1 Общая картина

На первый взгляд современная операционная система, например Linux, является достаточно сложной и состоит из большого количества частей, которые одновременно функционируют и взаимодействуют друг с другом. Так, веб-сервер может обмениваться данными с сервером базы данных, который, в свою очередь, использует совместную библиотеку, применяемую многими другими программами. Как же все это работает?

Наиболее эффективно понять устройство операционной системы можно с помощью *абстракции* — изящного способа сказать о том, что вы игнорируете большинство деталей. Например, когда вы едете в автомобиле, вам, как правило, не приходится задумываться о таких деталях, как крепежные болты, которые удерживают двигатель внутри машины, или же о людях, проложивших дорогу и поддерживающих ее в хорошем состоянии. Если вы едете в машине как пассажир, вам нужно знать лишь то, для чего предназначен автомобиль (он перемещает вас куда-либо), а также некоторые элементарные правила его использования (как обращаться с дверью и ремнем безопасности).

Если же вы ведете машину, вам необходимо знать больше. Вам потребуется изучить элементы управления (например, рулевое колесо и педаль газа), а также усвоить, что следует делать в случае неисправности.

Предположим, автомобиль движется рывками. Можно разбить абстракцию «автомобиль, который едет по дороге» на три части: автомобиль, дорога и ваш стиль вождения. Это поможет установить причину. Если дорога ухабиста, вам не придется винить машину или себя. Вместо этого вы можете попытаться выяснить, почему дорога испортилась, или же, если дорога новая, почему ее строители так отвратительно выполнили работу.

Разработчики программного обеспечения пользуются абстракцией как инструментом при создании операционных систем и приложений. Имеется множество терминов для абстрагированных разделов компьютерного ПО, в их число входят *подсистема*, *модуль* и *пакет*. Однако мы будем применять в данной главе термин *компонент*, поскольку он прост. При создании программного компонента, как правило, разработчиков не сильно заботит внутренняя структура других компонентов, однако им все же приходится думать о том, какие компоненты и как они могут использовать.

В этой главе приводится общий обзор компонентов, составляющих систему Linux. Хотя каждый из них обладает немыслимым количеством технических

деталей, относящихся к внутреннему устройству, мы не будем обращать на них внимания и сосредоточимся на том, что эти компоненты делают по отношению к системе в целом.

1.1. Уровни и слои абстракции в операционной системе Linux

Использование абстракций для разделения компьютерных систем на компоненты упрощает их понимание, но не приносит пользы, если отсутствует структура. Мы упорядочим компоненты в виде слоев, или уровней. *Слой*, или *уровень*, — это способ классификации (или группирования) компонентов в соответствии с их расположением между пользователем и аппаратными средствами. Браузеры, игры и т. п. расположены на верхнем слое; на нижнем слое мы видим память компьютера: нули и единицы. Операционная система занимает наибольшее число слоев между этими двумя.

В операционной системе Linux три главных уровня. На рис. 1.1 показаны уровни, а также некоторые компоненты внутри каждого из них. В основе расположены *аппаратные средства*. Они включают память, а также один или несколько центральных процессоров (CPU), выполняющих вычисления и запросы на чтение из памяти и запись в нее. Такие устройства, как жесткие диски и сетевые интерфейсы, также относятся к аппаратным средствам.

Уровнем выше располагается *ядро*, которое является сердцевиной операционной системы. Ядро — это программа, расположенная в памяти компьютера и отдающая распоряжения центральному процессору. Ядро управляет аппаратными средствами и выступает главным образом в качестве интерфейса между аппаратными средствами и любой запущенной программой.

Процессы — запущенные программы, которыми управляет ядро, — в совокупности составляют верхний уровень системы, именующийся *пространством пользователя*.

ПРИМЕЧАНИЕ

Более точным термином, чем «процесс», является термин «пользовательский процесс», вне зависимости от того, взаимодействует ли пользователь с этим процессом напрямую. Например, все вебсерверы работают как пользовательские процессы.

Существует важное различие между тем, как запускаются процессы ядра и процессы пользователя: ядро запускается в *режиме ядра*, а пользовательские процессы — в *режиме пользователя*. Код, работающий в режиме ядра, обладает неограниченным доступом к процессору и оперативной памяти. Это сильное преимущество, но оно может быть опасным, поскольку позволяет процессам ядра с легкостью нарушить работу всей системы. Область, которая доступна только ядру, называется *пространством ядра*.

