Занятие 6.

Тема: Процессы.

Вид занятия: лекция, практическое занятие.

Учебные вопросы:

1. Процессы в Linux. Идентификаторы процессов. Демоны.
2. Команда ps.
3. Права доступа процессов. Реальный и эффективный идентификаторы. Биты SUID и SGID.
4. Управление процессами. Сигналы.
5. Команды nice, nohup, kill, killall.

Время: 90 минут

Литература:

1. Робачевский А.М. «Операционная система Unix®». – СПб.: БВХ – Санкт-Петербург, 1999. – 528 с., ил.
2. Системная справочная служба Linux Man

Ход занятия.

1. При рассмотрении предыдущих тем мы с вами часто уполтребляли определение “процесс”. Но что такое процесс?

Процесс – понятие совокупности программного кода и данных, загруженных в память ЭВМ. Процесс это не запущенная программа (приложение) или команда, так как приложение может 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 выполение кода такого процесса приостановлено. Существует всего 3 состояния, в которых может нахзодиться процесс:

Работающий процесс – в данный момент код этого 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) и идентификаторы родительского процесса (PPID – Parent Process Identifier). 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 – 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, т.к. 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.

2. Для просмотра списка процессов в Linux существует команда ps.

ps [PID] options – просмотр списка процессов. Без параметров ps показывает все процессы, которы 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 – отсортировать по алфавиту

-w – показать полные строки описания процессов. Если они превосходят длину экрана, то перенести описание на следующую строку. Параметр -ww позволить убрать вообще все ограничения на длину отображаемой строки и делать вывод в 2 и более строк при необходимости.

Пример1:

```
[gserg@WEBMEDIA gserg]$ ps
  PID TTY          TIME CMD
 3126 pts/2    00:00:00 bash
 3158 pts/2    00:00:00 ps
[gserg@WEBMEDIA gserg]$_
```

Пример2:

```
[gserg@WEBMEDIA gserg]$ ps 3126
```

```
  PID TTY      STAT   TIME COMMAND
 3126 pts/2    S      0:00 /bin/bash
[gserg@WEBMEDIA gserg]$_
```

Пример3:

```
[gserg@WEBMEDIA gserg]$ 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
[gserg@WEBMEDIA gserg]$_
```

Пример4:

```
[gserg@WEBMEDIA gserg]$ ps -efw
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]
root         7     1  0 10:01 ?        00:00:00 [kupdated]
root         8     1  0 10:01 ?        00:00:00 [mdrecoveryd]
root        12     1  0 10:01 ?        00:00:00 [kjournald]
root       115     1  0 10:01 ?        00:00:00 devfsd /dev
root       211     1  0 10:01 ?        00:00:00 [khubd]
root       334     1  0 10:01 ?        00:00:00 [kjournald]
root       594     1  0 10:01 ?        00:00:00 [eth0]
root       730     1  0 10:01 ?        00:00:00 /sbin/dhcpcd -h
WEBMEDIA -Y -N eth0
root       772     1  0 10:02 ?        00:00:00 /sbin/dhcpcd -h
WEBMEDIA -Y -N eth0
rpc        820     1  0 10:02 ?        00:00:00 portmap
root       836     1  0 10:02 ?        00:00:00 syslogd -m 0
root       844     1  0 10:02 ?        00:00:00 klogd -2
root       879     1  0 10:02 ?        00:00:00 gpm -t ps/2 -m
/dev/psaux
xfs       1074     1  0 10:02 ?        00:00:07 xfs -port -1
-daemon -droppriv -user xfs
root      1130     1  0 10:02 ?        00:00:00 /usr/sbin/apmd -p
10 -w 5 -W -P /etc/sysconfig/apm-scripts/apmd_proxy
......
gserg     3122  2072  0 17:56 ?        00:00:00 xmms
gserg     3123  2072  1 17:56 ?        00:00:03 xmms
gserg     3124  1914  0 17:56 ?        00:00:02 kdeinit: konsole
-icon konsole.png -miniicon konsole.png
gserg     3126  3124  0 17:56 pts/2    00:00:00 /bin/bash
gserg     3172  3126  0 18:01 pts/2    00:00:00 ps -efw
[gserg@WEBMEDIA gserg]$_
```

3. 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 еще один набор 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 программой chmod:

chmod u+s filename – установка бита SUID

chmod g+s filename – установка бита SGID

Для установки этих битов в абсолютном режиме их стоить представить в виде трех бит: SUID, SGID, Sticky bit соответственно.

