Впервые опубликовано на developerWorks 29.08.2005

Обновлено 29.09.2005

Экзамен LPI 201: Ядро Linux

Администрирование, средний уровень (LPIC-2) тема 201

Компоненты ядра

Из чего состоит ядро?

В ядро Linux входит базовое ядро как таковое плюс некоторое количество модулей ядра. В большинстве случаев базовое ядро и большая коллекция модулей ядра, компилируемые одновременно и устанавливаемые или распространяемые вместе, основаны на коде, созданном Линусом Торвальдсом или измененном производителями дистрибутивов Linux. Базовое ядро всегда загружается в ходе загрузки системы и остается загруженным во время работы постоянно. Модули ядра первоначально могут быть загружены, а могут нет (хотя как правило часть из них загружена) и могут подгружаться или выгружаться во время работы.

Модульная структура ядра позволяет подключать дополнительные модули, скомпилированные позднее или отдельно от базового ядра. Дополнительные модули могут создаваться, когда вы добавляете оборудование к уже работающей системе Linux, а иногда могут поставляться третьими лицами. Модули ядра иногда распространяются в виде бинарных файлов, в результате чего ваши способности, как системного администратора, настраивать модули ядра оказываются не востребованы. В любом случае, загруженный модуль становится частью работающего ядра до тех пор, пока он не будет выгружен. Вопреки некоторым представлениям, модуль ядра не просто является интерфейсом прикладного программирования (API) для общения с базовым ядром, а становится частью работающего ядра.

Соглашения о наименовании ядра

Ядра Linux следуют соглашениям о наименовании/нумерации, что позволяет быстро получить важную информацию о загруженном ядре. В соглашении определены обозначения для `major` номера, `minor` номера, редакция и в некоторых случаях включена строка, описывающая производителя/настройки. Эти соглашения применяются для нескольких типов файлов, в том числе к архивам исходников ядра, патчам и, возможно, нескольким базовым ядрам (если вы запускаете то одно, то другое ядро).

Как и обычная разделенная точками последовательность, ядро Linux следует соглашению по разделению стабильной и экспериментальной веток. Для стабильных веток используется четный `minor` номер, тогда как для экспериментальных веток -- нечетный `minor` номер. Редакция -- просто последовательная нумерация, отражающая исправления ошибок и перенос нововведений в старые версии ядра. Кроме того, номер часто характеризует производителя или специальные возможности. Например:

- `linux-2.4.37-foo.tar.gz`: обозначает архив исходников для стабильной ветки ядра 2.4 от компании "Foo Industries"
- `/boot/bzImage-2.7.5-smp`: обозначает собранное экспериментальное базовое ядро 2.7 с возможностью поддержки SMP
- `patch-2.6.21.bz2`: обозначает патч для обновления более ранней стабильной версии 2.6 до редакции 21

Файлы ядра

Базовое ядро Linux может быть двух версий: `zImage`, ограниченной 508 Кбайт, и `bzImage` для более крупных ядер (приблизительно до 2.5 Мбайт). Как правило, современные дистрибутивы Linux используют ядро формата `bzImage`, что позволяет включать множество компонент. Вы можете предположить, что так как "z" в `zImage` означает сжатие с помощью `gzip`, то "bz" в `bzImage` может означать сжатие с помощью `bzip2`. Однако, "b" просто обозначает "big", а для сжатия по-прежнему используется `gzip`. В обоих случаях в каталоге `/boot/` базовое ядро часто переименовывается в `vmlinuz`. Как правило файл `/vmlinuz` является символьной ссылкой на файл с полным именем ядра, включающим номер версии, например, `/boot/vmlinuz-2.6.10-5-386`.

В каталоге `/boot/` есть несколько файлов, связанных с базовым ядром, которые вам уже знакомы (иногда они могут располагаться вместо этого в корневом каталоге файловой системы). `System.map` -- это таблица, отображающая адреса символов ядра. `initrd.img` иногда используется базовым ядром для создания упрощенной файловой системы на ram-диске, подключаемом на этапе загрузки для монтирования основной файловой системы.