В режиме пользователя, для сравнения, доступен лишь ограниченный (как правило, небольшой) объем памяти и разрешены лишь безопасные инструкции для процессора. *Пространством пользователя* называют участки оперативной памяти, которые могут быть доступны пользовательским процессам. Если какой-либо процесс завершается с ошибкой, ее последствия будут ограниченными и ядро сможет

их очистить. Это означает, что, если, например, произойдет сбой в работе браузера, выполнение научных расчетов, которые вы запустили на несколько дней в фоновом режиме, не будет нарушено.

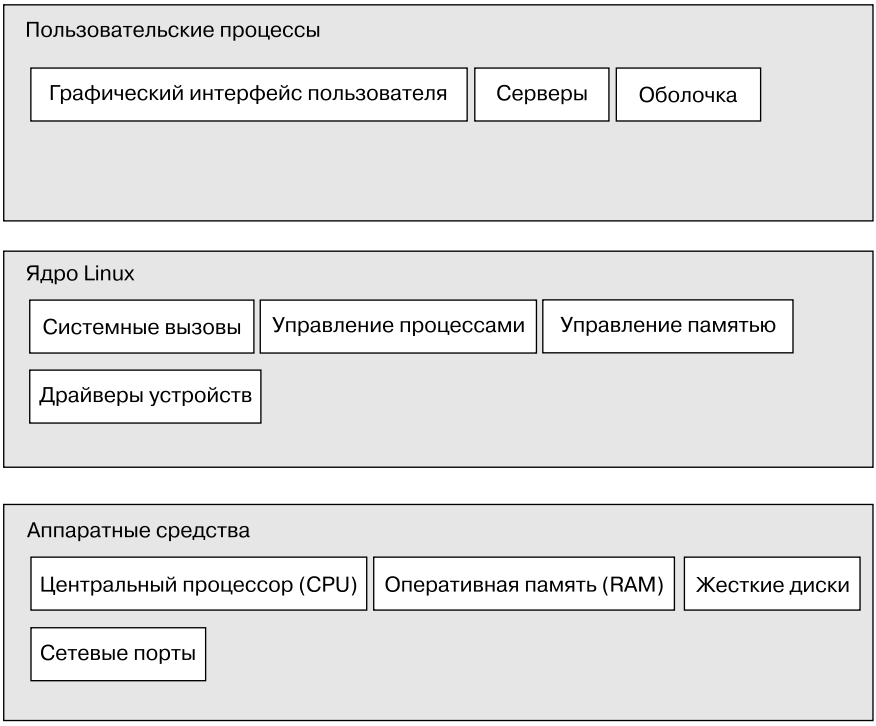

Рис. 1.1. Общая структура операционной системы Linux

Теоретически неконтролируемый пользовательский процесс не способен причинить существенный вред системе. В действительности же все зависит от того, что именно вы считаете «существенным вредом», а также от особых привилегий данного процесса, поскольку некоторым процессам разрешено делать больше, чем другим. Например, может ли пользовательский процесс полностью уничтожить данные на жестком диске? Если должным образом настроить разрешения, то сможет, и для вас это окажется крайне опасным. Для предотвращения этого существуют защитные меры, и большинству процессов просто не будет позволено сеять смуту подобным образом.

1.2. Аппаратные средства: оперативная память

Из всех аппаратных средств компьютера *оперативная память* является, пожалуй, наиболее важным. В своей самой «сырой» форме оперативная память — это всего лишь огромное хранилище для последовательности нулей и единиц. Каждый ноль или единица называется *битом*. Именно здесь располагаются запущенное ядро

и процессы — они являются лишь большими наборами битов. Все входные и выходные данные от периферийных устройств проходят через оперативную память также в виде наборов битов. Центральный процессор просто оперирует с памятью: он считывает из нее инструкции и данные, а затем записывает данные обратно в память.

Вам часто будет встречаться термин *«состояние»*, который будет относиться к памяти, процессам, ядру и другим частям компьютерной системы. Строго говоря, состояние — это какое-либо упорядоченное расположение битов. Например, если в памяти находятся четыре бита, то последовательности 0110, 0001 и 1011 представляют три различных состояния.

