1

Введение в ядро Linux

Даже после трех десятилетий использования операционная система (ОС) Unix все еще считается одной из самых мощных и элегантных среди всех существующих операционных систем. Со времени создания операционной системы Unix в 1969 году, это детище Денниса Ритчи (Dennis Ritchie) и Кена Томпсона (Ken Thompson) стало легендарным творением, системой, принцип работы которой выдержал испытание временем и имя которой оказалось почти незапятнанным.

Операционная система Unix выросла из Multics — многопользовательской операционной системы, проект по созданию которой потерпел неудачу в корпорации Bell Laboratories. По прекращении проекта Multics, сотрудники центра Bell Laboratories' Computer Sciences Research Center прекратили работу и так и не создали дееспособной диалоговой операционной системы. Летом 1969 года программисты корпорации Bell Labs разработали проект файловой системы, которая в конце концов была включена в операционную систему Unix. Томпсон осуществил реализацию операционной системы для реально не используемой платформы PDP-7. В 1971 году операционная система Unix была перенесена на платформу PDP-11, а в 1973 году переписана с использованием языка программирования C, что было беспрецедентным шагом в то время, но этот шаг стал основой для будущей переносимости. Первая версия операционной системы Unix, которая использовалась вне стен Bell Labs, называлась Unix System версии 6, ее обычно называют V6.

Другие компании перенесли операционную систему Unix на новые типы машин. Версии, полученные в результате переноса, содержали улучшения, которые позже привели к появлению нескольких разновидностей этой операционной системы. В 1977 году корпорация Bell Labs выпустила комбинацию этих вариантов в виде одной операционной системы Unix System III, а в 1982 году корпорация AT&T представила версию System V[1].

Простота устройства операционной системы Unix, а также тот факт, что эта система распространялась вместе со своим исходным кодом, привели к тому, что дальнейшие разработки начали проводиться в других организациях. Наиболее важным среди таких разработчиков был Калифорнийский университет в городе Беркли (University of California at Berkeley).

[1] Как насчет версии System IV? Ходят слухи, что это внутренняя экспериментальная версия.

Варианты операционной системы Unix из Беркли именовались Berkeley Software Distributions (BSD). Первая версия операционной системы Unix, разработанная в Беркли в 1981 году, называлась 3BSD. Следом за ней появились выпуски серии 4BSD: 4.0BSD, 4.1BSD, 4.2BSD и 4.3BSD. В этих версиях операционной системы Unix была добавлена виртуальная память, замещение страниц по требованию (demand paging) и стек протоколов TCP/IP. Последней официальной версией ОС Unix из Беркли была 4.4BSD, выпущенная в 1993 году, которая содержала переписанную систему управления виртуальной памятью. Сейчас разработка линии BSD продолжается в операционных системах Darwin, Dragonfly BSD, FreeBSD, NetBSD и OpenBSD.

В 1980–1990-х годах многие компании, разработчики рабочих станций и серверов, предложили свои коммерческие версии операционной системы Unix. Эти операционные системы обычно базировались на реализациях AT&T или Беркли и поддерживали дополнительные профессиональные возможности, которые обеспечивала соответствующая аппаратная платформа. Среди таких систем были Tru64 компании Digital, HP-UX компании Hewlett Packard, AIX компании IBM, DYNIX/ptx компании Sequent, IRIX компании SGI, Solaris компании Sun.

Первоначальное элегантное устройство операционной системы Unix в соединении с многолетними нововведениями и улучшениями, которые за ними последовали, сделали систему Unix мощной, устойчивой и стабильной. Очень небольшое количество характеристик ОС Unix ответственны за ее устойчивость. Во-первых, операционная система Unix проста: в то время как в некоторых операционных системах реализованы тысячи системных вызовов и эти системы имеют недостаточно ясное назначение, Unix-подобные операционные системы обычно имеют только несколько сотен системных вызовов и достаточно четкий дизайн. Во-вторых, в операционной системе Unix *все представляется в виде файлов*[2]. Такая особенность позволяет упростить работу с данными и устройствами, а также обеспечить это посредством простых системных вызовов: `open()`, `read()`, `write()`, `ioctl()` и `close()`. В-третьих, ядро и системные утилиты операционной системы Unix написаны на языке программирования C — это свойство делает Unix удивительно переносимой и доступной для широкого круга разработчиков операционной системой.

