Анатомия ядра Linux

История и архитектурная организация

М. Тим Джонс (mtj@mtjones.com)
инженер-консультант
Emulex

24.07.2007

Ядро Linux - это центральная часть большой и сложной операционной системы. При этом, несмотря на колоссальные размеры, оно имеет четкую структурную организацию в виде подсистем и уровней. В этой статье мы рассказываем об общей структуре ядра Linux и знакомим вас с его основными подсистемами и базовыми интерфейсами. Везде, где это возможно, мы приводим ссылки на другие статьи IBM, где вы сможете найти углубленную информацию.

Поскольку цель данной статьи - познакомить вас с ядром Linux и дать обзор его архитектуры и основных компонентов, давайте начнем с краткого обзора истории ядра Linux, затем рассмотрим архитектуру ядра Linux "с высоты птичьего полета", и, наконец, обсудим его основные подсистемы. Ядро Linux насчитывает свыше шести миллионов строк, поэтому данное введение не может быть исчерпывающим. Для получения более подробной информации пользуйтесь ссылками на дополнительные ресурсы.

Краткий обзор истории Linux

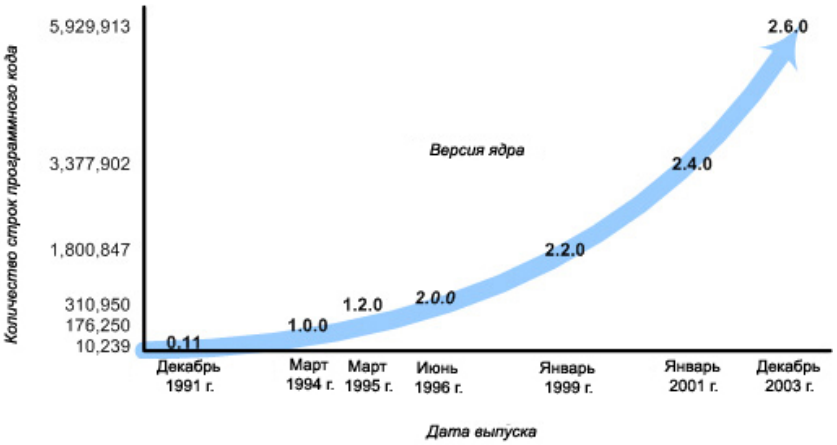
Хотя Linux, по всей видимости, является самой популярной операционной системой с открытым исходным кодом, на самом деле ее история в сравнении с другими операционными системами относительно коротка. На заре компьютерной эры программисты разрабатывали свои программы для "голой" аппаратуры, используя языки, понятные для этой аппаратуры. В отсутствие операционной системы использовать всю большую и дорогую вычислительную машину в каждый конкретный момент времени могло только одно приложение (и один пользователь). Первые операционные системы были разработаны в 1950-е годы, чтобы облегчить жизнь разработчиков. В качестве примера можно назвать General Motors Operating System (GMOS), разработанную для IBM 701, и FORTRAN Monitor System (FMS), созданную North American Aviation для IBM 709.

В 1960-е годы в Массачусетском Технологическом институте (MIT) и в ряде компаний была разработана экспериментальная операционная система Multics (Multiplexed Information and Computing Service) для машины GE-645. Один из разработчиков этой ОС, компания AT&T,

отошла от Multics и в 1970 году разработала свою собственную систему Unics. Вместе с этой ОС поставлялся язык C. При этом C был разработан и написан так, чтобы обеспечить переносимость разработки операционной системы.

Двадцать лет спустя Эндрю Танненбаум (Andrew Tanenbaum) создал микроядерную версию UNIX® под названием MINIX (minimal UNIX), которая могла работать на небольших персональных компьютерах. Эта операционная система с открытым исходным кодом вдохновила Линуса Торвальдса (Linus Torvalds) на разработку первой версии Linux в начале 1990-х (см. Рис. 1).

