МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ

УНИВЕРСИТЕТ имени академика С.П. КОРОЛЕВА»

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Загрузка процессов в Linux

Утверждено Редакционно-издательским советом университета

в качестве методических указаний к лабораторной работе

С А М А Р А

Издательство СГАУ

2010

УДК 004.451

Составитель А.М. С у х о в

Рецензент: к.т. н., доц. Попов С.Б.

Загрузка процессов в Linux: Методические указания к лабораторной работе/ Сост. А.М. Сухов. - Самара: Изд-во Самарского государственного аэрокосмического университета, 2010. 21с.

В настоящих методических указаниях приведен материал, необходимый для выполнения лабораторных работ по дисциплинам «Операционная система Linux на высокопроизводительных кластерах», «Перспективные информационные технологии»

Целью лабораторных работ является изучение основных возможностей по конфигурации сетевого соединения и приобретение навыков работы в операционной системе Linux.

Предназначено для слушателей ФПК СГАУ и студентов специальностей 010500, 010501

1. Цель лабораторной работы

- .

2. Основные сведения

Процесс это – совокупность программного кода и данных, загруженных в память ЭВМ. На первый взгляд процесс – это запущенная программа (приложение) или команда. Но это не совсем так. Некоторые приложения могут создавать несколько процессов одновременно.

Код процесса не обязательно должен выполняться в текущий момент времени, так как процесс может находиться в состоянии спящего. В этом случае выполнение кода такого процесса приостановлено. Существует всего 3 состояния, в которых может находиться процесс:

Работающий процесс – в данный момент код процесса выполняется.

Спящий процесс – в данный момент код процесса не выполняется в ожидании какого либо события (нажатия клавиши на клавиатуре, поступление данных из сети и т.д.)

Процесс-зомби – сам процесс уже не существует, его код и данные выгружены из оперативной памяти, но запись в таблице процессов остается по тем или иным причинам.

Каждому процессу в системе назначаются числовые идентификаторы (личные номера) в диапазоне от 1 до 65535 (**PID – Process Identifier – идентификатор процесса**) и идентификаторы родительского процесса (**PPID – Parent Process Identifier – идентификатор родительского процесса**). PID является именем процесса, по которому мы можем адресовать процесс в операционной системе при использовании различных средств просмотра и управления процессами. PPID определяет родственные отношения между процессами, которые в значительной степени определяют его свойства и возможности. Другие параметры, которые необходимы для работы программы, называют “окружение процесса”. Одним из таких параметров является **управляющий терминал** – имя терминального устройства, на которое процесс выводит информацию и с которого информацию получает. Управляющий терминал имеют далеко не все процессы. Процессы, не привязанные к какому-то конкретному терминалу называются “демонами” (daemons). Такие процессы, будучи запущенными пользователем, не завершают свою работу по окончании сеанса, а продолжают работать, так как они не связаны никак с текущим сеансом и не могут быть автоматически завершены. Как правило, с помощью демонов реализуются серверные службы, так например сервер печати реализован процессом-демоном cupsd, а сервер журналирования – syslogd.

Для просмотра списка процессов в Linux существует команда ***ps***. Формат команды следующий:

$ ps *[PID] [options]*

– просмотр списка процессов. Без параметров *ps* показывает все процессы, которые были запущены в течение текущей сессии, за исключением демонов. Options может принимать одно из следующих значений или их комбинации:

-a или *-e* – показать все процессы

-f – полный листинг

-w – показать полные строки описания процессов. Если они превосходят длину экрана, то перенести описание на следующую строку.

$ ps -ef

UID PID PPID C STIME TTY TIME CMD

root 1 0 0 10:01 ? 00:00:03 init [5]

root 2 1 0 10:01 ? 00:00:00 [keventd]

root 3 1 0 10:01 ? 00:00:00 [kapmd]

root 4 1 0 10:01 ? 00:00:00 [ksoftirqd_CPU0]

root 5 1 0 10:01 ? 00:00:24 [kswapd]

root 6 1 0 10:01 ? 00:00:00 [bdflush]

…

gserg 3126 3124 0 17:56 pts/2 00:00:00 /bin/bash

gserg 3160 3126 0 17:59 pts/2 00:00:00 ps -ef

Данная команда имеет много параметров, о которых вы можете прочитать в руководстве (man ps). Здесь описаны лишь наиболее часто используемые:

Параметр	Описание
-a	отобразить все процессы, связанных с терминалом (отображаются процессы всех пользователей)
-e	отобразить все процессы
-t список терминалов	отобразить процессы, связанные с терминалами
-u идентификаторы пользователей	отобразить процессы, связанные с данными идентификаторыми
-g идентификаторы групп	отобразить процессы, связанные с данными идентификаторыми групп
-x	отобразить все процессы, не связанные с терминалом

Процессы в ОС Linux обладают теми же правами, которыми обладает пользователь, от чьего имени был запущен процесс.

