running);
- **k** — отправить процессу сигнал. Программа спросит у вас PID процесса, а затем номер или название сигнала. По умолчанию посылается сигнал TERM;
- **r** — изменить приоритет процесса.

Количество и порядок отображаемых свойств процесса (столбцов) можно изменять. Всего доступно 26 свойств процесса (нумеруемых буквами английского алфавита), и соответствующие им заголовки столбцов перечислены в секции DESCRIPTIONS of Fields man-страницы команды **top**. На рис. 9.4 (команда **top** запущена с настройками по умолчанию) отображены следующие свойства:

- PID — PID процесса;
- USER — регистрационное имя владельца процесса;
- PR — приоритет;
- NI — показатель уступчивости (см. п.9.2.4);
- VIRT — занятая виртуальная память в килобайтах;
- RES — физическая (без подкачки) память, занятая процессом;
- SHR — объем разделяемой памяти, используемой процессом;
- S — состояние процесса;
- %CPU — доля процессорного времени, доставшаяся процессу с момента последнего обновления экрана;
- %MEM — доля занятой процессом физической памяти;
- TIME+ — процессорное время, израсходованное с момента запуска процесса, в секундах с сотыми долями;
- COMMAND — либо имя исполняемого файла программы, либо вся командная строка с аргументами. Режим переключается вводом в командную строку команды «c».

Процесс может находиться в одном из следующих состояний, отображаемых в столбце «S»:

- R (Running) — активен, то есть находится в основной памяти и ждет выделения ему процессорного времени либо уже выполняется;
- S (Sleeping) — выгружен из основной памяти;
- T (Traced) — приостановлен, например, в ходе отладки;
- D — состояние непрерываемого ожидания: процесс может быть «разбужен» только прямым (direct) сигналом от оборудования;
- Z — зомби. Это процесс, родитель которого не получил сигнала о завершении потомка и не очистил связанные с ним структуры ядра. То есть дочернего процесса нет, он не тратит процессорного времени и других ресурсов, а запись в таблице процессов осталась. Такие процессы нужно убивать вручную.

9.2.4. Приоритет процесса: команды nice и renice

Каждому процессу в системе назначен определённый приоритет, который учитывается планировщиком процессов при выделении процессу процессорного времени. Значение приоритета находится в диапазоне от -20 (наивысший приоритет) до 19 (наименьший: процесс выполняется только тогда, когда нет других претендентов на время процессора). Значение, обратное приоритету, называется показателем уступчивости (*nice*).

По умолчанию все процессы запускаются с базовым приоритетом, равным 0. Владелец процесса может в любой момент повысить его показатель уступчивости (понизить приоритет). Суперпользователь имеет право установить для любого процесса любое значение приоритета.

Если процесс отъедает слишком много ресурсов, то для нормального функционирования системы не обязательно его убивать: достаточно назначить ему низкий приоритет. Тогда планировщик предоставит ему меньше циклов процессора, и его выполнение займет больше времени, зато в течение этого времени можно будет выполнять другие задачи.

Для запуска процесса с приоритетом, отличным от базового, служит команда **nice**:

```
nice [ -n <приоритет> ] [командная_строка]
```

Например, копирование образа компакт-диска, будучи запущено с обычным приоритетом, может заблокировать остальные процессы. Поэтому запускать его нужно так:

```
$ nice -n 19 dd if=/media/cdrecorder of=~/my_cdrom.iso
```

Значение нового приоритета по умолчанию равно 10. Команда `nice` без аргументов выводит текущее значение базового приоритета.

Для изменения приоритета уже запущенного процесса служит команда **renice**:

```
renice -n <показатель_уступчивости> [-p PID] [-u UID]
```

Суперпользователь имеет право назначать процессу (или всем процессам указанного пользователя) отрицательный показатель уступчивости, то есть повышать их приоритет. Если указанное значение уступчивости выходит за границы диапазона [-20...19], то вместо него применяется соответствующее крайнее значение.

9.2.5. Фоновый режим: команды jobs, fg, bg

В п.3.2 я уже говорил, что родительский процесс может либо ждать завершения дочернего, либо продолжать свое выполнение. Если в роли родителя выступает командная оболочка, то это значит, что процессы, запущенные с одной консоли (или виртуального терминала), распадаются на две группы: те, которых оболочка ждет, — они взаимодействуют с пользователем, занимая консоль, — и те, после запуска которых с пользователем взаимодействует сама оболочка (консоль свободна). Эти группы называются передним и задним планами.

Чтобы запустить процесс на заднем плане (в фоновом или асинхронном режиме), нужно завершить командную строку управляющим оператором &.

Подкоманды оболочки **jobs, bg** (*background*) и **fg** (*foreground*) позволяют манипулировать заданиями, выполняющимися на переднем и заднем планах:

- Команда **jobs** выводит список процессов, которые выполняются в фоновом режиме,
- **fg** <номер_задания > переводит процесс на передний план,
- **bg** <номер_задания > — переводит процесс на задний план.

Номер задания — это не PID, а число, которое команда **jobs** выводит в квадратных скобках. С ключом **–l** она будет выводить, кроме того, и PID процесса.

Поскольку перечисленные команды — не самостоятельные утилиты, а подкоманды ***bash***, справку по ним нужно запрашивать так: `help <подкоманда>`.

9.3. Протоколирование системы

В любой UNIX-подобной системе есть стандартные файлы протоколов (журналов). В них попадают сообщения, генерируемые ядром, системны-

ми демонами, утилитами окружения. Эти файлы размещаются в каталоге `/var/log`. Прикладные программы обычно помещают свои протоколы в подкаталоги этого каталога, например, `/var/log/httpd` — журналы HTTP-сервера. Точное расположение журнала прикладной программы зависит от ее настройки.

За ведение журналов отвечает система регистрации событий syslog. Она позволяет сортировать сообщения по источникам и степени важности, а также перенаправлять их в журнальные файлы, на консоль или на другой компьютер в сети. Все подсистемы (демоны) и правильно написанные прикладные программы не ведут протоколов самостоятельно, а посылают сообщения в систему регистрации. Дальнейшая судьба этих сообщений зависит от политики системного администратора.

Система регистрации состоит из следующих компонент:

- демон **syslogd**, принимающий сообщения от других процессов и перенаправляющий их указанным адресатам;
- библиотечные функции **openlog()**, **syslog()**, **closelog()**, посредством которых процессы общаются с демоном **syslogd**. Функция **syslog()** записывает протокольное сообщение в гнездо (сокет) `/dev/log`, откуда его читает демон **syslogd**;
- команда **logger**, с помощью которой сообщение демону **syslogd** может передать командный интерпретатор.

Демон **syslogd** запускается инициализационным сценарием в ходе начальной загрузки системы и работает непрерывно. Убедиться в том, что он запущен, можно с помощью команды

```
$ ps -ef | grep syslogd | grep -v grep
```

Примечание

Последняя в этом конвейере команда нужна для того, чтобы отфильтровать из вывода команды ***grep*** сведения о самой команде ***grep***.

Если ваша система загружается в стиле System V, то сценарий запуска демона — это `/etc/rc.d/init.d/syslog`. Как обычно, запустить демон самостоятельно вы можете с помощью команды

```
/etc/rc.d/init.d/syslog start,
```

а остановить — `/etc/rc.d/init.d/syslog stop`.

Автоматический запуск системы **syslog** можно отключить или включить при помощи графического конфигуратора (**setup** для дистрибутива Red Hat, **drakxservices** для Mandrake, **system-config-services** для Fedora Core).

В Red Hat-совместимых дистрибутивах демон **syslogd** устанавливается из пакета **sysklogd**, в состав которого входит также демон **klogd**, принимающий сообщения ядра и отправляющий их на обработку демону **syslogd**.

Как и всякий демон, **syslogd** управляется конфигурационным файлом. По умолчанию это файл `/etc/syslog.conf`, однако ничто не мешает назвать его как угодно, а потом запускать **syslogd** с ключом `-f /path/to/config.file`. Об остальных немногочисленных ключах можно узнать по команде `man syslogd`.

По сигналу HUP демон закрывает все журнальные файлы, перечитывает конфигурационный файл и возобновляет процесс протоколирования. Чтобы изменения в конфигурационном файле вступили в силу, нужно послать демону **syslogd** сигнал HUP. Например, так:

```
$ kill -HUP `cat /var/run/syslogd.pid`
```

9.3.1. Конфигурационный файл /etc/syslog.conf

Это простой текстовый файл, каждая непустая и незакомментированная (знак комментария — #) строка которого имеет следующий формат:

```
<селектор>[;<селектор>...] <действие>
```

Селектор представляет собой правило отбора сообщений, а действие — указание, что с отобранными сообщениями дальше делать. Например, строка

```
user.*     -/var/log/user.log
```

диктует перенаправление сообщений от пользовательских программ в файл `/var/log/user.log`.

Действиями могут быть:

- **/абсолютное/имя/файла** — записать сообщение в файл. Префикс «-» перед именем файла запрещает синхронизировать файл после каждой записи. По умолчанию ***syslogd*** после записи каждой строки выполняет вызов *fsync()*. Это делается ради сохранности журналов, но при интенсивной записи может отнять половину процессорного времени. Поэтому следует отключать синхронизацию протоколов, в которые пишется много сообщений — например, отладочных.
- **@имя_машины** — переслать сообщение демону ***syslogd***, работающему на указанной машине. Имя должно быть известно DNS или прописано в `/etc/hosts`; в противном случае следует писать @IP-адрес.
- `/dev/console` — вывести сообщение на консоль.
- * — вывести сообщение на экраны всех работающих в данный момент пользователей.

Несколько селекторов, отбирающих сообщения разной категории, можно группировать, разделяя точкой с запятой, для того, чтобы над ними было выполнено одно и то же действие. Действия группировать нельзя. Чтобы направить одни и те же сообщения и в файл, и на удаленный компьютер, нужно вписать в конфигурационный файл две строки с одинаковыми селекторами.

Селектор, в свою очередь, имеет формат:

```
<средство> [,средство2...].<уровень_серьезности>
```

Средство (facility) — это категория программы, пославшей сообщение. Категория выбирается из списка ключевых слов, приведенного в таблице 9.2.

Категории программ, ведущих протокол *Таблица 9.2*

Ключевое слово	Назначение программы, пославшей сообщение
auth authpriv security	Программы, отслеживающие регистрацию пользователей в системе и повышение привилегий пользователя (login, su)
authpriv	Программы, отслеживающие повышение привилегий пользователя (команда su)
cron	Выполнение заданий по расписанию (cron и at)
daemon	Системные демоны, для которых не нашлось более подходящей категории
kern	Ядро
lpr	Подсистема печати
mail	Почтовые программы
news	Подсистема, обслуживающая телеконференции Usenet
uucp	Система UUCP
syslog	Сам демон syslogd
user	Все остальные программы
*	Любая программа
none	Никакая программа

Уровень серьезности (priority) определяет важность сообщения. Программы регистрируют любые сообщения — от отладочной информации до требований экстренного вмешательства — а демон **syslogd** игнорирует те из них, важность которых ниже, чем указано в конфигурационном файле. Уровень серьезности указывается ключевым словом, список которых в порядке возрастания важности приведен в таблице 9.3. Допускаются также ключевые слова * (все сообщения) и none (никаких сообщений).

Уровни серьезности сообщений *Таблица 9.3*

Ключевое слово	Означает
debug	Отладочные сообщения
info	Информационные сообщения о штатных ситуациях
notice	Замечания: сообщения о необычных ситуациях
warning	Предупреждения
err	Ошибки
crit	Критические ошибки
alert	События, требующие срочного вмешательства
emerg	События, угрожающие работе системы

В Red Hat-совместимых системах можно ставить перед уровнем серьезности дополнительные модификаторы «=» (регистрировать сообщения только указанного уровня) и «!» (игнорировать сообщения указанного и более высоких уровней). Можно также направлять сообщения не только в обычный файл, но и в именованный канал, поставив перед ним символ «|».

Листинг 9.4. Примерный файл /etc/syslog.conf

```
# Протоколирование аутентификации. Файл протокола
auth,authpriv.*          /var/log/auth.log
*.*;auth,authpriv.none        -/var/log/syslog
# Файл регистрации попыток доступа к системе. имеет
# ограниченный доступ. Обычно в этот файл записываются
# сообщения об удаленном доступе к этой машине, например,
# сообщения от демона FTP о том, какие пользователи и
# когда регистрировались на данном сервере.
authpriv.*            /var/log/secure

# Сообщения пользовательских программ
user.*                -/var/log/user.log

# Протоколировать все информационные сообщения, кроме
# почтовых
*.info;mail.none;          -/var/log/messages

# Протоколирование почты
#  Уровень отладки, информации и замечаний
mail.=debug;mail.=info;mail.=notice -/var/log/mail/info
#  Уровень предупреждений
mail.=warn                -/var/log/mail/warnings
#  Уровень ошибок
mail.err                  -/var/log/mail/errors

# Протоколирование демона cron.
cron.=debug;cron.=info;cron.=notice -/var/log/cron/info
cron.=warn                -/var/log/cron/warnings
cron.err                  -/var/log/cron/errors

# Протоколирование ядра
kern.=debug;kern.=info;kern.=notice -/var/log/kernel/info
kern.=warn                -/var/log/kernel/warnings
kern.err                  -/var/log/kernel/errors

# Очередь печати: сообщения уровня от "инфо" до "предупреждений"
lpr.info;lpr.!err         -/var/log/lpr/info
```

```
# Протоколирование демонов: сообщения всех уровней, кроме "инфо"
daemon.=debug;daemon.!=info        -/var/log/daemons

# Критические сообщения — всем тревога
*.emerg                  *

# Сохранять ошибки почты и новостей в отдельном файле
uucp,news.crit           -/var/log/spooler

# Загрузочные сообщения
local7.*                 -/var/log/boot.log
```

9.3.2. Сетевое протоколирование

Протоколы — это история жизни системы; они необходимы администратору для выявления и устранения неполадок, но они необходимы и злоумышленнику — для поиска уязвимости или для того, чтобы скрыть следы своего вторжения. Поэтому иногда бывает полезно вести журнал на другом, более безопасном, компьютере.

Протоколирование в сети — это перенаправление сообщений демону **syslogd**, запущенному на другой машине локальной сети, где они будут записаны в локальный файл. Таким образом, один демон **syslogd**, передающий сообщения, выступает в роли клиента, а другой такой же, но принимающий и сохраняющий их, — в роли сервера.

Сообщения передаются по протоколу UDP. Он не обеспечивает ни гарантированной доставки пакетов, ни секретности, но работает несколько быстрее, чем TCP. Следующая строка должна присутствовать (в незакомментированном виде!) в файлах `/etc/services` обоих компьютеров:

```
syslog 514/udp
```

Затем необходимо внести некоторые коррективы в файл конфигурации. На клиентской машине определите объекты протоколирования (селекторы), как и раньше, а в качестве действия укажите @имя_узла, где имя_узла — имя компьютера, на который будут перенаправлены сообщения. Это имя желательно указать в файле `/etc/hosts`, так как демон **syslogd** может быть запущен до сервера доменных имен или сервер DNS окажется недоступным.

На сервере протоколирования может быть необходимо перезапустить **syslogd** с другим набором ключей. В Red Hat-совместимых системах **syslogd** по умолчанию игнорирует сообщения из сети; для того, чтобы он начал их принимать, он должен быть запущен с ключом –r (это верно для пакета sysklogd версии 1.3 и новее. Ранние версии вели себя в точности наоборот).

Каждое сообщение в журнальном файле маркируется полным именем узла, с которого оно поступило (с указанием домена). Чтобы **syslogd** на сервере записывал только краткие имена узлов, нужно запустить его с ключом **–l <список_узлов>** или **–s <список_доменов>**. Узлы и домены в списках нужно разделять двоеточиями.

9.3.3. Протоколирование ядра. Демон klogd и команда dmesg

Демон **klogd** предназначен для перехвата и протоколирования сообщений ядра Linux. Ядро, в отличие от пользовательского процесса, не может выводить сообщения, пользуясь стандартной функцией **syslog**(): библиотека, содержащая эту функцию, доступна только обычным приложениям. В ядре есть свои средства вывода сообщений, приложениям недоступные. Обработку этих сообщений и организует демон **klogd**. Обычно эта обработка состоит в пересылке сообщений демону **syslogd**, хотя **klogd** может работать и самостоятельно.

Демон **syslogd** направляет сообщения ядра вперемешку с сообщениями от других источников в общий журнальный файл (обычно `/var/log/messages`, смотри `/etc/syslog.conf`), но ядро поддерживает и собственный кольцевой буфер сообщений. Для просмотра текущего состояния этого буфера предназначена команда **dmesg**. Ее вывод рекомендуется пропускать через фильтр **more** или **less**.

Анализ сообщений ядра может потребоваться для того, чтобы узнать, поддерживает ли ваше ядро то или иное устройство или функцию. Например, чтобы узнать, включена ли поддержка квотирования (п.7.2.3), выполните команду

```
$ dmesg | grep -i quot
```

Сообщения, которые ядро генерирует и сохраняет в буфере во время загрузки системы, направляются также в файл `/var/log/dmesg` (в не Red Hat-совместимых системах этот файл может называться `/var/log/boot.msg`). По окончании загрузки этот файл закрывается и его содержимое становится доступно для просмотра и анализа.

9.3.4. Что делать с протоколами дальше? Утилита logrotate

Журнальные файлы разрастаются быстро, и рано или поздно с ними приходится что-то делать: сжимать, архивировать или просто удалять. В дистрибутивах линии Red Hat и Debian для управления журналами — не только системы **syslog**, но и любых прикладных программ, — предназна-

чена утилита **logrotate**. Она устанавливается из пакета logrotate вместе с примерным конфигурационным файлом и сценарием, обеспечивающим ее периодический запуск (п.9.4). В результате установки этого пакета сценарий logrotate попадает в каталог `/etc/cron.daily`.

Как следует из названия утилиты **logrotate**, ее основное занятие — это ротация журнальных файлов. Ротация — это последовательное переименование предыдущих версий журналов

(`maillog.2` в `maillog.3`, `maillog.1` в `maillog.2`, `maillog` в `maillog.1`) и создание нового пустого журнального файла (`maillog`) для записи последующих сообщений. Ротация производится либо по истечении указанного времени, либо по превышении журнальным файлом указанного размера.

Список журналов, подлежащих обработке утилитой **logrotate**, и инструкции по обработке определяются конфигурационными файлами, имена которых в любом количестве передаются как аргументы при вызове утилиты (см. `man logrotate`). Директивы последующих конфигурационных файлов перекрывают директивы предыдущих, так что порядок имеет значение. Традиционно используется один файл `/etc/logrotate.conf`, а инструкции для дополнительных журналов включаются в него при помощи директивы include.

Листинг 9.5. Примерный конфигурационный файл /etc/logrotate.conf

```
# Секция глобальных параметров —
# инструкции для всех журналов
# Расписание ротации: еженедельно.
# Возможны варианты: daily, monthly,
# size <байт> — по достижении файлом указанного размера
weekly
# Сохранять 4 предыдущих журнала
rotate 4
# После ротации создавать новый пустой журнальный файл
create
# Сохранять старые журналы в сжатом виде (по умолчанию —
# при помощи gzip, директива compresscmd позволяет указать
# другую программу для сжатия)
compress

# В каталог /etc/logrotate.d прикладные программы помещают
# инструкции по управлению их собственными журналами.
# Директива include <каталог> требует включить все файлы,
# содержащиеся в этом каталоге.
include /etc/logrotate.d
```

```
# Секция локальных параметров — инструкции для
# перечисленных в ней журналов
/var/log/wtmp /var/log/lastlog {
 monthly
 create 0664 root utmp
 rotate 1
}
```

9.4. Выполнение заданий по расписанию

Пользователи ОС Windows привыкли к тому, что существует Мастер планирования заданий, позволяющий автоматически запускать приложения в заранее назначенное время. В UNIX-подобных ОС есть еще более мощный и гибкий диспетчер расписаний. Его удобно использовать для автоматизации обслуживания системы и выполнения других задач, не требующих взаимодействия с пользователем. Например, в нерабочее время можно автоматически выполнять резервное копирование данных или обновление системы через Интернет.

Создавать расписания и ставить задания в очередь на выполнение имеют право все, кому это явно не запрещено суперпользователем.

Сердцем системы периодического выполнения служат демоны ***atd*** и ***cron*** (в Red Hat-совместимых дистрибутивах он называется ***crond***).

9.4.1. Запуск задания в назначенное время: команда at

Демон ***atd*** обслуживает задания, назначенные для однократного выполнения либо в установленное время, либо тогда, когда нагрузка на систему снизится до приемлемого уровня.

Для постановки задания в очередь для выполнения в указанное время служит команда at:

```
at [-f <файл>] ЧЧ:ММ
```

Файл — это исполняемый файл либо сценарий командного интерпретатора. Если он не указан, то вы должны будете вводить команды в командной строке, завершая ввод, как обычно, нажатием комбинации клавиш Ctrl+D.

Команда **at** допускает и более сложные спецификации времени, чем ЧЧ:ММ; вы можете указать дату или такое время, как midnight, noon или teatime. Подробности вы найдете на man-странице команды **at**.

Команда **atq** служит для просмотра очереди заданий. Для непривилегированного пользователя она выводит задания, поставленные в очередь им, а для суперпользователя — все задания. Удалить задание из очереди вы можете командой **atrm**.

Команда **batch** используется для тех заданий, которые заведомо потребуют много процессорного времени. Задание, поставленное в очередь при помощи этой команды, начинает выполняться только тогда, когда нагрузка на систему (среднее число процессов, ожидающих выделения им кванта времени) снизится до заранее установленного значения (по умолчанию 0.8; подробности — `man atd`).

Очередь заданий находится в каталоге `/var/spool/at`. Файлы `/etc/at.allow` и `/etc/at.deny` содержат списки тех пользователей, которым разрешено и запрещено пользоваться системой **at**.

9.4.2. Диспетчер расписаний — демон cron

Этот демон запускается во время инициализации системы (сценарий `/etc/init.d/crond`), читает свои конфигурационные файлы и переходит в режим ожидания. Раз в минуту демон просыпается, проверяет дату последнего изменения конфигурационных файлов, перечитывает те из них, которые оказались изменены, и выполняет задания, назначенные на данную минуту.

В конфигурационных файлах, называемых еще crontab-файлами, хранятся расписания: в каталоге `/var/spool/cron` — по одному на зарегистрированного пользователя и `/etc/crontab` — суперпользовательский. Выполнив задание, демон **cron** отправляет почтовое сообщение о результатах выполнения владельцу crontab-файла или пользователю, указанному в переменной MAILTO этого файла.

Управление файлами расписаний

Пользовательские crontab-файлы не предназначены для ручного редактирования. Для управления ими служит команда **crontab**. Список тех, кому разрешено ее запускать, находится в файле `/etc/cron.allow`. Если этот файл отсутствует, то запускать **crontab** могут все, кроме тех, кто перечислен в файле `/etc/cron.deny`. Если отсутствуют оба файла, то запускать команду может только суперпользователь.

Формат команды:

```
crontab [-u <логин_имя> ] [ <файл> | [-l] | [-r] | [-e] | [-i]]
```

Если указан файл, то этим файлом замещается crontab-файл указанного пользователя; если нет, то crontab-файл редактируется на месте. Ключи означают следующее:

- **-l** — вывести на консоль содержимое файла расписания;
- **-r** — удалить файл расписания;
- **-i** — удалить, предварительно переспросив;
- **-e** — редактировать файл расписания, то есть открыть его в редакторе, указанном в переменной окружения $EDITOR (по умолчанию — **vi**).

Если не указаны ни файл, ни ключи, то **crontab** будет читать файл расписания со своего стандартного ввода. Закончив ввод строк, нажмите Ctrl+D.

Формат файла расписания

Каждая незакомментированная строка файла расписания имеет следующий формат:

```
минута час день месяц день_недели [логин_имя] команда
```

Поля спецификации времени могут содержать:

- символ *, соответствующий любому значению;
- число: 0–59 для минуты, 0–23 для часа, 1–31 для дня, 1–12 для месяца, 0–7 для дня недели (воскресенье — это и 0, и 7);
- диапазон чисел, разделенных дефисом: например, 1-5 в поле день_недели означает «с понедельника по пятницу»;
- числа или диапазоны, разделенные запятыми, действующими как ИЛИ.

Поля «день месяца» и «день недели» тоже объединяются как ИЛИ: задание будет выполнено в день, удовлетворяющий хотя бы одной из этих спецификаций.

Команда — это любая командная строка, допустимая правилами интерпретатора **sh**.

Например, следующая запись означает, что архивирование каталога `/home/den` будет производиться каждый день, кроме воскресенья, в семь часов утра:

```
0 7 * * 1-6 tar cfz /backup/home.den.gz /home/den
```

В дистрибутивы Linux обычно входит готовый системный файл расписаний `/etc/crontab` и сценарии для выполнения стандартных задач по обслуживанию системы (ротации журналов, ведения базы установленных пакетов программ и т.п.), размещенные в каталогах `/etc/cron.hourly`, `/etc/cron.daily`, `/etc/cron.weekly` и `/etc/cron.monthly`. Директива run-parts в файле `/etc/crontab` указывает, что следует выполнить все исполняемые файлы из указанного каталога.

Глава 10

РЕЗЕРВНОЕ КОПИРОВАНИЕ И ВОССТАНОВЛЕНИЕ ДАННЫХ

- СТРАТЕГИЯ РЕЗЕРВНОГО КОПИРОВАНИЯ
- ОБОРУДОВАНИЕ ДЛЯ РЕЗЕРВНОГО КОПИРОВАНИЯ
- ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ДЛЯ РЕЗЕРВНОГО КОПИРОВАНИЯ
- ДУБЛИРОВАНИЕ ДАННЫХ: ВВЕДЕНИЕ В RAID
- КАК НАЙТИ, СПРЯТАТЬ И БЕЗВОЗВРАТНО УНИЧТОЖИТЬ ДАННЫЕ

В обязанности системного администратора входит обеспечение сохранности пользовательских данных. Потерять данные легко: их уничтожают ошибки в программном обеспечении, сбои оборудования, сами невнимательные пользователи. В случае домашнего компьютера разгильдяй-пользователь, теряющий данные, и несчастный системный администратор, которому приходится их восстанавливать, это обычно одно и то же лицо, но тем важнее уметь их восстановить. Пенять-то не на кого.

В этой главе я сначала рассмотрю способы вернуть случайно удаленный файл, а потом остановлюсь на принципах и средствах резервного копирования информации.

10.1. Восстановление удаленного файла

Из второй главы вы узнали, что файл на файловой системе ext2 (и ext3) представляет собой индексный дескриптор (*inode*), указывающий на блоки, в которых и размещаются данные. Содержимое блоков удаленного файла не исчезает сразу: эти блоки помечаются как свободные и отводятся для размещения других файлов по мере необходимости в свободном месте. Если файловая система используется не слишком интенсивно, блоки будут перезаписаны нескоро, и данные еще можно восстановить.

Первым делом размонтируйте раздел, содержавший удаленный файл. Это нужно для того, чтобы предотвратить перезапись его блоков во время попыток восстановления.

Если размонтировать файловую систему не удается (вы получили сообщение «device is busy»), то проверьте, какие процессы к ней обращаются, и прервите их. Это можно сделать с помощью команды

```
fuser -m <файловая_система>
```

Если удаленный файл находился на корневой файловой системе, то ее размонтировать нельзя. Вам придется выключить компьютер, извлечь жесткий диск и заняться восстановлением файла на другой Linux-машине. Отсюда мораль: при установке Linux разбивайте жесткий диск так, чтобы

файловые системы /usr, /var и особенно /home (те, в которых данные меняются чаще всего) размещались на отдельных разделах.

Linux — тщательно документированная система, и пути решения распространенных проблем описываются в документах HOWTO. Восстановлению файлов посвящено руководство www.tldp.org/HOWTO/Ext2fs-Undeletion.html (его русский перевод можно найти, например, на http://linux.vitebsk.by/howto/Ext2fs-Undeletion.html).

Для восстановления файлов на файловой системе ext2 (не ext3!) разработана утилита **e2undel** (http://e2undel.sourceforge.net). С ней вы разберетесь сами, а сейчас я скажу пару слов о средствах, входящих в состав обыкновенного дистрибутива.

10.1.1. Midnight Commander

Любимый многими файловый менеджер **mc** имеет в своем составе средство восстановления файлов, которое иногда может помочь. Запустите ***mc*** от имени суперпользователя. Выполните команду меню **Команда → Восстановление файлов**. Затем введите имя файла устройства, на котором находится нужный раздел, без /dev: например, hda4. Через некоторое время **Midnight Commander** представит вам список удаленных индексных дескрипторов. Если вы не знаете номер i-узла удаленного файла, вам придется просмотреть их все, чтобы найти нужный вам файл. Но гарантии того, что вы его найдете, нет никакой.

10.1.2. Утилита debugfs

Эта утилита входит в состав пакета e2fsprogs и служит для интерактивного исследования и изменения состояния файловых систем типа ext2 и ext3. Порядок ваших действий будет следующим:

1. Размонтируйте файловую систему с удаленным файлом.
2. Откройте ее в режиме «только чтение»:
   debugfs <файл_устройства>.
3. В ответ на приглашение введите подкоманду **lsdel**, чтобы получить список удаленных i-узлов. Попытайтесь определить свой файл по атрибутам: владельцу, размеру, дате удаления и т.п.
4. Сохраните содержимое i-узла в файл на другой файловой системе, введя подкоманду dump <inode> имя_нового_файла. Номер i-узла указывайте в угловых скобках!
5. Выйдите по подкоманде quit.

Список остальных подкоманд **debugfs** вы можете получить по подкоманде help или узнать на man-странице.

10.2. Стратегия резервного копирования

Чтобы героически спасать файлы приходилось не слишком часто, следует заранее позаботиться об их надежной защите. Важнейшим средством защиты является резервное копирование.

Вам нужно хорошо продумать следующие пункты:

1. **Какая информация будет резервироваться (архивироваться)?**
   В первую очередь вам нужно архивировать данные пользователей, то есть каталог `/home`. Эти данные относятся к наиболее критичной категории данных. Восстановить систему вы сможете в течение не более чем двух-трех часов, а вот данные пользователей уже не восстановишь...
   На втором месте — файлы настройки системы, находящиеся в каталоге `/etc`. Архивирование этих данных позволит существенно сэкономить время, которое вам потребуется на восстановление системы после сбоя.
   И, наконец, на третьем месте — дистрибутивы программ, не входящих в состав дистрибутива Linux. Эти данные, как правило, не нуждаются в частом обновлении.
2. **Когда будет происходить создание резервных копий?**
   Самое удачное время для этого мероприятия — ночь. Во-первых, архивирование обычно не требует вмешательства оператора и поэтому его можно выполнять автоматически (п.9.4.2). Во-вторых, оно создает дополнительную нагрузку на систему, чему не обрадуются пользователи. В-третьих, открытые этими пользователями файлы создадут при копировании массу проблем.
3. **Кто этим будет заниматься?**
   Если речь идет о вашем домашнем компьютере, то этой ответственной задачей будете заниматься вы сами. На предприятии (особенно большом) необходимо определить, кто будет архивировать данные с каждого сервера сети: не будете же вы бегать по зданию со стримером, контролируя процесс создания резервных копий? В идеале, за каждым сервером должен быть закреплен человек, ответственный за процесс создания архива и поддержания его в должном состоянии.
4. **Как часто будет производиться архивирование?**
   Дома можно архивировать диск еженедельно и после обширных изменений. На предприятии зарекомендовала себя шестидневная схема. Вам понадобится шесть сменных носителей (кассет для стримера или магнитооптических дисков). В пятницу, после конца рабочей недели, создается резервная копия всего диска на первой ленте. С понедельника по четверг архивируются только новые и обновленные данные (каждый день — на свою ленту), и в следующую пятницу создается новая копия всего диска на шестую ленту. Таким образом, всегда можно восстановить данные на любой день последней недели.

Архивирование только новых данных обычно называется инкрементным. Отбирать файлы, дата последнего изменения которых свежее заданной, умеет архиватор ***tar*** (см. man-страницу).

10.3. Оборудование для резервного копирования

При выборе стратегии резервного копирования решающим фактором может оказаться соотношение размера диска и емкости имеющихся у вас внешних носителей. Привычные диски CD-RW вмещают всего 640 Мбайт, поэтому резервирование сколько-нибудь значительного объема данных нельзя организовать автоматически: кто-то должен менять диски вручную. Вот почему стоит обратить внимание на стримеры, позволяющие сохранять от 10 до 100 Гбайт сжатой информации на одном картридже. К тому же UNIX-программы резервного копирования создавались в те времена, когда ленточные накопители были единственным типом внешнего хранилища, и идеально приспособлены к работе с ними.

10.3.1. Стример

Linux, как и любая UNIX-система, обладает богатыми возможностями по созданию и сопровождению резервных копий с помощью стримеров. Стример — это потоковый накопитель на магнитной ленте. Стримеры работают в безостановочном режиме, обеспечивают запись и считывание данных с ленты сплошным потоком.

Существует два типа стримеров: стримеры, использующие интерфейс SCSI, и стримеры, использующие интерфейс FDC. Первые из них довольно дороги, что объясняется дороговизной самого контроллера SCSI. Хотя в последнее время наблюдается снижение цен на контроллеры и устройства SCSI. Эти стримеры подключаются к шине SCSI.

Второй тип, использующий интерфейс FDC, подключается к контроллеру гибких дисков. Это более дешевый и медленный вариант, и поэтому, если вы собираетесь использовать стример в профессиональных целях, лучше приобрести стример с интерфейсом SCSI. Второй тип более подходит для домашнего применения.

Основным преимуществом стримеров является их низкая стоимость, но у них есть ряд недостатков:

- с ними не так удобно работать, как с жестким или магнитооптическим диском;

- скорость передачи данных низка (хотя к дорогим стримерам с интерфейсом SCSI это не относится).

Подключение стримера с интерфейсом SCSI

ОС Linux поддерживает все возможные стримеры с интерфейсом SCSI. Это объясняется интеллектуальностью контроллера SCSI. Вы также можете использовать интерфейс LUN (*Logical Unit Number*), который является расширением интерфейса SCSI, для подключения стримера с автоматической заменой ленты.

Для подключения стримера вам потребуется перекомпилировать ядро системы (глава 20), включив опцию **SCSI Tape Support**. Также вам нужно установить тип контроллера SCSI в подразделе **SCSI Low-Level drivers**. Возможно, нужно будет включить режим **Probe all LUNs on each device**. После перезагрузки в вашей системе появится устройство `/dev/st0`.

Подключение стримера с интерфейсом FDC

В зависимости от типа вашего стримера вам нужно включить опцию **QIC-02 tape support** или **Ftape (QIC-80/Trawan) support**. О том, какую из этих опций нужно использовать, вы можете прочитать в документации, поставляемой со стримером. Эти опции находятся в разделе **Character devices**. После перезагрузки должно появиться устройство /dev/nrtf0.

10.3.2. Магнитооптический диск

Я решил написать этот параграф в силу большой распространенности магнитооптических дисков. Первые магнитооптические диски подключались к контроллеру SCSI, что не способствовало их широкому распространению из-за довольно высокой стоимости. После выпуска первых устройств с интерфейсом IDE цены на магнитооптические устройства значительно снизились.

Подключение магнитооптического привода подобно подключению жесткого диска. При этом следует помнить простое правило: не нужно подключать к одной шине магнитооптический привод и жесткий диск. Логика проста: магнитооптические диски обладают довольно низкой производительностью по сравнению с жестким диском, и использование двух этих устройств на одной шине снизит общую производительность дисковой подсистемы.

После подключения не забудьте выполнить процедуру AUTODETECT для данного устройства. ОС Linux определит магнитооптический привод как обыкновенный жесткий диск с интерфейсом IDE. Если вы поспешили

и, запустив Windows, чтобы полюбоваться новой буквой в списке доступных дисков, отформатировали ваш магнитооптический диск, то, скорее всего, в Linux он будет работать некорректно. Для обеспечения нормальной работы магнитооптического привода в Linux запустите ***fdisk*** для Linux и удалите все разделы, которые создала Windows. Затем создайте один первичный раздел и командой t измените его тип на FAT32.

Хочу отметить, что FAT32 может работать довольно медленно, но позволит сэкономить около 80 Мб дискового пространства при использовании магнитооптического диска размером 640 Мб.

10.4. Программное обеспечение для резервного копирования

10.4.1. Простое резервное копирование по сети

Простейший способ сохранить каталог со своими данными — это перекинуть его по сети на другой компьютер. Утилита **scp** (*Secure Copy*) позволяет копировать каталоги в защищенном зашифрованном виде не только в пределах локальной сети, но и по Интернету. Например, вы можете скопировать свои данные с рабочей машины на домашнюю (чаще наоборот, потому что компьютер назначения должен быть, разумеется, включен и подключен к сети).

Утилита **scp** служит клиентом демона **sshd** (п.11.3.2), который должен быть запущен на компьютере назначения. Она устанавливается из того же пакета OpenSSH.

Чтобы скопировать каталог `/home/den/mywork` с узла saraksh в свой домашний каталог на узле dhsilabs, введите следующую команду:

```
[den@saraksh~]$ scp -r mywork dhsilabs:/home/den
```

Ключ –r указывает, что нужно рекурсивно копировать подкаталоги. Вместо имени узла dhsilabs можно (или нужно, если это имя неизвестно в службе DNS) использовать IP-адрес.

Если защищенность ваших данных в процессе передачи для вас не так важна, как способность инструмента отличать «свежие» файлы и копировать только их, пользуйтесь утилитой **rsync**. Она должна быть установлена на обоих компьютерах — локальном и удаленном. Чтобы скопировать с узла dhsilabs только те файлы каталога `mywork`, которые были обновлены с момента последнего запуска **rsync**, за вычетом подкаталога `backup`, введите команду:

```
[den@saraksh~]$ rsync -az --exclude=backup \
dhsilabs:/home/den/mywork /home/den
```

Ключ **-z** сжимает передаваемые данные. Утилита **rsync** не шифрует данных, но предоставляет возможность подключить для шифрования оболочку **ssh**.

10.4.2. Управление стримером

Управление стримером выполняет программа **mt**. Она входит в состав пакета `mt-st`, который обычно входит в состав дистрибутива (в дистрибутивах, основанных на Red Hat, он присутствует). Программа **mt** использует устройство `/dev/nftape`, которое является символической ссылкой на `/dev/nrft0`. Если вы используете стример с интерфейсом SCSI, вам нужно сделать его ссылкой на `/dev/st0`.

После подключения стримера необходимо подготовить ленту к работе. Вся подготовка состоит из перетяжки ленты и ее форматирования. При перетяжке с поверхности пленки снимаются статические заряды. Перетяжку можно выполнить командой:

```
$ mt-st -f /dev/nftape retension
```

А вот форматирование вам придется выполнять с помощью программы для DOS, которая поставляется со стримером. Можно, конечно, использовать и другие программы. Стабильно работают **Conner Backup Basics**, **Norton Backup**, а также **QICstream**. По завершении этого процесса требуется инициализировать ленту:

```
$ mt-st -f /dev/nftape erase
```

Вот теперь можно приступать к архивации данных. Например, если вы хотите записать на ленту содержимое своего домашнего каталога, вы можете использовать следующую команду:

```
$ tar cfz /dev/nftape /home/den
```

Здесь я использовал команду **tar**, которая изначально предназначалась для работы с лентой (это видно из ее названия — *Tape Archive*). Ключ z указывает программе **tar** на необходимость сжать данные. Для записи без сжатия достаточно ключей cf. Для восстановления архива с ленты вы можете использовать команду

```
$ tar xfz /dev/nftape
```

Если вы не сжимали данные, то разархивируйте их с ключами xf.

Проверить целостность архива на ленте можно с помощью команды

```
$ tar df /dev/nftape
```

Чтобы разместить на одной ленте два или более архивов **tar**, вам понадобятся средства позиционирования головки и перемотки ленты. Это

делается при помощи программы **mt**. Например, перемотать ленту на две отметки начала файла в формате tar вперед можно таким образом:

$ mt -f /dev/nftape fsf 2

Для перемотки назад вместо операции **fsf** нужно использовать операцию **bsf**. Список операций команды **mt** вы можете найти на man-странице, а в таблице 10.1 приведены чаще всего употребляемые.

*Операции программы **mt*** *Таблица 10.1*

Операция	Назначение
eod	Перемотать ленту к концу записанных данных. Применяется для дозаписи файлов на ленту
eof	Записать метку конца файла в текущую позицию
erase	Стереть ленту
rewind	Перемотать ленту к началу
retension	Несколько раз промотать ленту, чтобы снять статические заряды
offline eject	Перемотать ленту к началу и извлечь из накопителя
fsf <число>	Перемотать ленту на <число> файлов вперед
bsf <число>	Перемотать ленту на <число> файлов назад
asf <номер>	Перемотать ленту к началу файла с указанным номером

10.4.3. Команды dump и restore

Пара утилит — **dump** и **restore** — это самое распространенное средство резервирования и восстановления данных в UNIX-системах. Они обслуживают файловую систему ext2 (ext3), то есть для архивирования ваших Windows-разделов их использовать не удастся. Эти утилиты входят в состав пакета dump, который в дистрибутивах Red Hat устанавливается по умолчанию; если в вашем дистрибутиве этого пакета нет, то скачайте его с `http://dump.sourceforge.net`.

Программа **dump** позволяет организовать инкрементное резервирование, сжимать полученный архив и разбивать его на тома, когда его размер превышает емкость внешнего носителя. Формат вызова:

```
dump [-уровень] [ключи] список_файлов
```

Номер уровня — это средство, при помощи которого программа dump выполняет инкрементное архивирование. Исторически использовались номера от 0 до 9, Linux-версия программы понимает любое целое число. Уровень 0 соответствует полному копированию всей файловой системы. Уровень N архивирует только те файлы, которые изменились с момента создания последнего архива уровня ниже N. Уровень архива и время его создания отмечаются в файле `/etc/dumpdates`, если команда `dump` запущена с ключом -u.

В качестве списка файлов можно указывать не только файлы и каталоги на смонтированных файловых системах, но и файл устройства, на котором находится файловая система, в данный момент размонтированная. Единственное ограничение — такую файловую систему можно архивировать только полностью (уровень 0).

Полный список ключей команды dump вы найдете на ее man-странице. Вот наиболее важные из них:

- **-f <файлы>** — список имен файлов, разделенный запятыми. Тома многотомного архива выводятся в указанные файлы в порядке их перечисления. Это могут быть файлы на диске, файлы устройств, стандартный вывод (символ «-»);
- **-u** — после успешного архивирования отметить дату и уровень в файле `/etc/dumpdates`;
- **-z[уровень_сжатия]** — сжимать выводимый архив средствами библиотеки zlib. Значение уровня сжатия по умолчанию равно 2; другой уровень нужно указывать без пробела.

Программа **restore** восстанавливает отдельные файлы и файловые системы, архивированные программой **dump**. Формат команды:

```
restore [режим] [ключи]
```

Важнейших режимов два:

- **-i** — интерактивное восстановление отдельных файлов и каталогов. Подкоманды этого режима (список которых можно получить по подкоманде **help**) позволяют перемещаться по архиву, оглавление которого прочитано с ленты, как по обычному дереву каталогов, и выбирать файлы для восстановления. Выбранный файл добавляется в список восстановления подкомандой **add**. Список восстановления извлекается с ленты подкомандой **extract**.
- **-r** — полное восстановление файловой системы. Пострадавшую файловую систему нужно сначала отформатировать командой **mkfs** (п.2.3), смонтировать и переместиться в ее корневой каталог. Потом восстановить с ленты архив уровня 0, а потом — все инкрементные архивы в порядке возрастания номера уровня.

10.4.4. Архиватор cpio

GNU-версия этого классического архиватора (*Copy In/Out*), используемая в Linux, поддерживает архивы как в собственном формате, так и в формате ***tar***. Программа ***cpio*** может работать в одном из трех режимов в зависимости от первого ключа:

- `cpio -o [ключи] < список_файлов > архив` : режим *copy-out*, копирование файлов в архив. Список файлов, по одному в строке, подается на стандартный ввод программы; в архив перенаправляется поток стандартного вывода. В качестве архива можно указывать файл на диске, файл устройства магнитной ленты или канал: например, можно пропустить вывод через какую-нибудь программу сжатия. Список файлов можно подготовить вручную, а можно использовать поток вывода другой программы, например, `ls -1` (с ключом «дефис-единица», выводящим файлы по одному в строке) или **find**.
- `cpio -i [ключи] [шаблоны] < архив` : режим *copy-in*, извлечение файлов из архива или вывод содержания архива. Архив читается из стандартного потока ввода. Извлечению подлежат только те файлы, имена которых удовлетворяют одному из шаблонов, перечисленных через пробел; если шаблоны не указаны, то извлекаются все файлы. Внимание: символы подстановки не совпадают с теми, которые использует оболочка. Символу «.» удовлетворяет любая подстрока в начале образца, а символу «/» в имени файла — любой набор символов.
- `cpio -p [ключи] каталог_назначения < список_файлов` : режим *copy-pass*, копирование дерева каталогов.

О ключах команды **cpio** можно узнать из ее man-страницы. Приведу несколько примеров ее использования.

Архивирование в формате **tar** под именем `archive.tar` файлов, имена которых вы вводите в командной строке:

```
$ cpio -o -H tar -O archive.tar
/opt/ctrl/ctrl.c
/opt/ctrl/ctrl.h
/opt/ctrl/ctrl.html
^D
$
```

Архивный файл будет создан с сохранением структуры каталогов. Чтобы убедиться в этом, запустите файловый менеджер **mc** и просмотрите содержимое архива `archive.tar`.

Архивирование текущего каталога с использованием перенаправления вывода команды **ls**:

```
$ ls -1 | cpio -o -H tar -O current_dir.tar
```

Извлечение файлов из архива:

```
$ cpio -i < current_dir.tar
```

В режиме *copy-in* ключ формата «`-H tar`» указывать необязательно, потому что программа ***cpio*** автоматически определяет формат входного архива.

Просмотр содержимого архива на магнитной ленте:

```
$ cpio -i -t < /dev/nftape
```

10.4.5. Программа AMANDA

Этот параграф я помещаю здесь, так сказать, «на вырост». Администратору домашней сети он вряд ли пригодится.

Программа AMANDA (*The Advanced Maryland Automatic Network Disk Archiver*) — это система резервного копирования, которая позволяет администратору локальной сети установить один главный бэкап-сервер для резервного копирования множества узлов локальной сети. При этом вам не придется подходить к каждому компьютеру, чтобы сделать резервную копию. AMANDA использует стандартные программы ***dump*** и ***tar***, поэтому вы можете выполнить резервное копирование большинства UNIX-станций. Последние версии программы позволяют выполнять резервное копирование рабочих станций MS Win9x/NT, но для этого нужно установить пакет SAMBA.

Скачать программу вы можете с сайта разработчика `www.amanda.org`. Вам потребуются 3 пакета: amanda, amanda-client, amanda-server. Первый пакет устанавливается как на сервере, так и на клиенте; второй пакет — только на клиенте, а третий — только на сервере.

Я рекомендую загрузить самую последнюю версию. Как правило, самые новые версии распространяются не в виде RPM-пакета, а в виде так называемого тарболла — файла с расширением tar.gz. Распаковав архивы, выполните команду:

```
$ ./configure --with-config=network --with-user=amanda
--with-group=operator
```

Опции **with-user** и **with-group** указывать обязательно. Желательно перед выполнением данной команды создать пользователя amanda и поместить его в группу `operator`. Название нашей конфигурации — network. Так же будет называться каталог с резервными копиями.

Если вы планируете делать резервное копирование с Windows-машины, нужно добавить опцию `--with-smbclient=/path/to/smbclient`.

Обратите внимание на то, что:

- На клиенте сначала нужно установить пакет amanda, затем — amanda-client. На сервере вместо пакета amanda-client нужно установить пакет amanda-server.
- Пользователя amanda нужно добавить на всех машинах сети, с которых предполагается архивирование данных.

После успешного завершения программы `configure` введите команды `make` и `make install`.

Основными конфигурационными файлами AMAND'ы служат `/etc/amanda/amanda.conf` и `/etc/amanda/disklist`. В этих файлах находится информация о стримере, о разделах, которые вы хотите резервировать, а также другая важная для AMAND'ы информация. Пока воспользуемся файлами по умолчанию. Откройте их и измените только специфические для вашей сети параметры.

В файле `disklist` нужно прописать имена узлов сети, которые вы хотите архивировать. При этом лучше прописывать IP-адреса, поскольку некоторые версии AMAND'ы иногда не могут преобразовать имя узла в IP-адрес (почему — сам не знаю, ведь все должно работать через DNS).

Теперь приступим к настройке узлов, резервную копию которых вам нужно создать. Если эти узлы используют устаревший суперсервер **inetd**, в файл `/etc/inetd.conf` нужно добавить такие строки:

```
amanda dgram udp wait amanda /usr/libexec/amandad amandad
amandaidx stream tcp nowait amanda /usr/libexec/amindexd amindexd
amidxtape stream tcp nowait amanda /usr/libexec/amidxtaped amidxtaped
```

Если же используется суперсервер **xinetd**, то в файл `/etc/xinetd.conf` нужно добавить следующие строки:

Листинг 10.1. Конфигурирование службы AMANDA

```
service amanda
{
 protocol     = udp
 socket_type    = dgram
 wait       = no
 user       = amanda
 server       = /usr/libexec/amandad
 log_on_failure   += USERID
}
service amandaidx
{
 protocol     = tcp
 socket_type    = stream
 wait       = no
 user       = amanda
 server       = /usr/libexec/amindexd
 log_on_failure   += USERID
}
service amidxtape
```

```
{
 protocol     = tcp
 socket_type    = stream
 wait       = no
 user       = amanda
 server       = /usr/libexec/amidxtaped
 log_on_failure   += USERID
}
```

Затем в файл /etc/.rhosts нужно добавить строку:

```
192.168.0.1 amanda
```

Чтобы изменения вступили в силу, желательно перезагрузить компьютер или перейти на первый уровень выполнения и обратно по команде **telinit**.

Теперь нужно пометить кассеты стримера. Это позволяет сделать команда **amlabel**. Зарегистрируйтесь как суперпользователь, вставьте в стример первую пленку и введите команду:

```
# amanda -c "amlabel network tape1"
```

Здесь «network» — это имя конфигурации, заданное в файле amanda.conf, а «tape1» — это метка вашей пленки. AMANDA запишет имя пленки в список пленок, поэтому у вас никогда не будет двух кассет с одной меткой.

Вам больше не нужна какая-нибудь лента? Для ее удаления из списка введите команду:

```
# amanda -c "amrmtape network tape1"
```

Команда amcheck позволяет проверить созданную вами конфигурацию и сообщить о возможных ошибках:

```
# amanda -c "amcheck network"
```

Вот теперь можно приступить к резервному копированию. Введите команду:

```
# amanda -c "amdump network" &
```

Архивирование данных по сети обычно занимает много времени, поэтому его следует запускать в фоновом режиме. Чтобы узнать состояние процесса архивирования, введите команду:

```
# amanda -c "amstatus network"
```

Если у вас возникла необходимость восстановить какую-нибудь рабочую станцию, зарегистрируйтесь на ней как суперпользователь, перейдите в корневой каталог и введите команду:

```
# amanda -c "amrestore network"
```

Программа amrestore соединится с сервером кассет. В ответ на приглашение сервера введите:

```
setdisk sd0f
```

После этого вы сможете просматривать файлы и каталоги резервной копии так же, как при работе с FTP-сервером. Если вы найдете файл или каталог, который хотите восстановить, введите команду:

```
add имя_файла_или_каталога
```

Выбранный вами файл будет добавлен в список восстановления программы amrestore. Для извлечения добавленных в этот список файлов введите команду extract.

За дальнейшими инструкциями обратитесь к документации по системе AMANDA.

10.5. Дублирование данных: введение в RAID

Идея надежности хранения данных волновала, волнует и будет волновать не одно поколение системных администраторов и пользователей. Используемые ОС Linux файловые системы ext2 и ext3 обладают достаточной степенью надежности, но зачастую этого мало.

Если существует вероятность потерять данные в результате выхода из строя жесткого диска, то единственным выходом из данной ситуации является использование массивов жестких дисков RAID. RAID (*Redundant Array of Independent (Inexpensive) Disk*) — это способ хранения данных с избыточностью на группе независимых (недорогих) жестких дисков. Под избыточностью понимается дублирование данных и хранение дополнительных кодов коррекции ошибок. Всего существует 6 уровней RAID (таблица 10.2).

Чаще всего используются массивы уровней 0, 1 и 5. Иногда встречаются комбинированные способы объединения данных в массив, например RAID-10 (RAID 0+1) — это чередование блоков данных на двух парах дублирующих друг друга дисководов.

RAID-5 использует дисковое пространство экономнее, чем RAID-1, так как избыточность представляет собой не полную копию информации, а только контрольную сумму. Но за высокую эффективность использования пространства приходится платить более низкой скоростью обмена данными.

Уровни RAID *Таблица 10.2*

Уровень	Описание
0 («полосатый», striping)	Группа дисководов без избыточности. Этот уровень предназначен для хранения больших объемов данных, не умещающихся на одном диске, и ускорения доступа к ним. Последовательные блоки одного файла хранятся на разных дисководах. Емкость массива равна суммарной емкости всех дисков, образующих массив
1 (дисковое зеркало, mirroring)	Дисководы, входящие в группу, содержат одинаковые данные и образуют один логический диск. Благодаря этому скорость чтения (но не записи) возрастает вдвое. Емкость массива равна емкости самого меньшего из дисков
2	Запись на разные дисководы производится методом чередования «страйпов» размером в один сектор с добавлением кодов исправления ошибок
3	То же, что уровень 2, но контрольные коды записываются на отдельный диск. При отказе одного из дисков оставшиеся диски можно использовать для восстановления хранившейся на нем информации
4	То же, что уровень 3, но размер страйпов — несколько секторов
5 (массив с вращающейся четностью)	То же, что уровень 4, но контрольные суммы хранятся не на одном дисководе, а на всех по очереди. При выходе из строя одного из дисков потерянные данные восстанавливаются с помощью контрольных сумм. Общая емкость массива вычисляется по формуле min_size*(n-1), где min_size — объем наименьшего из дисков, а n — количество дисков в массиве. Минимальное количество дисков равно трем

Организация массива RAID доступна не каждому из-за все еще высокой стоимости аппаратных контроллеров RAID. Хотя производители материнских плат пытаются поправить это, выпуская материнские платы со встроенными контроллерами RAID, но такие контроллеры не слишком универсальны и обладают слабыми возможностями.

ОС Linux поддерживает программные контроллеры RAID. Применение программных контроллеров имеет как свои преимущества, так и недостатки. К достоинствам относится возможность использования дисков с различными интерфейсами, например SCSI и IDE, для организации массива — программному контроллеру все равно, с чем работать. Недостатком же является дополнительная нагрузка на центральный процессор — он выполняет всю работу по обеспечению функционирования массива RAID.

Итак, приступим к созданию массива RAID. Вам понадобится любой дистрибутив с поддержкой программного контроллера RAID: такой возможностью обладают практически все современные дистрибутивы. Для включения поддержки RAID вам придется перекомпилировать ядро.

Включить поддержку RAID можно в разделе **Block devices** конфигуратора ядра (`make menuconfig`). Нужная опция называется **RAID n support**, где n — это номер уровня массива RAID. После этого нужно установить пакет raidtools, в состав которого входят программы ***raidhotadd***, ***raidhotremove***, ***mkraid*** и другие.

Для организации массива уровня RAID 1 нужно выделить два раздела и изменить тип этих разделов на Linux raid autodetect. Обратите внимание, я написал «два раздела», а не «два диска», так как конфигурируется программный контроллер. Конечно, лучше, чтобы эти разделы располагались на разных дисках, в противном случае от нашего массива будет мало толку.

Если ваше ядро поддерживает RAID, при загрузке системы вы должны увидеть примерно следующее:

```
md driver 0.90.0 MAX_MD_DEVS=256, MAX_REAL=12
raid5: measuring checksumming speed
raid5: MMX detected, trying high-speed MMX checksum routines
   pII_mmx : 980.694 MB/sec
   p5_mmx :  999.744 MB/sec
   8regs :   753.237 MB/sec
   32regs :    444.246 MB/sec
using fastest function: p5_mmx (999.744 MB/sec)
md.c: sizeof(mdp_super_t) = 4096
Partition check:
hda: hda1 hda2 < hda5 hda6 hda7 hda8 >
autodetecting RAID arrays
autorun ...
... autorun DONE.
```

Теперь отредактируйте файл /etc/raidtab.

Листинг 10.2. Примерный файл /etc/raidtab для уровня 1

```
# Имя устройства RAID
raiddev /dev/md0
# Уровень
raid-level 1
chunk-size 8
persistent-superblock 1
# Число дисков в массиве
nr-raid-disk 2
# Число дисков, которые будут использоваться в качестве замены,
# если один из дисков выйдет из строя
nr-spare-disk 0
```

```
# Определяем первый диск RAID
device /dev/hdb1
raid-disk 0
# Определяем второй диск RAID
device /dev/hdc1
raid-disk 1
```

После этого нужно создать устройство /dev/md0, для чего выполните следующую команду:

```
# mkraid /dev/md0
```

В некоторых случаях нужно будет использовать дополнительные параметры, о которых вы можете прочитать в справочной системе (man mkraid). Если инициализация прошла успешно, в файле /proc/mdstat вы увидите примерно следующее:

```
Personalities: [raid1]
read_ahead 1024 sectors
md0: active raid1 hdc1[1] hdb1[0]
```

10.6. Как найти, спрятать и безвозвратно уничтожить данные

Просмотреть содержимое ASCII-файла нетрудно, но что, если вам нужно узнать, что находится в двоичном файле неизвестного формата? Можно использовать шестнадцатеричный просмотрщик **hexdump**, который в зависимости от указанных ключей выводит содержимое файла в десятичном (-d), восьмеричном (-o) или каноническом ASCII+шестнадцатеричном (-C) виде:

```
$ hexdump -C /sbin/init | less
...
0000013001 00 00 00 47 4e 55 00 00 00 00 00 02 00 00 00    |....GNU.........|
...
```

Но таким способом искать в двоичном файле текстовые фрагменты не слишком удобно. Утилита **strings** извлекает из файла последовательности печатных символов длиной не меньше указанной (по умолчанию 4 символа). В качестве файла можно указать весь дисковый раздел:

```
$ strings -n 18 /dev/hda4 | grep "образец для поиска"
```

Особенности файловой системы ext2 подсказывают, что данные можно хранить и не в файлах. В самом деле, файловая система адресует блоки

размером 1, 2 или 4 Кб, но что если «хвост» файла не занимает блока целиком? Это место, «резерв» (*slack*), остается неиспользованным. Существуют утилиты, позволяющие записывать данные в «резервы», читать эти данные и затирать их, если необходимо. Эти данные будут невидимы для файловой системы, недоступны для обычного использования и необнаружимы средствами проверки целостности файла (*integrity checkers*). Таким способом можно хранить секретные данные.

Одна из таких утилит — **bmap**

(`ftp://ftp.scyld.com/pub/forensic_computing/bmap`).

Например, вот как эта команда записывает данные в «резерв», созданный файлом `/etc/passwd`:

```
# echo "здесь спрятана строка" | bmap --mode putslack /etc/passwd
```

А вот так эта команда показывает данные:

```
# bmap --mode slack /etc/passwd
getting from block 887048
file size was: 9428
slack size: 2860
block size: 4096
здесь спрятана строка
```

Для затирания «резерва» (удаления скрытой информации) используется команда:

```
# bmap --mode wipeslack /etc/passwd
```

Предотвратить восстановление удаленных файлов с конфиденциальной информацией можно, забив нулями освободившиеся блоки. Самый простой способ сделать это состоит в использовании стандартной Linux-утилиты **dd**. Вытереть пустое место на разделе `/home` можно так:

```
$ dd if=/dev/zero of=/home/bigfile
$ sync
$ rm /home/bigfile
$ sync
```

Утилита **shred** из пакета `coreutils`, входящего в подавляющее большинство дистрибутивов Linux, затирает файл перед его удалением, несколько раз записывая на его место случайные данные. К сожалению, ее применение имеет смысл не на всех файловых системах, а только на тех, которые переписывают файл «на месте». Файловые системы со включенным журналированием (п. 2.2.2), избыточностью (RAID), кэширующие и сжатые не позволяют удалить файл без возможности его восстановления.

Остатки конфиденциального файла могут уцелеть в аварийном дампе памяти, страницах памяти, выгруженных на диск (раздел подкачки), так что в целях дополнительной безопасности следует затирать весь раздел, `/swap` и (на всякий случай) раздел с `/tmp`. Утилита **shred** разрешает указывать в качестве файла, подлежащего затиранию, файл устройства (например, `/dev/hda4`).

На сайте The Hacker's Choice (`www.thc.org`) можно скачать пакет THC-SecureDelete, а также узнать побольше о безопасности данных и борьбе с ней.

Глава 11

БАЗОВОЕ КОНФИГУРИРОВАНИЕ СЕРВЕРА

- СЕРВЕРНЫЕ ТЕХНОЛОГИИ LINUX
- ОРГАНИЗАЦИЯ И СОСТАВ LINUX-СЕРВЕРА
- СУПЕРСЕРВЕР XINETD
- УДАЛЕННЫЙ ДОСТУП: SSH И TELNET

11.1. Серверные технологии Linux

Каждая операционная система имеет свое «призвание». Операционную систему Windows NT Server предпочтительнее использовать как сервер рабочих групп сетей Microsoft. Система Novell Netware лучше «смотрится» в роли файлового сервера и сервера печати. ОС UNIX первоначально разрабатывалась как интернет-сервер.

Средства для работы с Сетью встроены непосредственно в ядро этой операционной системы, а все необходимое программное обеспечение для организации сервера входит в состав дистрибутива. UNIX-система работает со всеми сетевыми протоколами (особенно с TCP/IP) лучше, чем любая другая операционная система для платформы Intel. Недаром говорят, что UNIX создан для сети, как птица для полета. Все перечисленные выше качества касаются также и ОС Linux.

Еще один важный аспект — документация системы. Все без исключения Unix-подобные системы очень хорошо документированы, и поэтому вся необходимая информация для настройки сервера, по сути, уже есть в вашем компьютере.

Где же применяются Linux-серверы? Прежде всего, это интернет-серверы. Вы можете спросить, почему именно Linux (Unix)? Почему не какая-нибудь другая операционная система, например, Windows NT (2000)? Давайте подумаем вместе. В начале 60-х годов по приказу Министерства обороны США была создана сеть Arpanet, которая и послужила в дальнейшем прототипом для создания Интернет. Как можно использовать NT-сервер в качестве интернет-сервера, если он был выпущен в 1996 году? А Интернет-то существовал с 70-х годов. И существовал именно благодаря Unix-системам. Так почему же не использовать для предоставления интернет-услуг родную операционную систему?

Многие правительственные и финансовые организации всего мира, например, Министерство иностранных дел Германии, используют Linux (SuSE Linux), а немецкий Dresdner Bank совместно с американской компанией CollabNet объявил о новой банковской информационной системе, построенной на основе Linux. И тут, как вы видите, дело не в деньгах — платить

или не платить за Linux, а в заботе организаций о своей информационной безопасности и надежности своих серверов. Как объяснить клиенту, что его счет «будет закрыт», поскольку «программа выполнила недопустимую операцию»? Тут даже созданный журнал ошибок не поможет. Недавно открытый сайт президента России, к которому предъявляются повышенные требования надежности, безопасности и производительности, базируется именно на основе Red Hat Linux.

Второй отраслью применения Linux-серверов является создание кластеров для произведения параллельных вычислений. По определению кластер — это несколько компьютеров, объединенных вместе для совместного решения одной задачи. Объединение компьютеров, как правило, производится с помощью высокоскоростной сети. На сегодняшний день создано специальное программное обеспечение, позволяющее собрать кластер даже в домашних условиях, например, PVM (Parallel Virtual Machine).

Помимо всего вышеуказанного, существует еще множество направлений, где используются Linux-серверы: WWW-серверы, FTP-серверы, почтовики, шлюзы, можно даже эмулировать домен NT с помощью пакета Samba. При этом нужно учитывать то, что все необходимое программное обеспечение уже входит в ваш дистрибутив:

1. веб-сервер Apache
2. FTP-серверы wu-ftpd и ProFTPD
3. Агенты MTA (Mail Transfer Agant) sendmail и postfix
4. Поддержка сети Microsoft — пакет Samba
5. DHCP (Dynamic Host Configuration Protocol)-сервер, который используется для автоматического назначения IP-адреса рабочим станциям в сети
6. Прокси-сервер SQUID
7. Брандмауэр IpChains и/или IpTables
8. Сервер баз данных MySQL
9. DNS-сервер
10. Специальный прокси-сервер Socks5.

Подробно о настройке всех этих служб, а также о многом другом вы сможете прочитать в моей книге «Linux-сервер своими руками», третье издание которой вышло в 2005 году в издательстве «Наука и Техника».

11.2. Организация и состав Linux-сервера

Что такое сервер? С точки зрения пользователей сети, сервер — это удаленный компьютер, выполняющий некоторые функции, например, прием и отправку электронной почты. С точки зрения нас, администраторов, сервер — это программа, выполняющая определенные функции. Раз

уж мы рассматриваем почтовый сервер, то на этом компьютере должна быть установлена специальная программа, которая будет отправлять и принимать сообщения.

Компьютер без этой программы (которую мы также будем называть сервером) — это просто рабочая станция. Мало просто установить операционную систему Linux, нужно еще установить программы-серверы, которые будут выполнять те самые функции, которых ждут от сервера.

Построение Linux-сервера нужно начать с настройки суперсервера — **xinetd** (в более старых дистрибутивах — **inetd**). Суперсервер — это основа основ: без него не будет работать большинство сетевых сервисов, таких как POP3, IMAP, FTP (если он не запускается отдельно). Сервер **xinetd** называется суперсервером, потому что он отвечает за установление TCP-соединения, то есть прослушивает пакеты и передает их на обработку другим программам, управляя таким образом другими серверами. Например, если в запросе клиента будет требование установить соединение с двадцать первым портом, то суперсервер вызовет сервер FTP: конечно, при условии, что соединение с 21 портом разрешено (в противном случае клиент получит сообщение «Connection refused»).

Примечание

По правде говоря, не все так просто, как я описал. На практике за установление TCP-соединений отвечает демон **tcpd** (в ранних версиях Linux его не было). Программы-сервисы (**httpd, ftpd**) могут постоянно находиться в памяти (режим standalone): в этом случае они сами обрабатывают пакеты, и суперсервер их уже не вызывает. Но это уже детали, и они картины не меняют.

Отнеситесь к настройке **xinetd** с должным вниманием: от того, как вы его настроите , будет зависеть работа вашего сервера.

После настройки **xinetd** можно приступить к настройке конкретных сетевых сервисов. Я вам рекомендую настраивать не все подряд (по принципу «чтобы было»), а только те сервисы, которые вам сейчас нужны. Например, если вы настраиваете почтовик, то нечего на нем устанавливать DNS-сервер, веб-сервер и FTP-сервер. Лучше хорошо настроить две-три нужные службы, чем использовать десяток (включая нужные и ненужные) с настройками «по умолчанию». Помните, что ненастроенная служба — это потенциальная дыра в системе безопасности вашего сервера.

Если вы настраиваете серверы провайдера, я настоятельно рекомендую разделить серверные функции между разными компьютерами. Не нужно, чтобы один компьютер обслуживал и входящие звонки, и почту, и разрешение имен. Для провайдера целесообразно разделить функции серверов так:

- Два отдельных DNS-сервера — первичный и вторичный. Пусть себе стоят в углу — DNS-серверы редко требуют администрирования, если, конечно, вы не выдаете доменные имена по десять раз за день.
- Главный сервер, который обслуживает одновременно и входящие звонки (dial-in), и почту.
- Отдельный веб-сервер. Обычно на веб-сервере устанавливаются интерпретаторы PHP, perl и сервер баз данных MySQL. Если пользователям нужен доступ к их файлам, можно настроить на этом же компьютере FTP, но я рекомендую вместо FTP использовать ssh. Конечно, если ваши пользователи пойдут на такие жертвы ради безопасности.

Если же вы ограничены в средствах, можно все это добро установить на одном компьютере. Надежность схемы «все в одном» значительно ниже, и в основном она используется для тестирования — проведения небольших экспериментов с сетевыми сервисами.

Сейчас мы подробно разберемся, что нужно устанавливать на ваш будущий Linux-сервер. Предположим, что вам нужно настроить веб-сервер. Тогда нужно установить и настроить следующее программное обеспечение:

- Суперсервер **xinetd** — вы уже знаете, для чего он нужен. Настройку любого Linux-сервера нужно начинать именно с настройки **xinetd**.
- Пакет apache (в некоторых дистрибутивах он называется **httpd**). Программа **Apache** выполняет функции веб-сервера: именно она передаст пользователю веб-страницу, когда тот введет URL страницы в окне браузера.
- Если ваши пользователи желают программировать на PHP, нужно установить пакет PHP. Связке веб-сервера Apache, интерпретатора PHP и сервера баз данных MySQL посвящена целая глава, поэтому мы не будем сейчас подробно на этом останавливаться.

Теперь рассмотрим второй распространенный случай — почтовый сервер. Почтовый сервер отвечает за отправку и прием сообщений электронной почты. Обычно он использует протоколы SMTP (*Simple Mail Transfer Protocol*) и POP (*Post Office Protocol*). Для создания почтового сервера нужно установить и настроить следующее программное обеспечение:

- Суперсервер **xinetd**.
- Почтовый агент (MTA, *Mail Transfer Agent*), которая будет отправлять и принимать сообщения. Обычно эту роль выполняет программа **sendmail**, а кроме нее довольно распространены программы **qmail** и **postfix**, выполняющие аналогичные функции.
- Пакет imap, обеспечивающий получение пользователями своей почты по протоколам POP3 или IMAP.

- Программу **procmail**, сортирующую почту. С ее помощью можно организовать автоответчик и другие полезные услуги.
- Программу **fetchmail**, позволяющую получать почту с других POP3-серверов.
- Желательно также установить какой-нибудь антивирус, например, KAV, и «прикрутить» его к **sendmail** (или другому SMTP-серверу): тогда входящие и исходящие сообщения будут автоматически проверяться на вирусы.

Третьим распространенным примером Linux-сервера является DNS-сервер. Для его организации вам достаточно установить и настроить суперсервер **xinetd** и сам DNS-сервер — пакет bind.

Следующий пример Linux-сервера — это шлюз. Шлюз позволяет соединить две сети различного типа, например, локальную сеть и Internet. Для его создания нужно установить и настроить следующее программное обеспечение:

- Суперсервер **xinetd**.
- Пакет **ppp**, содержащий сервер для работы с протоколом PPP (*Point to Point Protocol*). Данный пакет нужно устанавливать лишь в том случае, если вы сами подключаетесь к Интернету по протоколу PPP или вы настраиваете сервер входящих звонков (dial-in) для доступа удаленных пользователей в Интернет через ваш компьютер.
- Пакет **iptables** — это межсетевой экран, или брандмауэр. Именно эта программа будет выполнять функции шлюза, то есть переадресовывать пакеты из одной сети в другую (из локальной в глобальную). Кроме того, брандмауэр может отфильтровывать пакеты, что позволяет оградить вашу сеть от вторжения извне.
- Пакет **squid** — это прокси-сервер, который кэширует веб-страницы.

В таблице 11.1 перечислены лишь немногие функции, выполняемые сервером, и рекомендуемое программное обеспечение, которое необходимо для реализации этих функций. В графе «Дистрибутив» сказано, входит ли нужный пакет в состав популярных дистрибутивов.

Серверное программное обеспечение *Таблица 11.1*

Функция	Программное обеспечение	Дистрибутив
Авторизация удаленных пользователей (dialup)	ppp	Да
Автоматическое конфигурирование узлов сети	dhcp	Да
Совместный доступ к файлам	NFS, FTPd (ProFTPD, wu-ftpd)	Да
Доступ к сети Microsoft	samba	Да
Кэширование передаваемой информации	Squid	Да
Маршрутизация	route(d)	Да
Обмен сообщениями электронной почты	sendmail postfix qmail imap	Да Да Нет Да

Продолжение табл. 11.1

Функция	Программное обеспечение	Дистрибутив
Подсчет передаваемого по сети трафика	ядро Linux, IPChains (в старых дистрибутивах с ядром 2.2.x и меньше) или IPTables (в новых дистрибутивах с ядром 2.4.x и 2.6.x)	Да
Передача секретной информации	modSSL	Нет
Разрешение имени компьютера в IP-адрес	bind	Да
Сетевая печать	Lpd, Samba, CUPS	Да
Функции веб-сервера	apache	Да
Фильтрация пакетов IP-Маскарадинг	IPTables (IPChains в старых версиях Linux)	Да
Управление базой данных	MySQL PostgreSQL InterBase	Да Да Нет

11.3. Суперсервер xinetd

11.3.1. Установка суперсервера xinetd

Суперсервер **xinetd** появился в дистрибутивах давно (например, в Red Hat начиная еще с 7 версии) и стал достойной заменой **inetd**, использовавшемуся до него. Суперсервер **xinetd** обычно устанавливается по умолчанию во время установки системы. Одним из преимуществ этого суперсервера является наличие встроенных механизмов защиты, которые для **inetd** реализует отдельный демон **tcpd**. К тому же **xinetd**, в отличие от **inetd**, поддерживает IPv6. Даже если вы пока не планируете переходить на IPv6, установка **xinetd** будет очень полезной из-за расширенных функций суперсервера.

Примечание

В шестой версии протокола IP (IPv6, ранее именовавшегося IPng — *IP next generation*) используются 128-разрядные адреса получателей и отправителей (это в 4 раза больше, чем в версии 4). Адрес состоит из шестнадцати октетов и изображается в виде восьми пар октетов, разделенных двоеточиями. Адрес в формате IPv6 может выглядеть так: 3A3F:BC21:F133:56C4:A103:DB11:1000:400F

Если в вашем дистрибутиве xinetd не устанавливается по умолчанию, то рекомендую пойти по пути наименьшего сопротивления и установить **xinetd** из RPM-пакета. Можно также скачать последнюю версию **xinetd** по адресу `http://www.synack.net/xinetd` и установить его из исходных кодов.

Если вы собираете **xinetd** из исходников, вы можете сконфигурировать его (./configure) с одним из следующих флагов:

- **--with-libwrap**: с использованием **tcp wrappers**. С этой опцией **xinetd** будет сперва проверять ваши файлы `/etc/hosts.allow` и `/etc/hosts.deny` и только после этого запустит свой механизм контроля доступа. Библиотека **libwrap** должна быть установлена у вас заранее;
- **--with-loadavg**: с этой опцией **xinetd** остановит сервис, когда нагрузка достигнет определенного уровня;
- **--with-inet6**: включает поддержку IPv6.

Внимание! При включении IPv6 все IPv4-сокеты становятся IPv6-сокетами. Поэтому будьте готовы к тому, что вам понадобится перекомпилировать свое ядро с поддержкой IPv6. Если ваше ядро не поддерживает IPv6, выкачайте последнюю версию ядра по адресу `http://www.kernel.org`. Никаких ограничений на клиентские узлы включение IPv6 не накладывает: запросы IPv4 также будут обрабатываться.

11.3.2. Настройка суперсервера xinetd

Все настройки суперсервера **xinetd** сосредоточены в файле `/etc/xinetd.conf`. В составе дистрибутивов Red Hat и Mandrake уже имеется такой готовый файл, но в нем содержится лишь минимальный набор установок. В него включена строка, которая указывает суперсерверу прочитать все файлы в каталоге `/etc/xinetd.d` и воспринимать их как дополнительные конфигурационные файлы.

Все конфигурационные файлы — `/etc/xinetd.conf` и включаемые в него директивой **includedir** — состоят из записей, имеющих следующий вид:

```
service <имя_службы>
 {
    <атрибут> <оператор_присваивания > <значение> ...
    ...
 }
```

Имя_службы может иметь значение login, shell, telnet, ftp, pop3 и т.п.

Оператор присваивания может быть одним из следующих: «=», «+=», «-=». Большинство атрибутов может работать только с оператором «=». Назначение операторов таково:

- «=»: присвоить атрибуту значение;
- «+=»: добавить атрибуту еще одно значение;
- «-=»: удалить значение атрибута.

Атрибут может иметь несколько значений, указанных через пробел. Большинство атрибутов можно использовать в секции **defaults** конфигурационного файла. Эта секция размещается в начале файла, и все заданные в ней атрибуты применяются ко всем серверам (сервисам), находящимся под контролем ***xinetd***. Для каждого сервера (сервиса) можно индивидуально переопределить установки, заданные по умолчанию: если атрибут определен и в секции **defaults**, и в описании сервера (сервиса), то имеет силу значение, заданное в описании данного сервера (сервиса).

Среди атрибутов **xinetd** можно выделить следующие группы:

- **Запрет вызова сервера (сервиса)**. Запретить вызов какого-либо сервера (сервиса), находящегося под управлением **xinetd**, можно с помощью значения атрибута **disable=yes** (в **inetd** для этого пришлось бы закомментировать все строки описания сервиса). Того же эффекта можно достичь, если в секции **defaults** файла `/etc/xinetd.conf` указать `disables=список_серверов`. Список серверов (сервисов) состоит из их имен, разделенных пробелами.
- **Перенаправление**. С помощью атрибута **redirect** можно обращение к серверу (сервису), для которого указан этот атрибут, перенаправить на другой компьютер. Целевой компьютер указывается либо доменным именем, либо IP-адресом: **redirect=целевой_компьютер**. Этого же эффекта можно достичь и с помощью iptables, но, используя для этой цели **xinetd**, вы можете применять средства управления доступом суперсервера.
- **Протоколирование**. Определить, какая информация должна записываться в журнал в случае успешного запуска сервера (сервиса), а какая в случае неудачи, вы можете с помощью атрибутов **log_on_success** и **log_on_failure**. Подробное описание этих атрибутов приведено в таблице 11.2.
- **Ограничение на установление соединений**. Можно указать максимальное количество запросов от одного источника (сервиса), которое может обработать **xinetd** в единицу времени, а также максимальную загрузку **xinetd**, по достижении которой он будет отвергать все обращения, и приоритет обработки серверов (сервисов). Для этого предназначены атрибуты **per_source**, **instances**, **cps**, **nice** и **max_load** (таблица 11.2).
- **Атрибуты защиты и управления доступом**. В этих атрибутах заключается основное преимущество **xinetd** перед его предшественником **inetd**. С их помощью можно установить:
  - ограничения для отдельных узлов (атрибуты **only_from** и **no-access**);
  - ограничение по времени — временной интервал, в течение которого сервер (сервис) будет доступен для клиентов (атрибут **access_times**);
  - ограничения на использование интерфейсов — можно связать сервер с одним конкретным сетевым интерфейсом (атрибут **interface**).

Атрибуты сервисов под управлением **xinetd** *Таблица 11.2*

Атрибут	Возможные значения
id	Используется, если сервисы используют разные протоколы. Обычно совпадает с именем сервиса
type	Любая комбинация из следующих значений: 1. RPC — если это сервис RPC (Remote Procedure Call). Вызов удаленной процедуры используется в серверной части приложения. Механизм RPC скрывает от программиста детали сетевых протоколов нижележащих уровней. 2. UNLISTED — если сервис не описан в файле /etc/rpc для RPC сервисов или в /etc/services для не-RPC. 3. INTERNAL — если xinetd представляет этот сервис (для echo, time, daytime, chargen и discard). Если RPC-сервисы у вас работают некорректно после установки xinetd, вы можете использовать для них старый inetd — он прекрасно уживается с xinetd. В файле /etc/inetd.conf оставьте только RPC-сервисы, а остальное закомментируйте, а в /etc/xinetd.conf — наоборот
flags	Любая комбинация из следующих значений: 1. NODELAY — только для TCP-сервисов! Будет установлен флаг сокета — TCP_NODELAY. 2. DISABLE — отключить сервис. 3. KEEPALIVE — Только для TCP сервисов! Установка флага сокета SO_KEEPALIVE. 4. REUSE — установить флаг SO_REUSEADDR на сокет сервера. 5. INTERCEPT — перехватывать пакеты или принимать соединения по порядку, проверяя, что они приходят из нужных мест. 6. NORETRY — избегать повторных попыток в случае неудачи. 7. IDONLY — соединение будет приниматься только от идентифицированных пользователей. На удаленной машине должен работать сервер идентификации
disable	yes (отключить сервис) или no (не отключать)
socket_type	*Тип сокета. Может принимать следующие значения:* 1. stream — сокет stream, обычно используется службами, работающими на основе протокола TCP; 2. dgram — сокет dgram, обычно используется службами, работающими на основе протокола UDP; 3. raw — сокет raw для сервисов, требующих прямого доступа к IP; 4. *seqpacket — сокет* **seqpacket** *для сервисов, требующих надежной последовательной пересылки датаграмм*
protocol	tcp, udp и т.п.
wait	yes (ждать завершения сервиса на данном сокете) или no (не ждать). Значение yes обычно устанавливается для сокетов **dgram**, а значение no –для сокетов **stream**
user	Регистрационное имя пользователя, от имени которого будет запущен сервер (по умолчанию — root)
server	Абсолютный путь к запускаемому серверу
server_args	Аргументы, передаваемые серверу
only_from	Список хостов, IP-адресов или сетей, которым будет доступен сервис. Аналог файла hosts.allow
no_access	Список хостов, которым данный сервис не будет доступен. Аналог hosts.deny. Если ни один из атрибутов (**only_from** и **no_access**) не указан, то сервис будет доступен всем
access_times	Интервалы времени в форме ЧЧ:ММ-ЧЧ:ММ, на протяжении которых сервис будет доступен. Например, 9:00-18:00
log_type	Способ ведения сервисом журнала. Значения: 1. SYSLOG syslog_facility [syslog_level] — сообщения будут посылаться демону syslogd; 2. FILE file [soft_limit [hard_limit]] — сообщения будут писаться в указанный файл

Атрибут	Возможные значения
log_on_failure	Определяет, какая информация будет протоколироваться, если сервис по каким-либо причинам не запустился: 1. HOST — записывать адрес удаленного хоста. 2. USERID — если возможно, записывать идентификатор удаленного пользователя (используется протокол идентификации RFC 1413). 3. ATTEMPT — записывать факт неудачной попытки. 4. RECORD — записывать информацию с удаленного узла в случае невозможности запуска сервера
log_on_success	Определяет, какая информация будет протоколироваться в случае удачного запуска сервиса. Можно комбинировать любые из следующих значений: 1. PID — идентификатор запущенного серверного процесса. 2. HOST — адрес удаленного хоста. 3. USERID — если возможно, идентификатор удаленного пользователя (используется протокол идентификации RFC 1413). 4. EXIT — записывать, каким образом был произведен выход. 5. DURATION — продолжительность сессии
rpc_number	Номер сервиса RPC
rpc_version	Версия сервиса RPC
env	Список строк типа "имя=значение". Эти переменные будут добавлены в окружение перед тем, как сервер будет запущен
passenv	Список переменных окружения **xinetd**, которые должны быть переданы серверу
port	Порт сервиса. Если порт указан в файле /etc/services, то значение данного параметра должно совпадать с ним
redirect	Позволяет tcp-сервису делать перенаправление на другой хост. Значение задается в виде host:port
interface	Устанавливает интерфейс, через который будет работать сервис. Значением должен быть IP-адрес
bind	Синоним параметра **interface**
banner	Определяет имя файла, который будет показываться при соединении с сервисом
banner_success	Определяет имя файла, который будет показываться при удачном соединении
banner_fail	Определяет имя файла, который будет показываться при неудачном соединении
cps	Атрибут имеет два аргумента. Первый устанавливает количество соединений в секунду. Если это число будет превышено, сервис будет временно недоступен. Второй — число секунд, после которых сервис снова будет доступен
max_load	Определяет максимальную загрузку. При достижении максимума сервер перестает принимать запросы на соединение. Значение параметра — вещественное число типа float
per_source	Количество запросов от одного источника (сервиса), которое может обработать xinetd в единицу времени. Значением может быть либо число, либо UNLIMITED
instances	Максимальное количество процессов (серверов), которое **xinetd** может одновременно использовать для сервиса (по умолчанию лимита нет). Значением этого атрибута может быть либо число (большее значения атрибута **per_source**), либо UNLIMITED
nice	Устанавливает приоритет (показатель уступчивости) сервиса

Необязательно указывать для каждого сервиса все атрибуты. Можно указать только необходимые:

- socket_type
- user
- server
- wait.

Атрибут **protocol** указывается только для RPC-сервисов, а также для всех сервисов, которые не описаны в `/etc/services`. Атрибут **rpc_version** — только для RPC-сервисов. Атрибут **rpc_number** указывается только для RPC-сервисов, которые не перечислены в файле `/etc/rpc`. Атрибут **port** задается только для не-RPC сервисов, которые не описаны в `/etc/services`.

Следующие атрибуты поддерживают все операторы присваивания:

- only_from
- no_access
- log_on_success
- log_on_failure
- passenv
- env (кроме оператора «-=»).

В секции defaults, в которой описаны атрибуты по умолчанию, обычно указываются следующие атрибуты:

- log_type
- log_on_success
- log_on_failure
- only_from
- no_access
- passenv
- instances
- disabled
- enabled.

11.3.3. Запуск xinetd

Я надеюсь, что с настройкой все более-менее понятно. Если нет, то немного ниже вы найдете пример файла `/etc/xinetd.conf`. Сейчас же займемся запуском только что откомпилированного и настроенного суперсервера. А запускать его можно с ключами, самые нужные из которых приведены в таблице 11.3. За списком остальных обращайтесь к man-странице.

Ключи запуска **xinetd** *Таблица 11.3*

Ключ	Назначение
-f файл	Устанавливает альтернативный файл конфигурации, который должен использоваться вместо стандартного файла `/etc/xinetd.conf`
-pidfile pid_файл	Файл, в который будет записан PID процесса
-stayalive	Даже если ни один сервис не прописан, демон должен выполняться («остаться в живых»)
-d	режим отладки (debug mode)
-limit число	Ограничение на количество одновременно запущенных процессов

Как и любой демон, **xinetd** управляется сигналами:

- SIGHUP заставит **xinetd** перечитать файлы конфигурации и остановить службы, с этого момента отключенные;
- SIGQUIT остановит **xinetd**;
- SIGTERM остановит **xinetd**, остановив предварительно все запущенные им серверы;
- SIGUSR1 выведет дамп состояния в файл `/var/run/xinetd.dump`.

11.3.3.1. Защита xinetd

Суперсервер **xinetd** достаточно защищен, но его конфигурационный файл придется защитить вам самим. Естественно, нужно установить ему права доступа, разрешив только чтение и только суперпользователю:

```
# chmod 400 /etc/xinetd.conf
```

Чтобы никто не смог изменить, удалить, переименовать этот файл, установите ему атрибут «i» (только на файловой системе ext2 или ext3):

```
# chattr +i /etc/xinetd.conf
```

11.3.3.2. Пример файла конфигурации /etc/xinetd

Теперь, как и обещал, привожу пример файла конфигурации. В этом листинге перечислены чаще всего используемые сервисы с оптимальными атрибутами. Конечно же, вам предстоит решить, какие сервисы вы будете использовать, а какие нет. Возможно, вы также измените и их атрибуты — например, время работы того или иного сервиса.

Листинг 11.1. Фрагмент файла конфигурации /etc/xinetd

```
# Параметры по умолчанию для всех возможных сервисов
defaults
{
# Число серверов, которые могут быть активны одновременно
 instances      = 25
# Параметры протоколирования
 log_type     = FILE /var/log/servicelog
 log_on_success    = HOST PID
 log_on_failure    = HOST RECORD
# Параметры доступа
   only_from      = 111.11.111.0 111.111.112.0
 only_from      = localhost 192.168.1.0/32
# Отключенные сервисы
   disables       = tftp
}
```

```
service login
{
 flags        = REUSE
 socket_type    = stream
 protocol       = tcp
 wait      = no
 user      = root
 server       = /usr/etc/in.rlogind
 log_type     = SYSLOG local4 info
}
# Сервис telnet – эмуляция  терминала удаленных систем
# для интерфейса обратной петли
service telnet
{
 flags        = REUSE
 socket_type    = stream
 wait      = no
 user      = root
 server       = /usr/etc/in.telnetd
 bind      = 127.0.0.1
 log_on_failure     += USERID
}

# Сервис telnet – эмуляция терминала удаленных систем
service telnet
{
 flags        = REUSE
 disable        = yes
 socket_type    = stream
 wait      = no
 user      = root
 server       = /usr/etc/in.telnetd
 bind      = 192.231.139.175
 redirect       = 128.138.202.20 23
 log_on_failure     += USERID
}

service ftp
{
 socket_type    = stream
 wait      = no
 user      = root
 server       = /usr/etc/in.ftpd
 server_args    = -l
 instances    = 4
```

```
 log_on_success   += DURATION USERID
 log_on_failure     += USERID
 access_times     = 2:00-8:59 12:00-23:59
 nice      = 10
}

service name
{
 socket_type    = dgram
 wait      = yes
 user      = root
 server       = /usr/etc/in.tnamed
}

# Поддержка протокола TFTP (Trivial FTP). Этот протокол
# используется для обмена информацией между интеллектуальными
# маршрутизаторами и, скорее всего, вы не будете его
# использовать.
service tftp
{
 socket_type    = dgram
 wait      = yes
 user      = root
 server       = /usr/etc/in.tftpd
 server_args    = -s /tftpboot
}

# SMTP-сервис Qmail. Конфигурируется для запуска по
# требованию суперсервера xinetd
service smtp
{
   socket_type = stream
   protocol    = tcp
   wait    = no
   user    = qmaild
   id   = smtp
   server    = /var/qmail/bin/tcp-env
   server_args = /var/qmail/bin/qmail-smtpd
   log_on_success-= DURATION USERID PID HOST EXIT
   log_on_failure-= USERID HOST ATTEMPT RECORD
}

# Сервис finger, позволяющий узнать общедоступную информацию о
# пользователях системы, записанную в /etc/passwd.
service finger
```

```
{
 socket_type    = stream
 disable        = yes
 wait       = no
 user       = nobody
 server       = /usr/etc/in.fingerd
}

service echo
{
 type       = INTERNAL
 id      = echo-stream
 socket_type    = stream
 protocol       = tcp
 user       = root
 wait       = no
}
service echo
{
 type       = INTERNAL
 id      = echo-dgram
 socket_type    = dgram
 protocol       = udp
 user       = root
 wait       = yes
}

service rstatd
{
 type       = RPC
 disabled       = no
 flags       = INTERCEPT
 rpc_version    = 2-4
 socket_type    = dgram

 protocol       = udp
 server       = /usr/etc/rpc.rstatd
 wait       = yes
 user       = root
}
```

Как видно из примера, я сконфигурировал лишь самые необходимые сервисы. Доступ к сервисам в рассматриваемом примере могут получать только клиенты из сетей 111.111.111.0, 111.111.112.0 и 192.168.1.0. Можно также указывать адрес и маску подсети, например, 192.168.1.0/32. Вместо **sendmail** я использовал **qmail**. Можно было бы запускать **qmail** в режиме

standalone, но так как я не очень часто пользуюсь услугами 25-го порта, то мне удобнее запускать сервис **smtp** через **xinetd**. Из соображений безопасности я отключил некоторые сервисы: **finger, telnet**.

11.4. Удаленный доступ: ssh и telnet

11.4.1. Использование telnet

Сервис telnet обеспечивает базовую эмуляцию терминалов удаленных систем, поддерживающих протокол telnet над протоколом TCP/IP. Обеспечивается эмуляция терминалов Digital Equipment Corporation VT 100, Digital Equipment Corporation VT 52, TTY. Проще говоря, сервис telnet позволяет работать на удаленном компьютере так, как будто вы сидите непосредственно за ним. Протокол telnet описан в документе RFC 854, который вы найдете на сайте издательства `www.nit.com.ru`.

Для использования telnet на удаленном компьютере должен быть установлен telnet-сервер. На компьютере пользователя нужно установить программу-клиент. Любые команды, вводимые пользователем, обрабатываются telnet-сервером, а не локальным компьютером. Пользователь на своем компьютере лишь видит результат выполнения этих команд.

Практически в каждой операционной системе существует утилита telnet, которая является клиентом для протокола telnet (рис. 11.1).

```
D:\WINDOWS\System32\telnet.exe
Добро пожаловать в программу клиент Microsoft Telnet

Символ переключения режима: 'CTRL+]'

Microsoft Telnet> h

Команды могут быть сокращены. Поддерживаемыми командами являются:

c    - close                 закрыть текущее подключение
d    - display               отобразить параметры операции
o    - open имя_узла [Порт]  подключиться к сайту (по умолчанию, Порт = 23)
q    - quit                  выйти из telnet
set  - set                   установить параметры ("set ?" для вывода их списка)

sen  - send                  отправить строки на сервер
st   - status                вывести сведения о текущем состоянии
u    - unset                 сбросить параметры ("unset ?" для вывода их списка)

?/h  - help                  вывести справку
Microsoft Telnet> _
```

Рис. 11.1. *Telnet-клиент для Windows*

Сервис telnet был и остается одним из самых популярных способов удаленной регистрации и работы на удаленной машине. Однако основным его недостатком является то, что любая информация, в том числе и пароли, передается в открытом виде без какого-либо кодирования. Поэтому использование его в серьезных системах недопустимо, необходимо пользоваться лишь защищенными средствами (например, SSH).

Сервер telnet может в разных дистрибутивах называться по-разному. В Red Hat- дистрибутивах он находится в пакете telnet-server и в большинстве случаев устанавливается по умолчанию.

11.4.2. Настройка и использование SSH

Что такое SSH

SSH (*Secure Shell* — защищенная оболочка) — это протокол, обеспечивающий защищенную передачу данных. SSH использует криптографию открытого ключа для шифрования соединения между двумя машинами, а также для опознавания (аутентификации) пользователей. Протокол SSH можно использовать для безопасной регистрации на удаленном сервере или копирования данных между двумя машинами. Он позволяет предотвращать атаки способом присоединения посередине (*session hijacking*) и обмана сервера имен (*DNS spoofing*).

Протокол SSH поддерживает следующие алгоритмы шифрования:

- **BlowFish** — 64-разрядная схема шифрования. Этот алгоритм часто используется для высокоскоростного шифрования данных больших объемов.
- **Тройной DES** (*Data Encryption Standard*) — довольно старый стандарт шифрования данных, который в наше время рекомендуется использовать только для несекретных данных.
- **IDEA** (*International Data Encryption Algorithm*) — международный алгоритм шифрования информации. Этот алгоритм работает со 128-разрядным ключом и поэтому он более защищен, чем BlowFish и DES.
- **RSA** (*Rivest-Shamir-Adelman algorithm*) — алгоритм Ривеста-Шамира-Адельмана. Представляет собой схему шифрования с открытым и секретным ключами.

Поддержка нескольких алгоритмов шифрования позволяет обеспечивать наибольшую безопасность, так как в случае обнаружения уязвимости в одном алгоритме SSH переориентируется на использование других алгоритмов.

На данный момент существует две версии протокола SSH:

- **Протокол SSH версия 1**. У каждого узла есть свой RSA-ключ (обычно 1024 бит), который используется для идентификации узла. Этот ключ еще называется открытым. Дополнительно, при запуске демона, генерируется еще один RSA-ключ — ключ сервера (обычно 768 бит). Этот ключ создается заново каждый час и никогда не сохраняется на диске. Каждый раз при установке соединения с клиентом демон отправляет ему в ответ свой открытый ключ и ключ сервера. Клиент сравнивает полученный открытый ключ со своей базой данных, чтобы проверить, не изменился ли он. Затем клиент случайным образом генерирует 256-разрядное число и кодирует его, используя одновременно два ключа — открытый ключ и ключ сервера. Обе стороны используют этот случайный номер как ключ сессии, который используется для кодирования всех передаваемых во время сессии данных. Затем клиент пытается аутентифицировать себя, используя .rhosts-аутентификацию, аутентификацию RSA или же аутентификацию с использованием пароля. Обычно .rhosts-аутентификация небезопасна, поэтому она отключена.
- **Протокол SSH версия 2**. Версия 2 работает аналогично первой: каждый узел имеет определенный RSA-ключ, который используется для идентификации узла. Однако при запуске демона ключ сервера не генерируется. Безопасность соединения обеспечивается благодаря соглашению Диффи-Хелмана (*Diffie-Hellman key agreement*). Кроме того, в SSH2 были исправлены недостатки SSH1. Сессия может кодироваться следующими методами: 128-разрядный AES, Blowfish, 3DES, CAST128, Arcfour, 192-разрядный AES или 256-разрядный AES.

Программные пакеты, использующие эти протоколы, так и называются: ssh1 и ssh2. Сервером SSH служит демон **sshd**, который запускается на UNIX-машине, а клиентом — программа **ssh**, которая распространяется как для Linux, так и для Windows. Клиент ***ssh*** служит для обеспечения защищенной регистрации на удаленном компьютере. В пакет ssh входит еще и третья программа — **scp**, служащая для безопасного копирования файлов с локального компьютера на удаленный. Однако основным назначением SSH является все-таки авторизация пользователя при регистрации его на удаленном компьютере.

Оба программных продукта (SSH1 и SSH2) являются коммерческими и стоят денег. Хотя в какой-то момент разработчики одумались и сделали бесплатной SSH2 для Linux и *BSD, было уже поздно. Открытым обществом разработчиков на основе обоих протоколов SSH, с добавлением дополнительных возможностей и исправлением некоторых ошибок, был разработан ее бесплатный вариант OpenSSH.

Первая версия OpenSSH вышла еще в декабре 2001 года. В дистрибутив Fedora Core 3 включена третья версия этого продукта. Компьютеры, на которых установлена OpenSSH, прекрасно взаимодействуют с компью-

терами, на которых установлены коммерческие SSH1 или SSH2, то есть продукты полностью совместимы.

В дальнейшем, когда я буду говорить о SSH, я буду иметь в виду именно OpenSSH, которая поставляется со всеми современными дистрибутивами Linux. В целях безопасности рекомендуется отслеживать обновления и скачивать последнюю версию (в мае 2005 г. вышла четвертая) с сайта `www.openssh.org`.

Свободно распространяемая версия SSH состоит из следующих пакетов:

- openssh — основные файлы;
- openssh-clients — программа-клиент;
- openssh-server — ssh-сервер.

Чтобы служба SSH начала работать, необходимо запустить демон **sshd** на той машине, к которой предполагается подключение. Желательно добавить команду запуска в сценарий загрузки системы. Демон **sshd** работает по 22 порту (см. листинг 11.2). Можно запускать его из-под супердемона **xinetd/inetd**, но обычно **sshd** запускается самостоятельно — в режиме **standalone**.

Настройка SSH на сервере

Конфигурационный файл сервера **sshd** называется `/etc/ssh/sshd_config`. Справку по его синтаксису вы можете получить по команде `man 5 sshd_config`. В пакете `openssh-server` находится конфигурационный файл с типовыми настройками.

Чтобы оградить ваш компьютер от нежелательных вторжений извне, рекомендую вписать в этот файл директиву **allowedadress**, перечислив через пробел IP-адреса тех машин, с которых разрешен вход клиентов:

```
allowedadress 10.1.1.1 10.1.2.1 10.1.3.1
```

Листинг 11.2. Приметный файл конфигурации /etc/ssh/sshd_config

```
Port 22
# Сначала пытаемся работать по протоколу SSH 2, а потом,
# если та сторона не поддерживает вторую версию, — по SSH 1
Protocol 2,1
# Ключ для протокола SSH версии 1
HostKey /etc/openssh/ssh_host_key
# Ключи для протокола SSH2 — RSA и DSA
HostKey /etc/openssh/ssh_host_rsa_key
HostKey /etc/openssh/ssh_host_dsa_key
```

```
# Время жизни и размер ключа ssh версии 1
KeyRegenerationInterval 3600
# По умолчанию используется размер 768 бит,
# лучше установить 1024
ServerKeyBits 1024

# время, через которое ключи сервера будут созданы заново.
# Периодическая смена ключей повышает безопасность
системы.
KeyRegenerationInterval 1h

# Запрещаем регистрацию пользователя root по ssh.
# Это не исключает возможности удаленного
# администрирования: просто руту придется зайти под
# обычным пользователем, а затем  выполнить команду su.
# Зато злоумышленнику понадобится украсть
# не один, а два пароля: и root, и обычного пользователя.
PermitRootLogin no

# Протоколирование (раскомментируйте, если нужно
# вести журнал с помощью системы syslog)
#SyslogFacility AUTH
#LogLevel INFO

# Аутентификация
# Включает парольную аутентификацию
# и запрещает пустые пароли
PasswordAuthentication yes
PermitEmptyPasswords no

#StrictModes yes
# используем RSA-аутентификацию
RSAAuthentication yes
PubkeyAuthentication yes
# Аутентификация rhosts - обычно не используется,
# поэтому запрещаем ее:
# пользовательские файлы ~/.rhosts и ~/.shosts не
# будут использоваться
RhostsAuthentication no
IgnoreRhosts yes
# НЕ использовать PAM аутентификацию
PAMAuthenticationViaKbdInt no

# дополнительное время клиенту на то, чтобы
# аутентифицировать себя.
```

```
# Если за это время клиент не смог ввести пароль,
# соединение будет прекращено
LoginGraceTime 2m

# Следующие параметры нужны для того, чтобы заставить
# систему X Window работать по ssh. Подробнее вы
# сможете прочитать в документации по ssh
#X11Forwarding yes
#X11DisplayOffset 10
#X11UseLocalhost yes
#PrintMotd yes
#PrintLastLog yes
#KeepAlive yes
#UseLogin no
#UsePrivilegeSeparation yes
#Compression yes

# Путь к баннеру
Banner /some/path
# подсистема sftp-сервера
Subsystem sftp /usr/libexec/openssh/sftp-server
```

Запуск демона sshd

Перед первым запуском **sshd** необходимо сгенерировать файлы, содержащие ключи кодирования. В сценариях, осуществляющих запуск сервера **sshd,** обычно предусмотрена проверка наличия этих файлов. В случае их отсутствия они генерируются автоматически.

Ключи, с которыми можно запускать **sshd**, перечислены в таблице 11.4.

Ключи сервера sshd *Таблица 11.4*

Ключ	Назначение
-b биты	Определяет число битов для ключа сервера (по умолчанию 768). Эту опцию можно использовать, только если вы используете протокол SSH версии 1
-d	Режим отладки (DEBUG). В этом режиме сервер не переходит в фоновый режим, обрабатывает только одно соединение и подробно протоколирует свои действия в системном журнале. Ключ отладки особенно полезен для изучения работы сервера
-D	Так же, как и при использовании предыдущего ключа, сервер ***sshd*** не будет переходить в фоновый режим. Однако в отличие от -d ключ -D не переводит сервер в режим отладки
-e	Отправлять отладочные сообщения не в системный журнал, а на стандартный поток ошибок
-f файл	Задает альтернативный файл конфигурации вместо `/etc/ssh/sshd_config`
-g время	Предоставляет клиенту, не прошедшему аутентификацию, дополнительное время на ввод пароля. Значение 0 интерпретируется как бесконечное ожидание

Продолжение табл. 11.4

Ключ	Назначение
-h файл_ключа	Задает альтернативный файл открытого ключа (ключ узла). По умолчанию используется файл `/etc/ssh/ssh_host_key`. Этот ключ может понадобиться, чтобы запускать **sshd** от имени непривилегированного пользователя. Также ключ **-h** часто применяется при запуске **sshd** из сценариев, задающих различные настройки в зависимости от времени суток (в рабочее и нерабочее время)
-i	Используется, если нужно запускать **sshd** через суперсервер **xinetd**. Обычно демон **sshd** запускается отдельно при загрузке системы. Связано это с тем, что демону **sshd** требуется некоторое время для генерирования ключа сервера, прежде чем он сможет ответить на запросы клиентов. При запуске через суперсервер при каждом соединении суперсервер будет заново вызывать **sshd**, а тот — заново генерировать ключ. Однако на современных компьютерах задержка практически не заметна. Поэтому вполне можно запускать **sshd** и через суперсервер
-k время	Задает время, спустя которое ключ сервера будет создан заново. По умолчанию время составляет 1 час. Эту опцию можно использовать только с протоколом SSH версии 1
-p порт	Указывает альтернативный порт, который демон **sshd** будет прослушивать вместо порта 22
-q	«Тихий режим»: не протоколировать сессию. Обычно протоколируется начало аутентификации, результат аутентификации и время окончания сессии
-t	Тестовый режим. Применяется для проверки корректности файла конфигурации
-4	Разрешается использовать IP-адреса только в формате IPv4
-6	Разрешается использовать IP-адреса только в формате IPv6

Использование SSH-клиента

Клиентская программа **ssh** находится в пакете `openssh-clients` вместе с типовым конфигурационным файлом `/etc/ssh/ssh_config`. Настройки можно задавать и из командной строки, запуская **ssh** с соответствующими ключами. Основные ключи и аргументы перечислены в таблице 11.5.

