ВНУТРЕННЕЕ УСТРОЙСТВО LINUX

Дмитрий Кетов

bhv®

Дмитрий Кетов

ВНУТРЕННЕЕ УСТРОЙСТВО LINUX

Санкт-Петербург
«БХВ-Петербург»
2017

УДК 004.451
ББК 32.973.26-018.2
К37

Кетов Д. В.

К37 Внутреннее устройство Linux. — СПб.: БХВ-Петербург, 2017. — 320 с.: ил.

ISBN 978-5-9775-3580-9

Книга представляет собой введение во внутреннее устройство операционной системы Linux. Все положения наглядно проиллюстрированы примерами, разработанными автором и проверенными им на практике. Рассмотрены основные подсистемы ядра и их сущности — файлы и файловые системы, виртуальная память и отображаемые файлы, процессы, нити и средства межпроцессного взаимодействия, каналы, сокеты и разделяемая память. Раскрыты дискреционный и мандатный (принудительный) механизмы контроля доступа, а также привилегии процессов. Подробно описано пользовательское окружение и интерфейс командной строки CLI, оконная система X Window и графический интерфейс GUI, а также сетевая подсистема и служба SSH. Особое внимание уделено языку командного интерпретатора и его использованию для автоматизации задач эксплуатации операционной системы.

Для студентов, пользователей, программистов
и системных администраторов Linux

УДК 004.451
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Екатерина Капалыгина*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн обложки	*Марины Дамбиевой*

Подписано в печать 30.09.16.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 25,8.
Тираж 1500 экз. Заказ № 1619.
"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.

Первая Академическая типография "Наука"
199034, Санкт-Петербург, 9 линия, 12/28

ISBN 978-5-9775-3580-9

Оглавление

Введение

О чем эта книга?

Книга, которую вы держите в руках, адресована начинающим пользователям[1] операционной системы Linux и представляет собой ***иллюстрированное*** введение в ее ***внутреннее устройство*** — от ядра до сетевых служб и от утилит командной строки до графического интерфейса.

По моему скромному мнению, никакие книги не могут превратить читателя из новичка в эрудированного специалиста — ни обзорные, рассказывающие «ничего обо всем», ни узкоспециализированные, повествующие «все об одном». Без самостоятельного опыта, получаемого в результате исследования происходящих процессов и явлений, осмысление любого книжного знания практически невозможно.

Именно поэтому данная книга преследует всего одну важную цель — привить читателю ***навыки*** самостоятельного ***исследования*** постоянно ***эволюционирующей*** операционной системы Linux через ***умения*** пользоваться соответствующим инструментарием и на основе теоретических ***знаний*** о ее устройстве и философии[2].

Порядок и стиль изложения материала, иллюстративные изображения и листинги, приведенные в книге, являются результатом обобщения моего достаточно долгого опыта преподавания технических дисциплин, посвященных операционным системам в целом и операционной системе Linux в частности. Именно такой мне представляется картина «правильного» минимального набора необходимых знаний, умений и навыков, необходимых для построения качественной ***ментальной модели*** современной ОС Linux.

---

[1] Хотя термин «пользователь» зачастую ассоциируется с этаким домашним любителем или офисным работником, но никак не с IT-специалистом, книга адресована абсолютно всем, кому интересно понимать то, как устроена операционная система изнутри. Именно для ее компетентного применения в своей профессиональной деятельности такое понимание в первую очередь нужно программистам и системным администраторам — разработчикам и эксплуатационникам информационных систем на базе Linux.

[2] **W:[Философия UNIX]**.

Основная задача книги — иллюстративно изложить внутреннее устройство операционной системы Linux, связав «***теорию***» и «***практику***». В отличие от многих изданий, скупо и сухо излагающих теоретические аспекты построения операционной системы или излишне концентрирующихся на деталях конкретной реализации того или иного ее программного обеспечения, в книге рассматриваются ***абстрактные*** концепты внутреннего устройства ОС, иллюстративно подкрепляемые примерами анализа (а иногда и синтеза) ее ***конкретных*** компонент и связей между ними.

Все части операционной системы рассматриваются в контексте типичных задач, решаемых при их помощи на практике, а сама иллюстрация проводится при помощи соответствующего инструментария пользователя, администратора и разработчика.

Многие концепты и компоненты, инструменты и задачи будут иметь ***общность***, характерную для всех **W:[дистрибутивов Linux]**[1], но их реализации могут существенно ***различаться***. Поэтому, преследуя цель сохранить практическую значимость и жертвуя некоторой потерей общности излагаемого материала, повествование ведется в контексте отдельно взятой операционной системы, в качестве которой выступает дистрибутив **W:[Ubuntu Linux]**. Читатель может ***свободно*** скачать этот дистрибутив из Интернета и использовать в абсолютно любых целях, например, для проработки примеров, приведенных в книге.

Выбор в пользу дистрибутива **W:[Ubuntu Linux]**, в частности, сделан еще и по совокупности присущих ему свойств, как то: большое сообщество его пользователей (включая русскоговорящих), громадная пакетная база, регулярность обновлений и их стабильность, дружественность к начинающим и, наконец, широкая распространенность технических решений на его основе — от бизнес-решений и до сертифицированных разработок в оборонной промышленности.

Кому адресована книга?

Понимание внутреннего устройства важно как в системном администрировании для направленной диагностики проблем эксплуатации информационных систем, построенных на платформе Linux, так и для разработки программного обеспечения, учитывающего особенности ее внутреннего устройства.

Поэтому книга адресуется абсолютно всем, кто не любит использовать непрозрачный «черный ящик», кому интересна механика всех этих шестеренок, что вращаются внутри Linux, ежедневно решая задачи миллионов пользователей по всему

[1] А зачастую и для всех операционных систем семейства **W:[UNIX]**, к которому Linux принадлежит.

миру. Всем тем, кто в детстве разобрал не один игрушечный луноход в попытке узнать, как там чувствуют себя человечки в скафандрах.

Принятые соглашения и обозначения

Наиболее полезным навыком при исследовании любой незнакомой информационной системы является способность оперативно получать справочную информацию непосредственно изнутри ее, не прибегая к внешним источникам (вроде поиска по Google). Именно поэтому в данной книге *нигде* не встречаются даже частичные цитирования содержания справочных систем.

Классическим электронным справочником UNIX являются страницы руководства (**man**ual pages, подробнее *см. главу 2*), которые сгруппированы в тематические секции, а ссылки на страницу *page* в секции *N* принято записывать как **page(N)**, в том числе и в этой книге. Другая полезная информация, в том числе историческая, доступна из статей всемирной онлайн-энциклопедии Wikipedia, поэтому полезные ссылки на статью *useful àrticle* изображаются как **W:[useful article]**.

Разнообразные значки [☛☝?✋🗣🗝❶...❾⓪...⑨❮❯♞⇆⇵↰] на полях и в теле листингов позволяют акцентировать внимание на *нужном* и сопровождать *важное* дополнительными комментариями в тексте повествования.

Полужирный шрифт **command** в листингах идентифицирует вводимые пользователем команды или управляющие символы и последовательности, а нажатия на клавиши или их комбинации изображаются как Ctrl Alt F4. Курсив используется для выделения *понятий* и *терминов*, а полужирный курсив выделяет ***ключевые слова***, определяющие основное существо текста абзаца.

Значок указательного пальца ☛ и ☝ и вот такое «выделение маркером» укажет на те места в листингах, куда нужно обратить внимание. Номера ❶...❾⓪...⑨ и звездочка ★ позволят сослаться на конкретные места листингов в основном тексте.

Вопрос ? отметит те места листингов, которые должны вызывать недоумение вида «а это еще что?», а знак восклицания ! укажет на места, которые должны сопровождаться возгласами «ох тыж! ничего себе!». На особенно эмоциональные места листингов указывает значок 🗣, а поднятый вверх большой палец ☝, как и обычно, одобряет определенные команды листингов, которые приводят к долгожданному результату.

Аналогично, закрытый замок 🔒 символизирует закрытость (заблокированность) или отсутствие доступа, а открытый замок 🔓, наоборот, открытость (разблокированность), наличие доступа. Ключ 🗝 указывает на операцию открытия/закрытия чего-либо (например, ввод пароля при аутентификации).

Кинжал 🗡 указывает на выполнение некоторой деструктивной операции, а значки часов 🕐🕑🕒🕓 символизируют ожидание завершения достаточно длительной операции.

Для удобства чтения листингов значки ⇆⇵ указывают на «замещающий» вывод (т. е. без скроллинга), значок 🔄 обозначает очередную итерацию некоторого цикла, а значок 🚫 указывает на его окончание. Ну и, наконец, разнообразные стрелки ⤴↩▸→←⤒ связывают отдельные части листинга друг с другом.

Методические рекомендации

Многие иллюстративные листинги, приводящиеся в книге, нужно воспринимать не как «дополнение» к основному тексту, а как сам основной текст, просто на (пока) незнакомом и непонятном языке, а саму книгу — как учебник нового иностранного языка.

Получение читателем собственного отклика от взаимодействия с операционной системой представляется мне наиболее значимым результатом освоения содержимого этой книги. Поэтому чтение стоит непременно сопровождать самостоятельным выполнением команд листингов, т. к. успешное изучение нового иностранного языка никак невозможно без его непосредственного практического использования в процессе изучения.

Не менее важным для построения целостных взаимоотношений с системой является самостоятельное изучение первоисточников знаний[1] — ее внутренней документации, страниц руководства **man(1)**.

Для выполнения команд листингов и доступа к внутренним справочникам потребуется сама операционная система, что несложно получить, например, инсталлировав настольную версию Ubuntu Linux в виртуальную машину **W**:[VirtualBox].

Дополнительную пользу принесет ознакомление и с дополнительными источниками информации в Интернете, например со статьями на **http://help.ubuntu.ru**, а также поиск ответов на возникающие вопросы на **http://forum.ubunu.ru**, **http://askubuntu.com** и даже на **http://UNIX.stackexchange.com** и **http://stackoverflow.com**.

Кроме текстовых источников информации в Интернете можно воспользоваться услугами таких очаровательных онлайн-сервисов, как **http://explainshell.com** для объяснений конструкций и **http://www.shellcheck.net** для проверки проблем в конструкциях командного интерпретатора. Для проверки примеров из книги и разработки собственных сценариев командного интерпретатора незаменимую помощь может оказать сервис **http://www.tutorialspoint.com/execute_bash_online.php**.

---

[1] Это обязательно потребует знания технического английского языка, но без него вообще трудно рассчитывать на сколько-нибудь значительные успехи в отрасли информационных технологий.

Что должен знать читатель

В операционной системе Linux программное обеспечение поставляется в виде так называемых пакетов (package) — специальным образом подготовленных архивов, содержащих само *программное* обеспечение, его *конфигурационные* файлы, его данные и *управляющую* информацию. Управляющая информация пакета включает контрольные суммы устанавливаемых файлов, зависимости устанавливаемого пакета от других пакетов, краткое описание пакета, сценарии установки, сценарии удаления пакета и прочие данные, необходимые *менеджеру* пакетов.

В примерах, приведенных в листингах, часто встречаются программы, которые, возможно, не будут установлены «по умолчанию», поэтому важно знать и понимать, как устроена подсистема управления установкой и удалением программного обеспечения — так называемые менеджеры пакетов и зависимостей[1].

Менеджер пакетов производит непосредственную установку и удаление пакетов программного обеспечения, а также ведет их учет в системе. Вспомогательная «грязная» работа по подбору зависящих друг от друга пакетов, получению их из *репозиториев*[2] (например, скачивание с FTP/HTTP-серверов), выбору правильных версий пакетов, определению правильного их порядка установки достается *менеджеру зависимостей*.

Листинг B1. Пакетный менеджер dpkg

```
bart@ubuntu:~$ which date
/bin/date
❶ bart@ubuntu:~$ dpkg -S /bin/date
coreutils: /bin/date
❷ bart@ubuntu:~$ dpkg -L coreutils
...                 ...          ...          ...          ...
/bin/ln
/bin/touch
/bin/date
/bin/mkdir
/bin/sty
...                 ...          ...          ...          ...
```

[1] Условно, можно выделить две ветви операционной системы Linux — ветвь debian, к которой относятся дистрибутивы **W**:[Debian] и **W**:[Ubuntu], и ветвь redhat, куда нужно отнести **W**:[RHEL], **W**:[CentOS] и **W**:[Fedora]. В debian-ветви используется пакетный менеджер **dpkg** и построенные над ним менеджеры зависимостей **apt**, **aptitude**, **synaptic** и **software-center**, а в ветви redhat — пакетный менеджер **rpm** и основной менеджер зависимостей **yum**.

[2] Хранилище пакетов у разработчиков дистрибутива Linux.

```
bart@ubuntu:~$ dpkg -s coreutils
Package: coreutils
Essential: yes
Status: install ok installed
...                  ...             ...             ...             ...
Version: 8.13-3ubuntu3.2
Replaces: mktemp, timeout
Depends: dpkg (>= 1.15.4) | install-info
Pre-Depends: libacl1 (>= 2.2.51-5), libattr1 (>= 1:2.4.46-5), libc6 (>= 2.15), libselinux1
(>= 1.32)
Conflicts: timeout
Description: GNU core utilities
 This package contains the basic file, shell and text manipulation
 utilities which are expected to exist on every operating system.

 Specifically, this package includes:
 arch base64 basename cat chcon chgrp chmod chown chroot cksum comm cp
                      ...                    ...                    ...
 users vdir wc who whoami yes
Homepage: http://gnu.org/software/coreutils
Original-Maintainer: Michael Stone <mstone@debian.org>
```

При помощи менеджера пакетов (листинг B1) всегда можно узнать имя пакета ❶, в который входит та или иная компонента операционной системы (например, **/bin/date**), или, наоборот, узнать список компонент ❷, установленных из указанного пакета (например, **coreutils**).

Если при попытке выполнить ту или иную команду операционной системы Ubuntu Linux (это одна из причин, по которой иллюстрация в книге ведется именно с ее помощью) обнаружится, что нужный пакет с программным обеспечением не установлен, то при наличии доступа в Интернет можно тривиальным способом доустановить недостающие компоненты (листинг B2).

Листинг B2. Установка недостающего программного обеспечения

```
bart@ubuntu:~$ mail
Программа 'mail' на данный момент не установлена. Вы можете установить её, выполнив:
sudo apt-get install mailutils
bart@ubuntu:~$ sudo apt-get install mailutils
[sudo] password for bart:
Чтение списков пакетов... Готово
```

```
Построение дерева зависимостей
Чтение информации о состоянии... Готово
Предлагаемые пакеты:
  mailutils-mh
НОВЫЕ пакеты, которые будут установлены:
  mailutils
обновлено 0, установлено 1 новых пакетов, для удаления отмечено 0 пакетов, и 567 пакетов
не обновлено.
Необходимо скачать 401 kБ архивов.
После данной операции объём занятого дискового пространства возрастёт на 1 149 kB.
Получено:1 http://archive.ubuntu.com/ubuntu/ precise/universe mailutils i386 1:2.2+dfsg1-5
[401 kB]
Получено 401 kБ за 0с (846 kБ/с)
Выбор ранее не выбранного пакета mailutils.
(Чтение базы данных ... на данный момент установлено 281689 файлов и каталогов.)
Распаковывается пакет mailutils (из файла .../mailutils_1%3a2.2+dfsg1-5_i386.deb)...
Обрабатываются триггеры для man-db ...
Настраивается пакет mailutils (1:2.2+dfsg1-5) ...
                    ...                 ...                 ...
```

Совет для начинающих

И напоследок, самый важный совет для начинающих — начните!

Глава 1

Архитектура ОС Linux

1.1. Обзор внутреннего устройства

Операционная система (ОС), в общем, и **W:[Linux]**, в частности (рис. 1.1), представляет собой набор специализированных *программных* средств, обеспечивающих *доступ* потребителей (пользователей) к ресурсам (устройствам) и *распределение* последних между потребителями наиболее *удобным* и *эффективным* способом. Под ресурсами в первую очередь понимают аппаратные средства, такие как центральные процессоры, память, устройства ввода-вывода, дисковые и прочие накопители и другие периферийные устройства. Во вторую очередь к ресурсам относят такие «виртуальные» сущности (на рис. 1.1 не показаны), как процессы, нити, файлы, каналы и сокеты, семафоры и мьютексы, окна и прочие артефакты самой операционной системы, которые не существуют вне ее границ.

Как и во многих других операционных системах, в Linux выделяют два главных режима работы ее программных средств — ядерный режим (kernel mode), он же пространство ядра (kernel space), и пользовательский, внеядерный режим (user mode), или же пользовательское пространство (user space). Основное отличие этих двух режимов состоит в привилегиях доступа к аппаратным средствам — памяти и устройствам ввода-вывода, к которым разрешен полный доступ из режима ядра и ограниченный доступ из режима пользователя.

Совокупность работающих в ядерном режиме программ называют *ядром*, которое в Linux состоит из основы (или же, остова) и присоединяемых к ней объектов — динамически загружаемых модулей (подробнее *см. разд. 4.1.1*). Несмотря на компонентность, **W:[Ядро Linux]** относят к классу *монолитных* в силу того, что все его компоненты выполняются с *одинаковыми* (ядерными) привилегиями. В классе *микроядерных* систем, наоборот, компоненты ядра работают с *разными* привилегиями: основная компонента — микроядро (планировщик процессов/нитей и менеджер памяти) — с наибольшими привилегиями, менеджер ввода-вывода, драйверы устройств, файловые системы, сетевые протоколы и пр. — с другими, обычно меньшими привилегиями (возможно даже, с привилегиями пользовательского режима).

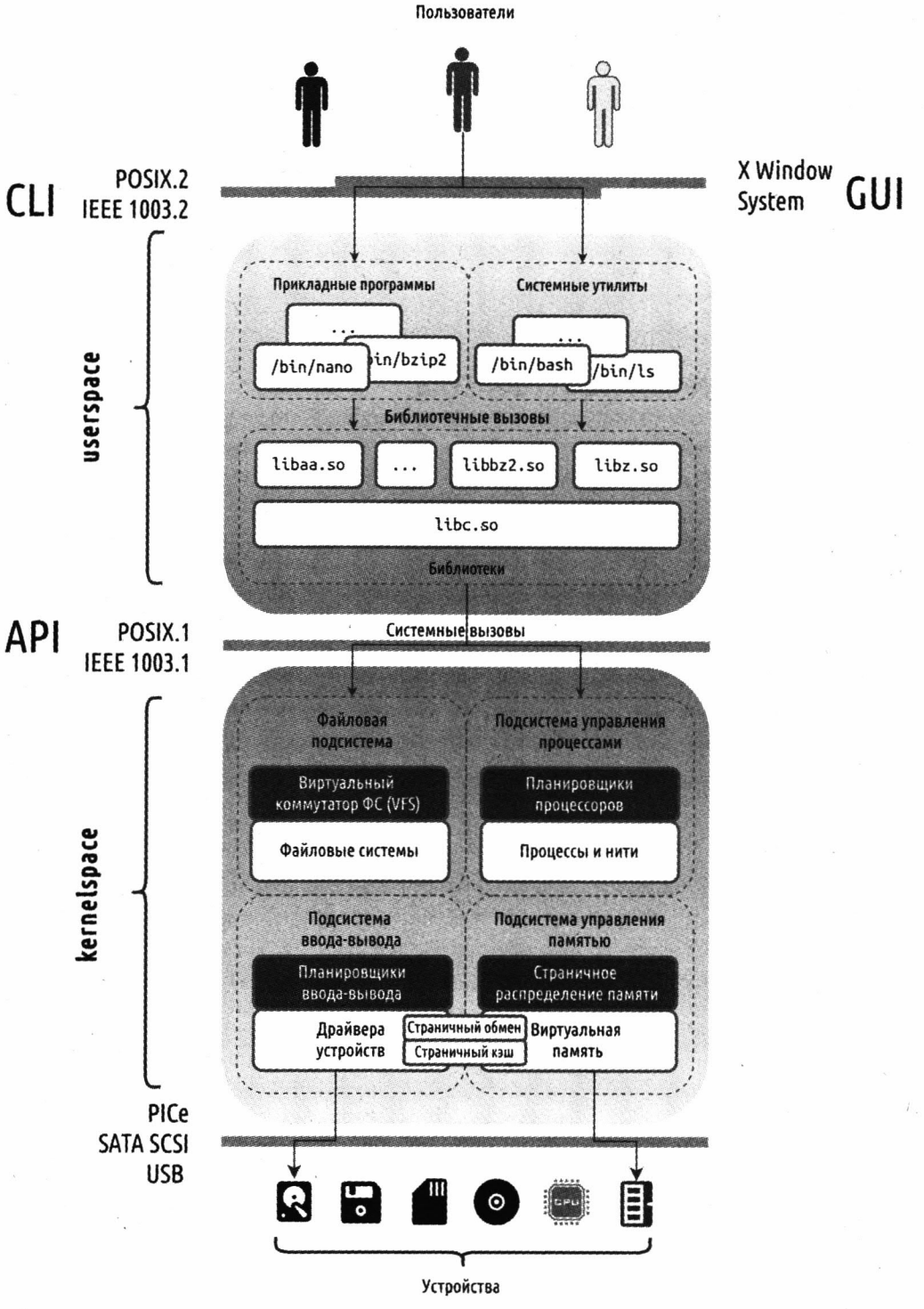


Рис. 1.1. Компоненты операционной системы Linux

1.2. Внеядерные компоненты: программы и библиотеки

Ядерные компоненты в основном обеспечивают решение задачи диспетчеризации (распределения) ресурсов между потребителями и предоставляют им базовый интерфейс доступа к ресурсам. Решение задачи обеспечения удобства доступа реализуется компонентами внеядерного режима — библиотеками динамической и статической компоновки (подробнее *см. разд. 4.1*).

Функции операционной системы, реализуемые ядерными компонентами, доступны внеядерным компонентам посредством *системных вызовов* — специализированных наборов обращений для получения услуг ядра. Системные вызовы выполняются в ядре, а вызываются при помощи основной внеядерной компоненты — библиотеки **libc.so** языка программирования W:[Си] (так исторически сложилось в силу того, что бóльшая часть системных программных средств написана на этом языке).

Функции, реализуемые внеядерными компонентами, доступны посредством *библиотечных вызовов* и вызываются/выполняются в самих библиотеках (например, алгоритмы сжатия информации в библиотеках **libz.so** и **libbz2.so**).

1.3. Ядерные компоненты: подсистемы управления процессами, памятью, вводом-выводом, файлами

Одна из основных задач ядра — распределение ресурсов между потребителями — вполне естественным образом приводит к тому, что среди ядерных компонент выделяют соответствующие (аппаратным ресурсам) менеджеры, так называемые подсистемы управления *процессами*, *памятью*, *вводом-выводом* и *файловую* подсистему.

Подсистема управления *процессами* распределяет время центральных процессоров (ЦП) между выполняющимися задачами (т. е. реализует W:[многозадачность]). Она создает и уничтожает такие сущности, как *процессы* и *нити (см. разд. 4.2)*, и организует одновременное (параллельное или псевдопараллельное) их выполнение при помощи планировщиков (англ. *scheduler*), реализующих алгоритмы распределения процессорного времени.

Подсистема *ввода-вывода* распределяет доступ к устройствам ввода-вывода (УВВ) между процессами и предоставляет им унифицированные интерфейсы *блочного, символьного (см. разд. 3.2.5)* и *пакетного* (сетевого) устройств *(см. разд. 6.1)*. Для устройств *внешней* памяти (дисковых или твердотельных накопителей, более медленных по сравнению с оперативной памятью) подсистема ввода-вывода организует W:[кэширование] при помощи подсистемы управления памятью.

Подсистема управления *памятью* (подробнее *см. разд. 4.7*) распределяет пространство оперативного запоминающего устройства (ОЗУ) между процессами при помо-

щи механизма страничного отображения — *выделяет* (и высвобождает) процессам страничные кадры физической памяти и *отображает* на страницы их адресного пространства. Кроме того, эта подсистема организует **W:[виртуальную память]** за счет механизма *страничного обмена* (**W:[подкачка страниц]**) — *вытесняет* неиспользуемые страницы процессов во внешнюю память и *загружает* их обратно по требованию при помощи подсистемы ввода-вывода.

Стоит отметить, что распределение ресурсов процессоров и устройств ввода-вывода происходит «во времени», т. е. в отдельные промежутки времени эти устройства выполняют операции только *одного* процесса или нити. Наоборот, распределение ресурсов запоминающих устройств происходит «в пространстве», т. к. информация *нескольких* процессов одновременно размещается в разных их *областях*.

Файловая подсистема ядра (подробнее *см. главу 3*) предоставляет процессам унифицированный интерфейс *файлового* доступа к внешней памяти (внешним запоминающим устройствам, ВЗУ — магнитным дисковым, твердотельным накопителям и т. п.) и распределяет между ними пространство ВЗУ при помощи *файлов* и *файловых систем*. Особенное назначение файловой подсистемы состоит еще и в том, что при помощи ее *файлового интерфейса* процессам предоставляется доступ и к другим подсистемам. Так, доступ к устройствам ввода-вывода организуется посредством специальных файлов устройств (блочных и символьных), например к CD/DVD-накопителю — через файл **/dev/sr0**, а к манипулятору мыши — через **/dev/input/mouse0**. Доступ к физической памяти и памяти ядра ОС организуется через файлы виртуальных устройств **/dev/mem** и **/dev/kmem**, а доступ процессов к страницам памяти друг друга — через файлы **/proc/PID/mem** псевдофайловой системы **procfs**. Даже доступ к внутренним параметрам и статистике ядра операционной системы возможен через файлы псевдофайловой системы **procfs**, а к списку обнаруженных ядром устройств ввода-вывода — через файлы псевдофайловой системы **sysfs**.

Кроме всех вышеперечисленных задач, файловая подсистема, подсистема ввода-вывода и подсистема управления процессами в совокупности предоставляют процессам средства межпроцессного взаимодействия, такие как сигналы *(см. разд. 4.8)*, каналы, сокеты и разделяемая память *(см. разд. 4.9)*.

1.4. Трассировка системных и библиотечных вызовов

Для наблюдения за обращениями программ к услугам ядерных компонент операционной системы, т. е. за системными вызовами, служит утилита **strace(1)**, предназначенная для трассировки — построения трасс выполнения той или иной программы. В листинге 1.1 представлена трассировка программы **date(1)** относительно

системных вызовов времени **time(2)**, **gettimeofday(2)**, **settimeofday(2)** и **clock_gettime(2)**, **clock_settime(2)**, где оказывается, что для получения и установки значения системного времени программа **date(1)** использует системные вызовы **clock_gettime(2)** и **clock_settime(2)**.

Листинг 1.1. Трассировщик системных вызовов strace

```
$ date
Пн. янв. 11 01:44:17 MSK 2016

$ strace -fe time,gettimeofday,clock_gettime date
clock_gettime(CLOCK_REALTIME, {1452465309, 213946474}) = 0
Пн. янв. 11 01:44:26 MSK 2016

$ strace -fe settimeofday,clock_settime date -s 10:00
clock_settime(CLOCK_REALTIME, {1452495600, 0}) = -1 EPERM (Operation not permitted)
date: невозможно установить дату: Операция не позволяется
Пн. янв. 11 10:00:00 MSK 2016
```

Листинг 1.2. Трассировщик системных вызовов ltrace

```
$ ltrace  -e localtime,asctime,ctime,strftime date
localtime(0xbf818448)                              = 0xb77c97e0
strftime(" \320\237\320\275.", 1024, " %a", 0xb77c97e0) = 6
strftime(" \321\217\320\275\320\262.", 1024, " %b", 0xb77c97e0) = 8
Пн. янв. 11 01:45:12 MSK 2016
+++ exited (status 0) +++
```

1.5. Интерфейсы прикладного программирования

Системные и библиотечные вызовы Linux, формирующие интерфейс прикладного программирования (API, application programming interface), соответствуют определенным промышленным спецификациям, в частности (практически идентичным друг другу) стандартам **W:[POSIX]**[1], **Portable Operating System Interface** и **SUS**, **W:[Single UNIX Specification]**, доставшимся «в наследство» от семейства операционных систем UNIX, членом которого Linux и является.

[1] Стандарт POSIX выпускается комитетом 1003 организации **W:[IEEE]**, поэтому имеет формальное обозначение IEEE 1003, а части стандарта POSIX.1 и POSIX.2 формально обозначаются IEEE 1003.1 и IEEE 1003.2 соответственно.

Стандарт POSIX условно делится на две части: **POSIX.1**, программный интерфейс (API) операционной системы, и **POSIX.2**, интерфейс командной строки (CLI) пользователя *(см. главу 2)* и командный интерпретатор *(см. главу 5)*.

1.6. В заключение

Совершенно очевидно, что в современной медицине без понимания анатомии живых организмов невозможно представить ни их терапию, ни хирургию. В информационных технологиях абсолютно аналогичным образом разработка и эксплуатация программного обеспечения будут успешны только на основе понимания внутреннего устройства операционной системы.

Рисунок 1.1 изображает эдакий «рентгеновский снимок» внутренностей операционной системы Linux, подробная анатомия которой шаг за шагом и раскрывается в последующих главах.

Глава 2

Пользовательское окружение ОС Linux

2.1. Командный интерфейс

Основным интерфейсом взаимодействия между ЭВМ и человеком в классической операционной системе **W:[UNIX]** был единственно возможный, диктуемый аппаратными устройствами ее времени[1], *командный интерфейс*. Называемый сегодня интерфейсом командной строки (Command Line Interface, **W:[CLI]**), он в неизменном виде сохранил все свои элементы — понятие *терминала*, двусторонний попеременный диалог при помощи клавиши ↵ENTER, управляющие символы и клавишу CTRL для их набора.

Терминал является оконечным (терминальным) оборудованием, предназначенным для организации человеко-машинного интерфейса, и состоит из устройств вывода — принтера или дисплея, и устройств ввода — клавиатуры, манипулятора «мышь» и пр.

Алфавитно-цифровой терминал позволяет вводить и выводить символы из некоторого заданного набора (например, семибитной кодировки **ascii(7)**) или другого набора символов **charsets(7)**), состоящего из букв алфавита, цифр, знаков препинания, некоторых других значков, и символов специального назначения для управления самим терминалом — *управляющих символов*.

Ранние, *печатающие терминалы*, представляли собой телетайпы **W:[телетайп]** (телепринтеры **W:[teleprinter]**), которые печатали символы из фиксированного набора на ленте или рулоне бумаги — слева направо, сверху вниз. Управляющие символы использовались для управления перемещением печатающей головки справа налево (символ **BS**), возврата головки к началу строки (символ **CR**), прокрутки рулона бумаги (символ **LF**) и пр.

Дисплейный терминал (видеотерминал на основе электронно-лучевой трубки) в упрощенном своем режиме эмулирует поведение печатающего терминала: пово-

---

[1] ЭВМ **W:[PDP-11]** и терминалы **W:[Teletype Model ASR-33]**.

рачивающийся рулон бумаги — при помощи *скроллинга* изображаемых строк снизу вверх, а перемещающуюся вдоль строки печатающую головку — при помощи *курсора*. В расширенном режиме видеотерминал используется как матрица символов, например в 24 строки по 80 символов в строке, и позволяет выводить символы в произвольное место матрицы, задавать символам стиль изображения, как то: мерцание, жирность, инвертирование, подчеркивание и цвет, и даже менять шрифты символов терминала. Для управления курсором, его позиционирования, смены стиля изображения символов и прочих возможностей видеотерминала применяются управляющие символы *(см. разд. 2.3)* и управляющие последовательности *(см. разд. 2.4)*.

Двусторонний попеременный диалог (рис. 2.1) командного интерфейса между пользователем и операционной системой представляет собой процесс ввода команд ❶ пользователем посредством клавиатуры и получения результата их выполнения ❷ на бумаге или дисплее алфавитно-цифрового терминала.

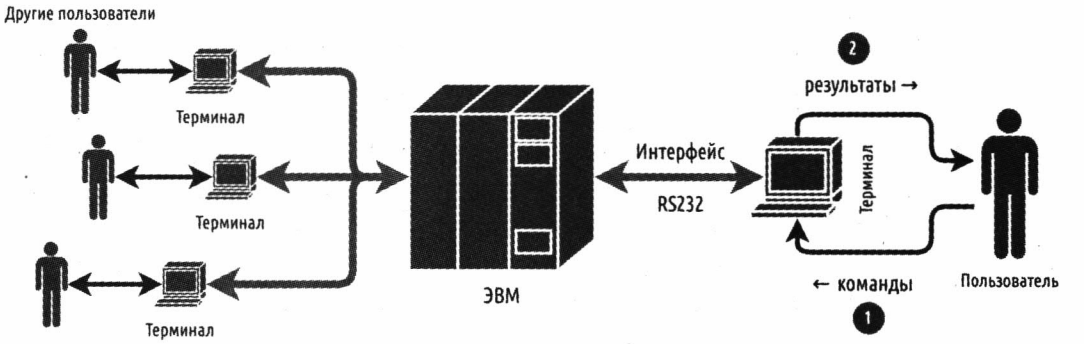


Рис. 2.1. Терминалы и командный интерфейс

В начале сеанса работы в многопользовательской среде операционной системы пользователь должен произвести регистрацию (**log**ging **in**) себя в системе (обычно говорят «произвести вход» в систему) при помощи предъявления имени своей учетной записи (login) и соответствующего и известного ему пароля (password, буквально — пропускное **pass** слово **word**) (рис. 2.2).

Процедура регистрации начинается с заставки операционной системы ❶ и приглашения к вводу имени учетной записи ❷, в ответ на которое пользователь вводит имя своей учетной записи ①. Затем, в ответ на приглашение к вводу пароля ❸, пользователь вводит пароль ② и при этом на алфавитно-цифровых терминалах никакие символы не изображаются. При положительном исходе регистрации пользователь получает сообщение ❹ о последней (**last**) успешной регистрации, сообщение дня и приглашение командного интерпретатора ❺.

Передача управления от пользователя к операционной системе на каждом шаге диалога происходит при помощи нажатия клавиши ↵ ENTER, а передача управления

в обратную сторону при помощи *приглашений* к вводу регистрационного имени, пароля, командного интерпретатора и пр. Приглашение командного интерпретатора исторически состояло из символа **$** или символа **%**, а при регистрации под учетной записью администратора — из символа **#**. Позднее приглашение развилось в **finn@ubuntu:~$** и состоит теперь из имени зарегистрировавшегося пользователя **finn**, собственного имени компьютера **ubuntu**, условного имени домашнего каталога пользователя, обозначенного символом **~**, и «классического» символа приглашения **$**.

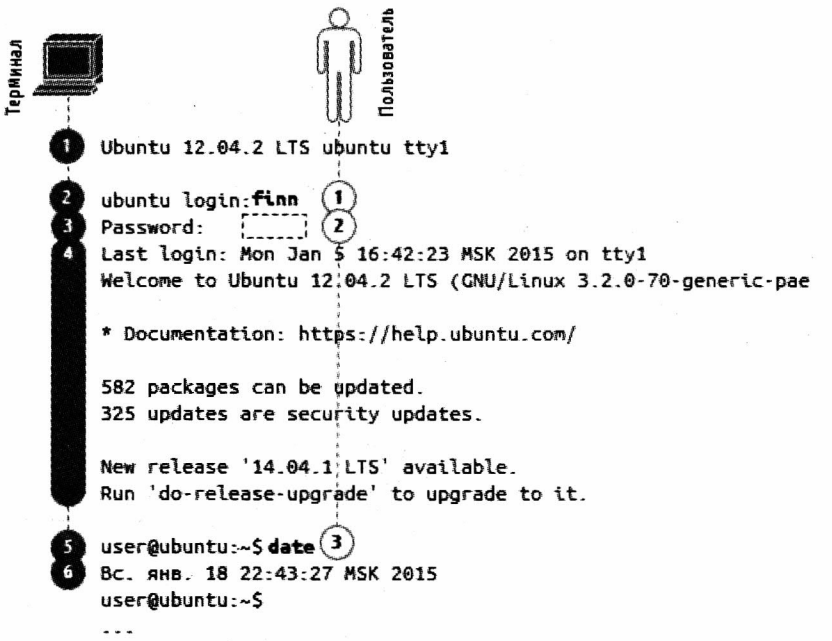


Рис. 2.2. Регистрация пользователя в системе

Сеанс командного интерфейса пользователя продолжается двусторонним попеременным диалогом с командным интерпретатором, где пользователь вводит команды ③ и получает результаты их выполнения ❻.

2.2. Виртуальные терминалы

На текущий момент времени многопользовательские системы с настоящими физическими терминалами, подключенными посредством интерфейса RS232 и его драйвера **ttyS(4)** к *большой* ЭВМ, — экзотическая редкость. На *персональных* ЭВМ для взаимодействия с пользователем имеются стандартные клавиатура, видеоадаптер и монитор, формирующие так называемую **W:[консоль]**, которая используется драйвером *виртуальных терминалов* для эмуляции нескольких физических терминалов (рис. 2.3).

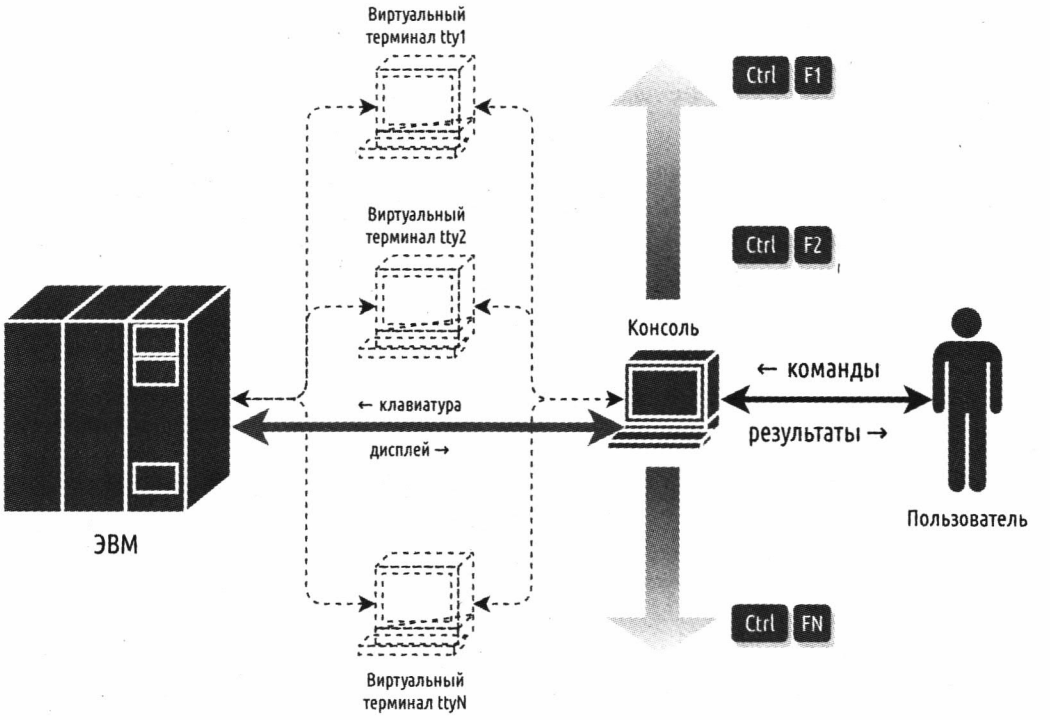


Рис. 2.3. Виртуальные терминалы

Узнать имя текущего терминала (а точнее — имя специального файла устройства[1] терминального драйвера, см. листинг 2.1), на котором выполнен вход в систему, позволяет команда **tty(1)**, а список всех терминальных входов пользователей — команды **users(1)**, **who(1)** и **w(1)**.

Листинг 2.1. Утилиты tty, users, who и w

```
finn@ubuntu:~$ tty ↵
/dev/tty1
finn@ubuntu:~$ users ↵
bubblegum finn iceking jake jake marceline
finn@ubuntu:~$ who ↵
iceking  tty4         2015-03-27 10:46
bubblegum tty5        2015-03-27 10:46
marceline tty3        2015-03-27 10:47
finn     tty1         2015-03-27 10:46
jake     tty7         2015-03-24 23:46
```

[1] Подробнее о специальных файлах устройств *см. в разд. 3.2.5.*

```
jake     pts/3        2015-03-27 10:47 (:0.0)
finn@ubuntu:~$ w
 10:47:52 up 2 days, 11:02,  9 users,  load average: 0,05, 0,18, 0,27
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
iceking  tty4                     10:46    1:20   0.05s  0.00s -sh
bubblegu tty5                     10:46    1:04   0.04s  0.00s -sh
marcelin tty3                     10:47   32.00s  0.06s  0.00s -sh
❶ finn     tty1                     10:46    1:04   0.50s  0.44s w
❷ jake     tty7                     Tue23   2days 25:36   2.88s gnome-session -
❸ jake     pts/3    :0.0            10:47    0.00s  0.21s  0.00s -bash
```

Драйвер виртуального терминала позволяет переключаться между эмулируемыми терминалами при помощи сочетания клавиш Ctrl F_1, ..., Ctrl F_{12} (первые двенадцать терминалов из 63 возможных), Alt ← для переключения на предыдущий, Alt → для переключения на следующий виртуальный терминал. При переключении из *графического* виртуального терминала на другой виртуальный терминал необходимо добавлять к сочетанию еще и клавишу Ctrl, т. к. сочетания с клавишей Alt востребованы самим графическим интерфейсом, например Alt F_4 закрывает активное окно. Таким образом, для переключения из графического на третий виртуальный терминал используется сочетание Alt Ctrl F_3. Также, драйвер виртуальных терминалов позволяет листать буфер вывода виртуального терминала при помощи сочетаний Ctrl PgUp и Ctrl PgDn (к сожалению, после переключения терминалов буфер пропадает).

Как и любым другим, драйвером виртуальных терминалов можно управлять при помощи специально предназначенных команд, например, программа **chvt(1)** позволяет переключаться на заданный терминал по его номеру, а команда программы **setfont(1)** — загружать шрифты, формирующие начертания алфавитно-цифровых знаков (см. листинг 2.12).

2.2.1. Псевдотерминалы

При работе в оконной системе X Window System *(см. главу 7)* используются графические терминалы, тогда как для командного интерфейса требуется алфавитно-цифровой терминал. В этом случае (рис. 2.4) он эмулируется при помощи драйвера *псевдотерминала* **pty(4)** (**p**seudo t**ty**) и приложения-посредника — эмулятора терминала (например, **xterm** или **gnome-terminal**), который связывает действительный обмен ① в графическом окне с мультиплексором псевдотерминалов ② **ptmx(4)** (**p**seudo**t**erminal **m**ultiple**x**er), а тот, в свою очередь, присоединен драйвером к подчиненному псевдотерминалу **pts(4)** (**p**seudo**t**erminal **s**lave) командного интерфейса.

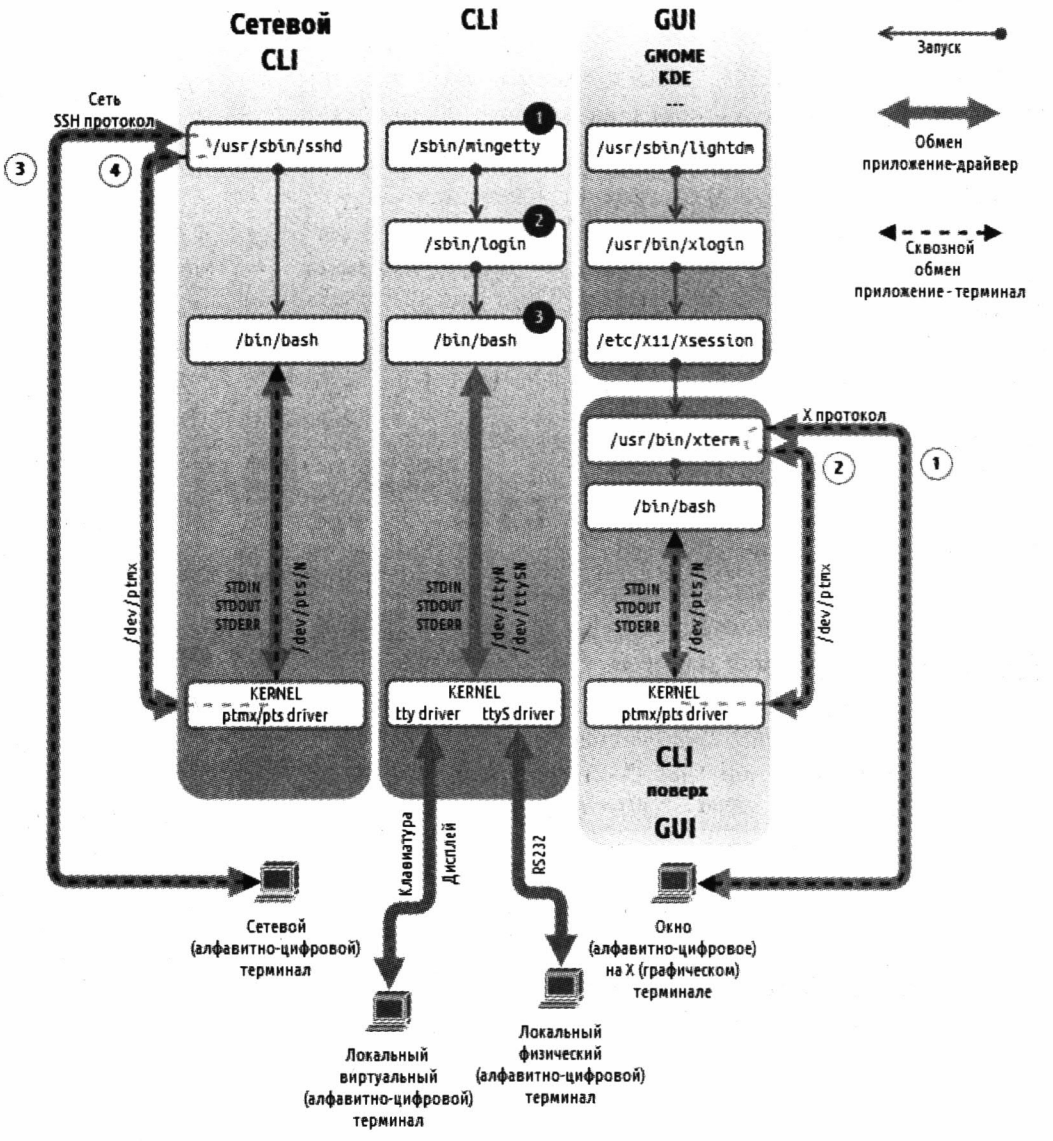


Рис. 2.4. Псевдотерминалы

Таким образом, командный интерпретатор и запускаемые им программы работают с воображаемым псевдотерминалом так, как будто окно графического приложения является настоящим физическим дисплеем и настоящей физической клавиатурой настоящего терминала. В примере из листинга 2.1 пользователь **finn** зарегистрирован в системе на первом виртуальном терминале **tty1**, а пользователь **jake** зарегистрирован на седьмом (графическом) виртуальном терминале **tty7** в оконной системе и работает с командным интерфейсом на третьем псевдотерминале **pts/3** в окне эмулятора терминала.

Аналогично, при подключении удаленного алфавитно-цифрового терминала посредством протоколов удаленного доступа (например, SSH) приложение-посредник (например, демон сетевой службы **sshd**, *см. разд. 6.4.1*) связывает действительный обмен ③ в сетевом соединении с мультиплексором псевдотерминалов ④.

Нужно заметить, что на этапе входа пользователя в систему посредством алфавитно-цифрового терминала (см. средний фрагмент с меткой **CLI** на рис. 2.4) последовательно запускаются: обработчик терминалов ❶, обработчик аутентификации и авторизации пользователей ❷, а затем командный интерпретатор ❸, например **bash(1)**. Именно **getty(1)** предъявляет пользователю (см. рис. 2.2) заставку операционной системы и приглашение к вводу имени пользователя, а **login(1)** — приглашение к вводу пароля, сообщение о последнем успешном входе и сообщение дня.

Аналогичные процессы происходят при любом входе пользователя в систему, например через псевдотерминалы «графического» или «сетевого» доступа (см. левый и правый фрагменты на рис. 2.4). В любом случае после аутентификации и авторизации основной программой (и первой в сеансе пользователя), интерпретирующей вводимые пользователем команды, является командный интерпретатор.

2.3. Управляющие символы

При *вводе* с терминала управляющие символы (табл. 2.1) служат командами драйверу терминала и в большинстве своем генерируются при помощи сочетания клавиш Ctrl (отсюда ее название **control** — управление) с одной из алфавитно-цифровых клавиш. В отдельных случаях, управляющие символы генерируются специально предназначенными для этого клавишами, например ↵ Enter, ↹ Tab или ← Backspace.

Таблица 2.1. Управляющие символы терминала

Управляющий символ			Клавиши	Код символа	
Нотация	Стандартное действие драйвера				
	ввод символа	вывод символа			
^C	intr	—	Ctrl C	0x03	ETX
^\	quit	—	Ctrl \ или Ctrl 4	0x1C	FS
^Z	susp	—	Ctrl Z	0x1A	SUB
^D	eof	—	Ctrl D	0x04	EOT
^?	erase	—	← Backspace или Ctrl ? или Ctrl 8	0x7F	DEL

Таблица 2.1 *(окончание)*

Управляющий символ			Клавиши	Код символа	
Нотация	Стандартное действие драйвера				
	ввод символа	вывод символа			
^H или \b	—	backspace	Ctrl H	0x08	BS
^W	werase	—	Ctrl W	0x17	ETB
^U	kill	—	Ctrl U	0x15	NAK
^I или \t	—	tab	⇆ Tab или Ctrl I	0x09	HT
^M или \r	eol	cr	↵ Enter или Ctrl M	0x0D	CR
^J или \n	eol	nl	Ctrl J	0x0A	LF
^S	stop	—	Ctrl S	0x13	DC3
^Q	start	—	Ctrl Q	0x11	DC1
^R	rprnt	—	Ctrl R	0x12	DC2
^V	lnext	—	Ctrl V	0x16	SYN
^N	—	so	Ctrl N	0x0E	SO
^O	—	si	Ctrl O	0x0F	SI
^[или \e	esc	esc	Esc или Ctrl [или Ctrl 3	0x1B	ESC

Так, например, нажатие клавиши ↵ Enter или эквивалентное сочетание Ctrl J, записывающееся как **^J**, генерирует управляющий символ **LF** (таким же действием обладает символ **CR**, **^M**), который сигнализирует драйверу терминала о завершении ввода строки (eol, **e**nd **o**f **l**ine) и необходимости «отдать команду на выполнение» (листинг 2.2).

Листинг 2.2. Управляющие символы ^J и ^M

```
finn@ubuntu:~$ date ↵
Вс. февр.  1 22:39:00 MSK 2015
finn@ubuntu:~$ hostname ^M
ubuntu
finn@ubuntu:~$ whoami ^J
finn
```

Нажатие клавиши ← Backspace или сочетания клавиш Ctrl ? приводит к генерации управляющего символа **DEL**, что заставляет драйвер выполнить управляющее действие **erase** (**^?**) — удалить последний набранный символ. Аналогично, **werase** (**^W**) и **kill** (**^U**) удаляют последнее набранное слово и всю набранную строку соответственно.

Управляющие символы **intr** (**^C**) и **quit** (**^**) — соответственно штатно и аварийно завершают запущенную ранее и выполняющуюся сейчас программу, а символ **susp** (**^Z**) временно приостанавливает выполняющуюся программу, что проиллюстрировано в листинге 2.3.

Листинг 2.3. Управляющие символы ^C и ^

```
finn@ubuntu:~$ dd if=/dev/dvd of=dvd.iso
^C6227352+0 записей получено
6227351+0 записей отправлено
скопировано 3188403712 байт (3,2 GB), 2,72618 с, 1,2 GB/с

finn@ubuntu:~$ dd if=/dev/cdrom of=cd.iso
^\Выход (сделан дамп памяти)
```

Символы **stop** (**^S**) и **start** (**^Q**) управляют потоком вывода (и, как следствие, скроллингом терминала), что можно использовать для временной приостановки вывода команд с многострочным выводом. Однако случайное нажатие **^S** может привести начинающего пользователя в замешательство — будет казаться, что терминал «завис», т. е. отсутствует реакция со стороны операционной системы на какие-либо нажимаемые клавиши и посылаемые символы, тогда как на самом деле отсутствует (приостановлен) лишь ее вывод — до нажатия **^Q**, **^C** или **^**.

Управляющий символ **eof** (**^D**) используется для оповещения драйвера о завершении ввода, при работе с интерактивными (ведущими с пользователем двусторонний попеременный диалог) программами (листинг 2.4).

Листинг 2.4. Управляющий символ ^D

```
finn@ubuntu:~$ mail dketov@gmail.com
Cc: ↵
Subject: Не забыть про Ctrl+D ↵
Символ ^D полезен для mail, at... где еще? ↵
^D
finn@ubuntu:~$ at 21:30
warning: commands will be executed using /bin/sh
at> mplayer ~/sounds/alarm.mp3 ↵
```

```
at> notify-send -i info 'Хватит работать' ↵
at>^D <EOT>
job 4 at Sat Jan 31 21:30:00 2015
finn@ubuntu:~$ lftp ftp.ubuntu.com
lftp ftp.ubuntu.com:~> get /ubuntu/pool/main/m/manpages/manpages_3.74.orig.tar.xz ↵
1291196 байтов перемещено за 6 секунд (226.2Кб/с)
lftp ftp.ubuntu.com:/>^D exit
```

Нужно заметить, что при работе с диалоговыми программами **^C** или **^** завершит выполняющуюся программу (**at**), не дав ей выполнить свое основное действие, или вообще будет проигнорирован (**ftp**, **mail**). Именно символ завершения ввода (**eof**, **e**nd **o**f **f**ile) сообщит драйверу о нежелании больше вести диалог с программой (который в свою очередь сообщит программе об отсутствии для нее вводимых данных).

В очень редких случаях, возможно, потребуется ввести сам управляющий символ, например **^C**, **^** или **^D**, *непосредственно* в выполняющуюся на терминале программу, что невозможно сделать соответствующими клавиатурными комбинациями, потому как управляющие символы будут поглощены драйвером терминала, что приведет к завершению программы, в которую вводятся символы. Для отмены (экранирования) специального назначения управляющих символов в пользу его *непосредственного* (литерального) значения служит управляющий символ (**l**iteral **next**) **lnext** (**^V**), сигнализирующий драйверу терминала об отмене специального назначения следующего за ним символа (листинг 2.5).

Листинг 2.5. Управляющий символ ^V

```
  finn@ubuntu:~$ tee cc.bin
☛ Ctrl+C:^C
  finn@ubuntu:~$ od -ca cc.bin
?  0000000
  finn@ubuntu:~$ tee cc.bin
  Ctrl+C:^V^C ↵ ☚
  Ctrl+C:
  ^D
  finn@ubuntu:~$ od -ca cc.bin
  0000000   C   t   r   l   +   C   : 003 ☚ \n
            C   t   r   l   +   C   : etx   nl
  finn@ubuntu:~$ hd cc.bin
  00000000  43 74 72 6c 2b 43 3a 03 0a                     |Ctrl+C:..|
  00000009                    ↳
```

Реакция драйвера терминала на получаемые управляющие символы и предпринимаемые им управляющие действия (а точнее, наоборот — управляющие символы, закрепленные за управляющими действиями) стандартно предопределена, но почти все эти соответствия могут быть просмотрены и изменены командой **stty(1)**, что иллюстрируется в листинге 2.6.

Листинг 2.6. Утилита stty

```
finn@ubuntu:~$ stty -a
speed 38400 baud; rows 38; columns 136; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <undef>; eol2 = <undef>;
swtch = <undef>; start = ^Q; stop = ^S; susp = ^Z;
rprnt = ^R; werase = ^W; lnext = ^V; flush = ^O; min = 1; time = 0;
-parenb -parodd cs8 hupcl -cstopb cread -clocal -crtscts
-ignbrk brkint -ignpar -parmrk -inpck -istrip -inlcr -igncr icrnl ixon -ixoff -iuclc -ixany
imaxbel iutf8
opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -ofdel nl0 cr0 tab0 bs0 vt0 ff0
isig icanon iexten echo echoe echok -echonl -noflsh -xcase -tostop -echoprt echoctl echoke
```

Кроме того, команда **stty(1)** позволяет получить (а также задать) и другие настройки драйвера терминала:

- скорость приемопередатчика последовательного интерфейса терминала (**speed 38400 baud**);
- количество изображаемых терминалом строк и столбцов (**rows 33; columns 119**);
- флаги режимов работы приемопередатчика интерфейса (**-parenb...hupcl -cstopb...-inpck**);
- флаги режимов обработки вводимых из терминала символов (**-istrip...-igncr icrnl...iutf8**);
- флаги режимов обработки выводимых на терминал символов (**opost...-ofdel**) и пр.

Так, например, флаг **icanon** включает или выключает (**-icanon**) «канонический» (**canon**ical) режим обработки вводимых (input) символов, т. е. возможности редактирования вводимой строки при помощи управляющих символов **^?** и **^U**, а также сигнализацию завершения ввода при помощи **^D**.

Флаг **iexten** включает «расширения» канонического режима стандарта POSIX, т. е. удаление последнего введенного слова при помощи **^W**, перерисовку введенной строки при помощи **^R** и ввод литеральных значений управляющих символов при помощи **^V**.

Управляющие символы **^C**, **^** и **^Z** штатного или аварийного завершения и приостановки выполняющейся программы активируются флагом **isig** (листинг 2.7), разрешающим или запрещающим (**-isig**) посылку сигналов (**sig**nal, *см. разд. 4.8*).

Флаг **icrnl** включает трансляцию вводимого символа **CR** в символ **LF**, что позволяет запускать команды клавишей ↵ENTER (неожиданно, правда?), потому как основной символ для этого действия все же — **LF** (так уж сложилось в UNIX со времен телетайпа ASR-33).

К счастью, большинство из этих параметров и флагов применимы только при работе с настоящими аппаратными терминалами и интерфейсами. Поэтому пояснение их назначения[1] можно опустить хотя бы просто потому, что оно требует дополнительных знаний специфики соответствующей аппаратуры, что не имеет ни особой актуальности, ни является предметом нашего рассмотрения.

Нужно также отметить, что некоторые[2] диалоговые программы «игнорируют» некоторые настройки терминала, например флаг «канонического» режима. Точнее, они всегда работают в «неканоническом» режиме и сами обрабатывают управляющие символы, зачастую переопределяя некоторые из них или добавляя обработку дополнительных, например, для командного интерпретатора **bash(1)** это **^R** (reverse-search-history), **^S** (forward-search-history), **^D** (delete-char) **^L** (clear-screen), **^A** (beginning-of-line), **^E** (end-of-line), **^F** (forward-char) **^B** (backward-char), **^P** (previous-history), **^N** (next-history) и др.

Листинг 2.7. Настройки драйвера терминала

```
finn@ubuntu:~$ stty
speed 38400 baud; line = 0;
iutf8
finn@ubuntu:~$ stty -isig
finn@ubuntu:~$ stty
speed 38400 baud; line = 0;
iutf8
-isig
```

[1] Исчерпывающе описанные в **termios(3)**, вместе с управляющими действиями и предопределенными им управляющими символами.

[2] В большинстве своем те, которые используют библиотеку **readline(3)** для расширенного редактирования вводимой строки или библиотеку **ncurses(3)** для работы с расширенным режимом терминала. К ним относятся командный интерпретатор **bash(1)**, постраничный листатель **less(1)**, текстовые редакторы **vi(1)** и **nano(1)**, Web-браузеры **links(1)** и **lynx(1)**, ftp-клиенты **lftp(1)** и **ncftp(1)**, файловый менеджер **mc(1)** и пр.

```
finn@ubuntu:~$ dd if=/dev/dvd of=dvd.iso
!☛ ^C^\^Z
... ... ...
```

При *выводе* информации на терминал, управляющие символы[1] (в отличие от алфавитно-цифровых символов, подлежащих изображению каким-либо значком) служат для управления терминалом (см. **ascii(7)**, **console_codes(7)** и табл. 2.1). Например, символ **CR** (carriage return) перемещает печатающую головку или курсор терминала в начало строки, символ **LF** (line feed) — в начало новой строки, символ **NL** (nl, new line) — на следующую строку, символ **HT** (horizontal tab) — на несколько символов вправо, а символ **BS** (back space) стирает один символ слева от курсора (или просто перемещает печатающую головку на один символ влево) и т. д.

Так, символ **LF**, используемый в «текстовых» файлах при выводе их на экран, позволяет разделять «логические» строки файла и изображать их на разных «физических» строках терминала, а символы **SO** (shift out) и **SI** (shift in) соответственно включают и выключают альтернативный шрифт терминала, содержащий другие знаки вместо маленьких букв латинского алфавита. Именно поэтому при ошибочном *выводе* на терминал «бинарного» файла, в содержимом которого весьма вероятно встретить символ **SO** ❶, активирующий альтернативный шрифт, возникает ощущение «испорченности» терминала, которую легко починить *выводом* символа **SI**, возвращающего терминал к стандартному шрифту. Для этого достаточно литерально *ввести* символ **SI** ❶, который будет расценен командным интерпретатором как (несуществующая) команда и *выведен* на экран в сообщении об ошибке ❷ (листинг 2.8).

Листинг 2.8. Управляющие символы SO (^N) и SI (^O)

```
finn@ubuntu:~$ cat /etc/localtime
... ❶...
#<#(1#X?|
        *▒3@FP *▒3@*▒! 8@8@LMTMMTMSTMDSTMSKMSDMSMEETEEST
MSK↑4
°i┼┼@┤b┤┼┼:·$ ^V^O ☚❶
❷ : команда не найдена
finn@ubuntu:~$ tee si-and-so.txt
hahaha^V^N+hahaha^V^O
hahaha→▒▒▒▒
finn@ubuntu:~$
```

[1] См. W:[управляющие символы].

2.4. Управляющие последовательности

В расширенном режиме видеотерминалов **W:[VT52]**, **W:[VT100]**, **W:[VT220]** появилась возможность *вывода* символов в произвольное место экрана и использования полужирного, затемненного, негативного, подчеркнутого и других начертаний символов. Возможности *ввода* дополнились функциональными клавишами, клавишами перемещения курсора, дополнительной клавиатуры и пр.

Для этого потребовались дополнительные управляющие символы, которые не поместились в кодировку **ascii(7)**, потому терминалы стали использовать управляющие последовательности символов[1] **console_codes(7)**, предваряемые управляющим символом **ESC** с кодом **0x1B**.

Так, например, последовательность **ESC # 8** вызовет визуальный тест выравнивания краев терминала, заполнением буквой **E** всех строк и столбцов, **ESC c** сбросит терминал в исходное состояние, **ESC [1 m** включит полужирное начертание, **ESC [2 m** — затемненное начертание, **ESC [4 m** — подчеркивание, а **ESC [0 m** вернет стандартное начертание символов.

Листинг 2.9. Управляющие последовательности терминала

```
finn@ubuntu:~$ tee esc.txt
❶ ^V^[[1mbold^V^[[0m,^V^[[2mdim^V^[[0m,^V^[[4munderscore^V^[[0m,^V^[[7mreverse^V^[[0m
bold,dim,underscore,reverse
^D
❷ finn@ubuntu:~$ cat esc.txt
bold,dim,underscore,reverse

finn@ubuntu:~$ od -c esc.txt
❸ 0000000 033  [  1  m  b  o  l  d 033  [  0  m  ,  033  [  2
0000020   m  d  i  m 033  [  0  m  ,  033  [  4  m  u  n  d
0000040   e  r  s  c  o  r  e 033  [  0  m  ,  033  [  7  m
0000060   r  e  v  e  r  s  e 033  [  0  m  \n
0000074
```

Управляющие символы и их последовательности являются обычными байтами и при литеральном вводе ❶ с терминала могут быть сохранены в файл (листинг 2.9), например при помощи команды **tee(1)**. При последующем выводе ❷ на терминал, например при помощи команды **cat(1)**, будут задействованы соответствующие расширенные возможности. Побайтное содержимое файла можно при этом

[1] См. **W:[управляющие последовательности ANSI]**.

увидеть ❸ на терминале посредством восьмеричного **od(1)** или шестнадцатеричного дампа **hexdump(1)**, **hd(1)**.

Многие «дополнительные» клавиши современных терминалов, такие как функциональные F1...F12, клавиши управления курсором ↓ ↑ → ←, скроллингом PgUp PgDn Home End и пр., *генерируют* (листинг 2.10) управляющие последовательности, которые обрабатываются, например, библиотекой **readline(3)** и используются для редактирования командной строки.

Листинг 2.10. Управляющие последовательности клавиатуры

```
   finn@ubuntu:~$ od -a
F1 ^[OQ↵
   0000000 esc   O   Q  nl
   0000004
   finn@ubuntu:~$ od -a
End^[OF↵
   0000000 esc   O   F  nl
   0000004
```

Несмотря на стандартизацию управляющих последовательностей, разные терминалы все же имеют различия, поэтому в операционной системе появились базы данных с описанием свойств и управляющих последовательностей терминалов **termcap(5)** и **terminfo(5)** (листинг 2.11). Узнать ESC-последовательности можно при помощи команды **infocmp(1)**, а вывести их на терминал — включить соответствующий режим — при помощи команды **tput(1)**.

Листинг 2.11. База данных управляющих последовательностей termcap(5)

```
finn@ubuntu:~$ infocmp
#    Reconstructed via infocmp from file: /lib/terminfo/l/linux
linux|linux console,
     am, bce, ccc, eo, mir, msgr, xenl, xon,
     colors#8, it#8, ncv#18, pairs#64,
     ... ... ...
     op=\E[39;49m, rc=\E8, rev=\E[7m ↵, ri=\EM, rmacs=\E[10m,
     ... ... ...
     sgr0=\E[0;10m ↵, smacs=\E[11m, smam=\E[?7h, smir=\E[4h,
     ... ... ...
     smpch=\E[11m, smso=\E[7m, smul=\E[4m ↵, tbc=\E[3g,
```

```
finn@ubuntu:~$ tput smul
finn@ubuntu:~$ tput rev
finn@ubuntu:~$ tput sgr0
finn@ubuntu:~$ tput smul | od -ac
0000000 esc   [   4   m
       033   [   4   m
0000004
```

Примером простейшей программы, использующей управляющие последовательности для форматирования символов при выводе на экран, является утилита просмотра справочных страниц **man(1)**. В качестве более изощренных программ, использующих управление расширенными возможностями терминала, можно привести **less(1)**, **nano(1)**, **mc(1)**, многие из которых используют для этого библиотеку **ncruses(3)**. А самым экстремальным примером использования управляющих последовательностей терминалов является **W:[ASCII-графика]** и **W:[ANSI-графика]**, реализующаяся библиотеками **aalib** и **caca**, при помощи которых на алфавитно-цифровом (!) терминале можно просматривать видеофильмы (листинг 2.12), например, при помощи видеоплеера **mplayer(1)**, поддерживающего эти библиотеки.

Листинг 2.12. Просмотр видео на алфавитно-цифровом терминале

```
finn@ubuntu:~$ setfont koi8r-8x8
finn@ubuntu:~$ mplayer -vo caca http://goo.gl/7BG1Yu
finn@ubuntu:~$ mplayer -vo aa:contrast=50 http://goo.gl/RQgznU
```

2.5. Основной синтаксис командной строки

Основой интерфейса командной строки UNIX является командный интерпретатор (КИ), являющийся первой и главной программой, запускаемой в сеансе пользователя. Двусторонний попеременный диалог с командным интерпретатором начинается с приглашения ❶, в ответ на которое пользователь вводит команду, отправляя ее на выполнение управляющим символом **LF 0x0A**, получает результат ее исполнения на терминале и новое приглашение, сигнализирующее о готовности КИ к исполнению очередной команды (рис. 2.5). Многие другие *диалоговые* программы,

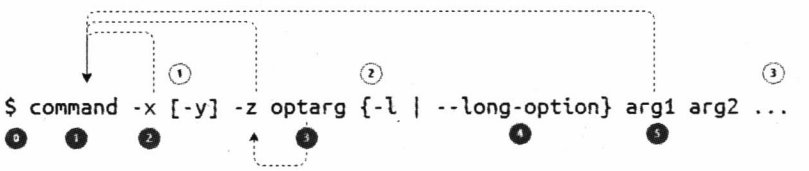


Рис. 2.5. Основной синтаксис командной строки

например **lftp**, так же будут придерживаться синтаксиса и соглашений, принятых в языке командного интерпретатора.

Базовый синтаксис (подчиняющийся второй части стандарта W:[POSIX]) языка любого[1] командного интерпретатора на самом деле достаточно прост и напоминает язык, близкий к естественному. Например, **rm -f -R Изображения Музыка** переводится на человеческий как «удалить (**rm**) без шума и пыли (**-f**) и со всеми потрохами (**-R**) каталоги **Изображения** и **Музыка**». Принято говорить, что команда состоит (см. рис. 2.5) из лексем (**лекс**ических **эле**ментов), разделенных пробельными символами — пробелами **SP 0x20** и табуляциями **HT 0x09** — в любом количестве и сочетании. Первая лексема ❶ — это название команды, за которой следуют ее параметры: сначала опции (они же — ключи, они же — модификаторы) без аргументов ❷ и ❹ или с аргументами ❸, а в конце — аргументы ❺ самой команды. Назва́ние команды — это глагол, указывающий ЧТО делать; опции — это наречия и прочие части речи, объясняющие КАК это делать; и наконец, аргументы задают то, с ЧЕМ это делать. В разнообразной документации при описании синтаксиса команды, ее опций и аргументов принято использовать квадратные скобки ① для указания необязательности опции или аргумента, фигурные скобки ② и вертикальную черту для указания выбора из вариантов и многоточие ③ для указания повторяемости.

Получив команду, интерпретатор определяет, является ли она *псевдонимом*, *встроенной* в интерпретатор, или реализуется *внешней* программой, подлежащей запуску (листинг 2.13).

Листинг 2.13. Утилиты which и type

```
finn@ubuntu:~$ which date
/bin/date
finn@ubuntu:~$ type date
date является /bin/date
finn@ubuntu:~$ type -a ls
ls является алиасом для `ls --color=auto'
ls является /bin/ls
finn@ubuntu:~$ type -a pwd
pwd встроена в оболочку
pwd является /bin/pwd
finn@ubuntu:~$ type which
which является /usr/bin/which
finn@ubuntu:~$ type type
type встроена в оболочку
```

[1] Диалектов языка командного интерпретатора достаточно много: Bourne shell **sh**, Korn shell **ksh**, C shell **csh** и т. д.

При наличии нескольких вариантов команды наивысший приоритет имеют псевдонимы, наименьший — внешние команды. Подстановку псевдонимов можно увидеть, включив трассировку выполнения команд интерпретатора при помощи команды **set** (листинг 2.14).

Листинг 2.14. Подстановка псевдонимов

```
finn@ubuntu:~$ type ls
ls является алиасом для `ls --color=auto'
finn@ubuntu:~$ set -x
finn@ubuntu:~$ ls -ASs
+ ls --color=auto -ASs
итого 36
12 examples.desktop   4 .bashrc         4 .profile      4 .lesshst
 4 .cache             4 .bash_history   4 .bash_logout
```

2.5.1. Опции командной строки

В истории развития операционной системы UNIX программы использовали разные способы задания своих опций:

- односимвольные, например **ls -a -l** (что эквивалентно **ls -l -a** или **ls -al** или **ls -la**);
- многосимвольные, например **find /var -xdev**;
- длинные, например **ps --help**;
- с аргументами, например **kill -n 15 1**, или **kill -n15 1**, или **du -B M**, или **du -BM**, или **find /etc -type d**, или даже **ls --sort=size**;
- «нестандартные», например **set +x**, **tar czf tar.tgz ~** или **dd if=/dev/dvd of=dvd.iso**.

Знак «минус», предваряющий опции, естественно используется для того, чтобы отличать[1] их от аргументов. Среди прочих, он был выбран потому, что редко встречается как первый символ в *аргументах* команд (в качестве которых зачастую выступают имена файлов). И еще потому, что на терминале классической[2] UNIX набор более логичного знака «плюс» (что могло бы означать «включить», «активировать») требовал достаточных усилий по нажатию клавиши ⇧ Shift. В результате получилось, что, например, в команде **set** опция **x** (execution trace) в форме **-x включает**, а **+x выключает** трассировку выполнения команд.

---

[1] В примере с командой **rm** лексема **-R** означает *опцию*, требующую выполнить рекурсию (recursion), но не *аргумент* — объект, подлежащий удалению.

[2] Очередной привет из прошлого от Teletype ASR-33.

В тех редких случаях, когда аргумент команды все же начинается с символа «минус» и тем самым похож на опцию (представим, что нужно выполнить действие над файлом с именем **-=filename=-**), специальная опция **--** сигнализирует о конце списка опций (листинг 2.15), за которым следуют лексемы, обязанные расцениваться как *аргументы*, вне зависимости от их написания.

Листинг 2.15. Конец списка опций

```
finn@ubuntu:~$ stat -=filename=-
stat: неверный ключ — "="
Попробуйте "stat --help" для получения более подробного описания.
finn@ubuntu:~$ stat -- -=filename=-
  Файл: "-=filename=-"
  Размер: 0           Блоков: 0          Блок В/В: 4096   пустой обычный файл
Устройство: fc00h/64512d        Inode: 26870044    Ссылки: 1
Доступ: (0600/-rw-------)  Uid: ( 1000/    finn)   Gid: ( 1000/    finn)
Доступ: 2015-02-24 01:33:21.574640275 +0300
Модифицирован: 2015-02-24 01:33:21.574640275 +0300
Изменён: 2015-02-24 01:33:21.574640275 +0300
 Создан: -
```

Короткие, односимвольные опции (например **-l -a**) без своих аргументов издревле можно было объединять в группы (**-la** или **-al**), однако их в этом случае сложно отличать от многосимвольных (**-xdev**) или односимвольных, склеенных со своими аргументами (**-BM**). Поэтому позже появились длинные (в так называемом GNU-стиле) опции, обозначаемые двумя знаками «минус», позволяющие навести некоторый порядок в виде **--block-size=M** вместо **-BM** или, предположим, **--dont-descent** вместо **-xdev**.

2.6. Справочные системы

Используемые в Linux электронные справочные системы (online help) являются логичным следствием как его родства с семейством операционных систем UNIX — страницы руководства **man(1)** (**man**ual pages), так и принадлежностью к свободному программному обеспечению под эгидой движения GNU — справочная система **info(1)**. Следует отметить, что понятие online в контексте справочных систем вовсе не означает их доступность через Интернет, как это часто, но ошибочно, воспринимается сегодня. В рассматриваемом контексте online означает немедленную доступность справочной информации непосредственно из программного обеспечения, по сравнению со справочной информацией, доступной в печатной, offline, форме.

2.6.1. Система страниц руководства

Самой известной справочной системой, сопровождающей UNIX практически с момента ее рождения, является справочная система страниц руководства, информация из которой доступна при помощи команд **man(1)**, **apropos(1)** и **whatis(1)**. Справочная система **man-pages(7)** состоит из отдельных страниц, посвященных отдельным командам, специальным файлам устройств, конфигурационным файлам, системным и библиотечным вызовам и другим понятиям, которые сгруппированы по восьми (обычно, но есть исключения из правил) секциям. Каждая секция имеет заголовочную страницу **intro**, описывающую назначение самой секции (листинг 2.16).

Листинг 2.16. Секции справочной системы man(1)

```
finn@ubuntu:~$ whatis intro
intro (2)            - Introduction to system calls
intro (7)            - Introduction to overview, conventions, and miscellany ...
intro (8)            - Introduction to administration and privileged commands
intro (3)            - Introduction to library functions
intro (1)            - Introduction to user commands
intro (6)            - Introduction to games
intro (4)            - Introduction to special files
intro (5)            - Introduction to file formats
finn@ubuntu:~$ whatis whatis
whatis (1)           - показывает описания справочн...
finn@ubuntu:~$ whatis apropos
apropos (1)          - поиск в именах справочных стр...
finn@ubuntu:~$ whatis man
man (1)              - доступ к справочным страницам
man (7)              - macros to format man pages
```

Естественным образом справочная система описывает сама себя, поэтому известнейшей «мантрой» в операционной системе является **man man**, т. е. запрос страницы руководства, посвященной самой команде **man(1)**. Сами страницы руководства написаны на языке разметки текста **roff**[1] и размещаются в сжатых gz-файлах «секционных» подкаталогов **man1**, **man2**, ..., **man8** каталога **/usr/share/man** (листинг 2.17). Страницы руководства частично поставляются переведенными на различные национальные языки, отличные от английского.

[1] Система подготовки текстов, доставшаяся в наследство от классической UNIX.

Листинг 2.17. Формат страниц справочной системы man

```
finn@ubuntu:~$ man -w man
/usr/share/man/ru/man1/man.1.gz
finn@ubuntu:~$ file /usr/share/man/ru/man1/man.1.gz
/usr/share/man/ru/man1/man.1.gz : gzip compressed data, from Unix, max compression
finn@ubuntu:~$ file -z /usr/share/man/ru/man1/man.1.gz
/usr/share/man/ru/man1/man.1.gz: troff or preprocessor input, UTF-8 Unicode text (gzip com-
pressed data, from Unix, max compression)
finn@ubuntu:~$ whatis file
file (1)             - determine file type
```

Команда **man(1)**, таким образом, ответственна за поиск указанной пользователем страницы, распаковку ее сжатого файла при помощи распаковщика **gzip(1)**, форматирования при помощи процессора **troff(1)** и (по умолчанию) вывод результата на терминал при помощи постраничного листателя **less(1)**. Именно процессор **troff** умеет посредством управляющих последовательностей нужного терминала раскрашивать вывод страниц руководства правильным образом, а листатель **less** прокручивать подготовленную справку вперед и назад.

Использование языка разметки позволяет форматировать страницу одинаково удобно для просмотра на разных терминалах с различным количеством столбцов, которые учитывает **troff**, и разным количеством строк, учитываемым **less**. Так, например, результат одинаково хорош и на псевдотерминале в окне эмулятора терминала **xterm** или **gnome-terminal** развернутого в любой размер, и на виртуальном терминале консоли с загруженным шрифтом любого размера. Более того, использование универсального языка разметки и соответствующий «драйвер» **troff** позволяет преобразовывать страницы руководства в самые разные виды. Например, в «принтерный» PostScript или PCL, пригодный для печати на принтере с высокой разрешающей способностью, или в HTML[1] для просмотра в html-браузере (листинг 2.18).

Листинг 2.18. Форматирование справочных страниц руководства для печати и для html-браузера

```
finn@ubuntu:~$ man -t man > man.print.1
finn@ubuntu:~$ file man.1.print
man.print.1: PostScript document text conforming DSC level 3.0
finn@ubuntu:~$ man -Tlj4 man > man.print.2
```

[1] При наличии установленного пакета **groff**.

```
finn@ubuntu:~$ file man.print.2
man.print.2: HP PCL printer data
finn@ubuntu:~$ man -Thtml man > man.html
finn@ubuntu:~$ file man.html
man.html: HTML document, ASCII text, with very long lines
```

При просмотре страниц руководства на терминале навигация по изображаемой справочной странице производится так, как предусмотрено используемым листателем, которым по умолчанию в большинстве случаев будет **less(1)**, приемник классического **more(1)**. Одним из самых полезных навигационных действий в справочных системах (табл. 2.2) является поиск регулярных выражений[1]. Собственно страницы руководства разбиты на разделы, заголовки которых размещаются в начале строк и записываются в верхнем регистре. Например, **SYNOPSIS** — краткий обзор, **EXAMPLES** — примеры, **FILES** — используемые программой (конфигурационные) файлы, **ENVIRONMENT** — переменные окружения и пр. Поэтому для перемещения к разделу **TOPIC** очень удобно использовать символ поиска **/** и выражение **^TOPIC**, что буквально означает: найти в начале строки — **^** слово — **TOPIC**.

Таблица 2.2. Клавиши навигации листателей страниц

Навигационное действие	Управляющий символ или клавиша less	Управляющий символ или клавиша more	Управляющий символ или клавиша info
Выход	Q или q	Q или q или ^C	Q или q или ^C
Справка содержимое этой таблицы	h или H	h или ?	h или ? или ^H
Вниз одну строку	CR или j или ↓	CR	↓ или ^N
Вверх одну строку	^P или k или ↑		↑ или ^P
Вниз один экран	f или SPACE или PGDN	f или SPACE	SPACE или PGDN
Вверх один экран	b или PGUP	b	DEL или PGUP
Поиск регулярных выражений вперед	/	/	/ или s
Поиск назад	?		
Повторить поиск	n	n	}

[1] Подробнее о регулярных выражениях *см. в разд. 5.8.*

Таблица 2.2 *(окончание)*

Навигационное действие	Управляющий символ или клавиша less	Управляющий символ или клавиша more	Управляющий символ или клавиша info
Повторить поиск в обратном направлении	N		{
В конец страницы (ноды)	G или > или End		e или End или Esc >
В начало страницы (ноды)	g или < или Home		b или Home или ESC <
Следующая нода	не применимо	не применимо	]
Предыдущая нода	не применимо	не применимо	[

2.6.2. Справочная система GNU

Еще одной системой документации является система **W:[GNU texinfo]**. В отличие от справочника **man**, выступающего по сути *кратким* руководством по командам, их опциям и аргументам, справочник **info** представляет собой *развернутое* руководство с примерами и объяснениями.

Справочная система состоит из предварительно подготовленных (гипер)текстовых страниц, размещенных в сжатых файлах каталога **/usr/share/info**, оглавлением которым служит так называемый «каталог» документации. Каждая страница структурирована при помощи иерархически упорядоченных, так называемых «нод», аналогов книжных разделов/подразделов/глав/секций.

Язык разметки **texinfo**, как и язык **roff** в системе страниц руководства, позволяет подготавливать разные представления справочной информации из единого источника при помощи специальных[1] инструментов, но в отличие от страниц руководства **man** только при наличии исходных файлов документации.

2.6.3. Встроенная справка командного интерпретатора

Как было указано ранее, команды интерпретатору могут приводить к вызову *внешних* программ операционной системы или исполняться непосредственно интерпретатором, будучи *встроенными* в него. Внешние программы зачастую документируются отдельными страницами руководства **man** или отдельными справочны-

[1] При наличии установленного пакета **texinfo** и исходных файлов документации **.texi**.

ми страницами **info**, тогда как встроенные команды являются частью интерпретатора и естественным образом описываются все вместе на справочной странице, посвященной интерпретатору (листинг 2.19).

Листинг 2.19. Справка по встроенным командам интерпретатора

```
finn@ubuntu:~$ type cd
cd встроена в оболочку
finn@ubuntu:~$ man cd
Нет справочной страницы для cd
finn@ubuntu:~$ whatis cd
cd: ничего подходящего не найдено.
finn@ubuntu:~$ man bash
                    ...                    ...                    ...
SHELL BUILTIN COMMANDS
       Unless otherwise noted, each builtin command documented in this section
       as accepting options preceded by - accepts -- to signify the end of the
       options.  The  :, true, false, and test builtins do not accept options
       and do not treat -- specially.  The exit, logout, break, continue, let,
                    ...                    ...                    ...
       cd [-L|[-P [-e]]] [dir]
              Change  the  current directory to dir.  The variable HOME is the
              default dir.  The variable CDPATH defines the  search  path  for
              the  directory  containing  dir.  Alternative directory names in
              CDPATH are separated by a colon (:).  A null directory  name  in
              CDPATH  is  the  same as the current directory, i.e., ``.''.  If
                    ...                    ...                    ...
```

Однако обращаться каждый раз к весьма внушительной справке по командному интерпретатору не совсем удобно, поэтому встроенная в командный интерпретатор команда **help** позволяет получить краткую справку по встроенным командам интерпретатора (листинг 2.20).

Листинг 2.20. Встроенная справка командного интерпретатора

```
finn@ubuntu:~$ help -d help
help - Display information about builtin commands.
finn@ubuntu:~$ help -d cd
cd - Change the shell working directory.
finn@ubuntu:~$ help cd
```

```
cd: cd [-L|[-P [-e]]] [каталог]
    Change the shell working directory.

    Change the current directory to DIR.  The default DIR is the value of the
    HOME shell variable.

    The variable CDPATH defines the search path for the directory containing
    DIR.  Alternative directory names in CDPATH are separated by a colon (:).
                ...                 ...                 ...
```

2.7. Пользователи и группы

Как было указано ранее, для начала работы в многопользовательской операционной системе пользователю необходимо «зарегистрироваться», предъявляя имя своей пользовательской учетной записи и пароль, подтверждающий право на ее использование. В результате регистрации в системе запускается командный интерпретатор — первая программа пользовательского сеанса.

Учетные записи (УЗ) служат для *авторизации*, т. е. для разграничения прав доступа субъектов (процессов пользователей или процессов системных служб) к объектам (файлам, другим процессам, системным вызовам и пр.).

Различают пользовательские и групповые учетные записи, при этом каждая пользовательская учетная запись *идентифицируется* уникальным числовым «пользовательским идентификатором» — UID (**U**ser **Id**entifier), а каждая групповая — таким же уникальным числовым «групповым идентификатором» GID (**G**roup **Id**entifier). Именно эти числовые идентификаторы и используются ядром операционной системы при определении и проверке прав доступа субъектов относительно объектов (листинг 2.21).

Учетные записи пользователей используются для *аутентификации* (проверки подлинности) их при регистрации в системе по *имени* и *паролю*, а учетные записи групп — для классификации пользовательских УЗ (по функциям, задачам, ролям, проектам или другими способами) и последующей раздачи прав доступа этим «классам» пользователей.

Листинг 2.21. Пользовательские идентификаторы UID и GID первичной и дополнительных групп

```
finn@ubuntu:~$ id
uid=1000(finn) gid=1000(finn) группы=1000(finn),4(adm),...,109(sambashare),110(admin)
```

Каждая учетная запись пользователя обязательно связана с одной групповой учетной записью, так называемой «первичной» группы пользователя. Исторически сло-

жилось, что в ранней UNIX членство пользователей в группах определялось динамически, т. е. после регистрации пользователя в системе ему (точнее — его процессу, *см. разд. 4.5.1*) выдавался идентификатор (и, как следствие, права доступа) только одной группы. Для выполнения действий в другой роли (получения других групповых идентификаторов) нужно было «заново зарегистрироваться в группе» при помощи команды **newgrp**, предъявляя имя и пароль (!) группы. Позднее (и до сих пор) членство в группе стало статическим, т. е. при регистрации в системе пользователю выдают идентификаторы всех групп, в которых он состоит, при этом первая группа носит название первичной (primary), а остальные — дополнительных (supplementary).

Кроме этих основных свойств, каждая учетная запись характеризуется именем *домашнего каталога* и именем *командного интерпретатора* (запускаемого при регистрации в системе). Дополнительно, учетная запись пользователя может содержать полное имя пользователя, рабочий и домашний телефоны, рабочий адрес и прочую информацию, которую можно посмотреть при помощи **pinky(1)** и **finger(1)** (листинг 2.22), а изменить при помощи **chfn(1)**.

Листинг 2.22. Свойства учетной записи пользователя

```
finn@ubuntu:~$ finger dvk
Login: dvk                              Name: Dmitry V. Ketov
Directory: /home/dvk                            Shell: /bin/bash
Office Phone: +7(812)703-02-02          Home Phone: ---
On since Tue Mar  3 13:48 (MSK) on tty2    28 seconds idle
On since Mon Mar  9 23:11 (MSK) on tty1    20 seconds idle
     (messages off)
On since Mon Mar  9 23:10 (MSK) on pts/1 from :1.0
   4 seconds idle
On since Sat Feb 14 16:08 (MSK) on tty7    28 days 21 hours idle
Mail last read Sun Mar  8 21:10 2015 (MSK)
No Plan.
```

Учетная запись системного администратора с привилегированными (а точнее — неограниченными в буквальном смысле) правами доступа обычно называется **root** и всегда имеет идентификатор **UID=0**. Учетные записи, «от лица которых» выполняются процессы системных служб, называются псевдопользовательскими и идентифицируются в диапазоне **UID=1—499** или **UID=1—999**, а начиная с **UID=500** (redhat) или **UID=1000** (debian) и далее идентифицируются учетные записи обычных пользователей.

2.7.1. Передача полномочий

Для выполнения определенных административных (привилегированных) действий, например для установки системного времени при помощи команды **date(1)**, нужны права доступа к определенным системным вызовам. В классической UNIX были предусмотрены простые правила[1] разграничения «все или ничего», т. е. *все* привилегированные действия были разрешены суперпользователю **root** с **UID=0**, и *никакие* привилегированные — всем остальным пользователям. В этих и подобных ситуациях для администрирования операционной системы непривилегированным пользователям необходимо временно воспользоваться правами суперпользователя, что реализуется посредством классической команды *явной* передачи полномочий **su(1)** — switch **u**ser, или более поздней команды **sudo(1)** — switch **u**ser **do** *контролируемой* передачи полномочий.

Основное различие между **su(1)** и **sudo(1)** заключается в том, что команда **su** реализует «повторную регистрацию в системе», требуя указать имя и ввести пароль того пользователя, чьи полномочия нужно получить. Напротив, команда **sudo** реализует явные правила **sudoers(5)** передачи полномочий, указанные в файле **/etc/sudoers**, и требует подтвердить передачу полномочий паролем того пользователя, который получает передаваемые полномочия (листинг 2.23).

Листинг 2.23. Передача полномочий

```
iceking@ubuntu:~$ su -l finn
Пароль: <пароль finn'a>
finn@ubuntu:~$ id
uid=1001(finn) gid=1001(finn) группы=1001(finn)
finn@ubuntu:~$ su -l jake
Пароль: <пароль jake'a>
jake@ubuntu:~$ id
uid=1002(jake) gid=1002(jake) группы=1002(jake)
jake@ubuntu:~$ ^D
finn@ubuntu:~$ ^D
iceking@ubuntu:~$ sudo -i -u finn
[sudo] password for iceking: <пароль iceking'a>
finn@ubuntu:~$ id
uid=1001(finn) gid=1001(finn) группы=1001(finn)
finn@ubuntu:~$ sudo -i -u jake
```

[1] На текущий момент времени в Linux реализована система POSIX.1e привилегий **capabilities(7)**, подробнее *см. разд. 4.5.2*.

```
[sudo] password for finn: <пароль finn'a>
finn is not in the sudoers file.  This incident will be reported.
```

Нужно заметить, что в Ubuntu Linux пароль суперпользователя **root** заблокирован, что не позволяет использовать учетную запись как «обычную» для регистрации в системе и превращает ее в «ролевую». Как следствие, привилегиями «роли» суперпользователя можно пользоваться лишь при помощи **sudo** и только непривилегированным пользователям, явно указанным в правилах передачи **sudoers(5)**.