Рис. 1. Краткая история основных выпусков ядра Linux

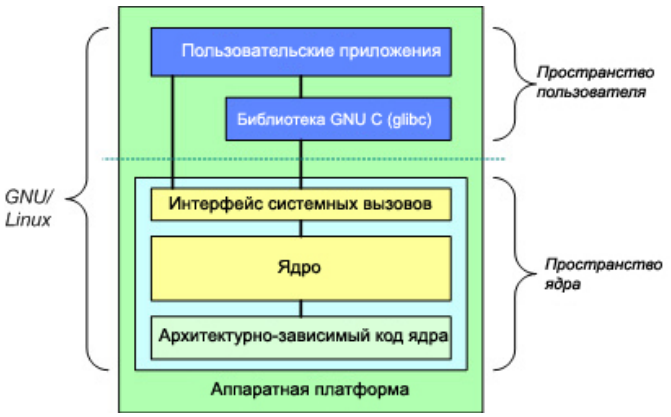


Linux быстро превратился из инициативы энтузиаста-одиночки во всемирный проект, в котором участвуют тысячи разработчиков. Одним из важнейших решений в судьбе Linux стало принятие лицензии GNU General Public License (GPL). GPL защитила ядро Linux от коммерческой эксплуатации и одновременно открыла путь к использованию разработок сообщества пользователей проекта GNU, основанного Ричардом Столлменом (Richard Stallman), объемы кода которого значительно превосходят даже объем ядра Linux. Это позволило использовать в Linux такие полезные приложения, как комплекс компиляторов GNU Compiler Collection (GCC) и различные командные оболочки.

Введение в ядро Linux

Перейдем к общему обзору архитектуры операционной системы GNU/Linux. Операционную систему можно условно разделить на два уровня, как показано на Рис. 2.

Рис. 2. Фундаментальная архитектура операционной системы GNU/Linux



На верхнем уровне находится пользовательское пространство (пространство приложений). Здесь исполняются приложения пользователя. Под пользовательским пространством располагается пространство ядра. Здесь функционирует ядро Linux.

Имеется также библиотека GNU C (glibc). Она предоставляет интерфейс системных вызовов, который обеспечивает связь с ядром и дает механизм для перехода от приложения, работающего в пространстве пользователя, к ядру. Это важно, поскольку ядро и пользовательское приложение располагаются в разных защищенных адресных пространствах. При этом, в то время как каждый процесс в пространстве пользователя имеет свое собственное виртуальное адресное пространство, ядро занимает одно общее адресное пространство. Более подробную информацию можно найти в литературе, ссылки на которую приведены в разделе "Ресурсы".

Ядро Linux можно, в свою очередь, разделить на три больших уровня. Наверху располагается интерфейс системных вызовов, который реализует базовые функции, например, `чтение` и `запись`. Ниже интерфейса системных вызовов располагается код ядра, точнее говоря, архитектурно-независимый код ядра. Этот код является общим для всех процессорных архитектур, поддерживаемых Linux. Еще ниже располагается архитектурно-зависимый код, образующий т.н. BSP (Board Support Package - пакет поддержки аппаратной платформы). Этот код зависит от процессора и платформы для конкретной архитектуры.

Свойства ядра Linux

Обсуждая архитектуру большой и сложной системы, можно рассматривать ее со многих разных точек зрения. Одна из целей архитектурного анализа может состоять в том, чтобы лучше понять исходный код системы. Именно этим мы здесь и займемся.