На самом деле операционная система воспринимает работающего в ней пользователя как набор запущенных от его имени процессов. Ведь и сам сеанс пользователя открывается в командной оболочке (или оболочке X) от имени пользователя. Поэтому когда мы говорим "права доступа пользователя к файлу" то подразумеваем "права доступа процессов, запущенных от имени пользователя к файлу".

Для определения имени пользователя, запустившего процесс, операционная система использует **реальные идентификаторы пользователя и группы**, назначаемые процессу. Но эти идентификаторы не являются решающими при определении прав доступа. Для этого у каждого процесса существует другая группа идентификаторов – **эффективные.**

Как правило, реальные и эффективные идентификаторы процессов одинаковые, но есть и исключения. Например, для работы утилиты *passwd* необходимо использовать идентификатор суперпользователя, так как только суперпользователь имеет права на запись в файлы паролей. В этом случае эффективные идентификаторы процесса будут отличаться от реальных. Возникает резонный вопрос – как это было реализовано?

У каждого файла есть набор специальных прав доступа – биты SUID и SGID. Эти биты позволяют при запуске программы присвоить ей эффективные идентификаторы владельца и группы-владельца соответственно и выполнять процесс с правами доступа другого пользователя. Так как файл *passwd* принадлежит пользователю *root* и у него установлен бит SUID, то при запуске процесс *passwd* будет обладать правами пользователя *root.*

Устанавливаются биты SGID и SUID командой *chmod*:

$ chmod u+s filename – установка бита SUID

$ chmod g+s filename – установка бита SGID

Мы с вами рассмотрели понятие процесса, способы отображения процессов и права доступа. Но для комфортной работы в операционной системе этого, согласитесь, мало. Необходимо еще эффективно управлять процессами. А для реализации управления мы сначала рассмотри строение таблицы процессов:

Родителем всех процессов в системе является процесс *init.* Его PID всегда 1, PPID – 0. Всю таблицу процессов можно представить себе в виде дерева, в котором корнем будет процесс *init.* Этот процесс хоть и не является частью ядра, но выполняет в системе очень важную роль – определяет текущий уровень инициализации системы и следит, чтобы были запущены программы, позволяющие пользователю общаться с компьютером (mingetty, X или другие).

Процессы, имена которых заключены в квадратные скобки, например *"[keventd]"* – это процессы ядра. Эти процессы управляют работой системы, а точнее такими ее частями, как менеджер памяти, планировщик времени процессора, менеджеры внешних устройств и так далее.

Остальные процессы являются пользовательскими, запущенными либо из командной строки, либо во время инициализации системы.

Жизнь каждого процесса представлена следующими фазами:

Создание процесса – на этом этапе создается полная копия того процесса, который создает новый. Например, вы запустили из интерпретатора на выполнение команду ls. Командный интерпретатор создает свою полную копию.

Загрузка кода процесса и подготовка к запуску – копия, созданная на первом этапе заменяется кодом задачи, которую необходимо выполнить и создается ее окружение – устанавливаются необходимые переменные и т.п.

Выполнение процесса

Состояние зомби – на этом этапе выполнение процесса закончилось, его код выгружается из памяти, окружение уничтожается, но запись в таблице процессов еще остается.

Умирание процесса – после всех завершающих стадий удаляется запись из таблицы процессов – процесс завершил свою работу.

Во время работы процесса, ядро контролирует его состояние, и в случае возникновения непредвиденной ситуации управляет процессом с помощью посылки ему сигнала. **Сигнал** – это простейший способ межпроцессорного (то есть между процессами) взаимодействия. Существует несколько типов сигналов. Для каждого из типов предусмотрено действие по умолчанию. Процесс может воспользоваться действием по умолчанию, или, если у него есть обработчик сигнала, то он может перехватить и обработать или игнорировать сигнал. Сигналы **SIGKILL и SIGSTOP** невозможно ни перехватить, ни игнорировать.

По умолчанию возможны несколько действий:

Игнорировать – продолжать работу, несмотря на то, что получен сигнал.

Завершить – завершить работу процесса.

завершить + core – завершить работу процесса и создать файл в текущем каталоге с именем core, содержащий образ памяти процесса (код и данные).

Остановить – приостановить выполнение процесса, но не завершать его работу и не выгружать код из памяти.