Формат команды:

```
ssh [ключи] [ключи_с_аргументами] [логин_имя@]хост.домен
[команда]
```

Если последним аргументом указана команда, то после успешного входа пользователя она выполняется на удаленной машине вместо командной оболочки по умолчанию. Таким образом можно работать не в командной строке, а запустить на удаленной машине графический сеанс.

Ключи программы **ssh** *Таблица 11.5*

Ключ	Назначение
-a	Отключает перенаправление аутентификации агента соединения
-A	Включает перенаправление аутентификации агента соединения
-c blowfish|3des|des	Позволяет выбрать алгоритм шифрования при использовании первой версии протокола SSH. Можно указать **blowfish**, **3des** или **des**
-C	Задает использование сжатия всех данных во всех выходных потоках с использованием gzip.

Продолжение табл. 11.5

Ключ	Назначение
-f	Данная опция переводит ssh в фоновый режим после аутентификации пользователя. Рекомендуется использовать для запуска программы X11. Например: **ssh –f host xterm**
-**i** идент_файл	Задает нестандартный идентификационный файл (для нестандартной RSA/DSA-аутентификации)
-**l** логин_имя	Указывает, от имени какого пользователя будет осуществляться регистрации на удаленной машине
-**p** *порт*	Определяет порт, к которому подключится программа **ssh** (по умолчанию используется порт 22)
-q	Переводит программу ssh в «тихий режим». При этом будут отображаться только сообщения о фатальных ошибках. Все прочие предупреждающие сообщения в стандартный выходной поток выводиться не будут
-v	Включает отображение всей отладочной информации
-x	Отключить перенаправление X11
-X	Включить перенаправление X11
-1	Использовать только первую версию протокола SSH (принудительно)
-2	Использовать только вторую версию протокола SSH (принудительно)
-4	Разрешается использовать IP-адреса только в формате IPv4
-6	Разрешается использовать IP-адреса только в формате IPv6

Аутентификация средствами SSH

Аутентификация в SSH может производиться одним из следующих четырех способов:

- **По принципу доверия**. При этом способе проверяется, внесено ли имя компьютера, с которого производится доступ, в файл `~/.rhosts` (или `~/.shosts`). Если его имя (IP-адрес) внесено, то пользователю разрешается доступ без проверки пароля. Этот способ очень уязвим для разнообразных атак (подмены IP-адреса и т.п.), так что использовать его категорически не рекомендуется. Для того, чтобы разрешить этот способ, нужно установить значение **no** для директивы **IgnoreRHosts** и **yes** для **RhostsAuthentication** в файле `/etc/openssh/sshd_conf`, а чтобы запретить — значения для этих директив поменять на противоположные.
- **Усиленная аутентификация по принципу доверия**. Этот способ в принципе повторяет предыдущий, за тем лишь исключением, что проверка имени компьютера (IP-адреса) производится в защищенном режиме. При этом используется шифрование открытым ключом. За включение и отключение данного механизма аутентификации отвечают директивы **RhostsRSAAuthentication** и **IgnoreRHosts**. Несмотря на некоторые усовершенствования, этот способ по-прежнему является небезопасным.

- **Аутентификация самого пользователя** с использованием шифрования с открытым ключом. На момент регистрации у пользователя должен быть доступ к файлу своего секретного ключа, и он должен предоставить пароль для его дешифровки. Этот способ аутентификации является самым надежным, но и самым неудобным. Кроме того, он вынуждает пользователей постоянно иметь под рукой файл с ключом. За включение и отключение этого способа отвечает директива **PubkeyAuthentication** (или **RSAAuthentication** в зависимости от версии).
- **Аутентификация с помощью пароля**. Это оптимальный способ: он удобен в использовании и в то же время достаточно безопасен. Именно он и используется в большинстве случаев. В отличие от telnet, пароль здесь передается в зашифрованном виде. Основной недостаток данного метода заключается в относительной слабости паролей (их длина зачастую составляет менее 8 символов). Это позволяет подбирать их с помощью специальных программ. За включение и отключение этого способа отвечает директива **PasswordAuthentication**.

Какой способ использовать — решать вам. Однако настоятельно не рекомендуется применение первых двух способов. Их использование допустимо лишь в исключительных случаях и в особых условиях.

Бесплатный SSH-клиент для Windows можно скачать у автора, Роберта Каллагана, сайт `http://www.zip.com.au/~roca/ttssh.html`.

Глава 12

РАЗДЕЛЕНИЕ РЕСУРСОВ: NFS И SAMBA

- NFS — СЕТЕВАЯ ФАЙЛОВАЯ СИСТЕМА
- SAMBA:LINUX-СЕРВЕР ДЛЯ WINDOWS-КЛИЕНТОВ
- СОВМЕСТНОЕ ИСПОЛЬЗОВАНИЕ КАТАЛОГОВ В LINUX MANDRAKE
- ПРОГРАММА LINNEIGHBORHOOD — ПРАВИЛЬНЫЙ ВЫБОР

12.1. NFS — сетевая файловая система

Сетевая файловая система (*Network File Sharing*) — это протокол, позволяющий монтировать файловые системы на удаленных компьютерах. При этом создается ощущение, что эти файловые системы располагаются локально, если не считать, конечно, скорости соединения. Сетевая файловая система чем-то напоминает службу «Доступ к файлам и принтерам» сети Microsoft.

Служба NFS основана на RPC (*Remote Procedure Call*): клиент вызывает функции в программе сервера. Клиентскому компьютеру не требуется дополнительного программного обеспечения, чтобы воспользоваться NFS: ядро Linux само определяет, что запрошенный файл находится на NFS-сервере, и автоматически генерирует RPC-вызов, чтобы получить доступ к файлу.

Версия 2 протокола NFS появилась в ядре 1.2. Начиная с версии ядра 2.2.18, поддерживается версия 3, в которой улучшена производительность благодаря безопасной асинхронной записи файлов, добавлены поддержка файлов размером больше 2 Гб и средства блокировки файлов. В ядра 2.4 и выше встроена полнофункциональная NFSv3, работающая поверх протокола TCP (в дополнение к обычному UDP), и большинство популярных дистрибутивов ее поддерживают. Четвертая версия NFS с повышенной безопасностью появилась в Red Hat Enterprise Linux 4.

FAQ и HOWTO-документы можно найти на домашней странице проекта `nfs.sourceforge.net`.

12.1.1. Клиент NFS: монтирование сетевого каталога

Монтирование осуществляется с помощью команды ***mount***:

```
# mount -t nfs -o timeo=30 nfsserver.domain.com:
/home/den /home/den/remote/
```

Прежде всего, нужно указать тип файловой системы: `-t nfs`. Параметр **timeo** задает время ожидания, равное 3 секундам. Интересующая нас

файловая система находится на компьютере `nfsserver.domain.com` и смонтирована там как `/home/den`. Мы же подмонтируем ее в подкаталог домашнего каталога на локальном компьютере `/home/den/remote/`.

Для монтирования сетевых файловых систем предназначены ключи команды mount, перечисленные в таблице 12.1.

Ключи команды mount для сетевых файловых систем *Таблица 12.1*

Ключ	Назначение
bg	Если первая попытка монтирования файловой системы NFS окажется неудачной, то попытки будут автоматически повторяться в фоновом режиме
fg	Если первая попытка монтирования файловой системы NFS окажется неудачной, то попытки будут автоматически повторяться в приоритетном режиме. Это значение установлено по умолчанию
soft	Задержка при выполнении операции, связанной с файлом, расположенным в сетевой файловой системе NFS (возникающая при сбое сервера или отключении сети), будет приводить к отправке приложению сообщения об ошибке ввода/вывода. И хотя некоторые приложения могут корректно обрабатывать такую ошибку, но большинство из них все-таки такой возможностью не обладают. Не рекомендуется устанавливать это значение, так как оно может привести к появлению испорченных файлов и потере данных
hard	Задержка при выполнении операции, связанной с файлом, расположенным в сетевой файловой системе NFS, будет приводить к приостановке, а затем возобновлению процесса с прерванного места. Таким образом будут предприниматься повторные попытки выполнения операции. Значение hard установлено по умолчанию
tcp	Монтирует сетевую файловую систему с помощью протокола TCP, а не UDP
rsize=1024	Задает размер информации, пересылаемый при чтении файлов за один раз. По умолчанию этот размер равен 1024 байта. Однако зачастую имеет смысл увеличить это значение (например, до 4096)
wsize=1024	Аналогично rsize, но для операции записи
noexec	Запрещает выполнение программ или сценариев в монтируемой файловой системе. При этом клиентом будет игнорироваться атрибут «исполнение»

Чтобы сетевая файловая система монтировалась автоматически при загрузке системы, нужно внести определенные записи в файл `/etc/fstab`. Например, такая запись для рассмотренного выше примера может иметь примерно следующий вид:

```
nfsserver.domain.com:/home/den  /home/den/remote/  nfs  bg,hard,rw 1  0
```

12.1.2. Настройка сервера NFS

Сервер NFS экспортирует файловые системы, то есть делает их доступными для использования с другого компьютера. Обслуживанием экспорта занимаются следующие демоны:

- **rpc.mountd**, обрабатывающий запрос на монтирование;
- **rpc.nfsd**, обеспечивающий доступ к файлам смонтированной файловой системы;

- **rpc.lockd** и **rpc.statd**, управляющие блокировкой файлов;
- **rpc.rquotad**, контролирующий квоты на экспортируемых файловых системах.

Эти демоны устанавливаются из пакета `nfs-utils` в каталог `/sbin` (реже `/usr/sbin`). Запускаются они в ходе начальной загрузки, и сценарии запуска находятся в том же пакете.

Как и другие RPC-службы, NFS зависит от демона **portmap** (или **rpc.portmap**), который должен запускаться первым. Загрузочные сценарии большинства современных дистрибутивов включают запуск этого демона. Если в вашем дистрибутиве демоны NFS по умолчанию не запускаются, отредактируйте загрузочные сценарии, организовав запуск в следующем порядке:

1. rpc.portmap;
2. rpc.mountd, rpc.nfsd;
3. rpc.statd, rpc.lockd (с ядром выше 2.2.18 этот демон вызывается из nfsd, и отдельно запускать его не нужно), rpc.rquotad.

Если на вашем компьютере запущен также сервер DNS, проследите за тем, чтобы сервер DNS запускался после демонов NFS. При соблюдении данного условия гарантируется корректная работа NFS.

Конфигурационным файлом сервера NFS служит `/etc/exports`, содержащий список экспортируемых файловых систем. В каждой строке записан один экспортируемый каталог, узел или домен, которому разрешено монтировать этот каталог, и — без пробела, в скобках, через запятую, — спецификации экспорта. Длинные строки можно разбивать на несколько строк при помощи символа обратного слэша.

Разрешается экспортировать любые каталоги, а не только корневые каталоги физических файловых систем. Единственное ограничение — файловые системы, физически расположенные на разных устройствах (разделах), должны экспортироваться по отдельности.

Листинг 12.1. Примерный файл /etc/exports

```
/pub        (ro,insecure,all_squash)
/home/den  denhome.domain.com(rw)
/mnt/cdrom    (ro)
/mnt/cdrom    comp11.domain.com(noaccess)
```

Давайте разберемся в данной конфигурации. К каталогу `/pub` имеют доступ в режиме «только чтение» все компьютеры сети. К каталогу `/home/den` доступ в режиме чтения и записи имеется только с компьютера denhome.domain.com. К каталогу `/mnt/cdrom` доступ в режиме «только чтение» имеют все компьютеры, кроме comp11.domain.com.

Другие спецификации экспорта перечислены в таблице 12.2.

Спецификации в файле /etc/exports *Таблица 12.2*

Спецификация	Назначение
secure	Требует, чтобы запросы исходили только из портов безопасного диапазона — с номерами, меньшими 1024. В Linux доступ к таким портам имеет только пользователь root. Значение secure по умолчанию включено
insecure	Отключает значение secure
ro	Устанавливает для экспортируемого каталога доступ в режиме «только чтение»
rw	Устанавливает для экспортируемого каталога доступ в режиме чтения и записи
noaccess	Запрещает доступ к конкретной ветви (подкаталогу) экспортируемого дерева каталогов
link_absolute	Оставляет все символические ссылки без изменений. Значение включено по умолчанию
link_relative	Конвертирует абсолютные ссылки в относительные
squash_uidssquash_gids	Указанные идентификаторы групп и пользователей будут конвертированы в анонимные
all_squash	Все идентификаторы групп и пользователей будут конвертированы в анонимные. При этом всем им будет запрещен доступ к экспортируемым каталогам. По умолчанию так не делается
no_all_squash	Значение, обратное all_squash. Активизировано по умолчанию
root_squash	Преобразует запросы от пользователя root (uid=0) в запросы от анонимного пользователя с целью запретить суперпользователю его привилегии при доступе к файловой системе. Это значение установлено по умолчанию
no_root_squash	Отменяет значение root_squash
anonuid=UID anonguid=GID	Задают идентификаторы анонимных пользователей. По умолчанию анонимным пользователем является пользователь nobody. Однако с помощью этих спецификаций вы можете сами указать, кого следует воспринимать в качестве анонимных пользователей

Каталоги, перечисленные в /etc/exports, экспортируются на этапе начальной загрузки. Добавить или удалить каталог в течение сеанса можно при помощи утилиты ***exportfs*** из пакета nfs-utils.

Для повышения безопасности доступа к разделяемым ресурсам служат конфигурационные файлы /etc/hosts.allow и /etc/hosts.deny (п.11.2.2).

12.2. Samba: Linux-сервер для Windows-клиентов

12.2.1. Samba на сервере

Из п.6.3 вы узнали, как использовать пакет Samba (www.samba.org) для просмотра общих ресурсов сети Windows. В этом параграфе я объясню, как настроить сервер Samba, чтобы открыть общий доступ к ресурсам компьютера под управлением Linux.

С помощью Samba вы сможете:

- предоставлять доступ к разделам Linux для рабочих станций Windows;
- получать доступ к ресурсам сети Microsoft;
- распечатывать документы на сетевых принтерах сети Microsoft, а также предоставить локальный принтер для использования в качестве сетевого.

Вам понадобятся основной пакет samba-common и серверный samba. Из серверного пакета samba устанавливаются демоны **smbd** и **nmbd**. Первый из них является носителем протокола SMB, а второй обеспечивает поддержку имен NetBIOS. Сразу же после их настройки ваш компьютер будет отображаться в сети.

После установки сервисы **smbd** и **nmbd** конфигурируются как запускаемые в ходе начальной загрузки. Возможно, вас не устраивает такой вариант (например, в тех случаях, когда обращение к ним планируется редко, и вы хотите освободить память). В этой ситуации никто не мешает добавить их в файл конфигурации суперсервера `/etc/xinetd.conf` и запускать «по требованию». При этом не забудьте только отключить их автозапуск с помощью конфигуратора системы.

12.2.2. Настройка Samba

В этом параграфе будет рассмотрена настройка пакета Samba «вручную», то есть без помощи конфигуратора. Вы же можете использовать конфигуратор (рис. 12.1), однако имейте в виду одно «но»: если вы будете настраивать Samba или любую другую службу сервера в другом дистрибутиве, знакомого вам конфигуратора в нем может не оказаться. Поэтому вы должны знать хотя бы назначение и расположение системных файлов той или иной службы сервера.

Основным конфигурационным файлом сервера Samba является файл `/etc/smb.conf`. Именно в нем задаются все используемые и предоставляемые ресурсы. Формат данного файла напоминает формат INI-файла Windows.

Файл `/etc/smb.conf` состоит из нескольких секций, в начале каждой из которых в квадратных скобках указывается ее имя. Параметры в каждой секции указываются в виде записей `Имя=Значение`.

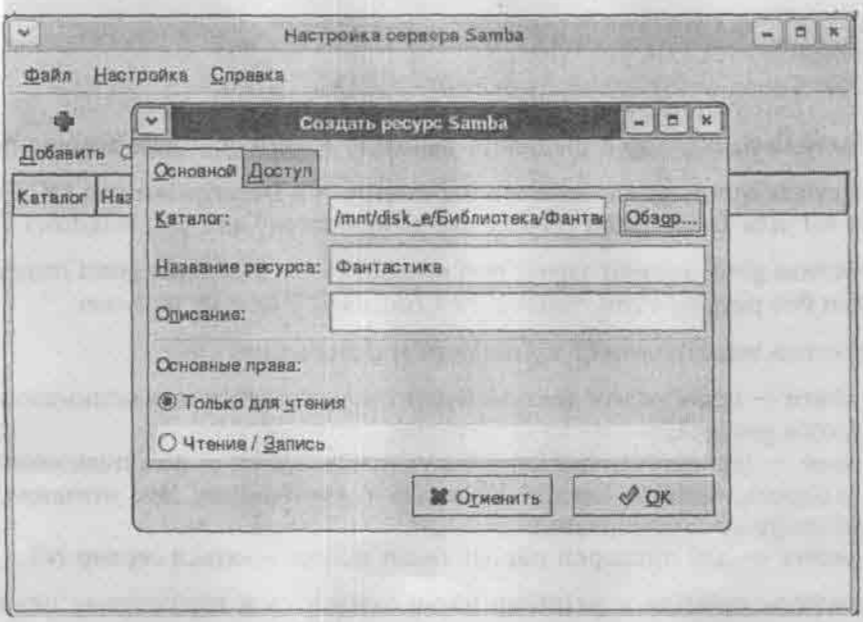


Рис. 12.1. *Конфигуратор system-config-samba в дистрибутиве Fedora Core 3*

Секция [global]

Основной является секция `[global]`, ее пример приведен в листинге 12.2.

Листинг 12.2. Пример секции global

```
[global]
workgroup = WORK
comment = Linux Server
guest account = guest
security = share
printing = bsd
printcap name = /etc/printcap
load printers = yes
client code page = 866
character set = koi8-r
encrypt passwords = Yes

log file = /var/log/samba/log.%m
max log size = 50
socket options = TCP_NODELAY SO_RCVBUF=8192 SO_SNDBUF=8192
dns proxy = no
```

```
wins support = yes
domain master = yes
interfaces = 192.168.1.1/24 192.168.2.1/24
```

Директива **workgroup** определяет рабочую группу или имя домена NT.

Директива **comment** аналогична параметру **NT Description** для ОС Windows NT или **Description** (Описание компьютера) для ОС Windows 9x.

Директива **guest account** задает имя пользователя. Значение **guest** означает доступ без регистрации, точнее, под гостевой учетной записью.

Директива **security** может принимать три значения:

- **share** — при каждом доступе будет запрашиваться имя пользовательского ресурса;
- **user** — для аутентификации будут использоваться имя пользователя и пароль, которые служат для входа в сетьWindows. Это значение используется по умолчанию;
- **server** — для проверки пароля будет использоваться сервер NT.

Директивы **printing** и **printcap name** относятся к подсистеме печати. Первая из них задает систему печати типа BSD, а вторая указывает, где расположен файл, содержащий информацию о принтерах. О настройке принтеров мы поговорим немного позже.

Директивы **client code page** и **character set** необходимы для корректного отображения русскоязычных имен файлов файловой системы Windows.

В ОС Windows NT, начиная с Service Pack 3, передача паролей по сети происходит в зашифрованном виде. Последние версии Samba поддерживают эту возможность. Чтобы воспользоваться ею, нужно установить значение директивы **encrypt password** равным **yes**. Если ваша версия Samba не поддерживает данную возможность, то вы можете отключить использование закодированных паролей в Windows. Правда, вам придется вручную править реестр на всех рабочих станциях Windows, так что проще, по-моему, обновить Samba. Но если вас все же интересует, какой именно параметр реестра Windows нужно изменить, я укажу его:

- В разделе реестра ОС Windows NT: [NT HKEY_LOCAL_MACHINE\System\Current ControlSet\Services\Rdr\Parameters] нужно создать ключ **EnablePlainTextPassword** типа DWORD и установить его значение, равное 1.
- В ОС Windows 9x вам нужно создать тот же ключ, но в разделе [HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\VxD\Parameters].
- В ОС Windows 2000 нужно внести изменения в раздел реестра [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\LanmanWorkStation\Parameters].

Директивы **log file** и **max log size** определяют имя файла протокола (журнала) и его максимальный размер.

Параметры сокетов задаются с помощью директивы **socket option**. Прежде чем устанавливать эти параметры, рекомендую прочитать `man smb.conf`.

Если в вашем компьютере установлено несколько сетевых интерфейсов, то вы можете сконфигурировать пакет Samba так, чтобы он использовал все интерфейсы. Как это сделал я, показано в листинге 12.2.

Секция [homes]

Следующая секция называется `[homes]` и определяет параметры совместно используемых ресурсов. Пример содержимого этой секции приведен в листинге 12.3.

Листинг 12.3. Секция [homes]

```
[homes]
comment = Home
browseable = yes
writable = yes
```

Директива **browseable=yes** разрешает отображение совместно используемых ресурсов в сети Microsoft. Директива **writable=yes** разрешает запись в каталоги (тот же эффект будет иметь директива **read only=no**).

Секция [public]

В секции `[public]` перечисляются расшариваемые каталоги (листинг 12.4).

Листинг 12.4. Секция [public]

```
[public]
comment = VFAT Partition
path = /mnt/disk_e
read only = no
```

12.2.3. Практические примеры настройки

Ваших знаний уже достаточно, чтобы самим произвести дальнейшую настройку. В качестве завершения этого параграфа я приведу пару практических примеров (листинг 12.5). Обратите внимание, что в файле `smb.conf` комментарии могут обозначаться либо решеткой (#), либо точкой с запятой (;).

Листинг 12.5.

```
; Каталог NETLOGON для входа в домен
[netlogon]
comment = Samba Netlogon Service
path = /var/netlogon
; Не устанавливайте значение yes
case sensitive = no
guest ok = yes
locking = no
writable = yes
browseable = yes

; Профиль для совместно используемых ресурсов
[Profiles]
path = /usr/local/samba/profiles
browseable = no
printable = no
guest ok = yes

; Каталог, используемый пользователем admin
; Пользователь admin должен существовать на сервере Samba
[admin]
comment = admin's directory
path = /home/admin
valid users = admin root
read only = no
```

12.2.4. Доступ к принтеру Linux для Windows-машин

Для обеспечения поддержки печати мы уже сделали почти все возможное. Директива секции [global] **load printers** загружает принтеры из файла `/etc/printcap` (листинг 12.2). Используется система печати BSD. Осталось только определить секцию [printers] файла `smb.conf`. В этой секции задаются глобальные параметры для всех принтеров, поэтому нет необходимости указывать их отдельно для каждого принтера.

Листинг 12.6. Секция [printers]

```
[printers]
comment = All Printers
security=server
path = /var/spool/lpd/lp
browseable = no
printable = yes
```

```
public = yes
writable = no
create mode = 0700
```

Некоторые директивы, используемые в этой секции (**browseable, writable, comment**), имеют те же значения, что и в секции **[homes]**. Директива **path** задает буферный каталог, в который файлы будут копироваться перед печатью (так называемый *спул* принтера). Директива **public** со значением **yes** разрешает печать из-под гостевой учетной записи, то есть всем желающим. Чтобы запретить печать из-под гостевой учетной записи, укажите **public=no**: в этом случае доступ к принтеру будут иметь только зарегистрированные на сервере пользователи. Вместо директивы **public** иногда используется ее синоним — директива **guest ok**.

Директива **writable** установлена в значении **no** для того, чтобы в буферный каталог принтера (спул) могли записываться только печатаемые файлы.

Возможно, вам потребуется разрешить печать только одному или нескольким определенным пользователям на каком-то определенном принтере. Сделать это можно так, как это показано в листинге 12.7.

Листинг 12.7. Разрешение печати определенному пользователю

```
[admprn]
valid user = root admin administator
path = /home/admin
printer = canon
public = no
writeable = no
printable = yes
```

Подключение к Windows-компьютеру сетевого принтера, расшаренного сервером или рабочей станцией под управлением Linux, осуществляется точно так же, как подключение обыкновенного сетевого принтера. В операционной системе Windows 98 для этого проделайте следующее:

1. Выполните команду меню **Пуск-> Настройка-> Принтеры**.
2. Активизируйте мастера **Установки принтера**.
3. Выберите тип принтера: **сетевой**.
4. Укажите путь к принтеру или нажмите на кнопку **Обзор** для автоматического выбора ресурса.
5. Далее установка сетевого принтера аналогична установке локального.

12.2.5. Доступ к Windows-принтеру с компьютера, работающего под Linux

Настройки сетевого принтера находятся в файле /etc/printcap. Примерное содержание этого файла с комментариями приведено в листинге 12.8.

Листинг 12.8. Файл /etc/printcap

```
# //nt_ws1/hp5m via smbprint
#
lp:\
# описание принтера
   :cm=HP 5MP Postscript hp5m on nt_ws1:\

# имя устройства, открываемого для вывода
   :lp=/dev/lp0:\

# каталог спула принтера (на локальной машине)
   :sd=/var/spool/lpd/lp:\

# файл учета использования принтера
   :af=/var/spool/lpd/lp/acct:\

# максимальный размер файла. Значение 0 означает
отсутствие ограничений
 :mx#0:\

# имя входного фильтра
   :if=/usr/bin/smbprint:
```

В пакет Samba входит сценарий **smbprint**. С помощью этого сценария можно распечатывать документы на сетевом принтере, используя сервисы SMB. Возможно, в состав вашего пакета он не входит, поэтому я воссоздал его в листинге 12.9. Этот листинг частично позаимствован из руководства по пакету Samba.

Листинг 12.9. Сценарий smbprint

```
#!/bin/sh -x
# (c) Andrew Tridgell
# Этот скрипт является фильтром для системы печати,
# использующей
# файл /etc/printcap
# Он использует программу smbclient для печати файла на
```

```
# сетевом принтере,
# который подключен к рабочей станции Windows.
#
#
# Эта запись создает unix-принтер, названный "smb",
# который будет печатать с помощью этого сценария. Вам
# необходимо создать каталог спула /usr/spool/smb с
# соответствующими правами и владельцем

# smb:lp=/dev/null:sd=/usr/spool/smb:sh:
# if=/usr/local/samba/smbprint
# Далее сценарий был изменен Майклом Гамильтоном
# так что сервер, сервис и пароль могут быть считаны из
# файла
# /usr/var/spool/lpd/PRINTNAME/.config
#
# Для того, чтобы это работало, запись в /etc/printcap
# должна включать файл учета использования (af=...):
#
#cdcolour:\
#   :cm=CD IBM Colorjet on 6th:\
#   :sd=/var/spool/lpd/cdcolour:\
#   :af=/var/spool/lpd/cdcolour/acct:\
#   :if=/usr/local/etc/smbprint:\
#   :mx=0:\
#   :lp=/dev/null:
#
# Файл /usr/var/spool/lpd/PRINTNAME/.config должен
# содержать
#   server=PC_SERVER
#   service=PR_SHARENAME
#password="password"
#
#Например,
#   server=PAULS_PC
#   service=CJET_371
#   password=""

#
# Файл для отладочной информации, можно изменить на /dev/null
#
logfile=/tmp/smb-print.log

spool_dir=/var/spool/lpd/lp
config_file=$spool_dir/.config
```

```
eval `cat $config_file`

echo "server $server, service $service" >> $logfile

(
    echo translate
    echo "print -"
    cat
) | /usr/bin/smbclient "\\\\$server\\$service" $password
-U $user -N -P >> $logfile
```

Теперь вы можете печатать на сетевом принтере. Но, тем не менее, я все же рекомендую прочитать руководство по пакету Samba для получения более подробной информации о печати на сетевых принтерах.

12.2.6. Конфигуратор SWAT

Конфигуратор SWAT (*Samba Web-based Administrative Tool*) предназначен для настройки пакета Samba через веб-интерфейс. Как и другие конфигураторы, SWAT предоставляет удобный графический интерфейс для администрирования сервера Samba. Основным преимуществом данного конфигуратора является то, что вам не нужно находиться за компьютером, который вы администрируете: администрировать сервера Samba вы можете с любого компьютера вашей сети. Как и при работе с другими конфигураторами, при работе со SWAT вам не нужно знать ни названия, ни расположения, ни формата конфигурационных файлов.

Для установки SWAT нужно установить пакет `samba-swat`.

После установки пакета проследите за тем, чтобы в вашем файле `/etc/services` была следующая запись:

```
swat 901/tcp
```

Конфигуратор SWAT для своей работы использует протокол TCP и порт 901. Тем не менее, вы можете назначить любой другой порт. При изменении номера порта не забудьте изменить номер порта в файле `/etc/inetd.conf` или `/etc/xinetd.conf`.

При этом, в конфигурационном файле `/etc/xinetd.conf` суперсервера **xinetd** должна присутствовать секция следующего содержания (листинг 12.10):

Листинг 12.10. Настройки службы SWAT

```
service swat
{
 disable   = no
 port    = 901
 socket_type = stream
 wait      = no
 only_from    = 127.0.0.1
 user    = root
 server    = /usr/sbin/swat
 log_on_failure += USERID
}
```

Если вы хотите конфигурировать сервер Samba с любого компьютера вашей сети, закомментируйте директиву **only_from= 127.0.0.1** или установите любые другие параметры доступа к SWAT.

Теперь нужно перезапустить суперсервер. Для этого введите команду:

```
# /etc/init.d/xinetd restart
```

Можно также заставить суперсервер перечитать свой конфигурационный файл:

```
# killall -HUP xinetd
```

Все! Настройка SWAT завершена, и теперь можно приступить к конфигурированию Samba с помощью SWAT. Для этого запустите свой любимый браузер и введите URL: http://host:901.

После установления соединения и авторизации вы увидите основное окно конфигуратора (рис. 12.2), в котором все самое нужное находится «под рукой» администратора: от документации до паролей пользователей.

В разделе **Globals** определяются значения глобальных переменных. Вы можете получить подсказку по тому или иному параметру, перейдя по ссылке **Help**. Установить значение по умолчанию можно, нажав на кнопку **Set Default**.

В разделе **Shares** определяются общие ресурсы, а в разделе **Printers** — общие принтеры. Состояние сервера Samba можно посмотреть в разделе **Status**. Здесь же можно запустить, остановить и перезапустить сервисы SMB и NMB. В этом разделе можно завершить любое соединение с сервером Samba, а также просмотреть состояние соединения.

В разделе **Passwords** определяются пользователи, которые имеют доступ к серверу Samba.

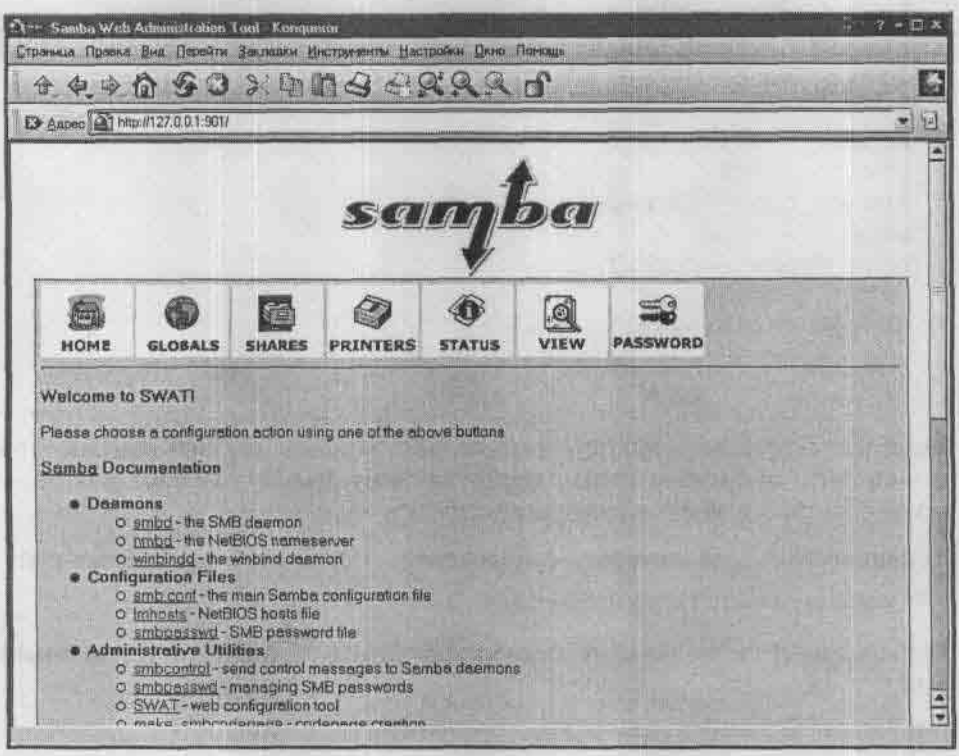


Рис. 12.2. *Окно конфигуратора SWAT*

12.2.7. Samba и безопасность

Вряд ли стоит разрешать доступ к вашему Samba-серверу всем желающим. Целесообразно разрешить доступ только определенным пользователям, которые должны быть зарегистрированы на сервере Samba. Напомню, что для создания пользователя используется команда adduser, а для изменения его пароля — passwd.

```
# adduser -s /bin/false samba-user
# passwd samba-user
```

Первая команда создает пользователя сервера Samba под именем `samba-user`. Параметр **-s** устанавливает «оболочку» для этого пользователя — `/bin/false`. Эта псевдооболочка не позволяет вводить команды, поэтому всем пользователям, которые не нуждаются в работе из командной строки, рекомендуется назначать ее из соображений безопасности.

Samba использует свой файл паролей — `/etc/samba/smbpasswd`. В нем всего три поля: регистрационное имя пользователя, UID и хешированный

пароль. Для добавления пользователя в файл /etc/samba/smbpasswd и изменения его пароля Samba использует команду **smbpasswd**:

```
# smbpasswd samba-user
```

Совет: если вы хотите добавить всех пользователей из файла /etc/passwd в файл /etc/samba/smbpasswd, то используйте для этого следующую команду:

```
# cat /etc/passwd | mksmbpasswd.sh > /etc/samba/smbpasswd
```

Не забудьте изменить права доступа к файлу /etc/samba/smbpasswd:

```
# chmod 600 /etc/samba/smbpasswd
```

Осталось только сделать небольшие изменения в секции [global] файла конфигурации /etc/smb.conf:

```
security = user
```

12.2.8. Оптимизация Samba

Ваш Samba-сервер медленно работает, и вы уже устали от постоянных жалоб пользователей? Лучшим выходом из данной ситуации будет покупка нового винчестера. Лучше всего покупать SCSI-винчестер со скоростью 10000 оборотов в минуту. В крайнем случае подойдет IDE-диск, поддерживающий режим ATA133 и работающий со скоростью 7200 оборотов в минуту. При покупке такого винчестера обратите внимание на следующие факторы:

- ваша материнская плата должна поддерживать режим ATA133;
- у вас должен быть специальный кабель для подключения жесткого диска (если материнская плата поддерживает ATA100/133, такой кабель у вас будет);
- покупайте винчестер с большим объемом кэша: на рынке можно найти несколько моделей винчестеров, различающихся маркировкой, но в одном случае вы получаете 2 Мб кэша, а во втором — 640 Кб (или 1 Мб, но все равно меньше, чем в более дорогой модели).

Если же денег на покупку такого винчестера нет, попробуем оптимизировать наши настройки. Во-первых, установите значение **yes** для директивы **wide links** в файле конфигурации smb.conf — это должно повысить производительность сервера, если в общих каталогах имеются ссылки.

Во-вторых, в дистрибутивах на ядре 2.4 есть демон **bdflush**. Он занимается записью буферов, содержащих модифицированные данные файловой системы, на диск. Настройки по умолчанию не очень эффективны:

```
30 64 64 256 500 3000 60 0 0
```

Попробуем немного оптимизировать настройки **bdflush**. Для этого откройте файл `/etc/sysctl.conf` и добавьте в него следующую строку:

```
vm.bdflush = 80 500 64 64 15 6000 6000 0 0
```

Этим мы говорим демону, чтобы он записывал буферы на диск, когда буфер кэша файловой системы заполнен на 80 процентов.

После этого нужно перезапустить сервис network:

```
# service network restart
```

Если вам нельзя ни на секунду останавливать сервис network, вместо модификации файла `/etc/sysctl.conf` введите следующую команду:

```
# sysctl -w vm.bdflush=«80 500 64 64 15 6000 6000 0 0»
```

Чтобы заставить ваш Samba-сервер работать быстрее, попробуем поэкспериментировать с кэшированием памяти: мы будем использовать минимум 60 процентов памяти для кэша.

```
sysctl -w vm.buffermem = «60 10 60»
```

Примечание

Последние два параметра (10 и 60) сейчас не используются. Значения по умолчанию — «2 10 60».

12.3. Совместное использование каталогов в Linux Mandrake

Конфигуратор **diskdrake-fileshare** позволяет очень быстро настроить пакет Samba для разрешения совместного использования каталогов («расшаривания» каталогов). Убедитесь, что запущены сервисы **nfs** и **smb**, если это не так, запустите их:

```
# service nfs start
# service smb start
```

Запустите конфигуратор **diskdrake-fileshare** (рис. 12.3), выберите опцию «Разрешить всех пользователей» и нажмите Ок.

Если вы выберете опцию «Выборочно», то разделять каталоги смогут только пользователи, входящие в группу `fileshare`.

Как только вы разрешите совместное использование каталогов, пользователи смогут расшарить их в своем файловом менеджере. Например, в **Konqueror** для расшаривания каталога нужно щелкнуть на нем правой кнопкой мыши и выбрать пункт меню **Share**.

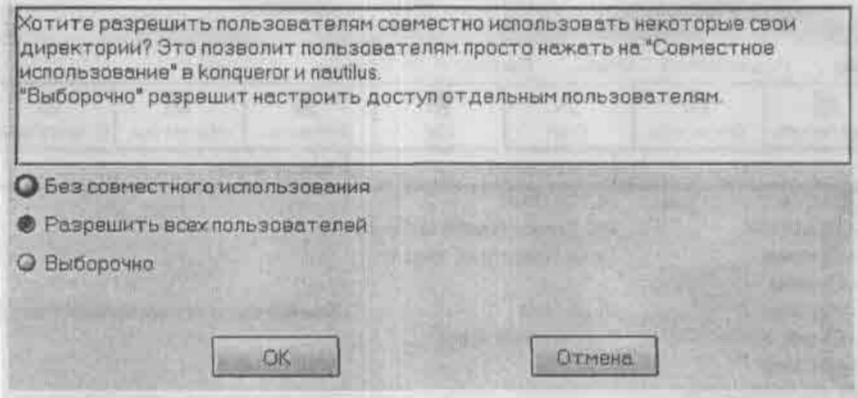


Рис. 12.3. Конфигуратор diskdrake-fileshare

12.4. Программа LinNeighborhood — правильный выбор

Доступ к совместно используемым ресурсам осуществляется с помощью программ **smbclient** и **smbmount**. Как их использовать, вы уже знаете. Не очень удобно, не правда ли? Конечно, если вы работаете в текстовом режиме и система X Window у вас не установлена, то другого выхода у вас нет. Но если у вас установлена графическая среда, то бороздить просторы сети Microsoft гораздо приятнее с помощью программы **LinNeighborhood** (Сетевое окружение Linux).

Установите программу **LinNeighborhood** (пакет и команда для запуска называются так же). Запустите ее и наслаждайтесь!

В верхней части окна отображаются все узлы в Сети, а в нижней — подключенные в данный момент общие ресурсы.

Вы видите, что в сети находятся две машины. Первая — это контроллер домена server.dhsilabs.com, а вторая — это моя машина `den.dhsilabs.com`.

Ресурс **public** в данный момент подключен к каталогу `/home/denis/mnt/server.dhsilabs.com/public`. С этим каталогом можно работать как с обыкновенным каталогом локальной файловой системы.

Чтобы подключить ресурс, выделите его и нажмите кнопку **Подключить** или дважды щелкните на нужном вам ресурсе. Появится окно **Диалог подключения**, в котором нужно указать необходимые параметры подключения.

Рис. 12.4. *Программа LinNeighborhood*

Кнопка **Добавить** позволяет добавить машину, которую вы часто используете. Совсем необязательно, чтобы машина находилась в одной с вами рабочей группе.

Нажав на кнопку **Настройки**, вы можете определить параметры программы, но предлагаемые параметры вполне приемлемы для большинства пользователей. Единственное, что нужно указать, так это имя рабочей группы.

Для поиска компьютера в доступной сети можно использовать команду меню **Опции → Просмотреть всю сеть**.

Глава 13

DNS — СЛУЖБА ИМЕН

- ВВЕДЕНИЕ В DNS
- НАСТРОЙКА КЛИЕНТА DNS
- НАСТРОЙКА СЕРВЕРА DNS
- КЭШИРУЮЩИЙ СЕРВЕР DNS
- ВТОРИЧНЫЙ СЕРВЕР DNS
- ПРОСМОТР DNS-ЗОНЫ. УТИЛИТА NSLOOKUP
- ОПТИМИЗАЦИЯ НАСТРОЕК СЕРВЕРА DNS
- ЗАЩИТА СЕРВЕРА DNS
- ИСПОЛЬЗОВАНИЕ ПОДПИСЕЙ ТРАНЗАКЦИЙ. МЕХАНИЗМ TSIG

13.1. Введение в DNS

Перед началом настройки DNS-сервера давайте разберемся, как он работает. Пространство имен DNS — это иерархическая древовидная структура доменов. Корень дерева доменов обозначается «.». Под ним находятся домены верхнего уровня. Наиболее известные из них вы знаете: com, gov, net, org и т.п. Выделением имен доменов верхнего уровня занимается ICANN — International Corporation for Assigned Names and Numbers. Администрация домена регистрирует домены следующего уровня и так далее.

Всемирная система доменных имен представляет собой распределенную базу данных. Каждый компьютер, подключенный к сети, является клиентом этой базы, а некоторые компьютеры — серверами.

Допустим, пользователь вводит в окне браузера адрес: http://www.yahoo.com. Чтобы установить соединение с компьютером www.yahoo.com, компьютеру пользователя необходимо знать его IP-адрес, поэтому операционная система пользователя пытается разрешить (перевести) имя компьютера в IP-адрес. С этой целью она обращается к серверу DNS. Сервер DNS сначала пытается разрешить имя данного компьютера, используя свой собственный кэш имен. Если требуемое имя компьютера в нем отсутствует, то сервер DNS должен получить его у другого сервера DNS, поэтому база данных DNS и называется распределенной.

Сначала первичный сервер обращается к одному из корневых серверов, список IP-адресов которых находится в его конфигурационных файлах (конкретно, в файле `named.ca`). Далее запрос обрабатывается рекурсивно: корневой сервер знает имена и IP-адреса официальных серверов DNS всех доменов второго уровня и перенаправляет запрос к серверу, который отвечает за домен com, а тот, в свою очередь, — к серверу DNS домена yahoo.com.

Сервер DNS домена yahoo.com возвращает IP-адрес компьютера www — 64.58.76.222 или все адреса, которые сопоставлены этому имени (многие

сетевые операционные системы, в том числе и Linux, позволяют одному имени сопоставлять несколько IP-адресов).

На самом деле, если выполнить разрешение имени `www.yahoo.com`, то сервер DNS возвратит следующие адреса:

64.58.76.222
64.58.76.228
64.58.76.223
64.58.76.176
64.58.76.224
64.58.76.177
64.58.76.227
64.58.76.179.

А официальное, или каноническое, имя компьютера www.yahoo.com — www.yahoo.akadns.net.

13.2. Настройка клиента DNS

Разрешение имен на стороне DNS-клиента выполняет модуль распознавания (резолвер). В ОС Linux он представляет собой набор библиотечных функций. Конфигурационным файлом распознавателя служит `/etc/resolv.conf`, содержащий список DNS-серверов (от 1 до 3), которым можно посылать запросы:

```
search subdomain.domain.com domain.com
nameserver 127.0.0.1   ; сервер по умолчанию
                       ; на локальном компьютере
nameserver 81.3.165.35 ; запасной – у провайдера
```

Директива **search** используется для поиска компьютера в том случае, если указано только имя узла без домена. Например, если вы введете в окне броузера http://host, то сначала будет выполнена попытка обращения к узлу host.subdomain.domain.com, а потом, если узел не будет найден, к узлу host.domain.com.

Комментариев в файле `resolv.conf` не полагается: произвольный текст можно вписывать только в строки с директивами **nameserver** после IP-адреса.

Кроме сервера имен, функции распознавателя могут просматривать и другие источники данных: файл /etc/hosts, СУБД NIS и т.п. Порядок просмотра определен в файле переключения служб `/etc/nsswitch.conf`:

```
hosts: files dns
```

13.3. Настройка сервера DNS

Локальная сеть с выходом в Интернет может пользоваться сервером имен, работающим на компьютере провайдера. Но, учитывая типичную для бывшего СССР скорость соединения, при которой на обращение к серверу DNS провайдера требуется от 10 до 30 секунд, имеет смысл установить собственный сервер DNS — обычно на шлюзе, который используется для выхода в Интернет.

Внутренняя сеть (intranet) может обойтись и файлами `/etc/hosts`, но при большом количестве узлов целесообразно завести DNS-сервер и для нее.

Самый известный сервер имен для Linux — это демон **named** из пакета BIND (Berkeley Internet Name Daemon). Вам понадобится установить еще пакет bind-libs с необходимыми библиотеками и bind-utils, содержащий утилиты командной строки (**dig, host, nslookup**).

Существуют три версии пакета BIND: 4, 8 и 9 (версий 5-7 никогда не было). Сейчас везде используется девятая версия, о которой я и буду рассказывать.

Для установки девятой версии BIND требуются:

- ядро 2.4 или выше;
- библиотека OpenSLL.

Для работы сервера должен быть активизирован сервис network.

Основным конфигурационным файлом named служит `named.conf`, который устанавливается по умолчанию в `/etc` для версии 9 или в `/etc/namedb` (версия 8). Синтаксис этого файла подобен языку С (листинг 13.1).

Листинг 13.1. Примерный файл named.conf

```
  logging {
    category cname {null; };
    };

 options {
    directory "/var/named";
    };

 zone "." {
    type hint;
    file "named.ca";
    };

 zone "dhsilabs.com" {
```

```
    type master;
    file "dhsilabs.com";
    notify no;
    };

  zone "0.0.127.in-addr.arpa" {
    type master;
    file "named.local";
    };

  zone "1.168.192.in-addr.arpa" {
    type master;
    file "192.168.1";
    notify yes;
    };
```

Рассмотрим этот пример подробнее. Обслуживаемая сервером зона (домен без поддоменов) — dhsilabs.com. Рабочий каталог сервера, от которого отсчитываются относительные пути файлов, — `/var/named`. Именно в этом каталоге сервер будет искать файлы `dhsilabs.com`, `named.local`, `192.168.1`, `named.ca`.

Блок **logging** определяет опции протоколирования. За ним следует задание параметров самого сервера — блок **options**. Параметр **directory** определяет рабочий каталог сервера. Этот параметр обязателен, но кроме него в блоке **options** могут присутствовать и другие (**forwarders**, **forward** и т.п.), о которых будет сказано несколько позже.

После блока параметров должны быть перечислены зоны, обслуживаемые сервером. Мы будем обслуживать зону dhsilabs.com. Информация об этой зоне хранится в файле `/var/named/dhsilabs.com`, с помощью которого наш сервер будет преобразовывать имена компьютеров в IP-адреса. Для обратного преобразования служит файл `/var/named/192.168.1`.

Зоны «.» и «0.0.127.in-addr.arpa» — особые. Я не буду их подробно описывать: их назначение вы поймете из дальнейшего текста главы. Файл `named.local` — это файл обратного соответствия, предназначенный для преобразования IP-адресов в имена, то есть, в частности, он используется для преобразования адреса 127.0.0.1 в имя `localhost`.

Файл `named.ca` содержит набор IP-адресов корневых DNS-серверов. При разрешении имени в IP-адрес или наоборот полученная информация кэшируется и остается в памяти сервера определенное время. В дальнейшем, если нужно разрешить имя в IP-адрес или наоборот, ваш DNS-сервер сначала будет искать необходимую ему информацию в кэше. Если ее там не окажется, то сервер обратится к одному из корневых серверов DNS.

Файл named.ca необходимо регулярно обновлять, чтобы он всегда содержал свежие данные (первый раз его нужно обновить сразу же после установки сервера, несмотря на то, что этот файл будет только что создан). Если ваш DNS-сервер предназначен для обслуживания только внутренней сети без выхода в Интернет, то зону корневых серверов нужно удалить.

Файл данных сервера имен dhsilabs.com непосредственно служит для преобразования имен в IP-адреса (листинг 13.2).

Записи в этом файле называются записями ресурсов. Формат записи ресурса следующий:

```
[имя_домена] [TTL] <тип_сети> <тип_записи> <данные>
```

Где:

- **Имя_домена** обязательно для первой записи в файле, оно всегда должно начинаться с первой колонки. Для следующих записей это поле можно опускать. Символ @ обозначает текущий домен.
- **Необязательное поле TTL** (Time to Live) — это время в секундах, в течение которого данные в кэше считаются достоверными. Если значение не указано, то оно берется из записи SOA (см. ниже).
- **Тип_сети** может содержать значение IN (Internet) или HS (информационная служба Hesiod).
- **Тип_записи**: типы записи ресурсов приведены в таблице 13.1. Остальные поля данных зависят от типа записи ресурса.

Пустые строки и строки, начинающиеся с точки с запятой, считаются комментариями.

Листинг 13.2. Файл dhsilabs.com

```
@   IN SOA  den.dhsilabs.com. hostmaster.dhsilabs.com. (
    93011120  ; серийный номер
    10800     ; обновление каждые 3 часа
    3600    ; повтор каждый час
    3600000     ; хранить информацию 1000 часов
    86400)    ; TTL записи - 24 часа
  IN   NS   den.dhsilabs.com.
  IN   A    192.168.1.1
  IN   MX   150  den.dhsilabs.com.

  den  IN   A    192.168.1.1
  IN   HINFO     INTEL CELERON (LINUX)
  IN   MX   100  den
  IN   MX   150  evg.dhsilabs.com.
```

```
ns      IN   CNAME     den.dhsilabs.com.
www     IN   CNAME     den.dhsilabs.com.
ftp     IN   CNAME     den.dhsilabs.com.
mail    IN   CNAME     den.dhsilabs.com.

evg     IN   A         192.168.1.2
IN      MX   100       den.dhsilabs.com.

localhost  IN   A   127.0.0.1
```

Тип записи SOA (*Start of Authority*) означает начало зоны. Для каждой зоны такая запись единственна и должна стоять первой по порядку. Она содержит имя зоны, почтовый адрес ее администратора (где знак @ заменен на «.») и параметры обновления данных. Круглые скобки служат для разбиения одной записи на несколько строк.

Имя домена может быть сокращенным или полным. Полностью определенные имена заканчиваются символом точки. Если точки нет, то имя считается сокращенным и к нему автоматически добавляется имя текущего домена: так, `den.dhsilabs.com` без точки будет интерпретировано как `den.dhsilabs.com.dhsilabs.com`. Не делайте этой распространенной ошибки, не забывайте ставить точку после имени домена.

Серийный номер — это число, на которое ориентируются подчиненные серверы, перезапрашивая данные у главного сервера зоны в том случае, если его серийный номер больше, чем у них. Обычно номер представляет собой дату последнего изменения файла данных зоны.

Типы записи ресурсов *Таблица 13.1*

Запись	Назначение
SOA	Начало полномочий: определение DNS-зоны
NS	Определение сервера имен
A	IP-адрес (IPv4), соответствующий имени компьютера. Для IPv6-адресов используется тип A6
PTR	Обратное преобразование: имя компьютера, соответствующее IP-адресу
MX	Mail eXchange: почтовый сервер, обслуживающий домен. Можно перечислить несколько почтовых серверов, указав приоритет: чем меньше число, тем приоритет выше
CNAME	Каноническое имя узла, к которому преобразуются псевдонимы: так, по адресу http://www.dhsilabs.com обращение будет производиться к den.dhsilabs.com
HINFO	Информация об узле: операционной системе и аппаратном обеспечении. Рекомендую не заполнять эту запись или использовать заведомо неправильные данные. Чем меньше информации о вашей сети получит злоумышленник, тем сложнее ему будет атаковать ее
TXT	Произвольный текст — комментарии или нестандартная информация

Содержание файлов обратного преобразования, устанавливающих соответствие между IP-адресами и именами при помощи записей типа PTR, приведено в листингах 13.3 и 13.4. Внимание! IP-адреса указываются в обратном порядке: 2.1.168.192. Если указан неполный IP, например, 1, то к нему будет добавлен адрес подсети 1.168.192.

Листинг 13.3. Файл named.local

```
@   IN   SOA  dhsilabs.com. root. dhsilabs.com. (
      199609203 ;серийный номер
      28800   ;обновление каждые 8 часов
      7200    ;повтор каждые 2 часа
      604800    ;хранить информацию 168 часов (1 неделю)
      86400)    ;TTL записи - 24 часа
    NS   dhsilabs.com.
1     PTR  localhost.
```

Листинг 13.4. Файл 192.168.1

```
@   IN SOA  den.dhsilabs.com. hostmaster.dhsilabs.com. (
      93011120     ; серийный номер
      10800      ; обновление каждые 3 часа
      3600       ; повтор каждый час
      3600000    ; хранить информацию 1000 часов
      86400 )      ; TTL записи - 24 часа
@   IN   NS   den.dhsilabs.com
1   IN   PTR  den.dhsilabs.com
2.1.168.192  IN   PTR  evg.dhsilabs.com
```

13.3.1. Обновление корневого кэша

Если вы настраиваете сервер DNS только для своей внутренней сети, которая не имеет выхода в Интернет, то не спешите обновлять файл кэша! Он вам вообще не нужен. Вы также должны удалить зону, описывающую корневой кэш в файле `named.conf`.

Обычно файл `named.ca` содержит примерно такую информацию:

Листинг 13.5. Файл корневых серверов

```
.   6D IN NS        G.ROOT-SERVERS.NET.
.   6D IN NS        J.ROOT-SERVERS.NET.
.   6D IN NS        K.ROOT-SERVERS.NET.
.   6D IN NS        L.ROOT-SERVERS.NET.
```

```
.    6D IN NS        M.ROOT-SERVERS.NET.
.    6D IN NS        A.ROOT-SERVERS.NET.
.    6D IN NS        H.ROOT-SERVERS.NET.
.    6D IN NS        B.ROOT-SERVERS.NET.
.    6D IN NS        C.ROOT-SERVERS.NET.
.    6D IN NS        D.ROOT-SERVERS.NET.
.    6D IN NS        E.ROOT-SERVERS.NET.
.    6D IN NS        I.ROOT-SERVERS.NET.
.    6D IN NS        F.ROOT-SERVERS.NET.

;; ADDITIONAL SECTION:
G.ROOT-SERVERS.NET.     5w6d16h IN A    192.112.36.4
J.ROOT-SERVERS.NET.     5w6d16h IN A    198.41.0.10
K.ROOT-SERVERS.NET.     5w6d16h IN A    193.0.14.129
L.ROOT-SERVERS.NET.     5w6d16h IN A    198.32.64.12
M.ROOT-SERVERS.NET.     5w6d16h IN A    202.12.27.33
A.ROOT-SERVERS.NET.     5w6d16h IN A    198.41.0.4
H.ROOT-SERVERS.NET.     5w6d16h IN A    128.63.2.53
B.ROOT-SERVERS.NET.     5w6d16h IN A    128.9.0.107
C.ROOT-SERVERS.NET.     5w6d16h IN A    192.33.4.12
D.ROOT-SERVERS.NET.     5w6d16h IN A    128.8.10.90
E.ROOT-SERVERS.NET.     5w6d16h IN A    192.203.230.10
I.ROOT-SERVERS.NET.     5w6d16h IN A    192.36.148.17
F.ROOT-SERVERS.NET.     5w6d16h IN A    192.5.5.241
```

Для установки файла корневого кэша следует установить пакет caching-nameserver, но я рекомендую получить и установить самую новую версию. Для этого подключитесь к Интернету, запустите сервер DNS, а затем выполните команду

```
# nslookup | tee ns
```

В ответ на приглашение утилиты **nslookup** введите две команды:

```
> set q=ns (или set type=ns)
> .
```

На экране вы увидите список корневых серверов DNS, который будет помещен в файл ns. Для преобразования файла ns в формат `named.ca` воспользуйтесь следующим awk-сценарием (листинг 13.6), вызвав его так:

```
# reformat ns named.ca
```

Листинт 13.6. Сценарий reformat

```
#!/bin/awk
awk ' BEGIN {
/root/ { print ". IN NS " $4"." }
/internet/ { print $1"." " 999999 IN A " $5 }
END '
```

Теперь осталось скопировать `named.ca` в каталог `/var/named`, и на этом — все.

Можно обновить корневой кэш и проще, воспользовавшись утилитой *dig*:

```
# dig @a.root-servers.net . ns > named.ca.new
```

или

```
# dig @198.41.0.4 . ns > named.ca.new
```

После этого остается только заменить старый файл `named.ca` новым файлом `named.ca.new`.

13.4. Кэширующий сервер DNS

Каждую зону обслуживает один главный сервер имен, хранящий официальную копию данных о зоне. Он называется «авторитетным», потому что его ответ — точный и окончательный. В зоне может быть также сколько угодно кэширующих серверов, у которых собственных данных нет: они накапливают данные, кэшируя ответы на свои запросы. Ответ кэширующего сервера неавторитетен, зато быстр. Обычно они используются для уменьшения DNS-трафика во внутренней сети.

Если собственного домена у вас нет, то имеет смысл возложить обработку DNS-запросов на провайдера, создав у себя кэширующий DNS-сервер. Вместо того, чтобы запрашивать последовательно несколько удаленных корневых серверов, он будет отсылать в сеть только один запрос на разрешение имени (DNS-серверу провайдера) и получать только один окончательный ответ. Это особенно полезно, если у вас плохое соединение с Интернетом.

13.4.1. Настройка кэширования на DNS-сервере

Для того, чтобы насладиться такой возможностью, следует в блок **options** файла `named.conf` добавить следующие параметры:

```
forward first;
forwarders {
  81.3.165.35;
  81.3.150.2;
};
```

Директива **forwarders** задает заключенный в фигурные скобки список IP-адресов DNS-серверов, которым ваш DNS-сервер будет переадресовывать запросы вместо того, чтобы отвечать на них самому. IP-адреса перечисляются через точку с запятой.

Директива **forward** может принимать одно из двух следующих значений:

- **only** — ваш DNS-сервер никогда не должен предпринимать попыток обработать запрос самостоятельно;
- **first** — ваш DNS-сервер должен пытаться сам обработать запрос, если указанные далее параметром **forwarders** сервера DNS не были найдены.

Без директивы **forwarders** директива **forward** не имеет смысла.

Таким образом, возвращаясь к настройке сервера, весь файл `named.conf` будет выглядеть так:

Листинг 13.7. Файл named.conf кэширующего сервера DNS

```
options {
    directory "/var/named";
    forward first;
    forwarders {
      81.3.165.35;
      81.3.150.2;
    };
    // Раскомментируйте следующую строку, если брандмауэр
    // мешает работе службы DNS
    // query-source port 53;
};

zone "." {
    type hint;
    file "named.ca";
    };

zone "0.0.127.in-addr.arpa" {
    type slave;
    file " named.local ";
    };
```

Обратите внимание, что в этом примере уже не поддерживается зона dhsilabs.com.

13.4.2. Возможные проблемы и их решение

Как правило, кэширующий сервер запускается на отдельном компьютере, который подключается к Интернету по коммутируемому соединению. Нужно учитывать, что сервер DNS сразу требует обращения к какому-нибудь сетевому ресурсу. В нашем же случае, если соединение не

установлено, то устройство `ppp0` существовать не будет, а **named** будет страшно ругаться на то, что сеть недоступна. При этом недоступным окажется даже интерфейс `lo`, а программа **nslookup**, если она нам понадобится без существования сети, просто «подвиснет», ожидая ответа от сервера DNS.

Есть два способа решить данную проблему. Какой использовать — это решать вам. Первый заключается в том, чтобы запускать сервер DNS после установления PPP-соединения и останавливать перед его разрывом.

Для управления демоном **named** служит утилита **rndc** (**ndc** в BIND 8). Ее можно использовать с параметрами start, stop, reload (перегрузить файлы данных зоны, если в них произошли изменения), restart. Команду `rndc start` следует включить в сценарий установления PPP-соединения, а команду `rndc stop` — в сценарий разрыва.

Второй способ состоит в том, чтобы при постоянно работающем сервере DNS подменять файл корневого кэша `named.ca`. В отсутствие PPP-соединения по этому имени находится пустой файл, а сценарий установки соединения содержит команду, копирующую на его место нормальный файл кэша `named.ca.ppp-on`. При использовании этого способа в ваших протоколах (журналах) будут регулярно появляться сообщения примерно такого содержания:

```
Jan 5 16:10:11 den named[10147]: No root nameserver for class IN
```

Для полноты картины хочу отметить, что, если при использовании DNS у вас возникают проблемы с монтированием удаленных файловых систем, запускайте сервер **named** после запуска **nfsd** и **mountd**.

13.5. Вторичный сервер DNS

Вы когда-нибудь обращали внимание, что у любого уважающего себя провайдера есть два сервера DNS — первичный (primary или master) и вторичный (secondary или slave)? Вторичный сервер копирует данные о зоне с первичного. Эта операция называется зонной пересылкой. В любой зоне должен быть хотя бы один вторичный сервер на тот случай, если с первичным сервером что-нибудь случилось или он просто не в состоянии обработать большое количество запросов клиентов. Получив отказ от первичного сервера, система разрешения имен обращается к вторичному. Для повышения надежности работы службы имен желательно включать вторичный сервер в другую сеть и в другую цепь питания.

Для вторичного сервера DNS, обслуживающего домен dhsilabs.com, секция зоны в файле `named.conf` будет выглядеть так:

```
zone " dhsilabs.com" {
 type slave;
 file " dhsilabs.com";
 masters { 192.168.1.1; 192.168.1.2; };
};
```

IP-адреса основных серверов DNS вашей сети указываются в директиве masters через точку с запятой. Вторичный сервер, в отличие от кэширующего, всегда должен иметь тип slave.

13.6. Просмотр DNS-зоны. Утилита nslookup

Утилита **nslookup**, служащая для просмотра информации о зоне (домене), входит в пакет `bind-utils` и в популярных дистрибутивах устанавливается по умолчанию. Она есть также в Windows NT/2000/XP.

Зоны бывают разные: одни содержат информацию о компьютерах в домене и служат для преобразования имени компьютера в IP-адрес и наоборот, другие содержат информацию о корневых серверах — зона «.». Последняя зона относится к типу **hint** — подсказка. Зоны для разрешения имен обычно имеют тип **master** (главный), а зоны вторичных серверов относятся к типу **slave** (подчиненный). Обычно просмотр зоны разрешается только определенным, доверенным узлам. Итак, запустите **nslookup**:

```
# nslookup
Default Server: ns4.obit.ru
Address: 81.3.165.35
>
```

Для того, чтобы получить информацию от сервера, нужно ввести в строку приглашения тип запроса: `set q=<тип>` (или `set type=<тип>`). Типы запросов перечислены в таблице 13.2.

Типы запросов *Таблица 13.2*

Тип	Описание
soa	Начало полномочий
a	Преобразование имени в IP-адрес узла
aaaa	Отображение IPv6-адреса узла
ns	Отображение информации о сервере DNS
ptr	Преобразование IP-адреса в имя узла
wks	Распространенные службы
hinfo	Информация об аппаратном обеспечении узла
mx	Информация о почтовых серверах домена
txt	Отображение записи общего назначения
cname	Отображение канонического имени
any	Отображение всех ресурсных записей

Теперь рассмотрим несколько практических примеров. Допустим, вы знаете имя узла — www.server.com. Давайте посмотрим, какая информация будет выведена при указании типа any:

```
>set q=any
>server.com
Server: myserver.domain.com
Address: 127.0.0.1
Non-authoritative answer:
server.com nameserver = comp1.server.com
server.com nameserver = comp2.server.com
server.com nameserver = comp3.server.com
Authoritative answers can be found from:
server.com nameserver = comp1.server.com
server.com nameserver = comp2.server.com
server.com nameserver = comp3.server.com
comp1.server.com internet address = 323.111.200.1
comp2.server.com internet address = 323.111.200.2
comp3.server.com internet address = 323.555.200.3
```

А сейчас посмотрим информацию о других узлах в этой сети:

```
>ls server.com.
[comp2.server.com]
server.com. 323.111.200.2
server.com. server = comp1.server.com
server.com. server = comp2.server.com
server.com. server = comp3.server.com
mail 323.111.200.17
gold 323.111.200.22
www.ie 323.111.200.11
jersild 323.111.200.25
comp1 323.111.200.1
comp3 323.111.200.3
parasit3 323.111.200.20
www.press 323.111.200.30
comp1 323.111.200.1
www 323.111.200.2
```

Но ответ мог быть и таким:

```
[server.com]
Can't list domain server.com: Query refused
```

Чтобы разрешить передачу зоны определенным узлам (а, значит, запретить всем остальным), в файле конфигурации DNS-сервера применяется директива **allow-transfer**. В следующем примере трансфер зоны разрешен

узлам 10.1.1.1 и 10.1.2.1, то есть все остальные узлы в ответ на запрос nslookup ls получат ответ «Query refused»:

```
options {
 allow-transfer {
 10.1.1.1;
 10.1.2.1;
};
};
```

Вторичный сервер DNS не передает никакой информации о зоне, поэтому обязательно укажите следующую строку в его файле конфигурации (в секции options):

```
allow-transfer { none; }
```

13.7. Оптимизация настроек сервера DNS

Как любой хороший администратор, вы хотите, чтобы ваш сервер DNS быстро обслуживал запросы клиентов. Но к вашему серверу могут подключаться пользователи не из вашей сети, а, например, из сети конкурирующего провайдера. Тогда ваш сервер будет обслуживать «чужих» клиентов. Непорядок! Директива **allow-query** позволяет указать адреса узлов и сетей, которым можно использовать наш сервер DNS:

```
allow-query { 192.168.1.0/24; localhost; };
```

В данном примере мы позволяем использовать наш сервер узлам из сети 192.168.1.0 и узлу `localhost`. Целесообразно разрешить рекурсивные запросы только из сети 192.168.1.0 и узлу `localhost`:

```
allow-recursion { 192.168.1.0/24; localhost; };
```

Обычно взлом любой сети начинается со сбора информации — о структуре сети, об установленном программном обеспечении и его версиях и т.п. Мы можем заставить сервер DNS не сообщать номер своей версии, а выдавать произвольное сообщение:

```
version "Made in USSR";
```

Все перечисленные директивы должны быть указаны в секции **options** файла конфигурации `/etc/named.conf`:

```
options {
 allow-query { 192.168.1.0/24; localhost; };
 allow-recursion { 192.168.1.0/24; localhost; };
 allow-transfer { 10.1.1.1; 10.1.2.1; };
 version "Made in USSR";
}
```

13.8. Защита сервера DNS

13.8.1. Настройка и запуск DNS-сервера в chroot-окружении

Из соображений безопасности рекомендуется запускать все сетевые сервисы в так называемом chroot-окружении (*change root*). Это файловая система, повторяющая структуру корневой файловой системы, но содержащая только те файлы, которые необходимы для запуска нашего сетевого сервиса. Взломав сетевой сервис и получив доступ к корневой файловой системе, злоумышленник не сможет повредить всей системе в целом, поскольку он получит доступ только к файлам данного сервиса. Некоторые сетевые службы не могут работать в chroot-окружении. BIND — может, и сейчас я покажу, как это организовать.

Не нужно создавать отдельный раздел на диске для каждого сетевого сервиса: нужно только создать каталог, например, `root-dns`, в который вы скопируете все файлы, необходимые для запуска сервера DNS. Потом, при запуске сервиса, будет выполнена команда **chroot** для этого сервиса, которая подменит файловую систему. А так как в каталоге `root-dns`, который станет каталогом /, имеются все необходимые файлы для работы BIND, то для сервиса запуск и работа в chroot-окружении будут совершенно прозрачными.

Сразу нужно оговорить, что настраивать chroot-окружение мы будем для девятой версии BIND, поскольку это значительно проще, чем для восьмой версии. В отличие от восьмой версии, где для настройки chroot-окружения нужно было копировать все бинарные файлы или библиотеки, необходимые для запуска BIND, для работы девятой версии достаточно скопировать только файлы конфигурации и зон, обслуживаемых сервером.

Создайте каталоги корневой файловой системы сервера DNS:

```
# mkdir -p /root-dns
# mkdir -p /root-dns/etc
# mkdir -p /root-dns/var/run/named
# mkdir -p /root-dns/var/named
```

Остановите сервер DNS, если он запущен:

```
# service named stop
```

Переместите файл конфигурации, файлы зон и файл `/etc/localtime` (он нужен для корректной работы сервера DNS со временем) в каталог `/root-dns`:

```
# mv /etc/named.conf /root-dns/etc/
```

```
# mv /var/named/* /root-dns/var/named/
# chown named.named /chroot/etc/named.conf
# chown -R named.named /root-dns/var/named/*
```

Защитите от редактирования и удаления файл конфигурации:

```
# chattr +i /root-dns/etc/named.conf
```

Добавьте в файл `/etc/sysconfig/named` строку:

```
ROOTDIR="/root-dns/"
```

Все, теперь можно запустить сервер ***named***:

```
# service named start
```

Проверьте, все ли сделано правильно:

```
$ ps -ax | grep named.
5380  ?    S  0:00 named -u named -t /root-dns/
5381  ?    S  0:00 named -u named -t /root-dns/
5382  ?    S  0:00 named -u named -t /root-dns/
```

13.9. Использование подписей транзакций. Механизм TSIG

В девятой версии BIND появилась возможность создавать подписи транзакций (TSIG — *Transaction SIGnatures*). Механизм TSIG работает так: сервер получает сообщение, подписанное ключом, проверяет подпись и, если она «правильная», сервер отправляет ответ, подписанный тем же ключом.

Механизм TSIG очень эффективен при передаче информации о зоне, уведомлений об изменении зоны и рекурсивных сообщений. Согласитесь, проверка подписи надежнее, чем проверка IP-адреса. Злоумышленник может вывести из строя вторичный сервер DNS банальной атакой на отказ, и, пока администратор будет «поднимать» вторичный сервер, он заменит свой IP-адрес адресом вторичного сервера. При использовании TSIG задача злоумышленника значительно усложняется: ведь ему придется «подобрать» 128-битный MD5-ключ, а вероятность такого подбора близка к нулю.

Итак, приступим к настройке. Остановите сервис **named**.

Сгенерируйте общие ключи для каждой пары узлов. Общие ключи используются при «общении» первичного и вторичного серверов DNS.

```
[root@dns]# dnssec-keygen -a hmac-md5 -b 128 -n HOST ns1-ns2
Kns1-ns2.+157+49406
```

Мы используем алгоритм HMAC-MD5, 128-битное шифрование, ns1-ns2 — это имя ключа. После выполнения этой команды будет создан файл `Kns1-ns2.+176+40946.private`. Откройте его в любом текстовом редакторе. Вы увидите примерно следующее:

```
Private-key-format: v1.2
Algorithm: 157 (HMAC_MD5)
Key: ms7dfts87Cjhj7FD9lk7a3==
```

Ключ «ms7dfts87Cjhj7FD9lk7a3==» и будет тем секретом, который будет передаваться между серверами. Запишите это значение на бумаге (которую потом нужно будет уничтожить) и удалите файлы `Kns1-ns2.+157+49406.key` и `Kns1-ns2.+157+49406.private`.

Добавьте в файлы конфигурации первичного и вторичного серверов DNS директивы, указывающие на использование ключа (листинги 13.8 и 13.9).

Листинг 13.8. Фрагмент файла named.conf первичного сервера DNS

```
key ns1-ns2 {
 algorithm hmac-md5;
 secret "ms7dfts87Cjhj7FD9lk7a3==";
};

# прописываем вторичный сервер DNS - 192.168.1.2:
server 192.168.1.2 {
 keys { ns1-ns2; };
};

options {
# разрешаем передачу зоны вторичному серверу DNS
 allow-transfer { 192.168.1.2; };
};
```

Листинг 13.9. Фрагмент файла named.conf вторичного сервера DNS

```
key ns1-ns2 {
 algorithm hmac-md5;
 secret "ms7dfts87Cjhj7FD9lk7a3==";
};

# прописываем первичный сервер DNS - 192.168.1.1:
server 192.168.1.1 {
 keys { ns1-ns2; };
};
```

```
options {
# никому не передаем зону
 allow-transfer { none };
};
```

Можно также настроить передачу зоны «по ключу». Для этого директива allow-transfer в файле конфигурации первичного сервера DNS должна выглядеть так:

```
allow-transfer { key ns1-ns2; };
```

Осталось только «спрятать» файлы конфигурации обоих серверов DNS от посторонних глаз — ведь они содержат ключи в открытом виде.

```
chmod 600 named.conf
```

Запустите сервис **named**. Теперь о его безопасности будет заботиться TSIG.

Глава 14

ПОЧТОВЫЙ СЕРВЕР

- УСТАНОВКА И НАСТРОЙКА SENDMAIL
- АУТЕНТИФИКАЦИЯ В SENDMAIL
- АГЕНТ ДОСТУПА — FETCHMAIL
- АВТОМАТИЧЕСКАЯ СОРТИРОВКА ВХОДЯЩЕЙ ПОЧТЫ — ПРОГРАММА PROCMAIL
- СОЗДАНИЕ СПИСКА РАССЫЛКИ
- ЗАЩИТА ПРОГРАММЫ SENDMAIL. ПРОГРАММА SMRSH

Неискушенные пользователи обычно принимают за систему электронной почты ту программу, с помощью которой они читают и пишут сообщения (**mail**, **pine**, **Netscape Messenger**, **MS Outlook Express**). На самом деле эта система состоит из нескольких компонентов (рис. 14.1):

- пользовательские агенты — те самые **mail**, **pine** и т.п., реализующие интерфейс к системе пересылки почты;
- транспортный агент (MTA, Mail Transfer Agent), пересылающий сообщения с одного компьютера на другой;
- агент доставки, сортирующий почту и помещающий ее в ящики пользователей или другое хранилище сообщений;
- агент доступа, скачивающий доставленную почту из хранилища по протоколу POP или IMAP, и агент подачи, доставляющий письма от пользователей на сервер исходящих сообщений по протоколу SMTP. В качестве агента подачи может работать транспортный агент.

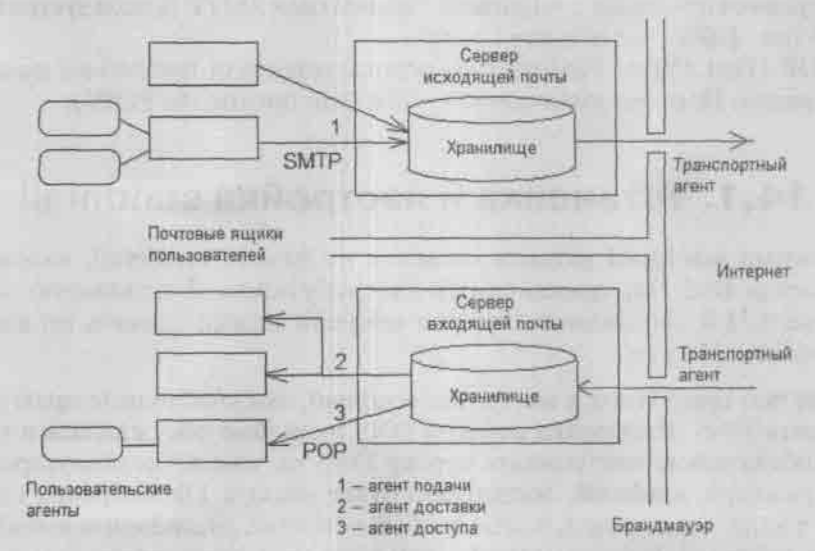


Рис. 14.1. *Структура почтовой системы*

При установке почтового сервера в сети организации или в любой другой компьютерной сети необходимо настраивать именно транспортный агент. Основными транспортными агентами пересылки почты на сегодняшний день являются **sendmail**, **postfix** и **qmail**. Кроме основной функции — пересылки сообщений электронной почты — каждый их них имеет собственные эксклюзивные возможности.

Старейшим транспортным агентом, фактическим стандартом, работавшим еще в самых ранних версиях UNIX, является **sendmail**. Разработчики **qmail** обратили особое внимание на обеспечение безопасности: до 1998 года реализация функций защиты в **sendmail** до такой степени оставляла желать лучшего, что в конференциях вообще не рекомендовалось ее использовать. Программа **postfix** считается проще других в настройке.

Я решительно предпочитаю **sendmail**, которая в настоящее время обеспечивает должный уровень безопасности и настраивается достаточно легко. В этой главе я объясню, как настроить **sendmail** для организации небольшого почтового сервера с использованием протокола SMTP. В качестве агента доставки я рассмотрю **procmail**, а в качестве агента доступа, работающего по протоколу POP3, — программу **fetchmail**. Для заинтересовавшихся я разместил на сайте `http://dkws.narod.ru` руководство по настройке **qmail**.

Вкратце напомню о протоколах SMTP и POP, которые будем конфигурировать:

- SMTP (Simple Mail Transfer Protocol) — сервис в сетях TCP/IP для передачи почтовых сообщений. Обычно для SMTP используется порт 25 (см. файл `/etc/services`).
- POP (Post Office Protocol) — используется для получения почты с сервера. Порт по умолчанию — 110 (для протокола POP3).

14.1. Установка и настройка sendmail

Программа **sendmail** устанавливается из пакета sendmail, входящего в состав Red Hat-совместимых дистрибутивов. Я использую пакет sendmail-8.11.0. Последнюю версию sendmail можно скачать по адресу `www.sendmail.org`.

Прежде чем приступить к настройке **sendmail**, вам необходимо правильно настроить DNS. Настройка сервера DNS подробно обсуждалась в главе 13. Необязательно настраивать сервер DNS на том же компьютере, где будет работать **sendmail**: достаточно будет указать DNS-сервера вашей сети в файле `/etc/resolv.conf`, чтобы система разрешения имен корректно работала. Впрочем, **sendmail** можно настроить для работы и без использования DNS, но этот вариант я рассматривать не буду.

14.1.1. Базовая настройка sendmail

Основным файлом конфигурации **sendmail** является `sendmail.cf`, расположенный в каталоге `/etc/mail` (в некоторых дистрибутивах — `/etc`, смотри `rpm -ql sendmail`). Об этом файле говорят, что он длиннее, чем лимузин у Билла Гейтса, и что его редактирование происходит в режиме «глаза боятся, руки делают». Если вы не верите мне, откройте этот файл, и вы убедитесь в этом. Редактировать данный файл вручную могут только профессионалы-администраторы или разработчики программы **sendmail**.

К счастью, базовую настройку **sendmail** можно выполнить при помощи графического конфигуратора **netconf**. К сожалению, этот конфигуратор (точнее, пакет linuxconf, в который он входит) в современные дистрибутивы включать перестали. Но не все потеряно: его можно скачать с сайта разработчика `http://www.solucorp.qc.ca/linuxconf`. Многие опытные пользователи Linux критически относятся к возможностям пакета linuxconf и предпочитают прямое редактирование конфигурационных файлов, но для новичка эти утилиты, несомненно, удобнее.

Запустите утилиту **netconf** из-под суперпользователя (рис. 14.2).

Выберите в меню **Mail delivery system**, затем **Configure basic information**. В поле **Present your system as** просто введите свое доменное имя. Затем обязательно отметьте флажок **Accept email for ваш_домен** (рис. 14.3).

Рис. 14.2. *Конфигуратор netconf*

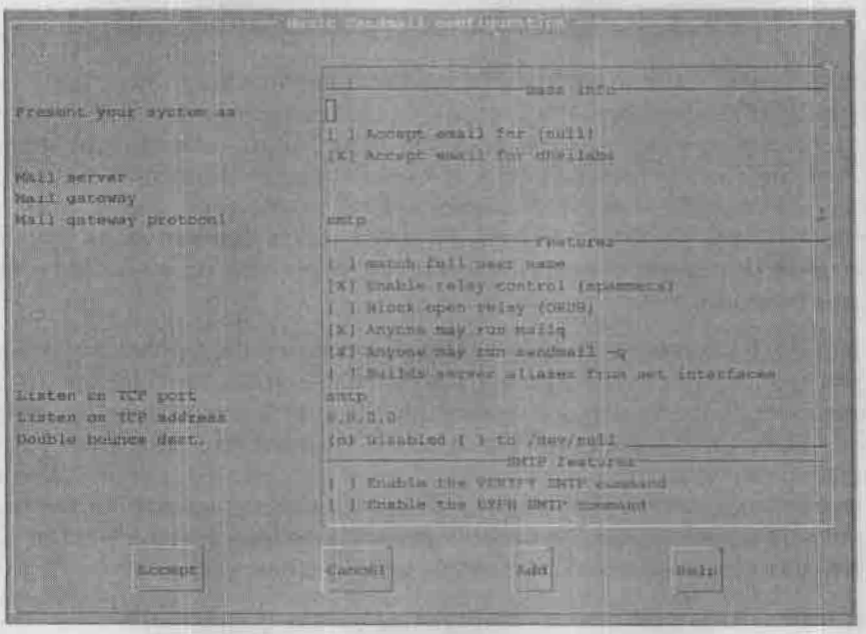


Рис. 14.3. *Основная конфигурация sendmail*

Если вы этого не сделаете, через ваш сервер можно будет перенаправить сообщения на другой сервер — а это лишний трафик через вас, который вам совсем ни к чему. Когда-то даже существовал такой вид атаки на отказ через электронную почту: письмо от несуществующего пользователя not_exists@A.com другому несуществующему пользователю not_exists@B.com отправлялось через компьютер host.com, который позволял перенаправлять сообщения (не включив режим **Accept email for** host.com). Обратно шло сообщение о том, что адресат не существует, но и оно не могло быть доставлено по той же причине... возникала косвенная рекурсия, и сервер host.com падал.

В качестве протокола отправки сообщений (**Mail gateway protocol**) установите **smtp**. Этой информации уже вполне достаточно, чтобы ваш **sendmail** функционировал.

Теперь сделайте так, чтобы **sendmail** принимал почту только с разрешенных адресов. Для этого даже не нужно настраивать сам **sendmail** — достаточно только подправить файлы `/etc/hosts.allow` и `/etc/hosts.deny`. Например, для запрещения доступа всем узлам, кроме компьютеров вашей сети, в файл `/etc/hosts.allow` добавьте строку

```
192.168.1.
```

Здесь имеется в виду, что сеть имеет адрес 192.168.1.0 и маску 255.255.255.0. Точка в конце образца требует сравнивать с образцом первые группы

цифр адреса узла. Подробнее о формате файлов `hosts.allow` и `hosts.deny` вы можете прочитать, введя команду `man hosts.allow`.

Теперь настала очередь POP3. После установки пакета imap у вас практически все настроено, т.е. я хочу сказать, что уже можно проверять конфигурацию. Перезапустите супердемон **inetd** (**xinetd**) и введите следующее:

```
$ telnet <имя_только_созданного_почтовика> 25
Trying 192.168.1.1 ...
Connected to 192.168.1.1
Escape character is '^]'
220 den.dhsilabs.com ESMTP Sendmail 8.11.0/8.8.7 Sun, 17
Jun 2001 10:54:22 +300
```

Это означает, что sendmail работает, осталось проверить, насколько правильно он это делает. С этой целью попробуйте отправить письмо:

```
mail from: me@my.host.com
220  2.1.0 me@my.host.com .... Sender Ok
rcpt to: den@den.dhsilabs.com
220  2.1.5 den@den.dhsilabs.com .... Recipient Ok
```

В качестве адресата (`rcpt to:`) укажите существующее регистрационное имя @ имя почтового сервера. После этого введите команду `data`, потом текст сообщения, а для окончания ввода поставьте точку в пустой строке. Программа **sendmail** сообщит, что сообщение помещено в очередь на отправку. Закончите telnet-сеанс командой `quit`.

Обратите внимание, что узла my.host.com не существует, а программа **sendmail** тем не менее рапортует, что «Sender Ok». Вот почему в настройках **sendmail** лучше включить флажок **Wait for DNS**.

Теперь нужно запустить какой-нибудь почтовый клиент, например **kmail**, и получить почту. Используйте следующие настройки сети в программе **kmail**: **Сеть → Отправка почты.** Там установите SMTP, порт 25, имя сервера — имя вашего почтовика (в рассматриваемом примере это den.dhsilabs.com). Затем добавьте учетную запись для POP3:

- Имя пользователя — den;
- Пароль — пароль, который используется для входа в систему;
- Сервер — den.dhsilabs.com;
- Порт — 110.

В результате вы должны получить то сообщение, которое только что послали. При этом возможны проблемы при разрешении имени. Чтобы их избежать, необходимо правильно настроить DNS или вместо имени почтового сервера использовать его IP-адрес. При добавлении нового пользователя не забудьте установить его пароль для входа в систему. Если

этого не сделать, а пытаться получить почту без указания пароля, то вы получите сообщение «Сбой аутентификации».

Базовая настройка программы **sendmail** с использованием конфигуратора выполняется очень просто и обычно устраивает всех... до тех пор, пока вашим почтовым сервером не заинтересуются спамеры. Для более тонкой настройки нужно ознакомиться с файлами конфигурации программы **sendmail**.

14.1.2. Редактирование конфигурационных файлов

Обычно для редактирования файла `sendmail.cf` используется макропроцессор **m4**. Сначала вы подготавливаете специальный mc-файл, где записаны настройки **sendmail** в более «читабельном» виде. Затем, отредактировав mc-файл, нужно запустить макропроцессор m4 для создания файла конфигурации sendmail:

```
# mv /etc/sendmail.cf /etc/sendmail.cf.orig
# m4 my_config.mc > /etc/sendmail.cf
```

Файл конфигурации по умолчанию, который используется макропроцессором **m4** для создания файла конфигурации программы sendmail (sendmail.cf), находится в каталоге `/usr/share/sendmail-cf/cf`. В более старых версиях программы **sendmail** он может быть расположен в каталоге `/usr/lib/sendmail`.

Как правило, этот файл называется `sendmail.mc`. Иногда он может называться и по-другому, например, `redhat.mc`, если вы используете операционную систему Red Hat или совместимую с ней.

Пример стандартного файла `/usr/share/sendmail-cf/cf/redhat.mc` приведен в листинге 14.1.

Листинг 14.1. Стандартный файл redhat.mc

```
divert(-1)
dnl This is the sendmail macro config file. If you make
changes to this file,
dnl you need the sendmail-cf rpm installed and then have
to generate a
dnl new /etc/sendmail.cf by running the following command:
dnl
dnl        m4 /etc/mail/sendmail.mc > /etc/sendmail.cf
dnl
include(`../m4/cf.m4')
VERSIONID(`linux setup for Red Hat Linux')dnl
```

```
OSTYPE(`linux')
define(`confDEF_USER_ID',``8:12'')dnl
undefine(`UUCP_RELAY')dnl
undefine(`BITNET_RELAY')dnl
define(`confAUTO_REBUILD')dnl
define(`confTO_CONNECT', `1m')dnl
define(`confTRY_NULL_MX_LIST',true)dnl
define(`confDONT_PROBE_INTERFACES',true)dnl
define(`PROCMAIL_MAILER_PATH',`/usr/bin/procmail')dnl
define(`ALIAS_FILE', `/etc/aliases')dnl
dnl define(`STATUS_FILE', `/etc/mail/statistics')dnl
define(`UUCP_MAILER_MAX', `2000000')dnl
define(`confUSERDB_SPEC', `/etc/mail/userdb.db')dnl
define(`confPRIVACY_FLAGS', `authwarnings,novrfy,noexpn,re
strictqrun')dnl
define(`confAUTH_OPTIONS', `A')dnl
dnl TRUST_AUTH_MECH(`DIGEST-MD5 CRAM-MD5 LOGIN PLAIN')dnl
dnl define(`confAUTH_MECHANISMS', `DIGEST-MD5 CRAM-MD5
LOGIN PLAIN')dnl
dnl define(`confTO_QUEUEWARN', `4h')dnl
dnl define(`confTO_QUEUERETURN', `5d')dnl
dnl define(`confQUEUE_LA', `12')dnl
dnl define(`confREFUSE_LA', `18')dnl
dnl FEATURE(delay_checks)dnl
FEATURE(`no_default_msa',`dnl')dnl
FEATURE(`smrsh',`/usr/sbin/smrsh')dnl
FEATURE(`mailertable',`hash -o /etc/mail/mailertable.
db')dnl
FEATURE(`virtusertable',`hash -o /etc/mail/virtusertable.
db')dnl
FEATURE(redirect)dnl
FEATURE(always_add_domain)dnl
FEATURE(use_cw_file)dnl
FEATURE(use_ct_file)dnl
FEATURE(local_procmail,`',`procmail -t -Y -a $h -d $u')dnl
FEATURE(`access_db',`hash -o /etc/mail/access.db')dnl
FEATURE(`blacklist_recipients')dnl
EXPOSED_USER(`root')dnl
dnl This changes sendmail to only listen on the loopback
device 127.0.0.1
dnl and not on any other network devices. Comment this out
if you want
dnl to accept email over the network.
DAEMON_OPTIONS(`Port=smtp,Addr=127.0.0.1, Name=MTA')
dnl NOTE: binding both IPv4 and IPv6 daemon to the same
port requires
```

```
dnl        a kernel patch
dnl DAEMON_OPTIONS(`port=smtp,Addr=::1, Name=MTA-v6,
Family=inet6')
dnl We strongly recommend to comment this one out if you
want to protect
dnl yourself from spam. However, the laptop and users on
computers that do
dnl not have 24x7 DNS do need this.
FEATURE(`accept_unresolvable_domains')dnl
dnl FEATURE(`relay_based_on_MX')dnl
MAILER(smtp)dnl
MAILER(procmail)dnl
Cwlocalhost.localdomain
```

С помощью директивы FEATURE можно подключить ту или иную функцию программы **sendmail**. Например, функция **mailertable** предназначена для переопределения маршрутизации для конкретных доменов. Вы можете легко расширить функциональные возможности программы **sendmail**, добавив нужные вам функции в mc-файл.

Предположим, вы хотите, чтобы названия компьютеров домена были скрыты. Это легко достигается с помощью добавления функции **masquerade_envelope** в ваш mc-файл. Для этого скопируйте файл `redhat.mc` в файл `hide_hosts.mc` и добавьте в конец файла `hide_hosts.mc` строки:

MASQUERADE_AS(my-domain.ru)dnl

FEATURE(masquerade_envelope)dnl

Затем выполните команду:

```
# m4 /usr/share/sendmail-cf/cf/hide_hosts.mc > /etc/sendmail.cf
```

Вот и все! Названия узлов будут скрыты. Описание прочих функций представлено в таблице 14.1.

Функции программы sendmail *Таблица 14.1*

Функция	Описание
access_db	Определяет таблицу доступа. В этой таблице указаны узлы, которым разрешена или запрещена отправка почты через ваш почтовый сервер. Эта функция эффективно используется для борьбы со спамом
accept_unresolvable_domains	Разрешает отправлять почту доменам, которые не могут быть распознаны
bestmx_is_local	Сообщения будут приниматься только в том случае, если запись MX сервера DNS указывает на этот почтовый сервер
blacklist_recipients	«Черный список». Еще одна функция для борьбы со спамом. Для ее работы необходима функция **access_db**

Продолжение табл. 14.1

Функция	Описание
dnsbl	Используется для работы с «черным списком». **dnsbl** — это сокращение от DNS Black List. В более ранних версиях эта опция называлась **rbl** (Resolve Black List)
domaintable	Используется для разрешения имен доменов
genericstable	Используется для изменения адреса отправки в сообщениях
local_procmail	Указывает, что доставлять почту нужно с помощью локальной утилиты **procmail**
mailertable	Переопределяет маршрутизацию для конкретных доменов
masquerade_entire_domain	Используется для маскировки (сокрытия) всего домена. Данная функция должна использоваться вместе с директивой MASQUERADE_AS (или MASQUERADE_DOMAIN), например, MASQUERADE_AS(f117.ru)dnl
masquerade_envelope	Позволяет скрыть имена узлов домена. Заменяет поле **received from** заголовка сообщения перед передачей сообщения другим MTA
redirect	Используется для перенаправления на другой почтовый сервер. Означает отказ от принятия почты с выдачей сообщения **please try <address>** (попытайтесь использовать этот адрес)
relay_based_on_MX	Разрешает перенаправление (ретрансляцию) почты только для узлов, которые указаны в записях MX сервера DNS
relay_hosts_only	Разрешает ретрансляцию только для узлов, указанных в **access_db**
relay_mail_from	Разрешает ретрансляцию, только если отправитель указан в списке RELAY базы **access_db**
smrsh	Использование ограниченной оболочки **sendmail**
use_cf_file	При указании этой функции **sendmail** будет обращаться к файлу `sendmail.cf` за списком доверенных пользователей
use_cw_file	При указании этой функции **sendmail** будет обращаться к файлу `sendmail.cw` за списком локальных узлов
virtusertable	Преобразует адрес получателя в адрес локального пользователя

В файле `/etc/mail/sendmail.cw` перечислены все псевдонимы вашего почтового сервера. Предположим, что имя вашего сервера mail.dhsilabs.ru. Если отправитель отправит почту по адресу den@mail.dhsilabs.ru, письмо будет без проблем доставлено пользователю den. А если кто-то отправит письмо по адресу den@dhsilabs.ru, то его доставка вызовет определенные трудности, так как не ясно, какому узлу домена dhsilabs адресовано сообщение. Для решения этой проблемы в файл sendmail.cw нужно поместить строку:

```
dhsilabs.ru
```

Теперь, когда будет приходить почта формата user@dhsilabs.ru, она будет доставлена почтовому серверу mail.dhsilabs.ru.

Напомню, что перед изменением файла `sendmail.cf` желательно остановить программу **sendmail**. Это делается с помощью команды:

```
# /etc/init.d/sendmail stop
```

Конечно, удобнее сначала отредактировать файл `sendmail.cf` с помощью **m4**, а потом выполнить команду `/etc/init.d/sendmail restart` для перезапуска программы **sendmail**.

14.2. Аутентификация в sendmail

14.2.1. Установка и настройка SASL

Программа Sendmail 8.10/8.11 поддерживает SMTP AUTH согласно стандарту RFC 2554. Аутентификация базируется на SASL (Simple Authentication and Security Layer). Она позволит вам несколько повысить безопасность вашей сети, но создаст определенные неудобства для пользователей, потому что не все почтовые клиенты (пользовательские агенты) ее поддерживают.

Первым делом убедитесь в том, что ваша сборка **sendmail** поддерживает библиотеку SASL:

```
# sendmail -d0.1 -bv root | grep SASL
```

При отсутствии поддержки SASL от вас потребуется перекомпилировать **sendmail**. Распаковав исходные коды в каталог sendmail-x.xx.xx, создайте файл `sendmail-x.xx.x/devtools/Site/site.config.m4`, в котором необходимо прописать следующие строки:

```
APPENDDEF(`confENVDEF', `-DSASL')
APPENDDEF(`conf_sendmail_LIBS', `-lsasl')
APPENDDEF(`confLIBDIRS', `-L/usr/lib/')
APPENDDEF(`confINCDIRS', `-I/usr/include/')
```

После этого запустите сценарий Build:

```
# ./Build
# ./Build install
```

Если вы все сделали правильно, ваш **sendmail** теперь должен поддерживать SMTP AUTH. Теперь вам потребуются библиотеки Cyrus SASL, исходные коды которых вы можете найти по адресу `ftp://ftp.andrew.cmu.edu/pub/cyrus-mail`.

Соберите библиотеку Cyrus SASL, выполнив следующую последовательность действий:

```
# tar -xzf cyrus-sasl-1.5.24.tar.gz
# cd cyrus-sasl-1.5.24/
# ./configure --prefix=/usr
```

```
# make
# make install
```

После установки библиотеки отредактируйте файл /usr/lib/sasl/Sendmail.conf. Если он не существует, создайте его. В конец этого файла необходимо добавить строку:

```
pwcheck_method: sasldb
```

Это укажет **sendmail**, что аутентификацию нужно проводить с использованием SASL. Теперь займитесь созданием базы данных всех пользователей, которые могут отправлять почту. Для этого используются две программы: **saslpasswd** и **sasldblistusers**. Они должны находиться в каталоге /sbin. Запускать их нужно от имени суперпользователя.

```
# saslpasswd -a sendmail newuser
password:<пароль для newuser>
```

Эту процедуру нужно провести для всех пользователей, которым разрешено отправление почты. Утилита **sasldblistusers** предназначена для просмотра всех записей в базе данных. После ее запуска вы должны увидеть что-то наподобие этого:

```
user: newuser realm: dhsilabs.com mech: CRAM-MD5
user: newuser realm: dhsilabs.com mech: DIGEST-MD5
user: newuser realm: dhsilabs.com mech: PLAIN
```

Отображенная информация означает, что пользователь newuser может аутентифицироваться тремя методами: CRAM-MD5, DIGEST-MD5, PLAIN. Рекомендую использовать метод CRAM-MD5, но в крайнем случае подойдет и PLAIN.

14.2.2. Настройка sendmail+SASL

В файл sendmail.mc внесите следующие строки:

```
TRUST_AUTH_MECH(`GSSAPI DIGEST-MD5 CRAM-MD5 PLAIN')dnl
define(`confAUTH_MECHANISMS', `GSSAPI DIGEST-MD5 CRAM-MD5
PLAIN')dnl
define(`confDEF_AUTH_INFO', `/etc/mail/auth/auth-info')dnl
FEATURE(`no_default_msa')dnl turn off default entry for MSA
DAEMON_OPTIONS(`Port=25, Name=MSA, M=E')dnl
```

Метод PLAIN, как самый ненадежный, можно было бы убрать из списка авторизации, но я рекомендую вам его оставить для совместимости с некоторыми почтовыми клиентами.

Запустите интерпретатор **m4**:

```
# m4 sendmail.mc > sendmail.cf
```

Скопируйте новый файл sendmail.cf на место старого, перезапустите **sendmail** и проверьте его работоспособность. Для этого запустите клиент **telnet** и присоединитесь к порту 25 вашего компьютера:

```
telnet localhost 25
Trying 127.0.0.1...
Connected to localhost
Escape character is '^]'.
220 local.sendmail.ORG ESMTP Sendmail 8.10.0/8.10.0; Thu,
9 Sep 1999 10:48:44 -0700 (PDT)
ehlo localhost
250-local.sendmail.ORG Hello localhost [127.0.0.1],
pleased to meet you
250-ENHANCEDSTATUSCODES
250-DSN
250-AUTH DIGEST-MD5 CRAM-MD5 PLAIN
250 HELP
quit
```

Теперь желательно добавить описания поддерживаемых вашим сервером методов аутентификации. Это делается для того, чтобы в заголовке письма появилось такое сообщение:

```
(auth_type is CRAM-MD5, user den)
```

Откройте файл sendmail.cf в любом текстовом редакторе и найдите следующие строки:

```
##########################
# Format of headers #
##########################
```

После них вам нужно добавить следующее:

```
$.$?{auth_type}(auth_type is ${тип}, user
${пользователь}$.)
```

14.2.3. Настройка почтовых клиентов с использованием аутентификации

Я рассмотрю настройку трех самых популярных почтовых клиентов:

1. **TheBat!.** Создайте учетную запись (**Account → New**). В качестве имени и пароля введите регистрационные данные пользователя, установленные на сервере с помощью команды passwd. Сервером входящей и исходящей почты назначьте только что созданный почтовый сервер den.dhsilabs.com. Нажмите кнопку **More** (рис. 14.4). В окне **Advanced SMTP Options** установите режим **Perform SMTP authentication**. Если

имя пользователя и пароль на сервере POP совпадают с именем пользователя и паролем на сервере SMTP, а это обычно так, установите режим **Use POP server login**. В противном случае укажите нужное имя пользователя и пароль.

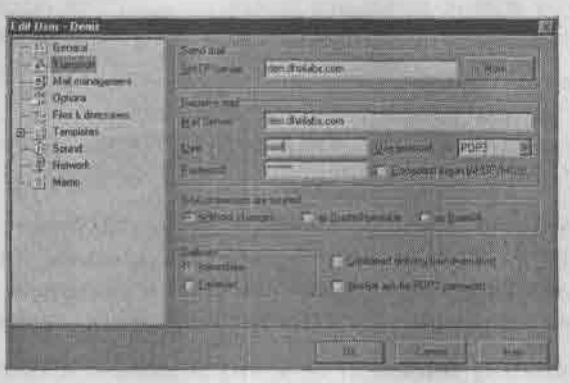


Рис. 14.4. *Настройка TheBat!*

2. **Outlook Express.** Создайте учетную запись (**Сервис → Учетные записи,** кнопка **Добавить**). В окне свойств учетной записи перейдите на вкладку **Серверы.** Включите режим **Проверка подлинности пользователя**, нажмите кнопку **Настройка** и установите параметры аутентификации.
3. **Netscape Messenger.** Выберите пункт меню **Edit → Preferences**. В окне **Preferences** (рис. 14.5) перейдите в раздел **Mail servers** и укажите необходимые вам параметры. Параметры протокола POP можно задать, выбрав почтовый сервер и нажав на кнопку **Edit**. **Netscape Messenger** версии 4.76 поддерживает только метод авторизации PLAIN.

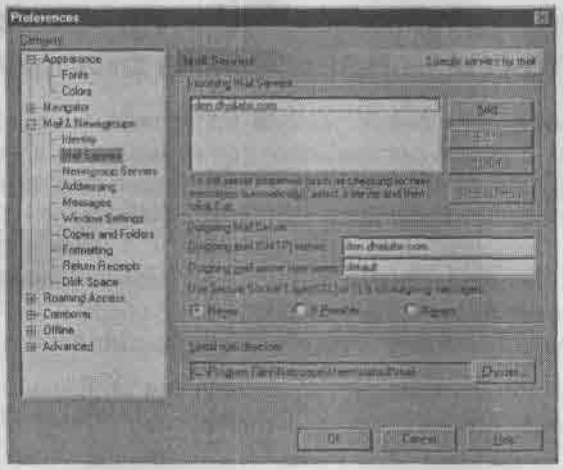


Рис. 14.5. *Настройка Netscape Messenger*

14.3. Агент доступа — fetchmail

Программа **fetchmail** используется для загрузки сообщений с сервера входящей почты в почтовый ящик пользователя. В домашнем каталоге того пользователя, от имени которого будет запускаться **fetchmail**, создайте файл `.fetchmailrc`. Добавьте в него следующие строки:

```
set postmaster "mail"
poll provider.ru proto POP3 no dns
   user "mail" pass "my_password" to mail here
options fetchall
```

Provider.ru — это имя почтового сервера, откуда вы будете забирать почту по протоколу POP3. При этом вы будете использовать имя пользователя mail и пароль my_password. Директива fetchall указывает программе получить всю почту и потом удалить полученные сообщения с сервера.

Запускать программу **fetchmail** можно как демон, а можно с помощью планировщика **cron**. В первом случае просто выполните команду:

```
$ fetchmail -d 12000
```

При этом **fetchmail** будет проверять наличие новой почты через каждые 20 минут.

Во втором случае отредактируйте пользовательский crontab-файл и введите новое задание:

```
$ crontab -e
0,20,40 * * * * /usr/bin/fetchmail
```

14.4. Автоматическая сортировка входящей почты — программа procmail

Лучше всего объяснять работу любой программы на практическом примере. Допустим, в вашей организации три отдела, и их адреса dep1@firma.ru, dep2@firma.ru и dep3@firma.ru. У вас также есть пользователь mail, на имя которого будет приходить вся почта. Вам нужно сортировать приходящую почту по названию отдела: например, если в теле письма или в одном из его заголовков упоминается название отдела, отправить это сообщение одному из пользователей отдела. Кроме того, вы хотите, чтобы рассылка проекта LinuxRSP отправлялась вам по адресу adm@firma.ru.

Создайте в домашнем каталоге пользователя mail файл `.procmailrc`, примерное содержание которого показано в листинге 14.2.

Листинг 14.2. Файл .procmailrc — правила сортировки почты

```
:0
* ^Subject:.*dep1
! dep1

:0
* ^Subject:.*dep2
! dep2

:0
* ^Subject:.*dep3
! dep3

:0
* ^Subject:.*LinuxRSP
! adm
```

Правила означают следующее: если в теме (заголовок Subject) присутствует название отдела, то сообщение будет отправлено нужному пользователю, который должен быть зарегистрирован в системе. Вместо имени пользователя можно указать адрес электронной почты.

Можно выполнить сортировку по любому другому полю. Например, последнее правило могло бы выглядеть так:

```
:0
* ^From:.* Subscribe.Ru
! adm
```

В этом случае, если кто-нибудь из пользователей вашей системы также подпишется на другую рассылку на сервере Subscribe.Ru, то вся почта попадет к пользователю adm.

