D-Link

Лекция 4.

Процессы в Linux. Команды для работы с процессами

D-Link

Лекция 4. Процессы в Linux. Команды для работы с процессами

- Понятие процесса в Linux
- Атрибуты процесса, состояние процесса
- Запуск процесса на переднем плане и в фоновом режиме
- Команда ps
- Команда top
- Команда at
- Команда nohup
- Команды nice и renice
- Сигналы. Команды kill и killall

Понятие процесса в Linux

Процесс можно определить как **программу, находящуюся в состоянии исполнения**.

Кроме, исполняемого кода каждый процесс содержит собственное **адресное пространство**, в котором содержатся данные процесса, а так же **набор ресурсов**, например, таких как открытые им файлы.

Linux, как и практически все современные операционные системы, является **многозадачной** операционной системой, т.е. позволяет выполнять много процессов одновременно. Операционная система позволяет имитировать запуск большего числа процессов, выделяя им процессорное время по очереди небольшими частями – т.н. **квантами**.

Каждый процесс может породить дополнительные процессы. При этом процесс, запустивший новый процесс, называется **родительским**, а новый процесс по отношению к создавшему его процессу называется **дочерним**. Процессы, порожденные одним и тем же родительским процессом, называются **братьями** (или **сестрами**).

В Linux реализована четкая **иерархия процессов** в системе. Корнем этой иерархии является процесс **init** имеет идентификатор, равный 1. От него порождаются все остальные процессы. Процесс **init** никогда не завершается.

Атрибуты процесса

Каждый процесс в ОС Linux характеризуется следующим набором **атрибутов**:

Идентификатор процесса (**Process IDentifier** - **PID**). Каждый процесс в системе имеет уникальный числовой идентификатор. Значение PID используется в ряде команд для задания процесса.

Идентификатор родительского процесса (**Parent PID** - **PPID**).

Реальный и **эффективный идентификаторы пользователя** (**UID**, **EUID**) и **группы** (**GID**, **EGID**). Данные атрибуты процесса говорят о его принадлежности к конкретному пользователю и группе пользователей. Реальные идентификаторы совпадают с идентификаторами пользователя, который запустил процесс, и группы, к которой он принадлежит. Эффективные идентификаторы показывают, от имени какого пользователя был запущен процесс. Они могут не совпадать с реальными идентификаторами, если для запускаемой программы установлен бит SGID или SUID. Права доступа процесса к ресурсам операционной системы определяются эффективными идентификаторами. Для управления процессом используются реальные идентификаторы. Все идентификаторы передаются от родительского процесса к дочернему.

Приоритет (**priority**) и **относительный приоритет** (**уступчивость**) процесса (**nice**). Приоритет определяет количество процессорного времени, которое выделяется процессу для выполнения. Пользователь может изменять значение только относительного приоритета (nice), который является своеобразной поправкой к основному значению приоритета и может изменяться в диапазоне от -20 до 19. Здесь меньшее значение соответствует более высокому приоритету.

Состояние процесса

Процесс может в текущий момент находиться в одном из следующих **состояний**:

- **Runable** (**R**) – процесс выполняется или готов к выполнению,
- **Sleeping** (**S**) – процесс находится в состоянии ожидания некоторого внешнего события (например, завершения операции ввода-вывода) и не выполняется. При получении сигнала процесс временно прерывает ожидание для обработки сигнала. Поэтому такие процессы можно завершить до окончания операции ввода-вывода.
- **Uninterruptible** (**D**) – состояние аналогично предыдущему с той разницей, что ожидание не прерывается для обработки сигналов. Такой процесс нельзя завершить до окончания операции ввода-вывода.
- **Stopped** (**T**) – процесс остановлен. Процесс переходит в это состояние когда получает соответствующий сигнал. Так же в этом состоянии находится процесс, если производится его отладка.
- **Zombie** (**Z**) – процесс-“зомби”. Этот процесс завершил свое выполнение и почти полностью выгружен из памяти. Единственная информация, которая хранится о данном процессе – код его завершения , значение которого может потребоваться родительскому процессу. После запроса кода завершения родительским процессом, процесс-“зомби” полностью удаляется из памяти.

Запуск процесса на переднем плане и в фоновом режиме

При запуске программы из консоли обычным способом она запускается как задача **переднего плана** (**foreground**). Программа переднего плана получает доступ к стандартному потоку ввода, что позволяет ей получать данные и команды от пользователя, и к стандартному потоку вывода, для выдачи данных пользователю.

Программа, запущенная в **фоновом режиме** (**background**) не может осуществлять интерактивное взаимодействие с пользователем через консоль, но ни в чем другом не ограничена.

Чтобы выполнить команду в фоновом режиме, после команды нужно поставить символ **&**.

```
$xeyes -center red &
[1] 2811
```

Для вывода информации о командах, запущенных в фоновом режиме, предназначена команда **jobs**:

```
$ jobs
[1]+  Running                 xeyes -center red &
```

Перевести команду из фонового режима на передний план можно с помощью команды **fg**.

```
$fg номер_задачи.
```

Для перевода команды с переднего плана в фоновый режим нужно нажать на клавиатуре Ctrl+Z (при этом процесс будет приостановлен) и ввести команду **bg**, которая продолжит выполнение процесса в фоновом режиме.

