Урок 7

Работа в Linux

Особенности загрузки Linux. Пространства ядра и пользовательское пространство. Потоки ядра. Программы и пользователи. Виртуальная файловая система. X11-сервер. Различия дистрибутивов GNU/Linux.

Введение

Изначально Linux – это ядро операционной системы, созданной Линусом Торвальдсом, которое было использовано проектом GNU. Операционная система получила название GNU/Linux. Ричард Столлман, один из основателей проекта GNU, настаивает, что операционная система должна называться не Linux, а GNU/Linux, указывая на то, что Linux – это ядро. В принципе на базе ядра Linux могут быть построены и другие операционные системы (например, Android), и наоборот, современные ОС, известные как Linux, могут быть собраны и на других ядрах – как непосредственно, так и с трансляцией вызовов. Примеры первого – Debian/kFreeBSD – дебиан на ядре FreeBSD, Debian/kHurd – дебиан на ядре Hurd. Пример второго – Ubuntu для Windows 10, через трансляцию системных вызовов подсистемой Linux. Для графических приложений также необходим X-Server: например, XMing.

Ядро Linux изначально было создано для процессора Intel 80386, поэтому избежало ряда детских проблем MS DOS и Windows, разрабатываемых ещё для процессоров 8086 и 80286, при отсутствии полноценных механизмов защиты памяти и процессов.

В отличие от ядер многих других операционных систем, Linux – монолитное ядро. Изначально добавление драйверов требовало перекомпиляции ядра. В настоящее время ядро Linux – модульное, то есть драйверы можно отключать и подключать, не перезагружая систему. При этом ядро Linux остаётся монолитным (модульное ядро – это разновидность монолитного), и возможность компилирования ядра с нужными драйверами сохраняется.

Порядок запуска LINUX

BIOS/UEFI

BIOS (Basic Input Output System) – базовая система ввода-вывода.

Изначально при поступлении питания на процессор выполняются команды из ROM. Это программа самотестирования POST, инициализация прерываний BIOS, считывание параметров из энергонезависимой памяти BIOS – разновидности оперативной памяти, которая питается постоянно от батарейки и используется системным таймером, а также хранит ряд настроек системы, в том числе порядок загрузки и указание, с какого диска грузиться.

Проблемы MS DOS были унаследованы даже при исполнении на 80486 и последующих процессорах: это 16-битный код, адресация 1 Мб памяти, исполнение в реальном режиме 8086. Традиционный BIOS обладает теми же ограничениями. Дальнейшее развитие привело к созданию EFI – Extensible Firmware Interface (расширяемый интерфейс прошивки).

В настоящее время стандарт развивается как UEFI – Unified Extensible Firmware Interface.

Так же, как BIOS предоставляла системные вызовы для базовых операций с вводом/выводом, UEFI предоставляет платформонезависимую среду драйверов, с которой операционная система может взаимодействовать для использования текста и графики до загрузки платформозависимых драйверов.

UEFI поддерживает механизмы загрузки операционных систем, в том числе и с GPT-дисков (если в MBR – Main Boot Record в первом секторе диска должен был находиться загрузчик, то в UEFI загрузчик уже содержится. Кроме того, таблица разделов GTP позволяет адресовать более 2,2 Тб).

Помимо этого, UEFI поддерживает механизмы расширений и оболочку наподобие COMMAND.COM с возможностью выполнения скриптов.

Новые технологии способны порождать и новые проблемы. Так, на компьютерах, поставляемых с ОС Windows, в UEFI включен по умолчанию режим Secure Boot. Этот режим предотвращает загрузку операционных систем, если они не подписаны ключом, загруженным в UEFI. В таком случае можно отключить режим Secure Boot либо добавить в прошивку UEFI подписанный ключ или выбрать дистрибутив Linux, в котором поддерживается режим Secure Boot – например, Ubuntu.

Итак, UEFI – это специфическая операционная система, традиция которой начинается ещё с прошивки в ROM BASIC в эпоху микрокомпьютеров.