А теперь создадим почтовый автоответчик. Существуют два типа автоответчиков. Первые посылают автоответ только на определенные сообщения (например, отправляют клиенту прайс-лист вашей организации по его требованию), а вторые — на все (например, сообщают, что вы сейчас заняты, прочитаете письмо тогда-то). Автоответчик первого типа настраивается при помощи правила

```
0:
* ^Subject.*Price
| (formail -r ; cat $HOME/pricelist.zip) | sendmail -t
```

А второй тип создается еще проще. Вы не определяете никаких условий, поэтому файл `info.txt`, содержащий ваш автоответ, будет послан любому отправителю:

```
0:
| (formail -r; cat $HOME/info.txt) | sendmail -t
```

Владельцем файла .procmailrc должен быть пользователь mail. Права доступа следует установить «600».

Вызывать агент доставки **procmail** можно с помощью правил программы **sendmail**, но сейчас мы рассмотрим другой способ. В этом же каталоге ($HOME/mail) создайте файл .forward с такими же правами доступа, как у .procmailrc. В файле .forward задаются правила перенаправления почты. Добавьте в него следующую строку:

```
|IFS=' ' && exec /usr/bin/procmail USER= <mail>
```

Используйте **procmail** с большой осторожностью, потому что если вы неправильно укажете условия сортировки, почта будет просто утеряна без возможности восстановления.

Строки файла конфигурации .procmailrc, которые начинаются с символа решетки (#) считаются комментариями.

Строки, начинающиеся с последовательности символов :0 или :0:, определяют правила, на основании которых procmail выполнит действие над сообщением. После символов :0 можно указать опции поиска и исполняемый файл, которому будет передано сообщение. Общий синтаксис такой:

```
:0 [опции] [: программа]
```

Опция H (header) означает, что условие будет применяться к заголовку письма, а опция B — к телу. Опция D указывает программе различать нижний и верхний регистры символов. По умолчанию используется опция H, то есть условие применяется только к заголовку, а верхний и нижний регистры не различаются. Подробнее об опциях вы прочитаете на man-странице программы **procmail**.

Условие задается с помощью регулярных выражений. Каждое условие начинается символом * и записывается в отдельной строке. Регулярные выражения задаются как обычно, а именно:

- Символ ^ указывает на начало строки, а $ — на ее конец.
- Символ . обозначает любой символ, кроме CR (возврат каретки).
- Символы ? и * читаются как «ноль или более раз».
- Символ + — «один или более раз» .
- Символ | обозначает логическую операцию ИЛИ : x|y — x ИЛИ y.
- [a-z] определяет любой символ из диапазона a..z.
- [^a-z] задает любой символ вне диапазона a..z.

После условия указывается одна команда. Если первый символ команды «!», то сообщение будет перенаправлено на все указанные почтовые

адреса, а если «|», то сообщение будет передано исполняемому файлу (программе), который указан после символа |. Вместо исполняемого файла можно указать переменную окружения, в которую будет записан результат.

Переменная окружения MAILDIR устанавливается в файле .procmailrc. Обычно она имеет значение $HOME/Mail.

Кроме переменной окружения MAILDIR, вы можете указать переменные окружения SENDMAIL и FORMAIL, которые содержат полный путь к программам **sendmail** и **formail** (фильтр-преобразователь сообщений в формат mailbox). Переменная окружения LOGFILE содержит имя файла протокола программы **procmail**, а переменная DEFAULT — имя файла, в который будут записываться сообщения, к которому **procmail** не может применить ни одно из правил.

Для иллюстрации я приведу свой файл конфигурации procmail.

Листинг 14.3. Мой файл .procmailrc

```
PATH=$HOME/bin:/usr/bin:/usr/sbin:/bin:/usr/local/bin:.
MAILDIR=/home/den/mail
DEFAULT=$MAILDIR/mbox
LOGFILE=$MAILDIR/from
LOCKFILE=$HOME/.lockmail

:0
* ^Subject.*Privet
privets

:0
* ^Subject.*Job
| (formail -r ; cat /home/den/vakancy.txt) | /usr/sbin/
sendmail -t
```

Если в теме сообщения было найдено слово «Privet», то все сообщения будут сохраняться в файле `/home/den/mail/privets`. Если тема сообщения содержит слово «Job», то по адресу отправителя будет автоматически отправлен файл `vakancy.txt`. Файл `vakancy.txt` должен быть текстовым — это не вложение.

Файл протокола, в который программа **procmail** запишет адрес отправителя, тему и размер сообщения, называется from.

14.5. Создание списка рассылки

Обычно системы рассылки создаются специально предназначенными для этого средствами: например, идеально подходят PHP в связке с MySQL.

Язык программирования PHP предназначен для создания веб-приложений и оснащен всеми необходимыми для этого функциями, а сервер баз данных MySQL обеспечит поддержку базы данных адресов подписчиков и параметры рассылки. Таким образом, если вы хотите создать собственный MailList.Ru, воспользуйтесь готовыми решениями или напишите собственное на PHP или Perl.

Однако иногда бывает полезно создать небольшую рассылку внутри одной организации. Приведенное далее решение не отличается оригинальностью и не претендует на звание лучшей системы рассылки. Это просто пример, из которого вы узнаете также, как использовать стандартную почтовую утилиту Linux — **mail**. Этот пользовательский агент входит в состав практически каждой UNIX-системы.

Допустим, у вас есть три отдела: отдел маркетинга, производственный отдел и администрация. К первому отделу относятся пользователи вашей системы `marina` и `oleg`, ко второму — `igor`, `dmitry`, `olya`, а к третьему — `president`, `director`, `secretar`. Периодически вам нужно отправлять сообщения в один из отделов. Число пользователей небольшое и, возможно, отправить сообщение можно было бы с помощью групп пользователей почтовой программы, которую вы используете. Однако сейчас я покажу, как элегантно это можно сделать средствами Linux.

Создайте файл `.mailrc` в вашем домашнем каталоге и добавьте в него строки псевдонимов (убедившись, что в вашей системе нет пользователей с именами `market`, `proizv` и `office`):

```
alias market marina oleg
alias proizv igor dmitry olya
alias office director secretar
```

В дальнейшем, чтобы отправить сообщение в производственный отдел, просто введите команду:

```
$ mail proizv
```

Программа **mail** попросит вас ввести тему, а затем текст сообщения. Для окончания ввода нажмите Ctrl+D, и **mail** отправит сообщения пользователям.

Если пользователей много, использовать механизм псевдонимов не очень удобно. Гораздо удобнее, чтобы программа **mail** брала список подписчиков из какого-нибудь файла. К сожалению, разработчики не предусмо-

трели такой возможности, однако с помощью небольшого сценария мы можем эту возможность организовать.

Создайте сценарий `smailer` в своем домашнем каталоге (листинг 14.4).

Листинг 14.4. Сценарий smailer

```
#!/bin/bash
DT=`date`
echo $DT >> log
for user in `cat users`
do
echo "Sending message to $user"
mail $user -s Subscribe < msg 2>> log
done
```

Сообщение, которое вы хотите отправить, запишите в файл `msg`, а список подписчиков по одному в строке — в файл `users`.

Программу **mail** можно использовать и для чтения почты. Для этого просто введите команду `mail`. Если в вашем почтовом ящике есть новые сообщения, программа выведет на экран нумерованный список, и вы сможете ввести номер сообщения, которое хотите прочитать. Для удаления сообщения используется команда **d <номер>** или **d <диапазон>**. Выйти из программы можно, введя команду q.

Программа **mail** — это исключительно пользовательский агент: она работает только с вашим локальным ящиком. В другие почтовые клиенты обычно встроен и агент доступа, забирающий сообщения с внешнего сервера, например, pop.mail.ru. Я советую использовать в качестве почтового клиента программу **kmail**, входящую в состав KDE.

Эта программа поддерживает несколько учетных записей электронной почты, в том числе и локальный ящик, отправку сообщений с помощью SMTP и локального транспортного агента, а также сообщения в формате HTML.

14.6. Защита программы sendmail. Программа smrsh

Однажды неизвестный хакер может заставить ваш **sendmail** выполнить какую-нибудь программу. Какая это будет программа — зависит от его фантазии: безобидная команда echo или «rm -rf /». О том, как это сделать, я писать не буду — лучше разберемся, как этого не допустить.

По умолчанию для запуска внешних программ используется оболочка /bin/sh (или /bin/bash), которая позволяет запускать любые программы без ограничения. Вместо оболочки /bin/sh рекомендуется использовать защищенную оболочку **smrsh**, которая позволяет определить, какие программы можно запускать, а какие — нет.

Определим список программ, которые можно запускать. В него войдут пользовательские агенты (**mail**), агент доставки **procmail** и, может быть, что-нибудь еще. Категорически нельзя разрешать запускать командные оболочки (**sh**, **bash**, **tcsh** и другие), потоковый редактор **sed**, интерпретаторы **perl**, **php** и программу **uuencode**.

Создайте в каталоге /etc/smrsh ссылки на программы, которые можно запускать:

```
$ cd /etc/smrsh
$ ln -s /bin/mail mail
$ ln -s /usr/bin/procmail procmail
```

Откройте файл sendmail.cf и замените строки:

```
Mprog,  P=/bin/sh, F=lsDFMoqeu9, S=EnvFromL/HdrFromL,
R=EnvToL/HdrToL, D=$z:/,
    T=X-Unix/X-Unix/X-Unix,
    A=sh -c $u
```

строками:

```
Mprog,     P=/usr/sbin/smrsh, F=lsDFMoqeu9, S=EnvFromL/Hdr-
FromL, R=EnvToL/HdrToL, D=$z:/,
    T=X-Unix/X-Unix/X-Unix,
    A=smrsh -c $u
```

После этого перезапустите программу **sendmail:**

```
# service sendmail restart
```

Глава 15

НАСТРОЙКА СЕРВЕРА FTP

- СЕРВЕР WU-FTPD
- СЕРВЕР PROFTP
- УТИЛИТЫ ОБСЛУЖИВАНИЯ FTP-СЕРВЕРА
- ВИРТУАЛЬНЫЙ УЗЕЛ FTP
- ЗАЩИТА FTP

FTP (File Transfer Protocol) — один из старейших протоколов Интернета — используется для обмена файлами между системами. Обычно на FTP-сервере размещают свободно распространяемое программное обеспечение, документацию, обновления программ, драйверы и прочую публичную информацию. Примером FTP-сервера может послужить сервер `ftp://ftp.redhat.com`. На этом сервере вы можете найти как саму операционную систему Linux Red Hat, так и обновления ее пакетов, а также дополнительные программы.

Доступ к серверу FTP осуществляется с помощью FTP-клиента. В любой сетевой операционной системе есть простейший FTP-клиент — программа **ftp**. Обычно для того, что бы начать работу с FTP-сервером, вы должны зарегистрироваться на нем, то есть ввести имя пользователя и пароль. После регистрации вы получаете доступ к своему каталогу. Существуют также общедоступные (анонимные) серверы, к которым имеют доступ все пользователи. Для регистрации на таких серверах обычно нужно использовать имя пользователя anonymous, а в качестве пароля — адрес электронной почты.

Над файлами и каталогами вы можете производить обычные операции: создание, удаление, копирование, перемещение, переименование. Как правило, при выполнении операции копирования вы либо копируете файлы на сервер (команда **put**) — загружаете на сервер, либо копируете файлы с сервера на свою локальную машину (команда **get**) — скачиваете с сервера. Работа с FTP-клиентом рассмотрена в п.6.4.5.3, а в этой главе я покажу, как создать FTP-сервер.

15.1. Сервер WU-FTPD

Сервер FTP **wu-ftpd**, разработанный в Вашингтонском университете, очень широко распространен. Он устанавливается из пакета **wu-ftpd**, входящего в состав практически каждого дистрибутива.

Демон **in.ftpd** может быть либо постоянно загружен в память (режим **standalone**), либо вызываться суперсервером **xinetd** (**inetd**) по мере не-

обходимости. Режим **standalone** и применяется, как правило, если FTP-серверу нужно часто обрабатывать запросы клиентов. Второй режим используется в целях экономии памяти, когда нагрузка на FTP-сервер не очень велика.

Чтобы запускать сервер FTP из-под супердемона, добавьте в файл `inetd.conf` следующую строку:

```
ftp stream tcp  nowait root /usr/sbin/tcpd  in.ftpd -l -a
```

Таким образом, FTP-сервер вызывается не напрямую, а через TCP-wrapper, чем обеспечивается дополнительная безопасность. Если вы используете супердемон **xinetd**, описание FTP-сервера должно выглядеть так (листинг 15.1).

Листинг 15.1. Фрагмент файла xinetd.conf

```
service ftp
{
 socket_type    = stream
 wait      = no
 user      = root
 server      = /usr/etc/in.ftpd
 server_args    = -l
 instances   = 4
 log_on_success   += DURATION USERID
 log_on_failure     += USERID
 access_times     = 2:00-8:59 12:00-23:59
 nice      = 10
}
```

Ключ **-l** регистрирует все сеансы FTP в службе **syslog**. Другие ключи, с которыми можно запускать FTP-сервер, перечислены в таблице 15.1.

Ключи командной строки сервера **wu-ftpd** *Таблица 15.1*

Ключ	Назначение
-d	Записывает отладочную информацию в журнал syslog
-l	Регистрирует все FTP-сеансы в журнале syslog
-L	Регистрирует в журнале syslog все команды, отправленные серверу FTP
-tсекунды	Устанавливает предел времени ожидания для пассивных клиентов (по умолчанию 15 минут). Если за этот промежуток времени от клиента не поступит ни одной команды, то FTP-сеанс с сервером будет разорван
-Tсекунды	Максимально допустимое время сеанса FTP (по умолчанию 2 часа)
-a	Разрешает использование файла конфигурации `ftpaccess`
-A	Запрещает использование файла конфигурации `ftpaccess`. Эта опция установлена по умолчанию
-i	Регистрирует в журнале `xferlog` файлы, полученные сервером FTP
-o	Регистрирует в журнале `xferlog` файлы, переданные сервером во время сеанса

15.1.1. Настройка WU-FTPD. Конфигурационные файлы

Сервер **wu-ftpd** использует пять файлов конфигурации:

- `/etc/ftpaccess` — основной файл конфигурации;
- `/etc/ftphosts` — файл, позволяющий запретить доступ к wu-ftpd с определенных узлов или определенным пользователям;
- `/etc/ftpusers` — этот файл содержит список локальных пользователей (зарегистрированных на сервере), которым запрещается взаимодействовать с wu-ftpd;
- `/etc/ftpservers` — позволяет задавать и использовать различную конфигурацию FTP-сервера для различных узлов;
- `/etc/ftpconversions` — определяет типы файлов архивов, которые будут использоваться при сжатии (архивировании) передаваемых данных. Само сжатие включается и выключается в файле `/etc/ftpaccess`.

Основной файл ftpaccess. Директивы сервера WU-FTPD

В этом файле содержатся директивы, которые управляют правами доступа и регистрацией пользователей, задают параметры TCP/IP-взаимодействия, виды регистрируемых событий, используются для конфигурирования анонимного доступа к FTP-серверу и т.п. Пример файла `/etc/ftpaccess` приведен в листинге 15.2.

Листинг 15.2. Примерный файл ftpaccess

```
class all  real,guest,anonymous     *

email root@localhost

loginfails 3

readme  README*   login
readme  README*   cwd=*

message /welcome.msg  login
message .message  cwd=*

compress  yes  all
tar   yes  all
chmod   no   guest,anonymous
delete       no guest,anonymous
```

```
overwrite     no   guest,anonymous
rename        no   guest,anonymous

log transfers anonymous,real inbound,outbound

shutdown /etc/shutmsg

passwd-check rfc822 warn
```

Директива **class** определяет класс пользователей, которые будут иметь доступ к серверу FTP. В примере 11.3 задан класс **all**, который состоит из следующих типов пользователей: настоящие (**real**), гости (**guest**), анонимные (**anonymous**). Под настоящими пользователями подразумеваются те, которые зарегистрированы на сервере, то есть их учетные записи хранятся в файле `/etc/passwd`.

С помощью директивы **email** можно указать адрес администратора сервера.

Директива **loginfails** задает максимальное количество попыток регистрации. Если это количество превышено, пользователь автоматически будет отключен. Значение по умолчанию для этой директивы равно 5.

Директива **message** определяет файл и событие, когда он должен быть отображен. Например, можно создать несколько файлов, один из которых будет отображаться при регистрации пользователя, а другой — при входе его в определенный каталог.

Директивы **chmod** и **delete** определяют, могут ли пользователи использовать одноименные команды FTP. А директивы **overwrite** или **delete** разрешают или запрещают определенным пользователям перезаписывать или удалять файлы на сервере. В приведенном примере пользователи классов **guest** и **anonymous** не могут выполнять ни одну из упомянутых операций.

Общий список директив сервера **wu-ftpd** перечислен в таблице 15.2.

Директивы сервера **wu-ftpd** *Таблица 15.2*

Директива	Описание
autogroup имя_группы имя_класса [...]	Разрешает доступ анонимным пользователям определенных классов к файлам, которые принадлежат к указанной группе
alias псевдоним каталог	Создает псевдоним для каталога на FTP-сервере. Псевдоним позволяет быстро (указав только псевдоним) перейти в соответствующий ему каталог из любого другого каталога на сервере
anonymous-root каталог [имя_класса]	Указывает каталог, который будет использоваться в качестве корневого для заданного класса пользователей. После успешной регистрации пользователя на FTP-сервере он автоматически попадет в соответствующий его классу каталог. Если имя класса не указано, то данная директива будет задавать корневой каталог для анонимных пользователей, для которых корневой каталог не определен явно

Продолжение табл. 15.2

Директива	Описание
banner /абсолютный/путь/к/файлу	Перед регистрацией клиента ему будет показано сообщение из указанного файла
bit-limit [raw] in | out | total макс_кол_байт [имя_класса]	Устанавливает ограничение на количество пересылаемой информации в байтах для пользователей указанного класса. Если имя класса не указать, то данное ограничение будет применяться ко всем пользователям, для которых ограничение не указано явно. Необязательный параметр **raw** позволяет ограничить весь объем пересылаемой информации (в том числе и служебной), а не только пересылаемых файлов. Значения **in**, **out**, **total** указывают поток данных (на сервер, от сервера или оба одновременно), подлежащий учету
class имя_класса типы_пользователей адреса_узлов	Создает класс пользователей с указанным именем. В качестве типа пользователей используются ключевые слова **anonymous** (анонимные пользователи), **guest** (гостевые пользователи) и **real** (зарегистрированные пользователи). Если указывается несколько типов, то они перечисляются через запятую без пробелов. В поле адреса_узлов указываются адреса узлов, пользователи только с которых будут принадлежать данному классу. Символ звездочка «*» означает все узлы. Адреса узлов могут указываться в виде одного из следующих форматов: • **IP-адрес**. Отдельный IP-адрес. • **IP-адрес:маска_сети**. • **IP-адрес/cidr**. IP-адрес с маской CIDR. • **!nameserved.** Указание этого идентификатора приводит к запрету доступа со всех узлов, имена которых не удается получить от DNS-сервера. • **/имя_файла**. Указывается абсолютное имя текстового файла, в котором содержится список IP-адресов (по одному в строке)
cdpath каталог	Определяет для директивы **cdpath** выражение, с помощью которого задается путь поиска при переходе в указанный каталог
compress yes | no имя_класса	Разрешает или запрещает сжатие данных перед отправкой (команда **compress**) для указанного класса пользователей
defaultserver private	Запрещает анонимный доступ к серверу
deny адреса_узлов /путь/к/файл_сообщения	Запрещает доступ к серверу для узлов с указанными адресами. При этом будет отображено сообщение из файл_сообщения. Адреса узлов могут указываться в виде одного из следующих форматов: • **IP-адрес** • **IP-адрес:маска_сети** • **IP-адрес/cidr** • **!nameserved** • **/имя_файла**
email адрес_почты	Почтовый адрес администратора сервера
file-limit [raw] in | out| total количество_файлов [имя_класса]	Устанавливает ограничение на количество пересылаемых файлов для пользователей указанного класса. Параметр **количество_файлов** как раз и задает максимально допустимое количество файлов. Значение остальных параметров такое же, как и для директивы **bit-limit**
guestgroup имя группы [имя группы....]	Всем пользователям, входящим в группу с указанным именем, будет разрешен гостевой доступ к серверу FTP
limit имя_класса максимум периоды файл_сообщения	Ограничивает число одновременно работающих пользователей, принадлежащих указанному классу, в определенное время суток. Параметр **максимум** задает максимально допустимое количество одновременно работающих пользователей. Параметр **периоды** задает временные интервалы. Клиенту, которому запрещается доступ к FTP-серверу в результате действия данной директивы, будет показано сообщение из файла
loginfails количество	Определяет максимальное число неудачных попыток регистрации пользователя, после которых он будет отключен. По умолчанию количество попыток равно 5
log commands типы_пользователей	Регистрирует в журнале команды, которые вводились пользователями указанных типов. В качестве типов пользователей указываются ключевые слова **anonymous, guest, real**

Продолжение табл. 15.2

Директива	Описание
log transfers тип_пользователей список_направлений	Регистрирует в журнале акты передачи файлов пользователями указанных типов. В качестве типов пользователей указываются ключевые слова **anonymous, guest, real.** В поле список_направлений задается направление передачи, подлежащее протоколированию: **inbound** (входящие файлы), **outbound** (исходящие). Если указываются оба направления, то они должны быть разделены запятой без пробела
message файл_сообщения действие	Отображает **файл_сообщения** во время регистрации или при переходе в другой каталог. Соответственно значение в поле **действие** может быть либо LOGIN (регистрация) или CWD=каталог (переход в каталог). Запись cwd=* задает любой каталог
noretrieve [class=имя_класса] список_файлов	Запрещает получение указанных в списке файлов. Если указан параметр **class**, то этот запрет распространяется только на пользователей заданного класса
readme файл действие	Во время регистрации или при смене каталога пользователь получит сообщение о существовании и времени модификации указанного файла. Параметр **действие** определяется так же, как и в директиве **message**
tar yes | no имя_класса	Разрешает или запрещает использование команды **tar** для указанного класса пользователей, то есть разрешает или запрещает архивирование файлов архиватором **tar** перед их пересылкой
virtual адрес	Разрешает использование виртуального FTP-узла

Кроме общих директив, сервер **wu-ftpd** имеет директивы, которые управляют правами доступа. Директивы прав доступа определяют, какие операции могут выполнять пользователи того или иного типа. Эти директивы перечислены в таблице 15.3.

Директивы прав доступа *Таблица 15.3*

Директива	Назначение
chmod yes | no типы_пользователей	Разрешает или запрещает выполнять команду **chmod** для пользователей указанных типов. В качестве типов пользователей указываются ключевые слова **anonymous, guest, real**
delete yes | no типы_пользователей	Разрешает или запрещает выполнять команду **delete** для пользователей указанных типов
overwrite yes | no типы_пользователей	Разрешает или запрещает пользователям указанных типов перезаписывать файлы на сервере
rename yes | no типы_пользователей	Разрешает или запрещает пользователям указанных типов переименовывать файлы на сервере
password-check rfc822|trivial|none enforce|warn	Задает уровень проверки пароля. При этом в качестве первого параметра указывается метод проверки пароля: • **none** — отключает проверку паролей; • **trivial** — все пароли должны обязательно содержать символ '@'; • **rfs822** — в качестве паролей должны указываться адреса электронной почты, задаваемые согласно стандарту RFC822 (рекомендую использовать именно это значение). Вторым параметром задается действие, которое должно производиться в тех случаях, когда пользователь введет неправильный пароль. Значение **warn** говорит о том, что пользователь просто будет проинформирован об ошибке в пароле и далее ему будет позволено заново зарегистрироваться на FTP-сервере. Если указать значение **enforce**, то пользователю будет выдано сообщение о неправильном пароле и ему в дальнейшем будет запрещен доступ к серверу
upload yes | no типы_пользователей	Разрешает или запрещает закачку файлов на сервер пользователям указанных типов

Файл ftphosts — параметры доступа для пользователей с указанных узлов

Файл `ftphosts` используется для разрешения или запрещения доступа определенных пользователей с указанных узлов. Например, вы можете разрешить доступ пользователю admin только с компьютера `admin.domain.ru` и запретить со всех остальных. А для других пользователей разрешить доступ со всех компьютеров. Таким образом, в файле могут быть записи двух видов: разрешающие и запрещающие. Формат записей в файле `ftphosts` следующий:

```
allow | deny  пользователь  узел [узел...]
```

Разрешающая запись **allow** разрешает пользователю регистрироваться с узлов, указанных в списке, но запрещает регистрацию со всех остальных. Запись **deny**, наоборот, запрещает доступ с определенных узлов, но разрешает со всех остальных. В листинге 15.2 приведен пример файла `ftphosts`.

Листинг 15.2. Пример файла ftphosts

```
allow admin 192.168.1.1
deny user 192.168.1.2 192.168.1.3
```

Файл ftpusers — список локальных пользователей, которым запрещено пользоваться WU-FTPD

Файл `/etc/ftpusers` содержит список локальных пользователей, которым запрещается обращение к серверу **wu-ftpd**. Эти пользователи не могут зарегистрироваться на сервере. При попытке регистрации будет выведено сообщение об ошибке **Login Incorrect**, даже если пользователь ввел правильный пароль.

Из соображений безопасности этот файл должен содержать хотя бы имена пользователей root, bin, nobody, daemon, news, uucp. Пустые строки, а также строки, начинающиеся с символа #, игнорируются. Полностью корректный с точки зрения безопасности файл представлен в листинге 15.3.

Листинг 15.3. Файл ftpusers

```
root
bin

adm
lp
sync
shutdown
```

```
halt
mail
news
uucp
operator
games
nobody
```

Файл ftpservers — разные настройки ftp-сервера для различных узлов

По умолчанию настройки **wu-ftpd** применяются ко всем клиентам, подключающимся к нему. Файл `/etc/ftpservers` позволяет задать режим, в котором для определенных узлов будут применяться свои индивидуальные настройки.

Файл `/etc/ftpservers` состоит из записей следующего вида:

```
IP-адрес(или имя узла)   Каталог
```

Если какой-либо пользователь подключится к ftp-серверу с одного из указанных в файле узлов, то для него будут применяться конфигурационные файлы из соответствующего каталога. Например, если в файле присутствует запись 192.168.1.2 `/etc/ftpd/user34`, то при обращении клиента 192.168.1.2 для него будут использоваться конфигурационные файлы из каталога `/etc/ftpd/user34`.

Файл ftpconversions — форматы сжатия

В файле `/etc/ftpconversions` задаются форматы сжатия, разрешенные для использования во время сеанса FTP. Обращаю ваше внимание на то, что само сжатие передаваемых данных включается и выключается соответствующей директивой в файле `ftpaccess`. Стандартный файл `ftpconversions` представлен в листинге 15.4.

Листинг 15.4 Файл ftpconversions

```
:.Z:  :  :/bin/compress -d -c %s:T_REG|T_ASCII:O_UNCOMPRESS:UNCOMPRESS
:   : :.Z:/bin/compress -c %s:T_REG:O_COMPRESS:COMPRESS
:.gz: :  :/bin/gzip -cd %s:T_REG|T_ASCII:O_UNCOMPRESS:GUNZIP
:   : :.gz:/bin/gzip -9 -c %s:T_REG:O_COMPRESS:GZIP
:   : :.tar:/bin/tar -c -f - %s:T_REG|T_DIR:O_TAR:TAR
: : :.tar.Z:/bin/tar -c -Z -f - %s:T_REG|T_DIR:O_COMPRESS|O_TAR:TAR+COMPRESS
:   : :.tar.gz:/bin/tar -c -z -f - %s:T_REG|T_DIR:O_COMPRESS|O_TAR:TAR+GZIP
```

Каждая запись этого файла состоит из восьми полей. Поля разделяются с помощью символа двоеточия. Эти поля содержат префиксы и постфиксы удаления и добавления, внешнюю команду, тип операции и описание.

Например, сжатый программой **gzip** файл должен иметь имя с суффиксом **gz**. Чтобы к имени файла был добавлен суффикс **gz** , запись в файле `ftpconversions` должна иметь постфикс **gz**.

Файл xferlog — журнал FTP-сервера

В файл `xferlog` записываются все транзакции, которые были произведены в ходе сеансов работы пользователей. С помощью ключей **–o** и **–i** сервера FTP можно выбрать тип транзакций, записываемых в журнал.

Рассмотрим листинг 11.7, в котором представлен фрагмент файла `xferlog`.

Листинг 15.5. Фрагмент файла xferlog

```
Wed Jan  9 11:49:35 2002 1 localhost.localdomain 1490 /
home/den/vmware.html a _ o r den ftp 0 * c
Wed Jan  9 11:50:08 2002 1 localhost.localdomain 281 /
home/den/w.out a _ o r den ftp 0 * c
Wed Jan  9 11:50:15 2002 1 localhost.localdomain 281 /
home/den/w.out a _ i r den ftp 0 * c
Wed Jan  9 11:52:08 2002 1 localhost.localdomain 888 /
home/den/ftphosts.html b _ i r den ftp 0 * c
```

Из первой записи видно, что пользователь den был зарегистрирован с удаленного узла localhost.localdomain. Начало передачи файла `/home/den/vmware.html` произошло в среду, 9 января 2002 года в 11:49. Общее время передачи — одна секунда. Общий объем переданной информации составляет 1490 байт.

Для передачи файла использовался режим ASCII (**a**), не было произведено никаких специальных операций (**_**). Файл `vmware.html` пользователь загружал с сервера, на что указывает направление передачи (**o**). Пользователь `den` является зарегистрированным пользователем системы (**r**). Символ **g** на этом месте означал бы гостевую регистрацию, а символ **a** — анонимную. Название службы, которая производила операцию — **ftp**.

Теперь рассмотрим четвертую запись. Тот же пользователь `den` передал на сервер файл `ftphosts.html`. Направление передачи — на сервер (**i**). Режим передачи — двоичный (**b**). Вторая и третья записи сообщают о загрузке с сервера и на сервер файла `w.out` в текстовом (**a**) режиме.

Остается только отметить, что файл `xferlog` используется обоими серверами FTP — **wu-ftpd** и **ProFTPD**.

15.2. Сервер ProFTP

15.2.1. Установка и запуск ProFTPD

Альтернативой, и, на мой взгляд, достаточно хорошей, серверу **wu-ftpd** является сервер **ProFTPD**. Он намного проще в конфигурировании, чем сервер **wu-ftpd**, и обладает достаточно гибкими возможностями. Для его установки достаточно установить пакет proftpd. Подобно серверу **wu-ftpd**, **ProFTPD** может запускаться автоматически при запуске системы или вызываться суперсервером при наличии запроса на установку соединения. Сервер ProFTPD может вызываться с ключами, указанными в табл. 15.4.

Параметры сервера ProFTPD *Таблица 15.4*

Ключ	Назначение
-h	Справочная информация
-n	Запускает сервер в автономном режиме. Для этого в файле конфигурации нужно указать режим запуска **standalone**
-d уровень_отладки	Устанавливает уровень отладки сервера (1-5)
-c файл_конфигурации	Задает использование альтернативного файла конфигурации вместо стандартного `/etc/proftpd.conf`
-p 0 | 1	Запрещает (0) или разрешает (1) использование постоянного пароля. Для получения более подробной информации смотрите документацию по серверу
-l	Выводит список всех модулей, откомпилированных для использования сервером ProFTPD
-v	Выводит версию

15.2.2. Настройка ProFTPD. Файл /etc/proftpd.conf

Сервер **ProFTPD** использует всего один файл конфигурации — `/etc/proftpd.conf`.

В листинге 15.7 представлен простейший файл конфигурации сервера ProFTPD.

Листинг 15.7. Пример файла конфигурации /etc/proftpd.conf

```
# Этот файл устанавливает один сервер и одну учетную запись
ServerName "My ProFTPD server"
ServerType   standalone
DefaultServer on
# Используем стандартный порт
Port 21
```

```
Umask 022
MaxInstances 30
# Пользователь и группа, обслуживающие сервер
User nobody
Group nobody
# Параметры корневого каталога. Блочная директива Directory
<Directory /*>
    # Директива, определяющая параметр AllowOverwrite
      AllowOverwrite on
</Directory>
```

Директивы конфигурации делятся на две группы: директивы, определяющие параметры, и блочные директивы. **Блочные директивы** конфигурирования похожи на тэги языка HTML: конечная директива имеет то же имя, что и начальная, но с наклонной чертой в начале. Например, начальная директива <Directory /*>, а конечная — </ Directory > (листинг 15.7).

Действия каждой пары директив распространяются только на блок, расположенный между ними. Директива **<Directory>** определяет свойства какого-нибудь каталога. В вышеприведенном листинге определяются свойства корневого каталога.

В таблице 15.5 представлены все директивы файла конфигурации сервера **ProFTPD**.

Директивы файла конфигурации сервера ProFTPD *Таблица 15.5*

Директива	Описание
AccessGrantMsg сообщение	Ответное сообщение, которое будет отправлено пользователю в случае его регистрации или получения анонимного доступа. Символы **%u** будут заменены на имя пользователя которое он ввел при регистрации
Allow from all | host | network [,host | network[, ...]]	Используется внутри блока **Limit**. Ограничивает доступ к серверу (а именно, разрешает доступ). По умолчанию **allow from all**
AllowAll	Разрешает доступ к блокам **Directory, Anonymous, Limit**
AllowForeignAddress on | off	Разрешает клиенту указывать при установке соединения адрес, который не соответствует ему. По умолчанию **off**. Может использоваться в блоках **VirtualHost, Anonymous, <Global>**
AllowGroup список_групп	Разрешает доступ определенным группам. Используется в блоке **Limit**
AllowUser список_пользователей	Разрешает доступ определенным пользователям. Используется в блоке **Limit**
AnonRequirePassword on | off	Требует пароль при анонимной регистрации. Пароль должен совпадать с паролем того пользователя, который запустил демон. По умолчанию данная опция выключена
<Anonymous directory>	Создает анонимную учетную запись, directory — корневой каталог анонимного сервера
AuthGroupFile путь	Позволяет указать путь к альтернативному файлу group. По умолчанию используется файл /etc/group

Продолжение табл. 15.5

Директива	Описание
AuthUserFile путь	Указывает альтернативный файл passwd
Bind Ip-адрес	Разрешает привязку дополнительного IP-адреса к основному или виртуальному хосту
DefaultRoot каталог	Задает корневой каталог по умолчанию
Deny from all | host | network	Запрещает доступ к серверу. Используется в блоке **Limit**
DenyAll	Запрещает анонимным пользователям доступ к объектам, указанным в блоке **Limit**
DenyUser *список_пользователей*	*Запрещает доступ определенным пользователям*
<Directory> путь	Используется в **VirtualHost, Anonymous** для того, чтобы определить особенные параметры доступа к каталогу и его подкаталогам
DisplayFirstChdir файл_сообщения	Указанный текстовый файл будет выводиться, когда пользователь впервые за время сеанса войдет в данный каталог. Используется в **VirtualHost, Directory, Anonymous**
DisplayLogin файл_сообщения	Этот файл будет отображен, когда пользователь зарегистрируется
<Global>	Используется для задания параметров, которые будут использоваться как основным, так и всеми виртуальными серверами
<Limit command>	Ограничение на выполнение данной FTP-команды, например, LOGIN, WRITE, READ, STOR
MaxClients number | none сообщение	Ограничение на количество клиентов. Приведенное сообщение будет отображено, если пользователю будет отказано в доступе. Блоки **Anonymous, Global**
MaxLoginAttempts	Максимальное количество попыток зарегистрироваться. По умолчанию 3. Блоки **VirtualHost**, **Global**
Order allow, deny | deny, allow	Порядок выполнения директив **Allow** и **Deny** в блоке **Limit**
PersistentPassword on | off	При значении **on** будут использованы системные файлы /etc/passwd и /etc/group, несмотря на то, что командой **chroot** корневой каталог был изменен
RequireValidShell on | off	Разрешает или запрещает регистрацию при использовании оболочек (shells), которые не указаны в файле /etc/shells
ServerAdmin email	Определяет email администратора сервера
ServerType	Определяет режим работы сервера **standalone** (по умолчанию) или **inetd**. В первом случае сервер будет запускаться автоматически из стартовых сценариев системы, во втором — его будет запускать сервер **inetd** при попытке соединения
TimeoutIdle секунды	Время в секундах, в течение которого пользователь имеет право не проявить активности. По умолчанию 60 (1 минута)
Umask маска	Определяет права доступа для созданного файла. Маска — число в восьмеричной системе, определяющее набор прав доступа (см. *главу* 2)
User имя_пользователя	Имя пользователя, присвоенное демону ProFTP
UserAlias псевдоним пользователь	Создает указанный псевдоним для указанного пользователя
<VirtualHost address>	Создает виртуальный сервер

15.2.3. Разграничение доступа к серверу ProFTP

Я считаю необходимым подробно рассмотреть блочную директиву Limit. Эта директива определяет вид и параметры доступа к тому или иному каталогу. Рассмотрим листинг 15.8.

Листинг 15.8. Пример использования директивы Limit

```
<Directory incoming>
   <Limit WRITE>
     AllowAll
   </Limit>
   <Limit READ>
     DenyAll
   </Limit>
 </Directory>
```

Директива **Directory** определяет свойства каталога `incoming`, а директива **Limit** задает вид доступа к этому каталогу. Команда WRITE директивы **Limit** вместе с директивой **AllowAll** разрешает всем пользователям записывать информацию в этот каталог. Команда READ директивы **Limit** задает ограничение на чтение этого каталога. В рассматриваемом случае чтение запрещено для всех пользователей.

Кроме команд WRITE и READ, в директиве **Limit** можно задавать команды STOR и LOGIN (таблица 15.6).

Команды директивы Limit, ограничивающие права доступа *Таблица 15.6*

Команда	Назначение
LOGIN	Ограничивает регистрацию
WRITE	Ограничивает запись
READ	Ограничивает чтение
STOR	Ограничивает прием файлов

В блоке **Limit** можно задавать директивы **Allow, AllowAll, AllowGroup, AllowUser, Deny, DenyAll, DenyUser**. Например, в листинге 15.9 запрещается доступ всем пользователям, кроме den. Пользователь den может регистрироваться со всех компьютеров, кроме компьютера с IP-адресом 111.111.111.111. Также запрещена регистрация из сети 192.168.2.0

Листинг 15.9. Пример блока Limit

```
<Limit LOGIN>
    DenyAll
    AllowUser den
    Deny from 111.111.111.111
    Deny from 192.168.2.
</Limit>
```

Для конфигурирования отдельного каталога может также использоваться файл `.ftpaccess`, который расположен в этом каталоге. В нем содержатся такие же директивы, что и в файле `proftpd.conf`, но файл `.ftpaccess` имеет приоритет перед файлом `proftpd.conf`.

Организация анонимного FTP-сервера

Анонимный FTP-сервер можно построить с помощью **wu-ftpd**, установив пакет anonftp. Этот пакет нельзя использовать вместе с сервером **ProFTPD**. Пакет anonftp поставляется в составе большинства дистрибутивов.

Сейчас рассмотрим, как организовать анонимный FTP-сервер с помощью сервера **ProFTPD**. Для организации анонимного доступа сервер **ProFTPD** имеет директиву **Anonymous**. При этом в блок **Anonymous** нужно поместить директивы, конфигурирующие анонимную службу. В самой же директиве **Anonymous** необходимо указать каталог, который будет использоваться в качестве корневого для анонимной службы. Сервер ProFTPD выполнит для этого каталога команду **chroot**, превращая этот каталог в корневой для удаленного пользователя. Перед тем, как сделать это, сервер ProFTPD прочитает все необходимые ему файлы конфигурации из реального каталога `/etc`.

При анонимной регистрации по умолчанию в качестве имени пользователя указывается anonymous, а вместо пароля — адрес электронной почты пользователя. Вы же можете изменить параметры анонимного доступа, добавив проверку пароля для анонимного пользователя с помощью директивы **AnonRequirePassword**. В следующем примере представлен типичный блок **Anonymous**, подходящий для большинства анонимных серверов (см. листинг 15.10).

Листинг 15.10. Типичный блок Anonymous

```
<Anonymous /var/ftp>
  User ftp
  Group ftp
  UserAlias anonymous ftp
  RequireValidShell off
```

```
  <Directory *>
    <Limit WRITE>
      DenyAll
    </Limit>
    <Limit STOR>
      AllowAll
    </Limit>
  </Directory>
</Anonymous>
```

Директивы **User** и **Group** задают имя пользователя для анонимного доступа. В обоих случаях применяется имя **ftp**. Для имени **ftp** определяется псевдоним anonymous. Вместо пароля нужно указать адрес электронной почты.

Директива **RequireValidShell** отключает проверку командного интерпретатора пользователя. По умолчанию сервер **ProFTPD** ищет список допустимых интерпретаторов в файле `/etc/shells`. Если используемый пользователем интерпретатор не указан в файле `/etc/shells`, то соединение будет разорвано. Директива **RequireValidShell off** отключает такую проверку.

Директива **<Directory *>** определяет свойства для всех каталогов. При этом всем пользователям запрещено записывать файлы на сервер, но разрешено скачивать файлы сервера на свой локальный компьютер.

Желательно также добавить в блок **Anonymous** директиву **MaxClients**, которая указывает максимальное число клиентов. Нужно учитывать нагрузку на сервер и пропускной канал для определения максимального числа анонимных клиентов. Настоящих пользователей сервера FTP по возможности не следует ограничивать, в отличие от анонимных. При медленном канале связи, например, 33 Кбит/сек, установите максимальное количество анонимных клиентов небольшим, например, 5 или даже 3. Конечно, число клиентов также зависит от объема информации, расположенной на сервере. Если размеры файлов невелики (например, это текстовые файлы), то число клиентов можно установить несколько большим (10-15).

15.3. Утилиты обслуживания FTP-сервера

При работе с серверами **wu-ftpd** и **ProFTPD** вы можете использовать программы **ftpshut** (останавливает сервер), **ftpwho** (выводит информацию о пользователях), **ftpcount** (сообщает о количестве установленных соединений). Инструментальные средства обоих серверов имеют похожие опции, но вспомогательные программы для **ProFTPD** выводят больше полезной информации.

Я рекомендую использовать расширенный вывод утилиты **ftpwho**. В этом режиме предоставляется больше информации (листинг 15.11).

Листинг 15.11. Результат работы программы ftpwho

```
# ftpwho -v
Master proftpd process 759:
 1113 2m55s  proftpd: ftp - localhost.localdomain:
 anonymous/den@den.com: IDLE
    (host: localhost.localdomain [127.0.0.1])
    (cwd: /)
 1150 0m20s  proftpd: den - localhost.localdomain: IDLE
    (host: localhost.localdomain [127.0.0.1])
    (cwd: /home/den)
Service class     -   2 users
```

15.4. Виртуальный узел FTP

Виртуальный сервер — это сервер, не существующий физически, но представляющийся пользователю как реальный сервер. ОС Linux может поддерживать несколько IP-адресов, благодаря чему имеется возможность создать виртуальные узлы. Если вы располагаете дополнительными IP-адресами, то они как раз могут использоваться для создания виртуальных узлов. При конфигурировании виртуальных FTP-узлов каждому виртуальному узлу нужно присвоить отдельный IP-адрес.

Виртуальные FTP-узлы нужны, если вы, например, хотите организовать несколько узлов FTP — например, для разных рабочих групп и для анонимных пользователей. Обслуживать сразу несколько FTP-узлов позволяет все тот же демон **proftpd**.

Настройка виртуального FTP-узла очень похожа на настройку виртуального веб-сервера, которая будет рассмотрена в следующей главе. Даже используется одна и та же директива VirtualHost, содержащая IP-адрес или доменное имя, прописанное в службе DNS. IP-адрес должен указывать на узел сети, на котором запущен **ProFTPD** (см. листинг 15.12).

Листинг 15.12. Пример использования директивы VirtualHost

```
<VirtualHost ftp.library.com>
 ServerName  "Online library"
 MaxClients  15
 MaxLoginAttempts   1
```

```
  DeferWelcome  on
  <Limit LOGIN>
    Allow  from 192.168.1
    Deny from all
  </Limit>

  <Limit WRITE>
    AllowUser libadmin
    DenyAll
  </Limit>

  <Anonymous  /var/ftp/library/books>
    User      library
    Group       library
    AnonRequirePassword  on
 </Anonymous>

  <Anonymous /var/ftp/library>
    User     ftp
    Group       ftp
    UserAlias  anonymous ftp
  </Anonymous>
 </VirtualHost>
```

В примере 15.12 также конфигурируются две анонимных учетных записи — library и ftp. Причем учетная запись library требует ввода пароля при регистрации. Пароль должен совпадать с паролем того пользователя, который запустил демон. Доступ к виртуальному серверу разрешен только для подсети 192.168.1.0. Записывать данные на сервер может только пользователь libadmin.

15.5. Защита FTP

Очень полезной, особенно, при организации виртуальных узлов, является конфигурационная директива **DefaultRoot**, позволяющая указать каталог, который представлялся бы пользователям как корневой. Например, значение **DefaultRoot "~"** настраивает сервер так, чтобы корневым каталогом каждого пользователя был его собственный домашний каталог.

А что делать, если нам нужно, чтобы пользователи видели все дерево файловой системы? Например, чтобы каждый мог посмотреть, в каком каталоге находится та или иная программа. В этом случае целесообразно ограничить действия над файлами в корневом каталоге (точнее, во всех каталогах, кроме домашнего каталога пользователя). Делается это так:

```
<Directory /*>
<LIMIT WRITE READ STOR>
    DenyAll
</LIMIT>
</Directory>
```

Пример файла конфигурации с использованием директивы **DefaultRoot** приведен ниже (листинг 15.13):

Листинг 15.13. Пример использования директивы DefaultRoot (/etc/proftpd.conf)

```
ServerName "My ProFTPD server"
ServerType   standalone
DefaultServer on

# Корневым для пользователя будет его домашний каталог
DefaultRoot "~"

# Используем стандартный порт
Port 21
Umask 022
MaxInstances 30
# Пользователь и группа, обслуживающие сервер
User nobody
Group nobody
# Параметры корневого каталога. Блочная директива Directory
<Directory /*>
    # Директива, определяющая параметр AllowOverwrite
     AllowOverwrite on
</Directory>

# Чтобы избежать атаки на отказ, нужно установить
# максимальное число клиентов, а также максимальное
# число неудачных попыток регистрации:
MaxClients 10
MaxLoginAttempts 2

# Иногда для взлома злоумышленник пытается использовать
# в качестве оболочки какую-нибудь программу, например,
# "оболочку" rm -R /.
# Обычно эта брешь в защите закрыта, но все же желательно
# требовать использования только законных оболочек:
RequireValidShell on
```

Глава 16

HTTP-СЕРВЕР APACHE

- УСТАНОВКА, НАСТРОЙКА APACHE. ФАЙЛЫ КОНФИГУРАЦИИ
- ФАЙЛ РОТАЦИИ ЖУРНАЛОВ /ETC/LOGROTATE.D/HTTPD
- СИСТЕМНЫЙ ФАЙЛ КОНФИГУРАЦИИ /ETC/SYSCONFIG/HTTPD
- СЦЕНАРИЙ ЗАПУСКА СЕРВЕРА APACHE /ETC/INIT.D/HTTPD
- ГРАФИЧЕСКИЕ КОНФИГУРАТОРЫ APACHE
- КАТАЛОГИ ПОЛЬЗОВАТЕЛЕЙ
- ВИРТУАЛЬНЫЙ HTTP-СЕРВЕР
- SSL И APACHE
- ПЕРЕКОДИРОВАНИЕ РУССКОЯЗЫЧНЫХ ДОКУМЕНТОВ «НА ЛЕТУ»
- СЕРВЕР KHTTPD — ВЕБ-СЕРВЕР УРОВНЯ ЯДРА

Эта глава посвящена популярному HTTP-серверу **Apache**. Этот сервер возник из веб-сервера NCSA, разработанного в Национальном центре разработок суперкомпьютеров Иллинойского университета. В 1994 году из проекта NCSA ушел главный разработчик, оставив многих последователей самостоятельно разбираться в своем сервере. Со временем начали появляться исправления и дополнения к серверу NCSA — заплатки (патчи). А в апреле 1995 года вышла первая версия сервера **Apache**, основанного на версии 1.3 сервера NCSA, которая просто вобрала в себя все известные на тот момент исправления NCSA. Отсюда появилось и само название **Apache** — «А PatCHy». Позже **Apache** стал самостоятельной разработкой, и сейчас он поддерживается группой программистов-добровольцев Apache Group.

Сервер **Apache** разрабатывался для ОС Linux и UNIX, но со временем были выпущены его версии и для ОС Windows и OS/2.

Хочу также отметить, что, кроме **Apache**, для ОС Linux существуют и другие веб-серверы: **Red Hat Secure Server**, **Apache-SSL**, **Netscape Enterprise Server** и т.д.

16.1. Установка Apache

В зависимости от дистрибутива, пакет, из которого устанавливается веб-сервер **Apache,** может называться **apache** или **httpd**, а пакет с документацией — **apache-docs** или **httpd-manual** соответственно. В первом случае вам понадобится установить еще пакет `apache-common`, содержащий необходимые файлы для запуска сервера.

После установки сервер конфигурируется для запуска в режиме **standalone**, то есть он будет постоянно находиться в памяти. Я не рекомендую изменять этот режим. Для запуска и останова сервера **Apache** вы можете воспользоваться командами:

```
# /etc/rc.d/init.d/httpd start
# /etc/rc.d/init.d/httpd stop
```

После успешной установки сервера отредактируйте файл `/etc/httpd/conf/httpd.conf`. В нем исправьте всего одну директиву — **ServerName**. Значением ее должно быть имя, зарегистрированное в службе DNS вашей сети. После этого запустите сервер. Откройте любой браузер и попробуйте обратиться к серверу с локального компьютера (`netscape http://localhost`), а потом с другого компьютера вашей сети (`netscape http://server.firma.ru`). Если в обоих случаях вы увидите приветствие сервера (рис. 16.1), значит, ваш сервер **Apache** нормально работает и можно приступать к его дальнейшему конфигурированию. Если в первом случае у вас произошла ошибка, значит, искать ее нужно на локальном уровне. При этом если сеть нормально работает, то, скорее всего, вы просто забыли запустить сервер. Появление ошибки во втором случае может быть связано с неправильной установкой директивы **ServerName** (например, назначенное вами имя веб-сервера не прописано в службе DNS).

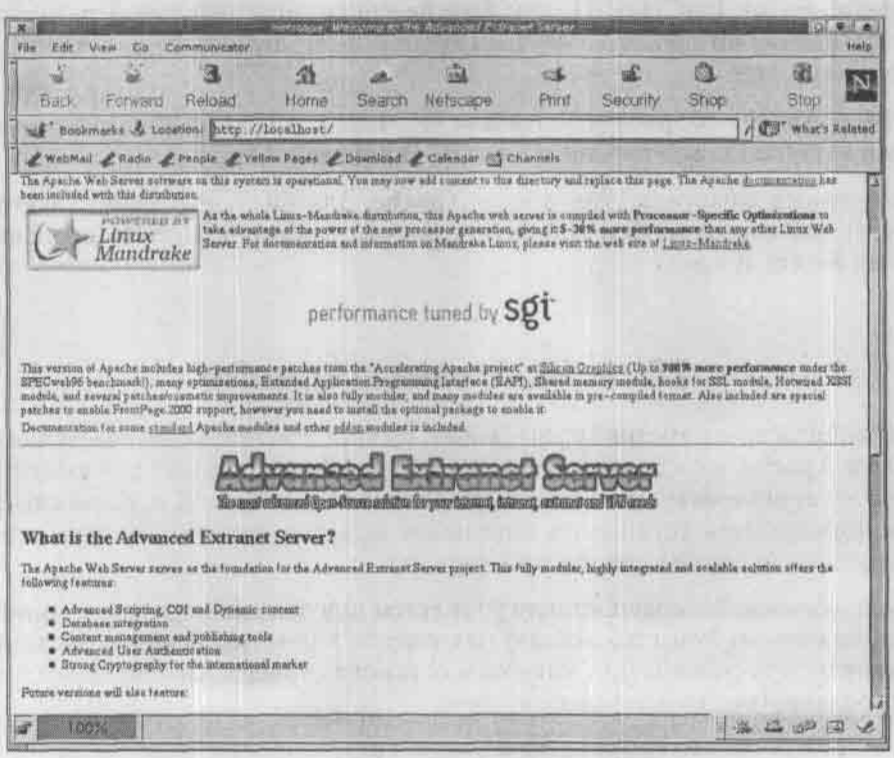


Рис. 16.1. Приветствие сервера Apache

16.2. Настройка Apache. Файлы конфигурации

После установки **Apache** следует отредактировать следующие файлы:

- `/etc/httpd/conf/httpd.conf` — основной файл конфигурации. Для **Apache** 2.x. этот файл может также называться `httpd2.conf`;
- `/etc/logrotate.d/apache` или `/etc/logrotate.d/httpd` (в версии 2.0) — файл ротации журналов;
- `/etc/sysconfig/httpd` — системный файл конфигурации;
- `/etc/init.d/httpd` — файл инициализации **Apache**.

Для старых версий **Apache** (до версии 1.3) характерно наличие файлов `srm.conf` и `access.conf`. Обычно эти файлы находятся в каталоге `/etc/httpd/conf`. В файле `srm.conf` задаются параметры документов, которые размещены на сервере. Файл `access.conf` содержит параметры доступа к серверу. Начиная с версии 1.3 рекомендуется все директивы, которые раньше находились в файлах `srm.conf` и `access.conf`, помещать в файл `httpd.conf`.

Файл `httpd.conf` — это основной файл конфигурации сервера. В нем содержится техническое описание работы сервера. Начиная с версии 1.3 появилось несколько дополнительных конфигурационных файлов: `apache-mime.types`, `vhosts/vhosts.conf`, `vhosts/VirtualHomePages.conf`, `vhosts/DynamicVHosts.conf`. В файле `apache-mime.types` содержатся типы MIME, поддерживаемые сервером **Apache**. Файлы `vhosts.conf`, `VirtualHomePages.conf`, `DynamicVHosts.conf` относятся к конфигурированию виртуальных веб-серверов, о которых речь пойдет немного позже.

Примечание

MIME (Multipurpose Internet Mail Extensions) — многоцелевое расширение электронной почты в сети Интернет. Используется не только при работе с электронной почтой, но и служит для описания различных типов данных, например, текстовых, графических. Описание типа MIME включает в себя наименование типа, подтипа и расширение (например, text/plain txt).

16.3. Основные настройки. Файл httpd.conf (httpd2.conf)

Как уже отмечалось ранее, этот файл содержит практически все директивы, необходимые для работы сервера. Директивы конфигурационного файла сервера **Apache** можно условно разделить на такие группы:

1. **Общие.** К общим директивам относятся глобальные директивы, влияющие на работу всего веб-сервера. Это директивы ServerName, ServerType, Port, User и Group, ServerAdmin, ServerRoot, PidFile, DocumentRoot, UserDir.
2. **Директивы протоколирования**: ErrorLog, TransferLog, HostnameLookups.
3. **Директивы ограничения доступа**: AllowOverride, Options, Limit.
4. **Директивы управления производительностью**: StartServers, MaxSpareServers, MinSpareServers, а также директива CacheNegotiatedDocs.
5. **Директивы обеспечения постоянного соединения с клиентом**: Timeout, KeepAlive, KeepAliveTimeout.
6. **Директивы настройки отображения каталога**. Оформить отображение каталогов можно с помощью директив настройки отображения каталогов: DirectoryIndex, FancyIndexing и AddIconByType.
7. **Директивы обработки ошибок**. Директивой обработки ошибок HTTP-сервера является директива ErrorDocument. С ее помощью можно установить реакцию на любую ошибку сервера, например, на ошибку 404 (документ не найден).
8. **Директивы перенаправления**: Redirect, Alias и ScriptAlias.
9. **Директивы для работы с многоязычными документами**: AddLanguage и LanguagePriority.
10. **Директивы обработки MIME-типов**. Настроить свой сервер для обработки различных MIME-типов можно с помощью директив DefaultType, AddEncoding, AddType, AddHandler и Action.
11. **Директивы создания виртуальных узлов**: VirtualHost, Listen, BindAddress.

Все эти директивы редактировать вам вряд ли придется — по умолчанию они содержат вполне разумные значения. Нужно будет задать только значения директив ServerName и ServerAdmin. Далее приведено описание директив, используемых в файле httpd.conf.

16.3.1. Общие директивы

Общие директивы изменяют глобальные параметры сервера — его имя, тип, порт, адрес администратора. Значения, указанные глобальными директивами, влияют на работу всего сервера.

- **ServerName** — директива, которая определяет имя сервера **Apache**. Здесь должно быть указано официальное имя сервера в таком виде, в котором появится в адресной строке браузера. Это имя должно быть зарегистрировано в службе DNS вашей сети.
- **ServerType** — директива, которая определяет тип сервера. По умолчанию используется значение standalone. Если вы хотите достичь

максимальной производительности вашего веб-сервера, не изменяйте это значение.

- **ServerAdmin** — директива, которая задает электронный адрес веб-мастера вашего веб-узла. Если возникнут какие-то проблемы, связанные с работой сервера, то по этому адресу будет отправлено соответствующее сообщение. Обычно используется значение webmaster@Your_Host.com. Пользователь webmaster, как правило, не существует реально в системе. Для определения имени (псевдонима) webmaster используется файл псевдонимов электронной почты `/etc/aliases`. Формат этого файла следующий: <псевдоним>:<регистрационное_имя>. После модификации этого файла нужно ввести от имени суперпользователя команду **newaliases**. В Windows псевдонимы создаются с помощью программы настройки установленного почтового сервера.
- **Port** — директива, задающая номер порта, который будет использоваться для установки соединения. По умолчанию используется порт 80. Если вы хотите запустить сервер **Apache** с использованием этого или любого другого порта, номер которого меньше 1024, вы должны обладать правами суперпользователя. Но даже если у вас нет таких прав, вы можете запустить сервер для работы с портом, номер которого превышает значение 1024. Обычно используется номер 8080 или 8000.
- **User** и **Group**. Директивы **User** и **Group** определяют идентификаторы пользователя и группы, от имени которых будет работать сервер. Данные идентификаторы присваиваются серверу, если он запущен в автономном режиме. Можно использовать как имена пользователей, так и их числовые эквиваленты — UID. По умолчанию используется имя пользователя nobody. Из соображений безопасности не рекомендуется изменять это значение и указывать имя реального пользователя. В этом случае веб-сервер получит доступ только к тем файлам, которые разрешены для чтения всем пользователям. Нужно заметить, что указанный пользователь и группа должны существовать в вашей системе. Ни в коем случае не запускайте сервер от имени пользователя root!
- **ServerRoot** — в этой директиве указывается местонахождение файлов конфигурации сервера **Apache**. По умолчанию для этих целей используется каталог `/etc/httpd`.
- **PidFile** — с помощью этой директивы указывается имя файла, в котором исходный процесс сервера будет регистрироваться. Этот файл содержит свой идентификатор процесса (PID). Данную информацию можно использовать для останова или перезапуска сервера при написании собственных сценариев. Этот файл будет создан, только если сервер **Apache** запущен в автономном режиме.
- **DocumentRoot** — директива, определяющая местонахождение корневого каталога документов вашего сервера.

- **UserDir** — эта директива задает названия подкаталога в домашнем каталоге пользователя, из которого берутся документы. В этом случае вы активизируете возможность использования пользовательских каталогов. Если вы не хотите включать эту возможность, укажите UserDir DISABLED. Более подробно эта директива будет рассмотрена позже.

16.3.2. Директивы протоколирования

Директивы протоколирования управляют процессом протоколирования работы сервера. С их помощью вы можете определить, что нужно записывать в журналы, а что — нет.

- **HostnameLookups** on | off.Сервер **Apache** ведет журнал доступа других компьютеров. Если вы включите данную опцию (on), то в журнал будет записано доменное имя компьютера-клиента. Если эта опция выключена (off), в журнал будет записан IP-адрес клиента. Включение данной опции замедляет работу сервера, так как требуется дополнительное время на ожидание ответа от сервера DNS.
- **ErrorLog** и **TransferLog** — эти директивы определяют расположение журналов сервера **Apache**. Обычно для этих целей используется каталог `/etc/httpd/logs`, который является ссылкой на каталог `/var/log/httpd` или на любой другой. В журнале `errorlog` протоколируются диагностические сообщения, а также сообщения об ошибках, которые порождают CGI-сценарии. В журнале `trasferlog` протоколируются запросы клиентов.

16.3.3. Директивы управления производительностью

Поэкспериментировав с этими директивами и выбрав оптимальные значения, можно добиться существенного повышения производительности вашего сервера.

Сервер **Apache** для каждого соединения запускает отдельную копию, которая будет обрабатывать запросы клиента. Управлять запущенными копиями позволяют директивы StartServers, MinSpareServers, MaxSpareServers.

- **StartServers, MaxSpareServers, MinSpareServers**. Для каждого нового соединения создается новая копия процесса сервера. Директива **StartServers** задает количество копий, которые будут созданы при запуске исходной копии сервера. При этом исходная копия сервера получает запросы и передает их свободным копиям. Это позволяет равномерно распределить нагрузку между отдельными процессами и повысить производительность сервера, однако на

практике все не так хорошо, как хотелось бы. Существенного прироста производительности можно добиться только в случае большой загрузки сервера. По умолчанию запускается пять копий сервера. Если число поступающих запросов превышает количество запущенных копий сервера, запускаются дополнительные процессы-серверы. Эти процессы не завершаются после обработки своего запроса, а продолжают находиться в памяти. Директива **MaxSpareServers** позволяет указать максимальное число таких процессов. Если это количество превышено, то лишние процессы завершаются. Если количество «серверов на подхвате» меньше, чем задано директивой **MinSpareServers**, запускаются дополнительные копии. Для работы этих директив необходимо, чтобы сервер был запущен в автономном режиме.

- **CacheNegotiatedDocs** — эта директива позволяет прокси-серверу, например, SQUID, не кэшировать документы, которые не генерируются автоматически, то есть в процессе выполнения различных сценариев. Согласно протоколу HTTP/1.0, сервер **Apache** с каждым пакетом посылает заголовок «Pragma: no-cache» прокси-серверу, что позволяет отключить кэширование документов (в протоколе HTTP/1.1 вместо Pragma используется Cache-Control). Если вы включите данную директиву, то вы разрешите прокси-серверу кэшировать документы. К сожалению, далеко не все прокси-серверы отключают кэширование после получения данного заголовка. При написании своего CGI-сценария вам, скорее всего, придется самому выводить заголовок Pragma (или Cache-Control) и мета-тэги, которые указывают на дату последнего обновления документа.

16.3.4. Директивы обеспечения постоянного соединения с клиентом

Эти директивы обеспечивают постоянное соединение с клиентом, а также управляют параметрами установленного соединения.

- **Timeout** — задает промежуток времени в секундах, в течение которого сервер продолжает попытки возобновления приостановленной передачи данных. Значение директивы **Timeout** распространяется не только на передачу, но и на прием данных. Если вам нужно получать большие файлы, рекомендую увеличить данное значение.
- **KeepAlive** — разрешает постоянные соединения, то есть такие соединения, в которых производится более одного запроса за один раз.
- **KeepAliveTimeOut** — данная директива определяет таймаут для постоянного соединения.
- **MaxClients**. Иногда поступающих запросов настолько много, что компьютеру не хватает ресурсов для загрузки новых копий сервера в

память и их выполнения. Директива **MaxClients** (значение по умолчанию — 150) определяет максимальное число копий сервера, которые могут выполняться одновременно.

- **MaxRequestsPerChild**. После обработки определенного количества запросов, указанного в этой директиве, копия сервера завершается, а вместо нее запускается новая.

16.3.5. Директивы создания виртуальных узлов

Довольно часто пользователь не может позволить себе такую роскошь, как собственный физический веб-сервер, то есть отдельный компьютер, постоянно подключенный к Интернет, на котором запущен сервер **Apache**. Поэтому современные веб-серверы поддерживают виртуальные узлы. У вас будет собственное имя, например, www.firma.ru, но в действительности ваши файлы будут помещены не на отдельном компьютере www.firma.ru, а будут находиться на сервере провайдера (или хостера) — www.isp.ru.

Благодаря технологии виртуальных узлов один сервер хостинг-провайдера обслуживает сотни или даже тысячи сайтов.

На своем домашнем веб-сервере вы также можете создать один или несколько виртуальных узлов, если, конечно, вам это нужно.

Listen — эта директива позволяет вам связывать **Apache** с определенным IP-адресом и/или дополнительными портами.

BindAddress — эта директива сообщает серверу, какой IP-адрес следует прослушивать. Значением данной директивы может быть «*» (любой адрес), IP-адрес или полное имя домена.

Блок директив **VirtualHost** будет рассмотрен позднее, при описании создания и использования виртуальных узлов.

16.3.6. Директивы настройки отображения каталогов

Эти директивы позволяют «украсить» ваш веб-сервер, например, сопоставить значок типу файла или определить оформление каталога.

- **DirectoryIndex** — позволяет задать название документа, который будет возвращен по запросу, не содержащему имени документа. Например, если вы введете в строке адреса броузера http://localhost, то будет возвращен один из указанных в директиве **DirectoryIndex** документов.С помощью данной директивы можно задать несколько имен файлов. Значениями по умолчанию являются index.html index.php index.htm index.shtml index.cgi Default.htm default.htm index.php.

- **FancyIndexing**. При получении запроса, не содержащего имя документа, сервер передаст один из файлов, указанных в директиве **DirectoryIndex**. Если такой файл не существует, клиенту будет возвращено оглавление каталога. При включении директивы **FancyIndexing** в оглавлении каталога будут использованы значки и описания файлов. Если директива **FancyIndexing** выключена, оглавление будет представлено в более простом виде.
- **AddIconByType** (TEXT, URL) mime-type — сопоставляет значок типу файла. Значок будет использоваться при выводе каталога, если включена директива **FancyIndexing**. Параметр TEXT определяет текстовое описание типа, которое увидят пользователи, использующие текстовый браузер или пользователи, у которых отключено отображение рисунков. Параметр URL определяет адрес значка, а параметр mime-type — это тип файла, с которым нужно сопоставить значок. Полный перечень MIME-типов приведен в файле `apache-mime.types`. В качестве имени файла можно задать не только MIME-тип, но и символы, которыми заканчивается имя файла (см. листинг 16.1), но для этого нужно использовать директиву **AddIcon** вместо **AddIconByType**.

Листинг 16.1. Фрагмент файла httpd.conf

```
AddIconByType (VID,/icons/movie.gif) video/*
AddIcon /icons/binary.gif .bin .exe
```

Первая директива в листинге сопоставляет типу video значок /icons/movie.gif. Вторая директива сопоставляет бинарным файлам *.bin и *.exe значок /icons/binary.gif. Значок по умолчанию задается директивой DefaultIcon.

16.3.7. Директивы обработки MIME-типов

Как вы помните, в Windows существует такое понятие, как расширение (или тип) файла. По расширению можно связать какую-либо программу с определенным типом файла. Например, когда вы щелкаете на файле с расширением .txt, запускается Блокнот, а при щелчке на файле .doc будет запущен Word. Точно такой же механизм реализован в **Apache**. Вы можете сопоставить типу файла программу-обработчик этого типа. Когда не был разработан модуль mod_php, программа **php** объявлялась обработчиком файла с расширением .php. Сервер запускал **php**, передавал ему файл, а потом возвращал пользователю результат.

- **DefaultType**. Если запрашиваемый клиентом тип не соответствует ни одному из MIME-типов, используется MIME-тип, указанный в директиве **DefaultType**.

- **AddEncoding**. Для сокращения времени передачи файла клиентам используется сжатие данных. Браузеры имеют встроенные программы для распаковки, запускаемые при получении архивов определенных MIME-типов. Именно эти MIME-типы и указываются в директиве **AddEncoding**.
- **AddType** — с помощью этой директивы можно добавить новый MIME-тип, который не указан в файле `apache-mime.types`.
- **AddHandler** и **Action**. Директива **AddHandler** позволяет сопоставить определенному MIME-типу какой-нибудь обработчик. А с помощью директивы **Action** можно определить какое-нибудь действие для обработчика. Использование этих директив, я думаю, лучше всего продемонстрировать на примере (листинг 16.2).

Листинг 16.2. Применение директив AddHandler и Action

```
AddHandler text/dhtml dhtml
Action text/dhtml /cgi-bin/dhtml-parse
```

16.3.8. Директивы для работы с многоязычными документами

Если ваш сайт имеет несколько языковых версий — например, русскую и английскую, — целесообразно настроить директивы управления языком. Тогда пользователю не нужно будет щелкать по ссылкам «In English/На Русском» — сервер, исходя из настроек браузера, сам определит, какой язык предпочитает пользователь, и отобразит нужную страницу.

- **AddLanguage**. В большинстве браузеров можно задать предпочитаемый язык. Директива **AddLanguage** сопоставляет расширение файла аббревиатуре языка. Для русского языка используется аббревиатура ru, для английского — en. При этом в корневом каталоге вашего сервера могут находиться несколько индексных файлов на разных языках. Например, для русского языка нужно использовать имя файла index.html.ru, а для английского — index.html.en.
- **LanguagePriority**. Если на вашем сервере размещены документы на разных языках, то с помощью директивы **LanguagePriority** можно указать приоритеты различных языков. Клиент вводит в адресной строке http://www.server.com/. Если в свойствах браузера имеется возможность задать предпочитаемый язык, то возвращен будет файл на нужном языке, если такой существует. Если же браузер клиента не поддерживает эту возможность, будет возвращен файл на языке, имеющем наиболее высокий приоритет. Для того, чтобы сервер поддерживал нужный вам язык, предварительно установите правильное значение директивы **AddLanguage**.

16.3.9. Директивы перенаправления

Довольно часто нужно перенаправить пользователя на другой ресурс: например, вы сменили хостера и из-за этого изменилось имя вашего сайта. Обычно при таком «переезде» у вас есть определенное время, чтобы сообщить вашим пользователям об этом. Проще всего установить на старом сервере перенаправление на новый — во-первых, пользователи узнают ваш новый адрес, а во-вторых, им не нужно будет вводить этот адрес вручную — сервер сделает все автоматически.

Возможно, вы просто перенесли файлы в другой каталог — вам так удобнее, но пользователи не знают об этом и по-прежнему обращаются к старому каталогу. Тогда создайте редирект на новый каталог, и сервер автоматически перенаправит пользователей на него.

- **Redirect**. Используйте эту директиву, когда нужно перенести документы в другой каталог или на другой сервер. Например, вам нужно перенести данные из каталога /users/den в каталог /den. Если при этом старый URL-адрес был http://www.host.com/users/den, то новый станет http://www.host.com/den/. Используйте для этого следующую директиву: `Redirect /users/den /den`. Можно также перенаправить запрос на другой сервер: `Redirect /users/den/ www.den.domain.com`. При этом допускается использование как нового, так и старого URL-адреса.
- **Alias** — с помощью директивы **Alias** можно предоставить доступ не только к файлам, находящимся в каталоге, указанном директивой DocumentRoot, и его подкаталогах, но и в других каталогах. По умолчанию определен только псевдоним для каталога /icons.
- **ScriptAlias** — аналогична директиве **Alias**, но позволяет задать месторасположение каталога для CGI-сценариев.

16.3.10. Директивы обработки ошибок

Такая директива всего одна, но она очень полезна. Например, произошла ошибка 404 (файл не найден). Вы можете сопоставить этой ошибке URL, на который будет перенаправлен браузер пользователя. Обычно перенаправление устанавливают на документ, содержащий логотип сайта и сообщение об ошибке.

ErrorDocument — директива, сопоставляющая коды ошибок сервера URL-адресам на этом же сервере.

16.3.11. Директивы управления доступом к отдельным каталогам

Вы можете определить отдельные параметры для каждого каталога вашего сервера — оформление каталога, параметры доступа к этому каталогу.

Блок директив Directory

Блок директив **Directory** определяет свойства каталога (см. листинг 16.3).

Листинг 16.3. Директива Directory

```
<Directory />
    Options Indexes Includes FollowSymLinks
    AllowOverride None
</Directory>
```

Свойства каталога можно указывать в директиве **Directory** или в файле `.htaccess`, который находится в том каталоге, для которого необходимо установить нужные параметры.

В блоке **Directory** могут находиться директивы управления доступом. К ним относятся директивы **AllowOverride**, **Options**, **Limit**. Рассмотрим по порядку все эти директивы. Директива **AllowOverride** может принимать значения, указанные в таблице 16.1.

*Значения директивы **AllowOverride*** *Таблица 16.1*

Значение	Описание
None	Сервер **Apache** будет игнорировать файлы .htaccess. Рекомендую установить данную опцию, так как это повысит производительность сервера
All	Пользователи имеют право переопределять в файлах .htaccess глобальные параметры доступа. Из соображений безопасности лучше не использовать этот режим
Options	Разрешает использовать директиву **Options**
Limit	Разрешает использовать директиву **Limit**
AuthConfig	Разрешает использование директив **AuthName, AuthType, AuthUserFile** и **AuthGroupFile**
FileInfo	Разрешает использовать в файлах .htaccess директивы **AddType** и **AddEncoding**

С помощью директивы **Options** можно определить функции сервера, которые будут доступны для использования в определяемом каталоге. Данную директиву можно использовать как в файле `httpd.conf`, так и в файлах `.htaccess`. Допустимые опции для директивы **Options** представлены в таблице 16.2.

Значения директивы **Options** *Таблица 16.2*

Значение	Описание
None	Не разрешается использование каких-либо функций
All	Разрешаются все функции
FollowSymLinks	Разрешается использовать символические ссылки. С точки зрения безопасности не рекомендуется использовать этот режим
SymLinksIfOwnerMatch	Разрешается использование символических ссылок, если они указывают на объекты, которые принадлежат тому же пользователю, что и сами ссылки
ExecCGI	Разрешается выполнение CGI-сценариев
Indexes	Если эта опция выключена, сервер не будет передавать содержимое каталога при отсутствии файла index.html
Includes	Разрешено использование серверных включений. Рекомендую отключить эту опцию, поскольку это сильно нагружает сервер
IncludesNoExec	Разрешает использование серверных включений, но запрещает запуск из них внешних программ

Директива **Limit** ограничивает доступ к файлам в определенном каталоге:

```
Limit метод
```

Параметр «метод» определяет метод передачи: GET или POST. Директиву **Limit** можно использовать внутри блоков **Directory**, **Location** или в файле `.htacccess`.

Блок директив Limit

В блоке **Limit** можно использовать такие директивы: **allow** (разрешить), **deny** (запретить), **order** (порядок), **require** (требуется). После директивы **allow** следует слово **from**, после которого можно указать IP-адрес, адрес сети, домен или просто имя компьютера. Слово **all** обозначает все компьютеры. Директива **order** определяет порядок выполнения директив **allow** и **deny**. Например, вам требуется запретить доступ всем компьютерам, кроме компьютеров, которые входят в домен ru (см. листинг 16.4).

Листинг 16.4. Директивы allow, deny

```
order deny, allow
deny from all
allow from ru
```

Следующий пример показывает, как разрешить доступ компьютерам только из вашей сети.

Листинг 16.5. Разрешение доступа подсети 192.168.1.0

```
order deny, allow
deny from all
allow from 192.168.1.
```

Кроме значений **allow,deny** и **deny,allow**, директива **order** может содержать значение **mutual-failure**. В этом случае в доступе будет отказано всем компьютерам, которые явно не указаны в списке **allow**.

Директиву require можно использовать для защиты каталога паролем. После названия директивы должен следовать список элементов: имена пользователей, групп, которые заданы в директивах **AuthUserFile** и **AuthGroupFile**. Можно использовать параметр **valid-user**, который укажет серверу предоставить доступ любому пользователю, имя которого имеется в директиве **AuthUserFile**, если он введет правильный пароль. Пример использования приведен в листинге 16.6.

Листинг 16.6. Использование директивы require

```
<Directory *>
    AuthUserFile /var/secure/.htpasswd
    AuthName Security
    AuthType Basic
    <Limit GET>
    order deny,allow
    deny form all
    allow from mydomain.ru
    require valid-user
    </Limit>
</Directory>
```

В листинге 16.6 для аутентификации используется файл паролей `.htpasswd`, который можно создать с помощью программы **htpasswd**. Директивы блока **Limit** разрешают доступ к любому каталогу сервера только пользователям домена `mydomain.ru`.

Кроме параметра **valid-user** допускается использование параметра users или groups. Данные параметры разрешают доступ только определенным пользователям или группам пользователей. Пример использования параметра users приведен в листинге 16.7.

Листинг 16.7. Применения параметра users

```
<Directory /users>
    AuthType Basic
    AuthUserFile /var/users/.htpasswd
    AuthName UsersDir
    <Limit GET POST>
    require users denis igor evg
    </Limit>
</Directory>
```

Блок директив Location

С помощью директив, расположенных в блоке **Location**, можно задать определенный URL-адрес, предназначенный для обозначения каталогов, файлов или групп файлов. Обозначить группу файлов можно с помощью шаблонов, например, шаблон *.html определяет все файлы, имена которых заканчиваются на .html. В URL-адрес не включается протокол и имя сервера. Пример описания блока **Location** представлен в листинге 16.8.

Листинг 16.8. Блок Location

```
<Location URL>
    директивы управления доступом
</Location>
```

16.4. Файл ротации журналов /etc/logrotate.d/httpd

Файл `/etc/logrotate.d/httpd` (или `/etc/logrotate.d/apache` — для версий **Apache** до 2.0) задает параметры ротации журналов веб-сервера, что позволяет поддерживать порядок в журнальном хозяйстве. Пример этого файла приведен в листинге 16.9.

Листинг 16.9. Файл /etc/logrotate.d/httpd (Apache 2.0)

```
/var/log/httpd/*_log {
    missingok
    notifempty
    sharedscripts
    postrotate
    /usr/bin/killall -HUP httpd
    endscript
}
```

Убедитесь, что режим доступа к файлу /etc/logrotate.d/httpd равен 0640 и владельцем этого файла является пользователь root.

16.5. Системный файл конфигурации /etc/sysconfig/httpd

Этот файл позволяет передать серверу **Apache** системную информацию, например, параметры запуска.

Предположим, что вы хотите запустить сервер **Apache** с включенной поддержкой SSL. Для этого в файл `/etc/sysconfig/httpd` добавьте строку:

```
OPTIONS="-DSSL"
```

Вам нужно только добавить нужные параметры в директиву OPTIONS, а обо всем остальном позаботится сценарий запуска `/etc/init.d/httpd`. Нужно отметить, что файл `/etc/sysconfig/httpd` появился в версии **Apache** 2.0.

16.6. Сценарий запуска сервера Apache /etc/init.d/httpd

Стандартный сценарий запуска веб-сервера **Apache** устанавливается из того же пакета, что и сам сервер. В версии **Apache** 2.0 можно вызывать сценарий запуска со следующими параметрами:

- **start** — запуск сервера;
- **stop** — завершение работы сервера;
- **restart** — перезапуск сервера;
- **status** — информация о работе сервера;
- **condrestart** — перезапуск сервера при наличии файла `/var/run/httpd.pid`. Этот файл создается при запуске сервера и удаляется при его останове. Если файл `httpd.pid` не удален, значит, сервер не был остановлен корректно: например, произошел сбой системы или банальное отключение питания.

Что же касается более старых версий **Apache**, то сценарии запуска позволяют указывать куда больше параметров, чем во второй версии:

- **start** — запуск сервера;
- **stop** — завершение работы сервера;
- **restart** — перезапуск сервера;

- **reload** — перезагрузка сервера. В отличие от перезапуска, когда сервер сначала останавливается командой kill (то есть, просто «убивается»), а потом запускается, при перезагрузке серверу передается сигнал HUP. Перезагрузка может понадобиться при изменении файла конфигурации сервера, чтобы изменения вступили в силу;
- **condrestart** — то же, что и одноименный параметр, описанный выше;
- **status** — информация о работе сервера;
- **fullstatus** — более подробная информация о работе сервера;
- **help** — подсказка;
- **configtest** — проверка файла конфигурации.

16.7. Графические конфигураторы Apache

Практически все параметры веб-сервера **Apache** можно установить, используя конфигуратор **netconf** (п.14.1.1). Запустите **netconf** от имени суперпользователя и выберите **Server Tasks**, а затем **Apache Web-server**. С помощью **netconf** вы легко можете определить виртуальные узлы, назначить параметры подкаталогов, определить спецификацию каталогов и модулей, а также установить параметры модуля mod_ssl (см. рис. 16.2), настройка которого рассмотрена далее в этой главе.

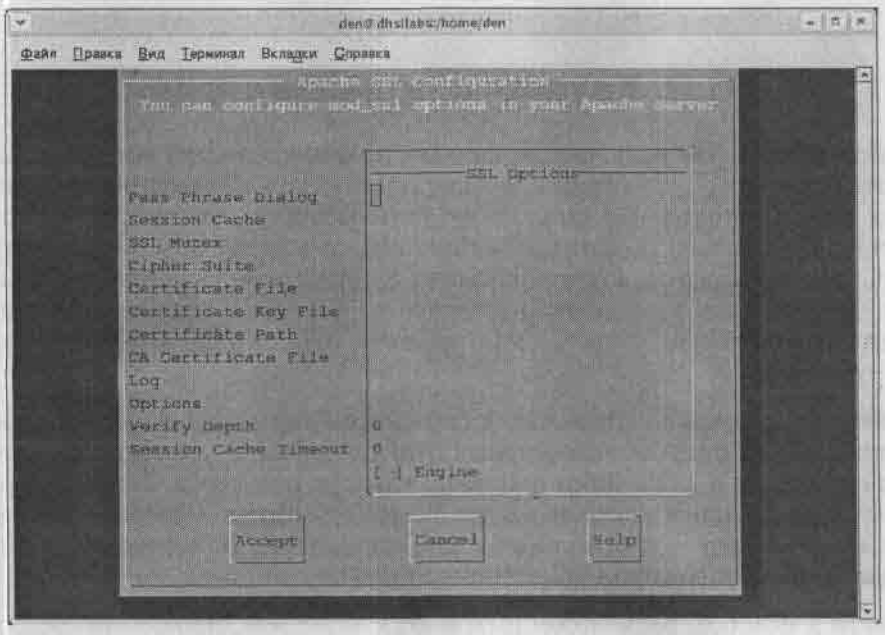


Рис. 16.2. *Конфигурирование модуля mod_ssl*

В дистрибутив Fedora Core включен более удобный конфигуратор **system-config-httpd** (рис. 16.3).

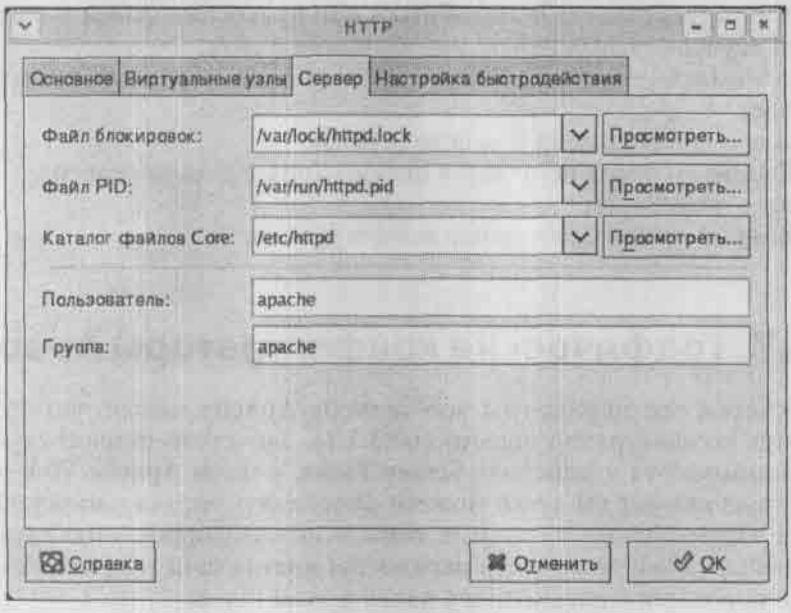


Рис. 16.3. *system-config-httpd*

16.8. Каталоги пользователей

Директива **UserDir** включает поддержку пользовательских каталогов. Эта директива определяет общее название подкаталога в домашних каталогах всех пользователей. По умолчанию используется каталог public_html. Данная возможность очень удобна при использовании ее в большой корпорации, где каждый сотрудник имеет собственную страничку. Раньше эта возможность часто использовалась на серверах, предоставляющих бесплатный хостинг. Может быть, помните адреса вида http://www.chat.ru/~mypage?

Сейчас же все чаще используется технология виртуальных серверов, которую мы рассмотрим в следующем пункте, но знать, что такое каталоги пользователей и как с ними работать, тоже не помешает. Тем более что домашние каталоги настраиваются намного быстрее и проще, чем виртуальный сервер — нужно всего лишь определить директиву **UserDir** и указать местоположение домашних каталогов.

Доступ к файлам, расположенным в этих каталогах, производится с помощью указания регистрационного имени пользователя после имени

сервера через тильду-слэш. Например, пусть имя сервера www.server.com, имя пользователя — denis, тогда URL-адрес будет выглядеть так: `http://www.server.com/~denis`. При этом сервер самостоятельно определит, где именно расположен домашний каталог пользователя. Если это каталог `/home/den`, то сервер передаст клиенту файл `/home/den/public_html/index.html`.

16.9. Виртуальный HTTP-сервер

Концепция виртуальных узлов позволяет одному серверу **Apache** поддерживать несколько сайтов. Пользователи видят отдельные веб-узлы, и получается, что один веб-сервер заменяет несколько. Это очень удобно, если нужно организовать персональные веб-сайты пользователей или собственные узлы подразделений компании, например, develop.mycompany.com.

Сервер **Apache** можно настроить несколькими способами: либо так, чтобы запускался один сервер, который будет прослушивать все обращения к виртуальным серверам (такой вариант настраивается при помощи директивы **VirtualHost**), либо запускать отдельный процесс для каждого виртуального сервера (в этом случае применяются директивы **Listen** и **BindAddress**). В этом параграфе я буду рассматривать первый вариант.

Внутри блока директивы **VirtualHost** можно использовать любые директивы, кроме **ServerType**, **BindAddress**, **Listen**, **NameVirtualHost**, **ServerRoot**, **TypesConfig**, **PidFile**, **MinRequestPerChild**, **MaxSpareServers**, **MinSpareServers**, так как некоторые из них относятся к основному HTTP-серверу (например, **ServerType**), а некоторые — ко второму варианту настройки виртуальных серверов и здесь неприемлемы. Обязательно должны присутствовать директивы **ServerName**, **DocumentRoot**, **ServerAdmin** и **ErrorLog**.

В зависимости от версии и от настроек **Apache** виртуальные узлы могут прописываться либо в файле `httpd.conf`, либо в файле `vhosts.conf`.

Виртуальные серверы можно идентифицировать по имени или по IP-адресу.

16.9.1. Виртуальные серверы с идентификацией по имени

Идентификация по имени имеет существенное преимущество перед идентификацией по IP-адресу: вы не ограничены количеством адресов, имеющимся в вашем распоряжении. Вы можете использовать любое количество

виртуальных серверов, и при этом вам не потребуются дополнительные адреса. Такое возможно благодаря использованию протокола HTTP/1.1. Данный протокол поддерживается всеми современными браузерами.

Поддержка виртуальных узлов обеспечивается директивами **VirtualHost** и **NameVirtualHost**. Если ваша система имеет только один IP-адрес, его нужно указать в директиве **NameVirtualHost**. Все блоки **VirtualHost** будут использовать этот IP-адрес.

Внутри блока **VirtualHost** записывается директива **ServerName**, задающая доменное имя для создаваемого виртуального сервера. Ее обязательно нужно записать, чтобы избежать поиска службой DNS — вы же не хотите, чтобы при неудачном поиске виртуальный сервер был заблокирован? Другие директивы в блоках **VirtualHost** описывают параметры каждого виртуального сервера в отдельности (листинг 16.10).

Листинг 16.10. Два виртуальных сервера — www и lib

```
ServerName den.dhsilabs.com

<NameVirtualHost 192.168.1.1>

<VirtualHost 192.168.1.1>
 ServerName www.dhsilabs.com
 ServerAdmin webmaster@den.dhsilabs.com
 DocumentRoot /var/httpd/www/html
 ErrorLog /var/https/www/logs/error.log
 TransferLog logs/access.log
</VirtualHost>

<VirtualHost 192.168.1.1>
 ServerName lib.dhsilabs.com
 ServerAdmin webmaster@den.dhsilabs.com
 DocumentRoot /var/httpd/lib/html
 ErrorLog /var/https/lib/logs/error.log
 TransferLog logs/access.log
</VirtualHost>
```

Если ваша система имеет только один IP-адрес, доступ к основному серверу напрямую будет невозможен: нужно включить его имя (в примере www.dhsilabs.com) в число виртуальных. Эти имена должны быть прописаны в службе DNS. При наличии двух IP-адресов можно один присвоить основному серверу, а другой — виртуальному.

Сервер **Apache** позволяет использовать несколько доменных имен для доступа к одному серверу, например:

ServerAlias www.dhsilabs.com www2.dhsilabs.com

При этом запросы, посланные по IP-адресам, которые присвоены вашим виртуальным узлам, должны соответствовать одному из указанных доменных имен. Чтобы зафиксировать запросы, не соответствующие ни одному из этих имен, нужно с помощью опции default:* создать виртуальный узел, который будет обслуживать такие запросы:

```
<VirtualHost _default_:*>
```

16.9.2. Виртуальные серверы с идентификацией по IP-адресу

В директиве **VirtualHost** в качестве адресов можно использовать доменные имена, но лучше указывать IP-адрес, причем действительный, а не виртуальный. В этом случае вы не будете зависеть от DNS при разрешении имени. Также потребуется один IP-адрес для вашего основного сервера. Если же распределить все адреса между виртуальными серверами, то нельзя будет получить доступ к основному серверу.

Листинг 16.11. Идентификация по IP-адресу

```
<VirtualHost 192.168.1.2>
 ServerName www.dhsilabs.com
 ServerAdmin webmaster@den.dhsilabs.com
 DocumentRoot /var/httpd/www/html
 ErrorLog /var/https/www/logs/error.log
</VirtualHost>

<VirtualHost lib.dhsilabs.com>
 ServerName lib.dhsilabs.com
 ServerAdmin webmaster@den.dhsilabs.com
 DocumentRoot /var/httpd/lib/html
 ErrorLog /var/https/lib/logs/error.log
</VirtualHost>
```

При конфигурировании виртуальных серверов можно использовать опцию **ExecCGI**, которая разрешает выполнение CGI-сценариев на виртуальном сервере. Листинги 16.12 и 16.13 демонстрируют настройку почтового веб-интерфейса.

Листинг 16.12. Файл httpd.conf

```
<Directory /home/httpd/mail>
 order deny,allow
 deny from all
 allow from localhost
 allow from 192.168
 allow from 123.123.123.123
 Options ExecCGI
</Directory>
```

Листинг 16.13. Файл vhosts.conf

```
<VirtualHost 123.123.123.123>
 ServerAdmin webmaster@den.dhsilabs.com
 DocumentRoot /home/httpd/mail
 ServerPath /mail
 ServerName wwwmail.dhsilabs.com
 ErrorLog logs/error_log
 TransferLog logs/access_log
# Error 403 — ошибка доступа извне, то есть почтовый
# интерфейс будет доступен только из локальной сети
 ErrorDocument 403 http://www.dhsilabs.com/messages/error403.html
</VirtualHost>
```

16.10. SSL и Apache

16.10.1. Установка SSL

SSL (Secure Sockets Layer) является методом шифрования, разработанным компанией Netscape для обеспечения безопасности передачи данных. Этот метод поддерживает несколько методов шифрования и обеспечивает аутентификацию как на уровне клиента, так и на уровне сервера. SSL работает на транспортном уровне и поэтому обеспечивает надежное шифрование всех типов данных. Более подробно о реализации SSL можно прочитать на сайте компании Netscape — `http://home.netscape.com/info/security-doc.html`.

Протокол S-HTTP является еще одним «безопасным» интернет-протоколом. Он был разработан для предоставления конфиденциальности данных, передаваемых через соединение. Конфиденциальность нужна, например, при передаче номеров кредитных карточек и прочей важной информации.

Модуль mod_ssl реализует в сервере **Apache** слой SSL, который осуществляет шифрование всего потока данных между клиентом и сервером. Для всех остальных частей веб-сервера модуль mod_ssl является прозрачным. Для работы в этом режиме требуется браузер, поддерживающий механизм SSL (этому условию удовлетворяют все распространенные сегодня браузеры).

Что касается установки, то вам необходим пакет OpenSSL (`http://www.openssl.org`), хотя, возможно, у вас в системе уже установлен этот пакет (в современных дистрибутивах он устанавливается по умолчанию). Если вы будете собирать OpenSSL из исходных текстов, то последовательность ваших действий такая:

```
# tar zxvf openssl-x.y.z.tar.gz # x.y.z - номер версии
# cd openssl-x.y.z
# ./config.
# make
# make install
# ldconfig
```

Перед выполнением команды **ldconfig** убедитесь, что в файле `/etc/ld.so.conf` прописан путь к библиотекам OpenSSL (по умолчанию это `/usr/local/ssl/lib`).

16.10.2. Подключение SSL к Apache

Версия mod_ssl, которую вам нужно установить, должна быть совместима с вашей версией **Apache**, иначе модуль mod_ssl будет некорректно работать или вообще откажется что-либо делать. Последние цифры в названии модуля указывают на совместимость с определенной версией **Apache**. Например, для **Apache** 1.3.14 нужен файл mod_ssl-2.7.1-1.3.14.tar.gz.

Для сборки mod_ssl из исходных текстов выполните команду:

```
#./configure --with-apache=../apache_1.3.14 --with-ssl=../openssl-0.9.5
```

В данном примере я использую OpenSSL 0.9.5. Теперь перейдите в каталог с **Apache**, откомпилируйте его и установите сертификат:

```
# cd ../apache-1.3.14
# make
# make certificate
# make install
```

Таким образом вы установите **Apache** в каталог, указанный в опции --prefix (по умолчанию `/usr/local/apache`).

Теперь попробуйте запустить **Apache**. Это можно сделать с помощью команды:

```
# usr/local/apache/bin/apachectl startssl
```

Параметр startssl необходим для включения SSL. Сервер **Apache** уже функционирует, однако обратиться по протоколу HTTPS вы еще не можете. Для этого вам нужно сконфигурировать виртуальные узлы, которые будут использовать протокол HTTPS. Но для начала необходимо настроить **Apache** для прослушивания порта 443 (это стандартный порт для протокола HTTPS). Добавьте в файл `/etc/httpd/conf/httpd.conf` следующие строки:

```
Listen 443
NameVirtualHost x.x.x.x:443
```

Теперь непосредственно приступите к созданию виртуального сервера, работающего по протоколу HTTPS, для чего продолжите редактирование файла `/etc/httpd/conf/httpd.conf`. Пример того, что необходимо при этом ввести, приведен в листинге 16.14. После этих обязательных директив вы можете конфигурировать свой виртуальный узел как обычно.

Листинг 16.14. Виртуальный HTTPS-сервер

```
<VirtualHost x.x.x.x:443>
# Эти строки нужны для поддержки SSL
SSLEngine on
SSLLogLevel warn
SSLOptions +StdEnvVars
SSLCertificateFile /usr/local/apache/conf/ssl.crt/server.
crt
SSLCertificateKeyFile /usr/local/apache/conf/ssl.key/server.key
SSLLog /usr/local/apache/logs/ssl_engine_log
# ------------------------------------

ServerName www.dhsilabs.com
ServerAdmin webmaster@den.dhsilabs.com
DocumentRoot /var/httpd/www/html
ErrorLog /var/https/www/logs/error.log

</VirtualHost>
```

Теперь нужно перезапустить сервер **httpd**. При запуске **Apache** потребует ввести пароль. Если вы не хотите вводить его при каждом запуске системы, то перейдите в каталог, где находится файл `ssl.key`, и выполните следующие команды:

```
# cp server.key server.key.org
# openssl rsa -in server.key.org -out server.key
# chmod 400 server.key
```

Почти все готово! Теперь сервер не должен запрашивать пароль и будет работать в нормальном режиме.

Чтобы **Apache** по умолчанию стартовал с поддержкой SSL, исправьте в файле `apachectl` (он устанавливается в каталог `/bin`, `/sbin` или `/usr/sbin`, см. `rpm -ql <пакет_Apache>`) условие start на startold, а startssl на просто start. Затем, находясь в каталоге `/usr/local/bin`, установите ссылку openssl:

```
# ln -s /usr/local/ssl/bin/openssl openssl
```

16.10.3. Генерирование сертификатов

Сертификат гарантирует безопасное подключение к веб-серверам и/или удостоверяет личность владельца. Идентификация обеспечивается путем применения личного ключа, известного только пользователю данной системы. Когда пользователь посещает защищенный узел для передачи секретной информации (например, номеров кредитных карточек) по протоколу HTTPS, узел автоматически посылает ему сертификат.

Итак, давайте приступим к генерированию сертификатов. Для этого сначала выполните команду:

```
# openssl genrsa -des3 -out server.key 1024
```

Она создаст файл `server.key`. После этого вы должны подать запрос в службу верификации:

```
# openssl req -new -key server.key -out server.csr
```

Здесь вам придется ответить на вопросы. Если вы ошибетесь — ничего страшного, все можно будет повторить заново. В том случае, если запрос сгенерирован правильно, вы должны получить такую надпись:

```
You now have to send this Certificate Signing Request (CSR)
to a Certifying Authority (CA) for signing
```

Отвечая на вопросы, будьте очень внимательны — ваши ответы увидит весь мир.

По всем правилам вы сейчас должны подписать сертификат у сертифицирующей организации. Это услуга платная, и оказывает ее компания ThawTe (`www.thawte.com`), которую в России представляет `www.solutions.rbc.ru`. Бесплатно вы можете получить только «самоподписанный» сертификат. Скопируйте `sign.sh` из пакета mod_ssl в каталог с ключами и подпишите себя сами:

```
# openssl req -new -x509 -days 365 -key server.key -out server.csr
#./sign.sh server.csr
```

Если на экране появится надпись

```
Now you have two files: server.key and server.crt.
These now can be used as following
```

то это означает, что все собрано правильно. Затем скопируйте новые файлы `server.key` и `server.crt` на место старых. Выполните команду **make** в каталоге с .crt-файлом.

В итоге вы получите полностью работающий **Apache**, защищенный SSL. Для сбора полной информации о работе SSL введите:

```
# openssl s_client -connect localhost:443 -state -debug
```

16.11. Пример файла httpd.conf

В этом параграфе приведен пример стандартной конфигурации сервера **Apache**. Каждому блоку листинга 16.15 сопутствуют комментарии, которые помогут вам разобраться с различными настройками сервера.