2.7.2. Хранилища учетных записей

Информация об идентификаторах UID и GID, именах пользователей и групп, их паролях и прочих свойствах учетных записей размещается (в простейшем случае) в файловых «хранилищах» — обычных текстовых файлах каталога **/etc**, формируя базы данных пользовательских **/etc/passwd**, **/etc/shadow** и групповых **/etc/group**, **/etc/gshadow** учетных записей. Формат и структура этих файлов хорошо документированы в руководстве **passwd(5)**, **shadow(5)** и **group(5)**, **gshadow(5)** и представляют собой простейшие текстовые таблицы, где свойства каждой учетной записи представлены набором столбцов одной строки, разделенных символом двоеточия : (листинг 2.24).

Листинг 2.24. Базы данных пользовательских учетных записей

```
finn@ubuntu:~$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
     ...              ...
dvk:x:1000:1000:Dmitry V. Ketov,,+7(812)703-02-02,,:/home/dvk:/bin/bash
     ...              ...
finn@ubuntu:~$ cat /etc/group
     ...
dvk:x:1000:
lpadmin:x:108:dvk
sambashare:x:109:dvk
admin:x:110:dvk
     ...
```

При использовании «коммутатора службы имен» (NSS, **W**:**[Name Service Switch]**) имеется возможность хранить базы данных учетных записей в любых хранилищах, включая сетевые службы каталогов NIS, NIS+, LDAP, активный каталог Microsoft Windows и даже реляционные сетевые базы данных SQL — при помощи соответствующих модулей NSS (листинг 2.25) и согласно настройкам коммутатора **nsswitch.conf(5)**.

Листинг 2.25. Хранилища пользовательских учетных записей и модули NSS

```
finn@ubuntu:~$ cat /etc/nsswitch.conf
    ...
passwd:         compat
group:          compat
shadow:         compat
    ...
finn@ubuntu:~$ find /lib -name 'libnss_*'
    ...
/lib/i386-linux-gnu/libnss_nisplus.so .2
/lib/i386-linux-gnu/libnss_compat.so.2
    ...
/lib/i386-linux-gnu/libnss_files.so.2
/lib/i386-linux-gnu/libnss_nis.so.2
/lib/i386-linux-gnu/libnss_winbind.so.2
    ...
```

2.8. Переменные окружения и конфигурационные dot-файлы

Для *одноразовой* параметризации выполнения команд служат их *индивидуальные* ключи, указываемые каждый раз при запуске команды, но иногда требуется установить некий параметр, который бы действовал в течение всего *сеанса* работы пользователя с системой, или *общий* параметр, который действовал бы для всех команд, запускаемых в сеансе. Таким механизмом является окружение **environ(7)** и переменные окружения, значения которых можно увидеть при помощи команды **env(1)**. Переменные окружения обычно документируются на странице руководства к тем программам, на которые воздействуют, как правило, в разделе **ENVIRONMENT**.

Например, переменная окружения **PATH** содержит перечисление разделенных символом **:** имен каталогов, где любой командный интерпретатор ищет одноименные запускаемым командам программы (листинг 2.26).

Листинг 2.26. Переменная окружения PATH

```
finn@ubuntu:~$ date
Чт. марта 12 01:24:47 MSK 2015
finn@ubuntu:~$ help -d unset
unset - Unset values and attributes of shell variables and functions.
```

```
finn@ubuntu:~$ unset PATH
finn@ubuntu:~$ date
bash: date: Нет такого файла или каталога
```

Переменная окружения **LANG** содержит идентификатор языка, на котором программы стараются общаться с пользователем (листинг 2.27), например **man(1)** ищет перевод страницы руководства.

Листинг 2.27. Переменная окружения LANG

```
finn@ubuntu:~$ LANG=ja_JP.UTF-8
finn@ubuntu:~$ date
2015年  9月 12日 土曜日 17:00:09 MSK
finn@ubuntu:~$ LANG=ka_GE.UTF-8
finn@ubuntu:~$ cal
  სექტემბერი 2015
კვ ორ სა ოთ ხუ პა შა
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30
finn@ubuntu:~$ unset LANG
finn@ubuntu:~$ whatis man
man (7)              - macros to format man pages
man (1)              - an interface to the on-line reference manuals
finn@ubuntu:~$ export LANG=ru_RU.UTF-8
finn@ubuntu:~$ whatis man
man (1)              - доступ к справочным страницам
man (7)              - macros to format man pages
```

Переменная окружения **PAGER** указывает имя программы «листателя», использующегося другими программами, вывод которых не умещается на один экран. Так, например, поступает **man(1)** при отображении отформатированной страницы руководства, **mail(1)** при просмотре длинного письма, **mysql(1)** или **psql(1)** при выводе большого количества результирующих строк ответа за запрос к базе данных. Наиболее распространенным листателем является **less(1)**, используемый как замена менее удобного «классического» **more(1)**.

Переменные окружения **EDITOR** и **VISUAL** указывают имя текстового редактора, который будет вызван другими программами при необходимости редактировать текст.

Например, **crontab(1)** использует указанный таким образом редактор для изменения списка периодических заданий, **mail(1)** для редактирования отправляемого сообщения, **mysql(1)** или **psql(1)** для редактирования длинных запросов к базе данных, а **lftp(1)** для редактирования списка «закладок». Очень часто в качестве редактора используется «классический» и достаточно непривычный **vi(1)**, который можно таким образом заменить более удобным **nano(1)**.

Переменная **BROWSER** указывает имя просмотрщика HTML (листинг 2.28), который будет использован другими программами при необходимости показать HTML-страницу, например, отформатированную таким образом страницу руководства **man(1)**.

Листинг 2.28. Переменная окружения BROWSER

```
finn@ubuntu:~$ man -H ls
sh: 1: exec: www-browser: not found
man: couldn't execute any browser from exec www-browser
finn@ubuntu:~$ export BROWSER=chromium-browser
finn@ubuntu:~$ man -H ls
В текущем сеансе браузера создано новое окно.
```

Некоторые программы имеют специальную переменную окружения, которая содержит ключи, применяемые каждый раз при вызове программы, например **MANOPT** для **man(1)**. Такие переменные для программ **XXX** чаще всего имеют имя **XXXOPT** или **XXX_OPTIONS** или даже **XXX**, например **ZIPOPT** для **zip(1)**, **TAR_OPTIONS** для **tar(1)**, **GZIP** для gzip(1) и **LESS/MORE** для **less(1)** и **more(1)** соответственно.

Установив, например, **MANOPT=-H** (**BROWSER=chromium-browser**, или **BROWSER=firefox**, или **BROWSER=links**, или **BROWSER=lynx**), можно просматривать страницы руководства в (одном из указанных графическом и/или текстовом) Web-браузере, а установив **GZIP=-9**, можно заставить упаковщик всегда использовать девятую (самую сильную, но самую медленную) степень сжатия.

Переменная окружения **PS1** изменяет приглашение командного интерпретатора и может содержать подстановки (документированные в **bash(1)**, см. раздел **PROMPTING**), например, **\u** — имя зарегистрированного в системе пользователя, **\h** — короткое собственное имя компьютера, **\w** — имя текущего каталога, **\$** — символ приглашения **$** для обычного пользователя и **#** для суперпользователя root, **\t** — время в 24-часовом формате, **\e** — управляющий символ **ESC** и пр.

Используя подстановки PS1 и управляющие **ESC**-последовательности **console_codes(7)** или воспользовавшись базой данных управляющих **ESC**-последовательностей **terminfo(5)** и командой **tput(1)**, можно модифицировать приглашение по своему предпочтению (листинг 2.29).

Листинг 2.29. Переменная окружения PS1 (вариант 1)

```
finn@ubuntu:~$ man 5 terminfo
     ...                    ...                    ...
enter_reverse_mode          rev        mr          turn on reverse
                                                   video mode
     ...                    ...                    ...
enter_dim_mode              dim        mh          turn on half-bright
                                                   mode
     ...                    ...                    ...
exit_attribute_mode         sgr0       me          turn off all
                                                   attributes
     ...                    ...                    ...
finn@ubuntu:~$ tput rev | od -a
0000000 esc   [   7   m
0000004
finn@ubuntu:~$ tput dim | od -a
0000000 esc   [   2   m
0000004
finn@ubuntu:~$ tput sgr0 | od -a
0000000 esc   (   B esc   [   m
0000006
finn@ubuntu:~$ PS1='\e[2m\t\n\e(B\e[m\u@\h \e[7m\w\e(B\e[m \$ '
13:17:25
finn@ubuntu ~ $ cd /etc
13:17:25
finn@ubuntu /etc $
```

Или даже можно воспользоваться поддержкой символов **unicode(7)** в **UTF-8(7)** представлении на терминале[1] (листинг 2.30).

Листинг 2.30. Переменная окружения PS1 (вариант 2)

```
finn@ubuntu:~$ stty
speed 38400 baud; line = 0;
eol = M-^?; eol2 = M-^?; swtch = M-^?;
```

[1] В этом примере эмулятор терминала в графическом интерфейсе по умолчанию имеет нужные шрифты, а на консоли необходимо загрузить подходящий Unicode-шрифт, например, командой **setfont Uni2-Terminus16**.

```
ixany iutf8 ⏎
finn@ubuntu:~$ PS1='\u\342\230\273 \h:\w\$ '
finn☻ubuntu:~$
```

Переменная окружения **TERM** устанавливает имя терминала, по которому программы, использующие управляющие ESC-последовательности (например, файловый менеджер **mc(1)**), берут их значения из базы данных **terminfo(5)**. Для эмуляторов терминала в графическом интерфейсе ее значение обычно **TERM=xterm** или **TERM=xterm-color**, для консоли Linux **TERM=linux**, а для настоящего аппаратного терминала W:[VT100] **TERM=vt100**. При неправильном значении переменной (листинг 2.31), например, могут «перестать работать» функциональные клавиши, просто потому что программа будет ожидать поступление определенной ESC-последовательности ❶, соответствующей функциональной клавише, а в реальности будет поступать другая ❷.

Листинг 2.31. Переменная окружения TERM

```
finn@ubuntu:~$ env
     ...
TERM=xterm
     ...
finn@ubuntu:~$ TERM=linux
finn@ubuntu:~$ infocmp
                    ...                          ...
❶       kf18=\E[32~, kf19=\E[33~, kf2=\E[[B ⏎, kf20=\E[34~,
                ...                          ...
finn@ubuntu:~$ od -a
F↵
  ^D
❷ 0000000 esc   O   Q  ⏎ nl
0000004
```

Стоит отметить, что переменные окружения сохраняют свои установленные значения в оперативной памяти командного интерпретатора и теряют их при завершении *сеанса*. Для установки *постоянных* значений параметров программ логично поместить их в какое-либо долгосрочное хранилище, например в файлы на диске. Специальные файлы и каталоги, сохраняющие конфигурационные параметры, по соглашению имеют имена, начинающиеся с точки (dot — точка), располагаются в домашнем каталоге пользователя и носят название dot-файлов (листинг 2.32).

Листинг 2.32. Конфигурационные dot-файлы

```
finn@ubuntu:~$ ls -A
     ...
.bash_history
.bash_logout
.bashrc
     ...
.profile

.lftp
.ssh
finn@ubuntu:~$ file .profile .bashrc .lftp .ssh
.profile: ASCII English text
.bashrc:  ASCII English text
.lftp:    directory
.ssh:     directory
```

Некоторые программы, например **lftp(1)**, **ssh(1)** или **ssh(1)**, имеют собственные конфигурационные файлы и/или каталоги, тогда как простейшие **less(1)**, **tar(1)** и **zip(1)** предполагают, что *«постоянные»* параметры по-прежнему задаются при помощи переменных окружения **LESS**, **TAR_OPTIONS** и **ZIPOPT**.

В таких случаях «постоянные» значения переменных сохраняются в каком-либо конфигурационном файле командного интерпретатора **bash**, например, считываемом в начале сеанса — **.profile** или при каждом запуске — **.bashrc** (листинг 2.33).

Листинг 2.33. Конфигурационные dot-файлы интерпретатора bash

```
finn@ubuntu:~$ less ~/.bashrc
...
GZIP=-9v
PS1='\e[2m\t\e(B\e[m \u@\h \e[7m\w\e(B\e[m \$
...
```

Конфигурационные dot-файлы представляют собой текст на некотором языке, понятном конфигурируемой программе, например, язык командного интерпретатора **bash(1)** используется в **.profile** и **.bashrc**.

В большинстве случаев имена dot-файлов и их язык документируются в страницах руководства к «их» программам, обычно в разделе **FILES**. Нередко конфигурационному файлу посвящается отдельная страница в пятой секции, например, **nanorc(5)** для dot-файла **.nanorc** текстового редактора **nano(1)** или **netrc(5)** для файла **.netrc** FTP-клиентов **ftp(1)** и **lftp(1)**, или **ssh_config(5)** для **.ssh/config** SSH-клиента **ssh(1)**.

В отдельных случаях, когда конфигурируется не конкретная программа, а общая для многих программ библиотека, например **readline(3)**[1], название переменных окружения, имя и язык конфигурационного файла можно получить (**INPUTRC** и **.inputrc** для **readline(3)**, соответственно) из страницы руководства самой библиотеки.

2.9. В заключение

Командный интерфейс Linux, каким бы «страшным» не казался, в реальности удивительно функционален для решения массы разнообразных задач, хотя он и не решает абсолютно все задачи одинаково эффективно. Например, его невероятно сложно и неудобно использовать для обработки графической информации, когда при взаимодействии с пользователем требуется ввести колоссальное количество «графических» данных, например, указать обтравочный[2] контур. В этом случае графический интерфейс с «непосредственным» манипулированием подойдет гораздо лучше.

Для начинающего пользователя интерфейс командной строки действительно является *непривычным*, что зачастую путают с *неудобством*, так толком и не разобравшись со всеми его возможностями. Вся сила языка командного интерпретатора в полной мере раскрывается в *главе 5*, до освоения которой читателю предлагается не делать скоропалительных выводов.

Естественные языки, которые используют люди для взаимодействия между собой, на порядок сложнее формального командного языка операционной системы. Однако использование глаголов (команд), существительных (аргументов) и наречий (опций) родного языка мало у кого вызывает чувство неудобства. Наоборот, странным покажется тот человеческий индивидуум, который попытается в обществе использовать непосредственное манипулирование, например, указывая (щелкая) пальцем в магазине на товары (значки) и мыча что-то нечленораздельное. Скорее всего, мы примем его за иностранца (или это будет ребенок), еще не в полной мере владеющего языком.

Именно *командный интерфейс* в современном виде — аудиоформе — больше не является уделом художественных фантастических произведений, где капитаны межгалактических кораблей командуют кораблям «включить защитное поле». Теперь мы все можем при помощи командного аудиоинтерфейса смартфона найти ближайшую пиццерию или маршрут к нужному месту. Надеюсь, что и алфавитно-цифровая форма командного интерфейса, доставшаяся «в наследство» от UNIX из 70-х годов прошлого века, вас тоже не особо испугает. Ok, Google?

---

[1] **readline(3)** — библиотека расширенного редактирования вводимой командной строки, которая используется **bash(1)**, **lftp(1)**, **mysql(1)** и пр.

[2] Выделить пушистого зверька — песца, сфотографированного сидящим в траве.

Глава 3

Подсистемы управления файлами и вводом-выводом

3.1. Файлы и дерево каталогов

Все операционные системы семейства **W:[UNIX]**, включая Linux, базируются на одной универсальной идее, заложенной в их общем предке, определившем основные черты семейства — операционной системе UNICS. В аббревиатуре UNICS[1], или же **UN**iplexed **I**nformation & **C**omputing **S**ervice, центральное место занимает идея «uniplex»ирования, или же односоставности (односложности) — идея решать разные задачи единым способом.

Одним из выражений этой идеи является утверждение о том, что *информация есть файл*, откуда бы эта информация в систему не поступала. При помощи файлов обеспечивается доступ к информации на устройствах хранения (записанной ранее), информации с устройств связи (принимаемой из каналов связи в реальном времени), информации из любых других источников. *Файл*, таким образом, является единицей обеспечения *доступа* к информации, а не единицей ее хранения, как в других операционных системах.

Одни файлы обеспечивают доступ к информации, хранимой на разнообразных носителях: магнитных дисках и дискетах, оптических CD/DVD/BD, твердотельных «дисках» и пр. Другие файлы обеспечивают доступ к информации, поступающей из/в устройств ввода-вывода — клавиатур, манипуляторов «мышь», тачпадов, сенсорных экранов, последовательных и параллельных портов, видеокамер, звуковых карт и пр. Особенные файлы обеспечивают доступ к информации о сущностях ядра операционной системы (процессы, нити, модули, драйвера и пр.).

Так или иначе, все файлы одинаково идентифицируются своими *именами*, упорядоченными в форме единой и единственной иерархической структуры, называемой *деревом каталогов* (рис. 3.1).

---

[1] Более поздняя аббревиатура UNIX произносится идентично, но на одну букву короче.

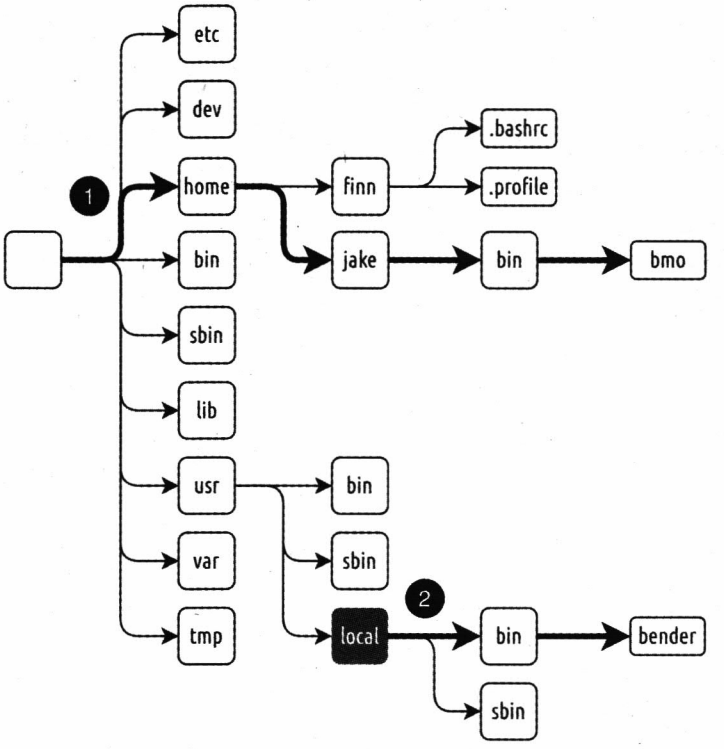


Рис. 3.1. Абсолютное путевое имя и относительное путевое имя

3.1.1. Путевые имена файлов

Глобально уникальным идентификатором файла в пределах операционной системы является его *абсолютное путевое имя*, вычисляющееся как путь от корня дерева каталогов до целевого файла, включая начало и конец пути.

Необходимо акцентировать внимание на том, что имя у корневого каталога отсутствует, т. е. является пустой строкой . Таким образом, абсолютное ❶ путевое имя файла `bmo` (см. рис. 3.1) записывается как разделенные символом **/** имена всех каталогов пути и имя самого файла, включая концы — ABS: /home/jake/bin/bmo.

Относительное путевое имя вычисляется как остаток пути от некоторого заранее заданного (называемого *рабочим*, WD — **w**orking **d**irectory) каталога до целевого файла, включая конец пути. Для рабочего каталога WD: **/usr/local** относительное ❷ путевое имя файла `bender` записывается как разделенные символом **/** имена всех каталогов остатка пути и имя самого файла — REL: bin/bender.

Для проверки правильности записи путевых имен всегда можно воспользоваться проверкой ABS = WD ◢ REL, например **/usr/local/bin/bender = /usr/local** ◢ bin/bender.

Некоторые каталоги дерева (например, каталоги первого уровня) носят устоявшиеся в семействе операционных систем UNIX имена (см. **hier(7)**) и дифференцируют содержание по смысловому признаку.

Например, каталог **/bin** (**bin**ary) предназначен для *системных* программ общего назначения, каталог **/usr**[1]**/bin** — для *прикладных* (условно) программ общего назначения, каталог **/usr/local/bin** — для *«локально»*[2] установленных прикладных программ общего назначения, а каталоги **bin** внутри домашних каталогов пользователей — для программ *персонального* назначения.

Аналогично определяется назначение каталогов **/sbin**, **/usr/sbin**, **/usr/local/sbin** с той лишь разницей, что каталоги **sbin** расшифровываются как superuser's **bin**aries и предназначаются для программ системного администрирования: системных, прикладных и «локально установленных» соответственно. Каталоги **/lib**, **/usr/lib** и **/usr/local/lib** аналогично содержат *системные* и *прикладные* библиотеки.

Каталог **/etc** содержит общесистемные конфигурационные файлы и с полным правом может[3] расшифровываться как **e**ditable **t**ext **c**onfiguration.

Каталог **/home** является контейнером домашних каталогов пользователей (кроме суперпользователя **root**). Каталог **/var** является хранилищем динамических данных (журнальные файлы **/var/log**, почтовые ящики **/var/mail**, разнообразные очереди **/var/spool** и тому подобное), а каталог **/tmp** является хранилищем временных данных.

Каталоги **/dev**, **/proc** и **/sys** содержат специальные файлы устройств[4] и псевдофайловых систем **proc** и **sysfs**.

3.2. Типы файлов

Файлы, как единицы обеспечения доступа к данным, различаются операционной системой по *типам*, указывающим источник информации. *Обычные* (regular) файлы и *каталоги* (directory) обеспечивают сохранение информации на тех или иных носителях. *Специальные файлы устройств* (special device file) позволяют обмениваться информацией с тем или иным аппаратным устройством ввода-вывода, а *именованные каналы* и *файловые сокеты* предназначены для обмена информацией между процессом одной программы и процессами других программ.

---

[1] **usr** — в ранней UNIX сокращение от **u**nix **s**ource **r**epositrary, современное сокращение от **u**nix **s**ystem **r**esources.

[2] Установленных системным администратором из сторонних источников, т. е. не из дистрибутива системы.

[3] **etc** — в ранней UNIX сокращение от лат. ***et c**etera* (и тому подобное) — содержал файлы, которым не нашлось место в других каталогах.

[4] Подробнее *см. разд. 3.2.5* и *3.4.4*.

В примере из листинга 3.1 в полном (**-l**, **l**ong) выводе команды **ls(1)** проиллюстрирован признак типа файла. Символом - обозначается обычный файл, символами **b** или **c** — специальные файлы блочного (**b**lock) или символьного (**c**haracter) устройств, символом **p** — именованный канал (**p**ipe), символом **s** — сокет (socket), а символом **l** — символическая ссылка (**l**ink).

Листинг 3.1. Признак типа файлов ls

```
finn@ubuntu:~$ ls -l /bin/ls /dev/sda /dev/tty /sbin/halt
-rwxr-xr-x 1 root   root   104508 нояб. 20  2012 /bin/ls
brw-rw---- 1 root   disk     8, 0 марта 24 23:45 /dev/sda
crw-rw-rw- 1 root   tty      5, 0 марта 27 13:47 /dev/tty
lrwxrwxrwx 1 root   root        6 янв.  18  2013 /sbin/halt -> reboot
finn@ubuntu:~$ ls -l /run/screen/S-finn/11322.pts-5.ubuntu /run/udev/control
prwx------ 1 finn   finn        0 марта 27 13:47 /run/screen/S-finn/11322.pts-5.ubuntu
srwxr-xr-x 1 root   root        0 марта 24 23:45 /run/udev/control
```

3.2.1. Обычные файлы

Обычные файлы содержат пользовательскую информацию: текст, изображения, звук, видео и прочие данные в виде *набора байтов* (см. ② на рис. 3.2, листинг 3.2). За структуру содержания и имена обычных файлов ответственны прикладные программы, а операционная система не накладывает никаких ограничений.

Листинг 3.2. Содержание обычных файлов

```
finn@ubuntu:~$ file /usr/share/man/man1/file.1.gz
/usr/share/man/man1/file.1.gz: gzip compressed data, from Unix, max compression
finn@ubuntu:~$ file /etc/passwd
/etc/passwd: ASCII text
finn@ubuntu:~$ file /etc/ls
/bin/ls: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked
(uses shared libs), for GNU/Linux 2.6.24,
BuildID[sha1]=0x83531f308f1fa18221be53eaf399303400c14638, stripped
finn@ubuntu:~$ file /usr/share/sounds/alsa/Noise.wav
/usr/share/sounds/alsa/Noise.wav: RIFF (little-endian) data, WAVE audio, Microsoft PCM,
16 bit, mono 48000 Hz
finn@ubuntu:~$ file /usr/share/backgrounds/Floorboards_by_Dawid_Huczyński.jpg
/usr/share/b...s/Floorb...Huczyński.jpg: JPEG image data, JFIF standard 1.01
```

Создать обычный файл можно при помощи любой программы, сохраняющей информацию в файл, например посредством текстовых редакторов **vi(1)**, **nano(1)** или

mcedit(1). Для создания пустого обычного файла можно воспользоваться командой touch(1) — см. листинг 3.5. Для удаления обычного файла предназначается команда rm(1) — см. листинг 3.6.

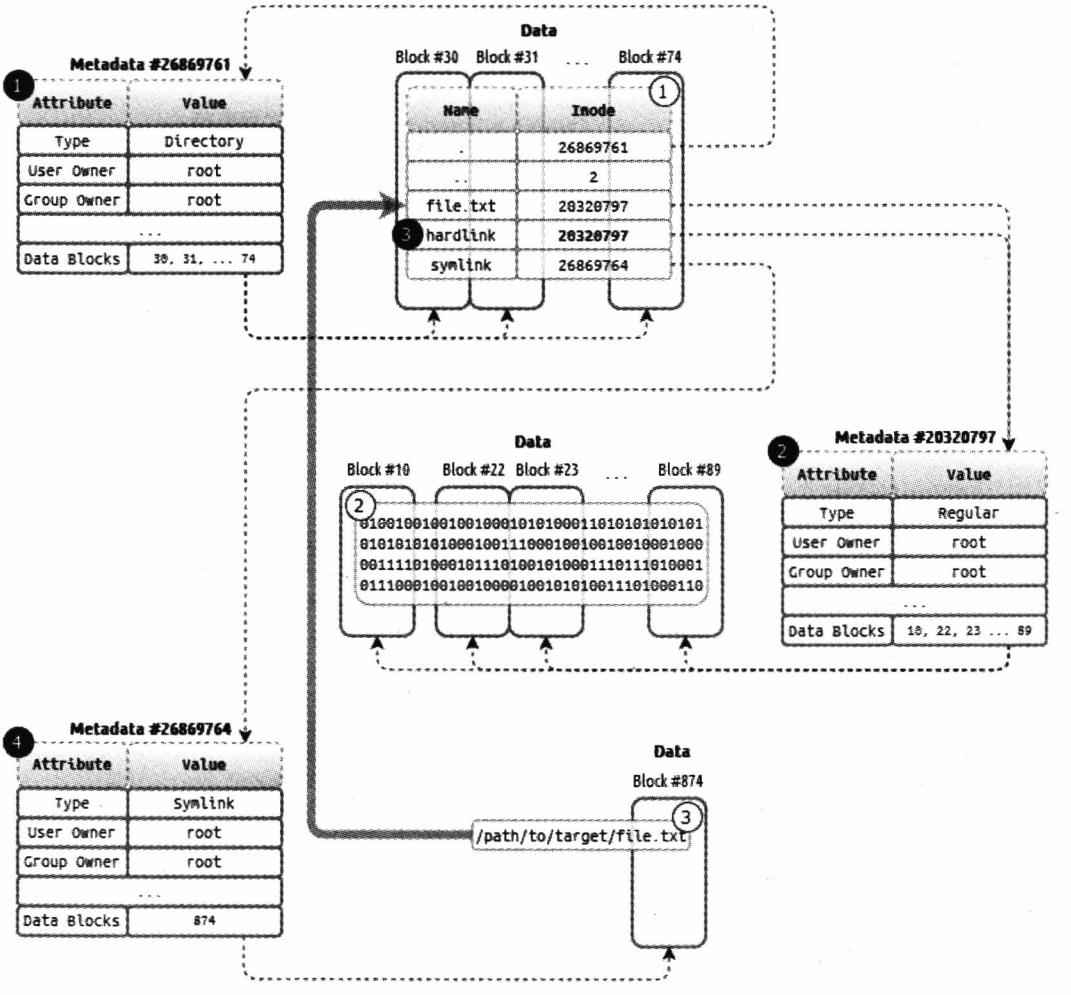


Рис. 3.2. Имена, метаданные и данные файлов

3.2.2. Каталоги

Файлы-каталоги, в отличие от обычных файлов, имеют служебное для операционной системы содержимое — *таблицу имен* (см. ① на рис. 3.2) файлов и соответствующих им номеров индексных дескрипторов (inode, index **node**), проиллюстрированных в листинге 3.3.

Листинг 3.3. Имена и номера индексных дескрипторов файлов

```
finn@ubuntu:~$ ls -ai
20332580 .             20318930 .bash_logout  20320866 examples.desktop
20316161 ..            20320868 .bashrc       20320867 .profile
20320797 .bash_history 20332712 .cache
```

Каждый индексный дескриптор содержит *метаданные* (см. ❶❷❹ на рис. 3.2) — список стандартных свойств файла, в том числе указывающих местоположение *данных* файла (набора блоков) на файловой системе. Полный набор метаданных (листинг 3.4) позволяет получить команда **stat(1)**, включая размер файла ❶, количество занимаемых блоков на диске ❷, тип файла ❸, номер индексного дескриптора ❹, права доступа ❺, владельцев ❻ и пр.

Листинг 3.4. Метаданные файла

```
finn@ubuntu:~$ stat .profile
  Файл: ".profile"
  Размер: 675 ❶       Блоков: 8 ❷        Блок В/В: 4096   ❸ обычный файл
Устройство: fc00h/64512d       Inode: ❹ 20320867    Ссылки: 1
❺ Доступ: (0644/-rw-r--r--) ❻ Uid: ( 1001/    finn)   Gid: ( 1001/    finn)
Доступ: 2015-04-01 00:48:07.220999337 +0300
Модифицирован: 2011-05-18 13:54:10.000000000 +0400
Изменён: 2015-03-11 23:36:12.060159729 +0300
 Создан: -
```

Для создания каталогов предназначена команда **mkdir(1)**, а для удаления **rmdir(1)**, при этом удалению подлежат только пустые каталоги (см. листинг 3.8).

3.2.3. Имена, данные, метаданные и индексные дескрипторы

Каждый раз, когда используется путевое (абсолютное или относительное) имя файла, производится итеративный поиск файла в дереве путем последовательного разбора на имя → метаданные → данные первого каталога пути, содержащего в свою очередь имена → метаданные → данные второго и т. д., пока в конце поиска не будут найдены имя → метаданные → данные указанного файла. Такая ссылочность позволяет сформировать удобную древовидную структуру для каталогизации файлов (называемую деревом каталогов), однако сам поиск является относительно длительной операцией.

3.2.4. Ссылки

Каталог как файл-список имен других файлов, которым сопоставлены номера индексных дескрипторов, не запрещает иметь два разных имени файла, указывающих на одни и теже *метаданные* (см. ❸ на рис. 3.2). Такой эффект носит название *жесткой ссылки*, создать которую можно при помощи команды **ln(1)** (листинг 3.5).

Листинг 3.5. Жесткая ссылка

```
finn@ubuntu:~$ touch readme
finn@ubuntu:~$ ls -li readme
20318653 -rw-r--r-- 1 finn finn 0 апр.   1 01:22 readme
finn@ubuntu:~$ ln readme readme.txt
finn@ubuntu:~$ touch README
finn@ubuntu:~$ ls -li readme readme.txt README
20318653 -rw-r--r-- 2 finn finn 0 апр.   1 01:22 readme
20319121 -rw-r--r-- 1 finn finn 0 апр.   1 01:23 README
20318653 -rw-r--r-- 2 finn finn 0 апр.   1 01:22 readme.txt
```

Более того, оба имени являются равнозначными, и нет возможности узнать, какое из них создано первым, из чего нужно заключить, что первое и единственное имя файла уже является его жесткой ссылкой (на номер индексного дескриптора). При добавлении файлу нового имени (жесткой ссылки) в его метаданных увеличивается счетчик количества имен (см. ❶ в листинге 3.6), а при удалении файла сначала удаляется имя и уменьшается счетчик количества имен ❷, и только при удалении последнего имени высвобождаются метаданные и данные файла.

Листинг 3.6. Счетчик имен файла

```
finn@ubuntu:~$ ln readme read.me
finn@ubuntu:~$ ls -li read*
20318653 -rw-r--r-- ❶ 3 finn finn 0 апр.   1 01:22 readme
20318653 -rw-r--r--   3 finn finn 0 апр.   1 01:22 read.me
20318653 -rw-r--r--   3 finn finn 0 апр.   1 01:22 readme.txt
finn@ubuntu:~$ rm readme
finn@ubuntu:~$ ls -li read*
20318653 -rw-r--r-- ❷ 2 finn finn 0 апр.   1 01:22 read.me
20318653 -rw-r--r--   2 finn finn 0 апр.   1 01:22 readme.txt
finn@ubuntu:~$ rm readme.txt
finn@ubuntu:~$ ls -li read*
20318653 -rw-r--r-- ❷ 1 finn finn 0 апр.   1 01:22 read.me
```

Нужно заметить, что удаление файла — двухшаговая операция, состоящая из удаления имени файла, а затем — удаления метаданных (и высвобождения блоков, занимавшихся этим файлом). Удаление метаданных файла не выполняется *вообще*, если у файла еще остались имена (жесткие ссылки), и не происходит *сразу*, если файл открыт *(см. разд. 3.3)* каким-либо процессом. Метаданные и блоки, занимаемые файлом, высвобождаются только при закрытии этого файла всеми открывшими его процессами, что проиллюстрировано в примере из листинга 3.7.

Команда **df(1)** измеряет доступное (свободное, **d**isk **f**ree) место на файловой системе указанного файла, тогда как команда **du(1)**, наоборот, измеряет занимаемое (**d**isk **u**sage) указанным файлом место на его файловой системе.

Листинг 3.7. Удаление открытого файла

```
  finn@ubuntu:~$ df -h .
  Файл.система            Размер Использовано  Дост Использовано% Смонтировано в
  /dev/mapper/ubuntu-root   455G          400G   32G          93% /
  finn@ubuntu:~$ du -sh astra-linux-1.3-special-edition-smolensk-disk3-devel.iso
☛ 2,8G astra-linux-1.3-special-edition-smolensk-disk3-devel.iso
\ finn@ubuntu:~$ rm astra-linux-1.3-special-edition-smolensk-disk3-devel.iso
  finn@ubuntu:~$ df -h .
  Файл.система            Размер Использовано  Дост Использовано% Смонтировано в
? /dev/mapper/ubuntu-root   455G          400G   32G          93% /
  finn@ubuntu:~$ lsof astra-linux-1.3-special-edition-smolensk-disk3-devel.iso
  COMMAND   PID   USER   FD   TYPE DEVICE   SIZE/OFF     NODE NAME
  fuseiso 16925   finn    3r   REG  252,0 2947385344 20316584 astra-linux-1.3-special-edition-
  smolensk-disk3-devel.iso
\ finn@ubuntu:~$ kill 16925
  finn@ubuntu:~$ df -h .
  Файл.система            Размер Использовано  Дост Использовано% Смонтировано в
! /dev/mapper/ubuntu-root   455G          397G   35G          92% /
```

Специальные имена текущего `.` и родительского `..` каталога на поверку тоже оказываются жесткими ссылками, поэтому у любого каталога по крайней мере два имени — свое «собственное» в родительском и специальное `.` в самом себе, а у каталогов с подкаталогами еще и имена `..` в каждом из дочерних (листинг 3.8).

Листинг 3.8. Имена каталогов

```
finn@ubuntu:~$ mkdir folder
finn@ubuntu:~$ ls -ldi folder
```

```
☛ 20357139 drwxr-xr-x    2 finn finn 4096 апр.   1 01:59 folder ☚
  finn@ubuntu:~$ cd folder
  finn@ubuntu:~/folder$ ls -lai
  итого 8
☛ 20357139 drwxr-xr-x ☛ 2 finn finn 4096 апр.   1 01:59 . ☚
  20332580 drwxr-xr-x    4 finn finn 4096 апр.   1 01:59 ..
  finn@ubuntu:~/folder$ mkdir child
  finn@ubuntu:~/folder$ cd child
  finn@ubuntu:~/folder/child$ ls -lai
  итого 8
  20357140 drwxr-xr-x    2 finn finn 4096 апр.   1 02:02 .
☛ 20357139 drwxr-xr-x ☛ 3 finn finn 4096 апр.   1 02:02 .. ☚
  finn@ubuntu:~/folder/child$ cd ../..
  finn@ubuntu:~$ stat folder/
    Файл: «folder/»
    Размер: 4096          Блоков: 8          Блок В/В: 4096   каталог
  Устройство: fc00h/64512d        Inode: 20357139    Ссылки: ☛ 3
  ...           ...               ...
  finn@ubuntu:~$ rmdir folder
  rmdir: не удалось удалить «folder»: Каталог не пуст
```

Существенным ограничением жесткой ссылки в дереве каталогов, куда смонтирована более чем одна файловая система, является локальность жесткой ссылки в пределах своей файловой системы в силу локальной значимости номеров индексных дескрипторов. Так как на каждой новой файловой системе номера индексных дескрипторов начинают нумероваться с нуля, то жесткая ссылка всегда указывает на метаданные файла в «своей» файловой системе и не может указывать на метаданные файла в «чужой» файловой системе общего дерева каталогов. Для преодоления этого ограничения служит *символическая ссылка* **symlink(7)**, являющаяся самостоятельным служебным типом (см. ❹ на рис. 3.2) и содержащая путевое имя (см. ③ на рис. 3.2 и ★ в листинге 3.9) к целевому файлу.

Листинг 3.9. Символическая ссылка

```
finn@ubuntu:~$ ln -s read.me readme.1st
finn@ubuntu:~$ ls -li read*
20318653 -rw-r--r-- 1 finn finn 0 апр.   1 01:22 read.me
20319944 lrwxrwxrwx 1 finn finn 7 апр.   2 00:03 readme.1st -> read.me ★
```

В случае с символической ссылкой при удалении целевого файла сама ссылка будет указывать в никуда и называться «сиротой» (orhpan). Попытка прочитать (лис-

тинг 3.10) такую ссылку приводит к странным, на первый взгляд, результатам: файл «существует» ❶ для команды **ls(1)**, но команда просмотра содержимого **cat(1)** говорит об обратном ❷. Ничего удивительного, если помнить, что **ls(1)** работает с именами файлов, а **cat(1)** — с их данными (которые действительно не существуют).

Листинг 3.10. Сиротская ссылка

```
finn@ubuntu:~$ rm read.me
finn@ubuntu:~$ ls read*
❶ readme.1st
finn@ubuntu:~$ cat readme.1st
❷ cat: readme.1st: Нет такого файла или каталога
finn@ubuntu:~$ ls -l read*
lrwxrwxrwx 1 finn finn 7 апр.   2 00:03 readme.1st -> read.me
finn@ubuntu:~$ cat read.me
cat: read.me: Нет такого файла или каталога
```

Символические ссылки могут ссылаться на *имена* друг друга неограниченное количество раз, а при попытке использовать одну из таких ссылок будут использованы *данные* файла, последнего в цепочке ссылок. По ошибке можно даже закольцевать (листинг 3.11) две или более ссылок, с чем разберется операционная система при чтении одной из ссылок.

Листинг 3.11. Кольцевые ссылки

```
finn@ubuntu:~$ ln -s readme.1st read.me
finn@ubuntu:~$ ls -l read*
lrwxrwxrwx 1 finn finn 10 апр.   2 00:41 read.me -> readme.1st
lrwxrwxrwx 1 finn finn  7 апр.   2 00:03 readme.1st -> read.me
finn@ubuntu:~$ cat read.me
cat: read.me: Слишком много уровней символьных ссылок
```

Основным назначением символических (и изначально, жестких) ссылок является «множественная каталогизация» файлов, т. е. разные наборы разных имен одних и тех же данных. Типичным примером использования ссылок является организация **boot(7)** сценариев запуска системных служб при старте операционной системы. Сами сценарии располагаются в каталоге **/etc/init.d**, а в каталогах **/etc/rcS.d**, **/etc/rc0.d**, ..., **/etc/rc6.d** размещены символические ссылки на эти сценарии, которые должны быть запущены с определенными параметрами при переключении состояния системы между так называемыми «уровнями исполнения» **runlevel(7)** (листинг 3.12).

Листинг 3.12. Сценарии запуска служб и их каталогизация по уровням исполнения

```
finn@ubuntu:~$ ls -l /etc/rc?[1].d
               ...                ...                ...
/etc/rc2.d:
               ...                ...                ...
lrwxrwxrwx 1 root root 20 марта 12  2015 /etc/rc2.d/S19postgresql -> ../init.d/postgresql
lrwxrwxrwx 1 root root 17 янв.  31  2015 /etc/rc2.d/S20postfix -> ../init.d/postfix
               ...                ...                ...
/etc/rc6.d:
               ...                ...                ...
lrwxrwxrwx 1 root root 17 янв.  31  2015 /etc/rc6.d/K20postfix -> ../init.d/postfix
lrwxrwxrwx 1 root root 20 марта 12  2015 /etc/rc6.d/K21postgresql -> ../init.d/postgresql
               ...                ...                ...
```

В этом примере сценарии **postfix** и **postgresql** имеют вторичные имена, начинающиеся с **K** в каталоге **rc6.d**, и другие вторичные имена, начинающиеся с **S** в каталоге **rc2.d**. Это символизирует необходимость запускать (start) службы **postfix** и **postgresql** при переключении системы на уровень исполнения № 2 и уничтожать (**k**ill) процессы этих служб при переключении системы на уровень исполнения № 6.

3.2.5. Специальные файлы устройств

Специальные файлы устройств предназначены для ввода данных с аппаратных устройств и вывода данных на них. Настоящую работу по вводу и выводу данных проделывает драйвер устройства, а специальные файлы (листинг 3.13) играют роль своеобразных «порталов» связи с драйверами. Различают *символьные* ❶ и *блочные* ❷ специальные файлы устройств, у которых минимальной единицей обмена информацией с драйверами является блок (обычно размером в 512 байт) или символ (1 байт), соответственно.

Листинг 3.13. Специальные файлы устройств

```
finn@ubuntu:~$ ls -l /dev/sd* /dev/input/mouse* /dev/video* /dev/snd/pcm*
❶ crw-r-----  1 root root   13 ①, 32 ② марта 24 23:45 /dev/input/mouse0
  brw-rw----  1 root disk    8,    0    марта 27 15:01 /dev/sda
  brw-rw----  1 root disk    8,    1    марта 24 23:46 /dev/sda1
  brw-rw----  1 root disk    8,    2    марта 24 23:45 /dev/sda2
```

[1] Подробнее о шаблонных символах **?**, ***** и пр. *см. разд. 5.3.*

```
❷ brw-rw----  1 root disk    8,    5     марта 24 23:45 /dev/sda5
  crw-rw---T+ 1 root audio 116,    4     марта 27 19:02 /dev/snd/pcmC0D0c
  crw-rw---T+ 1 root audio 116,    3     марта 27 13:48 /dev/snd/pcmC0D0p
  crw-rw---T+ 1 root audio 116,    2     марта 24 23:47 /dev/snd/pcmC0D3p
  crw-rw----+ 1 root video  81,    0     марта 24 23:45 /dev/video0
```

Все драйверы ядра пронумерованы главными (мажорными, **major**) числами ①, а аппаратные устройства, находящиеся под их управлением, — дополнительными (минорными, **minor**) числами ②.

Например, все IDE-диски работают под управлением драйвера **hd(4)**, имеющего $\mathbf{3}^{major}$ (первичный контроллер) и $\mathbf{22}^{major}$ (вторичный контроллер), и нумеруются как $\mathbf{0}_{minor}$ (мастер-диск) и $\mathbf{64}_{minor}$ (слэйв-диск). Аналогично, SCSI-диски работают под управлением драйвера **sd(4)**, имеющего $\mathbf{8}^{major}$, и нумеруются $\mathbf{0}_{minor}$ (первый диск), $\mathbf{16}_{minor}$ (второй диск) и т. д.

Основной характеристикой специальных файлов устройств является пара чисел **major**, **minor** (иногда называемых характеристическими числами), привязывающая их к конкретному драйверу и управляемому им устройству. Имена специальных файлов и их местоположение в дереве каталогов не имеют никакого значения, но по соглашению **MAKEDEV(8)** их принято располагать в каталоге **/dev** и именовать созвучно именам драйверов.

Специальными файлами дисковых устройств пользуются программы, управляющие структурами самого носителя, например таблицами разделов **fdisk(8)** и **parted(8)** (листинг 3.14), или механикой накопителя **eject(1)** (листинг 3.15), или файловыми системами разделов носителя **mount(8)**, **fsck(8)**, **mkfs(8)** и пр.

Листинг 3.14. Чтение таблицы разделов диска

```
finn@ubuntu:~$ sudo fdisk -l  /dev/sda
Диск /dev/sda: 500.1 Гб, 500107862016 байт
255 головок, 63 секторов/треков, 60801 цилиндров, всего 976773168 секторов
Units = секторы of 1 * 512 = 512 bytes
Размер сектора (логического/физического): 512 байт / 4096 байт
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
Идентификатор диска: 0x000c8d62

Устр-во Загр     Начало       Конец       Блоки   Id  Система
/dev/sda1   *       2048      499711      248832   83  Linux
/dev/sda2          501758   976771071   488134657    5  Расширенный
Partition 2 does not start on physical sector boundary.
/dev/sda5          501760   976771071   488134656   8e  Linux LVM
```

Листинг 3.15. Открытие лотка DVD

```
finn@ubuntu:~$ ls -l /dev/dvd
lrwxrwxrwx 1 root root 3 марта 27 14:54 /dev/dvd -> sr0
finn@ubuntu:~$ eject /dev/dvd
```

Особенное место занимают драйверы терминалов[1] *(см. главу 2)* — оконечных устройств для взаимодействия с пользователями. Аппаратные терминалы (например, VT100), подключающиеся посредством последовательного интерфейса RS232, доступны при помощи специальных файлов **/dev/ttyS*N*** драйвера приемопередатчика последовательного порта **ttyS(4)**. Виртуальные терминалы, реализующиеся консолью (стандартной клавиатурой и дисплеем в алфавитно-цифровом режиме — см. рис. 2.3), доступны при помощи специальных файлов **/dev/tty*N*** (листинг 3.16) драйверов консоли **console_ioctl(4)** и **tty_ioctl(4)**.

Листинг 3.16. Специальные файлы терминалов

```
finn@ubuntu:~$ ls -la /dev/tty?
crw--w---- 1 root      tty 4, 0 марта 24 23:45 /dev/tty0
crw------- 1 finn      tty 4, 1 марта 27 10:46 /dev/tty1
crw------- 1 jake      tty 4, 2 марта 27 10:46 /dev/tty2
crw------- 1 marceline tty 4, 3 марта 27 10:47 /dev/tty3
crw------- 1 iceking   tty 4, 4 марта 27 10:46 /dev/tty4
crw------- 1 bubblegum tty 4, 5 марта 27 10:46 /dev/tty5
crw-rw---- 1 root      tty 4, 5 марта 27 10:46 /dev/tty6
```

Именно эти специальные файлы терминалов используют команды **write(1)** и **wall(1)** для отсылки сообщений пользователям, зарегистрировавшимся в системе, **setfont(1)** для смены шрифтов, **chvt(1)** для переключения между терминалами, а **setleds(1)** для управления светодиодами клавиатуры (листинг 3.17).

Листинг 3.17. Включение светодиодов CAPS LOCK и SCROLL LOCK

```
finn@ubuntu:~$ tty
/dev/tty1
finn@ubuntu:~$ setleds -L +num +scroll
```

Кроме специальных файлов настоящих аппаратных устройств, в арсенале Linux имеются специальные файлы псевдоустройств, такие как **/dev/null**, **/dev/full** и

[1] См. **W**:[TTY абстракция].

/dev/zero, симулирующих всегда пустое, всегда полное и бесконечно нулевое устройство, см. **null(4)**.

Всегда пустое устройстово **/dev/null** широко применяется на практике в качестве «подавляющего» стока информации при перенаправлениях[1], а бесконечно нулевое устройство **/dev/zero** зачастую используют в качестве источника для получения нулевых файлов нужной величины.

Псевдоустройства **/dev/random** и **/dev/urandom** организуют доступ к ядерному генератору случайных и псевдослучайных чисел **random(4)**, основанных на внешних событиях периферийных устройств. Потоки случайных чисел используются в качестве источников энтропии для криптоалгоритмов, в частности используются библиотекой **W:[OpenSSL]**, или служат команде **shred(1)** источником случайных байтов для надежного стирания данных.

3.2.6. Именованные каналы и файловые сокеты

Именованные каналы и файловые сокеты являются простейшими средствами межпроцессного взаимодействия (IPC, **I**nter**P**rocess **C**ommunication) и служат программам для обмена информацией между собой. Разные программы выполняются в рамках различных процессов[2] (изолированных друг от друга), поэтому для общения нуждаются в специальных средствах взаимодействия. Таким средством могли бы стать обычные файлы, но их основное назначение состоит в *сохранении* информации на каком-либо накопителе, что будет при *обмене* информацией сопряжено с накладными расходами, например задержками записи/чтения дискового (т. е. механического) носителя. Предоставить процессам возможность использовать файловые операции для эффективного *взаимодействия* между собой призваны *именованные каналы* (named pipe) **pipe(7)**, они же *FIFO-файлы* (**f**irst **i**n **f**irst **o**ut) **fifo(7)** и *файловые сокеты* (socket) **unix(7)**. Каналы и сокеты используют для передачи данных от процесса к процессу оперативную память ядра операционной системы, а не память накопителя, как обычные файлы.

Основное отличие (подробнее *см. разд. 4.9*) именованного канала от сокета состоит в способе передачи данных. Через именованный канал организуется однонаправленная (симплексная) передача без мультиплексирования, а через сокет — двунаправленная (дуплексная) мультиплексированная передача.

Именованный канал обычно используют при взаимодействии процессов по схеме «поставщик — потребитель» (producer-consumer), когда один потребитель принимает информацию от одного поставщика (на самом деле от разных, но в различ-

---

[1] О перенаправлениях *см. разд. 5.3*.

[2] О процессах *см. главу 4*.

ные моменты времени). Например (в некоторых реализациях), программы **shutdown(8)**, **reboot(8)**, **poweroff(8)** и **telinit(8)** передают посредством именованного канала **/dev/initctl** команды перезагрузки, выключения питания и пр. диспетчеру **init(8)**, который выполняет соответствующие действия.

Сокет используют при взаимодействии по схеме «клиент — сервер» (client-server), т. е. один сервер принимает и отправляет информацию от многих и ко многим (одновременно) клиентам. Например, в целях централизованного сбора событийной информации разнообразные службы операционной системы (в частности служба периодического выполнения заданий **cron(8)**, служба печати **cupsd(8)** и даже команда **logger(1)**) передают посредством файлового сокета **/dev/log** сообщения о происходящих событиях службе **rsyslogd(8)** для централизованной журнализации.

3.3. Файловые дескрипторы

Основными операциями, предоставляемыми ядром операционной системы программам (а точнее — процессам[1]) для работы с файлами, являются системные вызовы **open(2)** **read(2)**, **write(2)** и **close(2)**. В соответствии со своими именами, эти системные вызовы предназначены для открытия и закрытия файла, для чтения из файла и записи в файл. Дополнительный системный вызов **ioctl(2)** (**i**nput **o**utput **c**ontro**l**) используется для управления драйверами устройств и, как следствие, применяется в основном для специальных файлов устройств.

При запросе процесса на открытие файла системным вызовом **open(2)** производится его однократный (относительно медленный, *см. разд. 3.2.2*) поиск имени файла в дереве каталогов и для запросившего процесса создается так называемый *файловый дескриптор* (описатель, от англ. *descriptor*). Файловый дескриптор «содержит» информацию, описывающую файл, например индексный дескриптор **inode** файла на файловой системе, номера **major** и **minor** устройства, на котором располагается файловая система файла, режим открытия файла, и прочую служебную информацию. При последующих операциях **read(2)** и **write(2)** доступ к самим данным файла происходит с использованием файлового дескриптора (что исключает медленный поиск файла в дереве каталогов).

Файловые дескрипторы пронумерованы и содержатся в таблице открытых процессом файлов, которую можно получить (листинг 3.18) при помощи диагностической программы **lsof(1)**. В обратную сторону получить список процессов, открывших тот или иной файл, можно при помощи программ **lsof(1)** и **fuser(1)**, что бывает полезно для идентификации программ, «занявших» файловую систему, подлежащую отмонтированию (см. ★ в листинге 3.21).

---

[1] Подробнее о процессах *см. главу 4*.

Листинг 3.18. Таблица файловых дескрипторов

```
① finn@ubuntu:~$ lsof -p $$[1]
  COMMAND   PID USER   FD ↵ TYPE DEVICE SIZE/OFF     NODE NAME
      ...               ...              ...
  bash    17975 finn   1u ❶ CHR  136,2      0t0        5 /dev/pts/2
      ...               ...              ...

② root@ubuntu:~# lsof /dev/log
  COMMAND  PID   USER   FD    TYPE     DEVICE SIZE/OFF NODE NAME
  rsyslogd 543 syslog   0u ❷ unix 0xefef5680      0t0 1338 /dev/log

  root@ubuntu:~# fuser /dev/log
  /dev/log:             543
  root@ubuntu:~# ps p 543
    PID TTY      STAT   TIME COMMAND
    543 ?        Sl     0:43 rsyslogd -c5

  root@ubuntu:~# lsof /var/log/syslog
  COMMAND  PID   USER   FD   TYPE DEVICE SIZE/OFF     NODE NAME
  rsyslogd 543 syslog   1w ❸ REG  252,0    29039 26214496 /var/log/syslog
```

В первом ① примере из листинга 3.18 показано получение списка файловых дескрипторов (столбец **FD**) процесса командного интерпретатора **bash** пользователя **finn**, на котором файловый дескриптор ❶ номер **1** описывает открытый на чтение и запись **u** специальный символьный **CHR** файл устройства **/dev/pts/2**. Во втором ② примере показано получение информации о процессе, открывшем файловый сокет **unix** с именем **/dev/log** (файловый дескриптор ❷ номер **0** на чтение и запись **u**) и обычный файл **REG** с именем **/var/log/sysog** (файловый дескриптор ❸ номер **1** на запись **w**).

Пронаблюдать за использованием системных вызовов файлового программного интерфейса в момент выполнения программам позволяет системный трассировщик **strace(1)** (листинг 3.19).

Листинг 3.19. Трассировка файлового программного интерфейса

```
finn@ubuntu:~$ date
Вт. сент. 15 19:05:41 MSK ↵ 2015
```

[1] О подстановках командного интерпретатора *см. разд. 5.4.2.*

```
finn@ubuntu:~$ strace -fe open,close,read,write,ioctl date
                ...                ...                ...
open("/etc/localtime", O_RDONLY|O_CLOEXEC) = 3
                ...                ...                ...
finn@ubuntu:~$ file /etc/localtime
/etc/localtime: timezone data, version 2, 13 gmt time flags, 13 std time flags, no leap
seconds, 77 transition times, 13 abbreviation char
finn@ubuntu:~$ ls -la /dev/dvd
lrwxrwxrwx 1 root root 3 сент. 15 19:08 /dev/dvd -> sr0
finn@ubuntu:~$ strace -fe open,close,read,write,ioctl eject
                ...                ...                ...
open("/dev/sr0", O_RDWR|O_NONBLOCK)     = 3
ioctl(3, CDROMEJECT, 0x804cb4e)         = 0
close(3)                                = 0

finn@ubuntu:~$ strace -fe open,read,write,close,ioctl setleds -L +num +scroll
                ...                ...                ...
ioctl(0, KDGKBLED, 0xbfe4f4ff)         = 0
ioctl(0, KDGETLED, 0xbfe4f4fe)         = 0
ioctl(0, KDSETLED, 0x3)                = 0
```

Предположив, что программа **date(1)** показывает правильное московское время, потому что узнаёт заданную временную зону MSK из некоего конфигурационного файла операционной системы, при трассировке ее работы можно установить его точное имя — **/etc/localtime**. Аналогично предположив, что программа **eject(1)** открывает лоток привода CD/DVD при помощи специального файла устройства, при трассировке можно узнать имя файла — **/dev/sr0**, номер файлового дескрпитора при работе с файлом — **3** и команду **CDROMEJECT** соответствующего устройству драйвера **ioctl_list(2)**. Трассировка команды **setleds(1)** показывает, что она вообще не открывает никаких файлов, но пользуется файловым дескриптором **0** так называемого стандартного потока ввода (прикрепленного к текущему терминалу[1]) и командами **KDGETLED** и **KDSETLED** драйвера консоли **console_ioctl(4)**.

3.4. Файловые системы

3.4.1. Файловые системы и процедура монтирования

Доступ к информации организуется при помощи файлов, упорядоченных в единое «воображаемое» дерево каталогов, тогда как «настоящими» *источниками* данных

[1] О стандартных потоках ввода-вывода *см. разд. 5.3.*

являются *файловые системы* — структуры, решающие задачи *хранения*[1] информации. Отображение *множества* файловых систем в *единое* дерево каталогов реализуется посредством процедуры *монтирования*. Таким образом, все, что наблюдается в дереве каталогов, в реальности размещается на файловых системах.

Состав дерева каталогов показывает команда **mount(8)**, равно как и присоединяет к нему — монтирует очередную файловую систему. Отсоединяет (отмонтирует) файловую систему от дерева каталогов команда **umount(8)**, но только при условии, что ни один файл на этой файловой системе не используется никакой программой (а правильнее — никаким процессом) операционной системы.

В примере на рис. 3.3 и в листинге 3.20 иллюстрируются: файловая система **W:[ext4]**, располагающаяся на дисковом накопителе, идентифицирующемся файлом устройства **/dev/sda2**, и смонтированная непосредственно в корень дерева каталогов ❶; файловая система **vfat** flash-накопителя ❹ на устройстве **/dev/sdb1**, смонтированная в **/media/flash**; файловая система **W:[ISO 9660]** CD-диска ❺ на устройстве /dev/sr0, смонтированная в **/media/cdrom**.

Кроме этого, в дерево каталогов смонтированы две псевдофайловые системы **proc** ❷ и **sysfs** ❸, считывающие из оперативной памяти ядра операционной системы информацию о процессах, обнаруженных устройствах, загруженных драйверах и предоставляющих «файловый» доступ к ней.

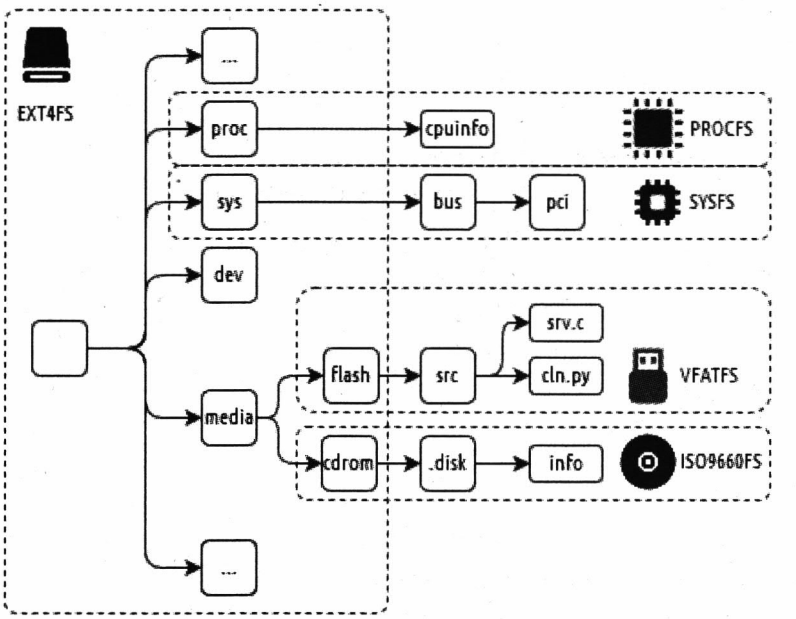


Рис. 3.3. Монтирование файловых систем

[1] Или извлечения информации откуда-либо (*см. разд. 3.4.4* и *3.4.5*).

Листинг 3.20. Состав дерева каталогов

```
finn@ubuntu:~$ mount
❶ /dev/sda2 on / type ext4 (rw,errors=remount-ro)
❷ proc on /proc type proc (rw,noexec,nosuid,nodev)
❸ sysfs on /sys type sysfs (rw,noexec,nosuid,nodev)
                ...             ...             ...
❹ /dev/sdb1 on /media/flash type vfat (rw,...)
❺ /dev/sr0 on /media/cdrom type iso9660 (ro,...)
```

Несмотря на то, что в современных дистрибутивах Linux обнаружение и процедуры монтирования файловых систем автоматизированы, операции монтирования/размонтирования могут быть произведены вручную (листинг 3.21).

Листинг 3.21. Процедуры монтирования/размонтирования файловых систем

```
finn@ubuntu:~$ mount /dev/dvd /media/cdrom
mount: только root может сделать это
finn@ubuntu:~$ sudo mount /dev/dvd /media/cdrom
mount: блочное устройство /dev/dvd защищен от записи, монтируется только для чтения
finn@ubuntu:~$ mount
                ...             ...             ...
/dev/dvd on /media/cdrom type iso9660 ☚ (ro)
                ...             ...             ...
finn@ubuntu:~$ cat /media/cdrom/.disk/info
Ubuntu 14.04.1 LTS "Trusty Tahr" - Release i386 (20140722.2)
finn@ubuntu:~$ umount /media/cdrom
umount: /media/cdrom не в fstab (а вы не root)
finn@ubuntu:~$ sudo umount /media/cdrom
finn@ubuntu:~$ cat /media/cdrom/.disk/info
cat: /media/cdrom/.disk/info: Нет такого файла или каталога
finn@ubuntu:~$ sudo umount /proc
umount: /proc: device is busy.
★         (In some cases useful info about processes that use
           the device is found by lsof(8) or fuser(1))
finn@ubuntu:~$ mount /dev/sdc1 /media/flash
finn@ubuntu:~$ mount
                ...             ...             ...
/dev/sdc1 on /media/flash type vfat ☚ (rw)
```

3.4.2. Дисковые файловые системы

Разные файловые системы **fs(5)**, как упоминалось ранее, предназначены для *хранения* информации на внешних носителях и преследуют различные цели, например, обеспечивают надежное хранение при помощи журнала транзакций или быстрый поиск метаданных файла (среди множества каталогов, подкаталогов и других файлов) по его имени, либо учитывают специфику свойств самого носителя и т. д. В большинстве случаев до сих пор носителями информации являются магнитные или оптические диски, благодаря чему файловые системы, размещаемые на них, зачастую называются «дисковыми» файловыми системами, даже если используются на твердотельных (flash) носителях.

Для магнитных дисков, характеризующихся возможностью чтения и записи блоков информации в произвольное место носителя (random access), в Linux на текущий момент времени используются «родные» файловые системы **W:[Ext2]**, **W:[Ext3]** и **W:[Ext4]**, специально разработанные **W:[ReiserFS]** и **W:[Reiser4]**, а также заимствованные **W:[XFS]** и **W:[JFS]**.

Для оптических CD/DVD-дисков, имеющих специфику записи в виде спиральной дорожки, применяются файловые системы **W:[ISO 9660]** и **W:[udf]**. Для USB-flash-накопителей в большинстве случаев используются заимствованные файловые системы **W:[FAT]** и **W:[NTFS]** в силу применения этих накопителей как мобильных средств переноса данных между разными компьютерами с различными операционными системами.

3.4.3. Сетевые файловые системы

Сетевые файловые системы, равно как и дисковые, обеспечивают *хранение* информации на внешнем носителе, которым в их случае выступает файловый сервер (например, домашний NAS, **N**etwork **A**ttached **S**torage), доступный по протоколу **NFS** (**N**etwork **F**ile **S**ystem, **W:[Network File System]**), **CIFS/SMB** (**C**ommon **I**nternet **F**ile **S**ystem или **S**erver **M**essage **B**lock, **W:[Server_Message_Block]**) или им подобным. Одноименные файловые системы **nfs** и **cifs/smb** используются для монтирования файлов сервера в дерево каталогов клиента. Таким образом, обычные (ничего не знающие ни про какие сетевые протоколы) программы, запускаемые в операционной системе клиента, используют файлы сетевого сервера точно так, как если бы они были размещены на локальных дисках, под управлением дисковых файловых систем.

В примере из листинга 3.22 программы **avconv(1)** и **avprobe(1)**, предназначенные для работы с «обычными» видеофайлами, используются для обработки записей сетевого видеорегистратора, видеофайлы которого доступны по протоколу **NFS**. Смонтированные при помощи сетевой файловой системы **nfs** в дерево каталогов файлы

сетевого регистратора становятся никак неотличимы от файлов *локальных* дисковых файловых систем.

Листинг 3.22. Сетевая файловая система NFS

```
finn@ubuntu:~$ mount -t nfs 182.168.1.10:/share/video /mnt/nas/video
finn@ubuntu:~$ mount
                ...                ...                ...
182.168.1.10:/share/video on /mnt/nas/video type nfs (rw)
                ...                ...                ...
finn@ubuntu:~$ cd /mnt/nas/video/screencasts
finn@ubuntu:~$ ls
                ...                ...                ...
20140523142626.mp4
                ...                ...                ...
finn@ubuntu:~$ file 20140523142626.mp4
20140523142626.mp4: ISO Media, MPEG v4 system, version 2
finn@ubuntu:~$ avprobe 20140523142626.mp4
                ...                ...                ...
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from '20140523142626.mp4':
                ...                ...                ...
  Duration: 00:00:07.94, start: 0.000000, bitrate: 11020 kb/s
    Stream #0.0(eng): Video: h264 (High), yuv420p, 1920x1080 [PAR 1:1 DAR 16:9], 10843 kb/s,
50 fps, 50 tbr, 50k tbn, 100 tbc
                ...                ...                ...
finn@ubuntu:~$ avconv -i 20140523142626.mp4 20140523142626.mkv
                ...                ...                ...
Output #0, matroska, to '20140523142626.mkv':
                ...                ...                ...
    Stream #0.0(eng): Video: mpeg4, yuv420p, 1920x1080 [PAR 1:1 DAR 16:9], q=2-31, 200 kb/s,
1k tbn, 50 tbc
                ...                ...                ...
Stream mapping:
  Stream #0:0 -> #0:0 (h264 -> mpeg4)
  Stream #0:1 -> #0:1 (aac -> libvorbis)
Press ctrl-c to stop encoding
frame=  395 fps= 72 q=31.0 Lsize=    2481kB time=7.94 bitrate=2561.5kbits/s dup=0 drop=1
video:2339kB audio:128kB global headers:4kB muxing overhead 0.457686%
```

Аналогично, в примере из листинга 3.23 геотеги файлов изображений сетевого видеорегистратора анализируются утилитой **exiv2(1)**, предназначенной для работы

с «обычными» изображениями. За счет файловой системы **cifs** и доступности видеорегистратора по протоколу **CIFS** его содержимое смонтировано в дерево каталогов так, словно *сетевой* регистратор является *локальным* дисковым накопителем.

Листинг 3.23. Сетевая файловая система CIFS/SMB

```
finn@ubuntu:~$ mount -t cifs -o username=guest //182.168.1.10/share/photos /mnt/nas/photos
Password:
finn@ubuntu:~$ mount
              ...              ...              ...
//182.168.1.10/share/photos on /mnt/nas/video type cifs (rw)
              ...              ...              ...
finn@ubuntu:~$ cd /mnt/nas/photos
finn@ubuntu:~$ ls
              ...              ...              ...
DSC_0034.JPG  DSC_0043.JPG  DSC_0062.JPG  DSC_0074.JPG  DSC_0100.JPG  DSC_0189.JPG
              ...              ...              ...
finn@ubuntu:~$ exiv2 -p a DSC_0043.JPG
              ...              ...              ...
Exif.GPSInfo.GPSLatitudeRef                 Ascii      2  North
Exif.GPSInfo.GPSLatitude                    Rational   3  60deg 10' 3.479"
Exif.GPSInfo.GPSLongitudeRef                Ascii      2  East
Exif.GPSInfo.GPSLongitude                   Rational   3  24deg 57' 23.294"
              ...              ...              ...
```

3.4.4. Специальные файловые системы

Развитие идеи файла, как единицы обеспечения доступа к информации, привело к тому, что абстракцию файловой системы перенесли и на другие сущности, доступ к которым стал организовываться в виде иерархии файлов. Например, информацию о процессах, нитях и прочих сущностях ядра операционной системы и используемых ими ресурсах предоставляет программам в виде файлов (!) *псевдофайловая* система **proc(5)**. Таким же образом, информацию об аппаратных устройствах, обнаруженных ядром операционной системы на шинах PCI, USB, SCSI и пр., предоставляет *псевдофайловая* система **sysfs**.

Различные утилиты, пользующиеся ядерной информацией, например показывающие нагрузку на операционную систему **uptime(1)** или списки процессов и загруженных модулей (драйверов) ядра операционной системы — **ps(1)** и **lsmod(8)**, пользуются псевдофайловой системой **proc**, в чем позволяет убедиться трассировка системных вызовов **open(2)** (листинг 3.24).

Листинг 3.24. Псевдофайловая система proc

```
finn@ubuntu:~$ strace -fe open uptime
              ...              ...              ...
open("/proc/version", O_RDONLY)         = 3
              ...              ...              ...
open("/proc/uptime", O_RDONLY)          = 3
              ...              ...              ...
open("/proc/loadavg", O_RDONLY)
              ...              ...              ...
15:42:32 up 12 days,  5:27,  7 users,  load average: 0.48 ☚, 0.33, 0.32
finn@ubuntu:~$ cat /proc/uptime
1056774.23 1667210.55
finn@ubuntu:~$ cat /proc/loadavg
0.48 ☚ 0.33 0.32 1/623 14277
```

Аналогично, утилиты, показывающие список устройств на шинах PCI, USB и SCSI — **lspci(8)**, **lsusb(8)** и **lsscsi(8)**, пользуются псевдофайловой системой **sysfs** (листинг 3.25).

Листинг 3.25. Псевдофайловая система sysfs

```
finn@ubuntu:~$ strace -fe open lspci -nn
              ...              ...              ...
open("/sys/bus/pci/devices/0000:00:02.0/resource", O_RDONLY) = 4
open("/sys/bus/pci/devices/0000:00:02.0/irq", O_RDONLY) = 4
open("/sys/bus/pci/devices/0000:00:02.0/vendor", O_RDONLY) = 4
open("/sys/bus/pci/devices/0000:00:02.0/device", O_RDONLY) = 4
open("/sys/bus/pci/devices/0000:00:02.0/class", O_RDONLY) = 4
              ...              ...              ...
00:02.0 VGA compatible controller [0300 ☚]: Intel Corporation 2nd Generation Core Processor
Family Integrated Graphics Controller [8086 ☚:0116 ☚] (rev 09)
finn@ubuntu:~$ cat /sys/bus/pci/devices/0000:00:02.0/vendor
0x8086 ☚
finn@ubuntu:~$ cat /sys/bus/pci/devices/0000:00:02.0/device
0x0116 ☚
finn@ubuntu:~$ cat /sys/bus/pci/devices/0000:00:02.0/class
0x030000 ☚
```

3.4.5. Внеядерные файловые системы

Сосуществование разных по своей сути файловых систем в едином дереве файлов и каталогов позволяет использовать для работы с его файлами «обычные» программы, ничего не знающие ни о местоположении, ни о свойствах самих источников информации этих файлов.

Таким же «файловым» образом можно организовать доступ к файлам внутри GZ/ZIP/...-архивов, файлам внутри ISO-образов CD/DVD-дисков, файлам зашифрованных каталогов, файлам других узлов сети, медиафайлам на медиаустройствах (плееры, смартфоны) и т. п. Файловая абстракция в определенных случаях оказывается удобной даже для осуществления доступа к информации, организованной совершенно «нефайловым» образом — к данным таблиц и представлений внутри реляционных баз данных, к элементам и атрибутам внутри XML-файлов и пр.

Для реализации такого «файлового» доступа к произвольным источникам информации используются так называемые *«внеядерные»* файловые системы **W**:[FUSE] (**F**ilesystem in **USE**rspace), реализуемые не ядерными модулями файловых систем (как **ext4**, **nfs**, **proc** и пр.), а обычными программами[1], запущенными в обычных процессах и работающими вне ядра.

В примере из листинга 3.26 показано, как можно без распаковки смонтировать сжатый архив исходных текстов ядра **linux-4.2.3.tar.xz** и прочитать отдельный файл **fuse.txt**.

Листинг 3.26. Внеядерная файловая система «fuse.archivemount»

```
finn@ubuntu:~$ wget https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.2.3.tar.xz
             ...              ...              ...
finn@ubuntu:~$ file linux-4.2.3.tar.xz
linux-4.2.3.tar.xz: XZ compressed data
finn@ubuntu:~$ archivemount linux-4.2.3.tar.xz ~/mnt/archive
                            ⓪ ① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩
finn@ubuntu:~$ mount
             ...              ...              ...
archivemount on /home/finn/mnt/archive type fuse.archivemount (rw,nosuid,...,user=finn)
finn@ubuntu:~$ cd ~/mnt/archive
finn@ubuntu:~/mnt/archive$ ls -l
итого 0
drwxrwxr-x 0 root root 0 окт.   3 14:52 linux-4.2.3
finn@ubuntu:~/mnt/archive$ cd linux-4.2.3/Documentation/filesystems
```

[1] Для чего в ядре все-таки требуется один универсальный модуль **fuse**.

```
finn@ubuntu:~/mnt/archive/linux-4.2.3/Documentation/filesystems$ less fuse.txt
finn@ubuntu:~/mnt/archive/linux-4.2.3/Documentation/filesystems$ cd ~
finn@ubuntu:~ $ fusermount -u ~/mnt/archive
```

В примере из листинга 3.27 показано, как при помощи сетевого протокола SSH *(см. разд. 6.4.1)* можно смонтировать часть дерева каталогов (домашний каталог пользователя **jake**) с удаленного узла **jake@grex.org** в каталог **~/mnt/net** локального дерева каталогов.

Листинг 3.27. Внеядерная файловая система «fuse.sshfs»

```
finn@ubuntu:~$ sshfs jake@grex.org: ~/mnt/net
jake@grex.org's password: P@s$w0rd
                          ⓿ ❶ ❷ ❸ ❹ ❺ ❻ ❼ ❽ ❾
finn@ubuntu:~$ mount
            ...           ...           ...
jake@grex.org: on /home/finn/mnt/net type fuse.sshfs (rw,nosuid,...,user=finn)
finn@ubuntu:~$ cd ~/mnt/net
finn@ubuntu:~/mnt/net$ ls -l
итого 488
-rw-rw-r-- 1 156620 131077 495604 окт.  10 20:28 OPENME.txz
-rw-r--r-- 1 156620 131077    106 окт.  10 20:30 README.gz
finn@ubuntu:~/mnt/net$ zless README.gz
finn@ubuntu:~/mnt/net$ cd ~
finn@ubuntu:~ $ fusermount -u ~/mnt/netw
```

В примере из листинга 3.28[1] показано, как можно смонтировать зашифровываемый каталог **/media/flash/secret** USB-flash-накопителя (уже смонтированного в каталог **/meida/flash** при помощи дисковой файловой системы **vfat**) в каталог **~/mnt/exposed** и скопировать туда файлы, подлежащие зашифровыванию. Теперь при утере накопителя можно не беспокоиться об информации, попавшей третьим лицам.

Листинг 3.28. Внеядерная файловая система «fuse.encfs»

```
finn@ubuntu:~ $ mount
            ...           ...           ...
/dev/sdc1 on /media/flash type vfat (rw)
            ...           ...           ...
finn@ubuntu:~ $ encfs /media/flash/secret ~/mnt/exposed
Директория "/media/flash/secret" не существует. Создать ее? (y,n) y
Директория "/home/finn/mnt/exposed" не существует. Создать ее? (y,n) y
```

[1] Во всех листингах даны оригинальные сообщения ОС.

```
Создание нового зашифрованого раздела.
Выберите одну из следующих букв:
 введите "x" для режима эксперта,
 введите "p" для режима максимальной секретности,
 любой другая буква для выбора стандартного режима.
?> ↵
Выбрана стандартная конфигурация.

Конфигурация завершена. Создана файловая система
с следующими свойствами:
Шифр файловой системы: "ssl/aes", версия 3:0:2
Шифр файла: "nameio/block", версия 3:0:1
Размер ключа: 192 бит
Размер блока: 1024 байт
Каждый файл содержит 8-ми байтный заголовок с уникальными IV данными.
Файловые имена зашифрованя с использованием IV цепочек.
File holes passed through to ciphertext.

Введите пароль для доступа к файловой системе.
Запомните пароль, так как в случае утери его,
будет невозможно востановить данные. Тем не менее
этот пароль можно изменить с помощью утилиты encfsctl.

Новый пароль EncFS: $ecr6t ↵
Повторите пароль EncFS: $ecr6t ↵
finn@ubuntu:~$ mount
encfs on /home/finn/mnt/exposed type fuse.encfs (rw,nosuid,nodev,..., user=finn)
finn@ubuntu:~$ cp ~/Изображения/IMG_20150801*.jpg ~/mnt/exposed
finn@ubuntu:~$ ls -l ~/mnt/exposed/
                ...                 ...                 ...
-rw-r--r-- 1 finn finn 1014408 окт.  10 19:06 IMG_20140801_123522.jpg
                ...                 ...                 ...
-rw-r--r-- 1 finn finn 1728838 окт.  10 19:06 IMG_20140801_124215.jpg
finn@ubuntu:~$ ls -l /media/flash/secret
                ...                 ...                 ...
-rw-r--r-- 1 finn finn 1014416 окт. 10 19:06 9WgX2m6hiNNz8,ii7UI1AIPetU3qGuLbRNyww9eHTiNNi-
                ...                 ...                 ...
-rw-r--r-- 1 finn finn 1728846 окт. 10 19:06 aavAA80-WGYg5O42x2A0I5VZacpZfK-4tEzP3SCX8bk5y-
finn@ubuntu:~$ file ~/mnt/exposed/IMG_20140801_123522.jpg
IMG_20140801_123522.jpg: JPEG image data, EXIF standard
```

```
finn@ubuntu:~$ file /media/flash/secret/9WgX2m6hiNNz8,ii7UI1AIPetU3qGuLbRNyww9eHTiNNi-
9WgX2m6hiNNz8,ii7UI1AIPetU3qGuLbRNyww9eHTiNNi-: data
finn@ubuntu:~$ fusermount -u /media/flash/secret
```

В примере из листинга 3.29 показано, как можно монтировать файловые системы **fuse** друг поверх друга в стек — смонтировать содержимое дерева каталогов FTP-сервера **mirror.yandex.ru**, далее смонтировать содержимое ISO-образа **FreeBSD-9.3-RELEASE-ia64-disc1.iso**, а затем смонтировать архив исходных текстов **src.txz**. При чтении файла страницы руководства **ls.1** файловые системы будут прозрачно и на лету (!) извлекать файл из архива, архив из образа и образ с сервера без предварительных скачиваний и распаковываний.

Листинг 3.29. Внеядерные файловые системы «fuse.curlftpfs», «fuse.fuseiso» и стекирование файловых систем

```
finn@ubuntu:~$ curlftpfs mirror.yandex.ru ~/mnt/net
                                    ⓵ ⓶ ⓷ ⓸ ⓹ ⓺ ⓻ ⓼ ⓽ ⓾
finn@ubuntu:~$ fuseiso ~/mnt/net/freebsd/releases/ISO-IMAGES/9.3/FreeBSD-9.3-RELEASE-ia64-disc1.iso ~/mnt/cd
                                    ⓵ ⓶ ⓷ ⓸ ⓹ ⓺ ⓻ ⓼ ⓽ ⓾
finn@ubuntu:~$ archivemount ~/mnt/cd/usr/freebsd-dist/src.txz ~/mnt/arcive
                                    ⓵ ⓶ ⓷ ⓸ ⓹ ⓺ ⓻ ⓼ ⓽ ⓾
finn@ubuntu:~$ mount
curlftpfs#ftp://mirror.yandex.ru/ on /home/finn/mnt/net type fuse (rw,...,user=finn)
fuseiso on /home/finn/mnt/cd type fuse.fuseiso (rw,nosuid,nodev,user=finn)
archivemount on /home/finn/mnt/archive type fuse.archivemount (rw,nosuid,nodev,user=finn)

finn@ubuntu:~$ man ~/mnt/archve/usr/src/bin/ls/ls.1
            ...              ...              ...
LS(1)                    BSD General Commands Manual

NAME
     ls — list directory contents
            ...              ...              ...

finn@ubuntu:~$ fusermount -u ~/mnt/archive
finn@ubuntu:~$ fusermount -u ~/mnt/cd
finn@ubuntu:~$ fusermount -u ~/mnt/net
```

3.5. Дискреционное разграничение доступа

В Linux, как и в любой многопользовательской системе, абсолютно естественным образом возникает задача разграничения доступа *субъектов* — пользователей к *объектам* — файлам дерева каталогов. Один из подходов к разграничению доступа — так называемый *дискреционный* (от англ. *discretion* — чье-либо усмотрение) — предполагает назначение *владельцев* объектов, которые по собственному усмотрению определяют *права доступа* субъектов (других пользователей) к объектам (файлам), которыми владеют.

Дискреционные механизмы разграничения доступа используются для разграничения прав доступа процессов *(см. разд. 4.5.1)* как обычных пользователей ❶, так и для ограничения прав системных программ ❷ (например, служб операционной системы), которые работают от лица псевдопользовательских учетных записей *(см. разд. 2.7)*.

В примере из листинга 3.30 при помощи команды **ps(1)** проиллюстрированы процессы операционной системы, выполняющиеся от лица разных учетных записей.

Листинг 3.30. Субъекты разграничения прав доступа: пользователи и псевдопользователи

```
finn@ubuntu:~$ ps axfu
☛ USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
  root          2  0.0  0.0      0     0 ?        S    Nov03   0:00 [kthreadd]
  root          3  0.0  0.0      0     0 ?        S    Nov03   0:03  \_ [ksoftirqd/0]
  ...           ...              ...
  root          1  0.0  0.0   3672  2016 ?        Ss   Nov03   0:01 /sbin/init
  syslog      999  0.0  0.0  31064  1528 ?        Sl   Nov03   0:02 rsyslogd -c5
  messageb   1064  0.0  0.0   4244  2024 ?        Ss   Nov03   0:06 dbus-daemon --system
  ...           ...              ...
  avahi      1154  0.0  0.0   3576  1748 ?        S    Nov03   0:00 avahi-daemon: running...
  ...           ...              ...
  daemon     1218  0.0  0.0   2472   348 ?        Ss   Nov03   0:00 atd
  nobody     8077  0.0  0.0   6820  1088 ?        S    09:35   0:01  \_ /usr/sbin/dnsmasq...
  ...           ...              ...
  whoopsie   1279  0.0  0.0  26244  4548 ?        Ssl  Nov03   0:01 whoopsie
  root       1847  0.0  0.0   5080  1384 ?        Ss   Nov03   0:00 /usr/lib/postfix/master
❷ postfix    1852  0.0  0.0   4936  1188 ?        S    Nov03   0:00  \_ qmgr -l -t fifo -u
  postfix   17763  0.0  0.0   4888  1204 ?        S    14:34   0:00  \_ pickup -l -t fifo -u -c
  ...           ...              ...
  root       1226  0.0  0.0  34040  7204 ?        Ssl  Nov03   0:00 lightdm
  root       1274  1.5  0.2  81988 17512 tty7     Ss+  Nov03  14:41  \_ /usr/bin/X :0 -auth ...
  root       1641  0.0  0.0  16644  3348 ?        Sl   Nov03   0:00  \_ lightdm --session-child 12 19
```

```
❶ finn      2000  0.0  0.1  52980  9448 ?        Ssl  Nov03   0:01      \_ gnome-session ...
  finn      2048  0.0  0.0   4080   208 ?        Ss   Nov03   0:00          \_ /usr/bin/ssh-agent ...
  ...          ...              ...
  finn      2132  1.2  1.0 392140 83672 ?        Sl   Nov03  11:40          \_ compiz
```

3.5.1. Владельцы и режим доступа к файлам

В рамках дискреционного разграничения доступа каждому файлу назначен *пользователь-владелец* ❶ и *группа-владелец* ❷ файла (листинг 3.31).

Листинг 3.31. Владельцы файлов

```
finn@ubuntu:~$ ls -la /etc/profile .profile
-rw-r--r-- 1 ❶ root ❷ root 665 марта  6  2013 /etc/profile
-rw-r--r-- 1 finn       finn 677 марта 22 12:33 .profile
finn@ubuntu:~$ stat .profile
     ...                    ...                    ...
Доступ: (0644/-rw-r--r--)  ❶ Uid: ( 1000/    finn)   ❷ Gid: ( 1000/    finn)
     ...                    ...                    ...
```

Назначаются владельцы файлов при их создании — обычно пользователем — владельцем файла становится пользователь, создавший файл, а группой — владельцем файла становится его первичная группа (листинг 3.32).

Листинг 3.32. Назначение владельцев файлов при создании

```
finn@ubuntu:~$ id
uid=1000(finn) gid=1000(finn) группы=1000(finn),4(adm),20(dialout),...,110(admin)
finn@ubuntu:~$ nano README
finn@ubuntu:~$ ls -la README
-rw-rw-r-- 1 finn finn 3 марта 26 23:26 README
```

Изменяются (листинг 3.33) пользователь — владелец ❶ файлов только суперпользователем ① **root** при помощи команды **chown(1)**, а группа-владелец — владельцем файла ❷ при помощи команды **chgrp(1)**, но только на ту ❸, к которой он сам принадлежит.

Листинг 3.33. Смена владельцев файлов

```
finn@ubuntu:~$ ls -la README
-rw-rw-r-- 1 finn finn 3 марта 26 23:26 README
finn@ubuntu:~$ chown jake README
```

```
❶ chown: изменение владельца «README»: Операция не позволяется
   finn@ubuntu:~$ id
   uid=1000(finn) gid=1000(finn) группы=1000(finn),4(adm),20(dialout),...,110(admin)
❷ finn@ubuntu:~$ chgrp admin README
   finn@ubuntu:~$ ls -la README
   -rw-rw-r-- 1 finn admin 3 марта 26 23:26 README
   finn@ubuntu:~$ chgrp daemon README
❸ chgrp: изменение группы для «README»: Операция не позволяется
① finn@ubuntu:~$ sudo chown jake README
   [sudo] password for user: <пароль finn'a>
   finn@ubuntu:~$ ls -la README
   -rw-rw-r-- 1 jake admin 3 марта 26 23:26 README
```

В примере из листинга 3.16 стоит обратить внимание на владельцев специальных файлов устройств виртуальных терминалов — ими назначены пользователи, совершившие вход в систему. Специальный файл устройства терминалов, на которых вход не осуществлен, принадлежит суперпользователю **root**, например как в случае с терминалом **/dev/tty9**. Такие назначения естественным образом делаются при входе пользователей в систему и для того, чтобы они могли распоряжаться правами устройства терминала, через который осуществили вход (см. листинг 3.41).

3.5.2. Базовые права доступа и дополнительные атрибуты

Для разграничения действий над файлами определены три базовых права доступа (базовые разрешения): чтение **r** — «**r**ead», запись **w** — «**w**rite» и выполнение **x** — «e**x**ecute», соответствующие разрешению выполнять системные вызовы **read(2)**[1], **write(2)** и **execve(2)**[2]. Каждое из базовых прав назначается на файл тому или иному пользователю или группе, разрешая соответствующую операцию.

В наследии классической UNIX определены только три субъекта (листинг 3.34), которым назначаются базовые права — пользователь-владелец (owner) ❶, группа-владелец (group owner) ❷ и все остальные (others) ❸. Совокупность их базовых прав называется *режимом доступа* (access mode) к файлу.

Базовое право может быть назначено **r**, **w** или **x** или отозвано **-**, поэтому в метаданных файла представляется одним битом, а для режима доступа требуется

---

[1] Точнее, системному вызову **open(2)** с флагами **O_RDONLY** и **O_WRONLY**, но для простоты можно считать **r** — **read(2)**, а **w** — **write(2)**.

[2] Подробнее о системном вызове **execve(2)** *см. разд. 4.3.*

девять бит: по три бита прав на каждый из трех субъектов доступа. Компактно режим доступа может быть записан соответствующим числом в восьмеричной системе счисления **rw-r--r--** → $\mathbf{110100100_2}$ → $\mathbf{644_8}$.

Листинг 3.34. Режим доступа к файлу

```
finn@ubuntu:~$ stat .profile
  Файл: «.profile»
              ❶↓    ↓❸              ...                    ...
Доступ: (0644/-rw- r-- r--)  Uid: ( 1000/    finn)   Gid: ( 1000/    finn)
                  ❷↑                ...                    ...
finn@ubuntu:~$ ls -l .profile
-rw-r--r-- 1 finn finn 677 марта 22 12:33 .profile
```

Проверка режима доступа (листинг 3.35) при операциях с файлами проверяется «слева направо» до первого совпадения. Если пользователь, осуществляющий операцию с файлом, является его владельцем, тогда используются *только* ❶ права владельца. В противном случае проверяется членство пользователя, осуществляющего операцию с файлом, в группе-владельцев файла, и тогда используются *только* ❷ права группы-владельцев. В других случаях используются права для всех остальных ❸, а для суперпользователя **root** вообще никакие проверки не осуществляются.