Модули ядра

Модули ядра содержат дополнительный код ядра, который может быть загружен после базового ядра. Модули обычно предоставляют одну или несколько функций:

- **Драйверы устройств (Device drivers)**: поддержка специфических типов оборудования
- **Драйверы файловой системы (File system drivers)**: предоставляют необязательную возможность чтения и/или записи специфической файловой системы
- **Системные вызовы (System calls)**: большинство поддерживается базовым ядром, но модули могут добавлять или изменять системные службы
- **Сетевые драйверы (Network drivers)**: реализуют соответствующие сетевые протоколы
- **Загрузчики исполняемых файлов (Executable loaders)**: анализируют и загружают исполняемые дополнительные форматы

Компиляция ядра

Получение исходников ядра

Первое, что нужно сделать, чтобы скомпилировать новое ядро, -- получить его исходные коды. Основное хранилище исходников ядра -- Linux Kernel Archives (kernel.org; см. Ресурсы). Производитель вашего дистрибутива мог включить в него обновленные исходники ядра, чтобы показать внесенные изменения. Например, вы можете получить и распаковать версию ядра с командами, подобными следующим:

Листинг 1. Получение и распаковка ядра

```
   % cd /tmp/src/
   % wget http://www.kernel.org/pub/linux/kernel/v2.6/linux-
2.6.12.tar.bz2
   % cd /usr/src/
   % tar jxvfy /tmp/src/linux-2.6.12.tar.bz2
```

Для распаковывания исходников ядра а каталог `/usr/src/` вам понадобятся права root'a. Однако, вы можете распаковать и собрать ядро в своем домашнем каталоге. Поищите на kernel.org другие форматы архивов и протоколов загрузки.

Проверка исходников ядра

Если вы благополучно получили и распаковали архив с исходниками ядра, в вашей системе должен появиться каталог `/usr/src/linux-2.6.12` (или, если вы распаковывали архив в другом месте, каталог с похожим названием). Важно, что тот каталог должен содержать файл README с текущей информацией. В этом каталоге содержатся подкаталоги с исходными файлами, в основном это файлы `.c` и `.h`. Главные действия по объединению этих файлов в работающее ядро прописаны в файле `Makefile`, который использует утилита `make`.

Конфигурирование сборки

После получения и распаковывания исходников ядра у вас может появиться желание сконфигурировать ядро. Команда `make` имеет три опции для настройки опций ядра: `config`, `menuconfig` и `xconfig`. Вообще, можно отредактировать файл `.config`, но делать это нежелательно (вы откажетесь от дополнительной информации и можете легко создать нерабочую конфигурацию).

Командой `make config` воспользоваться также непросто, как и отредактировать файл `.config` вручную; вам придется настраивать каждую опцию (их сотни) в определенном порядке, без возможности возврата к предыдущему действию. Команда `make menuconfig` предоставляет `curses` интерфейс, где вы можете выбрать только те опции, которые необходимо изменить. Команда `make xconfig` предоставляет симпатичный графический интерфейс (особенно красивый в Linux 2.6+).

Для значительной части опций ядра возможны три варианта выбора: (1) включить компоненту в базовое ядро (*include the capability in the base kernel*) ; (2) включить как модуль ядра (*include it as a kernel module*); (3) совсем не включать компоненту (*omit the capability entirely*). Вообще, в создании многочисленных модулей ядра нет ничего страшного (кроме незначительного увеличения времени компиляции), поскольку пока они не нужны, они и не загружены. Если дисковое пространство ограничено, можно не включать никаких возможностей.

Запуск процесса сборки

Теперь, чтобы собрать базовое ядро с выбранными опциями, следует выполнить следующие действия:

- `make dep`: необходимо только для ядра 2.4, для 2.6 не требуется.
- `make clean`: очистить предыдущие объектные файлы, это особенно полезно, если вы собираете данное ядро не первый раз.
- `make bzImage`: создать базовое ядро. В особых случаях для небольших образов ядра можно использовать `make zImage`. Вы также можете воспользоваться командой `make zlilo`, чтобы установить ядро прямо в загрузчик lilo, или командой `make zdisk`, чтобы создать загрузочную дискету. Вообще, лучше создавать образ ядра в каталоге типа `/usr/src/linux/arch/i386/boot/vmlinuz`, используя команду `make bzImage`, и затем копировать его оттуда вручную.
- `make modules`: создать все сконфигурированные загружаемые модули ядра.
- `sudo make modules_install`: установить все собранные модули в каталог `/lib/modules/2.6.12/`, название подкаталога совпадает с номером версии ядра.

Создание стартового ram-диска

Если вы создали важный загрузочный драйвер, стартовый ram-диск позволит загрузить его в процессе начальной загрузки. Это касается главным образом тех драйверов файловой системы, которые были собраны в виде модулей ядра. По существу, стартовый ram-диск -- некий магический корневой псевдо-раздел, который живет в памяти и позже выполняет `chroot` на реальный раздел диска (например, если ваш корневой раздел расположен на RAID). Более подробное описание вы найдете в следующих учебных пособиях этой серии.