Листинг 16.15. Пример файла `httpd.conf`

```
##
## httpd.conf -- файл конфигурации сервера HTTP Apache
##
# ---------------------------------------------------------------

# Установите имя сервера
ServerName www.dhsilabs.com
ResourceConfig /dev/null
AccessConfig /dev/null

# Поддержка динамических разделяемых объектов (Dynamic
# Shared Object -- DSO)
# Для более подробной информации о DSO прочтите файл
# README.DSO,
# входящий в дистрибутив Apache.
# Модуль расширяет возможности сервера Apache,
# добавляет в его состав новые функции.
# Подключить модуль можно так
# LoadModule foo_module libexec/mod_foo.so
# Вы можете найти документацию по модулям в файле
# "/var/www/manual/mod"
```

```
#LoadModule mmap_static_module modules/mod_mmap_static.so
LoadModule env_module          modules/mod_env.so

### The first module activates buffered logs.
# Первый модуль обеспечивает протоколирование.
# Он запишет информацию
# в протокол access_log, когда буфер объемом 4K
# переполнится.
#LoadModule config_buffered_log_module  modules/mod_log_
config_buffered.so
LoadModule config_log_module  modules/mod_log_config.so

LoadModule agent_log_module   modules/mod_log_agent.so
LoadModule referer_log_module modules/mod_log_referer.so
#LoadModule mime_magic_module modules/mod_mime_magic.so
LoadModule mime_module        modules/mod_mime.so
LoadModule negotiation_module modules/mod_negotiation.so
LoadModule status_module      modules/mod_status.so
LoadModule info_module        modules/mod_info.so

# Вы должны выбрать директиву mod_include
# или mod_include_xssi,
# но не обе одновременно! Директива mod_include более
# безопасна, но xxsi содержит больше функций.
LoadModule includes_module    modules/mod_include.so
#LoadModule includes_module   modules/mod_include_xssi.so

LoadModule autoindex_module   modules/mod_autoindex.so
LoadModule dir_module         modules/mod_dir.so
LoadModule cgi_module         modules/mod_cgi.so
LoadModule asis_module        modules/mod_asis.so
LoadModule imap_module        modules/mod_imap.so
LoadModule action_module      modules/mod_actions.so
#LoadModule speling_module    modules/mod_speling.so
LoadModule userdir_module     modules/mod_userdir.so
LoadModule proxy_module       modules/libproxy.so
LoadModule alias_module       modules/mod_alias.so

# Модуль mod_jserv должен быть объявлен до mod_rewrite.
<IfDefine HAVE_JSERV>
LoadModule jserv_module       modules/mod_jserv.so
</IfDefine>

LoadModule rewrite_module     modules/mod_rewrite.so
LoadModule access_module      modules/mod_access.so
LoadModule auth_module        modules/mod_auth.so
```

```
LoadModule anon_auth_module   modules/mod_auth_anon.so
#LoadModule dbm_auth_module   modules/mod_auth_dbm.so
#LoadModule db_auth_module    modules/mod_auth_db.so
LoadModule digest_module      modules/mod_digest.so
#LoadModule cern_meta_module  modules/mod_cern_meta.so
LoadModule expires_module     modules/mod_expires.so
LoadModule headers_module     modules/mod_headers.so
LoadModule usertrack_module   modules/mod_usertrack.so
#LoadModule example_module    modules/mod_example.so
#LoadModule unique_id_module  modules/mod_unique_id.so
LoadModule setenvif_module    modules/mod_setenvif.so

# В полном списке модулей должны быть перечислены все
# доступные модули (статические или разделяемые)
# в правильном порядке выполнения.
ClearModuleList
# AddModule mod_mmap_static.c
# AddModule mod_php.c
# AddModule mod_php3.c
# AddModule mod_php4.c
# AddModule mod_perl.c
# LoadModule php_module modules/mod_php.so
# LoadModule php3_module modules/mod_php4.so
# LoadModule php4_module modules/mod_php4.so
AddModule mod_env.c
AddModule mod_log_config.c
#AddModule mod_log_config_buffered.c
AddModule mod_log_agent.c
AddModule mod_log_referer.c
#AddModule mod_mime_magic.c
AddModule mod_mime.c
AddModule mod_negotiation.c
AddModule mod_status.c
AddModule mod_info.c
AddModule mod_include.c
#AddModule mod_include_xssi.c
AddModule mod_autoindex.c
AddModule mod_dir.c
AddModule mod_cgi.c
AddModule mod_asis.c
AddModule mod_imap.c
AddModule mod_actions.c
#AddModule mod_speling.c
AddModule mod_userdir.c
AddModule mod_proxy.c
AddModule mod_alias.c
```

```
# Модуль mod_jserv должен быть объявлен до mod_rewrite.
<IfDefine HAVE_JSERV>
AddModule mod_jserv.c
</IfDefine>

AddModule mod_rewrite.c
AddModule mod_access.c
AddModule mod_auth.c
AddModule mod_auth_anon.c
#AddModule mod_auth_dbm.c
#AddModule mod_auth_db.c
AddModule mod_digest.c
#AddModule mod_cern_meta.c
AddModule mod_expires.c
AddModule mod_headers.c
AddModule mod_usertrack.c
#AddModule mod_example.c
#AddModule mod_unique_id.c
AddModule mod_so.c
AddModule mod_setenvif.c

#----------- Name Space and Server Settings -------------
# Настройки пространства имен и сервера
# В этом разделе вы определяете, какие имена будут видеть
# пользователи  вашего HTTP-сервера. Этот файл также
# определяет настройки сервера,
# которые раньше содержались в отдельном файле srm.conf.
# Теперь этот файл входит в состав httpd.conf

# Директива DocumentRoot определяет местонахождение
# корневого каталога документов вашего сервера.

DocumentRoot /var/www/html

# Директива UserDir задает названия подкаталога в домашнем
# каталоге пользователя, из которого берутся документы в
# том случае, когда вы активизируете возможность
# использования пользовательских каталогов.
UserDir public_html

# Директива DirectoryIndex позволяет задать название
# документа, который будет
# возвращен по запросу, который не содержит имя документа.
DirectoryIndex index.html index.php index.htm index.shtml
index.cgi Default.htm default.htm index.php3
```

```
# Директива FancyIndexing определяет оформление
# каталога - стандартное или индексируемое.

FancyIndexing on

# Директивы AddIcon* указывают серверу, какие пиктограммы
# использовать для показа различных типов файлов

AddIconByEncoding (CMP,/icons/compressed.gif) x-compress x-gzip

AddIconByType (TXT,/icons/text.gif) text/*
AddIconByType (IMG,/icons/image2.gif) image/*
AddIconByType (SND,/icons/sound2.gif) audio/*
AddIconByType (VID,/icons/movie.gif) video/*

AddIcon /icons/binary.gif .bin .exe
AddIcon /icons/binhex.gif .hqx
AddIcon /icons/tar.gif .tar
AddIcon /icons/world2.gif .wrl .wrl.gz .vrml .vrm .iv
AddIcon /icons/compressed.gif .Z .z .tgz .gz .zip
AddIcon /icons/a.gif .ps .ai .eps
AddIcon /icons/layout.gif .html .shtml .htm .pdf
AddIcon /icons/text.gif .txt
AddIcon /icons/c.gif .c
AddIcon /icons/p.gif .pl .py
AddIcon /icons/f.gif .for
AddIcon /icons/dvi.gif .dvi
AddIcon /icons/uuencoded.gif .uu
AddIcon /icons/script.gif .conf .sh .shar .csh .ksh .tcl
AddIcon /icons/tex.gif .tex
AddIcon /icons/bomb.gif core
AddIcon /icons/back.gif ..
AddIcon /icons/hand.right.gif README
AddIcon /icons/folder.gif ^^DIRECTORY^^
AddIcon /icons/blank.gif ^^BLANKICON^^

# Директива DefaultIcon определяет пиктограмму
# по умолчанию.
DefaultIcon /icons/unknown.gif

# Директива AddDescription задает описание файла
# Формат: AddDescription "описание" filename

# Директива ReadmeName определяет имя файла README
# по умолчанию
# Формат: ReadmeName name
```

```
ReadmeName README
HeaderName HEADER

# Директива IndexIgnore определяет набор файлов, которые
# будут проигнорированы при индексировании
# Формат: IndexIgnore name1 name2...

IndexIgnore .??* *~ *# HEADER* README* RCS

# Директива AccessFileName определяет имя файла, содержащего
# директивы управления  доступом

AccessFileName .htaccess

# Директива TypesConfig задает местонахождение
# файла mime.types

TypesConfig /etc/httpd/conf/apache-mime.types

# С помощью директивы DefaultType можно указать
# MIME-тип по умолчанию для документов, тип которых
# сервер определить не может
DefaultType text/plain

# Директива AddEncoding разрешает браузеру распаковывать
# информацию "на лету"

AddEncoding x-compress Z
AddEncoding x-gzip gz

# AddLanguage разрешает определять язык документа
AddLanguage en .en
AddLanguage fr .fr
AddLanguage de .de
AddLanguage da .da
AddLanguage el .el
AddLanguage it .it

# Директива LanguagePriority определяет приоритет языков
LanguagePriority en fr de

# Директива Redirect позволяет перенаправить клиента
# на другой URL. Вы можете перенаправить клиента
# на другой узел или на URL, который находится в вашем
# пространстве имен, то есть на документ,
# который находится в одном из подкаталогов каталога
```

```
# DocumentRoot. Вы не  можете, например, перенаправить
# клиента к каталогу /etc, потому что он не
# находится в вашем пространстве имен.
# URL — это идентификатор ресурса, поэтому вы должны
# его указывать в виде
# протокол://адрес.домен, например, http://www.linux.ru.
# Если вы укажете просто каталог, например, /images, этот
# каталог должен быть подкаталогом каталога DocumentRoot,
# а не  корневого каталога вашей основной файловой
# системы. Формат: Redirect несуществующий_url url

# С помощью директивы Alias можно предоставить доступ
# не только к файлам, находящимся в каталоге, указанном
# директивой DocumentRoot, и его подкаталогах, но и в
# других каталогах. Формат:
# Alias несуществующее_имя нормальное_имя

Alias /icons/ /var/www/icons/

# ScriptAlias определяет расположение каталога сценариев CGI
# Формат: ScriptAlias подставное_имя настоящее_имя

ScriptAlias /cgi-bin/ /var/www/cgi-bin/
ScriptAlias /protected-cgi-bin/ /var/www/protected-cgi-bin/

# С помощью директивы AddType можно добавить новый
# тип MIME, который не указан в файле apache-mime.types.
# Формат: AddType type/subtype ext1

# Обычно для модуля PHP4 (он не является частью Apache)
# директива AddType используется так:
AddType application/x-httpd-php4 .php3 .phtml .php .php4
# AddType application/x-httpd-php3-source .phps
# Для PHP/FI (PHP2):
# AddType application/x-httpd-php .phtml
# ScriptAlias /_php/ /usr/bin/php
# Action application/x-httpd-php /usr/bin/php
# Action application/x-httpd-php3 /usr/bin/php
# Action application/x-httpd-php4 .

# Директива AddHandler позволяет сопоставить
# определенному типу MIME какой-нибудь обработчик.
# Формат: AddHandler action-name ext1

# Для использования сценариев CGI:
AddHandler cgi-script .cgi
```

```
# Для использования генерируемых сервером файлов HTML
AddType text/html .shtml
AddHandler server-parsed .shtml

# Раскомментируйте нижерасположенную строку,
# чтобы включить функцию
# Apache "отправь-как-есть" (send-as-is)
#AddHandler send-as-is asis

# Если вы хотите использовать карты изображений:
AddHandler imap-file map

# Для включения карт типов используйте:
#AddHandler type-map var

# С помощью директивы Action можно определить
# какое-нибудь действие для обработчика. Например, вы
# можете запустить какую-нибудь программу
# для обработки файла данного типа.
# Формат: Action media/type /cgi-script/location
# Формат: Action handler-name /cgi-script/location

# Директива MetaDir определяет имя каталога, в котором
# сервер Apache может найти информационные файлы meta.
# Эти файлы содержат дополнительные заголовки HTTP,
# которые будут добавлены к документу
# перед его передачей клиенту.

#MetaDir .web

# Директива MetaSuffix определяет имя суффикса файла,
# который содержит
# meta-тэги.

#MetaSuffix .meta

# Здесь можно определить сообщения об ошибках.
# Это можно сделать тремя способами:
# 1) обыкновенный текст. Символ "кавычка" обозначает текст
#  и клиенту не посылается
# ErrorDocument 500 "Ошибка сервера.
#
#  2) локальное перенаправление на документ или сценарий
# ErrorDocument 404 /missing.html
# ErrorDocument 404 /cgi-bin/missing_handler.pl
#
```

```
#  3) внешнее перенаправление
# ErrorDocument 402 http://some.other_server.com/
# subscription_info.html
#
# Модуль mod_mime_magic позволяет серверу использовать
# различные подсказки из файла для определения его типа.
# MimeMagicFile /etc/httpd/conf/magic

# Следующие директивы необходимы для браузеров Netscape 2.x и
# Internet Explorer  4.0b2

BrowserMatch "Mozilla/2" nokeepalive
BrowserMatch "MSIE 4\.0b2;" nokeepalive downgrade-1.0
force-response-1.0

# Следующие директивы отключают ответы HTTP/1.1 для браузеров,
# которые не поддерживают протокол HTTP/1.1

BrowserMatch "RealPlayer 4\.0" force-response-1.0
BrowserMatch "Java/1\.0" force-response-1.0
BrowserMatch "JDK/1\.0" force-response-1.0

#----------------- Настройки доступа --------------------

# В этом разделе определяются настройки сервера,
# которые управляют доступом к серверу. Раньше эти
# настройки находились в файле access.conf.

# Каждый каталог, к которому Apache может получить
# доступ, может быть сконфигурирован определенным образом.
# Можно запретить выполнение  некоторых операций, доступ
# определенных пользователей или узлов сети.
# Установки доступа распространяются на весь каталог
# и на все его подкаталоги.

# Прежде всего конфигурируем корневой каталог
# для установки полномочий доступа.

<Directory />
Options Indexes Includes FollowSymLinks
AllowOverride None
</Directory>

<Directory /home>

# Здесь должны быть определены директивы "Includes",
# "FollowSymLinks", "ExecCGI",  "MultiViews" или любая
```

```
# комбинация "Indexes"

Options Indexes Includes FollowSymLinks

AllowOverride All

# Разрешает доступ всем
order allow,deny
allow from all

</Directory>

# Каталоги /var/www/cgi-bin и /var/www/protected-cgi-bin
# должны быть определены с помощью директивы ScriptAliased

<Directory /var/www/cgi-bin>
AllowOverride All
Options ExecCGI
</Directory>

<Directory /var/www/protected-cgi-bin>
order deny,allow
deny from all
allow from localhost
#allow from .your_domain.com
AllowOverride None
Options ExecCGI
</Directory>

# Разрешает отчеты о состоянии сервера

<Location /server-status>
SetHandler server-status

order deny,allow
deny from all
allow from localhost, 127.0.0.1
# Установите здесь имя вашего домена
# allow from .your_domain.com
</Location>

# Разрешает доступ к файлам документации для локальной машины.
Alias /doc /usr/share/doc
<Directory /usr/share/doc>
order deny,allow
```

```
deny from all
allow from localhost, 127.0.0.1
# allow from .your_domain.com
Options Indexes FollowSymLinks
</Directory>

#---------------- Конфигурация сервера -------------------

# Тип сервера: inetd или standalone.

ServerType standalone

# Если вы используете тип inetd, перейдите
# к директиве "ServerAdmin"

# Директива Port — только для standalone-сервера.
# Если вы хотите запустить
# сервер Apache с использованием этого или любого
# другого порта, номер
# которого меньше 1024, вы должны обладать правами
# суперпользователя. Но
# даже если у вас нет таких прав, вы можете запустить
# сервер для работы с портом, номер которого превышает
# значение 1024. Обычно используется
# номер 8080 или 8000.

Port 80

# Сервер Apache ведет журнал доступа других компьютеров.
# Если вы включите следующую опцию, то в журнал будет
# записано доменное имя компьютера-клиента. Если эта
# опция выключена, то в журнал будет записан IP-адрес
# клиента. Включение данной опции замедляет работу
# сервера, так как ему требуется дополнительное время
# на ожидание ответа от сервера DNS.

HostnameLookups off

# Директивы User и Group определяют идентификаторы
# пользователя и группы, от имени которых запускается
# сервер в автономном режиме. Можно использовать как
# регистрационные имена, так и UID. По умолчанию
# используется имя пользователя nobody или apache. Из
# соображений безопасности не рекомендуется изменять
# это значение и присваивать имя реального пользователя.

User apache
```

```
Group apache

# Директива ServerAdmin задает электронный адрес веб-
# мастера вашего веб-узла. В случае возникновения ошибок
# именно по этому адресу будет отправлено сообщение.

ServerAdmin root@localhost

# В директиве ServerRoot указывается местонахождение
# файлов конфигурации сервера Apache. По умолчанию
# используется каталог /etc/httpd.

ServerRoot /etc/httpd

# Следующая директива используется для компьютеров,
# которые имеют несколько IP-адресов. Обычно данная
# директива используется для конфигурирования
# виртуальных узлов.
# BindAddress *

# Прослушивать порт 80
Listen 80

# Директивы ErrorLog и TransferLog определяют расположение
# журналов сервера Apache. Обычно используется каталог
# /etc/httpd/logs, который  является ссылкой на каталог
# /var/log/httpd или на любой другой.

ErrorLog logs/error_log

# LogLevel: устанавливает уровень протоколирования.
# Протоколируются предупреждающие сообщения сервера (warn)
# и ошибки. Если вы хотите протоколировать только ошибки,
# установите error

LogLevel warn

# Определяет формат файлов протокола, то есть информацию,
# которая будет протоколироваться. Обычно изменять эти
# значения не нужно.

LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\"
\"%{User-Agent}i\"" combined
LogFormat "%h %l %u %t \"%r\" %>s %b" common
LogFormat "%{Referer}i -> %U" referer
LogFormat "%{User-agent}i" agent
```

```
# Можно определить тип протокола
# Если вы хотите протоколировать общую информацию
#CustomLog logs/access_log common
# Если вы хотите протоколировать referer
#CustomLog logs/referer_log referer
# Если вы хотите протоколировать название пользовательских
# агентов (браузеров)
#CustomLog logs/agent_log agent
# По умолчанию используется комбинированный тип
# протоколирования, то есть
# будет протоколироваться вся информация

CustomLog logs/access_log combined

# С помощью директивы PidFile указывается имя файла, в
# котором исходный
# процесс сервера будет регистрироваться.

PidFile /var/run/httpd.pid

# ScoreBoardFile: Этот файл используется для сохранения
# внутренней информации процесса сервера.

ScoreBoardFile /etc/httpd/httpd.scoreboard

# Директива LockFile определяет файл блокировки, который
# используется сервером. Сервер должен быть скомпилирован
# с опцией
# USE_FCNTL_SERIALIZED_ACCEPT или
# USE_FLOCK_SERIALIZED_ACCEPT. Файл блокировки должен быть
# сохранен НА ЛОКАЛЬНОМ ДИСКЕ.
#
LockFile /etc/httpd /httpd.lock

ServerName new.host.name

# Директива UseCanonicalName появилась в версии
# Apache 1.3. Она разрешает
# использовать каноническое имя для сервера узла.

UseCanonicalName on

# Следующая директива позволяет прокси-серверу, например
# SQUID, не кэшировать  документы, которые не генерируются
# автоматически, то есть в процессе выполнения различных
# сценариев.
```

```
CacheNegotiatedDocs

# Директива Timeout задает промежуток времени в секундах,
# в течение которого сервер продолжает попытки
# возобновления приостановленной передачи данных.
# Значение директивы Timeout распространяется не только
# на передачу, но и на прием данных.

Timeout 300

# Директива KeepAlive разрешает постоянные соединения,
# то есть такие соединения, в которых производится более
# одного запроса за один раз.

KeepAlive off

# MaxKeepAliveRequests: Максимальное количество запросов,
# разрешенное в течение постоянного соединения. Установите
# 0 для снятия ограничения.

MaxKeepAliveRequests 100

# Директива KeepAliveTimeout  определяет таймаут для
# постоянного соединения.

KeepAliveTimeout 15

# Минимальное и максимальное число серверов в очереди
MinSpareServers 8
MaxSpareServers 20
StartServers 10

# Ограничивает общее количество клиентов. Когда это число
# будет превышено,новые клиенты получат отказ, поэтому это
# число не должно быть слишком  маленьким.

MaxClients 150

# После обработки определенного количества запросов,
# указанного в директиве MaxRequestsPerChild, копия
# сервера завершается, а вместо нее запускается новая.

MaxRequestsPerChild 500

# Директивы конфигурации прокси-сервера

# ProxyRequests On
```

```
# Для включения кэширования раскомментируйте строки ниже:

# CacheRoot /var/cache/httpd
# CacheSize 5
# CacheGcInterval 4
# CacheMaxExpire 24
# CacheLastModifiedFactor 0.1
# CacheDefaultExpire 1
# NoCache a_domain.com another_domain.edu joes.garage_sale.com

####################################
# Настройки производительности SGI   #
####################################
#
# Для использования этой функции раскомментируйте модуль
# mod_mmap_static в разделе описания модулей.
<IfModule mod_mmap_static.c>
QSC on
</IfModule>

# Если вы хотите использовать буферизированное
# протоколирование, раскомментируйте модуль
# mod_log_config_buffered в разделе описания модулей.

# Для использования карты памяти раскомментируйте эту строку:
# mmapfile /var/www/html/file_to_map_in_memory
#
# Если вы хотите настроить процессы-потомки, пожалуйста,
# прочитайте документацию на вашем сервере
# http://localhost/manual/misc/perf-mja.html.
# Эта страница объясняет, как привязать определенный
# IP-адрес или порт к другому процессору.
# SingleListen On

####################################
# Виртуальные серверы #
####################################
#
# Поддержка модуля perl
# Замечание: не удаляйте расположенные далее строки,
# иначе это может разрушить вашу конфигурацию.
<IfDefine PERLPROXIED>
 ProxyPass   /perl/ http://127.0.0.1:8200/
 ProxyPassReverse /perl/ http://127.0.0.1:8200/
</IfDefine>
```

```
# Файл, в котором находятся директивы конфигурирования
# виртуальных узлов.
Include conf/vhosts/Vhosts.conf

#
# Для поддержки динамических виртуальных узлов и
# виртуальных домашних каталогов, раскомментируйте
# следующие строки:
# LoadModule vhost_alias_modulemodules/mod_vhost_alias.so
# AddModule mod_vhost_alias.c
# Include conf/vhosts/DynamicVhosts.conf
# Include conf/vhosts/VirtualHomePages.conf

# Директивы конфигурирования PHP
Include conf/addon-modules/php.conf
```

16.12. Перекодирование русскоязычных документов «на лету»

С тех пор, когда в русском языке появилось слово «кодировка», появилась и проблема перекодировки. Стандартной кодировкой русского языка для большинства UNIX-серверов является КОИ8-R. Для применения в DOS компания Microsoft разработала альтернативную кодировку (ALT), известную также под названием CP-866.

Потом та же Microsoft создала кодировку Windows-1251 (ANSI), чем обеспечила проблемы с перекодировкой уже на локальном уровне: файлы, созданные в DOS, без предварительного перекодирования нельзя было прочитать в Windows, и наоборот. Заметьте, об Интернете и **Apache** я еще не сказал ни слова.

Кроме Microsoft, «облегчили» нам жизнь также компании Apple и Sun, разработав соответственно кодировки Apple и ISO8859-5. Компания IBM не отстала от них, разработав собственную кодировку русского языка.

В Интернете все эти кодировки смешались. Возникает задача: настроить автоматическое перекодирование документов из одной кодировки в другую. Для начала необходимо настроить хотя бы перекодирование «на лету» из KOI8 в Windows-1251, так как большинство клиентов в Сети используют именно эту кодировку (от Windows, как от смерти, не уйдешь).

16.12.1. Russian Apache: установка, настройка, использование

Обыкновенный англоязычный **Apache**, входящий в состав большинства дистрибутивов, не поможет вам решить эту проблему. Для корректных операций по перекодированию нужно загрузить и установить сервер **Russian Apache** или модуль Apache-RUS. Скачать как модуль, так и готовый сервер можно по адресу: `ftp://apache.lexa.ru/pub/apache-rus`. При этом старшая часть версии соответствует оригинальному серверу **Apache**, младшая — версии модуля Apache-Rus.

Рассматривать процесс установки и настройки я буду на примере не очень новой версии сервера — 1.3.3/PL27.3, это не принципиально. Последовательность действий для сборки сервера из исходных кодов стандартная:

```
# tar xvzf apache_1.3.3rusPL27.3.tar.gz
# cd apache_1.3.3rusPL27.3
# ./configure
# make
# make install
```

Настройка **Russian Apache** не отличается от настройки оригинального **Apache** за исключением настройки модуля перекодирования.

16.12.2. Настройка перекодировки русскоязычных документов

Директивы перекодирования (они находятся в файле `httpd.conf`) можно разделить на три группы.

Первые указывают, в какой кодировке хранятся файлы на диске. Их можно включать в блок **Location** или в файлы `.htaccess`:

```
# все файлы, кроме .txt, хранятся в кодировке koi8-r
CharsetSourceEnc koi8-r
# все файлы .txt хранятся в кодировке windows-1251
CharsetByExtension windows-1251 .txt
```

Вторые определяют названия (CharsetDecl) и псевдонимы (CharsetAlias) кодировок и таблиц символов (CharsetRecodeTable и CharsetWideRecodeTable). Они находятся в блоке <IfModule mod_charset.c> ... </IfModule> и не требуют изменений. Название языка (ru) должно быть определено в директивах AddLanguage и LanguagePriority.

```
CharsetDecl windows-1251  ru
CharsetAlias windows-1251 win x-cp1251 cp1251 cp-1251
```

С помощью третьей, самой многочисленной, группы вы можете настроить сервер для автоматической перекодировки символов на основании информации о клиенте. Например, определив, что клиент работает в операционной системе Windows и кодировкой браузера по умолчанию является Windows-1251, сервер самостоятельно перекодирует файлы в нужную кодировку. Если сервер сделает выбор неправильно, пользователь всегда сможет сам изменить кодировку вручную.

Существует три способа выбора кодировки:

- по каталогу:
  `http://www.server.ru/koi/file.html`
  `http://www.server.ru/win/file.html`
- по имени сервера:
  `http://koi.www.server.ru/file.html`
  `http://win.www.server.ru/file.html`
- по порту:
  `http://www.server.ru:8000/file.html`
  `http://www.server.ru:8001/file.html`

Для перекодирования по каталогу (точнее, по его префиксу) нужно добавить в блоке **VirtualHost** псевдоним, например:

```
Alias /koi /www/docs
```

Или же просто создать в нужном каталоге ссылку на самого себя:

```
# cd /www/docs
# ln -s . koi
```

Несмотря на свою простоту, этот способ имеет множество недостатков. Если у вас небольшой сервер, вы можете использовать перекодировку по каталогу. В другом случае лучше используйте перекодировку по имени сервера или по порту.

При использовании перекодировки по имени сервера следует обратить внимание на то, чтобы указанный вами сервер был прописан на сервере DNS. После регистрации поддомена (в качестве имени которого нужно использовать один из псевдонимов кодировки, указанный с помощью директивы CharsetAlias, например, koi или win) внесите следующие строки в ваш файл `httpd.conf`:

```
# Естественно, укажите здесь свой IP-адрес
<VirtualHost 111.111.111.1>
ServerName www.server.ru
ServerAlias *.www.server.ru
# далее следует обычная конфигурация
...
</VirtualHost
```

Если сервер DNS администрируете не вы и возможности прописать новый поддомен у вас нет, то используйте перекодировку по порту. Для этого закомментируйте директиву **Port** в файле `httpd.conf` и вместо нее добавьте следующие директивы:

```
Listen 80
Listen 8000
Listen 8001
Listen 8002
Listen 8003
CharsetByPort koi8-r 8000
CharsetByPort windows-1251 8001
CharsetByPort ibm866 8002
CharsetByPort iso-8859-5 8003
```

Номера портов при этом не очень важны. Правда, есть одна неприятность: если сеть клиента защищена брандмауэром, не позволяющим обращаться к выбранному вами порту, клиент не сможет установить соединение с вашим сервером.

Схема (порядок) выбора кодировки определяется директивой CharsetSelectionOrder:

- CharsetSelectionOrder Dirprefix Useragent Portnumber Hostname UriHostname — для выбора по каталогу;
- CharsetSelectionOrder Hostname UriHostname Useragent Portnumber Dirprefix — для выбора по имени домена;
- CharsetSelectionOrder Portnumber Useragent Hostname UriHostname Dirprefix — для выбора по порту.

16.13. Защита сервера Apache

По окончании настройки сервера запретим изменение и удаление файла конфигурации:

```
[root@webserver]# chattr +i /etc/httpd/conf/httpd.conf
```

После этого вы (и никто другой) не сможете изменить этот файл даже с помощью конфигуратора.

Желательно также установить права 511 для исполняемого файла сервера httpd:

```
# chmod 511 /usr/sbin/httpd
```

Далее, не нужно, чтобы посторонние глаза смогли посмотреть, а руки — изменить (и выполнить) файлы, находящиеся в каталогах `/etc/httpd/conf` и `/var/log/httpd`:

```
# chmod 700 /etc/httpd/conf/
# chmod 700 /var/log/httpd /
```

16.14. Сервер kHTTPd — веб-сервер уровня ядра

В операционной системе все процессы можно разделить на два типа: процессы уровня ядра и пользовательские процессы. Процесс уровня ядра запускается и работает очень быстро по сравнению с относительно неповоротливым пользовательским процессом. Однако пользовательские процессы безопаснее для здоровья системы, нежели процессы уровня ядра. Если произойдет ошибка в пользовательском процессе, то на работе системы это обстоятельство никак не отразится. А ошибка в процессе уровня ядра чревата «крахом» системы, который вам обойдется во много раз дороже, чем выигрыш на быстродействии процесса уровня ядра.

Процессы уровня ядра позволяют существенно повысить производительность веб-сервера, то есть скорость обработки HTTP-запросов. Начиная с версии ядра 2.4, в состав ядра входит веб-сервер **kHTTPd**. Скорость его работы значительно выше скорости веб-сервера **Apache**, который выполняется как пользовательский процесс.

16.14.1. Настройка kHTTPd

Так как **kHTTPd** является процессом уровня ядра, его настройка выполняется путем записи информации в файлы, хранящиеся в каталоге `/proc/sys/net/khttpd`.

Сначала нужно перекомпилировать ядро (этому процессу посвящена отдельная глава), включив в его состав **kHTTPd**. Для этого включите опцию **Kernel HTTPd Acceleration** в меню **Network Options**.

После этого нужно настроить веб-сервер **Apache** для работы по порту 8080 (директива **Port** в файле `httpd.conf`) и перезапустить его:

```
# service httpd restart
```

Загрузите модуль ядра khttpd:

```
# insmod khttpd
```

Укажите серверу **kHTTPd**, что запросы клиентов нужно обрабатывать через порт 80, а те запросы, с которыми он не может справиться, передавать **Apache**:

```
# echo 80 > /proc/sys/net/khttpd/serverport
# echo 8080 > /proc/sys/net/clientport
```

Какие запросы не может обработать **kHTTPd**? Ему «не по зубам» запросы, предполагающие запуск сценария. Все такие запросы будут перенаправлены **Apache**. Поэтому, если ваш веб-сервер предполагает в основном запуск CGI-сценариев (в том числе и PHP-сценариев), использовать **kHTTPd** нецелесообразно. Вместо повышения производительности вы добьетесь обратного.

Сообщите серверу **kHTTPd**, где нужно искать веб-страницы (в том же каталоге, который указан в директиве **DocumentRoot** сервера **Apache**):

```
# echo /var/www/html > /proc/sys/net/khttpd/documentroot
```

Если на вашем сервере установлен PHP, укажите каталог, в котором хранятся PHP-сценарии:

```
# echo /var/www/html/scripts > /proc/sys/net/khttpd/dynamic
```

Для запуска **kHTTPd** введите следующую команду:

```
# echo 1 > /proc/sys/net/khttpd/start
```

Всю эту работу можно автоматизировать, написав сценарий khttpd-start (листинг 16.16):

Листинг 12.16. Сценарий автоматического запуска kHTTPd

```
#!/bin/bash
# Загружаем модуль kHTTPd
insmod khttpd
# Указываем порт kHTTPd
echo 80 > /proc/sys/net/khttpd/serverport
# Указываем порт Apache
echo 8080 > /proc/sys/net/clientport
# Корневой каталог веб-сервера и каталог
# для хранения сценариев
echo /var/www/html > /proc/sys/net/khttpd/documentroot
echo /var/www/html/scripts > /proc/sys/net/khttpd/dynamic
# Запускаем kHTTPd
echo 1 > /proc/sys/net/khttpd/start
```

Так как **kHTTPd** — это процесс уровня ядра, к тому же экспериментальный, его использование может отрицательно повлиять на надежность работы системы. Помните, что сервер **kHTTPd** не может обеспечить такой же уровень надежности, как **Apache**. **kHTTPd** следует применять только в том случае, если **Apache** не справляется с нагрузкой и на веб-сервере не часто запускаются CGI-сценарии.

Глава 17

УСТАНОВКА И НАСТРОЙКА MYSQL. СВЯЗКА APACHE+ PHP+MYSQL

- УСТАНОВКА MYSQL
- КЛИЕНТСКАЯ ЧАСТЬ MYSQL
- УСТАНОВКА PHP И НАСТРОЙКА СВЯЗКИ APACHE+PHP+MYSQL
- ЗАЩИТА СЕРВЕРА MYSQL
- ВВЕДЕНИЕ В ЯЗЫК SQL

Ни один серьезный интернет-проект нельзя построить без использования баз данных. Большинство провайдеров предоставляет хостинг вместе с одним из серверов баз данных. Самым популярным из таких серверов считается **MySQL**, получивший широкое распространение благодаря своей простоте. Здесь я не буду ни рассматривать технические характеристики **MySQL**, ни сравнивать его с другими серверами баз данных (InterBase Server, IBM DB/2, Oracle) — достаточно сказать, что InterBase Server или Oracle более масштабируемы и поэтому лучше подходят для организации распределенной системы обработки информации, но для обычного интернет-проекта **MySQL** подходит практически идеально. В этой главе я опишу его установку, настройку и использование.

17.1. Установка MySQL

Прежде всего нужно установить пакеты, необходимые для работы **MySQL**. У меня MySQL версии 4.0.15, поэтому я установил такие пакеты (номера версий у вас, возможно, будут другими, и я их обозначил символами «x»):

- MySQL_GPL-4.x.x
- MySQL_GPL-client-4.x.x
- MySQL_GPL-shared-libs-4.x
- MySQL_GPL-bench-4.x.x
- MySQL_GPL-resolveip-4.x.x

После установки пакетов нужно создать системную базу данных **mysql**, содержащую таблицы **db**, **host** и **user**. Скорее всего, она уже создана, но, чтобы окончательно убедиться в этом, введите команду:

```
# mysql_install_db
```

17.1.1. Назначение пароля суперпользователя

Учетные записи и пароли всех пользователей, которые имеют право работать с сервером, содержатся в таблице **user**. Сразу же после создания базы **mysql** в эту таблицу внесен только один пользователь — root. По

умолчанию он не имеет пароля. Этот пароль нужно установить немедленно: не нужно объяснять, как это важно для безопасности системы. Для изменения пароля запустите сервер командой:

```
# safe_mysqld &
```

Эта команда запустит сервер в режиме демона и освободит консоль. Если все пакеты были установлены правильно, вы увидите сообщение:

```
mysql: ready for connections
```

Затем введите команду:

```
$ mysql -u root mysql
```

Эта команда запускает клиент **MySQL** (MySQL-монитор) и соединяется с сервером от имени пользователя root, даже если вы работаете под другой учетной записью. Последний аргумент указывает базу данных, которую требуется открыть.

SQL-запросы можно набирать в строке приглашения MySQL-монитора, заканчивая ввод точкой с запятой или командой \g (go), а можно редактировать в текстовом редакторе, введя команду \e (edit). Список команд MySQL-монитора можно получить по команде \h (help).

Измените пароль суперпользователя с помощью следующего запроса:

```
UPDATE user SET Password=PASSWORD('новый_пароль') WHERE
user='root';
```

Как вы заметили, это обычный SQL-запрос, обновляющий поле Password таблицы **user** для пользователя root. При вводе запроса обратите внимание на регистр названий полей: сервер MySQL различает прописные и строчные буквы!

Теперь нужно, чтобы MySQL принял изменения. Для этого выполните еще один запрос SQL:

```
FLUSH PRIVILEGES;
```

Завершите сеанс работы с MySQL-монитором, введя команду \q (quit). В следующий раз зарегистрироваться на сервере без пароля вы уже не сможете. Теперь нужно запускать MySQL-монитор следующей командой:

```
$ mysql -u root -p mysql
```

Ключ -p запросит при регистрации пароль. Имейте в виду: забытый пароль нельзя восстановить. Единственный выход из этого положения — удалить каталог `/var/lib/mysql/mysql` и создать базу **mysql** заново командой **mysql_install_db**.

Для принятия изменений можно также использовать программу **mysqladmin** с аргументом reload. Вызвать программу можно так:

```
$ mysqladmin -p reload
```

Параметр -p вам обязательно нужно использовать, так как вы только что установили пароль для пользователя root.

Установите права доступа к сценарию /etc/rc.d/init.d/mysqld:

```
# chmod +x /etc/rc.d/init.d/mysqld
```

Теперь можете перезапустить сервер командой

```
# /etc/rc.d/init.d/mysqld restart
```

17.1.2. Автозапуск сервера MySQL

Последнее, что вам осталось сделать — это добавить сервер MySQL в автозапуск. С этой целью перейдите в каталог /etc/rc.d/rc3.d и создайте символическую ссылку на файл /etc/rc.d/init.d/mysql:

```
# ln -s S14mysql /etc/rc.d/init.d/mysql
```

Префикс S14 определяет очередность запуска сервера **mysqld**. В данном случае он запустится после сервисов network (S10) и portmap (S11). У вас эти значения могут быть другими.

17.1.3. Пользователи сервера MySQL и их права

После установки сервера нужно завести пользователей, которые имеют право работать с сервером баз данных. Введите следующий запрос:

```
GRANT ALL PRIVILEGES ON *.* TO admin@localhost IDENTIFIED
BY 'пароль' WITH GRANT OPTION;
```

Введенный вами запрос создаст пользователя admin, который будет иметь право выполнять любые операции со всеми базами данных. Этот пользователь будет иметь право подключаться к серверу с компьютера localhost, используя пароль.

Маска *.* определяет, к каким базам данных и таблицам имеет право подключаться тот или иной пользователь. Первая звездочка определяет базу, а вторая — таблицу. Если вам нужно, чтобы пользователь admin имел право подключаться с любого узла, используйте знак процента вместо имени компьютера:

```
GRANT ALL PRIVILEGES ON *.* TO admin@"%" IDENTIFIED BY
'пароль' WITH GRANT OPTION;
```

Вместо всех полномочий вы можете определить, какие действия может выполнять с базой тот или иной пользователь. Если вы являетесь хостинг-провайдером и предоставляете доступ пользователю к его базе данных, то вы можете использовать следующий запрос:

```
GRANT CREATE,DROP,SELECT,INSERT,UPDATE,DELETE,INDEX ON
user.* TO user@% IDENTIFIED BY 'пароль';
```

Этот запрос позволяет пользователю user выполнять все операции с его базой данных.

Полный список полномочий представлен в таблице 17.1.

Полномочия пользователей сервера MySQL *Таблица 17.1*

Полномочия	Описание
SELECT, INSERT, UPDATE, DELETE	Право просматривать, добавлять, модифицировать, удалять данные в таблицах базы данных
INDEX	Право производить операции с индексами таблиц
REFERENCES	Право работать со ссылками в базах данных и таблицах
CREATE, DROP	Право создавать и удалять таблицы и базы данных
GRANT, ALTER	Право определять полномочия
RELOAD, SHUTDOWN, PROCESS	Право перезагружать, останавливать сервер и просматривать все процессы (подключения)

Если запрос GRANT у вас не работает, то вы можете внести пользователя непосредственно в таблицу **user** базы данных **mysql**. Структура таблицы **user** выглядит следующим образом:

```
Host User Password Select_priv Insert_priv Update_priv
Delete_priv Create_priv Drop_priv Reload_priv Shutdown_priv
Process_priv File_priv
```

Поля **Host**, **User**, **Password** — это, соответственно, узел, из которого пользователь может получить доступ, имя и пароль пользователя. Все остальные поля задают полномочия. Если выполнение какой-нибудь операции разрешено пользователю, значение поля должно быть равным «Y», в противном случае — «N».

Например, нам нужно создать пользователя admin, который должен иметь все полномочия. Это можно сделать с помощью такого запроса SQL:

```
INSERT INTO user(Host,User,Password,Select_priv,Insert_
priv,Update_priv,Delete_priv, Create_priv, Drop_priv,
Reload_priv, Shutdown_priv, Process_priv, File_priv)
VALUES ('localhost', 'admin', password('4td56ls12'),
'Y', 'Y', 'Y', 'Y','Y','Y','Y','Y','Y','Y');
```

Для создания обыкновенного пользователя используйте следующий запрос:

```
INSERT INTO user(Host, User, Password, Select_priv,
Insert_priv, Update_priv, Delete_priv) VALUES('%',
'user', password('123456'),'Y','Y','Y','Y');
```

17.2. Клиентская часть MySQL

Удобной программой для просмотра структуры базы данных является **mysqlshow**. Введите следующую команду:

```
$ mysqlshow -p mysql
```

В ответ вы увидите список таблиц, которые находятся в базе данных **mysql**.

Программа **mysqlshow** может вызываться с дополнительными ключами, перечисленными в таблице 17.2.

Ключи программы mysqlshow *Таблица 17.2*

Ключ	Назначение
--host=имя_узла	Задает имя узла, где работает сервер MySQL, к которому вы хотите подключиться
--port=номер_порта	Определяет номер порта для сервера MySQL
--socket=сокет	Указывает сокет
--user=имя_пользователя	Указывает, под каким именем зарегистрироваться на сервере MySQL
-p	Запрашивает ввод пароля

Для самих же операций с данными используется программа **mysql**. Она понимает те же ключи, что и **mysqlshow**, и много других, среди которых очень полезный ключ **-s**. Я рекомендую вам всегда его использовать. Этот ключ подавляет большинство ненужных сообщений, выводимых MySQL-клиентом, что существенно повышает производительность на медленных линиях связи.

17.3. Установка PHP и настройка связки Apache+PHP+MySQL

Сейчас мы произведем не только установку PHP, которая не вызывает особых проблем, но и настройку связки Apache+PHP+MySQL. Эту связку, очень полезную при создании веб-проектов, можно настроить двумя способами. Первый — использование программ, которые входят в состав дистрибутива и, как правило, устанавливаются из пакетов RPM.

Второй способ заключается в загрузке последних версий Apache, MySQL и PHP и в самостоятельной их сборке из исходных текстов. Первый способ я могу порекомендовать начинающим пользователям, так как он проще. Если же вы чувствуете уверенность в своих силах, приступайте сразу к чтению второго способа.

17.3.1. Первый способ: из пакетов RPM

Первую часть связки в дистрибутивах Red Hat выше 7.2 и Mandrake выше 8.1 настраивать не нужно: все настраивается во время установки системы. Поэтому вы можете сразу приступить к тестированию связки Apache+PHP (листинг 17.1).

Убедитесь, что сервер **Apache** установлен и корректно функционирует:

```
$ lynx http://localhost
```

Текстовый браузер **lynx** должен отобразить стартовую страницу **Apache**. После успешной проверки работы сервера остановите его командой:

```
# /etc/init.d/httpd stop
```

Проверьте наличие библиотеки **gd** — она необходима для работы с графикой в PHP:

```
$ rpm -qa | grep gd
$ rpm -ihv gd-1.8.4-4.i386.rpm
```

Загрузить последнюю версию PHP 5 можно по адресу http://www.php.net/downloads.php.

Скачайте (`www.php.net/downloads.php`) и установите пакет php, если вы его еще не установили. Вам также понадобится пакет php-mysql, обеспечивающий поддержку сервера MySQL языком PHP, и модуль **Apache**, обеспечивающий поддержку PHP (пакет mod_php).

Затем в файле `httpd.conf` раскомментируйте следующую строчку. После этого файлы с расширением .php будут правильно обрабатываться сервером:

```
AddType application/x-httpd-php4 .php
```

Теперь можно проверять правильность настройки двух компонент связки: **Apache** и PHP.

Напомню, что в большинстве современных серверных дистрибутивов сервер **Apache** уже установлен вместе с модулем **mod_php**, поэтому выполнять вышеизложенные шаги совсем необязательно, достаточно только проверить корректность работы модуля mod_php.

Для проверки работы модуля **mod_php** создайте тестовый файл `test.php` с таким содержимым:

Листинг 17.1. Файл test.php

```
<?
phpinfo();
?>
```

Этот файл сохраните в каталоге **DocumentRoot** сервера **Apache**. Обычно это каталог /var/www/html. Затем запустите любой браузер и введите адрес http://localhost/test.php. Вы должны увидеть в окне браузера сведения о PHP, сервере **Apache** и других компонентах и библиотеках (рис. 17.1).

Функция **phpinfo()** очень информативна: внимательно изучив информацию, которую она предоставляет, вы много узнаете о своем веб-сервере.

Теперь немного настроим PHP. С помощью функции **phpinfo()** узнайте, где расположен инициализационный файл PHP. Обычно он называется `php.ini` и находится в каталоге `/etc`. Откройте этот файл в любом текстовом редакторе и раскомментируйте следующую строку, убедившись, что в вашей системе есть файл `mysql.so` (он устанавливается при установке php-mysql):

```
extension=mysql.so
```

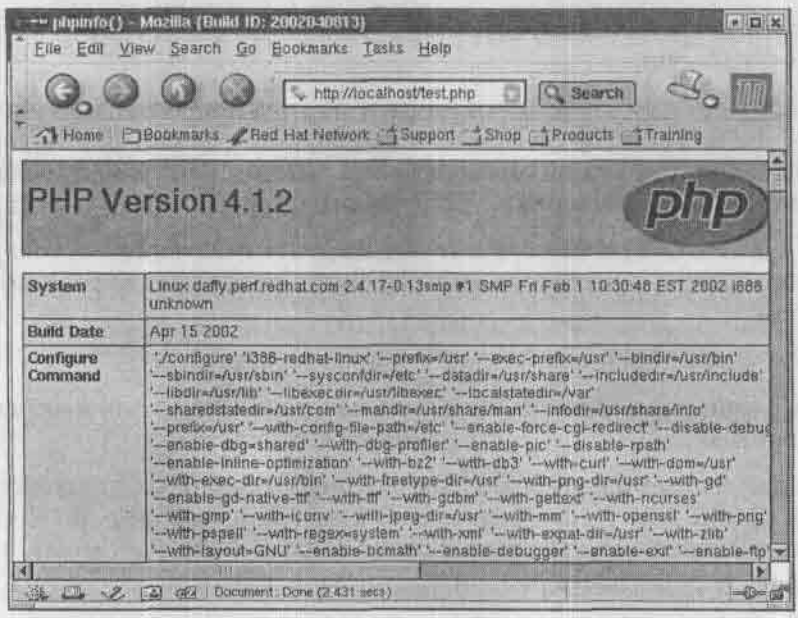


Рис. 17.1. *Функция phpinfo()*

После этого перейдите в секцию MySQL файла `php.ini` и установите параметры сервера MySQL по умолчанию:

```
mysql.default_port =
mysql.default_socket =
mysql.default_host =localhost
mysql.default_user =
mysql.default_password =
```

Эти параметры будут использоваться при установлении соединения с сервером, если в функциях PHP они не будут явно указаны. Никогда не указывайте пользователя root (а тем более его пароль) в качестве пользователя по умолчанию!

Теперь можно приступить к настройке сервера MySQL. Имеет смысл использовать версию MySQL не ниже 3.23, поскольку в этой версии появилась нормальная поддержка транзакций. Если вы устанавливаете MySQL версии 3.23 или выше, то установите еще пакет mod_auth_mysql, обеспечивающий базовую аутентификацию для сервера **Apache** с использованием таблиц MySQL.

При добавлении сервера MySQL в сценарии загрузки (/etc/rc.d/) обратите внимание на то, что сервер MySQL должен быть запущен ПЕРЕД сервером **Apache**.

17.3.2. Тестируем созданную конфигурацию

Теперь можно проверить работу всей связки Apache+PHP+MySQL. С этой целью создайте небольшой тестовый файл `mysql_test.php` в каталоге **DocumentRoot** (`/var/www/html`):

Листинг 17.2. Файл mysql_test.php

```
<?
// Используется имя пользователя root и пароль password
if(!mysql_connect("localhost","root","password"))
{
echo "Не могу соединиться с сервером\n";
echo mysql_error();
exit;
}
echo "Работает!"
?>
```

Как вы уже успели догадаться, если в окне браузера вы увидите слово «Работает!», значит, вы все сделали правильно.

17.3.3. Второй способ: из исходных текстов

У этого способа есть свои преимущества. Во-первых, у вас появится возможность использовать самые последние версии серверов Apache, MySQL и интерпретатора PHP, которых нет в составе даже самого нового дистрибутива Linux. Во-вторых, вы сами сможете контролировать процесс сборки и включать поддержку необходимых вам функций, исключив такую ситуацию, когда, например, разработчики пакетов RPM при сборке интерпретатора PHP забыли включить поддержку сервера MySQL. Мне попадался такой пакет php: функции **mysql_connect()** в нем просто не было.

Скачайте из Интернета последние версии **Apache**, **MySQL** и PHP. Предварительно удалите из системы старые версии, если они были установлены. После загрузки распакуйте исходные тексты в каталог /src.

Сначала установите сервер MySQL. С этой целью перейдите в каталог с исходными текстами MySQL и введите следующие команды (первая команда включает поддержку по умолчанию кодировки koi8-r):

```
# ./configure --with-charset=koi8_ru
# make
# make install
```

Затем аналогично установите **Apache**. Для получения информации обо всех возможных ключах сценария configure введите команду `configure --help`.

После этого распакуйте PHP и соберите его следующим образом:

```
# ./configure --with-mysql --with-apache=../apache_2.0.0
--with-mod_charset
# make
# make install
```

Первая команда конфигурирует интерпретатор PHP для работы с сервером баз данных MySQL и веб-сервером Apache. Естественно, вы должны правильно указать путь к исходным текстам **Apache** с помощью ключа --with-apache.

Затем вернитесь в каталог, содержащий исходные тексты **Apache**, и введите команду:

```
# ./configure --activate-module=src/modules/php4/libphp4.a
```

Перед этим нужно убедиться в существовании файла libphp4.a (если php собрался успешно, этот файл должен существовать). Если сценарий configure успешно завершил свою работу, введите команды `make` и `make install`.

Проверить, подключился ли модуль libphp, вы можете после установки сервера с помощью команды:

```
# httpd -l
```

В списке модулей должен быть модуль libphp4.c, а также модуль mod_charset.c — его вы подключили при первой сборке. После этого можно отредактировать файл `/etc/php.ini` и установить пароль для пользователя root сервера MySQL (не путайте пользователя root всей системы с пользователем root сервера MySQL!).

Теперь только остается добавить запуск серверов в сценарии автозагрузки системы. Напомню, что сервер MySQL должен запускаться до сервера **Apache**.

17.4. Защита сервера MySQL

По умолчанию для файла сокета `mysql.sock`, который используется соединениями сервера MySQL, установлены права доступа 0777. Это означает, что кто угодно может удалить этот файл. Если данный файл будет удален во время работы сервера, то ни один пользователь не сможет подключиться к серверу MySQL. Установите «бит прилипчивости» (sticky-бит) для каталога `/var/lib/mysql`, чтобы предотвратить удаление файлов из этого каталога:

```
# chmod +t /var/lib/mysql
```

Из соображений безопасности рекомендуется удалить базу данных **test**, которая создается при установке сервера и используется для его проверки:

```
# mysqladmin drop test -p
```

Не забудьте также удалить запись, соответствующую базе данных test, из таблицы **db** базы данных **mysql**:

```
# mysql -u root mysql -p
mysql> DELETE * FROM Db WHERE Db="test";
mysql> DELETE * FROM Db WHERE Db="test\_%";
mysql> quit
```

17.5. Введение в язык SQL

17.5.1. Общие понятия

Если вы раньше работали с какой-нибудь СУБД (Система Управления Базой Данных), вы смело можете пропустить этот пункт — ничего нового для себя вы не прочитаете. Но если вы сталкиваетесь с СУБД впервые, без основных терминов вам не обойтись.

- **Поле** (field) — это неделимый элемент данных в БД. Поле имеет имя и тип. Подробнее о типах полей мы поговорим чуть позже.
- **Запись** (record) — набор полей, содержащих связанную информацию. Например, запись с полями C_No, C_Name и C_Address содержит информацию о клиенте — его номер, имя и адрес.
- **Таблица** (table) — это набор записей одинаковой структуры. Если у нас есть запись структуры C_No, C_Name, C_Address, то все записи в таблице **Clients** будут иметь такую структуру.
- **База данных** (database) — это совокупность связанных таблиц. Например, в одной таблице может храниться информация о клиенте, а в другой — информация о заказе, который сделал клиент.
- **Индекс** (index) — используется для быстрого поиска нужной записи в базе данных. Обычно поиск производится по значению одного поля или по значению нескольких полей.
- **Первичный индекс** (index) — управляет порядком отображения записей в таблице. Поле первичного индекса должно быть уникальным, то есть в одной таблице не должно быть двух записей, в которых это поле принимает одно и то же значение. В нашей таблице **Clients**, очевидно, первичный индекс должен строиться по полю C_No — по номеру клиента, который должен быть уникален.
- **Вторичный индекс** (secondary index) — в отличие от первичного индекса может строиться по нескольким полям и не обязан быть уникальным. Вторичные индексы используются для связывания таблиц. Индексы также называются ключами.
- **Запрос** (query) — оператор, выбирающий записи и поля, удовлетворяющий заданному условию, из одной или нескольких таблиц.

17.5.2. Краткий практический курс SQL

Как работает сервер SQL? Клиент посылает запрос, в котором указывает, какую информацию хочет получить от сервера или какую операцию с данными собирается выполнить. В ответ сервер посылает клиенту ответ, в котором указывает, выполнил ли сервер его запрос, и, если выполнил, сообщает результат запроса.

Для описания запросов клиента был разработан целый язык — SQL (Structured Query Language, Структурированный язык запросов). С помощью запросов SQL вы можете:

- Создавать базы данных и таблицы.
- Добавлять информацию в таблицы.
- Удалять информацию.
- Модифицировать информацию.
- Получать нужные вам данные.

В этой книге мы не будем подробно рассматривать язык SQL — ему посвящены отдельные книги, в два раза толще этой, в которых описываются различные варианты языка SQL.

Перед началом работы с SQL вам нужно интуитивно понимать, что такое база данных. Если вы имеете представление о ней, можете смело пропустить этот абзац. База данных состоит из таблиц, как книга MS Excel состоит из листов. Каждая таблица состоит из записей, а каждая запись — из полей. Каждое поле имеет свой домен, то есть тип данных, которые можно записать в это поле. Поле типа INT может содержать только целые числа, а поле типа CHAR — строки.

Вот теперь можно приступать к созданию новой базы данных. Для создания баз данных и таблиц в языке SQL обычно используется запрос CREATE. В случае с MySQL для создания базы нужно использовать программу **mysqladmin**:

```
$ mysqladmin -u admin -p create sklad
```

Естественно, пользователь admin должен существовать и обладать соответствующими правами. Откройте созданную базу:

```
$ mysql -u admin -p sklad
```

Каждый запрос MySQL должен заканчиваться точкой с запятой. Если вы введете SELECT * FROM test, клиент **mysql** будет ждать ввода точки с запятой:

```
->
```

Давайте договоримся, что будем писать запросы согласно стандарту SQL, то есть для улучшения восприятия будем разбивать их на части. Программа **mysql** допускает ввод запроса во всю строчку. Например, запрос, записанный в стандарте SQL,

```
SELECT *
FROM S
WHERE Q > 10
```

в программе mysql можно записать так:

```
SELECT * FROM S WHERE Q > 10
```

Теперь создадим три таблицы — Товар, Клиенты и Заказы.

```
CREATE TABLE CLIENTS
(
C_NO        int         NOT NULL,
FIO         char(40)    NOT NULL,
ADDRESS     char(30)    NOT NULL,
CITY        char(15)    NOT NULL,
PHONE       char(11)    NOT NULL
);
```

Таблица CLIENTS содержит поля C_NO (номер клиента), FIO (Фамилия, Имя, Отчество), Адрес, Город и Телефон. Все эти поля не могут содержать пустого значения (NOT NULL).

Большинство серверов не требуют явного указания NOT NULL, но при этом значение по умолчанию может быть разным: одни сервер инициализируют столбцы значением NULL, а другие — NOT NULL. Поэтому лучше явно указать NOT NULL.

```
CREATE TABLE TOVAR
(
T_NO     int      NOT NULL,
DSEC     char(40)  NOT NULL,
PRICE    numeric(9,2)   NOT NULL,
QTY      numeric(9,2)   NOT NULL
);
```

Эта таблица будет содержать данные о товарах. Тип numeric (9,2) означает, что 9 знаков отводятся под целую часть и два — под дробную. QTY — это количество товара на складе.

```
CREATE TABLE ORDERS
(
O_NO     int      NOT NULL,
DATE     date     NOT NULL,
C_NO     int      NOT NULL,
T_NO     int      NOT NULL,
QUANTITY   numeric(9,2)  NOT NULL,
AMOUNT  numeric(9,2)   NOT NULL
);
```

Эта таблица содержит сведения о заказах — номер заказа (O_NO), дату заказа (DATE), номер клиента (C_NO), номер товара (T_NO), количество (QUANTITY) и стоимость заказа AMOUNT.

Теперь добавим данные в наши таблицы. Добавить данные можно с помощью оператора INSERT:

```
INSERT INTO CLIENTS
VALUES (1,'Иванов И.П.', 'Ленина 6', 'Кировоград','80522111111');
```

Добавляемые значения должны соответствовать тому порядку, в котором поля перечислены в операторе CREATE. Если вы хотите добавлять информацию в другом порядке, то вы должны указать этот порядок в операторе INSERT:

```
INSERT INTO CLIENTS (FIO,ADDRESS,C_NO,PHONE,CITY)
VALUES ('Петров', 'Пушкина 9',2,'-','Кировоград');
```

С помощью INSERT мы можем устанавливать значения только некоторых полей:

```
INSERT INTO CLIENTS (C_NO, FIO)
VALUES (1,'Петров');
```

В нашем примере этот запрос выполнен не будет, поскольку все остальные поля равны NULL (пустое значение), а наша таблица пустых значений не допускает.

Добавим данные в таблицу TOVAR:

```
INSERT INTO TOVAR
VALUES (1,'Монитор LG',550.74);
```

Обратите внимание, что мы пока еще не указали первичные ключи таблицы, поэтому нам никто не мешает добавить в таблицу одинаковые записи.

Добавить дату в поле DATE можно с помощью функции TO_DATE:

```
INSERT INTO ORDERS
VALUES (1,TO_DATE('01/01/02,'DD/MM/YY'),1,1,1,550.74);
```

Данная запись означает, что первого января 2002 года Иванов И.П. (C_NO=1) заказал один (QUANTITY=1) монитор LG (T_NO=1).

Предположим, что нам нужно обновить запись, например, клиент Иванов переехал в другой город. Это делается так:

```
UPDATE CLIENTS
SET CITY = 'Киев'
WHERE C_NO = 1;
```

Теперь удалим всех клиентов, номера которых превышают 10:

```
DELETE FROM CLIENTS
WHERE C_NO > 10;
```

Если вторая часть запроса DELETE — WHERE — не указана, значит, действие оператора распространяется на все записи указанной таблицы.

Добавление, изменение и удаление записей — это, безусловно, очень важные команды, но чаще всего вы будете использовать запрос SELECT, который выбирает из таблицы данные, удовлетворяющие условию.

Например, для вывода всех записей из таблицы CLIENTS, введите:

```
SELECT * FROM CLIENTS;
```

В результате вы получите такой ответ от сервера:

```
C_NO   FIO          ADDRESS      CITY        PHONE
1      Иванов И.П.  Ленина 6     Кировоград  80522111111
1      Иванов И.П.  Ленина 6     Кировоград  80522111111
2      Петров В.К.  Пушкина 9    Кировоград  80522112111
```

Обратите внимание на первые две записи — они одинаковые. Теоретически добавление одинаковых записей возможно — мы ведь не указали первичный ключ таблицы. Если вы хотите исключить одинаковые записи из ответа сервера (но не из таблицы!), введите запрос:

```
SELECT DISTINCT *
FROM CLIENTS;
```

Если вы хотите вывести только некоторые поля, то запрос должен выглядеть так:

```
SELECT DISTINCT FIO, PHONE
FROM CLIENTS;
```

Теперь займемся усложнением наших запросов. Выведем все товары, цена которых превышает 500 рублей.

```
SELECT *
FROM TOVAR
WHERE PRICE > 500;
```

Вы можете использовать другие операторы отношений: <,>,=,<>, >=,<=.

Если ваша компания обслуживает несколько однофамильцев и вы хотите вывести информацию обо всех Ивановых, используйте шаблон LIKE:

```
SELECT *
FROM CLIENTS
WHERE FIO LIKE '%Иванов%';
```

Запрос читается так: вывести всю информацию о клиентах, фамилия которых похожа на 'Иванов'.

Следующие два оператора эквивалентны:

```
SELECT *
FROM TOVAR
WHERE (PRICE > 100) AND (PRICE < 200);
```

и оператор

```
SELECT *
FROM TOVAR
WHERE PRICE BETWEEN 100 AND 200;
```

Если вы хотите выбрать данные из разных таблиц, перед именем поля нужно указывать имя таблицы. Следующий запрос выведет имена всех клиентов, которые хотя бы раз покупали у нас товар:

```
SELECT DISTINCT CLIENTS.FIO
FROM CLIENTS, ORDERS
WHERE CLIENTS.C_NO = ODREDS.C_NO;
```

При работе с оператором SELECT вам доступно несколько полезных функций, вычисляющих количество элементов (COUNT), сумму элементов (SUM), максимальное и минимальное значение (MAX и MIN), а также среднее значение (AVG).

Следующие операторы выведут, соответственно, количество записей в таблице CLIENTS и самый дорогой товар на складе:

```
SELECT COUNT(*)
FROM CLIENTS;
SELECT MAX(PRICE)
FROM TOVAR;
```

Оператор SELECT позволяет группировать возвращаемые значения. Например, клиент Иванов (C_NO=1) несколько раз заказывал у нас какой-то товар. Значит, его номер встречается в таблице ORDERS несколько раз.

Выведем имена всех клиентов, а также сумму заказа каждого клиента.

```
SELECT CLIENTS.FIO, SUM(ORDERS.AMOUNT) AS TOTALSUM
FROM CLIENTS, ORDERS
WHERE CLIENTS.C_NO = ORDERS.C_NO
GROUP BY ORDERS.C_NO;
```

Группировку выполняет оператор GROUP BY, который является частью оператора SELECT. Оператор GROUP BY можно ограничить с помощью HAVING. Этот оператор используется для отбора строк, возвращаемых GROUP BY. HAVING можно считать аналогом WHERE, но только для GROUP BY:

```
HAVING <условие>
```

Например, нас интересуют только клиенты, которые заказали товаров на общую сумму, превышающую 1000.

```
SELECT CLIENTS.FIO, SUM(ORDERS.AMOUNT) AS TOTALSUM
FROM CLIENTS, ORDERS
WHERE CLIENTS.C_NO = ORDERS.C_NO
GROUP BY ORDERS.C_NO
HAVING TOTALSUM > 1000;
```

Примечание

В этом запросе мы использовали псевдоним столбца TOTALSUM. В некоторых вариантах SQL для определения псевдонима не нужно писать служебное слово AS, а другие требуют применение знака равенства:

`SUM(ORDERS.AMOUNT) TOTALSUM` или
`TOTALSUM = SUM(ORDERS.AMOUNT.`

Пока мы не установили первичный ключ, сортировка нашей таблицы не выполняется. Записи будут отображены в порядке их занесения в таблицу. Для сортировки по полю C_NO результата вывода таблицы CLIENTS используется следующий запрос (сама таблица при этом не сортируется):

```
SELECT *
FROM CLIENTS
ORDER BY C_NO;
```

Предположим, что кто-то добавил в таблицу CLIENTS запись

```
1  Сидоров  Егорова 11  Кировоград80522345111
```

У нас получилось, что один и тот же номер сопоставлен разным клиентам. Тогда как определить, кто из них заказал монитор LG? Чтобы избежать подобной путаницы, нужно использовать первичные ключи:

```
ALTER TABLE CUSTOMER
ADD PRIMARY KEY (C_NO);
```

После этого запроса поле C_NO может содержать только уникальные значения. В качестве первичного ключа нельзя использовать поле, допускающее значение NULL.

Создать первичный ключ можно проще — при создании таблицы. Это делается так:

```
CREATE TABLE CLIENTS
(
```

```
C_NO    int     NOT NULL,
FIO   char(40)  NOT NULL,
ADDRESS char(30)  NOT NULL,
CITY    char(15)  NOT NULL,
PHONE char(11)  NOT NULL,
PRIMARY KEY (C_NO);
);
```

Таблица ORDERS содержит сведения о заказах. По полю C_NO этой таблицы идентифицируется заказчик. Предположим, что в таблицу ORDERS кто-то ввел значение, которого нет в таблице CLIENTS. Кто же заказал товар? Нам нужно не допустить подобной ситуации, поэтому введите следующий запрос на создание внешнего ключа:

```
ALTER TABLE ORDERS
ADD FOREIGN KEY(C_NO) REFERENCES CLIENTS;
```

Введенные в таблицу ORDERS номера клиентов C_NO должны существовать в таблице CLIENTS. Аналогично нужно добавить внешний ключ по полю T_NO. Эта возможность называется декларативной целостностью.

Команда ALTER используется не только для добавления ключей. Она предназначена для реорганизации таблицы в целом. Вы хотите добавить еще одно поле? Или установить список допустимых значений для каждого из полей? Все это можно сделать с помощью команды ALTER:

```
ALTER TABLE CLIENTS
ADD ZIP char(6) NULL;
```

Этот запрос добавляет в таблицу CLIENTS новое поле ZIP типа char. Обратите внимание, что вы не можете добавить новое поле со значением NOT NULL в таблицу, в которой уже есть данные.

Наша компания работает с клиентами только из Киева и Кировограда, поэтому целесообразно ввести список допустимых значений для таблицы CLIENTS:

```
ALTER TABLE CLIENTS
ADD CONSTRAINT INVALID_STATE SHECK (CITY IN
('Кировоград','Киев'));
```

Вам уже надоело работать с этой базой данных? Тогда с помощью запроса DISCONNECT отключитесь от нее, и, используя запрос CONNECT, подключитесь к другой базе данных. В некоторых вариантах SQL запрос DISCONNECT не работает, а вместо CONNECT нужно использовать оператор USE.

Теперь, когда вы уже знакомы с основами SQL, немного углубимся. Мы уже знаем, как добавлять первичный ключ, теперь добавим внешний ключ при создании таблицы:

```
CREATE TABLE T
(
/* Описания полей таблицы */
FOREIGN KEY KEY_NAME (LIST)
REFERENCES ANOTHER_TABLE [(LIST2)]
[ON DELETE OPTION]
[ON UPDATE OPTION]
);
```

Здесь KEY_NAME — это имя ключа. Оно не является обязательным, но я очень рекомендую его указывать: если вы не укажете имя ключа, то потом не сможете его удалить.

LIST — это список полей, входящих во внешний ключ. Список разделяется запятыми.

ANOTHER_TABLE — это другая таблица, по которой устанавливается внешний ключ, а необязательный элемент LIST2 — это список полей этой другой таблицы. Типы полей в списке LIST должны совпадать с типами полей в списке LIST2.

Необязательные параметры ON DELETE и ON UPDATE определяют действия, которые нужно произвести при удалении информации из таблицы и при ее обновлении. Например, нельзя так просто удалить клиента из таблицы клиентов, если в таблице заказов присутствуют записи его заказов: нарушится целостность базы. С помощью параметра ON DELETE мы можем указать серверу реакцию на удаление таких данных:

```
ON DELETE OPTION
```

Параметр OPTION может принимать одно их четырех значений: CASCADE, NO ACTION, SET DEFAULT, SET NULL.

Значение CASCADE означает, что номер удаляемого клиента будет удален из всех связанных таблиц. Например, если вы удалите клиента с номером 10 из таблицы клиентов, то из таблицы заказов будут удалены все заказы этого клиента.

Опция NO ACTION не разрешает удаление клиента до тех пор, пока его номер присутствует в связанной таблице. В нашем случае это означает, что сначала должны быть удалены все заказы клиента с номером 10.

С помощью опции SET_DEFAULT вы можете указать значение по умолчанию. Например, если вы укажете SET DEFAULT 1, то при удалении клиента с любым номером его заказы будут приписываться клиенту с номером 1, который, разумеется, всегда есть в таблице CLIENTS.

Опция SET NULL устанавливает значение NULL в качестве номера клиента, если тот удален из таблицы CLIENTS. В нашем примере это значение не допускается.

Две страницы назад мы добавили поле ZIP. А как его удалить? Стандартом SQL не предусмотрено удаление столбцов, но в MySQL мы все же можем это сделать:

```
ALTER TABLE CLIENTS
DROP ZIP;
```

Как удалить все записи? Очень просто:

```
DELETE *
FROM ORDERS;
```

Удалить таблицу еще проще:

```
DROP ORDERS;
```

В первом случае вы не удаляете таблицу: файл таблицы все еще остается на диске. Вы удалили только содержимое таблицы. Во втором случае вы полностью удаляете таблицу.

Естественно, удалить таблицу можно только при наличии соответствующих прав.

Напоследок рассмотрим два полезных примера. Предположим, что нам нужно установить ограничение на количество товара, которое можно продать клиенту. Допустим, в данный момент у нас нет такого количества товара на складе, следовательно, мы не можем оформить заказ. Ограничение данного типа можно определить с помощью запроса:

```
CREATE ASSERTION LIMIT
CHECK (ORDERS.QUANTITY <= TOVARS.QTY);
```

Установить минимальное количество для заказа можно так:

```
CREATE TABLE ORDERS
(
/* Определение полей */
FOREIGN KEY KEY1 (C_NO)
REFERENCES CLIENTS
ON DELETE NO ACTION
CHECK (QUANTITY >= 1)
);
```

Следующий запрос устанавливает минимальный размер заказа, если таблица уже существует:

```
CREATE ASSERTION LIMIT
CHECK (QUANTITY >= 1);
```

Глава 18

ПРОКСИ-СЕРВЕРЫ. SQUID И SOCKS

- ЧТО ТАКОЕ ПРОКСИ-СЕРВЕР?
- УСТАНОВКА SQUID
- НАСТРОЙКА SQUID
- ЗАПУСК SQUID
- СПИСКИ ACL
- ОТКАЗ ОТ РЕКЛАМЫ. БАННЕРНЫЙ ФИЛЬТР
- РАЗДЕЛЕНИЕ КАНАЛА С ПОМОЩЬЮ SQUID
- НАСТРОЙКА ПОДДЕРЖКИ ПРОКСИ У КЛИЕНТОВ
- ТЕХНОЛОГИЯ SOCKS5, ИЛИ КАК ИСПОЛЬЗОВАТЬ АСЬКУ ИЗ ЛОКАЛЬНОЙ СЕТИ

18.1. Что такое прокси-сервер?

Прокси-сервер (сервер-посредник) — это программа, которая выполняет HTTP/FTP-запросы от имени клиентов. Применение прокси-сервера дает возможность использовать фиктивные IP-адреса во внутренней сети (IP-маскарадинг), увеличивает скорость обработки запроса при повторном обращении (кэширование), а также обеспечивает дополнительную безопасность.

Нет смысла устанавливать прокси на своей домашней машине, так как функции кэширования выполняет сам браузер. Прокси-сервер стоит применять лишь в том случае, если в вашей сети есть несколько компьютеров, которым нужен выход в Интернет. Если один из них уже запрашивал какой-то интернет-ресурс, то следующий пользователь, обратившийся за этим же ресурсом, получит ответ не из Интернета, а из кэша прокси-сервера, то есть значительно быстрее.

SQUID — это нечто большее, чем просто посредник. Это своеобразный стандарт кэширования информации в сети Интернет. В силу повсеместной распространенности SQUID, в книге я уделил его конфигурированию большое внимание.

Прокси-сервер SQUID образуют несколько программ, в числе которых сам демон **squid**, а также программа **dnsserver** — программа обработки DNS-запросов. Когда запускается **squid**, то сначала он запускает заданное количество процессов **dnsserver**, каждый из которых работает самостоятельно и может осуществлять только один поиск в системе DNS. За счет этого уменьшается общее время ожидания ответа DNS.

18.2. Установка SQUID

Я использую версию **squid** 2.5. Пакет squid входит в состав современных дистрибутивов, а если его у вас почему-то нет, то скачать можно с `www.squid-cache.org`.

При сборке SQUID из исходных кодов первым шагом должна быть команда

```
# ./configure --prefix=/usr/local/squid
```

SQUID будет установлен в каталог, заданный ключом prefix. Другие ключи сценария configure перечислены в таблице 18.1.

Ключи сценария configure *Таблица 18.1*

Ключ	Назначение
--enable-icmp	Измерять путь до каждого HTTP-сервера при запросах с помощью ICMP
--enable-snmp	Включить SNMP-мониторинг
--enable-delay-pools	Управление трафиком
--disable-wccp	Отключить Web Cache Coordination Protocol
--enable-kill-parent-hack	Более корректный shutdown
--enable-splaytree	Позволяет увеличить скорость обработки ACL

18.3. Настройка SQUID

Сервер SQUID использует файл конфигурации `squid.conf`, который обычно располагается в каталоге `/etc/squid` (или `/usr/local/squid/etc` — более ранние версии). Подробнее на отдельных настройках мы остановимся чуть позже. А сейчас просто по шагам произведем настройку SQUID. Отредактируйте в этом файле следующие строки.

Укажите прокси-провайдера (тот сервер, который станет вашим «соседом» (neighbour, peer)):

```
cache_peer proxy.isp.ru
```

Установите объем памяти, разрешенный для кэша **squid**, в байтах, и каталог для дискового кэша:

```
cache_mem  65536
cache_dir  ufs  /usr/local/squid/cache 1024 16 256
```

где 1024 — количество мегабайтов, отводимое под кэш в указанном каталоге. В этом каталоге будут храниться кэшированные файлы. Стоит ли говорить, что если у вас несколько жестких дисков, то кэш нужно разместить на самом быстром из них.

Укажите узлы, которым разрешен доступ к прокси-серверу:

```
acl allowed_hosts src 192.168.1.0/255.255.255.0
acl localhost src 127.0.0.1/255.255.255.255
```

Укажите разрешенные SSL-порты:

```
acl SSL_ports port 443 563
```

Запретите метод CONNECT для всех портов, кроме указанных в acl SSL_ports:

```
http_access deny CONNECT !SSL_ports
```

Запретите доступ всем, кроме тех, кому можно:

```
http_access allow localhost
http_access allow allowed_hosts
http_access allow SSL_ports http_access deny all
```

Пропишите пользователей, которым разрешено пользоваться SQUID (в рассматриваемом примере это den, admin и developer):

```
ident_lookup on
acl allowed_users user den admin developer
http_access allow allowed_users
http_access deny all
```

Тэги maximum_object_size и maximum_object устанавливают ограничения на размер передаваемых объектов.

Ниже приведен пример запрета доступа к любому URL, который соответствует шаблону games, и разрешения доступа ко всем остальным:

```
acl GaMS url_regex games
http_access deny GaMS
http_access allow all
```

18.4. Запуск SQUID

Первый раз **squid** нужно запускать с ключом –z, чтобы создать и очистить каталог кэша:

```
# /usr/local/squid/bin/squid -z
```

Еще несколько полезных ключей, с которыми можно запускать **squid**, перечислены в таблице 18.2.

Ключи запуска **squid** *Таблица 18.2*

Ключ	Назначение
-a порт	Задает альтернативный порт для входящих HTTP-запросов
-d	Включает режим вывода отладочной информации в стандартный поток ошибок
-f файл	Задает альтернативный файл конфигурации, который должен будет использоваться вместо стандартного squid.conf
-h	Выдает справочную информацию
-k reconfigure	Посылает сигнал HUP, что приводит к тому, что SQUID заново прочитает свой конфигурационный файл
-k shutdown	Завершение работы прокси-сервера. При этом он подождет, пока будут завершены все соединения
-k interrupt	Немедленно завершить работу прокси-сервера, без ожидания завершения соединений
-k kill	Завершение без закрытия журналов
-u порт	Задает альтернативный порт для входящих ICP-запросов
-s	Включает журналирование с помощью syslog
-v	Выдает информацию о версии SQUID
-z	Очищает каталог кэша
-D	Запрещает DNS-тест при запуске
-N	Запрещает становиться демоном (фоновым процессом)
-Y	Более быстрое восстановление после сбоев

18.5. Расширенные настройки SQUID. Конфигурационный файл squid.conf

18.5.1. Параметры сети

В файле squid.conf могут быть заданы следующие параметры сети:

- **http_port** — порт для запросов клиентов. С этого порта прокси-сервер будет ожидать и обрабатывать запросы клиентов. Значение по умолчанию равно 3128;
- **icp_port** — порт для общения с соседями через ICP. Если «соседей» (peer) нет, то установите **icp_port** 0. По умолчанию используется значение 3130. При использовании этого параметра нужно установить ключ --enable-htcp для директивы htcp_port 4827;
- **tcp_outgoing_address** — при отправлении информации указанный адрес будет использован в качестве исходного. Значение по умолчанию: tcp_outgoing_address 255.255.255.255;
- **udp_outgoing_address** — то же самое, что и предыдущая директива — но только для ICP. Значение по умолчанию: udp_outgoing_address 255.255.255.255 . То же, но для ICP при приеме — директива udp_incoming_address со значением по умолчанию 0.0.0.0;

- **passive_ftp** on | off — по умолчанию этот режим включен, но если прокси-сервер находится за брандмауэром, то параметр passive_ftp нужно выключить.

18.5.2. Параметры соседей

«Соседи» — это другие кэширующие серверы, в кэшах которых SQUID ищет запрошенный ресурс перед тем, как обратиться к нему напрямую. Так, SQUID-сервер локальной сети может обратиться к серверу провайдера, региона и т.д. в расчете на то, что чем больше пользователей, тем больше шанс найти копию запрошенных данных ближе, чем по оригинальному адресу. Существует два типа «соседей»:

- parent (старший): если запрошенных данных не оказалось в кэше у parent, тот пересылает запрос дальше и возвращает подчиненному готовый ответ. Если SQUID получает отказ (TCP_DENIED) от parent, то обращается к ресурсу напрямую;
- sibling (равный): если запрошенных данных не оказалось в кэше у sibling, то он просто возвращает сообщение об этом, не предпринимая никаких дальнейших действий.

Каждый «сосед» прописывается отдельной строкой следующего формата:

```
cache_peer hostname type proxy-port icp-port options
```

где: hostname — имя узла-«соседа»;
type — тип соседа: parent — старший, sibling — одного уровня;
proxy-port — порт прокси-сервера;
icp-port — порт ICP;
options — параметры.

18.5.3. Управление кэшем

За управление кэшем отвечают следующие директивы:

- **cache_mem** <число> — задает размер оперативной памяти, отводимой под кэш. Размер этот указывается в байтах, килобайтах, мегабайтах (MB) или гигабайтах (GB). По умолчанию используется значение 8 MB;
- **cache_dir** <тип> <каталог> <размер> <1уровня_кат> <2уровня_кат> — задает местоположение кэша на диске и его параметры:
- тип — тип хранения. Практически всегда используется значение ufs;
- каталог — задает имя каталога, в котором будет храниться кэш;
- размер — размер (в мегабайтах) отводимого под кэш пространства на жестком диске;
- 1уровня_кат — максимальное число подкаталогов 1 уровня, которое может быть в указанном каталоге кэша;

- 2уровня_кат — максимальное количество подкаталогов, которое может быть в каждом из подкаталогов 1 уровня.

Значение по умолчанию: cache_dir ufs /usr/local/squid/cache 100 16 256. Допускается использование нескольких записей с директивой cache_dir для определения нескольких каталогов для размещения кэша;

- **cache_swap_high** <число> — процент заполнения кэша, по достижении которого начинается ускоренный процесс удаления старых объектов. Значение по умолчанию равно 95;
- **cache_swap_low** <число> — процент заполнения кэша, по достижении которого прекращается удаление старых объектов. Значение по умолчанию равно 90;
- **maximum_object_size** <число> KB — максимальный размер кэшируемого объекта. Значение по умолчанию равно 4096 KB;
- **minimum_object_size** — файлы меньшего размера не кэшируются. Значение по умолчанию: 0 KB.

18.5.4. Протоколирование

Ниже перечислены режимы протоколирования SQUID с указанием соответствующих журналов. Если какой-то журнал вам не нужен, установите none вместо имени файла.

- **cache_access_log** /usr/local/squid/logs/access.log — протоколирование запросов к SQUID;
- **cache_log** /usr/local/squid/logs/cache.log — протоколирование запусков процессов;
- **cache_store_log** /usr/local/squid/logs/store.log — протоколирование записи объектов в кэш.

18.5.5. Параметры внешних программ

В конфигурационном файле `squid.conf` могут быть заданы следующие параметры внешних программ и сервисов:

- **ftp_user** email-адрес — этот email-адрес будет использоваться вместо пароля при анонимном доступе к ftp-серверам;
- **dns_nameservers** список IP-адресов — этот список используется вместо того списка DNS-серверов, который определен в файле `/etc/resolv.conf`; значение по умолчанию — none;
- **cache_dns_program** /usr/local/squid/bin/dnsserver — указывает программу разрешения имен (сервер DNS);
- **authenticate_program** none — позволяет производить аутентификацию клиентов, делающих запросы. При этом должен быть определен ACL proxy_auth;

- **authenticate_program** /usr/local/squid/bin/ncsa_auth /usr/local/squid/etc/passwd — традиционная программа аутентификации. Определена в ../auth_modules/NCSA.

18.5.6. Параметры администрирования

Параметры администрирования, которые можно задать в файле `squid.conf`, таковы:

- **cache_mgr email** — почтовый адрес, на который будет послано письмо, если SQUID перестанет функционировать;
- **cache_effective_user** nobody — при запуске SQUID от имени root изменить UID на указанный в параметре cache_effective_user;
- **cache_effective_group** nogroup — при запуске SQUID от имени root изменить GID на указанный в параметре cache_effective_group;
- **visible_hostname** имя_узла — это имя будет упоминаться в сообщениях об ошибках;
- **hostname_aliases** имя — этот параметр задает список синонимов для имени узла.

18.6. Списки ACL

ACL (Access Control Lists) — списки контроля доступа. Довольно часто возникает необходимость группировки однотипных параметров в единое целое для их последующей обработки. Для эффективного решения этой задачи используются списки ACL. Например:

```
acl SSL_ports port 443 563
```

Эта запись означает, что создается список SSL_ports типа port. Элементами списка являются номера портов 443 и 563.

Добавить новый элемент к уже существующему списку можно так:

```
acl add SSL_ports port 999
```

Удалить ненужный элемент можно с помощью операции del:

```
acl del SSL_ports 999
```

Переименовать список позволяет операция ren:

```
acl ren SSL_ports Allowed_ports
```

Удалить все списки вместе с их содержимым позволяет операция flush:

```
acl flush
```

Стандарт ACL требует, чтобы перед именем списка обязательно был указан символ $. Строго говоря, все перечисленные выше примеры без этого символа неправильны. Однако большинство фильтров, например SQUID, пренебрегают этим требованием, и вы можете указывать имена списков без знака доллара.

Итак, ACL — это определение списка доступа, имеющее следующий формат:

```
acl <имя> <тип> <регулярное_выражение>
```

Типы, которые можно использовать при составлении списков ACL, перечислены в таблице 18.3.

Типы ACL *Таблица 18.3*

Тип	Назначение
Src IP-адрес/маска	IP-адрес клиентов
Src IP1-IP2/маска	Диапазон адресов
Dst IP-адрес/маска	URL узлов
Time [день] [Ч1:М1-Ч2:М2]	Время, где день — это одна буква из SMTWHFA
Port	Список портов
Port port1-port2	Диапазон портов
Proto	Протокол — HTTP или FTP
Method	Метод — GET или POST
Browser [-i] рег_выражение	Заголовок браузера клиента. [-i] — игнорируется регистр букв

18.6.1. Параметры доступа

Параметры доступа в файле `squid.conf` задаются следующими директивами:

- **http_access** allow | deny aclname — разрешать доступ к прокси по HTTP;
- **icp_access** allow | deny aclname — разрешать доступ к прокси по ICP;
- **miss_access** allow | deny aclname — разрешать получать ответ MISS («не найден») от вас;
- **cache_peer_access** cache-host allow | deny aclname — ограничить запросы к данному соседу — расширение для cache_peer_domain;
- **proxy_auth_realm** Squid proxy-caching web server — строка текста, которая будет выдана на экран клиента при запросе имени/пароля доступа к кэшу.

18.7. Отказ от рекламы. Баннерный фильтр

Вам не хочется тратить лишнее время на загрузку рекламных баннеров? Мне тоже. К счастью, SQUID позволяет достаточно просто решить эту проблему. Просто вставьте следующие строки в свой файл squid.conf:

```
acl good_url url_regex "/usr/local/etc/squid/acl/good_url"
acl bad_urlpath urlpath_regex "/usr/local/etc/squid/acl/
bad_urlpath"
acl bad_url url_regex "/usr/local/etc/squid/acl/bad_url"
http_access deny bad_urlpath !good_url
http_access deny bad_url!good_url
```

Соответственно, нужно будет создать три файла: good_url, bad_url_path и bad_url. В файл bad_url следует поместить URL с плохой репутацией, например:

```
^http://.*doubleclick
^http://.*-ad.flycast.com/server/img/
^http://1000.stars.ru/cgi-bin/1000.cgi
^http://12.16.1.10/~web_ani/
```

А в файл bad_url_path — «плохие» пути, например, такие, которые часто бывают у баннеров:

```
88x31.*gif
88x31.*GIF
100x80.*gif
100x80.*GIF
100x100.*gif
100x100.*GIF
120x60.*gif
120x60.*GIF
179x69.*gif
193x72.*gif
468x60.*gif
```

Примеры файлов good_url, bad_url_path и bad_url можно взять на моей домашней страничке http://dkws.narod.ru.

18.8. Разделение канала с помощью SQUID

Допустим, вам нужно настроить прокси-сервер таким образом, чтобы одна группа компьютеров работала в Интернете с одной скоростью, а другая — с другой. Это может потребоваться, например, для разграничения поль-

зователей, которые используют канал для работы, и пользователей, которые используют ресурсы канала в личных целях. Естественно, первым пропускная способность канала важнее, чем вторым. С помощью прокси-сервера SQUID можно разделить канал.

Для начала в файле конфигурации `squid.conf` укажите, сколько пулов, то есть групп пользователей, у вас будет:

```
delay_pools 2
```

Затем определите классы пулов. Всего существует три класса:

- Используется одно ограничение пропускной способности канала на всех.
- Одно общее ограничение и 255 отдельных для каждого узла сети класса С.
- Для каждой подсети класса В будет использовано собственное ограничение и отдельное ограничение для каждого узла.

В файл `squid.conf` добавьте следующие директивы:

```
delay_class 1 1   # определяет первый пул класса 1 для
                  # домашних пользователей
delay_class 2 2   # определяет второй пул класса 2 для
                  # служащих
```

Теперь задайте узлы, которые будут относиться к пулам:

```
acl home src адреса
acl workers src адреса
delay_access 1 allow home
delay_access 1 deny all
delay_access 2 allow workers
delay_access 2 deny all
```

Затем укажите ограничения:

```
delay_parameters 1 14400/14400
delay_parameters 2 33600/33600 16800/33600
```

Для пула класса 1 используется одно ограничение для всех компьютеров, входящих в пул — 14400 байт. Первое число задает скорость заполнения для всего пула (байт/сек). Второе — максимальное ограничение.

Для пула класса 2 используются ограничения на всю подсеть и отдельно на каждого пользователя. Если бы у нас был определен пул класса 3, то для него ограничения выглядели бы примерно так:

```
delay_parameters 3 128000/128000 64000/128000 12800/64000
```

Первые два числа задают соответственно скорость заполнения и максимальное ограничение для всех. Следующая пара чисел определяет

скорость заполнения для каждой подсети и максимальное ограничение, а третья — скорость заполнения и максимальное ограничение для индивидуального пользователя.

18.9. Настройка поддержки прокси у клиентов

После того, как вы настроили прокси-сервер, осталось напомнить процедуру настройки использования прокси для некоторых распространенных браузеров.

Настройка **Internet Explorer** под использование прокси-сервера производится следующим образом: в окне **Сервис → Свойства обозревателя → Подключение → Настройка сети** установите необходимые параметры, то есть имя прокси-сервера и его порт.

Если вам нужно настроить использование прокси-сервера для **Netscape Communicator**, то выберите из меню **Edit → Preferences → Advanced → Proxies → Manual Proxy Configuration → View** (рис. 18.1). В появившемся окне установите необходимые параметры, то есть имя прокси-сервера и его порт.

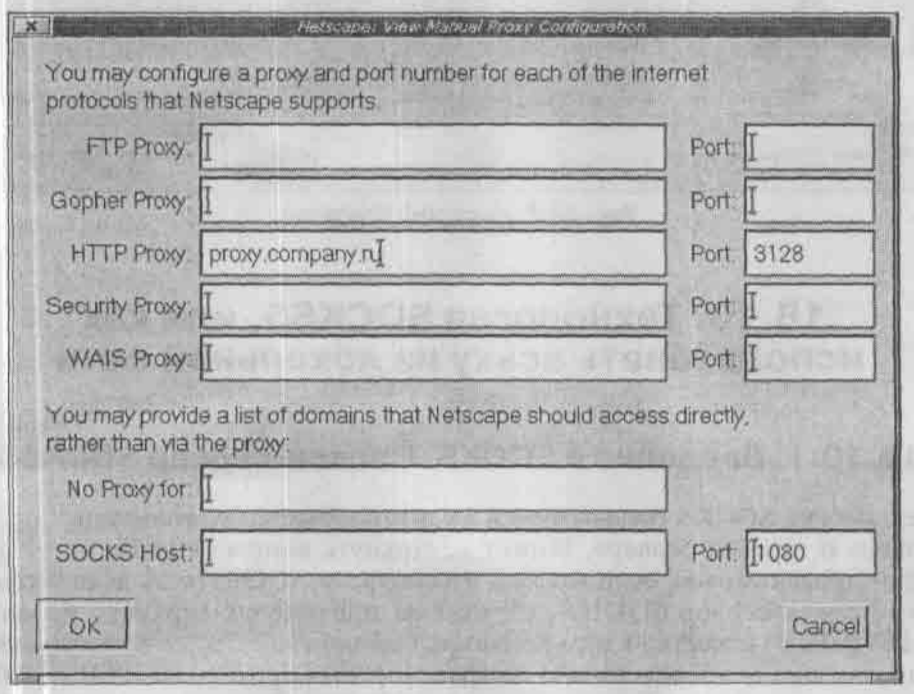


Рис. 18.1. Настройка Netscape Communicator

Теперь посмотрим, как настроить использование прокси-сервера для популярного Linux-браузера **Konqueror**. Выберите в строке меню команду **Настройка → Настроить Konqueror → Прокси** (рис. 18.2). Нажмите кнопку Настроить и укажите имя прокси-сервера и его порт.

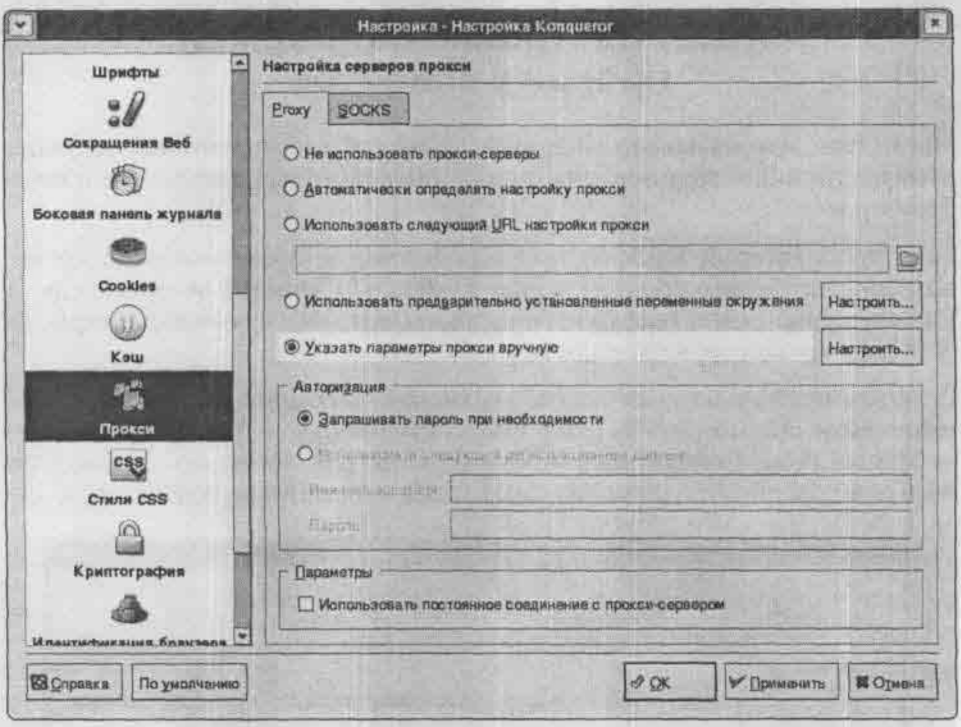


Рис. 18.2. *Настройка Konqueror*

18.10. Технология SOCKS5, или как использовать аську из локальной сети

18.10.1. Введение в SOCKS. Прокси-сервер SOCKS5

Технология SOCKS основывается на использовании одноименных протокола и прокси-сервера. Может возникнуть вопрос: «Зачем нам еще один прокси-сервер, если мы уже рассмотрели SQUID?». А дело в том, что прокси-сервер SQUID работает на протоколах верхнего уровня (HTTP, FTP) и жестко к ним привязан. Однако существуют приложения, работающие непосредственно на транспортных протоколах TCP и UDP и испытывающие проблемы при взаимодействии. Для решения этих про-

блем и был разработан протокол SOCKS. Он позволяет приложениям, работающим по TCP и UDP, использовать ресурсы сети, доступ к которым ограничен в силу архитектуры или настроек сети.

Классическим и основным примером является использование интернет-пейджера ICQ из локальной сети, защищенной брандмауэром (межсетевым экраном). У локального пользователя нет реального IP-адреса и прямого выхода в Интернет, а весь трафик направляется через сервер-шлюз сети, на котором установлен брандмауэр, не пропускающий трафика от ICQ.

Решает проблему пересечения межсетевых экранов клиент/серверными приложениями, работающими по протоколам TCP и UDP, установка сервера SOCKS на шлюзе. При этом через сервер SOCKS будет осуществляться перенаправление запросов на удаленную машину, а также прозрачная передача трафика после установки соединения.

Нужно отметить, что оба прокси-сервера (SOCKS5 и SQUID) могут быть установлены на одном сервере и функционировать одновременно, не мешая друг другу.

Примечание

Теоретически для решения проблемы пересечения бастиона можно было бы использовать IP-маскарадинг. При этом TCP/UDP-пакеты паковались бы в HTTP-сообщения (или другие протоколы верхнего уровня) и шлюз записывал бы в них свои заголовки (свой IP-адрес). В результате казалось бы, что эти пакеты исходят от него самого. Но такое решение имеет много минусов, обусловленных протоколами верхнего уровня и связанными с ними заморочками. Кроме того, в какой-то степени теряется анонимность: ведь при использовании SOCKS информация об IP вообще не передается (это не предусмотрено самой технологией). А еще, поскольку в SOCKS заголовков HTTP нет совсем, никто не сможет определить, что вы использовали прокси-сервер.

Прежде чем мы перейдем непосредственно к рассмотрению установки и настройки сервера SOCKS, стоит отметить еще одно его достоинство. Оно заключаются в том, что SOCKS-серверы могут без проблем выстраиваться в цепочку, позволяя вам еще эффективнее замести следы: направьте свой трафик через несколько прокси-серверов, и ваше сообщение уже никто не выследит. Некоторые прокси-серверы HTTP тоже могут выстраиваться в цепь, но это сопряжено с большими сложностями и проблемами. Кроме того, браузерами официально не предусмотрена поддержка таких цепочек.

Что касается версий протокола, то они отличаются следующим:

- SOCKS4 — основывается на TCP;
- SOCKS5 — работает как с TCP, так и с UDP. Кроме того, в нем расширена система адресации, поддерживается авторизация и удаленные DNS-запросы.

Клиентами сервера SOCKS5 являются популярные клиенты ICQ и **licq**, клиентская версия оболочки **ssh**, а также другие программы.

18.10.2. Настройка сервера SOCKS5

Популярный прокси-сервер, работающий по протоколу SOCKS5, поддерживает компания Permeo (`www.socks.permeo.com`). Я пользуюсь **socks5** v1.0 release 11 и настройку рассмотрю на его примере. Этот сервер не в полной мере некоммерческий (лицензия — не GPL), поэтому вам может быть удобнее использовать альтернативы — **DeleGate** (`www.delegate.org`) или **Dante** (`www.inet.no/dante`). О последнем я скажу в п.18.10.4.

Все настройки сервера **socks5** содержатся в конфигурационном файле `/etc/socks5.conf`. В большинстве случаев параметры по умолчанию вполне приемлемы. Сейчас мы рассмотрим пример конфигурационного файла (листинг 18.1), а потом разберемся, что все это означает.

Листинг 18.1. Файл /etc/socks5.conf

```
set SOCKS5_NOREVERSEMAP
set SOCKS5_NOSERVICENAME
set SOCKS5_NOIDENT
set SOCKS5_MAXCHILD 128
set SOCKS5_TIMEOUT 10

auth - - u
permit u - - - - - -
interface 192.168.0. - eth0
```

В первой строке мы отменяем обратное разрешение адресов, благодаря чему сервер будет работать заметно быстрее. Вторая строка означает, что мы будем протоколировать номера портов вместо имен сервисов. Теоретически это тоже должно повысить эффективность работы сервера. Параметр SOCKS5_NOIDENT запрещает рассылку клиентам ident-запросов. Четвертая строка устанавливает максимально допустимое число потомков сервера — не жадничайте. Пятая строка, как вы уже успели догадаться, устанавливает тайм-аут (10 секунд).

Вся остальная настройка сервера выполняется с помощью директив **auth** и **permit**. Первая устанавливает тип аутентификации, а вторая разрешает доступ определенным узлам и пользователям. Полный формат директивы **auth** такой:

```
auth <исходный_узел> <исходный_порт> <метод_аутентификации>
```

В приведенном примере мы будем запрашивать пароль у всех клиентов.

Формат директивы **permit**:

```
permit <аутентификация> <команда> <исх_узел> <узел_
назначения> <исх_порт> <порт_назнач> [список_
пользователей]
```

В примере я разрешаю доступ всем и отовсюду с использованием аутентификации. Следующий пример использования директивы **permit** демонстрирует гибкость этого прокси-сервера:

```
permit u cpubt 192.168. - - [100,1000] den
```

В этом примере мы разрешаем доступ пользователю den (с использованием пароля, конечно). Пользователь `den` имеет право использовать **Connect**, **Ping**, **UDP**, **BIND** и **Traceroute** (cpubt) с адресов 192.168.*.*. Диапазон входящих (первый «-») и исходящих (второй «-») портов — от 100 до 1000.

В дополнение к директиве **permit** можно использовать директиву **deny** такого же формата, но противоположного назначения (запрет доступа).

Директива **interface** в приведенном примере разрешает все соединения от компьютеров с адресами 192.168.0.* (наша внутренняя сеть) ко всем портам интерфейса `eth0`.

Имена и пароли пользователей сервера **socks5** содержатся в файле `/etc/socks5.passwd` в формате `<имя> <незашифрованный_пароль>`. После создания этого файла настройку **socks5** можно считать законченной.

18.10.3. Запуск сервера socks5

Запускается сервер следующей командой:

```
# /usr/local/bin/socks5 -f -s
```

При запуске с этими ключами демон должен перейти в фоновый режим и выводить диагностические сообщения на стандартный вывод (в нашем случае это экран). Если сервер сконфигурирован правильно, вы должны увидеть примерно следующее:

```
11410: Socks5 starting at Mon Mar 4 19:13:55 2002 in normal mode
```

После удачного запуска остановите сервер (`killall socks5`) и добавьте его запуск в сценарий автозагрузки системы.

18.10.4. Dante — еще один сервер SOCKS5

Этот сервер считается более простым в настройке. Он использует файл конфигурации `/etc/sockd.conf` (листинг 18.2).

Листинг 18.2. Примерный файл /etc/sockd.conf

```
internal: 192.168.0.1 port = 1080
external: 111.111.111.111
client pass {
    from: 192.168.0.0/16 to: 0.0.0.0/0
        }
pass      {
    from: 0.0.0.0/0 to: 192.168.0.0/16
    command: bindreply udpreply
    log: connect error
        }
```

Директива **internal** определяет ваш внутренний интерфейс (точнее, внутренний IP-адрес), а **external** — ваш настоящий IP (111.111.111.111). В блоке **client pass** указываются возможные клиенты вашего сервера (сеть 192.168.0.0), а в блоке **pass** — имена узлов, которые могут «общаться» с вашими клиентами. В приведенном примере разрешается отвечать клиентам со всех узлов (0.0.0.0). Протоколироваться будут только ошибки соединения.

18.10.5. Настройка клиентов SOCKS5 (ICQ и licq)

Настройку клиентов будем рассматривать на примере двух самых, наверное, популярных SOCKS-клиентов. Сначала рассмотрим настройку программы ICQ для Windows, а потом **licq** — ICQ-клиента для Linux.

Запустите программу ICQ и нажмите на кнопку **ICQ**. Из появившегося меню выберите команду **Preferences** и перейдите в раздел **Connections** на вкладку **Server** (рис.18.3). Включите режим использования прокси-сервера и установите тип прокси-сервера — SOCKS5. Потом перейдите на вкладку **Firewall** и установите параметры прокси-сервера: имя, порт, тип (socks5), имя пользователя и пароль (рис. 18.4).

С программой **licq** будет немножко сложнее. Во-первых, нужно установить на компьютере пользователя программу **runsocks**, входящую в состав пакета прокси-сервера (эту программу можно также найти в Интернете отдельно), и перекомпилировать **licq**, включив поддержку SOCKS5. Для этого перейдите в каталог, содержащий исходные тексты **licq**, и запустите сценарий `configure` с параметром **--enable-socks5**:

```
$ ./configure --enable-socks5
```

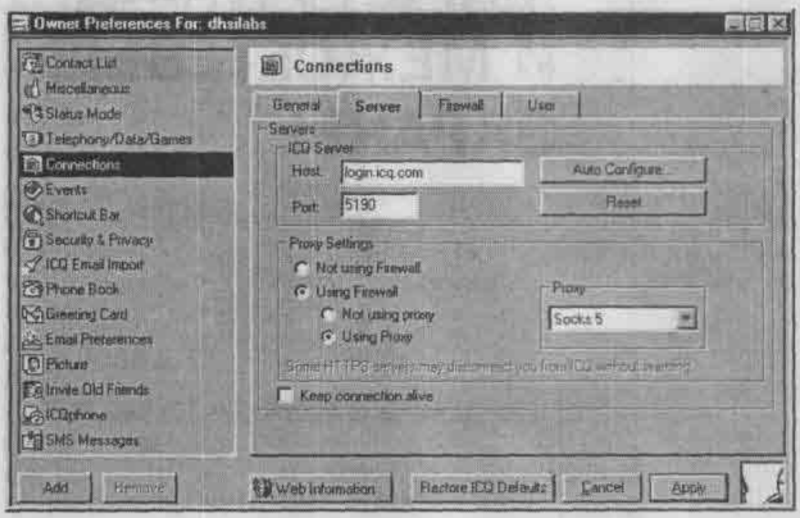


Рис. 18.3. *Свойства соединения ICQ*

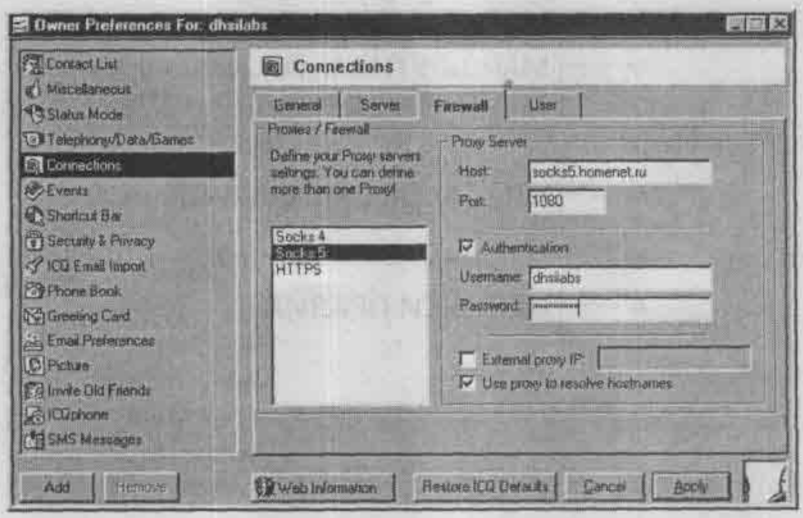


Рис. 18.4. *Параметры прокси-сервера*

После этого выполните привычные команды: `make; make install`.

Теперь нужно создать файл `/etc/libsocks5.conf` и добавить в него строку:

```
socks5 - - - - 192.168.0.1:port
```

192.168.0.1 — это адрес вашего SOCKS5-сервера, port — порт, необходимый клиенту (обычно 1080).

Глава 19

МАРШРУТИЗАЦИЯ И МЕЖСЕТЕВЫЕ ЭКРАНЫ

- ВВЕДЕНИЕ В МАРШРУТИЗАЦИЮ
- ПРОГРАММЫ МАРШРУТИЗАЦИИ В LINUX
- РАСШИРЕННЫЕ СРЕДСТВА МАРШРУТИЗАЦИИ. КОМПЛЕКС IPROUTE2
- ЧТО ТАКОЕ БРАНДМАУЭР
- ЦЕПОЧКИ ПРАВИЛ

19.1. Введение в маршрутизацию

Маршрутизация является очень ответственным механизмом, отвечающим за то, как будет происходить обмен данными, как будут пролегать информационные потоки. Скорость и эффективность обмена данными во многом зависят от того пути, по которому они следуют от одного компьютера сети к другому.

Маршрутизация имеет смысл в сильно разветвленных сетях и основывается на использовании таблиц маршрутизации. Такая таблица имеется на каждом узле, выступающем в роли маршрутизатора. В ней содержится информация об окружающих узлах и известных маршрутах, на основе которой маршрутизатор будет выбирать оптимальный путь для передачи данных.

При оценке и выборе маршрута используются так называемые метрики стоимости. В качестве такой метрики выступают какие-либо критерии, по которым можно оценить маршрут. Различные протоколы маршрутизации могут использовать различные метрики, то есть выбирать оптимальный маршрут, основываясь на анализе разных параметров. Рассмотрим основные протоколы маршрутизации:

- **RIP** — один из старейших протоколов маршрутизации, разработанный компанией Xerox. Метрикой стоимости у этого протокола является количество переходов, которое должен совершить пакет данных от отправителя к получателю. Этот протокол разрабатывался во времена небольших сетей, поэтому все узлы, находящиеся на расстоянии больше 15 переходов, он воспринимает как недосягаемые. Это значит, что в сетях, в которых в одну цепочку могут быть выстроены более 15 маршрутизаторов, использовать протокол RIP нельзя. Данный протокол использует в своей работе демон **routed**, который будет рассмотрен чуть позднее
- **RIP-2** представляет собой улучшенную версию протокола RIP. Одно из основных улучшений заключается в том, что вместе с адресом следующего перехода передается сетевая маска. Благодаря этому упрощается управление сетями, в которых есть подсети.

- **OSFP** является самым популярным и широко используемым протоколом маршрутизации. В его основе лежит специальный математический алгоритм, который позволяет высчитывать оптимальные маршруты. Этот протокол является топологическим, то есть учитывающим состояние канала. По сравнению с RIP он обладает следующими достоинствами: возможностью управления несколькими маршрутами и возможностью разделения сети на сегменты, которые будут предоставлять друг другу только высокоуровневые данные маршрутизации. Этот протокол рекомендуется использовать в сетях с разветвленной структурой, в которой может возникать большое количество дублирующихся маршрутов.

Следует понимать разницу между маршрутизацией и перенаправлением трафика. Маршрутизация представляет собой сложный процесс просчета и выбора наилучшего на данный момент маршрута. При этом используется большое количество входной информации, основывающейся как на предыдущем опыте работы маршрутизатора, так и на текущем состоянии сети. Значительная часть этой информации предоставляется соседними узлами, поэтому синхронизация таблиц маршрутизации у соседних узлов представляет собой существенную проблему.

Перенаправление трафика же является простой операций, направляющей пакеты данных в соответствии с определенным условием. Никакого выбора и расчета при этом не производится. Можно сказать, что перенаправление — это простейший (статический) метод маршрутизации. Мы рассмотрим его в параграфе, посвященном брандмауэрам.

19.2. Программы маршрутизации в Linux

19.2.1. Демон routed

Стандартной программой маршрутизации в Linux является демон **routed**. Этот демон, как правило, настраивается сам (динамически) и не требует конфигурирования. Обнаруженные маршруты он заносит в маршрутную таблицу ядра.

В своей работе демон **routed** использует протокол RIP. Чтобы воспользоваться преимуществами протоколов RIP-2 или OSFP, вы должны использовать другой демон — **gated**. Демон **routed** может работать либо в режиме сервера (-s), либо в режиме подавления сообщений. Во втором режиме он будет только получать от соседей маршрутную информацию, но сам ее отсылать не будет.