Вот список всех сигналов, существующих в системе на сегодняшний день:

Название	Действие по умолчанию	Значение
SIGABRT	Завершить + core	Сигнал отправляется, если процесс вызывает системный вызов abort()

Название	Действие по умолчанию	Значение
SIGTERM	Завершить	Сигнал обычно представляет своего рода предупреждение, что процесс вскоре будет уничтожен. Этот сигнал позволяет процессу соответствующим образом “подготовиться к смерти” – удалить временные файлы, завершить необходимые транзакции и т.д. Команда kill по умолчанию отправляет именно этот сигнал.
SIGTTIN	Остановить	Сигнал генерируется ядром (драйвером управляющего терминала) при попытке процесса фоновой группы осуществить чтение с управляющего терминала.
SIGTTOU	Остановить	Сигнал генерируется ядром (драйвером терминала) при попытке процесса фоновой группы осуществить запись на управляющий терминал.
SIGALRM	Завершить	Сигнал отправляется, когда срабатывает таймер, ранее установленный.
SIGBUS	Завершить + core	Сигнал свидетельствует о некоторой аппаратной ошибке. Обычно этот сигнал отправляется при обращении к недопустимому виртуальному адресу, для которого отсутствует соответствующая физическая страница.
SIGCHLD	Игнорировать	Сигнал, посылаемый родительскому процессу при завершении его потомка.
SIGSEGV	Завершить + core	Сигнал свидетельствует об обращении процесса к недопустимому адресу или области памяти, для которой у процесса недостаточно привилегий доступа.
SIGFPE	Завершить + core	Сигнал свидетельствует о возникновении особых ситуаций, таких как деление на 0 или переполнение операции с плавающей точкой.
SIGHUP	Завершить	Сигнал посылается лидеру сеанса, связанному с управляющим терминалом, что терминал отсоединился (потеря линии). Сигнал также посылается всем процессам текущей группы при завершении выполнения лидера.

Название	Действие по умолчанию	Значение
		Этот сигнал иногда используют в качестве простейшего средства межпроцессного взаимодействия. В частности, он применяется для сообщения демонам о необходимости обновить конфигурационную информацию. Причина выбора именно сигнала SIGHUP заключается в том, что демон по определению не имеет управляющего терминала и, соответственно, обычно не получает этого сигнала.
SIGILL	Завершить + core	Сигнал посылается ядром, если процесс попытается выполнить недопустимую инструкцию.
SIGINT	Завершить	Сигнал посылается ядром всем процессам при нажатии клавиши прерывания (<CTRL>+<C>)
SIGKILL	Завершить	Сигнал, при получении которого выполнение процесса прекращается. Этот сигнал нельзя не перехватить, не проигнорировать.
SIGPIPE	Завершить	Сигнал посылается при попытке записи в сокет, получатель данных которого завершил выполнение или закрыл файловый указатель на сокет.
SIGPOLL	Завершить	Сигнал отправляется при наступлении определенного события для устройства, которое является опрашиваемым (например, получен пакет по сети)
SIGPWR	Игнорировать	Сигнал генерируется при угрозе потери питания. Обычно он отправляется, когда питание системы переключается на источник бесперебойного питания (UPS).
SIGQUIT	Завершить	Сигнал посылается всем процессам текущей группы при нажатии клавиш <CTRL>+<\>.
SIGSTOP	Остановить	Сигнал отправляется всем процессам текущей группы при нажатии пользователем клавиш <CTRL>+<Z>. Получение сигнала вызывает останов выполнения процесса.
SIGSYS	Завершить + core	Сигнал отправляется ядром при попытке осуществления процессом недопустимого системного

Название	Действие по умолчанию	Значение
		вызова.
SIGUSR1	Завершить	Сигнал предназначен для прикладных задач как простейшее средство межпроцессного взаимодействия.
SIGUSR2	Завершить	Сигнал предназначен для прикладных задач как простейшее средство межпроцессного взаимодействия.

Немаловажную роль в жизни процессов играет также ***планировщик*** – это часть ядра, ответственная за многозадачность системы. Ведь в единицу времени на одном процессоре может выполняться только одна задача. Именно планировщик определяет, какой из запущенных процессов первым будет выполняться, какой вторым. Для этого у каждого процесса существует еще один параметр, называемый **приоритетом.** Для того, чтобы посмотреть приоритет процессов, нам необходимо использовать уже знакомую команду *ps* с параметром *-l* (long – расширенный вывод):

$ ps -l

F	*S*	*UID*	*PID*	*PPID*	*C*	*PRI*	*NI*	*ADDR*	*SZ*	*WCHAN*	*TTY*	*TIME*	*CMD*
0	*S*	*500*	*1554*	*1553*	*0*	*75*	*0*	*–*	*1135*	*wait4*	*pts/1*	*00:00:00*	*bash*
0	*R*	*500*	*1648*	*1554*	*0*	*81*	*0*	*–*	*794*	*–*	*pts/1*	*00:00:00*	*ps*

Во время своей работы, планировщик в первую очередь ставит на выполнение задачи с меньшим приоритетом. Так, приоритетом 0, обладают только критические системный задачи, а отрицательным приоритетом – процессы ядра. Задачам с большим приоритетом достается меньше процессорного времени и потому, работают они как правило, медленнее, и потребляют намного меньше системных ресурсов.