Листинг 3.35. Использование режима доступа к файлу

```
finn@ubuntu:~$ id finn
uid=1000(finn) gid=1000(finn) группы=1000(finn),4(adm),20(dialout),...,110(admin)
finn@ubuntu:~$ ls -l README.*
❶ -rw-r--r-- 1 finn        admin     2471 окт.  11 01:12 README.finn
  --w-r--r-- 1 finn        admin     2471 окт.  11 01:12 README.finn.locked
❷ -rw-r--r-- 1 jake        admin      776 окт.  11 01:12 README.jake
  -rw----r-- 1 bubblegum   admin      171 окт.  11 01:12 README.bubblegum
❸ -rw-r--r-- 1 marceline   marceline   16 окт.  11 01:12 README.marceline
  -rw-r----- 1 iceking     iceking     31 окт.  11 01:12 README.iceking
finn@ubuntu:~$ wc README.*
wc: README.admins: Отказано в доступе
wc: README.bubblegum: Отказано в доступе
  24  177 2471 README.finn
wc: README.finn.locked: Отказано в доступе
wc: README.iceking: Отказано в доступе
```

```
 12   70  776 README.jake
  1    1   16 README.marceline
 37  248 3263 итого
```

Режим доступа новых файлов

Назначается режим доступа файлов при их создании программой, создавшей файл, исходя из назначения файла, но с учетом пожеланий (точнее, нежеланий) пользователя. Так, например, текстовые редакторы назначают создаваемым (текстовым) файлам права **rw** для всех субъектов, а компиляторы назначают создаваемым (программным) файлам права **rwx** для всех субъектов. Пользователь может выразить свое нежелание назначать вновь создаваемым файлам те или иные права доступа для тех или иных субъектов, установив так называемую реверсивную (т. е. обратную, символизирующую НЕжелание) маску доступа при помощи встроенной команды интерпретатора **umask** (листинг 3.36).

Листинг 3.36. Реверсивная маска доступа

```
finn@ubuntu:~$ umask
0002
finn@ubuntu:~$ umask -S
u=rwx,g=rwx,o=rx
finn@ubuntu:~$ touch common.jnl
finn@ubuntu:~$ ls -l common.jnl
-rw-rw-r-- 1 finn finn 0 окт.  21 22:43 common.jnl
finn@ubuntu:~$ umask g-w,o-rwx
finn@ubuntu:~$ umask
0027
finn@ubuntu:~$ umask -S
u=rwx,g=rx,o=
finn@ubuntu:~$ touch group.jnl
finn@ubuntu:~$ ls -l group.jnl
-rw-r----- 1 finn finn 0 окт.  21 22:44 group.jnl
finn@ubuntu:~$ umask g=
finn@ubuntu:~$ umask
0077
finn@ubuntu:~$ umask -S
u=rwx,g=,o=
finn@ubuntu:~$ touch private.jnl
finn@ubuntu:~$ ls -l private.jnl
-rw------- 1 finn finn 0 окт.  21 22:44 private.jnl
```

Изменять режим доступа разрешено непосредственному пользователю — владельцу файла, но не членам группы-владельцев, что иллюстрирует листинг 3.37 при помощи команды chmod(1).

Листинг 3.37. Изменение режима доступа к файлу

```
finn@ubuntu:~$ id finn
uid=1000(finn) gid=1000(finn) группы=1000(finn),4(adm),20(dialout),...,110(admin)
finn@ubuntu:~$ ls -l README.*
-rw-r--r-- 1 finn        admin     2471 окт.  11 01:12 README.finn
-rw-r--r-- 1 jake        admin      776 окт.  11 01:12 README.jake
     ...                   ...                          ...
finn@ubuntu:~$ chmod g-r,o-r README.finn
finn@ubuntu:~$ ls -la README.finn
-rw-------- 1 finn        admin     2471 окт.  11 01:12 README.finn
finn@ubuntu:~$ chmod g-r,o-r README.jake
chmod: изменение прав доступа для «README.jake»: Операция не позволяется
```

Семантика режима доступа разных типов файлов

Права доступа `r`, `w`, `x` для обычных файлов представляются чем-то интуитивно понятным, но для других типов файлов это не совсем так. Например, каталог (см. рис. 3.2) содержит *список имен* файлов, поэтому право `w` для каталога — это право записи в этот список и право стирания из этого списка, что трансформируется в право *удаления* файлов из каталога и *создания* файлов в каталоге. Аналогично, право `r` для каталога — это право *просмотра списка имен* его файлов. И наконец, право `x` для каталога является правом *прохода* в каталог, т. е. позволяет обращаться к файлам внутри каталога по их имени (листинг 3.38).

Листинг 3.38. Права доступа к каталогу

```
finn@ubuntu:~$ mkdir folder
finn@ubuntu:~$ ls -lad folder/
drwxrwxr-x 2 finn finn 4096 окт.  12 00:37 folder/
finn@ubuntu:~$ cp /etc/magic folder
finn@ubuntu:~$ chmod u-w folder
finn@ubuntu:~$ ls -lad folder/
dr-xrwxr-x 2 finn finn 4096 окт.  12 00:40 folder/
finn@ubuntu:~$ cp /etc/localtime folder/
cp: невозможно создать обычный файл «folder/localtime»: Отказано в доступе
finn@ubuntu:~$ rm folder/magic
rm: невозможно удалить «folder/magic»: Отказано в доступе
```

```
finn@ubuntu:~$ ls -li folder/
итого 4
20318203 -rw-r--r-- 1 finn finn 111 окт.  12 00:40 magic
finn@ubuntu:~$ chmod u-r folder
finn@ubuntu:~$ ls -lad folder/
d--xrwxr-x 2 finn finn 4096 окт.  12 00:40 folder/
finn@ubuntu:~$ ls -li folder/
ls: невозможно открыть каталог folder/: Отказано в доступе
finn@ubuntu:~$ ls -li folder/magic
! 20318203 -rw-r--r-- 1 finn finn 111 окт.  12 00:40 folder/magic
finn@ubuntu:~$ chmod u-x folder/
finn@ubuntu:~$ ls -lad folder/
d---rwxr-x 2 finn finn 4096 окт.  12 00:40 folder/
finn@ubuntu:~$ ls -li folder/magic
ls: невозможно получить доступ к folder/magic: Отказано в доступе
finn@ubuntu:~$ chmod u+rw folder/
finn@ubuntu:~$ ls -lad folder/
drw-rwxr-x 2 finn finn 4096 окт.  12 00:40 folder/
finn@ubuntu:~$ cp /etc/localtime folder/
cp: не удалось выполнить stat для «folder/localtime»: Отказано в доступе
finn@ubuntu:~$ rm folder/magic
rm: невозможно удалить «folder/magic»: Отказано в доступе
finn@ubuntu:~$ ls -l folder/
ls: невозможно получить доступ к folder/magic: Отказано в доступе
итого 0
! -????????? ? ? ? ?            ? magic
finn@ubuntu:~$ chmod u=rwx folder/
finn@ubuntu:~$ ls -ld folder/
drwxrwxr-x 2 finn finn 4096 окт.  12 00:40 folder/
finn@ubuntu:~$ cd folder/
finn@ubuntu:~/folder$ ls -l
итого 4
-rw-r--r-- 1 finn finn  111 окт.  12 00:40 magic
finn@ubuntu:~/folder$ chmod a= magic
finn@ubuntu:~/folder$ ls -l magic
☛ ---------- 1 finn finn 111 окт.  12 00:40 magic
finn@ubuntu:~/folder$ rm magic
↰ rm: удалить защищенный от записи обычный файл «magic»? y
finn@ubuntu:~/folder$ ls -l
! итого 0
```

Для жестких ссылок права доступа не существуют вовсе — они просто являются теми же правами, что и права целевого файла, в силу того что права доступа хранятся в метаданных. Для символических ссылок семантика прав сохранена такой же, как и у жестких ссылок, с тем лишь различием, что права символических ссылок существуют отдельно от целевых файлов, но никогда не проверяются (см. **symlink(7)**). Для изменения прав доступа самих символических ссылок даже не существует специальной команды — при использовании **chmod(1)** со ссылкой всегда будут изменяться права целевого файла (листинг 3.39).

Листинг 3.39. Права доступа ссылок

```
finn@ubuntu:~$ ls -l README.finn
-rwxr--r-- 1 finn finn 2471 окт.  11 01:13 README.finn
finn@ubuntu:~$ ln README.finn read.me
finn@ubuntu:~$ ln -s README.finn readme.1st
finn@ubuntu:~$ ls -l README.finn read.me readme.1st
-rwxr--r-- 2 finn finn 2471 окт.  11 01:13 read.me
lrwxrwxrwx 1 finn finn   11 окт.  12 01:19 readme.1st -> README.finn
-rwxr--r-- 2 finn finn 2471 окт.  11 01:13 README.finn
finn@ubuntu:~$ chmod g+w read.me
finn@ubuntu:~$ ls -l README.finn read.me readme.1st
-rwxrw-r-- 2 finn finn 2471 окт.  11 01:13 read.me
lrwxrwxrwx 1 finn finn 11 окт.  12 01:19 readme.1st -> README.finn
-rwxrw-r-- 2 finn finn 2471 окт.  11 01:13 README.finn
finn@ubuntu:~$ chmod o-r readme.1st
finn@ubuntu:~$ ls -l README.finn read.me readme.1st
-rwxrw---- 2 finn  finn  2471 окт.  11 01:13 read.me
lrwxrwxrwx 1 finn  finn  11 окт.  12 01:19 readme.1st -> README.finn
-rwxrw---- 2 finn  finn  2471 окт.  11 01:13 README.finn
```

Для специальных файлов устройств, именованных каналов и сокетов право **x** не определено, а права **r** и **w** стоит воспринимать как права ввода и вывода информации на устройство и как права передачи и приема информации через средство взаимодействия.

Дополнительные атрибуты

Помимо базовых прав доступа **r**, **w** и **x**, для решения отдельных задач разграничения доступа используют дополнительные атрибуты **s**, Set user/group ID (SUID **S**et **U**ser **ID** или SGID, **S**et **G**roup **ID**) — атрибут неявного делегирования полномочий и **t**, s**T**icky — «липучка», атрибут ограниченного удаления.

Типичной задачей, требующей неявного делегирования полномочий, является проблема невозможности изменения пользователями свойств своих учетных записей, которые хранятся в двух файлах-таблицах — **passwd(5)** и **shadow(5)**, доступных на запись ❶ (и чтение ❷) только суперпользователю **root**. Однако (листинг 3.40) команды **passwd(1)**, **chsh(1)** и **chfn(1)**, будучи запущены обычным пользователем, прекрасно изменяют (!) пароль в таблице **/etc/shadow** и свойства пользовательской записи в таблице **/etc/passwd** за счет передачи полномочий ❸ *пользователя* — *владельца* программы тому пользователю, который ее запускает.

Листинг 3.40. Дополнительный атрибут SUID

```
finn@ubuntu:~$ ls -la /etc/passwd /etc/shadow
❶ -rw-r--r-- 1 root root   2338 сент. 11 11:51 /etc/passwd
❷ -rw-r----- 1 root shadow 1878 сент. 15 17:59 /etc/shadow
finn@ubuntu:~$ passwd
Смена пароля для finn.
(текущий) пароль UNIX:
Введите новый пароль UNIX:
Повторите ввод нового пароля UNIX:
passwd: пароль успешно обновлён
finn@ubuntu:~$ ls -la /etc/passwd /etc/shadow
-rw-r--r-- 1 root root   2338 сент. 11 11:51    /etc/passwd
-rw-r----- 1 root shadow 1878 окт.  19 23:51 ➥ /etc/shadow
finn@ubuntu:~$ chfn
Пароль:
Изменение информации о пользователе finn
Введите новое значение или нажмите ENTER для выбора значения по умолчанию
        Полное имя:
        Номер комнаты []:
        Рабочий телефон []: +7(812)703-02-02
        Домашний телефон []:
finn@ubuntu:~$ ls -l /etc/passwd /etc/shadow
-rw-r--r-- 1 root root   2357 окт.  19 23:53 ➥ /etc/passwd
-rw-r----- 1 root shadow 1878 окт.  19 23:51    /etc/shadow
finn@ubuntu:~$ ls -la /usr/bin/passwd /usr/bin/chfn
❸ -rwsr-xr-x 1 root root 40292 сент. 13  2012 /usr/bin/chfn
-rwsr-xr-x 1 root root 41284 сент. 13  2012 /usr/bin/passwd
```

За счет использования атрибута SUID получается, что пользователям, запускающим программы **chfn(1)**, **chsh(1)** и **passwd(1)**, для их исполнения временно делегируют-

ся права владельца этих программ (суперпользователя **root**) так, как будто сам суперпользователь их запустил.

Листинг 3.41. Дополнительный атрибут SGID

```
finn@ubuntu:~$ w
 00:03:53 up 12 days, 13:53,  7 users,  load average: 0,53, 0,51, 0,91
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
jake     tty2                      00:03    9.00s  0.52s  0.43s -bash
finn     tty1                      00:03   17.00s  0.51s  0.45s -bash
finn@ubuntu:~$ ls -l /dev/tty1 /dev/tty2
crw------- 1 finn tty 4, 1 окт.  20 00:03 /dev/tty1
crw------- 1 jake tty 4, 2 окт.  20 00:03 /dev/tty2
finn@ubuntu:~$ write jake
write: jake has messages disabled
...           ...              ...
                                          jake@ubuntu:~$ mesg y
...           ...              ...
finn@ubuntu:~$ ls -l /dev/tty1 /dev/tty2
finn@ubuntu:~$ ls -l /dev/tty2
crw--w---- 1 jake tty 4, 2 окт.  20 00:07 /dev/tty2
finn@ubuntu:~$ write jake
write: write: you have write permission turned off.

Hi, buddy, wazzzup?
^D
                                          jake@ubuntu:~$
                                          Message from finn@ubuntu on tty1 at 00:10 ...
                                          Hey buddy, wazzup?
                                          EOF
finn@ubuntu:~$ ls -Ll /usr/bin/write
-rwxr-sr-x 1 root tty 9728 марта 31  2012 /usr/bin/write
```

Аналогично (см. листинг 3.41) при использовании атрибута SGID, при передаче сообщений от пользователя к пользователю командой **write(1)** или **wall(1)**, запускающему эти программы пользователю, делегируются полномочия группы **tty**, имеющей доступ на запись к терминалам (специальным файлам устройств **/dev/tty*N***), владельцы которых разрешили такой доступ.

Именно за счет механизма SUID/SGID различные команды позволяют обычным, непривилегированным пользователям выполнять сугубо суперпользовательские дей-

ствия. Так, например, **su(1)** и **sudo(1)** позволяют выполнять команды одним пользователям от лица других пользователей, **mount(8)**, **umount(8)** и **fusermount(1)** — монтировать и размонтировать файловые системы, **ping(8)** и **traceroute(1)** — выполнять диагностику сетевого взаимодействия, **at(1)** и **crontab(1)** — сохранять в «системных» каталогах отложенные и периодические задания, и т. д.

Однако для каталогов атрибут SGID имеет совсем другой смысл. По умолчанию владельцем файла становится ❷ тот пользователь (и его первичная группа), который запустил программу, создавшую файл. Но для файлов, создаваемых в «общих» ❶ для какой-то группы пользователей каталогах, логичнее было бы назначать группой-владельцем создаваемых файлов эту общую группу ❸.

Листинг 3.42. Дополнительный атрибут SGID для каталога

```
bubblegum@ubuntu:~$ cd /srv/kingdom
bubblegum@ubuntu:/srv$ id
uid=1005(bubblegum) gid=1005(bubblegum) группы=1005(bubblegum),1007(candy)
bubblegum@ubuntu:/srv/kingdom$ ls -ld .
drwxr-xr-x 2 bubblegum bubblegum 4096 окт.  21 22:02 .
bubblegum@ubuntu:/srv/kingdom$ touch bananaguard1
bubblegum@ubuntu:/srv/kingdom$ ls -l
итого 0
-rw-rw-r-- 1 bubblegum bubblegum 0 окт.  21 22:02 bananaguard1
bubblegum@ubuntu:/srv/kingdom$ chgrp candy .
bubblegum@ubuntu:/srv/kingdom$ chmod g+ws .
bubblegum@ubuntu:/srv/kingdom$ ls -ld .
drwxrwsr-x 2 bubblegum candy 4096 окт.  21 22:02
...          ...             ...
finn@ubuntu:/srv/kingdom$ id
uid=1001(finn) gid=1001(finn) группы=1001(finn),1007(candy)
finn@ubuntu:/srv/kingdom$ touch bananaguard2
finn@ubuntu:/srv/kingdom$ ls -l
итого 0
-rw-rw-r-- 1 bubblegum bubblegum 0 окт.  21 22:02 bananaguard1
❸ -rw-rw-r-- 1 finn      candy     0 окт.  21 22:02 bananaguard2
```

В примере из листинга 3.42 за счет SGID-атрибута каталога, владельцем всех файлов, помещаемых в этот каталог, автоматически назначается группа-владелец самого каталога, а создатель (владелец) файла может теперь назначать нужные права доступа для всех членов *этой* группы к *своему* файлу — либо неявно при помощи реверсивной маски доступа, либо явно при помощи команды **chmod(1)**.

Атрибут-«липучка» **t** (sTicky) служит для ограничения действия базового разрешения **w** записи в каталоге. Например, временный каталог **/tmp** предназначается для хранения временных файлов любых пользователей и поэтому доступен на запись всем пользователям. Однако право записи в каталог дает возможность не только создавать в нем новые файлы, но и удалять *любые* существующие файлы (любых пользователей), что совсем не кажется логичным. Именно атрибут **t** ограничивает возможность удалять чужие файлы, т. е. файлы, не принадлежащие пользователю, пытающемуся их удалить.

Листинг 3.43. Дополнительный атрибут sTicky для каталога

```
finn@ubuntu:/srv/kingdom$ id
uid=1001(finn) gid=1001(finn) группы=1001(finn),1007(candy)
finn@ubuntu:/srv/kingdom$ ls -la
итого 8
drwxrwsr-x 2 bubblegum candy  4096 окт.  23 20:57 .
drwxr-xr-x 3 root      root   4096 окт.  21 21:57 ..
-rw-rw-r-- 1 bubblegum bubblegum 0 окт.  21 23:15 bananaguard1
-rw-rw-r-- 1 finn      candy     0 окт.  21 23:24 bananaguard2
...           ...              ...
finn@ubuntu:/srv/kingdom$ rm bananaguard1
rm: удалить защищенный от записи пустой обычный файл «bananaguard1»? y
finn@ubuntu:/srv/kingdom$ ls -l
итого 0
-rw-rw-r-- 1 finn candy 0 окт.  21 23:24 bananaguard2
...           ...              ...
bubblegum@ubuntu:/srv/kingdom$ chmod +t .
bubblegum@ubuntu:/srv/kingdom$ touch bananaguard1
bubblegum@ubuntu:/srv/kingdom$ ls -la
итого 8
drwxrwsr-t 2 bubblegum candy 4096 окт.  23 21:19 .
drwxr-xr-x 3 root      root  4096 окт.  21 21:57 ..
-rw-rw-r-- 1 bubblegum candy    0 окт.  23 21:19 bananaguard1
-rw-rw-r-- 1 finn      candy    0 окт.  23 21:19 bananaguard2
...           ...              ...
finn@ubuntu:/srv/kingdom$ rm bananaguard1
rm: невозможно удалить «bananaguard1»: Операция не позволяется
```

3.5.3. Списки контроля доступа POSIX

Режим доступа к файлу (access mode), определяющий базовые разрешения **r**, **w** и **x** только для трех субъектов доступа (владельца, группы-владельца и всех остальных), не является достаточно гибким и удобным инструментом разграничения доступа. *Списки* контроля доступа (**W**:[**ACL**], **a**ccess **c**ontrol **l**ists), согласно стандарту POSIX.1e, расширяют классический *режим* доступа к файлу дополнительными записями (рис. 3.4), определяющими права доступа для явно указанных пользователей и групп. Для просмотра и модификации записей в списках доступа используются утилиты **getfacl(1)** и **setfacl(1)** соответственно.

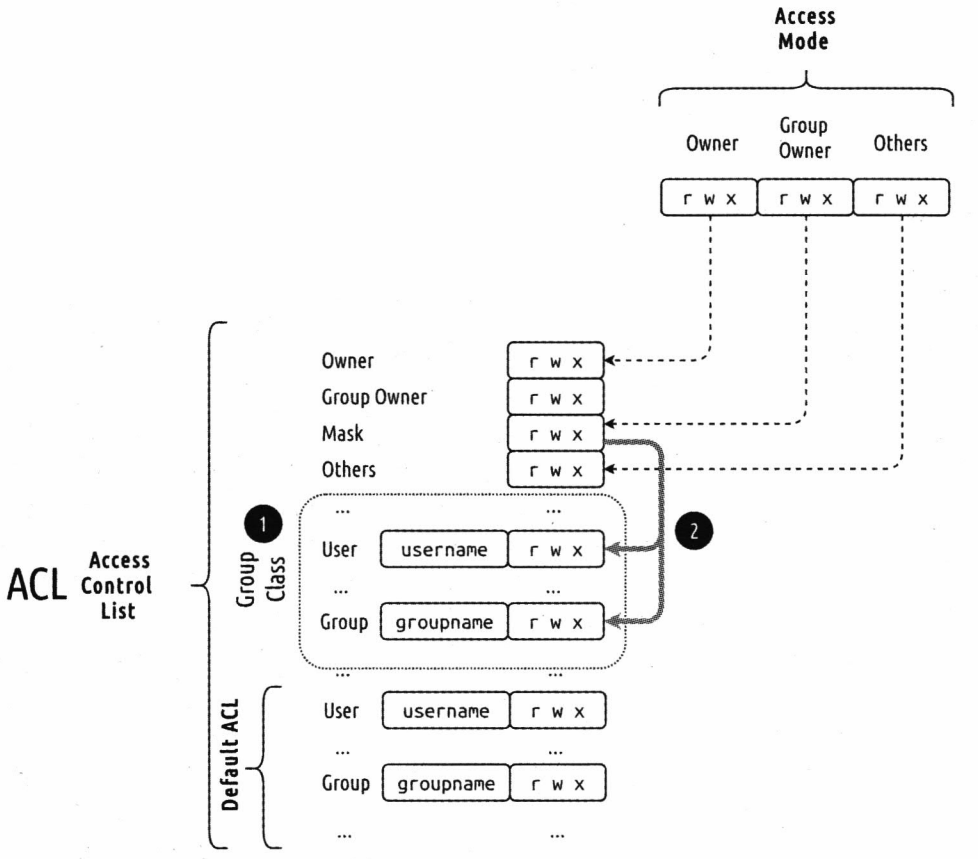


Рис. 3.4. Списки контроля доступа к файлам

В примере из листинга 3.44 для всех «остальных» (не входящих в группу **candy**) пользователей отзываются все права на каталог **/srv/kingdom/stash**, но для отдельного пользователя **jake** (не являющегося членом группы **candy**) назначаются права чтения, модификации и прохода в него **rwx**.

Листинг 3.44. Списки контроля доступа

```
    finn@ubuntu:/srv/kingdom$ ls -lad .
    drwxrwsr-t 2 bubblegum candy 4096 окт.  23 21:19 .
    finn@ubuntu:/srv/kingdom$ id
    uid=1001(finn) gid=1001(finn) группы=1001(finn),1007(candy)
    finn@ubuntu:/srv/kingdom$ mkdir stash
    finn@ubuntu:/srv/kingdom$ chmod o= stash/
    finn@ubuntu:/srv/kingdom$ ls -lad stash/
    drwxrws--- 2 finn candy 4096 нояб.  4 13:05 stash/
    finn@ubuntu:/srv/kingdom$ id jake
?   uid=1002(jake) gid=1002(jake) группы=1002(jake)
    finn@ubuntu:/srv/kingdom$ setfacl -m u:jake:rwx stash/
    finn@ubuntu:/srv/kingdom$ ls -lad stash/
    drwxrws---+ ☚ 2 finn candy 4096 нояб.  4 13:05 stash/
    finn@ubuntu:/srv/kingdom$ getfacl stash/
    ...          ...             ...             ...             ...
    user::rwx
☛   user:jake:rwx
    group::rwx
    mask::rwx
☛   other::---
```

Групповая маска

С расширением множеств субъектов, для которых определены права доступа при помощи списков контроля доступа, возникает вопрос об их смысловой совместимости с режимом доступа, в котором определены всего три множества: пользователь-владелец, группа-владелец и все остальные.

Программы, работающие в соответствии с режимом доступа, считают, что если и существует некоторое количество «выделенных» субъектов со специально определенными правами, отличными от «всех остальных», то *все* эти субъекты входят в группу владельцев. Списки контроля доступа позволяют «выделять» субъектов из числа любых пользователей и групп и определять их права произвольным образом. Когда программа, работающая в соответствии с режимом доступа, назначает права группе-владельцу, она вправе считать что *все* «выделенные» субъекты будут ограничены этими правами. Именно поэтому в список контроля доступа добавлена групповая *маска* прав, определяющая ограничения «выделенных» субъектов (называемых групповым *классом* субъектов) в их индивидуальных правах.

В примере из листинга 3.45 каталогу, которому определены индивидуальные права **rwx** для пользователя **jake** в списке контроля доступа, изменяют права группы-

владельцев ❶ классического режима доступа. Получившийся *режим* доступа ❶ означает, что *никому* кроме владельца файла не разрешено записывать в этот файл, что противоречит индивидуальным правам *списка* доступа. Противоречия устраняются маской ❷ *списка* доступа, ограничивающей эффективные права пользователя **jake** так, что бы это соответствовало *режиму* доступа *по смыслу*.

Листинг 3.45. Групповая маска ACL

```
    finn@ubuntu:/srv/kingdom$ ls -lad stash/
    drwxrws---+ 2 finn candy 4096 нояб.  4 13:05 stash/
    finn@ubuntu:/srv/kingdom$ getfacl stash/
    ...          ...          ...          ...          ...
    user::rwx
    user:jake:rwx
    group::rwx
    mask::rwx
    other::---
❶   finn@ubuntu:/srv/kingdom$ chmod g-w stash/
    finn@ubuntu:/srv/kingdom$ ls -lad stash/
❶   drwxr-s---+ 2 finn candy 4096 нояб.  4 13:05 stash/
    finn@ubuntu:/srv/kingdom$ getfacl stash/
    ...          ...          ...          ...          ...
    user::rwx
!   user:jake:rwx                #effective:r-x
!   group::rwx                   #effective:r-x
❷   mask::r-x
    other::---
```

Права по умолчанию

При создании новых файлов в каталогах с индивидуальными правами пользователей в списках доступа зачастую складывается ситуация, когда пользователи (имеющие доступ в каталоги) не получают нужных прав доступа к создаваемым файлам в этих каталогах. В большинстве случаев это противоречит здравому смыслу, т. к. все файлы некоторого каталога являются в определенном смысле «общими» для множества пользователей, которым разрешен доступ в сам каталог.

В примере из листинга 3.46 в каталоге **stash**, куда пользователю **jake** предоставлен индивидуальный доступ (см. листинг 3.45) при создании файла **README**, он в силу SGID для каталога (см. листинг 3.42) передается группе **candy**. В группу **candy** пользователь **jake** не входит (именно поэтому ему назначены индивидуальные права в листинге 3.44), в результате чего файл ему никак не будет доступен.

Проблема решается назначением ❷ каталогу **stash** прав доступа «по умолчанию» (default), которые будут унаследованы ❸ файлами, создающимися в этом каталоге.

Листинг 3.46. Права по умолчанию

```
finn@ubuntu:/srv/kingdom$ cd stash/
finn@ubuntu:/srv/kingdom$ umask 0007
finn@ubuntu:/srv/kingdom/stash$ touch README
finn@ubuntu:/srv/kingdom/stash$ ls -la README
-rw-rw---- ☚ 1 finn      candy ☚ 0 нояб.  4 14:16 README
finn@ubuntu:/srv/kingdom/stash$ id jake
❶ uid=1002(jake) gid=1002(jake) группы=1002(jake)
❷ finn@ubuntu:/srv/kingdom/stash$ setfacl -m d:u:jake:rw .
finn@ubuntu:/srv/kingdom/stash$ getfacl .
# file: .
...          ...             ...             ...             ...
default:user::rwx
☛ default:user:jake:rw-
default:group::rwx
default:mask::rwx
default:other::---
finn@ubuntu:/srv/kingdom/stash$ touch README.jake
finn@ubuntu:/srv/kingdom/stash$ ls -l README.jake
-rw-rw----+ ☚ 1 finn      candy    0 нояб.  4 14:17 README.jake
finn@ubuntu:/srv/kingdom/stash$ getfacl README.jake
# file: README.jake
...          ...             ...             ...             ...
❸ user:jake:rw-
...          ...             ...             ...             ...
```

3.6. Мандатное (принудительное) разграничение доступа

В Linux дискреционные механизмы разграничения доступа (DAC, **d**iscretionary **a**ccess **c**ontrol) являются основными и всегда активны. Их использование предполагает, что владельцы объектов правильно распоряжаются правами доступа к находящимся в их владении объектам. Например, пользовательские закрытые ключи, используемые службой W:[SSH] *(см. разд. 6.4.1)* в каталоге **~/.ssh** или ключи W:[GnuPG] в каталоге **~/.gnupg**, и прочие секретные данные (подобные ключи доступа в банковские информационные системы) должны быть *недоступны* никому, кроме их владельца.

Запускаемые пользователем программы выполняются от лица запустившего их пользователя и имеют доступ к файлам согласно установленным режимам или спискам доступа. В примере из листинга 3.47 клиент **ssh(1)**, браузер **firefox(1)** и коммуникатор **skype** имеют абсолютно равные возможности по чтению и модификации (!) пользовательского закрытого ключа **~/.ssh/id_rsa**, тогда как настоящим «владельцем» ключей является только **ssh(1)**.

Листинг 3.47. Необходимость MAC

```
finn@ubuntu:~$ ls -l .ssh
итого 8
-rw------- 1 finn finn 1675 нояб.  4 16:06 id_rsa
-rw-r--r-- 1 finn finn  393 нояб.  4 16:06 id_rsa.pub

finn@ubuntu:~$ ps fux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
finn     20650  0.0  0.1  13488  8576 pts/0    S    16:08   0:00 -bash
finn     20943  0.0  0.0   6668  2284 pts/0    S+   16:19   0:00  \_ ssh jake@grex.org
...        ...                 ...
finn     21094 13.1  1.5 592676 127820 pts/2   Rl+  16:27   0:09 /usr/lib/firefox/firefox
...        ...                 ...
finn     21790 29.7  2.1 575000 172084 ?       Sl   16:38   0:09 skype
```

Абсолютно естественно предполагать, что программы **firefox(1)** и **skype(1)** не имеют никаких *намерений* доступа к пользовательским ключам SSH. Можно даже *доверять* программе **firefox(1)**, штатно установленной из доверенного источника (дистрибутива), где она была изготовлена из открытых *исходных* текстов, подлежащих верификации. Однако нет никаких оснований доверять закрытому **skype**, поставляемому в бинарном виде.

Более того, предоставлять доступ программам **firefox(1)** и **skype(1)** к SSH-ключам пользователя нет никакой необходимости, во-первых, просто потому, что это выходит за рамки набора минимально необходимых условий их *целевого* функционирования. Во-вторых, практически в любой программе есть ошибки, используя которые злоумышленник может осуществлять *непреднамеренные* действия в свою пользу. Таким воздействиям особенно подвержены программы, использующие сетевой обмен с *недоверенной* внешней средой — клиенты и серверы сетевых служб операционной системы. Тем временем, дискреционный подход и механизмы служат для разграничения доступа разных пользователей к файлам, но никак не предназначены для разграничения доступа программ одного и того же пользователя к разным файлам этого пользователя.

Для разграничения доступа *субъектов* — программ к *объектам* — файлам дерева каталогов используют так называемый *мандатный* (от англ. *mandatory* — обязательный или принудительный) подход (MAC, **m**andotary **a**ccess **c**ontrol), предполагающий следование обязательным *правилам доступа* к файлам, назначаемым *администраторами* системы. Правила доступа строятся на основе знания о внутреннем устройстве программ и представляют собой описание набора минимально необходимых условий их *целевого* функционирования.

Таким образом, в мандатных правилах, ограничивающих доступ к SSH-ключам пользователя, только программе **ssh(1)** должен быть *разрешен* доступ для непосредственного выполнения своих прямых функций, а программам **firefox(1)** и **skype(1)** в доступе к SSH-ключам должно быть *отказано*.

3.6.1. Модуль принудительного разграничения доступа AppArmor

В Linux мандатные механизмы разграничения являются дополнительными и активируются по желанию пользователя или дистрибьютора. Так, например, в Ubuntu Linux по умолчанию устанавливается и активируется модуль принудительного разграничения доступа **apparmor(7)**. Модуль **W:[AppArmor]** имеет некоторое количество готовых к употреблению наборов мандатных правил (называемых *профилями*) для ограничения (англ. *confine*) доступа субъектов — программ к объектам операционной системы — файлам, сетевым протоколам, портам TCP/UDP и пр. Правила AppArmor идентифицируют программы и файлы на основе их полных путевых имен. При этом каждый профиль описывает ограничения для одной определенной запускаемой программы (а так же для других, запускаемых этой программой, т. е. «подчиненных» программ), а программы, для которых профили не определены, никак не ограничиваются (англ. *uniconfined*).

В примере из листинга 3.48 проиллюстрировано использование команды **aa-status(8)**, показывающей статус модуля принудительного контроля доступа AppArmor. В распоряжении модуля имеется профиль ❶ программы **skype** в режиме принуждения (**enforce**) и профиль ❷ программы **firefox(1)** в режиме оповещения (**complain**). Более того, процесс **skype** с идентификатором **30173** принуждается (**enforce**) модулем контроля к выполнению правил профиля, а процесс **firefox** с идентификатором **10335** только отслеживается (**complain**) на предмет нарушений правил.

Листинг 3.48. Модуль мандатного разграничения доступа AppArmor

```
finn@ubuntu:~$ sudo aa-status
apparmor module is loaded.
50 profiles are loaded.
26 profiles are in enforce mode.
```

```
❶    /usr/bin/skype
   ...          ...          ...
   24 profiles are in complain mode.
   ...          ...          ...
❷    /usr/lib/firefox/firefox{,*[^s][^h]}
   ...          ...          ...
   18 processes have profiles defined.
   3 processes are in enforce mode.
   ...          ...          ...
🔒    /usr/bin/skype (30173)
   6 processes are in complain mode.
🔓    /usr/lib/firefox/firefox{,*[^s][^h]} (10335)
   ...          ...          ...
   0 processes are unconfined but have a profile defined.
```

Нарушения мандатных правил процессами, находящимися в режиме оповещения (**complain**), приводят только к журналированию сообщений аудита об обнаруженных нарушениях. Попытки нарушения мандатных правил процессами, находящимися в режиме принуждения (**enforce**), пресекаются модулем контроля доступа в виде отказа в доступе к тому или иному запрашиваемому объекту.

Выполненный (листинг 3.49) при помощи команды **apparmor_parser(8)** анализ полного (**-p**) набора правил профиля программы **/usr/bin/firefox** показывает, что обращения к любым файлам каталога **.ssh** явно запрещены указанием **deny** и подлежат обязательному аудиту согласно указанию **audit**. При помощи команды **aa-enforce(8)** профиль переводится в режим **enforce**, в результате чего доступ **firefox(1)** к ключам пользователя оказывается запрещенным.

Листинг 3.49. Мандатные правила профиля firefox

```
   finn@ubuntu:~$ apparmor_parser -p /etc/apparmor.d/usr.bin.firefox
   ...          ...          ...          ...          ...
     audit deny ➧ @{HOME}/.ssh/** mrwkl,
   ...          ...          ...          ...          ...
   finn@ubuntu:~$ firefox ~/.ssh/id_rsa
?  -----BEGIN RSA PRIVATE KEY-----
   ...          ...          ...          ...          ...
   -----END RSA PRIVATE KEY-----
🗝  finn@ubuntu:~$ sudo aa-enforce firefox
   Setting /etc/apparmor.d/usr.bin.firefox to enforce mode.
   finn@ubuntu:~$ firefox ~/.ssh/id_rsa
!
```

Стоит отметить, что команды **aa-enforce(8)** и **aa-status(8)** выполняются от лица *суперпользователя*, единолично управляющего модулем принудительного контроля доступа AppArmor, но детальное рассмотрение синтаксиса правил и процедур управления модулем относится к задачам администрирования операционной системы, которые выходят за рамки этой книги.

3.6.2. Модуль принудительного разграничения доступа SELinux

Еще одна известная реализация мандатных механизмов разграничения доступа в RedHat/CentoOS/Fedora Linux называется **W:[SELinux]** (Security Enhanced Linux) и имеет несколько моделей применения и соответствующих им наборов мандатных правил, называемых *политиками*. Политики SELinux описывают ограничения доступа субъектов — программ к объектам операционной системы — файлам, сетевым портам протоколов TCP и UDP и пр. на основе *меток* безопасности.

Самая распространенная из моделей применения, похожая на AppArmor, представлена целевой (англ. *target*) политикой, предполагает ограничение прав доступа только *определенных* программ, тогда как остальные программы никак не ограничиваются. Другие модели применения представлены многоуровневой (MLS, multilayer security и ее вариантом MCS, multicategory security) и строгой (англ. *strict*) политиками.

В многоуровневой политике определяются уровни (и категории) доступа, например «не секретно», «для служебного пользования», «секретно», «совершенно секретно» и т. д., и реализуется принцип «no read up, no write down», т. е. запрещение читать файлы с меткой более высокого уровня и запрещение изменять файлы с меткой более низкого уровня. В строгой политике, ограничивающей права доступа *всех* процессов, запрещается все, что не разрешено явно.

Листинг 3.50. Модуль мандатного разграничения доступа SELinux

```
[lich@fedora ~]$ sestatus
SELinux status:                 enabled
SELinuxfs mount:                /sys/fs/selinux
SELinux root directory:         /etc/selinux
❶ Loaded policy name:           targeted
❷ Current mode:                 enforcing
Mode from config file:          enforcing
Policy MLS status:              enabled
Policy deny_unknown status:     allowed
Max kernel policy version:      29
```

В листинге 3.50 при помощи команды **sestatus(8)** показан статус модуля принудительного контроля доступа SELinux. Аналогично AppArmor, у модуля SELinux имеются разрешительный (**permissive**) и принудительный (**enforcing**) режимы ❷ работы. В разрешительном режиме нарушения мандатных правил процессами приводят только к журналированию сообщений аудита об обнаруженных нарушениях. Попытки нарушения мандатных правил процессами, находящимися в принудительном режиме, пресекаются модулем контроля доступа в виде отказа в доступе к тому или иному запрашиваемому объекту.

В отличие от AppArmor, модуль SELinux оперирует не профилями отдельных программ, а общим набором правил ❶, называемых политикой (**policy**). Сами правила политики описывают возможные отношения между процессами и файлами согласно их меткам безопасности. Каждая метка состоит из четырех компонент **user:role:type:level**, которые могут быть использованы лишь частично. Например, в целевой политике используются только составляющая «тип» (**type**) и принцип «соблюдения типов» (TE, type enforcement), согласно которому доступ программы к файлу разрешается, если они имеют «совместимые» типы.

В примере из листинга 3.51 показаны метки безопасности процессов **bash** и **ps** сеанса пользователя **lich** и процесса **httpd** системной службы Web-сервера. В *целевой* (**targeted**) политике программы, запускаемые в пользовательских сеансах, и, соответственно, их процессы никак не ограничиваются, что и обозначает тип **unconfined_t** их метки безопасности. Наоборот, процессы всех системных служб и являются собственно «целями» принудительного контроля доступа целевой политики, поэтому выполняются со своими типами, например **httpd_t** для службы Web-сервера.

Листинг 3.51. Мандатные метки процессов

```
[lich@centos ~]$ ps -Z
LABEL                              PID TTY          TIME CMD
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 8893 pts/0 00:00:00 bash
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 9454 pts/0 00:00:00 ps
[lich@centos ~]$ ps -ZC httpd
LABEL                              PID TTY          TIME CMD
system_u:system_r:httpd_t:s0      3262 ?        00:00:00 httpd
...           ...              ...              ...              ...
system_u:system_r:httpd_t:s0      3267 ?        00:00:00 httpd
```

Политика содержит правила назначения меток *процессам* тех *программ*, исполняемые файлы которых тоже имеют свои метки, проиллюстрированные в листинге 3.52. Так, например, целевая политика содержит правила, согласно которым

процессы Web-сервера **httpd** получают метку с типом **httpd_t**, потому что выполняют программы **/usr/sbin/httpd** с меткой **httpd_exec_t**.

Листинг 3.52. Мандатные метки файлов программ

```
[lich@centos ~]$ ls -Z /usr/sbin/httpd
-rwxr-xr-x. root root system_u:object_r:httpd_exec_t:s0 /usr/sbin/httpd
```

Файлы с данными тоже размечаются своими метками. Например, в листинге 3.53 иллюстрируются метки файлов в домашнем каталоге пользователя **lich**, где при помощи меток дифференцированы разные пользовательские данные *по смыслу*.

Листинг 3.53. Мандатные метки файлов

```
[lich@centos ~]$ ls -Za ~
drwxr-xr-x. lich lich unconfined_u:object_r:user_home_t:s0 Desktop
...          ...              ...              ...              ...
drwxr-xr-x. lich lich unconfined_u:object_r:audio_home_t:s0 Music
...          ...              ...              ...              ...
drwx------. lich lich unconfined_u:object_r:ssh_home_t:s0 .ssh
...          ...              ...              ...              ...
```

В частности, пользовательские ключи SSH в каталоге **~/.ssh** имеют метку с типом **ssh_home_t**, на основании чего и можно ограничить к ним доступ таких программ, как **skype** или **firefox**. Для этого нужно просто выполнять эти программы в процессах с такими типами в их метках, доступ которых к файлам с метками, имеющими тип **ssh_home_t**, ограничен правилами политики.

В листинге 3.54 при помощи команды **sesearch(1)** производится поиск разрешительных (**-A**, allow) правил в политике безопасности, разрешающих (**-p**, permissions) всем субъектам с исходным (**-s**, source) типом **unconfined_t** открывать (**open**) объекты файлового (**file**) класса (**-c**, class), имеющие целевой (**-t**, target) тип **ssh_home_t**.

Листинг 3.54. Мандатные правила типа unconfined_t

```
[lich@centos ~]$ sesearch -A -s unconfined_t -t ssh_home_t -c file -p open
Found 2 semantic av rules:
   allow files_unconfined_type file_type: file { ioctl read ... open audit_access } ;
❶  allow unconfined_t user_home_type: file { ioctl read write create ... rename open } ;
[lich@centos ~]$ seinfo -tssh_home_t -x
```

```
    ssh_home_t
...          ...          ...          ...          ...
❷     user_home_type
[lich@centos ~]$ seinfo -auser_home_type -x
    user_home_type
        audio_home_t
...          ...          ...          ...          ...
❸       ssh_home_t
...          ...          ...          ...          ...
        audio_home_t
```

В результате найдено правило ❶, разрешающее доступ субъектов с типом **unconfined_t** к объектам, типы которых «обобщаются» атрибутом **user_home_type**.

При помощи команды **seinfo(1)** уточняется, что тип (**-t**, type) **ssh_home_t** действительно обобщен атрибутом ❷ **user_home_type**, и, наоборот, атрибут **user_home_type** обобщает такие «пользовательские» типы ❸, как **ssh_home_t**, **audio_home_t**, **user_home_t** и пр.

Другими словами, всем процессам пользовательского сеанса (т. к. они имеют тип **unconfined_t**) разрешено обращаться к пользовательским SSH-ключам (имеющим тип **ssh_home_t**) и любым другим пользовательским данным (с типами **audio_home_t**, **user_home_t** и пр.).

Для ограничения программ в доступе к SSH-ключам пользователей потребуется некоторый другой тип (отличный от **unconfined_t**), которому правилами политики будет запрещено обращаться к файлам с типом **ssh_home_t**. Такими типами в целевой политике являются несколько типов «песочниц» (sandbox), например **sandbox_t** или **sandbox_net_t** (разрешающий сетевые соединения).

В примере из листинга 3.55 поиск ❶ разрешающих правил показывает, что процессам с типом **sandbox_net_t** не разрешено открытие файлов с типом **ssh_home_t**, но разрешены ❷ TCP-соединения. Таким образом, тип **sandbox_net_t** оказывается подходящим кандидатом процессов тех программ, которым нужно ограничить доступ к SSH-ключам и другим файлам пользователя.

Листинг 3.55. Мандатные правила типа sandbox_net_t

```
❶ [lich@centos ~]$ sesearch -A -s sandbox_net_t -t ssh_home_t -c file -p open
!
❷ [lich@centos ~]$ sesearch -A -s sandbox_net_t -c tcp_socket -p connect
Found 4 semantic av rules:
    allow sandbox_net_t sandbox_net_t: tcp_socket { ... read write ... connect ... shutdown } ;
...          ...          ...          ...          ...
[lich@centos ~]$ links -dump ~/.ssh/id_rsa
```

```
?  -----BEGIN RSA PRIVATE KEY-----
   ...          ...          ...          ...          ...
   -----END RSA PRIVATE KEY-----
    [lich@centos ~]$ sandbox -t sandbox_net_t links -dump ~/.ssh/id_rsa
❸  ELinks: Отказано в доступе
```

Запустить программу в процессе с соответствующей меткой позволяет команда **sandbox(8)**, специально предназначенная для формирования «песочниц» при помощи модуля SELinux и соответствующих **sandbox_*_t**-типов. В примере из листинга 3.55 текстовый Web-браузер **links(1)**, выполненный в процессе с типом **sandbox_net_t**, в действительности оказывается ограниченным ❸ в доступе к SSH-ключам пользователей, что и требовалось получить.

3.7. Дополнительные свойства файлов

3.7.1. Расширенные атрибуты файлов

Как было показано выше (см. листинг 3.4), базовые права доступа, дополнительные атрибуты SUID/SGID, владельцы, счетчик имен и другие *основные свойства* файлов хранятся в их метаданных. Кроме этого, файлам могут быть назначены списки контроля доступа (см. листинг 3.44) и метки безопасности SELinux (см. листинг 3.50), которые являются их *дополнительными свойствами* и хранятся за пределами метаданных, при помощи расширенных атрибутов **attr(5)**.

Каждый расширенный атрибут имеет имя вида **namespace.attrname**, при этом пространства имен **namespace** определяют назначение атрибута. Пространство имен **system** используется системными (ядерными) компонентами, например, для списков контроля доступа POSIX ACL. Пространство имен **security** используется системными компонентами безопасности, в частности для хранения привилегий исполняемых программ (**capabilities(7)**, *см. разд. 4.5.2*) и меток модуля принудительного контроля доступа SELinux (см. листинги 3.52 и 3.53). Пространства имен **trusted** и **user** предназначены для атрибутов внеядерных компонент — программ, выполняющихся привилегированным и обычными пользователями соответственно.

Для просмотра и назначения внеядерных (пользовательских) расширенных атрибутов используются утилиты **getfattr(1)** и **setfattr(1)**. Читать *пользовательские расширенные атрибуты* файла разрешено тем же субъектам, которым разрешено чтение *данных* этого файла. Аналогично устанавливать (изменять) и удалять *пользовательские расширенные атрибуты* файла могут субъекты, допущенные к записи данных этого файла.

Ядерные (системные) атрибуты обычно управляются специально предназначенными командами, например, **getfacl(1)** и **setfacl(1)** предназначены для списков контроля

доступа, команды **getcap(8)** и **setcap(8)** — для привилегий исполняемых программ, команды **chcon(1)** и **ls(1) -Z** — для мандатных меток. Системные атрибуты, как правило, всегда доступны для чтения, но их установка и изменение требуют определенных привилегий процесса *(см. разд. 4.5.2)*.

Листинг 3.56. Расширенные системные атрибуты файлов

```
finn@ubuntu:~$ getcap /usr/bin/gnome-keyring-daemon
/usr/bin/gnome-keyring-daemon = cap_ipc_lock+ep ❶

finn@ubuntu:~$ getfattr -d -m - /usr/bin/gnome-keyring-daemon
getfattr: Удаление начальных '/' из абсолютных путей
# file: usr/bin/gnome-keyring-daemon
security.capability=0sAQAAAgBAAAAAAAAAAAAAAAAAAAA=

finn@ubuntu:~$ cd /srv/kingdom/stash
finn@ubuntu:/srv/kingdom/stash$ ls -l README.jake
-rw-rw----+ 1 finn      candy    0 нояб.  4 14:17 README.jake
finn@ubuntu:/srv/kingdom/stash$ getfacl README.jake
# file: README.jake
# owner: finn
# group: candy
user::rw-
user:jake:rw- ❷
group::rwx                      #effective:rw-
mask::rw-
other::---
finn@ubuntu:/srv/kingdom/stash$ getfattr -d -m - README.jake
# file: README.jake
system.posix_acl_access=0sAgAAAAEABgD/////AgAGAOoDAAAEAAcA/////xAABgD/////IAAAAP////8=

finn@ubuntu:/srv/kingdom/stash$ setfattr -n user.color -v orange README.jake
finn@ubuntu:/srv/kingdom/stash$ setfattr -n user.flavour -v vanilla README.jake
finn@ubuntu:/srv/kingdom/stash$ getfattr -d README.jake
# file: README.jake
user.color="orange"
user.flavour="vanilla"
```

В примере из листинга 3.56 показано, что привилегии исполняемых программ ❶ на самом деле сохранены в атрибуте **security.capability**, списки контроля доступа ❷ —

в атрибуте **system.posix_acl_access**, а в атрибутах пространства имен **user** можно разместить любые значения. Наиболее известным применением пользовательских атрибутов (листинг 3.57) являются атрибуты **user.xdg.origin.url** и **user.xdg.referrer.url**, использующиеся браузером **chromium-browser(1)** для сохранения URL файлов, которые были загружены из Интернета.

Листинг 3.57. Расширенные пользовательские атрибуты файлов

```
finn@ubuntu:~/Downloads$ getfattr -d TLCL-13.07.pdf
# file: TLCL-13.07.pdf
user.xdg.origin.url="http://freefr.dl.sourceforge.net/project/linuxcommand/TLCL/13.07/
TLCL-13.07.pdf"
user.xdg.referrer.url="http://sourceforge.net/projects/linuxcommand/files/TLCL/13.07/
TLCL-13.07.pdf/download"
```

3.7.2. Флаги файлов

Кроме «общих» расширенных атрибутов, которые используются разными компонентами операционной системы, каждая файловая система зачастую имеет собственные атрибуты файлов, управляющие ее поведением и функциями при доступе к файлам. Так, файловые системы **ext2/ext3/ext4** управляются специальными атрибутами-флагами, например **a** (**a**ppend only), **i** (**i**mmutable), **s** (secure deletion), **S** (synchronous updates) и пр.

Флаг **s** заставляет файловую систему при удалении файла не только высвобождать принадлежащие ему блоки, но и обнулять их, а флаг **S** заставляет операции записи в файл выполняться синхронно (немедленно), минуя отложенную запись с использованием дискового кэша. Флаг **i** делает файл «неприкасаемым» — его нельзя ни изменить, ни удалить никому, даже суперпользователю **root**. Флаг **a** делает файл «накопительным» (листинг 3.58), т. е. никому не дает изменять ❶ имеющуюся в файле информацию или удалять ❸❹ файл, а позволяет только добавлять данные ❷ в его конец.

Устанавливать флаги файлов разрешено их владельцам, а установка отдельных флагов, например **a** или **i**, требует определенных привилегий процесса *(см. разд. 4.5.2)*. Для просмотра флагов файлов предназначена утилита **lsattr(1)**, а для их изменения — утилита **chattr(1)**, что иллюстрирует листинг 3.58 на примере флага **i**.

Листинг 3.58. Флаги файлов

```
finn@ubuntu:/srv/kingdom/stash$ lsattr
------------e- ./README.jake
------------e- ./README
```

```
finn@ubuntu:/srv/kingdom/stash$ chattr +i README.jake
chattr: Операция не позволяется while setting flags on README.jake
finn@ubuntu:/srv/kingdom/stash$ sudo chattr +i README.jake
finn@ubuntu:/srv/kingdom/stash$ lsattr README.jake
----i--------e- README.jake
finn@ubuntu:/srv/kingdom/stash$ date >[1] README.jake
-bash: README.jake: Операция не позволяется
finn@ubuntu:/srv/kingdom/stash$ rm README.jake
rm: невозможно удалить «README.jake»: Операция не позволяется
finn@ubuntu:/srv/kingdom/stash$ sudo rm README.jake
rm: невозможно удалить «README.jake»: Операция не позволяется ☚!
```

3.8. В заключение

Всестороннее рассмотрение разнообразных файлов, их свойств, атрибутов и контекстов использования неизбежно должно приводить к выводу, что файл является универсальной сущностью, позволяющей организовать однородный доступ к информации, вне зависимости от свойств ее источника. Специальные файлы устройств, именованные каналы и сокеты имеют файловую природу и могут обрабатываться совершенно «обычными» программами за счет идентичности их файлового программного интерфейса, наравне с файлами «обычных» (дисковых), сетевых, псевдофайловых и внеядерных файловых систем.

Подсистема управления процессами, о которой пойдет речь в следующей главе, тоже не обходится без файлов и использует механизм их отображения в память для организации виртуальной памяти и средств межпроцессного взаимодействия, таких как разделяемая память, очереди сообщений[2] и семафоры. Даже сетевые сокеты, рассмотрение которых отложено до *главы 6*, тоже на поверку оказываются файлами.

Таким образом, понимание файла, как основополагающей компоненты операционной системы, дает ключ к пониманию многих других ее частей, а навыки мониторинга файлов или трассировки файлового интерфейса позволяют заглянуть в корень практически всех ее механизмов.

---

[1] О перенаправлениях подробнее *см. разд. 5.3.*

[2] В реализации POSIX, но, к сожалению, не в реализации SYSV.

Глава 4

Управление процессами и памятью

Процессы операционной системы в большинстве случаев отождествляются с выполняющимися программами, что не совсем верно, точнее — совсем не верно. В современных операционных системах, включая Linux, между программой и процессом есть очевидная взаимосвязь, но далеко не такая непосредственная, как кажется на первый взгляд.

4.1. Программы и библиотеки

Программа представляет собой *алгоритм*, записанный на определенном языке, понятном исполнителю программы[1]. Различают *машинный* язык, понятный центральному процессору, и языки более *высоких уровней* (алгоритмические), понятные составителю программы — программисту.

Программы, составленные на языке высокого уровня, в любом случае перед исполнением должны быть транслированы (переведены) на язык исполнителя, что реализуется при помощи специальных средств — *трансляторов*. Различают два вида трансляторов программ — *компиляторы* и *интерпретаторы*. Компилятор транслирует в машинный код сразу всю программу целиком и не участвует в ее исполнении. Интерпретатор, наоборот, пошагово транслирует отдельные инструкции программы и немедленно выполняет их. Например, *командный интерпретатор* при интерактивном режиме пошагово выполняет команды, вводимые пользователем, а в пакетном режиме *(см. главу 5)* так же пошагово выполняет команды, записанные в файле сценария.

Алгоритм, в свою очередь, есть некоторый набор инструкций, выполнение которых приводит к решению некоторой задачи. В большинстве случаев, инструкции алгоритма имеют причинно-следственные зависимости и выполняются исполнителем *последовательно*. Однако если выделить «независимые» поднаборы инструкций

[1] Программа политической партии и программа научной конференции имеют тот же смысл, что и компьютерная программа, но предназначены для других исполнителей.

(независимые ветви), то их можно выполнять несколькими исполнителями одновременно — *параллельно*. Поэтому различают *последовательные* и *параллельные* алгоритмы и соответствующие им последовательные и параллельные программы. Некоторые программы реализуют алгоритмы *общего* назначения, например алгоритмы сжатия или шифрования информации, алгоритмы сетевых протоколов и т. д. Такие программы, востребованные не столько конечными пользователями, сколько другими программами, называют *библиотеками*.

Согласно **hier(7)**, откомпилированные до машинного языка программы размещаются в каталогах **/bin**, **/sbin**, **/usr/bin**, **/usr/sbin**, **/usr/local/bin**, **/usr/local/sbin**, а библиотеки — в каталогах **/lib**, **/usr/lib**, **/usr/local/lib**. Программы имеют специальный бинарный «запускаемый» формат **W:[ELF] executable** и зависят от библиотек, что проиллюстрировано в листинге 4.1 при помощи команды **ldd(1)** (**l**oa**d**er **d**ependencies). Каждая зависимость отображается именем библиотеки ❶ (**SONAME**, **s**hared **o**bject **name**), найденным в системе файлом библиотеки ❷ и адресом[1] в памяти процесса ❸ (32- или 48-битным, в зависимости от платформы), куда библиотека будет загружена.

Листинг 4.1. Программы и библиотеки

```
fitz@ubuntu:~$ which ls
/bin/ls
fitz@ubuntu:~$ file /bin/ls
/bin/ls: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked (uses
shared libs), for GNU/Linux 2.6.24, BuildID[sha1]=0x83531f308f1fa18221be53eaf399303400c14638,
stripped
fitz@ubuntu:~$ ldd /bin/ls
        linux-gate.so.1 =>  (0xb7744000)
        libselinux.so.1 => /lib/i386-linux-gnu/libselinux.so.1 (0xb7708000)
        librt.so.1 => /lib/i386-linux-gnu/librt.so.1 (0xb76ff000)
        libacl.so.1 => /lib/i386-linux-gnu/libacl.so.1 (0xb76f5000)
❶       libc.so.6 => ❷ /lib/i386-linux-gnu/libc.so.6 (0xb754b000) ❸
        libdl.so.2 => /lib/i386-linux-gnu/libdl.so.2 (0xb7546000)
        /lib/ld-linux.so.2 (0xb7745000)
        libpthread.so.0 => /lib/i386-linux-gnu/libpthread.so.0 (0xb752b000)
        libattr.so.1 => /lib/i386-linux-gnu/libattr.so.1 (0xb7525000)
fitz@ubuntu:~$ file /lib/i386-linux-gnu/libc.so.6
❹ /lib/i386-linux-gnu/libc.so.6: symbolic link to `libc-2.15.so'
```

[1] Для противодействия эксплуатации уязвимости в программах адрес выбирается случайным образом, см. **W:[ASLR]**.

Нужно заметить, что файла библиотеки **linux-gate.so.1** (реализующей интерфейс системных вызовов к ядру) не существует, т. к. она является виртуальной (VDSO, virtual dynamic shared object), т. е. предоставляется и отображается в память процесса самим ядром, «как будто» является настоящей библиотекой. Кроме того, библиотека **ld-linux.so.2** указана абсолютным путевым именем, поэтому поиск ее файла не производится.

Для большинства библиотек зависимость устанавливается при помощи SONAME вида **lib*NAME*.so.*X*** где **lib** — стандартный префикс (библиотека), **.so** — суффикс (разделяемый объект), **NAME** — имя «собственное», а **.X** — номер версии ее интерфейса (листинг 4.2). По имени SONAME в определенных (конфигурацией компоновщика — см. ld.so(8) и ldconfig(8)) каталогах производится поиск одноименного файла библиотеки, который на самом деле оказывается символической ссылкой ❹ на «настоящий» файл библиотеки. Например, для **6**-й версии интерфейса динамической библиотеки языка **c** (**libc.so.6**) настоящий файл библиотеки называется **libc-2.15.so**, что указывает на версию самой библиотеки как **2.15**.

Листинг 4.2. Версии библиотек

```
fitz@ubuntu:~$ file /lib/i386-linux-gnu/libacl.so.1
/lib/i386-linux-gnu/libacl.so.1: symbolic link to `libacl.so.1.1.0' ➥
```

Аналогично, в листинге 4.2 показано, что для **1**-й версии интерфейса динамической библиотеки списков контроля доступа **acl** (**libacl.so.1**) настоящий файл библиотеки называется **libacl.so.1.1.0**, а это указывает на версию самой библиотеки как **1.1.0**.

Такой подход позволяет заменять (исправлять ошибки, исправлять неэффективные алгоритмы и пр.) библиотеки (при условии неизменности их интерфейсов) *отдельно* от программ, зависящих от них. При обновлении библиотеки **libc-2.15.so**, например, до **libc-2.18.so** достаточно установить символическую SONAME-ссылку **libc.so.6** на **libc-2.18.so**, в результате чего ее начнут использовать все программы с зависимостями от **libc.so.6**. Более того, в системе может быть одновременно установлено любое количество версий одной и той же библиотеки, реализующих одинаковые или разные версии интерфейсов, выбор которых будет указан соответствующими SONAME-ссылками.

Листинг 4.3. Библиотеки — это незапускаемые программы

```
fitz@ubuntu:~$ file /lib/i386-linux-gnu/libc-2.15.so
/lib/i386-linux-gnu/libc-2.15.so: ELF 32-bit LSB shared object, Intel 80386, version 1
(SYSV), dynamically linked (uses shared libs),
BuildID[sha1]=0x87bb99bc340f1345950611a3d9cfec1cb49532dc, for GNU/Linux 2.6.24, stripped
```

```
fitz@ubuntu:~$:file /lib/i386-linux-gnu/libacl.so.1.1.0
/lib/i386-linux-gnu/libacl.so.1.1.0: ELF 32-bit LSB shared object, Intel 80386, version 1
(SYSV), dynamically linked, BuildID[sha1]=0xd48c066c7c7deba7c88505fe434d4601e3e91f50,
stripped
fitz@ubuntu:~$:ldd /lib/i386-linux-gnu/libacl.so.1.1.0
        linux-gate.so.1 =>  (0xb76fd000)
        libattr.so.1 => /lib/i386-linux-gnu/libattr.so.1 (0xb76ca000)
        libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xb7520000)
        /lib/ld-linux.so.2 (0xb76fe000)
```

Библиотеки имеют тот же бинарный формат **W:[ELF]**, что и «запускаемые» программы, но не «запускаемый» **executable**, а «совместно используемый» **shared object**. Библиотеки, являясь пусть и незапускаемыми, но программами, естественным образом тоже зависят от других библиотек, что показано в листинге 4.3. Практически, «запускаемость» **ELF**-файлов (листинг 4.4) зависит не от их типа, а от прав доступа и осмысленности точки входа — адреса первой инструкции, которой передается управление при попытке запуска. Например, библиотеку **libc-2.15.so** можно запустить, в результате чего будет выведена статусная информация.

Листинг 4.4. Запускаемые библиотеки

```
fitz@ubuntu:~$ ls -l /lib/i386-linux-gnu/libc-2.15.so
-rwxr-xr-x 1 root root 1730024 окт.   6  2012 /lib/i386-linux-gnu/libc-2.15.so
fitz@ubuntu:~$ /lib/i386-linux-gnu/libc-2.15.so
GNU C Library (Ubuntu EGLIBC 2.15-0ubuntu10.3) stable release version 2.15, by Roland McGrath
et al.
            ...          ...          ...          ...          ...
Available extensions:
        crypt add-on version 2.1 by Michael Glad and others
        GNU Libidn by Simon Josefsson
        Native POSIX Threads Library by Ulrich Drepper et al
        BIND-8.2.3-T5B
            ...          ...          ...          ...          ...
```

4.1.1. Ядро Linux

Не стоит забывать, что самой главной *программой* операционной системы является ее ядро, которое в Linux состоит из статического стартового модуля (листинг 4.5) в формате **ELF executable** и динамически пристыковываемых программных модулей формата **ELF relocatable** (листинг 4.6). Для выполнения процедуры начальной загрузки стартовый модуль упакован в «самораспаковывающийся» **gzip**-архив формата **bzImage** (**b**ig **z**ipped **image**), который состоит из программы распаковки и, собственно, запакованного стартового модуля.

В листинге 4.5 проиллюстрирован процесс ручного извлечения стартового модуля из архива **/boot/vmlinuz-3.13.0-49-generic** формата **bzImage** ❶, который предварительно копируется ❶ в **/tmp/vmlinuz**. Сначала, при помощи поиска ❷ сигнатуры **1f8b0800**, являющейся признаком заголовка **gzip**-архива, находится смещение запакованного стартового модуля. Затем, ❸ отступая один блок **skip=1** размером **bs=18109** байт от начала **/tmp/vmlinuz**, извлекается запакованный стартовый модуль в файл **/tmp/vmlinux.gz**. Наконец, ❹ стартовый модуль распаковывается в файл **/tmp/vmlinux**, который действительно оказывается статически скомпонованной (т. е. не использующей библиотеки **ELF shared object**) исполняемой **ELF**-программой.

Листинг 4.5. Ядро операционной системы

```
fitz@ubuntu:~$ uname -r
3.13.0-49-generic
fitz@ubuntu:~$ file /boot/vmlinuz-3.13.0-49-generic
/boot/vmlinuz-3.13.0-49-generic: regular file, no read permission
fitz@ubuntu:~$ ls -l /boot/vmlinuz-3.13.0-49-generic
-rw------- 1 root root 5937168 марта 25  2015 /boot/vmlinuz-3.13.0-49-generic
fitz@ubuntu:~$ sudo file /boot/vmlinuz-3.13.0-49-generic
❶ /boot/vmlinuz-3.13.0-49-generic: Linux kernel x86 boot executable bzImage, version 3.13.0-49-
  generic (buildd@komainu) #81~precise1-Ubuntu SMP Wed , RO-rootFS, swap_dev 0x5, Normal VGA
❶ fitz@ubuntu:~$ sudo cat /boot/vmlinuz-3.13.0-49-generic > /tmp/vmlinuz
❷ fitz@ubuntu:~$ grep -aboP '\x1f\x8b\x08\x00' /tmp/vmlinuz
☛ 18109:◆
❸ fitz@ubuntu:~$ dd skip=1 bs=18109 if=/tmp/vmlinuz of=/tmp/vmlinux.gz
326+1 записей получено
326+1 записей отправлено
скопировано 5919059 байт (5,9 MB), 0,0113558 с, 521 MB/с
fitz@ubuntu:~$ file /tmp/vmlinux.gz
/tmp/vmlinux.gz: gzip compressed data, from Unix, max compression
❹ fitz@ubuntu:~$ gunzip -c /tmp/vmlinux.gz > /tmp/vmlinux
gzip: /tmp/vmlinux.gz: decompression OK, trailing garbage ignored
fitz@ubuntu:~$ file /tmp/vmlinux
☛ /tmp/vmlinux: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically linked,
  BuildID[sha1]=0xef157e499a84b8027135b5cde271440fcd44a3a5, stripped
```

Динамические модули загружаются в пространство ядра и пристыковываются к стартовому модулю позднее, уже при работе операционной системы при помощи системных утилит **insmod(8)** или **modprobe(8)**. Для отстыковки и выгрузки ненужных модулей предназначена системная утилита **rmmod(8)**, для просмотра списка ❶

(см. листинг 4.6) загруженных модулей — **lsmod(8)**, а для идентификации свойств и параметров ❷ модулей — утилита **modinfo(8)**. Загрузка и выгрузка модулей реализуется специальными системными вызовами **init_module(2)** и **delete_module(2)**, доступ к списку загруженных модулей — при помощи файла **/proc/modules** псевдофайловой системы **proc(5)**, а идентификация свойств и параметров модулей — чтением специальных секций **ELF**-файлов модулей.

Листинг 4.6. Модули ядра

```
❶ fitz@ubuntu:~$ lsmod
Module                  Size  Used by
                        ...   ...          ...
i915                  733300  4
                        ...   ...          ...
btusb                  27671  0
                        ...   ...          ...
uvcvideo               72275  0
                        ...   ...          ...
r8169                  62856  0
                        ...   ...          ...
❷ fitz@ubuntu:~$ modinfo i915
filename:       /lib/modules/3.13.0-49-generic/kernel/drivers/gpu/drm/i915/i915.ko
license:        GPL and additional rights
description:    Intel Graphics
          ...          ...          ...          ...          ...
fitz@ubuntu:~$ file /lib/modules/3.13.0-49-generic/kernel/drivers/gpu/drm/i915/i915.ko
☛ /lib/modules/3.13.0-49-generic/kernel/drivers/gpu/drm/i915/i915.ko: ELF 32-bit LSB
relocatable, Intel 80386, version 1 (SYSV),
BuildID[sha1]=0x8e55af4617e7299b37468c6bef169d43df5152a5, not stripped
```

Динамические модули ядра зачастую являются драйверами устройств, что проиллюстрировано в листинге 4.7 при помощи утилит **lspci(8)** и **lsusb(8)**, которые сканируют посредством псевдофайловой системы **sysfs** списки обнаруженных ядром на шинах PCI и USB устройств и обслуживающих их драйверов.

Листинг 4.7. Драйверы устройств

```
fitz@ubuntu:~$ lspci -k
          ...          ...          ...          ...          ...
00:02.0 VGA compatible controller: Intel ... Integrated Graphics Controller (rev 09)
     Subsystem: Samsung Electronics Co Ltd Device c0b6
     Kernel driver in use: i915
     Kernel modules: i915
          ...          ...          ...          ...          ...
02:00.0 Ethernet controller: Realtek ...  8168B ... Gigabit Ethernet controller (rev 06)
     Subsystem: Samsung Electronics Co Ltd Device c0b6
```

```
        Kernel driver in use: r8169
        Kernel modules: r8169
fitz@ubuntu:~$ lsusb -t
/:  Bus 02.Port 1: Dev 1, Class=root_hub, Driver=ehci-pci/2p, 480M
    |__ Port 1: Dev 2, If 0, Class=hub, Driver=hub/6p, 480M
        |__ Port 3: Dev 15, If 0, Class='bInterfaceClass ... andled', Driver=btusb, 12M
        |__ Port 3: Dev 15, If 1, Class='bInterfaceClass ... andled', Driver=btusb, 12M
        |__ Port 4: Dev 4, If 0, Class=stor., Driver=usb-storage, 480M
/:  Bus 01.Port 1: Dev 1, Class=root_hub, Driver=ehci-pci/2p, 480M
    |__ Port 1: Dev 2, If 0, Class=hub, Driver=hub/6p, 480M
        |__ Port 4: Dev 3, If 0, Class='bInterfaceClass ... ed', Driver=uvcvideo, 480M
        |__ Port 4: Dev 3, If 1, Class='bInterfaceClass ... ed', Driver=uvcvideo, 480M
```

4.2. Процессы и нити

Сущность *процесса* неразрывно связана с мультипрограммированием и *многозадачностью* операционной системы. Например, в однозадачных операционных системах программы существуют, а процессы — нет. В *однозадачных* операционных системах единовременно *одна* последовательная программа выполняется *одним* исполнителем (центральным процессором), имея возможность безраздельно использовать все доступные ресурсы (память, устройства ввода-вывода и пр.).

В любой программе можно выделить перемежающиеся блоки инструкций, использующих или центральный процессор (ЦП), или устройства ввода-вывода (УВВ). При этом, центральный процессор вынужден простаивать при выполнении программой операций ввода-вывода, например, при ожидании окончания записи (или чтения) блока данных на внешний носитель, или при ожидании окончания передачи (или приема) сетевого кадра, или при ожидании событий с устройств человеко-машинного взаимодействия. С другой стороны, устройства ввода-вывода тоже вынуждены простаивать при выполнении программой вычислительных операций, например, ожидая результата, подлежащего выводу, или ожидая возникновения у программы потребности в новых исходных данных.

Используя такую модель поведения программ, можно провести анализ потребления ими ресурсов при выполнении. Например, компрессоры **gzip(1)**, **bzip(2)** и **xz(1)** считывают очередной блок данных исходного файла, относительно долго упаковывают его и записывают в результирующий файл, а затем повторяют процедуру до исчерпания блоков исходного файла. Количество времени, потраченного на вычислительные операции упаковки, будет много больше количества времени, потраченного на чтение исходных данных и запись результатов, поэтому нагрузка на ЦП будет высокой, а на УВВ — нет. Такой же анализ можно привести и для дубликатора **dd(1)**, копировщика **rsync(1)** или архиватора **tar(1)**, которые, наоборот, почти не вы-

полняют никаких вычислений, а сосредоточены на вводе-выводе больших объемов данных, поэтому при их использовании нагрузка на ЦП будет достаточно низкой, а на УВВ — высокой.

Для командного интерпретатора **bash(1)**, текстовых редакторов **nano(1)** и **vim(1)** и других *интерактивных* программ, взаимодействующих с пользователем, характерны длительные ожидания ввода небольших команд, простая и недолгая их обработка и вывод короткого результата. В результате коэффициент полезного использования и ЦП, и УВВ будет приближен к нулю.

Подобный анализ и желание увеличить коэффициенты полезного использования ресурсов привели к созданию *многозадачных* операционных систем, основывающихся на простой идее псевдоодновременного выполнения *нескольких* последовательных программ *одним* исполнителем. Для этого вместо *простоя* в ожидании окончания операции ввода-вывода, начатой некоторой программой, центральный процессор переключается на *выполнение* другой программы, тем самым увеличивая интегральный коэффициент его полезного использования.

С появлением *мультипрограммной смеси*[1] каждая из ее программ больше не может безраздельно использовать все доступные ресурсы (например, всю память — она одновременно нужна всем программам смеси), в связи с чем операционная система берет на себя задачи диспетчеризации (распределения) ресурсов между ними. В Linux, как и во многих других операционных системах, программы изолируются друг от друга в специальных «виртуальных» средах, обеспечивающих их *процесс* выполнения. Каждая такая среда называется *процессом* и получает долю доступных ресурсов — выделенный участок памяти, выделенные промежутки процессорного времени. Процесс эмулирует для программы «однозадачный» режим выполнения, словно программа выполняется в одиночку, и «безраздельное» использование ресурсов процесса, как будто это все доступные ресурсы.

Параллельные программы, как указывалось ранее, состоят из независимых ветвей, каждая из которых сама по себе укладывается в модель поведения последовательной программы, поэтому *одну* параллельную программу можно выполнять в *нескольких* процессах в псевдоодновременном режиме. *Процессы* операционной системы, таким образом, являются контейнерами для многозадачного выполнения программ, как последовательных, так и параллельных.

В листинге 4.8 при помощи команды **ps(1)** показаны процессы пользователя, упорядоченные в дерево, построенное на основе дочерне-родительских отношений между процессами. Уникальный идентификатор, отличающий процесс от других, выведен в столбце **PID** (**p**rocess **id**entifier), а имя и аргументы программы, запущенной в соответствующем процессе, — в столбце **COMMAND**.

---

[1] Набор программ, между которыми переключается процессор.

В столбце **STAT** показано текущее состояние процесса, например **S** (сон, sleep) или **R** (выполнение, running, или готовность к выполнению, runnable). Процессы, ожидающие завершения их операций ввода-вывода, находятся в состоянии *сна*, в противном случае либо *выполняются*, либо *готовы* к выполнению, т. е. ожидают, когда текущий выполняющийся процесс заснет, и процессор будет переключен на них. В столбце **TIME** показано чистое потребленное процессом процессорное время от момента запуска программы, увеличивающееся только при нахождении им в состоянии выполнения.

Листинг 4.8. Дерево процессов пользователя

```
fitz@ubuntu:~$ ps fx
  PID TTY      STAT   TIME COMMAND
 2156 ?        Ssl    0:02 gnome-session --session=ubuntu
 2410 ?        Ss     0:00  \_ /usr/bin/ssh-agent /usr/bin/dbus-launch --exit-with...
 2432 ?        Sl     0:51  \_ /usr/lib/gnome-settings-daemon/gnome-settings-daemon
 2451 ?        S      1:01  |   \_ syndaemon -i 2.0 -K -R -t
 2442 ?        Rl    27:21  \_ compiz
 2569 ?        Ss     0:00  |   \_ /bin/sh -c /usr/bin/compiz-decorator
 2570 ?        Sl     0:25  |   |   \_ /usr/bin/gtk-window-decorator
 7806 ?        Ss     0:00  |   \_ /bin/sh -c gnome-terminal
 7807 ? ☚      Sl     0:28  |       \_ gnome-terminal
 7813 ?        S      0:00  |           \_ gnome-pty-helper
❶ 8127 pts/1 ☚ Ss     0:00  |           \_ bash
❷ 10808 pts/1  R+     0:00  |           |   \_ ps fx
❶ 9321 pts/0   Ss     0:00  |           \_ bash
❷ 10792 pts/0  S+     0:00  |           |   \_ man ps
  10803 pts/0  S+     0:00  |           |       \_ pager -s
❶ 10610 pts/2  Ss     0:00  |           \_ bash
            ...          ...             ...             ...             ...
```

Управляющий терминал процесса, показанный в столбце **TTY**, используется для доставки ему интерактивных *сигналов (см. разд. 4.8)* при вводе управляющих символов *(см. разд. 2.3)* **intr ^C**, **quit ^** и пр. У части процессов управляющий терминал отсутствует, потому что они выполняют приложения, взаимодействующие с пользователем не посредством терминалов, а через графическую систему *(см. главу 7)*.

Процесс по своему определению изолирует свою программу от других выполняющихся программ, что затрудняет использование процессов для выполнения таких параллельных программ, ветви которых не являются полностью независимыми друг

от друга и должны обмениваться данными. Использование предназначенных для этого средств межпроцессного взаимодействия *(см. разд. 4.9)* при интенсивном обмене приводит к обременению неоправданными накладными расходами, поэтому для эффективного выполнения таких параллельных программ используются легковесные процессы (**LWP**, **l**ight-**w**eight **p**rocesses), они же нити[1] (threads). Механизм нитей позволяет переключать центральный процессор между параллельными ветвями одной программы, размещаемыми в *одном* (!) процессе. Нити никак не изолированы друг от друга, и им доступны абсолютно все ресурсы своего процесса, поэтому задача обмена данными между нитями попросту отсутствует, т. к. все данные являются для них общими.

В примере из листинга 4.9 показаны нити процесса в **BSD**-формате вывода. Выбор процесса производится по его идентификатору **PID**, предварительно полученному командой **pgrep(1)** по имени программы, выполняющейся в искомом процессе. В выводе наличие нитей процесса отмечает флаг **l** (**l**wp) в столбце состояния **STAT**, а каждая строчка без идентификатора **PID** символизирует одну нить. Так как в многонитевой программе переключение процессора производится между нитями, то и состояния сна **S**, выполнения или ожидания **R** приписываются отдельным нитям.

Листинг 4.9. Нити процессов, BSD-формат вывода

```
fitz@ubuntu:~$ pgrep skype
24244
fitz@ubuntu:~$ ps mp 24244
  PID TTY      STAT   TIME COMMAND
24244 ?        -      13:14 skype
    - -        Sl     11:13 -
    - -        Sl      0:00 -
    - -        Sl      0:00 -
                        ...          ...          ...
    - -        Sl      0:00 -
    - -        Sl      0:00 -
    - -        Sl      0:02 -
```

В листинге 4.10 показаны нити процесса в **SYSV**-формате вывода. Выбор процесса производится по имени его программы. Общий для всех нитей идентификатор их процесса отображается в столбце **PID**, уникальный идентификатор каждой нити —

[1] Существует еще один (неудачный, на мой взгляд) перевод понятия thread на русский язык — поток. Во-первых, он конфликтует с переводом понятия stream — поток, а во-вторых, в отличие от stream, thread никуда не течет. А вот процесс (process) содержит в себе нити (thread) абсолютно таким же образом, как и обычная веревка состоит из... нитей.

в столбце LWP (иногда называемый TID, thread identifier), а имя процесса (или собственное имя нити, если задано) — в столбце CMD.

Листинг 4.10. Нити процессов, SYSV-формат вывода

```
fitz@ubuntu:~$ ps -LC gnome-terminal
    PID ☛LWP TTY          TIME CMD
  16749 16749 ?        00:00:08 gnome-terminal
☛ 16749 16750 ?        00:00:00 dconf worker
  16749 16751 ?        00:00:00 gdbus ☚
  16749 16758 ?        00:00:00 gmain
```

4.3. Порождение процессов и нитей, запуск программ

Несмотря на очевидные различия, историю[1] возникновения и развития, нити и процессы объединяет общее назначение — они являются примитивами выполнения некоторого набора последовательных инструкций.

Процессы выполняют или разные последовательные программы целиком, или ветви одной параллельной программы, но в изолированном окружении со своим «частным» (private) набором ресурсов. Нити, наоборот, выполняют ветви одной параллельной программы в одном окружении с «общим» (shared) набором ресурсов. В многозадачном ядре Linux вообще используется универсальное понятие «задача», которая может иметь как общие ресурсы (память, открытые файлы и т. д.) с другими задачами, так и частные ресурсы для своего собственного использования.

Порождение нового процесса (рис. 4.1, *а*) реализуется при помощи системного вызова **fork(2)**, в результате которого ядро операционной системы создает новый дочерний (child) процесс PID_2 — полную копию (COPY) процесса-родителя (parent) PID_1. Вся (за небольшими исключениями) память процесса — состояние, свойства, атрибуты (кроме идентификатора PID) и даже содержимое (программа с ее библиотеками) — наследуется дочерним процессом. Даже выполнение порожденного и порождающего процесса продолжится с одной и той же инструкции их одинаковой программы. Такое клонирование обычно используют параллельные программы с ветвями *(см. разд. 4.3.1)*, выполняющимися в дочерних *процессах*.

Уничтожение процесса (например, при штатном окончании программы) производится с помощью системного вызова **exit(3)**. При этом родительскому процессу доставляется сигнал SIGCHILD, оповещающий о завершении дочернего процесса

[1] Откровенно говоря, нити, в общем, появились в операционных системах раньше, чем изолированные UNIX-процессы, в которые со временем вернулись UNIX-нити...

(см. разд. 4.8). Статус завершения **status**, переданный дочерним процессом через аргументы **exit(3)**, будет сохраняться ядром до момента его востребования родительским процессом при помощи системного вызова **wait(2)**, а весь этот промежуток времени дочерний процесс будет находиться в состоянии **Z** (**z**ombie[1]) (см. столбец **STAT** в листинге 4.8). Родительский процесс может завершиться раньше своих дочерних процессов, тогда логично предположить, что все «осиротевшие» процессы окажутся зомби по завершении, потому как просто некому будет востребовать их статус завершения. На самом деле этого не происходит, потому что «осиротевшим» процессам назначается приемный родитель, в качестве которого выступает прародитель всех процессов **init(8)** с идентификатором **PID=1**.

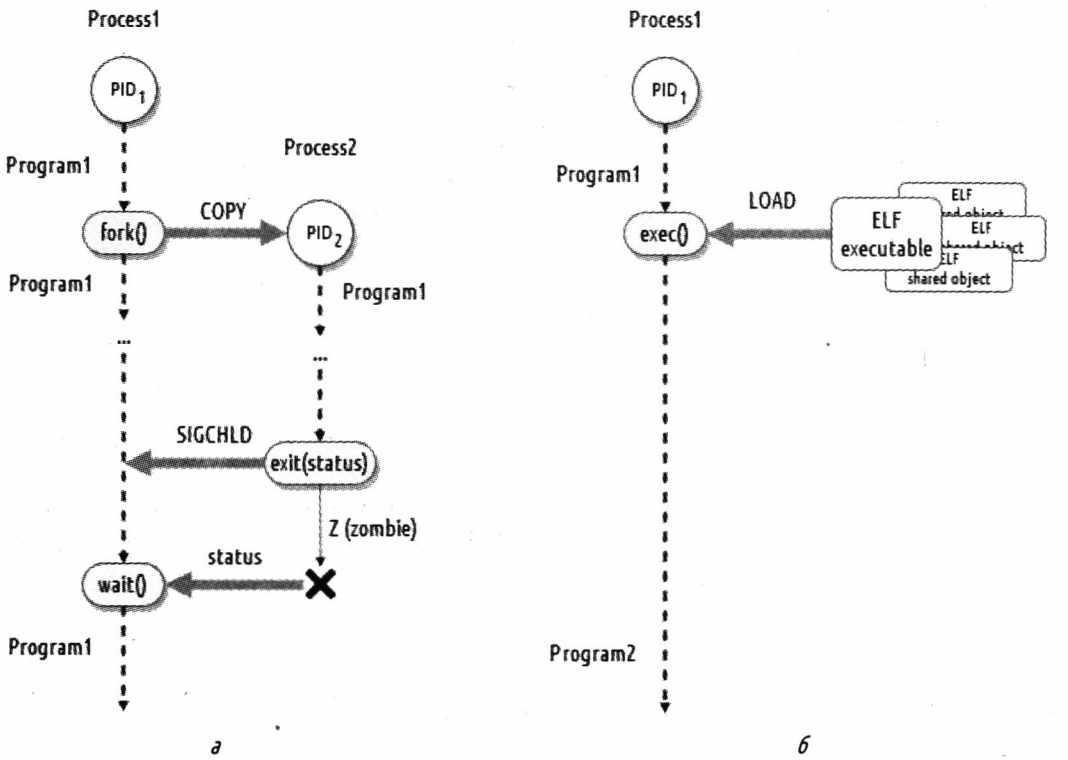


Рис. 4.1. Порождение процессов (*а*) и запуск программ (*б*)

Запуск новой программы (см. рис. 4.1, *б*) реализуется при помощи системного вызова **exec(3)**, в результате которого содержимое процесса PID_1 полностью замещается запускаемой программой и библиотеками, от которых она зависит, а свойства и атрибуты (включая идентификатор **PID**) остаются неизменными. Такое замещение

[1] Процесс почти мертв, но еще не совсем...

обычно используется программами, устанавливающими нужные значения свойств и атрибутов процесса и подготавливающими ресурсы процесса к выполнению запускаемой программы. Например, обработчик терминального доступа **getty(8)** *(см. главу 2)* открывает заданный терминал, устанавливает режимы работы порта терминала, перенаправляет на терминал стандартные потоки ввода-вывода, а затем замещает себя программой аутентификации **login(1)**.

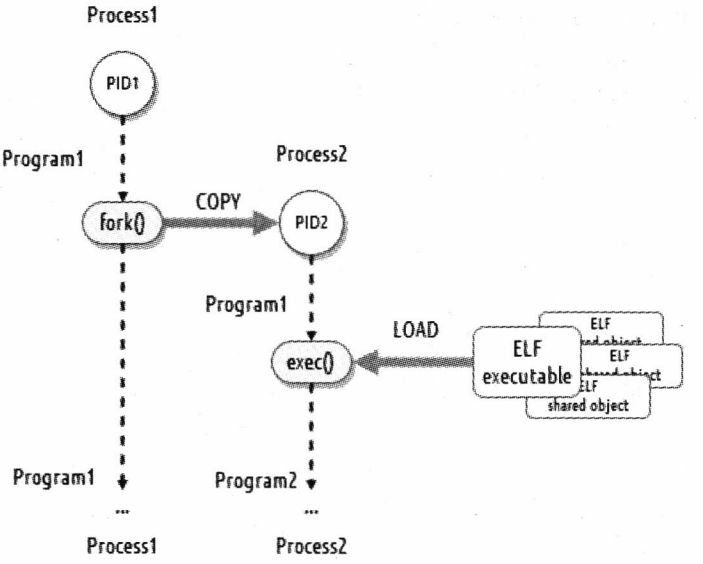


Рис. 4.2. Запуск программы в отдельном процессе

Для запуска новой программы в новом процессе используются оба системных вызова **fork(2)** и **exec(3)**, согласно принципу **fork-and-exec** «раздвоиться и запустить», показанному на рис. 4.2. В примере из листинга 4.8 дерево процессов сформировано именно на основе дочерне-родительских отношений между процессами, формирующимися при использовании принципа **fork-and-exec**. Например, командный интерпретатор **bash(1)** по командам **ps fx** или **man ps** порождает дочерние процессы ❷ и замещает их программами **ps(1)** и **man(1)**. Тем же образом действует графический эмулятор терминала **gnome-terminal(1)** — запуская новый сеанс пользователя ❶ на каждой из своих вкладок, он замещает свои дочерние процессы программой интерпретатора **bash(1)**.

Листинг 4.11 иллюстрирует команду интерпретатора, запущенную в «фоновом» режиме при помощи конструкции асинхронного списка (*см. разд. 5.6*). Аналогично всем предыдущим командам, интерпретатор использует **fork-and-exec** для запуска программы в дочернем процессе с идентификатором **23228**, но не дожидается его завершения при помощи системного вызова **wait(2)**, как обычно, а немедленно ❶ продолжает интерактивное взаимодействие с пользователем, сообщив ему **PID** по-

рожденного процесса и «номер задания» **[1]** команды «заднего фона» *(см. разд. 4.8.1)*. Оповещение о завершении своего дочернего процесса интерпретатор получит позже, при помощи сигнала **SIGCHLD**, и отреагирует соответствующим сообщением ❷ об окончании команды «заднего фона».

Листинг 4.11. Фоновое выполнение программ

```
fitz@ubuntu:~$ dd if=/dev/dvd of=plan9.iso &

❶ [1] 23228
fitz@ubuntu:~$ ps f
  PID TTY      STAT   TIME COMMAND
23025 pts/1    S      0:00 -bash
23228 pts/1    R      1:23  \_ dd if=/dev/dvd of=plan9.iso
23230 pts/1    R+     0:00  \_ ps f
fitz@ubuntu:~$
          ...          ...          ...          ...          ...
fitz@ubuntu:~$ 586896+0 записей получено
586896+0 записей отправлено
скопировано 300490752 байта (300 MB), 14,6916 с, 20,5 MB/с

❷ [1]+  Готово                dd if=/dev/dvd of=plan9.iso
```

В листинге 4.12 показана конвейерная *(см. разд. 5.3)* конструкция интерпретатора, при помощи которой осуществляется поиск самого большого файла с суффиксом **.html** вниз по дереву каталогов, начиная с **/usr/share/doc**. Эта конструкция реализуется при помощи **fork-and-exec** четырьмя параллельно порожденными дочерними процессами интерпретатора, в каждом из которых запущена программа соответствующей части конвейера, при этом дочерние процессы связаны неименованным каналом **pipe(2)** — простейшим средством межпроцессного взаимодействия *(см. разд. 4.9)*. Встроенная команда интерпретатора **wait** реализует одноименный системный вызов **wait(2)** и используется для ожидания окончания всех дочерних процессов конвейера, целиком запущенного в «фоновом» режиме.

Листинг 4.12. Параллельный запуск взаимодействующих программ

```
          ①↴                                    ②↴          ③↴          ④↴
fitz@ubuntu:~$ find /usr/share/doc -type f -name '*.html' | xargs -n1 wc -l | sort -k 1 -nr | head -1 &

[1] 12827
fitz@ubuntu:~$ ps fj
 PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
```

```
11715 11716 11716  9184 pts/0    14699 S     1006   0:01 -bash
11716 12824 12824  9184 pts/0    14699 S     1006   0:00  \_ find … -type f -name *.html
11716 12825 12824  9184 pts/0    14699 S     1006   0:00  \_ xargs -n1 wc -l
11716 12826 12824  9184 pts/0    14699 S     1006   0:00  \_ sort -k 1 -nr
11716 12827 12824  9184 pts/0    14699 S     1006   0:00  \_ head -1
11716 14699 14699  9184 pts/0    14699 R+    1006   0:00  \_ ps fj
fitz@ubuntu:~$ wait
41235 /usr/share/doc/libx11-dev/libX11/libX11.html
[1]+  Готово    find … -type f -name '*.html' | xargs -n1 wc -l | sort -k 1 -nr | head -1
```

4.3.1. Параллельные многопроцессные программы

Как указывалось ранее, параллельные программы зачастую используют процессы для выполнения отдельных ветвей. В эту категорию часто попадают программы сетевых служб, например сервер баз данных W:[PostgreSQL], служба удаленного доступа W:[SSH] и подобные. Листинг 4.13 иллюстрирует программу **postgres(1)**, выполняющуюся в шести параллельных процессах, один из которых — диспетчер ❶, четыре служебных ❷ и еще один ❸ вызван подключением пользователя **fitz** к одноименной базе данных **fitz**. При последующих подключениях пользователей к серверу будут порождены дополнительные дочерние процессы для обслуживания их запросов — по одному на каждое подключение.