Для добавления статических маршрутов вручную служит команда `route`. Рассмотрим пример такого маршрута. Пусть у нас есть две сетевые платы eth0 и eth1:

```
# ifconfig eth0 192.168.1.1 up
# ifconfig eth0 192.168.2.1 up
```

и нам нужно обеспечить маршрутизацию между подсетями 192.168.1.0 и 192.168.2.0. С этой целью объявляем, что машины, которые находятся в вашем локальном сегменте 192.168.1.*, «сидят» на первом интерфейсе и общаться с ними нужно напрямую:

```
# route add net 192.168.1.0 192.168.1.1 netmask 255.255.255.0 0
```

А с машинами с адресами 192.168.2.* будем разговаривать через eth1:

```
# route add net 192.168.2.0 192.168.2.1 netmask 255.255.255.0 0
```

Последний аргумент команды `route` — это метрика. Ее можно понимать как «расстояние до шлюза назначения» или «сколько пересадок между шлюзами придется сделать пакету по пути туда и обратно». Т.к. адреса 192.168.1.1 и 192.168.2.1 являются нашими собственными адресами, то это расстояние равно нулю.

Сетевые пакеты для IP-адресов, которые не лежат в нашей локальной сети, будем отправлять на машину 192.168.1.11, а она сама будет разбираться, что с ними делать:

```
# route add default 192.168.1.11 1
```

Таким образом мы объявили маршрут по умолчанию со значением метрики, равным 1.

Не забудьте только добавить вызовы команды `route` в загрузочный сценарий, потому что при перезагрузке правила маршрутизации ядра теряются.

Забегая несколько вперед, замечу, что такой статический маршрут представляет собой обычное правило перенаправления трафика, поэтому его можно реализовать и средствами пакетного фильтра IPTables:

```
# iptables -P FORWARD DROP
# iptables -A FORWARD -s 192.168.1.0/24 -d 192.168.2.0/24 -j ACCEPT
# iptables -A FORWARD -s 192.168.2.0/24 -d 192.168.1.0/24 -j ACCEPT
```

А вот более сложный пример, приведенный в документации по IPTables. Пусть у нас имеется одно-единственное соединение с Интернетом и мы не хотим, чтобы кто-либо вошел в нашу сеть извне:

```
## Загрузим модули для отслеживания соединений
# (не нужно, если они встроены в ядро)
```

```
# insmod ip_conntrack
# insmod ip_conntrack_ftp

## Создадим цепь block, которая будет блокировать
# соединения извне.
# iptables -N block
# iptables -A block -m state --state ESTABLISHED,RELATED
-j ACCEPT
# iptables -A block -m state --state NEW -i ! ppp0 -j ACCEPT
# iptables -A block -j DROP

## Весь входящий и маршрутизированный трафик будет
# проходить через block
# iptables -A INPUT -j block
# iptables -A FORWARD -j block
```

19.2.2. Демон gated — правильный выбор

В последнее время демон **gated** используется чаще, чем стандартный **routed**. Объясняется это тем, что **gated** более гибок в конфигурировании и обладает большими возможностями, в частности, им поддерживаются протоколы RIP-2 и OSFP.

Программа **gated** была разработана группой американских университетов для работы сети NFSNET. Она позволяет организовать многофункциональный шлюз, обслуживающий как внутреннюю, так и внешнюю маршрутизацию. На данный момент **gated** поддерживает следующие протоколы маршрутизации:

- RIP версий 1 и 2
- HELLO
- OSPF версии 2
- EGP версии 2
- BGP версии 2, 3 и 4.

Таблица 19.1 поможет вам сравнить возможности демонов **routed** и **gated**.

Протоколы, поддерживаемые gated и routed — *Таблица 19.1*

Демон	Протоколы внутренних маршрутизаторов			Протоколы внешних маршрутизаторов	
	RIP	HELLO	OSPF	BGP	EGP
routed	V1	-	-	-	-
gated, версия 2	V1	+	-	V1	+
gated, версия 3	V1,V2	+	V2	V2,V3	+

Рассмотрим классическое подключение локальной сети к Интернету. Пусть адрес нашей локальной сети 143.100.100.0, а на шлюзе установлены две сетевые платы с IP-адресами 143.100.100.1 и 143.100.200.1. Пусть в нашей сети есть машина с IP-адресом 143.100.100.5, на которой также установлен **gated**. Настроим **gated** сначала на этой рабочей станции, а потом — на сервере. Для настройки может использоваться утилита **gdc**, поставляемая вместе с самим **gated**.

Настройка **gated** осуществляется путем редактирования файла конфигурации `/etc/gated.conf`. Содержимое этого файла для рабочей станции приведено в листинге 19.1, а для сервера — в листинге 19.2.

Листинг 19.1. Файл конфигурации /etc/gated.conf для рабочей станции

```
# Это IP-адрес рабочей станции
interface 143.100.100.5 passive;
# используем протокол RIP (Route Internet Protocol)
rip yes;
```

Листинг 19.2. Файл конфигурации /etc/gated.conf для сервера

```
# Описываем интерфейсы и протокол
interface 143.100.100.1 passive;
interface 143.100.200.1 passive;
rip yes;

export  proto rip interface 143.100.100.1
{
 proto direct
 {
   announce 143.100.200.0 metric 0 ;
 } ;
} ;
export proto rip interface 143.100.200.1
{
 proto rip interface 143.100.100.1
 {
   announce all ;
 } ;
} ;
```

Первая директива **export** объявляет подсеть 143.100.100.0 (наша сеть) через интерфейс 143.100.100.1, который объявляется шлюзом в данную подсеть, то есть считается, что интерфейс 143.100.100.1 принадлежит узлу, входящему в эту сеть. Директива **proto direct** говорит о том, что пакеты

для подсети нужно посылать непосредственно на интерфейс, а нулевая метрика означает, что интерфейс стоит на шлюзе в подсеть.

Вторая директива **export** сообщает всем узлам подсети через интерфейс 143.100.200.1 все маршруты, которые данный шлюз получает из подсети 143.100.100.0 через интерфейс 143.100.100.1.

При написании директив **export** внешняя конструкция всегда определяет интерфейс, через который сообщается информация, а внутренняя — источник, через который эту информацию будет получать **gated**.

Рассмотрим пример из документации по **gated**, в котором нашу сеть через подсеть подключают к Интернету (листинг 19.3).

Листинг 19.3. Подключение через подсеть

```
rip yes;
export proto rip interface 136.66.12.3 metric 3
{
 proto rip interface 136.66.1.5
 {
   announce all ;
 } ;
} ;
export proto rip  interface 136.66.1.5
{
 proto rip interface 136.66.12.3
 {
   announce 0.0.0.0 ;
 } ;
} ;
```

Первая директива **export** говорит о том, что **gated** получает все, что передается в подсеть, связывающую данную сеть с Интернетом, через интерфейс 136.66.12.3 (речь идет о маршрутах, а не о самих данных).

Вторая директива **export** определяет место назначения — куда по умолчанию нужно отправлять информацию из сети, чтобы она достигла адресата, который расположен за пределами локальной сети. Адрес 0.0.0.0, соответствующий любой машине за интерфейсом 136.66.12.3, определяется через интерфейс 136.66.1.5 для всей локальной сети.

После настройки **gated** нужно перезапустить:

```
# service gated restart
```

19.3. Расширенные средства маршрутизации. Комплекс iproute2

19.3.1. Пакет iproute2

Начиная с версии ядра 2.2, сетевая подсистема Linux была значительно переработана, в нее было добавлено много новых возможностей — управление трафиком, маршрутизация на основе правил и т.п. Доступ к этим возможностям предоставляется пакетом iproute2, входящим в состав большинства современных дистрибутивов.

В случае необходимости можно скачать этот пакет с сайта `ftp://ftp.inr.ac.ru./ip-routing`.

19.3.2. Утилита ip

Утилита **ip**, входящая в состав пакета iproute2, объединяет в себе все возможности команд **ifconfig**, **arp** и **route**. Формат ее вызова такой:

```
ip [ключи]  объект [ команда [аргументы команды] ]
```

Ключи можно указывать следующие:

- **-s** — вывод статистической информации;
- **-f** — выбор протокола:
- -f inet — протокол IPv4;
- -f inet6 — протокол IPv6;
- -f link — сетевое устройство;
- **-r** — разрешать IP-адреса в имена узлов;
- **-V** — печать версии программы.

Аргумент **Объект** позволяет выбрать объект, с которым будем работать:

- **адрес** — IPv4 или IPv6 адрес устройства;
- **link** — устройство;
- **neighbour** — ARP-адрес;
- **route** — маршрутизация;
- **rule** — база данных правил маршрутизации;
- **madress** — широковещательный адрес;
- **tunnel** — туннель через IP.

Аргумент **Команда** описывает действие над **Объектом**:

- **ip link** — конфигурация сетевого устройства;
- **ip link set** — изменение параметров сетевого устройства;
- **ip link show** — просмотр параметров сетевого устройства.

При изменении параметров сетевого устройства вы можете задать следующие аргументы:

- **up** — включить;
- **down** — выключить;
- **arp on** или **arp off** — изменение флага NOARP на устройстве;
- **dynamic on** или **dynamic off** — изменяет флаг DYNAMIC на устройстве;
- **multicast on** или **multicast off** — изменяет флаг MULTICAST на устройстве;
- **name** — изменяет имя устройства;
- **mtu <Число>** — изменяет значение MTU на устройстве;
- **address <Адрес>** — изменяет адрес на устройстве;
- **broadcast <Адрес>** — изменяет широковещательный адрес на устройстве.

19.3.3. Просмотр параметров сетевого устройства

Команду **ip link show** лучше всего рассматривать на примерах. Для получения информации о состоянии устройства `eth0` введите команду:

```
# ip link ls dev eth0
eth0:  mtu 1500 qdisc cbq qlen 100
    link/ether 00:44:67:91:31:1d brd ff:ff:ff:ff:ff:ff
```

Получить статистику устройства `eth0` можно командой:

```
#ip -s link ls dev eth0
 2: eth0:  mtu 1500 qdisc cbq qlen 100
    link/ether 00:44:67:91:31 brd ff:ff:ff:ff:ff:ff
    RX: bytes  packets  errors  dropped overrun mcast
    xxxxxxxx   xxxxxx 0       0       0       0
    TX: bytes  packets  errors  dropped carrier collsns
    xxxxxxxx   xxxxxx 0       0       0       132934
```

Вместо xxxxxxxx xxxxxx вы увидите количество принятых(RX)/переданных(TX) байтов и пакетов.

19.3.4. Операции над адресами: команда ip address

Команда **ip address** управляет адресами на устройстве. **Объект** для нее — это IPv4 или IPv6 адрес. Эта команда показывает адреса и их свойства, а также добавляет новые адреса.

Чтобы добавить адрес 192.168.0.1/24 с маской подсети 255.255.255.0 со стандартным широковещательным адресом и именем eth0:Alias, введите команду

```
# ip addr add 192.168.0.1/24 brd + dev eth0 label eth0:Alias
```

Используются еще следующие варианты команды:

ip address delete предназначена для удаления адресов. Для удаления адреса 192.168.0.1/24 с устройства eth0 введите команду `ip addr del 192.168.0.1/24 dev eth0`.

ip address show выводит информацию об адресе.

19.3.5. Управление таблицей маршрутизации

Команда **ip route** управляет таблицей маршрутизации:

ip route add — добавить новый маршрут;
ip route change — изменить маршрут;
ip route replace — заменить маршрут.

Добавим маршрут к сети 192.168.0.0/24 через 192.168.1.1:

```
# ip route add 192.168.0.0/24 via 192.168.1.1
```

19.3.6. Динамическая маршрутизация

Команда **ip route** позволяет добавить динамический маршрут: шлюз будет выбираться в зависимости от текущей нагрузки на него. Всегда будет выбираться шлюз с минимальной нагрузкой.

Пусть у нас есть два устройства — ppp0 и ppp1. Маршрут по умолчанию — через устройство ppp0, но если этот маршрут недоступен, будет использоваться ppp1:

```
# ip route add default scope global nexthop \
  dev ppp0 nexthop dev ppp1
```

Для удаления маршрута используйте команду **ip route delete**:

```
# ip route del default scope global nexthop \
# dev ppp0 nexthop dev ppp1
```

19.3.7. Управление правилами маршрутизации

Для решения этой задачи предназначена команда **ip rule**. Маршрутизация производится в зависимости от:

- адреса получателя;
- адреса отправителя;
- IP-протокола;
- транспортного протокола.

По умолчанию используются три таблицы правил маршрутизации:

- Local — содержит таблицы для локальных и широковещательных адресов;
- Main — самая обыкновенная таблица маршрутизации;
- Default — пустая таблица по умолчанию.

Аргументы команды **ip rule**:

1. адрес отправителя;
2. адрес назначения;
3. имя интерфейса, с которого получен пакет;
4. метка пакета, которая устанавливается брандмауэром;
5. идентификатор таблицы маршрутизации: им может быть номер или строка из файла `/etc/iproute2/rt_tables`;
6. приоритет таблицы (число).

Вот несколько примеров. Требование пересылать пакеты с сети 192.168.0.0/24 согласно таблице Main:

```
# ip rule add from 192.203.80.0/24 table main prio 100
```

Теперь допустим, что у нас есть два канала в Интернет (два провайдера): ppp1 с адресом 193.168.99.99, который связан с 193.168.99.100, и ppp2 с адресом 193.168.100.99, связанный с 195.1.1.1. Пользователь ivanov хочет, чтобы мы его пакеты отправляли через ppp2 (второго провайдера):

```
# echo 200 ivanov >> /etc/iproute2/rt_tables
# ip rule add from 192.168.0.10 table ivanov
# ip rule ls
0: from all lookup local
32765: from 192.168.0.10 lookup ivanov
32766: from all lookup main
32767: from all lookup default
```

Теперь для этого пользователя назначим маршрут по умолчанию и очистим кэш-таблицы маршрутизации, чтобы наши изменения вступили в силу:

```
# ip route add default via 195.1.1.1 dev ppp2 table ivanov
# ip route flush cache
```

Рассмотрим еще один практический пример. Направим весь трафик на порт 21 через устройство eth1:

```
# iptables -A PREROUTING -i eth0 -t mangle -p tcp --dport
21 -j MARK --set-mark 2
```

Теперь нужно создать правила для помеченных пакетов

```
# echo 202 21.tbl >> /etc/iproute2/rt_tables
```

```
# ip rule add fwmark 2 table 21.tbl
# ip route add default via 192.168.0.10 dev eth1 table
21.tbl
# ip route flush cache
```

19.4. Что такое брандмауэр

Брандмауэр (межсетевой экран, бастион, firewall) — это компонент системы, обеспечивающий защиту сети от несанкционированного доступа.

Как известно, весь трафик в сети состоит из пакетов. Каждый пакет состоит из двух частей: заголовка и тела. В заголовке пакета находится информация об источнике, адресате, типе пакета, а также прочая информация, которая характерна для пакетов определенных типов. В теле пакета находится собственно передаваемая информация. Так вот, брандмауэр представляет собой пакетный фильтр: он просматривает заголовок каждого проходящего через него пакета, а потом решает, что с этим пакетом делать: принять, игнорировать или отказать с уведомлением отправителя.

Обычно брандмауэр используется на шлюзах, соединяющих две сети. Например, на шлюзе между локальной сетью и Интернетом можно запретить передавать или принимать пакеты некоторых категорий, отграничив таким образом свою хорошо организованную сеть от всемирного хаоса.

Фильтрация пакетов встроена в ядро ОС Linux. В старых дистрибутивах (ядро 2.2 и ниже) стандартным брандмауэром служит **IPChains**, в новых (ядро 2.4 и выше) — **IPTables**. Оба они позволяют защититься от таких распространенных атак, как пинг смерти, атака на отказ, IP-спуфинг, фрагментация пакетов. Возможности пакетного фильтра **IPTables** значительно шире, чем у **IPChains**, но на практике дополнительные возможности используются редко — в основном применяются средства, присутствовавшие уже в **IPChains**. Поэтому я начну изложение принципов и приемов настройки брандмауэра с этого устаревшего фильтра.

19.5. Цепочки правил

Ядро стартует с тремя списками правил: **input**, **forward**, **output**. Эти правила называются firewall-цепочками или просто цепочками. Цепочка — это набор правил вида «заголовок пакета: действие». Если заголовок пакета соответствует заголовку, указанному в правиле, выполняется заданное в правиле действие. Если первое правило к этому пакету неприменимо, то рассматривается следующее правило в цепочке. Если ни одно правило

не применимо к пакету, то пакет, скорее всего, будет отвергнут. Если какой-нибудь пакет отвергается, то сообщение об этом протоколируется службой **syslog**.

Пакет, поступающий извне, сначала проходит предварительную проверку. Она включает в себя сверку контрольной суммы (не поврежден ли пакет при пересылке) и «санитарную проверку» (не нарушит ли некорректный пакет работу кода проверки правил). Поврежденные и некорректные пакеты отвергаются.

Принятый пакет поступает на входной фильтр (цепочка **input**). Если входной фильтр пропускает пакет, то код маршрутизации проверяет адрес получателя, после чего пакет либо передается локальному процессу, либо маршрутизируется на другую машину. Пакет, предназначенный для другой машины, должен пройти через цепочку перенаправления **forward**. Если правила этой цепочки разрешают пакет, он будет маршрутизирован, в противном случае — отвергнут. Пакет также будет отвергнут, если ядро не поддерживает маршрутизации.

Прежде чем пакет выйдет из сетевого интерфейса, он должен пройти цепочку **output**. Любая программа может отправить пакет, который будет направлен в эту цепочку. Если она пропускает пакет, то он попадет в сеть — станет исходящим пакетом. Отвергнутый пакет не выйдет за пределы компьютера — он будет уничтожен.

Пакет, предназначенный для интерфейса lo (обратная петля или шлейфовый интерфейс), сначала проходит выходную цепочку, а потом попадает во входную цепочку интерфейса lo. Прохождение пакета через цепочки правил схематично представлено на рис. 19.1.

Рис. 19.1. *Принцип работы пакетного фильтра*

19.6. IPTables — пакетный фильтр для ядер 2.4.x. и 2.6.x

Фильтр **IPTables,** очень похожий на **IPChains**, называется так потому, что хранит цепочки правил в таблицах. Главной таблицей, в которой хранятся правила обработки всех стандартных типов трафика, является таблица **filter**. Именно с ней мы и будем работать. К стандартным таблицам относятся также:

- таблица **nat** — используется для построения NAT-преобразователей;
- таблица **mangle** — задает типы преобразования некоторых пакетов.

Для просмотра содержимого таблицы служит следующая команда:

```
iptables -L   -t   <имя_таблицы>
```

19.6.1. Что изменилось в IPTables по сравнению с IPChains

- Имена стандартных цепочек INPUT, OUTPUT и FORWARD теперь пишутся в верхнем регистре. Имена пользовательских цепочек по соглашению пишутся строчными буквами и могут иметь длину до 31 символа.
- Действие «отклонить пакет» теперь называется не DENY, а DROP.
- Действие MASQ называется MASQUERADE.
- Для перенаправления пакетов на пользовательское устройство используется цель QUEUE (раньше для этого использовалась опция –o).
- Опция -i теперь означает входящий интерфейс и работает только в цепочках INPUT и FORWARD. Правила в цепочках FORWARD и OUTPUT, которые использовали опцию –i, следует переписать с использованием опции -o.
- Если обнулить встроенную (стандартную) цепочку, то очистятся также и счетчики политик.
- Правила для протоколов TCP и UDP должны использоваться вместе с флагами --source-port и --destination-port (-sport и –dport). Эти флаги должны использоваться только с указанием протокола, например, -p tcp.
- Опция –y теперь называется –syn и должна указываться после флага –p.
- Действия REJECT и LOG реализованы отдельными модулями ядра.

19.6.2. Настройка ядра Linux для поддержки IPTables

Новые параметры конфигурации ядра, включающие поддержку **IPTables**, я представил в таблице 19.3. Возможно, в вашем дистрибутиве некоторые из них отключены, тогда вам придется пересобрать ядро. Следующая глава подробно рассказывает о том, как это сделать.

Параметры конфигурации ядра *Таблица 19.3*

Параметр	Назначение
CONFIG_IP_NF_IPTABLES	Необходим для работы IPTables. Не включив этот параметр, вы вообще не сможете использовать IPTables
CONFIG_PACKET	Позволяет использовать программы, которые работают непосредственно с сетевым устройством. Примером такой программы может послужить **tcpdump**
CONFIG_NETFILTER	Включите этот параметр, если вы собираетесь использовать ваш компьютер в качестве шлюза.
CONFIG_IP_NF_CONNTRACK	Позволяет отслеживать соединения. Этот параметр необходим для работы функций NAT или IP-маскарадинга. На компьютере-шлюзе включите этот параметр
CONFIG_IP_NF_FTP	Из-за большого количества FTP-запросов модуль IP_NF_CONNTRACK не в состоянии проследить все FTP-соединения, поэтому в помощь ему добавлен модуль CONFIG_IP_NF_FTP, отслеживающий только FTP-соединения. Включите этот параметр, если на вашем компьютере установлен FTP-сервер
CONFIG_IP_NF_MATCH_LIMIT	Необязательный параметр. Позволяет ограничить количество пакетов, передаваемых/принимаемых за определенный промежуток времени
CONFIG_IP_NF_MATCH_MAC	Позволяет блокировать пакеты, используя MAC-адрес (а не IP-адрес)

Все остальные опции, связанные с **IPTables**, содержат в своем названии слово MATCH, например, CONFIG_IP_NF_MATCH_MARK. Эти опции разрешают выполнять над пакетами определенные действия.

Фильтр **IPTables** может загружаться как модуль при первом запуске **iptables**, а может быть встроен в ядро постоянно (что рекомендуется). Модуль фильтра **filter** называется iptable_filter.o.

У вас может быть установлен модуль ipchains.o и соответствующий ему сервис — **ipchains**. Этот модуль несовместим с **iptables**, поэтому вы должны выбрать, какой из них вы будете использовать.

19.6.3. Первичная настройка IPTables. Задание политики по умолчанию

Если к пакету не может быть применено ни одно правило из стандартной цепочки, то судьбу этого пакета определяет политика цепочки — действие по умолчанию. Значением политики может быть либо ACCEPT (принять), либо DROP (отклонить), либо REJECT (отклонить с уведомлением источника). Первоначально встроенные цепочки не содержат ни одного правила, а политикой имеют ACCEPT, поэтому первое, что нужно сделать, — это установить в целях безопасности действие DROP:

```
# iptables -P  INPUT DROP
# iptables -P  OUTPUT DROP
# iptables -P  FORWARD DROP
```

19.6.4. Действия над цепочками

Рекомендуется создавать отдельные цепочки для различных объектов фильтрации. Это позволяет логически группировать правила. Для создания новой цепочки используется опция –N, за которой следует имя новой цепочки (напоминаю, что имена пользовательских цепочек пишутся строчными буквами):

```
iptables -N имя_цепочки
```

Ключи программы **iptables**, предназначенные для манипуляций с цепочками и правилами в них, точно такие же, как у программы **ipchains** (таблица 19.2).

19.6.5. Правила фильтрации

Задание правил фильтрации **IPTables** похоже на задание правил в **IPChains**. Для создания правила используется опция --append (или -A). После этой опции указывается имя цепочки и критерий выбора пакетов в этой цепочке. Затем указывается опция --jump (или -j), значением которой служит действие (ACCEPT, DROP и т.п.):

```
iptables  -A  INPUT  критерий_выбора  -j  действие
```

В качестве критерия выбора обычно указываются либо порты, либо IP-адреса, либо MAC-адрес. По этим параметрам и будет «вылавливаться» пакеты из цепочки. Можно задать сразу несколько критериев выбора, перечислив их друг за другом.

Фильтрация по портам, IP-адресу и MAC-адресу в **IPTables** задается с помощью следующих критериев выбора:

- **-p** протокол — задает протокол, по которому будет производиться отбор пакетов. Может принимать значение all, что означает «все протоколы»;
- **-s** адрес (или **--source** адрес) — задает исходный IP-адрес, по которому будет производиться отбор пакетов. В качестве значения может указываться сетевая маска или порт;
- **-d** адрес (или **--destination** адрес) — задает IP-адрес получателя, по которому будет производиться отбор пакетов. В качестве значения может указываться сетевая маска или порт;
- **-i** интерфейс — задает входной интерфейс (eth0, ppp0 и т.п.), с которого будет производиться отбор пакетов. Этот критерий работает только в цепочках INPUT и FORWARD. При задании интерфейса может использоваться знак «+», означающий «все интерфейсы заданного типа». Например, запись ppp+ означает все интерфейсы ppp (ppp0-pppN);

- **-o** интерфейс — задает выходной интерфейс (eth0, ppp0 и т.п.), с которого будет производиться отбор пакетов. Способ задания такой же, что и для предыдущего случая. Этот критерий работает только в выходных цепочках (OUTPUT);
- **--sport** порт [:порт] (или **--source-port** порт [:порт]) — задает исходный порт или диапазон исходных портов, по которому будет производиться отбор пакетов. Эта опция может использоваться только после опций **-p tcp** или **-p udp**. Диапазон портов задается номером первого и конечного портов, разделенных двоеточием;
- **--dport** порт [:порт] (или **--destination-port** порт [:порт]) — задает порт назначения или диапазон портов назначения, по которому будет производиться отбор пакетов. Эта опция может использоваться только после опций **-p tcp** или **-p udp**. Диапазон портов задается номером первого и конечного портов, разделенных двоеточием;
- **--port** порт [,порт] — позволяет задать порты, которые будут восприниматься и как порты источника, и как порты назначения. Использование этого критерия возможно только после использования **-p tcp** или **-p udp** и опции **-m multiport**;
- **-m** состояние — задает состояние соединения. В качестве состояния возможно указание большого количества вариантов. За подробной информацией обращайтесь к справочной системе. Некоторые варианты использования данного критерия приведены ниже, на практических примерах;
- **--mac-source** мак_адрес — задает MAC-адрес, по которому будет производиться отбор пакетов. Возможно только после использования критерия **-m mac**.

Рассмотрим теперь практические примеры. Выделить пакеты, приходящие от узла с MAC-адресом 11:12:13:14:15:16, можно с помощью правила:

```
iptables -A INPUT --mac-source 11:12:13:14:15:16  -j ……
```

Потом вы уже сами определите действие после опции –j. Если же нам нужно выделить все пакеты, кроме тех, которые присылает узел с этим MAC-адресом, то нужно использовать символ инверсии «!».

IPTables позволяет указывать несколько портов (не больше 15) через запятую, например:

```
iptables -A INPUT -p tcp -m multiport --sport 22,53,80,110 -j ……
```

Вместо портов источника вы можете указать порты назначения, используя опцию --dport.

Если вы хотите одновременно указать как порты источника, так и порты назначения, используйте опцию --port:

```
iptables -A INPUT -p tcp -m multiport --port 22,53,80,110
```

19.6.6. Фильтрация по отдельным пользователям

Если **IPChains** умел отфильтровывать только пакеты, исходящие от определенного компьютера, то теперь мы можем выделять пакеты отдельных пользователей. Для этого предназначены следующие критерии, которые могут использоваться только для исходящих пакетов в цепочке OUTPUT:

- **--uid-owner** UID — отбор пакетов по UID пользователя;
- **--gid-owner** GID — отбор по группе (GID);
- **--pid-owner** PID — отбор по идентификатору процесса;
- **--sid-owner** SID — отбор по идентификатору сеанса.

Например, выделить все пакеты, исходящие от пользователя с UID 500, можно так:

```
iptables -A OUTPUT -m owner --uid-owner 500
```

Аналогично мы можем ограничивать исходящие пакеты группы или процесса:

```
iptables -A OUTPUT -m owner --gid-owner 0
iptables -A OUTPUT -m owner --pid-owner 78
```

Естественно, мы можем сделать это только для исходящих пакетов, поскольку мы не знаем, какой UID имеет пользователь другой системы: информация об этом не передается по протоколу TCP.

Глава 20

НАСТРОЙКА ЯДРА

- МНОГООБРАЗИЕ ЯДЕР LINUX
- ЗАЧЕМ НАСТРАИВАТЬ ЯДРО?
- ДИНАМИЧЕСКИЕ ПАРАМЕТРЫ ЯДРА
- ЗАГРУЗОЧНЫЕ ПАРАМЕТРЫ ЯДРА
- КОМПИЛЯЦИЯ ЯДРА

20.1. Многообразие ядер Linux

В дереве развития ядер Linux достаточно много веток. И хотя в большинстве случаев придерживаются официальных версий ядер (ветка 2.6.x), упускать из внимания все остальные ни в коем случае нельзя.

2.6.x

Это главная ветка ядер Linux на настоящий момент, которую принято считать официальной и которая поддерживается самим Линусом Торвальдсом. Политика этой ветки более либеральна по сравнению с политикой ветки 2.4: в нее включаются не только исправления багов, но и такие изменения, для которых раньше заводили отдельную, экспериментальную, ветку. Все ядра этой ветки можно найти на сайте `www.kernel.org`.

2.4.x

В свое время ядра 2.4.x были основными — они встраивались во все дистрибутивы. Однако им на смену пришли ядра 2.6.x, и о ядрах классической ветки не стоило бы говорить, если бы не их исключительная стабильность (по сравнению с ядрами 2.6.x). И хотя они не обладают многими достоинствами и качествами 2.6.x, для тех, кому важна стабильность — они еще служат хорошую службу.

Более того, даже в некоторые современные дистрибутивы еще встраивается эти ядра (например, в Slackware 10.1). На сегодняшний момент эта ветка по прежнему очень активно поддерживается, а руководит данным процессом Марсело Тосатти, очень молодой программист бразильской компании Conectiva.

В качестве обновлений выступают исправления обнаруженных ошибок, ну и периодически — новые драйвера.

Так же как и ядра 2.6.x, ядра ветки 2.4.x. можно получить на сайте `www.kernel.org`.

2.6.x.y

Выше было сказано, что ядра ветки 2.6.x не отличаются особой стабильностью по сравнению с теми же 2.4.x. Даже были намерения сменить схему именования новых ядер. Однако этого не произошло, просто создали дополнительную ветку, которая выступает под грифом «stable» (то бишь стабильная), и которая известна под именем 2.6.x.y.

Ведут ветку –stable товарищи по имени Грег Кроа-Хартман и Крис Райт. В ней появляются только те обновления, которые непосредственно касаются повышения стабильности ядра и решают конкретные проблемы. Никакие обновления с новыми функциональными возможностями и т.п. в ядра этой ветки не вносятся.

2.6.x-mm

Если в предыдущем разделе мы рассмотрели наиболее стабильную ветку ядер Linux поколения 2.6, ветку 2.6.x-mm можно считать наименее стабильной. Она включает в себя все возможные патчи ядра, которые были выпущены.

Данная ветка является скорее экспериментальной, нежели имеющей практическое значение. Очень многое к Линусу в его ветку 2.6.x попадает именно через фильтр 2.6.x-mm. В частности сейчас в этих ядрах уже имеется поддержка файловой системы Reiser4, которая с легкостью выигрывает все тесты по производительности, и технологии FUSE (http://fuse.sourceforge.net), реализующей поддержку файловых систем в пользовательском пространстве.

Ведет ветку Эндрю Мортон, а патчи ветки доступны в двух видах:

- «все в одном» — единый мегапатч;
- «каждый сам по себе» — архив патчей, которые можно применять по отдельности.

Скачать их можно по адресу http://kernel.org/patchtypes/mm.html.

2.6.x-mm-jedi

Ветка 2.6.x-mm-jedi является, как это можно понять из названия, ответвлением 2.6.x-mm. И по сути считается stable-версией последней. Здесь собираются самые критичные исправления по различным версиям ядер 2.6.x-mm.

Доступ к данным патчам можно получить по адресу ftp://ftp.c9x.org/pub/linux-kernel.

2.6.x-pre и 2.6.x-rc

Ядра этих веток представляют собой предварительные версии ядер, которые впоследствии должны стать официальными. При этом принято считать, что ядра **-pre** являются наиболее «сырыми» и в них могут быть привнесены существенные изменения, а ядра **-rc** — уже более-менее стабильными.

2.6.x-tiny

Автор и ведущий данной ветки, Мэтт Мэколл, поставил себе задачу минимизировать занимаемые ядром дисковое пространство и объем памяти. Автор утверждает, что его ядро может быть запущено даже на машине с 2 Мб памяти.

Детально ознакомиться с этим направлением и получить само ядро можно на сайте http://selentic.com/tiny.

2.6.x-ac

Эта ветка, ведомая Аланом Коксом, призвана аккумулировать все исправления и патчи, касающиеся в основном драйверов. Кстати говоря именно это ядро используется в дистрибутивах Fedora Core.

Проект доступен по адресу http://kernel.org/patchtypes/ac.html

Прочие

Среди прочих следует также отметить:

- **2.6.x-ck** — в этой ветке аккумулируются патчи к диспетчерам ввода/вывода, а также направленные на общее повышение отзывчивости системы. Адрес: http://ck.kolivas.org/patches/2.6.
- **2.6.x-RT** — данная ветка направлена на обеспечение использования Linux в системах реального времени (мягкого и жесткого), насколько это вообще возможно сделать применительно к Linux.

20.2. Зачем настраивать ядро?

В главе 7 я уже говорил о том, что в обязанности ядра ОС Linux входит не только реализация концепций процессов, виртуальной памяти, файловой системы и тому подобных составляющих UNIX, но и организация взаимодействия с оборудованием компьютера. Это взаимодействие осуществляют драйверы устройств, которые в современных (выше 2.0)

ядрах могут как встраиваться в ядро статически, так и подключаться в виде модулей.

Хорошо известно, как многочисленно и разнообразно оборудование персональных компьютеров, на которых и работает ОС Linux. Ядро должно уметь адаптироваться к любой аппаратной среде. Этой цели служат параметры ядра — динамические или сообщаемые ему при загрузке — и механизм динамического подключения модулей.

Производители дистрибутивов поставляют ядро в некоторой базовой конфигурации, работающей на множестве вариантов оборудования, но не оптимизированной ни под один из них. Переконфигурировав ядро именно под свою аппаратную среду и задачи, можно существенно повысить производительность и увеличить надежность системы.

В этой главе я рассмотрю три основных способа настройки ядра:

- модификация динамических параметров ядра через псевдофайловую систему `/proc`;
- использование загрузчика для передачи ядру параметров на этапе начальной загрузки;
- пересборка ядра из исходных кодов.

20.3. Динамические параметры ядра

Файлы в каталоге `/proc` — это на самом деле информационные каналы, реализующие интерфейс между ядром и прикладными программами. Они разработаны для повышения гибкости ядра, позволяя системному администратору корректировать его поведение «на лету», без перезагрузки. Файловая система procfs называется виртуальной, потому что в действительности это карта работающей системы в иерархическом виде, создаваемая ядром и присоединяемая к обычной файловой системе. Некоторые команды (например, ***ps***) извлекают информацию о состоянии системы, не прибегая к системным вызовам, а читая непосредственно из `/proc`.

Хотя часть виртуальных файлов содержит текстовые данные, для их просмотра и изменения обычный ASCII-редактор неприменим: ведь они мало того что не существуют физически, так еще и ядро может в любое время внести в них изменения. Читать такие файлы нужно командой ***cat***, а записывать — перенаправляя вывод команды ***echo***. Например, чтобы увидеть максимальное количество файлов, которые можно открыть в одном процессе, введите команду

```
$ cat /proc/sys/fs/file-max
```

Большинство параметров ядра, предназначенных для динамической настройки на работающей системе, представлены файлами в каталоге

/proc/sys. Эти файлы доступны для записи суперпользователю. Например, если у вас обычный домашний компьютер, подключенный к Интернету по DSL или локальной сети, то производительность сети можно повысить, выключив некоторые параметры:

```
# echo "0" > /proc/sys/net/ipv4/tcp_sack
# echo "0" > /proc/sys/net/ipv4/tcp_timestamps
```

Особый интерес с точки зрения повышения производительности системы представляет коэффициент подкачки, находящийся в псевдофайле /proc/sys/vm/swappiness. Этот коэффициент, значение которого может быть от 0 до 100, указывает ядру, как часто следует выгружать страницы памяти на диск (свопить). Высокий коэффициент подкачки уместен в том случае, если вы работаете с несколькими громоздкими приложениями и переключаетесь между ними нечасто: каждое приложение будет работать быстрее, но переключение займет больше времени, ведь приложение, которое вы оставили без внимания на несколько минут, давно выгружено в своп-раздел. Например, если вы дизайнер и с утра до вечера не выходите из GIMP, то установите значение коэффициента равным 100:

```
# echo "100" > /proc/sys/vm/swappiness
```

Производительность вашего основного приложения (GIMP) станет максимальной. Если же вы целый день работаете с небольшими программками, между которыми часто переключаетесь, то установите коэффициент подкачки около 20. Значение по умолчанию равно 70. Возможно, вам больше всего подойдет именно это значение.

Значения динамических параметров ядра при перезагрузке не сохраняются. Чтобы сделать их постоянными, нужно вписать строки вида <параметр>=<значение> в файл /etc/sysctl.conf, откуда их в ходе начальной загрузки прочитает утилита ***sysctl***.

Имя параметра — это имя виртуального файла относительно каталога /proc/sys, в котором символ слэша заменяется на точку: vm.swappiness. Команда sysctl -a выводит список всех параметров, доступных для изменения.

20.4. Загрузочные параметры ядра

Полное описание параметров, которые можно передать ядру в ходе начальной загрузки, занимает достаточно много места, поэтому в этом параграфе я рассмотрю только основные из них. За более подробным их описанием вам следует обратиться к документу BootPrompt-HOWTO или к справочной системе (man bootparam). Большинство этих параметров

предназначено для того, чтобы сообщить ядру характеристики устройств, которые оно не может или не должно определить само.

Если ядро загружается средствами BIOS (например, с дискеты), то передать ему параметры невозможно: нужно использовать какой-либо загрузчик Linux. В главе 9 описано применение диспетчеров загрузки LILO и GRUB, то есть вы уже знаете, как указывать параметры в командной строке загрузчика или в его конфигурационном файле. Осталось разобраться с тем, какие это могут быть параметры.

Напоминаю, что синтаксис строки параметров следующий:

```
имя[=значение1][,значение2...] [имя2[=значение2.1][, значение2.2..]]
```

Значения разделяются запятой без пробелов, а параметры — пробелами. Пример строки параметров:

```
root=/dev/hda1 ether=9,0x300,0xd0000,0xd4000,eth0
```

20.4.1. Параметры корневой файловой системы

- **root**=устройство: указывает устройство, на котором находится корневая файловая система. В качестве устройства допустимо указывать:
  /dev/hdaN ... /dev/hddN — для IDE-дисков;
  /dev/sdaN ... /dev/sdeN — для SCSI-дисков;
  /dev/xdaN ... /dev/xdbN — для XT-совместимых дисков;
  /dev/fdN — флоппи-дисковод, где N=0 соответствует диску A:, а N=1 — диску B:
  /dev/nfs — псевдоустройство, указывающее ядру, что нужно загружаться по сети;
- **ro**: требует монтировать корневую файловую систему в режиме «только чтение». Используется по умолчанию;
- **rw**: задает монтирование корневой файловой системы в режиме «чтение/запись». При использовании этого параметра нельзя запускать программы типа ***fsck***. Перед запуском программы ***fsck*** нужно перемонтировать корневую файловую систему в режиме **ro**.

20.4.2. Объем памяти

Иногда нужно указать объем ОЗУ, отличный от того, который имеется на самом деле. Например, у вас чипсет Intel 810 с интегрированной видеоплатой, тогда вам нужно указать объем ОЗУ на 1 Мб меньше (а иногда даже на 2 Мб). Это связано с аппаратной особенностью чипсета. Более подробно об этом вы можете узнать на сайте компании Intel (`www.intel.com`).

Объем установленной памяти можно указать с помощью параметра mem:

```
mem= <число>
```

Число определяет объем памяти, установленной в компьютере, например, mem=16384K или mem=16M.

20.4.3. Управление RAMDISK

При создании загрузочных дискет для ОС Linux необходимо, чтобы на эти дискеты было помещено нужное программное обеспечение и чтобы для этого программного обеспечения хватило места. Обычно поступают следующим образом: создают сжатый архив всего необходимого программного обеспечения и помещают его на загрузочный диск. При загрузке системы в памяти создается виртуальный диск, на который это программное обеспечение распаковывается и записывается. Этот виртуальный диск называется RAM-диском.

Ядро не может быть включено в сжатый образ файловой системы RAM-диска, так как оно должно быть записано начиная с нулевого сектора, чтобы BIOS могло загрузить загрузочный сектор и ядро могло продолжить загрузку.

Если вы используете несжатый образ RAM-диска, то ядро может быть частью образа файловой системы. Такая дискета может быть загружена с помощью LILO.

В том случае, если вы для загрузки используете две дискеты (первая содержит ядро — boot, на второй находится образ файловой системы — root), образ файловой системы должен начинаться с нулевого сектора (смещение = 0).

Описываемые далее параметры задают режимы работы с RAM-диском.

- **load_ramdisk**=N: указывает, использовать RAM-диск (N=1) или нет (N=0). Значение по умолчанию равно 0.
- **prompt_ramdisk**=N: сообщает ядру, нужно ли запрашивать дискету, которая содержит образ файловой системы (N=1). Значение по умолчанию равно 1 (запрашивать).
- **ramdisk_start**=<смещение> : разрешает ядру находиться на дискете вместе со сжатым образом RAM-диска и указывает номер блока, с которого начинается RAM-диск.
- **ramdisk_size**=N: указывает максимальный размер (в Кб) RAM-диска. Начиная с версии ядра 1.3.48, память под виртуальный диск выделяется динамически. Значение по умолчанию равно 4096 (4 Мб).
- **noinitrd**: загружаться без использования initrd (см. п. 9.1.1.1).

20.4.4. Управление планировщиком ввода/вывода

Каждой программе, работающей под Linux, время от времени необходим доступ к диску. Ядро Linux решает, когда именно программа получит этот доступ. Часть ядра, отвечающая за планирование ввода/вывода, называется планировщиком ввода/вывода. Параметр **elevator** предназначен для указания планировщику алгоритма работы. Существует четыре различных алгоритма работы планировщика:

- Режим по умолчанию (noop) — для настольного компьютера он не подходит, и мы его даже рассматривать не будем.
- Упреждающее планирование (*Anticipatory Scheduling*) — при чтении программой данных с диска ядро пытается предугадать, какие данные программа будет читать при следующей операции чтения. Если ядро правильно угадало «мысли» программы, этот алгоритм позволяет существенно повысить производительность системы. Кроме всего прочего, эффективность этого алгоритма сильно зависит и от логики программы.

  ```
  elevator=as
  ```

- «Справедливая» очередь (*Complete Fairness Queuing*) — равные права для всех программ. Ядро равномерно планирует операции ввода/вывода для каждой программы, здесь нет каких-либо программ, которые могут монополизировать доступ к диску. Если несколько программ одновременно запросят доступ к диску, все программы получат ответ. Данный метод в некоторых случаях позволяет повысить производительность системы, а в других, наоборот, снижает общую производительность — все зависит от набора задач, решаемых на конкретной системе.

  ```
  elevator=cfq
  ```

- Deadline-планирование или планирование крайних сроков (*Deadline Queuing*) — все приложения, запросившие доступ к диску, ставятся в очередь. Из очереди извлекается одна программа, которая и получает практически монопольный доступ к диску. Пока эта программа работают, все остальные ожидают в очереди. По истечении определенного времени планировщик переводит эту программу в состояние ожидания и переключается на другую программу — следующую в очереди. Теперь вторая программа получает доминирующий доступ к диску. Потом третья, четвертая и т.д. Этот метод хорош для сервера баз данных, но не для рабочей станции.

  ```
  elevator=deadline
  ```

У каждого алгоритма есть свои преимущества и недостатки. Но только два из них подходят для настольного компьютера (рабочей станции) — второй и третий. В Интернете вы можете найти данные о том, что для настольного компьютера более подходит второй алгоритм.

Для выбора нужного режима перезагрузите компьютер и при загрузке ядра Linux передайте ему параметр elevator с одним из перечисленных выше значений.

20.4.5. Другие параметры ядра

- **debug**: сообщения ядра (важные и не очень) передаются через функцию printk(). Если сообщение очень важно, то его копия будет передана на консоль, а также функции klogd() для его регистрации на жестком диске. Сообщения передаются на консоль, потому что иногда невозможно запротоколировать сообщение на жестком диске (например, отказ самого диска). Предел того, что будет отображаться на консоли, задается переменной console_loglevel. По умолчанию на консоли отображается все, что выше уровня DEBUG (7). Список уровней можно найти в файле kernel.h.
- **init**=/путь: по умолчанию ядро пытается запустить демон ***init***, который продолжит загрузку согласно стартовым сценариям. Если программа ***init*** повреждена, то для аварийно-восстановительных работ вы можете запустить вместо нее командный интерпретатор (**init**=/bin/sh), в котором и будете ремонтировать систему.
- **no-hlt**: процессоры 386 (и выше) имеют инструкцию hlt, которая переводит процессор в режим пониженного потребления энергии, где он ожидает прерывания от устройства. Параметр **no-hlt** отключает использование инструкции hlt. Существование этого параметра обусловлено тем, что некоторые чипы 486DX-100 имеют проблемы с этой инструкцией. Кроме того, параметр **no-hlt** позволяет использовать Linux на бракованных процессорах.
- **no387**: отключает использование математического сопроцессора.
- **no-scroll**: отключает функцию прокрутки экрана во время загрузки.
- **reboot**=<режим>: параметр, задающий режим перезагрузки. Возможные значения: cold и warm, то есть «холодная» или «горячая» перезагрузка. Поддерживается ядрами версии 2.0 и выше.
- **single**: загружает систему в однопользовательском режиме — например, для ремонта.

20.5. Компиляция ядра

20.5.1. Зачем обновлять ядро?

Linux развивается быстрее любой другой операционной системы. Регулярно появляются новые версии ядра, реализующие новые функции. Например, едва успел выйти дистрибутив Fedora Core 4 на ядре 2.6.11, а

на www.kernel.org уже лежит стабильная версия 2.6.12.2. Еще чаще появляются драйверы к новым устройствам и заплатки с исправлением обнаруженных ошибок (например, прорех в системе защиты).

Допустим, новое ядро целиком еще можно установить из бинарного RPM-пакета (см. п.7.4.2). Но для подключения нового драйвера недостаточно просто скопировать его в каталог ядра: нужно интегрировать его код в структуры данных и таблицы ядра, для чего ядро придется строить заново из исходного кода. Исходные коды тем более необходимы для прикладывания заплатки (см. п. 7.4.3).

Пакет исходных текстов ядра может быть включен в ваш дистрибутив. Если это не так, скачайте архив с ftp://ftp.kernel.org/pub/linux/kernel (его размер 30-40 Мб в зависимости от версии) и распакуйте его в тот каталог, в котором будете производить сборку (обычно /usr/src/linux-<версия>).

20.5.2. Конфигурирование ядра

Когда вы строите из исходников прикладную программу, первым шагом сборки обычно бывает выполнение сценария configure. Ядро тоже нужно конфигурировать. Его настройки находятся в текстовом файле .config в каталоге исходных кодов. Этот файл можно редактировать вручную, но это крайне неудобно, поэтому предусмотрены диалоговые конфигураторы (п.7.2.3, рис. 7.2).

Если вы работаете в среде GNOME, выполните команду make gconfig, и вы увидите диалоговое окно, в котором можно выбрать нужные функции и драйверы устройств (Y) и отключить ненужные (N), а также указать, как следует включать в ядро выбранные устройства: статически или в виде модулей (M).

Осталось перечислить модули, доступные для конфигурирования, и дать рекомендации по их включению и отключению. Я расскажу об этом на примере современного ядра 2.6. О настройке ядра версии 2.4 рассказано в третьем издании моей книги «Самоучитель Linux», вышедшем в 2004 г. в издательстве «Наука и Техника», а если вы все еще используете ядро версии 2.2, то вам нужно первое издание этой книги.

20.5.2.1. Code maturity level options

Этот раздел позволяет включить в ядро экспериментальные модули, находящиеся еще в стадии разработки и предназначенные не для широкой публики, а для тестеров.

20.5.2.2. General setup

Support for paging of anonymous memory

Грозно звучит, не так ли? Я сначала даже не понял, что это. Оказывается, это просто поддержка свопа — своп-устройств и своп-файлов. Настоятельно рекомендуется не отключать эту опцию — сколько бы ни было оперативной памяти, а своп все равно пригодится.

System V IPC

Поддержка средств межпроцессного взаимодействия (*InterProcess Communication*) System V: очередей сообщений, семафоров, разделяемой памяти и т.д. Отключать не нужно, иначе процессы не смогут «общаться» друг с другом.

BSD Process Accounting

Учет процессов. С помощью специального системного вызова пользовательская программа может попросить ядро записать в специальный файл системную информацию: время создания процесса, идентификатор владельца, командную строку, использование ресурсов, например, памяти и терминалов и т.д. Чтобы все работало как нужно, не отключайте эту опцию.

Sysctl support

Включает поддержку программы *Sysctl*, позволяющей изменять параметры ядра без перекомпилирования во время загрузки. Поддержка *Sysctl* увеличивает размер ядра на 8Кб. Если ядро, которое вы компилируете, не предназначено для дисков загрузки/восстановления, включите эту опцию.

Kernel log buffer size

Задает размер буфера протокола ядра в зависимости от значения, указанного в программе конфигурирования ядра:

- 17 — 128 Кб (по умолчанию)
- 16 — 64 Кб
- 15 — 32 Кб (рекомендуется для SMP)
- 14 — 16 Кб
- 13 — 8 Кб
- 12 — 4 Кб.

Kernel .config support

Поддержка файлов `.config`, содержащих конфигурацию ядра.

20.5.2.3. Loadable module support

Если вы планируете использовать загружаемые модули, включите все функции. Можно создать компактную версию ядра, которая вообще не использует модули, при этом поддержка всех необходимых устройств будет включена непосредственно в ядро. В этом случае можно отключить все функции в этом разделе.

Enable loadable module support

Включить поддержку загружаемых модулей. Рекомендуется не отключать эту опцию, если вы собираете обычное ядро для настольной системы или сервера. Если же вы собираете компактное ядро, можно эту опцию выключить, а все необходимые модули включить в состав ядра.

Module unloading

Разрешить удаление модулей из ядра. Если эта опция выключена, вы не сможете удалить модуль из ядра после того, как он был загружен.

Forced module unloading

Принудительное удаление. Модуль будет удален из ядра, даже если какой-то процесс его использует. Такое удаление может быть опасным — ведь в результате удаления модуля устройства, которое в данный момент используется процессом, очень велика вероятность потери данных, поэтому лучше эту опцию не включать. Наоборот, когда вы занимаетесь программированием модулей, то есть созданием своих модулей, эта опция очень полезна, поскольку помогает сразу же выгрузить некорректно работающий модуль.

Module versioning support

Экспериментальная поддержка версий модулей. Включение данной опции позволяет использовать модули, откомпилированные для другой версии ядра. Нет никакой гарантии, что модуль, откомпилированный под другую версию ядра, будет работать стабильно с вашей версией, поэтому лучше выключите эту опцию.

Automatic kernel module loading

Обычно некоторые части ядра выполнены в виде модулей ядра. Когда ядру нужен тот или иной модуль, перед использованием модуля оно должно загрузить его (команда insmod). Если данная опция включена, ядро сможет автоматически загружать необходимые модули. Поэтому рекомендуется включить эту опцию.

20.5.2.4. Processor type and features

Здесь можно указать тип процессора и его функции, например, поддержка памяти более 1 Гб, MTRR, эмулирование математического сопроцессора.

Subarchitecture type

Тип архитектуры процессора:

- PC-compatible — PC-совместимый процессор, то есть процессор, использующий систему команд x86;
- Voyager (NCR) — SMP-архитектура, разработанная компанией NCP Corp;
- NUMAQ — позволяет запускать Linux на архитектуре NUMA (IBM/Sequent);
- SGI 320/540 — графические станции SGI.

Processor family

Эта функция используется для оптимизации работы процессора. Очень важно правильно указать тип процессора: после того, как я это сделал, производительность системы повысилась примерно в полтора раза, что стало заметно при загрузке системы. Если вы укажете тип процессора, например 486, 586, Pentium, PPro, ядро не обязательно будет запускаться на более ранней архитектуре. Так, если вы укажете Pentium, ядро будет работать на PPro (хотя и медленнее), но нет никакой гарантии, что оно запустится на 486. В табл. 20.1 приведены типы процессоров, которые рекомендуются для получения наибольшей производительности.

Типы процессоров *Таблица 20.1*

Тип	Процессоры
386	Процессоры производства AMD/Cyrix/Intel 386DX/DXL/SL/SLC/SX, Cyrix 486DLC/DLC2, UMC 486SX-S
486/Cx486	AMD/Cyrix/Intel/IBM DX4, 486DX/DX2/SL/SX/SX2 AMD/Cyrix 5x86 NexGen Nx586, UMC U5D или U5S
586/K5/5x86/6x86/6x86MX	Обычные (самые первые) процессоры Pentium, AMD K5. Не используются инструкции RDTSC (Read Time Stamp Counter)
Pentium-Classic	Классические процессоры Pentium — без поддержки MMX. Используются инструкции RDTSC

Продолжение табл. 20.1

Тип	Процессоры
Pentium-MMX	Процессоры Pentium с поддержкой MMX
Pentium Pro	Процессоры Pentium Pro/Celeron/Pentium II
Pentium II/Celeron (pre-Coopermine)	Процессоры Pentium II и Celeron (версия, которая была до Coopermine)
Pentium III/Celeron (Coopermine)/ Pentium III Xeon	Процессоры Pentium III и Celeron (Coopermine)
Pentium M	Мобильная версия процессора Pentium с пониженным энергопотреблением для ноутбуков
Pentium 4 Celeron (P4 based) / Pentium 4 M / Xeon	Процессоры Pentium 4, включая версию Pentium 4 M
K6/K6 II/K6 III	Процессоры AMD K6, K6-II, K6-III
Athlon/Duron/K7	Процессоры AMD Athlon/Duron
Opteron/Athlon64/ Hammer/K8	64-разрядные процессоры фирмы AMD
Elan/Crusoe	Процессоры Elan/Crusoe
WinChip-C6	Процессоры WinChip C6
WinChip 2A/Winchip-3	Процессоры WinChip 2A/Winchip-3
Cyrix III/C3	Процессоры IBM Cyrix III/C3

В моем случае ядро было оптимизировано под 586/K5. После того, как я установил Athlon/Duron/K7, Linux заработала быстрее (для справки: тогда у меня был AMD Duron 1.6 ГГц). Выбирайте именно ваш процессор — ваш классический Pentium не заработает быстрее и у него не появится поддержки MMX, если вы выберете Pentium MMX. Если вы попытаетесь таким образом ввести систему в заблуждение, она вообще может зависнуть, поскольку будет пытаться использовать MMX-инструкции, которые не поддерживаются процессором.

Generic x86 support

Базовая поддержка команд x86.

HPET Timer Support

HPET — это следующее поколение таймеров, пришедшее на смену классическому таймеру 8254. Просто включите эту опцию. Даже если ваша BIOS не поддерживает HPET, будет активизирована поддержка классического таймера 8254.

Symmetric multi-processing support

Скорее всего, у вас установлен один процессор. Тогда эту опцию вам нужно отключить — зачем включать лишний код в ядро? Если же вы счастливый обладатель мультипроцессорной машины, включите данную

опцию. При включении SMP укажите правильный тип процессора. Вы должны указать хотя бы 586. Ядро не запустится, если у вас выбран тип процессора 486. Также ядро не будет работать, если ваш компьютер оснащен процессором Pentium, а вы установили тип процессора PPro. Если у вас мультипроцессорная машина, вы должны также включить опцию **Enhanced Real Time Clock Support**. Опция **Advanced Power Managament** у вас будет отключена при использовании SMP.

Preemptible kernel

Данную опцию следует включить, если вам нужно ядро для RealTime-системы.

Local APIC support for uniprocessors

Поддержка внутреннего контроллера прерываний процессора (*Advanced Programmable Interrupt Controller*). Если у вас однопроцессорная система с процессором, оснащенным APIC, включите эту опцию. Если ваш процессор не поддерживает APIC, включение данной опции существенно снизит производительность системы.

У вас многопроцессорная система? Тогда APIC будет использоваться по умолчанию вне зависимости от значения этой опции.

Machine Check Exception

Позволяет процессору сообщать ядру о внутренних проблемах, например, о сбое.

Toshiba, DELL laptop support

Поддержка ноутбуков фирм Toshiba и DELL.

/dev/cpu/microcode

Включив эту опцию, вы сможете обновлять микрокод процессоров P Pro/ P II/ P III/ P4/ Xeon с помощью устройства /dev/cpu. Для работы этой опции нужно включить файловую систему /dev в разделе **File systems**. Информацию о микрокоде вы можете получить по адресу: `http://www.urbanmyth.org/microcode`. Если вы откомпилировали эту опцию как модуль, для его загрузки нужно прописать в вашем файле `/etc/modprobe.conf` строку:

```
alias char-major-10-184 microcode
```

/dev/cpu/*/msr

Поддержка регистров MSR. Может понадобиться в некоторых случаях для SMP-систем. Вы можете ее со спокойной совестью выключить или хотя бы откомпилировать как модуль.

/dev/cpu/*/cpuid

Поддержка информации о процессоре. Рекомендуется включить эту опцию. Загляните в файл /dev/cpu/0/cpuid — вы узнаете много интересного о своем первом (0) процессоре.

High Memory Support

Поддержка памяти более 1Гб.

Math emulation

Включите эту опцию, если вы используете один из следующих процессоров: 386SX/DX/SL/SLC без 80387, 486SL/SX/SX2.

MTRR

В семействе процессоров Intel P6 (Pentium Pro, Pentium II и выше) используются специальные регистры — Memory Type Range Registers (MTRR). Они задействуются для управления доступом процессора к различным диапазонам памяти. Включение этой опции может существенно повысить производительность системы, особенно если вы используете видеокарту PCI или AGP. Данную возможность поддерживают процессоры и посторонних производителей: Cyrix 6x86, 6x86MX, MII, AMD K6-2 (stepping 8 и выше), K6-3, Centaur C6. Некоторые BIOS устанавливают MTRR для первого процессора, но отключают для второго. Активизация данной опции решает также и эту проблему. Если вы не уверены, поддерживает ли ваш процессор MTRR, все равно включите эту опцию. Поддержка MTRR увеличит объем ядра всего лишь на 3Кб.

20.5.2.5. Power Management Options

В этом разделе вы найдете все опции, касающиеся управления питанием. В принципе, с опциями Power Management разобраться несложно и самому, в крайнем случае можно все оставить по умолчанию — опции вполне приемлемы. Но есть одна очень интересная экспериментальная (!) опция — **Software Suspend**. Если она включена, вы можете приостановить машину, а при следующей загрузке сразу же восстановить систему до того состояния, в котором она находилась до приостановки. Работает это так:

вы вводите команду swsusp или shutdown -z <время>, система записывает образ содержимого памяти на своп-раздел (или в своп-файл). Затем, при загрузке, вы вводите параметр ядра `resume=/dev/своп-раздел`, и система восстанавливается до исходного состояния. Если вы не ввели данный параметр, то будет невозможно задействовать своп-раздел, который использовался для приостановки системы (создания образа памяти). Опция **Suspend to disk** позволяет записывать образ памяти на диск, а не только на своп-раздел.

В этом же разделе можно включить опцию **Advanced Power Management**. Если вам нужно отключить функцию APM во время загрузки, введите в качестве параметра ядра apm=off.

При возникновении проблем (на старых компьютерах) попробуйте следующее:

1. Убедитесь, что раздел подкачки включен, а размер его достаточен.
2. Передайте ядру инструкцию **no-hlt**.
3. Попробуйте отключить поддержку сопроцессора (инструкция **no387**).
4. Передайте ядру инструкцию **floppy-nodma**.
5. Убедитесь, что процессор не «разогнан».
6. Установите новый вентилятор для процессора.

Что же касается APM, следует обратить внимание на следующие опции (таблица 20.2):

Опции APM *Таблица 20.2*

Опция	Описание
Enable PM at boot time	Включает APM во время загрузки системы. Если эта опция отключена, BIOS не будет управлять питанием устройств, входить в режимы **Standby** и **Suspend**, а также не будет производить никаких действий в ответ на вызовы процессора **CPU Idle**. Если ваш компьютер зависает во время загрузки, выключите эту опцию
Make CPU Idle calls when idle	Во время цикла простоя ядра разрешает вызовы к APM. Включение данной опции может привести к зависанию компьютера во время загрузки! Если компьютер использует несколько процессоров, эта опция игнорируется. Заметьте, речь идет о том, сколько процессоров именно *использует компьютер*, а не сколько их в нем установлено. Если у вас два процессора, а вы используете только один и поддержка SMP у вас отключена, данная опция игнорироваться не будет!
Enable console blanking using APM	Очищает текстовую консоль (не графическую!) при использовании APM. Некоторые ноутбуки могут использовать эту опцию для того, чтобы отключить подсветку LCD-экрана, когда активизирован хранитель экрана на одной из виртуальных консолей Linux.
RTC stores time in GMT	Если ваш аппаратный таймер сохраняет время в формате GMT, включите эту опцию, в противном случае отключите ее. Если опция выключена, сохраняется локальное время
Allow interrupts during APM BIOS calls	Обычно прерывания внешних устройств запрещены во время выполнения процедур APM. BIOS некоторых ноутбуков разрешает прерывания внешних устройств, например, IBM ThinkPad. По умолчанию данная опция выключена. Если вы не уверены, не включайте ее

20.5.2.6. Bus Options

В этом разделе описываются опции, касающиеся поддержки различных шин PCI, ISA, MCA. Конфигуратор ядра 2.4 держал их в разделе **General Setup**, а сейчас они вынесены в собственный раздел. Отключайте все, кроме шины PCI — не думаю, что у вас есть ISA или MCA-устройства.

Примечание

MCA — шина передачи данных, разработанная IBM, которая использовалась в системах PS1/PS2. Она давно снята с производства, современные устройства с ней не работают.

PCI access mode

Данная опция определяет режим доступа к PCI-устройствам. Если значение этой опции равно BIOS, значит, Linux будет использовать BIOS для определения PCI-устройств и их конфигураций. Однако на некоторых старых материнских платах BIOS не может корректно определить конфигурацию PCI-устройств. В этом случае нужно выбрать значение Direct, и Linux будет работать с PCI-устройствами напрямую, без BIOS. Если вы выберете Any, то Linux сначала попытается работать напрямую (так быстрее), а потом, если напрямую не получится, уже использовать BIOS.

20.5.2.7. Executable file formats

В этом разделе вы сможете включить поддержку различных форматов исполняемых файлов. Обычно это нужно, если вы хотите запускать в эмуляторах программы других операционных систем, например, DOS- или Windows-программы.

20.5.2.8. Device drivers

В этом разделе находятся опции, касающиеся драйверов устройств. Тут вы можете определить, какие устройства у вас установлены и какие вы хотите использовать в дальнейшем. По сравнению с программой настройки ядра 2.4 многие отдельные разделы «переехали» в этот раздел, например, **Parallel port support**, **MTD**, **PnP support** и т.п.

Memory Technology Devices (MTD)

В этом подразделе вы можете включить поддержку MTD-устройств. Самым ярким примером такого устройства может послужить flash-диск.

Parallel port support

Поддержка параллельного порта.

PnP support

Поддержка устройств Plug and Play.

Block devices

В этом подразделе вы можете настроить блочные устройства, то есть такие устройства, обмен данными с которыми выполняется поблочно (передаются целые блоки информации), а не посимвольно (за одну операцию ввода/вывода передается один байт). Ярким примером блочных устройств выступают дисковые накопители — дисковод для гибких дисков, жесткие диски и т.п.

Normal floppy disk support

Поддержка обычного дисковода для гибких дисков. Обычно поддержка данного устройства требуется, но если вы собираете ядро для сервера или ноутбука, то «флоппик» ему не нужен, поэтому можете эту опцию выключить.

XT hard disk support

Поддержка старых жестких дисков, которые устанавливались на компьютеры типа IBM XT. Отключите эту опцию — она вам не нужна, модуль тоже вам не нужен. Даже если вы где-то и найдете такой диск, вряд ли вы станете его подключать к современному компьютеру.

Parallel port IDE device support

Поддержка IDE-устройств, которые подключаются к компьютеру по параллельному порту. Очень часто к ноутбукам подключаются внешние IDE-устройства. Было время, когда шины USB не существовало, тогда был разработан интерфейс для подключения IDE-устройств по параллельному порту (по последовательному порту обмен информацией занимает много времени).

Compaq SMART2 support

Поддержка контроллеров SMART2 фирмы Compaq. Вряд ли этот параметр вам понадобится.

Compaq Smart Array 5xxx support

То же самое, что и предыдущая опция, но здесь включается поддержка контроллеров 5xxx.

Mylex DAC960/DAC1100 PCI RAID Controller support

Поддержка RAID-контроллера фирмы Mylex.

Micro Memory MM5415 Battery Backed RAM support (EXPERIMENTAL)

Поддержка специальных карт памяти. Честно говоря, я эти карты и в глаза не видел, и в руках не держал. Если кому-то интересна информация о них, посетите сайт: http://www.umem.com/.

Loopback device support

Вот эту опцию я отключать не советую — многие задачи требуют наличия устройства обратной петли. Поэтому, если хотите сделать ядро компактнее, включите эту опцию хотя бы в виде модуля. Модуль будет называться loop.

Network block device support

Поддержка сетевых блочных устройств. Просто включите эту опцию в виде модуля. Модуль называется nbd.

RAM disk support

Это очень полезная опция, позволяющая часть информации, находящейся на жестком диске, перенести в оперативную память для ускорения доступа к ней. Особенно данная опция полезна при создании загрузочных дисков, которые используются для восстановления системы. Если вы включите эту опцию в виде модуля, он будет называться rd.

Initial RAM disk (initrd) support

Инициализирующий RAM-диск — это RAM-диск, который загружается загрузчиком системы (LILO, GRUB, loadlin) и монтируется как корневая файловая система перед нормальной загрузкой. Он используется для загрузки модулей перед монтированием нормальной корневой системы. Включите эту опцию. Если же вы создаете загрузочный диск, то ее отключение недопустимо вообще!

Support for Large Block Devices

Поддержка больших блочных устройств — с размером более 2Тб. Отключайте эту опцию — жестких дисков на 2Тб в ближайшее время в продаже не предвидится.

ATA/ATAPI/MFM/RLL support

В этом подразделе вы можете включить/выключить поддержку ATA-устройств. Тут уж смотрите сами, какие устройства у вас есть и какие вы планируете использовать. Ненужные следует сразу отключать — нечего память забивать. Что нужно? Обыкновенный привод CD-ROM есть у всех. Даже если у вас в данный момент его нет, вы рано или поздно его подключите к своему компьютеру. Не будете же вы из-за этого перекомпилировать ядро? Поэтому опция **Include IDE/ATAPI CDROM support (BLK_DEV_IDECD)** относится к категории нужных. А вот поддержка **Silicon Image chipset** совершенно не нужна, хоть и встроена в ядро по умолчанию. Поддержку всех ATA-устройств следует отключить, если у вас сервер и все устройства — SCSI (только не забудьте SCSI включить!)

SCSI device support

Поддержка SCSI-устройств. Все комментарии аналогичны предыдущему пункту — все ненужное отключаем, оставляем самое необходимое. Вот почему мне нравится Linux — можно явно указать, что мне нужно, а что — нет. Ни в одной Windows такое сделать нельзя. Можно, конечно, удалить всю базу с драйверами, но я же сказал «явно».

Old CD-ROM drivers (not SCSI, not IDE)

В этом разделе вы можете включить поддержку старых (я бы сказал, древних) приводов CD-ROM, но, скорее всего, это вам не нужно, поэтому смело отключайте целый подраздел — сэкономим место на диске.

Multi-device support (RAID and LVM)

Поддержка RAID-массивов и LVM-томов (*Logical Volume Manager*). Если планируете использовать RAID, включите некоторые опции этого раздела. Вы можете указать, какие уровни RAID вам нужны, а какие нет.

IEEE 1394 (FireWire) support

Поддержка последовательной высокоскоростной шины IEEE 1394. Если у вас есть IEEE-адаптер, включите эту опцию. Внимание: поддержка IEEE экспериментальна (для ядра 2.6)!

QoS and/or fair queueing

В этом подразделе можно включить поддержку QoS (Quality of Service).

IrDA (infrared) support

Поддержка IrDA-устройств.

Bluetooth support

Поддержка Bluetooth. Обычно требует включения на современных ноутбуках.

ISDN subsystem

Поддержка технологии ISDN (Integrated Services Digital Networks, во Франции — RNIS). Технология ISDN постепенно уходит в прошлое — вместо нее используется ADSL. Если у вас есть возможность перейти на ADSL, сделайте это.

Telephony Support

Если у вас есть специальная карта, позволяющая подключить обыкновенный телефон для использования голосовых IP-приложений, включите поддержку телефонии. Вам не нужно включать поддержку телефонии для использования обычного модема.

Input device support

Поддержка различных устройств ввода — джойстиков, мышей, сенсорных панелей, клавиатур.

Character devices

Поддержка символьных устройств, например, стримеров.

Multimedia devices

Поддержка TV- и радиотюнеров.

Graphics support

Поддержка графических адаптеров. Выберите только те драйверы, которые необходимы для поддержки ваших видеокарт, а остальные отключите.

Sound

Поддержка систем ALSA (Advanced Linux Sound Architecture) и OSS (Open Sound System). Тут же драйверы звуковых плат.

USB support

Поддержка USB.

Networking support

Это довольно большой подраздел раздела **Device Drivers**, в котором можно включить поддержку как самой сети, так и отдельных ее компонентов. Поддержка сети нужна обязательно, даже если у вас нет сетевой платы или других сетевых устройств. Функции печати, а также графическая подсистема X Window требуют поддержки сети, а это значит, что, если сеть у вас отключена, вы не сможете ни документ распечатать, ни работать в графическом интерфейсе.

Сетевых опций довольно много, поэтому для их установки воспользуемся таблицей 20.3.

Опции сети *Таблица 20.3*

Опция	Назначение
Netlink device emulation	Опция обратной совместимости. Скоро будет удалена, но пока она нужна.
Unix domain sockets	Поддержка UNIX-сокетов. Не отключайте эту опцию
IPMI sockets	Поддержка IPMI-сокетов. Обычно не нужна
PF_KEY sockets	Требуется для IPsec, поэтому лучше не отключать ее или включить в виде модуля
TCP/IP networking	Поддержка TCP/IP обязательно должна быть включена!!!
IP: multicasting	Позволяет адресовать сразу несколько компьютеров. Данная опция полезна, если вы используете MBONE или другую магистраль для широковещательной передачи аудио- и видеоинформации
IP: advanced router	Включите, если предполагаете использовать данный компьютер в виде маршрутизатора, а заодно нужно включить и все ее компоненты, например, IP: policy routing — довольно интересная функция маршрутизатора. В двух словах ее не опишешь, а прочитать о ней можно по адресу: ttp://www.compendium.com.ar/policy-routing.txt
IP: kernel level autoconfiguration	Включает автоматическую конфигурацию IP-адреса сетевых устройств и таблицы маршрутизации во время загрузки ядра на основании информации, переданной ядру в командной строке или по протоколам BOOTP или RARP. Данную опцию имеет смысл включать только на бездисковых машинах, поскольку на обычных системах конфигурация сети задается в загрузочных сценариях
IP: tunnelling	Данная опция понадобится вам, если вы будете настраивать виртуальную частную сеть — VPN (Virtual Private Network)
IP: GRE tunnels over IP	Данная опция полезна при использовании маршрутизаторов Cisco. Для ее работы необходимо включить предыдущую опцию
IP: multicast routing	Данная опция позволяет настроить работу маршрутизатора так, чтобы он отправлял IP-пакеты по нескольким адресам. Очень полезно для широковещания аудио-видеоинформации по Интернету

Продолжение табл. 20.3

Опция	Назначение
IP: ARP daemon support (EXPERIMENTAL)	Поддержка ARP-демона. Можно включить на маршрутизаторе/шлюзе небольшой сети
IP: TCP syncookie support (disabled per default)	Вот эту опцию я бы включил из соображений безопасности, она поможет защитить вашу систему от так называемых SYN-наводнений, точнее, сообщит вам адрес атакующего хоста
IP: AH transformation	Поддержка AH-преобразования для IPSec. Если не уверены в том, что вы делаете, не отключайте эту опцию!
IP: ESP transformation	Поддержка ESP-преобразования для IPSec. Если не уверены в том, что вы делаете, не отключайте эту опцию!
IP: IPComp transformation	Поддержка IPComp-преобразования (сжатие данных, описано в RFC3173) для IPSec. Если не уверены в том, что вы делаете, не отключайте эту опцию!
IP virtual server support (EXPERIMENTAL)	Включение данной опции позволит вам построить виртуальный сервер, который будет использовать ресурсы нескольких физических серверов. Попросту говоря, данная опция позволяет собрать кластер. Раньше для создания кластеров использовались программные продукты посторонних разработчиков, а сейчас поддержка кластеров встроена в ядро. Если заинтересовались, посетите сайт: http://www.linuxvirtualserver.org/
The IPv6 protocol (EXPERIMENTAL)	Поддержка протокола IPv6. Пока он практически не используется, поэтому можно эту опцию смело отключить
DECnet Support	Данную опцию на просторах бывшего СССР вряд ли кто-то будет включать, точнее, вряд ли она кому-то понадобится
802.1d Ethernet Bridging	Если вы включите эту опцию, ваш компьютер превратится в Ethernet-мост, который будет соединять различные сегменты вашей локальной сети
Network packet filtering (replaces ipchains)	Поддержка нового поколения бастиона Netfilter, пришедшего на замену IPChains. Если вы настраиваете маршрутизатор, включите эту опцию. Для обыкновенных систем ее нужно выключить
IPsec user configuration interface	Поддержка IPSec. Если не уверены, просто включите эту опцию
Asynchronous Transfer Mode (ATM)	Поддержка ATM
The IPX protocol (IPX)	Поддержка протокола IPX (компания Novell)
Appletalk protocol support	Поддержка протокола компании Apple. Если в вашей сети есть хотя бы один Macintosh, включите эту опцию
CCITT X.25 Packet Layer (EXPERIMENTAL) (X25), LAPB Data Link Driver (EXPERIMENTAL) (LAPB)	Данные опции нужно включать, только если вам это действительно необходимо. Если вы не знаете, что они собой представляют, лучше их не трогать!
WAN router	Данная опция превращает ваш компьютер в маршрутизатор глобальной сети. Обычные маршрутизаторы не требуют включения этой опции. В случае включения данной опции в виде модуля модуль будет называться wanrouter
Fast switching	Перед включением этой опции настоятельно рекомендую прочитать документацию. Данная опция выбирает самый быстрый сетевой интерфейс для передачи данных. Внимание! Эта опция несовместима с опцией **Network packet filtering**
Forwarding between high speed interfaces	Не включайте эту опцию!

20.5.2.9.Filesystems

В разделе **Filesystems** вы можете включить поддержку следующих файловых систем:

- Second extended fs (ext2), до недавнего времени бывшей основной файловой системой Linux;
- Ext3 journalling file system — журналируемой версии ext2, используемой многими дистрибутивами в качестве основной файловой системы;
- ReiserFS — файловой системы Reiser;
- JFS filesystem — файловой системы JFS;
- XFS — файловой системы XFS;
- Minix FS — файловой системы Minix;
- CD-ROM/DVD Filesystems ISO 9660 — файловой системы, используемой для записи информации на CD-ROM.

Что включить, а что выключить? Первые две файловые системы, а также файловую систему ISO 9660 включите обязательно. Думаю, не нужно объяснять, почему. Файловую систему Minix можно сразу отключить — она давно устарела и не используется. Файловые системы Reiser, JFS, XFS относятся к разряду новых, но редко использующихся. В принципе, их нужно включить — вдруг кто-то принесет винчестер, на котором разделы будут содержать одну из этих файловых систем? Или просто вы захотите поэкспериментировать и отформатировать раздел в одной из этих систем.

Не забудьте включить средство автоматического монтирования сменных носителей — **Kernel automounter support**, особенно для рабочей станции! А вот поддержка квот (**Quota support**) окажется полезной, если вы настраиваете сервер.

В подразделе **DOS/FAT/NT Filesystems** вы можете включить поддержку следующих систем:

- Файловая система MS DOS. Включить ее нужно обязательно — в странах бывшего СССР до сих пор встречаются дискеты, записанные в этой файловой системе.
- VFAT (Windows-95): это основная файловая система операционных систем Windows 95 и 98.
- NTFS — файловая система ОС Windows NT, 2000, XP. Здесь же можно включить поддержку записи на раздел NTFS, которая по умолчанию отключена.

В разделе **Pseudo filesystems** вы можете включить так называемые псевдосистемы — файловые системы `/proc` и `/dev`.

В разделе **Miscellaneous filesystems** находятся опции включения поддержки других, редко используемых файловых систем, например, HPFS (High Performance File System), которая используется по умолчанию ОС IBM OS/2.

Включить поддержку файловых систем NFS и SMB (используется для монтирования удаленных Windows-разделов, читайте «общих дисков и папок») можно в разделе **Network File Systems**.

Раздел **Native Language Support** позволяет включить поддержку различных кодировок, в которых могут быть представлены имена файлов. Например, отключив кодировку cp-1251, при просмотре содержимого Windows-раздела вы увидите иероглифы вместо русских букв.

20.5.2.10.Kernel hacking

В этом разделе для вас найдутся две полезные опции, даже если вы не занимаетесь разработкой модулей ядра Linux. Опция **Prefer small over fast code** позволяет сделать ядро более маленьким, но более медленным. Маленький, но медленный код может пригодиться для создания загрузочной дискеты — там важен каждый байт. Вторую опцию **Kernel debugging** также можно отключить, если вы создаете системную дискету.

20.5.2.11.Cryptographic options

Различные опции, касающиеся криптографии.

В разделах **Library routines** и **Unofficial 3rd party kernel additions** я не нашел для себя ничего интересного.

20.5.3.Сборка ядра

Теперь, когда все устройства сконфигурированы, нужно сохранить файл конфигурации ядра и перейти непосредственно к этапу сборки ядра.

Для сборки вам понадобится программное обеспечение, необходимые версии которого перечислены в таблице 20.4.

Необходимое программное обеспечение *Таблица 20.4*

Программа/ библиотека	Минимально допустимая версия	Где взять
Gnu C Compiler	2.95.3	http://gcc.gnu.org
Gnu Make	3.78	ftp://ftp.gnu.org/gnu/make/
binutils	2.12	ftp://ftp.kernel.org/pub/linux/devel/binutils/
util-linux	2.10o	ftp://ftp.kernel.org/pub/linux/utils/util-linux/
module-init-tools	0.9.9	http://www.kernel.org/pub/linux/kernel/people/rusty/modules/

Продолжение табл. 20.4

Программа/ библиотека	Минимально допустимая версия	Где взять
procps	2.0.9	http://procps.sourceforge.net/
e2fsprogs (*)	1.29	http://e2fsprogs.sourceforge.net/
jfsutils (*)	1.0.14	http://www-124.ibm.com/jfs/
reiserfsprogs (*)	2.1.0	http://www.namesys.com/
nfs-utils	1.0.5	http://nfs.sourceforge.net/
pcmcia-cs	3.1.21	http://pcmcia-cs.sourceforge.net/
quota-tools	3.09	http://sourceforge.net/projects/linuxquota/
ppp	2.4.0	ftp://ftp.samba.org/pub/ppp/
isdn4k-utils	3.1pre1	http://www.isdn4linux.de/swpat.html
oprofile	0.5.3	http://oprofile.sourceforge.net/

(*) Данное программное обеспечение зависит от используемой файловой системы. Если вы используете только ext2, обновите только e2fsprogs, если JFS — то jfsutils. Если же вы используете все перечисленные файловые системы ext2, JFS, ReiserFS, вам нужно обновить все программы, отмеченные звездочкой.

Ваше старое ядро пока работает и, чтобы не сделать ничего непоправимого, нужно собирать новое ядро под новым именем. Найдите в Makefile (в самом начале файла) строчки:

```
VERSION=2
PATCHLEVEL=6
SUBLEVEL=<третья_цифра_версии_вашего_ядра>
EXTRAVERSION=
```

EXTRAVERSION — это суффикс, которым будет отличаться имя нового ядра. Дайте ему значение вроде «new» или «test». Это приведет к тому, что собранное вами ядро будет называться linux-2.6.x-new. Старое ядро никуда не денется, и при загрузке можно будет выбрать нужный вариант ядра.

Команда `make dep`, которая вводилась после конфигурирования ядер 2.4 и ниже, при сборке ядра 2.6 не используется. Вместо нее выполните следующую последовательность команд:

```
$ make bzImage
$ make modules
```

Эти пара команд соберет ядро и те модули, которые вы включили в него на этапе конфигурирования. Процесс сборки займет не меньше 20 минут, а то и значительно больше — в зависимости от быстродействия вашей системы и количества выбранных модулей.

Можно выполнять эти команды от имени непривилегированного пользователя. Даже нужно, поскольку идеологически правильнее работать под рутом только тогда, когда иначе нельзя. Без привилегий суперпользователя нельзя обойтись только на этапе установки ядра и модулей:

```
# make modules_install
# make install
```

Результатом успешной сборки и установки станут следующие файлы и каталоги:

```
/boot/vmlinuz-2.6.x-new
/boot/System.map-2.6.x-new
/boot/initrd-2.6.x-new.img
/lib/modules/2.6.x-new
```

Осталось добавить в конфигурационный файл вашего загрузчика (п.9.1.1) вариант загрузки с новым ядром. Если вы используете GRUB, впишите в `/boot/grub/grub.conf` следующие строки:

```
title Linux New Kernel
kernel /vmlinuz-2.6.x-new root=/dev/hda5 ro
initrd /initrd-2.6.x-new.img
```

Если вы вкомпилировали все драйверы, необходимые для загрузки системы, в ядро и поэтому не используете initrd, то строчку initrd можно удалить.

В случае, если ваш загрузчик — LILO, впишите в `/etc/lilo.conf` строки:

```
image=/boot/vmlinuz-2.6.x-new
 label=" Linux New Kernel"
 root=/dev/hda5
 initrd=/boot/initrd-2.6.x-new.img
 read-only
```

И занесите изменения в загрузочную запись:

```
# lilo
```

Теперь перезагрузите систему и попробуйте загрузиться с новым ядром. При появлении каких-либо ошибок вы всегда сможете загрузить старую версию.

Глава 21

СОЗДАЕМ КОНСОЛЬНОЕ ПРИЛОЖЕНИЕ

- КОМПИЛЯТОР GCC
- СБОРОЧНАЯ УТИЛИТА MAKE
- ПАКЕТ BINUTILS И ДРУГИЕ ПОЛЕЗНЫЕ ПРОГРАММЫ

21.1. Компилятор gcc

В 8 главе вы познакомились с языком командного интерпретатора и убедились, что на нем можно писать полезные сценарии. Но если вы раньше программировали под Windows, то интерпретатора вам маловато будет — понадобятся более мощные средства разработки. Разумеется, они в ОС Linux есть.

Основным С-компилятором в Linux служит **gcc** (GNU C Compiler). Если вы не установили средства разработки при установке дистрибутива, самое время сделать это сейчас. Запустите менеджер пакетов (рис. 21.1) и установите следующие пакеты:

- **gcc** — сам компилятор gcc;
- **gcc-c++** — поддержка C++;
- **gcc-doc** — документация по gcc (очень рекомендую установить);
- **compat-gcc** — пакет, содержащий средства gcc для обратной совместимости. Данный пакет вам нужен, если вы планируете писать программы для более старых систем, чем ваша;
- **compat-gcc-c++** — то же, что и compat-gcc, только с поддержкой C++;
- **compat-cpp** — препроцессор cpp для обратной совместимости;
- **binutils** — набор вспомогательных утилит, о которых мы поговорим в последнем пункте этой главы;
- **glibc-devel** — содержит библиотеки для разработки С-программ;
- **libbfd** — библиотека дескриптора двоичного файла (Binary File Descriptor library);
- **libstdc++-devel** — заголовочные файлы и библиотеки для программирования на C++;
- **make** — утилита, упрощающая процесс сборки больших проектов.

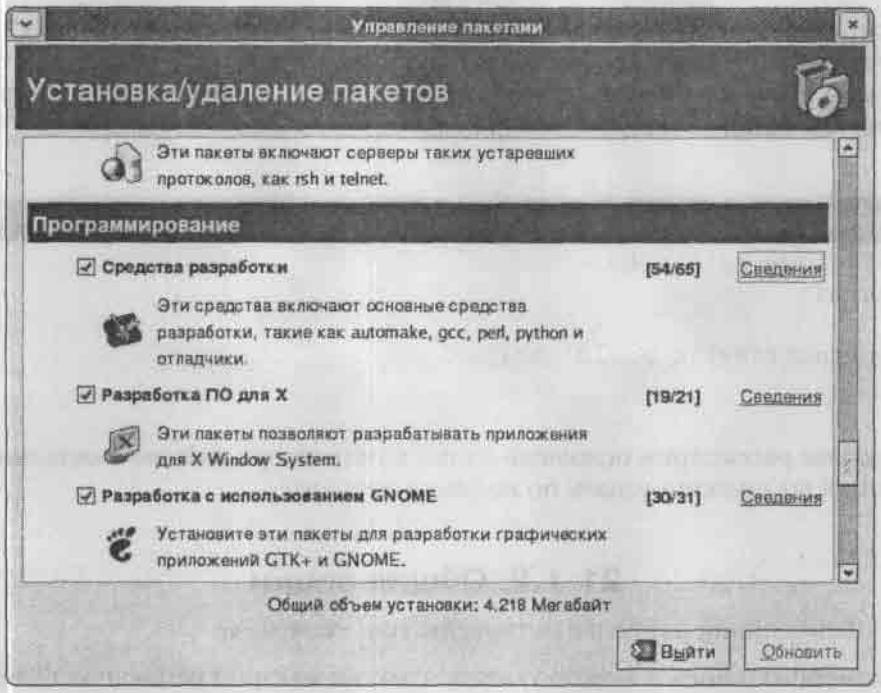


Рис. 21.1. *Менеджер пакетов Fedora Core — system-config-packages*

21.1.1. Вызов gcc

Формат вызова компилятора такой:

```
gcc [опции] helloworld.c
```

Вы написали свою программу на C++? Нет проблем: компиляторы C и C++ являются интегрированными. Чтобы сообщить компилятору, на каком языке вы писали свою программу, нужно указать определенную опцию при вызове компилятора. Но можно поступить проще: по умолчанию компилятор считает, что файлы с расширением .c написаны на языке C, файлы с одним из расширений .C, .cc, .cXX — на языке C++, а файлы с расширением .s — на языке ассемблера.

Как правило, перед компиляцией вызывается программа **cpp** — препроцессор языка C. Препроцессор создаст файл с расширением .i, если ваша программа написана на языке C, и файл с расширением .ii, если ваша программа написана на C++. Если ваша программа уже прошла препроцессорную подготовку, вы можете передать компилятору **gcc** файл с расширением .i или .ii и **gcc** уже не будет вызывать препроцессор для подготовки исходного текста. Как правило, программу **cpp** редко кто вызывает вручную.

Если не указывать никаких опций, то компилятор создаст в текущем каталоге файл a.out, содержащий исполняемый код. Для тех, кто изучал другой язык, я на всякий случай приведу текст первой программы на С (листинг 21.1).

Листинг 21.1. Файл helloworld.c

```
#include <stdio.h>
main()
{
 printf("Hello World!\n");
}
```

А сейчас рассмотрим основные опции компилятора. Обо всех остальных опциях вы сможете узнать по команде `man gcc`.

21.1.2. Общие опции

Из общих опций наиболее интересны три: **-x**, **-c** и **-o**.

С помощью опции -x можно указать язык, на котором написан исходный код, например,

```
gcc -x c helloworld.c
```

В качестве языка программы вы можете указать:

- **c**, если ваша программа написана на С;
- **objective_c**, если ваша программа написана на Objective C;
- **c-header**, если ваша программа является заголовком С;
- **c++**, если вы написали программу на С++;
- **assembler**, если вы написали программу на ассемблере.

Существуют и другие варианты, но они не столь важны.

Опция **-c** используется, если вы хотите только откомпилировать вашу программу, но не вызвать компоновщик. В результате будет создан объектный файл с расширением `.o`.

Опция **-o** позволяет указать имя результирующего файла. Это очень полезная опция, потому что имя `a.out` мало кого устраивает:

```
gcc -o helloworld helloworld.c
```

Очень полезна опция -v, которая выводит различную информацию о стадиях компиляции. Кроме того, эта опция выводит версию компилятора.

21.1.3. Опции языка

Из всех опций языка мне пригодилась лишь опция ANSI, которая выключает все функции GNU C, несовместимые со стандартом ANSI. К таким функциям относятся **asm**, **inline**, **typeof** и другие.

21.1.4. Опции препроцессора

Эти опции задают режим предварительной обработки исходного кода (до собственно компиляции).

Очень важной является опция **-include <файл>**. Она позволяет «прогнать» через препроцессор сперва содержимое указанного файла, а только после этого файл, который передан препроцессору. В результате указанный файл будет откомпилирован раньше, чем все остальные.

Опция **-nostdinc** запрещает использование системного каталога, содержащего файлы заголовков. При ее включении компилятор будет искать заголовки в каталогах, указанных в опции -I и в текущем каталоге.

Опция **–nostdinc++** запрещает использование стандартных файлов заголовков для языка C++.

21.1.5. Опции компоновщика

Опция компоновщика **-l** позволяет явно указать имя библиотеки, которая будет использоваться при сборке вашей программы. Например,

```
gcc -lmylibrary myfile.c
```

Компоновщик будет использовать файл `libmylibrary.a`, который он попытается найти в системных каталогах библиотек и каталогах, которые вы укажете с помощью опции -L.

Опция **-nostdlib** запрещает использовать все системные библиотеки. При этом будут использованы библиотеки только из тех каталогов, которые вы укажете с помощью опции -L.

Опция **-static** означает, что будет использована статическая линковка.

21.1.6. Опции каталогов

Две важнейшие опции каталогов: -I и -L. Первая позволяет указать путь для поиска заголовков (файлов с расширением .h), а вторая — библиотек. Например, если вы хотите, чтобы компилятор использовал файлы заголовков, которые находятся в каталоге `/root/include`, тогда укажите

опцию **-I/root/include**. Обратите внимание на отсутствие пробела между буквой I и первым символом пути.

Если вы укажете опцию -I-, то в каталогах, которые вы укажете в объявленной до этого опции -I, будет производиться поиск только пользовательских заголовков, то есть заголовков, указанных в директиве #include «файл». Пути для поиска системных заголовков, которые указываются директивой #include <файл>, останутся неизменными.

21.1.7. Опции отладки

Если вы хотите использовать отладчик, например **gdb**, для отладки своей программы, укажите опцию -g при вызове компилятора. Эта опция помещает в откомпилированный файл отладочную информацию, вследствие чего существенно увеличивается объем файла. Поэтому никогда не используйте эту опцию для окончательной версии продукта.

21.1.8. Опции оптимизации

Компилятор **gcc** позволяет оптимизировать код вашей программы. Другими словами, **gcc** сделает все для того, чтобы ваша программа была как можно меньше по размеру и как можно быстрее запускалась. Для включения режима оптимизации используйте опцию -O1. Вы можете поэкспериментировать с опциями -O2 и -O3, которые еще больше пытаются оптимизировать вашу программу, однако не перестарайтесь. Если ваша программа после такой оптимизации работает еще медленнее, чем до нее, или же некорректно работает, используйте опцию -O0 для отключения оптимизации.

Обо всех остальных опциях вы сможете прочитать в справочной системе.

21.2. Сборочная утилита make

Если вы уже собирали прикладную программу из исходных кодов, то обратили внимание на стандартную последовательность команд: `make; make install`.

Без утилиты **make** не обходится создание ни одного серьезного проекта. Эта утилита управляет сборкой большого проекта, состоящего из десятков и сотен файлов. Программа **make** может работать не только с компилятором gcc, но и с любым компилятором для любого языка программирования, способным запускаться из командной строки.

Директивы утилиты **make** служат для определения зависимостей между файлами проекта и находятся в файле по имени Makefile, расположенном в каталоге сборки.

Разберемся, как пишутся make-файлы. Общий формат make-файла выглядит так:

```
цель1: список_необходимых_файлов
 последовательность_команд
...
цельN: список_необходимых_файлов
 последовательность_команд
```

Цель — это метка для некоторой последовательности команд (например, install) или результирующий файл, который нужно «построить» — скомпилировать или скомпоновать.

Цели должны отделяться друг от друга хотя бы одной пустой строкой. Список необходимых файлов — это перечень файлов или других целей, которые нужны для достижения данной цели; он может быть и пустым.

Последовательность команд — это команды, которые нужно выполнить для достижения цели. Последовательность команд должна отделяться от начала строки символом табуляции, иначе вы получите ошибку «missing separator» (нет разделителя).

Make-файл может содержать комментарии — они начинаются символом #.

В make-файлах вы можете использовать макроопределения:

```
CC=gcc
PATH=/usr/include /usr/src/linux/include
MODFLAGS:= -O3 -Wall -DLINUX -I$(PATH)
...
$(CC) $(MODFLAGS) -c proga.c
```

Чтобы обратиться к макроопределению в команде или в другом макроопределении, нужно использовать конструкцию $(имя). Макроопределение может включать в себя другое, ранее определенное, макроопределение.

Формат запуска утилиты make:

```
make [-f файл] [ключи] [цель]
```

Ключ –f указывает файл инструкций, который нужно использовать вместо Makefile. Если этот ключ не указан, то **make** ищет в текущем каталоге файл Makefile и начинает собирать указанную цель. Если цель не указана, то выполняется первая встреченная в make-файле. Сборка выполняется рекурсивно: **make** сначала выполняет все цели, от которых зависит текущая цель. Если зависимость представляет собой файл, то **make** сравнивает

его время последней модификации со временем целевого файла: если целевой файл старше или отсутствует, то будет выполнена указанная последовательность команд. Если целевой файл моложе, то текущая цель считается достигнутой.

Примечание

Если нужно избежать пересборки какого-то из файлов проекта, то можно искусственно «омолодить» его командой **touch**, которая присвоит ему в качестве времени последней модификации текущее время. Если нужно, наоборот, принудительно пересобрать цель, то следует «омолодить» один из файлов, от которых она зависит.

Работа программы **make** заканчивается, когда достигнута цель, указанная в командной строке. Обычно это цель **all**, собирающая все результирующие файлы проекта. Другими распространенными целями являются **install** (установить собранную программу) и **clean** (удалить ненужные файлы, созданные в процессе сборки).

В листинге 21.2 представлен make-файл, собирающий небольшой проект из двух программ **client** и **server**, каждая из которых компилируется из одного файла исходного кода.

Листинг 21.2. Примерный make-файл

```
CC=gcc
CFLAGS=-O

all: client server

client: client.c
 $(CC) client.c -o client

server: server.c
 $(CC) server.c -o server
```

Обычно при вызове утилиты **make** не нужно задавать никаких ключей. Но иногда использование ключей бывает очень кстати (таблица 21.1).

Ключи команды **make** *Таблица 21.1*

Ключ	Назначение
-C каталог	Перейти в указанный каталог перед началом работы
-d	Вывод отладочной информации
-e	Приоритет переменным окружения. Если у нас установлена переменная окружения CC и в Makefile есть переменная с таким же именем, то будет использована переменная окружения
-f файл	Использовать указанный файл вместо Makefile
-i	Игнорировать ошибки компилятора
-I каталог	В указанном каталоге будет производиться поиск файлов, включаемых в Makefile
-j n	Запускать не более n команд одновременно
-k	Продолжить работу после ошибки, если это возможно
-n	Вывести команды, которые должны были выполниться, но не выполнять их
-o файл	Пропустить данный файл, даже если в Makefile указано, что он должен быть создан заново
-r	Не использовать встроенные правила
-s	Не выводить команды перед их выполнением
-w	Вывод текущего каталога до и после выполнения команды

21.3. Пакет binutils и другие полезные программы

Пакет `binutils` содержит утилиты для работы с бинарными файлами:

- **ld** — компоновщик: программа, связывающая объектные файлы и библиотеки в исполняемый файл;
- **ar** — работа с архивами (создания, модификация и извлечение);
- **nm** — вывод названий идентификаторов из двоичных файлов;
- **objcopy** — копирование и трансляция двоичных файлов;
- **objdump** — вывод информации из двоичных файлов;
- **ranlib** — генерирование индекса оглавления архива;
- **size** — вывод размеров секций архива или двоичного файла;
- **strings** — вывод строк, которые возможно прочитать, из двоичных файлов;
- **addr2line** — конвертирование адресов в памяти в строку в файле;
- **nlmconv** — конвертирует объектный код в NLM.

А теперь перечислим несколько полезных вспомогательных программ.

21.3.1. ansi2knr

Утилита **ansi2knr** предназначена для преобразования текстов программ, написанных в соответствии со стандартом ANSI C, в программы на «классическом» C Кернигана и Ричи. Формат вызова:

```
ansi2knr oldfile.c newfile.c
```

21.3.2. as

Программа **as** — это GNU-версия ассемблера, предназначенная для создания объектных файлов из программ, написанных на языке ассемблера. Формат вызова:

```
as [ключи] файл1 [файл2 ... файлN]
```

Ключи программы as *Таблица 21.2*

Ключ	Назначение
-a	Вывод листинга
-ad	Не выводить отладочные сообщения
-ad	Включение в листинг текста программы, написанной на языке высокого уровня, если компиляция проводилась с ключом –g
-al	Вывод листинга на ассемблере
-an	Не обрабатывать форм
-as	Вывод списка символов программы
-aфайл	Вывести листинг в указанный файл
-f	Быстрый режим. Директивы препроцессора не обрабатываются
-iпуть	Добавить указанный путь к include-пути
-MRI	Обеспечить MRI-совместимость
-o файл	Создание объектного файла с указанным именем
-R	Поместить сегмент данных в сегмент кода
-v	Вывод версии
-W	Не выводить предупреждения

21.3.3. bison

Программа **bison** — это грамматический разборщик (парсер): она создает C-программу, предназначенную для разбора определенной грамматики. Данная программа вам не понадобится до тех пор, пока вы не захотите написать собственный компилятор. Ключи программы представлены в таблице 21.3. Формат вызова:

```
bison [ключи] файл
```

Ключи программы bison *Таблица 21.3*

Ключ	Назначение
-b префикс	Использовать указанный префикс для имени входящего файла
-d	Создать заголовочный файл, содержащий информацию о типах грамматических образцов (токенов), которые определены в вашей грамматике
-l	Не вставлять код в существующие файлы
-o файл	Установить файл результата
-t	Включить отладочную информацию
-v	Записать созданную программу в файл y.output

21.3.4. flex

```
flex [параметры] файл
```

Это еще одна программа, которая пишет код за нас. **Flex** может написать программу на языке C, которая будет искать заданные образцы текста в текстовых файлах и выполнять определенные действия, заданные программистом. Если вам нужна эта программа, тогда самое время прочитать страницы руководства man flex.

21.3.5. gprof

Программы вроде **gprof** называются профайлерами. Они предназначены для определения быстродействия вашей программы. Для каждого вызова функции вашей программы профайлер выводит время ее выполнения. Вы как программист анализируете полученную информацию и, если нужно, оптимизируете исходный код вашей программы.

21.3.6. strip

Утилита **strip** удаляет таблицу символов из объектного файла.

21.4. Пример программы на C

В п. 9.2.3 я сказал о состояниях процесса и перечислил среди них состояние «зомби». Зомби — это процесс, который уже завершился, но его родитель еще не получил сигнала о его завершении и не удалил его структуру из таблицы процессов. Такое может произойти, когда процесс-родитель почему-либо не готов к завершению потомка. Сейчас мы искусственно создадим такого зомби. Процесс-родитель породит потомка

и уснет на 10 секунд. Потомок завершится через 2 секунды, а в течение 8 секунд он будет находиться в состоянии зомби. Напоминаю, что состояние процесса можно увидеть по команде **top**.

Листинг 21.3. Файл zombie.c

```
#include <unistd.h>
#include <signal.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <stdio.h>

int main() {
   int pid;
   int status, died;

   pid=fork();
   switch(pid) {
   case -1: printf("ошибка fork\n");
            exit(-1);
   case 0 : printf("   Я потомок процесса %d\n", getppid());
            printf("   Мой PID %d\n", getpid());
            // Ждем 2 секунды и завершаемся
            sleep(2);
            exit(0);
   default: printf("Я родитель.\n");
            printf("Мой PID %d\n", getpid());
            // Ждем завершения дочернего процесса
            // через 10 секунд, а потом убиваем его
             sleep(10);
            if (pid & 1)
               kill(pid,SIGKILL);
            died= wait(&status);
   }
}
```

Скомпилируйте файл `zombie.c` и запустите исполняемый файл `zombie`:

```
$ gcc -o zombie zombie.c
$ ./zombie
Я родитель.
Мой PID 1147
   Я потомок процесса 1147
   Мой PID 1148
```

Запомните последний номер и быстро переключитесь на другую консоль, где введите команду `top -p 1148`:

```
16:04:22  up 2 min,  3 users,  load average: 0,10, 0,10, 0,04
1 processes: 0 sleeping, 0 running, 1 zombie, 0 stopped
CPU states:  4,5% user,  7,6% system,  0,0% nice,  0,0% iowait, 87,8% idle
Mem:   127560k av,   76992k used,   50568k free,       0k shrd,    3872k buff
        24280k active,              19328k inactive
Swap:  152576k av,       0k used,  152576k free                   39704k cached

  PID USER     PRI  NI  SIZE  RSS SHARE STAT %CPU %MEM   TIME COMMAND
 1148 den       17   0     0    0     0 Z     0,0  0,0   0:00 zombie <defunct>
```

Мы видим, что в списке процессов появился один зомби (STAT=Z), который «проживет» в таком состоянии целых 8 секунд.

Глава 22

ОТЛАДКА, ТРАССИРОВКА И ОПТИМИЗАЦИЯ ПРОГРАММ

- ОШИБКИ И ОТЛАДКА
- ОТЛАДЧИК GDB
- ПРИМЕР ОТЛАДКИ ПРОГРАММЫ
- ТРАССИРОВКА СИСТЕМНЫХ ВЫЗОВОВ
- ПРОФАЙЛЕР GPROF

22.1. Ошибки и отладка

Самыми страшными являются не синтаксические, а так называемые логические ошибки. Ваша программа может содержать хоть сотню мелких синтаксических ошибок — там не так функцию написали, там забыли указать параметр, а где-то пропустили точку с запятой. После исправления всех этих ошибок программа будет работать.

Если же ваша программа содержит логическую ошибку — например, вы выбрали неправильный алгоритм или неправильно его использовали, — то компилятор может даже не выдать предупреждения. Вроде бы ошибок нет, программа работает, но результат выдает неправильный или в какой-то момент вообще рушится. Мне запомнился один афоризм: «Программа делает то, что вы ей сказали, но не то, что вам хочется». Это и есть самое удачное, на мой взгляд, описание логической ошибки.

Если вы заметили ошибку до того, как ваша программа «увидела свет», то можете считать, что вам повезло. Одно дело, когда программа бесплатная, другое, когда вы за нее получили деньги, а заказчик недоволен... А бывает и такое, что программа может работать один, два месяца и только потом ваша логическая ошибка «всплывает» наружу. Почему это произошло? Дать однозначный ответ сложно, даже когда видишь код программы: все зависит от ее специфики.

Например, если ваша программа использует какую-нибудь СУБД для обработки информации, вы могли установить размер поля меньший, чем нужно. Первые два месяца программа работала отлично, а в один прекрасный момент оператор ввел очень длинную фамилию очень важного клиента, и ваша программа не внесла эту информацию в базу. Но это тривиальная ошибка, и ее можно исправить очень быстро.

А вот когда вы пишете программу для управления устройством или для обработки показаний внешних датчиков, подключенных к компьютеру, бывает очень сложно найти ошибку, связанную с конфликтом на аппаратном уровне. Например, пользователь установил новое устройство, которое конфликтует с вашим контроллером. Или вы написали модуль для поддержки одного контроллера, а пользователь подключил два, и оба теперь не работают.

Какие же ошибки часто совершают начинающие (и не только) программисты? Самая тривиальная — неправильное использование операций инкремента и декремента. Например, следующие выражения не эквивалентны:

```
x = y++ + 10;
x = ++y + 10;
```

В первом случае переменной x будет присвоено значение 15, а во втором — 16.

Следующей по частоте является ошибка неучтенной единицы. Например, вам нужен массив, состоящий из 10 элементов, вы его объявляете:

```
int a[10];
```

А затем инициализируете его с помощью цикла:

```
for (i=0;i<=10;i++) a[i] = 0;
```

Этот фрагмент кода попытается инициализировать несуществующий элемент — a[10].