Остается только решить вопрос, а может ли пользователь управлять процессами и системными параметрами? Конечно может! Для этого в Linux есть набор инструментов, позволяющих изменять приоритет процесса, посылать процессам сигналы. О них мы с вами сейчас и поговорим.

Первый инструмент – команда nice:

$ nice -n command – позволяет изменять приоритет, с которым будет выполняться процесс после запуска. Без указания команды *command* выдает текущий приоритет работы. n по умолчанию равен 10. Диапазон приоритетов расположен от -20 (наивысший приоритет) до 19 (наименьший). Пример использования команды *nice*:

$ less .bashrc &

[1] 3070

$ ps -efl | grep less

0 T gserg 3070 3018 0 ***80*** *0 – 1004 finish 17:56 pts/3 00:00:00 less .bashrc*

Сравнивая цифры приоритета, заметим, что команда less в первом случае выполнялась с приоритетом 80, а во втором – 99. Таким образом, команда nice сделала свое дело – понизила приоритет задачи. Нужно учесть только несколько особенностей выполнения команды nice. Во-первых, команда понизит приоритет насколько это возможно (в примере на 19 вместо 20). Во-вторых – повышать приоритет задачи в системе может только суперпользователь.

Еще одна команда:

$ nohup command – позволяет процессу продолжить выполнение даже при потере управляющего терминала (SIGHUP). Эту команду выгодно использовать когда необходимо выполнить команду продолжительного действия. Вы запускаете команду и закрываете терминальный сеанс, а она при этом продолжает выполняться. Вывод команды *nohup* сохранит в файл nohup.out в текущем каталоге.

Самой часто используемой командой управления процессами можно по праву считать команду *kill:*

$kill -SIGNAL pid – посылает сигнал процессу с идентификатором pid. Если сигнал не указан, команда посылает процессу сигнал SIGTERM. Вот пример ее использования:

$ less &

[1] 1352

$ ps

PID TTY TIME CMD

1322 pts/2 00:00:00 bash

1352 pts/2 00:00:00 less

1353 pts/2 00:00:00 ps

$ kill -SIGKILL 1352

$ ps

PID TTY TIME CMD

1322 pts/2 00:00:00 bash

1355 pts/2 00:00:00 ps

[1]+ Killed less

Не менее популярной чем kill командой является *killall:*

$killall -s SIGNAL процесс – посылает сигнал всем процессам с именем процесс. Если сигнал не указан, посылает SIGTERM.

Сигнал для этой команды необходимо указывать без приставки SIG. Для получения соответствия цифрового вида и имени сигнала используется опция *-l* команды *killall*.

Программа top

Предназначена для вывода информации о процессах в реальном времени. Процессы сортируются по максимальному занимаемому процессорному времени, но вы можете изменить порядок сортировки (см. man top). Программа также сообщает о свободных системных ресурсах.

 # top

7:49pm up 5 min, 2 users, load average: 0.03, 0.20, 0.11

56 processes: 55 sleeping, 1 running, 0 zombie, 0 stopped

CPU states: 7.6% user, 9.8% system, 0.0% nice, 82.5% idle

Mem: 130660K av, 94652K used, 36008K free, 0K shrd, 5220K buff

Swap: 72256K av, 0K used, 72256K free 60704K cached

PID USER PRI NI SIZE RSS SHARE STAT %CPU %MEM TIME COMMAND

1067 root 14 0 892 892 680 R 2.8 0.6 0:00 top

1 root 0 0 468 468 404 S 0.0 0.3 0:06 init

2 root 0 0 0 0 0 SW 0.0 0.0 0:00 kflushd

3 root 0 0 0 0 0 SW 0.0 0.0 0:00 kupdate

4 root 0 0 0 0 0 SW 0.0 0.0 0:00 kswapd

5 root -20 -20 0 0 0 SW< 0.0 0.0 0:00 mdrecoveryd

Просмотреть информацию об оперативной памяти вы можете с помощью команды free, а о дисковой – df. Информация о зарегистрированных в системе пользователей доступна по команде w.

Команды выполнения процессов в фоновом режиме – jobs, fg, bg

Команда jobs выводит список процессов, которые выполняются в фоновом режиме, fg – переводит процесс в нормальные режим («на передний план» – foreground), а bg – в фоновый. Запустить программу в фоновом режиме можно с помощью конструкции *команда &*

Перенаправление ввода/вывода

Практически все операционные системы обладают механизмом перенаправления ввода/вывода. Linux не является исключением из этого правила. Обычно программы вводят текстовые данные с консоли (терминала) и выводят данные на консоль. При вводе под консолью подразумевается клавиатура, а при выводе – дисплей терминала. Клавиатура и дисплей – это, соответственно, стандартный ввод и вывод (stdin и

stdout). Любой ввод/вывод можно интерпретировать как ввод из некоторого файла и вывод в файл. Работа с файлами производится через их дескрипторы. Для организации ввода/вывода в UNIX используются три файла: stdin (дескриптор 1), stdout (2) и stderr(3).