Для ОС Unix характерно очень малое время создания нового процесса и уникальный системный вызов `fork()`. И наконец, операционная система Unix предоставляет простые и в то же время устойчивые средства межпроцессного взаимодействия, которые, в сочетании с быстрым созданием процессов, позволяют создавать простые утилиты, которые *умеют выполнять всего одну функцию, но делают это хорошо*, и могут быть связаны вместе для выполнения более сложных задач.

Сегодня Unix — современная операционная система, которая поддерживает многозадачность, многопоточность, виртуальную память, замещение страниц по требованию, библиотеки совместного использования, загружаемые по требованию, и сеть TCP/IP. Многие варианты операционной системы Unix поддерживают масштабирование до сотен процессоров, в то время как другие варианты ОС Unix работают на миниатюрных устройствах в качестве встраиваемых систем. Хотя разработка Unix больше не является исследовательским проектом, все же продолжаются разработки (с целью получить дополнительные преимущества) с использованием возможностей

[2] Да, конечно, не все, но многое представлено в виде файла. В современных операционных системах, таких как Plan9 (наследник Unix), практически все представляется в виде файлов.

операционной системы Unix, которая при этом остается практичной операционной системой общего назначения.

Операционная система Unix обязана своим успехом простоте и элегантности построения. В основе ее сегодняшней мощности лежат давние идеи Денниса Ритчи, Кена Томпсона и других разработчиков, обеспечившие возможность операционной системе Unix бескомпромиссно развиваться.

Потом пришел Линус: введение в Linux

Операционная система Linux была разработана Линусом Торвальдсом (Linus Torvalds) в 1991 году как операционная система для компьютеров, работающих на новом в то время микропроцессоре Intel 80386. Тогда Линус Торвальдс был студентом университета в Хельсинки и был крайне возмущен отсутствием мощной и в то же время свободно доступной Unix-подобной операционной системы. Операционная система DOS, продукт корпорации Microsoft, была для Торвальдса полезна только лишь, чтобы поиграть в игрушку "Принц Персии", и не для чего больше. Линус пользовался операционной системой Minix, недорогой Unix-подобной операционной системой, которая была создана в качестве учебного пособия. В этой операционной системе ему не нравилось отсутствие возможности легко вносить и распространять изменения исходного кода (это запрещалось лицензией ОС Minix), а также технические решения, которые использовал автор ОС Minix.

Поставленный перед такой проблемой, Линус решил написать свою операционную систему. Начал он с написания простого эмулятора терминала, который он подключал к большим Unix-системам в университете. Его эмулятор терминала постепенно рос, развивался и улучшался. Постепенно у Линуса появилась еще не совсем зрелая, но полноценная Unix-система. В 1991 году он опубликовал в Интернет ее первую версию.

По некоторым неясным причинам, использование операционной системы Linux и количество ее пользователей начали стремительно расти. Более важным для успеха Linux стало то, что эта операционная система привлекла многих разработчиков, которые начали изменять, исправлять и улучшать код. Благодаря соответствующему лицензионному соглашению, ОС Linux быстро стала совместным проектом, который разрабатывается многими людьми.

Сейчас Linux — это развитая операционная система, работающая на аппаратных платформах AMD x86-64, ARM, Compaq Alpha, CRIS, DEC VAX, H8/300, Hitachi SuperH, HP PA-RISC, IBM S/390, Intel IA-64, MIPS, Motorola 68000, PowerPC, SPARC, UltraSPARC и v850. Она работает в различных системах, как размером с часы, так и на больших супер-компьютерных кластерах. Сегодня коммерческий интерес к операционной системе Linux достаточно высок. Как новые корпорации, ориентирующиеся исключительно на Linux (Monta Vista или Red Hat), так и старые (IBM, Novell) предлагают решения на основе этой ОС для встраиваемых систем, десктопов и серверов.

Операционная система Linux является клоном Unix, но ОС Linux — это не Unix. Хотя в ОС Linux позаимствовано много идей от Unix, в Linux реализован API ОС Unix (как это определено в стандарте POSIX и спецификации Single Unix Specification), все же система Linux не является производной от исходного кода Unix, как это имеет место для других Unix-систем. Там, где это желательно, были сделаны отклонения от пути, по которому шли другие разработчики, однако это не

подрывает основные принципы построения операционной системы Unix и не нарушает программные интерфейсы.