Или еще один распространенный случай: программист забывает, что нумерация элементов массива начинается с 0, и не инициализирует первый элемент массива:

```
for (i=1;i<10;i++) a[i] = 0;
```

Особое место в зоопарке ошибок занимают ошибки, связанные с неправильным использованием указателей. Все эти ошибки можно условно разделить на три группы, которые я сейчас кратко перечислю.

1) Неправильное использование операторов * и &. Это самая распространенная группа ошибок начинающих программистов. Вот характерный пример такой ошибки:

```
/* неправильно */
char *s;
*s = (char *s)malloc(25);
/* правильно */
char *s;
s = (char *s)malloc(25);
```

2) Выделение недостаточного для адресации объекта объема памяти. Например, мы получим такую ошибку, если попытаемся скопировать в строку s (вышеприведенный фрагмент кода) строку, состоящую из 30 символов.

3) Использование неинициализированных указателей. Такие ошибки часто встречаются при работе с динамическими структурами. Например,

с линейными списками: вы забыли инициализировать главный элемент (head = NULL) и пытаетесь добавить в список новый элемент.

Использование рекурсивных вызовов может повлечь за собой **ошибку переполнения стека**, если вы неправильно зададите условие завершения рекурсии. Как правило, рекурсивная функция вызывает саму себя с несколько измененными параметрами. Рано или поздно такая функция должна, в зависимости от переданных параметров, возвратить какое-нибудь значение, а не опять вызвать саму себя.

Для облегчения поиска ошибок были созданы специальные программы — отладчики. Одним из самых удачных отладчиков для Linux является **gdb** (The GNU Debugger). Этот отладчик входит в состав всех распространенных дистрибутивов (за исключением их «урезанных» версий — для рабочих станций), и для его установки достаточно установить пакет gdb.

С помощью **gdb** вы сможете:

- запустить вашу программу с определенными аргументами;
- запустить программу в пошаговом режиме;
- установить точки останова (breakpoint);
- установить условие останова программы;
- узнать, что случилось, если программа неожиданно завершилась.

22.2. Отладчик gdb

Формат вызова отладчика **gdb** следующий:

```
gdb [-help] [-nx] [-q] [-batch] [-cd=dir] [-f] [-b bps]
[-tty=dev] [-s symfile] [-e prog] [-se prog] [-c core]
[-x cmds] [-d dir] [prog[core|procID]]
```

Ключи отладчика описаны в таблице 22.1.

Ключи командной строки **gdb** *Таблица 22.1*

Ключ	Назначение
-help или –h	Вывод краткого описания всех параметров
-nx или –n	Не обрабатывать команды файла инициализации .gdbinit
-q	Не выводить приветствие и информацию об авторских правах
-batch	Командный режим. Отладчик возвращает 0, если были выполнены все команды, указанные в файле, заданном параметром –x (и файле .gdbinit, если его использование разрешено). Если же хотя бы одна из команд не выполнена, возвращается ненулевое значение
-cd=каталог	Установить рабочий каталог (по умолчанию используется текущий каталог)
-f или –fullname	Данная опция нужна, если вы планируете использовать интерфейс текстового процессора **Emacs** для отладки ваших программ с помощью **gdb**. Для более подробного описания обратитесь к справочной системе

Продолжение табл. 22.1

-b bps (bits per second)	Установить скорость обмена информацией по последовательному интерфейсу, если вы отлаживаете вашу программу удаленно
-tty=терминал	Установить терминал в качестве стандартного ввода и вывода для отлаживаемой программы.
-s файл или -symbols=файл	Читает таблицу символов из указанного файла
-write	Разрешить запись в исполняемые и core-файлы
-e программа	Использовать указанную программу в качестве фильтра дампа
-se=файл	Читать таблицу символов из указанного файла и использовать указанный файл в качестве исполнимого
-core=файл или -c файл	Указать файл дампа
-command=файл или -x файл	Выполнить указанные в файле команды (используется в командном режиме)
-d каталог	Добавить каталог к списку поиска исходных текстов
[prog|core|procID]	Последний параметр задает объект, который нужно отлаживать. Вы можете задать программу (prog), или дамп-файл (core), который будет создан в случае ошибки программы (Segmentation fault), или же подсоединиться к уже запущенному процессу (procID)
-p PID	Подключиться к уже запущенному процессу (данная опция стала доступной в версии gdb 5.2)

Чтобы использовать **gdb** для отладки вашей программы, нужно добавить в исполняемый файл отладочную информацию. Для этого откомпилируйте вашу программу с опцией –g:

```
$ gcc -g -o prog prog.c
```

Данная опция включает отладочную информацию в родном для операционной системы формате, с которым может работать **gdb**.

Затем нужно вызвать **gdb** так:

```
$ gdb prog
```

Если после запуска вашей программы произошла ошибка и был создан дамп-файл (core), можно передать отладчику и этот файл:

```
$ gdb prog core
```

Можно также подключиться к уже запущенному процессу, для этого нужно передать его PID:

```
$ gdb 1111
```

Только убедитесь сначала в том, что у вас нет файла 1111, поскольку **gdb** сначала ищет исполняемый файл, затем core-файл, а уже затем PID.

После запуска отладчика в интерактивном режиме вы можете использовать команды, самые важные из которых перечислены в таблице 22.2. Об остальных можно узнать в справочной системе: **man gdb**.

Команды gdb *Таблица 22.2*

Команда	Назначение
break [файл:]функция	Установить точку останова
run [аргументы]	Запустить программу и передать ей указанные аргументы
bt	Обратная трассировка: отобразить стек программы
print выражение	Вывести значение выражения, операндами могут быть переменные, объявленные в вашей программе
C	Продолжить выполнение программы (после останова)
Next	Выполнить следующую строку. Это так называемый шаг «над» (step over). Если следующая строка — вызов функции, то мы выполним ее за один шаг — «перешагнем» ее
Step	Выполнить следующую строку. Это так называемый шаг «в» (step into). Если следующая строка — вызов функции, то мы будем последовательно выполнять все операторы тела функции
help [имя]	Вывести справку о команде отладчика или вывести общую информацию о нем
Quit	Выход

В данной таблице приведены далеко не все команды. Если вас интересует более полная информация, обратитесь к руководству по **gdb**.

22.3. Пример отладки программы

Давайте напишем программу, которая обнуляет элементы массива `a[]`. Да, программа ничего полезного не делает, но на ее примере можно продемонстрировать работу с отладчиком **gdb**.

Вот листинг программы:

Листинг 22.1. Демонстрационная программа, содержащая ошибку

```
#include <stdio.h>
int main()
{
 char developer[]="Denis";
 int a[10];
 int i;

 do
 {
    a[i]=0;
    i++;
 }
 while (i<10);
    return 0;
}
```

Назовем нашу программу test.c и откомпилируем ее:

```
$ gcc -g -o test test.c
```

Опция -g добавляет отладочную информацию для отладчика **gdb**, а опция -o указывает имя результирующего файла.

Просмотрите листинг программы. Программа не делает ничего подозрительного — она всего лишь в цикле do обнуляет все элементы массива — сначала нулевой элемент a[0] становится нулем, потом первый, второй и так далее.

Попробуйте запустить программу, и вы увидите сообщение: Segmentation fault. В чем же причина? Попробуем выяснить ее с помощью отладчика **gdb**. Запустите отладчик с параметром test (это имя нашего исполняемого файла):

```
$ gdb test
```

В отладчике **gdb** введем команду run для запуска программы. И что мы видим? Что программа получила сигнал SIGSEGV, то есть имеет место ошибка Segmentation fault. Эта ошибка произошла в строке 12: a[i]=0. Но что может быть опасного в этом операторе? Ошибка Segmentation fault может произойти по нескольким причинам, одной из которых является выход за пределы массива. Проверим это: введите команду print i.

Команда print выводит значение указанной переменной (или выражения). Ого! Оказывается, мы пытаемся обнулить не нулевой элемент массива, а 715910728-ой! Почему же так произошло? Может быть, это **gdb** нагло врет и вместо значения переменной i выводит непонятно что? Для проверки на лживость введите команду print developer, которая выведет на экран значение переменной developer. С переменной developer все нормально — ее значение «Denis» (рис. 22.1).

Так что же произошло? Так как переменная i не является глобальной, ее значение не обнуляется при запуске программы. Чтобы избежать подобной ошибки, нужно инициализировать переменную при объявлении:

```
int i=0;
```

Вот мы и нашли ошибку!

Что же еще можно сделать с помощью **gdb**? Можно установить **breakpoint**, то есть точку останова, прерывающую выполнение программы в указанном месте. Это нужно для того, чтобы проследить состояние некоторых переменных и/или стека программы перед запуском какой-нибудь функции или же для пошаговой трассировки программы. Для пошаговой трассировки программы установите точку останова для функции main:

```
break main
```

```
[denis@localhost test]$ gdb test
(gdb) run
Starting program: /home/denis/test/test

Program received signal SIGSEGV, Segmentation fault.
0x08048401 in main () at test.c:12
12        a[1]=0;
(gdb) print i
$1 = 715910728
(gdb) print developer
$2 = "Denis"
(gdb) bt
#0  0x08048401 in main () at test.c:12
#1  0x2aae6727 in __libc_start_main () from /lib/libc.so.6
(gdb)
```

Рис. 22.1. *Сессия gdb*

Для установки точки останова на другую функцию введите команду **break** и в качестве аргумента укажите имя функции. Теперь запустите программу на выполнение (команда **run**). Команде **run** можно передать также аргументы, с которыми должна запускаться программа, например, **run /home/denis/report.txt**.

Когда отладчик достигнет точки останова, он приостановит выполнение программы и будет ждать ваших инструкций. Затем введите команду **next** для выполнения следующей строки. Команда **next** — это команда трассировки «над» функцией, то есть оператор вызова функции будет выполнен за один шаг. Команда **step** используется для пошаговой трассировки функций. Если вы хотите продолжить нормальное выполнение программы после точки останова, введите команду **c**.

Для отображения стека программы предназначена команда **bt** (backtrace). Команда **list** используется для отображения исходного текста программы (рис. 22.2).

Естественно, весь текст программы не поместится на экране, поэтому отладчик отобразит только его часть. Чтобы еще раз не вводить команду **list**, просто нажмите Enter, и **gdb** выполнит предыдущую команду.

Команда **help** предназначена для отображения справки по командам отладчика, а программа **quit** — для выхода из него.

Рис. 22.2. Команда list

Рис. 22.3. Программа KDbg

Возможности программы **gdb** этим не ограничиваются. Отладчик **gdb** — это очень мощная программа, и я советую вам почаще использовать команду **help** — узнаете много полезного для себя.

Если вам удобнее использовать графический интерфейс, чем символьный, вы можете воспользоваться одним из интерфейсов к программе **gdb**. Один из них — **KDbg**. Все, что мы только что проделали с помощью команд отладчика **gdb**, вы можете сделать, используя меню интерфейса **KDbg** (рис. 22.3).

Программа **KDbg** понравилась мне еще тем, что позволяет удобно просматривать регистры, потоки, память, стек и прочее, имеющее непосредственное отношение к процессу отладки. Однако имейте в виду, что интерфейс **KDbg** сильно ограничивает возможности отладки, потому что позволяет выполнять лишь базовые функции.

Существуют и другие оболочки для отладчика **gdb**, например, **DDD**. Эта оболочка обладает чуть большими возможностями, чем **KDbg**, но все же она является лишь надстройкой над **gdb**. Оболочек много, а **gdb** — один. Вы можете выбрать оболочку на свой вкус, а я вообще предпочитаю **gdb** без всяких оболочек.

Рис. 22.4. *Оболочка DDD*

22.4. Трассировка системных вызовов

Вы когда-нибудь задумывались о том, какие системные вызовы использует наша программа во время своего выполнения? Если да, то этот пункт как раз для вас. Возможно, пока он только удовлетворит ваше любопытство, но через некоторое время эта информация станет вам по-настоящему необходима.

Проследить, какие системные вызовы использует наша программа, позволяет программа **strace**. Для ее установки нужно установить пакет strace.

Формат вызова команды **strace** следующий:

```
strace [ -dffhiqrtttTvxx ] [ -acolumn ] [ -eexpr ] ...
 [ -ofile ] [ -ppid ] ... [ -sstrsize ] [ -uusername ]
    [ command [ arg ... ] ]
```

Ключи программы перечислены в таблице 22.3.

*Ключи командной строки **strace*** *Таблица 22.3*

Ключ	Назначение
-c	Подсчитывать время, затраченное на каждый вызов и обработку ошибок. В конце трассировки будет представлен подробный отчет
-d	Выводить отладочные сообщения самой программы strace на стандартный вывод ошибки
-f	Трассировать дочерние процессы, созданные уже трассируемыми процессами
-ff	Данная опция применятся только вместе с опцией –o имя_файла. Каждый трассируемый процесс будет записан в файл имя_файла.pid
-F	Следовать вызовам vfork(). Данную опцию нельзя использовать вместе с опцией –f
-h	Вывести справку
-i	Выводить указатель инструкции во время системного вызова
-q	«Тихий режим». Подавляет вывод некоторых сообщений
-r	Выводить относительную метку времени для каждого вызова
-t	Перед каждой строкой выводить текущее время
-tt	То же, что и –t, но будут выводиться также микросекунды
-T	Показывать время, потраченное на системный вызов (то есть разницу между временем запуска и временем завершения вызова). Для каждого вызова
-v	Получение дополнительной информации
-V	Вывести номер версии strace
-x	Выводить не-ASCII строки в шестнадцатеричном формате
-xx	Выводить все строки в шестнадцатеричном формате
-a столбец	Выровнять возвращаемые вызовами значения в указанном столбце (по умолчанию 40)
-e выражение	Позволяет задать отслеживаемые события. За более подробной информацией обратитесь к справочной системе
-e trace=набор	Определить набор отслеживаемых вызовов. Например, trace=open,close,read,write

Продолжение табл. 22.3

Ключ	Назначение
-e trace=file	Будут отслеживаться только вызовы для работы с файлами (**open,stat,chmod,unlink и т.д.**)
-e trace=process	Отслеживаются вызовы для работы с процессами (fork, exec, wait и др.)
-e trace=network	Отслеживаются сетевые вызовы
-e trace=signal	Отслеживаются вызовы для работы с сигналами
-e trace=ipc	Отслеживаются IPC-вызовы
-e abbrev=набор	Сокращает вывод каждого члена структуры. Например, abbrev=all или abbrev=none
-e verbose=набор	Различать структуры различных системных вызовов, по умолчанию verbose=all
-e raw=set	Выводит не декодированные значения аргументов системных вызовов. Данный аргумент полезен, если вы не доверяете декодированию или хотите знать точное числовое представление аргумента
-e signal=набор	Определяет набор трассируемых сигналов. По умолчанию signal=all. Вы можете использовать восклицательный знак для отрицания, например, signal=!SIGIO означает, что сигнал SIGIO не будет трассирован
-e read=набор	Выполнять полный шестнадцатеричный и ASCII-дамп всех прочитанных вызовом read() данных. Например, чтобы видеть все данные, поступающие через дескрипторы 2 и 7, введите read=2,7
-e write=набор	То же, что и -e read, но только для записи
-o имя_файла	Перенаправить вывод программы в указанный файл. Данный файл будет полезен для дальнейшего анализа трассировки
-p pid	Присоединиться к процессу с PID=pid и начать трассировку
-s размер	Установить максимальный размер строки (по умолчанию 32). Имена файлов не рассматриваются как строки, поэтому всегда будут напечатаны полностью
-S критерий	Сортирует вывод гистограммы, которая выводится опцией –c, по заданному критерию: time (время), calls (вызовы), name (имя) и nothing (без сортировки)
-u имя_пользователя	Запустить программу от имени указанного пользователя. Эта опция будет полезной, если вы, зарегистрировавшись как root, будете проверять корректность работы программы, если бы она была запущена под другим пользователем

Вы даже не можете себе представить, какие системные вызовы использует такая маленькая программка:

Листинг 22.2. Файл prog.c

```
#include <stdio.h>
int main()
{
 printf("Hello\n");
 return 0;
}
```

Откомпилируйте эту программу (`gcc -o prog prog.c`) и запустите **strace**:

```
$ strace prog
```

Вы увидите следующий вывод:

```
execve("./a.out", ["./a.out"], [/* 21 vars */]) = 0
uname({sys="Linux", node="localhost.localdomain", ...}) = 0
brk(0)                                  = 0x80495b4
open("/etc/ld.so.preload", O_RDONLY)    = -1 ENOENT (No
such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 7
fstat64(7, {st_mode=S_IFREG|0644, st_size=31578, ...}) = 0
old_mmap(NULL, 31578, PROT_READ, MAP_PRIVATE, 7, 0) =
0x40014000
close(7)                                = 0
open("/lib/i686/libc.so.6", O_RDONLY)   = 7
read(7, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\
0`u\1B4\0"..., 1024) = 1024
fstat64(7, {st_mode=S_IFREG|0755, st_size=1401027, ...}) =
0
old_mmap(0x42000000, 1264928, PROT_READ|PROT_EXEC, MAP_
PRIVATE, 7, 0) = 0x42000000
mprotect(0x4212c000, 36128, PROT_NONE)  = 0
old_mmap(0x4212c000, 20480, PROT_READ|PROT_WRITE, MAP_
PRIVATE|MAP_FIXED, 7, 0x12c000) = 0x4212c000
old_mmap(0x42131000, 15648, PROT_READ|PROT_WRITE, MAP_
PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x42131000
close(7)                                = 0
munmap(0x40014000, 31578)               = 0
brk(0)                                  = 0x80495b4
brk(0x80495e4)                          = 0x80495e4
brk(0x804a000)                          = 0x804a000
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0),
...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_
ANONYMOUS, -1, 0) = 0x40014000
write(1, "Hello\n", 6)                  = 6
munmap(0x40014000, 4096)                = 0
_exit(0)                                = ?
```

Читать вызовы нужно так:

```
имя системного вызова = возвращаемое значение
```

В нашем случае мы вывели на консоль шесть символов, поэтому вызов write() возвратит значение 6.

В случае, если системный вызов завершился неудачно (обычно код ошибки –1), программа **strace** выводит не только код, но и описание ошибки:

```
open("/foo/bar", O_RDONLY) = -1 ENOENT (No such file or
directory)
```

Стандартные константы выводятся в их символьном представлении:

```
open("file.dat", O_WRONLY|O_APPEND|O_CREAT, 0666) = 3
```

Сигналы также выводятся в символьном представлении, например:

```
sigsuspend([] <unfinished ...>
--- SIGINT (Interrupt) ---
+++ killed by SIGINT +++
```

Структуры, точнее члены структур, заключаются в фигурные скобки и выводятся в формате имя_члена=значение, например:

```
lstat("/dev/null", {st_mode=S_IFCHR|0666, st_
rdev=makedev(1, 3), ...}) = 0
```

Символьные указатели печатаются как строки в C, то есть их значения заключены в двойные кавычки:

```
read(3, "root::0:0:System Administrator:/"..., 1024) = 422
```

22.5. Оптимизация программ. Профайлер gprof

Ваша программа работает медленно? Скорее всего, причина кроется в неэффективном, медленном алгоритме. Существуют программы, позволяющие определить время работы каждой функции вашей программы и всей программы в целом. Программы такого рода называются профайлерами. В вашем дистрибутиве может присутствовать один из профайлеров **gprof**, **prof**, **profiler**.

Сейчас мы рассмотрим программу **gprof** (The GNU Profiler), позволяющую определить время работы каждой функции. Основные ключи программы представлены в таблице 22.3.

Ключи командной строки **gprof** *Таблица 22.3*

Ключ	Назначение
-a	Не выводить информацию о статических функциях
-b	Не выводить описание каждого поля в итоговой таблице
-c	Включить эвристический анализ текстового сегмента объектного файла с целью создания статического графика вызовов
-e имя_функции	Не выводить отчет о работе указанной функции и обо всех функциях, которые из нее вызываются
-E имя_функции	Не обрабатывать указанную функцию и все функции, которые она вызывает
-f имя_функции	Выводить информацию только об указанной функции и обо всех функциях, которые из нее вызываются
-F имя_функции	Обрабатывать только указанную функцию и все функции, которые из нее вызываются
-k func1 func2	Не выводить информацию о вызове функции func2 из функции func1
-s	Создание итогового файла gmon.sum
-z	Вывести функции с нулевым процессорным временем

22.5.1. Использование профайлера

Для использования профайлера нужно скомпилировать программу с опцией компилятора **–pg** и без опции **–o**, так как профайлер по умолчанию работает с файлом `a.out`. После этого запустите файл `a.out`, чтобы он создал файл `gmon.out`. Теперь запустите профайлер с параметром **--nograph**:

```
$ gcc -pg 1.c
$ ./a.out
$ gprof -b --no-graph
```

Без ключа **–b** профайлер выведет описание полей итоговой таблицы:

- **Time**: время работы функции в процентном соотношении;
- **cumulative seconds**: сумма числа секунд этой функции и вызывающих ее функций;
- **self seconds**: число секунд, потраченное на работу этой функции в отдельности;
- **Calls**: число вызовов;
- **self ms/calls**: количество миллисекунд, на протяжении которых функция выполнялась;
- **total ms/calls**: количество секунд, на протяжении которых выполнялась функция и все функции, которые вызываются данной функцией;
- **name**: имя функции.

Чтобы было понятно, что означает каждое поле, рассмотрим листинг 22.3.

```
root@localhost:~/debug/2
Файл  Правка  Свойства  Справка
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  us/call  us/call  name
 50.48      1.09     1.09       10 109000.00 218000.00  function
 49.54      2.18     1.07       10 107000.00 107000.00  function2
[root@localhost 2]#
```

Рис. 22.5. *Программа gprof*

Листинг 22.3. Демонстрационная программа

```
#include <stdio.h>
int function2()
{
 int i;
 /* генерируем задержку */
 for (i=0;i<9999999;i++) ;
 return 777;
}

double function(void)
{
 int i;
 double x = 7.2323232323, y=324343.3434;
 /* генерируем задержку */
 for (i=0;i<9999999;i++) x/y;
 function2();
 return x/y;
}
```

```
int main()
{
 int i;
 double l;
 for (i=0;i<10;i++)
 {
   printf("%d\b",i);
   l=function();
 }
 return 0;
}
```

Как видно из листинга, функция `function()` вызывается 10 раз (см. поле calls). В свою очередь, она вызывает функцию `function2()`, следовательно, число вызовов этой функции тоже равно 10. Функция function() выполняется 1.09 секунд (self seconds), а так как ее никто не вызывает (кроме main), то поле `cumulative seconds` также равно 1.09. Функция `function2()` работает 1.07 секунд, но ее вызывает функция `function()`, которая работает 1.09 секунд. Следовательно, поле `cumulative seconds` для второй функции равно 1.09+1.07=2.16. Поле `self ms/call` эквивалентно полю `self seconds`, только его величина представлена в миллисекундах. Поле `total ms/call` обратно полю `cumulative call` и содержит время выполнения этой функции и всех дочерних функций, в то время как поле `cumulative call` содержит время этой функции и всех родительских.

22.5.2. Как оптимизировать программу

В качестве оптимизации программы могу вам порекомендовать предпринять следующие действия:

1. Запустите профайлер, и пусть он определит время работы всех функций.
2. Перепишите функцию (или функции), которые занимают больше всего процессорного времени. Возможно, вам придется изменить алгоритм работы этих функций.
3. Когда алгоритм изменить невозможно, можно попытаться переписать часть кода функции на языке ассемблера, если, конечно, это не противоречит идеологии вашей программы (возможно, вы хотите, чтобы она запускалась на как можно большем количестве разных платформ - тут без С не обойтись). Если переработка функции невозможна или помогла мало, то попробуйте оптимизировать программу с помощью опций компилятора. Прежде всего выключите отладочную информацию (не указывайте опцию –g) — и размер программы станет меньше. Используйте одну из опций оптимизации –O.

Глава 23

РАЗРАБОТКА ГРАФИЧЕСКОГО ПРИЛОЖЕНИЯ: БИБЛИОТЕКА GTK+

- ВВЕДЕНИЕ В GTK+
- БИБЛИОТЕКА GLIB
- ПЕРВАЯ ПРОГРАММА НА GTK+
- ВИДЖИТЫ

Сейчас мы поговорим о создании графического интерфейса для вашей Linux-программы. Как вы знаете, средствами одного C нормальный GUI не построишь, тем более что привычный пользователь Windows очень требователен не просто к наличию GUI, но и к дизайну формы (окна программы). Поэтому без дополнительных библиотек вам не обойтись. Самыми распространенными библиотеками для создания GUI являются библиотеки GTK+ и Qt. Рекомендуется использовать только эти библиотеки, поскольку велика вероятность того, что они уже будут установлены у пользователя (уж GNOME и KDE есть почти у всех). Рассмотрим подробно библиотеку GTK+, поскольку она, на мой взгляд, проще, чем Qt.

Скорее всего, GTK+ у вас уже установлена, но вам понадобится дополнительно установить пакет gtk+-devel, содержащий необходимые файлы для разработки GTK-программ.

23.1. Введение в GTK+

Первоначально библиотека GTK+ (далее GTK) была разработана для использования программой GIMP, отсюда и название — The GIMP Toolkit. Со временем библиотека стала использоваться для разработки других приложений для среды GNOME. Сейчас GTK — это объектно-ориентированный и кросс-платформенный инструмент для создания графического интерфейса приложений, причем созданные с использованием GTK приложения независимы от среды GNOME. Ваше приложение, написанное на GTK, будет отлично работать в KDE или любой другой среде — главное, чтобы библиотека GTK была установлена на компьютере.

Сама библиотека GTK+ стоит на трех китах:

- Библиотека Glib — предназначена для оперирования с различными типами данных. Данная библиотека будет полезной для разработки любых приложений, даже не GTK. Эта же библиотека содержит функции для работы с памятью и для обработки ошибок. О типах данных библиотеки Glib мы поговорим позже.

- Библиотека GDK (The GIMP Drawing Kit) — библиотека низкого уровня, ее функции взаимодействуют с функциями оконной системы (X Window). Используется для построения графических примитивов.
- Библиотека GTK — используется для создания графического интерфейса

23.2. Библиотека Glib

23.2.1. Стандартные типы данных библиотеки Glib

Библиотека Glib содержит типы данных, аналогичные типам данных языка C, а также такие структуры, как деревья, списки; функции для работы с памятью и обработки ошибок. Это сделано для того, чтобы ваше приложение могло быть легко портировано на другую платформу. Например, на одних компьютерах тип int может быть 32-разрадным, а на других — 64-разрядным (это зависит от архитектуры центрального процессора). А если вы используете тип gint, объявленный в библиотеке Glib, то можете дальше разрабатывать свое приложение и не беспокоиться о том, как оно будет работать на RISC-машине под управлением Windows NT. В таблице 23.1 перечислены типы данных библиотеки GLib, которые соответствуют типам данных C.

Таблица соответствия типов данных Glib и C *Таблица 23.1*

Тип данных C	Тип данных Glib
char	gchar
short	gshort
long	glong
int	gint
bool	gboolean
unsigned char	guchar
unsigned short	gushort
unsigned long	gulong
unsigned int	guint
float	gfloat
double	gdouble
long double	gldouble
void*	gpointer

Для использования этих типов данных, как и прочих возможностей библиотеки Glib, необходимо подключить заголовочный файл glib.h.

23.2.2. Функции для работы с памятью

Функции для работы с памятью библиотеки Glib выполняют те же действия, что и соответствующие им функции языка С. Вот их прототипы:

```
gpointer g_malloc( gulong size );
gpointer g_realloc( gpointer mem, gulong size );
void g_free( gpointer mem );
```

23.2.3. Строки и Glib

Библиотека Glib содержит довольно много функций для работы со строками, но я перечислю лишь самые с моей точки зрения интересные (таблица 23.2).

Некоторые строковые функции библиотеки Glib *Таблица 23.2*

Прототип	Описание
gchar* g_strchug (gchar* s)	Функция удаляет все пробелы в строке s, стоящие в начале строки (до первого печатного символа). Данная функция может пригодиться для контроля введенной пользователем информации
gchar* g_strchomp(gchar* s)	Функция удаляет все пробелы в строке s, стоящие в конце строки
gchar* g_strstrip(gchar* s)	Функция удаляет пробелы в начале и в конце строки
void g_strdown (gchar* s)	Функция переводит все буквы строки s в нижний регистр
void g_strup (gchar* s)	Функция переводит все буквы строки s в верхний регистр
void g_strreverse (gchar* s)	Функция преобразовывает прописные буквы в строчные и наоборот

23.4.4. Списки

Библиотека Glib содержит средства для работы с одно- и двусвязными списками. Особенность двусвязного списка заключается в том, что по нему можно перемещаться в обоих направлениях — назад и вперед. В файле gslist.h (Glib Single List) описаны средства для работы с односвязными списками, а в файле glist.h — для работы с двусвязным списком.

Вот структуры односвязного и двусвязного списков:

```
// односвязный список
typedef struct _GSList GSList;
struct _GSList
{
gpointer data;
GSList *next;   // указатель на следующий элемент списка
}
```

```
// двусвязный список
typedef struct _GList GList;
struct _GList
{
gpointer data;
GList *next;    // указатель на следующий элемент списка
GList *prev;    // указатель на предыдущий элемент списка
}
```

Поле data предназначено для хранения данных списка, причем они могут быть любого типа, ведь gpointer — это тип void*.

Работать со списками очень просто. Для начала нужно объявить список:

```
GList *list = NULL;
GSList *slist = NULL;
```

Затем добавить элементы в список. Это можно сделать с помощью двух функций — g_list_append() или g_slist_prepend() — в зависимости от используемого типа списка:

```
gchar *el = g_strdup("это первый элемент");
list = g_list_append(list, el);
```

Функции g_list_append() и g_slist_prepend() добавляют элемент в конец списка. Если вы хотите добавить элемент в начало списка, нужно использовать функции:

```
g_list_prepend(GList *list, gpointer data)
g_slist_prepend(GSList *list, gpointer data)
```

Чтобы вставить новый элемент в определенную позицию, нужно использовать функции:

```
GList *g_list_insert( GList *list, gpointer data, gint position )
GSList *g_slist_insert( GSList *list, gpointer data, gint position )
```

Здесь position — это номер элемента, перед которым нужно вставить новый элемент. Если position=0, то элемент будет добавлен в начало списка, то есть перед бывшим первым элементом.

Для удаления элемента используются функции:

```
GList *g_list_remove(GList *list, gpointer  data )
GSList *g_slist_remove(GSList *list, gpointer data)
```

Для передвижения по списку используются функции:

```
g_list_next(), g_slist_next() — на "шаг" вперед
g_list_prev() - назад
```

Вот небольшой пример работы со списком — вывод на консоль всех его элементов:

```
// double_list должен быть определен и содержать элементы
GList *list = double_list;

while (list!=NULL)
{
printf("%s\n",list->data);
list = g_list_next(list);
}
```

По окончании работы со списком не забудьте освободить память:

```
void g_list_free(GList *list);
void g_slist_free(GSList *slist);
```

Для сортировки списка используется функция:

```
GSList *g_slist_sort(GSList * slist, GCompareFunc f);
```

Первый параметр — это список, который нужно отсортировать. Второй — это функция сравнения двух элементов. Вот ее прототип:

```
typedef gint (*GCompareFunc) (gconstpointer a, gconstpointer b);
```

Данную функцию вы должны написать самостоятельно. Она должна принимать два параметра и возвращать целое значение:

- если a<b, то –1 (точнее, любое число меньше 0);
- если a==b, то 0;
- если a>b, то 1 (любое число больше 0).

Библиотека Glib также содержит средства для работы с деревьями — как бинарными, так и произвольными, но мы эти средства рассматривать не будем.

23.2.4. Таймеры в Glib

Библиотека Glib позволяет использовать таймеры в наших программах. Для этого нужно:

- подключить заголовочный файл gtimer.h;
- создать таймер функцией GTimer *g_timer_new();
- запустить таймер функцией g_timer_start(GTimer *timer);
- узнать время, отсчитанное таймером — g_timer_elapsed();
- при необходимости перезапустить таймер с помощью функции g_timer_reset(GTimer *timer)
- остановить таймер функцией g_timer_stop(GTimer *timer);
- уничтожить таймер — g_timer_destroy(GTimer *timer).

Стоит остановиться подробнее лишь на функции g_timer_elapsed(GTimer *timer, gulong *mcs). Данная функция возвращает число секунд, отсчитанное таймером. По адресу указателя *mcs записывается число микросекунд.

Пример использования таймера представлен в листинге 23.1.

Листинг 23.1. Использование таймера

```
#include <stdio.h>
#include <glib.h>
#include <gtimer.h>

int main()
{
 double sec;
 gulong ms;
 int i;

 GTimer *timer = g_timer_new();

 printf("Данный цикл будет работать не более 10 секунд\n");

 g_timer_start(timer);

 for (i=1; i>0;)
 {
   sec = g_timer_elapsed(timer,&ms);
   if (sec >=10)
   {
     g_timer_stop(timer);
     printf("Таймер остановлен. Мкс: %d\n",ms);
     break;
   }
 }

 g_timer_destroy(timer);

 return 0;
}
```

23.3. Первая программа на GTK+

23.3.1. Виджиты

Перед написанием самой простой GTK-программы нужно разобраться с терминологией GTK. Элементы графического интерфейса пользователя — окна, кнопки, поля ввода, переключатели и тому подобное — называются виджитами.

Основным элементом графического интерфейса является окно. Виджиты для размещения в окне помещаются в контейнер. В самом окне выравнивать виджиты можно с помощью вертикальных/горизонтальных боксов или же таблиц. Второй способ более гибок, хотя он может показаться вам сложнее.

Виджиты могут реагировать на сигналы, например, щелчок мышью. При этом вызывается функция-обработчик события (сигнала), если вы определили ее.

Работа с виджитами происходит по такой схеме:

1. создание виджита с помощью одной из функций библиотеки GTK;
2. определение свойств виджита;
3. определение сигналов виджита, если он должен реагировать на сигналы;
4. размещение виджита в контейнере, то есть привязка его к окну;
5. отображение виджита.

Нужно обязательно отобразить виджит, иначе его никто не увидит. Например, следующий фрагмент кода создает виджит — кнопку с текстом — и отображает ее.

```
GtkWidget *button;

...
/* Рисуем кнопочку с надписью Hello, All */
button = gtk_button_new_with_label ("Hello, All");

/* При нажатии кнопки будет вызвана функция hello() */
gtk_signal_connect (GTK_OBJECT (button), "clicked",
                    GTK_SIGNAL_FUNC (hello), NULL);

/* Помещаем кнопку в контейнер */
gtk_container_add (GTK_CONTAINER (window), button);

/* Отображаем кнопку. */
gtk_widget_show (button);
```

Первый оператор создает кнопку (button), второй — добавляет кнопку в контейнер. В данном случае контейнером является наше окно. Виджит window должен быть создан раньше: нельзя создать кнопку без окна. Точнее, можно, но тогда она не будет привязана к какому-либо окну и мы ее не увидим. Функция gtk_widget_show () отображает нашу кнопку. Не забудьте отобразить и само окно. Порядок отображения виджитов особой роли не играет, но рекомендуется главное окно отображать в последнюю очередь.

23.3.2. Окна

Сейчас мы напишем программу, которая будет формировать небольшое графическое окошко. Начнем сразу с исходного кода — так будет проще понять, что есть что.

Листинг 23.2. Простое окно (файл first.c)

```
#include <gtk/gtk.h>

int main( int   argc, char *argv[] )
{
   GtkWidget *window1;

   gtk_init( &argc, &argv );

   window1 = gtk_window_new( GTK_WINDOW_TOPLEVEL );

   gtk_window_set_title(GTK_WINDOW(window1),"Заголовок");

   gtk_widget_show( window1 );

   gtk_main();

   return 0;
}
```

Сначала мы подключаем заголовочный файл `gtk/gtk.h` — это необходимое условие для того, чтобы начать работу с библиотекой GTK. В первой строке программы мы объявляем наш основной (и единственный в этой программе) виджит — виджит основного окна:

```
GtkWidget *window1;
```

Обратите внимание, что виджит объявлен, но работать с ним пока нельзя. Сначала (обязательно до вызова первой GTK-функции) нужно

вызвать инициализирующую функцию gtk_init() и передать ей два два параметра — аргументы функции main(). После того, как библиотека инициализирована, нужно вызвать функцию gtk_window_new(), которая создает окно (напомню, что пока окно объявлено, но не создано). Теперь, когда виджит окна создан, можно установить его свойства и определить реакцию на сигналы. Установим свойство Title (заголовок) окна. Это делается с помощью функции gtk_window_set_title():

```
gtk_window_set_title(GTK_WINDOW(window1),"Заголовок");
```

Теперь можно отобразить наше окно:

```
gtk_widget_show( window1 );
```

Чтобы наше приложение могло реагировать на события оконной среды (например, щелчок мыши), нужно вызвать функцию gtk_main(). Функции gtk_init() и gtk_main() должны присутствовать в любой GTK-программе.

Теперь откомпилируем наше приложение. Для этого введем следующую команду в командной строке:

```
$ gcc first.c -o first `gtk-config --cflags` `gtk-config --libs`
```

Флаги `gtk-config --cflags` `gtk-config --libs` нужно использовать при компиляции любой GTK-программы. Если компиляция не удается, то проверьте, что вы используете апострофы (`), а не одинарные кавычки (‘), и что программа ***gtk-config*** у вас установлена.

Запустим нашу программу в эмуляторе терминала X Window (или оконной среды GNOME/KDE):

```
$ ./first
```

Вы увидите окно, изображенное на рис.23.1.

Рис. 23.1. *Простое окно*

Теперь закроем окно и перейдем к терминалу: окно закрыто, мы его больше не видим, а терминал не освобожден. Наша программа не реагирует на событие закрытия окна. По идее, когда графическая среда закрывает окно, программа должна завершить свою работу. А наша программа этого не делает. Значит, нужно «научить» ее реагировать на события (сигналы) оконной системы. Для этого нажмите в терминале Ctrl + C и отредактируйте исходный текст программы следующим образом:

Листинг 23.3. Добавим реакцию на закрытие окна

```
#include <gtk/gtk.h>

int main( int   argc, char *argv[] )
{
   GtkWidget *window1;

   gtk_init( &argc, &argv );

   window1 = gtk_window_new( GTK_WINDOW_TOPLEVEL );

   gtk_signal_connect( GTK_OBJECT( window1 ), "destroy",
         GTK_SIGNAL_FUNC( gtk_main_quit ), NULL );

   gtk_widget_show( window1 );

   gtk_main();

   return 0;
}
```

Функция **gtk_signal_connect()** устанавливает реакцию объекта `window1` на сигнал **destroy** и вызывает функцию **qtk_main_quit()** для завершения работы программы.

А что если нам при завершении работы программы нужно выполнить какие-нибудь специфические действия, например, удалить временные файлы? Тогда нужно написать свою функцию-обработчик события **destroy** (листинг 23.4).

Эта функция будет называется `destroy_window1()`, и мы «пропишем» ее в функции **gtk_signal_connect()** в качестве обработчика события закрытия окна вместо **gtk_main_quit()**. Делать она не будет ничего, просто вызовет стандартную функцию **gtk_main_quit()**.

Листинг 23.4. Добавляем собственную функцию-обработчик завершения работы

```
#include <gtk/gtk.h>

void destroy_window1( GtkWidget *widget, gpointer data);

int main( int   argc, char *argv[] )
{
   GtkWidget *window1;

   gtk_init( &argc, &argv );


   window1 = gtk_window_new( GTK_WINDOW_TOPLEVEL );

   gtk_signal_connect( GTK_OBJECT( window1 ), "destroy",
          (GtkSignalFunc)destroy_window1, &window1 );

   gtk_widget_show( window1 );

   gtk_main();

   return( 0 );
}


void destroy_window1( GtkWidget *widget, gpointer data)
{
   gtk_main_quit();
}
```

23.3.3. Изменение размеров окна

Вам кажется, что окно слишком маленькое и не подходит для нашей программы? Для изменения размеров окна лучше всего использовать функцию

```
void gtk_window_set_default_size(GtkWindow *window,
 gint width, gint heigth);
```

Эта функция устанавливает ширину окна window равной width, а высоту — heigth.

Для изменения позиции окна на экране используется функция

```
void gtk_widget_set_uposition(GtkWindget *widget,
 gint coord_x, gint coord_y);
```

Эту же функцию можно использовать для изменения расположения любого виджита — не обязательно, чтобы последний был окном.

Первая функция объявлена в файле `gtk/gtkwindow.h`, а вторая — в файле `gtk/gtkwidget.h`:

```
#include <gtk/gtkwindow.h>
#include < gtk/gtkwidget.h>
...
gtk_window_set_default_size(window1, 200, 300);
gtk_wigdet_set_uposition(window1, 50, 50);
```

23.3.4. Обработка сигналов

Перед тем, как перейти к следующему пункту, нужно еще раз рассмотреть функцию **gtk_signal_connect()**. Данной функции нужно передать четыре параметра:

- **GtkObject *object** — объект, которому может быть послан сигнал;
- **const gchar *name** — имя сигнала, например, «destroy»;
- **GtkSignalFunc func** — имя функции обратного вызова, то есть функции, которая будет вызвана для обработки сигнала;
- **gpointer data** — любые данные, которые будут переданы функции-обработчику.

Что такое сигнал? Как только пользователь переместил мышь, оконная среда посылает приложению сигнал, оповещающий о том, что мышь была перемещена. Как только пользователь щелкнул мышью, приложение получит сигнал об этом щелчке. Обрабатывать все сигналы может окно, но удобнее для каждого виджита установить собственную реакцию на события.

Функция, которая обрабатывает сигнал, называется по-разному: функция-обработчик (мы будем использовать именно это название), функция обратного вызова и callback-функция. Такой функции нужно передать два параметра (их передаст сама GTK):

- **GtkWidget *widget** — виджит;
- **gpointer data** — данные.

Параметры, которые нужно передать обработчику, зависят от передаваемого сигнала. Например, если бы мы передавали не сигнал «destroy», а сигнал «delete-event», то нужно было бы указать уже три параметра:

- **GtkWidget *widget** — виджит;
- **GdkEvent *event** — событие;
- **gpointer data** — данные.

Вот наиболее часто используемые сигналы:

- **button_press_event** — нажата левая кнопка мыши;
- **button_release_event** — левая кнопка отпущена;
- **motion_notify_event** — движение мыши;
- **delete_event** — удаление объекта;
- **destroy_event** — уничтожение объекта;
- **key_press_event** — нажата клавиша клавиатуры;
- **key_release_event** — клавиша отпущена;
- **enter_notify_event** — указатель мыши вошел в пределы объекта;
- **leave_notify_event** — указатель мыши вышел за пределы объекта;
- **focus_in_event** — объект стал активным (получи фокус);
- **focus_out_event** — объект не активен;
- **drag_begin_event** — начало перемещения объекта;
- **drag_request_event** — запрос на перемещение объекта;
- **drag_end_event** — перемещение объекта;
- **drop_enter_event** — объект перемещен.

Наиболее часто используемые события GDK (используются в функции-обработчике) перечислены ниже:

- GDK_NOTHING — не произошло никакого события;
- GDK_DELETE — удаление;
- GDK_DESTROY — уничтожение;
- GDK_MOTION_NOTIFY — уведомление о перемещении;
- GDK_BUTTON_PRESS — нажата любая кнопка мыши;
- GDK_1BUTTON_PRESS — нажатие первой кнопки мыши;
- GDK_2BUTTON_PRESS — нажатие второй кнопки мыши;
- GDK_3BUTTON_PRESS — нажата третья кнопка;
- GDK_BUTTON_RELEASE — кнопка (любая) отпущена;
- GDK_KEY_PRESS — нажата клавиша;
- GDK_KEY_RELEASE — клавиша отпущена;
- GDK_ENTER_NOTIFY — указатель мыши в пределах объекта (виджита);
- GDK_LEAVE_NOTIFY — указатель мыши вышел за пределы виджита;
- GDK_FOCUS_CHANGE — изменения фокуса ввода;
- GDK_OTHER_EVENT — другое событие.

23.3.5. Виджит событий — EventBox

Далеко не все виджиты связаны с окнами. Например, GtkLabel (надпись), GtkTable (контейнер-таблица), GtkHBox (горизонтальный контейнер), GtkVBox (вертикальный контейнер) и некоторые другие с окнами не связаны.

Если нужно, чтобы эти виджиты реагировали на определенные сигналы, нужно использовать виджит EventBox, позволяющий привязать сигнал к не связанному с окном виджиту. Следующая программа демонстрирует привязку события **button_press_event** к виджиту GtkLabel.

Листинг 23.5. Виджит EventBox

```
#include <gtk/gtk.h>

int main( int argc, char *argv[] )
{
   GtkWidget *window1;   // главное окно
   GtkWidget *event_box1;    // eventbox
   GtkWidget *label;     // надпись

    /* Инициализируем GTK */
   gtk_init( &argc, &argv );

   /* Создаем окно с заголовком "Надпись" */
   window1 = gtk_window_new( GTK_WINDOW_TOPLEVEL );
   gtk_window_set_title( GTK_WINDOW( window1 ), "Надпись" );
  /* Устанавливаем реакцию на закрытие окна */
   gtk_signal_connect(GTK_OBJECT(window1), "destroy",
         GTK_SIGNAL_FUNC(gtk_exit), NULL );

  /* устанавливаем ширину рамки контейнера — окна */

   gtk_container_set_border_width(GTK_CONTAINER(window1), 10);

  /* создаем event_box */
   event_box1 = gtk_event_box_new();
  /* помещаем event_box в контейнер */
   gtk_container_add( GTK_CONTAINER( window1 ), event_box1 );

  /* отображаем event_box */
   gtk_widget_show( event_box1);

  /* создаем надпись */
   label = gtk_label_new( "  -== Click here to exit. ==-  " );
```

```
  /* помещаем надпись в контейнер event_box */
   gtk_container_add(GTK_CONTAINER( event_box1 ), label);
  /* отображаем окно */
   gtk_widget_show(label);

  /* устанавливаем реакцию GtkLabel на щелчок */
  /*(при щелчке - выход) */
   gtk_widget_set_events(event_box1, GDK_BUTTON_PRESS_MASK );
   gtk_signal_connect( GTK_OBJECT( event_box1 ),
        "button_press_event",
                        GTK_SIGNAL_FUNC( gtk_exit ), NULL );

   gtk_widget_realize(event_box1);
  /* изменяем курсор над надписью -
     курсор превратится в руку */
   gdk_window_set_cursor( event_box1->window,
        gdk_cursor_new(GDK_HAND1));

  /* отображаем окно */
   gtk_widget_show(window1);

   gtk_main();

   return 0;
}
```

Откомпилируйте и запустите программу. Над надписью указатель мыши должен принять вид руки (как в браузере над ссылкой). При щелчке на надписи программа будет закрыта.

Если вы хотите создать надпись на русском языке, то подключите заголовок locale.h и вызовите функцию:

```
setlocale( LC_ALL, "ru_RU.KOI8-R");
```

Конечно, значение локали у вас может быть другим. Эту функцию нужно вызвать ДО инициализации GTK+.

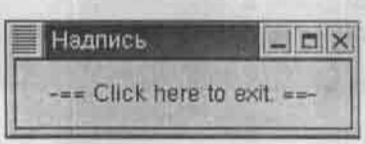


Рис. 23.2. *Надпись*

23.4. Виджиты

23.4.1. Рождение, смерть и состояния виджита

После создания виджита управление ресурсами и памятью, необходимыми ему, выполняется автоматически. Виджиты уничтожаются также автоматически — при разрушении главного окна. Но иногда бывает нужно самостоятельно уничтожить виджит. Сделать это можно с помощью функции:

```
void gtk_widget_destroy (GtkWidget *widget);
```

Эта функция объявлена в файле gtk/gtkwidget.h. При уничтожении виджита также уничтожаются все его дочерние виджиты.

Если вы освобождаете виджит из контейнера функцией:

```
void gtk_container_remove(GtkContainer *cont, GtkWidget *w);
```

то виджит также будет разрушен.

Иногда нужно переместить виджит из одного контейнера в другой без его уничтожения. Это можно сделать так (мы будем перемещать надпись):

```
gtk_widget_ref(GTK_WIDGET(label));
gtk_container_remove(GTK_CONTAINER(cont1), label);
gtk_container_add(GTK_CONTAINER(cont2), label);
```

«Спрятать» виджит можно с помощью функции

```
void gtk_widget_hide(GtkWidget *w);
```

Отобразить виджит снова поможет функция gtk_widget_show().

Виджит может находиться в одном из состояний:

- GTK_STATE_NORMAL — нормальное;
- GTK_STATE_ACTIVE — активное (например, нажата кнопка);
- GTK_STATE_PRELIGTH — над виджитом находится указатель мыши;
- GTK_STATE_SELECTED — виджит выбран (установлен фокус ввода);
- GTK_STATE_INSENSITIVE — виджит не реагирует на ввод (сигналы).

Определить состояние виджита можно так:

```
GTK_WIDGET(w)->state
```

или с помощью макроса:

```
GTK_WIDGET_STATE(wid),
```

описанного в файле gtk/gtkwidget.h

Сделать виджит неактивным можно так:

```
gtk_widget_set_sensitive(widget, FALSE);
```

Если второй параметр функции gtk_widget_set_sensitive() будет равен TRUE, виджит widget станет активным.

Чтобы наш виджит получил фокус ввода, нужно использовать функцию:

```
gtk_widget_grab_focus(widget);
```

23.4.2. Упаковка виджитов, поля ввода и кнопки

Для размещения (упаковки) виджита в окне используются контейнеры. Существуют два основных вида контейнеров. Первый вид в качестве прародителя использует объект класса GtkBin, а второй — объект класса GtkContainer. Контейнеры первого вида могут иметь только один дочерний виджит, поэтому они используются для создания специфических интерфейсов: одной кнопки, рамки, окна.

Контейнеры второго вида более функциональны — они могут иметь много дочерних виджитов. Чаще всего используются контейнеры:

- GtkHBox — позволяет размещать виджиты горизонтально;
- GtkVBox — используется для вертикального размещения виджитов;
- GtkFixed — позволяет размещать виджиты в фиксированных координатах;
- GtkTable — позволяет упаковывать виджиты в виде таблицы.

Наиболее удачным, на мой взгляд, является контейнер GtkTable, поэтому в этом параграфе мы рассмотрим именно его. GtkTable может с успехом заменить и горизонтальный, и вертикальный контейнеры — что нам стоит задать таблицу, состоящую или одной строки или одного столбца?

Кроме контейнера GtkTable, в этом параграфе будут рассмотрены:

- поля для ввода текста и обработка введенной информации;
- кнопки;
- файловый ввод/вывод.

Сейчас мы напишем небольшой конфигуратор, который будет вносить изменения в файл `/etc/resolv.conf`. Напомню вам формат этого файла:

```
domain firma.ru
nameserver 192.168.0.1
nameserver 192.168.0.2
```

Директива domain определяет наш домен, а две директивы nameserver — первый и второй DNS-серверы, соответственно. Наш конфигуратор не будет вносить изменения в настоящий файл `/etc/resolv.conf` — для этого нужны права суперпользователя. При желании можно будет потом скопировать файл `resolv.conf`, сгенерированный нашей программой, в каталог `/etc`.

На рисунке 23.2 изображена уже готовая программа. Работает она так. Когда пользователь введет что-нибудь в поле ввода и нажмет Enter, программа отобразит введенный им текст на консоли. Когда пользователь нажмет Ок, введенная им информация будет еще раз выведена на консоль и записана в файл. При нажатии кнопки Quit программа завершит свою работу. Она должна также завершить работу при нажатии кнопки закрытия окна — в GTK программист сам определяет реакции на стандартные кнопки.

Как видно из рисунка, нам понадобятся три поля ввода, три надписи и две кнопки. Поля ввода мы будем хранить в массиве:

```
GtkWidget *edit[3];
```

Создать поле для ввода можно с помощью функции **gtk_entry_new()**:

```
edit[i] = gtk_entry_new();
```

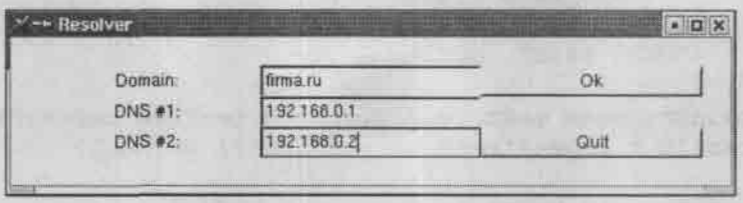


Рис. 23.3. *Учебный конфигуратор*

После создания поля необходимо вызвать функцию **gtk_entry_set_editable()**, иначе пользователь ничего не сможет ввести в это поле.

```
gtk_entry_set_editable(GTK_ENTRY(edit[i]), 1);
```

Ну и, само собой разумеется, нужно установить реакцию на нажатие клавиши Enter — сигнал activate:

```
gtk_signal_connect(GTK_OBJECT(edit[i]), "activate",
                   GTK_SIGNAL_FUNC(enter_callback), edit[i]);
```

Весьма желательно на этапе отладки программы видеть введенную информацию на консоли. Для этого нужно написать такую функцию **enter_callback()**, которая выводила бы содержимое поля на консоль. Получить введенную пользователем информацию очень легко:

```
domain = gtk_entry_get_text(GTK_ENTRY(edit[0]));
dns1 = gtk_entry_get_text(GTK_ENTRY(edit[1]));
dns2 = gtk_entry_get_text(GTK_ENTRY(edit[2]));
```

Реакция на нажатие кнопки Ok будет следующей:

```
void writetofile( GtkWidget *widget,
                  gpointer   data )
{
/* С помощью функции gtk_entry_get_text() мы получаем
введенный пользователем
текст из полей ввода */
domain = gtk_entry_get_text(GTK_ENTRY(edit[0]));
dns1 = gtk_entry_get_text(GTK_ENTRY(edit[1]));
dns2 = gtk_entry_get_text(GTK_ENTRY(edit[2]));

/* Выводим прочитанный текст на консоль */
g_print ("Domain %s\n", domain);
g_print ("DNS1 %s\n", dns1);
g_print ("DNS2 %s\n", dns2);

/* Перезаписываем файл resolv.conf в текущем каталоге */
if ((resolv = fopen("resolv.conf","w")) == NULL)
{
/* Наверное, нет места на диске или прав маловато... */
g_print ("ERR: Cannot open resolve.conf file\n");
gtk_main_quit ();
}

/* Запись в файл */
fprintf(resolv,"domain %s\n",domain);
fprintf(resolv,"nameserver %s\n",dns1);
```

```
fprintf(resolv,"nameserver %s\n",dns2);
fclose(resolv);

}
```

Если ваше окно должно содержать много надписей, то я рекомендую вам поступать так: объявить всего одну переменную, затем создать надпись, поместить ее в контейнер, затем опять создать надпись с использованием этой же переменной, поместить ее в контейнер и т.д. Примерно так:

```
label = gtk_label_new("Domain: ");

gtk_table_attach_defaults (GTK_TABLE(table), label, 0, 1, 0, 1);
gtk_widget_show (label);

label = gtk_label_new("DNS #1: ");
gtk_table_attach_defaults (GTK_TABLE(table), label, 0, 1, 1, 2);
gtk_widget_show (label);

label = gtk_label_new("DNS #2: ");
gtk_table_attach_defaults (GTK_TABLE(table), label, 0, 1, 2, 3);
gtk_widget_show (label);
```

Листинг 23.6 содержит полный код конфигуратора Resolver.

Листинг 23.6. Файл resolver.c

```
#include <gtk/gtk.h>
#include <stdlib.h>
#include <stdio.h>

gchar *domain, *dns1, *dns2;

/* Массив из трех полей ввода. Первое предназначено для
ввода имени домена, два вторых - [1] и [2] - для ввода
IP-адресов серверов DNS*/

GtkWidget *edit[3];

/* Наш файл */
FILE *resolv;

/* Функция записи в файл */
void writetofile( GtkWidget *widget,
                  gpointer   data )
```

```
{
/* С помощью функции gtk_entry_get_text() мы получаем
введенный
пользователем текст из полей ввода */
domain = gtk_entry_get_text(GTK_ENTRY(edit[0]));
dns1 = gtk_entry_get_text(GTK_ENTRY(edit[1]));
dns2 = gtk_entry_get_text(GTK_ENTRY(edit[2]));

/* Выводим прочитанный текст на консоль */
g_print ("Domain %s\n", domain);
g_print ("DNS1 %s\n", dns1);
g_print ("DNS2 %s\n", dns2);

/* Перезаписываем файл resolv.conf в текущем каталоге */
if ((resolv = fopen("resolv.conf","w")) == NULL)
{
/* Наверное, нет места на диске или прав маловато... */
g_print ("ERR: Cannot open resolve.conf file\n");
gtk_main_quit ();
}

/* Запись в файл */
fprintf(resolv,"domain %s\n",domain);
fprintf(resolv,"nameserver %s\n",dns1);
fprintf(resolv,"nameserver %s\n",dns2);
fclose(resolv);

}

/* Эта функция будет запущена, когда пользователь нажмет
кнопку закрытия окна или кнопку Quit */
gint delete_event( GtkWidget *widget,
                   GdkEvent  *event,
                   gpointer   data )
{

/* Функция gtk_main_quit() используется для завершения
работы GTK-программы. Не нужно для этого использовать
exit() */
    gtk_main_quit ();
    return(FALSE);
}
/* Когда пользователь введет текст и нажмет Enter,
введенный им текст
будет выведен на консоль */
void enter_callback( GtkWidget *widget,
```

```
                    GtkWidget *entry )
{
  domain = gtk_entry_get_text(GTK_ENTRY(entry));
  printf("Domain: %s\n", domain);
}

int main( int   argc,
          char *argv[] )
{

    GtkWidget *window;   /* Окно */
    GtkWidget *button;   /* Кнопка */
    GtkWidget *table;    /* Таблица для размещения виджитов */
    GtkWidget *label;    /* Надпись */

/* Как видите, все виджиты одного типа - GtkWidget,
поэтому мы могли бы обойтись даже тремя виджитами - для
окна, таблицы и для всех остальных элементов GUI*/
    int i;

    /* Инициализация любой GTK-программы */
    gtk_init (&argc, &argv);

    /* Создаем новое окно */
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

    /* Устанавливаем заголовок окна */
    gtk_window_set_title (GTK_WINDOW (window), "Resolver");

    /* Устанавливаем реакцию на кнопку закрытия окна.
Сигнал - delete_event Вызываем функцию delete_event(),
которая описана выше */
   gtk_signal_connect (GTK_OBJECT (window), "delete_event",
              GTK_SIGNAL_FUNC (delete_event), NULL);

    /* Устанавливаем рамку окна */
   gtk_container_set_border_width (GTK_CONTAINER (window), 20);

    /* Создаем таблицу 3x3 */
    table = gtk_table_new (3, 3, TRUE);
    /* Помещаем таблицу в контейнер. Обязательно! */
    gtk_container_add (GTK_CONTAINER (window), table);

    /* Рисуем надписи, помещаем их в таблицу и отображаем.
Обратите внимание, что в этом случае нам не нужно
объявлять отдельную переменную для каждой надписи */
```

```
 label = gtk_label_new("Domain: ");
/* О координатах ячеек поговорим после этого листинга */
gtk_table_attach_defaults (GTK_TABLE(table),
    label, 0, 1, 0, 1);
gtk_widget_show (label);

label = gtk_label_new("DNS #1: ");
gtk_table_attach_defaults (GTK_TABLE(table),
      label, 0, 1, 1, 2);
gtk_widget_show (label);

label = gtk_label_new("DNS #2: ");
gtk_table_attach_defaults (GTK_TABLE(table),
      label, 0, 1, 2, 3);
gtk_widget_show (label);

/* Заполняем наш массив полей ввода. По аналогии с
Delphi, я назвал массив edit[]*/
for(i=0; i<3; i++)
{
/* Новое поле */
edit[i] = gtk_entry_new();
/* Если забыть этот оператор, пользователь
ничего не сможет ввести */
gtk_entry_set_editable(GTK_ENTRY(edit[i]), 1);
/* Определяем одну для всех реакцию на сигнал
 activate - нажатие Enter*/
gtk_signal_connect(GTK_OBJECT(edit[i]), "activate",
                   GTK_SIGNAL_FUNC(enter_callback),
                   edit[i]);
/* Помещаем edit[i] в таблицу */
gtk_table_attach_defaults (GTK_TABLE(table),
      edit[i], 1, 2, i, i+1);
/* Показываем */
gtk_widget_show (edit[i]);
}

/* Создаем кнопку "Ok", помещаем в таблицу,
  определяем реакцию на нажатие и показываем */
button = gtk_button_new_with_label ("Ok");
gtk_table_attach_defaults (GTK_TABLE(table),
 button, 2, 3, 0, 1);
gtk_signal_connect(GTK_OBJECT(button),"clicked",
 GTK_SIGNAL_FUNC(writetofile),        NULL);
gtk_widget_show (button);
```

```
    /* То же самое для кнопки Quit */
    button = gtk_button_new_with_label ("Quit");
    gtk_table_attach_defaults (GTK_TABLE(table),
      button, 2, 3, 2, 3);
    gtk_signal_connect(GTK_OBJECT(button),"clicked",
        GTK_SIGNAL_FUNC(delete_event), NULL);
    gtk_widget_show (button);

    gtk_widget_show (table);   /* Показываем таблицу */
    gtk_widget_show (window); /* Показываем окно */

    /* Запускаем GTK-программу */
    gtk_main ();

    return 0;
}
```

Я старался писать подробные комментарии, но все же кое-что осталось в тумане. Это координаты ячеек. Рассмотрим нашу таблицу 3x3:

```
table = gtk_table_new (3, 3, TRUE);
0        1          2        3
 Domain      Поле       Ок
1
  DNS1       Поле
2
  DNS2       Поле      Quit
3
```

Сначала указываются координаты по X, затем — по Y. Вот координаты кнопки Ok: 2,3,0,1. Это означает, что кнопка будет расположена в последнем столбце (между 2 и 3), но в первой строке (между 0 и 1).

```
gtk_table_attach_defaults (GTK_TABLE(table), button, 2, 3, 0, 1);
```

Подробнее рассматривать контейнер GtkTable я не вижу смысла: основные операции, я думаю, вам понятны — это создание таблицы с указанием ее размерности и добавление в таблицу виджита функцией **gtk_table_attach_defaults()**. Еще раз напомню о необходимости отображения виджитов, помещенных в таблицу, и самой таблицы:

```
gtk_widget_show (table);
```

Теперь откомпилируем нашу программу:

```
$ gcc resolv.c -o resolv `gtk-config --cflags`
 `gtk-config --libs`
```

Программа **gtk-config** сообщает компилятору всю необходимую информацию о библиотеке GTK.

Обратите внимание на директиву

```
#include <gtk/gtk.h>
```

Обычно файлы заголовков GTK находятся в другом каталоге, например, gtk-1.2, но это не имеет значения — все необходимые параметры укажет программа **gtk-config**.

В заключение этого пункта перечислим события, характерные для кнопок (таблица 23.3).

События кнопок *Таблица 23.3*

Событие	Описание
clicked	Щелчок
pressed	Кнопка нажата мышью (и пока не отпущена)
released	Кнопка отпущена
enter	Указатель мыши в пределах кнопки
leave	Указатель мыши вышел за пределы кнопки

23.4.3. Переключатели

Переключатели бывают двух типов: зависимые (radio buttons) и независимые (checkbuttons). Переключатели являются кнопками, поэтому для них характерны те же события, что и для кнопок.

Начнем с независимых переключателей, так как они проще в реализации. Создать такой переключатель можно с помощью одной из функций:

```
GtkWidget *gtk_check_button_new( void );
GtkWidget *gtk_check_button_new_with_label ( gchar *label );
```

Первая создает переключатель без надписи (если вы хотите указать надпись отдельно), а вторая — с надписью, которая обычно отображается справа от переключателя. Затем нужно, как всегда, поместить виджиты в контейнер и отобразить.

Зависимые переключатели можно создать тоже с помощью двух аналогичных функций:

```
GtkWidget *gtk_radio_button_new( GSList *group );
GtkWidget *gtk_radio_button_new_with_label( GSList *group,
    ghar *label );
```

Параметр group указывает на принадлежность переключателя к группе. В пределах группы активным может быть только один переключатель. Группу можно создать функцией:

```
GSList *gtk_radio_button_group(GtkRadioButton *radio_button );
```

Однако существует другой способ, позволяющий обойтись без переменной группы — так мы сэкономим память, если групп много:

```
button2 = gtk_radio_button_new_with_label(
          gtk_radio_button_group (GTK_RADIO_BUTTON
          (button1)), "button2");
```

С помощью функции

```
void gtk_toggle_button_set_active( GtkToggleButton
      *toggle_button, gint state );
```

можно сделать одну из кнопок активной.

Следующий листинг демонстрирует работу с тремя зависимыми переключателями и вертикальным контейнером GtkVBox.

Листинг 23.7. Зависимые переключатели

```
#include <gtk/gtk.h>
#include <glib.h>

gint close_application( GtkWidget *widget,
                        GdkEvent  *event,
                        gpointer   data )
{
  gtk_main_quit();
  return(FALSE);
}

int main( int   argc,
          char *argv[] )
{
    GtkWidget *window = NULL;
    GtkWidget *box1;
    GtkWidget *box2;
    GtkWidget *button;
    GtkWidget *separator;
    GSList *group;

    gtk_init(&argc,&argv);
```

```
window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

gtk_signal_connect (GTK_OBJECT (window),
    "delete_event",
    GTK_SIGNAL_FUNC(close_application), NULL);

gtk_window_set_title (GTK_WINDOW (window),
    "Выберите дистрибутив");
gtk_container_set_border_width (GTK_CONTAINER
    (window), 0);

box1 = gtk_vbox_new (FALSE, 0);
gtk_container_add (GTK_CONTAINER (window), box1);
gtk_widget_show (box1);

box2 = gtk_vbox_new (FALSE, 10);
gtk_container_set_border_width (GTK_CONTAINER (box2), 10);
gtk_box_pack_start (GTK_BOX (box1), box2, TRUE, TRUE, 0);
gtk_widget_show (box2);

button = gtk_radio_button_new_with_label (NULL, "Red Hat");
gtk_box_pack_start (GTK_BOX (box2), button,
    TRUE, TRUE, 0);
gtk_widget_show (button);

group = gtk_radio_button_group (GTK_RADIO_BUTTON
    (button));
button = gtk_radio_button_new_with_label(group,
    "Mandrake");
gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
    (button), TRUE);
gtk_box_pack_start (GTK_BOX (box2), button,
    TRUE, TRUE, 0);
gtk_widget_show (button);

button = gtk_radio_button_new_with_label(
      gtk_radio_button_group (GTK_RADIO_BUTTON
      (button)), "ALT Linux");
gtk_box_pack_start (GTK_BOX (box2), button,
      TRUE, TRUE, 0);
gtk_widget_show (button);

separator = gtk_hseparator_new ();
gtk_box_pack_start (GTK_BOX (box1), separator,
      FALSE, TRUE, 0);
gtk_widget_show (separator);
```

```
    box2 = gtk_vbox_new (FALSE, 10);
    gtk_container_set_border_width (GTK_CONTAINER (box2), 10);
    gtk_box_pack_start (GTK_BOX (box1), box2, FALSE, TRUE, 0);
    gtk_widget_show (box2);

    button = gtk_button_new_with_label ("Ok");
    gtk_signal_connect_object (GTK_OBJECT (button),
        "clicked", GTK_SIGNAL_FUNC(close_application),
                                GTK_OBJECT (window));
    gtk_box_pack_start (GTK_BOX (box2), button,
        TRUE, TRUE, 0);
    GTK_WIDGET_SET_FLAGS (button, GTK_CAN_DEFAULT);
    gtk_widget_grab_default (button);
    gtk_widget_show (button);
    gtk_widget_show (window);

    gtk_main();

    return(0);
}
```

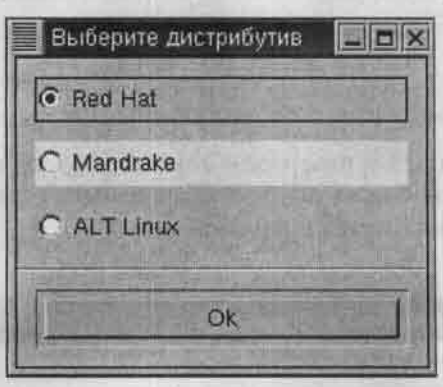


Рис. 23.4. *Зависимые переключатели*

23.4.4. Список

Виджит CList представляет собой список, состоящий из нескольких колонок. Ячейки такого списка могут содержать текстовые значения. Мы можем обратиться отдельно к каждой ячейке списка. Создать список можно одной из функций:

```
GtkWidget *gtk_clist_new ( gint columns );
GtkWidget *gtk_clist_new_with_titles( gint   columns,
gchar *titles[] );
```

Первая функция создает список без заголовков, а вторая с заголовками. Параметр columns задает число колонок.

Добавить элемент в список позволяют функции:

```
gint gtk_clist_prepend( GtkCList *clist, gchar    *text[] );
gint gtk_clist_append( GtkCList *clist, gchar    *text[] );
```

Первая функция добавляет новый элемент в начало списка, а вторая — в его конец. Если вам необходимо вставить элемент в определенную позицию, вам нужно использовать функцию:

```
void gtk_clist_insert( GtkCList *clist, gint row,
   gchar *text[] );
```

Она позволяет вставить новый элемент в строку row. Нумерация строк списка начинается с 0.

Для удаления элементов списка можно использовать одну из функций:

```
void gtk_clist_remove( GtkCList *clist, gint row );
void gtk_clist_clear( GtkCList *clist );
```

Первая удаляет строку row, а вторая очищает весь список.

Рассмотрим листинг 23.8, в котором демонстрируется работа со списком CList. Программа снабжена подробными комментариями, поэтому рекомендую внимательно читать исходный код.

Листинг 23.8. Применение виджита CList

```
#include <gtk/gtk.h>
/* Нужен для функции setlocale() */
#include <locale.h>

/* Добавляет список - обработчик кнопки Добавить */
void button_add_clicked( gpointer data )
{
    int indx;
```

```
    /* Простой список */
    gchar *dist[4][2] = {  { "1", "Red Hat Linux" },
                           { "2", "Mandrake Linux" },
                           { "3", "ALT Linux" },
                           { "4", "ASP Linux" } };

    for ( indx=0 ; indx < 4 ; indx++ )
        gtk_clist_append( (GtkCList *) data, dist[indx]);

    return;
}

/*Обработчик нажатия кнопки Очистить */
void button_clear_clicked( gpointer data )
{
    /* Очищаем список */
    gtk_clist_clear( (GtkCList *) data);

    return;
}

/* Функция прячет/отображает заголовки */
void button_hide_show_clicked( gpointer data )
{
    /*  0 = сейчас видим заголовки  */
    static short int flag = 0;

    if (flag == 0)
    {
        /* прячем заголовки */
        gtk_clist_column_titles_hide((GtkCList *) data);
        flag++;
    }
    else
    {
        /* Отображаем заголовки */
        gtk_clist_column_titles_show((GtkCList *) data);
    }

    return;
}

/* Данная функция будет вызвана, если пользователь выберет
элемент */
void selection_made( GtkWidget       *clist,
                     gint            row,
                     gint            column,
                     GdkEventButton *event,
```

```
                    gpointer        data )
{
    gchar *text;

    /*Получаем выбранный текст (элемент списка) */
    gtk_clist_get_text(GTK_CLIST(clist), row, column, &text);

    /* Просто выводим информацию на консоль */
    g_print("Вы выбрали ряд %d. Колонка %d,
      текст в ячейке %s\n\n", row, column, text);

    return;
}
int main( int    argc,
          gchar *argv[] )
{
    GtkWidget *window;
    GtkWidget *vbox, *hbox;
    GtkWidget *scrolled_window, *clist;
    GtkWidget *button_add, *button_clear, *button_hide_show;
    gchar *titles[2] = { "Номер", "Дистрибутив" };

    setlocale( LC_ALL, "ru_RU.KOI8-R");
                        // Нужно вызвать ДО  gtk_init()

    gtk_init(&argc, &argv);


    window=gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_widget_set_usize(GTK_WIDGET(window), 300, 150);

    gtk_window_set_title(GTK_WINDOW(window), "Список");
    gtk_signal_connect(GTK_OBJECT(window),
                       "destroy",
                       GTK_SIGNAL_FUNC(gtk_main_quit),
                       NULL);
    vbox=gtk_vbox_new(FALSE, 5);
    gtk_container_set_border_width(GTK_CONTAINER(vbox), 5);
    gtk_container_add(GTK_CONTAINER(window), vbox);
    gtk_widget_show(vbox);

    /* Создаем окно с полосками прокрутки и упаковываем в
 него список */
    scrolled_window = gtk_scrolled_window_new
          (NULL, NULL);
    gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW
          (scrolled_window), GTK_POLICY_AUTOMATIC,
          GTK_POLICY_ALWAYS);
```

```
    gtk_box_pack_start(GTK_BOX(vbox), scrolled_window,
          TRUE, TRUE, 0);
    gtk_widget_show (scrolled_window);

    /* Создаем список с двумя колонками */
    clist = gtk_clist_new_with_titles( 2, titles);

    /* Обработка выделения */
    gtk_signal_connect(GTK_OBJECT(clist), "select_row",
                       GTK_SIGNAL_FUNC(selection_made),
                       NULL);

    /* Устанавливаем тень для рамки списка */
    gtk_clist_set_shadow_type (GTK_CLIST(clist),
        GTK_SHADOW_OUT);
    /* Устанавливаем ширину для колонки. Колонки
   нумеруются с 0 */
    gtk_clist_set_column_width (GTK_CLIST(clist), 0, 150);

    /* Помещаем список в контейнер */
    gtk_container_add(GTK_CONTAINER(scrolled_window), clist);
    gtk_widget_show(clist);

    /* Создаем и размещаем кнопки Добавить список,
   Очистить, Спрятать/отобразить заголовок */
    hbox = gtk_hbox_new(FALSE, 0);
    gtk_box_pack_start(GTK_BOX(vbox), hbox, FALSE, TRUE, 0);
    gtk_widget_show(hbox);

    button_add = gtk_button_new_with_label("Добавить");
    button_clear = gtk_button_new_with_label("Очистить");
    button_hide_show = gtk_button_new_with_
        label("Спрятать/отобразить");

    gtk_box_pack_start(GTK_BOX(hbox), button_add,
        TRUE, TRUE, 0);
    gtk_box_pack_start(GTK_BOX(hbox), button_clear,
        TRUE, TRUE, 0);
    gtk_box_pack_start(GTK_BOX(hbox), button_hide_show,
        TRUE, TRUE, 0);

    /* Связываем обработчики */
    gtk_signal_connect_object(GTK_OBJECT(button_add),
 "clicked",GTK_SIGNAL_FUNC(button_add_clicked),
                              (gpointer) clist);
    gtk_signal_connect_object(GTK_OBJECT(button_clear),
"clicked", GTK_SIGNAL_FUNC(button_clear_clicked),
```

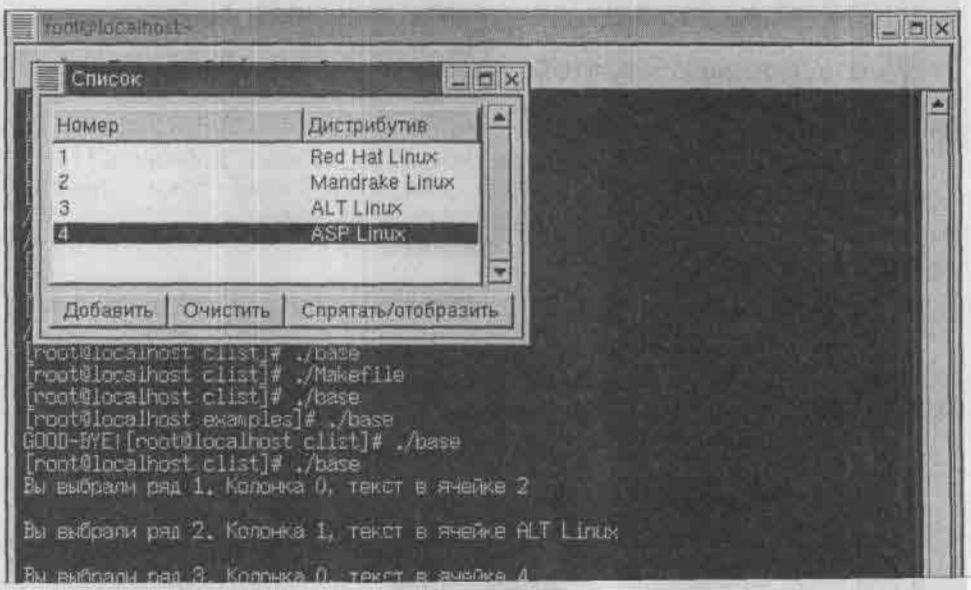
```
                                (gpointer) clist);
    gtk_signal_connect_object(GTK_OBJECT(button_hide_
show), "clicked", GTK_SIGNAL_FUNC(button_hide_show_
clicked),  (gpointer) clist);

    gtk_widget_show(button_add);
    gtk_widget_show(button_clear);
    gtk_widget_show(button_hide_show);

    gtk_widget_show(window);
    gtk_main();

    return(0);
}
```

Программа работает так: при нажатии кнопки **Добавить** создается список, состоящий из названий четырех популярных дистрибутивов Linux. Кнопка **Очистить** очищает список, а **Спрятать/отобразить** прячет или отображает заголовки списка. При щелчке на определенной ячейке списка на консоль выводится соответствующее сообщение — координаты ячейки и ее текст.




```
                    "clicked", (GtkSignalFunc)
                   file_ok_sel, file_dial );
```

```
void destroy( GtkWidget *widget, gpointer data);
static void button_click( GtkWidget *widget, gpointer data );


int main(int argc, char *argv[])
{
   GtkWidget *window;
   GtkWidget *button;
   GtkWidget *label;

   w_ctrl     ctrl;
   gchar     *caption;

   setlocale( LC_ALL, "ru_RU.KOI8-R");

   caption = g_strdup_printf("Доброго времени суток!");

   gtk_init( &argc, &argv );

   window = gtk_window_new( GTK_WINDOW_TOPLEVEL );

   gtk_signal_connect(GTK_OBJECT(window),"delete_event",
       GTK_SIGNAL_FUNC(delete_event),NULL);

   gtk_signal_connect(GTK_OBJECT(window), "destroy",
       GTK_SIGNAL_FUNC(destroy), &ctrl );

   gtk_window_set_title(GTK_WINDOW(window), caption);

   gtk_container_set_border_width( GTK_CONTAINER(window),
10 );

   button = gtk_button_new();
   label = gtk_label_new( " -==  Нажмите кнопку  ==-   " );

   ctrl.app_window = window;
   ctrl.label = label;

   gtk_container_add( GTK_CONTAINER( button ), label );
   gtk_container_add( GTK_CONTAINER( window ), button );

   gtk_signal_connect( GTK_OBJECT( button ), "clicked",
                       GTK_SIGNAL_FUNC( button_click ),
&ctrl );
```

```
   gtk_widget_show_all(window);

    gtk_main ();

    return(0);
}

// ******************************************************

void quit_confirm(GtkWidget *widget)
{
   GtkWidget *quit_form;
   GtkWidget *label;
   GtkWidget *yes_button, *no_button;

   quit_form = gtk_dialog_new();
   gtk_window_set_position(GTK_WINDOW( quit_form ),
                           GTK_WIN_POS_CENTER );

   gtk_container_set_border_width( GTK_CONTAINER
                                   ( quit_form ), 10 );

   label = gtk_label_new ( "\n Вы действительно хотите
выйти? \n" );
   yes_button = gtk_button_new_with_label( "Да" );
   no_button = gtk_button_new_with_label( "Нет" );


   gtk_signal_connect_object( GTK_OBJECT
    ( yes_button ), "clicked", GTK_SIGNAL_FUNC
    ( gtk_widget_destroy ), (gpointer)widget );

gtk_container_add (GTK_CONTAINER( GTK_DIALOG
    ( quit_form )->action_area ),  yes_button);

   gtk_signal_connect_object( GTK_OBJECT( no_but-
ton ), "clicked", GTK_SIGNAL_FUNC( gtk_widget_destroy
),(gpointer)quit_form );

   gtk_container_add (GTK_CONTAINER( GTK_DIALOG
      ( quit_form )->action_area ), no_button);

   gtk_container_add( GTK_CONTAINER( GTK_DIALOG
      ( quit_form )->vbox ), label);

   gtk_window_set_modal( GTK_WINDOW( quit_form ) ,TRUE );
```

```
    gtk_widget_show_all( quit_form );
}


gint delete_event(GtkWidget *widget, GdkEvent  *event,
gpointer data)
{
   quit_confirm(widget);

   return( TRUE );
}

void destroy( GtkWidget *widget, gpointer   data )
{
    printf("GOOD-BYE!");
    gtk_main_quit();
}


static void button_click( GtkWidget *widget, gpointer data )
{
   static gint i = 0;
   GtkWidget *app_window;
   GtkWidget *label;
   gchar msg[256];

   app_window = GTK_WIDGET( ((w_ctrl *)data)->app_window );
   label = GTK_WIDGET( ((w_ctrl *)data)->label );

   i++;
   sprintf(msg, "Вы нажали кнопку: %d раз(а)", i );
   gtk_label_set_text(GTK_LABEL( label ),msg);
}
```

Думаю, текст программы ясен без лишних комментариев. Нужно лишь пояснить один очень важный момент. Обратите внимание на то, что мы переопределили обработчик для события `delete_event`.

```
gtk_signal_connect(GTK_OBJECT(window), "delete_event",
 GTK_SIGNAL_FUNC(delete_event),NULL);
gtk_signal_connect(GTK_OBJECT(window), "destroy",
 GTK_SIGNAL_FUNC(destroy), &ctrl );
```

Если данный обработчик возвращает FALSE, то будет вызвана функция destroy(), которая уничтожит окно. Мы переписали функцию delete_event() так, чтобы она всегда возвращала TRUE, то есть функция destroy() вообще не будет вызвана. Но в таком случае наше окно вообще никогда не закроется, поэтому нужно, чтобы кто-то позаботился о закрытии окна. Это будет функция quit_confirm(), отображающая диалог завершения работы.

```
gint delete_event(GtkWidget *widget, GdkEvent  *event,
gpointer data)
{
quit_confirm(widget);
return( TRUE );
}
```

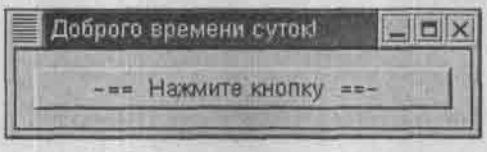


Рис. 23.6. *Программа только запущена*

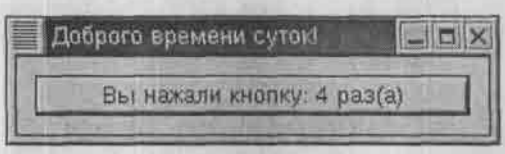


Рис. 23.7. *Пользователь нажал на кнопку 4 раза*

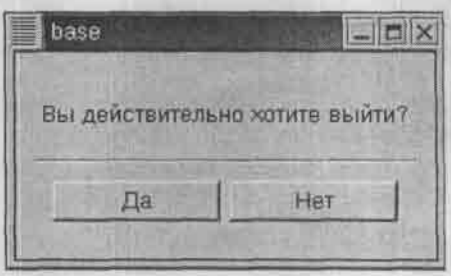


Рис. 23.8. *Диалог завершения работы*

Теперь рассмотрим обработчики событий кнопок Да и Нет диалога:

```
gtk_signal_connect_object( GTK_OBJECT( yes_button ),
«clicked», GTK_SIGNAL_FUNC( gtk_widget_destroy ),
(gpointer)widget );
gtk_signal_connect_object( GTK_OBJECT
( no_button ), «clicked», GTK_SIGNAL_FUNC
( gtk_widget_destroy ),(gpointer)quit_form );
```

Кнопка `yes_button` вызывает функцию **gtk_widget_destroy()** и передает ей параметр (gpointer)widget, то есть уничтожает главное окно приложения, а кнопка `no_button` передает функции **gtk_widget_destroy()** параметр (gpointer)quit_form, который указывает на окно диалога, то есть при нажатии этой кнопки будет закрыто само окно нашего диалога.

23.4.7. Меню

Меню программы вручную писать довольно неудобно, поэтому для разработки меню воспользуемся студией разработки графического интерфейса Glade.

23.4.8. Иерархия виджитов

```
GtkObject
  +GtkWidget
  | +GtkMisc
  | | +GtkLabel
  | | | +GtkAccelLabel
  | | | `GtkTipsQuery
  | | +GtkArrow
  | | +GtkImage
  | | `GtkPixmap
  | +GtkContainer
  | | +GtkBin
  | | | +GtkAlignment
  | | | +GtkFrame
  | | | | `GtkAspectFrame
  | | | +GtkButton
  | | | | +GtkToggleButton
  | | | | | `GtkCheckButton
  | | | | |   `GtkRadioButton
  | | | | `GtkOptionMenu
  | | | +GtkItem
  | | | | +GtkMenuItem
  | | | | | +GtkCheckMenuItem
```

```
| | | | | | `GtkRadioMenuItem
| | | | | `GtkTearoffMenuItem
| | | | +GtkListItem
| | | | `GtkTreeItem
| | | +GtkWindow
| | | | +GtkColorSelectionDialog
| | | | +GtkDialog
| | | | | `GtkInputDialog
| | | | +GtkDrawWindow
| | | | +GtkFileSelection
| | | | +GtkFontSelectionDialog
| | | | `GtkPlug
| | | +GtkEventBox
| | | +GtkHandleBox
| | | +GtkScrolledWindow
| | | `GtkViewport
| | +GtkBox
| | | +GtkButtonBox
| | | | +GtkHButtonBox
| | | | `GtkVButtonBox
| | | +GtkVBox
| | | | +GtkColorSelection
| | | | `GtkGammaCurve
| | | `GtkHBox
| | |   +GtkCombo
| | |   `GtkStatusbar
| | +GtkCList
| | | `GtkCTree
| | +GtkFixed
| | +GtkNotebook
| | | `GtkFontSelection
| | +GtkPaned
| | | +GtkHPaned
| | | `GtkVPaned
| | +GtkLayout
| | +GtkList
| | +GtkMenuShell
| | | +GtkMenuBar
| | | `GtkMenu
| | +GtkPacker
| | +GtkSocket
| | +GtkTable
| | +GtkToolbar
| | `GtkTree
| +GtkCalendar
```

```
| +GtkDrawingArea
| | `GtkCurve
| +GtkEditable
| | +GtkEntry
| | | `GtkSpinButton
| | `GtkText
| +GtkRuler
| | +GtkHRuler
| | `GtkVRuler
| +GtkRange
| | +GtkScale
| | | +GtkHScale
| | | `GtkVScale
| | `GtkScrollbar
| |   +GtkHScrollbar
| |   `GtkVScrollbar
| +GtkSeparator
| | +GtkHSeparator
| | `GtkVSeparator
| +GtkPreview
| `GtkProgress
|   `GtkProgressBar
+GtkData
| +GtkAdjustment
| `GtkTooltips
`GtkItemFactory
```

Глава 24

СТУДИЯ GLADE

- ЧТО ТАКОЕ GLADE?
- ЗНАКОМСТВО С GLADE
- РАБОТА С ПРОЕКТОМ
- СОЗДАНИЕ МЕНЮ
- ИНТЕРЕСНЫЕ ВИДЖИТЫ

24.1. Что такое Glade?

В предыдущей главе мы рассмотрели далеко не все виджиты и далеко не все сигналы, на которые должны реагировать рассмотренные виджиты. Это было сделано умышленно: для начала вам нужно было разобраться, что такое виджит и «с чем его едят», а также понять, что такое сигнал и функция-обработчик. С этого момента вам не нужно беспокоиться о том, что вы забыли имя виджита, имя сигнала или что означает та или иная маска сигнала, вам также не придется писать вручную прототипы функций обработчиков. За вас все сделает студия разработки графического интерфейса Glade. Почему вам не нужно знать имя виджита? Вам достаточно выбрать нужный вам виджит из палитры и поместить его в контейнер, причем контейнер также представляется визуально. Вам не нужно рисовать на бумажке или представлять перед собой координаты ячеек таблицы контейнера Glade, поскольку вы все видите перед собой. Если вы забыли (или не знаете) имя нужного вам сигнала, Glade при написании обработчика позволит вам выбрать нужный сигнал из списка. Если вам нужно замаскировать определенное событие, нет проблем: Glade не только отобразит доступные маски, но и выведет их описание. Код функций-обработчиков Glade также напишет самостоятельно — вам лишь нужно будет заполнить тело функции-обработчика, поскольку Glade пока еще не научился читать ваши мысли.

Что вам нужно для работы с Glade? Просто установите пакет glade (он входит в состав большинства современных дистрибутивов) и наслаждайтесь. Особенно приятно работать с Glade будет тем программистам, которые привыкли работать с визуальными средами разработки, например, Delphi или Visual C.

24.2. Знакомство с Glade

Наше знакомство с Glade мы начнем с разработки калькулятора, но прежде нужно познакомиться с интерфейсом самой студии. После запуска Glade вы увидите три окна:

- Главное окно Glade — используется для операций с проектом, например, создания, открытия, сохранения, указания опций.
- Палитра — набор виджитов, которые вы можете поместить в контейнер.
- Свойства — окно свойств виджитов, в котором отображаются свойства активного виджита.

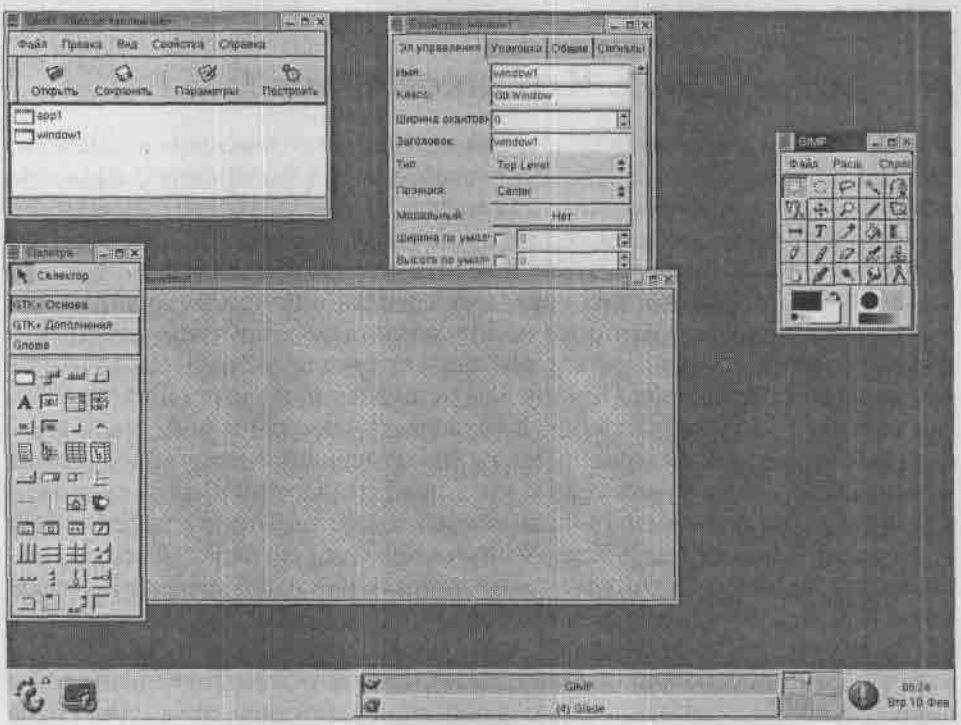


Рис. 24.1. *Студия Glade*

Палитра содержит три набора виджитов:

- GTK+ Основа — основные виджиты;
- GTK+ Дополнения — дополнительные виджиты;
- Gnome — специальные виджиты для создания GNOME-приложений.

К основным виджитам относятся:

- Окно — основной элемент вашего приложения.
- Главное меню — строка меню приложения.
- Панель инструментов — на этой панели обычно находятся кнопки быстрого доступа к некоторым элементам главного меню, например, Создать, Открыть, Сохранить, Выход и т.д.

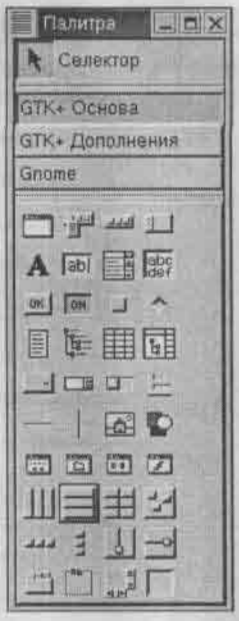


Рис. 24.2. *Палитра*

- Ползунок — позволяет изменять размеры некоторых виджитов; обычно ползунок ставится между двумя панелями инструментов.
- Метка — надпись.
- Поле ввода текста — название говорит само за себя.
- Комб. список — комбинированный список (Combo Box).
- Текстовый бокс — область для ввода нескольких строк текста; может использоваться в качестве основы простого текстового редактора.
- Кнопка — думаю, этот виджит в комментариях не нуждается.
- Кнопка-переключатель — кнопка, которая может находиться в одном из двух положений: включено или выключено.
- Флажок — независимый переключатель.
- Радиокнопка — зависимый переключатель.
- Список — простой список.
- Дерево — иерархическая схема элементов.
- Колоночный список — это список CList из главы 23.
- Колоночное дерево — что-то среднее между деревом и колоночным списком.
- Меню параметров — выпадающий список опций, напоминает список ComboBox (см. рис. 24.3, тип окна).
- Крутящаяся кнопка — не знаю, почему ее так назвали, на самом деле это поле с кнопками уменьшения/увеличения значения (как поле ввода ширины или высоты окна, см. рис. 24.3).

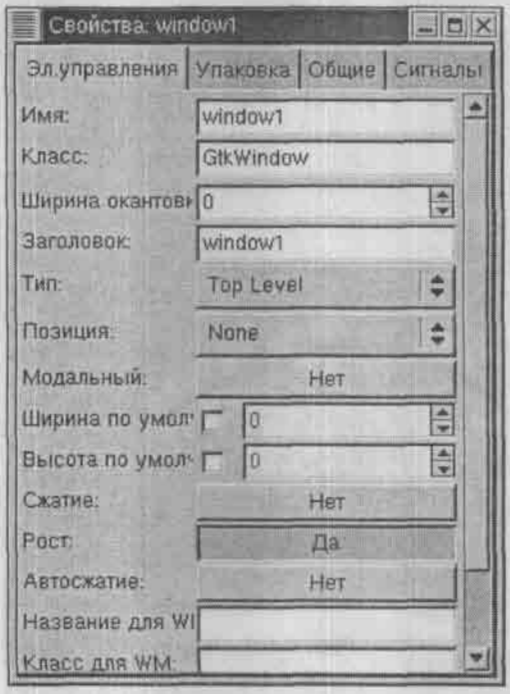


Рис. 24.3. *Окно свойств*

- Индикатор выполнения — вы использовали программу GnoRPM для установки пакетов? Пока пакет устанавливался, в небольшом окошке ползла небольшая шкала, информирующая нас о процессе установки. Это индикатор выполнения.
- Строка статуса — небольшая панель с текстовым полем, точнее, надписью, содержащей информацию о статусе приложения.
- Горизонтальный, вертикальный разделители — эти виджиты также не нуждаются в комментариях.
- Область для рисования — область, в которой можно рисовать мышью.
- Диалог — произвольное диалоговое окно.
- Диалог выбора файла — это наш старый знакомый, работу с которым мы рассматривали в прошлой главе.
- Диалог выбора шрифта — позволяет выбрать шрифт.
- Окно выбора цвета — позволяет выбрать цвет.
- Горизонтальный бокс — контейнер GtkHBox.
- Вертикальный бокс — контейнер GtkVBox.
- Таблица — контейнер GtkTable.
- Фиксированная позиция — контейнер GtkFixed.
- Горизонтальные панели, вертикальные панели — набор горизонтальных и вертикальных панелей.

- Записная книжка — виджит, состоящий из нескольких страниц, обычно используется для создания окна свойств.
- Рамка — небольшая рамка, обычно используется для объединения нескольких радиокнопок или других виджитов, устанавливающих параметры, в одну группу.
- Прокручиваемое окно — окно с полосами прокрутки.
- Окно просмотра.

Дополнительными виджитами являются:

- Горизонтальная, вертикальная шкала — эти элементы похожи на индикатор выполнения, но используются не для наблюдения за ходом процесса, а для установки значений (такие шкалы используются в диалоге выбора цвета).
- Горизонтальная, вертикальная линейка — аналогичны шкалам, но немного по-другому нарисованы (в виде линейки).
- Событийный бокс — это виджит EventBox, с которым мы уже знакомы.
- Метка с клавишей ускорения — надпись с клавишей быстрого доступа.
- Календарь — виджит для выбора даты.
- Ниспадающее меню — обычное ниспадающее меню.
- Кривая, Гамма-кривая — служат для отображения различных кривых.
- Горизонтальная, вертикальная полосы прокрутки — вместо этих виджитов лучше использовать прокручиваемое окно.
- Предварительный просмотр — область предварительного просмотра.
- Выбор шрифта, выбор цвета — виджиты выбора шрифта и цвета соответственно.
- Диалог ввода — диалог для ввода информации.
- Изображение — используется для вывода картинки.
- Пользовательский элемент управления.
- Стрелка.

На вкладке Gnome вы найдете следующие виджиты:

- Окно приложения Gnome — создает готовый шаблон окна приложения — с меню, панелью инструментов, кнопками быстрого доступа и строкой статуса.
- Диалоговое окно Gnome — произвольный диалог в стиле Gnome.
- Окно сообщений Gnome — используется для вывода сообщений.
- Окно Gnome «О программе».
- Выбор цвета Gnome — небольшая кнопка с изображением выбранного цвета, щелчок на которой приводит к появлению окна выбора цвета.
- Выбор шрифта Gnome — небольшая кнопка, щелчок по которой приводит к появлению окна выбора шрифта.

- Пиктограмма Gnome.
- Кнопка HRef-ссылки Gnome.
- Поле ввода Gnome — поле для ввода информации.
- Ввод файла Gnome — небольшое текстовое поле с кнопкой, щелкнув по которой, вы увидите диалог выбора файла.
- Ввод числа Gnome — поля для ввода числа.
- Правка даты — поле для редактирования даты.
- Шкала — красивая шкала, напоминающая спидометр.
- Часы.
- Строка приложения Gnome — напоминает строку статуса.
- Калькулятор — уже готовый калькулятор.
- GnomeCanvas — область для рисования.
- Список пиктограмм.
- Выбор пиктограммы — раскрывающийся список, позволяющий выбрать пиктограмму.
- Диалоговое окно свойств.
- Помощник — окно помощи Gnome.
- Картинка Gnome.
- Аниматор — отображает анимированное изображение.

Перечислять свойства и сигналы каждого виджита я не буду, поскольку в окне Свойств до такой степени все просто, что вы разберетесь без моих комментариев.

Итак, приступим. Создайте главный виджит — окно. Для этого просто щелкните мышью на иконке окна в Палитре. Сразу же после этого в окне Свойств вы можете установить свойства нашего окна.

Сейчас нас интересуют следующие свойства:

- Имя: имя виджита — это идентификатор, и оно должно соответствовать правилами написания имен идентификаторов, то есть никакой кириллицы!
- Заголовок: заголовок окна может содержать символы любого алфавита, только потом, в функции main(), не забудьте вызвать функцию локализации.
- Ширина окантовки: ширина рамки окна в пикселях.
- Тип — устанавливает тип окна:
- Top Level — главное окно;
- Dialog — диалоговое окно;
- Popup — всплывающее окно.
- Позиция — позиция окна на экране при запуске:
  - None — как при разработке;
  - Center — строго в центре;
  - Mouse — в текущей позиции указателя мыши.
- Ширина, высота — можно указать значения явно, а можно изменить размер окна с помощью мыши (второй способ часто оказывается удобнее).

Чтобы впоследствии увидеть окно свойств виджита window (нашего окна), нужно перейти в главное окно Glade и выбрать из списка окон нужное вам окно.

Чтобы удалить:

- **Контейнер**: щелкните на любом виджите контейнера правой кнопкой мыши, выберите имя контейнера и команду Удалить. Будут удалены также все дочерние виджиты контейнера.
- **Виджит:** щелкните на нем правой кнопкой мыши и выберите команду Удалить.
- **Строку контейнера**: удалите дочерний виджит, щелкните правой кнопкой на строке контейнера и выберите команду Удалить.
- **Окно**: в главном окне Glade выберите нужное вам окно и нажмите клавишу Del.

Добавьте вертикальный контейнер из двух строк для размещения виджитов. В верхней будет размещено текстовое поле ввода, а в нижней — контейнер-таблица, состоящая из 5 строк и 4 столбцов. В этой таблице будут расположены кнопки калькулятора.

Поместите в верхнюю строку контейнера GtkVBox текстовое поле. Перейдите в окно свойств и установите следующие свойства текстового поля (рис. 24.4):

- Вкладка **Эл. управления**
  - Редактировать: Да
  - Видимость текста: Да
  - Максимальная длина: 0
  - Текст: 0
- Вкладка **Общие**
  - Высота: 50
  - Видимость: Да
  - Чувствительность: Да
  - Фокусировка: Да
  - Имеет фокус: Нет
  - События: 0000000000000

После этого в первую ячейку таблицы добавьте кнопку и установите ее свойства следующим образом:

- Вкладка **Эл. управления**
  - Ширина окантовки: 0
  - Метка: ON
- Вкладка **Общие**
  - Ширина: 70
  - Высота: 70

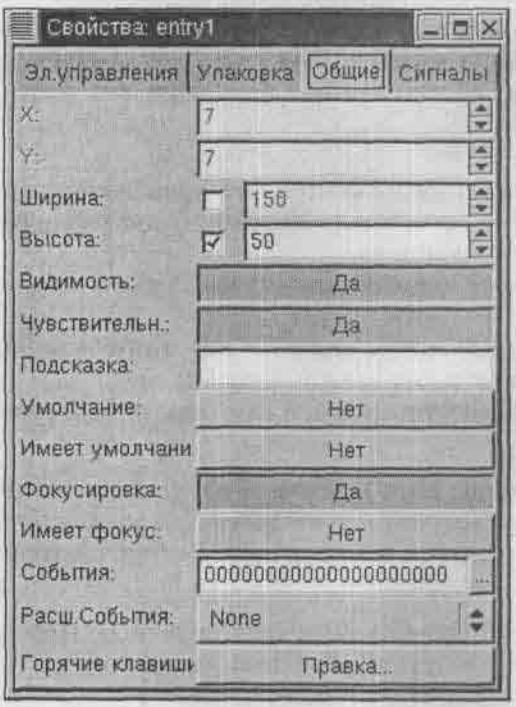


Рис. 24.4. *Свойства текстового поля: вкладка Общие*

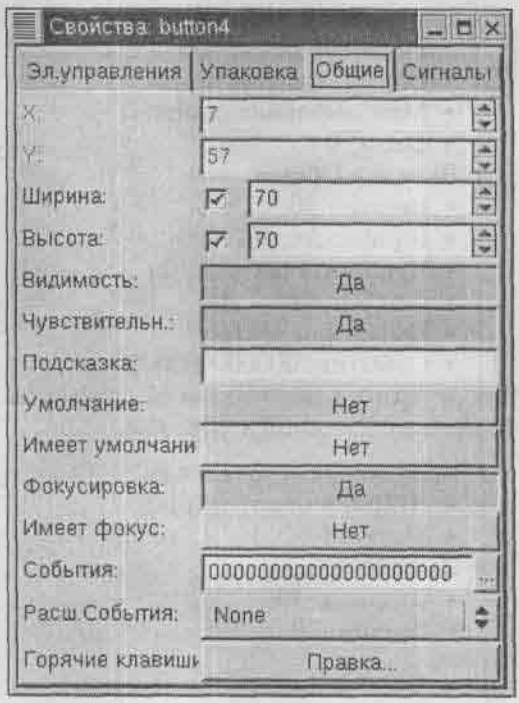


Рис. 24.5. *Вкладка Общие для кнопки*

Если вы забыли (или не знаете), что означает то или иное свойство виджита, подведите указатель мыши к метке поля свойства, и рядом с ним отобразится подсказка. Если же вам трудно установить маску для событий, нажмите кнопку «...» (рис. 24.5) рядом с полем выбора события, и вы увидите описание масок событий.

Скопируйте получившуюся кнопку в следующую ячейку (Ctrl+C, Ctrl+V.). Для новой кнопки установите свойство Метка: CE. Проверить, что находится в буфере обмена, можно с помощью команды главного меню Glade: Вид, Буфер обмена.

Точно так же создайте еще 16 кнопок и разместите их так, как показано на рисунке 24.6.

Сейчас установим функции-обработчики для наших кнопок. Выделите кнопку ON и перейдите в окно свойств на вкладку Сигналы.