Загрузчик (GRUB/LILO/MSDOS+LoadLin)

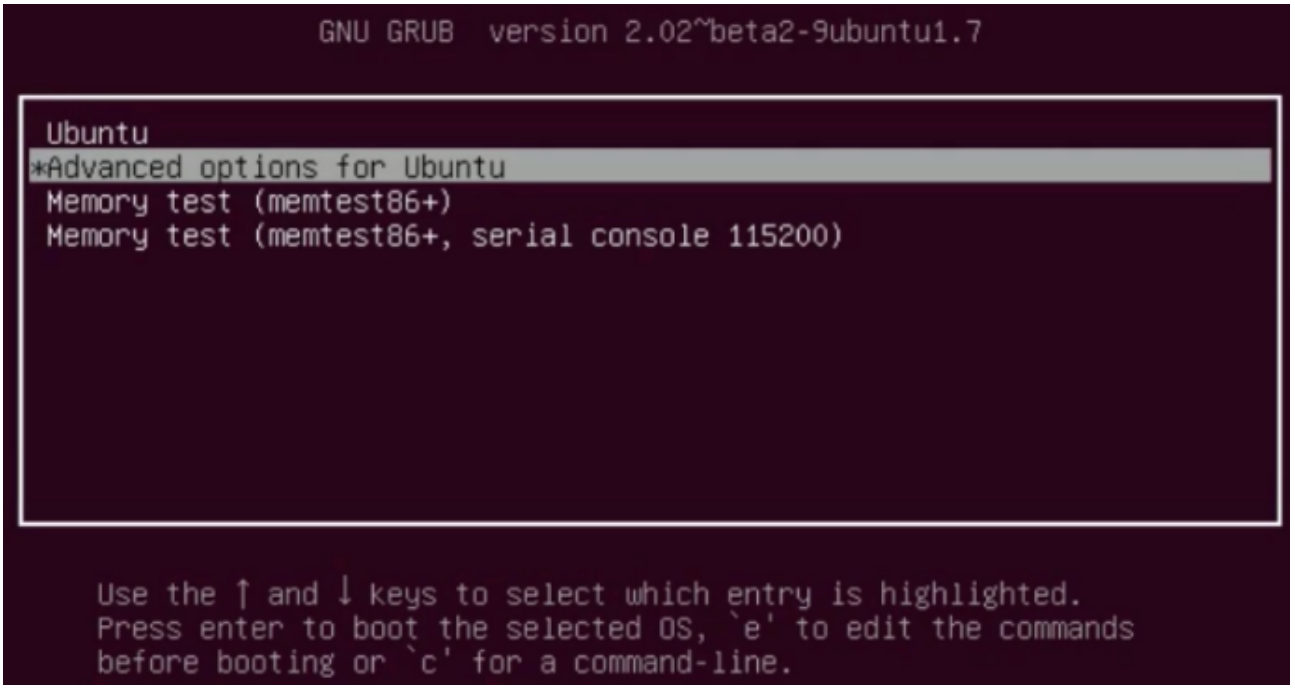
UEFI стартует запуск Windows или загрузчика Linux.

Среди загрузчиков Linux популярен прежде всего GRUB (Grand Unified Boot Loader; то, что обычно называется GRUB, – это на самом деле GRUB2). Кроме того, иногда используется загрузчик Lilo.

Как вы помните, в некоторой степени для Windows 9x загрузчиком выступала MS DOS. Такая возможность есть и в Linux при использовании загрузчика Loadlin. При запуске загрузчика из MS DOS загрузчик останавливает DOS и загружает ядро Linux, превращая DOS в подобие консоли GRUB. Linux не выполняется как приложение DOS, ядро DOS полностью выгружается – поэтому выхода из Linux в DOS в этом случае не будет (разве что через перезагрузку). Loadlin использовался, например, для загрузки с CD.

Загрузчик GRUB – это довольно сложная система по сравнению с LILO или Loadlin, своего рода однозадачная операционная система.