Если принять во внимание то, что процесс может с легкостью состоять из миллионов бит в памяти, зачастую проще использовать абстрактные термины, говоря о состояниях. Вместо описания состояния с применением битов вы говорите о том, что произошло или происходит в данный момент. Например, вы можете сказать «данный процесс ожидает входных данных» или «процесс выполняет второй этап процедуры запуска».

ПРИМЕЧАНИЕ

Поскольку при описании состояния обычно используются абстрактные понятия, а не реальные биты, для обозначения какого-либо физического размещения битов применяется термин «образ».

1.3. Ядро

Практически все, что выполняет ядро, касается оперативной памяти. Одной из задач ядра является распределение памяти на несколько подразделов, после чего ядро должно постоянно содержать в порядке информацию о состоянии этих подразделов. Каждый процесс использует выделенную для него область памяти, и ядро должно гарантировать то, что процессы придерживаются своих областей.

Ядро отвечает за управление задачами в четырех основных областях системы.

- **Процессы.** Ядро отвечает за то, каким процессам разрешен доступ к центральному процессору.
- **Память.** Ядру необходимо отслеживать состояние всей памяти: какая часть в данный момент отведена под определенные процессы, что можно выделить для совместного использования процессами и какая часть свободна.
- **Драйверы устройств.** Ядро выступает в качестве интерфейса между аппаратными средствами (например, жестким диском) и процессами. Как правило, управление аппаратными средствами выполняется ядром.
- **Системные вызовы и поддержка.** Обычно процессы используют системные вызовы для взаимодействия с ядром.

Теперь мы вкратце рассмотрим каждую из этих областей.

ПРИМЕЧАНИЕ

Подробности о работе ядра вы можете узнать из книг Operating System Concepts («Основные принципы операционных систем»), 9-е издание, авторы: Авраам Зильбершатц (Abraham Silberschatz), Питер Б. Гелвин (Peter B. Galvin) и Грег Гэнн (Greg Gagne) (Wiley, 2012) и Modern Operating Systems («Современные операционные системы»), 4-е издание, авторы: Эндрю С. Таненбаум (Andrew S. Tanenbaum) и Герберт Бос (Herbert Bos) (Prentice Hall, 2014).

1.3.1. Управление процессами

Управление процессами описывает запуск, остановку, возобновление и прекращение работы процессов. Понятия, которые стоят за процессами запуска и прекращения процессов, достаточно просты. Немного сложнее описать то, каким образом процесс использует центральный процессор в нормальном режиме работы.

В любой современной операционной системе несколько процессов функционируют «одновременно». Например, в одно и то же время вы можете запустить на компьютере браузер и открыть электронную таблицу. Тем не менее на самом деле все обстоит не так, как выглядит: процессы, которые отвечают за эти приложения, как правило, не запускаются *в точности* в один момент времени.

Рассмотрим систему с одним центральным процессором. Его могут использовать несколько процессов, но в каждый конкретный момент времени только один процесс может в действительности применять процессор. На практике каждый процесс использует процессор в течение малой доли секунды, а затем приостанавливается; после этого другой процесс применяет процессор в течение малой доли секунды; далее наступает черед третьего процесса и т. д. Действие, при котором какой-либо процесс передает другому процессу управление процессором, называется *переключением контекста*.

Каждый отрезок времени — *квант времени* — предоставляет процессу достаточно времени для выполнения существенных вычислений (и, конечно же, процесс часто завершает свою текущую задачу в течение одного кванта). Поскольку кванты времени настолько малы, человек их не воспринимает и ему кажется, что в системе одновременно выполняется несколько процессов (такая возможность известна под названием *«многозадачность»*).

Ядро отвечает за переключение контекста. Чтобы понять, как это работает, представим ситуацию, в которой процесс запущен в режиме пользователя, но его квант времени заканчивается. Вот что при этом происходит.

1. Процессор (реальное аппаратное средство) прерывает текущий процесс, опираясь на внутренний таймер, переключается в режим ядра и возвращает ему управление.
2. Ядро записывает текущее состояние процессора и памяти, которые будут необходимы для возобновления только что прерванного процесса.
3. Ядро выполняет любые задачи, которые могли появиться в течение предыдущего кванта времени (например, сбор данных или операции ввода/вывода).
4. Теперь ядро готово к запуску другого процесса. Оно анализирует список процессов, готовых к запуску, и выбирает какой-либо из них.
5. Ядро готовит память для нового процесса, а затем подготавливает процессор.
6. Ядро сообщает процессору, сколько будет длиться квант времени для нового процесса.
7. Ядро переводит процессор в режим пользователя и передает процессору управление.