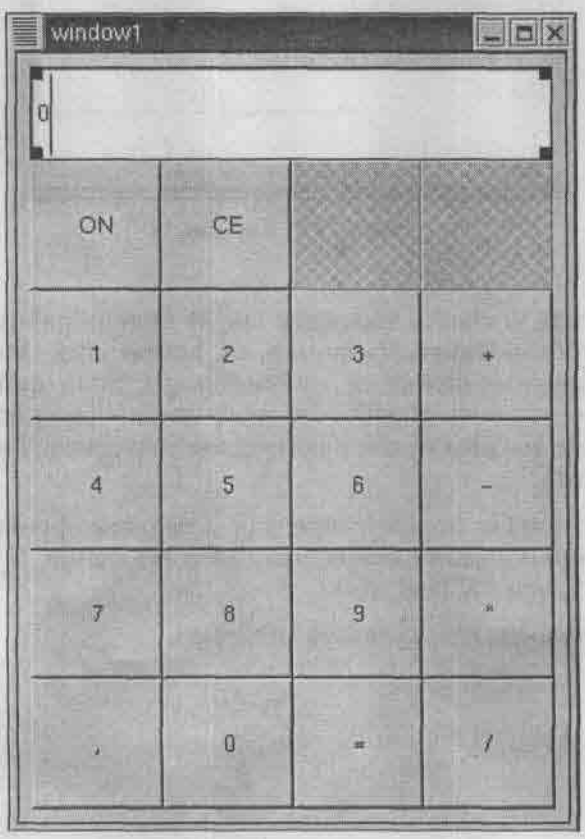


Рис. 24.6. *Калькулятор*

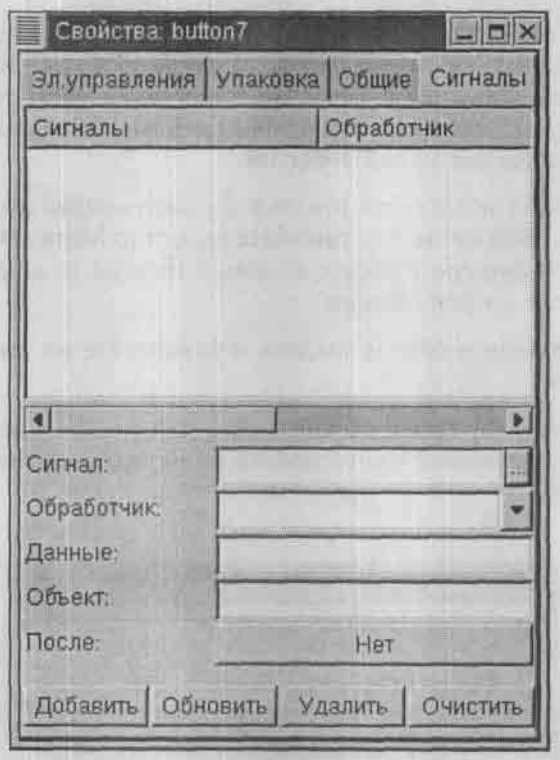


Рис. 24.7. *Сигналы*

Выберите сигнал clicked и нажмите **Ок**. В поле обработчика введите имя функции-обработчика, например, on_button_click. Можно выбрать одну из стандартных функций, например, gtk_main_quit(), но не для этой кнопки — это ведь обработчик включения калькулятора, поэтому код для него мы должны будем писать самостоятельно. Теперь нажмите кнопку **Добавить**.

Аналогично создайте обработчики для остальных функций. Советую давать функциям понятные имена, например, on_button_N_click, где N — число от 0 до 9, или on_plus_click.

Вот и все, интерфейс калькулятора построен.

24.3. Работа с проектом

Прежде всего нужно сохранить наш проект. Нажмите кнопку **Сохранить** в главном окне Glade. Перед сохранением Glade предложит установить свойства проекта:

- Вкладка «Общие»
  - Каталог проекта /home/ваше_имя/Проекты/Проект — сразу же измените Проекты на Projects и Проект на имя нашего проекта — calc. Использовать написанные кириллицей названия не запрещается, но и не рекомендуется.
  - Название проекта: Calc.
  - Название программы: calc — так будет назван исполнимый файл.
  - Файл проекта: calc.glade.
  - Каталог исходного кода: src (~/Projects/Calc/src).
  - Каталог картинок: pixmap.
  - Язык: C.
  - Включить поддержку Gnome: Да.
- Вкладка «Параметры C»
  - Все оставить по умолчанию.

В результате в каталоге ~/Projects/Calc будет создан файл проекта calc.glade. Этот файл полностью описывает наш проект на языке XML. Фрагмент этого файла приведен в листинге 25.1 — просто для общего развития.

Листинг 25.1. Фрагмент файла проекта calc.glade

```
<?xml version="1.0"?>
<GTK-Interface>

<project>
  <name>Calc</name>
  <program_name>calc</program_name>
  <directory></directory>
  <source_directory>src</source_directory>
  <pixmaps_directory>pixmaps</pixmaps_directory>
  <language>C</language>
  <gnome_support>True</gnome_support>
  <gettext_support>True</gettext_support>
</project>

<widget>
  <class>GtkWindow</class>
```

```
  <name>window1</name>
  <border_width>7</border_width>
  <title>window1</title>
  <type>GTK_WINDOW_TOPLEVEL</type>
  <position>GTK_WIN_POS_NONE</position>
  <modal>False</modal>
  <allow_shrink>False</allow_shrink>
  <allow_grow>True</allow_grow>
  <auto_shrink>False</auto_shrink>

  <widget>
    <class>GtkVBox</class>
    <name>vbox2</name>
    <homogeneous>False</homogeneous>
    <spacing>0</spacing>

    <widget>
      <class>GtkEntry</class>
      <name>entry1</name>
      <height>50</height>
      <can_focus>True</can_focus>
      <editable>True</editable>
      <text_visible>True</text_visible>
      <text_max_length>0</text_max_length>
      <text>0</text>
      <child>
<padding>0</padding>
<expand>False</expand>
<fill>False</fill>
      </child>
    </widget>
...

  </widget>
...

</widget>
 ...
</GTK-Interface>
```

Но кроме этого файла в каталоге Calc ничего нет. А где же исходный код? А где картинки? Чтобы Glade сгенерировала исходный код, нажмите кнопку **Построить** в главном окне Glade. В результате в каталоге calc будет создана структура подкаталогов, показанная на рис. 24.8.

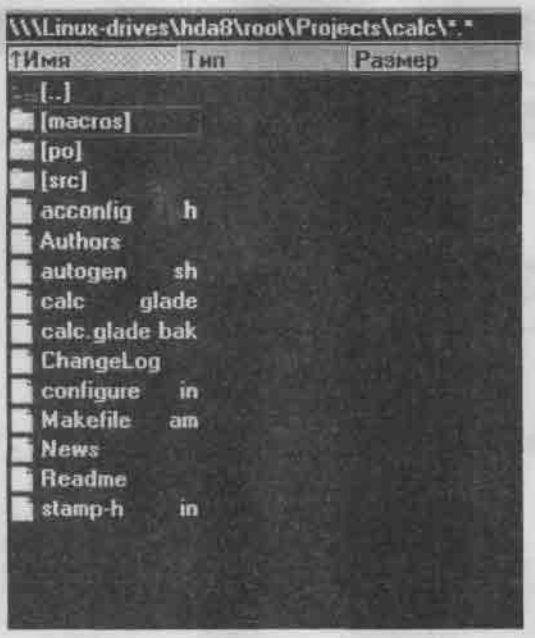


Рис. 24.8. *Структура каталогов проекта*

В каталоге `macros` находятся макросы для поддержки среды Gnome. Каталог `po` предназначен только для вас — в него вы будете вносить сведения об изменениях в проекте. Каталог `src` содержит исходный код проекта.

О каталоге `src` нужно поговорить подробнее. В нем находятся следующие файлы:

- interface.*, support.* — эти файлы сгенерированы Glade, и вам не нужно их редактировать;
- callback.* — функции-обработчики. Автоматически сгенерированные функции первоначально не делают ничего, это только заготовки. Чтобы ваша программа что-нибудь делала, вам нужно отредактировать файл callback.c;
- make.c — вы можете редактировать этот файл, если вам это нужно.

Рассмотрим файл callback.c — его нужно редактировать в первую очередь. Glade создала для вас заготовки функций-обработчиков следующего вида:

Листинг 25.2. Фрагмент файла callback.c

```
#ifdef HAVE_CONFIG_H
#  include <config.h>
#endif

#include <gnome.h>

#include "callbacks.h"
#include "interface.h"
#include "support.h"


void
on_button_clicked              (GtkButton       *button,
                               gpointer
                               user_data)
{

}

void
ce_button_clicked              (GtkButton       *button,
                              gpointer
                              user_data)
{

}
...

void
on_button9_clicked            (GtkButton       *button,
                                     gpointer

                             user_data)
{

}
```

Вписав в эти заготовки код, выполняющий нужные вам действия, можно попытаться собрать программу. Для этого перейдите в каталог `calc` и введите команду make.

24.4. Создание меню

Если вам нужно стандартное меню приложения, состоящее из пунктов Файл, Правка, Вид и т.п., используйте виджит **Окно приложения Gnome** (рис. 24.9) — вы его найдете на страничке виджитов Gnome.

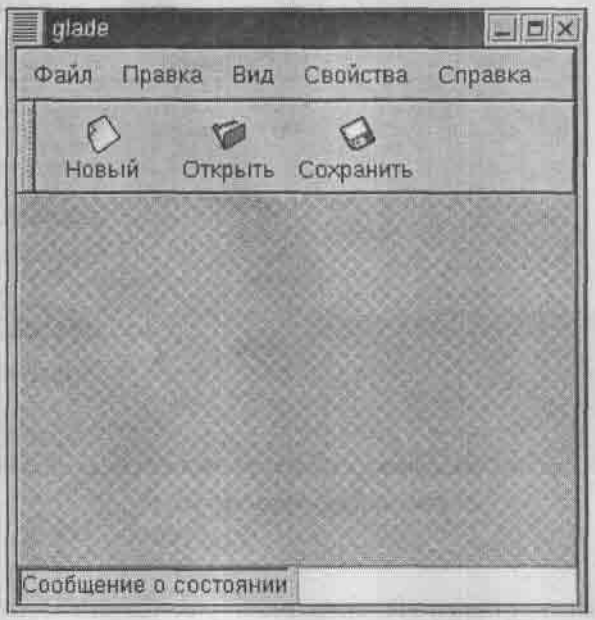


Рис. 24.9. *Окно приложения Gnome*

Это уже готовый шаблон окна с меню, панелью инструментов и строкой статуса. Удобно?

Если же вам нужно нестандартное меню, то выберите виджит Меню, который находится на вкладке основных виджитов окна Палитра. В окне свойств меню нажмите кнопку Правка меню. В открывшемся окне редактора вы можете создавать пункты меню.

Для создания пункта меню нажмите кнопку **Добавить** и введите следующую информацию:

- Метка — эту надпись увидит пользователь.
- Имя — это идентификатор пункта меню.
- Обработчик — имя функции-обработчика.
- Иконка — иконка, соответствующая пункту меню..
- Подсказка — обычно эта подсказка отображается в строке статуса при выборе пункта меню.

Рис. 24.10. *Редактор меню*

- Тип элемента — обычный, зависимый (радиокнопка) или независимый (флажок) переключатель.
- Ускоритель — горячая комбинация клавиш.

С помощью стрелок вы можете изменять положение пунктов меню.

24.5. Интересные виджиты

Интересных виджитов намного больше, чем будет описано в этом пункте, — каждый виджит по-своему интересен. Но мне больше всего понравились следующие виджиты:

- Шкала — виджит, напоминающий спидометр, находится на вкладке элементов Gnome.
- Выбор цвета — если этот виджит расположить в контейнере, то получится окно выбора цвета, которое очень похоже на стандартное окно выбора цвета среды Gnome (вкладка Дополнительно).
- Выбор шрифта — позволяет пользователю выбрать шрифт (вкладка Дополнительно).

- Диалоговое окно свойств — уже готовое окно свойств (напоминает окно свойств проекта), содержащее вкладки и кнопки **Ок**, **Применить**, **Закрыть**, **Справка** (вкладка Gnome).

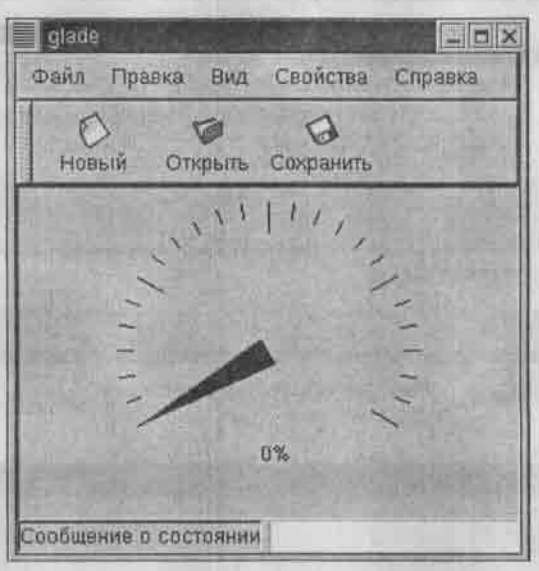


Рис. 24.11. *Шкала*

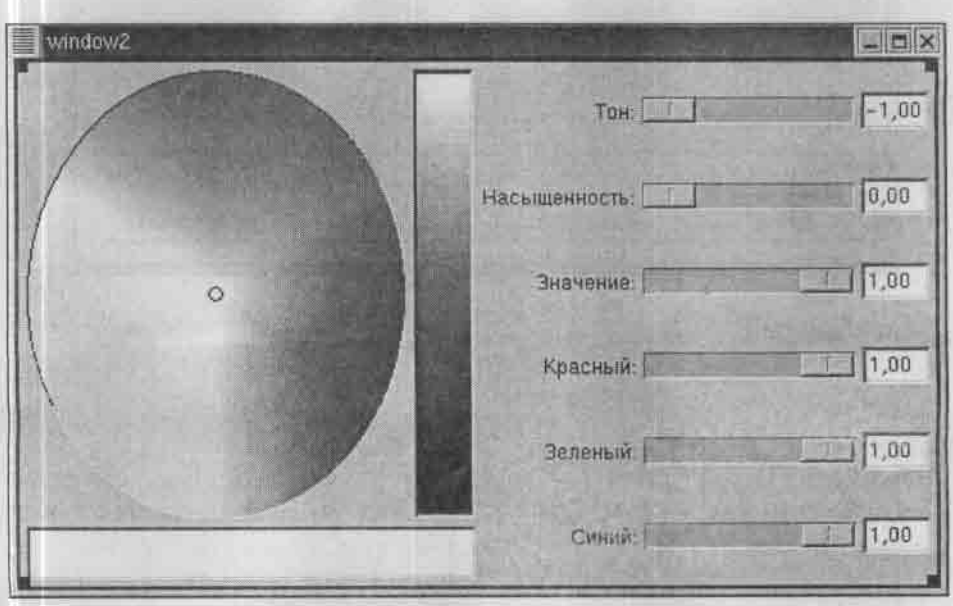


Рис. 24.12. *Выбор цвета*

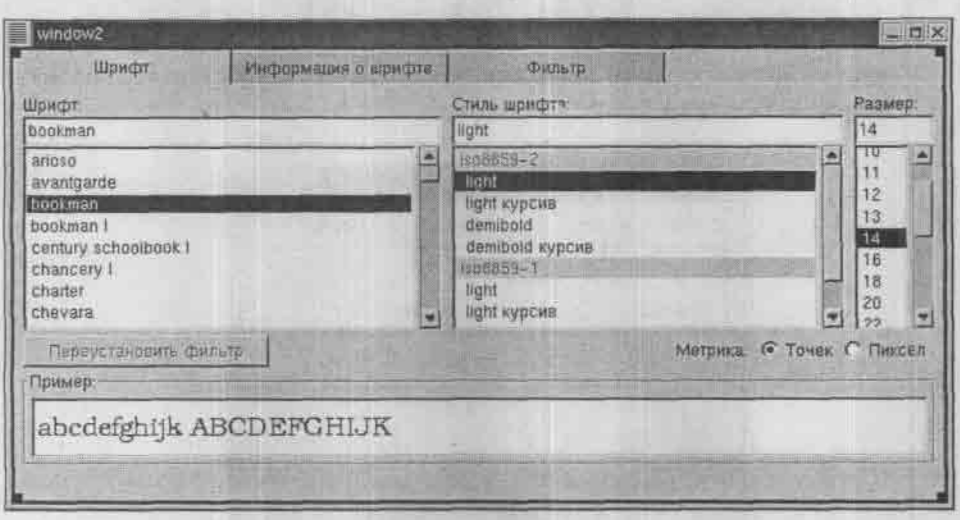


Рис. 24.13. *Выбор шрифта*

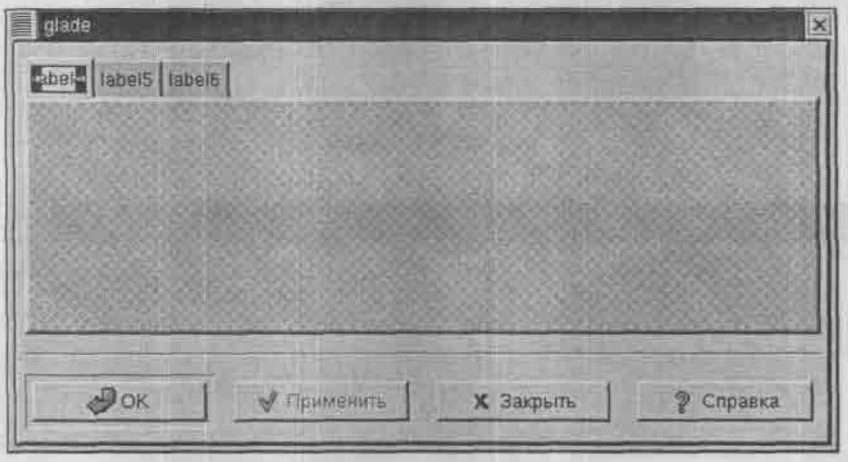


Рис. 24.14. *Окно свойств*

Глава 25

ПАКЕТ DIALOG

- ЧТО ТАКОЕ DIALOG?
- СООБЩЕНИЯ
- ВИДЖИТ YES-NO
- ОКНО ВВОДА ТЕКСТА
- ЗАВИСИМЫЕ И НЕЗАВИСИМЫЕ ПЕРЕКЛЮЧАТЕЛИ
- ОРГАНИЗАЦИЯ МЕНЮ
- КАЛЕНДАРЬ
- ШКАЛА ПРОГРЕССА

25.1. Что такое Dialog?

Пакет dialog служит для вывода диалоговых окон в сценариях **bash**. Элементы пользовательского интерфейса мы будем, как и в GTK+, называть виджитами. Пакет dialog позволяет использовать следующие виджиты:

- **Infobox, Msgbox** — выводят информационные сообщения;
- **Inputbox** — принимает ввод текстовой информации;
- **Calendar** — позволяет выбрать дату;
- **Radiolist** — список зависимых переключателей;
- **Checklist** — список независимых переключателей;
- **Menubox** — используется для организации меню;
- **Gauge** — шкала прогресса.

Установив пакет dialog, в каталоге `/usr/share/doc/dialog-<версия>` вы найдете много примеров применения этого замечательного пакета.

25.2. Сообщения

Начнем с самого простого — отображения небольших текстовых сообщений. Для этой цели можно использовать два виджита — Infobox или Msgbox. Первый отличается от второго тем, что не ждет подтверждения пользователя о прочтении, а закрывается через некоторое время. Виджит Msgbox ждет, пока пользователь нажмет кнопку **Ok** или клавишу Enter.

Рассмотрим пример использования виджита Msgbox:

Листинг 25.1. Виджит Msgbox

```
#!/bin/sh
DIALOG=${DIALOG=dialog}

$DIALOG --title "MESSAGE BOX" --clear \
        --msgbox "Этот виджит используется для вывода
сообщений.  Ждет, пока пользователь нажмет Enter" 10 41
case $? in
  0)
    echo "OK";;
  255)
    echo "Нажата ESC.";;
esac
```

Теперь разберемся, что есть что. Параметр --title программы dialog задает заголовок виджита (рис. 25.1) Параметр --msgbox сообщает программе тип виджита, который нужно отобразить. После этого параметра нужно указать текст сообщения, заключенный в кавычки. В тексте можно использовать управляющие последовательности (\b,\n,\t,\a и др.). После сообщения задаются размеры виджита.

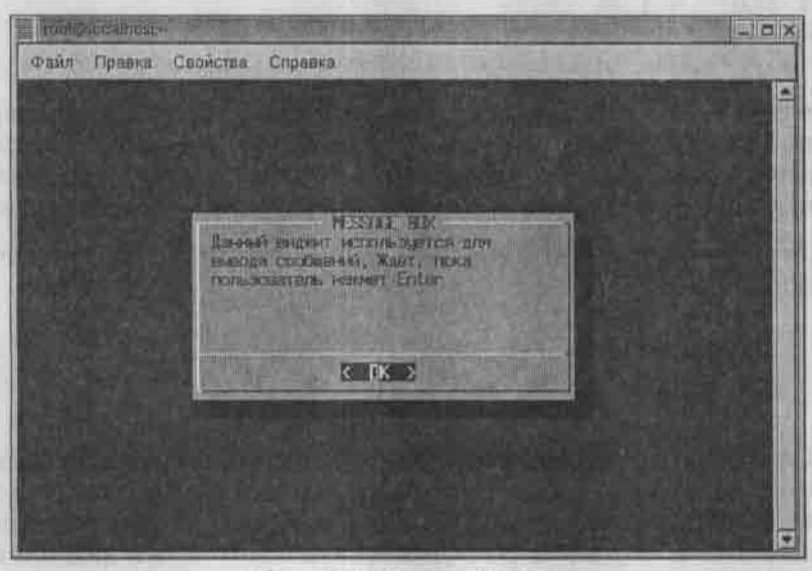


Рис. 25.1. *Виджит Msgbox*

Конструкция case проверяет возвращенное программой dialog значение. Если пользователь нажал **Ок** (или <Ввод>), то программа возвращает 0, а если клавишу Esc, то 255.

Виджит Infobox следует использовать так:

Листинг 25.2. Использование виджита Infobox

```
#! /bin/sh
DIALOG=${DIALOG=dialog}

# ждем указанное количество секунд
left=10
unit="секунд"

while test $left != 0
do
$DIALOG --sleep 1 \
```

```
        --title "INFO BOX" \
        --infobox "Это информационный бокс. Он отличается от
        msgbox тем, что не ждет, пока пользователь нажмет Enter,
        а прекращает работу по истечении времени
        (в данном примере - 10 секунд).
У вас $left $unit чтобы прочитать это сообщение..." 10 52
left=`expr $left - 1`
test $left = 1 && unit="second"
done
```

Параметр `--sleep 1` означает, что программа `dialog` будет перерисовывать окно через одну секунду. Параметр `--title` задает заголовок виджита, `--infobox` — определяет тип виджита, после определения виджита следует отображаемое сообщение, а потом — размер виджита.

В цикле while мы проверяем, сколько секунд осталось, уменьшая значение переменной left.

25.3. Виджит Yes-no

Часто встречаются диалоговые окна, спрашивающие пользователя, согласен ли он с действиями программы, например, «Вы точно хотите выйти?» или «Удалить этот файл?». Пользователю же предлагается два варианта ответа — Да или Нет. Для организации такого диалога предназначен виджит yes-no.

Следующий листинг демонстрирует работу с этим виджитом:

Листинг 25.3. Работа с виджитом yes-no

```
#!/bin/sh
DIALOG=${DIALOG=dialog}

$DIALOG --title "YES/NO BOX" --clear \
        --yesno "Отформатировать /dev/hda1?" 5 41

case $? in
  0)
    echo "Да.";;
  1)
    echo "Нет.";;
  255)
    echo "ESC.";;
esac
```

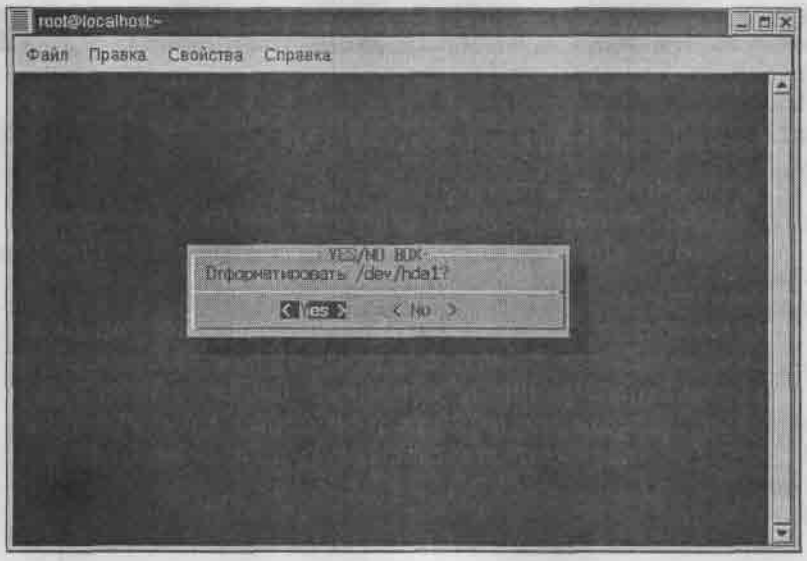


Рис. 25.2. *Виджит yes-no*

Виджит используется так же, как и предыдущие: название виджита (yesno), текстовое сообщение, размеры виджита. Программа dialog возвращает следующие значения:

- 0, если пользователь нажал кнопку **Yes**;
- 1, если пользователь нажал кнопку **No**;
- 255, если пользователь нажал клавишу Esc.

25.4. Окно ввода текста

Следующим по частоте применения после информационных виджитов и виджита yes-no следует виджит ввода текстовой информации — Inputbox. Принцип работы данного виджита следующий:

1. Мы определяем имя временного файла, в который будет записано введенное пользователем сообщение.
2. Вызываем программу `dialog` с параметром `--inputbox`.
3. Перенаправляем вывод программы во временный файл (программа выведет введенное пользователем значение).
4. Выводим или обрабатываем каким-либо другим способом содержимое временного файла.

Листинг 25.4. Виджит Inputbox

```
#!/bin/sh
DIALOG=${DIALOG=dialog}
tempfile=`tempfile 2>/dev/null` || tempfile=/tmp/test$$
trap "rm -f $tempfile" 0 1 2 5 15

$DIALOG --title "INPUT BOX" --clear \
        --inputbox "Данный виджит используется для ввода
информации\n\n
Введите свое имя:" 16 51 2> $tempfile

retval=$?

case $retval in
  0)
    echo "Вы ввели `cat $tempfile`";;
  1)
    echo "Нажата Cancel";;
  255)
    if test -s $tempfile ; then
      cat $tempfile
    else
      echo "Нажата ESC."
    fi
    ;;
esac
```

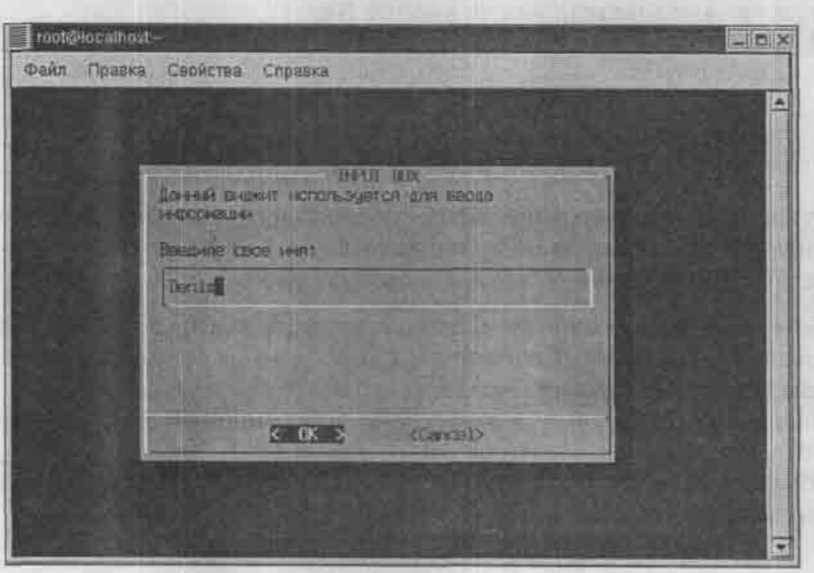


Рис. 25.3. Виджит InputBox

Значения, возвращаемые программой `dialog`:

- 0, если пользователь что-то ввел и нажал Enter или просто нажал <Ввод>, не введя ничего (тогда временный файл будет пуст);
- 1, если пользователь нажал Cancel;
- 255, если пользователь нажал Esc.

25.5. Зависимые и независимые переключатели

Программа `dialog` позволяет вам использовать в своих сценариях зависимые и независимые переключатели. Зависимые переключатели реализуются виджитом **radiolist**, а независимые — **checklist**.

Листинг 25.5. Независимые переключатели

```
#! /bin/sh
DIALOG=${DIALOG=dialog}
tempfile=`tempfile 2>/dev/null` || tempfile=/tmp/test$$
trap "rm -f $tempfile" 0 1 2 5 15

$DIALOG --backtitle "Пример" \
        --title "Независимые переключатели" \
        --checklist "Данный пример демонстрирует работу
независимых \n\
переключателей, реализуемых с помощью пакета Dialog \n\
Используйте ПРОБЕЛ для включения или выключения
переключателей. \n\n\
  Какие произведения вы читали?" 20 61 5 \
        "Почти как люди"  "Саймак" off \
        "Фауст"    "Гете" ON \
        "Мастер и Маргарита" "Булгаков" off \
        "Мир теней"    "Саймак" off \
        "Демон"  "Лермонтов" on 2> $tempfile

retval=$?

choice=`cat $tempfile`
case $retval in
  0)
    echo "Вы выбрали '$choice'";;
  1)
    echo "Нажата Cancel";;
  255)
    echo "Нажата ESC";;
esac
```

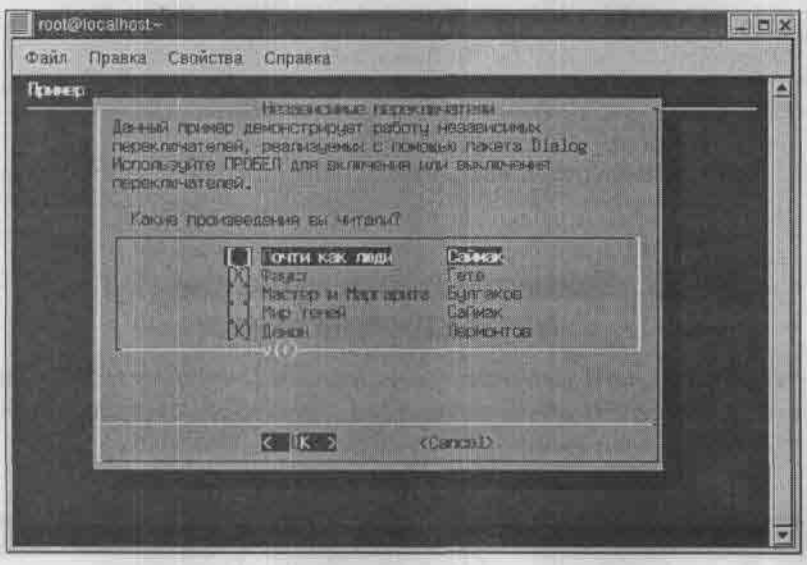


Рис. 25.4. *Независимые переключатели*

Выбранные пользователем значения помещаются во временный файл, который вам нужно будет обработать. Сейчас мы просто выведем его на консоль. Элемент списка **checklist** может находиться в одном из положений — On или Off. Эти значения не чувствительны к регистру. Во временный файл выводятся только включенные переключатели.

Работа с виджитом **radiolist** осуществляется так же, как и в виджитом **checklist**, за исключением того, что может быть активным лишь один элемент и во временный файл будет записан именно этот элемент.

Листинг 25.6. Зависимые переключатели

```
#! /bin/sh
DIALOG=${DIALOG=dialog}
tempfile=`tempfile 2>/dev/null` || tempfile=/tmp/test$$
trap "rm -f $tempfile" 0 1 2 5 15

$DIALOG --backtitle "Пример" \
        --title "RADIOLIST BOX" --clear \
        --radiolist "Это список зависимых переключателей \n\
Он позволяет выбрать только один вариант из списка \n\
  Какую из этих книг вы читали последней?" 20 81 5 \
        "Почти как люди"  "Саймак" off \
        "Фауст"    "Гете" off \
```

```
        "Мастер и Маргарита" "Булгаков" ON \
        "Мир теней"    "Саймак" off \
        "Демон"  "Лермонтов" on 2> $tempfile

retval=$?

choice=`cat $tempfile`
case $retval in
  0)
    echo "Вы выбрали '$choice'";;

  1)
    echo "Нажата Cancel";;
  255)
    echo "Нажата ESC";;
esac
```

25.6. Организация меню

Виджит **MenuBox** очень похож на **radiolist** — они прямо-таки братья-близнецы, только у **menubox** нет слева переключателя включено/выключено. Использовать **MenuBox** нужно так же, как и **radiolist**, но не указывая **on** или **off** для элементов меню.

Листинг 25.7. Меню

```
#!/bin/sh
DIALOG=${DIALOG=dialog}
tempfile=`tempfile 2>/dev/null` || tempfile=/tmp/test$$
trap "rm -f $tempfile" 0 1 2 5 15

$DIALOG --clear --title "MENU BOX" \
        --menu "Этот виджит поможет вам организовать
небольшое меню \n\
MENU BOX предоставляет пользователю выбрать один вариант
из списка \n\
Данный виджит также позволяет прокручивать возможные
варианты \n\
Вы можете использовать стрелки ВВЕРХ/ВНИЗ,
а также клавиши  \n\
1-9 для выбора.\n\
          Выберите вашу ОС:" 20 71 4 \
        "Linux"  "А что, разве есть другие операционные
        системы" \
```

```
        "FreeBSD" "Это лучшая ОС" \
        "Windows" "Мы кроме Windows ничего не видели..." \
        "MSDOS"  "На моей двойке ничего другого не
запускается :(" 2> $tempfile

retval=$?

choice=`cat $tempfile`

case $retval in
  0)    echo "Ваш выбор '$choice'";;
  1)
    echo "Нажата Cancel";;
  255)
    echo "Нажата Esc.";;
esac
```

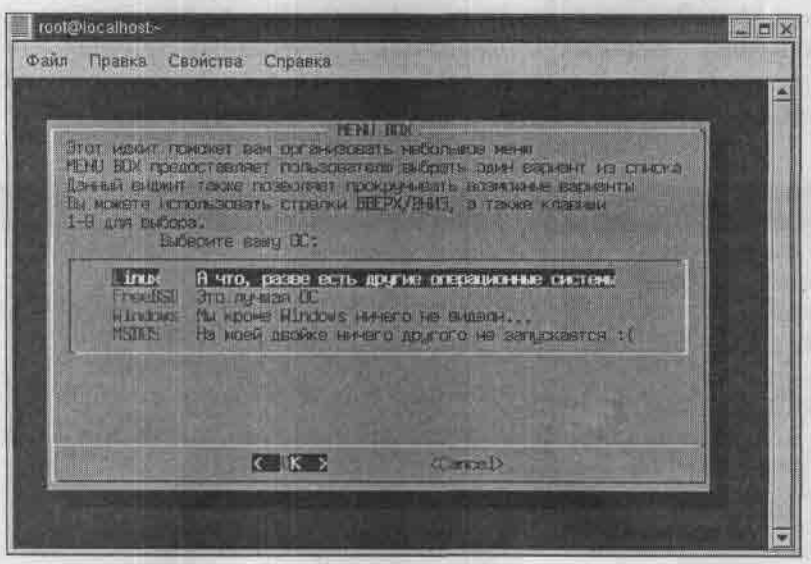


Рис. 25.5. *Виджит menubox*

25.7. Календарь

Этот виджит позволяет пользователю удобно ввести дату. Работать с ним нужно так: с помощью клавиши Tab выбираете нужный элемент управления: кнопка **Ok**, **Cancel**, поле изменения месяца, поле изменения года, поле выбора числа; с помощью стрелок вверх/вниз указываете нужное значения месяца, года, числа и нажимаете <Ввод>. Выбранная вами дата будет отображена на консоли.

Листинг 25.8. Календарь

```
#!/bin/sh
: ${DIALOG=dialog}

USERDATE=`$DIALOG --stdout --title "CALENDAR" --calendar
"Выберите дату..." \
0 0 1 1 2004`

case $? in
  0)
    echo "Вы выбрали дату: $USERDATE.";;
  1)
    echo "Нажата Cancel.";;
  255)
    echo "Диалог закрыт";;
esac
```

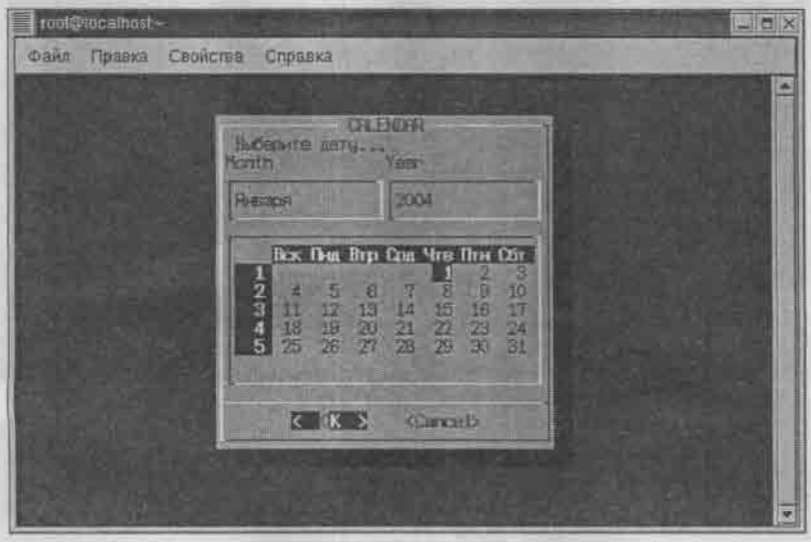


Рис. 25.6. Календарь

25.8. Шкала прогресса

Для информирования пользователя о ходе процесса, например, копирования или обработки файла, целесообразно использовать виджит **gauge** (шкала прогресса).

Листинг 25.9. Шкала прогресса

```
#!/bin/sh
DIALOG=${DIALOG=dialog}

PCT=10
(
while test $PCT != 100
do
echo "XXX"
echo $PCT
echo "Выполнено\n\
($PCT %)"
echo "XXX"
PCT=`expr $PCT + 10`
# засыпаем на 1 секунду, 1 секунда — это 10%
sleep 1
done
) |

$DIALOG --title "Шкала" --gauge "Шкала" 20 70 0
```

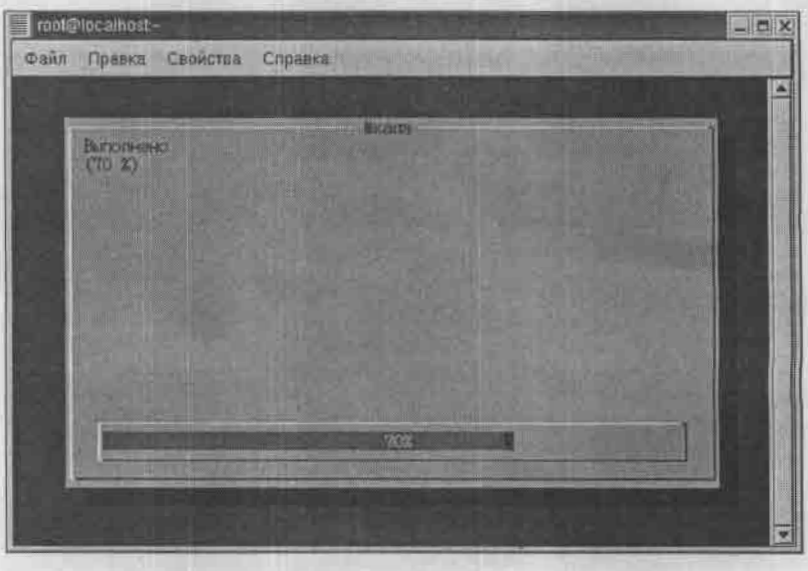


Рис. 25.7. *Шкала прогресса*

Глава 26

ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ В LINUX

- СПОСОБЫ ВЗАИМОДЕЙСТВИЯ
- ПОЛУДУПЛЕКСНЫЕ КАНАЛЫ
- КАНАЛЫ ТИПА FIFO
- ОСНОВНЫЕ ПРИНЦИПЫ SYSTEM V IPC
- ОЧЕРЕДИ СООБЩЕНИЙ
- СЕМАФОРЫ
- РАЗДЕЛЯЕМЫЕ СЕГМЕНТЫ ПАМЯТИ

26.1. Способы взаимодействия

Процессы, как и люди, могут «общаться» между собой, то есть обмениваться информацией. В главе 3 мы бегло рассмотрели два средства межпроцессного взаимодействия (IPC, *Inter-Process Communication*): полудуплексные каналы (конвейеры) и сигналы, но в UNIX-системах таких средств значительно больше. В этой главе я перечислю остальные средства IPC и покажу, как использовать их в программном коде.

С давних времен существуют именованные каналы FIFO (*First In — First Out*) и сетевые гнезда (сокеты). Вместе с конвейерами и сигналами они составляют IPC типа BSD. Компания AT&T вместе с операционной системой System V предложила три новых вида IPC:

- семафоры;
- разделяемая память;
- очереди сообщений.

В операционной системе Linux поддерживаются оба типа IPC — System V и BSD, то есть в Linux мы можем использовать все вышеперечисленные способы IPC.

26.2. Полудуплексные каналы

Напомню, что канал — это способ связи стандартного вывода одного процесса со стандартным вводом другого. Каналы — старожилы UNIX: они появились еще в самых первых версиях UNIX. Полудуплексные каналы позволяют обмениваться информацией только в одном направлении. Если процесс-предок передает информацию со своего стандартного вывода на стандартный ввод потомка — это пример полудуплексного канала.

Что такое перенаправление ввода/вывода и как его использовать из командной строки, вы уже знаете (п.3.4.6). Сейчас я покажу, как осуществить перенаправление программным путем, то есть без вмешательства пользователя.

Вызов **system**() порождает дочерний процесс, позволяя ему читать данные со стандартного ввода (stdin) и писать на стандартный вывод (stdout). Иногда нам нужно передать данные дочернему процессу или, наоборот, получить информацию от порожденного процесса. Другими словами, мы хотим, чтобы дочерний процесс получал данные не со стандартного ввода, а от родительского процесса или/и выводил информацию не на стандартный вывод, а передавал ее процессу-предку. Ввод/вывод между процессами осуществляется с помощью системного вызова **popen**(). Этот вызов должен быть выполнен ДО вызова fork(), чтобы файловые дескрипторы были унаследованы дочерним процессом.

```
FILE * popen(const char * команда, const char * режим_доступа);
```

Первый параметр — это название программы, которую мы хотим запустить в дочернем процессе. Второй параметр определяет режим доступа. Установите значение «r», если вам нужно читать вывод дочернего процесса, если же вам нужно передать информацию на стандартный ввод порожденного процесса, установите значение «w». Режима двустороннего обмена не существует.

Вызов popen() возвращает указатель FILE* или пустой указатель NULL, если вызов не удался. Так же, как и при работе с обыкновенными файлами, после завершения операции ввода/вывода вы должны закрыть канал вызовом **pclose**(). Во время работы с потоком рекомендую использовать вызов **fflush**(), чтобы предотвратить задержки из-за буферизации.

Теперь несколько простых примеров. Предположим, что нам нужно вывести на стандартный вывод имена всех текстовых файлов, содержащихся в текущем каталоге. Это можно очень просто сделать с помощью вызова **system**():

```
system("ls *.txt");
```

Это уж совсем тривиальная задача — мы просто выводим данные, но никак не обрабатываем их. Как получить все имена текстовых файлов и обработать их в программе?

```
// открываем поток
FILE *fp = popen("ls *.txt", "r");

// в цикле читаем имена всех текстовых файлов
while ((fname = fgets(...,fp);) != EOF)
 {
// обрабатываем полученное значение переменной fname
 }

// закрываем поток
pclose(fp);
```

Этот фрагмент кода в особых комментариях не нуждается. Сначала мы создаем поток для чтения (доступ «r») информации от порожденного процесса (ls *.txt). Затем в цикле while читаем имена файлов до тех пор, пока не будет достигнут конец файла. После окончания операции ввода/вывода закрываем поток вызовом **pclose(fp)**.

Вот теперь мы готовы к тому, чтобы рассмотреть более серьезный пример. В этом примере мы будем передавать данные дочернему процессу. Задача такова: у нас есть две программы. Первая программа передает второй какую-нибудь информацию, вторая обрабатывает ее и выводит на стандартный вывод результат.

Листинг 26.1. Родительский процесс

```
#include <stdio.h>
#include <sys/wait.h>
#include <unistd.h>

int main()
{
 char buff[1024]={0};
 FILE * cp;        // cp - child process - дочерний процесс
 int status;

 // Открываем канал. Дочерний процесс - /usr/bin/child
 cp=popen("/usr/bin/child", "w");
   if (!cp)
    {
      printf("не могу открыть канал.\n");
      exit(1)
     }

 printf("Введите информацию для передачи дочернему
процессу " );

 // читаем ввод пользователя
 fgets(buff, sizeof (buff), stdin);

 // передаем данные дочернему процессу
 fprintf(cp, "%s\n", buff);
 // "выталкиваем" содержимое буфера в канал
 fflush(cp);

 // закрываем канал и проверяем состояние вызова pclose()
 status=pclose(cp);
```

```
 if (!WIFEXITED(status))
  printf("ошибка при закрытии канала\n");

 printf("Завершение работы родительского процесса\n");
 return 0;
}
```

Листинг 26.2. Дочерний процесс — исходный код /usr/bin/child

```
#include <stdio.h>
int main()
{
 char buff[1024]={0};
 fgets(buff, sizeof (buff), stdin);
 printf("Прочитал со стандартного ввода: %s\n",buff);
 printf("Завершение работы дочернего процесса\n");
 return 0;
}
```

26.3. Каналы типа FIFO

Канал FIFO — это канал, основанный на принципе очереди: «первым вошел, первым вышел». От обычного канала канал FIFO отличается следующим:

- Канал FIFO сохраняется в файловой системе в виде файла, поэтому каналы FIFO называются именованными.
- С именованным каналом, как с обычным файлом, могут работать все процессы, а не только предок и потомки.
- В отличие от полудуплексного канала, находящегося в ядре, канал FIFO находится в файловой системе и остается там даже после завершения обмена данными. Для следующего использования канала его не нужно заново создавать.

Создать именованный канал можно с помощью командного интерпретатора:

```
$ mknod myFIFO p
$ mkfifo a=rw myFIFO
```

или системного вызова mknod():

```
int mknod( char *pathname, mode_t mode, dev_t dev );
```

Функция **mknod()** используется не только для создания каналов FIFO. Она может создать любой i-узел (*inode*) файловой системы: файл, устройство, канал FIFO. Функция возвращает 0, если создание узла прошло

успешно, или -1, если произошла ошибка. Проанализировать ошибку можно с помощью переменной errno, которая равна:

- EFAULT, ENOTDIR, ENOENT — неправильно задан путь;
- EACCESS — у вас недостаточно прав;
- ENAMETOOLONG — слишком длинный путь.

Пример создания FIFO-канала:

```
mknod("FIFO", S_IFIFO|0666, 0);
```

В текущем каталоге будет создан канал FIFO с правами доступа 0666.

Указывая права доступа создаваемого файла, помните, что они находятся под влиянием **umask**. Поэтому, если вы хотите установить истинное значение прав доступа, используйте системный вызов **umask(0)**, чтобы временно отключить влияние **umask**:

```
umask(0);
mknod("FIFO", S_IFIFO|0666, 0);
```

Рассмотрим программу, создающую FIFO-канал и ожидающую данных по этому каналу. Программа после создания канала будет ожидать данных по этому каналу и не завершится до тех пор, пока вы не «убьете» процесс.

Листинг 26.3. Процесс-читатель

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <unistd.h>
#include <linux/stat.h>

/* Наш канал называется FIFO, он будет создан в текущем
каталоге */
#define FIFO   "FIFO"

void main(void)
{
   FILE *fp;
/* Буфер для чтения */
   char buf[128];

   /*  Создаем канал, если он еще не создан, права доступа
0666 */
   umask(0);
   mknod(FIFO, S_IFIFO|0666, 0);
```

```
/* Ожидаем данные */
   while(1)
   {
      fp = fopen(FIFO, "r");
      fgets(buf, 128, fp);
      printf("Получена строка: %s\n", buf);
      fclose(fp);
   }

}
```

Теперь рассмотрим процесс-писатель, который будет записывать данные в FIFO-канал. Этот процесс не завершится до тех пор, пока процесс-читатель не прочитает их:

Листинг 26.4. Процесс-писатель writefifo.c

```
#include <stdio.h>
#include <stdlib.h>

#define FIFO   "FIFO"

void main(int argc, char *argv[])
{
   FILE *fp;

   if ( argc != 2 ) {
      printf("USAGE: writefifo <string>\n");
      exit(1);
      }

   fp = fopen(FIFO, "w");

   fputs(argv[1], fp);

   fclose(fp);

}
```

Запустите процесс-читатель, затем перейдите на другую консоль и запустите «писателя» с аргументом — строкой «info». На первой консоли вы увидите сообщение:

```
Получена строка: info
```

При использовании каналов FIFO нужно учитывать механизм их блокирования. Если процесс открыл канал для записи, то он блокируется до тех пор, пока другой процесс не откроет его для чтения. Аналогично, если какой-то процесс откроет FIFO-канал для чтения, он будет блокирован, пока другой процесс не запишет в канал данные. Если блокировка процесса нежелательна, можно использовать опцию O_NONBLOCK

```
open(fd, O_NONBLOCK);
```

Ясное дело, что тогда нужно немного модифицировать исходный код: вызовы **fclose()**, **fputs()**, **fgets()** использовать уже нельзя, вместо них нужно использовать соответствующие вызовы **close()**, **write()**, **read()**.

И последнее, что нужно помнить при программировании FIFO-каналов: идеология FIFO-каналов предполагает наличие «читателей» и «писателей». Если «писатель» пишет в канал, у которого нет «читателя», из ядра будет послан сигнал SIGPIPE.

26.4. Основные принципы System V IPC

Каждый объект IPC, то есть семафор, очередь сообщений или разделяемый сегмент памяти, имеет свой идентификатор, позволяющий ядру однозначно идентифицировать объект. Идентификатор уникален только для объектов данного типа, а не для всей системы. Например, в системе может быть очередь сообщений с идентификатором «111» и семафор с ID «111», но никогда не будет двух разных очередей с идентификатором «111».

Для получения уникального идентификатора системе нужен ключ (IPC Key), который согласовывается с процессом-сервером и процессом-клиентом. Ключ осуществляет связь между процессом-сервером и процессом-клиентом подобно тому, как маршрутизатор осуществляет связь между двумя компьютерами в разных подсетях.

Ключ генерируется приложением самостоятельно. Для этого используется функция **ftok()**:

```
key_t ftok( char *pathname, char proj );
```

В случае успеха возвращается ключ, а в случае ошибки -1.

При создании ключа нужно учитывать, что сгенерированные ключи могут повторяться, поэтому после получения нового ключа нужно проверить, не используется ли он уже.

Ключ генерируется на основе номера **inode** первого аргумента и символов второго. Это не гарантия уникальности, поэтому и получается, что функция может возвращать уже используемый ключ.

```
key_t   key1;
key_t   key2;

key1 = ftok ("/tmp/app", 'a');
key2 = ftok (".", 'd');
```

Просмотреть статус всех объектов IPC можно с помощью команды ***ipcs***, выводящей состояние всех разделяемых сегментов памяти, семафоров и очередей сообщений.

Для удаления объекта IPC используется команда

```
ipcrm <msg | sem | shm> IPC_ID
```

Так как идентификатор IPC уникален только для объектов IPC определенного типа, вы должны указать тип объекта (**msg** — очередь сообщений, **sem** — семафор, **shm** — сегмент памяти) и его идентификатор.

Для работы с объектами IPC не забудьте подключить следующие заголовочные файлы:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>  /* для очередей */
#include <sys/sem.h>  /* для семафоров */
#include <sys/shm.h>/* для разделяемых сегментов памяти */
```

26.5. Очереди сообщений

26.5.1. Основные структуры ядра для работы с очередями

Очередь сообщений — это связный список, находящийся в адресном пространстве ядра. Каждая очередь имеет свой уникальный идентификатор IPC.

Структура ядра msgbuf (описана в файле `/usr/src/linux/include/linux/msg.h`) является буфером сообщений:

```
struct msgbuf {
    long mtype;          /* тип сообщения */
    char mtext[1];       /* текст сообщения */
};
```

Тут все ясно: тип сообщения и само сообщение. Используя тип сообщения, вы можете помещать в одну очередь разные сообщения, а не создавать еще одну очередь. Например, у нас есть два приложения — клиент и сервер. Вы можете использовать для них одну и ту же очередь: сообщения клиента будут с номером 1 (mtype = 1), а сообщения сервера — с номером 0 (mtype = 0).

Ясное дело, что сообщения из одного символа нас не устраивают, поэтому вы можете переопределить структуру msgbuf в своей программе:

```
struct my_buf {
long mtype;
 char mtext[128];
}
```

Вы также можете добавлять новые поля в эту структуру (но только в своей программе! Код ядра модифицировать не нужно):

```
struct my_buf {
long mtype;
 char mtext[128];
 char info[50];
 int    status;
}
```

Не бойтесь создавать свои структуры: ядру все равно, с какими данными работать, вам нужно учитывать только максимальный размер сообщения, который определен в файле /usr/src/linux/include/linux/msg.h:

```
#define MSGMAX 4056
```

4056 байтов — это максимальный размер не ваших данных, а всей структуры, включая тип сообщения. Размер типа long равен 4 байтам.

Сами сообщения хранятся ядром в структуре msg, которая также определена в файле msg.h:

```
struct msg {
   struct msg *msg_next;   /* указатель на след. сообщение
в очереди */
   long   msg_type;      /* тип сообщения */
   char  *msg_spot;        /* адрес самого сообщения
(текста) */
   short  msg_ts;          /* размер сообщения (текста) */
};
```

Сообщения хранятся в виде односвязного списка. Первый член структуры msg_next — это указатель на следующее сообщение в очереди. Второй член msg_type — это тип сообщения, такой же, как в структуре msg_buf.

Следующий член структуры — это указатель на начало текста сообщения, а последний член msg_ts — размер текста сообщения.

Каждый тип объекта IPC представляется в ядре определенной структурой. Для очередей сообщений это структура msqid_ds (описана в файле /usr/src/linux/include/linux/msg.h).

```
struct msqid_ds {
   struct ipc_perm msg_perm; /* информация о правах
                                доступа */
   struct msg *msg_first;    /* указатель на первое
                                сообщение в очереди */
   struct msg *msg_last;     /* указатель на последнее
                                сообщение в очереди*/
   time_t msg_stime;         /* время последнего вызова
                                msgsnd */
   time_t msg_rtime;         /* время последнего вызова
                                msgrcv */
   time_t msg_ctime;         /* время последнего изменения */
   struct wait_queue *wwait;
   struct wait_queue *rwait;
   ushort msg_cbytes;
   ushort msg_qnum;
   ushort msg_qbytes;        /* максимальное число байтов
                                на очередь */
   ushort msg_lspid;         /* pid последнего
                                испустившего msgsnd */
   ushort msg_lrpid;         /* последний полученный pid */
};
```

- msg_perm

  Это структура типа ipc_perm (ipc_perm определена в файле linux/ipc.h). Данная структура содержит информацию о владельце и правах доступа:

```
struct ipc_perm {
   key_t  key;
   ushort uid;    /* uid и gid владельца */
   ushort gid;
   ushort cuid;   /* uid и gid создателя */
   ushort cgid;
   ushort mode;   /* режим доступа */
   ushort seq;    /* системное поле. Вас оно не касается. */
};
```

- msg_first

  Указатель на первое сообщение в очереди.

- msg_last
  Указатель на последнее сообщение в очереди.

- msg_stime
  Время отправки последнего сообщения из очереди.

- msg_rtime
  Время последнего изъятия сообщения из очереди.

- msg_ctime
  Время последнего изменения очереди.

- wwait и rwait
  Указатели в очередь ожидания ядра, которые используются, когда очередь переполнена и процесс вынужден ждать из-за этого.

- msg_cbytes
  Суммарный объем всех сообщений в очереди.

- msg_qnum
  Количество сообщений в очереди.

- msg_qbytes
  Максимальный размер очереди.

- msg_lspid
  PID процесса, который послал последнее сообщение в очереди.

- msg_lrpid
  PID процесса, который получил сообщение из очереди.

26.5.2. Создание очереди сообщений

Для создания очереди сообщений используется системный вызов **msgget()**. Этот же вызов используется для подключения к уже существующей очереди:

```
int msgget( key_t key, int msgflg );
```

Первый аргумент — это ключ, который мы получаем с помощью системного вызова **ftok()**. Второй аргумент — это режим доступа к очереди:

- IPC_CREAT — создать очередь, если она не была создана ранее.
- IPC_EXCL — если использовать вместе с IPC_CREAT, то в случае, если очередь существует, мы получим ошибку.

Если использовать только IPC_CREAT (без IPC_EXCL), то вызов **msgget()** всегда возвращает идентификатор очереди, даже если очередь уже существует (происходит подключение к очереди). Если использовать IPC_EXCL вместе с IPC_CREAT, также будет создана новая очередь, но если очередь уже существует, подключения не произойдет, а функция **msgget()** возвратит -1 (ошибка).

Вместе с режимом IPC_CREAT можно указывать права доступа к очереди с помощью операции OR:

```
IPC_CREAT | 0660
```

Если произошла ошибка и **msgget()** вернул -1, то переменная errno устанавливается следующим образом:

- EACCESS — у вас нет прав доступа к объекту IPC;
- EEXIST — очередь уже существует, создание невозможно, но возможно подключение к очереди;
- EIDRM — очередь помечена для удаления;
- ENOENT — очередь не существует (в случае подключения);
- ENOMEM — не хватает памяти для создания очереди;
- ENOSPC — не хватает адресного пространства (то есть превышено максимальное количество очередей).

Следующий код создает очередь сообщений:

```
key_t key;   /* ключ IPC */
int id;    /* ID очереди сообщений */
/* создаем ключ */
key = ftok (".", 'd');
/* создаем очередь */
   if ((id = msgget ( key, IPC_CREAT | 0660 )) == -1)
   {
      printf("Ошибка при создании очереди\n");
   }
```

26.5.3. Постановка сообщения в очередь

Для постановки сообщения в очередь используется вызов **msgsnd()**:

```
int msgsnd( int msqid, struct msgbuf *msgp, int msgsz, int msgflg );
```

Первый аргумент — это идентификатор очереди, в которую нужно добавить сообщение. Данный идентификатор мы предварительно получаем с помощью системного вызова msgget(). Второй параметр — это указатель на буфер сообщения. Третий аргумент — это длина сообщения без учета типа сообщения (4 байта). Последний аргумент обычно устанавливают равным 0 или IPC_NOWAIT, если вы не хотите, чтобы процесс был бло-

кирован при постановке сообщения в очередь, в случае переполнения очереди. По умолчанию (когда флаг равен 0), если очередь переполнена, ваш процесс будет блокирован до тех пор, пока сообщение не будет поставлено в очередь.

Как обычно, в случае успеха вызов возвращает 0, а если произошла ошибка, то -1. С помощью errno можно анализировать ошибку:

- EAGAIN — очередь переполнена, а вы используете флаг IPC_NOWAIT, то есть сообщение будет удалено и вам нужно заново поставить его в очередь (отсюда и название ошибки — AGAIN (опять));
- EACCESS — у вас недостаточно прав для записи сообщения в очередь;
- EFAULT — неверный адрес буфера **msgp** (невозможно получить доступ к этому адресу);
- EIDRM — очередь сообщений удалена;
- EINVAL — ошибка в аргументах, например, неправильное значение идентификатора очереди, отрицательный тип сообщения, неправильный размер сообщения и т.д.
- ENOMEM — не хватает памяти.

Следующий фрагмент кода демонстрирует постановку сообщения в очередь:

```
int res, length;     /* результат операции и длина
сообщения */
struct my_buf *buf; /* само сообщение */

/* определяем длину сообщения - 4 байта */
length = sizeof(struct my_buf) - sizeof(long);

   if((res = msgsnd( id, &buf, length, 0)) == -1)
   {
 printf("Ошибка при постановке сообщения в очередь\n");
   }
```

Наверное, вам уже не терпится увидеть реально работающий пример, а не куски кода, которые только отчасти связаны между собой. В листинге 26.5 представлена программа, создающая очередь сообщения и записывающая в нее сообщение.

Листинг 26.5. Пример работы с очередью

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include <linux/ipc.h>
#include <linux/msg.h>

main()
{
   int id;    /* Идентификатор очереди */
   key_t key;      /* Ключ */
   int res, length; /* Результат операции и длина
сообщения */

   struct my_buf {
      long   mtype;      /* тип сообщения */
/* Далее следуют произвольные поля - они зависят от
сообщения */
      int    op_type;    /* тип операции */
      int    l_op;    /* первый операнд */
      int    r_op;    /* второй операнд */
   } msg;

   /* Генерируем IPC-ключ */
   key = ftok(".", 'd');

   /* Создаем очередь или присоединяемся к уже
существующей */
   if ((id = msgget ( key, IPC_CREAT | 0660 )) == -1)
   {
      printf("Ошибка при создании очереди\n");
      exit(1);
   }

    /* Заполняем сообщение */
   msg.type    = 1;  /* тип сообщения, должен быть
положительным! */
   msg.op_type = 0;  /* тип операции */
   msg.l_op = 6;
   msg.r_op = 5;

/* определяем длину сообщения - 4 байта */
length = sizeof(struct my_buf) - sizeof(long);

   if((res = msgsnd( id, &buf, length, 0)) == -1)
   {
 printf("Ошибка при постановке сообщения в очередь\n");
 exit(1);
   }
}
```

После запуска этой программы запустите программу **ipcs** и посмотрите на статус только что отправленного сообщения. Теперь напишем программу, которая получит это сообщение.

26.5.4. Получение сообщений очереди

Для получения сообщения используется системный вызов msgrcv():

```
int msgrcv( int msqid, struct msgbuf *msgp, int msgsz,
long mtype, int msgflg)
```

Первый аргумент определяет очередь, из которой нужно получить сообщение. Второй аргумент — это адрес буфера, в который будет записано сообщение. Третий аргумент — это ограничитель длины сообщения. Четвертый аргумент — это тип сообщения. Ядро будет искать в очереди наиболее старое сообщение данного типа и вернет его копию. Если **mtype**=0, то ядро вернет самое старое сообщение независимо от типа.

После успешного получения сообщения оно удаляется из очереди.

В случае успеха вызов **msqrcv**() возвращает число байтов, скопированных в буфер, или -1 в случае ошибки. Переменная errno устанавливается следующим образом:

- E2BIG — длина сообщения больше, чем ограничитель msgsz;
- EACCESS — у вас недостаточно прав;
- EFAULT — недоступен адрес буфера;
- EIDRM — очередь уничтожена ядром;
- EINTR — операция прервана поступившим сигналом;
- EINVAL — ошибка в аргументах, например, отрицательный размер сообщения или неверный номер очереди;
- ENOMSG — нет сообщения, удовлетворяющего условию. Посылается, если установлен флаг IPC_NOWAIT, в противном случае процесс будет ждать нужного сообщения.

Последний аргумент предоставляет дополнительные возможности по работе с сообщениями. Если установлен бит MSG_NOERROR в **msgflg**, то если размер сообщения больше, чем **msgsz**, оно будет обрезано и вы получите только **msgsz** байтов. Если флаг SMG_NOERROR не устанавливать, вы получите ошибку E2BIG.

Следующий код получает сообщение из очереди:

```
int   id;                /* ID очереди */
int   res, length;       /* результат операции и длина */
struct my_buf buf;       /* буфер */
int type=1;              /* тип сообщения */
```

```
length = sizeof(struct my_buf) - sizeof(long);

if((res = msgrcv( id, &buf, length, type, 0 )) == -1)
 {
      printf("Ошибка!");

/* можно проанализировать ошибку */
      if(errno==E2BIG) printf("Сообщение слишком большое\n");
      if(errno==EACCESS) printf("Нет доступа\n");

 /* и т.д. */

      exit(1);
 }
```

26.5.5. Проверка наличия сообщения в очереди

Наверное, вы не хотите, чтобы ваша программа ждала, пока в очереди появится нужное сообщение. Используя особенности системного вызова **msgrcv**(), можно написать код проверки наличия сообщения определенного типа в очереди. Напишем функцию **msg_exists**(), которая будет возвращать TRUE, если сообщение есть в очереди, или FALSE, если сообщения в очереди нет.

```
int msg_exists(int id, long type )
{
   int   res;
   if((result = msgrcv( id, NULL, 0, type, IPC_NOWAIT )) == -1)
   {       if(errno == E2BIG)
           return(TRUE);
   }
   return(FALSE);
}
```

В вызове msgrcv() отсутствует адрес буфера и длина сообщения. Этим мы специально провоцируем ошибку, а вызов IPC_NOWAIT отказывает от блокировки процесса. Мы проверяем errno; если он равен E2BIG, значит, сообщение есть в очереди. Ошибка E2BIG порождается потому, что мы установили размер сообщения равным 0.

26.5.6. Тотальный контроль

До сих пор мы рассматривали только системные вызовы для работы с сообщениями очереди, сейчас рассмотрим системный вызов **msgctl()**, предназначенный для контроля самой очереди.

```
int msgctl ( int msgqid, int cmd, struct msqid_ds *buf );
```

Первый аргумент — это ID очереди, второй — команда, которую нужно выполнить:

- IPC_STAT — записывает в буфер buf структуру msqid_ds для очереди сообщений с идентификатором msgqid.
- IPC_SET — устанавливает значение ipc_perm структуры msqid. Значение берется из буфера buf.
- IPC_RMID — удаляет очередь.

Системный вызов возвращает 0 в случае успеха и -1, если произошла ошибка. Переменная errno устанавливается следующим образом:

- EACCESS — недостаточно прав.
- EFAULT — невозможно получить доступ к адресу буфера buf или неверный адрес.
- EIDRM — очередь была уничтожена прямо во время запроса.
- EINVAL — ошибка в аргументах, например, неправильный ID-очереди или отрицательный размер сообщения.
- EPERM — у вас нет прав на запись в очередь.

Структура **msqid_ds** уже рассматривалась ранее, поэтому не вижу смысла приводить ее описание еще раз.

Если подытожить, то все, что мы можем сделать с очередью — это удалить ее и изменить права доступа или информацию о владельце (его UID и GID). Как удалить, думаю, понятно. Напишем функцию **change_mode()**, которая будет изменять права доступа к нашей очереди. Ей нужно передать два параметра — идентификатор очереди и новый режим доступа в виде строки, например, "0660".

```
int change_mode( int id, char *mode )
{
   struct msqid_ds buf;

   /* Получаем копию структуры в буфер buf */
   if( msgctl( id, IPC_STAT, &buf) == -1 )
   {
      return(-1);
   }
```

```
    /* Изменяем права доступа */
    sscanf(mode, "%ho", &buf.msg_perm.mode);

   /* Модернизируем внутреннюю структуру */
    if( msgctl( id, IPC_SET, &buf ) == -1 )
    {
       return(-1);
    }

    return(0);
}
```

Наша функция возвращает 0 в случае успеха или -1, если произошла ошибка.

На этом обзор средств для работы с очередями сообщений можно считать законченным, теперь с чистой совестью перейдем к следующему средству IPC — семафорам.

26.6. Семафоры

Семафор — это объект IPC, управляющий доступом к общим ресурсам (устройствам). Семафоры не позволяют одному процессу захватить устройство до тех пор, пока с этим устройством работает другой процесс. Семафор может находиться в двух положениях: 0 (устройство занято) и 1 (устройство свободно).

Одиночный семафор используется редко, практически никогда. Для контроля доступа к ресурсам обычно используются множества семафоров, даже если это множество состоит всего из одного семафора. Например, пусть у нас есть три принтера. Когда вы посылаете задание на печать, диспетчер печати просматривает множество семафоров принтеров и выясняет, есть ли свободный принтер. Если да, то он начинает печатать ваше задание, если же нет, диспетчер ставит ваше задание в очередь печати.

Еще один пример использования семафоров — это счетчики ресурсов. Представим, что вместо принтера есть некий контроллер, позволяющий выполнять 100 заданий одновременно. Когда он свободен, значение семафора равно 100. По мере поступления заданий диспетчер контроллера уменьшает значение семафора на 1, а по мере их выполнения увеличивает на 1. Когда значение достигает 0, новое задание ставится в очередь до освобождения контроллера.

Как и в случае с очередями сообщений, для семафоров в ядре Linux есть своя структура — semid_ds, которая описана в файле /usr/src/linux/include/linux/sem.h:

```
struct semid_ds {
   struct ipc_perm sem_perm;          /* права доступа */
   time_t          sem_otime;        /* время последней
операции */
   time_t          sem_ctime;        /* время последнего
изменения */
   struct sem      *sem_base;        /* указатель на первый
семафор */
   struct wait_queue *eventn;        /* очереди ожидания */
   struct wait_queue *eventz;
   struct sem_undo *undo;            /* запросы undo в этом
массиве */
   ushort          sem_nsems;        /* номера семафоров в
массиве */
};
```

Обратите внимание: в структуре есть указатель на первый семафор. Тип указателя — sem. Данный тип описывает семафор:

```
struct sem {
   short   sempid;    /* pid последней операции */
   ushort  semval;    /* текущее значение семафора */
   ushort  semncnt;    /* число процессов, ожидающих
освобожд. рес */
   ushort  semzcnt;   /* число процессов, ожидающих освоб.
всех рес.  */
};
```

- sem_pid
  PID процесса, который произвел последнюю операцию над семафором.

- sem_semval
  Текущее значение семафора.

- sem_semncnt
  Число процессов, ожидающих увеличения значения семафора, то есть освобождения ресурсов.

- sem_semzcnt
  Число процессов, ожидающих освобождения всех ресурсов.

26.6.1. Создание множества семафоров

Для создания множества семафоров или подключения к уже существующему множеству используется системный вызов **semget**():

```
int semget ( key_t key, int nsems, int semflg );
```

Первый аргумент — это ключ IPC, который, как обычно, создается системным вызовом **ftok**(). Он сравнивается с ключами других семафоров и в зависимости от значения **semflg** решается, создавать новое множество или подключиться к уже существующему. Значение **semflg**:

- IPC_CREAT — создать новое множество семафоров;
- IPC_EXCL — при использовании с IPC_CREAT порождает ошибку, если семафор уже существует.

При создании семафора, как и при создании очереди сообщений, мы можем указать права доступа:

```
IPC_CREAT | 0660
```

Второй аргумент системного вызова semget() задает требуемое количество семафоров. Оно ограничено в файле sem.h:

```
#define SEMMSL  32 /* <= 512 */
```

Данный аргумент игнорируется, если вы подключаетесь к уже существующему множеству, а не создаете новое.

Функция **semget**() возвращает идентификатор семафора или -1 в случае ошибки. Переменная `errno` устанавливается следующим образом:

- EACCESS — у вас не хватает полномочий для выполнения операции;
- EEXISTS — множество существует, его нельзя создать;
- EIDRM — множество помечено для удаления;
- ENOENT — множество не существует, не было ни одной операции IPC_CREAT;
- ENOMEM — не хватает памяти;
- ENOSPC — достигнуто максимальное количество семафоров.

Функция для открытия существующего семафора может выглядеть так:

```
void open_sem(int *sid, key_t key)
{

   if((*sid = semget(key, 0, 0666)) == -1)
   {
      printf("Семафор не существует !!!!\n");
      exit(1);
```

```
    }

}
```

Для создания множества семафоров можно использовать следующую функцию:

```
void create_sem(int *sid, key_t key, int n)
{
   int c=0;     /* счетчик */
   union semun sems;

   if(n > SEMMSL) {
      printf("Превышен лимит. Максимальное число семафоров
%d\n",SEMMSL);
      exit(1);
   }

   if((*sid = semget(key, n, IPC_CREAT|IPC_EXCL|0666)) == -1)
   {
      printf("Множество уже существует\n");
      exit(1);
   }

   sems.val = SEM_RESOURCE_MAX;

   /* Инициализируем все элементы одним значением */
   semctl(*sid,c,SETALL,sems);

   /* Если нужно установить разные значения, нужно
использовать SETVAL,
   например
   for(c=0; c<n; c++)
      semctl(*sid, c, SETVAL, sems);
   */
}
```

26.6.2. Выполнение операций над семафорами

Для выполнения операций над множеством семафоров служит системный вызов **semop**():

```
int semop( int semid, struct sembuf *sops, unsigned nsops);
```

Первый аргумент — это идентификатор семафора, возвращаемый вызовом **semget**(). Второй — это массив операций, которые нужно выполнить над семафорами. Последний аргумент — это количество операций в массиве.

Второй аргумент представляет собой массив типа sembuf:

```
struct sembuf {
   ushort sem_num;   /* номер семафора в массиве */
   short  sem_op;    /* операция над семафором */
   short  sem_flg;   /* флаги */
};
```

- sem_num
  Номер семафора, над которым нужно выполнить операцию

- sem_op
  Выполняемая операция. Может быть отрицательным или положительным числом. Если число отрицательно, значение семафора будет уменьшено, а если положительным — увеличено. Не забывайте, освобождая ресурс, увеличивать значение семафора — за вас никто это не сделает. Если sem_op = 0, то процесс «заснет» и не «проснется» до тех пор, пока значение семафора не станет 0.

- sem_flg

Флаги операции, например, IPC_NOWAIT. Если IPC_NOWAIT не установлен, то процесс «заснет» до тех пор, пока не освободится указанное количество ресурсов (пока другой процесс не освободит их).

Чтобы лучше понять, что такое semop(), вернемся к нашим принтерам. Пусть у нас есть всего один принтер, умеющий выполнять только одно задание за раз. Начальное значение семафора принтера будет равно 1.

Перед посылкой задания на принтер нужно убедиться, что он свободен, то есть получить от семафора значение 1. Заполним массив sembuf необходимой для выполнения операции информацией:

struct sembuf prn_lock = { 0, -1, IPC_NOWAIT };

Здесь 0 — это номер семафора: у нас всего один принтер, а нумерация начинается с нуля. -1 — это операция, запрашивающая единицу ресурса. Если принтер свободен, то после выполнения этой операции значение семафора принтера будет равно 0.

Мы также установили флаг IPC_NOWAIT, чтобы вызов прошел немедленно. Если принтер занят, вызов вернет ошибку:

```
if(semop(sid, &prn_lock, 1) == -1) printf("Принтер занят\n");
```

Первый аргумент — это идентификатор объекта IPC, второй — это массив операций. Последний аргумент semop() говорит о том, что у нас есть только одна структура типа sembuf, то есть нам нужно выполнить только одну операцию.

После выполнения задания мы должны сообщить семафору об освобождении ресурса:

```
struct sembuf prn_unlock = { 0, 1, IPC_NOWAIT };
semop(sid, &prn_unlock, 1);
```

В случае успеха, когда выполнены все операции, системный вызов semop() возвращает 0. В случае ошибки возвращается -1, а errno равна:

- E2BIG — количество операций (аргумент nsops) превышает разрешенное число операций;
- EACCESS — не хватает полномочий;
- EAGAIN — операция не может быть выполнена (при использовании флага IPC_NOWAIT), такую операцию нужно повторить снова;
- EFAULT — указатель sops указывает на ошибочный адрес;
- EIDRM — множество помечено на удаление;
- EINTR — прервано сигналом;
- EINVAL — неверный semid;
- ENOMEM — не хватает памяти для создания структуры Undo-операции;
- ERANGE — значение семафора вышло за пределы допустимых значений.

26.6.3. Контроль семафора

Для контроля семафора используется системный вызов **semctl**():

```
int semctl ( int semid, int semnum, int cmd, union semun arg );
```

Первый аргумент — это идентификатор семафора, второй — номер семафора во множестве семафоров (нумерация начиняется с 0). В отличие от очереди сообщений, где достаточно было указать только идентификатор очереди, при работе с семафорами нужно обязательно указывать номер конкретного семафора, над которым вы хотите выполнить операцию. Третий аргумент — это команда, которую нужно выполнить над семафором. Возможные команды представлены в таблице 26.1.

Команды управления семафорами *Таблица 26.1*

Команда	Назначение
IPC_STAT	Запоминает структуру semid_ds для множества по адресу buf объединения semun (чуть позже мы подробно рассмотрим эту структуру и объединение semun)
IPC_SET	Устанавливает значение члена ipc_perm структуры semid_ds
IPC_RMID	Удаляет множество
GETALL	Позволяет получить значения всех семафоров. Значения возвращаются в виде массива unsigned short, на который указывает член объединения array
GETNCNT	Возвращает число процессов, которые ожидают ресурсы в данный момент
GETPID	Возвращает PID процесса, выполнившего последний вызов semop()
GETVAL	Возвращает значение одного семафора
GETZCNT	Возвращает число процессов, которые ожидают полного освобождения ресурса
SETALL	Устанавливает значение семафоров. Значения берутся из члена array объединения semun
SETVAL	Устанавливает значение конкретного семафора. Значение берется из элемента val объединения semun

Последний аргумент — это объединение (union) аргументов, которые можно использовать для управления семафором. Рассмотрим подробнее это объединение, объявленное в файле `/usr/src/linux/include/linux/sem.h`:

```
union semun {
   int val;                  /* значение для SETVAL */
   struct semid_ds *buf;    /* буфер для IPC_STAT и IPC_SET */
   ushort *array;           /* массив для GETALL и SETALL */
   struct seminfo *__buf;   /* буфер для IPC_INFO */
   void *__pad;
};
```

Первый член этого объединения **val** используется для установки значения одного семафора при использовании команды SETVAL.

Член **buf** используется командами IPC_STAT и IPC_SET. Это копия внутренней структуры данных семафора.

Указатель на массив **array** используется командами GETALL и SETALL для получения или установки значений всех семафоров во множестве.

Последние два члена объединения специфичны только для Linux — в других UNIX-системах вы их не найдете. Эти элементы использует ядро.

В случае успеха системный вызов **semctl**() возвращает натуральное число, а в случае ошибки -1. Переменная errno равна:

- EACCESS — не хватает полномочий;
- EFAULT — адрес arg ошибочен;
- EIDRM — множество помечено для удаления;

- EINVAL — неправильный аргумент semid;
- EPERM — у вас нет прав для выполнения команды cmd;
- ERANGE — значение семафора вышло за пределы допустимых значений.

Пример получения значения семафора с номером N из множества sid:

```
int val;
val=semctl(sid, N, GETVAL, 0);
```

Предположим, что нам нужно вывести состояние всех трех имеющихся принтеров:

```
int c, val;
for (c=0; x<3; c++)
{
  val=semctl(sid,c,GETVAL,0);
  printf("Принтер %d: %d\n",c,val);
}
```

А вот код инициализации всех семафоров множества semid:

```
union semun opts;
int c;

opts.val = 1;     /* первоначальное значение семафоров */

for (c=0;c<5;c++) semctl(semid, c,SETVAL,opts);
```

Довольно часто возникают определенные сложности с установкой прав доступа к множеству семафоров. Рассмотрим следующий код, позволяющий установить права доступа к множеству **semid**. Права доступа задаются в виде строки, например, «0660».

```
void sem_change_mode(int semid, char *mode)
{
   int res;
   struct semid_ds semds;
      union semun opts;

   /* Нужно указать нашу локальную копию структуры */
   opts.buf = &semds;

   if((res = semctl(semid, 0, IPC_STAT, opts)) == -1)
   {
      printf("Error");
      exit(1);
   }
```

```
    printf("Старые права доступа %o\n", opts.buf->sem_perm.mode);

    /* Изменяем права доступа */
    sscanf(mode, "%ho", &opts.buf->sem_perm.mode);

    /* Обновляем внутреннюю структуру */
    semctl(sid, 0, IPC_SET, opts);

}
```

26.7. Разделяемые сегменты памяти

Сегменты памяти разделяются между несколькими процессами. Один процесс создает сегмент памяти, а другие в любом количестве — используют. Разделяемые сегменты памяти — это самый быстрый способ IPC.

Для каждого разделяемого сегмента памяти ядро поддерживает специальную структуру — **shmid_ds**, описанную в файле `/usr/src/linux/include/linux/shm.h`:

```
    struct shmid_ds {
       struct ipc_perm shm_perm;          /* права доступа */
       int    shm_segsz;     /* размеры сегмента в байтах */
       time_t shm_atime;     /* время последней привязки */
       time_t shm_dtime;     /* время последней отвязки */
       time_t shm_ctime;     /* время последнего изменения */
       unsigned short shm_cpid;      /* PID создателя */
  /* PID последнего процесса-пользователя сегмента */
       unsigned short shm_lpid;
       short  shm_nattch;            /* число привязок */
       unsigned short shm_npages;
                     /* размеры сегмента (в страницах) */
        /* массив указателей на $frames -> S$*/
       unsigned long *shm_pages;
          struct vm_area_struct *attaches;
                     /* дескрипторы для привязок */
};
```

Я немного сократил эту структуру, оставив описание только нужных нам полей. Полагаю, что вы в нем разберетесь. Возможно, вас заинтересовали термины «привязка» и «отвязка». Привязка — это размещение сегмента в адресном пространстве процесса, подключение к разделяемому сегменту памяти (РСП). Отвязка, соответственно, — отключение. Поле shm_nattch содержит количество привязок к РСП на данный момент.

Для создания нового РСП используется системный вызов **shmget()**. Этот же вызов используется для подключения к уже существующему РСП.

```
int shmget ( key_t key, int size, int shmflg );
```

Первый аргумент — это ключ IPC, полученный с помощью **ftok()**, второй — размер РСП в байтах, а третий — флаги системного вызова **shmget**. Если установлен флаг IPC_CREAT, системный вызов создаст новый РСП или подключится к уже существующему сегменту, если обнаружится, что уже есть такой сегмент (с таким же значением ключа). Если установлен флаг IPC_EXCL вместе с IPC_CREAT (сам по себе он бесполезен) подключение к существующему РСП запрещается.

Системный вызов **shmget()** возвращает идентификатор РСП или -1, если произошла ошибка. Переменная errno устанавливается так:

- EACCESS — не хватает полномочий для доступа к сегменту;
- EINVAL — неправильно заданы размеры сегмента;
- EEXISTS — сегмент уже существует, создание невозможно. Вы получите эту ошибку, если будете использовать флаг IPC_EXCL вместе с IPC_CREAT при условии, что сегмент уже существует;
- IDRM — сегмент помечен на удаление или уже удален;
- ENOMEM — не хватает памяти для создания сегмента.

Приведем пример функции открытия/создания РСП:

```
int open_shms( key_t key, int size )
{
   return (shmget( key, size, IPC_CREAT | 0660 )) == -1));
}
```

После получения идентификатора РСП мы должны «привязаться» к этому сегменту, то есть разместить сегмент в своем адресном пространстве. Для этого используется системный вызов **shmat()** (*shared memory attachment*):

```
int shmat ( int shmid, char *shmaddr, int shmflg );
```

Первый аргумент — это идентификатор РСП, который мы получаем с помощью предыдущего вызова. Второй аргумент — это адрес привязки. Если указать вместо адреса ноль, то ядро само найдет нераспределенную область.

Третий аргумент — это флаги. Обычно используется два флага:

- SHM_RND — переданный адрес будет округлен до ближайшей страницы (если вы сами указываете адрес);
- SHM_RDONLY — РСП будет доступен только для чтения.

В случае успеха **shmat()** возвращает адрес, по которому сегмент был привязан к процессу, или -1, если произошла ошибка. Переменная errno может принимать всего три значения:

- EACCESS — нет доступа;
- ENOMEM — не хватает памяти;
- EINVAL — ошибка в параметрах, то есть неправильное значение ID или адреса привязки (shmaddr).

Пример привязки:

```
char *ptr;
prt = shmat(sh_id,0,0);
```

После привязки сегмента к адресному пространству доступны операции чтения и записи, которые очень напоминают работу с простыми указателями.

Для снятия привязки используется системный вызов shmdt():

```
int shmdt ( char *shmaddr );
```

В случае ошибки данный системный вызов возвращает -1. Значение errno только одно: EINVAL, то есть вы неправильно указали адрес привязки. После отвязки значение элемента shm_nattch структуры shmid_ds уменьшается на 1. Если больше нет привязок, то есть shm_nattch = 0, сегмент будет удален ядром.

Для управления РСП используется системный вызов shmctl():

```
int shmctl ( int shmid, int cmd, struct shmid_ds *buf );
```

Первый аргумент — это идентификатор РСП, второй — команда, а третий — буфер для команд IPC_STAT/IPC_SET. Команд для управления три:

- IPC_STAT — сохраняет структуру shmid_ds по адресу buf;
- IPC_SET — берет значение элемента ipc_perm структуры shmid_ds и устанавливает его для сегмента. Значение берется из buf;
- IPC_RMID — помечает сегмент для удаления, само удаление произойдет, как только последний процесс отвяжется от сегмента. Если сегмент помечен на удаление, ни один процесс не сможет привязаться к сегменту.

В случае успеха системный вызов **shmctl()** возвращает 0 или -1,если произошла ошибка. Переменная errno устанавливается так:

- EACCESS — нет прав;
- EFAULT — ошибочный адрес buf;
- EIDRM — сегмент помечен на удаление;
- EINVAL — неправильный идентификатор сегмента.

Вот теперь мы готовы к написанию демонстрационной программы для работы с разделяемыми сегментами памяти.

Листинг 26.6. Демонстрационная программа shm_demo.c

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

/* размер нашего сегмента – 256 байтов */
#define SIZE 256

int main(int argc, char *argv[])
{
   key_t key;          /* ключ */
   int shmid, c;       /* идентификатор */
   char *ptr;          /* указатель, через который мы будем
                       работать с сегментом */

/* Если аргументы не указаны … */
   if(argc == 1)
    {
    printf("shm_demo usage:\n");
    printf("shm_demo -w string  записать строку в сегмент\n");
    printf("shm_demo -r  прочитать инф. из сегмента\n");
    printf("shm_demo -d       удалить сегмент\n");
    printf("shm_demo -m mode        изменить права доступа\
n");
    exit(1);
    }

/* Генерируем ключ IPC */
   key = ftok(".", 'D');

   /* Пытаемся создать сегмент*/
   if((shmid = shmget(key, SIZE, IPC_CREAT|IPC_EXCL|0660)) == -1)
   {
      printf("Сегмент существует, подключаемся к нему…\n");

/* Используем shmget без IPC_EXCL */
      if((shmid = shmget(key, SIZE, 0)) == -1)
      {
         printf("Ошибка в shmget\n");
```

```
        exit(1);
      }
   }
   else
   {
      printf("Создаем новый сегмент\n");
   }

   /* Привязываемся к сегменту */
   if((ptr = shmat(shmid, 0, 0)) == -1)
   {
      perror("shmat");
      exit(1);
   }

/* Разбираем параметры командной строки:
 w - запись в сегмент
 r - чтение
 d - удаление сегмента
 m - изменение прав доступа
*/
   switch(tolower(argv[1][1]))
   {
      case 'w': shm_write(shmid, ptr, argv[2]);
                break;
      case 'r': shm_read(shmid, ptr);
                break;
      case 'd': shm_rm(shmid);
                break;
      case 'm': shm_change_mode(shmid, argv[2]);
                break;
   }
}
/* Функция для записи в сегмент: ей нужно передать ID
сегмента, адрес привязки и записываемую информацию */
shm_write(int shmid, char *ptr, char *info)
{
   strcpy(ptr, info);
}

/* Функция чтения информации из сегмента */
shm_read(int shmid, char *ptr)
{
   printf("Информация из сегмента: %s\n", ptr);
}
```

```
/* Функция удаления сегмента */
shm_rm(int shmid)
{
   shmctl(shmid, IPC_RMID, 0);
   printf("Сегмент помечен на удаление\n");
}
/* Функция изменения прав доступа. Ей нужно передать
идентификатор сегмента и права доступа в виде строки,
например, "0666" */
shm_change_mode(int shmid, char *mode)
{
   struct shmid_ds mds;

   shmctl(shmid, IPC_STAT, &mds);

   printf("Старые права доступа: %o\n", mds.shm_perm.mode);

   sscanf(mode, "%o", &mds.shm_perm.mode);

   shmctl(shmid, IPC_SET, &mds);

   printf("Новые права доступа: %o\n", mds.shm_perm.mode);
}
```

Использовать программу нужно так:

./shm_demo –w строка
: запись строки в сегмент

./shm_demo –r
: чтение строки из сегмента

./shm_demo –m права
: изменение прав доступа

./shm_demo –d
: удаление сегмента

Выполните команду

```
$ ./shm_demo -w string
```

А затем запустите утилиту ***ipcs***. Вы увидите, что наша программа создала разделяемый сегмент памяти:

```
------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x44063781 0          root       660        256        0
```

```
------ Semaphore Arrays --------
key        semid      owner      perms      nsems      status

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
```

Затем выполните команду:

```
$ ./shm_demo -r
```

Вы получите информацию:

```
Информация из сегмента: string
```

Попробуем изменить права доступа, а затем просмотреть информацию командой ***ipcs***:

```
$./shm_demo -m 0666

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x44063781 0          root       666        256        0


------ Semaphore Arrays --------
key        semid      owner      perms      nsems      status

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
```

Глава 27

СОЗДАНИЕ СЕТЕВОГО ПРИЛОЖЕНИЯ В LINUX

- ПРОТОКОЛ TCP/IP
- ПРОТОКОЛ ICMP
- ПРОГРАММИРОВАНИЕ СОКЕТОВ

В главе 6 я ввел основные сетевые понятия, перечислил самые популярные протоколы и вкратце рассказал об основе сети Интернет — протоколе TCP/IP. В этой главе я собираюсь подробнее остановиться на протоколах TCP/IP и ICMP и показать, какие средства вы можете использовать для написания собственных приложений для работы в сети.

27.1. Протокол TCP/IP

27.1.1. Многоуровневая архитектура стека TCP/IP

Протокол TCP/IP был создан в конце 60-х — начале 70-х годов агентством DARPA Министерства Обороны США (U.S. Department of Defense Advanced Research Projects Agency). Основные этапы развития этого протокола отмечены в таблице 27.1.

Этапы развития протокола TCP/IP *Таблица 27.1*

Год	Событие
1970 г.	Введен в использование протокол NCP (Network Control Protocol) для узлов сети Arpanet
1972 г.	Вышла первая спецификация Telnet (см. RFC 318)
1973 г.	Введен протокол FTP (RFC 454)
1974 г.	Разработана программа TCP (Transmission Control Program)
1981 г.	Опубликован стандарт протокола IP (RFC 791)
1982 г.	Объединение протоколов TCP и IP в одно целое — TCP/IP
1983 г.	Сеть Arpanet переведена на протокол TCP (ранее использовался протокол NCP)
1984 г.	Введена доменная система имен DNS

Как видите, все стандарты интернет-протоколов опубликованы в документах RFC. Документы RFC (*Request for Comments*) — это документы, в которых описывается устройство сети Интернет. Они создаются сообществом Интернет (Internet Society, ISOC). Любой член ISOC может опубликовать свой стандарт в документе RFC. Документы RFC делятся на пять типов:

1. Требуется (Required) — данный стандарт должен быть реализован на всех основных узлах TCP/IP.
2. Рекомендуется (Recommended) — обычно такие спецификации RFC также реализуются.
3. Выборочно (Elective) — реализация не обязательна.
4. Ограниченное использование (Limited use) — не рекомендуется для всеобщего применения.
5. Не рекомендуется (Not recommended) — не рекомендуются.

Протоколы семейства TCP/IP можно представить в виде модели, состоящей из четырех уровней: прикладного, основного, межсетевого и сетевого (таблица 27.2).

Уровни стека протоколов TCP/IP *Таблица 27.2*

Уровень 1	Прикладной уровень (уровень приложения, Application Layer)
Уровень 2	Основной (транспортный) уровень (Transport Layer)
Уровень 3	Межсетевой уровень (уровень Internet, Internet Layer)
Уровень 4	Уровень сетевых интерфейсов (Network Interface Layer)

Каждый из этих уровней выполняет определенную задачу для организации надежной и производительной работы сети.

27.1.1.1. Уровень сетевого интерфейса

Этот уровень лежит в основании всей модели протоколов семейства TCP/IP. Уровень сетевого интерфейса отвечает за отправку в сеть и прием из сети кадров, которые содержат информацию. *Кадр* (frame) — это единица данных, которыми обмениваются компьютеры в сети Ethernet. Для обозначения блоков данных определенных уровней используются термины кадр (frame), пакет (packet), дейтаграмма (datagram), сегмент (segment). Все эти термины обозначают транспортируемые отдельно блоки данных, и их можно считать синонимами. Название блока пересылаемых данных изменяется в зависимости от уровня (рис. 27.1).

27.1.1.2. Межсетевой уровень

Протоколы Интернет инкапсулируют блоки данных в пакеты (дейтаграммы) и обеспечивают необходимую маршрутизацию. К основным интернет-протоколам относятся:

- IP (*Internet Protocol*) — предназначен для отправки и маршрутизации пакетов.
- ARP (*Address Resolution Protocol*) — используется для получения MAC-адресов (аппаратных адресов) сетевых адаптеров.
- ICMP (*Internet Control Message Protocol*) — предназначен для отправки извещений и сообщений об ошибках при передаче пакетов.

Рис. 27.1. *Пересылка блока данных в стеке протоколов TCP/IP*

- IGMP (*Internet Group Management Protocol*) — используется узлами для сообщения маршрутизаторам, которые поддерживают групповую передачу, о своем участии в группах.
- RIP (*Route Internet Protocol*) и OSPF (*Open Shortest Path First*) — протоколы маршрутизации.

На этом уровне реализуется передача пакетов без установки соединения — дейтаграммным способом. Межсетевой уровень обеспечивает перемещение пакетов по сети с использованием наиболее рационального маршрута (протокол OSPF). Основная функция межсетевого уровня — передача пакетов через составную сеть, поэтому этот уровень также называется уровнем Интернет.

27.1.1.3. Транспортный (основной) уровень

Этот уровень обеспечивает сеансы связи между компьютерами. Существует два транспортных протокола: TCP (*Transmission Control Protocol*) и UDP (*User Datagram Protocol*). Протокол TCP ориентирован на установление соединения, то есть перед передачей данных компьютеры «договариваются» между собой. Обычно по этому протоколу передаются большие объемы данных или данные, для которых требуется подтверждение их

приема. Этот протокол используется большинством сетевых приложений, так как обеспечивает достаточную надежность при передаче данных.

Протокол UDP не ориентирован на соединение и не гарантирует доставку пакетов (дейтаграмм), зато он работает быстрее TCP. Обычно по этому протоколу передаются небольшие объемы данных. Ответственность за доставку данных несет сетевая программа.

27.1.1.4. Уровень приложений

Этот уровень является вершиной модели TCP/IP. На этом уровне работают практически все распространенные утилиты и службы: DNS, Telnet, WWW, Gopher, WAIS, SNMP, FTP, TFTP, SMTP, POP, IMAP.

Таблица 27.3 показывает соответствие уровней стека протокола TCP/IP семиуровневой модели OSI.

Соответствие уровней стека TCP/IP модели OSI *Таблица 27.3*

Уровень модели OSI	Протокол	Уровень стека TCP/IP
7,6	WWW (HTTP), FTP, TFTP, SMTP, POP, telnet, WAIS, SNMP	1
5,4	TCP, UDP	2
3	IP, ICMP, RIP, OSPF, ARP	3
2,1	Ethernet, PPP, SLIP	4

27.1.2. Структура пакетов IP и TCP

Протокол IP не ориентирован на соединение, поэтому не обеспечивает надежную доставку данных. Поля, описание которых приведено в таблице 27.4, представляют собой IP-заголовок и добавляются к пакету при его получении с транспортного уровня.

Структура заголовка IP-пакета *Таблица 27.4*

Поле	Описание
Source IP-address	IP-адрес отправителя пакета
Destination IP-address	IP-адрес получателя пакета
Protocol	Протокол: TCP или UDP
Checksum	Контрольная сумма для проверки целостности пакета
TTL (Time to Live)	Время жизни пакета: определяет, сколько секунд дейтаграмма может находиться в сети. Предотвращает бесконечное блуждание пакетов в сети. Значение TTL автоматически уменьшается на одну или более секунд при проходе через каждый маршрутизатор сети
Version	Версия протокола IP — 4 или 6 (4 бита)
Header Length	Длина заголовка пакета (4 бита). Минимальный размер заголовка — 20 байтов

Продолжение табл. 27.4

Поле	Описание
Type of Service ()	Тип обслуживания: обозначение требуемого для этого пакета качества обслуживания при доставке через маршрутизаторы IP-сети. Здесь определяются приоритет, задержки, пропускная способность (8 битов)
Total Length	Длина дейтаграммы IP-протокола (16 битов)
Identification	Идентификатор пакета. Если пакет фрагментирован (разбит на части), то все фрагменты имеют одинаковый идентификатор (16 битов)
Fragmentation Flags	3 бита для флагов фрагментации и 2 бита для текущего использования
Fragmentation Offset	Смещение фрагмента: указывает на положение фрагментов относительно начала поля данных IP-пакета. Если фрагментации нет, смещение равно 0x0 (13 битов)
Options and Padding	Опции

Протокол TCP в отличие от протокола IP ориентирован на установление соединения и обеспечивает надежную доставку данных. Структура TCP-пакета описана в таблице 27.5.

Структура заголовка TCP-пакета *Таблица 27.5*

Поле	Описание
Source port	Порт TCP узла-отправителя
Destination Port	Порт TCP узла-получателя
Sequence Number	Номер последовательности пакетов
Acknowledgement Number	Номер подтверждения: порядковый номер байта, который локальный узел рассчитывает получить следующим
Data Length	Длина TCP-пакета
Reserved	Зарезервировано для будущего использования
Flags	Флаги: описание содержимого сегмента
Window	Показывает доступное место в окне протокола TCP
Checksum	Контрольная сумма для проверки целостности пакета
Urgent Pointer ()	Указатель срочности: при отправке срочных данных (поле Flags) в этом поле задается граница области срочных данных

27.2. Протокол ICMP

27.2.1. Для чего используется протокол ICMP

Протокол межсетевых управляющих сообщений используется для диагностических целей. Например, система А передала системе Б неверный пакет. Система Б с помощью протокола ICMP может «сказать» системе А, что посланный ею пакет некорректен. Также по этому протоколу производится диагностика сети. Многие из вас используют утилиту ***ping***,

чтобы определить, доходят ли отправленные вами пакеты до какой-то определенной машины, даже не подозревая о том, что ***ping*** использует как раз протокол ICMP.

С помощью протокола ICMP можно определить не только тип неисправности, но и максимально уточнить ее причину. Например, тип сообщения 3 («Адресат недостижим») свидетельствует о том, что посланный нам пакет не может «добраться» до адресата. Кроме типа сообщения, возвращается его код, позволяющий детализировать неисправность. Например, код 0 означает, что сеть недоступна, 1 — узел недоступен, 2 — протокол недоступен и т.д. В таблице 27.8 ниже перечислены типы и коды сообщений ICMP.

Обязательно ли посылать сообщение об ошибке, получив некорректный пакет? Нет, из главы 19 вы узнали, что такие пакеты (например, отправленные на запрещенный для доступа извне порт) можно просто игнорировать. Первый подход «вежливее», второй безопаснее. Представьте, что злоумышленник отправляет с 1000 машин 1000 пакетов с указанием неверного порта на вашу систему. Тогда ваша система только тем и будет заниматься, что отправлять сообщения об ошибках. Это называется атакой на отказ. Конечно, современные брандмауэры позволяют избавиться от этого неприятного явления, но не у всех они установлены и настроены должным образом.

27.2.2. Структура ICMP-пакета

Для лучшей диагностики ошибки вместе с ICMP-пакетом передается заголовок исходного пакета, вызвавшего ошибку. Также передается специальный указатель на позицию заголовка, позволяющий определить, что именно вызвало ошибку. В случае с неправильным номером порта указатель будет установлен на поле, содержащее номер порта. Благодаря этому система-источник может вычислить процесс и сокет, вызвавшие ошибку.

Структуры различных ICMP-пакетов отличаются друг от друга в зависимости от типа пакета. Например, пакет, сообщающий о недоступности адресата (Destination Unreachable Message), выглядит так:

Сообщение Destination Unreachable Message — *Таблица 27.6*

Тип	Код	Контрольная сумма
Не используется		
Интернет-заголовок плюс первые 64 бита оригинального сообщения (пакета)		

Что такое тип и код, вы уже знаете, с контрольной суммой тоже все ясно. А последнее поле необходимо, поскольку оно поможет идентифицировать процесс, вызвавший ошибку.

Точно такую же структуру имеют сообщение об истечении лимита времени (Time Exceeded Message) и сообщение об обрыве источника Source Quench Message (тип 4).

А вот сообщение о неверном параметре (например, указан неверный номер порта) выглядит уже по-другому (таблица 27.7).

Сообщение Parameter Problem Message *Таблица 27.7*

Тип	Код	Контрольная сумма
Указатель	Не используется	
Интернет-заголовок плюс первые 64 бита оригинального сообщения (пакета)		

Поле Указатель в сообщении о неверном параметре (Parameter Problem Message) указывает на то место в заголовке, которое вызвало ошибку.

Сообщение о переадресации (Redirect Message) имеет следующую структуру:

Сообщение Redirect Message *Таблица 27.8*

Тип	Код	Контрольная сумма
Адрес шлюза		
Интернет-заголовок плюс первые 64 бита оригинального сообщения (пакета)		

Чтобы понять, что такое сообщение о переадресации, рассмотрим следующий пример. Система Б определяет, что посланный системой А пакет некорректен. Системе Б нужно отправить системе А сообщение об ошибке. Система Б определяет, что единственным маршрутом назад для данного пакета является маршрут через систему А. Тогда система Б посылает системе А два пакета: первый с сообщением о некорректном пакете, а второе — сообщение переадресации, докладывающее, что у системы А проблемы с таблицей маршрутизации, которая, возможно, содержит ошибку.

Сообщения типа эхо-запрос (ping, тип 8) и эхо-ответ (pong, тип 0) имеют следующую структуру:

Сообщения Echo или Echo Reply Message *Таблица 27.9*

Тип	Код	Контрольная сумма
Идентификатор		Последовательность
Данные		

Поля Идентификатор и Последовательность могут использоваться источником эха для передачи вспомогательной информации. Например, идентификатор может использоваться как порт при использовании протоколов TCP/UDP для идентификации службы, а номер последовательности может увеличивается на единицу при отправке каждого запроса (то есть выступать в роли счетчика).

27.2.3. Тип и код ICMP-сообщения

В следующей таблице перечислены все типы ICMP-сообщений. Об их структуре вы можете прочитать в документе RFC 792. Типы 17 и 18 описаны в документе RFC 950.

Типы ICMP-сообщений *Таблица 27.8*

Тип	Код	Название сообщения	Описание
0		Echo Reply Message	Эхо-ответ
	0		Код всегда равен 0
3		Destination Unreachable Message	Адресат недоступен
	0		Сеть недоступна
	1		Узел недоступен — что-то случилось с компьютером, возможно, он просто выключен
	2		Протокол недоступен — запрашиваемый протокол не поддерживается
	3		Порт недоступен — на машине ни одна служба не связана с указанным номером порта
	4		Длина пакета слишком велика, а в его заголовке установлен флаг DF (Don't Fragment), то есть не фрагментировать. Для передачи большого пакета его нужно фрагментировать (разбить на части), а так как установлен флаг DF, фрагментация, а следовательно, и передача пакета невозможна
	5		Ошибочный маршрут источника
4		Source Quench Message	Обрыв источника
	0		Код всегда равен 0
5		Redirect Message	Переадресация
	0		Переадресация пакетов для сети
	1		Переадресация пакетов для узла
	2		Переадресация пакетов для сети и типа обслуживания (TOS, Type Of Service)
	3		Переадресация пакетов для узла и типа обслуживания (TOS, Type Of Service)
8		Echo Message	Эхо-запрос
	0		Код всегда равен 0
11		Time Exceeded Message	Превышен лимит времени
	0		При передаче превышено «время жизни» (TTL, Time To Live)

Тип	Код	Название сообщения	Описание
	1		Превышено время реассемблирования (сборки) фрагментов
12		Parameter Problem Message	Ошибочный параметр
	0		Указатель на ошибочный параметр (табл.27.7)
13		Timestamp Message	Запрос временной метки
	0		Код всегда равен 0
14		Timestamp Reply Message	Ответ о временной метке
	0		Код всегда равен 0
15		Information Request	Информационный запрос (запрашивается номер нашей сети)
	0		Код всегда равен 0
16		Information Reply Message	Информационный ответ (возвращается номер нашей сети)
	0		Код всегда равен 0
17 (*)		Information Request	Информационный запрос (запрашивается маска адреса)
	0		Код всегда равен 0
18 (*)		Information Reply Message	Информационный ответ (возвращается маска адреса)
	0		Код всегда равен 0

27.2.4. Функции для работы с протоколом ICMP

Для работы с протоколом ICMP существует 12 основных функций. Все эти функции описаны в файле `/usr/src/linux/net/ipv4/icmp.c`. У вас нет этого файла? Тогда установите исходники ядра (странно, почему вы до сих пор этого не сделали).