Листинг 4.13. Параллельные многопроцессные сервисы

```
   fitz@ubuntu:~$ ps f -C postgres
     PID TTY      STAT   TIME COMMAND
❶  1427 ?        S      0:01 /usr/lib/postgresql/9.1/bin/postgres -D ...
❷  1984 ?        Ss     0:25  \_ postgres: writer process
|  1985 ?        Ss     0:22  \_ postgres: wal writer process
|  1986 ?        Ss     0:05  \_ postgres: autovacuum launcher process
↳  1987 ?        Ss     0:04  \_ postgres: stats collector process
❸ 21619 ?        Ss     0:00  \_ postgres: fitz ➥ fitz [local] idle
   fitz@ubuntu:~$ ssh ubuntu
   fitz@ubuntu's password:
                  ...          ...          ...          ...          ...
   Last login: Sat Nov 21 13:29:33 2015 from localhost
   fitz@ubuntu:~$ ps f -C sshd
     PID TTY      STAT   TIME COMMAND
①   655 ?        Ss     0:00 /usr/sbin/sshd -D
②  21975 ?        Ss     0:00  \_ sshd: fitz [priv]
```

```
③ 22086 ?        S      0:00      \_ sshd: ☛ fitz@pts/1
   fitz@ubuntu:~$ ^Dвыход
   Connection to ubuntu closed.
```

Аналогично, при удаленном доступе по протоколу SSH программа **sshd(8)**, выполняясь в качестве диспетчера ① в одном процессе, на каждое подключение порождает один свой клон ②, который, выполнив аутентификацию и авторизацию пользователя в системе, порождает еще один свой клон ③, имперсонирующийся в пользователя и обслуживающий его запросы.

4.3.2. Параллельные многонитевые программы

Для управления нитями в Linux используют стандартный **POSIX**-интерфейс **pthreads(7)**, реализующийся библиотекой **W:[NPTL]**, которая является частью библиотеки **libc** (см. ★ в листинге 4.4). Интерфейс предоставляет «нитевой» вызов создания нити **pthread_create(3)**, который является условным аналогом «процессных» **fork(2)** и **exec(3)**, вызов завершения и уничтожения нити **pthread_exit(3)**, условно аналогичный **exit(2)**, и вызов получения статуса завершения нити **pthread_join(3)**, условно аналогичный **wait(2)**.

В качестве типичных примеров использования нитей можно привести сетевые сервисы, которые для параллельного обслуживания клиентских запросов используют нити вместо процессов. Например, **WEB**-сервер **apache(8)**, как показано в листинге 4.14, использует два многонитевых процесса по 27 нитей в каждом, что позволяет экономить память (за счет работы всех нитей процесса с общей памятью) при обслуживании большого количества одновременных клиентских подключений.

Листинг 4.14. Параллельные многонитевые сервисы

```
fitz@ubuntu:~$ ps f -C apache2
  PID TTY      STAT   TIME COMMAND
21352 ?        Ss     0:00 /usr/sbin/apache2 -k start
21355 ?        S      0:00  \_ /usr/sbin/apache2 -k start
21357 ?        Sl ☛   0:00  \_ /usr/sbin/apache2 -k start
21358 ?        Sl ☛   0:00  \_ /usr/sbin/apache2 -k start
fitz@ubuntu:~$ ps fo pid,nlwp,cmd -C apache2
  PID  NLWP CMD
21352     1 /usr/sbin/apache2 -k start
21355     1  \_ /usr/sbin/apache2 -k start
21357 ☛  27  \_ /usr/sbin/apache2 -k start
21358    27  \_ /usr/sbin/apache2 -k start
```

```
fitz@ubuntu:~$ ps -fLC rsyslogd
UID          PID  PPID   LWP  C NLWP STIME TTY          TIME CMD
syslog       936     1   936  0    4 Nov19 ?        00:00:01 rsyslogd -c5
syslog       936     1   961  0    4 Nov19 ?        00:00:14 rsyslogd -c5
syslog       936     1   962  0 ☛  4 Nov19 ?        00:00:00 rsyslogd -c5
syslog       936     1   963  0    4 Nov19 ?        00:00:01 rsyslogd -c5
```

Аналогично, сервис централизованной журнализации событий **rsyslogd(8)** использует нити для параллельного сбора событийной информации из разных источников, ее обработки и журнализации. Одна нить считывает события ядра из **/proc/kmsg**, вторая принимает события других служб из файлового сокета **/dev/log**, третья фильтрует поток приятных событий и записывает в журнальные файлы каталога **/var/log/*** и т. д. Параллельная обработка потоков поступающих событий при помощи нитей производится с минимально возможными накладными расходами, что позволяет достигать колоссальной производительности по количеству обрабатываемых сообщений в единицу времени.

Распараллеливание используется не только для псевдоодновременного выполнения ветвей параллельной программы, но и для их настоящего одновременного выполнения несколькими центральными процессорами. В примере из листинга 4.15 показано, как сокращается время сжатия ISO-образа файла при использовании параллельного упаковщика **pbzip2(1)** по сравнению с последовательным **bzip2(1)**. Для измерения времени упаковки применяется встроенная команда интерпретатора **time**, при этом сначала измеряется время упаковки ❶ и время распаковки ❷ последовательным упаковщиком, а затем — время упаковки ① и время распаковки ② параллельным упаковщиком. Команды упаковки запускаются на «заднем фоне», оценивается наличие процессов и нитей паковщиков, после чего они переводятся на «передний фон» встроенной командой интерпретатора **fg** (**fore**ground) и оцениваются затраты времени.

Листинг 4.15. Параллельные многонитевые утилиты

```
fitz@ubuntu:~$ ls -lh plan9.iso
-rw-r--r-- 1 fitz fitz 287M нояб. 28 15:47 plan9.iso
❶ fitz@ubuntu:~$ time bzip2 plan9.iso &
[1] 5545
fitz@ubuntu:~$ ps f
  PID TTY      STAT   TIME COMMAND
 4637 pts/0    S      0:00 -bash
 5545 pts/0    S      0:00  \_ -bash
 5546 pts/0    R ☛    0:12  |   \_ bzip2 plan9.iso
 5548 pts/0    R+     0:00  \_ ps f
```

```
fitz@ubuntu:~$ ps -fLp 5546
UID        PID  PPID   LWP  C NLWP STIME TTY          TIME CMD
fitz      5546  5545  5546 96 ☚ 1 10:50 pts/0    00:00:22 bzip2 plan9.iso
fitz@ubuntu:~$ fg
time bzip2 plan9.iso
                                  ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰ ⑱ ⑲ ⑳ ⑴
real 0m54.780s ☚
user 0m51.772s
sys  0m0.428s
fitz@ubuntu:~$ ls -lh plan9.iso.bz2
-rw-r--r-- 1 fitz fitz 89M нояб. 28 15:47 plan9.iso.bz2
❷ fitz@ubuntu:~$ time bzip2 -d plan9.iso.bz2
                                  ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰ ⑱ ⑲ ⑳ ⑴
real 0m20.705s ☚
user 0m19.044s
sys  0m1.168s
① fitz@ubuntu:~$ time pbzip2 plan9.iso &
[1] 5571
fitz@ubuntu:~$ ps f
  PID TTY      STAT   TIME COMMAND
 4637 pts/0    S      0:00 -bash
 5571 pts/0    S      0:00  \_ -bash
 5572 pts/0    Sl ☚   0:03  |   \_ pbzip2 plan9.iso
 5580 pts/0    R+     0:00  \_ ps f
fitz@ubuntu:~$ ps -fLp 5572
UID        PID  PPID   LWP  C NLWP STIME TTY          TIME CMD
fitz      5572  5571  5578 92 ☚ 8 10:52 pts/0    00:00:43 pbzip2 plan9.iso
      ...                    ...                ...                ...                ...
fitz      5572  5571  5579  1    8 10:52 pts/0    00:00:00 pbzip2 plan9.iso
fitz@ubuntu:~$ fg
time pbzip2 plan9.iso
                                  ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰ ⑱ ⑲ ⑳ ⑴
real 0m24.259s ☚
user 1m22.940s
sys  0m1.888s
fitz@ubuntu:~$ ls -lh plan9.iso.bz2
-rw-r--r-- 1 fitz fitz 89M нояб. 28 15:47 plan9.iso.bz2
② fitz@ubuntu:~$ time pbzip2 -d plan9.iso.bz2
                                  ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰ ⑱ ⑲ ⑳ ⑴
```

```
real 0m7.384s
user 0m25.972s
sys  0m1.396s
```

В результате оценки оказывается, что последовательный упаковщик **bzip2(1)** использует один однонитевой процесс и затрачивает ≈ 54,7 с реального времени на упаковку, из них ≈ 51,7 с проводит в пользовательском режиме **user** и лишь ≈ 0,4 с в режиме ядра **sys** (выполняя системные вызовы, например, **read(2)** или **write(2)**). Соотношение между временем режимов говорит о вычислительном характере программы, т. е. о существенном превалировании времени вычислительных операций упаковки над временем операций ввода-вывода для чтения исходных данных и записи результатов (что подтверждает анализ *разд. 4.2*). Это означает, что нагрузка последовательного упаковщика на центральный процессор близка к максимальной, и его параллельная реализация для псевдоодновременного выполнения ветвей (которые практически никогда не спят) лишена смысла.

Параллельный упаковщик **pbzip2(1)** использует один многонитевой процесс из восьми нитей и затрачивает ≈ 24,4 с реального времени на упаковку, при этом ≈ 1 мин 22,9 с (!) проводит в пользовательском режиме и ≈ 1,8 с в режиме ядра. Прирост производительности упаковки и, как следствие, сокращение времени упаковки достигаются за счет настоящего параллельного выполнения нитей на разных процессорах (ядрах процессора). Соотношение между реальным временем упаковки и суммарно затраченным временем режима пользователя, которое примерно в 3 раза больше, означает использование в среднем трех процессоров для параллельного выполнения вычислительных операций упаковки.

4.3.3. Двойственность процессов и нитей Linux

Как указывалось ранее, процессы и нити в ядре Linux сводятся к универсальному понятию «задача». Задача, все ресурсы которой (память, открытые файлы и т. д.) используются совместно с другими такими же задачами, является «нитью». И наоборот, «процессами» являются такие задачи, которые обладают набором своих частных, индивидуальных ресурсов.

Универсальный системный вызов **clone(2)** позволяет указать, какие ресурсы станут общими в порождаемой и порождающей задачах, а какие — частными. Системные вызовы порождения POSIX-процессов **fork(2)** и POSIX-нитей **pthread_create(3)** оказываются в Linux всего лишь «обертками» над **clone(2)**, что проиллюстрировано в листингах 4.16 и 4.17.

В примере из листинга 4.16 архиватор **tar(1) PID=11801** создает при помощи системного вызова **clone(2)** дочерний процесс **PID=11802**, в который помещает программу компрессора **gzip(1)**, используя системный вызов **execve(2)**. В результате параллель-

ной работы двух взаимодействующих процессов будет создан компрессированный архив **docs.tgz** каталога **/usr/share/doc**.

Листинг 4.16. Системный вызов clone — порождение нового процесса

```
fitz@ubuntu:~$ strace -fe clone,fork,execve tar czf docs.tgz /usr/share/doc
execve("/bin/tar", ["tar", "czf", "docs.tgz", "/usr/share/doc"], [/* 24 vars */]) = 0
clone(Process 11802 attached
child_stack=0, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0xb7df0728) = 11802
tar: Удаляется начальный `/' из имен объектов
          ...          ...          ...          ...          ...
[pid 11802] execve("/bin/gzip", ["gzip"], [/* 24 vars */]) = 0
                                ⓰ ⓱ ⓲ ⓳ ⓴ ㉑ ㉒ ㉓ ㉔ ㉕
Process 11801 suspended
Process 11801 resumed
Process 11802 detached
--- SIGCHLD (Child exited) @ 0 (0) ---
```

В примере из листинга 4.17 компрессор **pbzip2(1) PID=11857** создает при помощи системного вызова **clone(2)** семь «дочерних» нитей ❶...❼ **PID=11858→11862**, которые имеют общую память **CLONE_VM**, общие открытые файлы **CLONE_FILES** и прочие общие ресурсы.

Листинг 4.17. Системный вызов clone — порождение новой нити

```
fitz@ubuntu:~$ strace -fe clone,fork,execve pbzip2 plan9.iso
execve("/usr/bin/pbzip2", ["pbzip2", "plan9.iso"], [/* 24 vars */]) = 0
❶ clone(Process 11858 attached (waiting for parent)
Process 11858 resumed (parent 11857 ready)
child_stack=0xb7cb6424,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|
CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tidptr=0xb7cb6ba8, {entry_number:6,
base_addr:0xb7cb6b40, limit:1048575, seg_32bit:1, contents:0, read_exec_only:0,
limit_in_pages:1, seg_not_present:0, useable:1}, child_tidptr=0xb7cb6ba8) = 11858
❷ [pid 11857] clone(Process 11859 attached (waiting for parent)
Process 11859 resumed (parent 11857 ready)
child_stack=0xb74b5424,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLO
NE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tidptr=0xb74b5ba8, {entry_number:6,
base_addr:0xb74b5b40, limit:1048575, seg_32bit:1, contents:0, read_exec_only:0,
limit_in_pages:1, seg_not_present:0, useable:1}, child_tidptr=0xb74b5ba8) = 11859
4✕ ...          ...          ...          ...          ...
```

```
❼ [pid 11857] clone(Process 11864 attached (waiting for parent)
  Process 11864 resumed (parent 11857 ready)
  child_stack=0xb4cb0424,
  flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLO
  NE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tidptr=0xb4cb0ba8, {entry_number:6,
  base_addr:0xb4cb0b40, limit:1048575, seg_32bit:1, contents:0, read_exec_only:0,
  limit_in_pages:1, seg_not_present:0, useable:1}, child_tidptr=0xb4cb0ba8) = 11864
                                    ⓯ ⓰ ⓱ ⓲ ⓳ ⓴ ㉑ ㉒ ㉓ ㉔
  Process 11862 detached
  Process 11860 detached
  Process 11863 detached
  Process 11861 detached
  Process 11864 detached
  Process 11858 detached
  Process 11859 detached
```

4.4. Дерево процессов

Процессы, попарно связанные дочерне-родительскими отношениями, формируют дерево процессов операционной системы. Первый процесс **init(8)**, называемый *прародителем процессов* порождается ядром операционной системы после инициализации и монтирования корневой файловой системы, откуда и считывается программа **/sbin/init**. Прародитель процессов всегда имеет **PID=1**, а его основной задачей является запуск разнообразных системных служб, включая запуск обработчиков *(см. разд. 2.2.1)* алфавитно-цифрового терминального доступа **getty(8)**, менеджера дисплеев *(см. разд. 7.5.3)* графического доступа и службы дистанционного доступа **SSH** *(см. разд. 6.4.1)*. Кроме того, **init(8)** назначается приемным родителем для осиротевших процессов и может (в зависимости от реализации, см «альтернативы» **W:[init]**) отслеживать аварийные завершения запускаемых ими служб и перезапускать их.

В примере из листинга 4.18 показано дерево процессов, построенное при помощи специальной команды **pstree(1)**, а в листинге 4.19 — «классическое» представление дерева процессов при помощи команды **ps(1)**.

Листинг 4.18. Дерево процессов

```
fitz@ubuntu:~$ pstree -cnAh
init-+-upstart-udev-br
     |-udevd-+-udevd
     |       `-udevd
     |-sshd
```

```
|-rsyslogd-+-{rsyslogd}
|          |-{rsyslogd}
|          `-{rsyslogd}
   ...           ...              ...
|-cupsd
|-NetworkManager-+-{NetworkManager}
|                |-{NetworkManager}
|                |-dhclient
|                `-dnsmasq
   ...           ...              ...
|-getty
|-getty
|-login---bash---passwd ❸
|-getty
|-getty
|-cron
|-atd
|-acpid
|-lightdm-+-{lightdm}
|         |-Xorg
|         |-{lightdm}
|          -lightdm-+-{lightdm}
|                   -gnome-session-+-ssh-agent
|                                  |-{gnome-session}
|                                  |-{gnome-session}
|                                  |-{gnome-session}
   ...           ...               ...
|                                  |-compiz-+-{compiz}
|                                  |        |-{compiz}
|                                  |        |-{compiz}
   ...           ...               ...
|                                  |         -sh---gnome-terminal-+-{gnome-terminal}
|                                  |                              |-{gnome-terminal}
|                                  |                              |-gnome-pty-helpe
|                                  |                              |-bash---pstree
                        ...            ...              ...
```

Процессы операционной системы принято классифицировать на *системные* (ядерные), *демоны* и *прикладные*, исходя из их назначения и свойств (листинг 4.19).

Прикладные процессы ❸ выполняют обычные *пользовательские* программы (например, утилиту `man`), для чего им выделяют индивидуальную память, объем которой указан в столбце `VSZ` вывода команды `ps(1)`. Такие процессы обычно интерактивно взаимодействуют с пользователем посредством управляющего терминала (за исключением графических программ), указанного в столбце `TTY`.

Демоны (daemons) ❷ выполняют *системные* программы, реализующие те или иные службы операционной системы. Например, **cron(8)** реализует службу периодического выполнения заданий, **atd(8)** — службу отложенного выполнения заданий, **rsyslogd(8)** — службу централизованной журнализации событий, **sshd(8)** — службу дистанционного доступа, **udevd(8)** — службу «регистрации» подключаемых устройств, и т. д. Демоны запускаются на ранних стадиях загрузки операционной системы и взаимодействуют с пользователем не интерактивно при помощи терминала, а опосредованно — при помощи своих утилит. Таким образом, отсутствие управляющего терминала в столбце **TTY** отличает[1] их от прикладных процессов.

Листинг 4.19. Процессы ядра, демоны, прикладные процессы

```
fitz@ubuntu:~$ ps faxu
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          2  0.0  0.0      0     0 ?        S    Nov19   0:00 [kthreadd]
root          3  0.0  0.0      0     0 ?        S    Nov19   0:08  \_ [ksoftirqd/0]
root          5  0.0  0.0      0     0 ?        S<   Nov19   0:00  \_ [kworker/0:0H]
root          7  0.0  0.0      0     0 ?        S    Nov19   1:17  \_ [rcu_sched]
root          8  0.0  0.0      0     0 ?        S    Nov19   0:00  \_ [rcu_bh]
                                ...   ...                     ...
root          1  0.0  0.0   3652  1988 ?        Ss   Nov19   0:01 /sbin/init
                               ...   ...                     ...
root        394  0.0  0.0   3280  1536 ?        Ss   Nov19   0:00 /sbin/udevd --daemon
root        655  0.0  0.0   6680  2420 ?        Ss   Nov19   0:00 /usr/sbin/sshd -D
syslog      936  0.0  0.0  31064  1540 ?        Sl   Nov19   0:15 rsyslogd -c5
                              ...   ...                     ...
root       1344  0.0  0.0   2620   932 ?        Ss   Nov19   0:00 cron
daemon     1345  0.0  0.0   2472   348 ?        Ss   Nov19   0:00 atd
                              ...   ...                     ...
root       2395  0.0  0.0   3920  2000 tty1     Ss   Nov19   0:00 /bin/login --
fitz       9953  0.0  0.1  13516  8552 tty1     S    Nov20   0:00  \_ -bash
fitz      10058  0.0  0.0   6352  1712 tty1     S+   Nov20   0:00      \_ man ps
fitz      10069  0.0  0.0   5836   860 tty1     S+   Nov20   0:00          \_ pager -s
```

Системные (ядерные) ❶ процессы[2] выполняют параллельные части *ядра* операционной системы, поэтому не обладают ни индивидуальной виртуальной памятью **VSZ**, ни управляющим терминалом **TTY**. Более того, ядерные процессы не выполняют отдельную программу, загружаемую из **ELF**-файла, поэтому их имена **COMMAND** являются условными и изображаются в квадратных скобках.

---

[1] Зачастую демоны имеют суффикс **d** в конце названия, например **sshd** — это **s**ecure **sh**ell **d**aemon, а **rsyslogd** — **r**ocket **sys**tem **log**ging **d**aemon, и т. д.

[2] Правильнее — ядерные нити, т. к. выполняются они в общей памяти ядра операционной системы.

4.5. Атрибуты процесса

Процесс в операционной системе является основным активным субъектом, взаимодействующим с окружающими его объектами — файлами и файловыми системами, другими процессами, устройствами и пр. Возможности процесса выполнять те или иные действия по отношению к другим объектам определяются его специальными свойствами — атрибутами процесса.

4.5.1. Маркеры доступа

Возможности процесса по отношению к объектам, доступ к которым разграничивается при помощи дискреционных механизмов (в частности, к файлам дерева каталогов — *см. разд. 3.5*), определяются значениями его атрибутов, формирующих его DAC-маркер доступа, а именно — атрибутами **RUID**, **RGID**, **EUID**, **EGID**, см. **credentials(7)**.

Эффективные идентификаторы **EUID** (**e**ffective **u**ser **id**entifier) и **EGID** (**e**ffective **g**roup **id**entifier) указывают на «эффективных» пользователя и группу, использующихся дискреционными механизмами для определения прав доступа процесса к файлам и другим объектам согласно назначенному им режиму или списку доступа *(см. разд. 3.5.3)*. Атрибуты **RUID** (**r**eal **u**ser **id**entifier) и **RGID** (**r**eal **g**roup **id**entifier) указывают на «настоящих» пользователя и группу, «управляющих» процессом.

Первому процессу пользовательского сеанса (в случае регистрации в системе с использованием алфавитно-цифрового терминала — командному интерпретатору) назначают атрибуты **RUID/EUID** и **RGID/EGID** равными идентификаторам зарегистрировавшегося пользователя и его первичной группы. Последующие процессы пользовательского сеанса наследуют значения атрибутов, т. к. порождаются в результате клонирования при помощи **fork(2)**. В примере из листинга 4.20 при помощи команды **id(1)** показаны значения **EUID/EGID** пользовательского сеанса и их наследование от командного интерпретатора, что явным образом подтверждает команда **ps(1)**.

Листинг 4.20. DAC-маркер доступа процесса — атрибуты RUID, EUID, RGID, EGID

```
fitz@ubuntu:~$ id
uid=1006(fitz) gid=1008(fitz) группы=1008(fitz)
fitz@ubuntu:~$ ps fo euid,ruid,egid,rgid,user,group,tty,cmd
 EUID  RUID  EGID  RGID  USER  GROUP  TT       CMD
1006  1006  1008  1008  fitz  fitz   pts/2    -bash
1006  1006  1008  1008  fitz  fitz   pts/2     \_ ps fo euid,uid,egid,...,tty,cmd
```

Изменение идентификаторов **EUID/EGID** *процесса* происходит при срабатывании механизма неявной передачи полномочий, основанного на дополнительных атрибутах

SUID/SGID файлов *программ*. При запуске таких программ посредством системного вызова **exec(3)** атрибуты EUID/EGID запускающего процесса устанавливаются равными идентификаторам UID/GID владельца запускаемой программы. В результате процесс, в который будет загружена такая программа, будет обладать правами владельца программы, а не правами пользователя, запустившего эту программу.

В листинге 4.21 приведен типичный пример использования механизма неявной передачи полномочий при выполнении команд **passwd(1)** и **wall(1)**. При смене пароля пользователем при помощи программы **/usr/bin/passwd** ее процесс получает необходимое право записи ① в файл **/etc/shadow** (см. листинг 3.40) в результате передачи полномочий ❶ суперпользователя **root** (**UID=0**). При передаче широковещательного сообщения всем пользователям при помощи **/usr/bin/wall** необходимо иметь право записи ② в их файлы устройств **/dev/tty***, которое появляется ❷ в результате передачи полномочий группы **tty** (GID=5).

Листинг 4.21. Атрибуты файла SUID/SGID и атрибуты процесса RUID, EUID, RGID, EGID

```
   fitz@ubuntu:~$ who
   fitz     pts/0        2015-11-22 00:52 (:0.0)
   fitz     pts/1        2015-11-22 00:53 (:0.0)
   fitz     pts/2        2015-11-22 01:06 (:0.0)
   fitz@ubuntu:~$ ls -la /etc/shadow /dev/pts/*
   crw--w---- 1 fitz tty    136, 2 февр. 14 22:05 /dev/pts/1
   crw--w---- 1 fitz tty    136, 2 февр. 14 22:05 /dev/pts/1
   crw--w---- 1 fitz tty    136, 3 февр. 14 22:18 /dev/pts/2
②  crw--w---- 1 jake tty    136, 3 февр. 14 22:18 /dev/pts/2
   c--------- 1 root root     5, 2 февр.  8 14:21 /dev/pts/ptmx
①  -rw-r----- 1 root shadow  2595 февр. 14 22:18 /etc/shadow
   fitz@ubuntu:~$ ls -l /usr/bin/passwd /usr/bin/wall
   -rwsr-xr-x 1 ☛ root   root 41284 сент. 13  2012 /usr/bin/passwd
   -rwxr-sr-x 1    root ☛ tty 18036 марта 30  2012 /usr/bin/wall
   fitz@ubuntu:~$ ls -ln /usr/bin/passwd /usr/bin/wall
   -rwsr-xr-x 1 ☛ 0    0 41284 сент. 13  2012 /usr/bin/passwd
   -rwxr-sr-x 1    0 ☛ 5 18036 марта 30  2012 /usr/bin/wall
   fitz@ubuntu:~$ ps ft pts/1,pts/2 o pid,ruid,rgid,euid,egid,tty,cmd
     PID  RUID  RGID  EUID  EGID TT       CMD
   27883  1006  1008  1006  1008 pts/2    bash
❷  27937  1006  1008  1006  ☛ 5 pts/2     \_ wall
   27124  1006  1008  1006  1008 pts/1    bash
❶  27839  1006  1008  ☛ 0  1008 pts/1     \_ passwd
```

По отношению к объектам, доступ к которым ограничивается при помощи мандатных механизмов *(см. разд. 3.6)*, возможности процесса определяются значениями его MAC-маркера доступа, а именно — атрибутом мандатной метки **LABEL**. Как и **RUID/EUID/RGID/EGID**, атрибут **LABEL** назначается первому процессу сеанса пользователя явным образом, а затем наследуется при клонировании процессами-потомками от процессов-родителей. В примере из листинга 4.22 при помощи команды **id(1)** показан атрибут **LABEL** сеанса пользователя, а при помощи команды **ps(1)** — его явное наследование от процесса-родителя. Аналогично изменениям **EUID/EGID** процесса, происходящим при запуске **SUID**-ной/**SGID**-ной программы, изменение метки **LABEL** *процесса* происходит (согласно мандатным правилам ❶) в системном вызове **exec(3)** при запуске *программы*, помеченной соответствующей мандатной меткой файла. Так, например, при запуске программы **/usr/sbin/dhclient** с типом **dhcpc_exec_t** ее мандатной метки ❷ процесс приобретает тип **dhcpc_t** своей мандатной метки ❸, в результате чего существенно ограничивается в правах доступа к разным объектам операционной системы.

Листинг 4.22. MAC-маркер доступа процесса — мандатная метка selinux

```
fitz@ubuntu:~$ ssh lich@fedora
lich@fedora's password:
Last login: Sat Nov 21 14:25:16 2015
[lich@centos ~]$ id -Z
staff_u:staff_r:staff_t:s0-s0:c0.c1023
[lich@centos ~]$ ps Zf
LABEL                              PID TTY      STAT   TIME COMMAND
staff_u:staff_r:staff_t:s0-s0:c0.c1023 31396 pts/0 Ss   0:00 -bash
staff_u:staff_r:staff_t:s0-s0:c0.c1023 31835 pts/0 R+   0:00  \_ ps Zf
staff_u:staff_r:staff_t:s0-s0:c0.c1023 31334 tty2 Ss+   0:00 -bash
[lich@centos ~]$ sesearch -T -t dhcpc_exec_t -c process
Found 19 semantic te rules:
   ...          ...             ...              ...              ...
❶  type_transition NetworkManager_t dhcpc_exec_t : process dhcpc_t;
   ...          ...             ...              ...              ...
[lich@centos ~]$ ls -Z /usr/sbin/dhclient
❷ -rwxr-xr-x. root root system_u:object_r:dhcpc_exec_t:s0 /usr/sbin/dhclient
[lich@centos ~]$ ps -ZC dhclient
LABEL                           PID TTY          TIME CMD
❸ system_u:system_r:dhcpc_t:s0  2120 ?        00:00:00 dhclient
system_u:system_r:dhcpc_t:s0    4320 ?        00:00:00 dhclient
```

4.5.2. Привилегии

Еще одним важным атрибутом процесса, определяющим его возможности по использованию системных вызовов, являются привилегии процесса cababilities(7). Например, обладание привилегией CAP_SYS_PTRACE разрешает процессам трассировщиков strace(1) и ltrace(1), использующих системный вызов ptrace(2), трассировать процессы *любых* пользователей (а не только «свои», EUID которых совпадает с EUID трассировщика). Аналогично, привилегия CAP_SYS_NICE разрешает изменять приоритет, устанавливать привязку к процессорам и назначать алгоритмы планирования *(см. разд. 4.6)* процессов и нитей *любых* пользователей, а привилегия CAP_KILL разрешает посылать сигналы *(см. разд. 4.8)* процессам *любых* пользователей.

Явная привилегия «владельца» CAP_FOWNER позволяет процессам изменять режим и списки доступа *(см. разд. 3.5.1)*, мандатную метку *(см. разд. 3.6.2)*, расширенные атрибуты *(см. разд. 3.7.1)* и флаги *(см. разд. 3.7.2)* любых файлов так, словно процесс выполняется от лица владельца файла. Привилегия CAP_LINUX_IMMUTABLE разрешает управлять флагами файлов **i**, **i**mmutable и **a**, **a**ppend *(см. разд. 3.7.2)*, а привилегия CAP_SETFCAP — устанавливать «файловые» привилегии *(см. далее)* запускаемых программ.

Необходимо отметить, что именно обладание полным набором привилегий делает пользователя **root** (**UID=0**) в Linux — суперпользователем. И наоборот, обычный, непривилегированный пользователь (в смысле **UID≠0**) не обладает никакими явными[1] привилегиями. Назначение[2] привилегий процесса происходит при запуске программы при помощи системного вызова exec(3), исполняемый файл которого помечен «файловыми» привилегиями.

В примере из листинга 4.23 иллюстрируется получение списка привилегий процесса при помощи утилиты getpcaps. Как и ожидалось, процесс **postgres** (**PID=1427**), работающий от лица обычного (непривилегированного, в смысле **UID≠0**) псевдопользователя **postgres**, не имеет ❶ никаких привилегий, а процесс **NetworkManager** (**PID=1263**), работающий от лица суперпользователя **root** (**UID=0**), имеет полный ❷ набор привилегий. Однако процесс **dhclient** (**PID=14026**) выполняется от лица «суперпользователя», лишенного ❸ большинства своих привилегий, т. к. после порождения своего дочернего процесса **NetworkManager** уменьшил его привилегии до минимально необходимого набора, достаточного для выполнения своих функций[3].

---

[1] Неявно он обладает привилегией владельца для всех своих объектов.

[2] Здесь допущено намеренное упрощение механизма наследования и назначения привилегий при fork(2) и exec(3) без потери смысла.

[3] Что способствует обеспечению защищенности операционной системы.

Листинг 4.23. Привилегии (capabilities) процесса

```
fitz@ubuntu:~$ ps fo user,pid,cmd -C NetworkManager,dhclient,postgres
USER       PID CMD
root      1263 NetworkManager
root     14026  \_ /sbin/dhclient -d -4 -sf /usr/lib/NetworkManager/nm-dhcp-client...
postgres  1427 /usr/lib/postgresql/9.1/bin/postgres -D /var/lib/postgresql/9.1/main ...
postgres  1984  \_ postgres: writer process
postgres  1985  \_ postgres: wal writer process
postgres  1986  \_ postgres: autovacuum launcher process
postgres  1987  \_ postgres: stats collector process
fitz@ubuntu:~$ getpcaps 1427
❶ Capabilities for `1427': =
❷ fitz@ubuntu:~$ getpcaps 1263
Capabilities for `1263': =
cap_chown,cap_dac_override,cap_dac_read_search,cap_fowner,cap_fsetid,cap_kill,cap_setgid,
cap_setuid,cap_setpcap,cap_linux_immutable,cap_net_bind_service,cap_net_broadcast,
cap_net_admin,cap_net_raw,cap_ipc_lock,cap_ipc_owner,cap_sys_module,cap_sys_rawio,
cap_sys_chroot,cap_sys_ptrace,cap_sys_pacct,cap_sys_admin,cap_sys_boot,cap_sys_nice,
cap_sys_resource,cap_sys_time,cap_sys_tty_config,cap_mknod,cap_lease,cap_audit_write,
cap_audit_control,cap_setfcap,cap_mac_override,cap_mac_admin,cap_syslog,35,36+ep
❸ fitz@ubuntu:~$ getpcaps 14026
Capabilities for `14026': =
cap_dac_override,cap_net_bind_service,cap_net_raw,cap_sys_module+ep
```

В листинге 4.24 показан типичный пример использования *отдельных* привилегий там, где классически применяется неявная передача *всех* полномочий суперпользователя при помощи механизма SUID/SGID. Например, «обычная» утилита **ping(1)** для выполнения своей работы должна создать «необработанный» **raw(7)** сетевой сокет, что является с точки зрения ядра привилегированной операцией. В большинстве систем программа **/bin/ping** будет SUID-ной ❶ во владении суперпользователем **root**, чьи права и будут передаваться при ее запуске. С точки зрения защищенности системы это не соответствует здравому смыслу, подсказывающему наделять программы минимально необходимыми возможностями, достаточными для их функционирования. Для создания «необработанных» **raw(7)** и пакетных **packet(7)** сокетов достаточно только привилегии **CAP_NET_RAW**, а весь суперпользовательский набор привилегий более чем избыточен.

Листинг 4.24. Делегирование привилегий программе ping(1)

```
fitz@ubuntu:~$ ls -l /bin/ping
❶ -rwsr-xr-x 1 root root 34740 нояб.  8  2011 /bin/ping
fitz@ubuntu:~$ ping ubuntu
```

```
PING ubuntu (127.0.1.1) 56(84) bytes of data.
64 bytes from ubuntu (127.0.1.1): icmp_req=1 ttl=64 time=0.074 ms
^C
--- ubuntu ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.074/0.074/0.074/0.000 ms
❷ fitz@ubuntu:~$ sudo chmod u-s /bin/ping
fitz@ubuntu:~$ ping ubuntu
🔒 ping: icmp open socket: Operation not permitted
❸ fitz@ubuntu:~$ sudo setcap cap_net_raw+ep /bin/ping
fitz@ubuntu:~$ ls -l /bin/ping
-rwxr-xr-x 1 root root 34740 нояб.  8  2011 /bin/ping
fitz@ubuntu:~$ getcap /bin/ping
/bin/ping = cap_net_raw+ep
fitz@ubuntu:~$ ping ubuntu
PING ubuntu (127.0.1.1) 56(84) bytes of data.
🔓 64 bytes from ubuntu (127.0.1.1): icmp_req=1 ttl=64 time=0.142 ms
^C
--- ubuntu ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.142/0.142/0.142/0.000 ms
```

При отключении передачи полномочий ❷ программа **/bin/ping** лишается возможности выполнять свои функции, а при назначении ей при помощи команды **setcap(8)** «файловой» привилегии **CAP_NET_RAW** ❸ функциональность возвращается в полном объеме, т. к. приводит к установке «процессной» привилегии **CAP_NET_RAW** при запуске этой программы. Для просмотра привилегий, делегируемых при запуске программ, используется парная команда **getcap(8)**.

Аналогично, при использовании анализаторов сетевого трафика **tshark(1)** (листинг 4.25) и/или **wireshark(1)**, вызывающих для захвата сетевых пакетов утилиту **dumpcap(1)**, необходимо открывать как «необработанные» **raw(7)**, так и пакетные **packet(7)** сетевые сокеты, что требует той же привилегии **CAP_NET_RAW**. Классический способ применения анализаторов пакетов состоит в использовании явной передачи *всех* полномочий суперпользователя (при помощи **su(1)** или **sudo(1)**) при их запуске, что опять не соответствует минимально необходимым и достаточным требованиям к разрешенным возможностям программ.

Листинг 4.25. Делегирование привилегий программе tshark(1)

```
fitz@ubuntu:~$ tshark
tshark: There are no interfaces on which a capture can be done
```

```
itz@ubuntu:~$ strace -fe execve tshark
execve("/usr/bin/tshark", ["tshark"], [/* 23 vars */]) = 0
Process 8951 attached
[pid  8951] execve("/usr/bin/dumpcap", ["/usr/bin/dumpcap", "-D", "-Z", "none"],
[/* 23 vars */]) = 0
Process 8951 detached
--- SIGCHLD (Child exited) @ 0 (0) ---
tshark: There are no interfaces on which a capture can be done
fitz@ubuntu:~$ sudo setcap cap_net_raw+ep /usr/bin/dumpcap
[sudo] password for fitz:
fitz@ubuntu:~$ getcap /usr/bin/dumpcap
/usr/bin/dumpcap = cap_net_raw+ep
fitz@ubuntu:~$ tshark -i wlan0
Capturing on wlan0
  0.307205 fe80::895d:9d7d:f0b3:a372 -> ff02::1:ff96:2df6 ICMPv6 86 Neighbor Solicitation
  0.307460 SuperMic_74:0e:90 -> Spanning-tree-(for-bridges)_00 STP 60 Conf. Root =
32768/0/00:25:90:74:0e:90  Cost = 0  Port = 0x8001
            ...          ...          ...          ...          ...
  0.982542   10.2.0.126 -> 10.2.0.255   NBNS 92 Name query NB WPAD<00>
  0.982811 04:18:d6:4e:53:88 -> Broadcast    ARP 60 Who has 10.2.0.1?  Tell 10.2.0.43.
```

Для эффективного использования анализаторов трафика непривилегированными пользователями достаточно делегировать их процессам захвата пакетов привилегию **CAP_NET_RAW** при помощи «файловых» привилегии **CAP_NET_RAW** для программы захвата **/usr/bin/dumpcap**, что и проиллюстрировано в листинге 4.25.

4.5.3. Другие атрибуты

Переменные окружения (листинг 4.26) и текущий рабочий каталог (листинг 4.27) на поверку тоже оказываются атрибутами процесса, которые можно получить при помощи команд **ps(1)** и **pwdx(1)** соответственно.

Листинг 4.26. Переменные окружения процесса

```
fitz@ubuntu:~$ ps fe
  PID TTY      STAT   TIME COMMAND
21872 pts/2    S      0:00 -bash TERM=xterm LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:...
22904 pts/2    R+     0:00  \_ ps fe SHELL=/bin/bash TERM=xterm USER=fitz ...
```

Листинг 4.27. Текущий рабочий каталог процесса

```
fitz@ubuntu:~$ ps fx
  PID TTY      STAT   TIME COMMAND
```

```
22984 pts/0    S      0:00 -bash
23086 pts/0    S+     0:00  \_ man ps
23097 pts/0    S+     0:00      \_ pager -s
21872 pts/2    S      0:00 -bash
23103 pts/2    R+     0:00  \_ ps fx
fitz@ubuntu:~$ pwdx 23097 22984
23097: /usr/share/man
22984: /home/fitz
```

4.6. Классы и приоритеты процессов

Переключение центрального процессора между задачами (процессами и нитями) выполняет специальная компонента подсистемы управления процессами, называемая *планировщиком* (scheduler). Именно планировщик определенным образом выбирает из множества неспящих, готовых к выполнению (runable) задач одну, которую переводит в состояние выполнения (running). Процедуры, определяющие способ выбора и моменты выполнения выбора, называются *алгоритмами планирования*. Выбор задачи, подлежащей выполнению, естественным образом происходит в моменты времени, когда текущая выполнявшаяся задача переходит в состояние сна (sleep) в результате выполнения операции ввода-вывода. *Вытесняющие* алгоритмы планирования, кроме всего прочего, ограничивают непрерывное время выполнения задачи, принудительно прерывая ее выполнение по исчерпанию выданного ей кванта времени (timeslice) и *вытесняя* ее во множество готовых, после чего производят выбор новой задачи, подлежащей выполнению.

По умолчанию для пользовательских задач используется вытесняющий алгоритм CFS (completely fair scheduler), согласно которому процессорное время распределяется между неспящими задачами справедливым (fair) образом. Для каждой задачи определяется выделяемая справедливая (в соответствии с ее относительным «приоритетом») доля процессорного времени, которую она должна получить при конкуренции за процессор. Для двух задач с любыми *одинаковыми* приоритетами должны быть выделены *равные* доли (в 50% процессорного времени), а при *различии* в приоритетах на *одну* ступень разница между выделяемыми долями должна составить ≈10% процессорного времени (т. е. 55 и 45% соответственно). Для удовлетворения этого требования алгоритм планирования CFS назначает каждой ступени приоритета соответствующий[1] «вес» задачи, а процессорное время делит между всеми неспящими задачами пропорционально их весам. Таким образом, две задачи

[1] Шкала весов учитывает только требование 10% разницы между *двумя* задачами с различием в относительных приоритетах на *одну* ступень.

с любыми одинаковыми приоритетами будут иметь равные веса $w_i = w_j = w$, а доли процессорного времени составят $\mu_i = w_i/(w_i + w_j) = 1/2$ и $\mu_j = w_j/(w_i + w_j) = 1/2$. Для двух задач с приоритетами, отличающимися на одну ступень $w_i \neq w_j$, а $\mu_i - \mu_j = 1/10$, откуда несложно получить, что $w_i/w_j = 11/9$ — правило построения шкалы[1] весов, а $\mu_1 = 11/20 = 0{,}55$ и $\mu_2 = 9/20 = 0{,}45$, что и требовалось получить.

Для дифференциации задач используют 40 относительных POSIX-приоритетов на шкале от −20 до +19, называемых «любезностью» задачи **NICE**. Относительный приоритет буквально определяет, насколько «любезна» будет задача по отношению к остальным готовым к выполнению задачам при конкуренции за процессорное время освободившегося процессора. Наименее «любезным», с относительным приоритетом −20 (наивысшим) планировщик выделит большую долю процессорного времени, а наиболее «любезным», с приоритетом +19 (наинизшим) — меньшую. При отсутствии конкуренции, когда количество готовых к выполнению задач равно количеству свободных процессоров, приоритет не будет играть никакой роли.

В примере из листинга 4.28 при помощи команды **bzip(2)** запущены два процесса сжатия ISO-образа с наилучшим качеством одновременно друг другу на «заднем фоне». В выводе свойств **pcpu** (**p**ercent **cpu**, процент потребляемого процессорного времени), **pri** (**pri**ority), **ni** (**ni**ce) и **psr** (**p**rocesso**r** number) их процессов при помощи **ps(1)** оказывается, что они потребляют практически одинаковые ❶ доли (проценты) процессорного времени, и это для одинаковых программ вполне соответствует интуитивным ожиданиям. После повышения любезности (понижения относительного приоритета) одного из них ❷ при помощи команды **renice(1)** до значения +10 отношение потребляемых долей процессорного времени не изменилось ❸, что означает отсутствие конкуренции за процессор, подтверждаемое как столбцом **PSR**, показывающим номер процессора, выполняющего программу, так и командами **nproc(1)** и **lscpu(1)**.

Листинг 4.28. Относительный приоритет NICE

```
fitz@ubuntu:~$ bzip2 --best -kf plan9.iso &
[1] 12944
fitz@ubuntu:~$ bzip2 --best -kf plan9.iso &
[2] 12945
fitz@ubuntu:~$ ps fo pid,pcpu,pri,ni,psr,cmd
  PID %CPU PRI  NI PSR CMD
12808  0.1  19   0   0 -bash
```

[1] Только $w_i/w_j = 11/9 = 1{,}2(2)$, тогда как в ядре Linux взято $w_i/w_j = 1{,}25$.

```
❶ 12944 94.5  19   0   2  \_ bzip2 --best -kf plan9.iso
  12945 96.0  19   0   1  \_ bzip2 --best -kf plan9.iso
  12946  0.0  19   0   3  \_ ps fo pid,pcpu,pri,ni,psr,cmd
❷ fitz@ubuntu:~$ renice +10 12945
  12945 (process ID) old priority 0, new priority 10
  fitz@ubuntu:~$ ps fo pid,pcpu,pri,ni,psr,cmd
    PID %CPU PRI  NI PSR CMD
  12808  0.1  19   0   0 -bash
❸ 12944 94.8  19   0 ☛2  \_ bzip2 --best -kf plan9.iso
? 12945 97.0   9  10 ☛0  \_ bzip2 --best -kf plan9.iso
  12948  0.0  19   0   1  \_ ps fo pid,pcpu,pri,ni,psr,cmd
  fitz@ubuntu:~$ nproc
  4
  fitz@ubuntu:~$ lscpu
  Architecture:          i686
  CPU op-mode(s):        32-bit, 64-bit
  Byte Order:            Little Endian
  CPU(s):                4 ☚
  On-line CPU(s) list:   0-3
                        ...          ...          ...          ...
❹ fitz@ubuntu:~$ taskset -p -c 3 12808
  pid 12808's current affinity list: 0-3
  pid 12808's new affinity list: 3
❺ fitz@ubuntu:~$ nice -n 5 time bzip2 --best -kf  plan9.iso &
  [1] 29331
❺ fitz@ubuntu:~$ nice -n 15 time bzip2 --best -kf  plan9.iso &
  [2] 29333
  fitz@ubuntu:~$ ps fo pid,pcpu,pri,ni,psr,cmd
    PID %CPU PRI   NI PSR CMD
  28573  0.0  19    0   3 -bash
  29331  0.0   9    5   3  \_ time bzip2 --best -kf plan9.iso
❻ 29332 91.4☚ 9 ☛  5   3  |   \_ bzip2 --best -kf plan9.iso
  29333  0.0   4   15   3  \_ time bzip2 --best -kf plan9.iso
! 29334  9.7☚ 4 ☛ 15   3  |   \_ bzip2 --best -kf plan9.iso
  29336  0.0  19    0   3  \_ ps fo pid,pcpu,pri,ni,psr,cmd
  fitz@ubuntu:~$ wait
  51.10user 0.22system 0:56.86elapsed 90%CPU (0avgtext+0avgdata 31248maxresiden
  0inputs+180560outputs (0major+1004minor)pagefaults 0swaps
  [1]-  Готово          nice -n 5 time bzip2 --best -kf plan9.iso
  53.79user 0.20system 1:43.08elapsed 52%CPU (0avgtext+0avgdata 31520maxresident)k
  0inputs+180560outputs (0major+1515minor)pagefaults 0swaps
  [2]+  Готово          nice -n 15 time bzip2 --best -kf plan9.iso
```

Проиллюстрировать действие относительного приоритета NICE на многопроцессорной системе можно, создав искусственную конкуренцию двух процессов за один процессор. Для этого при помощи команды taskset(1) устанавливается привязка (affinity) ❹ командного интерпретатора (PID=12808) к процессору 3 (привязка, как и прочие свойства и атрибуты процесса, наследуется потомками). Затем при помощи команды nice(1) запускаются ❺ две программы упаковки с относительными приоритетами 5 и 15 в режиме измерения потребления времени при помощи команды time(1). В результате доли процессорного времени распределяются неравномерно, причем их разница зависит от *разницы* в относительных приоритетах (и от свойств конкурирующих процессов, но в примере они одинаковые). Дождавшись завершения процессов заднего фона, при помощи встроенной команды wait можно оценить разницу в реальном времени выполнения упаковки, вызванную неравным распределением процессора между процессами упаковщиков.

Кроме приоритетной очереди, планировщик Linux позволяет использовать еще два алгоритма планирования — FIFO и RR, предназначенные для задач реального времени. Вытесняющий алгоритм RR (round robin) организует простейшее *циклическое* обслуживание с фиксированными квантами времени, тогда как FIFO (first in first out) является его невытесняющей модификацией, позволяя задаче выполняться непрерывно долго, до момента ее засыпания. По сути, оба алгоритма организуют задачи в одну приоритетную очередь (PQ, priority queue) со статическими приоритетами на шкале от 1 до 99, выбирая для выполнения всегда самую высокоприоритетную из множества готовых.

Перевод задачи под управление тем или алгоритмом планирования производится при помощи назначения ей политики планирования (scheduling policy) посредством команды chrt(1). Различают пять политик планирования, три из которых — SCHED_OTHER, SCHED_BATCH и SCHED_IDLE — реализуются алгоритмом CFS, а две оставшиеся — SCHED_FIFO, SCHED_RR — одноименными алгоритмами FIFO и RR. Политика SCHED_OTHER, она же SCHED_NORMAL, применяется по умолчанию и обслуживает класс задач TS (time sharing), требующих «интерактивности», а политика SCHED_BATCH предназначается для выполнения «вычислительных» задач класса пакетной B (batch) обработки. Разница между политиками состоит в том, что планировщик в случае необходимости всегда прерывает задачи класса B в пользу задач класса TS, но никогда наоборот. Политика SCHED_IDLE формирует класс задач, выполняющихся только при «простое» (idle) центрального процессора за счет выделения им планировщиком CFS очень небольшой доли процессорного времени. Для этого задачам IDL-класса назначают минимально возможный вес — меньший, чем вес самых «любезных» (NICE = +19) задач TS-класса.

В примере из листинга 4.29 проиллюстрировано распределение процессорного времени алгоритмом планирования CFS между (конкурирующими за один процессор)

одинаковыми процессами TS-, B- и IDL-классов, назначенных им при запуске упаковщиков bzip2(1) посредством команды chrt(1).

Листинг 4.29. Классы процессов

```
fitz@ubuntu:~$ ps f
  PID TTY      STAT   TIME COMMAND
12058 pts/0    Ss     0:00 bash
12065 pts/0    R+     0:00  \_ ps f
fitz@ubuntu:~$ taskset -p -c 3 12058
pid 12058's current affinity list: 0-3
pid 12058's new affinity list: 3
fitz@ubuntu:~$ chrt -b 0 time bzip2 --best -kf plan9.iso &
[1] 12410
fitz@ubuntu:~$ chrt -o 0 time bzip2 --best -kf plan9.iso &
[2] 12412
fitz@ubuntu:~$ chrt -i 0 time bzip2 --best -kf plan9.iso &
[3] 12414
fitz@ubuntu:~$ ps fo pid,pcpu,class,pri,ni,psr,cmd
  PID %CPU CLS   PRI  NI PSR CMD
12058  0.1 TS     19   0   3 -bash
12410  0.0 B      19   -   3  \_ time bzip2 --best -kf plan9.iso
12411 50.0 B      19   -   3  |   \_ bzip2 --best -kf plan9.iso
12412  0.0 TS     19   0   3  \_ time bzip2 --best -kf plan9.iso
12413 49.7 TS     19   0   3  |   \_ bzip2 --best -kf plan9.iso
12414  0.0 IDL    19   -   3  \_ time bzip2 --best -kf plan9.iso
12415  0.1 IDL    19   -   3  |   \_ bzip2 --best -kf plan9.iso
12471  0.0 TS     19   0   3  \_ ps fo pid,pcpu,class,pri,ni,psr,cmd
fitz@ubuntu:~$ wait
53.85user 0.26system 1:45.98elapsed 51%CPU (0avgtext+0avgdata 31248maxresident)k
0inputs+180560outputs (0major+1004minor)pagefaults 0swaps
53.96user 0.22system 1:46.04elapsed 51%CPU (0avgtext+0avgdata 31248maxresident)k
0inputs+180560outputs (0major+1004minor)pagefaults 0swaps
52.74user 0.27system 2:41.54elapsed 32%CPU (0avgtext+0avgdata 31248maxresident)k
0inputs+180560outputs (0major+1004minor)pagefaults 0swaps
[1]   Готово          chrt -b 0 time bzip2 --best -kf plan9.iso
[2]-  Готово          chrt -o 0 time bzip2 --best -kf plan9.iso
[3]+  Готово          chrt -i 0 time bzip2 --best -kf plan9.iso
```

В листинге 4.30 показана конкуренция процессов под управлением RR-планировщика, использующего статические приоритеты. Так как операция назначения

политик планирования «реального времени» FIFO и RR является привилегированной, то сначала ❶ командный интерпретатор переводится в RR-класс (**-r**) с наивысшим статическим приоритетом **99** при помощи команды chrt(1), выполняемой от лица суперпользователя **root**. При последующих запусках упаковщиков класс будет унаследован и не потребует повышенных привилегий. Два процесса упаковщиков запускаются ❷ командой chrt(1) с одинаковыми статическими приоритетами **1**, привязанные к одному процессору командой taskset(1), в результате чего получают равные доли процессорного времени, что вполне соответствует интуитивным ожиданиям от вытесняющего циклического планировщика RR. Нужно отметить, что шкала статических приоритетов 1→99 классов RR и FIFO, как и шкала «любезности» NICE +19 →–20 классов TS и B, отображаются на общую шкалу приоритетов PRI так, что верхняя часть шкалы PRI: 41→139 соответствует статическим приоритетам, а нижняя часть шкалы PRI: 0→39 соответствует «любезности».

Листинг 4.30. Классы процессов реального времени

```
   fitz@ubuntu:~$ ps f
     PID TTY      STAT   TIME COMMAND
☛  15313 pts/0    S      0:00 -bash
   15520 pts/0    R+     0:00  \_ ps f
❶  fitz@ubuntu:~$ sudo chrt -pr 99 15313
   fitz@ubuntu:~$ ps fo pid,psr,cls,ni,pri,pcpu,comm
     PID PSR CLS  NI PRI   %CPU COMMAND
   15313   0  RR   - 139 ☚ 0.1 bash
   15550   1  RR   - 139   0.0  \_ ps
❷  fitz@ubuntu:~$ chrt -r 1 taskset -c 2 bzip2 --best -kf plan9.iso &
   [1] 15572
❷  fitz@ubuntu:~$ chrt -r 1 taskset -c 2 bzip2 --best -kf plan9.iso &
   [2] 15573
   fitz@ubuntu:~$ ps fo pid,psr,cls,ni,pri,pcpu,comm
     PID PSR   CLS  NI   PRI %CPU   COMMAND
   15313   0    RR   -   139  0.1   bash
   15572   2 ☚ RR   - ☛  41 51.8 ☚ \_ bzip2
   15573   2 ☚ RR   - ☛  41 48.8 ☚ \_ bzip2
   15597   1    RR   -   139  0.0    \_ ps
❸  fitz@ubuntu:~$ chrt -r 2 taskset -c 2 bzip2 --best -kf plan9.iso &
   [3] 15628
   fitz@ubuntu:~$ ps fo pid,psr,cls,ni,pri,pcpu,comm
     PID PSR CLS  NI   PRI %CPU    COMMAND
   15313   0  RR   -   139  0.1    bash
   15572   2  RR   -    41 48.3 ☚? \_ bzip2
```

```
★ 15573   2  RR   -    41 47.5 ☚? \_ bzip2
  15628   2  RR   - ☚ 42 93.3 ☚! \_ bzip2
  15630   1  RR   -   139  0.0     \_ ps
  fitz@ubuntu:~$ top -b -n1 -p 15572,15573,15628
  top - 14:44:01 up  4:27,  2 users,  load average: 5.07, 3.42, 2.50
  Tasks:   3 total,   3 running,   0 sleeping,   0 stopped,   0 zombie
  Cpu(s): 18.5%us,  3.4%sy,  0.6%ni, 76.0%id,  1.4%wa,  0.0%hi,  0.1%si,  0.0%st
  Mem:   8192144k total,  3917972k used,  4274172k free,   151936k buffers
  Swap:  4104188k total,        0k used,  4104188k free,  2713336k cached

    PID USER      PR  NI  VIRT  RES  SHR S  %CPU %MEM    TIME+  COMMAND
  15628 fitz   ☚  -3   0  9536 7880  468 R ☚ 100  0.1   1:14.96 bzip2
  15572 fitz   ☚  -2   0  9536 7880  460 R ☚   0  0.1   1:30.95 bzip2
  15573 fitz   ☚  -2   0  9536 7884  464 R ☚   0  0.1   1:30.01 bzip2
```

Добавление ❸ третьего процесса упаковщика со статическим приоритетом **2** приводит к резкому перекосу выделяемой доли процессорного времени в его пользу. Это объясняется тем, что алгоритм планирования **RR** (равно как и **FIFO**) всегда выбирает процесс с самым высоким статическим приоритетом из множества готовых, поэтому процессам с более низкими приоритетами процессорное время будет выделено только при засыпании[1] всех процессов с большими приоритетами.

Странный результат ★, изображаемый командой **ps(1)**, объясняется «несовершенством» ее способа расчета доли процессорного времени **%CPU**, выделяемой процессу. Расчет производится как отношение чистого потребленного процессорного времени (за все время существования процесса) к промежутку реального времени, прошедшему с момента порождения процесса, что соответствует *среднему*, но не *мгновенному* значению потребляемой доли. Гораздо более ожидаемый результат получает команда **top(1)**, выполняющая расчет *мгновенной* доли процессорного времени как отношение чистого потребленного процессорного времени (за небольшой промежуток наблюдения) к реальному времени наблюдения.

4.7. Память процесса

Еще одним ресурсом, подлежащим распределению между процессами, является оперативная память. В Linux, как и во многих других современных операционных системах, для управления памятью используют механизм *страничного отображения*, реализуемого ядром операционной системы при помощи устройства управле-

[1] А упаковщик, хоть и выполняет операции ввода-вывода, в реальности практически не засыпает в связи с использованием дискового кэша.

ния памятью — **W:[MMU]**. При этом процессы работают с *виртуальными* адресами (virtual address) «воображаемой» памяти, отображаемыми устройством MMU на физические адреса (physical address) настоящей оперативной памяти. Для отображения (рис. 4.3) вся оперативная память (RAM) условно разбивается на «гранулы» — *страничные кадры* размером в 4 Кбайт, которые затем выделяются процессам. Таким образом, память процесса условно состоит из *страниц* (page), которым в специальных *таблицах страниц* (page table) сопоставлены выделенные страничные кадры (page frame). При выполнении процесса преобразование его виртуальных адресов в физические выполняется устройством MMU «на лету» при помощи его индивидуальной таблицы страниц.

Именно механизм страничного отображения позволяет эффективно распределять память между процессами путем размещения страниц процессов в *произвольные*

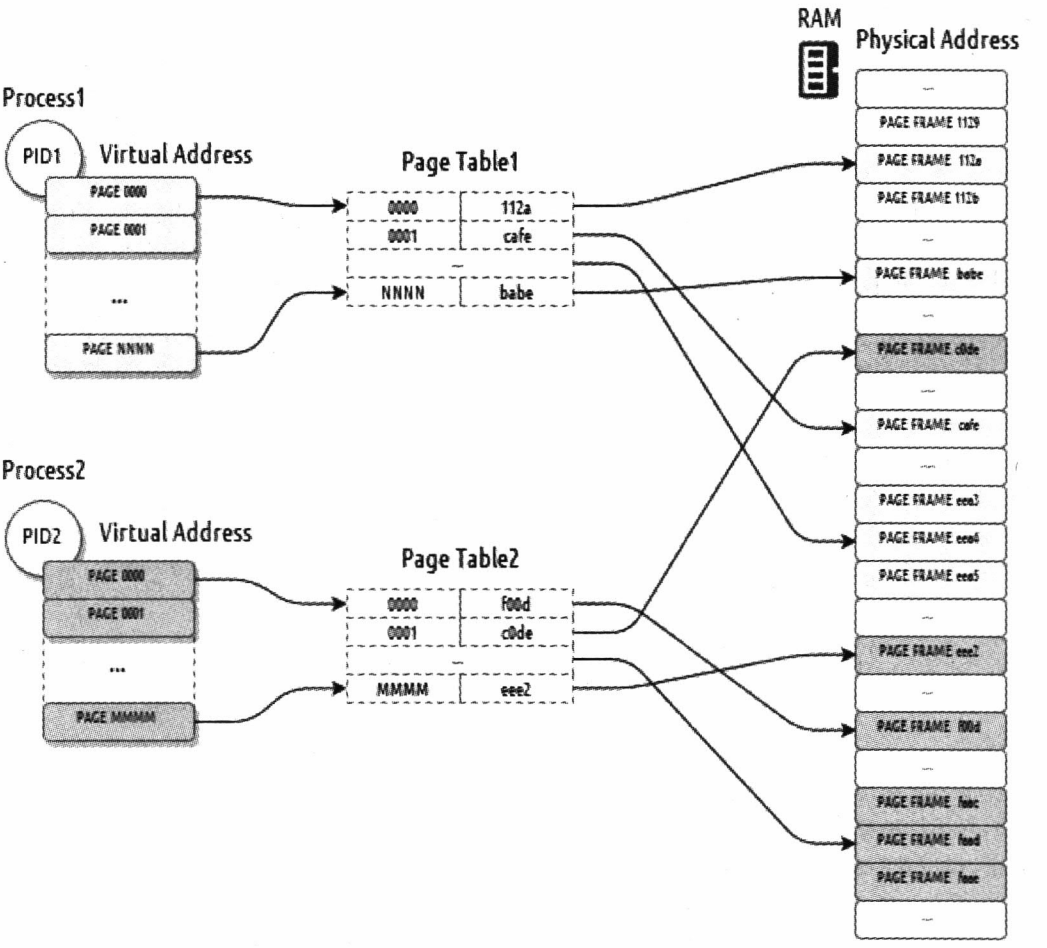


Рис. 4.3. Страничное отображение и распределение памяти

свободные страничные кадры. Кроме этого, механизм страничного отображения позволяет выделять процессам память *по требованию*, добавляя дополнительные страницы и отображая их на свободные страничные кадры. Аналогично, ненужные процессу страницы могут быть удалены, а соответствующие страничные кадры высвобождены для использования другими процессами.

4.7.1. Виртуальная память

Помимо задачи распределения памяти между процессами, механизм страничного отображения используется ядром операционной системы и для решения задачи нехватки оперативной памяти. При определенных обстоятельствах имеющиеся в распоряжении свободные страничные кадры оперативной памяти могут быть исчерпаны. Одновременно с этим оказывается, что большую часть времени процессы используют лишь малую часть выделенной им памяти, а находясь в состоянии сна, не используют память вовсе.

Увеличить коэффициент полезного использования памяти позволяет еще одна простая идея[1] (рис. 4.4) — высвобождать ❶ страничные кадры при помощи выгруз-

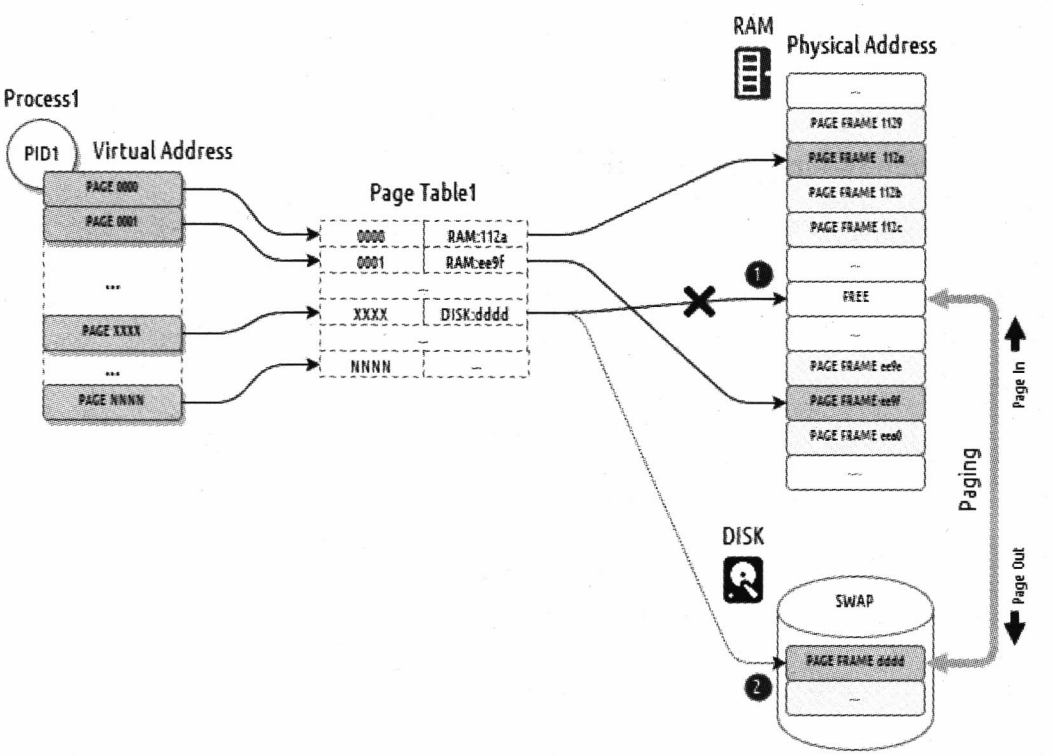


Рис. 4.4. Страничный обмен

[1] Подобная идее многозадачности *(см. разд. 4.2)*.

ки ❷ (page out) неиспользуемых страниц процессов во вторичную память (в специальную область «подкачки» SWAP, например, на диске), а при обращении к выгруженной странице — загружать (page in) ее обратно перед использованием. За счет такого *страничного обмена*[1] (paging или page swapping) организуется W:[виртуальная память], т. е. видимость бóльшего количества (оперативной) памяти для размещения процессов, чем есть на самом деле.

В примере из листинга 4.31 в столбцах VSZ и RSS вывода команды ps(1) показано потребление памяти процессами (в килобайтах). В столбце VSZ (virtual size) указывается суммарный объем всех страниц процесса (в том числе и выгруженных), а в столбце RSS (resident set size) — суммарный объем всех его страничных кадров в оперативной памяти, т. е. ее *реальное* потребление процессом.

Листинг 4.31. Виртуальная и резидентная память процесса

```
fitz@ubuntu:~$ ps fu
USER       PID %CPU %MEM    VSZ    RSS TTY      STAT START   TIME COMMAND
fitz     18690  0.0  0.0  10308   5432 pts/1    Ss   20:51   0:00 bash
fitz     19393  2.8  1.7 609744 143316 pts/1    Sl   21:27   0:11  \_ /usr/.../firefox
fitz     19526  0.0  0.0   6104    700 pts/1    R+   21:33   0:00  \_ ps fu
```

4.7.2. Отображение файлов в память

Страничный обмен, помимо организации виртуальной памяти, имеет еще одно важнейшее применение. Именно на его основе реализуется незаменимый механизм отображения файлов в память процесса, доступный при помощи системных вызовов mmap(2)/munmap(2) (и дополнительных mlock(2), mprotect(2), msync(2), madvise(2) и др.).

Для отображения файла (рис. 4.5) в память процесса ему выделяют страницы ❶ в необходимом количестве, но не страничные кадры. Вместо этого, в таблице страниц формируются такие записи ❶, как будто эти страницы уже были выгружены ранее (!) в отображаемый файл ❷. При последующем обращении (on demand) процесса к какой-либо странице отображенной памяти, под нее выделяют ❸ страничный кадр и заполняют ❹ (read) соответствующим содержимым файла. Любые последующие изменения, сделанные процессом в отображенных страницах, сохраняются ❺ обратно (write back) в файл, если отображение выполнено «разделяемым» (shared) способом. Для страниц, отображенных «частным» (private) способом, используется принцип COW (copy-on-write) ❻, согласно которому любые попытки

[1] Зачастую используют термин W:[подкачка страниц].

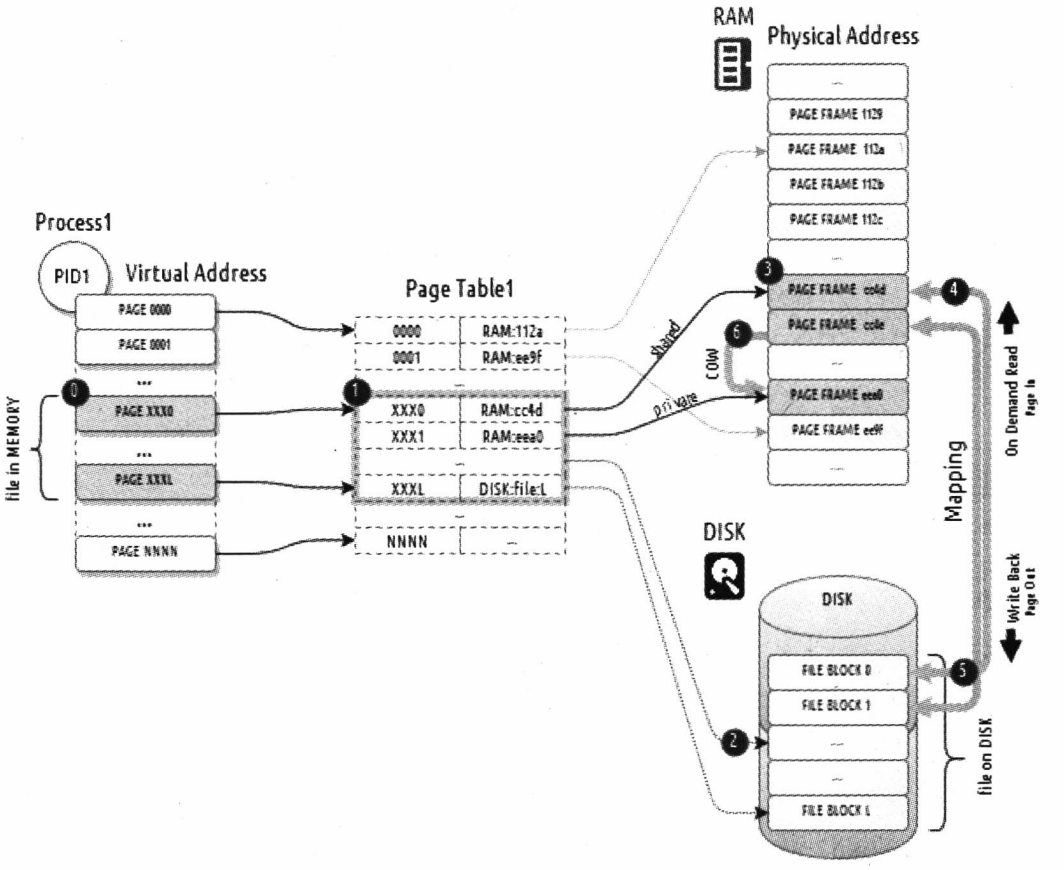


Рис. 4.5. Отображение файла в память

их изменения (write) приводят к созданию их копий (copy), куда и попадают изменения.

Таким образом, страницы отображенного файла, которые никогда не были востребованы процессом, не будут вовсе занимать оперативной памяти. Это обстоятельство широко используется для «загрузки» в процесс его программы и библиотек. В листинге 4.32 при помощи команды **pmap(1)** показана карта (отображения файлов) памяти процесса командного интерпретатора **bash(1)**.

Листинг 4.32. Карта памяти процесса

```
fitz@ubuntu:~$ ps f
  PID TTY      STAT   TIME COMMAND
26958 pts/0    Ss     0:00 bash
28540 pts/0    R+     0:00  \_ ps f
fitz@ubuntu:~$ which bash
```

```
/bin/bash
fitz@ubuntu:~$ readelf -l /bin/bash
           ...          ...             ...             ...             ...
Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
           ...          ...             ...             ...             ...
①  LOAD           0x000000 0x08048000 0x08048000 0xdb0c8 0xdb0c8 R E 0x1000
②  LOAD           0x0dbf04 0x08124f04 0x08124f04 0x04870 0x09820 RW  0x1000
           ...          ...             ...             ...             ...
   GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x4
   GNU_RELRO      0x0dbf04 0x08124f04 0x08124f04 0x000fc 0x000fc R   0x1
fitz@ubuntu:~$ pmap -d 26958
26958:   bash     ❹
Address   Kbytes Mode  Offset           Device    Mapping
❶ 08048000     880 r-x-- 0000000000000000 0fc:00000 bash
  08124000       4 r---- 00000000000db000 0fc:00000 bash
❷ 08125000      20 rw--- 00000000000dc000 0fc:00000 bash
  0812a000      20 rw--- 0000000000000000 000:00000   [ anon ]
❹ 086b9000    3676 rw--- 0000000000000000 000:00000   [ anon ]
           ...          ...             ...             ...             ...
❸ b753b000    1676 r-x-- 0000000000000000 0fc:00000 libc-2.15.so
  b76de000       8 r---- 00000000001a3000 0fc:00000 libc-2.15.so
  b76e0000       4 rw--- 00000000001a5000 0fc:00000 libc-2.15.so
  b76e1000      16 rw--- 0000000000000000 000:00000   [ anon ]
           ...          ...             ...             ...             ...
  bfe78000     132 rw--- 0000000000000000 000:00000   [ stack ]
  mapped: 10312K    writeable/private: 3908K    shared: 240K
```

В память процесса интерпретатора отображен исполняемый ELF-файл его программы ❶❷ и ELF-файлы всех библиотек ❸, от которых она зависит. Отображение ELF-файлов выполняется частями — *сегментами* (при помощи readelf(1) можно получить их список), в зависимости от их назначения. Так, например, сегмент программного кода ① отображен в страницы ❶, доступные на чтение **r** и выполнение **x**, сегмент данных ② отображен в страницы ❷, доступные на чтение **r** и запись **w**, и т. д.

Более того, выделение страниц памяти по требованию в процессе работы процесса ❹ реализуется при помощи «воображаемого» отображения некоторого несуществующего, *«анонимного файла»* **[anon]** на страничные кадры. Необходимо отметить, что механизм виртуальной памяти при освобождении неиспользуемых страниц выгружает в специальную область подкачки SWAP (см. рис. 4.4) только «анонимные» страничные кадры и «анонимизированные», полученные копированием при изменении (согласно принципу COW). Неанонимные измененные кадры выгружаются непосредственно в соответствующие им файлы, а неизмененные освобождаются вовсе без выгрузки, т. к. уже «заранее выгружены».

В примере из листинга 4.33 иллюстрируются два способа выделения памяти по требованию: *явный* — при помощи системного вызова **mmap(2)** с аргументом **MAP_ANONYMOUS**, и *неявный*[1] — при помощи системного вызова **brk(2)**. Явный способ ❺ позволяет выделять *новые* сегменты памяти процесса, тогда как неявный способ ❹ изменяет размер предопределенного «сегмента данных» процесса, позволяя увеличивать и уменьшать его по желанию, перемещая так называемый «break» — адрес конца этого сегмента.

Листинг 4.33. Системные вызовы mmap/munmap и brk — выделение и высвобождение памяти

```
fitz@ubuntu:~ $ ldd hostname
        linux-gate.so.1 =>  (0xb7709000)
        libnsl.so.1 => /lib/i386-linux-gnu/libnsl.so.1 (0xb76cf000)
        libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xb7526000)
☛       /lib/ld-linux.so.2 (0xb770a000)
fitz@ubuntu:~$ strace hostname
❶ execve("/bin/hostname", ["hostname"], [/* 43 vars */]) = 0
   brk(0)                                = 0x9e82000
   access("/etc/ld.so.nohwcap", F_OK)    = -1 ENOENT (No such file or directory)
   mmap2(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb770e000
   access("/etc/ld.so.preload", R_OK)    = -1 ENOENT (No such file or directory)
┌  open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
❷ fstat64(3, {st_mode=S_IFREG|0644, st_size=122101, ...}) = 0
└▶ mmap2(NULL, 122101, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb76f0000      ①
   close(3)                              = 0
   access("/etc/ld.so.nohwcap", F_OK)    = -1 ENOENT (No such file or directory)
┌  open("/lib/i386-linux-gnu/libnsl.so.1", O_RDONLY|O_CLOEXEC) = 3
❸ read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0`1\0\0004\0\0\0"..., 512) = 512
|  fstat64(3, {st_mode=S_IFREG|0644, st_size=92016, ...}) = 0
├▶ mmap2(NULL, 104424, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0xb76d6000
└▶ mmap2(0xb76ec000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|...|MAP_DENYWRITE, 3, 0x15) = 0xb76ec000 ②
❹ mmap2(0xb76ee000, 6120, PROT_READ|PROT_WRITE, MAP_PRIVATE|...|MAP_ANONYMOUS, -1, 0) = 0xb76ee000
   close(3)                              = 0
   access("/etc/ld.so.nohwcap", F_OK)    = -1 ENOENT (No such file or directory)
┌  open("/lib/i386-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
❺ read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0000\226\1\0004\0\0\0"..., 512) = 512
|  fstat64(3, {st_mode=S_IFREG|0755, st_size=1730024, ...}) = 0
```

[1] Доставшийся в наследство от классической ОС **UNIX**.

```
├► mmap2(NULL, 1739484, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0xb752d000
└► mmap2(0xb76d0000, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|...|MAP_DENYWRITE, 3, 0x1a3) = 0xb76d0000     ③
❸ mmap2(0xb76d3000, 10972, PROT_READ|PROT_WRITE, MAP_PRIVATE|...|MAP_ANONYMOUS, -1, 0) = 0xb76d3000
  close(3)                              = 0
  mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb752c000
  mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb752b000
  set_thread_area({entry_number:-1 -> 6, base_addr:0xb752b6c0, limit:1048575, seg_32bit:1, contents:0,
  read_exec_only:0, limit_in_pages:1, seg_not_present:0, useable:1}) = 0
③ mprotect(0xb76d0000, 8192, PROT_READ)  = 0
② mprotect(0xb76ec000, 4096, PROT_READ)  = 0
  mprotect(0x804b000, 4096, PROT_READ)   = 0
  mprotect(0xb7731000, 4096, PROT_READ)  = 0
① munmap(0xb76f0000, 122101)             = 0                          ↥ /lib/ld-linux.so.2
-------------------------------------------------------------------------------------------
  brk(0)                                = 0x9e82000                    ↧ /bin/hostname
❹ brk(0x9ea3000)                        = 0x9ea3000
  uname({sys="Linux", node="ubuntu", ...}) = 0
  fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 2), ...}) = 0
❺ mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb770d000
  write(1, "ubuntu\n", 7ubuntu
  )                 = 7
  exit_group(0)
```

Трасса команды **hostname(1)**, показанная в листинге 4.33, поясняет работу загрузчика и компоновщика (loader, ld) динамических библиотек **ld-linux(8)**.

Системный вызов **exec(2)** ❶ отображает для запуска в память процесса не только заданный ELF-файл **/bin/hostname**, но и (указанный в этом ELF-файле) загрузчик библиотек **/lib/ld-linux.so.2**, которому и передается управление до самой программы.

Загрузчик библиотек, в свою очередь, отображает ❶ в процесс свой «конфигурационный» файл **/etc/ld.so.cache**, а затем посегментно ❷ отображает файлы всех библиотек ❷ и выделяет им ❸ требуемую дополнительную память. Загруженные библиотеки присоединяются (линкуются или же компонуются, linking) к программе **/bin/hostname**, после чего страницам их отображенных сегментов назначается ②③ соответствующий режим доступа системным вызовом **mprotect(2)**. По завершении компоновки отображение конифгурационного файла **/etc/ld.so.cache** снимается ① при помощи **munmap(2)**, а управление передается исходной программе.

4.7.3. Потребление памяти

Суммарное распределение страниц памяти по сегментам процесса можно получить при помощи третьего набора столбцов (активировав его клавишами G,3) команды top(1), как показано в листинге 4.34. В столбце VIRT изображается суммарный объем (в килобайтах) всех страниц процесса, а в столбце RES — объем резидентных страниц (находящихся в страничных кадрах оперативной памяти). В столбце SWAP указывается объем всех страниц, находящихся во вторичной памяти — как «анонимных» страниц, выгруженных в специальную область подкачки, так и «файловых» страниц, возможно, никогда не загружавшихся в оперативную память.

Столбцы CODE и DATA показывают объемы (в килобайтах) памяти, выделенной под сегменты кода и данных, а столбец SHR — объем резидентных страниц, которые используются (или могут быть использованы) совместно с другими процессами.

Листинг 4.34. Распределение памяти по назначению

```
   fitz@ubuntu:~$ top -p 26958
G3 top - 11:05:01 up 15 days, 47 min,  8 users,  load average: 0.27, 0.51, 0.51
   Tasks:   1 total,   0 running,   1 sleeping,   0 stopped,   0 zombie
   Cpu(s):  8.6%us,  2.4%sy,  0.0%ni, 88.6%id,  0.4%wa,  0.0%hi,  0.0%si,  0.0%st
   Mem:   8192144k total,  7037720k used,  1154424k free,   238984k buffers
   Swap:  4104188k total,    35376k used,  4068812k free,  4356372k cached

     PID %MEM  VIRT SWAP  RES CODE DATA  SHR nFLT nDRT S  PR  NI %CPU COMMAND
   26958  0.1 10288 4936 5352  880 3852 1544    0    0 S  20   0    0 bash
     ...          ...             ...             ...             ...            ...
```

Механизм отображения считывает содержимое файла в страничные кадры только один раз, вне зависимости от количества процессов, отображающих этот файл в свою память. В случае отображения одного файла разными процессами (рис. 4.6) их страницы *совместно* отображаются на одни и те же страничные кадры, за исключением страниц, скопированных (согласно принципу COW) при изменении.

Такое поведение механизма отображения позволяет эффективно использовать оперативную память за счет использования разными программами одинаковых разделяемых библиотек. Так как ELF-файлы библиотек «загружаются» в память процессов при помощи отображения mmap(2), то в результате каждая библиотека размещается в оперативной памяти лишь единожды, вне зависимости от количества ее «использований».

Интегральная статистика по использованию виртуальной памяти может быть получена при помощи команды free(1), как показано в листинге 4.35.

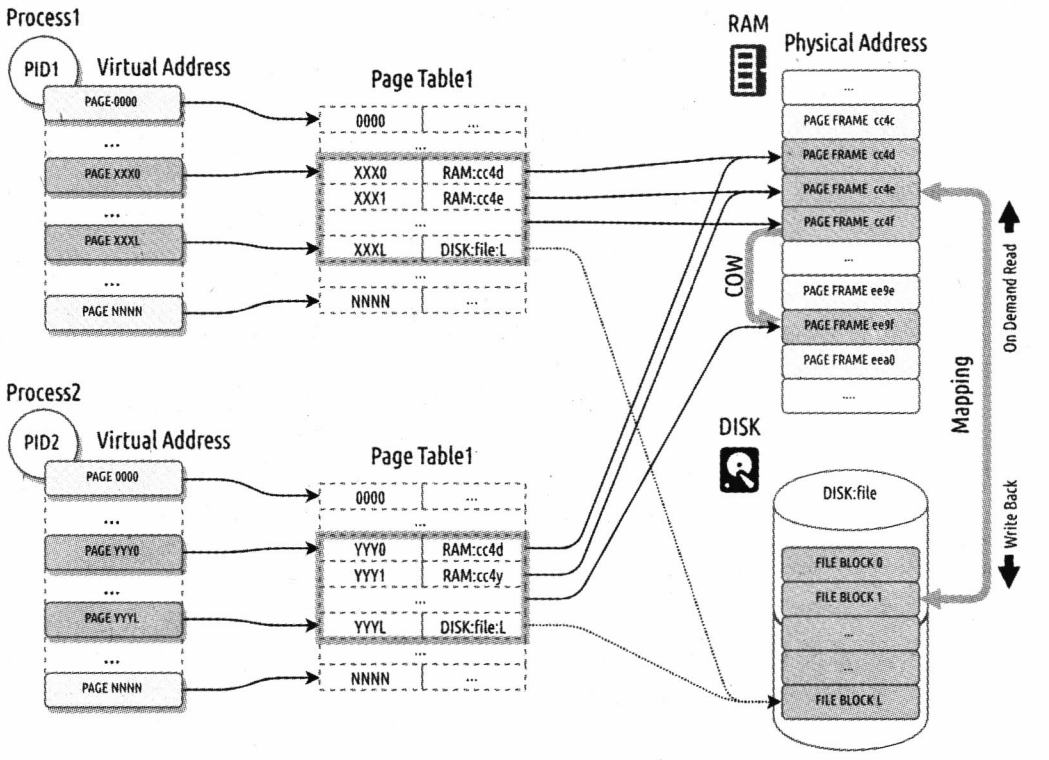


Рис. 4.6. Совместное использование памяти

Листинг 4.35. Статистика использования памяти

```
fitz@ubuntu:~$ free -m
             total       used       free     shared    buffers     cached
Mem:          8000       5523       2476          0        219       3430
-/+ buffers/cache:       1873       6126
Swap:         4007         17       3990
```

Строка **Mem:** содержит статистику использования оперативной памяти, а строка **Swap:** — статистику специальной области подкачки. В столбце **total** указан суммарный объем всех доступных страничных кадров, а в столбцах **used** и **free** — суммарные объемы использованных и свободных страничных кадров, соответственно.

В столбце **cached** указан объем *страничного кэша* (page cache), т. е. суммарный объем страничных кадров оперативной памяти, использованных под отображение файлов в память. Аналогично, в столбце **buffers** указывается объем *буферного кэша*, т. е. суммарный объем памяти, использованной ядром для кэширования «не-

отображаемых» сущностей: метаданных файлов, дисковых блоков при прямом вводе-выводе на устройства и пр.

В столбцах **used** и **free** строки **-/+ buffers/cache** указываются объемы использованной и свободной памяти «за вычетом» страничного и буферного кэшей, т. е. «чистая» память, выделенная процессам по требованию под данные, сгенерированные в процессе работы.

В листинге 4.36 показан пример потребления памяти процессом текстового редактора **vi(1)** при попытке редактирования громадного файла в 1 Гбайт. Процесс загружает файл целиком, в результате чего он целиком оказывается в резидентных страницах ❶ сегмента данных ❸ процесса, что естественным образом увеличивает «чистый» расход оперативной памяти системы ❷. После принудительного завершения процесса при помощи команды **kill(1)** память естественным образом высвобождается.

Листинг 4.36. Потребление памяти процессами

```
                                                                    pts/0
fitz@ubuntu:~$ dd if=/dev/urandom of=big bs=4069 count=262144
262144+0 записей получено
262144+0 записей отправлено
скопировано 1073741824 байта (1,1 GB), 148,956 с, 7,2 MB/c
fitz@ubuntu:~$ ls -lh big
-rw-r--r-- 1 fitz fitz 1,0G дек.   3 12:26 big
fitz@ubuntu:~$ free -m
             total       used       free     shared    buffers     cached
Mem:          8000       3947       4052          0         42       2712
-/+ buffers/cache:       1192 ←     6807
Swap:         4007          0       4007
fitz@ubuntu:~$ vi big
```

```
                                                                    pts/1
fitz@ubuntu:~$ ps f
  PID TTY      STAT   TIME COMMAND
20595 pts/1    S      0:00 -bash
21087 pts/1    R+     0:00  \_ ps f
20437 pts/0    S      0:00 -bash
21085 pts/0    Rl+    0:08  \_ vi big
                              ① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪
fitz@ubuntu:~$ ps up 21085
USER       PID %CPU %MEM    VSZ   RSS      TTY      STAT START   TIME COMMAND
❶ fitz   21085 96.4 21.8 1826416 1786444 ← pts/0 Sl+  21:14   0:18 vi big
```

```
fitz@ubuntu:~$ free -m
             total       used       free     shared    buffers     cached
Mem:          8000       5687       2312          0         42       2709
❷ -/+ buffers/cache:       2935 ☚     5065
Swap:         4007          0       4007
fitz@ubuntu:~$ top -b -n1 -p 21085
top - 21:14:43 up  3:56,  3 users,  load average: 0.53, 0.40, 0.47
Tasks:   1 total,   0 running,   1 sleeping,   0 stopped,   0 zombie
Cpu(s):  6.0%us,  5.8%sy,  0.2%ni, 83.7%id,  4.3%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   8192144k total,  5826856k used,  2365288k free,    43932k buffers
Swap:  4104188k total,        0k used,  4104188k free,  2777504k cached

  PID %MEM  VIRT SWAP   RES  CODE DATA   SHR nFLT nDRT S  PR  NI %CPU COMMAND
❸ 21085 21.8 1783m  39m ☛ 1.7g 2148 1.7g ☚ 5676    0    0 S  20   0    0 vi
fitz@ubuntu:~$ kill 21085
fitz@ubuntu:~$ free -m
             total       used       free     shared    buffers     cached
Mem:          8000       3945       4054          0         42       2709
-/+ buffers/cache:       1193 ☚     6806
Swap:         4007          0       4007
```

4.8. Механизм сигналов

Механизм сигналов **signal(7)** является простейшей формой межпроцессного взаимодействия и предназначен для внешнего управления процессами. Каждый сигнал имеет свой обработчик, определяющий поведение процесса при отсылке ему этого сигнала. Этот обработчик является некоторым набором инструкций программы (подпрограммой), которым передается управление при доставке сигнала процессу. Каждому процессу назначаются обработчики «по умолчанию», в большинстве случаев приводящие к завершению процесса.

В листинге 4.37 показано применение сигнала штатного прерывания процесса **№2 (SIGINT)**, отсылаемого драйвером терминала всем процессам «переднего» фона при получении символа **^C (ETX)**, т. е. нажатии клавиш **Ctrl+C** ❶ на этом терминале. Для процессов «заднего» фона сигнал может быть отослан явно ❷, при помощи команды **kill(1)**, предназначенной, несмотря на ее название[1], для отсылки сигналов.

[1] В большинстве случаев обработчик завершит процесс, отсюда и название.