Переключение контекста дает ответ на важный вопрос: *когда* работает ядро? Ответ следующий: ядро работает *между* отведенными для процессов квантами времени, когда происходит переключение контекста.

В системе с несколькими процессорами дело обстоит немного сложнее, поскольку ядру нет необходимости прекращать управление текущим процессором, чтобы позволить запуск какого-либо процесса на другом процессоре. И тем не менее, чтобы извлечь максимальную пользу из всех доступных процессоров, ядро все же так поступает (и может применить определенные хитрости, чтобы получить дополнительное процессорное время).

1.3.2. Управление памятью

Поскольку ядро должно управлять памятью во время переключения контекста, оно наделено этой сложной функцией. Работа ядра сложна, поскольку необходимо учитывать следующие условия:

- ядро должно располагать собственной областью памяти, к которой не могут получить доступ пользовательские процессы;
- каждому пользовательскому процессу необходима своя область памяти;
- какой-либо пользовательский процесс не должен иметь доступ к области памяти, предназначенной для другого процесса;
- пользовательские процессы могут совместно использовать память;
- некоторые участки памяти для пользовательских процессов могут быть предназначены только для чтения;
- система может применять больше памяти, чем ее есть в наличии, задействовав в качестве вспомогательного устройства дисковое пространство.

У ядра есть помощник. Современные процессоры содержат *модуль управления памятью* (MMU), который активизирует схему доступа к памяти под названием *«виртуальная память»*. При использовании виртуальной памяти процесс не обращается к памяти напрямую по ее физическому расположению в аппаратных средствах. Вместо этого ядро настраивает каждый процесс таким образом, словно в его распоряжении находится вся машина. Когда процесс получает доступ к памяти, модуль MMU перехватывает такой запрос и применяет карту адресов памяти, чтобы перевести местоположение памяти, полученное от процесса, в физическое положение памяти на компьютере. Однако ядро все же должно инициализировать, постоянно поддерживать и изменять эту карту адресов. Например, во время переключения контекста ядро должно изменить карту после отработавшего процесса и подготовить его для наступающего.

ПРИМЕЧАНИЕ

Реализация карты адресов памяти называется таблицей страниц.

О том, как отслеживать производительность памяти, вы узнаете из главы 8.

1.3.3. Драйверы устройств и управление ими

Задача ядра по отношению к устройствам довольно проста. Как правило, устройства доступны только в режиме ядра, поскольку некорректный доступ (например, когда пользовательский процесс пытается выключить питание) может вызвать от-

каз в работе компьютера. Еще одна проблема заключается в том, что различные устройства редко обладают одинаковым программным интерфейсом, даже если они выполняют одинаковую задачу: например, две различные сетевые карты. По этой причине драйверы устройств традиционно являются частью ядра и стремятся предоставить унифицированный интерфейс для пользовательских процессов, чтобы облегчить труд разработчиков программного обеспечения.

1.3.4. Системные вызовы и поддержка

Существуют и другие типы функций ядра, доступные для пользовательских процессов. Например, *системные вызовы* выполняют специальные задачи, которые пользовательский процесс не может выполнить хорошо в одиночку или вообще не может справиться с ними. Так, все действия, связанные с открытием, чтением и записью файлов, вовлекают системные вызовы.

Два системных вызова — `fork()` и `exec()` — важны для понимания того, как происходит запуск процессов:

- `fork()`. Когда процесс осуществляет вызов `fork()`, ядро создает практически идентичную копию данного процесса;
- `exec()`. Когда процесс осуществляет вызов `exec(`*`program`*`)`, ядро запускает программу *program*, которая замещает текущий процесс.

За исключением процесса `init` (глава 6), *все* пользовательские процессы в системе Linux начинаются как результат вызова `fork()`, и в большинстве случаев осуществляется вызов `exec()`, чтобы запустить новую программу, а не копию существующего процесса. Простым примером является любая программа, которую вы запускаете из командной строки, например команда `ls`, показывающая содержимое каталога. Когда вы вводите команду `ls` в окне терминала, запущенная внутри окна терминала оболочка осуществляет вызов `fork()`, чтобы создать копию оболочки, а затем эта новая копия оболочки выполняет вызов `exec(ls)`, чтобы запустить команду `ls`. На рис. 1.2 показана последовательность процессов и системных вызовов для запуска таких программ, как `ls`.