После выставления необходимых прав добавьте в начало числа цифру для установки специальных бит:

```
Пример5:
[gserg@WEBMEDIA gserg]$chmod 7777 filename
[gserg@WEBMEDIA gserg]$ls filename
-rwsrwsrwt  1 gserg  gserg   23811 Aug 29 11:00 filename
[gserg@WEBMEDIA gserg]$
```

4. 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. Но 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 сситеме очень важную роль, о которой мы с вами поговорим на 16-ом занятии.

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. Процесс может воспользоваться действием по умолчанию, или, если у него есть обработчик сигнала, то он может перехватить или игнорировать сигнал. Сигналы `SIGKILL` и `SIGSTOP` невозможно не перехватить, не 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:

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

Название	***Действие по умолчанию***	***Значение***
		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. 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 вызова.
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.
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):

```
[gserg@WebMedia gserg]$ 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
[gserg@WebMedia gserg]$
```

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.

5. 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.

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).

```
[gserg@WebMedia gserg]$ less .bashrc &
[1] 3070
[gserg@WebMedia gserg]$ ps -efl | grep less
0 T gserg  3070 3018 0 80 0 - 1004 finish 17:56 pts/3    00:00:00
less .bashrc
0 S gserg  3072 3018 0 80 0 - 949  pipe_w 17:57 pts/3    00:00:00
grep less
[gserg@WebMedia gserg]$ nice -n 20 less .bashrc &
[1] 3081
[gserg@WebMedia gserg]$ ps -efl | grep less
0 T gserg  3081 3018 0 99 19 - 1003 finish 18:01 pts/3   00:00:00
less .bashrc
0 S gserg  3083 3018 0 81 0  - 950  pipe_w 18:01 pts/3   00:00:00
grep less
[gserg@WebMedia gserg]$
```

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.

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.

kill -SINAL pid – посылает сигнал процессу с идентификатором pid. Если сигнал не указан, команда посылает процессу сигнал SIGTERM.

```
[gserg@WebMedia gserg]$ less &
[1] 1352
[gserg@WebMedia gserg]$ 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
[gserg@WebMedia gserg]$ kill -SIGKILL 1352
[gserg@WebMedia gserg]$ ps
  PID TTY          TIME CMD
 1322 pts/2    00:00:00 bash
 1355 pts/2    00:00:00 ps
[1]+  Killed                  less
[gserg@WebMedia gserg]$_
```

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.

```
[gserg@WebMedia gserg]$ less ./.bashrc&
[1] 1374
[gserg@WebMedia gserg]$ less ./.bashrc&
[2] 1375

[1]+  Stopped                 less ./.bashrc
[gserg@WebMedia gserg]$ less ./.bashrc&
[3] 1376
```

```
[2]+  Stopped                 less ./.bashrc
[gserg@WebMedia gserg]$ ps
  PID TTY          TIME CMD
 1322 pts/2    00:00:00 bash
 1374 pts/2    00:00:00 less
 1375 pts/2    00:00:00 less
 1376 pts/2    00:00:00 less
 1377 pts/2    00:00:00 ps

[3]+  Stopped                 less ./.bashrc
[gserg@WebMedia gserg]$ killall -s KILL less
[1]   Killed                  less ./.bashrc
[2]-  Killed                  less ./.bashrc
[3]+  Killed                  less ./.bashrc
[gserg@WebMedia gserg]$
```