Одна из наиболее интересных особенностей операционной системы Linux — то, что это не коммерческий продукт; наоборот, это совместный проект, который выполняется через всемирную сеть Интернет. Конечно, Линус остается создателем Linux и занимается *поддержкой* ядра, но работа продолжается группой мало связанных между собой разработчиков. Фактически кто угодно может внести свой вклад в операционную систему Linux. Ядро Linux, так же как и большая часть операционной системы, является *свободно распространяемым* программным обеспечением и имеет *открытый исходный код*[3].

В частности, ядро Linux выпускается под лицензией GNU General Public License (GPL) версии 2.0. В результате каждый имеет право загружать исходный код и вносить в него любые изменения. Единственная оговорка — любое распространение внесенных вами изменений должно производиться на тех же условиях, которыми пользовались вы при получении исходного кода, включая доступность самого исходного программного кода[4].

Операционная система Linux предоставляет много возможностей для многих людей. Основными частями системы являются ядро, библиотека функций языка C, компилятор, набор инструментов, основные системные утилиты, такие как программа для входа в систему (login) и обработчик команд пользователя (shell). В операционную систему Linux может быть включена современная реализация системы X Windows, включая полно-функциональную среду офисных приложений (desktop environment), такую как, например, GNOME. Для ОС Linux существуют тысячи свободных и коммерческих программ. В этой книге под понятием *Linux*, в основном, имеется в виду *ядро Linux*. Там, где это может привести к неопределенностям, будет указано, что имеется в виду под понятием Linux — вся система или только ядро. Строго говоря, термин Linux относится только к ядру.

Обзор операционных систем и ядер

Из-за неуклонного роста возможностей и не очень качественного построения некоторых современных операционных систем, понятие операционной системы стало несколько неопределенным. Многие пользователи считают, что то, что они видят на экране, — и есть операционная система. Обычно, и в этой книге тоже, под *операционной системой* понимается часть компьютерной системы, которая отвечает за основные функции использования и администрирования. Это включает в себя ядро и драйверы устройств, системный загрузчик (boot loader), командный процессор и другие интерфейсы пользователя, а также базовую файловую систему и системные утилиты. В общем, только *необходимые* компоненты. Термин *система* обозначает операционную систему и все пользовательские программы, которые работают под ее управлением.

Конечно, основной темой этой книги будет *ядро* операционной системы. Интерфейс пользователя — это внешняя часть операционной системы, а ядро — вну-

[3] Для тех, кому интересно, дискуссия по поводу отличия свободного кода от открытого доступна в Интернет по адресам `http://www.fsf.org` и `http://www.opensource.org`.

[4] Вероятно, вам нужно прочесть лицензию GNU GPL, если вы еще не читали ее. В файле COPYING, в исходном коде ядра, есть копия этой лицензии. В Интернет лицензия доступна по адресу `http://www.fsf.org`.

тренняя. В своей основе ядро — это программное обеспечение, которое предоставляет базовые функции для всех остальных частей операционной системы, занимается управлением аппаратурой и распределяет системные ресурсы. Ядро часто называют *основной частью* (core) или *контроллером* операционной системы. Типичные компоненты ядра — обработчики прерываний, которые обслуживают запросы на прерывания, планировщик, который распределяет процессорное время между многими процессами, система управления памятью, которая управляет адресным пространством процессов, и системные службы, такие как сетевая подсистема и подсистема межпроцессного взаимодействия. В современных системах с устройствами управления защищенной памятью ядро обычно занимает привилегированное положение по отношению к пользовательским программам. Это включает доступ ко всем областям защищенной памяти и полный доступ к аппаратному обеспечению. Состояние системы, в котором находится ядро, и область памяти, в которой находится ядро, вместе называются *пространством ядра* (или режимом ядра, kernel-space). Соответственно, пользовательские программы выполняются в *пространствах задач* (пользовательский режим, режим задач, user-space). Пользовательским программам доступно лишь некоторое подмножество машинных ресурсов, они не могут выполнять некоторые системные функции, напрямую обращаться к аппаратуре и делать другие недозволенные вещи. При выполнении программного кода ядра система находится в пространстве (режиме) ядра, в отличие от нормального выполнения пользовательских программ, которое происходит в режиме задачи.