Листинг 4.37. Штатное прерывание процесса (^C, intr, SIGINT)

```
fitz@ubuntu:~$ stty -a
speed 38400 baud; rows 24; columns 80; line = 0;
① intr = ^C; ➥ quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = M-^?; eol2 = M-^?;
            ...          ...          ...          ...          ...
② isig ➥ icanon iexten echo echoe echok -echonl -noflsh -xcase -tostop -echoprt
echoctl echoke
fitz@ubuntu:~$ dd if=/dev/zero of=/dev/null
❶ ^C782349+0 записей получено
782349+0 записей отправлено
скопировано 400562688 байт (401 MB), 0,531532 с, 754 MB/с

fitz@ubuntu:~$ dd if=/dev/zero of=/dev/null &
[1] 23418
fitz@ubuntu:~$ ^C
            ...          ...          ...          ...          ...
fitz@ubuntu:~$ ^C
fitz@ubuntu:~$ jobs -l
[1]+ 23418 Выполняется  dd if=/dev/zero of=/dev/null &
❷ fitz@ubuntu:~$ kill -SIGINT 23418
fitz@ubuntu:~$ 25318811+0 записей получено ➥
25318810+0 записей отправлено
скопировано 12963230720 байт (13 GB), 17,1001 с, 758 MB/с

[1]+  Прерывание     dd if=/dev/zero of=/dev/null
```

Нужно отметить, что настройки драйвера терминала позволяют как переопределить ① символ, получение которого приведет к выполнению действия **intr** (посылка сигнала **SIGINT**), так и совсем выключить ② отсылку подобных сигналов управления процессами при получении управляющих символов.

В листинге 4.38 показано аналогичное применение сигнала аварийного завершения процесса **№ 3 (SIGQUIT)**, посылаемого процессам «переднего» фона при получении драйвером управляющего символа **^\ (FS)**, генерируемого терминалом при нажатии **Ctrl+**. Обработчик сигнала не только завершит процесс, но и попытается (если позволяют настройки) сохранить дамп памяти процесса в файл **core(5)** для последующей отладки.

Листинг 4.38. Аварийное прерывание процесса (^\, quit, SIGQUIT)

```
fitz@ubuntu:~$ dd if=/dev/zero of=/dev/null
^\Выход (сделан дамп памяти)
```

```
fitz@ubuntu:~$ dd if=/dev/zero of=/dev/null &
[1] 23429
fitz@ubuntu:~$ kill -SIGQUIT 23429
[1]+  Выход              (подготовлен дамп ядра) dd if=/dev/zero of=/dev/null
```

Некоторые процессы (например, демоны, или графические приложения) не имеют управляющего терминала, поэтому не могут быть завершены интерактивно при помощи Ctrl+C или Ctrl+\. В этом случае используется сигнал штатного завершения № 15 (SIGTERM), что проиллюстрировано в листинге 4.39.

Листинг 4.39. Штатное завершение процесса (SIGTERM)

```
fitz@ubuntu:~$ dd if=/dev/zero of=/dev/null &
[1] 23444
fitz@ubuntu:~$ kill -SIGTERM 23444
fitz@ubuntu:~$
[1]+  Завершено       dd if=/dev/zero of=/dev/null
```

Для приостановки процесса, т. е. временного исключения его из процедур распределения процессорного времени планировщиком, предназначен сигнал № 19 (SIGSTOP), а для возобновления процесса — сигнал № 18 (SIGCONT). В листинге 4.40 показано, что приостановленный (sTopped) процесс не потребляет процессорного времени.

Листинг 4.40. Приостановка и возобновление процесса (SIGSTOP и SIGCONT)

```
fitz@ubuntu:~$ pbzip2 big &
[1] 1647
fitz@ubuntu:~$ top -b -n1 -p 1647
          ...          ...          ...          ...          ...
  PID USER      PR  NI  VIRT  RES  SHR S  %CPU %MEM    TIME+  COMMAND
 1647 fitz      20   0  102m  33m  900 S  387  0.4   0:25.09 pbzip2
fitz@ubuntu:~$ pkill -STOP pbzip2
fitz@ubuntu:~$ ps f
 PID TTY      STAT   TIME COMMAND
1640 pts/2    S      0:00 -bash
1647 pts/2    T      0:24  \_ pbzip2 big
1651 pts/2    R+     0:00  \_ ps f
fitz@ubuntu:~$ jobs -l
[1]+  1647 Остановлено (сигнал)            pbzip2 big
fitz@ubuntu:~$ top -b -n1 -p 1647
          ...          ...          ...          ...          ...
```

```
  PID USER      PR  NI  VIRT  RES  SHR    S  %CPU %MEM    TIME+  COMMAND
 1647 fitz      20   0  102m  34m  900 ☛ T  ☛ 0  0.4   0:42.90 pbzip2
fitz@ubuntu:~$ kill -CONT 1647
fitz@ubuntu:~$ top -b -n1 -p 1647
          ...          ...          ...          ...          ...
  PID USER      PR  NI  VIRT  RES  SHR S  %CPU %MEM    TIME+  COMMAND
 1647 fitz      20   0  102m  34m  900 S ☛370  0.4   0:51.82 pbzip2
```

Сигналы могут быть перехвачены, если процесс назначит собственный обработчик, а также проигнорированы, тогда при их доставке вообще никакой обработчик не вызывается. Исключение составляют некоторые «безусловные» сигналы, такие как **№ 9 (SIGKILL)** — безусловное завершение или **№ 19 (SIGSTOP)** — безусловная приостановка процесса. Игнорирование и перехват сигналов показаны в листинге 4.41 на примере командного интерпретатора **bash(1)**. Ни попытки «интерактивного» завершения ❶ при помощи сигналов **SIGINT** и **SIGQUIT**, ни явная посылка ❷ сигналов **SIGINT** и **SIGTERM** не приводят к завершению командного интерпретатора. К желаемому результату приводит только явная отсылка ❸ сигнала **SIGKILL**.

Листинг 4.41. Игнорирование и перехват сигналов

```
  fitz@ubuntu:~$ ps f
    PID TTY      STAT   TIME COMMAND
  23025 pts/1    S      0:00 -bash
  23771 pts/1    R+     0:00  \_ ps f
  fitz@ubuntu:~$ bash
  fitz@ubuntu:~$ ps f
    PID TTY      STAT   TIME COMMAND
  23025 pts/1    S      0:00 -bash
  23636 pts/1    S      0:00  \_ bash
  23692 pts/1    R+     0:00      \_ ps f
❶ fitz@ubuntu:~$ ^C
❶ fitz@ubuntu:~$ ^\
  fitz@ubuntu:~$ ps f
    PID TTY      STAT   TIME COMMAND
  23025 pts/1    S      0:00 -bash
!  23636 pts/1    S      0:00  \_ bash
  23692 pts/1    R+     0:00      \_ ps f
❷ fitz@ubuntu:~$ kill -SIGINT 23636
  fitz@ubuntu:~$ ps f
    PID TTY      STAT   TIME COMMAND
```

```
   23025 pts/1    S      0:00 -bash
!  23636 pts/1    S      0:00  \_ bash
   23701 pts/1    R+     0:00      \_ ps f
❷ fitz@ubuntu:~$ kill -SIGTERM 23636
   fitz@ubuntu:~$ ps f
     PID TTY      STAT   TIME COMMAND
   23025 pts/1    S      0:00 -bash
!  23636 pts/1    S      0:00  \_ bash
   23708 pts/1    R+     0:00      \_ ps f
❸ fitz@ubuntu:~$ kill -SIGKILL 23636
   Убито
   fitz@ubuntu:~$ ps f
     PID TTY      STAT   TIME COMMAND
   23025 pts/1    S      0:00 -bash
   23771 pts/1    R+     0:00  \_ ps f
```

Диспозицию сигналов, т. е. информацию об игнорировании (IGNORED), перехвате (CAUGHT), временной блокировке (BLOCKED) или ожидании доставки (PENDING) сигналов процессов можно получить при помощи команды **ps(1)**, как показано в листинге 4.42. Диспозиция изображается битовой маской, где каждый N-й бит маски (нумерация от младших к старшим) соответствует сигналу N. Например, командный интерпретатор игнорирует сигналы, представленные (шестнадцатеричной) маской 00384004_{16}, что в двоичном представлении составляет $11100001000000000000100_2$ и указывает на игнорируемые 3-й, 15-й, 20-й, 21-й и 22-й сигналы т. е. SIGQUIT, SIGTERM, SIGTSTP, SIGTTIN и SIGTTOU. Для перевода из шестнадцатеричного в двоичное представление использован стековый калькулятор **dc(1)**, которому было велено из входной **i** (input) системы счисления по основанию **16** в выходную **o** (output) систему счисления по основанию **2** напечатать **p** (print) число **00384004**, а для просмотра имен сигналов по их номерам использована встроенная команда kill.

Листинг 4.42. Диспозиция сигналов

```
fitz@ubuntu:~$ ps s
  UID   PID   PENDING   BLOCKED   IGNORED    CAUGHT STAT TTY        TIME COMMAND
 1006  23025 00000000  00010000  00384004  4b813efb S    pts/5      0:00 -bash
 1006  23773 00000000  00000000  00000000  73d3fef9 R+   pts/5      0:00 ps s
fitz@ubuntu:~$ dc -e 16i2o00384004p
11100001000000000000100
222120----15------87654321
```

```
fitz@ubuntu:~$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
...             ...             ...             ...             ...
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
...             ...             ...             ...             ...
63) SIGRTMAX-164) SIGRTMAX
```

4.8.1. Сеансы и группы процессов: управление заданиями

Одним из основных применений сигналов при интерактивной работе пользователя в системе является механизм управления «заданиями», которыми пользуются командные интерпретаторы и подобные интерактивные программы, например **lftp(1)**.

Для удобства управления процессами при помощи сигналов они объединяются в *группы* и *сеансы* (см. **credentials(7)**), проиллюстрированные в листинге 4.43 при помощи команды **ps(1)** атрибутами **PGID** (**p**rocess **g**roup **id**entifier) и **SID** (**s**ession **id**entifier). Процесс (создавший группу), чей идентификатор **PID** совпадает ② с идентификатором **PGID** группы, носит название *лидера группы*. Процесс (создавший сеанс), чей идентификатор **PID** совпадает ① с идентификатором **SID** сеанса, называется *лидером сеанса*. Нужно отметить, что лидер сеанса в столбце **STAT** отмечается флагом **s**, а процессы группы переднего фона — флагом **+**. Только одна группа сеанса, называемая «терминальной» **TGPID**, является группой «переднего» (foreground) фона, остальные группы сеанса являются группами «заднего» (background) фона.

Командный интерпретатор формирует свои задания «заднего» ❶ или «переднего» фона ❷, помещая процессы заданий в соответствующие группы. Механизм управления заданиями всегда посылает «терминальные» сигналы **^C SIGINT**, **^\ SIGQUIT** всем процессам текущей «терминальной» группы. Для смены терминальной группы используется сигнал **№ 20 SIGTSTP** (**t**erminal **st**op **sig**nal), также отсылаемый всем процессам «терминальной» группы при получении драйвером терминала управляющего символа **^Z** (**SUB**), генерируемого клавишами **Ctrl**+**Z**. Обработчик сигнала **SIGTSTP** по умолчанию приостанавливает процессы, и управление возвращается к командному интерпретатору ❸, группа которого становится «терминальной». При помощи встроенных команд **fg** (**fo**re**g**round), **bg** (**b**ack**g**round) можно продолжить (**SIGCONT**) выполнение указанного задания (всех процессов его группы) на «переднем» или «заднем» фоне, а при помощи команды **jobs -l** получить список всех заданий вместе с номерами их групп процессов.

Листинг 4.43. Сеансы и группы процессов — задания интерпретатора

```
❶ fitz@ubuntu:~$ dd if=/dev/zero of=/dev/null &
  [1] 3181
```

```
❹ fitz@ubuntu:~$ ps jf
   PPID   PID  PGID   SID  TTY      TPGID STAT   UID   TIME COMMAND
①  3094  3099  3099  3099 pts/0     3159 Ss     1000   0:00 bash
②  3099  3181  3181  3099 pts/0     3182 R      1000   0:08  \_ dd if=/dev/zero of=...
②  3099  3182  3182  3099 pts/0     3182 R+     1000   0:00  \_ ps jf
   fitz@ubuntu:~$ man dd
   ...  ...     ...     CTRL+Z  ...     ...     ...
❸ [2]+  Остановлено  man dd
   fitz@ubuntu:~$ jobs -l
   [1]-  3181 Выполняется  dd if=/dev/zero of=/dev/null &
   [2]+  3193 Остановлено  man dd
   fitz@ubuntu:~$ ps jf
   PPID   PID  PGID   SID  TTY      TPGID STAT   UID   TIME COMMAND
    3094  3099  3099  3099 pts/0     3310 Ss     1000   0:00 bash
    3099  3181  3181  3099 pts/0     3310 R      1000   4:48  \_ dd if=/dev/zero of=
    3099  3193  3193  3099 pts/0     3310 T      1000   0:00  \_ man dd
    3193  3203  3193  3099 pts/0     3310 T      1000   0:00  |   \_ pager -s
    3099  3310  3310  3099 pts/0     3310 R+     1000   0:00  \_ ps jf
   fitz@ubuntu:~$ fg %1
   dd if=/dev/zero of=/dev/null
   ^Z
   [1]+  Остановлено  dd if=/dev/zero of=/dev/null
   fitz@ubuntu:~$ ps f
     PID TTY      STAT   TIME COMMAND
    3099 pts/0    Ss     0:00 bash
    3181 pts/0    T      26:44  \_ dd if=/dev/zero of=/dev/null
    3193 pts/0    T      0:00  \_ man dd
    3203 pts/0    T      0:00  |   \_ pager -s
    3937 pts/0    R+     0:00  \_ ps ffitz@ubuntu:~$ bg 1
   [1]+ dd if=/dev/zero of=/dev/null &
   fitz@ubuntu:~$ fg %2
   ...  ...     ...     Q       ...     ...     ...
   man dd
   fitz@ubuntu:~$ jobs
   [1]+  Остановлено  dd if=/dev/zero of=/dev/null
   fitz@ubuntu:~$ fg
   dd if=/dev/zero of=/dev/null
   ^C11771330744+0 записей получено
   11771330743+0 записей отправлено
   скопировано 6026921340416 байт (6,0 TB), 8400,83 с, 717 MB/с
```

Кроме переключения группы «переднего» фона, механизм управления заданиями координирует «совместный» доступ процессов к управляющему терминалу. При «одновременном» вводе информации с одного терминала несколькими процессами результат оказывается непредсказуем, т. к. нет возможности предугадать порядок и объемы считываемой информации. Поэтому ввод (input) разрешен только процессам группы «переднего» фона, а группа формируется так, что только один из них в реальности будет производить чтение. Процессам группы «заднего» фона ввод запрещен, а любые попытки подавляются при помощи сигнала **SIGTTIN** (terminal stop on input signal), доставка которого приводит к приостановке процесса.

В примере из листинга 4.44 при составлении текста письма посредством команды **mail(1)** ее процесс был временно приостановлен ❶ при помощи **^Z** и **SIGTSTP** для получения доступа к командному интерпретатору. Попытка продолжить ❷ выполнение задания **mail** на «заднем» фоне не увенчалась успехом, т. к. была подавлена ❸ за чтение терминала. Продолжение задания на «переднем» фоне ❹ дает возможность закончить ввод текста письма и завершить ввод управляющим символом **^Z** (EOT).

Листинг 4.44. Приостановка при вводе из заднего фона (SIGTTIN)

```
fitz@ubuntu:~$ mail dketov@gmail.com
Subject: schedtool(1) вместо taskset, chrt и nice/renice
^Z
❶ [2]+  Остановлено  mail dketov@gmail.com
fitz@ubuntu:~$ which schedtool
/usr/bin/schedtool
fitz@ubuntu:~$ dpkg -S /usr/bin/schedtool
schedtool: /usr/bin/schedtool
❷ fitz@ubuntu:~$ bg
? [1]+ mail dketov@gmail.com &
? (continue)
? fitz@ubuntu:~$ jobs -l
❸ [1]+ 12025 Остановлено (ввод с терминала)                mail dketov@gmail.com
fitz@ubuntu:~$ ps fj
  PID TTY      STAT   TIME COMMAND
 8992 pts/2    Ss     0:00 bash
12025 pts/2    T      0:00  \_ mail dketov@gmail.com
12063 pts/2    R+     0:00  \_ ps f
 8897 pts/0    Ss+    0:00 bash
❹ fitz@ubuntu:~$ fg
mail dketov@gmail.com
```

```
(continue)
Утилита schedtool(1) из одноименного пакета schedtool заменяет "стандартные"
taskset/nice/renice/chrt↵
^D
Cc:↵
fitz@ubuntu:~$
```

При «одновременном» выводе информации на один терминал несколькими процессами результат точно так же непредсказуем, как и при вводе. В итоге будет получена смесь перемежающихся строчек разных процессов, однако по умолчанию вывод (output) разрешен как процессам группы «переднего» фона, так и процессам всех групп «заднего» фона. Настроечный флаг драйвера терминала **tostop** (terminal **o**utput stop) позволяет запретить вывод из заднего фона так же, как и ввод. При запрещенном выводе из заднего фона все попытки будут подавляться сигналом **SIGTTOU** (terminal stop on **ou**tput **sig**nal), приостанавливающим процесс. В листинге 4.45 проиллюстрировано действие сигнала **SIGTTOU** при включении настроечного флага **tostop** посредством команды **stty(1)**.

Листинг 4.45. Приостановка при выводе из заднего фона (SIGTTOU)

```
fitz@ubuntu:~$ find / -type f -size 0 & ☚
!?      ...      ^C ☚   ...      ^\ ☚   ...      ^Z ☚   ...              ...      ...
                                ① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪
fitz@ubuntu:~$ stty -a
speed 38400 baud; rows 24; columns 80; line = 0;
            ...          ...            ...            ...            ...
isig icanon iexten echo ... -echonl -noflsh -xcase -tostop -echoprt echoctl echoke
fitz@ubuntu:~$ stty tostop
fitz@ubuntu:~$ find / -type f -size 0 &
[1] 356
fitz@ubuntu:~$ jobs -l
[1]+   356 Остановлено (вывод на терминал)                find / -type f -size 0
fitz@ubuntu:~$ ps f
  PID TTY      STAT   TIME COMMAND
32535 pts/1    S      0:00 -bash
  356 pts/1    T  ☚   0:00  \_ find / -type f -size 0
  360 pts/1    R+     0:00  \_ ps f
```

4.9. Межпроцессное взаимодействие

Кроме сигналов, которые могут использоваться как простейшие средства межпроцессного взаимодействия (IPC, **i**nter-**p**rocess **c**ommunication), для эффективного обмена информацией между процессами используются каналы, сокеты, очереди сообщений и разделяемая память, а для синхронизации действий процессов над совместно используемыми объектами — семафоры.

4.9.1. Неименованные каналы

Самым простым средством обмена информацией между родственными процессами (родитель и любые его потомки) являются *неименованные каналы*. Канал является «двусторонним однонаправленным безымянным файлом», с одного конца в который можно только записывать информацию, а с другого конца — только считывать. В отличие от «обычного» файла, при открытии которого создается только один файловый дескриптор (и для чтения, и для записи файла), при создании канала создаются сразу два дескриптора — один для передачи (записи) в канал, а другой — для приема (чтения) из канала. При порождении дочерних процессов файловые дескрипторы наследуются, что и позволяет им взаимодействовать как с родительским процессом, так и между собой.

Использование неименованных каналов в системе широко распространено. Например, командный интерпретатор использует их для организации конвейерной обработки *(см. разд. 5.3)*. Архиватор **tar(1)** использует каналы аналогичным способом — для упаковки архивов «на лету» при помощи «внешних» упаковщиков. В примере из листинга 4.46 посредством трассировщика **strace(1)** отслеживается системный вызов **pipe(2)**, при помощи которого **tar(1)** создает неименованный канал для связи с упаковщиком **xz(1)** в дочернем процессе. Вся группа процессов архиватора **tar(1)** приостановлена сигналом **^Z SIGTSTP**, после чего при помощи команды **lsof(1)** показаны файловые дескрипторы открытых файлов обоих процессов.

Листинг 4.46. Неименованные каналы

```
fitz@ubuntu:~$ strace -fe pipe,execve tar cJf /tmp/docs.tgz  /usr/share/doc
pipe([3, 4])                            = 0
Process 3967 attached
tar: Удаляется начальный `/' из имен объектов
          ...          ...          ...          ...          ...
[pid  3967] execve("/usr/bin/xz", ["xz"], [/* 43 vars */]) = 0
^Z
[1]+  Остановлено  strace -fe pipe tar cJf /tmp/docs.tgz /usr/share/doc
fitz@ubuntu:~$ ps f
  PID TTY      STAT   TIME COMMAND
```

```
 5472 pts/2    Ss     0:07 bash
 3965 pts/2    T      0:00  \_ strace -fe pipe tar cJf /tmp/docs.tgz /usr/share/doc
 3966 pts/2    t      0:00  |   \_ tar cJf /tmp/docs.tgz /usr/share/doc
 3967 pts/2    t      0:01  |       \_ xz ◄
 3968 pts/2    R+     0:00  \_ ps f
fitz@ubuntu:~$ lsof -p 3966
COMMAND  PID  USER   FD   TYPE DEVICE SIZE/OFF       NODE NAME
tar     3966  fitz  cwd    DIR  252,0    69632   20316162 /home/fitz
tar     3966  fitz  rtd    DIR  252,0     4096          2 /
tar     3966  fitz  txt    REG  252,0   305968    7208993 /bin/tar
          ...           ...              ...                ...
tar     3966  fitz   0u   CHR  136,2      0t0          5 /dev/pts/2
tar     3966  fitz   1u   CHR  136,2      0t0          5 /dev/pts/2
tar     3966  fitz   2u   CHR  136,2      0t0          5 /dev/pts/2
tar     3966  fitz   3r   DIR  252,0   106496    3801096 /usr/share/doc
tar     3966  fitz ◄ 4w  FIFO   0,8      0t0 ◄ 1358926 pipe ◄
          ...           ...              ...                ...
fitz@ubuntu:~$ lsof -p 3967
   COMMAND  PID  USER   FD   TYPE DEVICE SIZE/OFF       NODE NAME
xz      3967  fitz  cwd    DIR  252,0    69632   20316162 /home/fitz
xz      3967  fitz  rtd    DIR  252,0     4096          2 /
xz      3967  fitz  txt    REG  252,0    67460    3802386 /usr/bin/xz
          ...           ...              ...                ...
xz      3967  fitz ◄ 0r  FIFO   0,8      0t0 ◄ 1358926 pipe ◄
xz      3967  fitz   1w   REG  252,0  1466368   26874966 /tmp/docs.tgz
xz      3967  fitz   2u   CHR  136,2      0t0          5 /dev/pts/2
```

Нужно отметить, что в процессе архиватора **PID = 3966** файловый дескриптор передающей части канала сохранил свой номер, а в дочернем процессе упаковщика **PID = 3967** файловый дескриптор принимающей части канала был перенаправлен на **STDIN**, как того и ожидает упаковщик **xz(1)**.

4.9.2. Именованные каналы

Именованные каналы повторяют поведение неименованных каналов, но предназначены для обмена информацией между *неродственными* процессами. Любые две запущенные программы могут организовать однонаправленный канал передачи путем открытия файла канала по заранее согласованному *имени* на запись «с одной стороны» и на чтение «с другой». Файл канала должен быть предварительно создан в дереве каталогов при помощи специального системного вызова **mkfifo(3)**, а его последующее открытие осуществляется «обычным» системным вызовом **open(2)**. Именованные каналы на текущий момент времени используются достаточно редко

и практически вытеснены именованными локальными сокетами, но в некоторых случаях являются вполне достаточным средством взаимодействия. В примере из листинга 4.47 показан терминальный мультиплексор **W:[GNU screen]**, который в Ubuntu Linux все еще использует именованные каналы для однонаправленного взаимодействия между своими отключенным ❶ (detached) и повторно подключающимся (reattach, **-r**) ❷ экземплярами.

Листинг 4.47. Именованные каналы

```
fitz@ubuntu:~$ tty
/dev/pts/0
fitz@ubuntu:~$ screen
```

pts/2

```
fitz@ubuntu:~$ tty
/dev/pts/2
fitz@ubuntu:~$        ...     Ctrl+A C        ...
```

pts/4

```
fitz@ubuntu:~$ tty
/dev/pts/4
fitz@ubuntu:~$        ...     Ctrl+A D        ...
```

```
❶ [detached from 22261.pts-14.ubuntu]
fitz@ubuntu:~$ tty
/dev/pts/0
fitz@ubuntu:~$ ps fp 22261 t pts/2,pts/4
  PID TTY      STAT   TIME COMMAND
22261 ?        Ss     0:00 SCREEN
22262 pts/2    Ss+    0:00  \_ /bin/bash
28020 pts/4    Ss+    0:00  \_ /bin/bash
fitz@ubuntu:~$ screen -ls
There is a screen on:
        22261.pts-14.ubuntu     (27.02.2016 23:25:17)   (Attached)
1 Socket in /var/run/screen/S-fitz.
fitz@ubuntu:~$ ls -l /var/run/screen/S-fitz
итого 0
☛ prwx------ 1 fitz fitz 0 февр. 27 23:25 22261.pts-14.ubuntu
❷ fitz@ubuntu:~$ screen -r
```

pts/4

```
fitz@ubuntu:~$ tty
/dev/pts/4
fitz@ubuntu:~$ ^D
```

```
                                                                    pts/2
fitz@ubuntu:~$ tty
/dev/pts/2
fitz@ubuntu:~$ ^D
    [screen is terminating]
```

Терминальные мультиплексоры **screen(1)** и **tmux(1)** позволяют запускать несколько пользовательских «вторичных» сеансов одновременно, переключаться между ними, отсоединяться от них и снова присоединяться к ним из «первичного» сеанса. При этом «первичный» сеанс может быть организован как угодно, его можно завершать и начинать другим способом. Например, запустив терминальный мультиплексор на алфавитно-цифровом виртуальном терминале и отключившись от него, можно подключиться к нему и продолжить работу уже из графического эмулятора терминала или по сети с использованием службы удаленного доступа SSH *(см. разд. 6.4.1)*.

4.9.3. Неименованные локальные сокеты

Каналы являются однонаправленными средствами взаимодействия процессов, поэтому слабо подходят для двунаправленного обмена, например для организации обратной связи между процессами. В большинстве случаев именованные каналы позволяют эффективно реализовать только простейшую модель взаимодействия «поставщик → потребитель», тогда как для реализации модели «клиент ⇆ сервер» используют специальное средство взаимодействия, называемое *сокетом*[1]. Неименованные локальные (файловые) сокеты, как и неименованные каналы, являются «безымянными файлами», только *двунаправленными*, и так же предназначены для взаимодействия родственных процессов. Для создания пары соединенных сокетов используется системный вызов **socketpair(2)**, создающий пару файловых дескрипторов, каждый из которых используется одновременно и для приема (чтения), и для передачи (записи) информации. Применение неименованных файловых сокетов проиллюстрировано в листинге 4.48 на примере синхрокопировщика **rsync(1)**. Копировщик оказывается параллельной программой, использующей сокеты для взаимодействия своих параллельных ветвей.

Листинг 4.48. Неименованные локальные (файловые) сокеты

```
fitz@ubuntu:~$ strace -fe socketpair,execve rsync -a /usr/share/doc /tmp/
execve("/usr/bin/rsync", ["rsync", "-a", "/usr/share/doc", "/tmp/"], [/* 42 vars */]) = 0
```

[1] Сокет — устоявшаяся русская калька с англ. *socket*, буквально означающая «разъем», например, 220 В розетку и вилку, или сетевую розетку и вилку RJ-45, или 3,5 мм гнездо для наушников и соответствующий штекер.

```
socketpair(PF_FILE, SOCK_STREAM, 0, [3, 4]) = 0
socketpair(PF_FILE, SOCK_STREAM, 0, [5, 6]) = 0
Process 5268 attached
[pid  5268] socketpair(PF_FILE, SOCK_STREAM, 0, [3, 4]) = 0
Process 5269 attached
^Z
[1]+  Остановлено  strace -fe socketpair,execve rsync -a /usr/share/doc /tmp/
fitz@ubuntu:~$ ps f
  PID TTY      STAT   TIME COMMAND
 4838 pts/5    Ss     0:00 bash
 5266 pts/5    T      0:00  \_ strace -fe socketpair execve rsync -a /usr/share
 5267 pts/5    t      0:00  |   \_ rsync -a /usr/share/doc /tmp/
 5268 pts/5    t      0:00  |       \_ rsync -a /usr/share/doc /tmp/
 5269 pts/5    t      0:00  |           \_ rsync -a /usr/share/doc /tmp/
 5301 pts/5    R+     0:00  \_ ps f
fitz@ubuntu:~$ lsof -p 5269
COMMAND  PID   USER   FD   TYPE     DEVICE SIZE/OFF     NODE NAME
rsync   5269   fitz  cwd    DIR      252,0   139264 26869761 /tmp
...      ...    ...  ...    ...      ...     ...      ...    ...
rsync   5269   fitz    0u  unix 0x00000000      0t0  1752340 socket
...      ...    ...  ...    ...      ...     ...      ...    ...
rsync   5269   fitz    4u  unix 0x00000000      0t0  1751443 socket
fitz@ubuntu:~$ lsof -p 5268
COMMAND  PID   USER   FD   TYPE     DEVICE SIZE/OFF     NODE NAME
rsync   5268   fitz  cwd    DIR      252,0   139264 26869761 /tmp
...      ...    ...  ...    ...      ...     ...      ...    ...
rsync   5268   fitz    1u  unix 0x00000000      0t0  1752343 socket
...      ...    ...  ...    ...      ...     ...      ...    ...
rsync   5268   fitz    3u  unix 0x00000000      0t0  1751442 socket
fitz@ubuntu:~$ lsof -p 5267
COMMAND  PID   USER   FD   TYPE     DEVICE SIZE/OFF     NODE NAME
rsync   5267   fitz  cwd    DIR      252,0    20480  3801093 /usr/share
...      ...    ...  ...    ...      ...     ...      ...    ...
rsync   5267   fitz    4u  unix 0x00000000      0t0  1752341 socket
rsync   5267   fitz    5u  unix 0x00000000      0t0  1752342 socket
```

4.9.4. Именованные локальные сокеты

Именованные локальные сокеты, как и именованные каналы, предназначаются для взаимодействия неродственных процессов и широко распространены в системе. В листинге 4.49 показаны сокеты разных служб операционной системы и их процессы-владельцы, выясненные при помощи утилиты **fuser(1)**. Так, например, сокет **/run/wpa_supplicant/wlan0** используется для внешнего управления W:[WPA] «просите-

лем»[1] **wpa_supplicant(8)**, в частности при помощи утилиты **wpa_cli(8)**, что проиллюстрировано при помощи трассировки «сокетных» системных вызовов **socket(2)**, **connect(2)**, **send(2)** и **recv(2)**.

Листинг 4.49. Именованные локальные (файловые) сокеты

```
   fitz@ubuntu:~$ sudo find /run -type s
   [sudo] password for fitz:
?  /run/wpa_supplicant/wlan0
                                                ...
   /run/nscd/socket
                                                ...
   /run/acpid.socket
                                                ...
   /run/avahi-daemon/socket
   /run/cups/cups.sock
                                                ...
   /run/dbus/system_bus_socket
   /run/udev/control
   fitz@ubuntu:~$ sudo fuser -v /run/wpa_supplicant/wlan0 /run/udev/control /run/nscd/socket
                        ПОЛЬЗ-ЛЬ     PID ДОСТУП КОМАНДА
   /run/wpa_supplicant/wlan0:
                        root       1353 F.... wpa_supplicant
   /run/udev/control:   root        397 F.... udevd
   /run/nscd/socket:    root      31825 F.... nscd
   fitz@ubuntu:~$ sudo ls -l /run/wpa_supplicant/wlan0
☛  srwxrwx--- 1 root root 0 февр. 28 09:11 /run/wpa_supplicant/wlan0
   fitz@ubuntu:~$ sudo strace -fe socket,connect,send,recv wpa_cli status
   Selected interface 'wlan0'
   socket(PF_FILE, SOCK_DGRAM, 0)          = 3
☛  connect(3, {sa_family=AF_FILE, path="/var/run/wpa_supplicant/wlan0"}, 110) = 0
   send(3, "STATUS", 6, 0)                 = 6
   recv(3, "bssid=9c:37:f4:76:c5:68\nssid=474"..., 2047, 0) = 153
   bssid=9c:37:f4:76:c5:68
   ssid=474
   id=0
   mode=station
   pairwise_cipher=CCMP
   group_cipher=TKIP
   key_mgmt=WPA2-PSK
   wpa_state=COMPLETED
   ip_address=192.168.100.4
```

[1] Предназначен для выполнения аутентификации при подключении к беспроводным сетям.

4.9.5. Разделяемая память, семафоры и очереди сообщений

Разделяемая память

Каналы и сокеты являются удобными средствами обмена информацией между процессами, но их использование при интенсивном обмене или обмене объемными данными приводит к значительным накладными расходам. *Разделяемая память* является специализированным средством взаимодействия, имеющим минимальные издержки использования.

В листинге 4.50 при помощи команды **ipcs(1)** показаны созданные в системе сегменты разделяемой памяти, массивы семафоров и очереди сообщений — средства межпроцессного взаимодействия **svipc(7)**, «унаследованные» Linux от **W:[UNIX System V]**. Экземпляры средств **System V IPC** идентифицируются при помощи глобально уникальных ключей ❶, или локальных[1] идентификаторов ❷ **shmid** (**sh**ared **m**emory **id**entifier), **semid** (**sem**aphore **id**entifier) и **msqid** (**m**essage **q**ueue **id**entifier), и сродни файлам имеют владельцев и права доступа.

Листинг 4.50. Разделяемая память, очереди сообщений и семафоры (System V IPC)

```
fitz@ubuntu:~$ ipcs

------ Сегменты совм. исп. памяти --------
ключ ❶     shmid ❷    владелец   права      байты      nattch     состояние
0x00000000 11567105   fitz       600        393216     2          назначение
0x00000000 10944514   fitz       700        1694000    2          назначение
             ...          ...        ...          ...          ...
0x00000000 11206666   fitz       600        393216     2          назначение

------ Массивы семафоров --------
ключ   semid      владелец права nsems

------ Очереди сообщений --------
ключ   msqid      владелец права исп. байты сообщения
```

Разделяемая память **System V IPC** реализуется ядром на основе механизма страничного отображения при помощи совместного использования страничных кадров страницами разных процессов (см. рис. 4.6). При помощи системного вызова **shmget(2)** один из взаимодействующих процессов создает сегмент памяти, состоящий из

[1] Идентификаторы IPC (сродни индексным и файловым дескрипторам файлов) изменяются при пересоздании/переоткрытии экземпляра, а ключи IPC (подобно именам файлов) — нет.

страничных кадров без отображения на какой-либо файл. Впоследствии кадры этого сегмента при помощи системного вызова **shmat(2)** (**sh**ared **m**emory **at**tach) отображаются на страницы всех взаимодействующих процессов, за счет чего эти процессы и используют выделенную память совместно.

Иллюстрация применения разделяемой памяти приведена в листинге 4.51, где сегмент разделяемой памяти с идентификатором 8421380_{10} был создан (cpid, **c**reator **pid**) процессом **PID=8636** программы X-клиента **skype**, а последнее обращение к нему (lpid, **l**ast operation **pid**) осуществлялось процессом **PID=1461** программы X-сервера **Xorg(1)**. В данном конкретном случае X-клиент и X-сервер *(см. разд. 7.1)* для эффективного обмена значительными объемами растровых изображений используют расширение X-протокола **W**:**[MIT-SHM]**, основанное на использовании разделяемой памяти.

Листинг 4.51. Разделяемая память (System V IPC)

```
fitz@ubuntu:~$ ipcs -m -p

------ Shared Memory Creator/Last-op PIDs --------
shmid      владелец   cpid       lpid
11567105   fitz       16738      31382
10944514   fitz       4207       1461
            ...        ...        ...          ...          ...
❶ 8421380   fitz       8636       1461
            ...        ...        ...          ...          ...
fitz@ubuntu:~ $ dc -e 16o10i8421380p
☛ 808004
fitz@ubuntu:~$ ps -fp 16738,31382,4207,1461,8636
UID        PID  PPID  C STIME TTY          TIME CMD
root     ☛ 1461  1410  0 Feb22 tty7     01:23:02 /usr/bin/X :0 -auth /var/run/lig
fitz       4207     1  0 Feb26 ?        00:05:52 /usr/lib/virtualbox/VirtualBox
fitz     ☛ 8636     1  1 Feb26 ?        00:39:15 skype
fitz      16738     1  1 Feb27 ?        00:20:35 /usr/lib/chromium-browser/chromi
fitz@ubuntu:~$ pmap 8636
8636:   skype
            ...          ...                  ...          ...          ...
99e23000   2176K rw-s-    [ shmid=0x838005 ]
9a043000   3828K rw-s-    [ shmid=0x830006 ]
9cc2f000    484K rw-s-    [ shmid=0x828007 ]
❷ 9f541000    384K rw-s- ☛ [ shmid=0x808004 ]
            ...          ...                  ...          ...          ...
bf905000    132K rw---    [ stack ]
 total   632708K
```

```
fitz@ubuntu:~$ sudo pmap 1461
1461:   /usr/bin/X :0 -auth /var/run/lightdm/root/:0 -nolisten tcp vt7 -novtswitch ...
          ...           ...             ...             ...             ...
b5a5a000    384K rw-s-    [ shmid=0xb28003 ]
❸ b5aba000  384K rw-s- ☚  [ shmid=0x808004 ]
b5b3e000    384K rw-s-    [ shmid=0xab000a ]
          ...           ...             ...             ...             ...
bfcd9000    144K rw---    [ stack ]
 total   107220K
```

Анализ карт памяти взаимодействующих процессов при помощи команды **pmap(1)** подтверждает, что страничные кадры сегмента разделяемой **s** памяти $\mathbf{8421380}_{10}$ ($\mathbf{808004}_{16}$) были отображены на страницы обоих взаимодействующих процессов по разным виртуальным адресам ❶ и ❷.

Еще один вариант реализации разделяемой памяти, пришедший в Linux из **W:[BSD]**, основывается на «разделяемом» (shared) отображении одного файла в память разных процессов при помощи «штатного» механизма **mmap(2)**. В листинге 4.52 показан типичный пример использования совместного отображения файла **/var/cache/nscd/hosts** в память процесса **PID = 14566**, **nscd** (**n**ame service **c**ache **d**aemon) и многих других процессов, пользующихся его услугами.

Служба имен (name service) предназначена для извлечения свойств различных каталогизируемых сущностей по их имени. Например, по имени интернет-узла из каталога системы доменных имен DNS служба имен извлекает IP-адрес этого узла, и наоборот, по IP-адресу — имя. При большом количестве повторяющихся запросов к службе имен в течение промежутка времени, в который изменения запрашиваемой информации в соответствующем каталоге объектов не происходит, «бесполезные» повторные запросы к каталогу могут быть сокращены при помощи сохранения и использования предыдущих ответов (кэширования), чем и занимается демон **nscd(8)**.

В частности, ответы из DNS сохраняются процессом **nscd** в страничных кадрах памяти отображенного файла **/var/cache/nscd/hosts** ❶, а остальные процессы совместно их используют, отображая тот же файл в страницы своей памяти ❷.

Листинг 4.52. Разделяемая память (BSD)

```
fitz@ubuntu:~ $ sudo fuser -v /var/cache/nscd/hosts
                     ПОЛЬЗ-ЛЬ    PID ДОСТУП КОМАНДА
/var/cache/nscd/hosts:
                     root       14566 F...m nscd ☚
                     fitz       16724 ....m ubuntu-geoip-pr
                     fitz       16738 ....m chromium-browse
```

```
                    fitz      15895 ....m unity-scope-vid
                    fitz      8636  ....m skype
fitz@ubuntu:~$ sudo pmap 14566
14566:   /usr/sbin/nscd
             ...          ...              ...              ...          ...
❶ b1240000  32768K rw-s- ➧ /var/cache/nscd/hosts
             ...          ...              ...              ...          ...
bf8c5000    132K rw---     [ stack ]
 total   153556K
fitz@ubuntu:~$ pmap 8636
8636:   skype
             ...          ...              ...              ...          ...
❷ ab21e000    212K r--s- ➧ /var/cache/nscd/hosts
             ...          ...              ...              ...          ...
bf905000    132K rw---     [ stack ]
 total   632708K
```

Организация межпроцессного взаимодействия при помощи разделяемой памяти на основе «штатного» механизма отображения файлов в память процессов имеет один значительный недостаток. Так как изменяемые страничные кадры такого общего сегмента памяти требуют сохранения в *дисковый* файл, то производительность взаимодействия ограничивается операциями *дискового* ввода-вывода (!), а не скоростью работы оперативной памяти.

В примере с демоном **nscd(8)** этого вполне достаточно, тем более что создаваемый им кэш все равно должен сохраняться при перезагрузках. В случаях, когда взаимодействие процессов требует максимальной производительности, разделяемая память на основе отображения *дисковых* файлов в память является не лучшим механизмом. Элегантное решение данной проблемы используется в Linux-реализации разделяемой памяти стандарта **POSIX**[1], что проиллюстрировано в листинге 4.53. Псевдофайловая система **tmpfs** специально придумана для «временного» размещения файлов непосредственно в оперативной памяти, что в совокупности со «штатным» механизмом их отображения в память взаимодействующих процессов и дает желаемые характеристики производительности.

Листинг 4.53. Разделяемая память (POSIX)

```
fitz@ubuntu:~$ findmnt /dev/shm
TARGET   SOURCE FSTYPE OPTIONS
/run/shm        tmpfs  rw,nosuid,nodev,relatime
```

[1] См. **shm_overview(7)**, а также **POSIX**-семафоры **sem_overview(7)** и **POSIX**-очереди сообщений **mq_overview(7)**.

```
fitz@ubuntu:~$ ls -l /dev/shm
lrwxrwxrwx 1 root root 8 нояб 19 09:50 /dev/shm -> /run/shm
fitz@ubuntu:~$ sudo fuser -v /dev/shm/*
                     ПОЛЬЗ-ЛЬ    PID ДОСТУП КОМАНДА
            ...          ...          ...          ...          ...
/run/shm/pulse-shm-2012073053:
                     fitz      16738 ....m chromium-browse
/run/shm/pulse-shm-2024923292:
                     fitz       2907 ....m pulseaudio
                     fitz       8636 ....m skype
            ...          ...          ...          ...          ...
fitz@ubuntu:~$ pmap 8636
8636:   skype
93cee000  65540K rw-s- ☚ /run/shm/pulse-shm-2024923292
            ...          ...          ...          ...          ...
bf905000    132K rw---      [ stack ]
 total   632708K
fitz@ubuntu:~$ pmap 2907
2907:   /usr/bin/pulseaudio --start --log-target=syslog
            ...          ...          ...          ...          ...
ad4ff000  65540K r--s- ☚ /run/shm/pulse-shm-2024923292
            ...          ...          ...          ...          ...
ad5a5000     64K rw-s-  /dev/snd/pcmC0D0c
b15ec000     64K rw-s-  /dev/snd/pcmC0D0p
            ...          ...          ...          ...          ...
bfe15000    132K rw---      [ stack ]
 total   167156K
```

В примере из листинга 4.53 показано, как разделяемую память **POSIX** использует звуковая служба **pulseaudio(1)**, позволяющая осуществлять совместный (мультиплексированный) доступ приложений к устройствам аудиоввода и аудиовывода **/dev/snd/***.

Семафоры и очереди сообщений

Разделяемая память требует синхронизации действий процессов из-за эффекта гонки (race), возникающего между конкурентными, выполняющимися параллельно процессами. Для синхронизации процессов при совместном доступе к разделяемой памяти и прочим разделяемым ресурсам предназначено еще одно специализированное средство их взаимодействия — *семафоры*. В большинстве случаев семафорами **System V** (листинг 4.54) или **POSIX** пользуются многопроцессные сервисы, такие как, например, SQL-сервер **postgres(1)**, который синхронизует доступ своих параллельных процессов к сегментам их общей памяти.

Листинг 4.54. Семафоры (System V IPC)

```
fitz@ubuntu:~$ sudo ipcs -s
------ Сегменты совм. исп. памяти --------
ключ    shmid      владелец права байты nattch     состояние
           ...          ...            ...              ...             ...
0x0052e2c1 378241026  postgres   600        30482432   4
           ...          ...            ...              ...             ...

------ Массивы семафоров --------
ключ    semid      владелец права nsems
0x0052e2c1 1081344    postgres   600        17
0x0052e2c2 1114113    postgres   600        17
0x0052e2c3 1146882    postgres   600        17
0x0052e2c4 1179651    postgres   600        17
0x0052e2c5 1212420    postgres   600        17
0x0052e2c6 1245189    postgres   600        17
0x0052e2c7 1277958    postgres   600        17
```

Очереди сообщений являются средствами взаимодействия между процессами, реализующими еще один интерфейс передачи сообщений (message passing inerface), подобно каналам и сокетам. По своей природе они похожи на дейтаграммный SOCK_DGRAM режим передачи поверх именованных локальных сокетов unix(7). Основное отличие очередей сообщений от сокетов заключается в том, что время их жизни не ограничивается временем существования процессов, которые их создали. На практике очереди сообщений являются настолько малораспространенными, что их иллюстрация на среднестатистической инсталляции Linux практически невозможна.

4.10. В заключение

Выполняющиеся программы являются основными активными сущностями, инструкции которых при помощи механизма системных вызовов потребляют ресурсы, находящиеся под управлением операционной системы. Распределением этих ресурсов и занимаются подсистемы управления процессами, управления памятью и ввода-вывода, в достаточно детальной мере рассмотренные в этой главе. Основной задачей этих подсистем является организация *эффективного* распределения ресурсов между массой их потребителей — процессами и нитями.

Фактическая эффективность их работы при прочих равных будет во многом зависеть от понимания пользователем их внутренних алгоритмов и значений конфигурационных параметров этих алгоритмов, в зависимости от характеристик самих

потребителей и желаемых результатов. Например, эффективность распределения процессорного времени будет напрямую зависеть от свойств процессов и нитей, их приоритетов, их классов и процессорных привязок.

Кроме того, понимание алгоритмов работы подсистем может ответить на многие вопросы о количестве потребляемых ресурсов и дать ответ об их достаточности или недостатке. Например, важно понимать, что недостаток ресурса оперативной памяти вовсе не определяется суммарными размерами виртуальной памяти, потребленной процессами, а напрямую связан с суммарными размерами их резидентной памяти.

Навыки мониторинга и трассировки потребления ресурсов процессами помогут сделать массу полезнейших выводов о свойствах выполняющихся в них программ, что чрезвычайно полезно при разработке качественного программного обеспечения. Не лишними эти навыки будут и при выборе качественного программного обеспечения для эксплуатации в заданных условиях и с требуемыми характеристиками.

Глава 5

Программирование на языке командного интерпретатора

Командный интерпретатор является основой интерфейса командной строки, первой и главной программой, запускающейся в *интерактивном* сеансе пользователя. Кроме этого, он широко используется и в *пакетном* режиме работы, когда команды записываются в файл *сценария* «пьесы» и «проигрываются по ролям» при его запуске. В этом случае сценарий является простейшей интерпретируемой программой на языке соответствующего командного интерпретатора.

5.1. Интерпретаторы и их сценарии

На текущий момент времени существует достаточное количество диалектов языка командного интерпретатора: POSIX-совместимые **ash(1)** и **dash(1)**, авторские диалекты **W**:[Korn shell] **ksh(1)** и **W**:[Bourne shell] **bash(1)**, диалекты с синтаксисом, подобным языку программирования Си **csh(1)** и **tcsh(1)** и прочие[1].

Кроме языка командного интерпретатора, языки **W**:[Perl], **W**:[Python] или **W**:[Tcl] так же имеют свои интерпретаторы и практически всегда используются в пакетном режиме обработки своих сценариев.

Для запуска нужного интерпретатора используют универсальный комментарий **W**:[shebang], записываемый в первую строчку сценария (листинг 5.1) и указывающий полный путь к программе интерпретатора, которая вызывается для интерпретации запускаемого сценария.

Листинг 5.1. Интерпретаторы и sha-bang

```
bender@ubuntu:$ file /bin/which
which: POSIX shell script, ASCII text executable
bender@ubuntu:$ head -1 /bin/which
```

[1] Ультрасовременные **zsh(1)** или **fish(1)** хороши для интерактивной работы в системе, но для пакетной обработки команд не имеют особенного смысла.

```
#! /bin/sh
bender@ubuntu:~$ file /bin/gunzip
/bin/gunzip: Bourne-Again shell script, ASCII text executable
bender@ubuntu:~$ head -1 /bin/gunzip
#!/bin/bash
bender@ubuntu:~$ file /usr/sbin/iotop
/usr/sbin/iotop: a /usr/bin/python script, ASCII text executable
bender@ubuntu:~$ head -1 /usr/sbin/iotop
#!/usr/bin/python
bender@ubuntu:~$ file /usr/bin/lsdev
/usr/bin/lsdev: a /usr/bin/perl script, ASCII text executable
bender@ubuntu:~$ head -1 /usr/bin/lsdev
#!/usr/bin/perl
bender@ubuntu:~$ file /usr/bin/netwag
/usr/bin/netwag: a /usr/bin/wish script, ASCII text executable, with CRLF, LF line
terminators, with overstriking
bender@ubuntu:~$ head -1 /usr/bin/netwag
#!/usr/bin/wish
```

Сами сценарии представляют собой обычные текстовые файлы, подготавливаемые в любом текстовом редакторе, однако размещаются в каталогах и имеют права (см. ❶ и ❷, листинг 5.2) подобно «обычным» исполняемым W:[ELF]-программам.

Листинг 5.2. Сценарии интерпретаторов

```
bender@ubuntu:~$ cat hello.sh
#!/bin/sh

echo "Hello, World!"
bender@ubuntu:~$ hello.sh
hello.sh: команда не найдена
bender@ubuntu:~$ env
            ...          ...             ...             ...             ...
☛ PATH=/home/bender/bin:/usr/lib/lightdm/lightdm:/usr/local/sbin:/usr/local/bin:
/usr/sbin:/usr/bin:/sbin:/bin: /usr/games
            ...          ...             ...             ...             ...
bender@ubuntu:~$ pwd
/home/bender
bender@ubuntu:~$ mkdir bin
❶ bender@ubuntu:~$ mv hello.sh /home/bender/bin
bender@ubuntu:~$ hello.sh
bash: /home/bender/bin/hello.sh: Отказано в доступе
```

```
bender@ubuntu:~$ ls -la bin/hello.sh
-rw-rw-r-- 1 bender bender 32 янв.  17 15:23 bin/hello.sh
❷ bender@ubuntu:~$ chmod a+x bin/hello.sh
bender@ubuntu:~$ ls -la bin/hello.sh
-rwxrwxr-x 1 bender bender 32 янв.  17 15:23 bin/hello.sh
bender@ubuntu:~$ hello.sh
Hello, World!
```

5.2. Встроенные и внешние команды

Основное назначение любого командного интерпретатора в интерактивном или пакетном режиме — запускать команды, которые приводят или к запуску программы, «внешней» по отношению к самому интерпретатору, или к выполнению каких-либо «встроенных» действий самим командным интерпретатором (листинг 5.3). Например, команда **cd**, изменяющая текущий каталог, является встроенной (и по-другому реализована быть не может, потому что должна изменить атрибут **CWD** процесса самого интерпретатора). Команда **pwd**, наоборот, может быть внешней (и показывать при запуске атрибут **CWD** своего процесса, унаследованного в момент запуска от командного интерпретатора), но для интерпретаторов Bourne/Korn shell зачастую имеет и встроенную реализацию.

Листинг 5.3. Встроенные и внешние команды

```
? bender@ubuntu:~$ which -a cd
bender@ubuntu:~$ type -a cd
cd встроена в оболочку
bender@ubuntu:~$ which -a pwd
/bin/pwd
bender@ubuntu:~$ type -a pwd
pwd встроена в оболочку
pwd является /bin/pwd
```

5.3. Перенаправление потоков ввода-вывода

Для программирования сценариев на языке командного интерпретатора одной из важнейших его способностей является возможность организации *сохранения результатов* в файлы и возможность *считывания исходных* данных из файлов при выполнении команд.

Командный интерпретатор организует перенаправления потоков ввода-вывода внешних и встроенных команд при помощи конструкций **[n]>file** или **[n]>>file** и

[n]<file (рис. 5.1). Символы **<** и **>** естественным образом идентифицируют направление выполняемых перенаправлений — ввода или вывода, а необязательное число **n** уточняет номер перенаправляемого потока (при умалчивании **n** перенаправляется стандартный поток вывода № 1, **stdout(3)** при выводе и стандартный поток ввода № 0, **stdin(3)** при вводе). Перенаправление вывода **>** выполняется с *усечением* старого содержимого, а перенаправление **>>** — с *добавлением* к старому содержимому файла **file**.

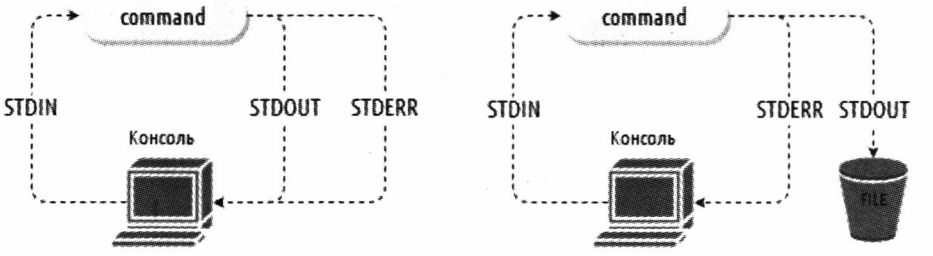


Рис. 5.1. Стандартные потоки ввода-вывода и перенаправление потока вывода STDOUT

Для перенаправления заданного потока **N** некоторой запускаемой команды **command** в файл **file** командный интерпретатор порождает дочерний процесс при помощи системного вызова **fork(2)**, затем в новом процессе открывает файл **file** при помощи системного вызова **open(2)**, перенаправляет поток **N** в открытый файл посредством системного вызова **dup2(2)** и, наконец, запускает в этом процессе программу **command** при помощи системного вызова **execve(2)**.

В примере из листинга 5.4 при помощи «разрезателя» текста **cut(1)** вырезается по разделителю (**-d**) **:** (двоеточие) **1**-е поле (**-f**) файла **passwd(5)**, содержащего свойства учетных записей пользователей, зарегистрированные в системе, а *результат сохраняется* в файле **users** при помощи перенаправления потока **STDOUT**. Затем, при помощи потокового редактора файлов **sed(1)** выводятся (**5,8p**) только (**-n**) с **5**-й по **8**-ю строки полученного файла, что дает логины с 5-го по 8-го пользователей.

Листинг 5.4. Перенаправление стандартного потока вывода

```
bender@ubuntu:~$ sed -n 5,8p /etc/passwd
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
bender@ubuntu:~$ cut -f 1 -d : /etc/passwd 1> users
bender@ubuntu:~$ sed -n 5,8p users
```

```
sync
games
man
lp
```

При выполнении некоторых команд, например как в листинге 5.5, при поиске файлов посредством **find(1)** может выводиться достаточное количество сообщений об «ошибках» доступа в тот или иной каталог, которые мешают анализировать результаты поиска. В этом случае их удобно убрать с терминала путем перенаправления потока № 2, **stderr(3)**, например, в файл **eaccess**. В результате на терминале будут отражены только имена (**-name**) файлов, соответствующие критерию имени ***.xml**, найденные в каталоге **/etc** и всех его подкаталогах.

Листинг 5.5. Перенаправление стандартного потока ошибок

```
bender@ubuntu:~$ find /etc -name '*.xml'
/etc/obex-data-server/imaging_capabilities.xml
            ...          ...              ...            ...            ...
? find: `/etc/lvm/backup': Отказано в доступе
find: `/etc/lvm/cache': Отказано в доступе
find: `/etc/cups/ssl': Отказано в доступе
            ...          ...              ...            ...            ...
/etc/ImageMagick/log.xml
bender@ubuntu:~$ find /etc -name '*.xml' 2>eaccess
/etc/obex-data-server/imaging_capabilities.xml
            ...          ...              ...            ...            ...
/etc/ImageMagick/log.xml
```

На практике часто оказывается, что мешающий вывод сообщений об «ошибках» доступа, как в листинге 5.6, оказывается ненужным вовсе, тогда его *подавляют* при помощи перенаправления потока **STDERR** в файл **/dev/null** специального всепожирающего псевдоустройства **null(4)**. Подсчитав количество строк (**-l**) в результирующем файле **empty** и зная, что одна строка в нем соответствует одному найденному ранее командой **find(1)** имени файла, размер (**-size**) которого равен 0 байт, можно получить количество «пустых» файлов в системе.

Листинг 5.6. Подавление стандартного потока ошибок

```
bender@ubuntu:~$ find / -size 0 1>empty 2>/dev/null
bender@ubuntu:~$ wc -l empty
90521 empty
```

В случаях, когда необходимо использовать содержимое файла в качестве *исходных данных* для выполнения программ, вместо ввода этих данных с терминала, как в

примере из листинга 5.7, применяют перенаправление потока **STDIN** (рис. 5.2). Таким образом, например, сохраненный в файл весьма внушительный список установленных в системе пакетов (полученный командой **dpkg(1)**) можно отправить по почте командой **mail(1)** в качестве текста сообщения.

Листинг 5.7. Перенаправление стандартного потока ввода

```
bender@ubuntu:~$ dpkg -l > installed-pacakges
bender@ubuntu:~$ mail dketov@gmail.com 0< installed-pacakges
bender@ubuntu:~$ rm -f installed-pacakges
```

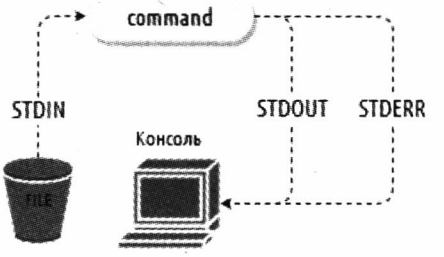


Рис. 5.2. Перенаправление потока ввода STDIN

Однако чаще на практике необходимо использовать *результат выполнения* одной команды в качестве *исходных данных* другой команды, не сохраняя эти данные в промежуточный (ненужный) файл. В этом случае используют конструкцию **command$_1$ | command$_2$ | ...**, называемую *конвейерной*[1] обработкой (рис. 5.3). Нужно заметить, конструкцией конвейерной обработки можно сцеплять более чем две команды, причем все они будут выполняться параллельно в дочерних процессах командного интерпретатора. При этом стандартные потоки ввода и вывода пар процессов будут связаны простейшим средством IPC, реализуемым ядром ОС —

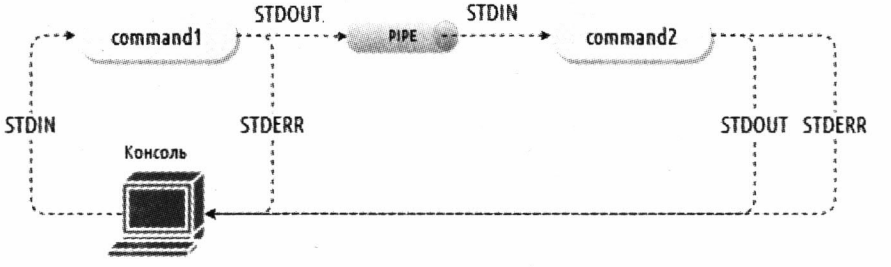


Рис. 5.3. Конвейерная обработка

[1] Или просто *конвейером*, или *каналом* (pipe), или даже *трубой*.

неименованным каналом, создаваемым системным вызовом **pipe(2)**, куда и будут перенаправлены потоки команд при помощи системного вызова **dup2(2)**.

При использовании конвейерной обработки в примере из листинга 5.8 отправка списка пакетов, установленных в системе почтовым сообщением, происходит без промежуточного файла, ровно как и подсчет количества файлов, размер которых равен 0 байт (листинг 5.9).

Листинг 5.8. Конвейерная обработка: отправка списка установленных пакетов по электронной почте

```
bender@ubuntu:~$ dpkg -l | mail dketov@gmail.com
```

Листинг 5.9. Конвейерная обработка: подсчет количества пустых файлов

```
bender@ubuntu:~$ find / -size 0 2>/dev/null | wc -l
91715
```

Листинг 5.10. Конвейерная обработка: вывод части файла

```
bender@ubuntu:~$ getent passwd | cut -f 1 -d : | head -8 | tail -5
sys
sync
games
man
lp
```

В примере из листинга 5.10 конвейерная обработка используется для последовательного получения строкового представления свойств всех доступных[1] пользовательских учетных записей при помощи **getent(1)**, затем вырезания из этих строк при помощи **cut(1)** 1-го поля (**-f**) посредством разделителя **:** (**-d**), далее выбора 8-ми первых строк командой **head(1)**, а потом выбора 5-ти последних строк предыдущего результата командой **tail(1)**.

Листинг 5.11. Конвейерная обработка: генерация трех случайных восьмизначных паролей

```
bender@ubuntu:~$ tr -dc a-zA-Z0-9 </dev/urandom | fold -w 8 | head -3
QCh67VnC
3nwAG7P9
51IS1Lxz
```

[1] Учетные записи пользователей, зарегистрированных непосредственно в системе (локально), хранятся в файле **/etc/passwd**, см. **passwd(5)**. Все доступные в системе учетные записи, в том числе, например, централизованные в (сетевых) LDAP-каталогах организации, могут быть получены при помощи команды **getent(1)** благодаря службе имен.

В листинге 5.11 конвейерная обработка в совокупности с перенаправлением потока **STDIN** используется для генерации трех случайных строк, которые могут использоваться, например, в качестве начальных паролей пользовательских учетных записей. Для этого из специального файла псевдоустройства **urandom(4)**, генерирующего случайную последовательность байтов, отбираются командой транслитерации **tr(1)** только строковые и заглавные буквы латинского алфавита и цифры **a-zA-Z0-9** путем удаления (**-d**) из выходного потока символов, не (**-c**) попавших в заданный набор. Затем команда **fold(1)** разбивает свой входной поток по 8 символов в строке выходного потока, после чего команда **head(1)** выбирает только 3 первые строки.

Листинг 5.12. Конвейерная обработка: удаление пустых файлов

```
bender@ubuntu:~$ find /tmp -user bender -size 0 -print0 2>/dev/null | xargs -0 rm -f
```

В листинге 5.12 конвейерная обработка вместе с командой **xargs(1)** используется для того, чтобы во временном каталоге **/tmp** удалить все файлы, принадлежащие (**-user**) пользователю **bender**, размер (**-size**) которых равен нулю. Сначала при помощи команды **find(1)** производится поиск файлов согласно указанным критериям и вывод списка их имен с разделением имен нулевым символом (**-print0**). Затем команда **xargs(1)** последовательно «применяет» команду **rm(1)** безусловного (**-f**) удаления файла для каждого имени из списка. Следует отметить, что такая конструкция работает с абсолютно любыми именами файлов, включая файлы с пробелами, табуляциями, переводами строк и другими управляющими символами в имени, т. к. имена файлов в списке разделяются нулевым символом, а он единственный запрещен к использованию в именах файлов.

Аналогично, в листинге 5.13, используя конвейер, при помощи команды **groups(1)** можно вывести групповое членство всех пользователей, доступных в системе.

Листинг 5.13. Конвейерная обработка: просмотр группового членства пользователей системы

```
bender@ubuntu:~$ getent passwd | cut -f 1 -d : | xargs groups
          ...          ...          ...          ...          ...
finn : finn candy
jake : jake
bubblegum : bubblegum candy
          ...          ...          ...          ...          ...
fitz : fitz sudo
skillet : skillet
bender : bender
          ...          ...          ...          ...          ...
```

С помощью конвейеров и универсального спискового «применяtеля» команд **xargs(1)** можно организовать (листинг 5.14) параллельный запуск (**-P**) упаковки целого списка отобранных командой **find(1)** файлов, использовав параллельный упаковщик **pbzip2(1)**, в несколько упаковочных нитей (**-p**) на каждый файл. В результате получим запускаемые парами процессы по две активные нити на каждый, что эффективно загружает четырехъядерный процессор в течение упаковки всего списка файлов.

Листинг 5.14. Конвейерная обработка: параллельная упаковка ISO-образов

```
bender@ubuntu:~$ find . -name '*.iso' | xargs -P 2 pbzip2 -p2 &
bender@ubuntu:~$ ps f
  PID TTY      STAT   TIME COMMAND
 4723 pts/1    S      0:00 -bash
 4903 pts/1    S      0:00  \_ xargs -P 2 -n1 pbzip2 -p2
 4904 pts/1    Sl     0:08  |   \_ pbzip2 -p2 ./dvd.iso
 4905 pts/1    Sl     0:08  |   \_ pbzip2 -p2 ./plan9.iso
 4856 pts/1    R+     0:00  \_ ps f
bender@ubuntu:~$ ps -fL
UID        PID  PPID   LWP  C NLWP STIME TTY          TIME CMD
bender    4723  4722  4723  0    1 01:18 pts/1    00:00:00 -bash
bender    4903  4723  4903  0    1 01:21 pts/1    00:00:00 xargs -P 2 -n1 pbzip2
            ...           ...             ...              ...              ...
bender    4904  4903  4910 99    6 01:21 pts/1    00:00:01 pbzip2 -p2 ./dvd.iso
bender    4904  4903  4913 99 ☚  6 01:21 pts/1    00:00:01 pbzip2 -p2 ./dvd.iso
bender    4904  4903  4914  0    6 01:21 pts/1    00:00:00 pbzip2 -p2 ./dvd.iso
            ...           ...             ...              ...              ...
bender    4905  4903  4911 99 ☚  6 01:21 pts/1    00:00:01 pbzip2 -p2 ./plan9.is
bender    4905  4903  4912 99    6 01:21 pts/1    00:00:01 pbzip2 -p2 ./plan9.is
bender    4905  4903  4915  0    6 01:21 pts/1    00:00:00 pbzip2 -p2 ./plan9.is
bender    4916  4723  4916  0    1 01:21 pts/1    00:00:00 ps -fL
```

5.4. Подстановки командного интерпретатора

Еще одной важной способностью командного интерпретатора, необходимой для разработки сценариев, является возможность подставлять различные вычисляемые значения в качестве аргументов запускаемых команд.

5.4.1. Подстановки имен файлов

Простейшими вычисляемыми командным интерпретатором значениями являются имена файлов, соответствующие некоторым шаблонам. Язык шаблонных выраже-

ний крайне прост и основывается на понятии *метасимволов*, т. е. символов со специальными значениями (табл. 5.1).

Таблица 5.1. Шаблонные метасимволы

Метасимвол	Значение
?	Любой один символ
*	Любое количество любых других символов
[ab...z]	Любой символ из набора a, b, ..., z
[!ab...z] или [^ab...z]	Любой символ НЕ из набора a, b, ..., z
~	Домашний каталог пользователя

Использование метасимволов в команде интерпретатора заставляет его искать файлы, чьи имена соответствуют составленному шаблонному выражению, и *подставлять* их вместо самих шаблонных выражений.

В примере из листинга 5.15 шаблонное выражение **y*** используется для подстановки имен файлов, начинающихся с буквы **y**, за которой следует любое количество любых символов. Выражение **???.*** означает имена файлов, которые содержат три любых символа, потом символ **.** (точка), за которым следует любое количество любых символов. Сама работа подстановок интерпретатора может быть отслежена в режиме трассировки, который включается ❶ командой **set -x** (а выключается, как ни странно, командой **set +x**). В этом режиме интерпретатора видно, что выражение **[0-9]***, означающее имена файлов, начинающиеся с цифры, сначала вычисляется ❶ до соответствующего списка имен, который подставляется вместо самого шаблонного выражения, и только потом выполняется команда ls(1), в аргументах которой оно было использовано. Аналогично вычисляется ❷ и подставляется шаблонное выражение **[!a-z0-9]***, означающее имена файлов, начинающихся с символов, не являющихся ни буквой, ни цифрой. Таким образом, ни одна команда самостоятельно не вычисляет подстановки имен файлов, а пользуется результатами вычислений так, как будто они были заданы непосредственно в качестве ее аргументов.

Листинг 5.15. Просмотр файлов man-страниц по критерию имени

```
bender@ubuntu:~$ cd /usr/share/man/man1
bender@ubuntu:/usr/share/man/man1$ ls y*
yacc.1.gz       yelp.1.gz  youtube-dl.1.gz     yuvsplittoppm.1.gz
```

```
ybmtopbm.1.gz  yes.1.gz   ypdomainname.1.gz  yuvtoppm.1.gz
bender@ubuntu:/usr/share/man/man1$ ls ???.*
7za.1.gz  cpp.1.gz  ftp.1.gz  ldd.1.gz  moc.1.gz  s2p.1.gz  tee.1.gz  wmc.1.gz
a2p.1.gz  cut.1.gz  g++.1.gz  lft.1.gz  mrd.1.gz  sar.1.gz  tex.1.gz  wrc.1.gz
          ...       ...       ...       ...       ...       ...       ...
cmp.1.gz  erb.1.gz  kvm.1.gz  mft.1.gz  rev.1.gz  tbl.1.gz  vlc.1.gz
col.1.gz  fmt.1.gz  lcf.1.gz  mmd.1.gz  rsh.1.gz  tcc.1.gz  who.1.gz
bender@ubuntu:/usr/share/man/man1$ set -x ❶
bender@ubuntu:/usr/share/man/man1$ ls -l [0-9]*
+ ls --color=auto -l 2to3.1.gz 2to3-2.7.1.gz 2to3-3.2.1.gz 411toppm.1.gz 7z.1.gz 7za.1.gz ❶
lrwxrwxrwx 1 root root   13 июня  18  2013 2to3.1.gz -> 2to3-2.7.1.gz
-rw-r--r-- 1 root root  563 февр. 28  2014 2to3-2.7.1.gz
           ...          ...          ...          ...          ...
-rw-r--r-- 1 root root 2013 февр. 18  2012 7z.1.gz
-rw-r--r-- 1 root root 2032 февр. 18  2012 7za.1.gz
bender@ubuntu:/usr/share/man/man1$ ls [!a-z0-9]*
+ ls --color=auto -l '[.1.gz' ❷
lrwxrwxrwx 1 root root 9 нояб. 20  2012 [.1.gz -> test.1.gz
```

В режиме трассировки командного интерпретатора видна еще и подстановка псевдонимов, которая заменяет «псевдоним» команды пользователя, например **ls** (листинг 5.16), ее «настоящим значением».

Листинг 5.16. Псевдонимы командного интерпретатора

```
bender@ubuntu:~$ alias
          ...          ...          ...          ...          ...
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -alF'
alias ls='ls --color=auto'
```

5.4.2. Подстановки параметров

Более важным видом подстановок командного интерпретатора являются подстановки значений *параметров* — специальных сущностей, имеющих эти самые значения. Различают три типа параметров — *переменные*, *позиционные* параметры и *специальные* параметры. Значения переменных могут быть изменены в любой момент времени при помощи операции присваивания, тогда как значения позиционных параметров задаются один раз при их инициализации, а значения специальных параметров вычисляются предопределенным образом в зависимости от окружающей среды и обстоятельств.

Переменные — именованные параметры

Самый простой тип параметров — это переменные командного интерпретатора, и их «глобальное» подмножество — переменные окружения. *Переменные окружения (см. главу 2)* параметризируют пользовательский сеанс работы в системе и видны абсолютно всем командам сеанса, тогда как переменные командного *интерпретатора* видны только ему.

В примере из листинга 5.17 командами **env(1)** и **set** показано количество переменных окружения и переменных командного интерпретатора соответственно. При помощи операции присваивания **VARIABLE=value** вводится новая переменная интерпретатора ❶, а посредством команды **export** ❷ эта переменная «экспортируется» в переменные окружения, что видно во флагах объявления переменных на выводе команды **typeset**. Стоит отметить, что команда **set** интерпретатора **bash(1)** по умолчанию выводит не только объявленные переменные, но и объявленные функции, что отключается ❶ установкой POSIX-совместимого режима работы.

Листинг 5.17. Переменные интерпретатора и переменные окружения

```
❶ bender@ubuntu:~$ set -o posix
  bender@ubuntu:~$ env | wc -l; set | wc -l; typeset -p VARIABLE
  23
  119
  -bash: typeset: VARIABLE: не найден
❶ bender@ubuntu:~$ VARIABLE=value
  bender@ubuntu:~$ env | wc -l; set | wc -l; typeset -p VARIABLE
  23
☛ 120
  declare -- VARIABLE="value"
❷ bender@ubuntu:~$ export VARIABLE
  bender@ubuntu:~$ env | wc -l; set | wc -l; typeset -p VARIABLE
☛ 24
  120
  declare -x VARIABLE="value"
  bender@ubuntu:~$ set +o posix
```

Присваивание значения «новой» переменной, как показано в листинге 5.18, устанавливает именно переменную командного *интерпретатора*, которая не видна «глобально» в сеансе пользователя и требует экспортирования для работы в качестве переменной *окружения*.

Листинг 5.18. Присвоение значений переменным

```
bender@ubuntu:~$ LANG=ko_KR.utf8
bender@ubuntu:~$ date
!  2015. 12. 19. (토) 11:11:28 MSK
bender@ubuntu:~$ TZ=Europe/Stockholm
bender@ubuntu:~$ date
?  2015. 12. 19. (토) 11:11:37 MSK
bender@ubuntu:~$ declare -p LANG TZ
declare -x LANG="ko_KR.utf8"
declare -- TZ="Europe/Stockholm"
bender@ubuntu:~$ export TZ
!  2015. 12. 19. (토) 09:11:49 CET ☚
```

Так как имена и значения переменных располагаются в оперативной памяти процесса командного интерпретатора, то их *время жизни* и *область видимости* ограничиваются процессом их интерпретатора. Это означает, что переменные теряют свои значения при завершении интерпретатора, в котором были установлены, а переменные, установленные в одном интерпретаторе, не *видны* в других интерпретаторах. Только установленные переменные *окружения* экспортируются (копируются) процессам других команд при запуске, но любые их последующие изменения происходят отдельно и локально в каждом из процессов.

Для разового экспорта переменных окружения предназначается утилита **env(1)**, хотя в большинстве диалектов языка командного интерпретатора используется просто оператор присвоения значения переменным перед запускаемой командой (листинг 5.19).

Листинг 5.19. Временное присвоение значений переменным окружения

```
bender@ubuntu:~$ env LANG=be_BY.utf8 TZ=Europe/Minsk ls -l
итого 222976
-rw-r--r-- 1 bender bender 114650029 ☚ Сне.  12 10:36 dvd.iso.gz
-rw-r--r-- 1 bender bender 113666114   Сне.  14 00:02 plan9.iso.gz
bender@ubuntu:~$ LANG=fi_FI.utf8 TZ=Europe/Helsinki ls -l
итого 222976
-rw-r--r-- 1 bender bender 114650029 ☚ joulu 12 09:36 ☚dvd.iso.gz
-rw-r--r-- 1 bender bender 113666114   joulu 13 23:02 plan9.iso.gz
bender@ubuntu:~$ typeset -p LANG
```

```
declare -x LANG="ru_RU.UTF-8"
-bash: typeset: TZ: не найден
bender@ubuntu:~$ ls -l
итого 222976
-rw-r--r-- 1 bender bender 114650029 ☛ дек.  12 10:36 dvd.iso.gz
-rw-r--r-- 1 bender bender 113666114   дек.  14 00:02 plan9.iso.gz
```

Для подстановки значения параметров при выполнении команд используется конструкция **$parameter**, где символ **$** является требованием подстановки, а **parameter** — идентификатором параметра, например, *именем* переменной, *номером* позиционного параметра или *символом* специального параметра.

В режиме трассировки команд интерпретатора из листинга 5.20 видно, что командный интерпретатор вместо указанной переменной подставляет ее значение до выполнения команды, после чего команда выполняется так, как будто ее аргументы были заданы непосредственно подставленными значениями.

Листинг 5.20. Подстановка значений переменных

```
bender@ubuntu:/tmp$ env
SSH_AGENT_PID=3260
SHELL=/bin/bash
USER=bender
          ...          ...          ...          ...          ...
USERNAME=bender
MAIL=/var/mail/bender
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
PWD=/tmp
LANG=ru_RU.UTF-8
          ...          ...          ...          ...          ...
HOME=/home/bender
          ...          ...          ...          ...          ...
LOGNAME=bender
bender@ubuntu:/tmp$ set -x
bender@ubuntu:/tmp$ file $SHELL
☛+ file /bin/bash
/bin/bash: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked (uses
shared libs), for GNU/Linux 2.6.24, BuildID[sha1]=0xf199a4a89ac968c2e0e99f2410600b9d7e995187,
stripped
bender@ubuntu:/tmp$ ps p $SSH_AGENT_PID
☛ + ps p 3260
  PID TTY      STAT   TIME COMMAND
 3260 ?        Ss     0:02 /usr/bin/ssh-agent /usr/bin/dbus-launch ...
bender@ubuntu:/tmp$ cd $HOME
```

```
☛ + cd /home/bender
  bender@ubuntu:~$
```

Позиционные параметры

Позиционные параметры используются сценариями командного интерпретатора для передачи их фактических параметров и идентифицируются номерами, нумерующими аргументы сценария, указанные при его запуске. Например, в листинге 5.21 показан достаточно простой сценарий **ssh-copy-id(1)**, использующийся для копирования публичного ключа на сервер службы SSH *(см. разд. 6.4.1)* для последующей «ключевой» аутентификации вместо «парольной». В тексте сценария используются подстановки **$1** и **$0**, символизирующие первый формальный аргумент ①, переданный в сценарий при его запуске, и нулевой формальный аргумент ⓪ — имя самого сценария. При последующем запуске сценария в режиме трассировки (явно указывая интерпретатор **sh** с опцией **-x**) ему передаются фактические аргументы ❼ и ❶, которые соответствующим образом подставляются в командах, выполняемых интерпретатором согласно тексту сценария.

Листинг 5.21. Подстановка значений позиционных параметров

```
bender@ubuntu:~$ file /usr/bin/ssh-copy-id
/usr/bin/ssh-copy-id: POSIX shell script, ASCII text executable
bender@ubuntu:~$ cat /usr/bin/ssh-copy-id
#!/bin/sh
   ...          ...          ...          ...          ...          ...
                ①↴
if [ "$#" -lt 1 ] || [ "$1" = "-h" ] || [ "$1" = "--help" ]; then
           ⓪↴
  echo "Usage: $0 [-i [identity_file]] [user@]machine" >&2
  exit 1
fi
   ...          ...          ...          ...          ...          ...

                     ⓪↴           ❶↴
bender@ubuntu:~$ sh -x /usr/bin/ssh-copy-id -h
                     $0↥          $1↥
+ ID_FILE=/home/bender/.ssh/id_rsa.pub
❶☛+ [ -i = -h ]
+ [ x != x ]
+ eval
+ [ -z  ]
```

```
+ [ -r /home/bender/.ssh/id_rsa.pub ]
+ GET_ID=cat "/home/bender/.ssh/id_rsa.pub"
+ eval cat "/home/bender/.ssh/id_rsa.pub"
+ cat /home/bender/.ssh/id_rsa.pub
+ [ -z ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDenTy5ffPzKq9L1/Mo1SfEOySDQDctJOquHLUjGL+CF6weuAsBzS+Z3I8aeDq0G
gbYgSNG2LVaWUFhNMUD3mXDOGAZloVAqRX+8lucIVF0XvSCo/RiNieaYiEZxw+MgMZy2tuZJIvaECO6+9g/V/gLc5Mkp+
rGmnQ6aBvaanGq2EMLlFEmc+GCJ/aS6OHP1z4inqAbUaUqsUVhKNF4nPN6gYwnBTI34HwtNVp6BMcmUdl+bbri5yrE8NN
lT1zXldkp2JiRVU4cXUUJ0IVId0Dz6l957X4yDo1oHLocXMNvK+lauga6+bErhWzdHeZAg7JxJ9LSxu8cqCnPzcMBtF2V
bender@ubuntu ]
+ [ 1 -lt 1 ]
❶+ [ -h = -h ]
❶+ echo Usage: /usr/bin/ssh-copy-id [-i [identity_file]] [user@]machine
Usage: /usr/bin/ssh-copy-id [-i [identity_file]] [user@]machine
+ exit 1
```

Специальные параметры

Специальные параметры идентифицируются символами **#**, **?**, **!**, **@**, **$** и вычисляются в зависимости от окружения и обстоятельств выполнения сценария в заранее предопределенные значения. Так, например, в листинге 5.22 показана подстановка параметра **$**, вычисляющегося в идентификатор процесса самого командного интерпретатора, а в листинге 5.23 — подстановка параметра **!**, вычисляющегося в идентификатор процесса последнего асинхронного (параллельного с самим идентификатором) дочернего процесса. В режиме трассировки команд видно, что, как и любые другие подстановки, они выполняются до запуска команд, в которых были использованы. В результате сами команды выполняются так, словно параметры их были заданы непосредственным образом.

Листинг 5.22. PID текущего процесса

```
bender@ubuntu:~$ set -x
bender@ubuntu:~$ ps up $$
+ ps up 32649
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
bender   32649  0.1  0.1  13536  8604 pts/1    S    10:28   0:00 -bash
```

Листинг 5.23. PID последнего дочернего асинхронного процесса

```
bender@ubuntu:~$ set -x
bender@ubuntu:~$ dd if=/dev/dvd of=dvd.iso &
[1] 399
+ dd if=/dev/dvd of=dvd.iso
```

```
bender@ubuntu:~$ ps up $!
+ ps up 399
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
bender     399 14.0  0.0   5604   588 pts/1    D    10:36   0:00 dd if=/dev/dvd
```

Используя специальный параметр **?**, можно узнать статус завершения (код возврата) программы, выполнившейся последней. При этом *нулевой* статус завершения символизирует успешное ее выполнение, а любое другое, *отличное от нулевого* значение есть «номер» ошибки. В листинге 5.24 команда **which(1)** завершилась с успехом при поиске программы, запускающейся по внешней команде **ls**, и неуспехом — при поиске программы, соответствующей встроенной команде **cd**.

Листинг 5.24. Статус завершения процесса/программы

```
bender@ubuntu:~$ which ls
/bin/ls
bender@ubuntu:~$ echo $?
0
bender@ubuntu:~$ which cd
bender@ubuntu:~$ echo $?
1
```

5.4.3. Подстановки вывода команд

Еще одним видом подстановок, выполняемых командным интерпретатором, являются подстановки *вывода команд*. Конструкции вида **$(command)** (или ее более старая форма **`command`**) используется для подстановки результата вывода команды **command** на поток **STDOUT** в место ее использования.

На рис. 5.4 показана широко распространенная подстановка *результата* работы команды $\textbf{command}_2$ в качестве *аргументов*[1] команды $\textbf{command}_1$, используемая в виде $\textbf{command}_1$ $\textbf{\$(command}_2\textbf{)}$. Так, например, можно, зная полный путь (который можно получить при помощи **which(1)**) к определенной утилите, узнать при помощи **dpkg(1)** пакет программного обеспечения, которому он принадлежит, что показано в примере из листинга 5.25 в режиме трассировки.

[1] Для сравнения, при конвейерной обработке *результаты* выполнения одной команды передаются в качестве *исходных* данных другой программе.

Листинг 5.25. В каком пакете утилита?

```
bender@ubuntu:~$ which lspci
/usr/bin/lspci
bender@ubuntu:~$ set -x
❶ bender@ubuntu:~$ dpkg -S `which lspci`
☛ ++ which lspci
☛ + dpkg -S /usr/bin/lspci
pciutils: /usr/bin/lspci
❷ bender@ubuntu:~$ dpkg -S $(which lspci) | cut -f 1 -d : | xargs dpkg -s
Package: pciutils
          ...          ...          ...          ...          ...
Description: Linux PCI Utilities
 This package contains various utilities for inspecting and setting of
 devices connected to the PCI bus.
Homepage: http://atrey.karlin.mff.cuni.cz/~mj/pciutils.shtml
Original-Maintainer: Anibal Monsalve Salazar <anibal@debian.org>
```

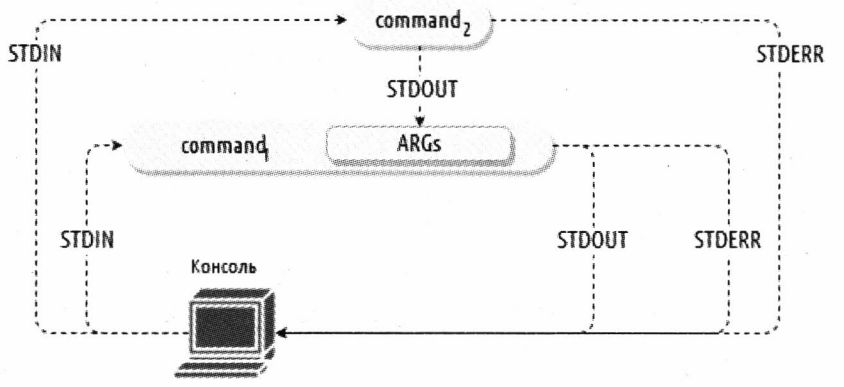


Рис. 5.4. Подстановка вывода команд

Стоит заметить, что команда **xargs(1)** выполняет подобную работу, что и подстановка вывода команд, т. е. передает в качестве *аргументов* одной команды результаты вывода другой. В примере ❷ из листинга 5.25 используются оба варианта, где подстановка вывода **which(1)** выполняется командным интерпретатором, а вывод первой команды **dpkg(1)** подставляется при помощи **xargs(1)**, будучи предварительно отфильтрованным при помощи **cut(1)**.

Подстановка вида **$(command)**, в отличие от **`command`**, может без особых ухищрений многократно вкладываться сама в себя. В примере из листинга 5.26 имя текущего пользователя ①, полученное при помощи команды **id(1)**, подставляется в качестве

аргумента команды поиска find(1) для поиска его пустых файлов ②, список которых подставляется в качестве аргумента команды rm(1) для их удаления.

Листинг 5.26. Удаление пустых файлов пользователя

```
bender@ubuntu:~$ set -x
                                    ①↴
bender@ubuntu:~$ rm -f $(find /tmp -user $(id -un) -size 0)
                       ②↴
+++ id -un
❶ ++ find /tmp -user bender -size 0
find: `/tmp/.org.chromium.Chromium.N29MCl': Отказано в доступе
find: `/tmp/.wine-1000': Отказано в доступе
          ...          ...          ...          ...          ...
❷ + rm -f /tmp/wireshark_eth0_20151204134033_OMZ6wa /tmp/wireshark_eth0_20151204134033_...
```

Важно понимать, что подстановки заменяются их вычисляемыми значениями в любом месте команды, как, например, в листинге 5.27 — в части имени архива, получаемого при помощи команды tar(1) из файлов каталога **~/.config**.

Листинг 5.27. Архив с датой создания в имени

```
bender@ubuntu:~$ set -x
bender@ubuntu:~$ tar cjf dotconfig-$(date +%F).tbz2 ~/.config
+ tar cjf dotconfig-2015-12-13.tbz2 /home/bender/.config
tar: Удаляется начальный `/' из имен объектов
tar: /home/bender/.config/chromium: файл изменился во время чтения
tar: /home/bender/.config/geeqie/.command: сокет проигнорирован
```

5.4.4 Подстановки арифметических выражений

Подстановки вывода команд и параметров являются для программирования на языке командного интерпретатора практически самыми важными конструкциями. Например, используя операцию присвоения, подстановку вывода и внешнюю команду expr(1), предназначенную для вычисления *арифметических* выражений, можно вычислять значения одних переменных на основе других, как показано в примере ❶ из листинга 5.28. Нужно отметить, что аргументы команды expr(1) отделяются пробелами, как и у любой другой команды, например ls(1).

Запуск внешних команд влечет за собой накладные расходы в виде системных вызовов fork(2) и execve(2), поэтому во многих диалектах языка командного интерпретатора реализованы аналогичная встроенная команда *арифметических* вычисле-

ний **let** и *арифметическая* подстановка в виде стандартной POSIX-конструкции **$((expression))**, а в некоторых еще и в нестандартном виде **$[expression]**. Использование команды **let** в примере ❷ из листинга 5.28, наоборот, исключает пробелы в арифметическом выражении, т. к. оно целиком является одним ее аргументом. В случае использования арифметической подстановки в примере ❸ пробельные символы могут быть применены произвольным образом.

Листинг 5.28. Арифметические действия командного интерпретатора

```
❶ bender@ubuntu:~$ CIRCLE=`expr 2 * $RADIUS * 355 / 113`
❷ bender@ubuntu:~$ let CIRCLE=2*RADIUS*355/113
❸ bender@ubuntu:~$ CIRCLE=$((2 * RADIUS * 355/113))
```

Подстановки арифметических выражений полезно применять для пересчета значений в аргументах разных команд, использующих «неудобные» единицы измерения. Так, например, в листинге 5.29 при поиске библиотек больше 10 Мбайт нужно задавать количество байтов (символов), которое и составит искомые десять килобайт килобайтов.

Листинг 5.29. Библиотеки больше 10 Мбайт

```
bender@ubuntu:~$ set -x
bender@ubuntu:~$ find /lib /usr/lib -name '*.so.*' -size +$((1024*1024*10))c
+ find /lib /usr/lib -name '*.so.*' -size +10485760c
/usr/lib/libicudata.so.48.1.1
/usr/lib/libwireshark.so.1.1.7
...          ...          ...          ...          ...
/usr/lib/i386-linux-gnu/libQtGui.so.4.8.1
/usr/lib/libwebkitgtk-1.0.so.0.13.4
```

Пример из листинга 5.30 содержит две вложенные друг в друга подстановки — подстановку вывода команды **date(1)** в арифметическую подстановку. Команда **find(1)** при поиске модифицированных (**-mtime**, modification time) файлов использует единицы измерения «дней назад», и при поиске файлов в каталоге **/var/lib/dpkg/info**[1], изменившихся в период с 20.12.2015 по 25.12.2015 приходится рассчитывать количество «дней назад» до этих дат. Для расчета используются команда **date(1)** и **W:[UNIX-время]**, вычисляющая как целое количество секунд, прошедших от 01.01.1970 00:00 UTC до нужного момента, и арифметическая подстановка, вычис-

[1] Это база данных пакетного менеджера **dpkg(1)** об установленных в систему пакетах программного обеспечения.

ляющая количество «дней назад». Сначала вычисляется количество «секунд назад» как разница между текущим UNIX-временем и UNIX-временем указанной даты, после чего вычисляется количество «дней назад» путем деления на 60 секунд в минуте, 60 минут в часе и 24 часа в дне. При помощи команды **stat(1)** подтверждается правильность поиска путем анализа даты модификации найденных файлов.