Команда ps

Команда **ps** выводит информацию о запущенных процессах.

```
$ ps [ключи]
```

Эта команда имеет, пожалуй, самый запутанный синтаксис из всех команд Linux. Она пытается одновременно поддерживать сразу три варианта опций: опции UNIX, опции BSD, опции GNU. Часто ключи команды дублируют друг друга.

Если запустить команду без параметров, то она выведет список выполняющихся процессов, которые запущенны текущим пользователем и ассоциированы с текущим терминалом.

```
$ ps
  PID TTY          TIME CMD
 2171 pts/0    00:00:00 bash
 2191 pts/0    00:00:00 ps
```

Если мы хотим просмотреть список всех запущенных процессов, то нужно задать в команде ключ **-A** или **-e**.

Иерархию процессов отображает специальная команда **pstree**.

Команда top

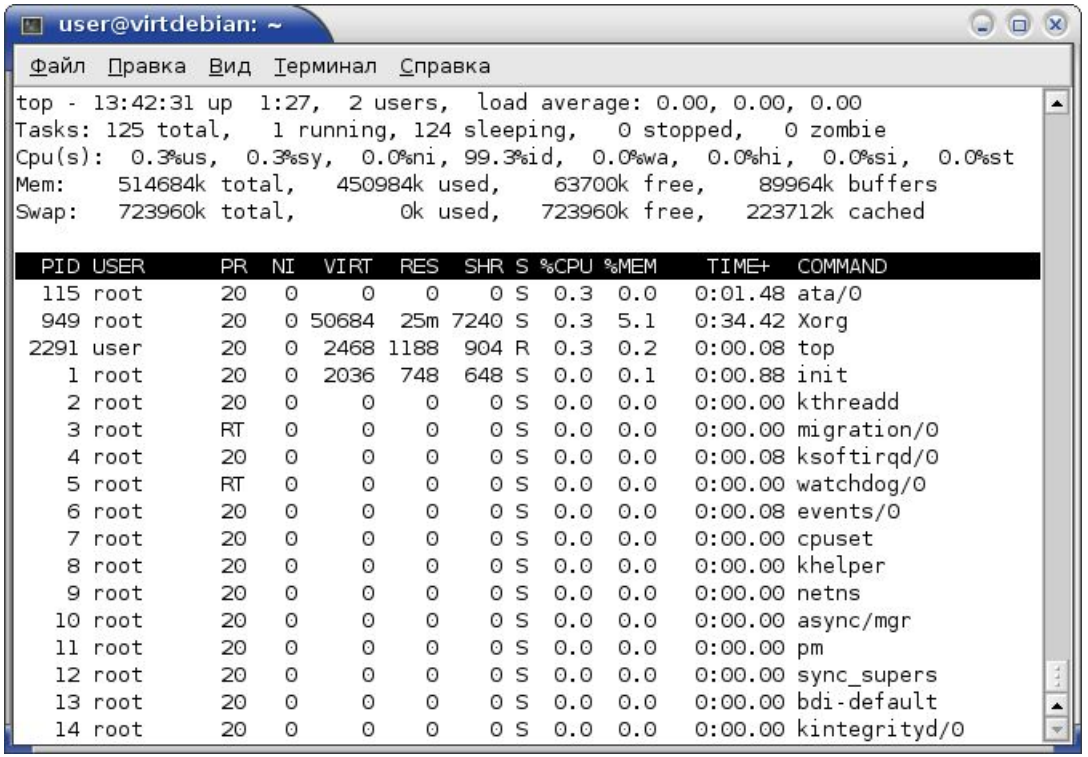
Команда **top** отображает информацию о запущенных процессах и загрузке системы в режиме реального времени.

```
$top [ключи]
```

При выполнении **top** в верхней части окна отображается текущее время, время, прошедшее с момента запуска системы, число пользователей в системе, число запущенных процессов и число процессов, находящихся в разных состояниях, данные об использовании ЦПУ, памяти и свопа.

В утилите **top** можно вводить команды с клавиатуры в интерактивном режиме.

h - отображение справки по работе с программой,
k - отправка сигнала процессу.
M - сортировать процессы по объёму занятой ими памяти.
u - вывести процессы заданного пользователя, имя которого будет запрошено командой.
r - эта команда используется для изменения динамического приоритета выбранного процесса.
q - завершение команды



Команда at (1)

Команда **at** предназначена для выполнения одной или нескольких команд в заданный момент времени в будущем.

```
$at [ключи] время
```

Команда **at** читает список выполняющихся команд из стандартного потока ввода или из указанного файла. Если команды вводятся пользователем с клавиатуры, то после завершения ввода нужно нажать Ctrl+D.

Введенные команды будут выполнены однократно в указанное время.

Время может задаваться в различных форматах.