В ядре Linux реализован целый ряд важных архитектурных элементов. И на самом общем, и на более детальных уровнях ядро можно подразделить на множество различных подсистем. С другой стороны, Linux можно рассматривать как монолитное целое, поскольку все базовые сервисы собраны в ядре системы. Такой подход отличается от архитектуры с микроядром, когда ядро предоставляет только самые общие сервисы, такие как обмен информацией. ввод/вывод, управление памятью и процессами, а более конкретные сервисы реализуются

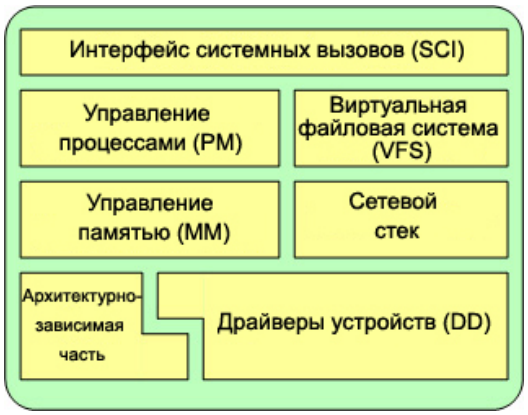
в модулях, подключаемых к уровню микроядра. Каждая из этих точек зрения имеет свои достоинства, но я здесь не буду вдаваться в это обсуждение.

С течением времени ядро Linux стало более эффективным с точки зрения использования памяти и процессорных ресурсов и приобрело исключительную стабильность. Однако самый интересный аспект Linux, учитывая размер и сложность этой системы - это ее переносимость. Linux можно откомпилировать для огромного количества разных процессоров и платформ, имеющих разные архитектурные ограничения и потребности. Например, Linux может работать на процессоре как с блоком управления памятью (MMU), так и без MMU. Поддержка процессоров без MMU реализована в версии ядра uClinux. Более подробную информацию см. в разделе "Ресурсы".

Основные подсистемы ядра Linux

Давайте рассмотрим некоторые основные компоненты ядра Linux, следуя структуре, изображенной на рис. 3.

Рис. 3. Один из возможных взглядов на архитектуру ядра Linux



Интерфейс системных вызовов

SCI - это тонкий уровень, предоставляющий средства для вызова функций ядра из пространства пользователя. Как уже говорилось, этот интерфейс может быть архитектурно зависимым, даже в пределах одного процессорного семейства. SCI фактически представляет собой службу мультиплексирования и демультиплексирования вызова функций. Реализация SCI находится в ./linux/kernel, а архитектурно-зависимая часть - в ./linux/arch. Более подробные сведения об этом компоненте можно найти в разделе Ресурсы.

Управление процессами

Управление процессами сконцентрировано на исполнении процессов. В ядре эти процессы называются *потоками* (threads); они соответствуют отдельным виртуализованным объектам процессора (код потока, данные, стек, процессорные регистры). В пространстве пользователя обычно используется термин *процесс*, хотя в реализации Linux эти две концепции (процессы и потоки) не различают. Ядро предоставляет интерфейс программирования приложений (API) через SCI для создания нового процесса (порождения копии, запуска на исполнение, вызова функций Portable Operating System Interface [POSIX]),

остановки процесса (kill, exit), взаимодействия и синхронизации между процессами (сигналы или механизмы POSIX).

Еще одна задача управления процессами - совместное использование процессора активными потоками. В ядре реализован новаторский алгоритм планировщика, время работы которого не зависит от числа потоков, претендующих на ресурсы процессора. Название этого планировщика - O(1) - подчеркивает, что на диспетчеризацию одного потока затрачивается столько же времени, как и на множество потоков. Планировщик O(1) также поддерживает симметричные многопроцессорные конфигурации (SMP). Исходные коды системы управления процессами находятся в ./linux/kernel, а коды архитектурно-зависимой части - в ./linux/arch). Более подробную информацию об этом алгоритме см. в разделе Ресурсы.

Управление памятью

Другой важный ресурс, которым управляет ядро - это память. Для повышения эффективности, учитывая механизм работы аппаратных средств с виртуальной памятью, память организуется в виде т.н. *страниц* (в большинстве архитектур размером 4 КБ). В Linux имеются средства для управления имеющейся памятью, а также аппаратными механизмами для установления соответствия между физической и виртуальной памятью.