Листинг 5.30. Пакеты программного обеспечения, установленные в определенный период

```
bender@ubuntu:~$ set -x
bender@ubuntu:~$ find /var/lib/dpkg/info -name '*.list' \ ↵
> -mtime +$[ ($(date +%s) - $(date -d 2015-12-25 +%s)) / (24*60*60) ] \ ↵
> -mtime -$[ ($(date +%s) - $(date -d 2015-12-20 +%s)) / (24*60*60) ] | ↵
> xargs stat -c %y:%n
+ xargs stat -c %y:%n
++ date +%s
++ date -d 2015-12-25 +%s
++ date +%s
++ date -d 2015-12-20 +%s
+ find /var/lib/dpkg/info -name '*.list' -mtime +1 -mtime -6
2015-12-24 15:59:50.697681116 +0300:/var/lib/dpkg/info/flex.list
2015-12-20 18:02:51.656846947 +0300:/var/lib/dpkg/info/bison.list
```

В этом примере строка первой *команды* конвейера для удобства ввода разбивается на три *экранные строки* посредством *экранирования* (см. ниже) управляющего символа перевода строки ↵ при помощи метасимвола одиночного экранирования ****. В этом случае командный интерпретатор каждую последующую экранную строку предваряет «вторичным» приглашением **PS2**, а не «первичным» приглашением **PS1**. Символ перевода строки после символа конвейера **|** экранирования не требует, т. к. командному интерпретатору очевидна необходимость ввода второй команды конвейера.

5.5. Экранирование

Перед запуском введенной команды интерпретатор анализирует ее на наличие так называемых *метасимволов*, т. е. символов, имеющих *специальное* значение. Так, например, *пробел* отделяет аргументы команды друг от друга, *шаблонные* выражения на основе **?**, *****, и **[]** вычисляются в имена файлов, *доллар* **$** активирует подстановку команд, параметров или арифметических выражений, а символы **<**, **>**, **|** — перенаправления потоков. В определенных командах требуется использовать *литеральное* (буквальное) значение метасимволов, как, например, в листинге 5.31 для

манипулирования файлами, в именах которых содержатся пробелы. Для отмены специального значения метасимволов используется их *экранирование* при помощи конструкций **\m**, **'$m_1m_2m_3...m_N$'** или **"$m_1m_2m_3...m_N$"**, «отменяющих» метасимволы **m**. **m_1**, ..., **m_N**.

Листинг 5.31. Экранирование пробельных символов

```
bender@ubuntu:/usr/share/onboard/themes$ ls -l
итого 88
            ...          ...             ...              ...              ...
-rw-r--r-- 1 root root 2637 февр. 16  2012 Classic Onboard.colors
-rw-r--r-- 1 root root  436 февр. 16  2012 Classic Onboard.theme
            ...          ...             ...              ...              ...
❶ bender@ubuntu:/usr/share/onboard/themes$ file Classic Onboard.colors
? Classic:         ERROR: cannot open `Classic' (No such file or directory)
? Onboard.colors: ERROR: cannot open `Onboard.colors' (No such file or directory)
❷ bender@ubuntu:/usr/share/onboard/themes$ file Classic\ Onboard.colors
Classic Onboard.colors: XML document text
❸ bender@ubuntu:/usr/share/onboard/themes$ file "Classic Onboard.colors"
Classic Onboard.colors: XML document text
❹ bender@ubuntu:/usr/share/onboard/themes$ file 'Classic Onboard.colors'
Classic Onboard.colors: XML document text
```

В примере из листинга 5.31 показано экранирование символа пробела в имени файла, который привел ❶ к разбиению имени файла на два аргумента команды **file(1)**, ни один из которых естественно не является именем какого-либо файла. В примере ❷ используется одиночное экранирование пробела при помощи **\m**, а в примерах ❸ и ❹ — множественное экранирование всех символов в кавычках **''** и **""**, включая пробел.

В менее тривиальных случаях, как в примере ❶ из листинга 5.32, попытка выполнения команды **find(1)** с вполне валидными аргументами приводит к неожиданным и странным результатам. В режиме трассировки команд интерпретатора становится очевидно, что шаблонное выражение ***.gz**, предназначавшееся *самой* команде поиска файлов, было подставлено интерпретатором *до* запуска команды, что привело ее в недоумение. Для правильной передачи шаблонных выражений самим командам их следует экранировать, как в примере ❷.

Листинг 5.32. Экранирование шаблонных выражений

```
❶ bender@ubuntu:~$ find . -name *.gz
?🗣 find: все пути должны предшествовать выражению: plan9.iso.gz
```

```
Использование: find [-H] [-L] [-P] [-Оуровень] [-D help|tree|search|stat|rates|opt|exec]
[путь...]❶ [выражение]❷
bender@ubuntu:~$ set -x
bender@ubuntu:~$ find . -name *.gz
+ find . -name dvd.iso.gz plan9.iso.gz
     ❶↑   ❷↑            ?↑
find: все пути должны предшествовать выражению: plan9.iso.gz
Использование: find [-H] [-L] [-P] [-Оуровень] [-D help|tree|search|stat|rates|opt|exec]
[путь...] [выражение]
❷ bender@ubuntu:~$ find . -name "*.gz"
+ find . -name '*.gz'
./dvd.iso.gz
./plan9.iso.gz
```

Аналогично, в примере ❶ из листинга 5.33 при попытке скачивания с Web-сервера ресурса со сложным W:[URL] при помощи команды **wget(1)** командный интерпретатор постарался разбить команду на список заданий, встретив метасимвол **&**, формирующий асинхронные задания *(см. разд. 4.8.1)*. Для правильной передачи строк, содержащих метасимволы, их следует экранировать, как в примере ❷.

Листинг 5.33. Экранирование символов списка команд &

```
❶ bender@ubuntu:~$ wget https://r4---sn-xjpm-4g5e.googlevideo.com/videoplayback?signature=D017
06B19078430C4AB2E7E0E8A1C03A685048F5.53412B263BA6F90E9DB1ADD7F175E6AB4F7F18A3&expire=14511669
72&pl=24&mv=u&mt=1451145210&ms=au&id=o-ANXmoZGwigBoC32QKD0mA9iQcFzRyI8UhFWlgl5Wll8c&mn=sn-xjp
m-4g5e&mm=31&key=yt6&ipbits=0&mime=video%2Fmp4&source=youtube&sver=3&dur=5104.210&itag=18&fex
p=9407536%2C9412913%2C9413141%2C9416126%2C9416984%2C9417206%2C9418203%2C9418777%2C9420452%2C9
420540%2C9421665%2C9422596%2C9423662%2C9424114%2C9424629%2C9424845%2C9425448%2C9425953%2C9426
410%2C9426494&ratebypass=yes&lmt=1394506818301594&ip=185.53.168.70&requiressl=yes&sparams=dur
%2Cid%2Cip%2Cipbits%2Citag%2Clmt%2Cmime%2Cmm%2Cmn%2Cms%2Cmv%2Cpl%2Cratebypass%2Crequiressl%2C
source%2Cupn%2Cexpire&upn=tkAO65sWcj8&title=%D0%A4%D0%B8%D0%BB%D1%8C%D0%BC%20%D0%BE%20GNU/Lin
ux%20Revolution%20OS%20[RU] -O revolution-os.mp4
[1] 5215
             ...          ...          ...          ...          ...
?🗣* --2015-12-26 19:00:05--  https://r4---sn-xjpm-
4g5e.googlevideo.com/videoplayback?signature=D01706B19078430C4AB2E7E0E8A1C03A685048F5.53412B2
63BA6F90E9DB1ADD7F175E6AB4F7F18A3
             ...          ...          ...          ...          ...
[23] 5237
[2]   Готово           expire=1451166972
             ...          ...          ...          ...          ...
Распознаётся r4---sn-xjpm-4g5e.googlevideo.com (r4---sn-xjpm-4g5e.googlevideo.com)...
Подключение к r4---sn-xjpm-4g5e.googlevideo.com (r4---sn-xjpm-
4g5e.googlevideo.com)|194.122.81.15|:443... соединение установлено.
HTTP-запрос отправлен. Ожидание ответа... 403 Forbidden
2015-12-26 19:00:05 ОШИБКА 403: Forbidden.
```

❷ bender@ubuntu:~$ **wget 'https://r4---sn-xjpm-4g5e.googlevideo.com/videoplayback?signature=D01706B19078430C4AB2E7E0E8A1C03A685048F5.53412B263BA6F90E9DB1ADD7F175E6AB4F7F18A3&expire=1451166972&pl=24&mv=u&mt=1451145210&ms=au&id=o-ANXmoZGwigBoC32QKD0mA9iQcFzRyI8UhFWlgl5Wll8c&mn=sn-xjpm-4g5e&mm=31&key=yt6&ipbits=0&mime=video%2Fmp4&source=youtube&sver=3&dur=5104.210&itag=18&fexp=9407536%2C9412913%2C9413141%2C9416126%2C9416984%2C9417206%2C9418203%2C9418777%2C9420452%2C9420540%2C9421665%2C9422596%2C9423662%2C9424114%2C9424629%2C9424845%2C9425448%2C9425953%2C9426410%2C9426494&ratebypass=yes&lmt=1394506818301594&ip=185.53.168.70&requiressl=yes&sparams=dur%2Cid%2Cip%2Cipbits%2Citag%2Clmt%2Cmime%2Cmm%2Cmn%2Cms%2Cmv%2Cpl%2Cratebypass%2Crequiressl%2Csource%2Cupn%2Cexpire&upn=tkAO65sWcj8&title=%D0%A4%D0%B8%D0%BB%D1%8C%D0%BC%20%D0%BE%20GNU/Linux%20Revolution%200S%20[RU] -O linux-revolution-os.mp4' -O revolution-os.mp4**

```
           ...          ...          ...          ...          ...
Распознаётся r4---sn-n8v7zn76.googlevideo.com (...)... 74.125.13.211, 2a00:1450:4011:1a::13
Подключение к r4---sn-n8v7zn76.googlevideo.com (...)|74.125.13.211|:443... соединение установлено.
HTTP-запрос отправлен. Ожидание ответа... 200 OK
Длина: 260800224 (249M) [video/mp4]
Сохранение в: «revolution-os.mp4»

100%[=======================================================================>]
260 800 224 2,85M/s   за 86s

2015-12-26 19:19:23 (2,91 MB/s) - «revolution-os.mp4» сохранён [260800224/260800224]
```

В листингах 5.32 и 5.33 использованы разные способы множественного экранирования '' и "", хотя в этих примерах между ними нет разницы. Экранирование метасимволов в двойных кавычках "" носит название *слабого* экранирования, потому что не распространяется на метасимволы подстановки **$** и `` и метасимвол одиночного экранирования ****. Экранирование метасимволов в одинарных кавычках '' носит название *сильного* экранирования, потому как распространяется на абсолютно любые метасимволы.

Листинг 5.34. Сильное и слабое экранирование

```
bender@ubuntu:~$ cal
    Декабрь 2015
Вс Пн Вт Ср Чт Пт Сб
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
❶ bender@ubuntu:~$ notify-send $(date) $(cal)
Invalid number of options.
bender@ubuntu:~$ set -x
```

```
❷ bender@ubuntu:~$ notify-send $(date) $(cal)
   ++ date
   ++ cal
                                  ①1                          ②1③1④1  ...1
   + notify-send $'\320\241\320\261.' $'\320\264\320\265\320\272.' 26 20:21:59 MSK 2015
   $'\320\224\320\265\320\272\320\260\320\261\321\200\321\214' 2015 $'\320\222\321\201'
   $'\320\237\320\275' $'\320\222\321\202' $'\320\241\321\200' $'\320\247\321\202'
   $'\320\237\321\202' $'\320\241\320\261' 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
   22 23 24 25 $'_\b2_\b6' 27 28 29 30 31
❸ bender@ubuntu:~$ notify-send '$(date)' '$(cal)'
   + notify-send '$(date)' '$(cal)'
❹ bender@ubuntu:~$ notify-send "$(date)" "$(cal)"
   ++ date
   ++ cal
   + notify-send 'Сб. дек. 26 20:28:28 MSK 2015' '    Декабрь 2015
   Вс Пн Вт Ср Чт Пт Сб
         1  2  3  4  5
    6  7  8  9 10 11 12
   13 14 15 16 17 18 19
   20 21 22 23 24 25 26
   27 28 29 30 31'
```

В примерах из листинга 5.34 необходимо текущее время и календарь на текущий месяц отправить пользователю в качестве уведомления в сеансе графического интерфейса. При помощи команды **notify-send(1)** можно послать уведомление, заголовок которого передается первым ее параметром, а текст уведомления — вторым. Прямое решение ❶ с подстановками вывода команд **date(1)** и **cal(1)** не срабатывает и завершается со странным результатом. В режиме трассировки команд интерпретатора ❷ видно, что пробелы ①②③... в выводе команд **date(1)** и **cal(1)** были естественным образом подставлены как аргументы команды **notify-send(1)**, что привело к передаче более чем двух ожидаемых аргументов. Решение с сильным экранированием ❸ посылает уведомление с литеральным написанием подстановок, без их выполнения, что тоже не является ожидаемым результатом. Слабое экранирование ❹ работает, как ожидается, с выполнением подстановок и экранированием пробелов после выполнения подстановки, в результате чего формируются два правильных аргумента команды **notify-send(1)**.

5.6. Списки команд

Перенаправления и подстановки командного интерпретатора позволяют производить различные вычисления, но не разрешают *управлять ходом* вычислений — выполнять различные действия в зависимости от результата вычислений или циклически

повторять вычисления. Для управления ходом выполнения сценариев на языке командного интерпретатора служат *списки* команд.

Простейшие списки формирует пользователь при интерактивной работе с командным интерпретатором, последовательно запуская команды при помощи управляющего символа перевода строки ↵ или параллельно запуская задания *(см. разд. 4.8.1)* посредством метасимвола **&** запуска задач в «фоновом» режиме.

В пакетном режиме работы такие списки называются *простыми* и формируются конструкциями **command$_1$; command$_2$; ...** и **command$_1$ & command$_2$ & ...**, называемыми *простым синхронным* (листинг 5.35) и *простым асинхронным* (листинг 5.36) списками соответственно.

Листинг 5.35. Простой последовательный (синхронный) список команд

```
bender@ubuntu:~$ ① dd if=/dev/dvd of=dvd.iso ↵
                                              ↓
                                  ⑦ ⑥ ⑤ ④ ③ ② ① ⑧ ⑨ ⑩ ⑪
586896+0 записей получено
586896+0 записей отправлено
скопировано 300490752 байта (300 MB), 5,10918 с, 58,8 MB/c
bender@ubuntu:~$ ② ls -lh dvd.iso
-rw-r--r-- 1 fitz fitz 287M дек.   3 13:42 dvd.iso
               ①↴              ②↴
bender@ubuntu:~$ bzip2 -v dvd.iso ; ls -lh dvd.iso.bz2
                                  ↓
❶   dvd.iso:  3.251:1,  2.461 bits/byte, 69.24% saved, 300490752 in, 92444196 out.
❷ -rw-r--r-- 1 fitz fitz 89M дек.   3 13:42 dvd.iso.bz2
```

Интерактивный режим взаимодействия ①② с командным интерпретатором заставляет пользователя дожидаться завершения текущей выполняющейся программы и появления приглашения командного интерпретатора для того, чтобы запустить последующую команду. Последовательный список позволяет организовать подобное последовательное выполнение посредством командного интерпретатора.

Листинг 5.36. Простой параллельный (асинхронный) список команд

```
            ①↴                          ②↴                          ③↴
bender@ubuntu:~$ time xz --best -k plan9.iso -S .b.xz & time xz --fast -k plan9.iso -S .f.xz & ps f
                                          ↓                           ↓
[1] 11978
[2] 11979
  PID TTY      STAT   TIME COMMAND
```

```
   11716 pts/0    S      0:00 -bash
   11978 pts/0    S      0:00  \_ -bash
❶  11981 pts/0    R      0:00  |   \_ xz --best -k plan9.iso -S .b.xz
   11979 pts/0    S      0:00  \_ -bash
❷  11982 pts/0    R      0:00  |   \_ xz --fast -k plan9.iso -S .f.xz
   11980 pts/0    R+     0:00  \_ ps f
   bender@ubuntu:~$ wait ; ls -lh plan9.*
                                  ⑦ ⑥ ⑤ ④ ③ ② ① ⑨ ⑧ ⑦ ⑥
❷  real 0m37.791s
   user 0m37.476s
   sys  0m0.264s
                                  ⑦ ⑥ ⑤ ④ ③ ② ① ⑨ ⑧ ⑦ ⑥
❶  real 3m7.409s
   user 3m6.656s
   sys  0m0.472s
   [1]-  Готово                  time xz --best -k plan9.iso -S b.xz
   [2]+  Готово                  time xz --fast -k plan9.iso -S f.xz
   -rw-r--r-- 1 bender bender 287M    нояб. 28 15:47 plan9.iso
❶  -rw-r--r-- 1 bender bender  61M ➥ нояб. 28 15:47 plan9.iso.b.xz
❷  -rw-r--r-- 1 bender bender  89M ➥ нояб. 28 15:47 plan9.iso.f.xz
```

При помощи асинхронного списка (см. листинг 5.36), наоборот, организуется запуск сразу нескольких процессов в «фоновом» режиме, дождаться завершения которых позволяет встроенная команда интерпретатора **wait**.

5.6.1. Условные списки

Условные списки команд представляют собой компактные, но достаточно выразительные конструкции, управляющие ходом выполнения сценария. Как указывалось выше (см. специальный параметр **?** в *разд. 5.4.2*), каждая команда имеет статус завершения — нулевой при *успешном* выполнении и отличный от нуля при *неуспешном*. Условный список «И» формируется конструкцией вида **command$_1$ && command$_2$ && ...** и выполняет команды *последовательно*, запуская очередную команду, только если предыдущая закончилась *успешно* (заканчивая свое выполнение после первой *неуспешной* команды). Условный список «ИЛИ» вида **command$_1$ || command$_2$ || ...**, наоборот, очередную команду запускает только в случае *неуспеха* предыдущей (заканчивая свое выполнение после первой *успешно* выполнившейся команды).

В примере из листинга 5.37 безусловная попытка ❶ смонтировать ISO-образ диска неуспешна, потому что эта операция уже была выполнена ранее. Условный

список «ИЛИ» ❷ организует проверку «смонтированности» файловой системы в указанный каталог. Если первая команда списка **findmnt(1)** завершится неуспехом (если никакая файловая система в указанный каталог не смонтирована), то в этом случае будет запущена команда монтирования **fuseiso(1)**. В противном случае, результатом успешно выполнившейся команды **findmnt(1)** будет вывод информации о файловой системе, уже смонтированной в целевой каталог.

Листинг 5.37. Список «ИЛИ»

```
❶ bender@ubuntu:~$ fuseiso dvd.iso ~/.dvd
  fuse: mountpoint is not empty
  fuse: if you are sure this is safe, use the 'nonempty' mount option
❷ bender@ubuntu:~$ set -x
  bender@ubuntu:~$ findmnt ~/.dvd || fuseiso dvd.iso ~/.dvd
  + findmnt /home/bender/.dvd
  TARGET          SOURCE FSTYPE  OPTIONS
  /home/bender/.dvd fuseiso fuse.fus rw,nosuid,nodev,relatime,user_id=1000,group_id=1010
```

Условный список «И» в примере из листинга 5.38, наоборот, пытается размонтировать файловую систему, только если она была смонтирована ранее.

Листинг 5.38. Список «И»

```
bender@ubuntu:~$ fusermount -u ~/.dvd
fusermount: entry for /home/bender/.dvd not found in /etc/mtab
bender@ubuntu:~$ set -x
bender@ubuntu:~$ findmnt ~/.dvd && fusermount -u ~/.dvd
+ findmnt /home/bender/.dvd
```

Условные списки можно комбинировать, например, в виде **command$_1$ && command$_2$ || command$_3$ ...** или **command$_1$ || command$_2$ && command$_3$**. Сначала будет выполнена первая команда и начнется анализ ее статуса завершения по условиям списка. Если после команды указано условие «И» **&&**, то при ее успешном завершении будет выполнена *следующая команда*, а при неуспешном анализ продолжится на *следующем условии* и т. д. И наоборот, если после команды указано условие «ИЛИ», то при ее неуспешном завершении будет выполнена *следующая команда*, а при успешном анализ продолжится на *следующем условии*. Как только будет найдена очередная команда для выполнения, она будет выполнена и начнется анализ *ее статуса* выполнения по условиям списка, следующим за ней.

Так, например, в листинге 5.39 комбинированный список «ИЛИ-И» всегда приводит к выводу списка файлов из каталога, куда смонтирован ISO-образ, вне зави-

симости от того, был ли он туда смонтирован до запуска списка или был смонтирован командами при его выполнении.

Листинг 5.39. Комбинация списков «И» и «ИЛИ»

```
bender@ubuntu:~$ set -x
bender@ubuntu:~$ findmnt ~/.dvd || fuseiso dvd.iso ~/.dvd && ls -a ~/.dvd
❶ + findmnt /home/bender/.dvd
❷ + fuseiso dvd.iso /home/bender/.dvd
❸ + ls --color=auto -a /home/bender/.dvd
.       acme    bootdisk.img  env      LICENSE.afpl  mips    pbsraw   sparc
..      adm     cfg           fd       LICENSE.gpl   mnt     power    sys
386     amd64   cron          lib      lp            n       power64  tmp
9load   arm     dist          LICENSE  mail          NOTICE  rc       usr
bender@ubuntu:~$ findmnt ~/.dvd || fuseiso dvd.iso ~/.dvd && ls -a ~/.dvd
❶ + findmnt /home/bender/.dvd
TARGET           SOURCE  FSTYPE   OPTIONS
/home/bender/.dvd fuseiso fuse.fus rw,nosuid,nodev,relatime,user_id=1008,group_i
❸ + ls --color=auto -a /home/bender/.dvd
.       acme    bootdisk.img  env      LICENSE.afpl  mips    pbsraw   sparc
..      adm     cfg           fd       LICENSE.gpl   mnt     power    sys
386     amd64   cron          lib      lp            n       power64  tmp
9load   arm     dist          LICENSE  mail          NOTICE  rc       usr
```

5.6.2. Составные списки: ветвление

Условные списки «И» и «ИЛИ» являются простейшей формой *ветвления* хода выполнения сценария в зависимости от успеха или неудачи выполнения той или иной команды. При помощи специальной команды[1] **test(1)**, позволяющей выполнять проверки *логических выражений*, можно осуществлять ветвление сценария, например, в зависимости от определенных условий или значений тех или иных параметров.

В листинге 5.40 показано, что команда **test(1)** и ее «красивая» форма **[** изначально являются внешними командами, что так же влечет за собой накладные расходы на системные вызовы **fork(2)** и **execve(2)**. Поэтому в большинстве интерпретаторов обе формы команды **test(1)** реализованы еще и как встроенные команды.

---

[1] Аналогично специальной команде **expr(1)**, предназначенной для вычисления арифметических выражений.

Листинг 5.40. Команды test и [

```
bender@ubuntu:~$ which test
/usr/bin/test
bender@ubuntu:~$ which [
/usr/bin/[
bender@ubuntu:~$ type -a test
test встроена в оболочку
test является /usr/bin/test
bender@ubuntu:~$ type -a [
[ встроена в оболочку
[ является /usr/bin/[
bender@ubuntu:~$ test -f /etc/passwd
bender@ubuntu:~$ echo $?
0
bender@ubuntu:~$ [ -w /etc/passwd
-bash: [: пропущен `]'
bender@ubuntu:~$ [ -w /etc/passwd ] ➥
bender@ubuntu:~$ echo $?
1
```

Команда **test(1)** выполняет проверку логических выражений и заканчивается успехом, если проверяемое выражение *истинно*, и неуспехом, если проверяемое выражение *ложно*. Именно это ее свойство используется для реализации ветвления. Например, во втором примере из листинга 5.41 проверяется наличие блочного (**-b**) файла устройства **/dev/cdrom** и при наличии запускается команда **eject(1)**, предписывающая драйверу устройства открыть лоток привода CD/DVD.

Листинг 5.41. Ветвление при помощи условных списков

```
bender@ubuntu:~$ which clear && clear || tput clear
bender@ubuntu:~$ [ -b /dev/cdrom ] && eject /dev/cdrom
```

Условные списки удобно использовать для ветвления, если в каждой ветви выполняется по одной команде, как в первом примере из листинга 5.41, где посредством **which(1)** проверяется наличие команды **clear(1)**, которая при наличии и вызывается для очистки терминала. В противном случае, терминал очищается при помощи управляющей последовательности, которую выводит команда **tput(1)**.

При необходимости выполнить в каждой ветви сценария несколько команд, используется конструкция ветвления «ЕСЛИ» вида

if [!] list; then list; [elif [!] list; then list;] ... [else list;] fi

являющаяся *составным списком* и использующая *ключевые слова* языка командного интерпретатора: **if**, **then**, **elif**, **else**, **fi**. Стоит заметить, что в самой конструкции *логические выражения* отсутствуют, а ветвление основано на статусах завершения списков команд, указывающихся после ключевых слов **if** и **elif**. В листинге 5.42 из ISO-образа извлекается каталог **sys/src**, но двумя разными способами. Если установлен архиватор **7z(1)** и обнаружен командой **which(1)**, то выполняется «успешная ветвь» ① с его использованием, иначе выполняется «неуспешная» ветвь с использованием монтирования/размонтирования ISO-образа при помощи **fuseiso(1)**.

Листинг 5.42. Простое ветвление по результату выполнения команд

```
❶ bender@ubuntu:~$ if which 7z ↵
☛ > then ↵
↓① >   7z x plan9.iso sys/src ↵
   > else ↵
↓② >   fuseiso dvd.iso ~/dvd ↵
↓  >   cp -a ~/dvd/sys/src . ↵
↓  >   fusermount -u ~/dvd ↵
   > fi ↵
❷ bender@ubuntu:~$ if ! findmnt ~/dvd ↵
   > then ↵
   >   fuseiso dvd.iso ~/dvd ↵
   > fi ↵

❸ bender@ubuntu:~$ if test -b /dev/cdrom; then eject /dev/cdrom; fi
```

В примере ❷ из листинга 5.42 используется признак отрицания **!**, который заставляет **if** действовать «наоборот», т. е. запускать ветвь **then**, если список после **if** закончился *неуспешно*, и ветвь **else** в противном случае. В примере ❸ стоит обратить внимание на то, что списки, использующиеся в конструкции **if**, могут отделяться символом **;**, а не символом перевода строки ↵, как в примерах ❶ и ❷.

В листинге 5.43 приведен маленький сценарий на языке командного интерпретатора, исполняющийся интерактивно в режиме трассировки команд. В нем используются практически все конструкции командного интерпретатора: перенаправление потоков, конвейер, подстановки команд, параметров, арифметических выражений, экранирование и составной список ветвления.

Листинг 5.43. Простое ветвление по результату проверки условий

```
   bender@ubuntu:~$ set -x
❶  bender@ubuntu:~$ load=$(awk '{printf "%d", $1*100 + $2*10 + $3}' < /proc/loadavg)
   ++ awk '{printf "%d", $1*100 + $2*10 + $3}'
   + load=86
❷  bender@ubuntu:~$ power=$(( $(nproc)*100 ))
   ++ nproc
   + power=400
❸  bender@ubuntu:~$ temp=$(( $(cat /sys/class/thermal/thermal_zone0/temp)/1000 ))
   ++ cat /sys/class/thermal/thermal_zone0/temp
   + temp=55
❹  bender@ubuntu:~$ if [ $load -ge $power -a $temp -gt 80 ]↵
   > then↵
   >   notify-send "Обнаружена перегрузка: t=$temp C" "$(top -bn1 | head -5)" ↵
   > fi↵
⑦  + '[' 86 -ge 400 -a 55 -gt 80']'
```

Сначала ❶ вычисляется «интегральная» нагрузка на систему **load** путем «взвешивания» с весами 100, 10 и 1 статистики за последние 1, 5 и 15 мин о среднем количестве процессов, стоящих в очереди на получение процессорного времени или доступа к устройству ввода-вывода. Измеряемая ядром статистика считывается из *трех* столбцов *одной* строки файла **/proc/loadavg** псевдофайловой системы **proc(5)**. Взвешивание выполняется при помощи процессора текстовых таблиц **W**:[AWK], основное назначение которого состоит в разбиении строк на столбцы и построчном выполнении указанных действий над ними. В данном случае **awk(1)** выводит в формате (**printf**) целого числа (**%d**) результат взвешивания на основе значений статистики из первого (**$1**), второго (**$2**) и третьего (**$3**) столбцов.

Затем ❷ вычисляется «мощность» вычислительной системы **power** путем умножения 100 (процентов) на количество ядер процессора. Следующим шагом ❸ вычисляется температура **temp** процессора в градусах Цельсия (°C), рассчитываемая путем деления нацело измеряемой ядром (из псевдофайловой системы **sysfs**) температуры на 1000 (миллиградусов).

Завершает сценарий составной список ветвления **if** ❹, проверяющий при помощи «красивой» формы **[** команды **test(1)** логическое выражение |*нагрузка* ≥ *мощность* И *температура* > 80°C| на истинность и уведомляющий при помощи **nodtify-send(1)** пользователя о перегрузке, в сообщении которого будет приведена текущая статистика потребления ресурсов из первых пяти строк вывода команды **top(1)**.

Еще один вид составного списка, предназначенного для организации *множественного* ветвления, реализуется конструкцией

case word in [[(] pattern$_1$ [| pattern$_2$]...) list$_1$;;]... esac

с ключевыми словами **case**, **in**, **esac** и признаком окончания ветви **;;**. При множественном ветвлении используются не логические[1], а *шаблонные* выражения, заимствованные у подстановок имен файлов. Для определения ветви исполнения слово **word** проверяют на соответствие шаблонам **pattern$_i$** слева направо (сверху вниз), при совпадении с одним из которых выполняется соответствующий список команд **list$_j$**, и дальше никакие проверки не производятся.

В примере из листинга 5.44 сильно заэкранированный текст небольшого сценария командного интерпретатора присваивается переменной **PROMPT_COMMAND**, выполнение команд которой происходит каждый раз перед выводом первичного приглашения **PS1**. В роли такой команды выступает составной список множественного ветвления **case**, который анализирует статус завершения последней выполненной интерпретатором команды при помощи подстановки значения специального параметра **$?**.

Если статус завершения равен **0** (успех) или **127** (команда не найдена), то вспомогательной «выдуманной» переменной **PROMPT_COLOR** присваивается управляющая последовательность *(см. разд. 2.4)* «нормального» (**sgr0**) режима вывода на терминал. Если статус завершения равен **130** (штатное завершение по сигналу **SIGINT**, например, при нажатии на **^C**) или **131** (аварийное завершение по сигналу **SIGQUIT**, например, при нажатии на **^**), то используется управляющая последовательность «негативного» (**rev**) вывода ❶. Если же команда завершилась со статусом **1** или с любым другим ошибочным статусом, то будет использована управляющая последовательность «жирного» (**bold**) начертания ❷.

Листинг 5.44. Множественное ветвление

```
bender@ubuntu:~$ PROMPT_COMMAND='↵
> case $? in ↵
>   0|127) PROMPT_COLOR=`tput sgr0`;; ↵
>   13[01]) PROMPT_COLOR=`tput rev`;; ↵
>   1) PROMPT_COLOR=`tput bold`;; ↵
>   *) echo "exit code = $?"; PROMPT_COLOR=`tput bold`;; ↵
> esac'↵
bender@ubuntu:~$ PS1='`echo $PROMPT_COLOR`\u@\h \w\$ `tput sgr0`'
bender@ubuntu ~$ dd
```

[1] При «обычном» ветвлении **if-then-else** для определения ветви исполнения тоже используются не логические выражения, а статусы завершения списков.

```
^C0+0 записей получено
0+0 записей отправлено
скопировано 0 байт (0 B), 0,900673 с, 0,0 kB/c

❶ bender@ubuntu ~$ date
Пн. янв.  4 15:39:18 MSK 2016 MSK 2016
bender@ubuntu ~$ grep bender /etc/shadow
grep: /etc/shadow: Отказано в доступе
exit code = 2
❷ bender@ubuntu ~$
```

5.6.3. Составные списки: циклы

Последний важный вид составных списков предназначен для многократного циклического выполнения команд в сценариях командного интерпретатора. Различают цикл с *параметром*, реализуемый конструкцией

```
for name in [words ...]; do list; done
```

и циклы с *условием* «ПОКА»

```
while [!] list; do list; done
```

и «ДО»

```
unitl [!] list do list; done
```

с ключевыми словами **for**, **in**, **while**, **until**, **do**, **done**, соответственно.

В примере из листинга 5.45 цикл с параметром используется для создания галереи миниатюр фотографий при помощи составного списка **for** и подстановки вывода команд. В режиме трассировки команд интерпретатора видно, что команда **convert(1)** в теле цикла выполняется столько раз, сколько слов (найденных имен файлов) было подставлено из вывода команды **find(1)**. При этом переменная **file** на каждой итерации цикла **()** принимает очередное значение из подставленного списка. В результате последовательных подстановок разных значений переменной **$file** были выполнены одинаковые преобразования разных файлов изображений.

Листинг 5.45. Цикл с параметром: создание галереи миниатюр фотографий

```
bender@ubuntu:~$ set -x
bender@ubuntu:~$ for file in $(find DCIM -name '*.jpg') ↵
> do ↵
>   convert $file -resize 100x $(basename $file .jpg).mini.jpg ↵
> done ↵
```

```
❶ ++ find DCIM -iname '*.jpg'
❷ + for file in '$(find DCIM -name '\''*.jpg'\'')'
   ++ basename DCIM/DSC_0067.jpg .jpg
   + convert DCIM/DSC_0067.jpg -resize 100x DSC_0067.mini.jpg
❷ + for file in '$(find DCIM -name '\''*.jpg'\'')'
   ++ basename DCIM/DSC_0189.jpg .jpg
   + convert DCIM/DSC_0189.jpg -resize 100x DSC_0189.mini.jpg
              ...          ...          ...          ...          ...
❷ + for file in '$(find DCIM -iname '\''*.jpg'\'')'
   ++ basename DCIM/DSC_0062.jpg .jpg
   + convert DCIM/DSC_0062.jpg -resize 100x DSC_0062.mini.jpg
⃠
```

В этом примере для формирования результирующих имен файлов используется подстановка вывода команды **basename(1)** для отрезания «расширений» **.jpg** от имен файлов, к которым затем приклеиваются новые «расширения» **.mini.jpg**.

В примере из листинга 5.46 составной список **for** используется вместе с подстановкой команды **seq(1)**, формирующей список чисел 1, ..., 254, очередные значения из которого принимает переменная **node**. На каждой итерации цикла ❷ используется подстановка значения **$node** для формирования IP-адреса очередного узла локальной сети, доступность которого проверяется при помощи команды **ping(1)**.

Листинг 5.46. Цикл с параметром: проверка доступности узлов локальной сети

```
bender@ubuntu:~$ for node in $(seq 1 254) ↵
> do ↵
>   ping -c 1 -W 1 192.168.1.$node ↵
> done ↵
++ seq 1 254
❷ + for node in '$(seq 1 254)'
+ ping -c 1 -W 1 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.

--- 192.168.1.1 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
              ...          ...          ...          ...          ...
❷ + for node in '$(seq 1 254)'
+ ping -c 1 -W 1 192.168.1.45
PING 192.168.16.45 (192.168.16.45) 56(84) bytes of data.
64 bytes from 192.168.16.45: icmp_req=1 ttl=62 time=5.25 ms
```

```
   --- 192.168.16.45 ping statistics ---
   1 packets transmitted, 1 received, 0% packet loss, time 0ms
   rtt min/avg/max/mdev = 5.257/5.257/5.257/0.000 ms
                ...          ...             ...            ...            ...
() + for node in '$(seq 1 254)'
   + ping -c 1 -W 1 192.168.1.254
   PING 192.168.1.1 (192.168.1.254) 56(84) bytes of data.

   --- 192.168.1.254 ping statistics ---
   1 packets transmitted, 0 received, 100% packet loss, time 0ms
⦸
```

В примере из листинга 5.47 на основе составного списка **while** организован «индикатор прогресса» процесса фонового сжатия ISO-образа диска при помощи опроса его состояния. Цикл выполняется, *пока* успешна команда **ps(1)**, опрашивающая процесс, PID которого передан при помощи подстановки специального параметра **$!**. При каждой итерации цикла **()** команда **ls(1)** выводит размер выходного файла **dvd.iso.bz2**, а команда **sleep(1)** приостанавливает выполнение на **1** секунду.

Листинг 5.47. Цикл «ПОКА»: ожидание завершения процесса

```
   bender@ubuntu:~$ bzip2 -kf dvd.iso &
   [1] 5773
   bender@ubuntu:~$ set -x
   bender@ubuntu:~$ while ps p $! ; do ls -lh dvd.iso.bz2; sleep 1 ; done
() + ps p 5773
   PID TTY      STAT   TIME COMMAND
☛  5773 pts/0    R      0:05 bzip2 -kf dvd.iso
   + ls --color=auto -lh dvd.iso.bz2
   -rw------- 1 bender bender 9,6M ☚ дек.  12 13:38 dvd.iso.bz2
              ...          ...             ...            ...            ...
   + sleep 1
() + ps p 5773
   PID TTY      STAT   TIME COMMAND
☛  5773 pts/0    R      0:58 bzip2 -kf dvd.iso
   + ls --color=auto -lh dvd.iso.bz2
   -rw------- 1 bender bender 87M ☚ дек.  12 13:39 dvd.iso.bz2
   [1]+  Готово                 bzip2 -kf dvd.iso
   + sleep 1
⦸ + ps p 4523
☛ PID TTY      STAT   TIME COMMAND
```

В примере из листинга 5.48 составной список **while** используется в качестве генератора пар имен файлов для массового параллельного переименования. Для этого стандартный поток ввода построчно считывается в цикле при помощи встроенной команды интерпретатора **read**, причем на каждой итерации цикла переменная **FN** принимает значение очередной строки, а команда **read** завершается успешно до исчерпания строк. В результате выполнения встроенной команды **echo** и при помощи подстановки значения переменной **$FN** и подстановок вывода команд **dirname(1)** и **basename(1)** на стандартном потоке вывода составного списка формируются пары имен файла — исходное считанное путевое имя и синтезированное путевое имя с измененным «расширением файла». Исходный поток имен формируется командой **find(1)**, а результирующий поток передается команде **xargs(1)**, которая в свою очередь передает пары имен (**-n2**) командам **mv(1)**, запускаемым параллельно в количестве ядер процессора, которое было получено командой **nproc(1)**.

Листинг 5.48. Цикл «ПОКА»: массовое параллельное переименование файлов

```
bender@ubuntu:~$ find DCIM -name '*.jpg' | ↵
> while read FN ; do echo $FN $(dirname $FN)/$(basename $FN .jpg).jpeg ; done | ↵
> xargs -n2 -P $(nproc) mv ↵
```

Аналогично, в примере из листинга 5.49 составной список **while** используется в качестве генератора ❶ классификатора файлов по их контрольным MD5-суммам. При помощи команд **find(1)**, **xargs(1)** и **md5sum(1)** формируется поток строк, в первом столбце которых будет выведена контрольная сумма W:[MD5] файла, а во втором столбце — его имя. Полученный поток построчно считывается в цикле при помощи встроенной команды интерпретатора **read**, причем на каждой итерации переменные **sum** и **file** принимают значения соответственно первого и второго столбцов строки, при этом во временном каталоге создаются файлы классификатора, именованные значениями **$sum**, в содержимое которых добавляются имена **$file**. Если контрольные суммы нескольких обрабатываемых файлов одинаковые, то их имена будут добавлены в разные строки одного и того же файла классификатора. Остается только найти те файлы классификатора, которые содержат более одной строки, они и будут содержать имена дубликатов. Для поиска по классификатору ❷ используется аналогичная связка **find(1)**, **xargs(1)** и **wc(1)**, формирующая на потоке вывода текстовую таблицу по два столбца в строке, в первом из которых указано количество строк в файле классификатора, а во втором — его имя. Затем, при помощи процессора текстовых таблиц **awk(1)** на стандартный поток вывода печатаются только те имена файлов классификатора из второго столбца строк таблицы (**print $2**), в первом столбце которых содержится число, большее единицы (**$1 > 1**). Полученный поток имен файлов отправляется на **xargs(1)** для вывода имен (**-t**) и содержимого при помощи **cat(1)**. Необходимо заметить, что сам классификатор раз-

мещен в каталоге файловой системы W:[tmpfs], использующей для хранения оперативную память, и имеет случайное имя, сгенерированное при помощи **mktemp(1)**.

Листинг 5.49. Цикл «ПОКА»: поиск дублирующихся фотографий

```
bender@ubuntu:~$ df -h /run/shm
Файл.система   Размер Использовано  Дост Использовано% Смонтировано в
none             4,0G          26M  3,9G            1% /run/shm
kotoff@ubuntu:~$ findmnt /run/shm
TARGET   SOURCE FSTYPE OPTIONS
/run/shm        tmpfs  rw,nosuid,nodev,relatim
bender@ubuntu:~$ T=$(mktemp -d /run/shm/XXXXXX)
bender@ubuntu:~$ find DCIM -type f -print0 | xargs -0 -n1 md5sum |↵
❶ > while read sum file ; do echo $file >> $T/$sum ; done↵
❷ bender@ubuntu:~$ find $T -type f | xargs -n1 wc -l |↵
   > awk '$1 > 1 { print $2 }' | xargs -n1 -t cat
☛ cat /dev/shm/fjYfn/4f7ba191573ded309bc0f04e08309d8a
   DCIM/Camera/IMG_20140731_212625+.jpg
   DCIM/Camera/IMG_20140731_212625.jpg
☛ cat /dev/shm/fjYfn/4329ae8006b6935d0c7fd66b57e07c0c
   DCIM/DSC_0046+.JPG
   DCIM/DSC_0046.JPG
```

В листинге 5.50 проиллюстрирована вторая форма цикла с условием — составной список **until**, который используется для ожидания доступности подключения к Интернету путем опроса наличия связи с узлом **8.8.8.8** (публичный DNS-сервер компании Google) при помощи команды **ping(1)**. Нужно отметить, что использование **until list do ...; done** может быть заменено эквивалентным **while ! list do ...; done** с указанием признака отрицания **!**.

Листинг 5.50. Цикл «ДО»: ожидание доступности подключения к Интернету

```
bender@ubuntu:~$ set -x
bender@ubuntu:~$ until ping -c1 -w1 8.8.8.8 ; do sleep 1;date ; done
() + ping -c1 -w1 8.8.8.8
☛ connect: Network is unreachable
   + sleep 1
   + date
   Сб. дек. 19 09:11:32 MSK 2015
            ...          ...          ...          ...          ...
() + ping -c1 -w1 8.8.8.8
```

```
connect: Network is unreachable
+ sleep 1
+ date
Сб. дек. 19 09:11:39 MSK 2015
+ ping -c1 -w1 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_req=1 ttl=56 time=11.6 ms

--- 8.8.8.8 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 11.670/11.670/11.670/0.000 ms
```

5.6.4. Функции

Как и во многих языках программирования, командный интерпретатор имеет средства структуризации сценариев при помощи функций. Составной *именованный* список команд, называемый *функцией*, объявляется при помощи (Bourne- и POSIX-диалекты) конструкций

name() compound-list

или (Korn-диалект)

function name compound-list

с использованием ключевого слова **function**, где **compound-list** — это составной список, например, **if**, **case**, **for** или **while**. Сформировать составной список из конвейера, простого или условного списка можно при помощи конструкций **{ list; }** или **(list)**, позволяющих выполнить **list** в том же или в отдельном дочернем процессе интерпретатора.

После объявления функция может быть неоднократно вызвана (как и любая другая внешняя или встроенная команда) с разными фактическими параметрами, значения которых в самой функции доступны при помощи подстановки позиционных параметров *(см. разд. 5.4.2)*.

В примере из листинга 5.51 объявляется, а затем вызывается без параметров функция, выполняющая вывод идентификаторов пользовательских учетных записей, доступных в системе (по аналогии с листингом 5.13).

Листинг 5.51. Список пользователей, зарегистрированных в операционной системе

```
bender@ubuntu:~$ function getusers↵
> {↵
>   getent passwd | cut -f 1 -d : | xargs -n1 id↵
> }↵
```

```
bender@ubuntu:~$ type -a getusers
getusers является функцией
getusers ()
{
    getent passwd | cut -f 1 -d : | xargs -n1 id
}
bender@ubuntu:~$ getusers
uid=0(root) gid=0(root) группы=0(root)
            ...          ...          ...          ...          ...
uid=1001(finn) gid=1001(finn) группы=1007(candy),1001(finn)
uid=1002(jake) gid=1002(jake) группы=1002(jake)
            ...          ...          ...          ...          ...
uid=1003(iceking) gid=1003(iceking) группы=1003(iceking)
uid=1004(marceline) gid=1004(marceline) группы=1004(marceline)
uid=1005(bubblegum) gid=1005(bubblegum) группы=1007(candy),1005(bubblegum)
uid=1006(fitz) gid=1008(fitz) группы=27(sudo),1008(fitz)
uid=1007(skillet) gid=1009(skillet) группы=1009(skillet)
            ...          ...          ...          ...          ...
uid=1008(bender) gid=1010(bender) группы=1010(bender)
            ...          ...          ...          ...          ...
```

В листинге 5.52 объявляется функция, являющаяся универсальным экстрактором «архивов». Сначала ❶ при помощи «красивой» формы **[** команды **test(1)** определяется наличие файла **(-f)**, путевое имя которого будет передано первым фактическим параметром при вызове функции. Затем ❷, если заданный файл найден, при помощи множественного ветвления и шаблонов, «примеряемых» к имени заданного файла, определяется и выполняется в соответствующей ветви команда распаковки **tar(1)**, **gunzip(1)**, **bunzip2(1)**, **rar(1)**, **unzip(1)**, **uncompress(1)** или **7z(1)**. При запуске функции в режиме трассировки видно, как работает передача аргументов в функцию, как выполняются списки ветвления и множественного ветвления.

Листинг 5.52. Универсальный экстрактор архивов

```
bender@ubuntu:~$ extract () ↵
❶ > if [ -f $1 ] ↵
  > then ↵
❷ >   case $1 in ↵
  >     *.tar)           tar xf  $1;; ↵
  >     *.tar.bz2|*.tbz2) tar xjf $1;; ↵
  >     *.tar.gz|*.tgz)   tar xzf $1;; ↵
  >     *.gz) gunzip      $1;; ↵
  >     *.bz2) bunzip2    $1;; ↵
```

```
>      *.rar) rar x      $1;; ↵
>      *.zip) unzip      $1;; ↵
>      *.Z)   uncompress $1;; ↵
>      *.7z|*.iso) 7z x  $1;; ↵
>      *) echo "Неизвестен распаковщик для '$1'"; return 1;; ↵
>   esac ↵
> else ↵
>   echo "'$1' не является файлом"; return 1 ↵
> fi
bender@ubuntu:~$ type -a extract
extract является функцией
extract ()
{
          ...          ...          ...          ...          ...
}
bender@ubuntu:~$ set -x
bender@ubuntu:~$ extract dvd.iso
+ extract dvd.iso
+ '[' -f dvd.iso ']'
+ case $1 in
+ 7z x dvd.iso

7-Zip 9.20  Copyright (c) 1999-2010 Igor Pavlov  2010-11-18
p7zip Version 9.20 (locale=ru_RU.UTF-8,Utf16=on,HugeFiles=on,4 CPUs)

Processing archive: dvd.iso
          ...          ...          ...          ...          ...
bender@ubuntu:~$ extract /tmp
+ extract /tmp
+ '[' -f /tmp ']'
+ echo ''\''/tmp'\'' не является файлом'
'/tmp' не является файлом
+ return 1
bender@ubuntu:~$ extract lynx_2.8.8dev.9-2ubuntu0.12.04.1_all.deb
+ extract lynx_2.8.8dev.9-2ubuntu0.12.04.1_all.deb
+ '[' -f lynx_2.8.8dev.9-2ubuntu0.12.04.1_all.deb ']'
+ case $1 in
+ echo 'Неизвестен распаковщик для '\''lynx_2.8.8dev.9-2ubuntu0.12.04.1_all.deb'\'''
Неизвестен распаковщик для 'lynx_2.8.8dev.9-2ubuntu0.12.04.1_all.deb'
+ return 1
```

Объявленные функции, как и переменные, располагаются в оперативной памяти процесса командного интерпретатора, в силу чего их *время жизни* и *область видимости* так же ограничиваются процессом их интерпретатора. В этом смысле объявленные функции практически неотличимы от встроенных команд интерпретатора и могут расцениваться как его «расширения».

Сохранить объявленные функции и присвоенные переменные нельзя, но можно их повторно объявить и присвоить, воспользовавшись инициализационными dot-файлами *(см. разд. 2.8)* интерпретатора, например сценариями **.bashrc** или **.profile**.

5.7. Сценарии на языке командного интерпретатора

Сценарий на языке командного интерпретатора представляет собой текстовый файл со списком команд, подлежащих выполнению в пакетном режиме. Достаточное количество программного обеспечения в системе написано на языке командного интерпретатора и представлено сценариями в каталогах **/usr/bin** и **/bin**. Сценарии неотличимы от любых других программ и доступны пользователями как внешние команды, формируя таким образом «расширения» операционной системы.

Пользователи, расширяющие таким образом операционную систему за счет собственных сценариев, располагают их обычно в каталоге **~/bin** и наделяют правом исполнения (см. ❶ и ❷ в листинге 5.2). Аналогично, «локальные» расширения, выполняемые системными администраторами для всех пользователей операционной системы, располагаются в **/usr/local/bin** (см. листинг 5.68).

В листинге 5.53 приведен сценарий **ldr** (**l**oa**d**er **r**equirements), обратный утилите **ldd(1)** (**l**oa**d**er **d**ependencies). Если при помощи **ldd(1)** можно увидеть библиотеки, от которых зависит заданная программа, то сценарий **ldr**, наоборот, показывает программу, использующую заданную библиотеку (листинг 5.54).

В сценарии объявлена ❶ функция **usage**, формирующая сообщение пользователю о правильных параметрах запуска сценария и использующаяся при некорректно заданных параметрах ❷ и ❺-①. В соответствии с соглашениями о выводе сообщение об «ошибке» перенаправлено ❶-① на поток **stderr** при помощи перенаправления потока **stdin** конструкцией **[n]>&m**. По соглашению о статусе завершения с «ошибкой» встроенная команда **exit** возвращает ненулевое значение ❶-②. Корректность параметра **$1** — имени искомой библиотеки — проверяется на «пустоту» ❷ при помощи «красивой» формы **[** команды **test(1)**.

Сам сценарий выполняется следующим образом: сначала ❸ переменной **ldpath** присваивается список имен каталогов, используемых динамическим компоновщиком **ld.so(8)** для поиска библиотек. Для этого из конфигурационных файлов компоновщика **/etc/ld.so.conf.d/*.conf** фильтруются (см. листинг 5.57) строки, незакомментированные символом **#**. Затем ❹, в зависимости от формы задания имени целевой

библиотеки, переменной **what** присваивается имя ее файла. Если библиотека была сразу задана абсолютным путевым именем своего файла, соответствующим шаблонному выражению **/*/lib*.so.[0-9]**, то оно используется непосредственно. Если библиотека была задана «предметным» именем ***NAME***, то при помощи команды **find(1)** организуется поиск абсолютного путевого имени файла библиотеки по шаблону **lib*NAME*.so.[0-9]** в каталогах компоновщика **ld.so(8)**, перечисленных в переменной **$ldpath**, а найденное имя присваивается переменной **what**. Корректность результата «вычисления» имени файла библиотеки **$what** проверяется ❺ при помощи «красивой» формы **[** команды **test(1)** на предмет существования (**-f**) такого файла. На очередном шаге сценария ❻ переменной **where** присваивается список каталогов (перечисленных в переменной окружения **PATH**), содержащих «доступные» пользователю исполняемые файлы программ. Для этого при помощи транслитератора **tr(1)** символы-разделители **PATH**-списка — двоеточия заменяются требуемыми символами-разделителями — пробелами.

Листинг 5.53. Сценарий поиска программ, использующих указанную библиотеку

```
   bender@ubuntu:~$ cd bin
   bender@ubuntu:~/bin$ cat ldr
   #!/bin/sh
❶ usage() {
|①   echo "Usage: $(basename $0) name | /.../libname.so.0" >&2 ←
|②   exit 1
↳ }

❷ [ -z "$1" ] && usage

❸ ldpath=$(grep -h '^[^#]' /etc/ld.so.conf.d/*.conf)

❹ case $1 in
|①   /*/lib*.so.[0-9]) what=$1;;
|②   *) what=$(find $ldpath -name "lib$1.so.[0-9]" 2>/dev/null);;
↳ esac

❺ if ! [ -f "$what" ]
|  then
|    echo "Library $1 is not found in" $LDPATH"
|①   usage
↳ fi
```

```
❻ where=$(printenv PATH | tr ':' ' ')

❼⓪find $where -type f |
|①  xargs file -L |
|②  grep ':.*ELF' |
|③  cut -f 1 -d : |
|④  while read exe
|   do
|⑤    ldd $exe | grep -q $what && echo $exe
↳   done
```

Целевая работа сценария выполняется одним конвейером ❼, в котором при помощи **find(1)** ищутся ⓪ регулярные файлы (**-type f**) в каталогах **$where**, список имен которых при помощи **xargs(1)** передается ① для классификации **file(1)**. Построенный классификатор **имя:класс** фильтруется ② при помощи **grep(1)**, из которого затем отделяются имена ③ посредством **cut(1)**. Последняя команда конвейера — составной список **while** циклично перебирает ④ имена файлов, считывая их встроенной командой **read** в переменную **exe**, где на каждой итерации цикла проверяется зависимость исполняемого файла **$exe** от заданной библиотеки. Зависимость устанавливается ⑤ по факту успешного завершения **grep(1)**, определяющего только наличие (**-q**) в выводе **ldd(1)** строки с именем путевого файла **$what**.

Листинг 5.54. Утилита ldd(1) и сценарий ldr

```
bender@ubuntu:~$ ldr gtk-3
/usr/sbin/unity-greeter
/usr/bin/gnome-session
      ...          ...          ...          ...          ...
/usr/games/sol
bender@ubuntu:~$ ldd /usr/games/sol
    linux-gate.so.1 =>  (0xb76f6000)
      ...          ...          ...          ...          ...
☛   libgtk-3.so.0 => /usr/lib/i386-linux-gnu/libgtk-3.so.0 (0xb6fd3000)
      ...          ...          ...          ...          ...
    libexpat.so.1 => /lib/i386-linux-gnu/libexpat.so.1 (0xb62a0000)
bender@ubuntu:~$ ldr pthread | wc -l
 ⑦  ⑧  ⑨  ⑩  ⑪  ⑫  ⑬  ⑭  ⑮  ⑯  ⑰
☛ 991
```

5.8. Инструментальные средства обработки текста

Данные, которые генерируют, обрабатывают и потребляют внешние и встроенные команды и конструкции интерпретатора, представляют собой *текстовые* потоки с произвольной структурой. Чаще всего в потоке можно выделить *строки*, отделяемые друг от друга управляющим символом перевода строки **\n** (**CR**, **^J** с кодом **0x0A**). Иногда в каждой строке выделяют *поля*, отделяемые друг от друга пробельными символами — управляющим символом горизонтальной табуляции **\t** (**HT**, **^I** с кодом **0x09**) или символом пробела (**SPC** с кодом **0x20**) либо каким-то другим символом, зачастую символом двоеточия **:** или символом вертикальной черты **|**.

Для манипуляции текстовыми данными используют **W:[регулярные выражения]** — формальный язык *поиска подстрок*, удовлетворяющих определенным правилам (табл. 5.2). Регулярные выражения подобны шаблонным выражениям, которые применяются в подстановках имен файлов *(см. разд. 5.4.1)* и в команде **find(1)** при поиске файлов по критерию имени (**-name**).

Таблица 5.2. Базовые регулярные выражения

Метасимвол	Значение
.	Любой одиночный символ
c*	Любое количество символов c
.*	Любое количество любых символов
[ab...z]	Любой символ из набора a, b, ..., z
[^ab...z]	Любой символ НЕ из набора a, b, ..., z
^	Начало строки
$	Конец строки

Регулярные выражения (RE, **R**egular **E**xpressions) учитывают *строковую* структуру обрабатываемых данных (табл. 5.3) и по сравнению с шаблонными выражениями вводят два дополнительных метасимвола — **^** и **$**, обозначающих начало и конец строки соответственно. Различают традиционные для UNIX *базовые*[1] (BRE, **B**asic RE), *расширенные*[2] (ERE, **E**xtended RE) и *Perl-совместимые* регулярные выражения (PCRE, **P**erl **C**ompatible RE).

---

[1] Во всех иллюстративных листингах используются **BRE**, а изучение **ERE** и **PCRE** выходит за рамки этой книги.

[2] В стандарте **W:[POSIX]**.

Таблица 5.3. Регулярные выражения в сравнении с шаблонными выражениями

Шаблонные выражения	Регулярные выражения	Значение
?	.	Любой одиночный символ
*	.*	Любое количество любых символов

Основной набор инструментальных средств, используемых для обработки текстовых потоков, включает в себя фильтр строк **grep(1)**, транслитератор **tr(1)**, фильтр символов и полей **cut(1)**, склейщик строк и полей **paste(1)**, процессор таблиц **awk(1)** и потоковый редактор **sed(1)**.

5.8.1. Фильтр строк grep

Имя команды **grep(1)** восходит к команде **g** древнейшего текстового редактора **ed(1)**, которая использовала регулярные выражения **re** для глобального поиска строк и применения других команд редактора к ним, например команды **п**ечати **p**. В целом, встроенная команда редактора **ed(1)** записывалась как **g/re/p**, что затем и дало название эквивалентной внешней команде **grep(1)**, фильтрующей (и печатающей на поток вывода) строки, соответствующие заданному регулярному выражению (рис. 5.5).

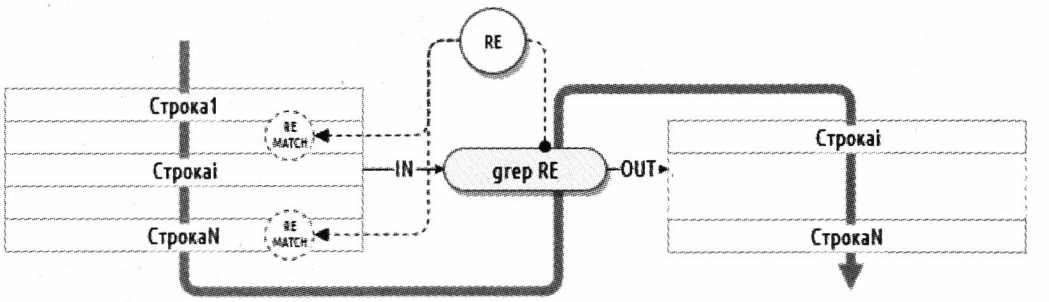


Рис. 5.5. Фильтр строк grep(1)

В примерах из листингов 5.55 и 5.56 **fgrep(1)** и **grep(1)** используются для фильтрации строк, содержащих (**-F**) фиксированные слова, что быстрее, чем обработка полных регулярных выражений.

Листинг 5.55. Выборка строк: список сценариев интерпретатора

```
bender@ubuntu:~ $ file -Li /*bin/* /usr/*bin/* | fgrep shellscript
/bin/bzcmp:                                   text/x-shellscript; charset=us-ascii
          ...          ...             ...             ...             ...
```

```
/bin/setupcon:                          text/x-shellscript; charset=utf-8
          ...          ...              ...             ...             ...
/usr/sbin/upgrade-from-grub-legacy:     text/x-shellscript; charset=us-ascii
```

Листинг 5.56. Выборка строк: RSS-память процессов браузера chromium

```
bender@ubuntu:~$ ps axo rss,comm | grep -F chromium
296628 chromium-browse
28432 chromium-browse
 7444 chromium-browse
166672 chromium-browse
57948 chromium-browse
 6400 chromium-browse
190752 chromium-browse
84872 chromium-browse
68028 chromium-browse
```

В примере из листинга 5.57 отфильтровываются строки конфигурационного файла команды **wget(1)**, закомментированные символом **#** в их начале. Для этого выбираются строки, соответствующие регулярному выражению **^[^#]**, которое требует в начале строки **^** наличия символа, не **^** входящего в набор **[#]**.

Листинг 5.57. Выборка строк: фильтрация комментариев

```
bender@ubuntu:/etc$ wc -l /etc/wgetrc
126 /etc/wgetrc
bender@ubuntu:/etc$ grep '^[^#]' /etc/wgetrc
passive_ftp = on
```

В листинге 5.58 при помощи регулярного выражения **[^0-9][0-9][0-9][^0-9]** выбираются строки с двузначными числами — содержащие два символа подряд из набора **[0-9]** (цифры), непосредственно перед которыми и после которых находятся символы не **^** из набора **[0-9]** (не цифры).

Листинг 5.58. Выборка строк с двузначными числами

```
bender@ubuntu:/etc$ grep [^0-9][0-9][0-9][^0-9] /etc/services
systat          11/tcp          users
daytime         13/tcp
        ...         ...             ...             ...             ...
z3950       210/tcp         wais            # NISO Z39.50 database
#> Ports are used in the TCP [45,106] to name the ends of logical
```

```
rtcm-sc104      2101/tcp                    # RTCM SC-104 IANA 1/29/99
x11             6000/tcp x11-0              # X Window System
              ...           ...               ...               ...               ...
# The following is probably Kerberos v5  --- ajt@debian.org (11/02/2000)
linuxconf       98/tcp                              # LinuxConf
```

5.8.2. Фильтр символов и полей cut

Команда **cut(1)** применяется для «вырезания» указанных (порядковым номером) символов или полей (по заданному разделителю) каждой строки (рис. 5.6). Несмотря на название команды, с содержимым файла никаких действий не производится, а символы и поля «выделяются» на поток вывода, что позволяет считать команду фильтром *полей*, аналогичным фильтру *строк* **grep(1)**.

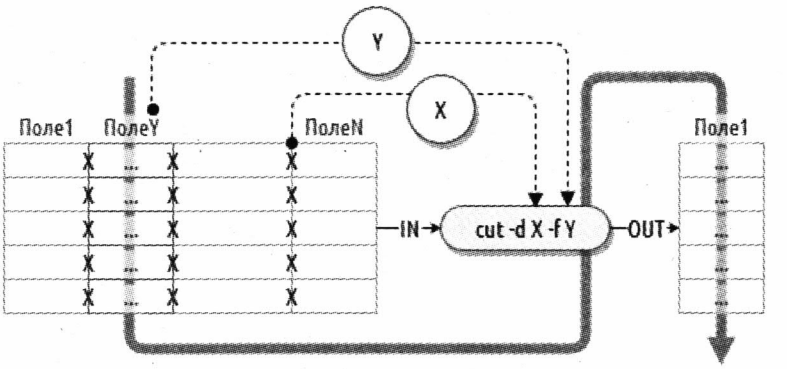


Рис. 5.6. Фильтр столбцов и полей cut(1)

В примере из листинга 5.59 при помощи **cut(1)** отфильтровывается первое поле (**-f 1 -d :**) тех строк классификатора файлов **file(1)**, которые были предварительно отфильтрованы по наличию слова **shellscript**. В результате на поток **stdout** будут выведены только имена файлов, классифицированных как сценарии командного интерпретатора.

Листинг 5.59. Выборка строк и полей с помощью grep и cut: список имен сценариев интерпретатора

```
bender@ubuntu:~ $ file -Li /*bin/* /usr/*bin/* | grep shellscript | cut -f 1 -d :
/bin/bzcmp
          ...          ...          ...          ...          ...
/bin/setupcon
          ...          ...          ...          ...          ...
/usr/sbin/upgrade-from-grub-legacy
```

Аналогично, в примере из листинга 5.60 из вывода команды **ps(1)** отбираются строки свойств процессов с именем **chromium**, затем выделяются символы с **1**-го по

8о-й каждой строки (содержащие суммарное количество резидентной памяти процесса **rss**, помещенное в первые **:8** символов). Отобранные статистические данные склеиваются командой **paste(1)** в одну строку (**-s**) посредством символа **+** в качестве разделителя (**-d**) и полученное таким образом арифметическое выражение отправляется на калькулятор **bc(1)** для подсчета. В результате будет получено суммарное потребление физической памяти всеми процессами Web-браузера **chromium**.

Листинг 5.60. Выборка столбцов и склейка полей: суммарная RSS-память процессов браузера chromium

```
bender@ubuntu:~$ ps axo rss:8,comm | grep chromium | cut -c 1-8 | paste -s -d + | bc
907176
```

5.8.3. Процессор текстовых таблиц awk

Команда **awk(1)** предназначается для обработки текстовых таблиц, столбцы которых разделяются пробельными символами, а строки — символом перевода строки (рис. 5.7). Язык процессора **W:[AWK]**[1] использует регулярные выражения **RE** для выделения *строк*, подлежащих обработке, и подстановочные выражения **$1**, **$2**, ..., **$N** для выделения *столбцов*. Каждая инструкция языка **AWK** записывается как **/RE/{ ACTION $i ... }** и заставляет процессор выполнить действие **ACTION** со столбцами **$i** каждой строки, соответствующей регулярному выражению **RE**.

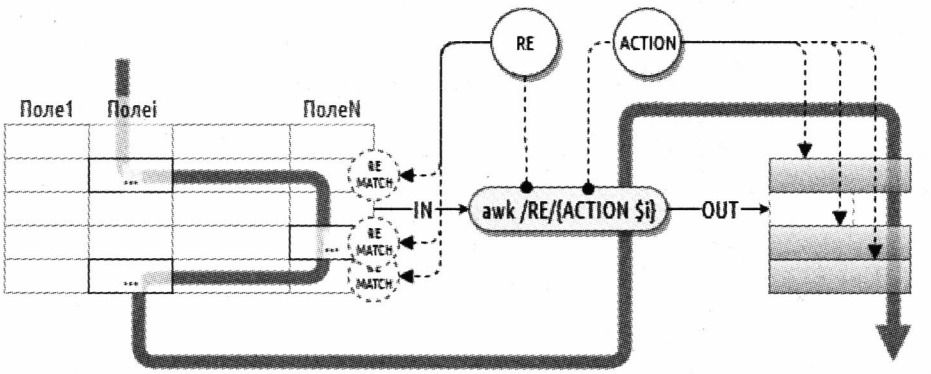


Рис. 5.7. Процессор текстовых таблиц awk(1)

Так, например, в листинге 5.61 при помощи **awk(1)** вместо **grep(1)** и **cut(1)** выделяются имена классифицированных командой **file(1)** файлов, являющихся сценариями командного интерпретатора.

---

[1] Имя процессора образовано от заглавных букв фамилий его разработчиков — **A**ho, **W**einberger и **K**ernighan.

Листинг 5.61. Выборка строк и полей с помощью awk: список имен сценариев интерпретатора

```
bender@ubuntu:~$ file -Li /*bin/* /usr/*bin/* | awk -F: '/shellscript/ { print $1 }'
/bin/bzcmp
          ...          ...          ...          ...          ...
/bin/setupcon
          ...          ...          ...          ...          ...
/usr/sbin/upgrade-from-grub-legacy
```

В примере из листинга 5.62 при помощи **awk(1)** подсчитывается суммарное потребление **RSS**-памяти процессами **chromium** при помощи простейшей **AWK**-программы с тремя инструкциями. Инструкция **BEGIN { sum = 0 }** выполняется *до* анализа строк, инструкция **END { print sum }** — *после* анализа всех строк, а инструкция **/chromium/ { sum += $1 }** выполняется *при* анализе строк и прибавляет к значению переменной **sum** целочисленное значение первого столбца **$1** тех строк, в содержимом которых будет найдена строка **chromium**.

Листинг 5.62. Суммирование по строкам и полям: суммарная RSS-память процессов браузера chromium

```
bender@ubuntu:~$ ps axo rss,comm | ↵
> awk 'BEGIN { sum = 0 } /chromium/ { sum += $1 } END { print sum }'
907176
```

5.8.4. Потоковый редактор текста sed

Потоковый редактор **sed(1)** (stream **ed**itor) применяется для пакетного (неинтерактивного) редактирования текстовых файлов, имеющих строчную структуру (рис. 5.8). Инструкции языка **W**:[SED] используют регулярные выражения **RE** для выделения *строк*, подлежащих редактированию при помощи команд **CMD**: вставки и добавления строк **i** (**i**nsert) и **a** (**a**ppend), замены строк **c** (**c**hange), удаления строк **d** (**d**elete), замены подстрок **s** (substitute) и др.

В примере из листинга 5.63 при помощи редактора **sed(1)** эмулируется поведение **grep(1)** путем удаления (команда **d**) тех строк потока, которые не (отрицание регулярного выражения **!**) содержат подстроку **chromium**.

Листинг 5.63. Выборка строк: удаление лишних

```
bender@ubuntu:~$ ps axo rss,comm | sed '/chromium/!d'
296628 chromium-browse
          ...          ...          ...          ...          ...
84872 chromium-browse
68028 chromium-browse
```

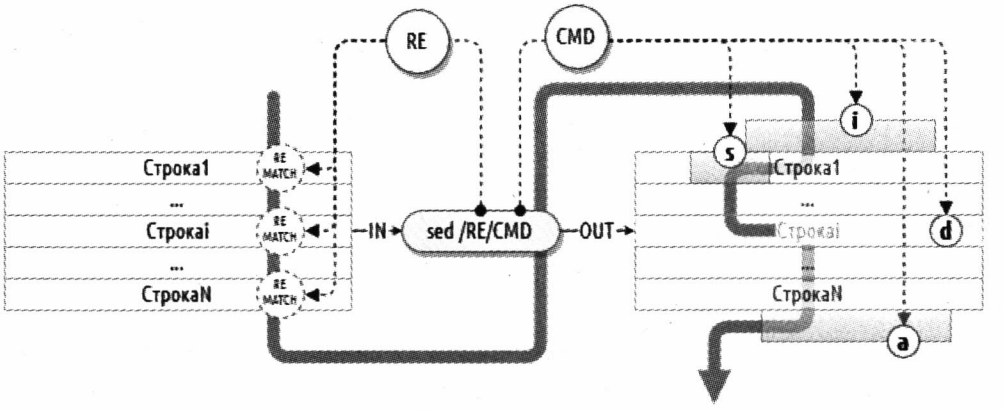


Рис. 5.8. Потоковый редактор sed(1)

Аналогично, в листинге 5.64 фильтруются строки файла **/etc/wgetrc** путем удаления из вывода строк, не соответствующих регулярному выражению **^[^#]** (в начале строки содержащих знак, не похожий на знак комментария **#**).

Листинг 5.64. Выборка строк: фильтрация комментариев

```
bender@ubuntu:~$ sed '/^[^#]/!d' /etc/wgetrc
passive_ftp = on
```

Кроме выбора отдельных строк инструкциями вида **/RE/CMD**, **sed(1)** позволяет адресовать блоки строк, подлежащих редактированию. При помощи инструкций вида **/RE1/,/RE2/ CMD** команда редактирования применяется ко всем строкам, начиная с той, которая соответствует выражению **RE1**, заканчивая той, которая соответствует выражению **RE2**.

Так, например, в листинге 5.65 из потока вывода команды **lspci(8)** выделяется блок текста, начиная со строки, содержащей слово **Ethernet**, вплоть до следующей пустой строки (**^$** между началом и концом которой не содержится ни одного символа).

Листинг 5.65. Выборка блоков текста

```
bender@ubuntu:~$ lspci -v | sed '/Ethernet/,/^$/!d'
02:00.0 Ethernet controller: Realtek Semiconductor Co., Ltd. RTL8111/8168B ... (rev 06)
        Subsystem: Samsung Electronics Co Ltd Device c0b6
        Flags: bus master, fast devsel, latency 0, IRQ 41
        I/O ports at 2000 [size=256]
        Memory at c0404000 (64-bit, prefetchable) [size=4K]
        Memory at c0400000 (64-bit, prefetchable) [size=16K]
```

```
Capabilities: <access denied>
Kernel driver in use: r8169
Kernel modules: r8169
```

В листинге 5.66 показан типичный пример самого распространенного варианта использования **sed(1)** с командой **s** — замены одних подстрок другими. При помощи трех нехитрых подстановок отсортированный командой **sort(1)** вывод команды **find(1)** превращается в дерево файлов и каталогов, аналогичное тому, что строит команда **tree(1)**. Команда подстановки подстрок имеет форму **/RE/s/FND/RPL/F**, при этом во всех строках, соответствующих **RE**, будут найдены подстроки, соответствующие **FND**, и заменены на **RPL** согласно флагам замены **F**. Регулярное выражение **RE** может отсутствовать, тогда замены производятся во всех строчках, а регулярные выражения **FND** и **RPL** могут быть ограничены любым другим символом, например **:** вместо **/**. Флаги уточняют место замены. Например, флаг **g** (**g**lobal) требует произвести столько замен, сколько будет найдено слева направо соответствий выражению **FND**, а флаг **i** (case **i**nsensitive) указывает на нечувствительность выражения **FND** к регистру букв.

Таким образом, подстановка **s:^..::** выполняет удаление **^..** двух любых символов в начале строки (замену на «ничто»). Это отрежет ненужные префиксы **./** относительных путевых имен, выводимых как результат **find(1)**. Подстановка **s:[^/.][^/]*$:+-- &:** заменяет подстроки, состоящие из любого символа кроме слеша и точки **[^/.]** и следующих за ним любого количества ***** любых кроме слеша **[^/]** символов вплоть до конца строки **$**, на то же самое значение **&**, перед которым поставлены символы **+--**. Такое действие изобразит имена найденных **find(1)** файлов и каталогов как **+-- file**. Последняя подстановка **s:[^/]*/:| :g** глобально заменит подстроки из любого количества ***** символов кроме слеша **[^/]**, заканчивающиеся слешем **/**, на вертикальную черту с тремя пробелами **| **, что заменит элементы путевого имени найденных файлов в нужное количество уровней иерархии дерева в выводе.

Листинг 5.66. Замена подстрок: дерево файлов и каталогов

```
bender@ubuntu:~$ cd /lib/terminfo
bender@ubuntu:/lib/terminfo$ find . | sort -f |
> sed -e 's:^..::' -e 's:[^/.][^/]*$:+-- &:' -e 's:[^/]*/:|  :g'
.
+-- a
|  +-- ansi
+-- c
|  +-- cons25
          ...          ...          ...          ...          ...
```

```
+-- v
|   +-- vt100
|   +-- vt102
|   +-- vt220
|   +-- vt52
        ...          ...          ...          ...          ...
+-- x
|   +-- xterm
        ...          ...          ...          ...          ...
|   +-- xterm-vt220
|   +-- xterm-xfree86
```

Еще один типичный пример использования потокового редактора **sed(1)** проиллюстрирован в листинге 5.67, где во всех модулях некоторой программы на языке **W:[python]** имя модуля **module** нужно заменить его новым именем **Module**. Найденные командой **find(1)** имена модулей ***.py** проверяются командой **grep(1)** на предмет наличия в их тексте слова (**-w**) **module**, затем их имена (**-l**) отправляются потоковому редактору, который выполняет замену подстрок, замещая (**-i**) содержимое непосредственно обрабатываемого файла. Замены производятся в строках, содержащих слово **import**, при этом только подстроки **module**, слева **\<** и справа **\>** от которых находятся границы слов, т. е. только слова **module**, заменяются на **Module**.

Листинг 5.67. Массовая замена подстрок в файлах

```
bender@ubuntu:~$ find . -name '*.py' | xargs grep -wl module |
> xargs sed -i '/import/ s:\<module\>:Module:g'
```

В очень большом количестве случаев **sed(1)** применяется для редактирования конфигурационных файлов системы, например в сценариях установки и удаления пакетов программного обеспечения. В примере из листинга 5.68 показан небольшой сценарий интерпретатора, позволяющий «включать»/«выключать» параметры в конфигурационном файле **/etc/sysctl.conf**, используя удаление/установку символа комментария **#** в начало строк при помощи **sed(1)**. Для этого в трех инструкциях **sed(1)** используются две команды подстановки **s** и команда ветвления **t** (test). Первая подстановка **s/^#//** удаляет комментарий из начала строки, а вторая подстановка **s/^/#/** добавляет его в начало. Команда **t** тестирует результат выполнения первой инструкции подстановки и в случае успеха передает управление на конец списка инструкций. В результате выполнения инструкций либо комментарий удаляется из начала строки первой инструкцией, если он там присутствует (на чем список инструкций завершается), либо комментарий добавляется в начало строки второй инструкцией.

Листинг 5.68. Редактирование конфигурационных файлов

```
bender@ubuntu:~$ grep net.ipv4.ip_forward /etc/sysctl.conf
#net.ipv4.ip_forward=1
bender@ubuntu:~$ sudo sed -i '/net.ipv4.ip_forward/ s/^#//' /etc/sysctl.conf
bender@ubuntu:~$ grep net.ipv4.ip_forward /etc/sysctl.conf
net.ipv4.ip_forward=1
bender@ubuntu:~$ cat /usr/local/bin/toggle-comment
#!/bin/sh
[ -n "$1" -a -f "$2" ] &&
  sed -i -e "/$1/ s/^#//" -e t -e "/$1/ s/^/#/" $2 ||
  echo "Usage: toggle-comment RE file"
bender@ubuntu:~$ sudo toggle-comment net.ipv4.ip_forward /etc/sysctl.conf
bender@ubuntu:~$ grep net.ipv4.ip_forward /etc/sysctl.conf
#net.ipv4.ip_forward=1
```

Несмотря на простоту инструментальных средств обработки текста, таких как **awk(1)** и **sed(1)**, энтузиасты реализуют с их помощью (вкупе с управляющими **ESC**-последовательностями терминала и символами кодировки **utf-8**) даже простые игры, такие как крестики-нолики (**tic-tac-toe**), тетрис (**tetris**), **sokoban**, **arkanoid**, **2048** и даже (!) шахматы[1].

Более подробное изложение концепций и практик применения регулярных выражений, **sed(1)** и **awk(1)**, можно найти в книгах **ISBN**:[1-56592-225-5], **ISBN**:[5-93286-121-5], а полноценный курс программирования на языке командного интерпретатора — в **ISBN**:[5-7315-0114-9] и **ISBN**:[5-93286-029-4].

5.9. В заключение

Командный интерпретатор вместе с утилитами обработки текста формирует среду, позволяющую практически без ограничений решать разнообразные задачи автоматизации, что и находит широкое применение в виде соответствующих сценариев в операционной системе. Сценарии применяются практически повсеместно — при запуске и останове служб операционной системы, при постинсталляционном конфигурировании установленных пакетов программного обеспечения, при компиляции и компоновке программ и т. д.

Располагая таким могучим инструментом, командный интерфейс перестает быть для пользователя просто интерактивным способом взаимодействия с операционной

[1] См. https://habrahabr.ru/post/191006/.

системой, а превращается в полноценное средство разработки решений его произвольных задач. Вместе с освоением языка командного интерпретатора сам пользователь шаг за шагом превращается из чужака и пришельца в нативного обитателя этой экосистемы, аборигена цифровых джунглей Linux. Для такого пользователя командный интерфейс больше не представляется рудиментом и тяжким наследием прошлого, а *дополняет* графический интерфейс до единого гармоничного целого.

Основной вид профессиональной деятельности такого пользователя не имеет особого значения. Фотографы и дизайнеры, аудио- и видеоинженеры, 3D-моделеры и инженеры САПР, типографские работники и прочие профессионалы находят свою прелесть в написании и использовании сценариев пакетной обработки своих фотографий, аудио- и видеоматериалов, моделей, чертежей и массы другой информации. Вручив ежедневную рутину командному интерпретатору, они переходят на следующий уровень развития, где в полную силу посвящают себя решению творческих задач.

Не желаете присоединиться?

Глава 6

Сетевая подсистема

Сетевая подсистема Linux организует сетевой обмен пользовательских приложений и, как следствие, сетевое взаимодействие самих пользователей. Часть сетевой подсистемы, выполняющаяся в режиме *ядра*, естественным образом ответственна за управление сетевыми *устройствами* ввода-вывода, но кроме этого на нее также возложены задачи маршрутизации и транспортировки пересылаемых данных, которые решаются при помощи соответствующих сетевых *протоколов*. Таким образом, именно ядерная часть сетевой подсистемы обеспечивает процессы средствами *сетевого* межпроцессного взаимодействия (network IPC).

Внеядерная часть сетевой подсистемы отвечает за реализацию сетевых служб, предоставляющих пользователям *прикладные* сетевые услуги, такие как передача файлов, обмен почтовыми сообщениями, удаленный доступ и т. д.

6.1. Сетевые интерфейсы, протоколы и сетевые сокеты

Непосредственное, *физическое* взаимодействие сетевых узлов через каналы связи между ними реализуется аппаратурой сетевых адаптеров. Сетевые адаптеры, как и любые другие устройства ввода-вывода, управляются соответствующими драйверами (листинг 6.1), реализующимися в большинстве случаев в виде динамических модулей ядра.

Листинг 6.1. Драйверы сетевых устройств

```
lumpy@ubuntu:~$ lspci
          ...           ...           ...           ...           ...
01:00.0 Network controller: Intel Corporation Centrino Wireless-N 130 (rev 34)
02:00.0 Ethernet controller: ..., Ltd. RTL8111/8168B PCI Express ... Ethernet controller ...
lumpy@ubuntu:~$ lspci -ks 01:00.0
01:00.0 Network controller: Intel Corporation Centrino Wireless-N 130 (rev 34)
     Subsystem: Intel Corporation Centrino Wireless-N 130 BGN
```

```
        Kernel driver in use: iwlwifi
        Kernel modules: iwlwifi ☚
lumpy@ubuntu:~$ lspci -ks 02:00.0
02:00.0 Ethernet controller: ..., Ltd. RTL8111/8168B PCI Express ... Ethernet controller ...
        Subsystem: Samsung Electronics Co Ltd Device c0b6
        Kernel driver in use: r8169
        Kernel modules: r8169 ☚
lumpy@ubuntu:~$ modinfo iwlwifi r8169 | grep ^description
description:    Intel(R) Wireless WiFi driver for Linux
description:    RealTek RTL-8169 Gigabit Ethernet driver
```

В отличие от несетевых устройств, большинство которых имеют специальный файл в каталоге **/dev**, сетевые устройства представляются в системе своими *интерфейсами*. Список доступных интерфейсов, их параметры и статистику можно получить при помощи «классической» UNIX-команды **ifconfig(8)** или специфичной для Linux команды **ip(8)** (листинги 6.2 и 6.3).

Листинг 6.2. Сетевые интерфейсы (UNIX ifconfig(8))

```
lumpy@ubuntu:~$ ifconfig -a
    eth0      Link encap:Ethernet  HWaddr e8:03:9a:0a:73:40
              UP BROADCAST MULTICAST  MTU:1500  Metric:1
              ...          ...          ...          ...          ...

    lo        Link encap:Локальная петля (Loopback)
              inet addr:127.0.0.1  Mask:255.0.0.0
              inet6 addr: ::1/128 Scope:Host
              UP LOOPBACK RUNNING  MTU:65536  Metric:1
              ...          ...          ...          ...          ...

    wlan0     Link encap:Ethernet  HWaddr b8:03:05:a2:28:4e
              inet addr:192.168.100.5  Bcast:192.168.100.255  Mask:255.255.255.0
              inet6 addr: fe80::ba03:5ff:fea2:284e/64 Scope:Link
              UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
              ...          ...          ...          ...          ...
lumpy@ubuntu:~$ ifconfig wlan0
wlan0     Link encap:Ethernet  HWaddr b8:03:05:a2:28:4e
☚           inet addr:192.168.100.5  Bcast:192.168.100.255  Mask:255.255.255.0
          inet6 addr: fe80::ba03:5ff:fea2:284e/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:23103842 errors:0 dropped:0 overruns:0 frame:0
          TX packets:15591575 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2010967763 (2.0 GB)  TX bytes:2828583098 (2.8 GB)
```

Листинг 6.3. Сетевые интерфейсы (Linux ip(8))

```
lumpy@ubuntu:~$ ip link show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state DOWN qlen 1000
    link/ether e8:03:9a:0a:73:40 brd ff:ff:ff:ff:ff:ff
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP qlen 1000
    link/ether b8:03:05:a2:28:4e brd ff:ff:ff:ff:ff:ff
lumpy@ubuntu:~$ ip -s link show dev wlan0
    3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP qlen 1000
        link/ether b8:03:05:a2:28:4e brd ff:ff:ff:ff:ff:ff
        RX: bytes  packets  errors  dropped overrun mcast
        2026956879 23261031 0       0       0       0
        TX: bytes  packets  errors  dropped carrier collsns
        3118714118 15863377 0       0       0       0
lumpy@ubuntu:~$ ip addr show dev wlan0
    3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP qlen 1000
        link/ether b8:03:05:a2:28:4e brd ff:ff:ff:ff:ff:ff
        inet 192.168.100.5/24 brd 192.168.100.255 scope global wlan0
           valid_lft forever preferred_lft forever
        inet6 fe80::ba03:5ff:fea2:284e/64 scope link
           valid_lft forever preferred_lft forever
```

За *логическое* взаимодействие (адресацию, маршрутизацию, обеспечение надежной доставки и пр.) отвечают сетевые протоколы, тоже в большинстве случаев реализующиеся соответствующими модулями ядра. Нужно отметить, что в примере из листинга 6.4 показан список *динамически* загруженных модулей, среди которых присутствует «нестандартный» TCP vegas, но нет IP, TCP, UDP и прочих «стандартных» протоколов стека TCP/IP. На текущий момент времени сложно вообразить применение Linux без подключения к IP-сети, поэтому модули стандартных протоколов TCP/IP стека скомпонованы в ядро *статически* и являются частью «стартового» *(см. разд. 4.1.1)* модуля.

Листинг 6.4. Драйверы сетевых протоколов

```
lumpy@ubuntu:~$ lsmod
Module                  Size  Used by
tcp_vegas              13603  0
esp4                   12868  0
```

```
ah4                    12800  0
xfrm_algo              14869  2 esp4,ah4
          ...            ...             ...             ...             ...
bnep                   19107  2
          ...            ...             ...             ...             ...
bluetooth             356727  24 bnep,rfcomm,btusb
          ...            ...             ...             ...             ...
mac80211              564463  1 iwldvm
          ...            ...             ...             ...             ...
lumpy@ubuntu:~$ modinfo tcp_vegas bnep mac80211 | grep ^description
description:    TCP Vegas
description:    Bluetooth BNEP ver 1.3
description:    IEEE 802.11 subsystem
```

Доступ процессов к услугам ядерной части сетевой подсистемы реализует интерфейс *сетевых сокетов* **socket(7)**, являющихся основным (и единственным) средством сетевого взаимодействия процессов в Linux.

Разные *семейства* (address family) сокетов соответствуют различным стекам сетевых протоколов. Например, стек TCP/IP v4 представлен семейством **AF_INET**, см. **ip(7)**, стек TCP/IP v6 — семейством **AF_INET6**, см. **ipv6(7)**, и даже локальные (файловые) сокеты имеют собственное семейство — **AF_LOCAL**, см. **unix(7)**.

Для просмотра статистики по использованию сетевых сокетов применяют «классическую» **UNIX**-команду **netstat(8)** или специфичную для Linux команду **ss(8)**. В листингах 6.5 и 6.6 иллюстрируется использование этих команд для вывода информации обо всех (**-a**, all) сокетах протоколов (**-u**, udp) UDP и (**-t**, tcp) TCP стека TCP/IP v4 (**-4**), порты и адреса которых выведены в числовом (**-n**, numeric) виде, а также изображены процессы (**-p**, process), их открывшие.

Листинг 6.5. Сетевые сокеты (UNIX netstat(8))

```
lumpy@ubuntu:~$ sudo netstat -4autpn
Активные соединения с интернетом (servers and established)
Proto Recv-Q Send-Q Local Address  Foreign Address  State       PID/Program name
tcp        0      0 0.0.0.0:5900   0.0.0.0:*        LISTEN      28880/vino-server
tcp        0      0 127.0.0.1:53   0.0.0.0:*        LISTEN      5999/dnsmasq
tcp        0      0 0.0.0.0:22     0.0.0.0:*        LISTEN      655/sshd
tcp        0      0 127.0.0.1:631  0.0.0.0:*        LISTEN      1064/cupsd
tcp        0      0 192.168.100.5:22 ➥ 192.168.100.3:57929 ➥ ESTABLISHED 15841/sshd: ...
udp        0      0 0.0.0.0:5353   0.0.0.0:*                    15094/avahi-daemon:
udp        0      0 0.0.0.0:46347  0.0.0.0:*                    15094/avahi-daemon:
udp        0      0 127.0.0.1:53   0.0.0.0:*                    5999/dnsmasq
udp        0      0 0.0.0.0:68     0.0.0.0:*                    5995/dhclient
```

Сетевые сокеты идентифицируются парой адресов (собственным, local, и чужим[1], foreign), принятыми в их семействе. Например, для семейства TCP/IP адрес сокета состоит из (сетевого) IP-адреса и (транспортного) номера порта, причем нули имеют специальное — «неопределенное» значение. Так для прослушивающего[2] (LISTEN) сокета **0.0.0.0** в собственном IP-адресе означает, что он принимает соединения, направленные на любой адрес любого сетевого интерфейса, а **0.0.0.0** в чужом адресе указывает на то, что взаимодействие еще не установлено. Для сокетов с установленным (ESTABLISHED) взаимодействием оба адреса имеют конкретные значения, определяющие участников взаимодействия, например **192.168.100.5:22** и **192.168.100.3:57929**.

Листинг 6.6. Сетевые сокеты (Linux ss(8))

```
lumpy@ubuntu:~$ sudo ss -4atupn
Netid  State    Recv-Q Send-Q  Local Address:Port  Peer Address:Port
udp    UNCONN   0      0             *:5353             *:*     users:(("avahi-daemon",15094,13))
udp    UNCONN   0      0             *:46347            *:*     users:(("avahi-daemon",15094,15))
udp    UNCONN   0      0       127.0.0.1:53             *:*     users:(("dnsmasq",5999,4))
udp    UNCONN   0      0             *:68               *:*     users:(("dhclient",5995,6))
tcp    LISTEN   0      5             *:5900 ☚           *:*     users:(("vino-server",28880,14))
tcp    LISTEN   0      5       127.0.0.1:53             *:*     users:(("dnsmasq",5999,5))
tcp    LISTEN   0      128           *:22 ☚             *:*     users:(("sshd",655,4))
tcp    LISTEN   0      128     127.0.0.1:631            *:*     users:(("cupsd",1064,10))
tcp    LISTEN   0      100     127.0.0.1:25             *:*     users:(("master",2021,12))
```

Из примеров листингов 6.5 и 6.6. видны 5 «слушающих» (LISTEN) сокетов TCP и 4 «несоединенных» (UNCONN) сокета UDP, открытых разными службами операционной системы. Например, 22 порт TCP открыл сервер **sshd**, PID = 655 службы удаленного доступа W:[SSH], а 5900 порт TCP — сервер **vino-server**, PID = 28880 службы удаленного доступа к рабочему столу пользователя W:[VNC].