При запуске Linux необходимо успеть зажать Shift. Тогда появится окно загрузчика, которое по умолчанию скрыто.



GRUB2 имеет несколько конфигурационных файлов, позволяет загружать несколько операционных систем, настраивать цвет и шрифт текста, показывать при загрузке splash-экран, отображать или отключать меню выбора при загрузке и т.д.

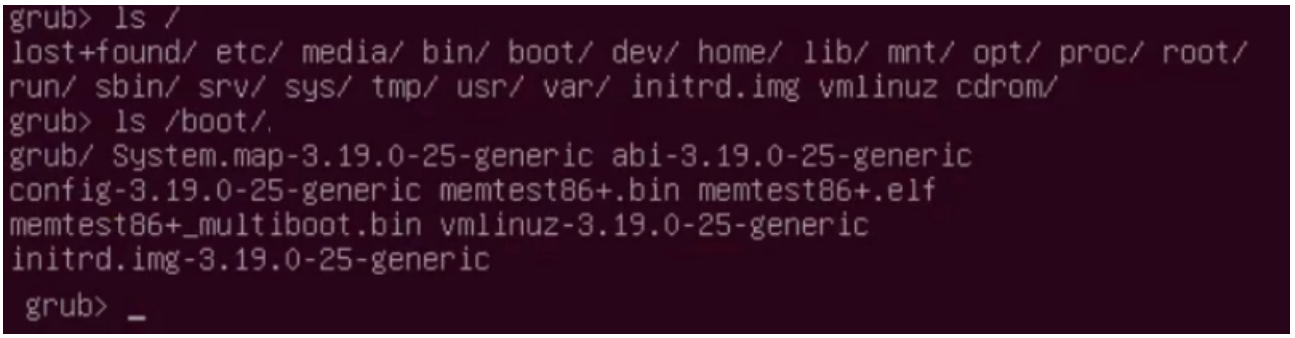
В меню по умолчанию доступна загрузка ОС, а также Memtest (не требующая операционной системы программа предназначенная для тестирования RAM). Здесь же могут присутствовать новые пункты – например, загрузка Windows.

Для редактирования команд перед загрузкой можно нажать «e» или перейти в режим консоли «c».

Нажмем «c».

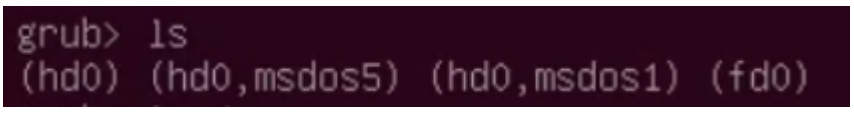
GRUB сообщит, что перед нами bash-подобная система. В ней работает автодополнение по TAB, вызов ранее введённых команд клавишей «вверх».

Чтобы получить список команд, можно набрать help. Среди прочих команд в списке есть команды ls, cat. GRUB понимает файловые системы ext2 и ext4, но не понимает LVM: в этом случае обычно создаётся отдельный раздел на ext2 вне томов LVM, и система загружается с него.



Обратите внимание на файлы, содержащиеся в разделе /boot, и сравните их с содержанием стартового меню. Попробуйте самостоятельно с помощью cat посмотреть содержимое одного из системных файлов (например, /etc/shadow), сделайте выводы.

Особым образом работает команда ls без параметров:



Обратите внимание, что система именования дисков подобна имеющейся в Linux (/dev/sda, /dev/sda1, /dev/sda2, /dev/sdb1, /dev/sdb2).

(hd0) соответствует диску /dev/sda.

(hd0,msdos5) означает 5 раздел на диске hd0 (/dev/sda) и соответствует /dev/sda5. Это логический раздел, очевидно, предназначенный для swap.

(hd0,msdos1) означает 1 раздел на диске hd0 (/dev/sda) и соответствует /dev/sda1. Это первичный раздел – очевидно, он будет монтироваться как корень (/) в Linux.

В консоли GRUB мы тоже имеем некий аналог монтирования диска в корень, однако по большей части это не столько монтирование, сколько указание значения переменной.