Прикладные программы, работающие в системе, взаимодействуют с ядром с помощью интерфейса *системных вызовов* (system call) (рис. 1.1). Прикладная программа обычно вызывает функции различных библиотек, например *библиотеки функций* языка C, которые, в свою очередь, обращаются к интерфейсу системных вызовов для того, чтобы отдать приказ ядру выполнить определенные действия от их имени. Некоторые библиотечные вызовы предоставляют функции, для которых отсутствует системный вызов, и поэтому обращение к ядру — это только один этап в более сложной функции. Давайте рассмотрим всем известную функцию `printf()`. Эта функция обеспечивает форматирование и буферизацию данных и лишь после этого один раз обращается к системному вызову `write()` для вывода данных на консоль. Некоторые библиотечные функции соответствуют функциям ядра один к одному. Например, библиотечная функция `open()` не делает ничего, кроме выполнения системного вызова `open()`. В то же время некоторые библиотечные функции, как, например, `strcpy()`, надо полагать, вообще не используют обращения к ядру. Когда прикладная программа выполняет системный вызов, то говорят, что *ядро выполняет работу от имени прикладной программы*. Более того, говорят, что прикладная программа *выполняет системный вызов в пространстве ядра*, а ядро выполняется в *контексте процесса*. Такой тип взаимодействия, когда прикладная программа *входит* в ядро через интерфейс системных вызовов, является фундаментальным способом выполнения задач.

В функции ядра входит также управление системным аппаратным обеспечением. Практически все платформы, включая те, на которых работает операционная система Linux, используют *прерывания* (interrupt). Когда аппаратному устройству необходимо как-то взаимодействовать с системой, оно генерирует прерывание, которое прерывает работу ядра в асинхронном режиме[5].

[5] Иными словами, заранее неизвестно, в какой момент времени это событие произойдет и в каком состоянии будет система в этот момент времени. — *Прим. перев.*

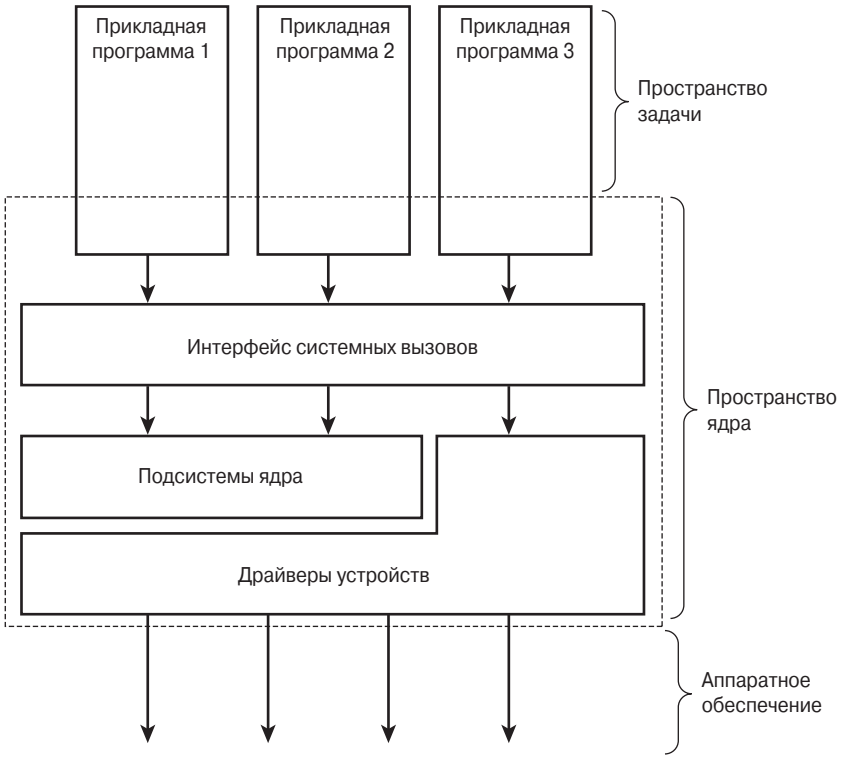

Рис. 1.1. *Взаимодействие между прикладными программами, ядром и аппаратным обеспечением*