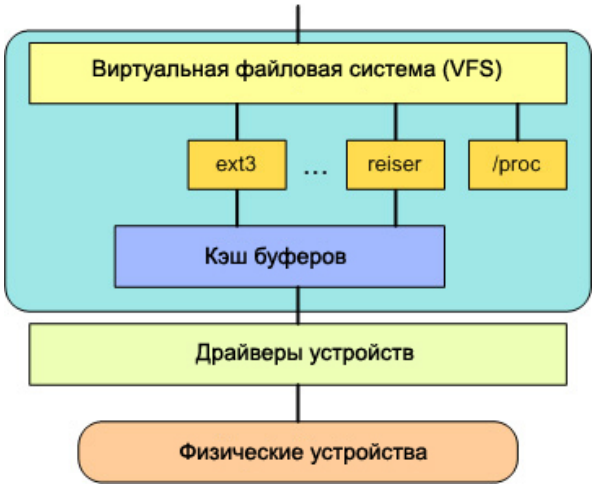
Однако управление памятью - это значительно больше, чем просто управление буферами по 4 КБ. Linux предоставляет абстракции над этими 4 КБ буферами, например, механизм распределения slab allocator. Этот механизм управления базируется на 4 КБ буферах, но затем размещает структуры внутри них, следя за тем, какие страницы полны, какие частично заполнены и какие пусты. Это позволяет динамически расширять и сокращать схему в зависимости от потребностей вышележащей системы.

В условиях наличия большого числа пользователей памяти возможны ситуации, когда вся имеющаяся память будет исчерпана. В связи с этим страницы можно удалять из памяти и переносить на диск. Этот процесс обмена страниц между оперативной памятью и жестким диском называется *подкачкой*. Исходные коды управления памятью находятся в ./linux/mm.

Виртуальная файловая система

Еще один интересный аспект ядра Linux - виртуальная файловая система (VFS), которая предоставляет общую абстракцию интерфейса к файловым системам. VFS предоставляет уровень коммутации между SCI и файловыми системами, поддерживаемыми ядром (см. Рис. 4).

Рис. 4. VFS предоставляет коммутационную матрицу между пользователями и файловыми системами



На верхнем уровне VFS располагается единая API-абстракция таких функций, как открытие, закрытие, чтение и запись файлов. На нижнем уровне VFS находятся абстракции файловых систем, которые определяют, как реализуются функции верхнего уровня. Они представляют собой подключаемые модули для конкретных файловых систем (которых существует более 50). Исходные коды файловых систем находятся в ./linux/fs.

Ниже уровня файловой системы находится кэш буферов, предоставляющий общий набор функций к уровню файловой системы (независимый от конкретной файловой системы). Этот уровень кэширования оптимизирует доступ к физическим устройствам за счет краткосрочного хранения данных (или упреждающего чтения, обеспечивающего готовность данных к тому моменту, когда они понадобятся). Ниже кэша буферов находятся драйверы устройств, реализующие интерфейсы для конкретных физических устройств.

Сетевой стек

Сетевой стек по своей конструкции имеет многоуровневую архитектуру, повторяющую структуру самих протоколов. Вы помните, что протокол Internet Protocol (IP) - это базовый протокол сетевого уровня, располагающийся ниже транспортного протокола Transmission Control Protocol, TCP). Выше TCP находится уровень сокетов, вызываемый через SCI.

Уровень сокетов представляет собой стандартный API к сетевой подсистеме. Он предоставляет пользовательский интерфейс к различным сетевым протоколам. Уровень сокетов реализует стандартизованный способ управления соединениями и передачи данных между конечными точками, от доступа к "чистым" кадрам данных и блокам данных протокола IP (PDU) и до протоколов TCP и User Datagram Protocol (UDP). Исходные коды сетевой подсистемы ядра находятся в каталоге ./linux/net.