Добавьте в виртуальной машине второй диск (/dev/sdb), создайте раздел (/dev/sdb1) и отформатируйте его в ext2 или ext4. Создайте несколько файлов.

В Linux:

Создаём раздел:

```
# fdisk /dev/sdb
Команда (m для справки) : m
Команда (m для справки) : p
Команда (m для справки) : n
Select (default p): p
Команда (m для справки) : p
Команда (m для справки) : w
```

Некоторые команды fdisk:

- m – справка;
- p – вывести перечень разделов;
- n – создать новый раздел (требуется указать p – первичный или e – расширенный);
- w – сохранить изменения;
- q – выход без сохранений.

Форматируем раздел:

```
# mkfs -t ext4 /dev/sdb1
# mkdir /mnt/diskb
# mount /dev/sdb1 /mnt/diskb
```

Самостоятельно создайте в /mnt/diskb один или несколько каталогов и файлов (или скопируйте).

В GRUB:

```
grub> set root=(hd1,msdos1)
grub> ls
grub> set root=(hd0,msdos1)
grub> ls
```

Сравните результаты вывода ls.

Для процесса загрузки важно, чтобы root был установлен на тот раздел, с которого вы будете загружать систему. Если вы не меняли этот показатель, дополнительно устанавливать root нет необходимости: он указан по умолчанию.

Для загрузки понадобится:

- загрузить в память ядро командой linux;
- загрузить в память образ виртуальной файловой системы командой initrd;
- стартовать загрузку командой boot.

```
grub> linux /boot/vmlinuz-3.19.0-25-generic root=/dev/sda1
grub> initrd /boot/initrd.img-3.19.0-25-generic
grub> boot
```

Важно: версии ядра и образа initrd должны совпадать.

Для запуска в однопользовательском режиме для linux также необходимо указать параметр single.

В случае, если вы передумали загружать систему, можно остановить компьютер командой halt или вызвать перезагрузку командой reboot.

Загрузка ядра

Когда образ ядра загружен в память командой linux и образ ядра initrd также загружен, после команды boot управление передаётся ядру. Ядро стартует, распаковывает образ initrd в память, монтирует его, после чего загружает необходимые модули ядра. initrd можно не использовать, если самостоятельно скомпилировать ядро с драйверами для нужного оборудования. Иногда так делают для увеличения скорости загрузки. В случае ошибки (например, если не указать файловую систему для монтирования в корень) будет запущена initframes. В этом случае останется доступным RAM-диск.

Система инициализации

После того как драйвера (модули ядра) загружены и корень диска смонтирован, запускаются процессы init и поток ядра kthread. При этом kthread служит родителем для потоков ядра, а init запускает все программы, работающие в пользовательском окружении.

Отметим, что Линус Торвальдс создавал Linux, в большей степени основываясь на System V, но с некоторой оглядкой на BSD (например, реализация сокетов Беркли). Таким образом, первой системой инициализации была SysV Init. Она обладала рядом недостатков, в частности, последовательной работой и отсутствием обратной связи от процессов. В результате появились новые системы, такие, как Upstart и System D – последняя не относится к unix-way, но практически доминирует в новых дистрибутивах GNU/Linux.

System V/Upstart/SystemD образуют определенный аспект различий между разными линуксами: одни и те же процессы в них происходят по-разному и задаются разными конфигурационными файлами. Впрочем, такие различия могут всплывать и между разными версиями одной ОС. В целом различия в конфигурационных сервисах и файлах могут меняться от Линукса к Линуксу, что необходимо иметь в виду при переходе от систем, основанных на Red Hat (Fedora, Centos и т.д.), к системам, основанным на Debian (в т.ч. Ubuntu и её клонам). При этом процессы в системах выполняются одни и те же: стартуют такие сервисы (демоны), как cron, syslog, запускаются службы терминалов и т.д.