Простейший формат – HH:MM, например – 13:15. Так же при задании времени могут использоваться слова noon, midnight и приставки AM и PM для указания времени дня. Кроме времени может указываться дата выполнения команд.

Ключами команды могут быть:

-m – отправить пользователю письмо по электронной почте после завершения выполнения команд.

-f файл – считать команды из указанного файла.

При использовании команды **at** следует помнить, что запускаемые ей команды будут выполняться **без привязки к терминалу и графической среде**.

Команда at (2)

```
$ echo 'ls > /home/user/test/attestfile' | at 12:45
warning: commands will be executed using /bin/sh
job 6 at Wed Apr 13 12:45:00 2011
$ date
Срд Апр 13 12:42:19 MSD 2011
$ cat /home/user/test/attestfile
cat: /home/user/test/attestfile: Нет такого файла или
каталога
$ date
Срд Апр 13 12:45:12 MSD 2011
$ cat /home/user/test/attestfile
attestfile
myscr
test2
testtextfile
```

Команда nohup

Когда мы выходим из командной оболочки, она посылает всем запущенным из нее программам сигнал на завершение работы. Если же мы хотим, чтобы программа продолжила выполняться после завершения командной оболочки, то запускать такую программу нужно с помощью специальной команды **nohup**. С помощью данной команды запускаются различные системные службы.

```
$ nohup команда [ключи] &
```

Если стандартный поток ввода команды связан с терминалом, то он перенаправляется на устройство /dev/null, а стандартный вывод, связанный с терминалом, направляется в файл nohup.out, расположенный в домашнем каталоге пользователя.

Команды nice и renice

Задать относительный приоритет некоторой команды при ее запуске можно с помощью команды **nice**.

```
$ nice [ключи] [команда [аргументы_команды]]
```

Основным ключом команды **nice** является ключ **-n**, который задает значение относительного приоритета запускаемой команды, по умолчанию используется значение 10.

```
$ nice -n 5 egrep -f pattern_file text_file
```

Команда **renice** позволяет изменить относительный приоритет уже работающего процесса.

```
$ renice отн_приоритет [ключи]
```

Ключи команды задают процессы, для которых изменяется относительный приоритет. Возможно три варианта:

-p ID ... - процесс задается одним или несколькими идентификаторами процесса,

-g ID ... - процесс задается одним или несколькими идентификаторами групп пользователей,

-u username ... - процесс задается одним или несколькими именами пользователей.

Сигналы (1)

В Linux существует механизм оповещения процессов об асинхронных событиях, называемый **сигналами**. События могут быть как внешними по отношению к процессу, например, когда пользователь нажал Ctrl+C, так и внутренними, например деление на ноль. Сигналы обрабатываются процессами **асинхронно**, т.е. при получении сигнала операционная система прерывает нормальное выполнение процесса и передает управление обработчику сигнала.

Когда процесс получает сигнал, он должен его обработать одним из трех способов:

- просто игнорировать сигнал,
- захватить и обработать сигнал,
- выполнить действие по умолчанию.

Передать процессу сигнал может или операционная система, или другая программа, или пользователь с помощью специальной команды.

D-Link

Сигналы (2)

Имя сигнала	Номер сигн.	Описание	Действие поумолч.
SIGBUS	7	Аппаратная ошибка или ошибка выравнивания	Завершиться с созданием дампа ядра
SIGCHLD	17	Завершился дочерний процесс	Игнорировать
SIGCONT	18	Процесс продолжил выполняться после того, как был остановлен	Игнорировать
SIGFPE	8	Арифметическое исключение	Завершиться с созданием дампа ядра
SIGHUP	1	Управляющий терминал процесса был закрыт (чаще всего это связано с тем, что пользователь выходит из системы)	Завершиться
SIGILL	4	Процесс попытался выполнить недопустимую функцию	Завершиться с созданием дампа ядра
SIGINT	2	Пользователь ввел символ прерывания (Ctrl+C)	Завершиться
SIGKILL	9	Завершение процесса, которое невозможно захватить	Завершиться
SIGPWR	30	Сбой питания	Завершиться
SIGSEGV	11	Нарушение доступа к памяти	Завершиться с созданием дампа ядра
SIGSTOP	19	Приостановить выполнение процесса	Остановиться
SIGTERM	15	Завершение процесса с возможностью захвата	Завершиться
SIGTSTP	20	Пользователь ввел символ приостановки (Ctrl+Z)	Остановиться

Команды kill и killall

Основной командой, с помощью которой можно отправить сигнал процессу является команда **kill**. Как видно из названия, в большинстве случаев она используется для завершения процесса.

```
$ kill [ключи] идентификатор_процесса …
```

Если ключи в команде не заданы, то процессу посылается сигнал **TERM** (15), который приводит к его корректному завершению.

Для посылки процессу сигнала используется следующий синтаксис:

```
$ kill -s сигнал идентификатор_процесса …
```

Часто одна команда запускает сразу несколько процессов. Чтобы их завершить нужно использовать команду **killall**.

```
$ killall [ключи] имя_команды
```

Для посылки процессу сигнала используется следующий синтаксис:

```
$ killall -s сигнал имя_команды
```