Обычно каждому типу прерываний соответствует номер. Ядро использует номер прерывания для выполнения специального обработчика прерывания (interrupt handler), который обрабатывает прерывание и отправляет на него ответ. Например, при вводе символа с клавиатуры, контроллер клавиатуры генерирует прерывание, чтобы дать знать системе, что в буфере клавиатуры есть новые данные. Ядро определяет номер прерывания, которое пришло в систему и выполняет соответствующий обработчик прерывания. Обработчик прерывания обрабатывает данные, поступившие с клавиатуры, и дает знать контроллеру клавиатуры, что ядро готово для приема новых данных. Для обеспечения синхронизации выполнения ядро обычно может запрещать прерывания: или все прерывания, или только прерывание с определенным номером. Во многих операционных системах обработчики прерываний не выполняются в контексте процессов. Они выполняются в специальном *контексте прерывания* (interrup context), который не связан ни с одним процессом. Этот специальный контекст существует только для того, чтобы дать обработчику прерывания возможность быстро отреагировать на прерывание и закончить работу.

Контексты выполнения заданий полностью определяют всю широту возможных действий ядра. Фактически, можно заключить, что в операционной системе Linux процессор в любой момент времени выполняет один из трех типов действий.

- Работа от имени определенного процесса в режиме ядра в контексте процесса.

- Работа по обработке прерывания в режиме ядра в контексте прерывания, не связанном с процессами.
- Выполнение кода пользовательской программы в режиме задачи.

Ядро Linux в сравнении с классическими ядрами Unix

Благодаря общему происхождению и одинаковому API, современные ядра Unix имеют некоторые общие характерные черты. За небольшими исключениями ядра Unix представляют собой монолитные статические бинарные файлы. Это значит, что они существуют в виде больших исполняемых образов, которые выполняются один раз и используют одну копию адресного пространства. Для работы операционной системы Unix обычно требуется система с контроллером управления страничной адресацией памяти (memory management unit); это аппаратное обеспечение позволяет обеспечить защиту памяти в системе и предоставить каждому процессу уникальное виртуальное адресное пространство. В списке литературы приведены мои любимые книги по устройству классических ядер операционной системы Unix.

Сравнение решений на основе монолитного ядра и микроядра

Операционные системы, в соответствии с особенностями построения, можно разделить на две большие группы: с монолитным ядром и с микроядром. (Есть еще третий тип — экзоядро, которое пока еще используется, в основном, только в исследовательских операционных системах, но уже начинает пробивать дорогу в большой мир.)

Монолитное ядро является самым простым, и до 1980-х годов все ядра строились именно таким образом. Монолитное ядро реализовано в виде одного большого процесса, который выполняется в одном адресном пространстве. Такие ядра обычно хранятся на диске в виде одного большого статического бинарного файла. Все службы ядра существуют и выполняются в одном большом адресном пространстве ядра. Взаимодействия в ядре выполняются очень просто, потому что все, что выполняется в режиме ядра, — выполняется в одном адресном пространстве. Ядро может вызывать функции непосредственно, как это делает пользовательское приложение. Сторонники такой модели обычно указывают на простоту и высокую производительность монолитных ядер.

Микроядра не реализуются в виде одного большого процесса. Все функции ядра разделяются на несколько процессов, которые обычно называют *серверами*. В идеале, в привилегированном режиме работают только те серверы, которым абсолютно необходим привилегированный режим. Остальные серверы работают в пространстве пользователя. Все серверы, тем не менее, поддерживаются независимыми друг от друга и выполняются каждый в своем адресном пространстве. Следовательно, прямой вызов функций, как в случае монолитного ядра, невозможен. Все взаимодействия внутри микроядра выполняются с помощью передачи сообщений. Механизм межпроцессного взаимодействия (Inter Process Communication, IPC) встраивается в систему, и различные серверы взаимодействуют между собой и обращаются к “службам” друг друга путем отправки сообщений через механизм IPC. Разделение серверов позволяет предотвратить возможность выхода из строя одного сервера при выходе из строя другого.

Кроме того, модульность системы позволяет одному серверу вытеснять из памяти другого. Поскольку механизм IPC требует больше накладных расходов, чем обычный вызов функции, и при этом может потребоваться переключение контекста из пространства пользователя в пространство ядра и наоборот, то передача сообщений приводит к падению производительности по сравнению с монолитными ядрами, в которых используются обычные вызовы функций.