Уровни выполнения

Это по большей части устаревшее понятие не имеет отношения к кольцам защиты: они всего лишь определяют, какой набор сервисов будет запускаться или останавливаться. Традиционно выделяют 7 уровней выполнения (по числу линий в коммутаторах телефонной линии), но фактически их может быть до 9:

- 0 – останов системы (все сервисы завершаются);
- 1 – однопользовательский режим;

- S – псевдоним для 1;
- 2-3 – многопользовательский режим (иногда различают режим без поддержки сети и с поддержкой сети. Какой именно у режима уровень выполнения – зависит от операционной системы);
- 4-5 – графический режим (обычно 5; в этом режиме стартует X-Server).

В SystemD поддерживаются уровни выполнения, но фактически это псевдонимы для набора зависимостей.

Терминалы

Независимо от того, работаете ли вы только в консоли или в графической среде, система предоставляет набор до 6 (или более) текстовых терминалов.

Переключение из одного текстового терминала в другой происходит по команде Alt-F1 – Alt-F6.

Переключение в оконную систему – Alt-F7.

Переключение в текстовый терминал из X11-Server – Ctrl-Alt-F1 – Ctrl-Alt-F6.

Текстовые терминалы обслуживаются процессом getty и имеют соответствующие файлы устройств /dev/ttyX (где X имеет значения от 1 до 6, например, /dev/tty1, в редких случаях – больше 6).

Фактически терминалы эмулируют настоящие терминалы («печатные машинки» прошлых поколений ЭВМ).

Из второго терминала можно отправить сообщение в первый:

```
# echo Test >/dev/tty1
```

Графическая система X11

В отличие от системы Windows, где задачи на отрисовку окон, элементов управления, кнопок отправляются через системные запросы к WinAPI, в GNU/Linux (и не только) применяется принципиально иная архитектура, основанная на идее клиент-серверного взаимодействия.

Для X11 существует большой набор сред рабочего стола (Gnome, Unity, KDE, XDE), в зависимости от которых может меняться вид оконного интерфейса, но логику работы приложений это не затронет.

Запускаемое приложение в среде X11 является клиентом X11-сервера. Учитывая это, можно запускать графические приложения даже на машинах, где X11-сервер не запущен. В таком случае наблюдается интересная ситуация: ssh-клиент подключается к ssh-серверу, на котором графическое приложение выступает в качестве X11-клиента, подключаемого к X11-серверу на стороне ssh-клиента.

Для Windows также существуют реализации X11-сервера – например, XMing. В таком случае можно запускать удалённые графические программы через putty.

Виртуальная файловая система

Существенным отличием Linux и других UNIX-подобных систем от таких систем, как DOS и Windows, является то, что работа в них ведётся только с виртуальной файловой системой, то есть фактически, с одним деревом, в иерархии которого находится тот или иной файл независимо от того, где он расположен на самом деле (на каком разделе, на диске или по сети, или же создан из какой-либо

структуры ядра). Более того, Linux следует концепции «всё есть файл», и поэтому устройства, драйверы, процессы – всё в нём отображается в структуре файловой системы, благодаря чему достигается существенная гибкость в работе (скрипты могут получить доступ к любой необходимой информации, как и сам пользователь).

root

Корень – / – является точкой монтирования всей системы. Корень (/) в Linux не аналогичен, например, указанию вида C:\ в MS DOS и WINDOWS, то есть / не есть оглавление некоего системного раздела, это именно корень. При этом в / может быть смонтировано содержимое – например, /dev/sda1, – но именно смонтировано. Первоначально при загрузке монтируется виртуальный образ initrd и уже после размонтируется, а монтируется, скажем, /dev/sda1.

initrd

Init RAM disk – образ файловой системы, который распаковывается в RAM-диск и монтируется в корень. Он необходим для загрузки модулей ядра и выполнения простейших скриптов. В настоящее время образ initrd представляет собой архив cpio сжатый gzip.