[1] Адрес удаленного приложения, с которым установлено соединение.

[2] Прослушивающие сокеты используются «серверными» приложениями, пассивно ожидающими входящие соединения с ними.

6.2. Конфигурирование сетевых интерфейсов и протоколов

6.2.1. Ручное конфигурирование

Для функционирования разных стеков протоколов сетевым интерфейсам должны быть предварительно назначены корректные сетевые адреса и сконфигурированы прочие параметры, что может быть выполнено вручную администратором или автоматически специальными службами этих стеков.

Ручное назначение сетевых адресов стека TCP/IP выполняется при помощи команд **ifconfig(8)** или **ip(8)**, а простейшая диагностика — при помощи команды **ping(1)**, как проиллюстрировано в листинге 6.7.

Листинг 6.7. Ручное конфигурирование сетевых интерфейсов

```
lumpy@ubuntu:~$ sudo ifconfig eth0 10.0.0.10 up
lumpy@ubuntu:~$ sudo ifconfig eth0
eth0      Link encap:Ethernet  HWaddr e8:03:9a:0a:73:40
          inet addr:10.0.0.10  Bcast:10.255.255.255  Mask:255.0.0.0
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:2103481 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2510799 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:636720606 (636.7 MB)  TX bytes:3451876716 (3.4 GB)
lumpy@ubuntu:~$ ping -c 1 10.0.0.10
PING 10.0.0.1 (10.0.0.10) 56(84) bytes of data.
64 bytes from 10.0.0.10: icmp_req=1 ttl=64 time=0.134 ms

--- 10.0.0.10 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.134/0.134/0.134/0.000 ms

lumpy@ubuntu:~$ sudo ip address add 172.16.16.172/16 dev eth0
lumpy@ubuntu:~$ ip address show dev eth0
2: eth0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state DOWN qlen 1000
    link/ether e8:03:9a:0a:73:40 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.10/8 brd 10.255.255.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet 172.16.16.172/16 scope global eth0
       valid_lft forever preferred_lft forever
```

```
lumpy@ubuntu:~$ ping -c 1 172.16.16.172
PING 172.16.0.1 (172.16.16.172) 56(84) bytes of data.
64 bytes from 172.16.16.172: icmp_req=1 ttl=64 time=0.121 ms

--- 172.16.16.172 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.121/0.121/0.121/0.000 ms
```

Просмотр таблиц маршрутизации и ручное конфигурирование IP-маршрутов выполняется при помощи команд **route(8)** или **ip(8)**, а простейшая диагностика — при помощи команды **traceroute(1)**. В примере из листинга 6.8 показана процедура ручного добавления маршрута «по умолчанию» через интернет-шлюз **10.0.0.1** с последующей диагностикой доступности узлов за ним.

Листинг 6.8. Ручное конфигурирование таблицы маршрутизации

```
lumpy@ubuntu:~$ sudo ip route add 0.0.0.0/0 ❶ via 10.0.0.1 ❷
lumpy@ubuntu:~$ route -n
Таблица маршутизации ядра протокола IP
Destination Gateway Genmask Flags Metric Ref Use Iface
0.0.0.0 ❶       10.0.0.1    ❷ 0.0.0.0         UG    0      0        0 eth0
10.0.0.0        0.0.0.0         255.0.0.0       U     0      0        0 eth0
172.16.0.0      0.0.0.0         255.255.0.0     U     0      0        0 eth0
lumpy@ubuntu:~$ ip route show
default ❶ via 10.0.0.1 ❷ dev eth0  proto static
10.0.0.0/8 dev eth0  proto kernel  scope link  src 10.0.0.1
172.16.0.0/16 dev eth0  proto kernel  scope link  src 172.16.0.1
lumpy@ubuntu:~$ traceroute bad.horse
traceroute to bad.horse (162.252.205.157), 30 hops max, 60 byte packets
 1  10.0.0.1 (10.0.0.1)  1.025 ms  1.080 ms  1.236 ms
          ...          ...          ...          ...          ...
16  bad.horse (162.252.205.130)  231.840 ms  230.451 ms  230.294 ms
17  bad.horse (162.252.205.131)  225.436 ms  229.919 ms  234.079 ms
18  bad.horse (162.252.205.132)  236.432 ms  224.638 ms  237.708 ms
19  bad.horse (162.252.205.133)  247.277 ms  244.117 ms  230.279 ms
20  he.rides.across.the.nation (162.252.205.134)  248.204 ms  236.705 ms  233.131 ms
21  the.thoroughbred.of.sin (162.252.205.135)  264.369 ms  261.428 ms  274.971 ms
22  he.got.the.application (162.252.205.136)  261.681 ms  264.942 ms  264.636 ms
23  that.you.just.sent.in (162.252.205.137)  252.373 ms  264.443 ms  248.169 ms
24  it.needs.evaluation (162.252.205.138)  269.286 ms  266.914 ms  267.678 ms
25  so.let.the.games.begin (162.252.205.139)  269.224 ms  260.428 ms  274.339 ms
```

```
26 a.heinous.crime (162.252.205.140)  297.513 ms  289.543 ms  292.710 ms
27 a.show.of.force (162.252.205.141)  273.900 ms  290.666 ms  295.939 ms
28 a.murder.would.be.nice.of.course (162.252.205.142)  292.588 ms  279.658 ms  276.165 ms
29 bad.horse (162.252.205.143)  282.689 ms  292.198 ms  283.833 ms
30 bad.horse (162.252.205.144)  301.530 ms  286.082 ms  302.532 ms
```

6.2.2. Автоматическое конфигурирование

За автоматическое конфигурирование сетевых интерфейсов отвечает менеджер сетевых подключений — системная служба **networkmanager(8)**, отслеживающая физическую активацию сетевых адаптеров (подключение сетевого кабеля Ethernet или поключения к сети Wi-Fi) и запускающая «подчиненные» службы, в частности W:[DHCP]-клиент **dhclient(8)** для автоматического получения IP-адреса и простейший W:[DNS]-сервер **dnsmasq(8)** для подключения к доменной системе имен.

Для взаимодействия с менеджером сетевых подключений предназначены команды **nmcli(1)**, **nm-tool(1)** и GUI-приложение **nm-applet(1)**, позволяющие опрашивать его состояние и управлять его действиями.

Листинг 6.9. Конфигурирование сетевых интерфейсов (автоматически)

```
lumpy@ubuntu:~$ ps axfwwww
          ...             ...             ...             ...             ...
1121  ?   Ssl    0:04 NetworkManager
15537 ?   S      0:00  \_ /sbin/dhclient -d -4 ... wlan0
15541 ?   S      0:00  \_ /usr/sbin/dnsmasq ... --conf-file=/var/run/nm-dns-dnsmasq.conf ...
          ...             ...             ...             ...             ...
lumpy@ubuntu:~$ nmcli dev
УСТРОЙСТВО ТИП              СОСТОЯНИЕ
eth0       802-3-ethernet   отключено
wlan0 ☚    802-11-wireless  подключено
lumpy@ubuntu:~$ nmcli dev list iface wlan0
GENERAL.УСТРОЙСТВО:                 wlan0
          ...             ...             ...             ...             ...
GENERAL.АППАРАТНЫЙ АДРЕС:  B8:03:05:A2:28:4E
GENERAL.СОСТОЯНИЕ:                  100 (подключено)
          ...             ...             ...             ...             ...
IP4.АДРЕС[1]:                       ip = 192.168.100.4/24 ❶, gw = 192.168.100.1 ❷
IP4.DNS[1]:                             192.168.100.1 ❸
          ...             ...             ...             ...             ...
lumpy@ubuntu:~$ ip address show dev wlan0
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP qlen 1000
    link/ether b8:03:05:a2:28:4e brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.4/24 ① brd 192.168.100.255 scope global wlan0
       valid_lft forever preferred_lft forever
```

```
    inet6 fe80::ba03:5ff:fea2:284e/64 scope link
       valid_lft forever preferred_lft forever
lumpy@ubuntu:~$ ip route show
default via 192.168.100.1 ② dev wlan0  proto static
          ...          ...          ...          ...          ...
lumpy@ubuntu:~$ cat /var/run/nm-dns-dnsmasq.conf
server=192.168.100.1 ③
```

В листинге 6.9 проиллюстрирован результат работы менеджера подключений при подключении к Wi-Fi-сети сетевого интерфейса **wlan0**. Полученные при помощи DHCP-клиента конфигурационные параметры были активированы автоматически: IP-адреса и маска ❶ назначены на интерфейс ①, шлюз «по умолчанию» ❷ задан в соответствующем маршруте ②, а адрес DNS-сервера ❸ указан в конфигурационном файле ③ DNS-сервера.

6.3. Служба имен и DNS/mDNS-резолверы

Основными идентификаторами сетевого взаимодействия в стеке протоколов TCP/IP являются числа — IP-*адреса* узлов и *номера* портов TCP/UDP, «человеческое» использование которых достаточно[1] неудобно. Использование строковых *имен* для узлов и портов приводит к необходимости отображать «человеческие» имена в «протокольные» числа и обратно, что возложено на *службу имен* (name service). Как указывалось ранее, служба имен вообще предназначается для организации доступа приложений к свойствам каталогизируемых сущностей по их имени: к UID пользователя по имени его учетной записи, к IP-адресу по имени узла, к номеру порта TCP/UDP по имени сетевой службы, использующей его, и т. д.

Отображение имен сущностей на их свойства зачастую выполняется разными способами в разных каталогах, что определяется конфигурацией коммутатора службы имен **nsswitch.conf(5)** (name service switch **conf**iguration) и наличием ее соответствующих модулей **lib*nss*_*.so.?**.

В листинге 6.10 иллюстрируется такая конфигурация отображения имен *узлов* **hosts** ⓪, которая использует сначала файловую таблицу ① **/etc/hosts** — см. **hosts(5)**, а затем — службы **W**:[mDNS] ②④ и **W**:[DNS] ③.

Аналогично, номера портов сетевых *служб* **services** отображаются сначала с использованием файла индексированной базы данных (соответствующий модуль **libnss_db.so.2** оказался не установлен), а затем с использованием файловой таблицы **/etc/services** — см. **services(5)**.

---

[1] Если с 32-битным IPv4-адресом еще можно было совладать, то 128-битный адрес IPv4 не оставляет человеку практически никаких шансов.

Листинг 6.10. Служба имен и ее модули

```
lumpy@ubuntu:~$ grep hosts /etc/nsswitch.conf
⓪              ①↴   ②↴                          ③↴ ④↴
hosts:         files mdns4_minimal [NOTFOUND=return] dns mdns4
lumpy@ubuntu:~$ grep services /etc/nsswitch.conf
services:      db files
lumpy@ubuntu:~$ find /lib -name 'libnss*'
               ...          ...          ...          ...          ...
❸ /lib/i386-linux-gnu/libnss_dns.so.2
❶ /lib/i386-linux-gnu/libnss_files.so.2
               ...          ...          ...          ...          ...
❹ /lib/libnss_mdns4.so.2
❷ /lib/libnss_mdns4_minimal.so.2
               ...          ...          ...          ...          ...
```

Файловые таблицы имен **/etc/hosts** и **/etc/services** имеют тривиальный формат ❶❷, сопоставляющий имена узлов и сервисов — их IP-адресам и портам протоколов TCP и UDP, что проиллюстрировано в листинге 6.11. Утилита службы имен **getent(1)**, позволяющая выбирать указанную сущность по ее типу и имени, используется в качестве диагностики ❸ коммутатора службы имен и его модулей.

Листинг 6.11. Файловые таблицы имен

```
lumpy@ubuntu:~$ cat /etc/hosts
❶ 127.0.0.1     localhost
   127.0.1.1     ubuntu

   # The following lines are desirable for IPv6 capable hosts
   ::1     ip6-localhost ip6-loopback
   fe00::0 ip6-localnet
   ff00::0 ip6-mcastprefix
   ff02::1 ip6-allnodes
   ff02::2 ip6-allrouters
   lumpy@ubuntu:~$ grep http /etc/services
   # Updated from http://www.iana.org/assignments/port-numbers and other
   # sources like http://www.freebsd.org/cgi/cvsweb.cgi/src/etc/services .
❷ http          80/tcp          www             # WorldWideWeb HTTP
   http          80/udp                          # HyperText Transfer Protocol
   https         443/tcp                         # http protocol over TLS/SSL
   https         443/udp
   http-alt      8080/tcp webcache # WWW caching service
   http-alt      8080/udp
```

```
❸ lumpy@ubuntu:~$ getent hosts ubuntu
  127.0.1.1       ubuntu
  lumpy@ubuntu:~$ getent services 53/udp
  domain                  53/tcp
```

Соответствия IP-адресов именам «серверных» узлов, например публичных Web-, почтовых и прочих серверов, обычно регистрируются их администраторами в «таблицах» на *ответственных* (authoritative) серверах службы DNS. Для доступа к ним соответствующий модуль службы имен (см. ③, листинг 6.10) использует стандартный DNS-клиент (**resolver(3)**, DNS-резолвер), которому указываются (в конфигурационном файле **resolv.conf(5)**) IP-адреса ближайших *кэширующих* DNS-серверов, например серверов провайдера услуг Интернета.

В примере из листинга 6.12 показано, что в качестве кэширующего сервера выступает простейший DNS-сервер **dnsmasq(8)**, который автоматически настраивается на «вышестоящие» кэширующие DNS-серверы и (пере)запускается менеджером сетевых подключений при каждой активации нового соединении.

Для диагностики DNS-модуля службы имен (равно как и любого другого ее модуля) используется команда **getent(8)**, а для непосредственной диагностики DNS-серверов — команда **host(1)**.

Листинг 6.12. DNS-клиент

```
lumpy@ubuntu:~$ cat /etc/resolv.conf
# Dynamic resolv.conf(5) file for glibc resolver(3) generated by resolvconf(8)
#     DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
☛ nameserver 127.0.0.1
lumpy@ubuntu:~$ sudo ss -4autpn sport = :domain
Netid  State    Recv-Q Send-Q  Local Address:Port  Peer Address:Port
udp    UNCONN   0      0       127.0.0.1:53 ☚      *:* users:(("dnsmasq",5999,4))
tcp    LISTEN   0      5       127.0.0.1:53        *:* users:(("dnsmasq",5999,5))
lumpy@ubuntu:~$ getent hosts bhv.ru
☛ 85.249.46.2     bhv.ru
lumpy@ubuntu:~$ host bhv.ru
bhv.ru has address 85.249.46.2 ☚
bhv.ru mail is handled by 50 relay2.peterlink.ru.
bhv.ru mail is handled by 30 relay1.peterlink.ru.
lumpy@ubuntu:~$ host 8.8.8.8
8.8.8.8.in-addr.arpa domain name pointer google-public-dns-a.google.com.
```

Сетевые устройства (принтеры, камеры, видеорегистраторы и пр.) и «клиентские» узлы локальных сетей, динамически получающие случайные IP-адреса при помощи DHCP, на ответственных серверах DNS почти никогда не регистрируются, а использование их имен в локальной сети (в «домене» **.local**) становится возможным благодаря службе mDNS.

Серверы mDNS запускаются на каждом «клиентском» узле и регистрируют у себя соответствия собственных IP-адресов своему имени, а затем используют многоадресную (**m**ulticast) рассылку стандартных запросов DNS для получения информации друг у друга. Сервером mDNS, как показано в примере из листинга 6.13, является **avahi-daemon(8)**, реализующий еще и службу DNS-SD (DNS **s**ervice **d**iscovery), которая позволяет узлам локальной сети обнаруживать (discovery) услуги (service), предоставляемые другими узлами. При помощи **avahi-browse(1)** проиллюстрирован список всех (**-a**, all) имен ❶ и типов ❷ услуг, объявленных узлами сети и сохраненных в локальном кэше (**-c**, **c**ache), а также результаты (**-r**, **r**esolve) запросов на получение информации об услугах.

Листинг 6.13. mDNS/DNS-SD-клиент

```
lumpy@ubuntu:~$ sudo ss -4autpn sport = :mdns
Netid  State    Recv-Q Send-Q  Local Address:Port  Peer Address:Port
udp    UNCONN   0      0       *:5353 ◄           *:* users:(("avahi-daemon",15094,13))

lumpy@ubuntu:~$ avahi-browse -arcl
+   eth0 IPv4 HP LaserJet 700 M712 [4C6BF5] ◄❶ ❷► UNIX Printer                local
          ...              ...              ...              ...              ...
+   eth0 IPv4 AXIS 211M - 00408C81D401              RTSP Realtime Streaming Server local
          ...              ...              ...              ...              ...
+   eth0 IPv4 NVR(SMB)                              Microsoft Windows Network      local
+   eth0 IPv4 NVR(NFS)                              Network File System            local
          ...              ...              ...              ...              ...
=   eth0 IPv4 HP LaserJet 700 M712 [4C6BF5]         UNIX Printer                   local
   hostname = [NPI4C6BF5.local]
   address = [192.168.17.68]
   port = [515]
   txt = ["Scan=F" "UUID=56ff35dd-1065-4e5e-9385-3c26b8946b79" "Color=F" "Duplex=T" "Binary=T"
"Transparent=T" "note=" "adminurl=http://NPI4C6BF5.local." "priority=40" "usb_MDL=HP LaserJet
700 M712" "usb_MFG=Hewlett-Packard" "product=(HP LaserJet 700 M712)" "ty=HP LaserJet 700 M712"
"URF=V1.1,CP99,RS600,MT1-2-3-5-12,W8,PQ4,IS20-21-22-23,DM1,OB1" "pdl=application/postscript"
"rp=BINPS" "qtotal=4" "txtvers=1"]
          ...              ...              ...              ...              ...
=   eth0 IPv4 AXIS 211M - 00408C81D401              RTSP Realtime Streaming Server local
   hostname = [axis-00408c81d401.local]
   address = [192.168.17.142]
   port = [554]
```

```
   txt = ["path=mpeg4/1/media.amp"]
=   eth0 IPv4 NVR(SMB)                          Microsoft Windows Network      local
   hostname = [NVR.local]
   address = [192.168.17.90]
   port = [445]
   txt = []
=   eth0 IPv4 NVR(NFS)                          Network File System            local
   hostname = [NVR.local]
   address = [192.168.17.90]
   port = [2049]
   txt = []

lumpy@ubuntu:~$ avahi-resolve --name NVR.local NPI4C6BF5.local axis-00408c81d401.local
NVR.local               192.168.17.90
NPI4C6BF5.local         192.168.17.68
axis-00408c81d401.local 192.168.17.142

lumpy@ubuntu:~$ getent hosts NVR.local NPI4C6BF5.local axis-00408c81d401.local
192.168.17.90    NVR.local
192.168.17.68    NPI4C6BF5.local
192.168.17.142   axis-00408c81d401.local
```

Для диагностики mDNS-модуля службы имен неизменно используется команда **getent(8)**, а для непосредственной диагностики mDNS-сервера **avahi-daemon(8)** — специальная команда **avahi-resolve(1)**.

6.4. Сетевые службы

6.4.1. Служба SSH

Служба **W:[SSH]** предназначена для организации *безопасного* (secure) доступа к сеансу командного интерпретатора (**sh**ell) удаленных сетевых узлов. Изначально разрабатывалась как замена *небезопасным* R-утилитам **W:[Rlogin]**, **W:[Rsh]** и протоколу сетевого алфавитно-цифрового терминала **W:[telnet]**.

При сетевом взаимодействии в «открытой» публичной сети, такой как Интернет, «безопасность» обычно понимают как *конфиденциальность* (плюс *целостность*) передаваемых данных и *аутентичность* (подлинность) взаимодействующих сторон.

Конфиденциальность данных, т. е. их недоступность некоторой третьей стороне, не участвующей во взаимодействии, обеспечивается в протоколе SSH при помощи

симметричного шифрования с использованием общего *сеансового* ключа. Симметричные алгоритмы шифрования используют для зашифровки и расшифровки информации один и тот же ключ, поэтому конфиденциальность целиком и полностью сводится к *секретности* ключа, т. е. его недоступности третьей стороне.

Сеансовый ключ устанавливается обеими взаимодействующими сторонами при помощи *асимметричного* алгоритма открытого согласования ключей (**W:[протокол Диффи — Хеллмана]**), использующего две пары дополнительных ключей. Обе стороны взаимодействия случайным образом выбирают *закрытые* ключи и на их основе вычисляют парные *открытые* ключи, которыми публично обмениваются. Общий (сеансовый) ключ *вычисляется* каждой стороной на основе *своего закрытого* ключа и *чужого открытого* ключа, что не может повторить третья сторона, т. к. может подслушать только передачу открытых ключей и не владеет[1] закрытыми ключами участников взаимодействия.

При активном вмешательстве третьей стороны во взаимодействие путем перехвата и подмены сообщений согласования ключей, злоумышленник может выдавать себя за другую сторону каждому из участников взаимодействия, став посредником, см. **W:[MITM]** (**m**an **i**n **t**he **m**iddle). В этом случае взаимодействующие стороны согласуют свои сеансовые ключи со злоумышленником (!), предполагая, что согласовали их с подлинным участником взаимодействия. Как следствие, все передаваемые данные «естественным» образом станут доступными злоумышленнику, причем наличие посредника никак не будет обнаружено.

Обеспечение подлинности (аутентичности) взаимодействующих сторон в протоколе SSH основывается на *аутентификации сервера* клиентом при помощи *асимметричных* алгоритмов цифровой подписи с использованием закрытого и открытого ключей *сервера*. Закрытый ключ используется для подписывания сообщений, а парный ему открытый ключ — для проверки подписи, корректность которой удостоверяет в том, что сообщение было сгенерировано подлинным владельцем закрытого ключа.

В сообщение протокола Диффи — Хеллмана сервер добавляет свой открытый ключ цифровой подписи (так называемый host key), а само сообщение подписывает закрытым ключом. Клиент извлекает этот открытый ключ из сообщения и с помощью проверки корректности его цифровой подписи удостоверяется в подлинности владельца вложенного ключа. Остается только убедиться, что владельцем вложенного ключа и является целевой сервер.

В примере из листинга 6.14 иллюстрируется первое присоединение к SSH-серверу **grex.org** (**162.202.67.158**), в результате чего был получен открытый ключ алгоритма

---

[1] Обратная задача вычисления закрытых ключей на основе открытых ключей практически не решаема, на чем и основывается вся асимметричная криптография.

W:[ECDSA], который, *возможно*, действительно принадлежит этому серверу, а не злоумышленнику посередине соединения. Единственный способ это проверить — *заранее знать* действительный открытый ключ SSH-сервера **grex.org** и побитно *сверить* его с присланным ключом, что достаточно затруднительно сделать человеку. Поэтому на практике сверяют короткие хэш-суммы действительного и присланного ключей, называемые «отпечатками пальца» (fingerprint), совпадение которых гарантирует совпадение[1] ключей. Хэш-суммы открытых ключей SSH-серверов заранее известны и открыто публикуются, например для **grex.org** — на Web-странице **http://grex.cyberspace.org/faq.xhtml#sshfinger**. После ручной сверки ключа ❶ при первом подключении он сохраняется в файле **~/.ssh/known_hosts** для автоматической сверки при последующих подключениях.

Листинг 6.14. Первое присоединение к SSH-серверу

```
☛ lumpy@ubuntu:~$ ssh jake@grex.org
  The authenticity of host 'grex.org (162.202.67.158)' can't be established.
  ECDSA key fingerprint is 30:82:50:ba:6c:26:68:01:52:64:dc:54:83:8e:95:7e.
  Are you sure you want to continue connecting (yes/no)? yes ↵ ❶
  Warning: Permanently added 'grex.org,162.202.67.158' (ECDSA) to the list of known hosts.
🗝 jake@grex.org's password: P@s$w0rd ↵ ❷
                ...          ...          ...          ...          ...
☛ grex$ uname -a ↵
  OpenBSD grex.org 5.0 GENERIC.MP#0 i386
  grex$ whoami ↵
  jake
  grex$ w ↵
   8:40AM  up 86 days,  8:33, 15 users, load averages: 1.41, 1.17, 1.03
  USER    TTY FROM                 LOGIN@  IDLE WHAT
  cross    p4 spitfire.i.gajen     23Dec15     0 -bash
                ...          ...          ...          ...          ...
  jake ❸☛pf 95-55-86-216.dyn☚❹  8:37AM     0 w
  pbbl     q6 192.94.73.30        Mon05AM     0 alpine
  ilk      q8 99-194-162-175.d     1:34AM  7:05 screen -r
  grex$ tty
  /dev/ttypf ❸
  grex$ logout ↵
  Connection to grex.org closed.
☛ lumpy@ubuntu:~$
```

[1] При этом подобрать другой ключ с такой же хэш-суммой практически невозможно.

После успешного установления соединения SSH пользовательский сеанс аутентифицируется одним из способов, например с помощью пароля ❷, а затем в интерактивном режиме запускается начальный командный интерпретатор аутентифицированного пользователя. При этом на стороне сервера используется псевдотерминал ❸ и эмулируется терминальный вход в систему *(см. разд. 2.2.1)*, считающийся сетевым ❹.

Листинг 6.15. Выполнение отдельной команды

```
lumpy@ubuntu:~$ ssh jake@grex.org uptime
jake@grex.org's password: P@sSW0rd ↵
 8:44AM  up 86 days,  8:37, 15 users, load averages: 1.33, 1.19, 1.06
lumpy@ubuntu:~$
```

Кроме интерактивного сетевого входа, SSH позволяет удаленно выполнять отдельные команды (см. листинг 6.15), при этом псевдотерминал не используется, а терминальный вход не эмулируется. Вместо этого стандартные потоки ввода-вывода **STDIN**, **STDOUT** и **STDERR** выполняемой команды просто перенаправляются через сетевое соединение ssh-клиенту. Это зачастую используется для удаленного копирования файлов. Например, в листинге 6.16 при помощи архиватора **tar(1)** создается сжатый архив каталога **/usr/src/sys** на *удаленном* узле **grex.org**, а результат перенаправляется в *локальный* файл **openbsd-kernel-source.tgz**.

Листинг 6.16. Копирование при помощи ssh

```
lumpy@ubuntu:~$ ssh jake@grex.org tar czf - /usr/src/sys > openbsd-kernel-source.tgz
jake@grex.org's password: P@sSW0rd ↵
tar: Removing leading / from absolute path names in the archive
                                ① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫
lumpy@ubuntu:~$
```

Аутентификация пользовательского сеанса по паролю имеет некоторое неудобство — необходимость вводить пароль заново при каждом новом соединении SSH. Кроме того, пароль передается внутри соединения SSH, поэтому существует угроза подбора пароля злоумышленником. Более прогрессивный способ аутентификации пользователя основан на использовании асимметричных алгоритмов цифровой подписи (аналогично аутентификации сервера в протоколе Диффи — Хеллмана) и ключей *пользователя*. Для этого открытый ключ пользователя единожды регистрируется на сервере, а при аутентификации подписывается его закрытым ключом и высылается серверу повторно. Сервер в свою очередь проверяет цифровую

подпись присланного ключа и удостоверяется в подлинности его владельца, а побитное сравнение с заранее зарегистрированным ключом пользователя указывает на то, что владелец присланного ключа и есть целевой пользователь.

В листинге 6.17 иллюстрируется процедура применения ключей аутентификации пользователя. Сначала при помощи команды **ssh-keygen(1)** генерируется закрытый (private) W:[RSA]-ключ в локальном файле **~/.ssh/id_rsa** и парный ему открытый (public) ключ в локальном файле **~/.ssh/id_rsa.pub**. Закрытый ключ защищается (шифруется) от несанкционированного использования парольной фразой (passphrase) ⓪, которая в отличие от пароля никогда по сети не передается, а лишь обеспечивает доступ к самому ключу. Нужно заметить, что использование пустой парольной фразы (как в примере) потенциально *небезопасно*, т. к. в случае кражи или разглашения ключей ими может воспользоваться любая третья сторона.

Затем при помощи команды **ssh-copy-id(1)** открытый ключ регистрируется на удаленном сервере **grex.org** в файле **~/.ssh/authorized_keys**, что происходит с предъявлением пароля ❷. Последующие SSH-соединения ❸ аутентифицируются парой ключей, в результате отпадает необходимость вводить пароль[1].

Листинг 6.17. Доступ по ключу

```
❶ lumpy@ubuntu:~$ ssh-keygen
  Generating public/private rsa key pair.
  Enter file in which to save the key (/home/lumpy/.ssh/id_rsa): ↵
  Enter passphrase (empty for no passphrase): ↵ ⓪
  Enter same passphrase again: ↵ ⓪
  Your identification has been saved in /home/lumpy/.ssh/id_rsa.
  Your public key has been saved in /home/lumpy/.ssh/id_rsa.pub.
  The key fingerprint is:
  b5:27:3a:cc:6c:b1:70:1f:90:cc:a4:dc:1e:c8:20:d4 lumpy@ubuntu
❷ lumpy@ubuntu:~$ ssh-copy-id jake@grex.org
🗝 jake@grex.org's password: P@s$w0rd ↵
  Now try logging into the machine, with "ssh 'jake@grex.org'", and check in:

    ~/.ssh/authorized_keys

  to make sure we haven't added extra keys that you weren't expecting.
❸ lumpy@ubuntu:~$ ssh jake@grex.org ls
```

[1] Появляется необходимость вводить парольную фразу, но в примере она (непозволительно) пустая.

```
! OPENME.txz
  README.gz
❸ lumpy@ubuntu:~$ ssh jake@grex.org zcat README.gz
  А в файле OPENME.txz находится кое-что полезное ;)
❸ lumpy@ubuntu:~$ ssh jake@grex.org file OPENME.txz
  OPENME.txz: data
```

Использовать зашифрованные парольными фразами ключи и одновременно не вводить парольную фразу при каждом подключении позволяет SSH-агент **ssh—agent(1)**, который удерживает в оперативной памяти расшифрованные единожды закрытые ключи пользователя и генерирует с их помощью цифровые подписи по запросу. Листинг 6.18 иллюстрирует аутентификацию пользователя по ключу, защищенному парольной фразой ❶, а затем передачу этого расшифрованного ключа агенту при помощи команды **ssh-add(1)**, что позволяет избавиться от необходимости ввода парольной фразы ключа все время, пока запущен процесс **ssh-agent** (обычно — до окончания сеанса). Наличие SSH-агента, запущенного в сеансе пользователя, обнаруживают две переменные окружения — **SSH_AGENT_PID** и **SSH_AUTH_SOCK**, содержащие соответственно PID агента и имя локального сокета для связи с ним.

Листинг 6.18. SSH-агент

```
🔒 lumpy@ubuntu:~$ ssh jake@grex.org file /usr/bin/ssh
🗝 Enter passphrase for key '/home/lumpy/.ssh/id_rsa': ... ... ... ... ... ↵ ❶
  /usr/bin/ssh: ELF 32-bit LSB executable, Intel 80386, version 1, for OpenBSD, dynamically
  linked (uses shared libs), stripped
  lumpy@ubuntu:~$ ssh-add ❷
🗝 Enter passphrase for /home/lumpy/.ssh/id_rsa: ... ... ... ... ... ↵
  Identity added: /home/lumpy/.ssh/id_rsa (/home/lumpy/.ssh/id_rsa)
🔓 lumpy@ubuntu:~$ ssh jake@grex.org ldd /usr/bin/ssh
  /usr/bin/ssh:
       Start    End      Type Open Ref GrpRef Name
       1c000000 3c017000 exe  1    0   0      /usr/bin/ssh
       0dbc0000 2dbc4000 rlib 0    1   0      /usr/lib/libgssapi.so.5.0
       01cc6000 21cd3000 rlib 0    1   0      /usr/lib/libkrb5.so.18.0
       0098a000 209c7000 rlib 0    1   0      /usr/lib/libcrypto.so.19.0
       0a148000 2a14f000 rlib 0    1   0      /usr/lib/libz.so.4.1
       0b61a000 2b648000 rlib 0    1   0      /usr/lib/libc.so.60.1
       0890e000 0890e000 rtld 0    1   0      /usr/libexec/ld.so
  lumpy@ubuntu:~$ env | grep SSH
  SSH_AGENT_PID=21655
```

```
SSH_AUTH_SOCK=/tmp/ssh-Eehhsbx21654/agent.21654
lumpy@ubuntu:~$ ls -l /tmp/ssh-Eehhsbx21654/agent.21654
srw------- 1 lumpy lumpy 0 марта 28 23:07 /tmp/ssh-Eehhsbx21654/agent.21654
```

Протокол SSH получил широкое распространение далеко за рамками своего изначального предназначения. Кроме непосредственного удаленного доступа, он используется другими командами для своих нужд. Например (см. листинг 6.19), команды **scp(1)** и **sftp(1)** используют **ssh(1)** для безопасного сетевого копирования файлов ❶❷, а команда **rsync(1)** — для сетевой синхронизации (копирование изменившихся) файлов ❸.

Все эти (и другие, подобные им) команды используют **ssh(1)** для запуска некоторой «серверной» программы на удаленном узле (в частности **scp** и **rsync** запускают сами себя, а **sftp** запускает **sftp-server(8)**), с которой и взаимодействуют через защищенное соединение.

Листинг 6.19. Копирование файлов поверх SSH

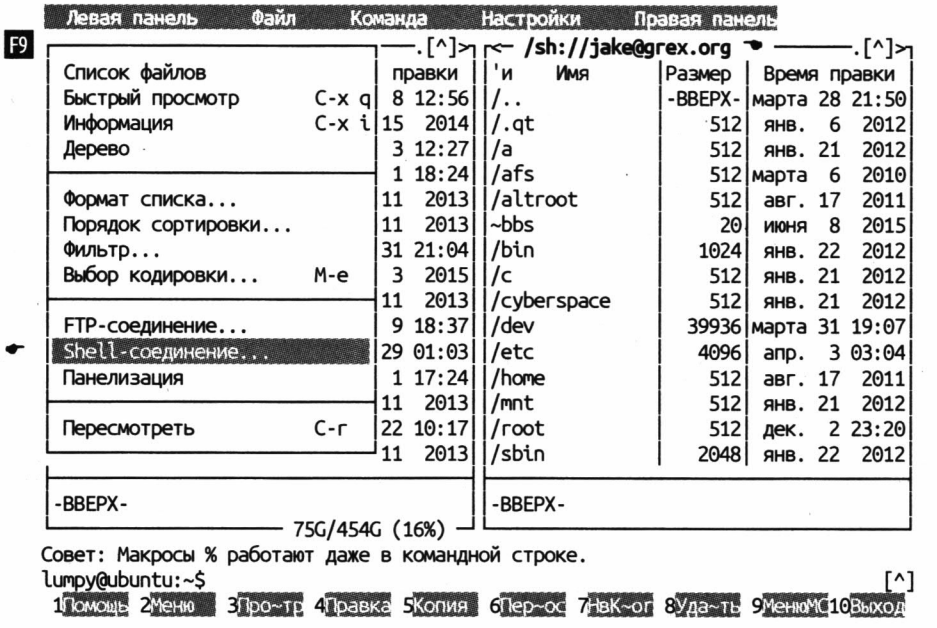
```
❶ lumpy@ubuntu:~$ scp *.pdf jake@grex.org:
  tcpdump.pdf                                   100%   37KB  37.3KB/s   00:00
  Wireshark_Display_Filters.pdf                 100%   38KB  38.0KB/s   00:00
❷ lumpy@ubuntu:~$ sftp jake@grex.org
  Connected to grex.org.
  sftp> ls
  OPENME.txz                          README.gz
  Wireshark_Display_Filters.pdf       linuxperftools.png
  tcpdump.pdf
  sftp> exit
❸ lumpy@ubuntu:~$ rsync -avrz jake@grex.org:/usr/share/man/man1 .
  receiving incremental file list
  man1/
  man1/Mail.1
            ...          ...          ...          ...          ...
  man1/sparc64/mksuncd.1
  man1/vax/
  man1/zaurus/

  sent 10646 bytes  received 4796404 bytes  27081.97 bytes/sec
  total size is 14861868  speedup is 3.09
```

Как оказывается на практике, использование «файловой» надстройки **sftp-server(1)** для безопасного доступа к дереву каталогов удаленных узлов имеет достаточно

широкое распространение. В частности, терминальный файловый менеджер **mc(1)**, **W:[Midnight Commander]**, тоже является SSH:sftp-клиентом, что иллюстрирует листинг 6.20. Более того, при помощи FUSE-файловых систем **sshfs(1)** (см. листинг 3.29) или **gvfs**, файлы SSH:sftp-серверов могут быть смонтированы в дерево каталогов так, что вообще любые программы смогут ими воспользоваться.

Листинг 6.20. Файловый менеджер mc(1) — клиент SSH (sftp)



6.4.2. Почтовые службы SMTP, POP/IMAP

Электронная почта, пожалуй, является самым ранним приложением сетевой подсистемы операционных систем семейства UNIX. Изначально электронные письма пересылались непосредственно между конечными сетевыми узлами при помощи службы **W:[sendmail]** с использованием протокола **W:[SMTP]**, а для отправки или чтения писем применялась утилита **mail(1)**.

Вместо **sendmail** может быть использована абсолютно любая реализация агента пересылки почты (**W:[mail transport agent]**, MTA), например **W:[postfix]** или **W:[exim]**, но обычно его функцию делегируют почтовым серверам провайдера услуг Интернета или серверам публичных сервисов типа **yandex.ru** или **gmail.com**.

Листинг 6.21 иллюстрирует «классическую» схему электронной почты с «локальным» MTA, использующую команду **mail(1)** для составления исходящих ❶ и чтения входящих ❸ писем и команду **mailq(1)** для просмотра очередей обработки почты ❷.

Листинг 6.21. Обработка почты локальной почтовой системой

```
❶ lumpy@ubuntu:~$ mail -s Тест dketov@gmail.com
   Это тест ;) ↵
   Cc: ↵
❷ lumpy@ubuntu:~$ mailq
   Mail queue is empty
❸ lumpy@ubuntu:~$ mail
   Mail version 8.1.2 01/15/2001.  Type ? for help.
   "/var/mail/lumpy" ☚: 1 message 1 new
   >N  1 MAILER-DAEMON@ubu  Thu Jan  7 15:09   77/2653  Undelivered Mail Returned t
☚ & 1 ↵
   Message 1:
   From MAILER-DAEMON  Thu Jan  7 15:09:35 2016
   X-Original-To: lumpy@ubuntu
   Date: Thu,  7 Jan 2016 15:09:35 +0300 (MSK)
   From: MAILER-DAEMON@ubuntu (Mail Delivery System)
   Subject: Undelivered Mail Returned to Sender
   To: lumpy@ubuntu
             ...          ...          ...          ...          ...
   This is the mail system at host ubuntu.☚

   I'm sorry to have to inform you that your message could not
   be delivered to one or more recipients. It's attached below.
             ...          ...          ...          ...          ...
★  <dketov@gmail.com>: host gmail-smtp-in.l.google.com[74.125.205.27] said:
|      550-5.7.1 [95.55.94.237] The IP you're using to send mail is not
|      Authorized to 550-5.7.1 send email directly to our servers. Please use the
|   ☚ SMTP relay at your 550-5.7.1 service provider ☚ instead. Learn more at 550
|      5.7.1 https://support.google.com/mail/answer/10336 tw4si41552991lbb.77 -
└      gsmtp (in reply to end of DATA command)
             ...          ...          ...          ...          ...
   & q ↵
   Saved 1 message in /home/lumpy/mbox ☚
❶ lumpy@ubuntu:~$ mail handy@happytreefriends.ru
   Subject: См. hands(4) ... ↵
   ... про /dev/hands ;) ↵
   Cc: ↵
❷ lumpy@ubuntu:~$ mailq
   -Queue ID- --Size-- ----Arrival Time---- -Sender/Recipient-------
```

```
AF0491900315      309 Thu Jan  7 15:50:08  lumpy@ubuntu
        (connect to happytreefriends.ru[94.76.205.132]:25: Connection refused)
                                          handy@happytreefriends.ru

-- 0 Kbytes in 1 Request.
```

Современные требования к условиям корректной пересылки почтовых сообщений (например, ★ в листинге 6.21) зачастую оказываются чересчур строгими, а содержание собственной локальной почтовой системы — неоправданно сложным. В большинстве случаев конечные пользователи пользуются услугами «внешних» почтовых серверов, организующих полный цикл обработки почты — от приема исходящих сообщений до обслуживания почтовых ящиков. Исходящие сообщения отправляются таким серверам с помощью протокола **W**:[SMTP], а доступ к почтовым ящикам — с помощью протоколов **W**:[POP3] или **W**:[IMAP].

В примере из листинга 6.22 показан терминальный клиент современных почтовых систем **mutt(1)**, поддерживающий «защищенные» протоколы электронной почты SMTPs, IMAPs и POPs, использующие протокол **W**:[SSL][1] для криптозащиты сетевых соединений.

Для отправки ❶ сообщений используются учетная запись пользователя **lumpy.moose** и сервер «исходящих» сообщений **smtp.yandex.ru**, принимающий почту по протоколу SMTPs ①, а для чтения ❷ писем из почтового ящика пользователя **lumpy.moose** могут быть использованы протоколы IMAPs ② или POPs ③ и серверы «входящих» сообщений **imap.yandex.ru** и **pop.yandex.ru**, соответственно.

Листинг 6.22. Обработка почты публичной почтовой службой

```
❶ lumpy@ubuntu:~$ mutt -s Тест dketov@gmail.com
                  ...      ...          ...          ...          ...
  lumpy@ubuntu:~$ cat ~/.muttrc
  set from=lumpy.moose@yandex.ru
  set smtp_url=smtps://lumpy.moose@smtp.yandex.ru
                  ①
❷ lumpy@ubuntu:~$ mutt -f imaps://lumpy.moose@imap.yandex.ru
                          ②
  Определяется адрес сервера imap.yandex.ru...
  Устанавливается соединение с imap.yandex.ru...
  SSL/TLS-соединение с использованием TLS1.2 (RSA/AES-128-CBC/SHA1)
```

[1] Использующий весьма похожие на SSH способы обеспечения конфиденциальности данных и аутентичности взаимодействующих сторон.

```
Пароль для lumpy.moose@imap.yandex.ru: ... ... ... ... ... ↵
q:Выход  d:Удалить  u:Восстановить  s:Сохранить  m:Создать  r:Ответить  g:Всем
   1     Jan 07 Яндекс           (9,9K) Соберите всю почту в этот ящик
   2     Jan 07 Команда Яндекс. ( 15K) Как читать почту с мобильного
   3 N + Mar 30 Яндекс.Паспорт  (8,4K) Доступ к аккаунту восстановлен

---Mutt: imaps://lumpy.moose@imap.yandex.ru/INBOX [Msgs:3 New:1 33K]---(threads/date)-(all)---

❷ lumpy@ubuntu:~$ mutt -f pops://lumpy.moose@pop.yandex.ru
                  ③
          ...      ...      ...      ...      ...
```

6.4.3. Служба WWW

Служба W:[WWW] знакома каждому современному пользователю и в комментариях особо не нуждается. Одной ее заметной особенностью в Linux, пожалуй, является существование терминальных Web-браузеров **links(1)**, **lynx(1)**, **elinks(1)** и **w3m(1)**, позволяющих работать с «текстовой» частью гипертекстовых Web-ресурсов, что проиллюстрировано с помощью **lynx(1)** в примере из листинга 6.23.

Листинг 6.23. Терминальные браузеры lynx, links и w3m

```
lumpy@ubuntu:~$ lynx http://www.kernel.org
#                                                          The Linux Kernel Archives (p1 of 3)
  #The Linux Kernel Archives Atom Feed Latest Linux Kernel Releases

                                   The Linux Kernel Archives

    * About
    * Contact us
    * FAQ
    * Releases
    * Signatures
    * Site news

  Protocol          Location
  HTTP      https://www.kernel.org/pub/
  GIT       https://git.kernel.org/
  RSYNC     rsync://rsync.kernel.org/pub/

  Latest Stable Kernel:
  Download 4.5

  mainline:   4.6-rc1      2016-03-26 [tar.xz] [pgp] [patch]              [view diff] [browse]
  mainline:   4.5          2016-03-14 [tar.xz] [pgp] [patch]              [view diff] [browse]
  stable:     4.4.6        2016-03-16 [tar.xz] [pgp] [patch] [inc. patch] [view diff] [browse] [changelog]
  longterm:   4.1.20       2016-03-17 [tar.xz] [pgp] [patch] [inc. patch] [view diff] [browse] [changelog]
  longterm:   3.18.29      2016-03-17 [tar.xz] [pgp] [patch] [inc. patch] [view diff] [browse] [changelog]
  longterm:   3.14.65      2016-03-16 [tar.xz] [pgp] [patch] [inc. patch] [view diff] [browse] [changelog]
-- Нажмите пробел для перехода на следующую страницу --
  Стрелки: Вверх, Вниз - перемещение. Вправо - переход по ссылке; Влево - возврат.
```

Кроме Web-браузеров, предназначенных для интерактивной работы пользователей, в сценариях на языке командного интерпретатора зачастую используются неинтерактивные пользовательские агенты **wget(1)** и **curl(1)**, позволяющие автоматизировать Web-взаимодействие. Так, например, в листинге 6.24 при помощи **wget(1)** показано скачивание файла в режиме «с докачкой» (**-c**, continue), а **curl(1)** применяется для обращения к Google Geocoding API.

Листинг 6.24. Пользовательские агенты wget и curl

```
lumpy@ubuntu:~$ wget -c http://www.brendangregg.com/Perf/linuxperftools.png
--2016-03-31 12:01:03--  http://www.brendangregg.com/Perf/linuxperftools.png
Распознаётся www.brendangregg.com (www.brendangregg.com)... 97.74.144.194
Подключение к www.brendangregg.com (www.brendangregg.com)|97.74.144.194|:80... соединение установлено.
HTTP-запрос отправлен. Ожидание ответа... 200 OK
Длина: 523561 (511K) [image/png]
Сохранение в: «linuxperftools.png»

42% [=====================>                                ] 224 157     --.-K/s   за 11s

2016-03-31 12:16:14 (19,5 KB/s) - Ошибка чтения, позиция 224157/523561 (Время ожидания соединения истекло).
Продолжение попыток.

--2016-03-31 12:16:15--  (попытка: 2) http://www.brendangregg.com/Perf/linuxperftools.png
Подключение к www.brendangregg.com (www.brendangregg.com)|97.74.144.194|:80... соединение установлено.
HTTP-запрос отправлен. Ожидание ответа... 206 Partial Content
Длина: 523561 (511K), 299404 (292K) осталось [image/png]
Сохранение в: «linuxperftools.png»

100%[++++++++++++++++++++++++++++++=============================================>] 523 561      213K/s   за 1,4s

2016-03-31 12:07:41 (213 KB/s) - «linuxperftools.png» сохранён [523561/523561]

lumpy@ubuntu:~$ curl -s http://maps.googleapis.com/maps/api/geocode/xml?address=СПб+Обручевых+1 |
> fgrep -A 3 '<location>'
   <location>
    <lat>59.9342802</lat>
    <lng>30.3350986</lng>
   </location>
```

6.4.4. Служба FTP

Протокол **W:[FTP]** является «ископаемым»[1] даже по сравнению с **W:[SMTP]**, однако все еще широко используется для организации доступа к обширным файловым хранилищам. Основной особенностью протокола является отделение потока команд от потоков данных, что позволяет организовать параллельную передачу нескольких файлов одновременно. За счет этой особенности появляется возможность (практически не используемая, как небезопасная) передачи файлов не между файловым сервером и клиентом (как «обычно»), а между двумя (!) файловыми серверами, см. **W:[FXP]**.

В листинге 6.25 иллюстрируются **lftp(1)** — один из самых распространенных терминальных FTP-клиентов, имеющий массу полезных возможностей, как то: «задачи» заднего фона ❷, зеркалирование файловых поддеревьев (включая FXP) ❷, неинтерактивная работа ❸ и т. д.

Листинг 6.25. FTP-клиент lftp

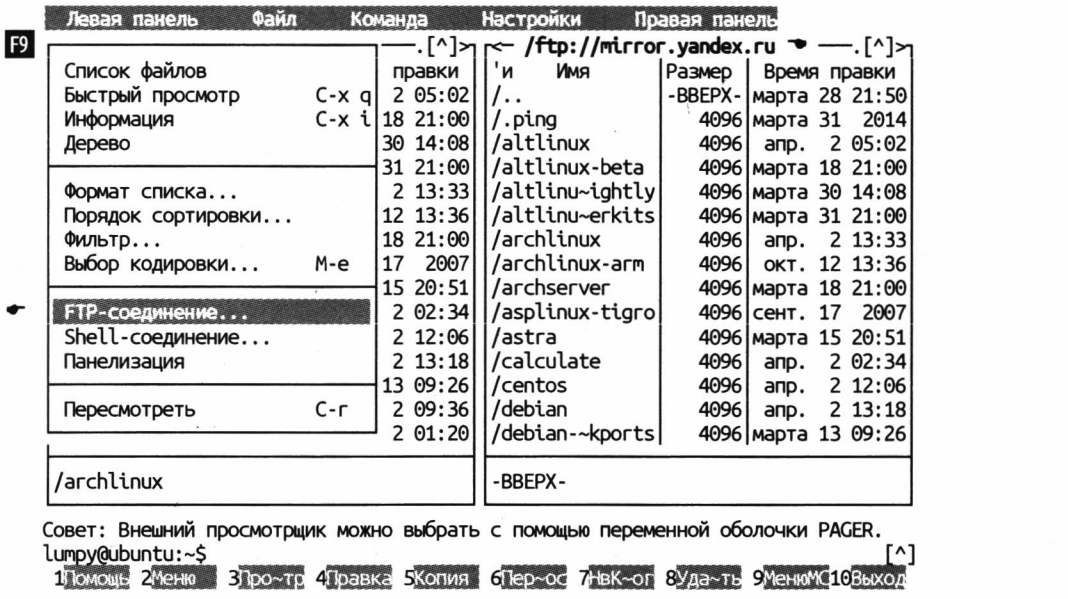
```
lumpy@ubuntu:~$ lftp mirror.yandex.ru
lftp mirror.yandex.ru:~> cd archlinux/iso/latest/ ↵
lftp mirror.yandex.ru:/archlinux/iso/latest> ls *.iso ↵
-rw-r--r--    1 ftp      ftp      743440384 Mar 01 16:04 archlinux-2016.03.01-dual.iso
lftp mirror.yandex.ru:/archlinux/iso/latest> get archlinux-2016.03.01-dual.iso & ↵
                                                                              ❶↑
[0] get archlinux-2016.03.01-dual.iso &
    `archlinux-2016.03.01-dual.iso' в позиции 0
lftp mirror.yandex.ru:/archlinux/iso/latest> cd /mirrors/gnu ↵
lftp mirror.yandex.ru:/mirrors/gnu> mirror hurd &
                                    ↑❷
[1] mirror hurd &
    cd `hurd' [Ожидание ответа...]
lftp mirror.yandex.ru:/mirrors/gnu> jobs ↵
[0] get archlinux-2016.03.01-dual.iso
    `archlinux-2016.03.01-dual.iso' в позиции 279913700 (37%) 1.59Мб/с ост:4m [Получение данных]
[1] mirror hurd
 \mirror old
  \transfer hurd-0.1.tar.gz
```

[1] Первая публикация спецификации протокола (RFC141) датируется 1971 годом, см. https://tools.ietf.org/html/rfc114.

```
        `hurd-0.1.tar.gz' в позиции 88264 (7%) [Получение данных]
    lftp mirror.yandex.ru:/ubuntu> quit ↵
❸ [25836] Moving to background to complete transfers...
```

Кроме массы специализированных FTP-клиентов **ftp(1)**, **lftp(1)**, **ncftp(1)**, **gftp(1)**, протокол FTP поддерживается и другими программными средствами, скажем, различными файловыми менеджерами, что иллюстрирует листинг 6.26 на примере **mc(1)**.

Листинг 6.26. Файловый менеджер mc(1) — клиент FTP



Кроме всего прочего, клиентами FTP являются еще и внеядерные FUSE-файловые системы **curlftpfs** (см. листинг 3.29) или **gvfs**, позволяющие монтировать файлы FTP-серверов в дерево каталогов для их использования вообще любыми программами.

6.4.5. Служба NFS

Служба **W:[Network File System]** изначально разрабатывалась для *прозрачного* сетевого использования файловых систем сервера так, как будто они были непосредственно примонтированы в дерево каталогов клиента. В отличие от **F**ile **T**ransfer-протокола, предназначенного для скачивания (transfer) файлов, протокол NFS является ретранслятором системных вызовов **open(2)**, **close(2)**, **read(2)**, **write(2)**, **seek(2)** и пр. с клиента на сервер. Это позволяет клиенту выполнять операции чтения/записи с любой частью файла, без его передачи целиком.

NFS-клиент

Клиент протокола NFS непосредственно реализован в ядре Linux при помощи модулей **nfs**, **nfsv2**/**nfsv3**/**nfsv4** и используется посредством штатной операции монтирования **mount(8)**, тем самым делая доступными серверные файлы любым клиентским программам.

В примере из листинга 6.27 показана процедура монтирования файловой системы **/Qmultimedia** NFS-сервера **NVR.local** ❷ в каталог **/mnt/network/nvr/Qmm** при помощи протокола NFS v3 (**-t nfs -o vers=3**). Для получения списка экспортируемых (**-e**, export) сервером файловых систем ❶ используется команда **showmount(8)**, которая также является специализированным NFS-клиентом.

Листинг 6.27. Монтирование NFS

```
❶ lumpy@ubuntu:~$ showmount -e NVR.local
  Export list for NVR.local:
  /Qweb *
  /Qusb *
  /Qrecordings *
  /Qmultimedia *
  /Qdownload *
  /Public *
  /Network Recycle Bin 1 *
  lumpy@ubuntu:~$ sudo mkdir -p /mnt/network/nvr/Qmm
❷ lumpy@ubuntu:~$ sudo mount -t nfs -o vers=3 NVR.local:/Qmultimedia /mnt/network/nvr/Qmm
  lumpy@ubuntu:~$ findmnt -t nfs
  TARGET                    SOURCE                 STYPE  OPTIONS
  /mnt/network/nvr/Qmm      NVR.local:/Qmultimedia nfs    rw,relatime,vers=3,...
  lumpy@ubuntu:~$ ls /mnt/network/nvr/Qmm
❸ -rw-r--r-- 1 toothy ☜ users ☜  678696 авг.  20  2014 IMG_20140719_125651.jpg
  -rw-r--r-- 1 toothy    users     649685 авг.  20  2014 IMG_20140719_125713.jpg
  ...          ...              ...              ...              ...
  -rw-r--r-- 1 toothy    users     814607 авг.  20  2014 IMG_20140820_111355.jpg
  lumpy@ubuntu:~$ id toothy
  uid=1010(toothy) gid=1012(toothy) группы=1012(toothy)
```

Необходимо отметить, что права доступа и идентификаторы UID/GID владельцев файлов ❸ передаются NFS-протоколом в неизменном виде, поэтому базы пользовательских учетных записей всех клиентов (и сервера) должны быть согласованы, например, при помощи их централизованного хранения в каталоге LDAP.

NFS-сервер

Функционирование NFS-сервера имеет свою специфику, связанную с использованием NFS-протоколом принципа RPC (**r**emote **p**rocedure **c**all, удаленного вызова процедур) в реализации **W:[SUN RPC]**. Серверы, использующие **SUN RPC**, не имеют «зафиксированного»[1] номера порта TCP/UDP, а используют произвольный, случайно выбираемый порт. Вместо этого, в **SUN RPC** закрепляются номера «программ» (program), предоставляющих определенные «услуги» (service), а соответствующий порт регистрируется в служебной RPC-программе **portmapper**[2], что проиллюстрировано в листинге 6.28 при помощи утилиты **rpcinfo(8)**.

Листинг 6.28. Удаленный вызов процедур RPC: динамические номера портов и portmapper

```
lumpy@ubuntu:~$ rpcinfo -p NVR.local
   program vers proto   port  service
    100000    3   tcp    111  portmapper
    100000    3   udp    111  portmapper
       ...     ...    ...          ...          ...          ...
    100003    3   tcp   2049  nfs
    100003    3   udp   2049  nfs
       ...     ...    ...          ...          ...          ...
    100005    3   udp ☛ 53964  mountd
    100005    3   tcp ☛ 39835  mountd
       ...     ...    ...          ...          ...          ...
```

При помощи **portmapper** организуется обнаружение NFS-серверов в локальной сети посредством широковещательного (**-b**, **b**roadcast) поиска ❶ зарегистрированных RPC-программ NFS v3 (листинг 6.29). Кроме этого, некоторые серверы NFS (например, сетевые устройства хранения данных **W:[NAS]** или сетевые видеорегистраторы **W:[NVR]**) регистрируются в службе mDNS/DNS-SD, что обнаруживается ❷ при помощи **avahi-browse(1)**.

Листинг 6.29. Обнаружение NFS-серверов

```
❶ lumpy@ubuntu:~$ rpcinfo -b nfs 3
   192.168.17.90.8.1       NVR.local
   ...          ...             ...          ...          ...
```

[1] Как, например, порт 22 закреплен за службой SSH, порт 25 — за SMTP, а порт 80 — за HTTP-протоколом службы WWW.

[2] Порт самой RPC-программы **portmapper** все же закреплен за номером 111 TCP/UDP, что позволяет клиентам обращаться **portmapper** сервера и находить порты других RPC-программ по их номерам.

```
❷ lumpy@ubuntu:~$ avahi-browse -rcl _nfs._tcp
...          ...          ...          ...          ...
+   eth0 IPv4 NVR(NFS)                 ☛ Network File System          local
...          ...          ...          ...          ...
=   eth0 IPv4 NVR(NFS)                   Network File System          local
   hostname = [NVR.local]
   address = [192.168.17.90]
   port = [2049]
   txt = []
```

Сервер NFS предоставляет клиентам некоторое количество RPC-услуг (**-s**, services), показанных в листинге 6.30, при помощи **rpcinfo(8)**. Выделяют базовые RPC-программы **mountd** ② и **nfs** ①, позволяющие монтировать файловые системы NFS и обращаться к их файлам, и дополнительные RPC-программы[1] **nlockmgr** ③ и **status** ④, реализующие механизм блокировки файлов.

Листинг 6.30. RPC-программы службы NFS

```
lumpy@ubuntu:~$ rpcinfo -s NVR.local
   program version(s) netid(s)                         service     owner
    100000  4,3,2     tcp6,tcp,udp,udp6                 portmapper  superuser
    100003  4,3,2     udp6,tcp6,udp,tcp           ①     nfs         superuser
    100005  3,2,1     tcp6,udp6,tcp,udp           ②     mountd      superuser
    100021  4,3,2,1   tcp6,udp6,tcp,udp           ③     nlockmgr    superuser
    100024  1         tcp6,udp6,tcp,udp           ④     status      superuser
```

6.4.6. Служба SMB/CIFS

Служба **W**:[CIFS] (**c**ommon **i**nternet **f**ile **s**ystem), заимствованная из семейства операционных систем Windows, предназначена (аналогично «родной» NFS) для совместного использования файлов. Основным протоколом службы **CIFS** является протокол SMB (server **m**essage **b**locks), который аналогично NFS ретранслирует системные вызовы к файлам.

Имена NetBIOS

Отличительной особенностью протокола **SMB**, доставшейся ему в наследство от транспорта **W**:[NetBIOS], является возможность использования еще одного вида имен узлов (в дополнение к DNS- и mDNS-именам, *см. разд. 6.3*) — так назы-

[1] См. **W**:[NLM], **n**etwork **l**ock **m**anager и **W**:[NSM], **n**etwork **s**tatus **m**onitor.

ваемых «имен NetBIOS», и собственной службы NBNS[1] (**n**et**b**ios **n**ame **s**ervice), отображающей имена NetBIOS на IP-адреса.

Листинг 6.31. Имена узлов NetBIOS и их IP-адреса

```
lumpy@ubuntu:~$ nmblookup WINXP
querying WINXP on 192.168.100.255
192.168.100.3 WINXP<00>
```

В листинге 6.31 иллюстрируется использование утилиты **nmblookup(1)**, предназначенной для диагностики службы NBNS, при помощи которой «плоское» имя NetBIOS **WINXP** отображается на соответствующий ему IP-адрес. Именно при помощи службы NBNS и широковещательного поиска специальных «групповых» имен NetBIOS реализуется основной способ обнаружения CIFS-серверов локальной сети, что выполняет утилита **smbtree(1)**, иллюстрируемая ❶ в листинге 6.32.

В редких отдельных случаях серверы CIFS обнаруживаются ❷ зарегистрированными в службе mDNS/DNS-SD, что характерно (как и в случае серверов NFS) для сетевых устройств хранения данных **W**:**[NAS]** или сетевых видеорегистраторов **W**:**[NVR]**.

Листинг 6.32. Обнаружение CIFS-серверов

```
❶ lumpy@ubuntu:~$ smbtree -N
  WORKGROUP
      \\WINXP
              \\WINXP\C$              Стандартный общий ресурс
              \\WINXP\ADMIN$          Удаленный Admin
              \\WINXP\media           Фото, видео и т. д.
              \\WINXP\D$              Стандартный общий ресурс
              \\WINXP\IPC$            Удаленный IPC
      \\NVR
              \\NVR\Qrecordings
              \\NVR\Qmultimedia
              \\NVR\Qdownload
              \\NVR\IPC$

❷ lumpy@ubuntu:~$ avahi-browse -rcl _smb._tcp
  ...          ...          ...          ...          ...
```

[1] Служба NBNS похожа на DNS и mDNS одновременно, но несовместима с ними.

```
+   eth0 IPv4 NVR(SMB)                          Microsoft Windows Network      local
...          ...              ...              ...              ...
=   eth0 IPv4 NVR(SMB)                          Microsoft Windows Network      local
    hostname = [NVR.local]
    address = [192.168.17.90]
    port = [445]
    txt = []
```

CIFS-клиенты

Различают две разных реализации клиента CIFS — внеядерную **smbclient(1)** (аналогичную «интерактивным» FTP-клиентам) и ядерную (аналогичную NFS-клиенту), реализующуюся модулем ядра **cifs**. Использование ядерного модуля позволяет монтировать общие файловые ресурсы (share) серверов CIFS непосредственно в дерево каталогов клиента, что дает возможность любым его программам использовать серверные файлы.

В примерах из листинга 6.33 показано использование CIFS-клиента **smbclient(1)** для получения списка (**-L**, list) разделяемых ресурсов ❶ (share) узла **WINXP**, равно как и для подключения ❷ к его публичному (**-N**, no password) разделяемому ресурсу **media** с последующим скачиванием файлов целиком.

Листинг 6.33. Клиент SMB/CIFS

```
❶ lumpy@ubuntu:~$ smbclient -NL //WINXP
  Domain=[WINXP] OS=[Windows 5.1] Server=[Windows 2000 LAN Manager]

        Sharename       Type      Comment
        ---------       ----      -------
        C$              Disk      Стандартный общий ресурс
        D$              Disk      Стандартный общий ресурс
        ADMIN$          Disk      Удаленный Admin
        IPC$            IPC       IPC Service
        media           Disk      Фото, видео и т. д.

❷ lumpy@ubuntu:~$ smbclient -N //WINXP/media
  Domain=[WINXP] OS=[Windows 5.1] Server=[Windows 2000 LAN Manager]
  smb: \> cd DCIM\ ↵
  smb: \DCIM\> dir ↵
    .                                   D        0  Thu Jan  7 20:57:36 2016
    ..                                  D        0  Thu Jan  7 20:57:36 2016
    Dd595.jpg                           A  4494092  Sun Feb 13 16:24:04 2011
```

```
  Dd596.jpg                          A  3842680  Sun Feb 13 16:14:56 2011
        ...          ...                 ...                 ...          ...
  Dd680.jpg                          A  4087313  Sun Feb 13 03:10:45 2011
  Dd681.jpg                          A  4108278  Sun Feb 13 15:58:38 2011

                61192 blocks of size 1048576. 10915 blocks available
smb: \DCIM\> get Dd680.jpg ↵
getting file \DCIM\Dd680.jpg of size 4087313 as Dd680.jpg (72572,9 KiloBytes/sec)
(average 72573,0 KiloBytes/sec)
smb: \DCIM\> quit ↵
```

Листинг 6.34 иллюстрирует использование ядерного модуля **cifs** для монтирования публичного (**-o guest**) ресурса **media** с узла **WINXP** в каталог **/mnt/network/winxp/media**.

Клиент **smbclient(1)** имеет встроенный механизм NBNS, поэтому без проблем подключается к узлу **WINXP**, а при монтировании **cifs** механизм NBNS недоступен ❶, что требует подсказки ❷ в виде IP-адреса сервера.

Листинг 6.34. Монтирование ресурса SMB/CIFS

```
lumpy@ubuntu:~$ sudo mkdir -p /mnt/network/winxp/media
lumpy@ubuntu:~$ sudo mount -t cifs -o guest //WINXP/media /mnt/network/winxp/media
❶ mount error: could not resolve address for WINXP: Unknown error
lumpy@ubuntu:~$ nmblookup WINXP
querying WINXP on 192.168.100.255
192.168.100.3 WINXP<00>
❷ lumpy@ubuntu:~$ sudo mount -t cifs -o guest,ip=192.168.100.3 //WINXP/media /mnt/network/winxp/media
lumpy@ubuntu:~$ findmnt -t cifs
TARGET                    SOURCE         FSTYPE OPTIONS
/mnt/network/winxp/media  //WINXP/media cifs   rw,relatime,vers=1.0,cache=strict,...
lumpy@ubuntu:~$ ls -l /mnt/network/winxp/media/DCIM/
итого 346228
-rwxr-xr-x 1 root ➥ root ➥ 4494092 февр. 13  2011 Dd595.jpg
          ...          ...              ...              ...              ...
-rwxr-xr-x 1 root     root    4108278 февр. 13  2011 Dd681.jpg
```

6.5. Средства сетевой диагностики

Диагностика сетевого обмена существенно облегчает решение разнообразных задач, связанных с эксплуатацией или разработкой сетевых приложений. К сетевым диагностическим специальным средствам относят анализаторы пакетов и сетевые сканеры, которые применяются самостоятельно или вместе с трассировщиками системных и библиотечных вызовов.

6.5.1. Анализаторы пакетов tcpdump и tshark

Анализаторы пакетов предназначены для перехвата данных, поступающих из сети на сетевые интерфейсы или отправляющиеся в сеть с сетевых интерфейсов. Современные анализаторы пакетов кроме, собственно, захвата пакетов осуществляют их детальный протокольный разбор, а также позволяют отфильтровывать подлежащие анализу пакеты по ряду критериев.

В листинге 6.35 показан пример использования наиболее распространенного, «классического» анализатора пакетов **tcpdump(1)**, анализирующего пакеты на интерфейсе (**-i**, interface) **wlan0**, адресованные порту **port 53**.

Листинг 6.35. Анализатор пакетов tcpdump

```
                                                                    pts/1
lumpy@ubuntu:~$ host 2x2tv.ru
2x2tv.ru ① has address 178.63.2.134 ②
2x2tv.ru mail is handled by 20 relay2.tnt-tv.ru.
2x2tv.ru mail is handled by 30 relay3.tnt-tv.ru.
2x2tv.ru mail is handled by 10 mx.tnt-tv.ru.
```

```
                                                                    pts/2
lumpy@ubuntu:~$ sudo tcpdump -i wlan0 port 53
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on wlan0, link-type EN10MB (Ethernet), capture size 65535 bytes
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on wlan0, link-type EN10MB (Ethernet), capture size 65535 bytes

15:11:32.139394 IP ubuntu.local.40694 > 192.168.100.1.domain: 27739+ A? 2x2tv.ru. ① (26)
15:11:32.144674 IP 192.168.100.1.domain > ubuntu.local.40694: 27739 1/0/0 A 178.63.2.134 ② (42)
15:11:32.145260 IP ubuntu.local.22022 > 192.168.100.1.domain: 37673+ AAAA? 2x2tv.ru. (26)
15:11:32.147959 IP 192.168.100.1.domain > ubuntu.local.48132: 41476 NXDomain 0/0/0 (44)
15:11:32.149187 IP ubuntu.local.58056 > 192.168.100.1.domain: 62332+ MX? 2x2tv.ru. (26)
15:11:32.154191 IP 192.168.100.1.domain > ubuntu.local.58056: 62332 3/0/2 MX mx.tnt-tv.ru. 10, ... (130)
^C
6 packets captured
6 packets received by filter
0 packets dropped by kernel
```

Анализу подвергается работа DNS-клиента **host(1)**, отображающего имя домена **2x2tv.ru** на IP-адрес и имена его почтовых адресов (MX-записи DNS). В результате анализа захваченных пакетов наблюдаются запросы и ответы DNS-протокола к локальному кэширующему серверу **192.168.100.1**, который на вопрос ① **A?** адреса

IPv4, соответствующего имени **2x2tv.ru**, отвечает ② адресом **A 178.63.2.134**, а на вопрос **AAAA?** адреса IPv6 отвечает **NXDomain** (отсутствует такая DNS-запись).

Терминальный[1] анализатор пакетов **tshark(1)**, позволяющий проводить детальный анализ прикладных протоколов, таких как SSH, HTTP, FTP, NFS и пр., проиллюстрирован в листинге 6.36. Здесь анализируется работа пользовательского агента **curl(1)**, запрашивающего Web-ресурс по адресу **http://ipinfo.io/city**. В результате захвата пакетов на интерфейсе (**-i**, interface) **wlan0**, адресованных порту **port 80**, просматриваются (**-R**, read filter) пакеты, содержащие **http**-запросы, при этом детальному анализу (**-V**, view) подвергается только (**-O**, only) их **http**-содержимое.

В результате анализа, например, можно сделать вывод ★ о программном обеспечении Web-сервера, обслуживающего сайт **http://ipinfi.io**.

Листинг 6.36. Анализатор пакетов wireshark

pts/1

```
lumpy@ubuntu:~$ curl http://ipinfo.io/city
Saint Petersburg
```

pts/2

```
lumpy@ubuntu:~$ sudo tshark -i wlan0 port 80 -V -O http,data-text-lines -R http
tshark: Lua: Error during loading:
 [string "/usr/share/wireshark/init.lua"]:45: dofile has been disabled
Running as user "root" and group "root". This could be dangerous.
    Capturing on wlan0
    Frame 38: 229 bytes on wire (1832 bits), 229 bytes captured (1832 bits)
    Ethernet II, Src: IntelCor_a2:28:4e (b8:03:05:a2:28:4e), Dst: 9c:37:f4:76:c5:58 (9c:37:f4:76:c5:58)
    Internet Protocol Version 4, Src: 192.168.100.5 (192.168.100.5), Dst: 52.28.249.93 (52.28.249.93)
    Transmission Control Protocol, Src Port: 43564 (43564), Dst Port: http (80), Seq: 1, Ack: 1, Len: 163
    Hypertext Transfer Protocol ☚
        GET /city HTTP/1.1\r\n
            [Expert Info (Chat/Sequence): GET /city HTTP/1.1\r\n]
                [Message: GET /city HTTP/1.1\r\n]
                [Severity level: Chat]
                [Group: Sequence]
            Request Method: GET
            Request URI: /city
            Request Version: HTTP/1.1
        User-Agent: curl/7.22.0 (i686-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23 ...
        Host: ipinfo.io\r\n
```

[1] Гораздо удобнее, конечно, использовать его графический вариант — **wireshark(1)**.

```
    Accept: */*\r\n
    \r\n
    [Full request URI: http://ipinfo.io/city]

Frame 54: 313 bytes on wire (2504 bits), 313 bytes captured (2504 bits)
Ethernet II, Src: 9c:37:f4:76:c5:58 (9c:37:f4:76:c5:58), Dst: IntelCor_a2:28:4e (b8:03:05:a2:28:4e)
Internet Protocol Version 4, Src: 52.28.249.93 (52.28.249.93), Dst: 192.168.100.5 (192.168.100.5)
Transmission Control Protocol, Src Port: http (80), Dst Port: 43564 (43564), Seq: 1, Ack: 164, Len: 247
Hypertext Transfer Protocol ➧
    HTTP/1.1 200 OK\r\n
        [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
            [Message: HTTP/1.1 200 OK\r\n]
            [Severity level: Chat]
            [Group: Sequence]
        Request Version: HTTP/1.1
        Status Code: 200
        Response Phrase: OK
    Access-Control-Allow-Origin: *\r\n
    Content-Type: text/html; charset=utf-8\r\n
    Date: Sun, 13 Mar 2016 12:37:31 GMT\r\n
    Server: nginx/1.6.2\r\n ★
    Set-Cookie: first_referrer=; Path=/\r\n
    Content-Length: 17\r\n
        [Content length: 17]
    Connection: keep-alive\r\n
    \r\n
Line-based text data: text/html
Saint Petersburg\n ➧

^C2 packets captured
```

6.5.2. Сетевой сканер nmap

Сетевой сканер **W:[nmap]** предназначен для поиска служб по их открытым портам на указанных узлах сети. В примере из листинга 6.37 показан процесс и результаты сканирования узла **192.168.100.1** (беспроводной маршрутизатор, арендованный у провайдера Интернета).

Сканирование выполнялось способом TCP connect scan ❶, т. е. предпринималась попытка установить соединение TCP с каждым из 1000 «популярных» портов. В результате оказывается, что на узле открыты 7 портов, 3 из которых недоступ-

ны ❷ (вероятно, отфильтрованы по IP-адресу), а оставшиеся 4 (в частности, порты 53/DNS и 80/WWW) доступны ❸ для присоединения.

Листинг 6.37. Сетевой сканер nmap

```
lumpy@ubuntu:~$ nmap -n -vvv --reason 192.168.100.1

Starting Nmap 5.21 ( http://nmap.org ) at 2016-04-03 17:48 MSK
Initiating Ping Scan at 17:48
Scanning 192.168.100.1 [2 ports]
Completed Ping Scan at 17:48, 0.00s elapsed (1 total hosts)
❶ Initiating Connect Scan at 17:48
Scanning 192.168.100.1 [1000 ports]
Discovered open port 80/tcp on 192.168.100.1
Discovered open port 53/tcp on 192.168.100.1
Discovered open port 49152/tcp on 192.168.100.1
Discovered open port 49153/tcp on 192.168.100.1
Completed Connect Scan at 17:48, 1.58s elapsed (1000 total ports)
Nmap scan report for 192.168.100.1
Host is up, received conn-refused (0.055s latency).
Scanned at 2016-04-03 17:48:53 MSK for 2s
Not shown: 993 closed ports
Reason: 993 conn-refused
PORT      STATE    SERVICE REASON
❷ 21/tcp    filtered ftp     no-response
22/tcp    filtered ssh     no-response
23/tcp    filtered telnet  no-response
❸ 53/tcp    open     domain  syn-ack
80/tcp    open     http    syn-ack
49152/tcp open     unknown syn-ack
49153/tcp open     unknown syn-ack

Read data files from: /usr/share/nmap
Nmap done: 1 IP address (1 host up) scanned in 1.63 seconds
```

6.5.3. Мониторинг сетевых соединений процессов

Интерфейс сокетов в целом является продолжением идеи «файлов», специально предназначенных для межпроцессного взаимодействия, некоторым развитием «файловой» природы простейших именованных и неименованных каналов.

Таким образом, сокеты сетевых семейств **ip(7)**, **ipv6(7)** и пр. обладают «файловыми» свойствами точно так же, как и локальные сокеты **unix(7)**. Например, каждый сетевой сокет идентифицируется при помощи файлового (!) дескриптора в таблице открытых файлов процесса, что проиллюстрировано в листинге 6.38 при помощи **lsof(1)** и **ss(8)**.

Листинг 6.38. Файловые дескрипторы сетевых сокетов

```
lumpy@ubuntu:~$ pgrep lftp
8715
lumpy@ubuntu:~$ lsof -i :5900
COMMAND  PID  USER   FD   TYPE   DEVICE SIZE/OFF    NODE NAME
lftp    8715 lumpy  cwd   DIR    252,0     4096 20430868 /home/lumpy
lftp    8715 lumpy  rtd   DIR    252,0     4096        2 /
lftp    8715 lumpy  txt   REG    252,0   905324  3811570 /usr/bin/lftp
              ...          ...            ...         ...          ...
lftp    8715 lumpy   0u   CHR    136,1      0t0        4 /dev/pts/1
lftp    8715 lumpy   1u   CHR    136,1      0t0        4 /dev/pts/1
lftp    8715 lumpy   2u   CHR    136,1      0t0        4 /dev/pts/1
lftp    8715 lumpy   3r   DIR    252,0     4096 20430868 /home/lumpy
lftp    8715 lumpy ☛ 4u   IPv4 16686880     0t0      TCP ubuntu.local:35561->mirror.yandex.ru:ftp (ESTAB...
lumpy@ubuntu:~$ sudo ss -p dport = :21
State      Recv-Q Send-Q      Local Address:Port       Peer Address:Port
ESTAB      0      0           192.168.100.5:35561      213.180.204.183:ftp      users:(("lftp",8715,4☚))
```

Более детально проследить за жизненным циклом сетевого сокета позволяет трассировка «сокетных» системных вызовов **socket(2)**, **connect(2)**, **bind(2)**, **listen(2)**, **accept(2)**, **send(2)** и **recv(3)**.

В примере из листинга 6.39 показана трассировка «клиентского» сокета пользовательского агента **curl(1)**, загружающего Web-ресурс **http://www.gnu.org/graphics/agnuheadterm-xterm.txt**.

Системный вызов **socket(2)** ❶ создает потоковый сокет семейства **ip(7)**, которому назначается первый свободный файловый дескриптор **FD=3**, после чего системный вызов **connect(2)** инициирует установку соединения ❷ этого сокета с портом **80** узла **208.118.235.148**. После установки соединения системный вызов **send(2)** отсылает Web-серверу команду **W:[HTTP]** протокола **GET** на получение запрашиваемого ресурса, а несколько системных вызовов **recv(2)** получают запрошенный ресурс.

Листинг 6.39. Сетевой клиент: системные вызовы socket(2), connect(2), send(2) и recv(2)

```
lurpy@ubuntu:~$ strace -fe trace=network curl http://www.gnu.org/graphics/agnuheadterm-xterm.txt
...  ...  ...  ...  ...
❶ socket(PF_INET, SOCK_STREAM, IPPROTO_TCP) = 3
setsockopt(3, SOL_SOCKET, SO_KEEPALIVE, [1], 4) = 0
❷ connect(3, {sa_family=AF_INET, sin_port=htons(80), sin_addr=inet_addr("208.118.235.148")},
16) = -1 EINPROGRESS (Operation now in progress)
getsockopt(3, SOL_SOCKET, SO_ERROR, [0], [4]) = 0
getpeername(3, {sa_family=AF_INET, sin_port=htons(80), sin_addr=inet_addr("208.118.235.148")}, [16]) = 0
getsockname(3, {sa_family=AF_INET, sin_port=htons(51090), sin_addr=inet_addr("192.168.100.4")}, [16]) = 0
❸ send(3, "GET /graphics/agnuheadterm-xterm"..., 192, MSG_NOSIGNAL) = 192
❹ recv(3, "HTTP/1.1 200 OK\r\nDate: Wed, 06 J"..., 16384, 0) = 1400
recv(3, "m\342\226\204\33[48;5;242m\33[38;5;244m\342\226\204\33[4"..., 14770, 0) = 1400
recv(3, ";5;59m\342\226\200\33[38;5;240m\342\226\204\33[48;5;24"..., 13370, 0) = 7000
recv(3, "  \33[48;5;251m \33[48;5;231m      "..., 6370, 0) = 1400
recv(3, "8;5;242m\33[38;5;237m\342\226\204\33[48;5;16m"..., 4970, 0) = 4970
```

Жизненный цикл «серверных» сокетов чуть более сложен и предполагает использование одного «слушающего» сокета для клиентских подключений и по одному «обслуживающему» сокету на каждого подключенного клиента.

В примере из листинга 6.40 показана трассировка простейшего Web-сервера, реализованного модулем **SimpleHTTPServer** языка программирования **W:[python]**. Web-сервер запускается из «рабочего» каталога **/usr/share/doc** и предоставляет Web-доступ ко всем файлам этого каталога, используя «нестандартный» порт **8000**.

Для начала при помощи **socket(2)** создается ❶ потоковый сокет семейства **ip(7)**, которому назначается первый свободный файловый дескриптор **FD = 3**. Этот сокет и будет выступать в роли «слушающего», т. е. принимающего клиентские соединения, поэтому ему «привязывается» ❷ адрес **0.0.0.0** и порт **8000** (куда и будут поступать клиентские соединения) при помощи системного вызова **bind(2)**, а сам сокет переводится в слушающее состояние ❸ системным вызовом **listen(2)**. Все входящие клиентские соединения ставятся в очередь «слушающего» сокета и изымаются из нее системным вызовом **accept(2)**, который создает для каждого клиентского соединения собственный сокет (клонируя слушающий). При поступлении клиентского соединения ❹ был создан новый «обслуживающий» сокет с файловым дескритором **FD = 4**, используя который посредством **recv(2)** была получена **W:[HTTP]** команда **GET** на доступ к «корневому» ресурсу сервера, а с помощью нескольких системных вызовов **send(2)** в ответ был направлен сформированный HTML-список файлов в каталоге **/usr/share/doc**.

Листинг 6.40. Сетевой сервер: системные вызовы socket(2), bind(2), listen(2), accept(2), send(2) и recv(2)

pts/1

```
lumpy@ubuntu:~$ cd /usr/share/doc
lumpy@ubuntu:/usr/share/doc$ strace -fe trace=network python -m SimpleHTTPServer
❶ socket(PF_INET, SOCK_STREAM, IPPROTO_IP) = 3
setsockopt(3, SOL_SOCKET, SO_REUSEADDR, [1], 4) = 0
❷ bind(3, {sa_family=AF_INET, sin_port=htons(8000), sin_addr=inet_addr("0.0.0.0")}, 16) = 0
getsockname(3, {sa_family=AF_INET, sin_port=htons(8000), sin_addr=inet_addr("0.0.0.0")}, [16]) = 0
getsockname(3, {sa_family=AF_INET, sin_port=htons(8000), sin_addr=inet_addr("0.0.0.0")}, [16]) = 0
              ...           ...           ...           ...           ...
❷ listen(3, 5)                          = 0
getsockname(3, {sa_family=AF_INET, sin_port=htons(8000), sin_addr=inet_addr("0.0.0.0")}, [16]) = 0
Serving HTTP on 0.0.0.0 port 8000 ...
❹ accept(3, {sa_family=AF_INET, sin_port=htons(55094), sin_addr=inet_addr("127.0.0.1")}, [16]) = 4 ↵
❺ recv(4 ↵, "GET / HTTP/1.1\r\nUser-Agent: Wget"..., 8192, 0) = 111
localhost - - [03/Apr/2016 19:52:14] "GET / HTTP/1.1" 200 -
❻ send(4 ↵, "HTTP/1.0 200 OK\r\n", 17, 0)   = 17
send(4, "Server: SimpleHTTP/0.6 Python/2."..., 37, 0) = 37
send(4, "Date: Sun, 03 Apr 2016 16:52:14 "..., 37, 0) = 37
send(4, "Content-type: text/html; charset"..., 40, 0) = 40
send(4, "Content-Length: 140946\r\n", 24, 0) = 24
send(4, "\r\n", 2, 0)                   = 2
send(4, "<!DOCTYPE html PUBLIC \"-//W3C//D"..., 8192, 0) = 8192
              ...           ...           ...           ...           ...
send(4, "video-qxl/</a>\n<li><a href=\"xser"..., 1682, 0) = 1682
shutdown(4, 1 /* send */)               = 0^C
```

pts/2

```
lumpy@ubuntu:~$ wget http://ubuntu:8000
--2016-04-03 19:52:14--  http://ubuntu:8000/
Распознаётся ubuntu (ubuntu)... 127.0.1.1
Подключение к ubuntu (ubuntu)|127.0.1.1|:8000... соединение установлено.
HTTP-запрос отправлен. Ожидание ответа... 200 OK
Длина: 140946 (138K) [text/html]
Сохранение в: «index.html.1»

100%[======================================>] 140 946     --.-K/s   за 0,003s

2016-04-03 19:52:14 (51,0 MB/s) - «index.html.1» сохранён [140946/140946]
```

6.6. В заключение

Сетевая подсистема ОС Linux чрезвычайно развита на всех ее уровнях — от сетевых интерфейсов и протоколов до прикладных сетевых служб. На сегодняшний день колоссальное количество сетевых устройств работают под управлением Linux — маршрутизаторы, сетевые хранилища, медиаплееры, TV-боксы, планшеты, смартфоны и прочие «встраиваемые» и мобильные устройства.

К сожалению, рассмотреть весь пласт сетевых возможностей в рамках этой книги не представляется возможным, т. к. потребует от читателя серьезного понимания устройства и функционирования самих сетевых протоколов стека TCP/IP, что не является предметом настоящего рассмотрения.

Основополагающим результатом текущей главы должно стать понимание принципов организации сетевого взаимодействия в Linux, необходимое и достаточное в качестве базы для последующего самостоятельного расширенного и углубленного изучения. Не менее полезными в практике администратора и программиста будут навыки использования инструментов трассировки и мониторинга сетевых сокетов, а в особенно «непонятных» ситуациях — навыки применения анализаторов пакетов.

Исключительное место (эдакий «швейцарский нож») среди прочих сетевых инструментов Linux займет служба SSH, применение которой найдется уже в следующей главе, при распределенном использовании оконной системы X Window System, являющейся основой современного графического интерфейса пользователя.

Глава 7

Графическая система X Window System

Графическая система **W:[X Window System]** предоставляет приложениям операционной системы возможность представления графической информации на графических устройствах вывода, в большинстве случаев — на дисплеях растровых видеотерминалов.

Основной принцип графической системы **X(7)** позволяет множеству графических приложений осуществлять «одновременный» графический вывод за счет *окон*, совместно отображаемых на дисплеях. Содержимое окон определяется их владельцами — графическими приложениями, а управление окнами (их создание, уничтожение, отображение на дисплеях, перекрытие другими окнами, перемещение, изменение размеров и пр.) и возложено на *оконную* систему.

Действительной особенностью оконной системы **X(7)** является ее устройство в виде сетевой службы, что позволяет разным компонентам системы выполняться в виде отдельных процессов на разных узлах сети и взаимодействовать при помощи соответствующих средств, в большинстве случаев — с использованием сетевых сокетов семейств **ip(7)** и **ipv6(7)**. При выполнении компонент оконной системы на одном узле в целях повышения производительности используют локальные (файловые) сокеты **unix(7)** и даже разделяемую память (*см. разд. 4.9.5*, листинг 4.51).

Как любая другая сетевая служба, оконная система **X(7)** состоит из X-сервера и X-клиентов, взаимодействующих между собой посредством **W:[X протокола]**.

7.1. X-сервер

Первичной компонентой оконной системы **X(7)** является X-сервер, основная задача которого заключается в управлении оборудованием графического вывода и ввода. Под управлением X-сервера находятся графические дисплеи (видеоадаптеры и подключенные к ним мониторы), устройства «графического» интерфейса с пользователем (манипуляторы «мышь», трекболы, тачпады, графические планшеты и пр.), а кроме того, устройства «символьного» взаимодействия с пользователем — клавиатуры.

Именно X-сервер принимает подключения от X-клиентов и согласно их запросам создает окна, изменяет их размер, отображает или скрывает окна на дисплеях, сообщает положение курсора, рисует текст, линии, точки, прямоугольники, дуги, полигоны и пр. В обратную сторону X-сервер отправляет X-клиентам информацию о событиях нажатия клавиш клавиатур, кнопок мыши и планшетов, оповещает о движении курсора и т. д.

Листинг 7.1. Аппаратный X-сервер

```
homer@ubuntu:~$ pgrep -l Xorg
1460 Xorg
homer@ubuntu:~$ ps o pid,tty,cmd p 1460
  PID TT       CMD
 1460 tty7 ⓪  /usr/bin/X :0 ① -auth /var/run/lightdm/root/:0 -nolisten tcp vt7 ...
❶ homer@ubuntu:~$ sudo lsof -p 1460 -a /dev
COMMAND  PID USER   FD   TYPE DEVICE SIZE/OFF  NODE NAME
Xorg    1460 root  mem    CHR  226,0          1563 /dev/dri/card0
Xorg    1460 root    7u   CHR    4,7      0t0  1049 /dev/tty7 ⓪
Xorg    1460 root    8u   CHR  10,63      0t0  1026 /dev/vga_arbiter
Xorg    1460 root    9u   CHR  226,0      0t0  1563 /dev/dri/card0
Xorg    1460 root   12u   CHR  13,66      0t0  8232 /dev/input/event2
          ...          ...              ...           ...              ...
Xorg    1460 root   17u   CHR  13,68      0t0  1590 /dev/input/event4
❷ homer@ubuntu:~$ sudo lsof -p 1460 -a -U
COMMAND  PID USER   FD   TYPE     DEVICE SIZE/OFF    NODE NAME
Xorg    1460 root    1u  unix 0xefbbad00      0t0   12475 socket
Xorg    1460 root    3u  unix 0xefbbaa80      0t0   12476 /tmp/.X11-unix/X0 ②
Xorg    1460 root    6u  unix 0xefbba800      0t0   12483 socket
          ...          ...              ...           ...              ...
Xorg    1460 root   60u  unix 0xe3dacf00      0t0  626936 socket
❸ homer@ubuntu:~$ sudo lsof -p 1460 -a -i
!☛
```

Листинг 7.1 иллюстрирует сервер **Xorg(1)**, использующий специальные файлы ❶ устройств видеокарты **/dev/dri/card0** и устройств пользовательского ввода **/dev/input/*** для организации на виртуальном терминале ⓪ **/dev/tty7** графического дисплея ① с номером[1] **:0**.

[1] При наличии нескольких видеокарт можно запустить несколько X-серверов с номерами графических дисплеев **:1**, **:2** и т. д., например, для одновременной работы разных пользователей в оконной системе X.

Для взаимодействия с X-клиентами сервер создает ❷ локальный (файловый) сокет ② **/tmp/.X11-unix/X0**, но не создает сетевой сокет ❸ (в силу своих параметров запуска **-nolisten tcp**), поэтому подключение X-клиентов по сети к такому серверу невозможно.