Создание стартового ram-диска осуществляется при помощи команды `mkinitrd`. Чтобы узнать, какие опции имеет команда `mkinitrd`, включенная в ваш дистрибутив Linux, обратитесь к странице `man` этой команды. Самое простое -- запустить команду, подобную следующей:

Листинг 2. Создание ram-диска

```
% mkinitrd /boot/initrd-2.6.12 2.6.12
```

Инсталляция собранного ядра Linux

Успешно собрав базовое ядро и связанные с ним модули (это займет какое-то время -- на медленных машинах до нескольких часов), вы должны скопировать образы ядра (`vmlinuz` или `bzImage`) в свой каталог `/boot/`.

После того как вы скопировали необходимые файлы в `/boot/` и установили модули ядра при помощи `make modules_install`, необходимо сконфигурировать загрузчик, обычно это `lilo` или `grub`, для доступа к соответствующему ядру (ядрам). Информацию о конфигурировании `lilo` и `grub` вы найдете в следующем учебном пособии этой серии.

Дополнительная информация

На сайте kernel.org есть много полезных ссылок, по которым можно получить дополнительную информацию о компонентах ядра и требованиях для сборки. Чрезвычайно полезная и подробная информация содержится в руководстве **Kernel Rebuild Guide** Квана Лоу (Kwan Lowe).

Приложение патчей к ядру

Получение патчей

Исходники ядра Linux распространяются в виде дерева основных исходников в сочетании с множеством небольших патчей. Обычно это позволяет получить самое свежее ядро через максимально быстрые каналы. Это соглашение позволяет прикладывать специальные патчи, полученные не с kernel.org, а из других источников.

Если вы хотите применить несколько уровней изменений, вам необходимо получить всю серию патчей последовательно (по возрастанию). Например, предположим, что к моменту чтения этого пособия доступно ядро 2.6.14 и вы загрузили ядро 2.6.12. Вы должны сделать следующее:

Листинг 3. Последовательное получение патчей

```
% wget http://www.kernel.org/pub/linux/kernel/v2.6/patch-2.6.13.bz2
% wget http://www.kernel.org/pub/linux/kernel/v2.6/patch-2.6.14.bz2
```

Распаковывание и применение патчей

Чтобы применить патч, необходимо сначала распаковать архив при помощи `bzip2` или `gzip`, в зависимости от формата сжатия архива, а затем приложить патч. Например:

Листинг 4. Распаковывание и применение патчей

```
% bzip2 -d patch2.6.13.bz2
% bzip2 -d patch2.6.14.bz2
% cd /usr/src/linux-2.6.12
% patch -p1 < /path/to/patch2.6.13
% patch -p1 < /path/to/patch2.6.14
```

Применив патчи, продолжите компиляцию, как описано в предыдущем разделе. Команда `make clean` удалит дополнительные объектные файлы, которые, возможно, не соответствуют новым изменениям.

Настройка ядра

О настройке

Настройка ядра описана в разделе этого пособия, рассказывающем о сборке ядра (точнее, в опциях `make [x|menu]config`). Когда базовое ядро и его модули собраны, вы можете включить или отменить возможности ядра в порядке подключения дополнительных возможностей, запуска различных профилей и подключения памяти.

В этом разделе рассматриваются способы изменения поведения ядра в ходе работы системы.

Поиск информации о загруженном ядре

Linux (и другие UNIX-подобные операционные системы) использует специальные, как правило совместимые способы хранения информации о загруженном ядре (или других запущенных процессах). Специальный каталог `/proc/` содержит псевдо-файлы и подкаталоги, содержащие массу информации о загруженной системе.

В ходе работы системы Linux каждый процесс создает подкаталог со своим номером, каждый из которых содержит несколько статусных файлов. Здесь хранятся сводные данные о командах пользовательских уровней и системных средствах, но основная часть данных расположена в каталоге файловой системы `/proc/`.

Специфические данные, касающиеся статуса самого ядра, находятся в каталоге `/proc/sys/kernel`.

Подробнее о текущих процессах

Хотя статус процессов, особенно пользовательских, не имеет отношения к ядру как таковому, важно иметь о них представление, если вы намерены заниматься отладкой основного ядра. Простейший способ получить сводку процессов -- выполнить команду `ps` (также существуют графические средства). Зная ID процесса, вы можете исследовать запущенный процесс. Например:

Листинг 5. Исследование запущенного процесса

```
% ps
```

```
  PID TTY          TIME CMD
16961 pts/2    00:00:00 bash
17239 pts/2    00:00:00 ps
% ls /proc/16961
binfmt   cwd@     exe@  maps  mounts  stat   status
cmdline  environ  fd/   mem   root@   statm
```