Распаковать и изучить образ можно следующим способом:

```
mkdir /tmp/initrd&cd /tmp/initrd
cp /boot/initrd.img-3.19.0-25-generic /tmp/initrd/img.gz
gzip -d img.gz
mv img img.cpio
cpio -i < img.img
ls
mc
```

procfs

procfs – это виртуальная файловая система, отображающая информацию о процессах и системе. В директории /proc находится некоторая информация о системе, а в директориях с PID процесса – о соответствующих процессах.

Примеры:

```
# cat /proc/interrupts
# cat /proc/cpuinfo
# cat /proc/2/stat
# cat /proc/2267/environ
```

procfs содержит также ряд полезных подразделов с информацией о прерываниях, файловых системах, системными параметрами.

Интерфейс позволяет не только получать информацию о процессах, но и изменять их. Например, можно разрешить IP-Forwarding, чтобы машина могла работать как маршрутизатор:

```
echo "1" > /proc/sys/net/ipv4/ip_forward
```

Важно, что маршрутизация выполняется на уровне ядра, без привлечения пользовательского пространства. Сделано это в интересах быстродействия. Маршрутизация, сетевой фильтр и другие механизмы реализованы внутри ядра.

sysfs

sysfs так же, как и profs, предоставляет для пользовательского пространства информацию, доступную ядру. В /etc/sysfs содержатся структурированные в виде текстовых файлов данные о найденном ядром оборудовании: это шины, блочные и символьные устройства (в том числе информация о PCI, закрепленных аппаратных прерываниях и т.д.), драйверы.

devfs

devfs – важный элемент системы, реализующий концепцию «всё есть файл». Если мы хотим отправить сообщение в терминал 1, пишем:

```
# echo Test >/dev/tts1
```

Существующие в директории /dev файлы – особого рода имена устройств, с которыми можно работать. Точнее, это именованные каналы (FIFO), позволяющие взаимодействовать с драйверами устройств.

Ряд устройств создаётся операционной системой автоматически. К ним относятся терминалы, устройства дисков (hda – IDE, sda – SATA).

Устройства делятся на блочные и символьные: с символьными можно работать как с обычными текстовыми файлами, блочные работают блоками (пример – устройства дисков).

У данных файлов нет размера, но они обладают такими атрибутами, как старший и младший номер устройства. Старший номер определяет тип устройства, младший – порядковый номер среди однотипных устройств.

Посмотрите, какие старшие и младшие номера присутствуют у дисковых устройств (/dev/sda, /dev/sda1, /dev/sdb).

Не всегда файл нужного устройства будет присутствовать. В этом случае помогает команда mknod. Она позволяет с ключом b создавать блочные устройства, с ключом u – символьные, с ключом p – именованные каналы.

Примеры:

```
# mknod /dev/test1 b 200 1
# mknod /dev/test2 u 201 1
# mknod /tmp/test3 p
```

Фактически это ещё один механизм межпроцессного взаимодействия.

После перезагрузки созданные файлы устройств в директории /dev не сохраняются. Поэтому они должны создаваться или с помощью mknod скриптами при загрузке, или с помощью аналогичного системного вызова соответствующего модуля ядра.

Ядро, приложения, пользователи, доступ

Структура ОС: ядро, модули ядра, утилиты, демоны

Итак, у современных процессоров существует несколько колец защиты. Из них в Linux используется только два: с наивысшим и наинизшим приоритетом. Ядро работает в кольце 0, пользовательское пространство – в кольце 3. Помимо защиты памяти, данный механизм предоставляет и защиту операций ввода-вывода. Не все прерывания могут быть вызваны из пользовательского пространства. Для доступа к ядру предоставляется несколько механизмов: один из них – int 0x80, другой – sysenter. Обработчики прерываний работают в кольце 0. При вызове прерываний, осуществлении системных вызовов осуществляется переключение контекста.

На уровне ядра работает, собственно, само ядро Linux. Ядро Linux многопоточно. Потоки ядра видны в выводе ps aux – точнее, они не являются ни потоками, ни процессами, когда речь идет о прикладных программах. В принципе потоки прикладных программ также являются технически процессами. Потоки ядра являются третьим элементом параллелизма, которые имеют много общего как с потоками, так и с процессами.