Драйверы устройств

Подавляющее большинство исходного кода ядра Linux приходится на драйверы устройств, обеспечивающие возможность работы с конкретными аппаратными устройствами. В дереве

исходных кодов Linux имеется подкаталог драйверов, в котором, в свою очередь, имеются подкаталоги для различных типов поддерживаемых устройств, таких как Bluetooth, I2C, последовательные порты и т.д. Исходные коды драйверов устройств находятся в ./linux/drivers.

Архитектурно-зависимый код

Хотя основная часть Linux независима от архитектуры, на которой работает операционная система, в некоторых элементах для обеспечения нормальной работы и повышения эффективности необходимо учитывать архитектуру. В подкаталоге ./linux/arch находится архитектурно-зависимая часть исходного кода ядра, разделенная на ряд подкаталогов, соответствующих конкретным архитектурам. Все эти каталоги в совокупности образуют BSP. В случае обычного настольного ПК используется каталог i386. Подкаталог для каждой архитектуры содержит ряд вложенных подкаталогов, относящихся к конкретным аспектам ядра, таким как загрузка, ядро, управление памятью и т.д. Исходные коды архитектурно-зависимой части находятся в ./linux/arch.

Интересные особенности ядра Linux

Помимо переносимости и эффективности, ядро Linux обладает целым рядом других интересных функций, которые не были освещены в вышеприведенном рассмотрении.

Linux, как широко используемая на практике операционная система с открытым исходным кодом, является отличной испытательной площадкой для новых протоколов и их усовершенствований. Linux поддерживает большое количество сетевых протоколов, включая традиционный TCP/IP и его высокоскоростные расширения (для сетей быстрее Gigabit Ethernet [GbE] и 10 GbE). Linux также поддерживает такие протоколы, как Stream Control Transmission Protocol (SCTP), реализующий множество дополнительных функций, отсутствующих в TCP (применяется в качестве альтернативного протокола транспортного уровня).

Следует отметить, что ядро Linux является динамическим (поддерживает добавление и удаление программных компонентов без остановки системы). Эти компоненты называются динамически загружаемыми модулями ядра. Их можно вводить в систему при необходимости, как во время загрузки (если найдено конкретное устройство, для которого требуется такой модуль), так и в любое время по желанию пользователя.

Еще одно недавнее усовершенствование Linux - возможность ее использования в качестве операционной системы для других операционных систем (т.н. гипервизора). Недавно в ядро было внесено усовершенствование, получившее название Kernel-based Virtual Machine (KVM, виртуальная машина на базе ядра). В результате этой модификации в пространстве пользователя был реализован новый интерфейс, позволяющий исполнять поверх ядра с поддержкой KVM другие операционные системы. В таком режиме можно не только исполнять другие экземпляры Linux, но и виртуализовать Microsoft® Windows®. Единственное ограничение состоит в том, что используемый процессор должен поддерживать новые инструкции виртуализации. Более подробную информацию см. в разделе Ресурсы.

Дальнейшее изучение

В этой статье мы лишь в самых общих чертах рассказали об архитектуре ядра Linux и его особенностях и возможностях. Подробную информацию о содержимом ядра можно найти в каталоге с документацией, который имеется в любом дистрибутиве Linux. Обязательно ознакомьтесь с разделом Ресурсы в конце данной статьи, где имеются ссылки на более подробную информацию по многим обсуждаемым здесь темам.

Об авторе

М. Тим Джонс

М. Тим Джонс (M. Tim Jones) является архитектором встраиваемого программного обеспечения и автором работ: *Программирование Приложений под GNU/Linux*, *Программирование AI-приложений* и *Использование BSD-сокетов в различных языках программирования*. Он имеет опыт разработки процессоров для геостационарных космических летательных аппаратов, а также разработки архитектуры встраиваемых систем и сетевых протоколов. Сейчас Тим работает инженером-консультантом в корпорации Эмулекс (Emulex Corp.) в г.Лонгмонт, Колорадо.