Символ > используется для перенаправления стандартного вывода в файл.

Пример:

$ cat > newfile.txt

Стандартный ввод команды cat будет перенаправлен в файл newfile.txt, который будет создан после выполнения этой команды. Если файл с этим именем уже существует, то он будет перезаписан. Нажатие Ctrl + D остановит перенаправление и прерывает выполнение команды cat.

Символ < используется для переназначения стандартного ввода команды. Например, при выполнении команды cat < file.txt в качестве стандартного ввода будет использован файл file.txt, а не клавиатура.

Символ >> используется для присоединения данных в конец файла (append) стандартного вывода команды. Например, в отличие от случая с символом >, выполнение команды cat >> newfile.txt не перезапишет файл в случае его существования, а добавит данные в его конец.

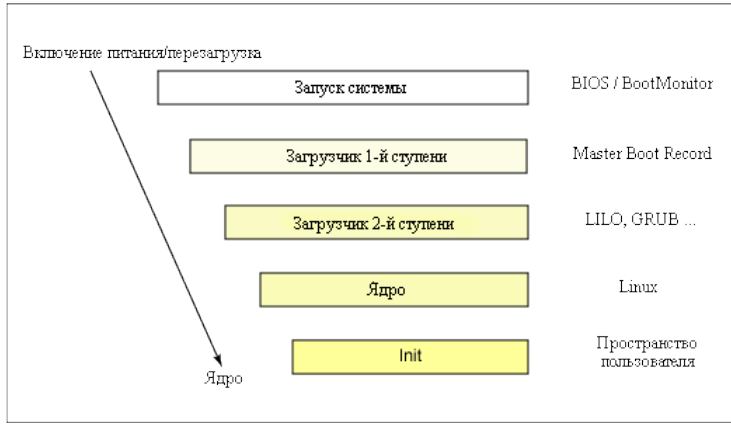
Символ | используется для перенаправления стандартного вывода одной программы на стандартный ввод другой. Напрмер, ps -ax | grep httpd.

Общие сведения о начальной загрузке

Когда-то давно термин bootstrapping (загрузка) в компьютерной области означал загрузку бумажной ленты, на которой хранилась программа начальной загрузки, или же ввод программы начальной загрузки вручную при помощи расположенных на передней панели переключателей адреса/данных/управления. Современные компьютеры оборудованы устройствами, которые значительно упрощают процесс первоначальной загрузки – однако это не означает, что этот процесс является простым.

Давайте сначала бросим самый общий взгляд на процесс начальной загрузки Linux, чтобы охватить картину полностью. Затем мы более подробно рассмотрим, что происходит на каждом этапе процесса. Ссылки на исходный код, которые будут приводиться постоянно в процессе изложения, помогут при изучении дерева исходных кодов ядра и подскажут, где получить дополнительную информацию.

Рисунок 1. Вид на процесс начальной загрузки в Linux



При первоначальной загрузке системы или при ее перезагрузке системы процессор выполняет код, который расположен в хорошо известном месте. В персональном компьютере это место соответствует базовой системе ввода/вывода (BIOS), которая хранится в расположенной на системной плате микросхеме энергонезависимой flash-памяти. Центральный процессор (CPU) встраиваемой системы обращается к reset-вектору для получения адреса программы, которая хранится по известному адресу в flash/ROM-памяти. В любом случае это приводит к одному и тому же результату. Так как персональные компьютеры отличаются намного большей универсальностью, BIOS должна определить, какие именно устройства являются кандидатами на выполнение начальной загрузки. Мы подробно рассмотрим данный процесс позже.

После того, как устройство, с которого будет осуществляться начальная загрузка, найдено, начальный загрузчик первой ступени загружается в оперативную память и начинается его выполнение. Этот начальный загрузчик имеет размер менее 512 байт (один сектор), и его задачей является загрузка начального загрузчика второй ступени.

После того, как в оперативную память загружается и начинает выполняться начальный загрузчик второй ступени, на экране обычно отображается заставка и в память загружаются Linux вместе с необязательным начальным RAM-диском (временная система корневых файлов). После того как эти образы будут загружены, начальный загрузчик второй ступени передает управление ядру и выполняется декомпрессия и инициализация ядра. На этой стадии начальный загрузчик второй ступени проверяет аппаратное обеспечение системы, выполняет нумерацию подключенных устройств, монтирует корневое устройство и затем загружает необходимые модули ядра. После завершения этих задач запускается первая пользовательская программа (init) и затем выполняется инициализация системы высокого уровня.

Таков в общих чертах процесс начальной загрузки в Linux. Теперь давайте углубимся чуть далее и рассмотрим некоторые подробности начальной загрузки в Linux.