В этом пособии не будет исследоваться вся информация, содержащаяся в псевдо-файлах процессов. Для примера приведен фрагмент файла `status`:

Листинг 6. Фрагмент псевдо-файла status

```
$ head -12 /proc/17268/status
Name:   bash
State:  S (sleeping)
Tgid:   17268
Pid:    17268
PPid:   17266
TracerPid:      0
Uid:    0       0       0       0
Gid:    0       0       0       0
FDSize: 256
Groups: 0
VmSize:     2640 kB
VmLck:         0 kB
```

Процессы ядра

Каталог `/proc/` наряду с данными о пользовательских процессах содержит полезную информацию о загруженном ядре. Особенно важен каталог `/proc/sys/kernel/`:

Листинг 7. Каталог /proc/sys/kernel/

```
% ls /proc/sys/kernel/
acct           domainname  msgmni       printk         shmall
threads-max
cad_pid        hostname    osrelease    random/        shmmax
version
cap-bound      hotplug     ostype       real-root-dev  shmmni
core_pattern   modprobe    overflowgid  rtsig-max      swsusp
core_uses_pid  msgmax      overflowuid  rtsig-nr       sysrq
ctrl-alt-del   msgmnb      panic        sem            tainted
```

Содержимое этих псевдо-файлов отображает информацию о загруженном ядре. Например:

Листинг 8. Просмотр псевдо-файла ostype

```
% cat /proc/sys/kernel/ostype
Linux
% cat /proc/sys/kernel/threads-max
```

```
4095
```

Уже загруженные модули ядра

Как и другая информация о запущенной системе Linux, данные о загруженном ядре хранятся в каталоге файловой системы `/proc/`, точнее в каталоге `/proc/modules`. Однако, как правило, доступ к этим данным можно получить при помощи утилиты `lsmod` (которая просто показывает заголовки необработанного содержимого файла `/proc/modules`); команда `cat /proc/modules` выведет такую же информацию. Рассмотрим пример:

Листинг 9. Содержимое файла /proc/modules

```
% lsmod
Module                  Size  Used by    Not tainted
lp                      8096   0
parport_pc             25096   1
parport                34176   1  [lp parport_pc]
sg                     34636   0  (autoclean) (unused)
st                     29488   0  (autoclean) (unused)
sr_mod                 16920   0  (autoclean) (unused)
sd_mod                 13100   0  (autoclean) (unused)
scsi_mod              103284   4  (autoclean) [sg st sr_mod sd_mod]
ide-cd                 33856   0  (autoclean)
cdrom                  31648   0  (autoclean) [sr_mod ide-cd]
nfsd                   74256   8  (autoclean)
af_packet              14952   1  (autoclean)
ip_vs                  83192   0  (autoclean)
floppy                 55132   0
8139too                17160   1  (autoclean)
mii                     3832   0  (autoclean) [8139too]
supermount             15296   2  (autoclean)
usb-uhci               24652   0  (unused)
usbcore                72992   1  [usb-uhci]
rtc                     8060   0  (autoclean)
ext3                   59916   2
jbd                    38972   2  [ext3]
```

Загрузка дополнительных модулей ядра

Для загрузки модулей ядра есть два инструмента. Команда `modprobe` -- немного более высокого уровня. Она регулирует загрузку взаимозависимых модулей, то есть при загрузке модулей ядра загружаются и те, модули, от которых они зависят. Однако, `modprobe` на самом деле только надстройка над `insmod`.

Например, предположим, вы хотите загрузить в ядро возможность поддержки *Reiser file system* (надеюсь, она еще не встроена в ядро). Вы можете использовать опцию `modprobe -nv`, чтобы просто посмотреть, что сделала бы команда, но на самом деле ничего не загружать:

Листинг 10. Проверка зависимостей при помощи modprobe

```
%  modprobe -nv reiserfs
/sbin/insmod /lib/modules/2.4.21-
```

```
0.13mdk/kernel/fs/reiserfs/reiserfs.o.gz
```

В нашем случае зависимостей нет. В других случаях они могут быть (и их придется урегулировать вручную, если использовать команду `modprobe` без опции `-n`). Пример:

Листинг 11. Еще один вывод команды modprobe

```
% modprobe -nv snd-emux-synth
/sbin/insmod /lib/modules/2.4.21-0.13mdk/kernel/drivers/sound/
   soundcore.o.gz
/sbin/insmod /lib/modules/2.4.21-0.13mdk/kernel/sound/core/
   snd.o.gz
/sbin/insmod /lib/modules/2.4.21-0.13mdk/kernel/sound/synth/
   snd-util-mem.o.gz
/sbin/insmod /lib/modules/2.4.21-0.13mdk/kernel/sound/core/seq/
   snd-seq-device.o.gz
/sbin/insmod /lib/modules/2.4.21-0.13mdk/kernel/sound/core/
   snd-timer.o.gz
/sbin/insmod /lib/modules/2.4.21-0.13mdk/kernel/sound/core/seq/
   snd-seq.o.gz
/sbin/insmod /lib/modules/2.4.21-0.13mdk/kernel/sound/core/seq/
   snd-seq-midi-event.o.gz
/sbin/insmod /lib/modules/2.4.21-0.13mdk/kernel/sound/core/
   snd-rawmidi.o.gz
/sbin/insmod /lib/modules/2.4.21-0.13mdk/kernel/sound/core/seq/
   snd-seq-virmidi.o.gz
/sbin/insmod /lib/modules/2.4.21-0.13mdk/kernel/sound/core/seq/
   snd-seq-midi-emul.o.gz
/sbin/insmod /lib/modules/2.4.21-0.13mdk/kernel/sound/synth/emux/
   snd-emux-synth.o.gz
```

Предположим, вы хотите загрузить модуль ядра сейчас. Вы можете воспользоваться командой `modprobe`, которая попутно загрузит и все зависимости, но чтобы увидеть все в подробностях, используйте команду `insmod`.

На основании вышеизложенного вы могли сделать предположение, что надо запустить, например, `insmod snd-emux-synth`. Но если сделать это без предварительной загрузки зависимостей, вы получите сообщение о "неразрешенных символах" ("unresolved symbols"). Давайте попробуем вместо этого использовать Reiser file system:

Листинг 12. Загрузка модуля ядра

```
% insmod reiserfs
Using /lib/modules/2.4.21-0.13mdk/kernel/fs/reiserfs/reiserfs.o.gz
```

Довольно успешно, ваше ядро теперь поддерживает новую файловую систему. Вы можете монтировать разделы, читать их, производить в них запись и так далее. Другие возможности системы могут быть подключены подобным образом.

Удаление загруженных модулей ядра

Выгрузка модулей, как и их загрузка, может быть выполнена при помощи высокоуровневой команды `modprobe` или низкоуровневой `rmmod`. Инструмент высокого уровня осуществляет выгрузку в обратном порядке. Команда `rmmod` просто удаляет отдельный модуль ядра, но может не справиться с этой задачей, если модуль используется (обычно из-за зависимостей). Например:

Листинг 13. Попытка выгрузки модулей, имеющих зависимости

```
% modprobe snd-emux-synth
% rmmod soundcore
soundcore: Device or resource busy
% modprobe -rv snd-emux-synth
# delete snd-emux-synth
# delete snd-seq-midi-emul
# delete snd-seq-virmidi
# delete snd-rawmidi
# delete snd-seq-midi-event
# delete snd-seq
# delete snd-timer
# delete snd-seq-device
# delete snd-util-mem
# delete snd
# delete soundcore
```

Однако, если ничто не препятствует удалению модуля, команда `rmmod` выгрузит его из памяти. Например:

Листинг 14. Выгрузка модулей, не имеющих зависимостей

```
% rmmod -v reiserfs
Checking reiserfs for persistent data
```

Автоматическая загрузка модулей ядра

При желании вы можете заставить модули ядра загружаться автоматически, воспользовавшись загрузчиком модулей ядра, входящим в последние версии Linux, или демоном `kerneld` в более старых версиях. Если вы используете эти приемы, в случае, если ядро обнаружит, что к нему обращаются через неподдерживаемые специфические системные вызовы, оно попытается загрузить соответствующий модуль ядра.

Однако, за исключением систем с недостаточным количеством памяти, обычно нет причин отказываться от загрузки необходимых модулей ядра в процессе загрузки системы (подробнее об этом рассказывается в следующем учебном пособии). Загрузчик модулей ядра включен в некоторые дистрибутивы.

Автоматическая выгрузка модулей ядра

Наряду с автоматической загрузкой возможна и автоматическая выгрузка модулей, в основном в системах с недостаточным количеством памяти, например, встроенных системах Linux. Однако, следует знать, что модули ядра могут быть загружены с опцией `insmod --autoclean`, которая помечает их как незагруженные, если они не используются в данное время.

Раньше демон `kerneld` периодически вызывал команду `rmmod --all` для удаления неиспользуемых модулей ядра. В особых случаях (если не используется демон `kerneld`, не входящий в свежие системы Linux), можно добавить в `crontab` команду `rmmod --all`, запускаемую примерно раз в минуту. Но обычно это бывает лишним, поскольку модули ядра как правило используют намного меньше памяти, чем типичные пользовательские процессы.