Рис. 1.2. Запуск нового процесса

ПРИМЕЧАНИЕ

Системные вызовы обычно обозначаются с помощью круглых скобок. В примере, показанном на рис. 1.2, процесс, который запрашивает ядро о создании другого процесса, должен осуществить системный вызов fork(). Такое обозначение происходит от способа написания вызовов в языке программирования C. Чтобы понять эту книгу, вам не обязательно знать язык C. Помните лишь о том, что системный вызов — это взаимодействие между процессом и ядром. Более того, в этой книге упрощены некоторые группы системных вызовов. Например, вызов exec() обозначает целое семейство системных вызовов, выполняющих сходную задачу, но отличающихся программной реализацией.

Ядро также поддерживает пользовательские процессы, функции которых отличаются от традиционных системных вызовов. Самыми известными из них являются *псевдоустройства*. С точки зрения пользовательских процессов, псевдоустройства выглядят как обычные устройства, но реализованы они исключительно программным образом. По сути, формально они не должны находиться в ядре, но они все же присутствуют в нем из практических соображений. Например, устройство, которое генерирует случайные числа (`/dev/random`), было бы сложно реализовать с необходимой степенью безопасности с помощью пользовательского процесса.

ПРИМЕЧАНИЕ

Технически пользовательский процесс, который получает доступ к псевдоустройству, все же вынужден осуществлять системный вызов для открытия этого устройства. Таким образом, процессы не могут полностью обойтись без системных вызовов.

1.4. Пространство пользователя

Область оперативной памяти, которую ядро отводит для пользовательских процессов, называется *пространством пользователя*. Поскольку процесс является лишь состоянием (или образом) в памяти, пространство пользователя обращается также к памяти за всей совокупностью запущенных процессов. Вам также может встретиться термин *«участок пользователя» (userland)*, который применяется вместо пространства пользователя.

Большинство реальных действий системы Linux происходит в пространстве пользователя. Несмотря на то что все процессы с точки зрения ядра являются одинаковыми, они выполняют различные задачи для пользователей. Системные компоненты, которые представляют пользовательские процессы, организованы в виде элементарной структуры — сервисного уровня (или слоя). На рис. 1.3 показан примерный набор компонентов, связанных между собой и взаимодействующих с системой Linux. Простые службы расположены на нижнем уровне (ближе всего к ядру), сервисные программы находятся в середине, а приложения, с которыми работает пользователь, расположены вверху. Рисунок 1.3 является крайне упрощенной схемой, поскольку показаны только шесть компонентов, но вы можете заметить, что верхние компоненты находятся ближе всего к пользователю (пользовательский интерфейс и браузер); компоненты среднего уровня располагают почтовым сервером, который использует браузер; в нижней части присутствует несколько малых компонентов.

Нижний уровень состоит, как правило, из малых компонентов, выполняющих простые задачи. Средний уровень содержит более крупные компоненты, такие как почтовая служба, сервер печати и база данных. Компоненты верхнего уровня выполняют сложные задачи, которые зачастую непосредственно контролирует пользователь. Если один компонент желает воспользоваться другим, то этот второй компонент находится либо на том же сервисном уровне, либо ниже.

Рисунок 1.3 только приблизительно отображает устройство пространства пользователя. В действительности в пространстве пользователя нет правил. Например, большинство приложений и служб записывают диагностические сообщения, ко-

торые называются *журналами*. Большинство программ использует стандартную службу `syslog` для записи сообщений в журнал, но некоторые предпочитают вести журнал самостоятельно.