Запуск системы

Запуск системы определяется той аппаратной платформой, на которой выполняется начальная загрузка Linux. На встраиваемых платформах при включении питания системы или при перезагрузке используется bootstrap-среда. В качестве примеров можно привести U-Boot, RedBoot и MicroMonitor от компании Lucent. Эти программы хранятся в специальной области flash-памяти, расположенной во встраиваемой системе: они предоставляют средства для загрузки образа ядра Linux во flash-память и в дальнейшем обеспечивают выполнение этого ядра. В дополнение к функциям по хранению и загрузке образа Linux эти мониторы загрузки также выполняют тестирование системы на каком-то уровне и инициализацию аппаратного обеспечения. Во встраиваемых системах подобные мониторы загрузки обычно объединяют функции загрузчиков первой и второй ступени.

Просмотр содержимого MBR

Для просмотра содержимого MBR используйте следующую команду:

 # dd if=/dev/hda of=mbr.bin bs=512 count=1

 # od -xa mbr.bin

Команда *dd*, которая выполняется с правами root, считывает первые 512 байт с */dev/hda* (первый IDE-жесткий диск) и записывает их в файл *mbr.bin*. Команда *od* выполняет печать полученного двоичного файла в форматах hex и ASCII.

На персональных компьютерах загрузка Linux начинается в BIOS с адреса 0xFFFF0. Первым действием, которое выполняет BIOS, является тестирование при включении питания (POST, power-on self test). Задачей такого тестирования является выполнение проверки аппаратного обеспечения. Вторым действием POST является выполнение нумерации и инициализации для локальных устройств.

Если учитывать различное назначение функций BIOS, то можно считать, что BIOS состоит из двух частей: кода POST и сервисов времени выполнения. После завершения POST соответствующий код удаляется из памяти, однако сервисы времени выполнения BIOS остаются в памяти и доступны для операционной системы.

Для выполнения загрузки операционной системы сервисы времени выполнения BIOS выполняют поиск таких устройств, которые являются активными и способны выполнять загрузку – причем поиск выполняется в порядке, который определяется настройками, сохраненными в CMOS-памяти. В качестве загрузочных устройств могут выступать флоппи-диски, CD-ROM, разделы на жестком диске, подключенное к сети устройство или даже портативный USB-накопитель.

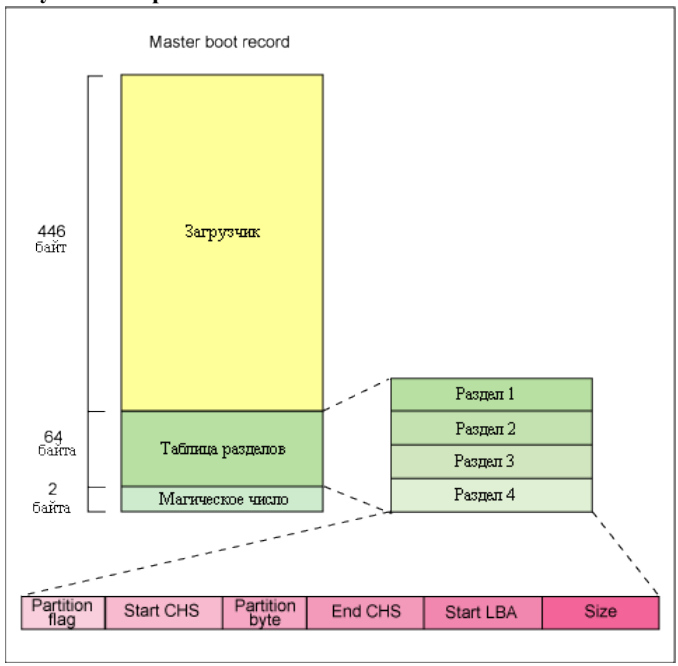
Обычно загрузка Linux производится с жесткого диска, на котором в MBR содержится первичный начальный загрузчик. MBR представляет собой сектор размером 512 байт, который располагается в первом секторе диска (сектор 1 цилиндра 0, головка 0). После того как MBR загружается в память, BIOS передает ему управление.

Загрузчик 1-й ступени

Первичный начальный загрузчик, хранящийся в MBR, представляет собой образ размером 512 байт, который содержит как программный код, так и небольшую

таблицу разделов (см. рисунок 2). Первые 446 байт представляют собой собственно первичный загрузчик, который содержит как программный код, так и текст сообщений об ошибках. Следующие 64 байта представляют собой таблицу разделов, которая содержит запись для каждого из четырех разделов диска (по 16 байт каждая). В конце MBR располагаются два байта, которые носят название "магического числа" (0xAA55). Это магическое число служит для целей проверки MBR.