Потоки ядра:

```
ps -ef
```

```
UID          PID  PPID  C STIME TTY          TIME CMD
root           1     0  0 февр.18 ?  00:00:08 /sbin/init
root           2     0  0 февр.18 ?  00:00:00 [kthreadd]
root           3     2  0 февр.18 ?  00:00:12 [ksoftirqd/0]
root           5     2  0 февр.18 ?  00:00:00 [kworker/0:0H]
root           7     2  0 февр.18 ?  00:00:34 [rcu_sched]
root           8     2  0 февр.18 ?  00:00:00 [rcu_bh]
root           9     2  0 февр.18 ?  00:00:00 [migration/0]
root          10     2  0 февр.18 ?  00:00:14 [watchdog/0]
root          11     2  0 февр.18 ?  00:00:00 [khelper]
root          12     2  0 февр.18 ?  00:00:00 [kdevtmpfs]
root          13     2  0 февр.18 ?  00:00:00 [netns]
root          14     2  0 февр.18 ?  00:00:00 [perf]
root          15     2  0 февр.18 ?  00:00:00 [khungtaskd]
root          16     2  0 февр.18 ?  00:00:00 [writeback]
root          17     2  0 февр.18 ?  00:00:00 [ksmd]
root          18     2  0 февр.18 ?  00:00:01 [khugepaged]
root          19     2  0 февр.18 ?  00:00:00 [crypto]
root          20     2  0 февр.18 ?  00:00:00 [kintegrityd]
root          21     2  0 февр.18 ?  00:00:00 [bioset]
root          22     2  0 февр.18 ?  00:00:00 [kblockd]
root          23     2  0 февр.18 ?  00:00:00 [ata_sff]
root          24     2  0 февр.18 ?  00:00:00 [md]
root          25     2  0 февр.18 ?  00:00:00 [devfreq_wq]
root          29     2  0 февр.18 ?  00:00:00 [kswapd0]
```

Потоки ядра отображаются в квадратных скобках. Обратите внимание, что PPID всех родительских потоков – это PID потока kthreadd.

Два процесса – init и kthreadd – имеют PPID, равный 0. Они запускаются при старте ядра.

Несложно подсчитать число потоков ядра

```
ps -ef | grep -F '[' | wc -l
```

```
oga@uho:~$ ps -ef|grep -F '['|wc -l
116
oga@uho:~$
```

Создаются потоки ядра вызовом kernel_thread().

Пользователи, разграничения полномочий

Linux – многопользовательская система, в которой файлы имеют атрибуты пользователя владельца и группы. У каждого пользователя имеется UID – user identifier. Не только файлы принадлежат пользователям, но и программы выполняются от имени пользователя. Соответственно принимается решение, может ли данный процесс поменять файл или нет.

Различаются 3 типа пользователей:

- root – пользователь с UID=0. Может иметь доступ ко всем файлам, менять права независимо от установленных прав. При этом root также работает в пользовательском пространстве. Не следует путать права доступа к root и уровень ядра. Попытка создать новый файл в procfs выдаст ошибку «доступ запрещён». В настоящее время, как правило, доступ к системе непосредственно через root не используется. Механизм временного получения UID=0 реализуется с помощью sudo и групп, указанных в sudoers;
- демоны (UID = от 1 до 999) – псевдопользователи. Для них установлен несуществующий пароль, а в качестве оболочки указано /usr/sbin/nologin. Тем не менее процессы-демоны (apache2/httpd, vsftpd, bind/named) запускаются от имени специальных пользователей;
- пользователи (UID=1000 и выше) – учётные записи пользователей-людей.

Демоны

Демонами называются программы, работающие в фоновом режиме. Они необходимы либо для корректной работы системы (демоны логирования, планировщик cron и т.д.), либо для обслуживания серверов (vsftpd, openssh-server, apache2 и т.д.). Как правило, они запускаются без связи с терминалом и конфигурируются в рамках настройки соответствующей системы инициализации. Тем не менее возможен запуск демона и в консоли – тогда на экран будет выдаваться журнальная/отладочная информация.