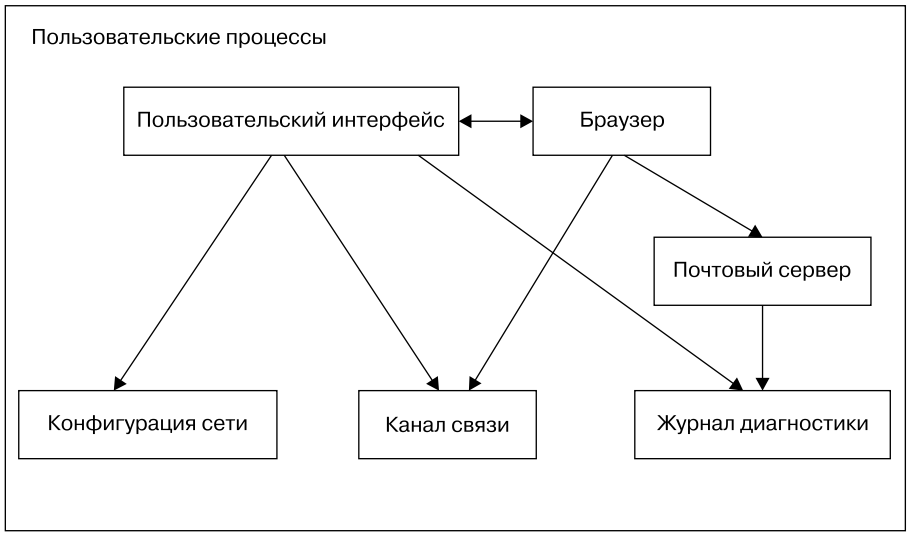

Рис. 1.3. Типы процессов и взаимодействий

Кроме того, некоторые компоненты пространства пользователя бывает трудно отнести к какой-либо категории. Серверные компоненты, например веб-сервер или сервер базы данных, можно рассматривать как приложения очень высокого уровня, поскольку они выполняют довольно сложные задачи. Такие приложения можно поместить в верхней части рис. 1.3. В то же время пользовательские приложения могут зависеть от серверных, когда необходимо выполнять задачи, с которыми они не могут справиться самостоятельно. В таком случае серверные компоненты следовало бы поместить на средний уровень.

1.5. Пользователи

Ядро системы Linux поддерживает традиционную концепцию пользователя системы Unix. *Пользователь* — это сущность, которая может запускать процессы и обладать файлами. С пользователем связано *имя пользователя*. Например, в системе может быть пользователь `billyjoe`. Однако ядро не работает с именами пользователей, вместо этого оно идентифицирует пользователя с помощью простого числового *идентификатора пользователя* (в главе 7 рассказывается о том, как идентификаторы сопоставляются с именами пользователей).

Пользователи существуют главным образом для того, чтобы соблюдались права доступа и ограничения. У каждого процесса из пространства пользователя существует пользователь-*владелец*, а о процессах говорят, что они запущены

в качестве владельцев. Пользователь может прервать или изменить ход принадлежащих ему процессов (в определенных пределах), но не может вмешаться в процессы других пользователей. Кроме того, пользователи могут обладать файлами и предоставлять совместный доступ к ним для других пользователей.

В системе Linux обычно присутствуют дополнительные пользователи помимо тех, которые соответствуют реальным людям, работающим в системе. Более подробно об этом рассказывается в главе 3, но самым важным пользователем является `root`. Этот пользователь — исключение из приведенных выше правил, поскольку он может прерывать и изменять ход процессов другого пользователя, а также выполнять чтение любого локального файла. По этой причине пользователь `root` известен как `superuser`. О человеке, который может работать как пользователь `root`, говорят, что у него есть *root-доступ*. В традиционной системе Unix это администратор.

ПРИМЕЧАНИЕ

Работать с правами root может оказаться опасно. Будет сложно выявить и исправить ошибки, поскольку система позволит вам выполнить что угодно, даже если вы пытаетесь причинить ей вред. Системщики постоянно стараются сделать так, чтобы root-доступ не был необходим, насколько это возможно. Например, при переключении между сетями беспроводного доступа на ноутбуке. В то же время, каким бы могущественным ни был пользователь root, он все-таки работает в режиме пользователя системы, а не в режиме ядра.

Группы состоят из пользователей. Основная цель групп заключается в том, чтобы пользователь мог предоставлять файлы для совместного доступа другим пользователям группы.

1.6. Заглядывая вперед

Итак, вы увидели, из чего состоит работающая система Linux. Пользовательские процессы создают среду, с которой вы непосредственно взаимодействуете. Ядро управляет процессами и аппаратными средствами. Обе эти составляющие — ядро и процессы — располагаются в памяти.

Теоретическая информация — это замечательно, но вы не сможете изучить детали системы Linux, только читая о ней. В следующей главе вы начнете свое путешествие, освоив некоторые основы пространства пользователя. Попутно вы узнаете о более обширных частях системы Linux, о которых не говорилось в этой главе: о долговременных запоминающих устройствах (жестких дисках, файлах и т. п.). Вам ведь необходимо где-то хранить свои программы и данные.