В современных операционных системах с микроядром, большинство серверов выполняется в пространстве ядра, чтобы избавиться от накладных расходов, связанных с переключением контекста, кроме того, это дает потенциальную возможность прямого вызова функций. Ядро операционной системы Windows NT, а также ядро Mach (на котором базируется часть операционной системы Mac OS X) — это примеры микроядер. В последних версиях как Windows NT, так и Mac OS X все серверы выполняются только в пространстве ядра, что является отходом от первоначальной концепции микроядра.

Ядро ОС Linux монолитное, т.е. оно выполняется в одном адресном пространстве, в режиме ядра. Тем не менее ядро Linux позаимствовало некоторые хорошие свойства микроядерной модели: в нем используется преемптивное ядро, поддерживаются потоки пространства ядра и возможность динамической загрузки в ядро внешних бинарных файлов (модулей ядра). Ядро Linux не использует никаких функций микроядерной модели, которые приводят к снижению производительности: все выполняется в режиме ядра с непосредственным вызовом функций, вместо передачи сообщений. Следовательно, операционная система Linux — модульная, многопоточная, а выполнение самого ядра можно планировать.

Прагматизм снова победил.

По мере того как Линус и другие разработчики вносили свой вклад в ядро Linux, они принимали решения о том, как развивать ОС Linux без пренебрежения корнями, связанными с Unix (и, что более важно, без пренебрежения API ОС Unix). Поскольку операционная система Linux не базируется на какой-либо версии ОС Unix, Линус и компания имели возможность найти и выбрать наилучшее решение для любой проблемы и даже со временем изобрести новые решения! Ниже приводится анализ характеристик ядра Linux, которые отличают его от других разновидностей Unix.

- Ядро Linux поддерживает динамическую загрузку модулей ядра. Хотя ядро Linux и является монолитным, оно дополнительно поддерживает динамическую загрузку и выгрузку исполняемого кода ядра по необходимости.
- Ядро Linux поддерживает симметричную многопроцессорную обработку (SMP). Хотя большинство коммерческих вариантов операционной системы Unix сейчас поддерживает SMP, большинство традиционных реализаций ОС Unix такой поддержки не имеет.
- Ядро Linux является преемптивным. В отличие от традиционных вариантов ОС Unix, ядро Linux в состоянии вытеснить выполняющееся задание, даже если это задание работает в режиме ядра. Среди коммерческих реализаций ОС Unix преемптивное ядро имеют только операционные системы Solaris и IRIX.
- В ядре Linux используется интересный подход для поддержки многопоточности (threads): потоки ни чем не отличаются от обычных процессов. С точки зрения ядра все процессы одинаковы, просто некоторые из них имеют общие ресурсы.
- В ядре Linux отсутствуют некоторые функции ОС Unix, которые считаются плохо реализованными, как, например, поддержка интерфейса STREAMS, или отвечают “глупым” стандартам.
- Ядро Linux является полностью открытым во всех смыслах этого слова. Набор функций, реализованных в ядре Linux, – это результат свободной и открытой модели разработки операционной системы Linux. Если какая-либо функция ядра считается маловажной или некачественной, то разработчики ядра

не обязаны ее реализовать. В противоположность этому, внесение изменений при разработке ядра Linux занимает "элитарную" позицию: изменения должны решать определенную практическую задачу, должны быть логичными и иметь понятную четкую реализацию. Следовательно, функции некоторых современных вариантов ОС Unix, такие как память ядра со страничной реализацией, не были реализованы. Несмотря на имеющиеся различия, Linux является операционной системой со строгим наследованием традиций ОС Unix.

Версии ядра Linux

Ядро Linux поставляется в двух вариантах: стабильном (stable) и разрабатываемом (development). Версии стабильного ядра — это выпуски продукции промышленного уровня, которая готова для широкого использования. Новые стабильные версии ядра обычно выпускаются для исправления ошибок и для предоставления новых драйверов устройств. Разрабатываемые версии ядра, наоборот, подвержены быстрым изменениям. По мере того как разработчики экспериментируют с новыми решениями, часто вносятся радикальные изменения в ядро.

Ядра Linux стабильных и разрабатываемых версий можно отличить друг от друга с помощью простой схемы присваивания имен (рис. 1.2.). Три числа, которые разделяются точкой, определяют версию ядра. Первое число — значение старшей (major) версии, второе — значение младшей (minor), третье число — значение редакции (выпуска, revision). Значение младшей версии также определяет, является ли ядро стабильным или разрабатываемым; если это значение четное, то ядро стабильное, а если нечетное, то разрабатываемое. Так, например, версия 2.6.0 определяет стабильное ядро. Ядро имеет старшую версию 2, младшую версию 6 и редакцию 0. Первые два числа также определяют "серию ядер", в данном случае серия ядер — 2.6.