Пакетные менеджеры

Как уже говорилось, между разными системами есть различия – в частности, между системами, основанными на Red Hat (Fedora, Centos и т.д.) и основанными на Debian (в т.ч. Ubuntu и её клоны).

В RedHat (включая Centos) принята система, базирующаяся на .rpm-файлах (red hat package manager). Соответственно для работы с пакетами используется утилита rpm и ее обёртка yum.

В Debian (включая Ubuntu) принята система, базирующаяся на .deb-файлах. Соответственно для работы с пакетами используется утилита dpg и её обертка apt.

Интересные примеры

Работа с модулями ядра

Пример подключения модуля:

```
modprobe 8021q
```

Пример удаления модуля ядра:

```
rmmod 8021q
```

Список модулей:

```
lsmod
```

Загрузка модуля из файла:

```
insmod /lib/modules/3.19.0-25-generic/kernel/net.8021q.ko
```

Запуск графических программ на удалённой машине

В Linux установите openssh-server.

В windows установите XMing.

Воспользуйтесь putty, в настройках установите X11-Forwarding.

Войдите в сессию и запустите какое-либо графическое приложение.

Сравните варианты запуска.

```
$ nautilus
```

```
$ nautilus&
```

Практическое задание

Для работы понадобится установленный Linux (например, Ubuntu).

1. Зайти в консольный режим GRUB:
    - просмотреть содержимое директорий дисков;

- вывести на экран содержимое какого-либо текстового файла (например, .conf) с помощью cat;
- перед перезагрузкой в Linux в виртуальной машине создать новый диск, отформатировать его, создать раздел и несколько файлов;
- изучить работу команд set root, ls, cat на примере разделов разных дисков.

2. Запустить из GRUB Linux в многопользовательском и однопользовательском режиме (консоль восстановления).
3. Распаковать initrd, изучить, какие скрипты и программы в нём содержатся.
4. В консоли Linux просмотреть дерево /proc, изучить, какие полезные сведения можно использовать из находящихся в директории данных.
5. * Установить X-Server (например, XMing) на Windows-машине, запустить десктопное приложение Linux на рабочем столе Windows, подключившись через putty и включив туннелирование X11 трафика.
6. * Установить openvpn-сервер, проследить с помощью tcpdump или wireshark движение трафика. Обратить внимание, какие части выполняются в пользовательском пространстве, а какие – на уровне ядра.
7. * Написать (или найти готовый код и разобрать) модуль ядра, запускающий поток ядра.

Примечание. Задания со звёздочкой предназначены для тех, кому недостаточно заданий 1-4 и требуются более сложные задачи. Задания разные – выберите то, что больше подходит вам.

Дополнительные материалы

1. http://rus-linux.net/MyLDP/boot/uefi-boot-linux.html
2. http://www.multiboot.ru/DUET.htm
3. http://rus-linux.net/kos.php?name=/papers/boot/boot-03initrd.html
4. https://habrahabr.ru/company/smart_soft/blog/184174/
5. Потоки ядра http://www.ibm.com/developerworks/ru/library/l-linux_kernel_70/index.html
6. sysfs http://rus-linux.net/MyLDP/BOOKS/Moduli-yadra-Linux/05/kern-mod-05-10.html
7. http://mydebianblog.blogspot.ru/2013/02/sysfs-linux.html
8. http://src-code.net/fajlovaya-sistema-sysfs/
9. procfs https://mnorin.com/nemnogo-o-direktorii-proc-v-linux.html

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. http://rus-linux.net/MyLDP/BOOKS/lpg-04/node6.htm
2. http://linuxdoc.ru/mknod.html
3. http://rus-linux.net/kos.php?name=/papers/boot/boot-03initrd.html
4. http://www.k-max.name/linux/yadro-linux-poluchenie-informacii-i-upravlenie/