- **icmp_address()** — отправка ответа на запрос о маске адреса;
- **icmp_discard()** — удаляет ICMP-пакет;
- **icmp_echo()** — эхо-запрос;
- **icmp_init()** — инициализирует служебные подпрограммы протокола ICMP в операционной системе;
- **icmp_out_count()** — увеличивает счетчик отправленных пакетов;
- **icmp_rcv()** — прием ICMP-пакета;
- **icmp_redirect()** — отправка сообщения переадресации;
- **icmp_send()** — отправка ICMP-сообщения;
- **icmp_timestamp()** — ответ на запрос о времени создания;
- **icmp_unreach()** — отправляет сообщение об ошибке;
- **xrlim_allow()** — решает, отправлять ли ICMP-пакет или нет;
- **xrlim_init()** — ограничение скорости передачи ICMP-пакетов (в версии ядра 2.0).

27.2.4.1. Технические подробности

Прежде чем перейти к рассмотрению функций ICMP, разберемся, как же ICMP-пакеты принимаются операционной системой. Собственно, ICMP-пакет принимается операционной системой Linux так же, как и любой другой пакет. Драйвер сетевой платы (или другого сетевого устройства) собирает полный пакет данных, затем он строит структуру **sk_buff**.

Листинг 27.1. Структура sk_buff

```
struct sk_buff {
 /* Эти два члена должны быть первыми */
 struct sk_buff* next;/* Следующий буфер в списке*/
 struct sk_buff*·prev;/* Предыдущий буфер в списке*/

 struct sk_buff_head * list; /* "Голова" списка */
 struct sock *sk;      /* Сокет */
 struct timeval stamp; /* Время прибытия пакета */
 struct net_device  *dev;  /* Сетевое устройство */

 /* Заголовок транспортного уровня */
 union
 {
   struct tcphdr *th;
   struct udphdr *uh;
   struct icmphdr*icmph;
   struct igmphdr*igmph;
   struct iphdr  *ipiph;
   struct spxhdr *spxh;
   unsigned char *raw;
 } h;

 /* Заголовок сетевого уровня */
 union
 {
   struct iphdr  *iph;
   struct ipv6hdr*ipv6h;
   struct arphdr *arph;
   struct ipxhdr *ipxh;
   unsigned char *raw;
 } nh;

 union
 {
```

```
   struct ethhdr *ethernet;
   unsigned char *raw;
 } mac;

 struct  dst_entry *dst;


 char   cb[48];

 unsigned int  len;      /* Длина данных */
   unsigned int  data_len;
 unsigned int  csum;       /* Контрольная сумма */
 unsigned char   __unused,  /* Не используется */
          cloned,   /* Заголовок должен  клонироваться */
          pkt_type,   /* Класс пакета */
          ip_summed;  /* контрольная сумма IP */
 __u32    priority;   /* Приоритет пакета */
 atomic_t users; /* Счетчик пользователей — см. datagram.
c,tcp.c      */
 unsigned short protocol;   /* Протокол пакета */
 unsigned short security;   /* Уровень безопасности */
 unsigned int  truesize;   /* Размер буфера */

 unsigned char *head;    /* Заголовок буфера */
 unsigned char *data; /* Указатель заголовка данных */
 unsigned char *tail; /* Указатель "хвоста" */
 unsigned char *end;  /* Конечный указатель */

 void      (*destructor)(struct sk_buff *);
#ifdef CONFIG_NETFILTER
        unsigned long  nfmark;
        /* Cache info */
        __u32     nfcache;
        /* Ассоциированное соединение */
        struct nf_ct_info *nfct;
#ifdef CONFIG_NETFILTER_DEBUG
        unsigned int nf_debug;
#endif
#endif /*CONFIG_NETFILTER*/

#if defined(CONFIG_HIPPI)
 union{
   __u32  ifield;
 } private;
#endif
```

```
#ifdef CONFIG_NET_SCHED
        __u32           tc_index;     /* Индекс контроля
трафика */
#endif
};
```

Данная структура описана в файле /usr/src/linux/include/linux/skbuff.h. После формирования структуры sk_buff она передается драйвером функции netif_rx.

```
int netif_rx(struct sk_buff *skb);
```

Функция **netif_rx()** описана в файле /usr/src/linux/net/core/dev.c. Она получает пакет от драйвера сетевого устройства и ставит его в очередь (очередь называется backlog) протокола высшего уровня. Функция возвращает следующие значения:

- NET_RX_SUCCESS — пакет удачно поставлен в очередь;
- NET_RX_CN_LOW — имеется небольшая «пробка» при постановке пакета в очередь, но скоро она «рассосется»;
- NET_RX_CN_MOD — «пробка» чуть больше — средней «длины»;
- NET_RX_CN_HIGH — очень большая «пробка»;
- NET_RX_DROP — пакет был удален.

Если в очереди backlog находятся более 300 пакетов, новый пакет будет удален без какого-либо предупреждения.

Драйвер сетевого устройства при формировании структуры sk_buff устанавливает ее поле protocol. Функция **netif_rx()**, как уже было сказано, передает пакет «наверх», затем функция **net_bh()** использует значение поля protocol, установленное драйвером, для вызова соответствующей протоколу программы и передает этой программе пакет. Для протокола ICMP такой программой является функция **icmp_rcv()**, описанная в файле /usr/src/linux/net/ipv4/icmp.c.

```
int icmp_rcv(struct sk_buff *skb);
```

Функция icmp_rcv выполняет следующие действия:

1. Увеличивает значение счетчика входящих пакетов ICMP_INC_STATS_BH(IcmpInMsgs).
2. Удаляет пакет, если его размер слишком мал.
3. Проверяет контрольную сумму: если она ошибочна, увеличивает счетчик некорректных пакетов.
4. Проверяет тип ICMP-пакета. Если тип пакета превышает максимальное значение 18, то пакет удаляется. При этом увеличивается счетчик некорректных пакетов.

5. Вызывает функцию-обработчик ICMP-сообщения в зависимости от значения типа сообщения.

Соответствие номеров типа служебным функциям (описаны в файле icmp.c) *Таблица 27.9*

Тип	Функция	Описание
0, 1, 2, 6, 7, 9, 10, 12, 14-16, 18	void icmp_discard (struct sk_buff *skb)	Удаляет пакет
3,4,11	void icmp_unreach (struct sk_buff *skb)	Данная функция определяет максимальный размер модуля передачи (MTU, Maximum Transmission Unit), а также передает информацию об ошибке любому модулю, которому нужна эта информация
5	void icmp_redirect (struct sk_buff *skb)	Пакет переадресации, который означает примерно следующее: «Не посылайте мне пакет по адресу xxx.xxx.xxx.xxx, потому что он будет отправлен обратно»
8	void icmp_echo (struct sk_buff *skb)	Функция для отправки эхо-запроса. Эхо-ответ (тип 0) отправляется ядром автоматически (если вы не отключили эту возможность)
13	void icmp_timestamp (struct sk_buff *skb)	Запрос о времени создания пакета — один из самых эффективных методов измерения производительности глобальных сетей в реальном времени. Метод заключается в следующем: на тестируемый узел посылаются небольшие пакеты, которые сразу же удаляются. Информация из последнего пакета копируется в пакет эхо-ответа, в этот пакет также вставляется информация о текущем времени. После этого пакет ставится в очередь как обычно
17	void icmp_address (struct sk_buff *skb)	Отправка ответа на запрос о маске адреса

Рассмотренные выше функции обрабатывают входящие ICMP-пакеты. Но мы ведь можем не только принимать ICMP-пакеты, но и передавать, поэтому есть также и «исходящие» функции.

Для отправки ICMP-сообщений используется функция

```
void icmp_send(struct sk_buff *skb_in, int type, int code,
u32 info);
```

Данная функция отправляет сообщения типа type, с кодом code и телом info.

После отправки пакета неплохо бы проверить, будет ли он реально отправлен. Это можно сделать с помощью функции

```
int xrlim_allow(struct dst_entry *dst, int timeout);
```

Эта функция определяет, отправлять или нет текущий пакет из очереди. Если пакет должен быть удален, функция возвращает 0, иначе — 1.

После отправки ICMP-пакета нужно увеличить счетчик отправленных пакетов. Это можно сделать с помощью функции icmp_out_count():

```
void icmp_out_count(int type);
```

Данная функция увеличивает счетчик оправленных пакетов типа type.

На этом обзор протокола ICMP в Linux заканчивается и мы переходим к рассмотрению механизма гнезда (сокета), с помощью которого можно реализовать двунаправленный обмен данными между двумя компьютерами.

27.3. Программирование сокетов

27.3.1. Что такое сокет?

Сокет — это двунаправленный канал между двумя компьютерами в сети, который обеспечивает конечную точку соединения. «Двунаправленный» означает, что данный могут передаваться в двух направлениях — от клиента к серверу и наоборот. Понятие сокета — абстрактное, это как бы программный соединитель, через который обмениваются данными программа-сервер и программа-клиент.

Сокет-интерфейс используется для получения доступа к транспортному уровню протокола TCP/IP и представляет собой набор системных вызовов операционной системы и библиотечных функций на языке C. Все эти функции можно условно разделить на три группы:

- управляющие функции;
- функции установления связи;
- функции сетевого ввода/вывода.

Общий алгоритм работы сетевой программы, использующей сокеты:

1. Подготовить (создать) сокет — функция socket().
2. Связать сокет — функция bind().
3. Установить связь с удаленным компьютером (клиенту — установить связь, а серверу — ожидать установления связи).
4. Произвести обмен данными — функции recv() и send().
5. Завершить сеанс связи — close() и shutdown().

Библиотечные функции для работы с сокетами находятся в заголовочном файле sys/socket.h, поэтому для любой сетевой программы обязательна следующая директива:

```
#include <sys/socket.h>
```

27.3.2. Создание и связывание сокета

Основная задача управляющих функций — организовать взаимодействие двух компьютеров, точнее процессов, а также завершить сеанс связи этих процессов. К управляющим функциям относятся функции:

- **socket()** — создание сокета;
- **bind()** — связывание сокета;
- **close()** и **shutdown()** — завершение сеанса связи.

Начнем по порядку, а именно, с функции **socket()**. Ее прототип следующий:

```
#include <sys/types.h>
#include <sys/socket.h>
extern int socket (int __domain, int __type,
                   int __protocol) __THROW;
```

Первый аргумент определяет набор протоколов. Особо вдаваться в подробности не будем — просто всегда в качестве параметра domain передавайте значение AF_INET, что означает использование стека протоколов TCP/IP.

Аргумент **type** позволяет установить режим работы: с установлением соединения и без такового — значения SOCK_STREAM и SOCK_DGRAM соответственно. Для непосредственного доступа к протоколам IPv4 используется параметр SOCK_RAW. Для его использования нужно подключить заголовочный файл:

```
#include <netinet/in.h>
```

Третий параметр лучше всего установить равным 0. В этом случае будет выбран протокол по умолчанию в зависимости от режима работы:

TCP, если мы выбрали режим SOCK_STREAM;

UDP, если мы выбрали SOCK_DGRAM.

Если вы установили значение SOCK_RAW, вы можете указывать в качестве последнего параметра непосредственно значения из файла `/etc/protocols`. Фрагмент этого файла приведен ниже.

Листинг 27.2. Фрагмент файла /etc/protocols

```
ip  0  IP        # Протокол Интернета
icmp  1  ICMP    # Протокол ICMP
igmp  2  IGMP    # Протокол IGMP
                 # (Internet Group Management Protocol)
ggp 3  GGP       # Протокол GGP (gateway-gateway )
tcp 6  TCP       # Протокол TCP
udp 17 UDP       # Протокол UDP
```

Если сокет создан успешно, функция возвращает дескриптор сокета — целое положительное число. В случае ошибки функция возвращает значение –1 (отрицательное число). Вот небольшой пример:

```
int sock;
sock = socket (AF_INET, SOCK_STREAM, 0);
if (sock==-1) {
printf("Ошибка при создании сокета\n");
exit(1);
}
```

Чтобы связать созданный нами сокет с локальным портом, например, 1234, нужно использовать системный вызов **bind**():

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

extern int bind (int fd, struct sockaddr *addr, socklen_t
len)  __THROW;
```

Первый аргумент функции задает дескриптор нашего сокета. Второй — это указатель на структуру типа sockaddr. Все структуры данного типа определены в файле `socket.h`:

```
# define __SOCKADDR_ALLTYPES \
  __SOCKADDR_ONETYPE (sockaddr) \
  __SOCKADDR_ONETYPE (sockaddr_at) \
  __SOCKADDR_ONETYPE (sockaddr_ax25) \
  __SOCKADDR_ONETYPE (sockaddr_dl) \
  __SOCKADDR_ONETYPE (sockaddr_eon) \
  __SOCKADDR_ONETYPE (sockaddr_in) \
  __SOCKADDR_ONETYPE (sockaddr_in6) \
  __SOCKADDR_ONETYPE (sockaddr_inarp) \
  __SOCKADDR_ONETYPE (sockaddr_ipx) \
  __SOCKADDR_ONETYPE (sockaddr_iso) \
  __SOCKADDR_ONETYPE (sockaddr_ns) \
  __SOCKADDR_ONETYPE (sockaddr_un) \
  __SOCKADDR_ONETYPE (sockaddr_x25)
```

Мы программируем для сети TCP/IP, поэтому будем использовать структуру **sockaddr_in** (для IPv4) или **sockaddr_in6** (для IPv6).

Последний аргумент — это длина выбранной нами структуры (sockaddr_in) в байтах.

Структура **sockaddr**_in определена в файле `in.h` так:

```
struct sockaddr_in
  {
    __SOCKADDR_COMMON (sin_);
    in_port_t sin_port;        /* Номер порта */
    struct in_addr sin_addr;        /* IP-адрес  */

    unsigned char sin_zero[sizeof (struct sockaddr) -
         __SOCKADDR_COMMON_SIZE -
         sizeof (in_port_t) -
         sizeof (struct in_addr)];
  };

/* Для IPv6.  */
struct sockaddr_in6
  {
    __SOCKADDR_COMMON (sin6_);
    in_port_t sin6_port; /* Порт транспортного уровня */
    uint32_t sin6_flowinfo;  /* Информация  потока IPv6 */
    struct in6_addr sin6_addr;      /* адрес IPv6 */
    uint32_t sin6_scope_id;  /* IPv6-идентификатор */
  };
```

Поля структуры **sockaddr_in** означают следующее:

- **sin_** — набор используемых протоколов. Так как мы используем TCP/IP, данное поле должно содержать значение AF_INET;
- **sin_port** — номер порта;
- **sin_addr** — структура, определяющая адрес узла;
- **sin_zero** — обычно не используется.

Структура **struct in_addr**, определяющая адрес узла, также описана в файле `in.h`:

```
struct in_addr
  {
    in_addr_t s_addr;
  };
```

Обычно поле **s_addr** должно принимать значение INADDR_ANY — сейчас поясню почему. Структура **sockaddr_in** должна быть заполнена ДО вызова функции bind(). Если поле **sin_addr.s_addr** принимает значение INADDR_ANY, то функция bind() автоматически привяжет к сокету адрес локального компьютера и нам не нужно будет указывать его явно — так наша программа будет универсальной.

Функция **bind**() возвращает 0 в случае успеха, и −1, если произошла ошибка. Вот небольшой пример использования этой функции:

```
struct sockaddr_in client;
...
client.sin_family = AF_INET;
client.sin_addr.s_addr = INADDR_ANY;
client.sin_port = 1235;

bind (sock, (struct sockaddr *)&client, sizeof(client));
```

27.3.3. Установление связи с удаленным компьютером

Устанавливать связь можно как на стороне сервера, так и на стороне клиента. На стороне клиента используется только один вызов — **connect**(), который «спрашивает» у сервера: «Могу ли я подключиться?», то есть передает запрос на установление соединения. На сервере используются функции:

- **listen**() — ожидание клиента;
- **accept**() — подтверждение запроса клиента на установление соединения.

Сервер должен постоянно прослушивать сокет — ожидать новых клиентов. Как только новый клиент посылает запрос на установление соединения, сервер может либо разрешить ему подключиться (connect), либо запретить (например, если сервер уже обслуживает другого клиента).

Функция listen()

Вызов **listen**() «заставляет» программу-сервер работать в режиме ожидания запроса на соединение от клиента. Прототип этой функции следующий:

```
#include <sys/socket.h>
extern int listen (int __fd, int __n) __THROW;
```

Первый параметр — это дескриптор сокета, а второй — максимальное количество запросов на установление связи (другими словами, максимальное количество клиентов).

Как и функция bind(), функция listen() в случае успеха возвращает 0. Пример вызова функции:

```
if (listen (sock1, 3)!=0) {
printf("Ошибка при вызове listen(sock1,3)\n");
exit(1);
};
```

Функция connect()

Используется программой-клиентом для отправки запроса на подключение к серверу. Прототип функции следующий:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

extern int connect (int __fd, struct sockaddr_in *addr,
socklen_t __len)  __THROW;
```

Первый параметр — это дескриптор сокета, созданного функцией **socket**() и привязанного функцией **bind**(). Привязку сокета функцией **bind**() выполнять не обязательно: если сокет не был привязан до вызова **connect**(), привязка будет выполнена автоматически.

Второй параметр — это указатель на структуру типа **sockaddr_in**, содержащую информацию о сервере: его IP-адрес, номер порта, а также семейство протоколов.

Последний параметр — это размер структуры **sockaddr_in** в байтах. В случае успеха функция возвращает 0, а в случае ошибки –1.

Вот пример использования вызова connect:

```
struct sockaddr_in server;
struct hostent *h;
...
// определяем IP-адрес сервера
h = gethostbyname ("server.domain.ru");
memcpy ((char *)&server.sin_addr,h->h_addr,h->h_length);
// Определяем порт сервера
server.sin_port = 1234;
// Определяем семейство протоколов
server.sin_family = AF_INET;

// Вызов функции connect()
connect (sock, &server, sizeof(server));
```

Если вы используете режим без установления соединения (SOCK_DGRAM), вызов **connect** () необязателен.

Функция accept()

Если максимальное число клиентов не превышено, сервер может принять запрос клиента. Для этого используется функция accept(). Данная функция используется только при работе в режиме с установлением соединения. Прототип функции следующий:

```
#include <sys/socket.h>
#include <netinet/in.h>
extern int accept (int __fd, struct sockaddr_in *addr,
socklen_t *__restrict __len) __THROW;
```

Первый параметр — это дескриптор сокета, второй — указатель на структуру, где можно разместить адрес клиента, причем данную структуру инициализировать не нужно. Последний параметр — размер структуры, указанной во втором параметре.

Системный вызов **accept**() работает так. Сначала он извлекает из очереди **listen**() запрос на соединение и создает новый сокет, через который будет производиться обмен данными с клиентом, например,

```
// получаем сокет клиента
sock2 = accept (sock1, &client, &ans_len);
// передаем клиенту информацию
write (sock2, MSG_TO_SEND, sizeof(MSG_TO_SEND));
```

Если вызов **accept**() завершился успехом, структура **addr**, задаваемая во втором параметре, будет содержать IP-адрес клиента.

Если очередь **listen**() пуста, то наш сервер будет ожидать появления нового клиента. В случае ошибки функция **accept**() возвращает отрицательное значение.

27.3.4. Функция gethostbyname()

Пользователям обычно удобнее указать символьное имя сервера, чем его IP-адрес. Для разрешения имени служит функция **gethostbyname**(). Вот ее прототип:

```
#include <netinet/in.h>
#include <netdb.h>
struct hostent *gethostbyname (char *name);
```

Данная функция возвращает указатель на структуру типа **hostent**, содержащую следующие поля:

- **char *h_name** — доменное имя узла;
- **char **h_aliases** — псевдонимы узла, если таковые определены;
- **char *h_addr** — IP-адрес узла;
- **int h_addrtype** — набор используемых протоколов (в нашем случае — AF_INET);
- **int h_lenght** — длина адреса узла.

Примеры использования функции:

```
struct hostent *h;
h=gethostbyname (*argv);
```

```
if (h==NULL) {
  printf ("Невозможно разрешить имя: `%s`\n",*argv);
  exit (1);
}
// Выводим IP-адрес. Вывод в виде: имя -> адрес
printf ("%s -> %s \n",*argv,inet_ntoa(*((struct in_addr
*)h->h_addr_list[0])));
```

Узнать свой собственный адрес можно с помощью функции getsockname():

```
extern int getsockname (int __fd, __SOCKADDR_ARG __addr,
      socklen_t *__restrict __len) __THROW;
```

Ей нужно передать три параметра — дескриптор сокета, адрес структуры, которая будет содержать информацию о нашем узле (его адрес). Третий параметр будет содержать длину адресной структуры.

27.3.5. Функции сетевого ввода/вывода

После успешного установления соединения можно начать обмен данными. Для отправки и получения данных можно использовать обыкновенные функции для работы с файлами — read() и write(), только вместо дескриптора файла нужно указывать дескриптор сокета. Однако рекомендуется использовать системные вызовы **send**() и **recv**(), которые предназначены именно для работы с сокетами. Эти системные вызовы будут рассмотрены ниже.

Если вы работаете в режиме без установления соединения, вам нужно использовать функции **sendto**() и **recvfrom**(). Первая функция отправляет данные, а вторая — принимает. Функция **sendto**() вместе с данными позволяет указать адрес получателя, а **recvfrom**() возвращает не только полученные данные, но и адрес отправителя.

Обмен данными в режиме SOCK_STREAM

Для отправления данных используется функция send():

```
#include <sys/types.h>
#include <sys/socket.h>
extern ssize_t send (int __fd, __const void *__buf,
size_t __n, int __flags) __THROW;
```

Первый параметр — дескриптор сокета, второй — указатель на область памяти, которая содержит передаваемые данные. Третий параметр — это размер передаваемых данных в байтах. Последний параметр позволяет определить поведение функции **send**(): если он равен 0, то вызов **send**() полностью аналогичен вызову **write**().

Нужно отметить особенность работы этой функции: если буфер сокета **__fd** переполнен, функция переводит программу в состояние ожидания освобождения буфера. Такое может случиться, если узел-приемник по каким-то причинам не успевает принять данные.

Функция возвращает число байтов отправленных данных или –1 в случае ошибки.

Для приема данных используется функция recv():

```
#include <sys/types.h>
#include <sys/socket.h>
extern ssize_t recv (int __fd, void *__buf, size_t __n,
int __flags) __THROW;
```

Первый параметр, как обычно, задает дескриптор сокета. В случае успешного приема данных они будут размещены в буфере **__buf** — второй параметр функции **recv**(). Третий параметр задает размер области, на которую указывает второй параметр. Если четвертый параметр (флаги) принимает значение 0, то вызов **recv**() аналогичен вызову **read**(). Четвертый параметр может принимать следующие значения:

- MSG_PEEK — прочитанные данные не удаляются. Следующий вызов функции recvfrom() опять возвратит эти данные.
- MSG_WAITALL — процесс будет блокирован до получения всего запрошенного объема данных, а не до получения первого сообщения. Только для сокетов SOCK_STREAM!

Если через указанный сокет ничего нельзя принять, функция переводит программу в состояние ожидания — до появления данных в канале связи.

Функция возвращает количество принятых байтов или –1 в случае ошибки.

Обмен данными в режиме SOCK_DGRAM

Функция sendto() позволяет отправить данные по протоколу UDP (без установления соединения), указав при этом узел-приемник:

```
extern ssize_t sendto (int __fd, __const void *__buf,
       size_t __n,
       int __flags, __CONST_SOCKADDR_ARG __addr,
       socklen_t __addr_len) __THROW;
```

Назначение первых четырех аргументов такое же, как и функции **send**(), а последние два аргумента задают структуру типа **struct sockaddr_in**, содержащую информацию об адресе узла-приемника, и размер этой структуры соответственно. Аргумент `__addr` — это адрес структуры **sockaddr_in**, а не она сама!

Как и функция **send**(), функция **sendto**() возвращает количество байтов отправленных данных или –1, если произошла ошибка.

Функция **recvfrom**() позволяет получить данные по протоколу UDP:

```
extern ssize_t recvfrom (int __fd, void *__restrict __buf,
size_t __n, int __flags,
__SOCKADDR_ARG __addr,
socklen_t *__restrict __addr_len) __THROW;
```

Назначение первых четыре аргументов такое же, как и у функции recv(). Предпоследний аргумент позволяет указать структуру, в которую будет записана информация об адресе узла-отправителя. Помните: нужно передать адрес структуры, а не саму структуру. Последний параметр задает длину этой структуры.

Функция возвращает количество принятых данных или –1 в случае ошибки. Проверить ошибку можно и по-другому: если структура адреса узла отправителя пуста (равна NULL), значит, произошла ошибка.

27.3.6. Завершение сеанса связи

Для закрытия сеанса связи можно использовать один из двух системных вызовов: **close**() или **shutdown**().

Системный вызов **close**() также используется для закрытия файлов. Вот прототип этой функции:

```
int close(int __fd);
```

Данной функции нужно передать всего один параметр — дескриптор сокета.

Однако вызов **close**() использовать не рекомендуется из-за специфики его работы: он закрывает сокет грубо, не дожидаясь завершения передачи данных. В результате использования **close**() вероятность повреждения принимаемых или передаваемых данных очень высока. В принципе, использовать **close**() можно на клиенте, но на сервере это недопустимо: сначала нужно использовать **shutdown**(), а потом уже **close**().

Вызов **shutdown**() используется для завершения сеанса связи, при этом еще не переданные данные будут переданы другой стороне. Прототип функции:

```
extern int shutdown (int __fd, int __how) __THROW;
```

Первый параметр — это дескриптор сокета, а второй может принимать одно из трех значений:

- SHUT_RD (или 0) — передать данные, которые еще не переданы, но их отправка уже началась, и больше не принимать данные для чтения.
- SHUT_WR (или 1) — передать данные и запретить прием данных через сокет.
- SHUT_RDWR (или 2) — передать данные и запретить вообще обмен через сокет — ни приема, ни передачи.

27.3.7. Программа-сервер

В этом пункте мы напишем две программы — сервер и клиент. Программа-сервер после запуска сразу же перейдет в режим ожидания («прослушивания») новых клиентов. Максимальное количество клиентов — 3. Как только подключится клиент, сервер отправит ему сообщение «What *is* your name?», в ответ на которое клиент передаст свое имя — «Denis». Сервер прочитает переданную клиентом информацию и выведет ее на консоль. Клиент, в свою очередь, выведет на консоль запрос сервера.

С целью упрощения исходного кода как сервера, так и клиента, обработку ошибок производить не будем, поэтому будьте готовы к тому, что ваш клиент выдаст сообщение Segmentation fault в ответ на неверно заданные параметры. Я рекомендую в качестве имени сервера использовать `localhost` и обе программы запускать на одном компьютере — это же только демонстрация.

Вот исходный код программы-сервера:

Листинг 27.3. Программа-сервер

```
#include <sys/types.h>
#include <netdb.h>
#include <memory.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>

#define SERVER_PORT 1234
#define BUF_SIZE 64
#define MSG_TO_SEND "What is your name?\n"

int main () {

int sock1, sock2;

int ans_len, total=0;
```

```
char buffer[BUF_SIZE];
struct sockaddr_in sin, client;

sock1 = socket (AF_INET, SOCK_STREAM, 0);
memset ((char *)&sin, '\0', sizeof(sin));

sin.sin_family = AF_INET;
sin.sin_addr.s_addr = INADDR_ANY;
sin.sin_port = SERVER_PORT;

bind (sock1, (struct sockaddr *)&sin, sizeof(sin));

printf("Server running...\n");

listen (sock1, 3);

while (1) {
    ans_len = sizeof(client);
    sock2 = accept (sock1, &client, &ans_len);
    write (sock2, MSG_TO_SEND, sizeof(MSG_TO_SEND));
    total+=1;
    ans_len = read (sock2, buffer, BUF_SIZE);
    write (1, buffer, ans_len);
    printf("Client no %d\n",total);
    shutdown (sock2, 0);
    close (sock2);
 };
return 0;
}
```

Теперь разберемся, что есть что. Сначала мы определяем некоторые макросы: номер порта, который будет прослушивать сервер, размер буфера передаваемых данных и текст запроса клиенту.

Стандартные номера портов определены в файле `netinet/in.h`:

```
enum
  {
    IPPORT_ECHO = 7,    /* Echo service.  */
    IPPORT_DISCARD = 9,    /* Discard transmissions service.  */
    IPPORT_SYSTAT = 11,    /* System status service.  */
    IPPORT_DAYTIME = 13, /* Time of day service.  */
    IPPORT_NETSTAT = 15,    /* Network status service.  */
    IPPORT_FTP = 21,    /* File Transfer Protocol.  */
    IPPORT_TELNET = 23,    /* Telnet protocol.  */
    IPPORT_SMTP = 25,    /* Simple Mail Transfer Protocol.
```

```
 */
    IPPORT_TIMESERVER = 37,  /* Timeserver service.  */
    IPPORT_NAMESERVER = 42,  /* Domain Name Service.  */
    IPPORT_WHOIS = 43,   /* Internet Whois service.  */
    IPPORT_MTP = 57,

    IPPORT_TFTP = 69,    /* Trivial File Transfer Proto-
col.  */
    IPPORT_RJE = 77,
    IPPORT_FINGER = 79,    /* Finger service.  */
    IPPORT_TTYLINK = 87,
    IPPORT_SUPDUP = 95,    /* SUPDUP protocol.  */


    IPPORT_EXECSERVER = 512, /* execd service.  */
    IPPORT_LOGINSERVER = 513,      /* rlogind service.  */
    IPPORT_CMDSERVER = 514,
    IPPORT_EFSSERVER = 520,

    /* UDP ports.  */
    IPPORT_BIFFUDP = 512,
    IPPORT_WHOSERVER = 513,
    IPPORT_ROUTESERVER = 520,

    /* Ports less than this value are reserved for privi-
leged processes.  */
    IPPORT_RESERVED = 1024,

    /* Ports greater this value are reserved for (non-
privileged) servers.  */
    IPPORT_USERRESERVED = 5000
  };
```

Нам понадобятся сразу два сокета: первый — это сокет сервера, а через второй сокет мы будем производить обмен данными с клиентом.

```
int sock1, sock2;
```

Следующие переменные: `ans_len` используется для хранения размера передаваемой клиентом информации — фактического размера структуры **struct sockaddr_in**, а `total` — это счетчик числа клиентов, используемый для вывода порядкового номера клиента.

Переменная `buffer` размера BUF_SIZE — это наш буфер для обмена информацией. Нам нужны две структуры типа `sockaddr_in` — одна для сервера (sin) и одна для клиента (client).

В строке

```
sock1 = socket (AF_INET, SOCK_STREAM, 0);
```

мы создаем наш, «серверный», сокет: набор протоколов — TCP/IP, режим — с установлением соединения.

Затем мы инициализируем структуру sin:

```
memset ((char *)&sin, '\0', sizeof(sin));

sin.sin_family = AF_INET;           // TCP/IP
sin.sin_addr.s_addr = INADDR_ANY;   // можем работать на
                                    // любом адресе
sin.sin_port = SERVER_PORT;         // указываем порт (1234)
```

После создания сокета и инициализации структуры sin, нужно связать наш сокет с адресом и портом сервера:

```
bind (sock1, (struct sockaddr *)&sin, sizeof(sin));
```

Оператор `listen (sock1, 3)` означает, что мы будем прослушивать сокет sock1 (порт 1234) и максимальное число клиентов не должно превышать 3.

Как и любой нормальный сервер, мы должны работать в бесконечном цикле, постоянно обрабатывая запросы клиентов. В бесконечном цикле мы:

1. получаем размер структуры client

   ```
   ans_len = sizeof(client);
   ```

2. создаем сокет sock2, через который будем обмениваться данными с клиентом. Если в очереди listen нет клиентов, мы переходим в состояние ожидания

   ```
   sock2 = accept (sock1, &client, &ans_len);
   ```

3. как только подключится клиент, мы отправим ему сообщение MSG_TO_SEND

   ```
   write (sock2, MSG_TO_SEND, sizeof(MSG_TO_SEND));
   ```

4. увеличиваем счетчик клиентов

   ```
   total+=1;
   ```

5. получаем размер прочитанных данных, сами данные записываются в буфер buffer

   ```
   ans_len = read (sock2, buffer, BUF_SIZE);
   ```

6. выводим прочитанные данные на стандартный вывод

   ```
   write (1, buffer, ans_len);
   ```

7. завершаем сеанс связи

```
shutdown (sock2, 0);
```

8. закрываем сокет

```
close (sock2);
```

Конечно, любой нормальный сервер при поступлении определенных сигналов, например, SIG_HUP, должен корректно перезапуститься или вообще завершить работу. Наш сервер этого не делает — обработку сигналов, я надеюсь, вы можете добавить сами.

Теперь мы можем откомпилировать нашу программу:

```
$ gcc -o server server.c
```

Запускаем:

```
./server
```

Программа перешла в состояние ожидания новых клиентов.

27.3.8. Программа-клиент

Программа-клиент несколько проще, чем сервер. Вот ее листинг:

Листинг 27.4. Программа-клиент

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <memory.h>
#include <stdio.h>

#define SERVER_HOST "localhost"
#define SERVER_PORT 1234
#define CLIENT_PORT 1235
#define MSG "Denis\n"

main () {
int sock;
int ans_len;
int BUF_SIZE = 64;

char buffer[BUF_SIZE];
```

```
struct hostent *h;
struct sockaddr_in client, server;

sock = socket (AF_INET, SOCK_STREAM, 0);
memset ((char *)&client, '\0', sizeof(client));

client.sin_family = AF_INET;
client.sin_addr.s_addr = INADDR_ANY;
client.sin_port = CLIENT_PORT;

bind (sock, (struct sockaddr *)&client, sizeof(client));
memset ((char *)&client, '\0', sizeof(server));

h = gethostbyname (SERVER_HOST);

server.sin_family = AF_INET;

memcpy ((char *)&server.sin_addr,h->h_addr,h->h_length);

server.sin_port = SERVER_PORT;

connect (sock, &server, sizeof(server));

ans_len = recv (sock, buffer, BUF_SIZE, 0);
write (1, buffer, ans_len);
send (sock, MSG, sizeof(MSG), 0);
close (sock);
exit (0);
}
```

Константа MSG — это сообщение, которое будет передано серверу. Как и в случае с сервером, нам понадобятся две структуры типа **sockaddr_in**:

```
struct hostent *h;
struct sockaddr_in client, server;
```

Структура типа hostent нам нужна для получения адреса сервера.

Создаем сокет, заполняем информацию о клиенте и связываем сокет:

```
sock = socket (AF_INET, SOCK_STREAM, 0);
memset ((char *)&client, '\0', sizeof(client));

client.sin_family = AF_INET;
client.sin_addr.s_addr = INADDR_ANY;
client.sin_port = CLIENT_PORT;

bind (sock, (struct sockaddr *)&client, sizeof(client));
```

Перед подключением к серверу нужно определить его IP-адрес:

```
h = gethostbyname (SERVER_HOST);
```

Подключаемся к серверу:

```
server.sin_family = AF_INET;   // набор протоколов
memcpy ((char *)&server.sin_addr,h->h_addr,h->h_length);
// задаем адрес сервера
server.sin_port = SERVER_PORT; // указываем порт сервера

connect (sock, &server, sizeof(server));
```

После подключения к серверу принимаем его запрос, выводим на стандартный вывод, отправляем серверу свое сообщение и закрываем сокет:

```
ans_len = recv (sock, buffer, BUF_SIZE, 0);
write (1, buffer, ans_len);
send (sock, MSG, sizeof(MSG), 0);
close (sock);
```

27.3.9. Установка опций сокета

Поскольку мы используем набор протоколов AF_INET, то в этом пункте будем рассматривать только те опции сокетов, которые относятся к этому набору. Для работы с опциями сокета используются две функции:

- **getsockopt()** — получение опций сокета;
- **setsockopt()** — установка опций сокета.

Прототипы этих функций выглядят так:

```
#include <sys/socket.h>

int getsockopt (int sd, int level,
      int option_name,
      void *restrict option_value,
      socklen_t *restrict option_len);

int setsockopt (int sd, int level,
      int option_name,
      const void *option_value,
      socklen_t option_len);
```

Первый параметр, sd, — это дескриптор сокета. Второй параметр — уровень доступа (существует только один уровень — SOL_SOCKET). Следующий параметр, option_name, — это название опции, значение которой

вы хотите изменить (см. таблицу 27.10). Последние два параметра — это значение опции и его размер.

Наиболее часто используемые опции сокетов *Таблица 27.10*

Название опции	Описание
SO_DEBUG	Включить/выключить (1/0) запись отладочной информации для сокета
SO_BROADCAST	Включить/выключить (1/0) отправку широковещательных сообщений
SO_REUSEADDR	Опция разрешает/запрещает использование локальных адресов
SO_KEEPALIVE	Сохраняет неактивные соединения "в живых" путем посылки сообщений. Если данный сокет не отвечает на сообщения, соединение будет разорвано, а процессу, который осуществлял запись в сокет, будет послан сигнал SIGPIPE. Для включения KEEPALIVE нужно установить значение 1, для выключения — 0
SO_SNDBUF	Устанавливает размер буфера отправки, значение целого типа
SO_RCVBUF	Устанавливает размер буфера приема, значение целого типа
SO_SNDTIMEO	Установка таймаута для отправки сообщений. По умолчанию таймаут равен 0, то есть его вообще нет. Нужно передать значение типа struct timeval
SO_RCVTIMEO	Установка таймаута для приема сообщений. По умолчанию таймаут равен 0, то есть его вообще нет. Нужно передать значение типа struct timeval
TCP_NODELAY	Отключить (1) механизм буферизации сообщений, то есть они будут отправляться сразу, без задержки. Для включения механизма буферизации нужно указать значение 0
TCP_MAXSEG	Установить максимальный сегмент данных. Значение целого типа
TCP_NOPUSH	Не использовать проталкивание (1)
TCP_NOOPT	Не использовать опции TCP (1) Для использования опций передайте значение 0

В случае успешной установки параметра функция **setsockopt**() возвращает 0; в случае ошибки возвращается -1, а переменная errno устанавливается следующим образом:

- EBADF — неверный дескриптор сокета;
- ENOTSOCK — указанный дескриптор является файлом, а не сокетом;
- EFAULT — нет доступа к адресу, на который указывает указатель `optval`, то есть данный адрес находится за пределами видимости приложения.

Функция **getsockopt**() возвращает значение параметра. Кроме вышеперечисленных параметров, функция **getsockopt**() может использовать следующие параметры:

- SO_ERROR — возвращает номер ошибки (будет в возвращаемом значении);
- SO_TYPE — возвращает тип сокета.

Рассмотрим небольшой пример работы с опциями сокетов. Мы установим размер буфера TCP.

Листинг 27.5. Установка опций сокета

```
#include "sock.h"
#include "stdio.h"
main()
{
  int sd;                    /* дескриптор сокета */
  int optval;                /* значение опции */
  int optlen;                /* длина optval    */

  int new_buffsize = 8192;     /* новый размер буфера */

/* создаем сокет  */
  sd = socket(AF_INET, SOCK_STREAM, 0);

/* считывание длины буфера TCP */
  optlen = sizeof(optval);

getsockopt(sd, SOL_SOCKET, SO_SNDBUF, &optval, &optlen);
printf("Size of send buffer %d\n", optval);

getsockopt(sd, SOL_SOCKET, SO_RCVBUF, &optval, &optlen);
printf("Size of recv buffer %d\n", optval);

/* изменяем длину буфера */
setsockopt(sd, SOL_SOCKET, SO_RCVBUF,
           &new_buffsize,sizeof(new_buffsize));
setsockopt(sd, SOL_SOCKET, SO_SNDBUF,
           &new_buffsize,sizeof(new_buffsize));

/* выводим измененную информацию */
getsockopt(sd, SOL_SOCKET, SO_SNDBUF, &optval, &optlen);
printf("New size of send buffer %d\n", optval);

getsockopt(sd, SOL_SOCKET, SO_RCVBUF, &optval, &optlen);
printf("New size of recv buffer %d\n", optval);
}
```

27.3.10. Сигналы и сокеты

С сокетами связаны три сигнала:

- SIGIO — сокет готов к вводу/выводу. Сигнал посылается процессу, который связан с сокетом;
- SIGURG — сокет получил экспресс-данные (мы их использовать не будем, поэтому особо останавливаться на них нет смысла);

- SIGPIPE — запись в сокет больше невозможна. Сигнал посылается процессу, связанному с сокетом. Например, функция write() вызывает сигнал SIGPIPE, если удаленный процесс завершен или связь по сети невозможна.

Пример обработки сигнала SIGPIPE приведен ниже.

Листинг 27.6. Обработка сигнала SIGPIPE

```
#include "sock.h"
#include <signal.h>

/* обработчик сигнала SIGPIPE        */
sigpipe_handler()
{
  err_quit("Получен SIGPIPE \n");
}

main()
{
  int sock;          /* дескриптор сокета */

/* установка обработчика сигнала SIGPIPE      */
  signal(SIGPIPE, sigpipe_handler);

/* работа с сокетом */

}
```

27.3.11. Мультиплексирование

В этой главе мы рассматривали пример программы-сервера, обрабатывающей запросы только от одного клиента. На практике все выглядит намного сложнее: серверу приходится одновременно обрабатывать запросы многих клиентов. Для мультиплексирования запросов клиентов используется системный вызов select(). Этот вызов использует, например, суперсервер **xinetd**.

Листинг 27.7. Мультиплексирование запросов

```
#include "sock.h"
#include <sys/time.h>
main()
{
```

```
  int sock;          /* дескриптор исходного сокета */
  int new_sock;  /* дескриптор, полученный с помощью accept  */
  int retval;        /* возвращаемое значение  */
  struct sockaddr_in server;        /* адрес сокета  */
  fd_set readv;        /* переменная для select */
  fd_set writev;    /* переменная для select */
  struct timeval tout;   /* тайм-аут для select  */

/* бесконечный цикл ожидания  */

for (;;)  {
/* процесс ждет операцию ввода-вывода на сокете ;
одновременно можно ждать и другие операции */
  FD_ZERO(&readv);
  FD_ZERO(&writev);
  FD_SET(sock, &readv);
  FD_SET(sock, &writev);
  tout.tv_sec = 10;      /* 10 секунд */

  retval = select(sock+1, &readv, &writev, 0, &to);

  /* если select возвращает нулевое значение, значит
тайм-аут*/
  if (retval == 0)  {
        err_ret("timeout");
        continue;
  }
/* в противном случае, ищем соответствующий дескриптор */
  if ( (FD_ISSET(sock, &readv)) || (FD_ISSET(sock,
&writev))) {

/* прием связи с сокета */
     new_sock = accept(sock, (struct sockaddr *) 0, (int *)0);

/* работа с сокетом new_sock  */

…

/* закрытие текущей связи   */
       close (new_sock);

  } else  {
err_ret("Это не сокет! Проверьте все дескрипторы\n");
        }
  }
}
```

Системный вызов select() принимает 5 аргументов:

```
int select(int fd, fd_set *input, fd_set *output,
           fd_set *error,
           struct timeval *timeout);
```

Первый аргумент, **fd**, — это файловый дескриптор, который может быть сокетом. Следующие три аргумента задают множества файловых дескрипторов для ожидания условий ввода (input), вывода (output) и ошибок (error). Последний аргумент — это таймаут.

Множества файловых дескрипторов инициализируются с помощью трех макросов:

```
FD_ZERO(fd_set);
FD_SET(fd, fd_set);
FD_CLR(fd, fd_set);
```

Первый макрос полностью очищает множество, следующие два макроса, соответственно, добавляют и удаляют файловый дескриптор. Мы использовали два макроса для ввода и два для вывода. Сначала мы полностью очистили множество, а потом добавили в него соответствующие дескрипторы:

```
FD_ZERO(&readv);
FD_ZERO(&writev);
FD_SET(sock, &readv);
FD_SET(sock, &writev);
```

Особого разговора требует последний параметр — таймаут. Таймаут можно задавать в секундах и миллисекундах. Например, следующие операторы объявляют таймаут длительностью 2 секунды и 5 миллисекунд:

```
struct timeval tout;
/* тайм-аут для select   */

  tout.tv_sec = 2;        /* 2 секунды */
  tout.tv_usec = 5;       /* 5 миллисекунд */
```

Если вы хотите не использовать таймаут (то есть ждать бесконечно), укажите NULL в качестве последнего аргумента.

Функция **select**() возвращает число файловых дескрипторов, на которых выполнились ожидаемые условия (ввод/вывод/ошибка) или -1 при ошибке.

Вот еще один пример использования функции **select**(). Мы будем ожидать ввода из файла и из сокета. Если будет достигнут таймаут в 20 секунд, пользователь увидит соответствующее сообщение; в противном случае он увидит сообщение: «Получен ввод из файла/сокета».

Листинг 27.8. Еще один пример использования select()

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/select.h>

int             k;
int             sock;
int             fd;
int             max_fd;
fd_set          input;
struct timeval timeout;

/* инициализация файла и сокета */
...

/* Инициализируем множество ввода */
FD_ZERO(input);
FD_SET(fd, input);
FD_SET(sock, input);

max_fd = (sock > fd ? sock : fd) + 1;

/* Задаем таймаут */
timeout.tv_sec  = 20;

k = select(max_fd, &input, NULL, NULL, &timeout);

if (k < 0)
  perror("Ошибка при вызове select");
else if (k == 0)
  puts("TIMEOUT");
else
{
  /* Получен ввод */
  if (FD_ISSET(fd, input))
    printf("Получен ввод из файла");
  if (FD_ISSET(sock, input))
    printf("Получен ввод из сокета");
}
```

Вроде бы код программы очень прост, но комментария заслуживает макрос FD_ISSET. С его помощью мы проверяем, есть ли во множестве ввода ввод из какого-либо источника.

27.3.12. Неблокирующие операции

Некоторые функции для работы с сокетами блокируют программу в случае, если удаленный процесс не осуществил требуемую операцию. Примеры таких функций:

- accept();
- connect();
- read();
- write().

Блокирование процесса очень нежелательно, поскольку во время ожидания можно было бы заняться чем-нибудь другим: например, обработать информацию, поступившую с другого сокета. Вы можете объявить сокеты неблокирующими с помощью системного вызовы **ioctl**().

Особенности работы некоторых функций в неблокирующем режиме:

- функция **accept**() сразу же завершает работу с ошибкой EWOULDBLOCK;
- функция **connect**() тоже завершает работу, но с другой ошибкой: EINPROGRESS;
- функции чтения (**read**(), **recv**(), **recvfrom**()) возвращают -1 или 0, если нет считываемых данных.

Ясное дело, что в таком режиме нужно периодически проверять наличие данных — ведь теперь процесс не будет их ожидать: если их нет, то функции просто возвратят -1 или 0.

Пример создания неблокирующих сокетов приведен ниже:

Листинг 27.9. Использование системного вызова ioctl()

```
#include "sock.h"
#include <sys/ioctl.h>

void main()
{
   int sock;
   int on = 1, off = 0;        /* значение для ioctl()  */
/* Создаем неблокирующий сокет */
   ioctl(sock, FIONBIO,&on);
...
}
```

Глава 28

ПРОГРАММИРОВАНИЕ ЯДРА

- КАРКАС МОДУЛЯ
- КОМПИЛЯЦИЯ МОДУЛЯ
- РАБОТА С УСТРОЙСТВАМИ
- ОПЕРАЦИИ НАД УСТРОЙСТВОМ. ПОИСК УСТРОЙСТВ

Из главы 7 вы узнали, что драйверы устройств в Linux выполнены в виде модулей ядра, и познакомились с пакетом module-init-tools (он же modutils для ядер 2.4), содержащим утилиты для выполнения основных операций над модулями ядра. В этой главе я покажу, как создать собственный модуль, позволяющий расширить возможности ядра операционной системы.

28.1. Каркас модуля

Что будет делать ваш модуль, зависит от вас — это может быть драйвер устройства или просто небольшой модуль, дополняющий ядро нужной вам функцией.

Для начала напишем каркас модуля на языке С. Этот каркас можно будет скомпилировать, но в результате получится модуль, который не делает ничего. Он просто послужит вам основой для написания настоящих, серьезных модулей.

Листинг 28.1. Каркас модуля ядра Linux (module1.c)

```
#define MODULE
#define __KERNEL__
#include <linux/module.h>

int init_module()
{
return 0;
}

void cleanup_module()
{

}
```

Теперь разберемся, что означает каждая строчка кода нашего будущего модуля. Первые две строчки делают обыкновенную программу модулем ядра Linux. Это директивы препроцессора cpp, обязательные для каждого модуля. Если вы не укажете их, компилятор сгенерирует совсем не тот код, которого мы от него ожидали.

Третья строка подключает заголовочный файл module.h, в котором находятся все необходимые для создания модуля определения .

Функция init_module() вызывается при загрузке модуля ядром. Если загрузка модуля прошла успешно, функция возвращает 0, в противном случае она должна возвратить любое другое значение — код ошибки.

Функция cleanup_module() вызывается при удалении модуля. Тип возвращаемого ею значения не определен, поэтому проверить, успешно ли произошло удаление, сама функция не позволяет.

Названия функций init_module() и cleanup_module() необязательны — вы можете назвать их по-другому, но для этого вам нужно использовать функции module_init() и module_exit(), которые определены в заголовочном файле init.h.

Переделаем наш шаблон так, чтобы он не использовал стандартные имена функций init_module() и cleanup_module():

Листинг 28.2. Шаблон модуля с переименованием стандартных функций (module2.c)

```
#define MODULE
#define __KERNEL__
#include<linux/module.h> // для модуля
#include<linux/init.h>   // module_init() и module_exit()

int start() {  return 0; }

void stop() {}

module_init(start);
module_exit(stop);
```

Функциям module_init() и module_exit() нужно передать имена функций, которые будут вызваны при инициализации и удалении модуля соответственно. Зачем это нужно? Так, для общего развития — чтобы вы знали, для чего используются эти функции. Ваш модуль не станет работать лучше, если вы переименуете стандартные функции инициализации и удаления.

Помните, для чего используется программа modinfo? Да, для получения информации о модуле. Давайте добавим эту информацию в наш модуль.

Листинг 28.3. Информация о модуле (module3.c)

```
#define MODULE
#define __KERNEL__
#include <linux/module.h>

MODULE_AUTHOR("Denis Kolisnichenko dhsilabs@mail.ru");
MODULE_DESCRIPTION("Linux kernel module");

int init_module()  { return 0; }

void cleanup_module() { return 0; }
```

Макросы `MODULE_AUTHOR` и `MODULE_DESCRIPTION` определены в заголовочном файле module.h, поэтому никаких дополнительных заголовочных файлов подключать не нужно.

При необходимости модуль может выводить на консоль сообщения, например, о невозможности инициализации устройства. Выводимые модулем сообщения будут запротоколированы службой протоколирования ядра — демоном klogd. Выводить сообщения нужно не с помощью функции printf(), а функцией printk(). В этом случае сообщения не только окажутся на системной консоли, но и будут запротоколированы в файле `/var/log/messages`.

Сейчас мы напишем модуль, который будет выводить сообщения при загрузке и при удалении.

Листинг 28.4. Использование функции printk() (module.c)

```
#define MODULE
#define __KERNEL__
#include <linux/module.h>
#include <linux/kernel.h> // printk

MODULE_AUTHOR("Denis Kolisnichenko dhsilabs@mail.ru");
MODULE_DESCRIPTION("Linux kernel module");

int init_module()
{
printk("My module: Starting...\n");
```

```
return 0;
}

void cleanup_module()
{
printk("My module: Stopping...\n");
return 0;
}
```

28.2. Компиляция модуля

Компилировать мы будем файл module.c. Для этого понадобится установленный компилятор gcc, заголовочные файлы и исходные тексты ядра. Если вы дочитали книгу до этой главы, то у вас уже должны быть установлены пакеты:

1. cpp — препроцессор cpp;
2. binutils — набор различных утилит (as, gprof, ld);
3. glibc-kerheaders — заголовочные файлы ядра;
4. glibc-devel — вспомогательные файлы для разработки приложений с использованием стандартной библиотеки C;
5. gcc — компилятор gcc.

Осталось установить пакет kernel-source — исходные тексты ядра Linux. Кроме того, нужно убедиться, что ваше ядро поддерживает динамически загружаемые модули (п. 20.3.2.3). Если опция **Enable loadable module support** выключена, ее нужно включить, сохранить файл конфигурации ядра и перекомпилировать ядро.

Компилятор gcc нужно вызвать со множеством опций, поэтому для облегчения себе работы мы напишем make-файл (п. 21.2):

Листинг 28.5. Makefile для сборки модуля

```
CC=gcc
PATH=/usr/include /usr/src/linux-2.4/include
MODFLAGS:= -O3 -Wall -DLINUX -D__KERNEL__ -I$(PATH)
module.o: module.c
$(CC) $(MODFLAGS) -c module.c
```

Опции компилятора означают следующее:

- O3: будет использован третий уровень оптимизации (что это такое, вы узнаете в справочной системе *gcc*: `man gcc`);
- Wall: включаем все предупреждения;

- DLINUX: генерируем код для Linux;
- I$(PATH): определяем путь поиска заголовочных файлов. По умолчанию компилятор ищет файлы заголовков в каталоге /usr/include, но там может и не быть нужных файлов. Например, для дистрибутива ALT Linux (ядро 2.4.21) файлы заголовков находятся в каталоге /usr/include/linux-2.4.21rel-std-up.

Поместите make-файл в тот же каталог, где находится module.c, и выполните команду `make`. После ее выполнения вы получите файл module.o, который будет находиться в том же каталоге.

```
# insmod module.o
```

Вы увидите сообщение `My module: Starting...` Это же сообщение будет записано в файл протокола `/var/log/messages`.

28.3. Работа с устройствами

Мы только что написали модуль ядра, который можно установить, удалить, который выводит сообщения, но ничего полезного он не делает. Усложним нашу задачу: напишем модуль, управляющий некоторым устройством `/dev/device`. Данного устройства в нашей системе нет, поэтому нам его нужно создать, но этим мы займемся чуть позже.

Перед тем, как приступить к написанию драйвера устройства, нужно поговорить о самих устройствах. Устройства бывают трех типов:

- Символьные: чтение и запись устройства выполняются посимвольно. Примером такого устройства может послужить последовательный порт.
- Блочные: чтение и запись устройства выполняются блоками, как правило, по 512 или 1024 байта. Самый яркий пример блочного устройства — жесткий диск.
- Сетевые: файлов этих устройств вы не найдете в каталоге /dev, поскольку использование файловой системы, то есть работа с этими устройствами как с файлами, неэффективно. Пример — сетевая карта (ethX).

Существуют устройства, которые нельзя отнести ни к одному типу, поскольку они не принимают и не передают данные. Пример: RTC (Real Time Clock).

Чтобы сделать устройство доступным для системы и пользовательских программ, нужно его зарегистрировать. В заголовочном файле `fs.h` определены функции для регистрации символьных и блочных устройств. Все они начинаются с префикса register. Наше устройство будет сим-

вольным, поэтому для его регистрации мы будем использовать функцию register_chrdev. Вот ее прототип:

```
extern int register_chrdev(unsigned int, const char *,
struct file_operations *);
```

Первый аргумент — это старший номер (*major number*) устройства, определяющий его тип. Если старший номер равен 0, то функция возвратит свободный старший номер для устройства нашего вида (символьное устройство).

Чтобы понять, для чего нужен старший номер, вспомним каталог `/dev`, содержащий файлы устройств. Файл `/dev/tty1` — это терминал с номером 1. Старший номер определяет тип устройства — терминал (tty), а младший — его номер в системе (1). Человеку проще работать не с номерами, а с символьными именами устройств, поэтому каждому старшему номеру соответствует символьное обозначение.

При регистрации устройства нужно указать его тип — старший номер устройства. Но для этого нужно знать, какие номера свободны. Проще всего указать первым аргументом 0 — тогда функция возвратит первый свободный старший номер символьного устройства для вашей системы. Если старший номер указать явно, может возникнуть конфликт номеров, и наше устройство не будет зарегистрировано.

Второй параметр определяет имя устройства («device»). Последний параметр очень важен. Это структура указателей на функции для работы с нашим устройством. Наш модуль содержит таблицу доступных функций, а операционная система вызывает нужную функцию, когда пользовательской программе нужно выполнить операцию с файлом устройства (открытие/закрытие, чтение/запись). Таблица функций символьного устройства хранится в структуре file_operations, которая передается при регистрации устройства.

После регистрации драйвера устройства происходит поиск устройств данного типа. Причем этот поиск должен произвести сам модуль. Для простоты будем считать, что у нас всего два устройства. Нам нужно создать эти два устройства, предварительно вычислив старший номер устройства. Напишем модуль, который помимо регистрации устройства выводил бы его старший номер — потом мы его будем использовать при создании устройства.

Листинг 28.6. Драйвер устройства /dev/device (без структуры file_operations)

```
#define MODULE
#define __KERNEL__

#include  <linux/module.h>
#include  <linux/init.h>
#include  <linux/kernel.h>
#include  <linux/fs.h>     // регистрация устройств
#include  <linux/ioport.h> // работа с портами ввода/вывода
#include  <linux/sched.h>  // резервирование прерывания

// Имя нашего устройства
#define DEV_NAME "device"

// Порты ввода-вывода нашего устройства
#define PORT_START 0x2000
#define PORT_QTY 10

// Память нашего устройства
#define MEM_START 0x20000000
#define MEM_QTY 0x20

// Номер прерывания для нашего устройства
#define IRQ_NUM 9

MODULE_AUTHOR("Denis Kolisnichenko dhsilabs@mail.ru");
MODULE_DESCRIPTION("Linux kernel module");

// Старший номер файла устройства
static int Major;

// Структура file_operations — пока пустая,
// но вскоре мы ее напишем
struct file_operations FO;

// Обработчик прерывания
void irq_handler(int irq, void *dev_id, struct pt_regs
*regs)
{
return;
}
```

```
int init_module()
{
// Регистрируем устройство
printk("My module: starting…\n");

Major = register_chrdev(0, DEV_NAME, &FO);

if (Major < 0) {
// Устройство не зарегистрировано
printk("My module: registration failed\n");
return Major;
}
printk("My module: device registered, major number = %d\n",Major);

// Резервирование портов ввода-вывода
printk("My module: allocating io ports\n");

if (check_region(PORT_START, PORT_QTY))
{
printk("My module: allocation io ports failed\n");
return -EBUSY;
}

request_region(PORT_START, PORT_QTY, DEV_NAME);
printk ("My module: io ports allocated\n");

// Резервирование памяти
if (check_mem_region(MEM_START, MEM_QTY))
{
printk("My module: memory allocation failed\n");
release_region(PORT_START, PORT_QTY);
return -EBUSY;
}

request_mem_region(MEM_START, MEM_QTY, DEV_NAME);
printk ("My module: memory allocated\n");

// Резервирование прерывания
if (request_irq(IRQ_NUM, irq_handler, 0, DEV_NAME, NULL))
{
printk("My module: IRQ allocation failed\n");
release_mem_region(MEM_START, MEM_QTY);
release_region(PORT_START, PORT_QTY);
return -EBUSY;
}
```

```
printk ("My module: IRQ allocated\n");
return 0;
}

void cleanup_module()
{
// Освобождаем порты ввода-вывода
release_region(PORT_START, PORT_QTY);
printk("My module: release io ports\n");

// Освобождаем память
release_mem_region(MEM_START, MEM_QTY);
printk("My module: release memory\n");

// Освобождаем прерывание
free_irq(IRQ_NUM,NULL);
printk("My module: release irq\n");

// Отменяем регистрацию устройства
if (unregister_chrdev(Major, DEV_NAME) < 0){
   printk("My module: cannot to unregister device\n");
}

printk("My module: device unregistered\n");
return;
}
```

При загрузке модуля вы увидите следующее сообщение:

```
My module: device registered, major number = 255
```

Конечно, кроме этого сообщения будут и другие, но нас они не интересуют. Почему именно это сообщение так важно для нас? В первой части сообщения говорится, что наше устройство успешно зарегистрировано, а во второй сообщается старший номер устройства, который мы будем использовать для создания устройств /dev/device0 и /dev/device1.

Вы не забыли, что нам еще нужно создать два устройства типа device, чтобы программы могли работать с ними? Перейдите в каталог `/dev` и от имени суперпользователя выполните команды:

```
# mknod device c 255 0
# mknod device c 255 1
```

Здесь 255 — это старший номер устройства (у вас он будет другим), 0 и 1 — младшие номера устройств. После выполнения данных команд будут созданы два файла устройств — `/dev/device0` и `/dev/device1`.

После регистрации устройства функцией register_chrdev() мы пытаемся захватить диапазон портов. Для этого предназначена функция request_region(), но перед ее вызовом мы должны убедиться, что нужный нам диапазон не используется (функция check_region()). Затем, если нужно, мы резервируем память для нашего устройства. Для резервирования памяти используется функция request_mem_region(), а для проверки возможности захвата памяти предназначена функция check_mem_region(). После успешной регистрации памяти можно попытаться захватить IRQ — функция request_irq().

Предположим, что на каком-то этапе регистрации модуля произошла ошибка. Если мы не смогли зарегистрировать порты ввода/вывода, вряд ли имеет смысл продолжать работу. Если же ошибка произошла при резервировании памяти, то перед завершением работы модуля нам нужно освободить порты ввода/вывода, которые мы зарегистрировали на предыдущем этапе. Аналогично поступаем при ошибке захвата IRQ — освобождаем порты и память. Функции `release_mem_region()`,`release_region` и `free_irq()` используются для освобождения памяти, портов и IRQ соответственно.

Обратите внимание: мы написали драйвер так, что он захватывает порты и память от имени одного устройства — DEV_NAME. В реальности все гораздо сложнее: нужно захватывать ресурсы для каждого устройства данного типа. К тому же придется предусмотреть поиск устройств модулем: в нашем случае мы знаем, что устройств только два, но у конечного пользователя таких устройств может быть больше или меньше, поэтому наш модуль не будет универсален, если он будет поддерживать только два устройства.

28.4. Операции над устройством. Поиск устройств

Наш модуль пока еще не может называться «драйвером» в прямом смысле этого слова: устройство-то он регистрирует, но не позволяет выполнить ни одной операции с ним — ведь структура file_operations пуста.

Кроме структуры file_operations нам еще понадобится структура для хранения информации о состоянии устройства, а так как устройств у нас два, то нужен также массив структур для хранения состояния каждого устройства. Индексами массива будут младшие номера устройств.

```
// Структура для хранения состояния устройства
struct device_state
{
```

```
int dev_open;        // 1 – устройство открыто, 0 - закрыто
ssize_t byte_read;   // Количество прочитанных
                     // из устройства байтов
ssize_t byte_write;  // Количество записанных байтов
};
// Массив для хранения информации о состоянии устройств
static struct device_state state[2];
```

В принципе, можно обойтись и без кода поиска устройств — без него модуль будет проще (а значит, надежнее), да и работать он будет быстрее. Обойти поиск устройств можно следующим образом. Мы не знаем, сколько устройств типа device будет у конечного пользователя. Поэтому вместо массива state (он будет описан ниже) нужно использовать динамический список, который будет содержать информацию о каждом устройстве типа device. При загрузке модуля список будет содержать всего один элемент — для устройства `/dev/device0`. Если устройств этого типа в системе нет вообще, будем просто считать, что устройство `device0` закрыто, а при попытке обращения к нему будем сообщать, что оно занято. По мере поступления запросов программ на открытие других устройств `/dev/deviceX` будем добавлять в наш список новые элементы.

Если же вам все-таки хочется узнать конкретное количество устройств /dev/deviceX, установленных у пользователя, можно просто просмотреть содержимое каталога /dev и посчитать количество файлов device*.

Все готово для того, чтобы написать функцию открытия устройства.

Листинг 28.7. Функция открытия устройства

```
static int device_open(struct inode *inode, struct file *fp)
{
struct device_state *dev_state;

printk("My module: try  to open device with minor number
%d\n", MINOR(inode->i_rdev));
dev_state = &state[MINOR(inode->i_rdev)];

if(dev_state->dev_open)
{
printk("Devise is busy\n");
return -EBUSY;
}
dev_state->dev_open = 1;
dev_state->byte_read = 0;
dev_state->byte_write = 0;
```

```
MOD_INC_USE_COUNT;

return 0;

}
```

Младший номер устройства мы получаем с помощью вызова MINOR (inode->i_rdev). Если устройство уже открыто, мы выводим сообщение: Devise is busy. В противном случае устанавливаем флаг открытия устройства, обнуляем byte_read и byte_write, а также увеличиваем счетчик использования данного модуля (MOD_INC_USE_COUNT).

Функция закрытия устройства сбрасывает флаг dev_open и уменьшает счетчик использования устройства.

Листинг 28.8. Функция закрытия устройства

```
static int device_close(struct inode *inode, struct file *fp)
{
struct device_state *dev_state;

printk("My module: try to close device with minor number
%d\n", MINOR(inode->i_rdev));

dev_state = &state[MINOR(inode->i_rdev)];
if(!dev_state->dev_open)
{
printk("Device is not open\n");
return 0;
}

dev_state->dev_open=0;
MOD_DEC_USE_COUNT;

return 0;
}
```

Теперь нам нужно указать ядру, какие функции нужно использовать для открытия и закрытия устройства:

```
struct file_operations FO =
{
open: device_open,
release: device_close
};
```

Полный код модуля устройства device вместе с функциями открытия и закрытия устройства, а также структурой file_operations приведен в следующем листинге:

Листинг 28.9. Модуль устройства device (module.c)

```
#define MODULE
#define __KERNEL__

#include <linux/module.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/fs.h>      // регистрация устройств
#include <linux/ioport.h>  // работа с портами ввода/вывода
#include<linux/sched.h> // резервирование прерывания

// Имя нашего устройства
#define DEV_NAME "device"

// Порты ввода-вывода нашего устройства
#define PORT_START 0x2000
#define PORT_QTY 10

// Память нашего устройства
#define MEM_START 0x20000000
#define MEM_QTY 0x20

// Номер прерывания для нашего устройства
#define IRQ_NUM 9

MODULE_AUTHOR("Denis Kolisnichenko dhsilabs@mail.ru");
MODULE_DESCRIPTION("Linux kernel module");

// Старший номер файла устройства
static int Major;

// Структура file_operations - пока пустая, но вскоре мы
ее напишем
struct file_operations FO
{
open: device_open,
release: device_close
};
```

```
// Структура для хранения состояния устройства
struct device_state
{
int dev_open;      // 1 – устройство открыто, 0 - закрыто
ssize_t byte_read;  // Количество прочитанных байтов из
устройства
ssize_t byte_write;    // Количество записанных байтов
};
// Массив для хранения информации о состоянии устройств
static struct device_state state[2];


// Обработчик прерывания
void irq_handler(int irq, void *dev_id, struct pt_regs
*regs)
{
return;
}

int init_module()
{

// Регистрируем устройство
printk("My module: starting…\n");

Major = register_chrdev(0, DEV_NAME, &FO);

if (Major < 0) {
// Устройство не зарегистрировано
printk("My module: registration failed\n");
return Major;
}
printk("My module: device registered, major number = %d\
n",Major);

// Резервирование портов ввода-вывода
printk("My module: allocating io ports\n");

if (check_region(PORT_START, PORT_QTY))
{
printk("My module: allocation io ports failed\n");
return -EBUSY;
}

request_region(PORT_START, PORT_QTY, DEV_NAME);
printk ("My module: io ports allocated\n");
```

```
// Резервирование памяти
if (check_mem_region(MEM_START, MEM_QTY))
{
printk("My module: memory allocation failed\n");
release_region(PORT_START, PORT_QTY);
return -EBUSY;
}

request_mem_region(MEM_START, MEM_QTY, DEV_NAME);
printk ("My module: memory allocated\n");

// Резервирование прерывания
if (request_irq(IRQ_NUM, irq_handler, 0, DEV_NAME, NULL))
{
printk("My module: IRQ allocation failed\n");
release_mem_region(MEM_START, MEM_QTY);
release_region(PORT_START, PORT_QTY);
return -EBUSY;
}
printk ("My module: IRQ allocated\n");
return 0;
}

void cleanup_module()
{
// Освобождаем порты ввода-вывода
release_region(PORT_START, PORT_QTY);
printk("My module: release io ports\n");

// Освобождаем память
release_mem_region(MEM_START, MEM_QTY);
printk("My module: release memory\n");

// Освобождаем прерывание
free_irq(IRQ_NUM,NULL);
printk("My module: release irq\n");

// Отменяем регистрацию устройства
if (unregister_chrdev(Major, DEV_NAME) < 0){
   printk("My module: cannot to unregister device\n");
}

printk("My module: device unregistered\n");
return;
}
```

```
static int device_open(struct inode *inode, struct file *fp)
{
struct device_state *dev_state;

printk("My module: try  to open device with minor number
%d\n", MINOR(inode->i_rdev));
dev_state = &state[MINOR(inode->i_rdev)];

if(dev_state->dev_open)
{
printk("Devise is busy\n");
return -EBUSY;
}
dev_state->dev_open = 1;
dev_state->byte_read = 0;
dev_state->byte_write = 0;

MOD_INC_USE_COUNT;
return 0;
}

static int device_close(struct inode *inode, struct file
*fp)
{
struct device_state *dev_state;

printk("My module: try to close device with minor number
%d\n", MINOR(inode->i_rdev));

dev_state = &state[MINOR(inode->i_rdev)];
if(!dev_state->dev_open)
{
printk("Device is not open\n");
return 0;
}

dev_state->dev_open=0;
MOD_DEC_USE_COUNT;

return 0;
}
```

Теперь модуль для абстрактного устройства device готов. Вы можете написать небольшую программку, которая пыталась бы выполнить операции с нашим устройством: открыть его и закрыть — других операций мы не определили. Для определения других действий используется та же структура file_operations. Листинг 28.10 показывает, как она объявлена в файле `/usr/src/linux-2.4/include/linux/fs.h`.

Листинг 28.10. Фрагмент файла /usr/src/linux-2.4/include/linux/fs.h

```
struct file_operations {
 struct module *owner;
 loff_t (*llseek) (struct file *, loff_t, int);
 ssize_t (*read) (struct file *, char *, size_t, loff_t *);
 ssize_t (*write) (struct file *, const char *,
                   size_t, loff_t *);
 int (*readdir) (struct file *, void *, filldir_t);
 unsigned int (*poll) (struct file *,
                struct poll_table_struct *);
 int (*ioctl) (struct inode *, struct file *, unsigned
int, unsigned long);
 int (*mmap) (struct file *, struct vm_area_struct *);
 int (*open) (struct inode *, struct file *);
 int (*flush) (struct file *);
 int (*release) (struct inode *, struct file *);
 int (*fsync) (struct file *, struct dentry *,
             int datasync);
 int (*fasync) (int, struct file *, int);
 int (*lock) (struct file *, int, struct file_lock *);
 ssize_t (*readv) (struct file *, const struct iovec *,
unsigned long, loff_t *);
 ssize_t (*writev) (struct file *, const struct iovec *,
unsigned long, loff_t *);
 ssize_t (*sendpage) (struct file *, struct page *, int,
size_t, loff_t *, int);
 unsigned long (*get_unmapped_area)(struct file *,
                unsigned long, unsigned long,
                unsigned long, unsigned long);
};
```

Как использовать структуру file_operations, думаю, ясно. Например, нам нужно описать обработчики записи и чтения устройства — функции device_write() и device_read():

```
struct file_operations FO =
{
open: device_open,
release: device_close
read: device_read,
write: device_write
};
```

Обработчики чтения и записи пишутся «по образу и подобию» обработчиков открытия и закрытия устройства, то есть сначала нам нужно определить младший номер с помощью вызова MINOR(), а затем произвести операцию с устройством.

Приложение

ТАБЛИЦЫ СООТВЕТСТВИЯ WINDOWS- И LINUX-ПРОГРАММ

- РАБОТА В ИНТЕРНЕТ
- РАБОТА С ФАЙЛАМИ
- ПРИКЛАДНЫЕ И СИСТЕМНЫЕ ПРОГРАММЫ
- ОФИСНЫЕ ПРИЛОЖЕНИЯ
- МУЛЬТИМЕДИА
- РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ
- СУБД
- КОМПИЛЯЦИЯ МОДУЛЯ
- МАТЕМАТИЧЕСКИЕ ПАКЕТЫ
- ИГРЫ

С помощью приведенных ниже таблиц вы сможете подобрать достойный аналог для вашей Windows-программы. При заполнении таблицы я руководствовался простым правилом: чтобы Windows-программа и ее Linux-аналог были в одной весовой категории, то есть обладали примерно одинаковыми возможностями.

Работа в Интернет

Windows-программа	Linux-Аналог	Комментарий
Internet Explorer	Netscape Navigator/Mozilla	Думаю, что обе эти программы в представлении и особых комментариях не нуждаются — мы их знаем еще со времен Windows
	Konqueror	Вряд ли Завоеватель сможет выступить в роли Internet Explorer'а. Но для походов на не очень «навороченные» (с использованием Flash, VRML, апплеты Java) сайты его возможностей вполне хватит
	Galeon	Данный браузер основан на движке Mozilla, поэтому он с успехом может заменить самого Mozill'y!
	IE for Linux	Компания Microsoft портировала программу IE на платформу Linux. Честно говоря, «ослика ИА для Линукс» я в глаза не видел, но судя по отзывам, программа линуксоидам не понравилась
Opera	Opera for Linux	Уже вышла восьмая версия этого популярного браузера, и она доступна пользователям Linux. Но рекомендуется воспользоваться стабильной версией 7.54 — это проверенная временем, надежная версия
Mozilla for Windows	Mozilla for Linux	Linux-аналог обладает всеми возможностями Windows-программы
Outlook	Ximian Evolution	Полный аналог знаменитой программы Outlook — программы похожи друг на друга как две капли воды
	K Mail	K Mail вполне справится с возложенной на него задачей — служить заменителем для Outlook. Организация учетных записей K Mail очень напоминает Outlook, что делает эти две программы несколько похожими друг на друга. Конечно, K Mail и Outlook не очень похожи внешне, но идея остается прежней. Нужно заметить, что K Mail не заменит полную версию Outlook, скорее всего, он «потянет» на роль аналога Outlook Express
Outlook (чтение новостей)	Knode	Программа обладает достаточно удобным интерфейсом, что позволяет ставить ее в один ряд с клиентом новостей программы Outlook
	Netscape Messenger	Messenger — это более зрелый, по сравнению с Knode, продукт, позволяющий читать не только почту по протоколам POP3 и IMP3, но и новости

Windows-программа	Linux-Аналог	Комментарий
The Bat!	Sylpheed	Программа The Bat! давно стала культовой, а ее разработчики, видно, не спешат портировать программу на платформу Linux. К счастью, мир не без добрых людей: разработчик Hiroyuki Yamamoto (скорее всего, японец) написал Linux-аналог программы The Bat! Программа Sylpheed заслуживает отдельного разговора, поэтому о ней мы поговорим немного позже
The Bat!	K Mail	Программа K Mail тоже может выступать в роли аналога для программы The Bat!, но с меньшим успехом, чем программа Sylpheed
FlashGet, Reget, Go!zilla менеджеры закачки файлов)	Downloader for X	Мне очень понравилась эта программа — по своим возможностям она не уступает ни одной из перечисленных Windows-программ
	Kget (или Caitoo)	Довольно неплохой загрузчик файлов
	Wget	Данная программа работает в консоли, то есть не имеет графического интерфейса, но возможности данной программы заслуживают уважения
Teleport Pro (полная загрузка сайта)	Downloader for X	Программа Downloader for X может использоваться как для загрузки одного файла, так и для закачки целых сайтов
	Wget	То же самое можно сказать и о программе wget
FTP-клиенты (Bullet Proof FTP)	1. Gftp 2. lfgp 3. ncftp	Первые две программы более удобны, поскольку они являются графическими приложениями. Третья программа — ncftp — работает в консоли, но обладает довольно полезными функциями, например, докачка файла в случае разрыва соединения или закачка файлов по шаблону
ICQ, MSN, AIM	1. licq 2. aicq 3. micq 4. Gaim 5. Simple Instant messenger	Существует много разновидностей ICQ-клиента. Вы можете выбрать один из них на свой вкус
Mirc (IRC-клиент)	Xhat, Kvirc, Ksirc	С IRC-клиентами также никогда не было проблем — это только три наиболее удобные на мой взгляд IRC-клиента
Firewall (BlackICE, ATGuard, ZoneAlarm)	Iptables или ipchains	Обе эти программы являются стандартными и устанавливаются при установке любого дистрибутива Linux. Использовать графические оболочки для iptables или ipchains (Kmyfirewall, Firewall Builder) я не рекомендую, поскольку ни одна оболочка не позволяет гибко настроить firewall
VisualRoute, CyberKit	Xtraceroute, VisualRout for Linux, traceroute, ping, tcpdump	Программу VisualRoute с успехом можно заменить программой Xtraceroute или ее текстовой версией traceroute. Пакет CyberKit можно заменить набором стандартных программ — traceroute, ping, nslookup и tcpdump: полученная смесь будет мощнее, чем CyberKit
Фильтрация данных и роутинг (BlackICE, ATGuard, WinRoute, WinGate)	Squid и route	Обе эти программы являются стандартными и обладают значительно большими возможностями, чем их Windows-аналоги
VentaFax	efax Gfax	Эти программы смогут заменить знаменитую программу VentaFax
Vdialer, E-type dialer (дозвон к провайдеру)	Kppp Gppp Kinternet	Звонить к провайдеру Linux также умеет — не сомневайтесь, ведь Linux создан для сети, как птица для полета

Windows-программа	Linux-Аналог	Комментарий
Загрузка канала	kpppload kisdnload	Первая программа отображает график загрузки PPP-канала, а вторая — канала ISDN
Remote Administrator	telnet ssh X-terminal	В отличие от Windows, где для удаленного администрирования нужно устанавливать отдельную программу, в Linux есть стандартные средства, с помощью которых можно управлять удаленной машиной
HyperTerminal	minicom	Небольшая программа с большими возможностями

Работа с файлами

Windows-программа	Linux-Аналог	Комментарий
Norton Commander, Volcov Commander, FAR	Midnight Commander	Программа Midnight Commander полностью заменяет популярную программу NC. Интерфейс программ практически одинаковый: MC, как и NC, обладает двумя панелями синего цвета и работает в текстовом режиме. Так как MC — это файловый менеджер Unix (Linux), то он обладает специфическими для этой ОС функциями: операции с правами доступа, создания символических ссылок, поддержка сети (NFS, FTP)
Windows Commander (Total Commander)	Midnight Commander	Программа MC обладает всеми функциями указанных Windows-программ. Правда, в отличие от Windows Commander, MC работает в текстовом режиме. Хотя имеется возможность запуска MC в терминале X: при этом программа будет реагировать на события X (например, щелчок мыши на кнопке Копировать)
	LinCommader	Удобный файловый менеджер. Программа доступна по адресу http://www.ussr.to/Russia/vvi/lcmd.html
	Krusader	Программа обладает удобным интерфейсом и предназначена для работы в KDE. Поддерживает различные форматы архивов, обладает встроенными FTP-клиентом и просмотрщиком графических файлов. Программа доступна по адресу http://krusader.sourceforge.net/

Прикладные и системные программы

Windows-программа	Linux-Аналог	Комментарий
Блокнот	kedit, gedit, xedit	Любая из этих программ в состоянии заменить Блокнот
Bred, Rpad32, Aditor	Kate	Программа Kate поддерживает 38 различных кодировок и подсветку синтаксиса для C, C++, Java, Pascal, PHP, HTML, Bash и других языков программирования
Программы-переводчики (Prompt)	Пока нет ни одного нормального переводчика	
Словари (Socrat, Lingvo)	Mueller, Mova, Ksocrat	Любой из этих словарей способен заменить Lingvo, хотя я не обещаю, что замена будет равноценной

Windows-программа	Linux-Аналог	Комментарий
RAR/WinRAR	RAR for Linux	Версия популярного архиватора для Linux – комментарии излишни
	Ark	Программа входит в состав утилит KDE (пакет kdeutil) и поддерживает распространенные форматы архивов
WinZIP	Ark	Все тот же Ark
	GnoZIP	Удобный архиватор для среды GNOME. Программу можно качать по адресу http://www.geocities.com/SiliconValley/9757/gnozip.html
PGP	PGP for Linux	Полный аналог Windows-версии
DrWeb, AVP	DrWeb for Linux, AVP for Linux	Программы доступны по адресам http://www.drweb.ru/unix/ и http://www.kaspersky.ru/, соответственно
Msconfig	DrakConf, LinuxConf (в Mandrake) setup (в Red Hat) redhat-config-XXXX (в RH 8.0)	Думаю, с конфигураторами проблем у вас не будет
System Commander, PowerQuest Boot Magic	LILO Grub ASPLoader Acronis OS Selector	Первые два загрузчика являются стандартными, и вы можете использовать любой из них на свой вкус. Третий — это стандартный загрузчик ОС ASP Linux, а последний — это альтернативный загрузчик, доступный по адресу http://www.acronis.ru/
Диспетчер задач	ps, Top, Gtop, Ktop,...	Без комментариев
MBMonitor, PCAlert (измерение температур и напряжений на материнской плате)	KHealthCare (для KDE)	Программа не входит в состав KDE, поэтому ее нужно акачать самостоятельно http://homepages.fh-giessen.de/~hg7229/khealthcare/main.html
PowerQuest Partition Magic	DiskDrake	Полный аналог PQ Partition Magic. Входит в состав Mandrake и Mandrake-подобных дистрибутивов
	FIPS	Поставлялся на компакт-диске с дистрибутивами компании Red Hat до версии 9.0. Сейчас эту программу можно найти в Интернете, но стоит ли? Ведь она уже не развивается. Поэтому лучше воспользуйтесь программой GNU Parted — очень, очень удобная программа.
	GNU Parted	Программа доступна по адресу http://www.gnu.org/software/parted/parted.html
Мониторинг S.M.A.R.T-атрибутов винчестера	Smartsuite Hddtemp-0.3 IDEload-0.2 Ide-smart Smartsuite-2.1 Smartmontools	Оценить работу данных программ не смог по техническим причинам — мой винчестер не относится к разряду «умных» (smart)
Fdisk	Fdisk	Практически каждая ОС обладает своим аналогом программы fdisk — ОС Linux не является исключением из правил
Scandisk	fsck	Программа fsck умеет проверять не только файловые системы ext2 и ext3, но и любые другие при наличии плагина файловой системы

Офисные приложения

Windows-программа	Linux-Аналог	Комментарий
MS Office	Open Office	Данный пакет будет достойной заменой офисному стандарту де-факто — MS Office. Надеюсь, вы уже успели в этом убедиться
	Star Office	Это коммерческий вариант Open Office. Точнее, Open Office — это бесплатный вариант Open Office. Не знаю почему, но мне больше нравится Open Office. Нет, не потому, что он бесплатный, а какой-то более дружественный и более «шустрый»
	K Office	Скорее всего, данный пакет не сможет удовлетворить все ваши запросы, особенно, если вы до этого работали с MS Office
Adobe Photoshop	GIMP	Программу GIMP можно с успехом использовать в непрофессиональных целях вместо Photoshop. Почему в непрофессиональных? GIMP немного «прихрамывает» при калибровке цветов и размеров, поэтому в профессиональной полиграфии его использование чревато небольшими (или большими) неприятностями
Adobe Acrobat	K Office, TeX, LyX,...	Многие Linux-программы, работающие с документами, могут конвертировать их в формат PDF
Adobe Acrobat Reader	Xpdf	Данная программа (версия 0.9) поддерживает форматы PDF 1.3, 1.4 (Acrobat 4, 5, соответственно). Xpdf присутствует во всех дистрибутивах, поэтому вам не придется ее долго искать в Интернет — она уже установлена в вашей системе
	Acrobat Reader for Linux	Тут уж вообще комментарии излишни — компания «Adobe» давно выпускает версии Acrobat'a (начиная с версии 2.0) для Linux и других Unix-платформ
Corel Draw!	Corel Draw for Linux	Как вы, наверное, знаете, компания Corel давно выпустила версию Corel Draw для Linux. Честно говоря, мне программа не понравилась. Во-первых, программа довольно медленна, потому что она запускается посредством эмулятора Wine и ее нельзя назвать Linux-программой в полном смысле этого слова.Во-вторых, мне очень не понравилось, как программа работает с русскими шрифтами — русский язык больше похож на китайские иероглифы. Причем программа поддается настройке только в операционной системе Corel Linux. В-третьих, программа очень не стабильна. Исходя из всего этого, позвольте мне отрекомендовать вам следующую программу
	Open Draw	Данная программа входит в состав пакета Open Office. Возможно, она не заменит Corel Draw полностью, однако если выбирать между Corel Draw for Linux и Open Draw, я бы выбрал последнее
	Karbon	Неплохой векторный редактор, появившийся в KDE 3.3 (до этого основным векторным редактором KDE был K Illustrator). Программа проста и даже проще, чем OO Draw, поэтому рассчитывать на ее функциональность не приходится. Однако попробуйте поработать с программой – вдруг вам понравится
Page Maker	TeX, LateX	Издательская система TeX, разработанная Д. Кнутом, является одной из самых лучших издательских систем. Единственный ее недостаток — это отсутствие графического интерфейса пользователя. Хотя, этот недостаток иногда перерастает в достоинство — программу можно использовать даже на стареньких 386-х компьютерах
	Scribus	Данная программа обладает интерфейсом пользователя, системой управления цветом (Color Management System), достаточно удобна в использовании

Windows-программа	Linux-Аналог	Комментарий
Композеры и HTML-редакторы	Quanta Plus	Очень рекомендую данный HTML-редактор. С помощью Quant'ы вы не только отредактируете вашу страничу и обновите ее на вашем сервере, но и проверите правописание, а также соответствие написанного коду HTML-стандарта
Fine Reader	Kooka	Работа программы Kooka оставляет желать лучшего. К тому же Kooka не является OCR, а только сканирует документ. Я бы вам посоветовал запускать Fine Reader через Vm Ware или Win4Lin. Кроме программы Kooka имеется программка Gocr, но она умеет распознавать только английский текст
3D Max	Innovation 3D	Innovation в полной мере не заменит вам 3D Max, но можно надеяться, что в скором времени, это случится — в программу Innovation 3D можно добавлять плагины
ACDSee	GQView	GQView послужит равноценной заменой популярному просмотрщику графических файлов — ACDSee
MS Money	GNUcash	Небольшая программка, позволяющая управлять своими финансами
Финансы без проблем	Финансы без проблем для Linux	Портированная версия «Финансов» для Linux. Программа доступна на компакт-диске
1С: Бухгалтерия	Ananas	Не следует надеяться, что Ananas полностью заменит Бухгалтерию от 1С. Программу можно скачать по адресу http://ananas.linux.ru.net/
AutoCAD	Kivio, QCAD, Varcon,...	В Интернете вы найдете не только эти программы, но и сопровождающую их документацию – поэкспериментируйте с ними, возможно, для себя вы найдете доступную замену AutoCAD

Мультимедиа

Windows-программа	Linux-Аналог	Комментарий
AudioGraber	Grip cdda2wav	Программа Grip более функциональна, чем AudioGraber, а MP3-кодек lame обеспечивает качественное сжатие музыки в формате MP3
WinAMP	XMMS	Программа XMMS — это полный аналог программы WinAmp, XMMS даже поддерживает скины WinAMP
Проигрыватель Audio CD	KsCD	Программа является стандартным проигрывателем компакт-дисков в Linux
Windows Media Player	Xine	Xine поддерживает форматы видео MPEG, MP4, DVD, AVI (Video for Windows)
Nero, Easy CD Creator	Xcdroast KonCd cdrecord	С помощью этих программ вы сможете записать собственный CD. Подробно о процессе записи CD вы сможете прочитать в этой книге
Преобразование DVD в DivX	dvdrip	Программа позволяет преобразовать один DVD-диск в несколько дисков формата DivX
Flash Player	Плагины для Netscape/Mozilla/ Konqueror	Браузеры Netscape/Mozilla/Konqueror обладают встроенными плагинами для просмотра Flash

Windows-программа	Linux-Аналог	Комментарий
Macromedia Fireworks	GIMP	Программа Fireworks с успехом заменится тем же GIMP'ом
Main Actor	Main Actor for Linux	Для программы Main Actor давно создан ее полноценный аналог для Linux. Программа ничем не уступает аналогичной Windows-программе. Main Actor for Linux может также послужить заменой программе Windows Movie Maker
Программы просмотра телепередач через TV-тюнер	Xawtv GnomeTV KwinTV	Программа Xwatv является стандартной и обладает большими возможностями по сравнению с GnomeTV и KwinTV. Xawtv рассматривается в главе 17 «Видео и видеомонтаж в Linux»
RealPlayer	RealPlayer for Linux	Портированная версия RealPlayer. Скачать можно здесь: http://scopes.real.com/real/player/unix/unix.html
QuickTime Player	QuickTime Player для Linux	Полный аналог Windows-версии
Sound Forge	Wave Forge, Sox	Данные программы можно скачать по адресам http://www.tfm.ro/waveforge/ и http://home.sprynet.com/~cbagwell/sox.html соответственно
Lame (MP3-кодек)	Lame for Linux	Портированная версия

Разработка программного обеспечения

Windows-программа	Linux-Аналог	Комментарий
Delphi	Kylix и Kylix Personal Edition	*Kylix Personal Edition является бесплатной программой*
Pascal	Freepascal	Свободнораспространяемый 32-битный компилятор
TurboPascal	Freepascal + Motor	Motor — это редактор с подсветкой синтаксиса, подобный тому, который используется в среде Turbo Pascal. Motor доступен по адресу http://konst.org.ua/motor
Turbo C	GCC + Motor	GCC — это the GNU C Compiler
Borland C++ Builder	Qt3 Designer + Kdevelop	В полной степени Qt3 Designer все же заменить C Builder не может, но написать небольшое приложение с использованием библиотеки Qt, можно
	Code Forge	*Среда разработки приложений*
Basic	Hbasic	Любителям Бейсика рекомендую заглянуть на страничку http://hbasic.sourceforge.net/
	SmallBasic	Если Hbasic вам не понравится, загляните сюда http://smallbasic.sourceforge.net/
VisualProlog	GNU Prolog	Программа доступна по адресу http://pauillac.inria.fr/~diaz/gnu-prolog
Turbo Assembler	NASM	Программа входит в состав многих дистрибутивов. Домашняя страничка — http://www.web-sites.co.uk/nasm
FrontPage	Mozilla Composer	Композер Mozilla поддерживает принцип WYSIWYG
J Builder	J Builder for Linux	Портированная версия
Hiew	Biew	Программа доступна по адресу http://biew.sourceforge.net/
PHP	PHP for Linux	Входит в состав любого дистрибутива

СУБД

Windows-программа	Linux-Аналог	Комментарий
MS Access	Open Office + MySQL	Open Office можно использовать в качестве графического интерфейса для СУБД MySQL. Полученная смесь будет более мощная, чем Access
	GDBM	Возможно, в вашем дистрибутиве будет СУБД GDBM
	IBM DB2 for Linux	Заменять Access СУБД от IBM — это все равно, что с пушки по воробьям палить. Но все-таки, IBM DB2 обладает удобным клиентом, написанным на Java. Единственный недостаток — графическая версия клиента, в отличие от текстовой, притормаживает — сказывается кроссплатформенность
InterBase Server	InterBase Server for Linux	Портированная версия
IBM DB2	IBM DB2 for Linux	Портированная версия. Версия для Linux, как и для Windows, платная
MySQL for Windows	MySQL for Linux	Вот сейчас я немного сомневаюсь, какая программа была разработана раньше: или Windows-версия, которая была портирована на Linux, или наоборот. Лично я сначала увидел Linux-версию, а потом уж версию для Windows. В любом случае замена есть
Oracle	Oracle for Linux	Версия для Linux, как и для Windows, платная

Математические пакеты

Windows-программа	Linux-Аналог	Комментарий
Maple	Maple for Linux	Существует версия Maple для Linux
MathCAD	Gap	По своим функциям программа Gap приближается к популярному математическому пакету
Редактор формул	OO Math	OO Math входит в состав офисного пакета Open Office
Statistica	Probability and Statistics Utilities for Linux users	Программа доступна по адресу http://www.geocities.com/SiliconValley/Network/6885/
	Gnumeric	Я, конечно, далек от вопросов математической статистики, но мне показалось, что программа Gnumeric сможет помочь решить задачи в этой области

Игры

Windows-программа	Linux-Аналог	Комментарий
Quake 1, 2, 3	Quake 1, 2, 3 for Linux	Портированные версии. Для работы этих версий нужны некоторые файлы Windows-версии
DOOM	LxDOOM, DOOM Legecy	Программы доступны по адресам http://lxdoom.linuxgames.com/, http://doomlegacy.sourceforge.net/
CounterStrike	CS Linux	Прошу извинения, но я не любитель игр, поэтому оценить программу не смог
Return to Castle Wolfenstein	Return to Castle Wolfenstein для Linux	Портированная версия. Подробнее можно прочитать здесь: http://www.activision.com/games/wolfenstein
Urban Terror	Urban Terror для Linux	Портированная версия
Unreal Tournament 2003	Unreal Tournament 2003 for Linux	Портированная версия
Civilization	FreeCiv	Довольно неплохая игрушка, очень похожая на Civilization
StarCraft	FreeCraft	Скачать можно здесь http://freecraft.org/
Стандартные игры Windows	Игры KDE (пакет kdegames)	Эти игры понравятся любителям небольших игрушек, помогающих «убить» время
Эмулятор Sony PlayStation (ePSXe for Windows)	ePSXe for Linux	Портированная версия

Издательство «Наука и Техника»

Д.Н. Колисниченко

LINUX-сервер своими руками

ISBN: 5-94387-103-9
Размер: 165x235
Объем: 752 с.: ил.

Издание 3-е, переработанное и дополненное

Вы держите в руках третье издание книги, ставшей бестселлером в своей области. В ней вы найдете подробное описание настроек сетевых сервисов, позволяющих на основе ОС Linux создать сервер требуемой конфигурации и функциональности. Узнаете, как должен быть организован Linux-сервер для выполнения тех или иных задач. Ознакомитесь с практическими примерами настройки. Научитесь администрировать Linux. Благодаря этой книге вы сможете настроить сервер любого типа: от сервера локальной сети до Интернет-сервера и сервера удаленного доступа.

Во третьем издании книга была немного дополнена и существенно обновлена (в соответствии с появлением новых дистрибутивов Mandrake, Fedora Core). Более широко стало рассмотрено создание Linux-сервера для Windows-сетей. Книга подойдет как для профессиональных, так и для начинающих администраторов, поскольку изложение материала начинается с установки ОС Linux, а в первой главе дано описание основных сетевых технологий и протоколов (курс молодого администратора).

В одном из множества благодарных отзывов по первому изданию говорится: «Наконец-то нашел книгу, в которой все понятно и толково описано». И этими словами можно выразить общее мнение читателей о ней. Так что если вам нужно настроить сервер на основе Linux, узнать его внутренний мир — эта книга для вас.

серия — *Секреты Мастерства* — серия

Издательство «Наука и Техника»

Д.Н. Колисниченко

Самоучитель LINUX.
Установка, настройка, использование

ISBN: 5-94387-193-4
Размер: 165x235
Объем: 688 с.: ил.

Издание 4-е, переработанное и дополненное

Данная книга является самым полным руководством пользователя Linux. Предназначена для всех, кто хочет эффективно использовать у себя на компьютере эту систему. Изложение материала происходит последовательно, начиная с этапа установки и заканчивая специальными технологиями Linux. Подробно описаны внутреннее устройство Linux, настройка оборудования и оптимизация системы. Обсуждаются возможные проблемы. Особое внимание уделено защите и восстановлению Linux после сбоев.

В книге детально рассмотрен весь круг задач, который может стоять перед пользователем: от редактирования текста, работы в Интернет и запуска Windows-игр, до компилирования ядра, администрирования системы и записи компакт-дисков. Учитываются особенности разных дистрибутивов. Подробно описаны мультимедиа-инструменты Linux: фотомонтаж, просмотр и редактирование видео, настройка TV-тюнера, обработка звука и многое другое. Рассмотрено множество программ и приложений под Linux: OpenOffice, KOffice, The Gimp, MainActor, WineX, VMWare, Lan Billing и т.д.

Автор книги — опытный и известный консультант по использованию Linux. Книга написана простым и понятным языком. Лучший выбор для начинающих линуксоидов.

серия — ПРОСТО О СЛОЖНОМ — серия

Издательство «Наука и Техника»

М.В. Юдин,
А.В. Куприянова

Самоучитель работы на НОУТБУКе

ISBN: 5-94387-241-8
Размер: 165x235
Объем: 512 с.: ил., цв. вклейки

Издание 2-е, стереотипное

Эта книга позволяет освоить работу на ноутбуке «с нуля», даже без каких-либо предварительных компьютерных навыков. Начинается изложение с детальных правил пользования ноутбуком, правильного ухода за ним, а также общего описания его внутреннего и внешнего устройства. Далее приводится описание стандартных компьютерных технологий (Windows XP, Word, Excel, Интернет, Электронная почта) с учетом особенностей работы на ноутбуке.

«Изюминкой» книги являются следующие разделы: подключение ноутбука к телевизору и использование его в качестве универсального CD/DVD-плеера, домашний кинотеатр на базе ноутбука, сотовый телефон + ноутбук = мобильный Интернет (как подключить к ноутбуку сотовый телефон и настроить выход в Интернет через него (настройка GPRS-соединения)), цифровая фотостудия на основе ноутбука (подключение цифрового фотоаппарата и работа с цифровыми фотографиями), контроль и правильная эксплуатация аккумуляторных батарей ноутбука, а также многое другое.

Книга написана простым и доступным языком. Содержит множество наглядных иллюстраций. Лучший выбор для начинающих.

серия — ПРОСТО О СЛОЖНОМ — серия

Книги ПОЧТОЙ

Издательство «Наука и Техника» принимает заказы на продажу собственной печатной продукции по почте наложенным платежом.

Оплата производится на почте при получении книг, для организаций возможна оплата по безналичному расчету после выставления счета. Для этого Вам необходимо оформить бланк заказа и отправить его нам.

Для жителей России:
193029 Санкт-Петербург, а/я 44,
ООО «Наука и Техника»
тел/факс (812)-567-70-26, 567-70-25
E-mail: nit@mail.wplus.net

Для жителей Украины:
02166 Киев-166, ул. Курчатова, 9/21,
«Наука и Техника»
тел/факс (044)-516-38-66
E-mail: nits@voliacable.com

Перечень рассылаемых книг размещен на сайте издательства: www.nit.com.ru

Жители Украины могут приобрести книги других издательств:

Радиоэлектроника..... ☐
Компьютерные технологии..... ☐
Техника безопасности на предприятиях ..... ☐
Транспорт ..... ☐

Примечание. Заказанные бесплатные каталоги высылаются по электронной почте или вкладываются в посылку с книгами ***(отдельно письмом каталоги не рассылаются)***.

ЗАКАЗ НА КАТАЛОГИ

Заполняйте поля аккуратно большими отдельными буквами.

Информация для приобретения книг почтой частными лицами

1. Фамилия, имя, отчество _____
2. Почтовый адрес: индекс _____ страна _____
область _____ город, поселок _____
улица _____ дом _____ корпус _____ кв. _____
телефон (_____) _____
адрес электронной почты (если он у Вас есть) : E-mail: _____

Информация для выставления счета организациям

Название _____ ИНН _____
Телефон/факс (_____) _____
Контактное лицо _____
Адрес для отправки заказа: индекс _____ страна _____
область _____ город, поселок _____
улица _____ дом _____
адрес электронной почты (если он у Вас есть) : E-mail: _____

БЛАНК ЗАКАЗА (принимаются ксерокопии)

Автор	Название	Цена Россия (руб.)	Цена Украина (грн.)	Год	Объем	Заказ (экз.)
Серия: Компьютерная шпаргалка						
Егоров	МиниЖелтые страницы Интернет. Компьютерная шпаргалка	16	4	2006	80	
Егоров	Поиск в Интернет. Компьютерная шпаргалка	16	4	2006	80	
Золотарева	Электронная почта. Компьютерная шпаргалка	16	4	2006	80	
Колосков	Microsoft Windows XP. Компьютерная шпаргалка	16	4	2006	80	
Кузнецова	Microsoft Word 2003: работаем с текстом. Компьютерная шпаргалка	16	4	2006	80	
Матвеев	Вычисления и расчеты в Excel 2003. Комп. шпаргалка	16	4	2006	80	
Юдин	Microsoft Excel 2003: работаем с таблицами. Компьютерная шпаргалка	16	4	2006	80	
Серия: Просто о сложном						
Алешков	Программы-переводчики. Осваиваем сами	54	12	2005	144	
Антоненко	«Толстый» самоучитель работы на компьютере	131	29	2005	544	
Воробьев	Nero Buruing ROM. Записываем CD и DVD	54	10	2005	192	
Герасимов	AutoCAD 2002. Популярный самоучитель	135	28	2004	496	
Главенка	45 лучших программ для созд., копир., обработки CD и DVD	143	32	2005	368	
Дмитриев	Настройки BIOS. Изд. 2-е	76	17	2005	288	
Жарков	AutoCAD 2004. Эффективный самоучитель. Изд. 2-е	164	37	2005	560	
Жарков	AutoCAD 2005: эффективный самоучитель	173	39	2005	600	
Колисниченко	Самоучитель PHP 5. 2-е издание	169	38	2005	576	
Колисниченко	Самоучитель LINUX. Установка, настр., использ. 4-е изд.	***	***	2006	688	
Колисниченко	Сделай сам комп. сеть. Монтаж, настройка, обслуж.	127	28	2004	400	
Колосков	Windows XP. Популярный самоучитель. 2-е изд.перер. и доп.	108	24	2005	368	
Кузнецова	Установка и переустановка Windows. 4-е изд.	43	10	2005	128	
Кузнецова	Microsoft Windows XP. Краткое руководство	63	14	2005	256	
Марек	Ассемблер на примерах. Базовый курс	109	24	2005	240	
Матвеев	Самоучитель MS Windows XP. Все об использ. и настр. 4-е изд.	***	***	2006	624	
Пестриков	Турбо Паскаль 7.0. Изучаем на примерах. Изд. 2-е	109	22	2004	368	
Подольский	Печать на ПК слепым десятипальцевым методом. 3-е изд.	***	***	2005	96	
Пономарев	Самоучитель работы на компьютере + **цв.вклейки**	106	24	2005	368	
Серогодский	Excel 2003 + **цв.вклейки**. Эффективный самоучитель	147	33	2005	400	
Сухарев	Turbo Pascal 7.0. Теория и практика программир. 2-е изд.	165	34	2004	640	
Юдин	Самоучитель работы на ноутбуке. 2-е изд + **цв.вклейки**	175	39	2005	512	
Серия: Секреты мастерства						
Колисниченко	Linux-сервер своими руками. 3-е изд. перер. и дополненное	202	45	2005	752	
Мозговой	Классика программирования: алгоритмы, языки, автоматы, компиляторы. Практический подход	164	37	2006	320	
Сухарев	Основы Delphi. Профессиональный подход	184	37	2004	600	
Финков	Интернет. Шаг второй: от пользоват. к профессионалу + **CD**	88	17	2002	768	
Юдин	Ноутбук: особенности испол. и настройки + **цв.вклейки**	184	37	2005	384	
Серии: Профи и др.						
Вебер	Knowledge-технологии в консалтинге и управл.предпр.+ **CD**	127	18	2003	176	
Гургенидзе	Мультисервисные сети и услуги широкополосного доступа	109	18	2003	400	
Есипов	Информатика (учебник), 3-е изд.	102	21	2003	400	
Куприянов	Техническое обеспечение цифровой обработки сигналов	66	12	2000	752	
Кучеров	Источники питания ПК и периферии. 3-е изд.	142	32	2005	432	
Кучерявый	Пакетная сеть связи общего пользования	132	27	2004	272	
Кучерявый	Управл. трафиком и качество обслуживания сети Интернет	128	29	2004	336	
Никамин	Цифровая звукозапись. Технологии и стандарты	76	12	2002	256	
Щеглов	Защита комп. информации от несанкционир. доступа	164	20	2004	384	