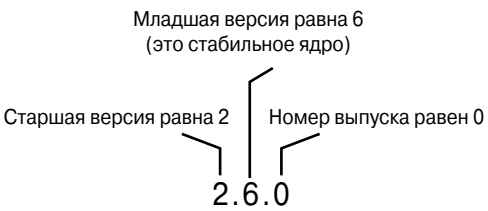

Рис. 1.2. *Соглашение о присваивании имен ядрам*

Разработка ядра соответствует различным фазам. Вначале разработчики ядра работают над новыми функциями, что напоминает хаос. Через определенное время ядро оказывается сформировавшимся, и в конце концов объявляется замораживание функций.

Начиная с этого момента никакие новые функции не могут быть добавлены в ядро. Однако работа над существующими функциями может быть продолжена. После того как ядро становится почти стабильным, осуществляется замораживание кода. В этом случае допускаются только исправления ошибок. Вскоре после этого (можно надеяться) ядро выпускается в виде первой, новой, стабильной версии. Например, при стабилизации серии ядер 2.5 получается серия 2.6.

Все это не правда

По крайней мере — не совсем. Приведенное только что описание процесса разработки ядра технически правильное. Раньше процесс происходил именно так, как описано. Тем не менее летом 2004 года на ежегодном саммите для приглашенных разработчиков ядра Linux было принято решение продолжить разработку серии 2.6 ядра Linux и в ближайшем будущем не переходить на серию разрабатываемого ядра 2.7. Такое решение было принято потому, что ядро 2.6 получилось хорошим; оно, в основном, стабильно и на горизонте нет никаких новых функций, которые требуют серьезного вторжения в ядро.

Кроме того, и, возможно, это главное — существующая система поддержки, которая обеспечивается Линусом Торвальдсом и Эндрю Мортоном, работает чрезвычайно хорошо. Разработчики ядра уверены, что процесс разработки может продолжаться таким образом, что серия ядер 2.6 будет оставаться стабильной и в ней будут появляться новые возможности. Время рассудит, но уже сейчас результаты выглядят хорошо.

Эта книга базируется на ядрах стабильной серии 2.6.

Сообщество разработчиков ядра Linux

Когда вы начинаете разрабатывать код ядра Linux, вы становитесь частью глобального сообщества разработчиков ядра Linux. Главный форум этого сообщества — *список рассылки разработчиков ядра Linux* (linux-kernel mailing list). Информация по поводу подписки на этот форум доступна по адресу `http://vger.kernel.org`. Следует заметить, что это достаточно перегруженный сообщения список рассылки (количество сообщений порядка 300 в день) и что другие читатели этого списка (разработчики ядра, включая Линуса) не очень склонны заниматься ерундой. Однако этот список рассылки может оказать неоценимую помощь в процессе разработки; здесь вы сможете найти тестологов, получить экспертную оценку и задать вопросы.

В последних главах приведен обзор процесса разработки ядра и более полное описание того, как успешно принимать участие в деятельности сообщества разработчиков ядра.

Перед тем как начать

Эта книга посвящена ядру Linux: как оно работает, почему оно работает и чему следует уделить внимание. Далее будут описаны принципы работы и реализация основных подсистем ядра, а также интерфейсы и программная семантика. Эта книга касается практических вопросов, и в ней используется подход на основании золотой серединки указанных выше направлений. Такой интересный подход в сочетании с анекдотами из личной практики автора и советами по хакерским приемам позволяет быть уверенным в том, что книга станет хорошим стартом.

Я надеюсь, что у читателей есть доступ к системе Linux и дереву исходного кода ядра. В идеале предполагается, что читатель — это пользователь операционной системы Linux, который уже "копался" в исходном программном коде, но все же нуждается в некоторой помощи для того, чтобы все связать воедино. В принципе, читатель может и не быть пользователем Linux, но хочет разобраться в устройстве ядра из чистого любопытства. Тем не менее, для того чтобы самому научиться писать программы — исходный код незаменим. Исходный программный код *свободно* доступен — пользуйтесь им!

Удачи!