Основные свойства X-сервера и управляемых им графических дисплеев можно получить при помощи утилиты **xdpyinfo(1)**, см. листинг 7.2.

Листинг 7.2. Свойства дисплея X-сервера

```
homer@ubuntu:~$ xdpyinfo | grep -A 4 screen
default screen number:    0
number of screens:    1

screen #0:
  dimensions:    1366x768 pixels (361x203 millimeters)
  resolution:    96x96 dots per inch
  depths (7):    24, 1, 4, 8, 15, 16, 32
root window id:    0xae
```

Оконная система **X(1)** и ее X-протокол на текущий момент времени обросли массой *расширений*, одним из которых является расширение **W:[RANDR]**, позволяющее управлять сменой графического разрешения, ориентацией, подключением и отключением видеовыходов дисплея X-сервера и т. д. В примере из листинга 7.3 иллюстрируется утилита **xrandr(1)**, реализующая расширение **RANDR**, которая может использоваться как для запроса ❶ списка видеовыходов (и поддерживаемых ими видеорежимов работы), так и для переключения ❷ заданного видеовыхода в тот или иной видеорежим и смены ориентации изображения.

Листинг 7.3. Расширения X-сервера: RANDR

```
homer@ubuntu:~$ xdpyinfo | grep RANDR
    RANDR
❶ homer@ubuntu:~$ xrandr
Screen 0: minimum 320 x 200, current 1366 x 768, maximum 8192 x 8192
LVDS1 connected 1366x768+0+0 (normal left inverted right x axis y axis) 309mm x 174mm
   1366x768       60.1*+
   1360x768       59.8     60.0
   1024x768       60.0
   800x600        60.3     56.2
   640x480        59.9
```

```
VGA1 disconnected (normal left inverted right x axis y axis)
HDMI1 disconnected (normal left inverted right x axis y axis)
DP1 disconnected (normal left inverted right x axis y axis)
❷ homer@ubuntu:~$ xrandr --output LVDS1 --mode 1024x768 --rotate left
```

Расширение **GLX** является реализацией программного интерфейса **W:[OpenGL]** для оконной системы **X(1)** и предназначено для передачи «команд» OpenGL внутри X-протокола. Программный интерфейс OpenGL используется приложениями, интенсивно работающими с 2D/3D-графикой (в большинстве случаев[1] — игры, системы моделирования и визуализации) и нацеленными на аппаратный рендеринг «команд» OpenGL на стороне сервера.

Утилита **glxinfo(1)**, показанная в листинге 7.4, предназначена для запроса свойств **GLX** расширения и свойств самого OpenGL-рендерера.

Листинг 7.4. Расширения X-сервера: GLX

```
homer@ubuntu:~$ xdpyinfo |grep GLX
    GLX
    SGI-GLX
homer@ubuntu:~$ glxinfo | grep -E "render(er|ing)|version"
direct rendering: Yes
server glx version string: 1.4
client glx version string: 1.4
GLX version: 1.4
OpenGL renderer string: Mesa DRI Intel(R) Sandybridge Mobile x86/MMX/SSE2
OpenGL version string: 3.0 Mesa 8.0.4
OpenGL shading language version string: 1.30
```

Необходимо заметить, что все эти утилиты — **xdpyinfo(1)**, **xrandr(1)** и **glxinfo(1)** — являются простейшими X-клиентами и используют X-протокол для опроса X-сервера.

7.2. X-клиенты и X-протокол

Второй важной компонентой оконной системы **X(7)** являются X-клиенты — приложения, получающие в свое распоряжение окна и отображающие в них графическую информацию. Точнее, X-клиенты всего лишь взаимодействуют с сервером

[1] Хотя на сегодняшний день даже «обычные» пользовательские окружения **GNOME** и **KDE** интенсивно используют OpenGL для спецэффектов.

при помощи X-протокола и могут вовсе не создавать окон, как это делают «простейшие» **xdpyinfo(1)**, **xrandr(1)** и **glxinfo(1)**, **xlsclients(1)**, **xwininfo(1)**, **xprop(1)**, но «полновесные» X-клиенты всегда создают хотя бы одно окно.

Список созданных окон (упорядоченный в дерево[1]) и системные *атрибуты* каждого из них можно запросить у X-сервера при помощи **xwininfo(1)**. В листинге 7.5 построено полное дерево окон (начиная с «корневого» окна самого X-сервера), среди которых отобраны окна с именем **gnome-terminal** (предположительно принадлежащие **gnome-terminal(1)**, узнать явно нет возможности). При этом оказывается, что только одно окно изображается ❷ X-сервером на дисплее, а еще одно находится в скрытом состоянии ❶.

Листинг 7.5. Дерево окон X-сервера и их атрибуты

```
homer@ubuntu:~$ xwininfo -tree -root | grep gnome-terminal
     0x4400001 "Терминал": ("gnome-terminal" "Gnome-terminal")  10x10+10+10  +10+10
           0x4400006 "homer@ubuntu: ~": ("gnome-terminal" "Gnome-terminal")  882x530+0+0  +483+51

homer@ubuntu:~$ xwininfo -id 0x4400001
            ...             ...             ...             ...             ...
  Map State: IsUnMapped ❶
homer@ubuntu:~$ xwininfo -id 0x4400006
            ...             ...             ...             ...             ...
  Absolute upper-left X:  483
  Absolute upper-left Y:  51
          ...             ...             ...             ...             ...
  Width: 882
  Height: 530
  Depth: 32
          ...             ...             ...             ...             ...
  Map State: IsViewable ❷
          ...             ...             ...             ...             ...
```

Кроме системных атрибутов, каждое окно наделяется *свойствами* (properties), широко используемыми для взаимодействия между X-клиентами (см. **W**:[ICCCM]), особенно между «обычными» клиентами и оконным менеджером (**w**indow **m**anager) *(см. разд. 7.2)*. Оконный менеджер является «особенным» X-клиентом, обрабатывающим события создания окон «обычных» X-клиентов. Именно он добавляет к «чужим» создаваемым окнам «свои» декорирующие элементы: заголовок (title) окна — для его перемещения, бордюр (border) — для изменения его размеров и т. д.

---

[1] Окна в **X Window System** связаны дочерне-родительскими отношениями, определяемыми при создании окна.

В листинге 7.6 показаны некоторые свойства окна, установленные программой-владельцем окна для оконного менеджера. Например, свойство **WM_NAME** используется оконным менеджером для текста заголовка (отображаемых) окон, свойство **WM_LOCALE_NAME** указывает на язык и кодировку текста, содержащегося в **WM_NAME**, свойство **WM_CLIENT_MACHINE** содержит имя сетевого узла X-клиента, а свойство **WM_COMMAND** — команду, при помощи которой был запущен клиент.

Листинг 7.6. Свойства окон X-сервера

```
homer@ubuntu:~$ xprop -id 0x4400001
   ...          ...                    ...                ...          ...
WM_CLASS(STRING) = "gnome-terminal", "Gnome-terminal"
WM_COMMAND(STRING) = { "gnome-terminal" }
WM_CLIENT_MACHINE(STRING) = "ubuntu"
WM_LOCALE_NAME(STRING) = "ru_RU.UTF-8"
WM_ICON_NAME(STRING) = "gnome-terminal"
WM_NAME(COMPOUND_TEXT) = "Терминал"
   ...          ...                    ...                ...          ...
homer@ubuntu:~$ xprop -id 0x4400006
WM_NAME(STRING) = "homer@ubuntu: ~"
```

Свойства окон X-клиентов могут использоваться не только оконным менеджером, но и другими клиентами, например **xlsclients(1)**, который опрашивает X-сервер (см. листинг 7.7) и выводит список «клиентов», имеющих окна с установленными свойствами **WM_CLIENT_MACHINE** и **WM_COMMAND**, которые в действительности устанавливаются (не всегда) X-клиентами только для «основных» окон.

Листинг 7.7. Список клиентов X-сервера

```
homer@ubuntu:~$ xlsclients -l | grep -C 3 gnome-terminal
Window 0x4400001:
  Machine:  ubuntu
  Name:  <unknown type COMPOUND_TEXT (363) or format 8>
  Icon Name:  gnome-terminal
  Command:  gnome-terminal
  Instance/Class:  gnome-terminal/Gnome-terminal
Window 0x3000001:
  Machine:  ubuntu
  Name:  gdu-notification-daemon
```

Взаимодействие X-клиентов и X-сервера происходит при помощи локальных или сетевых сокетов в зависимости от их взаимного месторасположения и согласно

адресу подключения, указываемому на стороне X-клиентов при помощи переменной окружения **DISPLAY** в формате **host:number**.

Адрес подключения состоит из имени (или IP-адреса) узла X-сервера **host** и номера его дисплея **number** (например, **DISPLAY=ubuntu.local:0** или **192.168.0.5:0**). В качестве **host** может выступать любое имя, для которого можно получить IP-адрес, используя службу имен *(см. разд. 6.3)*, а в случае локального взаимодействия **host** не указывается вовсе (т. е. **DISPLAY=:0**).

Номер дисплея указывает на экземпляр X-сервера, запущенного на узле **host** для управления некоторым набором оборудования (графическими дисплеями — видеокартами и мониторами, клавиатурами и пр.).

Проиллюстрировать разные виды X-взаимодействия проще всего при помощи «виртуальных» X-серверов **Xnest(1)** и **Xephyr(1)**. В отличие от «обычного» X-сервера, организующего доступ X-клиентов к аппаратному дисплею, эти серверы организуют доступ своих клиентов к виртуальному «дисплею», в качестве которого выступает их собственное окно аппаратного сервера.

В листинге 7.8 показан запуск X-сервера **Xnest(1)** с номером дисплея **:1** (т. к. дисплей **:0** обычно занят ❶ аппаратным X-сервером), который создает сокет ❷ семейства **ip(7)** для подключения сетевых клиентов и сокет ❸ семейства **unix(7)** для локальных.

Листинг 7.8. Виртуальный X-сервер Xnest

```
   homer@ubuntu:~$ Xnest :0 &
   Fatal server error:
❶ Server is already active for display 0
        If this server is no longer running, remove /tmp/.X0-lock
        and start again.
   homer@ubuntu:~$ Xnest :1 &
   [1] 10365
   homer@ubuntu:~$ lsof -p 10365
   COMMAND   PID   USER  FD   TYPE    DEVICE SIZE/OFF     NODE NAME
   Xnest   10365  homer cwd    DIR     252,0    69632 20316162 /home/kotoff
   Xnest   10365  homer rtd    DIR     252,0     4096        2 /
   Xnest   10365  homer txt    REG     252,0  1754384  3831580 /usr/bin/Xnest
             ...          ...            ...           ...          ...
❷ Xnest   10365  homer 1u  IPv4  17552891      0t0      TCP *:x11-1 (LISTEN) ☚
   Xnest   10365  homer 2u   CHR     136,1      0t0        4 /dev/pts/1
   Xnest   10365  homer 3u  unix 0x00000000     0t0 17552892 socket
❸ Xnest   10365  homer 4u  unix 0x00000000     0t0 17552893 /tmp/.X11-unix/X1 ☚
   Xnest   10365  homer 5r   REG     252,0    31246  6817286 /usr/lib/xorg/protocol.txt
   Xnest   10365  homer 6u  unix 0x00000000     0t0 17552896 socket
```

В примере из листинга 7.9 показана трассировка демонстрационного X-клиента **xeyes(1)**, иллюстрирующая использование клиентом локального ❶ и сетевого ❷ сокетов, в зависимости от значения переменной окружения **DISPLAY**.

Листинг 7.9. Сетевые и локальные подключения клиента X-сервера

```
homer@ubuntu:~$ DISPLAY=:1 strace -fe connect xeyes
❶ connect(3, {sa_family=AF_FILE, path=@"/tmp/.X11-unix/X1" ↩}, 20) = 0
^C
homer@ubuntu:~$ DISPLAY=ubuntu.local:1 strace -fe connect xeyes
   ...          ...          ...          ...          ...          ...
❷ connect(3, {sa_family=AF_INET, sin_port=htons(6001 ↩), sin_addr=inet_addr("192.168.100.5 ↩")}, 16) = 0
^C
```

При работе в оконной системе переменную **DISPLAY** обычно устанавливают в начале сеанса, а затем при запуске X-клиентов их вывод будет попадать на нужный X-сервер, как это показано на примере[1] сервера **Xnest(1)** и клиентов **xterm(1)**, **xcalc(1)** и **xeyes(1)** в листинге 7.10 и на рис. 7.1.

Впоследствии при запуске одними X-клиентами других X-клиентов (например, по команде пользователя из эмулятора терминала **xterm(1)**, см. листинг 7.10) переменная окружения наследуется *(см. разд. 4.5.3)* порожденными процессами, и вывод запускаемых клиентов «интуитивным» (так же как и стандартные потоки ввода-вывода) образом попадает на тот же X-сервер.

Листинг 7.10. X-сеанс пользователя

```
homer@ubuntu:~$ export DISPLAY=:1
homer@ubuntu:~$ xterm & xcalc & xeyes &
[1] 2803
[2] 2804
[3] 2805
```

Нужно заметить, что все окна по умолчанию открываются в левом верхнем углу (координата +0+0) экрана, а возможность управлять их размером и местоположением отсутствует. Каждый X-клиент может самостоятельно управлять размером и местоположением своих окон, и даже многие из них запоминают при выходе эти параметры в своих dot-файлах *(см. разд. 2.8)*, а при последующем запуске вос-

[1] Все иллюстрации дальше делаются именно при помощи «виртуальных» X-серверов **Xnest(1)** или **Xephyr(1)** и могут быть без проблем воспроизведены пользователем, уже работающим в оконной системе.

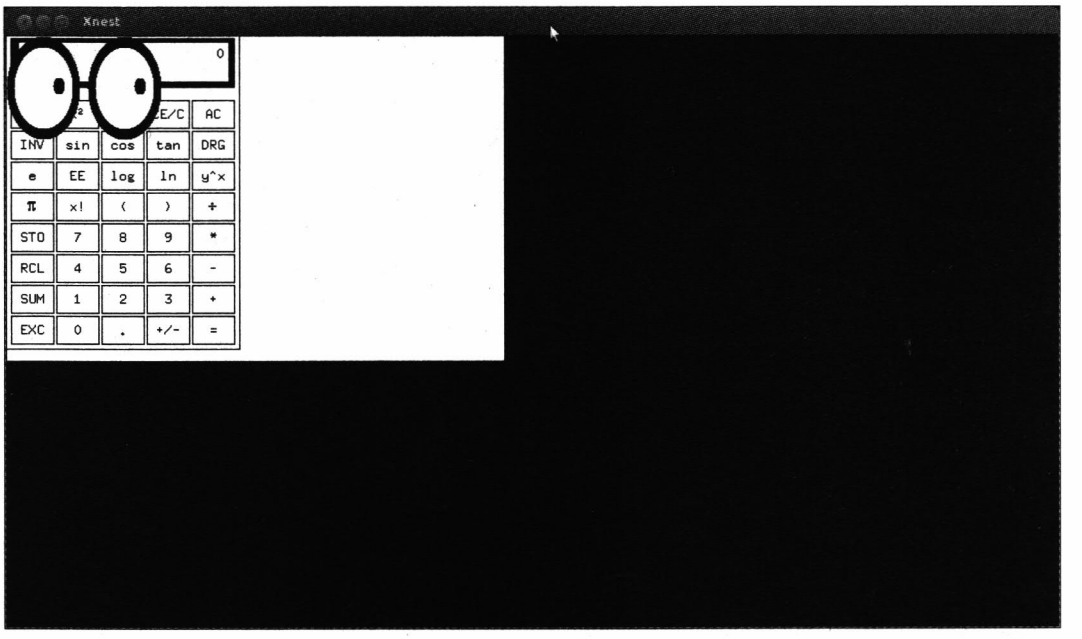


Рис. 7.1. X-клиенты на дисплее X-сервера: X-сеанс пользователя

станавливают окна в прежних размерах и местах, но интерактивным перемещением окон и изменением их размера занимается *оконный менеджер*.

7.3. Оконные менеджеры

Третьей важной составляющей оконной системы, позволяющей пользователю *интерактивно* манипулировать окнами, является оконный менеджер. Одним из самых ранних оконных менеджеров является **W:[twm]**, проиллюстрированный на рис. 7.2 и в листинге 7.11.

Листинг 7.11. Оконный менеджер twm

```
homer@ubuntu:~$ twm &
[4] 2880
twm: warning: font for charset MICROSOFT-CP1251 is lacking.
             ...          ...          ...          ...          ...
```

Кроме возможности интерактивного перемещения, изменения размеров окон, сворачивания их в значок на «рабочем столе» и разворачивания[1], оконный менеджер

[1] Аналогом привычной «панели задач» в **twm(1)** являются приложения, свернутые в значки, которые располагаются непосредственно на «рабочем столе», т. е. корневом окне X-сервера.

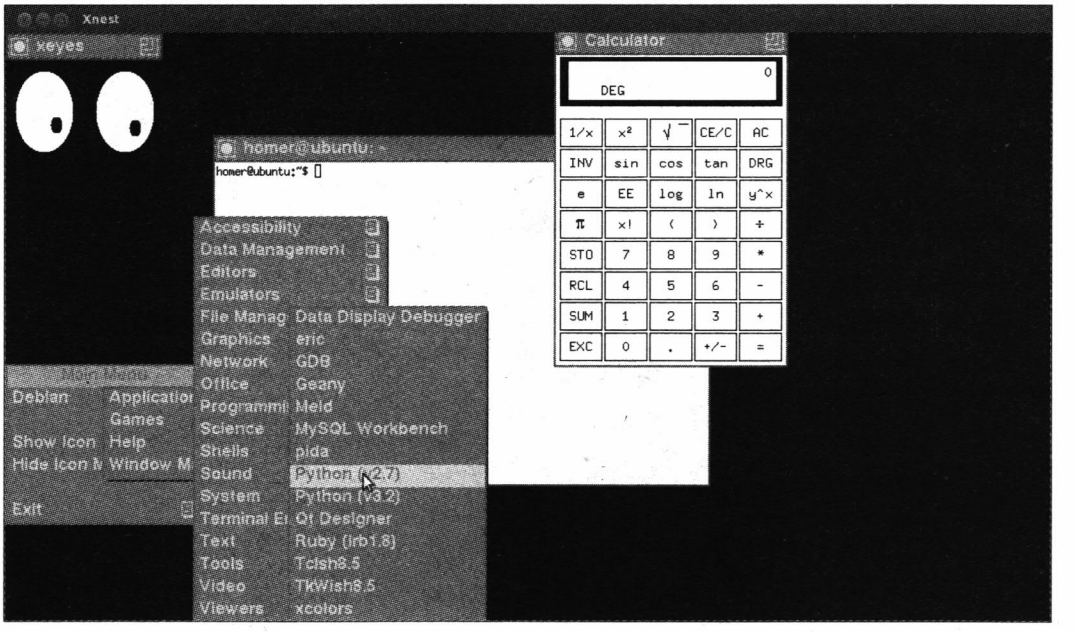


Рис. 7.2. Оконный менеджер twm — Tab Window Manager

twm(1) имеет «главное меню» (вызываемое нажатием левой кнопки мыши непосредственно на «рабочем столе»), позволяющее запускать графические приложения.

Другим, не менее древним оконным менеджером является **W:[olwm]**, доставшийся в наследство от настольного окружения W:[OpenWindows] из операционной системы **W:[SunOS]**. В листинге 7.12 представлена попытка запуска оконного менеджера **olwm(1)** при уже запущенном **twm(1)**, закончившаяся неудачно, потому что управлять окнами одного X-сервера одномоментно может только один ❶ оконный менеджер.

После завершения ❷ предыдущего оконного менеджера новый менеджер заново декорирует окна тех же X-клиентов в своем стиле, как показано на рис. 7.3. Аналогично **twm(1)**, приложения сворачиваются в значки «рабочего стола», а для запуска графических приложений используется главное меню (вызываемое нажатием правой кнопки мыши).

Листинг 7.12. Оконный менеджер olwm

```
homer@ubuntu:~$ olwm
❶ olwm: fatal X protocol error -- BadAccess (attempt to access private resource denied)
    request major code:  2 (X_ChangeWindowAttributes)
    request minor code:  0
    resource ID in failed request:  0x114
    serial number of failed request:  113
☛ Perhaps there is another window manager running?
```

```
homer@ubuntu:~$ kill %twm
homer@ubuntu:~$:jobs %+
[4]+  Готово                  twm
homer@ubuntu:~$ olwm &
[4] 2942
```

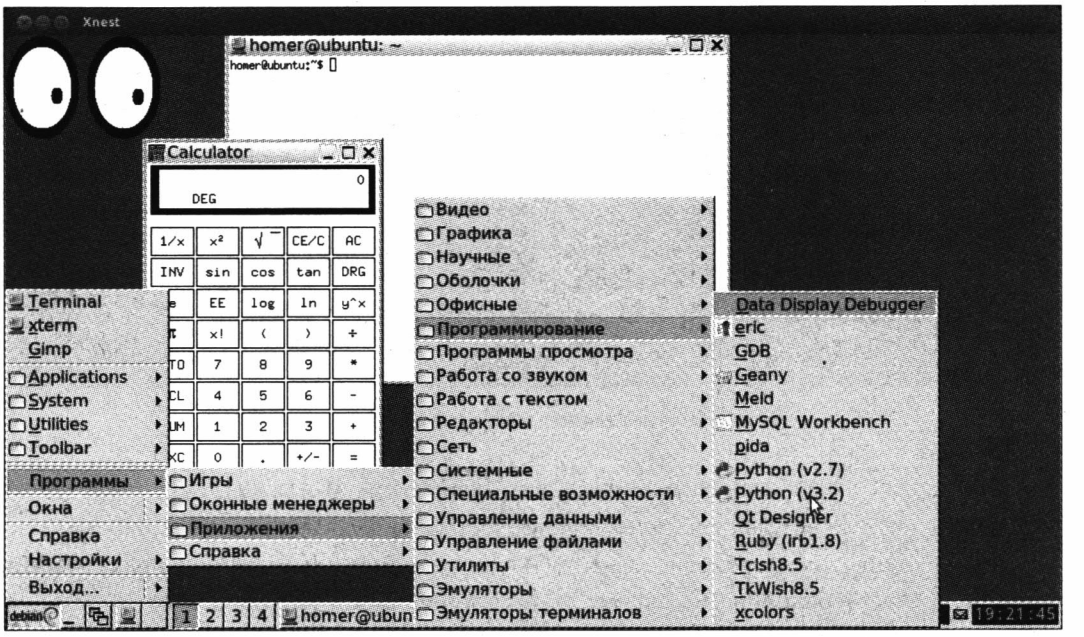


Рис. 7.3. Оконный менеджер olwm — OpenLook Window Manager

Еще один оконный менеджер «из прошлого», **W:[Motif Window Manager]**, **mwm(1)**, являвшийся частью настольного окружения **W:[CDE]**, проиллюстрирован в листинге 7.13 и на рис. 7.4. Как и **twm(1)** и **olwm(1)**, окна приложений под управлением **mwm(1)** при минимизации сворачиваются в значки на «рабочем столе», а главное меню вызывается правой кнопкой мыши.

Листинг 7.13. Оконный менеджер mwm

```
homer@ubuntu:~$ kill %olwm
homer@ubuntu:~$ jobs %+
[4]+  Выполняется  olwm &
homer@ubuntu:~$ kill -KILL %+
homer@ubuntu:~$ jobs %+
[4]+  Убито                   olwm
homer@ubuntu:~$ mwm &
[4] 3025
```

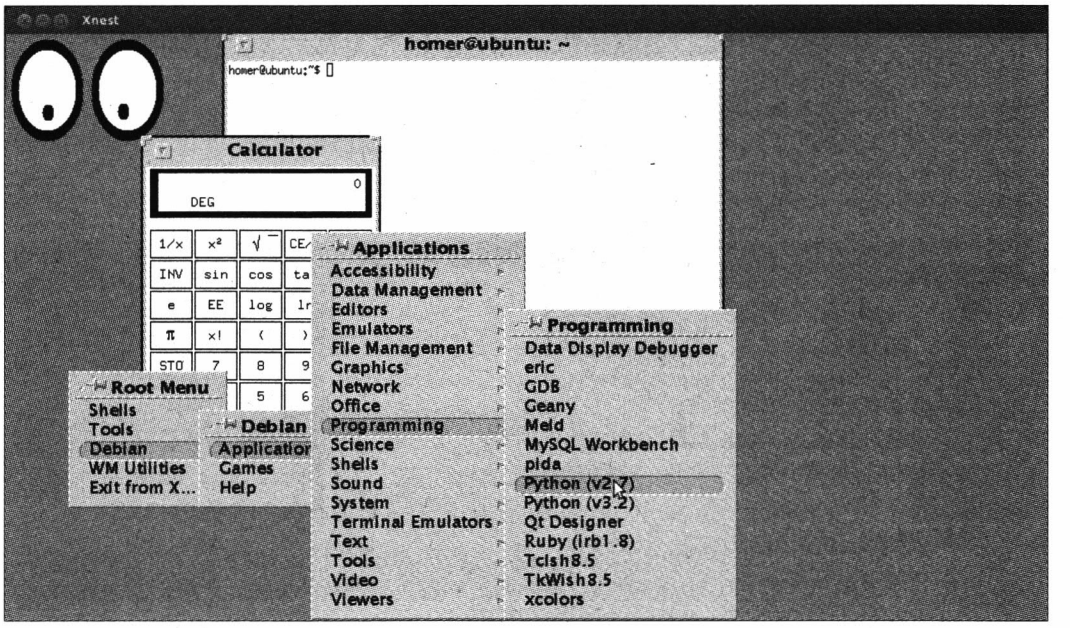


Рис. 7.4. Оконный менеджер mwm — Motif Window Manager

Более поздние оконные менеджеры, как например **W:[IceWM]**, представленный в листинге 7.14 и на рис. 7.5, зачастую имеют «панель задач» снизу, кнопку «пуск» с главным меню слева панели задач, область уведомлений («трей») справа панели задач и прочие «современные» элементы пользовательского интерфейса.

Листинг 7.14. Оконный менеджер icewm

```
homer@ubuntu:~$ kill %mwm
homer@ubuntu:~$ jobs %+
[4]+  Выполняется  mwm &
homer@ubuntu:~$ kill -9 %+
homer@ubuntu:~$ jobs %+
[4]+  Убито                mwm
homer@ubuntu:~$ icewm-session &
[4] 3120
          ...          ...          ...          ...          ...
Xlib:  extension "RANDR" missing on display ":1"
homer@ubuntu:~$ pstree 3120
icewm-session─┬─icewm
              ├─icewmbg
              └─icewmtray
```

Нужно заметить ★, что сеанс **icewm(1)** запускается командой **icewm-session(1)** и состоит из трех компонент: самого оконного менеджера **icewm(1)** и двух «подчинен-

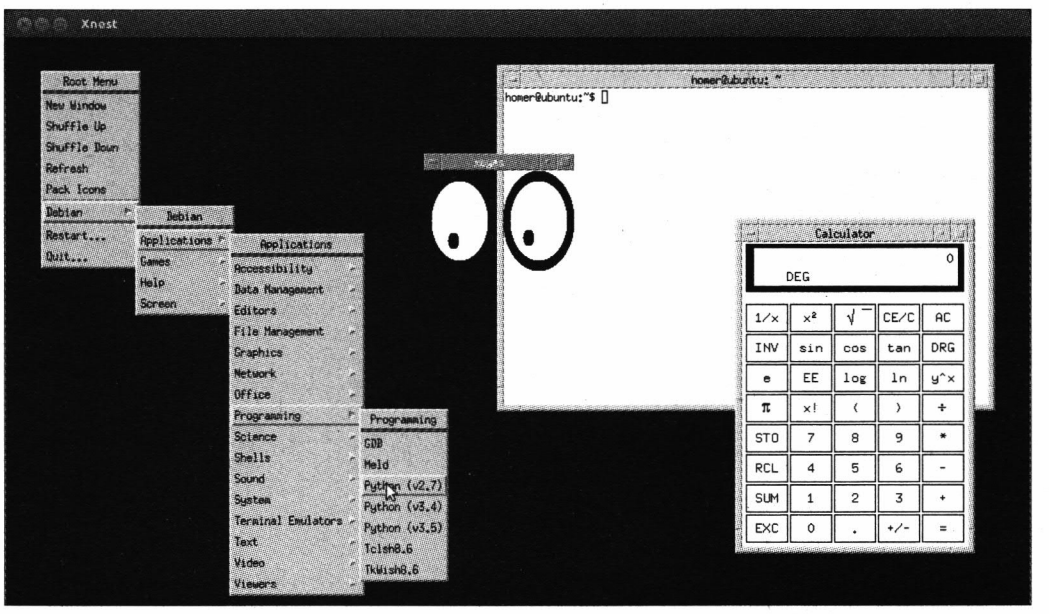


Рис. 7.5. Оконный менеджер icewm — ICE Window Manager

ных» **icewmbg(1)** и **icewmtray(1)**, управляющих изображением на рабочем столе и областью уведомлений, соответственно.

7.4. Настольные пользовательские окружения

Современные графические пользовательские среды[1], такие как **W:[GNOME]**, **W:[KDE]**, **W:[XFCE]** и **W:[LXDE]**, с точки зрения оконной системы **X(1)** являются всего лишь набором взаимодействующих (по собственным законам[2] этой среды) X-клиентов, запускаемых *менеджером сеансов*.

На рис. 7.6 и в листинге 7.15 проиллюстрирована среда GNOME, запущенная менеджером сеансов ❶ **gnome-session(1)**, состоящая из достаточно большого ❷ количества компонент, включая оконный менеджер ❸ **compiz(1)**.

Листинг 7.15. Настольное окружение GNOME

```
homer@ubuntu:~$ kill %icewm
          ...          ...          ...          ...          ...
Xlib:  extension "RANDR" missing on display ":1".
```

[1] Зачастую используют метафору «рабочего стола», на что именуются «настольным окружением» (desktop environment).

[2] В современных средах практически безальтернативно используется **W:[D-Bus]**, но в более ранних версиях использовались **W:[DCOP]** в KDE и **W:[CORBA]** в GNOME.

```
homer@ubuntu:~$ jobs %+
[4]+  Готово                  icewm-session
homer@ubuntu:~$ gnome-session --session=gnome &>/dev/null </dev/null &
[4] 8496
homer@ubuntu:~$ pstree 8496
gnome-session─┬─bluetooth-apple───2*[{bluetooth-apple}]
              ├─compiz─┬─sh───gtk-window-deco───2*[{gtk-window-deco}]
              │        ├─sh───gnome-terminal─┬─bash───pstree
              │        │                     ├─gnome-pty-helpe
              │        │                     └─3*[{gnome-terminal}]
              │        └─3*[{compiz}]
              ├─deja-dup-monito───2*[{deja-dup-monito}]
              ├─evolution-alarm───2*[{evolution-alarm}]
              ├─gdu-notificatio---2*[{gdu-notificatio}]
              ├─gnome-screensav───2*[{gnome-screensav}]
              ├─gnome-settings-─┬─syndaemon
              │                 └─3*[{gnome-settings-}]
              ├─gnome-shell───5*[{gnome-shell}]
              ├─gpg-agent
              ├─nautilus───2*[{nautilus}]
              ├─nm-applet───2*[{nm-applet}]
              ├─ssh-agent
              ├─update-notifier───3*[{update-notifier}]
              ├─vino-server───2*[{vino-server}]
              ├─zeitgeist-datah───{zeitgeist-datah}
              └─3*[{gnome-session}]
```

Среда KDE, представленная в листинге 7.16 и на рис. 7.7, запускается при помощи специальной команды **startkde(1)** и состоит из еще большего числа компонент ❶,

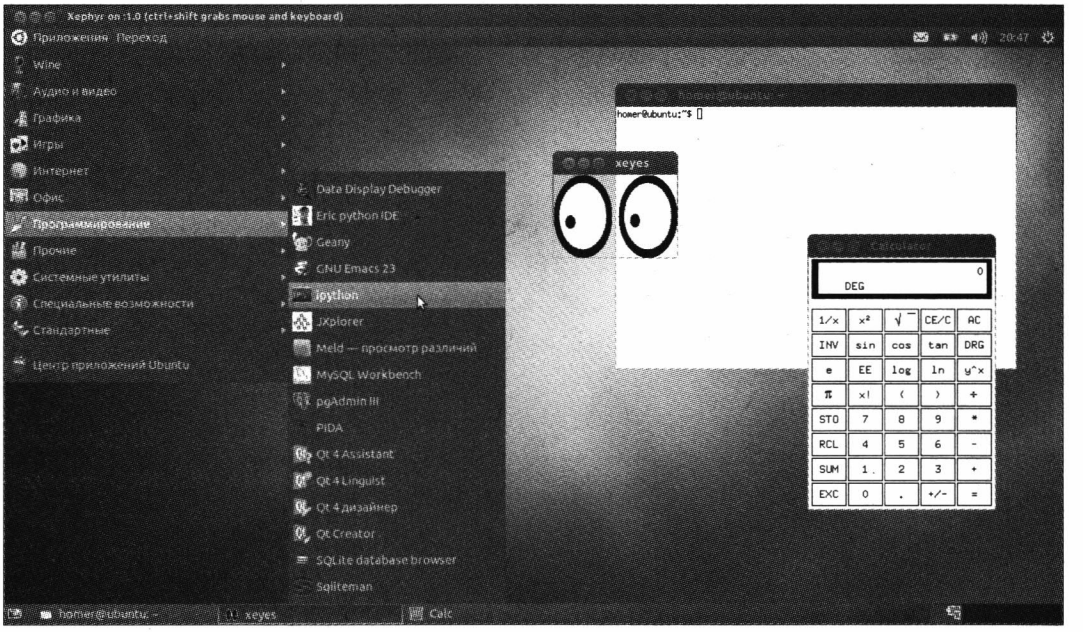


Рис. 7.6. Настольное окружение GNOME

среди которых неизменно присутствуют менеджер сеансов **ksmserver** и оконный менеджер **kwin**.

Листинг 7.16. Настольное окружение KDE

```
homer@ubuntu:~$ kill %gnome-session
homer@ubuntu:~$ jobs %+
[4]+  Завершено       gnome-session --session=gnome &>/dev/null < /dev/null
homer@ubuntu:~$ startkde &>/dev/null </dev/null &
[4] 7391
homer@ubuntu:~$ pstree 7391
startkde──kwrapper4
homer@ubuntu:~$ pstree $(whoami)
                ...          ...          ...          ...          ...
kded4──2*[{kded4}]

kdeinit4─┬─deja-dup-monito──2*[{deja-dup-monito}]
         ├─2*[kio_trash──{kio_trash}]
         ├─klauncher
         ├─ksmserver─┬─kwin──6*[{kwin}]
         │           └─{ksmserver}
         ├─nepomukserver─┬─nepomukservices─┬─virtuoso-t──5*[{virtuoso-t}]
         │               │                 └─11*[{nepomukservices}]
         │               ├─2*[nepomukservices]
         │               ├─2*[nepomukservices──2*[{nepomukservices}]]
         │               └─{nepomukserver}
         └─zeitgeist-datah──{zeitgeist-datah}
                ...          ...          ...          ...          ...
zeitgeist-fts─┬─cat
              └─{zeitgeist-fts}
```

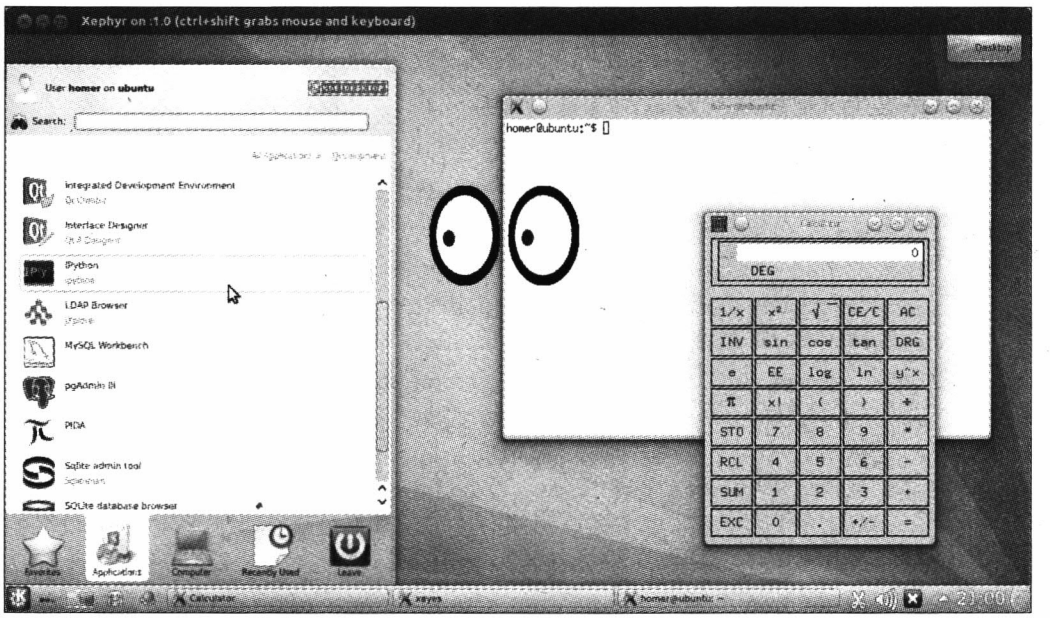


Рис. 7.7. Настольное окружение KDE

7.5. Библиотеки интерфейсных элементов

Возможности X-протокола позволяют прорисовывать лишь простейшие графические примитивы, как то: линии, точки, прямоугольники, дуги, полигоны, растровые изображения и текст — при помощи растровых или векторных шрифтов, поддерживаемых X-сервером. Все интерфейсные элементы управления приложениями, такие как кнопки, полосы прокрутки, закладки, меню, списки и пр., должны быть самостоятельно составлены X-клиентами из простейших примитивов. Для решения этой задачи X-клиенты используют специальные *библиотеки интерфейсных элементов* (*виджетов*, widget), такие как **W:[Xaw]**, **Xview**, **W:[Motif]**, **W:[Tk]**, **W:[Qt]**, **W:[Gtk]** и др.[1].

Самими «простыми» (доставшимися в наследство от UNIX) библиотеками виджетов являются библиотеки Xaw и Motif, разработанные на основе базовой Xt, **W:[X Toolkit Intrinsics]**, а так же «альтернативная»[2] XView.

Более «современным» видом виджетов обладают наиболее распространенные на сегодняшний день библиотеки Qt и Gtk, на основе которых разрабатываются пользовательские среды KDE и GNOME.

В примере из листинга 7.17 иллюстрируется использование различных X-библиотек, среди которых библиотека самого X-протокола ❶ **W:[Xlib]**, библиотеки виджетов Xt и Xaw ❷ и библиотеки расширений X-протокола ❸, например **W:[XRender]**.

Листинг 7.17. Библиотеки Xlib, Xt и Xaw

```
homer@ubuntu:~$ ldd $(which xeyes) | grep -i libX
        libXext.so.6 => /usr/lib/i386-linux-gnu/libXext.so.6 (0xb7781000)
        libXmu.so.6 => /usr/lib/i386-linux-gnu/libXmu.so.6 (0xb7768000)
❷       libXt.so.6 => /usr/lib/i386-linux-gnu/libXt.so.6 (0xb770b000)
❶       libX11.so.6 => /usr/lib/i386-linux-gnu/libX11.so.6 (0xb75d7000)
❸       libXrender.so.1 => /usr/lib/i386-linux-gnu/libXrender.so.1 (0xb75cd000)
        libXau.so.6 => /usr/lib/i386-linux-gnu/libXau.so.6 (0xb73a2000)
        libXdmcp.so.6 => /usr/lib/i386-linux-gnu/libXdmcp.so.6 (0xb739b000)
homer@ubuntu:~$ ldd $(which xcalc) | grep -i Xaw
❷       libXaw.so.7 => /usr/lib/i386-linux-gnu/libXaw.so.7 (0xb76fb000)
```

[1] См. **W:[wxWidgets]**, **W:[FLTK]**.

[2] См. **W:[OpenLook]**.

Аналогично, в листинге 7.18 показаны X-клиенты, использующие библиотеки виджетов **XView** и **Motif**, а в листинге 7.19 — клиенты, использующие **Tk**, **Gtk** и **Qt**.

Листинг 7.18. Библиотеки декорирования XView и Motif

```
homer@ubuntu:~$ ldd $(which clock) | grep -i libXView
        libxview.so.3 => /usr/lib/libxview.so.3 (0xb7595000)
homer@ubuntu:~$ ldd $(which mwm) | grep -i libXm
        libXm.so.4 => /usr/lib/i386-linux-gnu/libXm.so.4 (0xb7458000)
        libXmu.so.6 => /usr/lib/i386-linux-gnu/libXmu.so.6 (0xb70f2000)
```

Листинг 7.19. Библиотеки декорирования Tk, Gtk и Qt

```
homer@ubuntu:~$ ldd $(which wish) | grep -i libTk
        libtk8.5.so.0 => /usr/lib/libtk8.5.so.0 (0xb7620000)
homer@ubuntu:~$ ldd $(which nautilus) | grep -i libGtk
        libgtk-3.so.0 => /usr/lib/i386-linux-gnu/libgtk-3.so.0 (0xb70f1000)
homer@ubuntu:~$ ldd $(which kcalc) | grep -i libQtGui
        libQtGui.so.4 => /usr/lib/i386-linux-gnu/libQtGui.so.4 (0xb5fec000)
```

7.6. Запуск X Window System

7.6.1. Локальный запуск X-клиентов

В примерах из листингов 7.8 и 7.10 показан «ручной» запуск компонент оконной системы X; что достаточно неудобно, т. к. требует установить правильное значение переменной окружения **DISPLAY**, запустить X-сервер до запуска первого клиента, а завершить после завершения последнего и т. д. Поэтому для автоматизации запуска X-сервера и всех клиентов X-сеанса пользователя предназначается специальная команда **xinit(1)**, проиллюстрированная в листинге 7.20.

Листинг 7.20. Запуск сеанса IceWM с отображением на сервере Xnest

```
homer@ubuntu:~$ xinit /usr/bin/icewm-session -- /usr/bin/Xnest :1 &
          ...          ...             ...             ...             ...
homer@ubuntu:~$ ps f
  PID TTY      STAT   TIME COMMAND
28163 pts/0    Ss     0:00 bash
28218 pts/0    S      0:00  \_ xinit /usr/bin/icewm-session -- /usr/bin/Xnest :1
28219 pts/0    Sl     0:00  |   \_ /usr/bin/Xnest :1
28226 pts/0    S      0:00  |   \_ /usr/bin/icewm-session
28231 pts/0    R+     0:00  \_ ps f
```

Запускатель **xinit(1)** может быть сконфигурирован пользовательскими dot-файлами **~/.xinitrc** и **~/.xserverrc**, представляющими собой сценарии командного интерпретатора, которые запускают клиентскую и серверную части оконной системы соответственно.

В примере из листинга 7.21 показан сценарий **.xserverrc**, позволяющий **xinit(1)** использовать виртуальный X-сервер **Xnest(1)** по умолчанию на указанном дисплее, что сокращает командную строку запуска.

Листинг 7.21. Запуск сеанса TWM с отображением на сервере «по умолчанию» (Xnest)

```
homer@ubuntu:~$ cat .xserverrc
#!/bin/sh
exec /usr/bin/Xnest "$@"
homer@ubuntu:~$ xinit /usr/bin/twm -- :2 &
          ...          ...          ...          ...          ...
homer@ubuntu:~$ ps f
  PID TTY      STAT   TIME COMMAND
28268 pts/1    Ss     0:00 bash
28322 pts/1    S      0:00  \_ xinit /usr/bin/twm -- :2
28323 pts/1    Sl     0:01  |   \_ /usr/bin/Xnest :2
28330 pts/1    S      0:00  |   \_ /usr/bin/twm
30112 pts/1    R+     0:00  \_ ps f
```

В примере из листинга 7.22 показан сценарий **.xinitrc**, позволяющий **xinit(1)** использовать X-сеанс среды KDE по умолчанию, что еще более сокращает командную строку запуска.

Листинг 7.22. Запуск локальной X Window System (окружение KDE с отображением на сервере Xnest)

```
homer@ubuntu:~$ cat .xserverrc
#!/bin/sh
exec /usr/bin/Xnest "$@"
homer@ubuntu:~$ cat .xinitrc
#!/bin/sh
exec startkde "$@"
homer@ubuntu:~$ xinit -- :3 &
          ...          ...          ...          ...          ...
homer@ubuntu:~$ ps f
  PID TTY      STAT   TIME COMMAND
30810 pts/2    Ss     0:00 bash
30996 pts/2    S      0:00  \_ xinit -- :3
30997 pts/2    Sl     0:04  |   \_ /usr/bin/Xnest :3
```

```
31004 pts/2    S      0:00  |   \_ /bin/sh /usr/bin/startkde
31064 pts/2    S      0:00  |       \_ kwrapper4 ksmserver
31179 pts/2    R+     0:00  \_ ps f
```

Нужно заметить, что все три пользовательских сеанса, запущенных из листингов 7.20–7.22, могут существовать одновременно, т. к. используют разные экземпляры X-сервера.

7.6.2. Дистанционный запуск X-клиентов

Оконная система X изначально проектировалась для распределенной работы ее компонент на различных узлах сети, что достаточно широко используется на практике, когда нужно запускать графические приложения на удаленном узле, их окна отображать на локальном дисплее.

В большинстве случаев для дистанционного запуска используется **ssh(1)**, что проиллюстрировано в листинге 7.23. Попытка ❶ «прямого» запуска **xeyes(1)** на узле **centos** от лица пользователя **lich** не увенчалась успехом потому, что в большинстве инсталляций аппаратный X-сервер на дисплее **:0** не принимает сетевые соединения (см. листинг 7.1). Попытка ❷ запуска локального виртуального сервера **Xnest(1)** на дисплее **:1** с перенаправлением ему вывода дистанционного **xeyes(1)** тоже оказалась неудачной, но уже по другим причинам.

Листинг 7.23. Запуск дистанционного X-клиента

```
❶ homer@ubuntu:~$ ssh -f lich@centos "DISPLAY=ubuntu.local:0 xeyes"
  Error: Can't open display: ubuntu.local:0
❷ homer@ubuntu:~$ Xnest :1 &
                ...          ...          ...          ...          ...
  homer@ubuntu:~$ ssh -f lich@centos "DISPLAY=ubuntu.local:1 xeyes"
☛ No protocol specified
  Error: Can't open display: ubuntu.local:1
```

При сетевом взаимодействии X-клиентов и X-сервера для аутентификации клиентских подключений используется механизм, основанный на предъявлении общего (известного обеим сторонам) «секрета», называемого «волшебной печенькой» (см. **W**:[magic cookie]), использование которых проиллюстрировано в примере из листинга 7.24.

На стороне сервера «печеньки» всех клиентов, которым разрешено подключение, размещаются при помощи утилиты **xauth(1)** в «банке с печеньками» (jar) ❶, откуда извлекаются сервером для проверки при подключении клиента. На стороне клиента «печеньки» при помощи той же утилиты **xauth(1)** размещаются ❷ в «банке»

~/.Xauthority, откуда извлекаются библиотекой **Xlib** для предъявления серверу при соединении с ним.

Листинг 7.24. Аутентификация дистанционного X-клиента

```
homer@ubuntu:~$ mcookie
8f36c904dc0c9934c506c21ea7860eb2
❶ homer@ubuntu:~$ xauth -f cookie-jar ☚
xauth:  file cookie-jar does not exist
Using authority file cookie-jar
xauth> add ubuntu:1 MIT-MAGIC-COOKIE-1 ☚ 8f36c904dc0c9934c506c21ea7860eb2 ↵
xauth> exit↵
Writing authority file cookie-jar
homer@ubuntu:~$ Xnest :1 -auth cookie-jar ☚ &
          ...          ...          ...          ...          ...
homer@ubuntu:~$ ssh lich@centos
```

lich@centos

```
Last login: Fri Jan  8 17:43:04 2016 from ubuntu
[lich@centos ~]$ xauth
❷ xauth:  file /home/lich/.Xauthority does not exist
Using authority file /home/lich/.Xauthority
xauth> add ubuntu.local:1 MIT-MAGIC-COOKIE-1 ☚ 8f36c904dc0c9934c506c21ea7860eb2 ↵
xauth> exit↵
Writing authority file /home/lich/.Xauthority
[lich@centos ~]$ logout
Connection to centos closed.
❸ homer@ubuntu:~$ ssh -f lich@centos "DISPLAY=ubuntu.local:1 xeyes"
```

Необходимость установки общего секрета и переменной окружения **DISPLAY** и необходимость запуска дополнительного виртуального X-сервера (или необходимость активации приема сетевых соединений аппаратным сервером) делают «ручной» запуск дистанционных X-клиентов неудобным «чуть более, чем полностью». Вместе с этим, передача «волшебных печенек» (как и любых других сообщений X-протокола) от X-клиента к X-серверу по сети происходит незащищенным образом, что может быть легко использовано злоумышленником. Именно поэтому на практике используют *туннелирующие* возможности протокола SSH, позволяющие удобным автоматизированным способом решить все вышеперечисленные задачи и проблемы.

В примере из листинга 7.25 показано поведение SSH-сервера при туннелировании X-протокола. По запросу (**-X**) от SSH-клиента SSH-сервер начинает эмулировать

❹ поведение X-сервера, устанавливает ❶ переменную окружения **DISPLAY**, указывающую на «дисплей» **:10** на «том же» узле **localhost**, и создает ❷ «волшебную печеньку» для этого дисплея.

При последующем запуске ❸ X-клиента **xeyes(1)** им будет установлено соединение с «SSH-эмулятором» X-сервера на **localhost:10**, а SSH-сервер перенаправит (туннелирует) это соединение X-протокола обратно SSH-клиенту внутри зашифрованного соединения SSH.

Листинг 7.25. SSH-туннелирование X-протокола (SSH-сервер)

```
homer@ubuntu:~$ ssh -X lich@centos
Last login: Fri Jan  8 17:48:34 2016 from ubuntu
[lich@centos ~]$ echo $DISPLAY
❶ localhost:10.0
[lich@centos ~]$ xauth list
❷ centos/unix:10  MIT-MAGIC-COOKIE-1  80c749073282be2001c33bd43e577aa4 ↩
❸ [lich@centos ~]$ strace -fe connect xeyes
          ...          ...          ...          ...          ...
connect(3, {sa_family=AF_INET, sin_port=htons(6010), sin_addr=inet_addr("127.0.0.1")}, 16) = 0
          ...          ...          ...          ...          ...
+++ exited with 0 +++
[lich@centos ~]$ sudo lsof -nP -i 4:6010
COMMAND   PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
❹ sshd    14221 lich   10u  IPv4  44578      0t0  TCP 127.0.0.1:6010 (LISTEN)
[lich@centos ~]$ logout
Connection to centos closed.
```

Листинг 7.26 иллюстрирует поведение SSH-клиента в режиме (**-X**) X-туннелирования, при котором он эмулирует «X-клиента» и соединяется с аппаратным (!) X-сервером через локальный сокет **/tmp/.X11-unix/X0**. При получении от SSH-сервера перенаправленных соединений X-протокола они ретранслируются SSH-эмулятором «X-клиента» локальному X-серверу, тем самым вывод *дистанционных* X-клиентов изображается в окнах *локального* X-сервера.

Листинг 7.26. SSH-туннелирование X-протокола (SSH-клиент)

```
homer@ubuntu:~$ strace -fe connect ssh -f -X lich@centos xeyes
          ...          ...          ...          ...          ...
connect(3, {sa_family=AF_INET, sin_port=htons(22), sin_addr=inet_addr("10.0.0.99")}, 16) = 0
          ...          ...          ...          ...          ...
connect(7, {sa_family=AF_FILE, path="/tmp/.X11-unix/X0"↩}, 110) = 0
```

Так как SSH-туннелирование X-протокола позволяет перенаправлять любое количество X-соединений внутри одного SSH-соединения, то дистанционно можно за-

пускать целые сеансы пользовательских сред, таких как GNOME (листинг 7.27), с использованием **xinit(1)** и виртуального X-сервера **Xnest(1)**.

Листинг 7.27. Запуск дистанционного сеанса GNOME с отображением на локальном сервере Xnest

```
homer@ubuntu:~$ xinit /usr/bin/ssh -X lich@centos gnome-session -- /usr/bin/Xnest :4 &
          ...          ...              ...              ...              ...
homer@ubuntu:~$ ps f
  PID TTY      STAT   TIME COMMAND
27759 pts/0    Ss     0:00 bash
27829 pts/0    S      0:00  \_ xinit /usr/bin/ssh -X lich@centos gnomes-session -- ...
27830 pts/0    Sl     0:00  |   \_ /usr/bin/Xnest :4
27837 pts/0    S      0:00  |   \_ /usr/bin/ssh -X lich@centos gnome-session
27928 pts/0    R+     0:00  \_ ps f
```

7.6.3. Управление X-дисплеями: XDMCP-менеджер и протокол

В большинстве современных инсталляций Linux работа пользователей в системе осуществляется сразу с использованием оконной системы X и какой-либо пользовательской среды, например GNOME.

Оконная система запускается сразу при старте операционной системы и не требует использования **xinit(1)** или **startx(1)**. Если при работе в автоматически запущенной оконной системе (листинг 7.28) проследить дерево процессов от командного интерпретатора до прародителя процессов **init(1)**, то обнаружится *менеджер дисплеев* ❶ (например, **lightdm(1)**), осуществляющий запуск пользовательского сеанса ❷ (**gnome-session(1)**). Более того, аппаратный X-сервер дисплея **:0** тоже окажется запущенным ❸ этим менеджером.

Листинг 7.28. Локальный графический вход

```
homer@ubuntu:~$ pstree -ap -s $$
init,1
  └─lightdm,32494
❶ ☛   └─lightdm,32749 --session-child 14 47
❷         └─gnome-session,329 --session=ubuntu
              └─compiz,404
                  └─sh,791 -c gnome-terminal
                      └─gnome-terminal,792
                          └─bash,1411
                              └─pstree,1485  -a -s 1411

homer@ubuntu:~$ ps f --pid 32494,32749 --ppid 32494,32749
  PID TTY      STAT   TIME COMMAND
32494 ?        Ssl    0:00 lightdm
```

```
❸ 32500 tty7     Ss+    0:10  \_ /usr/bin/X :0 -auth /var/run/lightdm/root/:0 ...
  32749 ?        Sl     0:00  \_ lightdm --session-child 14 23
    329 ?        Ssl    0:00      \_ gnome-session --session=ubuntu
```

Менеджер дисплеев является специальной компонентой оконной системы, управляющей автоматическим запуском ее X-серверов и X-сеансов. Именно он запускает аппаратные X-серверы для обслуживания дисплеев в указанном количестве (по умолчанию один дисплей **:0**), а затем «графическим» образом производит аутентификацию (*см. разд. 2.2.1*, рис. 2.4) пользователя в системе и запускает менеджер сеансов пользовательской среды.

Кроме этого, менеджер дисплеев предназначен и для запуска дистанционных пользовательских X-сеансов, но делает это при помощи специального протокола **W:[XDMCP]**, а не посредством протокола SSH, как при «ручном» дистанционном их запуске.

В листинге 7.29 и на рис. 7.8 показано дистанционное подключение виртуального X-сервера **Xnest(1)** с узла **terminal** к узлу **ubuntu.local** по протоколу XDMCP (**-query**), в результате чего менеджер дисплеев **lightdm(1)** (при помощи X-протокола) отобразил экран аутентификации пользователей в системе узла **ubuntu**. При успешной аутентификации пользователя будет запущен его X-сеанс на узле **ubuntu.local** так, что вывод всех X-клиентов будет направлен на дистанционный X-сервер узла **terminal**.

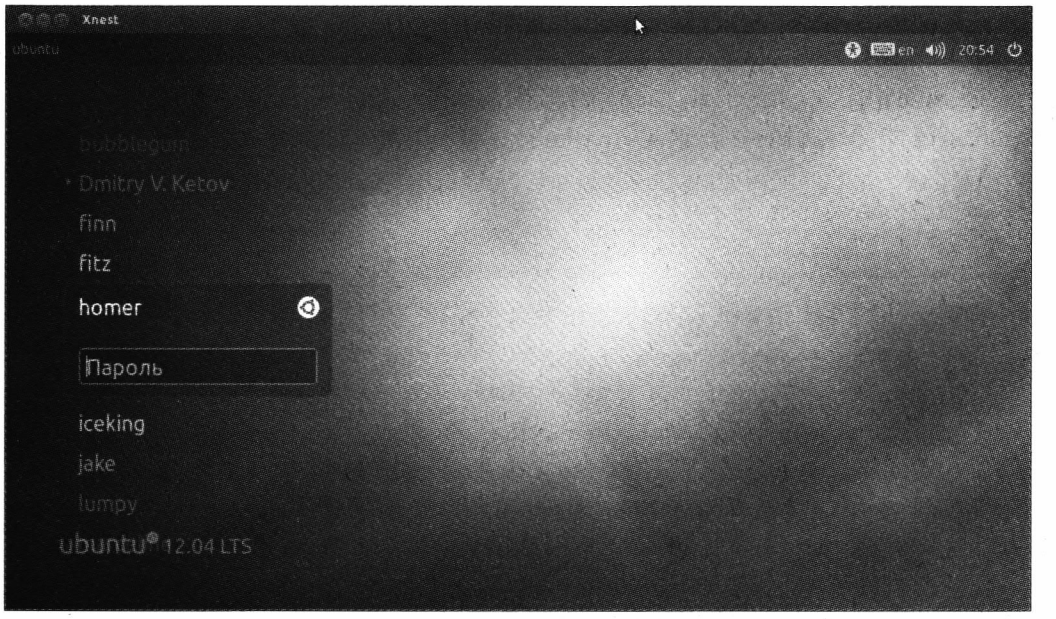


Рис. 7.8. Дистанционный графический вход

Листинг 7.29. Дистанционный графический вход в ubuntu

```
homer@ubuntu:~$ sudo lsof -p 32494 -a -i4
COMMAND   PID USER   FD   TYPE  DEVICE SIZE/OFF NODE NAME
lightdm 32494 root   11u  IPv4 3031884      0t0  UDP *:xdmcp

bart@terminal:~$ Xnest :1 -query ubuntu.local
```

Надо заметить, что при использовании дистанционного XDMCP-запуска данные XDMCP- и X-протоколов при передаче в публичной сети оказываются никак не защищены (включая согласование «волшебной печеньки» и ее последующее использование), поэтому на практике безальтернативно используют SSH-запуск дистанционных сеансов с туннелированием X-протокола внутри криптографически защищенного SSH-соединения.

7.7. Программный интерфейс X Window System

7.7.1. Трассировка X-библиотек и X-протокола

Программный интерфейс оконной системы X, являющейся обычной сетевой службой, представлен библиотеками и соответствующими (библиотечными) вызовами к ним. Естественным образом, при наблюдении за работой разнообразных компонент утилита **strace(1)** трассировки системных вызовов (к ядру) оказывается далеко не лучшим инструментом, а ее место занимает утилита трассировки библиотечных вызовов **ltrace(1)**.

В листинге 7.30 показан пример трассировки библиотечных вызовов библиотек **Xt**, **Xlib** и **Xrender** при работе простейшего демо-клиента **xeyes(1)**.

Листинг 7.30. Трассировка библиотек X Window System

```
homer@ubuntu:~$ ldd $(which xeyes) | grep libX
        libXext.so.6 => /usr/lib/i386-linux-gnu/libXext.so.6 (0xb772e000)
        libXmu.so.6 => /usr/lib/i386-linux-gnu/libXmu.so.6 (0xb7715000)
        libXt.so.6 => /usr/lib/i386-linux-gnu/libXt.so.6 (0xb76b8000)
        libX11.so.6 => /usr/lib/i386-linux-gnu/libX11.so.6 (0xb7584000)
        libXrender.so.1 => /usr/lib/i386-linux-gnu/libXrender.so.1 (0xb757a000)
        libXau.so.6 => /usr/lib/i386-linux-gnu/libXau.so.6 (0xb7350000)
        libXdmcp.so.6 => /usr/lib/i386-linux-gnu/libXdmcp.so.6 (0xb7349000) homer@ubuntu:~$
ltrace -n 2 -l /usr/lib/i386-linux-gnu/libXt.so.6 \↵
> -l /usr/lib/i386-linux-gnu/libX11.so.6 \↵
```

```
> -l /usr/lib/i386-linux-gnu/libXrender.so.1 xeyes
          ...          ...          ...          ...          ...
❶ XtAppAddTimeOut(0x93ec868, 50, 0x804a780, 0x9405ba0, 0xbfd378e0)       = 0x9405ea0
|  XtWindowOfObject(0x9405ba0, 50, 0x804a780, 0xb756ccab, 0x93eefdc)      = 0x600000c
|☛ XQueryPointer(0x93ed2f8, 0x600000c, 0xbfd37924, 0xbfd37928, 0xbfd3792c) = 1
↳  XtWidgetToApplicationContext(0x9405ba0, 0x40131942, ..., ..., 0xbfd378e0) = 0x93ec868
          ...          ...          ...          ...          ...
❷ XtAppAddTimeOut(0x8d84868, 50, 0x804a780, 0x8d9dba0, 0xbfa7c630)       = 0x8d9e0e8
|  XtWindowOfObject(0x8d9dba0, 50, 0x804a780, 0xb7553cab, 0x8d86fdc)      = 0x600000c
|  XQueryPointer(0x8d852f8, 0x600000c, 0xbfa7c674, 0xbfa7c678, 0xbfa7c67c) = 1
|☛ XFillRectangle(0x8d852f8, 0x600000c, 0x8d9de10, 44, 49)                = 1
|  XRenderFindStandardFormat(0x8d852f8, 2, 0x65cb1fc7, 0xbfc28ee3, 0)     = 0x8d9d660
|☛ XRenderCompositeDoublePoly(0x8d852f8, 3, 0x600000a, 0x600000f, 0x8d9d660) = 0
|  XFillRectangle(0x8d852f8, 0x600000c, 0x8d9de10, 122, 50)               = 1
|  XRenderFindStandardFormat(0x8d852f8, 2, 0x47cc50bf, 0xbfc542b9, 0)     = 0x8d9d660
|  XRenderCompositeDoublePoly(0x8d852f8, 3, 0x600000a, 0x600000f, 0x8d9d660) = 0
↳  XtWidgetToApplicationContext(0x8d9dba0, 0x4024e60f, ..., ..., 0xbfa7c630) = 0x8d84868
          ...          ...          ...          ...          ...
```

Зная назначение библиотечных вызовов **XQueryPointer(3)**, **XFillRecangle(3)** и **XRenderCompositeDoublePoly(3)**, можно составить модель функционирования X-клиента, в котором легко увидеть цикл ❶ опроса положения курсора при помощи **XQueryPointer(3)** и цикл ❷ перерисовки глаз стиранием их старого изображения при помощи **XFillRectangle(3)** и отрисовки нового положения посредством **XRenderCompositeDoublePoly(3)**.

Еще одним инструментом, разрешающим трассировать сообщения самого X-протокола, является **xtrace(1)**, позволяющая увидеть обмен между X-клиентом и X-сервером (листинг 7.31).

Листинг 7.31. Трассировка X-протокола

```
homer@ubuntu:~$ xtrace xeyes | grep -E 'QueryPointer|FillRectangle|RENDER'
          ...          ...          ...          ...          ...
000:<:0144:  8: Request(38): QueryPointer window=0x0600000c

000:>:0144:32: Reply to QueryPointer: same-screen=true(0x01) root=0x000000ae
child=None(0x00000000) root-x=957 root-y=40 win-x=23 win-y=-11 mask=0
          ...          ...          ...          ...          ...
000:<:00c5: 20: Request(70): PolyFillRectangle drawable=0x0600000c gc=0x06000007
rectangles={x=16 y=55 w=13 h=18};

000:<:00c6:304: RENDER-Request(149,10): Trapezoids op=Over(0x03) src=0x0600000a xSrc=0 ySrc=0
dst=0x0600000f maskFormat=0x00000024 trapezoids={top=54.831375 bottom=55.656631
          ...          ...          ...          ...          ...
left={x1=20.236710 y1=70.672775 x2=22.805756 y2=71.498032}; right={x1=25.374802 y1=70.672775
x2=22.805756 y2=71.498032};};
```

Кроме **xtrace(1)**, для анализа сообщений X-протокола при передаче по сети можно использовать анализатор сетевых пакетов **wrieshark(1)**, ровно таким же образом, как и для анализа сетевых сообщений любых других сетевых служб.

Большинство современных X-клиентов не прибегают напрямую к помощи низкоуровневых библиотек, таких как **Xlib** и **Xrender**, а используют высокоуровневые библиотеки виджетов, отрисовывающие их при помощи сообщений X-протокола. В примере из листинга 7.32 показана простейшая программа на языке программирования C, использующая библиотеку виджетов Gtk, а в листинге 7.33 приведены ее компиляция и трассировка ее библиотечных вызовов к **libgtk-x11-2.0.so.0**.

Листинг 7.32. Язык C и библиотека виджетов Gtk

```
homer@ubuntu:~$ cat hello.c
#include <gtk/gtk.h>

static void terminate(GtkWidget *widget, gpointer data) {
❾     gtk_main_quit();
}

int main(int argc, char *argv[]) {
    GtkWidget *window, *button, *label, *vbox;

❶     gtk_init (&argc, &argv);

❷     window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
❸     vbox = gtk_vbox_new(TRUE, 0);
③     gtk_container_add(GTK_CONTAINER (window), vbox);

❹     label = gtk_label_new("Hello, world!");
④     gtk_container_add(GTK_CONTAINER (vbox), label);

❺     button = gtk_button_new_with_label("Quit");
⑤     gtk_container_add(GTK_CONTAINER (vbox), button);

❻     g_signal_connect(button, "clicked", G_CALLBACK (terminate), NULL);
❼     gtk_widget_show_all (window);
❽     gtk_main ();

    return 0;
}
```

При работе программы сначала инициализируется библиотека виджетов ❶, которая инициирует соединение с X-сервером и аутентифицирует клиента при помощи «волшебной печеньки».

Интерфейс программы выстраивается из виджетов окна (window) ❷ и вертикального контейнера (vbox) ❸, который добавляется в это окно ③. В контейнер последовательно добавляются ④ и ⑤ виджеты текстовой метки (label) ❹ и кнопки (button) ❺, сигнал нажатия (**clicked**) на которую связывается ❻ с обработчиком — функцией **terminate**.

По запросу отображения окна ❼ библиотека прорисовывает виджеты помощи X-протокола, после чего запускается главный цикл ❽ обработки поступающих событий X-протокола (щелчки кнопкой мыши, нажатия клавиш и т. д.), который прекращается ❾ при срабатывании обработчика сигнала нажатия (**clicked**) на виджет кнопки.

Листинг 7.33. Трассировка библиотеки Gtk

```
homer@ubuntu:~$ gcc hello.c -o hello $(pkg-config --cflags --libs gtk+-2.0)
homer@ubuntu:~$ lddtree hello | grep libgtk
    libgtk-x11-2.0.so.0 => /usr/lib/i386-linux-gnu/libgtk-x11-2.0.so.0
homer@ubuntu:~$ ltrace -n2 -l /usr/lib/i386-linux-gnu/libgtk-x11-2.0.so.0 ./hello
❶ gtk_init(0xbfc98440, 0xbfc98444, 0x8049ff4, 0x80488b1, -1)      = 1
❷ gtk_window_new(0, 0xbfc98444, 0x8049ff4, 0x80488b1, -1)         = 0x8302000
❸ gtk_vbox_new(1, 0, 0x8049ff4, 0x80488b1, -1)                    = 0x82ab858
③ gtk_container_get_type(1, 0, 0x8049ff4, 0x80488b1, -1)          = 0x82e29b0
③ gtk_container_add(0x8302000, 0x82ab858, 0x8049ff4, 0x80488b1, -1) = 0
❹ gtk_label_new(0x8048960, 0x82ab858, 0x8049ff4, 0x80488b1, -1)   = 0xb5b06608
④ gtk_container_get_type(0x8048960, 0x82ab858, 0x8049ff4, 0x80488b1, -1) = 0x82e29b0
④ gtk_container_add(0x82ab858, 0xb5b06608, 0x8049ff4, 0x80488b1, -1) = 0
❺ gtk_button_new_with_label(0x804896e, 0xb5b06608, 0x8049ff4, 0x80488b1, -1) = 0x830a010
⑤ gtk_container_get_type(0x804896e, 0xb5b06608, 0x8049ff4, 0x80488b1, -1) = 0x82e29b0
⑤ gtk_container_add(0x82ab858, 0x830a010, 0x8049ff4, 0x80488b1, -1) = 0
❼ gtk_widget_show_all(0x8302000, 0x8048973, 0x8048764, 0, 0)      = 2
❽ gtk_main(0x8302000, 0x8048973, 0x8048764, 0, 0 <unfinished ...>
❾   gtk_main_quit(0x82ab858, 0xb73d0f30, 0xb72e5160, 0xb72e5243, 0x830a010) = 0
<... gtk_main resumed> )                                          = 0
+++ exited (status 0) +++
```

Аналогичным образом выглядят и работают программы, использующие Gtk и на других языках программирования, например на Python, что проиллюстрировано в листинге 7.34.

Листинг 7.34. Интерпретатор Python и библиотека интерфейсных элементов Gtk

```
homer@ubuntu:~$ cat gtk-hello.py
#!/usr/bin/python

import gtk

w = gtk.Window()
box = gtk.VBox()
w.add(box)

l = gtk.Label("Hello, world!")
box.add(l)

b = gtk.Button("Quit")
box.add(b)

def terminate(o):
  gtk.main_quit()

b.connect("clicked", terminate)
w.show_all()
gtk.main()
```

Каждая библиотека виджетов при ее использовании для построения пользовательского интерфейса обладает своей спецификой, но в целом имеет много общего с другими библиотеками интерфейсных элементов. Для сравнения, в листинге 7.35 приведена программа на языке Python, использующая библиотеку Qt, в которой можно найти много общего с программой из листинга 7.34, написанной на том же языке, но использующей библиотеку Gtk.

Листинг 7.35. Интерпретатор Python и библиотека интерфейсных элементов QtGui

```
homer@ubuntu:~$ cat qt-hello.py
#!/usr/bin/python

import sys
from PyQt4 import QtGui, Qt

a = QtGui.QApplication(sys.argv)

w = QtGui.QWidget()
box = QtGui.QVBoxLayout()
w.setLayout(box)
```

```
l = QtGui.QLabel("Hello, world!")
w.layout().addWidget(l)

b = QtGui.QPushButton("Quit")
w.layout().addWidget(b)

def terminate():
  a.quit()

w.show()
a.connect(b, Qt.SIGNAL("clicked()"), terminate)
a.exec_()
```

Библиотека виджетов **W:[Tk]** среди прочих других библиотек занимает, пожалуй, особенное место за счет языка **W:[Tcl]**, в качестве расширения которого специально и разрабатывалась. В отличие от языков C и Python, приведенных выше, и многих других «настоящих» систем программирования, Tcl (Tool Command Language) является инструментальным языком, родственным семейству языков командного интерпретатора **bash(1)**, **ksh(1)**, **csh(1)**, **zsh(1)** и пр.

Tcl-интерпретатор имеет «обычную» оболочку **tclsh(1)**, **Tcl sh**ell, и «оконную» оболочку **wish(1)**, **wi**ndowing **sh**ell, со встроенными командами к библиотеке виджетов Tk. Среди прочих вариантов X-клиента с использованием различных языков программирования и библиотек виджетов пример из листинга 7.36 имеет самый простой и очевидный синтаксис, похожий на синтаксис языка командного интерпретатора.

Листинг 7.36. Командный интерпретатор wish: язык Tcl и библиотека интерфейсных элементов Tk

```
homer@ubuntu:~$ cat hello.tcl
#!/usr/bin/wish

label .l -text "Hello, world!"
button .b -text Quit -command exit

pack .l .b
```

7.8. В заключение

Графический интерфейс, с которого начинается первое «визуальное» знакомство любого начинающего пользователя с современной операционной системой Linux, на поверку оказывается самой «сложной» ее подсистемой.

Оконная система X Window System использует в своей работе практически все[1] рассмотренные сущности операционной системы — программы и библиотеки, процессы и нити, драйверы устройств и их специальные файлы, локальные (файловые) сокеты, сетевую подсистему и сетевые сокеты, разделяемую память, отображение файлов в память и даже службу SSH. Не менее изощренными (а может быть, и более) в использовании услуг, предоставляемых приложениям разнообразными компонентами операционной системы, являются графические среды пользователей, такие как GNOME или KDE.

Таким образом, понимание происходящих в недрах Linux процессов, стоящих за графическим интерфейсом, пожалуй, сможет служить хорошей мерой проникновения в предмет внутреннего устройства этой замечательной операционной системы.

Надеюсь, что это понимание доставит вам такое же искреннее удовольствие, как и мне.

Д. К.

[1] Даже выход первой версии ядра Linux был практически «приурочен» к моменту успешного запуска и стабильной работы оконной системы X.

Заключение

Изначально содержимое этой книги задумывалось как практическая иллюстрация внутреннего устройства различных компонент операционной системы, в равной степени подкрепляемая примерами их анализа и синтеза.

Аналитическая часть удалась примерно в той степени, в которой и задумывалась, а примеры синтеза выходили достаточно блеклыми, особенно по сравнению с теми, которые уже представлены в соответствующих изданиях, целиком и полностью посвященных программному окружению и разработке. Более того, сделать эти примеры доступными для их успешного понимания читателем, не владеющим базовыми знаниями по программированию на языке C (Си), не удалось вовсе.

Поэтому все эти не столь многочисленные примеры перекочевали в приложение к книге, которое в виде архива доступно на сайте издательства **www.bhv.ru** со страницы книги или по ссылке:

ftp://ftp.bhv.ru/9785977535809.zip

Большинство примеров являются простейшими и достаточно грубыми моделями уже существующих в операционной системе программ.

Так, например, работу с файлами и ссылками иллюстрируют модели программ **touch(1)**, **ln(1)**, **rm(1)**, **mv(1)**, **cp(1)**, а работу со специальными файлами устройств — модель программы **eject(1)**, открывающая лоток привода CD/DVD, и специальная программа **mouse**, обрабатывающая перемещения манипулятора «мышь». Управление процессами продемонстрировано при помощи программы **shell**, моделирующей простейший командный интерпретатор, а управление нитями представляет программа **pdu** — параллельная модель измерителя файлов в дереве каталогов **du(1)**. Сетевое взаимодействие иллюстрируется простейшими TCP-клиентом, моделирующим **telnet(1)** (или **netcat(1)**), и TCP-сервером, моделирующим сервер службы удаленного доступа, похожей на **telnetd(8)** (или **sshd(8)**, только без криптозащиты).

Завершаются иллюстрации четырьмя вариантами простейшей GUI-программы **W:[Hello,_world!]**, выполненной на языках C, Python и Tcl и являющейся X-клиентом оконной системы X Window, использующим ее графические библиотеки интерфейсных элементов Qt, Gtk+ и Tk.

Кроме собственно иллюстрационных программ, архив содержит вспомогательные сценарии на языке командного интерпретатора, формирующие «программного гида», который поможет откомпилировать и запустить эти программы и вдобавок прокомментирует результат их выполнения в тех частях, где анализ результата не будет тривиальным. Сами сценарии «программного гида» тоже являются живыми примерами практик и приемов программирования на языке командного интерпретатора, анализируя которые можно расширить восприятие материала соответствующей главы, посвященной этой теме.

Как и ожидалось с самого начала, эта книга является лишь иллюстративным введением во внутреннее устройство ОС Linux, оставляющим за кадром массу специфичных деталей реализации конкретных версий ее ядер или ее дистрибутивов. Выстроив «правильную» начальную ментальную модель операционной системы, все эти тонкости можно без значительных усилий почерпнуть из собственной практики, документации, интернет-общения или других книг (см. список литературы).

Список литературы

Начинающим

Граннеман С. Linux. Карманный справочник. — М.: Вильямс, 2015. — 416 с. ISBN 978-5-8459-1956-4.

Пайк Р., Керниган Б. UNIX. Программное окружение. — СПб.: Символ-плюс, 2003. — 416 с. ISBN 5-93286-029-4.

Реймонд Э. Искусство программирования для UNIX. — М.: Вильямс, 2016. — 544 с. ISBN 978-5-8459-2064-5.

Тейнсли Д. Linux и UNIX: программирование в shell. Руководство разработчика. — СПб.: БХВ-Петербург, 2001. — 464 с. ISBN 5-7315-0114-9.

Фридл Дж. Регулярные выражения. — СПб.: Символ-плюс, 2008. — 608 с. ISBN 5-93286-121-5.

Dougherty D., Robbins A. sed & awk. — O'Reilly Media, 1997. — 432 p. ISBN 1-56592-225-5.

Программистам

Лав Р. Linux. Системное программирование. — СПб.: Питер, 2016. — 448 с. ISBN 978-5-496-01684-1.

Роббинс А. Linux: программирование в примерах. — СПб.: КУДИЦ-Пресс, 2008. — 656 с. ISBN 978-5-91136-056-6.

Стивенс У. Р., Рago С. UNIX. Профессиональное программирование. — СПб.: Символ-плюс, 2014. — 1104 с. ISBN 978-5-93286-216-2.

Kerrisk M. The Linux Programming Interface. — No Starch Press, 2010. — 1552 p. ISBN 978-1-59327-220-3.

Бесстрашным

Лав Р. Ядро Linux. Описание процесса разработки. — М.: Вильямс, 2014. — 496 с. ISBN 978-5-8459-1944-1.

Bovet D. P., Cesati M. Understanding the Linux Kernel. — O'Reilly Media, 2005. — 944 p. ISBN 978-0-596-00565-8.

Kroah-Hartman G. Linux Kernel in a Nutshell. — O'Reilly Media, 2006. — 202 p. ISBN 978-0-596-10079-7.

McKellar J., Rubini A., Corbet J., Kroah-Hartman G. Linux Device Drivers, 3rd Edition. — O'Reilly Media, 2005. — 640 p. ISBN 978-0-596-00590-0.

Предметный указатель

М

Н

О

П

Р

С

Т

У

Ф

Ц

Ш

Э

Я

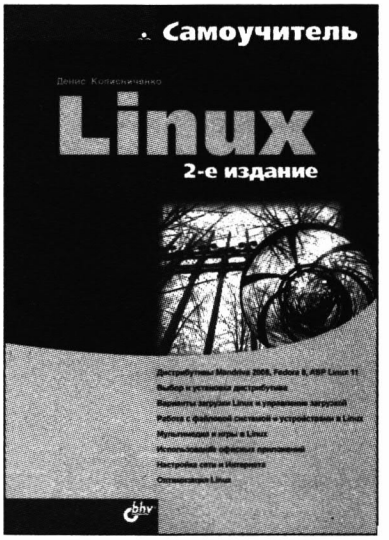
www.bhv.ru

Колисниченко Д.

Самоучитель Linux, 2-е издание

Отдел оптовых поставок:
e-mail: opt@bhv.spb.su

Ответы на актуальные вопросы по настройке и использованию операционной системы Linux



- Дистрибутивы Mandriva 2008, Fedora 8, ASP Linux 11
- Выбор и установка дистрибутива
- Варианты загрузки Linux и управление загрузкой
- Работа с файловой системой и устройствами в Linux
- Мультимедиа и игры в Linux
- Использование офисных приложений
- Настройка сети и Интернета
- Оптимизация Linux

Самоучитель ориентирован на домашнего и офисного непрофессионального пользователя и поможет ему самостоятельно настроить и оптимизировать операционную систему Linux. Материал ориентирован на современные версии дистрибутивов — Mandriva 2008, Fedora 8, ASP Linux 11. В книге есть ответы практически на все вопросы, возникающие при ежедневной работе в Linux. Рассмотрены типичные ситуации: вход в систему, работа с файловой системой, использование графического интерфейса, установка программного обеспечения, настройка сети и Интернета, работа в Интернете, использование офисных, мультимедийных и игровых приложений, работа с периферийными устройствами, а также оптимизация операционной системы Linux.

Колисниченко Денис Николаевич, инженер-программист и системный администратор. Имеет богатый опыт эксплуатации и создания локальных сетей от домашних до уровня предприятия на базе операционной системы Linux, автор более 10 книг компьютерной тематики, в том числе «Самоучитель Linux», «99 советов по Linux», «Ubuntu Linux. Краткое руководство пользователя», «Серверное применение Linux» и др.

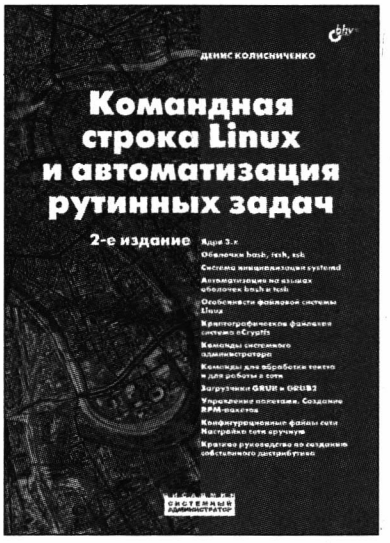
www.bhv.ru

Колисниченко Д.

Командная строка Linux и автоматизация рутинных задач, 2-е изд.

Отдел оптовых поставок:
e-mail: opt@bhv.spb.su

Эффективная работа в командной строке Linux



- Ядро 3.x
- Оболочки bash, tcsh, zsh
- Система инициализации systemd
- Автоматизация на языках оболочек bash и tcsh
- Особенности файловой системы Linux
- Криптографическая файловая система eCryptfs
- Команды системного администратора
- Команды для обработки текста и для работы в сети
- Загрузчики GRUB и GRUB2
- Управление пакетами. Создание RPM-пакетов
- Конфигурационные файлы сети. Настройка сети вручную
- Краткое руководство по созданию собственного дистрибутива

Рассмотрены команды Linux, основы работы в командной строке, а также настройка системы с помощью программ, обладающих только текстовым интерфейсом. Работа с системой выполняется только в режиме консоли, что требует определенной квалификации пользователя. Подробно описаны наиболее полезные команды Linux, особенности файловой системы Linux (в том числе криптографическая файловая система eCryptfs), системы инициализации (в том числе systemd), загрузчики GRUB и GRUB2. С позиции пользователя оценены интерактивные возможности оболочки zsh. Даны практические примеры разработки сценариев на языках оболочек bash и tcsh. Рассмотрено управление пакетами для наиболее актуальных на данный момент дистрибутивов. Описаны конфигурационные файлы сети, даны рекомендации по настройке сети вручную без использования конфигураторов. Для энтузиастов Linux написана отдельная глава о разработке собственного дистрибутива Linux и создании загрузочного LiveCD. Во втором издании появилось описание ряда полезных утилит (dd, sed и др.), псевдофайловой системы /proc, процесса создания RPM-пакетов.

Колисниченко Денис Николаевич, инженер-программист и системный администратор. Имеет богатый опыт работы в операционной системе Linux. Автор более 40 книг компьютерной тематики, в том числе «Linux. От новичка к профессионалу», «Linux на ноутбуке», «Самоучитель Linux openSUSE 11.2», «Серверное применение Linux», «Самоучитель системного администратора Linux» и др.

www.bhv.ru

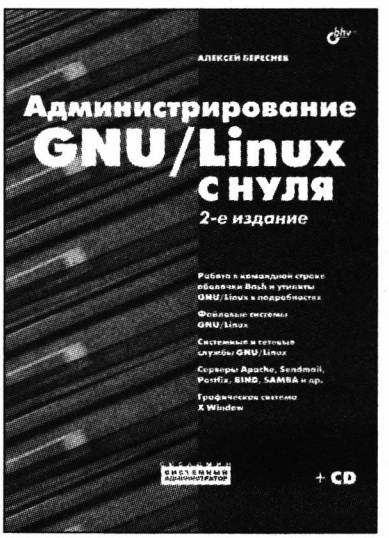
Береснев А.
Администрирование GNU/Linux с нуля, 2-е издание

Отдел оптовых поставок:
e-mail: opt@bhv.spb.su

Гарантия быстрого и эффективного освоения администрирования GNU/Linux

Книга дает ясное, современное и практичное понимание GNU/Linux и программного обеспечения GNU, помогает сделать быстрый старт для начинающих и систематизировать свои знания для профессионалов. Отлично подобранные примеры позволяют эффективно подготовиться к сдаче международного сертификационного экзамена Linux Professional Institute.

Евгений Сафонов, Certified Novell Engineer



- Работа в командной строке оболочки Bash и утилиты GNU/Linux в подробностях
- Файловые системы GNU/Linux
- Системные и сетевые службы GNU/Linux
- Серверы Apache, Sendmail, Postfix, BIND, SAMBA и др.
- Графическая система X Window

Интерфейс командной строки, традиционно являющийся основным путем администрирования UNIX и GNU/Linux, с непривычки может напугать. Десятки служб и сотни команд, предоставляемые GNU/Linux, впечатляют начинающих администраторов. Автор прилагает усилия для наиболее комфортного знакомства новичка с GNU/Linux и продвижения его к вершинам профессионализма. Материал подобран таким образом, чтобы читатель овладел фундаментальными знаниями основ GNU/Linux, а затем постепенно получал все более сложные знания в области администрирования. При этом материал не привязан к какому-либо одному дистрибутиву GNU/Linux. Приведенные в книге примеры настройки системных служб и сетевых сервисов просты, но показательны. Рассмотрены не только аспекты ежедневного администрирования, как, например, управления пользователями и обслуживания файловых систем, но и настройка таких популярных служб, как BIND, Apache и SAMBA. Помимо прочего, книга позволяет подготовиться к сдаче сертификационных экзаменов LPI-101 и LPI-102.

Береснев Алексей Лингардович, автор учебных курсов по GNU/Linux, FreeBSD и SUN Solaris, сертифицированный преподаватель CISCO (CCSI), имеет сертификаты Linux Professional Institute LPI-I и LPI-II и многолетний опыт эксплуатации UNIX и GNU/Linux. В настоящее время работает коммерческим директором НОУ «ЦИФТ» — старейшего учебного центра по информационным технологиям на Урале.

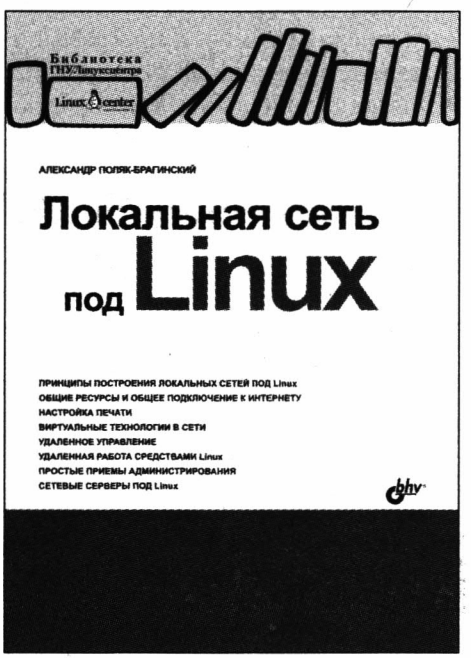
www.bhv.ru

Поляк-Брагинский А.

Локальная сеть под Linux

Отдел оптовых поставок:
e-mail: opt@bhv.spb.su

Локальная сеть дома и в офисе простыми средствами



- Принципы построения локальных сетей под Linux
- Общие ресурсы и общее подключение к Интернету
- Настройка печати
- Виртуальные технологии в сети
- Удаленное управление
- Удаленная работа средствами Linux
- Простые приемы администрирования
- Сетевые серверы под Linux

Для тех, кто решил построить простую локальную сеть под управлением Linux у себя дома или в офисе, эта книга станет незаменимым помощником. Рассмотрены вопросы создания и обслуживания небольшой локальной сети под управлением Linux. Даны практические приемы применения виртуальных технологий, позволяющих удешевить сеть и получить максимум функциональности при минимальных затратах, опробовать различные версии операционной системы Linux в сети и использовать выбранный вариант без затрат на оборудование. Описаны простые средства администрирования сети, средства операционной системы Linux для удаленного доступа к компьютерам сети. Для комфортной работы с книгой и исключения необходимости поиска дополнительной литературы приведены описания самых необходимых процедур и команд Linux.

Поляк-Брагинский Александр Владимирович, руководитель проекта по автоматизации на транспорте в успешной коммерческой компании, специалист в области организации и эксплуатации малых и средних локальных сетей. Автор книг «Сеть своими руками», «Сеть под Microsoft Windows. Экспресс-курс», «Администрирование сети на примерах», «Локальные сети. Модернизация и поиск неисправностей», «Локальная сеть дома и в офисе. Народные советы», «Первые шаги с Windows Vista. Руководство для начинающих», «Linux и Windows в домашней сети», «Локальная сеть. Самое необходимое».

ВНУТРЕННЕЕ УСТРОЙСТВО LINUX

- Пользовательское окружение и интерфейс командной строки CLI
- Файлы, каталоги и файловые системы
- Дискреционное, мандатное разграничение доступа и привилегии
- Процессы и нити
- Виртуальная память и отображаемые файлы
- Каналы, сокеты и разделяемая память
- Сетевая подсистема и служба SSH
- Оконная система X Window и графический интерфейс GUI
- Программирование на языке командного интерпретатора

Книга, которую вы держите в руках, адресована студентам, начинающим пользователям, программистам и системным администраторам операционной системы Linux. Она представляет собой введение во внутреннее устройство Linux — от ядра до сетевых служб и от утилит командной строки до графического интерфейса.

Все части операционной системы рассматриваются в контексте типичных задач, решаемых на практике, и поясняются при помощи соответствующего инструментария пользователя, администратора и разработчика.

Все положения наглядно проиллюстрированы примерами, разработанными и проверенными автором с целью привить читателю навыки самостоятельного исследования постоянно эволюционирующей операционной системы Linux.

Кетов Дмитрий Владимирович, преподаватель в области операционных систем и сетевых технологий Санкт-Петербургского политехнического университета с многолетним стажем, инженер в Санкт-Петербургском исследовательском центре LG Russia R&D Lab. Профессионально занимается теорией построения и практикой разработки операционных систем и системного программного обеспечения.

ISBN 978-5-9775-3580-9

БХВ-ПЕТЕРБУРГ
191036, Санкт-Петербург,
Гончарная ул., 20
Тел.: (812) 717-10-50,
339-54-17, 339-54-28
E-mail: mail@bhv.ru
Internet: www.bhv.ru