Рисунок 2. Строение MBR



Задача первичного загрузчика - отыскать и загрузить вторичный загрузчик (загрузчик второй ступени). Он делает это, просматривая таблицу разделов в поиске активного раздела. Когда первичный загрузчик обнаруживает активный раздел, он просматривает оставшиеся разделы с целью убедиться, что они не являются активными. После завершения этой проверки с устройства в оперативную память считывается загрузочная запись активного раздела.

Загрузчик 2-й ступени

Вторичный загрузчик или загрузчик второй ступени было бы более логично назвать загрузчиком ядра. Его задачей на данной стадии является загрузка ядра Linux и, возможно, загрузка начального RAM-диска.

Загрузчики GRUB разных ступеней

Директория /boot/grub содержит загрузчики stage1,stage1.5 и stage2, а также некоторые альтернативные загрузчики (например, CR-ROM использует iso9660_stage_1_5).

Загрузчики для среды x86, которые объединяют в себе загрузчики первой и второй стадии, носят название Linux Loader (LILO) или GRand Unified Bootloader (GRUB). Так как LILO имел некоторые недостатки, которые были исправлены в GRUB, то далее мы будем рассматривать именно GRUB.

Одним из наибольших достоинств GRUB является то, что он способен понимать используемые в Linux файловые системы. Вместо того чтобы подобно LILO, обращаться непосредственно к секторам жесткого диска, загрузчик GRUB способен загружать ядро Linux с файловых систем ext2 или ext3. Это достигается благодаря превращению двухступенчатого загрузчика в трехступенчатый. Ступень 1 (MBR) загружает загрузчик 1.5-ступени, способный понимать файловую систему, в которой хранится образ ядра Linux. Примерами могут служить reiserfs_stage1_5(для загрузки из файловой системы с журналированием Reiser) или e2fs_stage1_5 (для загрузки из файловых систем ext2 или ext3). После того, как загрузчик 1.5 ступени загружен и выполняется, может быть загружен загрузчик 2-й ступени.

После загрузки 2 ступени GRUB способен по запросу показать список имеющихся ядер (которые определяются в/etc/grub.conf, с поддержкой мягких ссылок из /etc/grub/menu.lst и/etc/grub.conf). Вы можете выбрать нужное ядро и даже передать ему дополнительные параметры ядра. Также существует возможность воспользоваться оболочкой с поддержкой командной строки, что обеспечивает большую степень контроля над процессом загрузки.

После того как загрузчик второй стадии загружен в память, он обращается к файловой системе и выполняет загрузку в память установленного по умолчанию образа ядра и образainitrd. Когда эти образы готовы к работе, загрузчик 2-й стадии вызывает образ ядра.

Ядро

Ручная загрузка в GRUB

Из командной строки в GRUB можно загрузить нужное ядро с указанным образом initrd следующим образом:

grub> ***kernel /bzImage-2.6.14.2***
  [Linux-bzImage, setup=0x1400, size=0x29672e]

grub> ***initrd /initrd-2.6.14.2.img***
  [Linux-initrd @ 0x5f13000, 0xcc199 bytes]

grub> ***boot***

Uncompressing Linux... Ok, booting the kernel.

Если известно название ядра, которое вы хотите загрузить, то просто введите символ прямого слэша (/) и затем нажмите клавишу Tab. После этого GRUB отобразит список ядер и образов initrd.

После того как образ ядра оказывается в памяти и ему передается управление от загрузчика 2-й ступени, наступает стадия ядра. Однако образ ядра не является исполняемым, это сжатый образ ядра. Обычно это zImage (сжатый образ размером менее 512KB) или bzImage (большой сжатый образ, размером более 512KB), который был сжат при помощи zlib. В начале такого образа ядра располагается программа, которая выполняет минимальную настройку аппаратного обеспечения и затем распаковывает ядро, хранящееся внутри образа ядра, и помещает его в верхнюю область памяти. Если имеется образ начального RAM-диска, то программа также перемещает его в память и помечает для дальнейшего использования, а затем вызывает само ядро, после чего начинается загрузка ядра.

Функция initrd позволяет создать компактное ядро Linux, где драйверы скомпилированы как загружаемые модули. Эти загружаемые модули обеспечивают доступ ядра к дискам и файловым системам, которые имеются на этих дисках. Также имеются драйверы для других аппаратных устройств. Так как корневая файловая система представляет собой *файловую систему* на диске, то функция initrd обеспечивает для загрузчика возможность обратиться к диску и смонтировать действительную корневую файловую систему. Во встраиваемой системе без жесткого диска initrd может представлять собой окончательную файловую систему, или же окончательная файловая система может монтироваться при помощи сетевой файловой системы (Network File System, NFS).

Init

После загрузки и инициализации ядра запускается первое приложение в пространстве пользователя. Это первая из вызываемых программ, которые скомпилированы со стандартной библиотекой C. До этого момента процесса стандартные C-приложения еще не выполнялись.

На обычных настольных системах с операционной системой Linux обычно первым запускаемым приложением является /sbin/init. Однако это обязательно. Во встраиваемых системах редко требуется такая обширная инициализация, которую обеспечивает init (которая конфигурируется при помощи /etc/inittab). Во многих случаях можно запустить простой shell-скрипт, который запускает необходимые встраиваемые приложения.

Заключение

Как и сама операционная система Linux, процесс загрузки ядра является чрезвычайно гибким и универсальным и поддерживает большое количество процессоров и аппаратных платформ. В самом начале загрузчик loadlin обеспечивал простой способ загрузки Linux без поддержки каких-либо необязательных аксессуаров. Загрузчик LILO расширил круг поддерживаемых функций, однако его недостатком являлось отсутствие поддержки файловых систем. Последнее поколение загрузчиков, таких как

GRUB, позволяет загружать Linux с самых различных файловых систем (начиная с Minix и заканчивая Reiser).

Список контрольных вопросов

1. Дайте определение процесса и перечислите его состояния
2. Изменение прав доступа процессов
3. Типы межпроцессорных сигналов
4. Что такое планировщик задач?
5. Какие основные команды управления процессами Вы знаете?
6. Порядок первоначальной загрузки в Linux
7. Загрузчики LILO и GRUB
8. Init – первое пользовательское приложение

Задания по лабораторной работе

1. Отобразить процессы, связанные с данными идентификаторыми групп
2. Определите приоритет процесса *bash*
3. Измените приоритет процесса *nano*
4. Убейте процесс редактирования документа
5. Назовите текущий процесс, потребляющий наибольшее количество оперативной памяти
6. Запустите любой процесс в фоновом режиме
7. Осуществите процесс ручной загрузки GRUB

Список литературы

1. *Уэлш. М. и~др.,* Руководство по установке и использованию системы **Linux.** М.: ИЛКиРЛ, 1999

2. Руководство администратора Linux. Установка и настройка. 2-е издание, Эви Немет, Гарт Снайдер, Трент Хейн

3. Linux. Библия пользователя, Кристофер Негус

4. Linux для чайников , 6-е издание, Ди-Анн Лебланк

5. Разработка ядра Linux, 2-е издание, Роберт Лав

6. Библиотека Qt 4. Программирование прикладных приложений в среде Linux., Чеботарев Арсений Викторович

7. Red Hat Linux Fedora 4. Полное руководство, Пол Хадсон, Эндрю Хадсон, Билл Болл, Хойт Дафф

8. Искусство программирования для Unix, Эрик С. Реймонд

9. Linux для "чайников", 5-е издание, Ди-Анн Лебланк

10. Red Hat Linux. Секреты профессионала, Наба Баркакати

11. Использование Linux, Apache, MySQL и PHP для разработки Web-приложений, Джеймс Ли, Брент Уэр

12. Секреты хакеров. Безопасность сетей - готовые решения, 4-е издание, Стюарт Мак-Клар, Джоэл Скембрей, Джордж Курц

13. FreeBSD: полный справочник., Родерик Смит

14. Секреты хакеров. Безопасность Linux — готовые решения, 2-е издание, Брайан Хатч, Джеймс Ли, Джордж Курц

15. Red Hat Linux 8. Библия пользователя, Кристофер Негус

16. Серверы Linux. Самоучитель, Птицын Константин Александрович

17. Безопасность Linux, 2-е издание, Скотт Манн, Эллен Л. Митчелл, Митчелл Крелл

18. Сетевые средства Linux, Родерик Смит

19. Руководство администратора Linux, Эви Немет, Гарт Снайдер, Трент Хейн

20. Сети TCP/IP, том 3. Разработка приложений типа клиент/сервер для Linux/POSIX, Дуглас Камер, Дэвид Л. Стивенс

21. Секреты хакеров. Безопасность Linux — готовые решения, Брайан Хатч, Джеймс Ли, Джордж Курц

22. Программирование для Linux. Профессиональный подход, Марк Митчелл, Джеффри Оулдем, Алекс Самьюэл

23. Использование Linux, 6-е издание. Специальное издание, Дэвид Бендел, Роберт Нейпир

24. Создание сетевых приложений в среде Linux, Шон Уолтон

25. Освой самостоятельно Linux за 24 часа, 3-е издание,

26. Система электронной почты на основе Linux. Руководство администратора, Ричард Блам

27. Системное администрирование Linux, М. Карлинг, Стефан Деглер, Джеймс Деннис

28. Робачевский А.М. «Операционная система Unix®». – СПб.:БВХ – Санкт-Петербург, 1999. – 528 с., ил.

Учебное издание

Загрузка процессов в Linux

Методические указания

Составитель: Сухов Андрей Михайлович

Изд-во Самарского государственного
аэрокосмического университета.
443086 Самара, Московское шоссе, 34.