

Основы системного администрирования и сетевых технологий

УРОК №5

Процессы

Процессом называется выполняемая в данный момент программа или ее потомки.

Каждый процесс запускается от имени какого-то пользователя. Процессы, которые начинают выполняться при загрузке операционной системы, обычно выполняются от имени пользователей `root` или `nobody`.

Каждый пользователь может управлять поведением процессов, запущенных от его имени. Суперпользователь может управлять любыми процессами.

Каждый процесс имеет имя и уникальный номер – идентификационный номер процесса (Process IDentificator - PID). Именно PID используется ядром операционной системы и утилитами для управления процессами.

Передний план и фоновый режим

Процессы могут выполняться на переднем плане (foreground) и в фоновом режиме (background).

На переднем плане в каждый момент времени в каждом терминале выполняется только один процесс. Процесс переднего плана – это процесс, с которым взаимодействует пользователь, он получает информацию с клавиатуры (стандартный ввод) и посылает результаты на экран (стандартный вывод).

Передний план и фоновый режим

Фоновый процесс после запуска отключается от клавиатуры и экрана, т.е. не ожидает ввода данных со стандартного ввода и не выводит информацию на стандартный вывод. Командная оболочка не ожидает окончания запущенного в фоновом режиме процесса, что позволяет пользователю запустить еще один процесс. Обычно фоновые процессы выполняются в течении длительного времени и не требуют вмешательства пользователя.

Процессы могут быть *отложенными*. Отложенный процесс – это процесс, который в данный момент не выполняется и временно остановлен. Отложенный процесс можно продолжить как на переднем плане, так и в фоновом режиме. При возобновлении отложенного процесса он начнет выполняться с того места, на котором был остановлен.

Для выполнения программы в режиме переднего план достаточно просто набрать ее имя в командной строке. Для запуска программы в качестве фонового процесса необходимо в командной строке набрать имя программы и добавить знак амперсанта (&), отделенного пробелом от имени программы.

Передний план и фоновый режим

При запуске программы в фоновом режиме выводится сообщение, похожее на следующее:

[1] 134,

где:

[1] – номер запущенного фонового процесса;

134 – идентификационный номер (PID) запущенного фонового процесса.

Номер фонового процесса уникален только для пользователя, запустившего этот процесс, PID уникален для операционной системы и однозначно идентифицирует в ней процесс.

Остановка и возобновление процесса

Чтобы перевести процесс, выполняющийся на переднем плане, в фоновый режим необходимо сначала остановить выполнение процесса, а затем продолжить его выполнение в фоновом режиме.

Для остановки выполнения программы необходимо нажать комбинацию клавиш Ctrl+z. Будет выведено сообщение, похожее на

[1]+ Stopped

и пользователь получит возможность вводить команды в командную строку.

Чтобы перевести процесс в фоновый режим, необходимо выполнить команду `bg %1`

Здесь: %1 – имя переменной командной оболочки, в которой хранится PID фонового процесса с номером 1.

Остановка и возобновление процесса

Возврат из фонового режима на передний план выполняет команда `fg %1`

Между фоновым и остановленным процессами существует существенная разница. Остановленный процесс не выполняется и не потребляет ресурсы процессора, однако занимает оперативную память. В фоновом режиме процесс продолжает выполняться.

Чтобы остановить выполнение фонового процесса, необходимо сначала переместить процесс на передний план, а потом остановить.

Завершение работы процесса

Чтобы завершить (не остановить, а завершить) процесс переднего плана можно воспользоваться комбинацией клавиш `Ctrl+C`. Можно также попытаться использовать комбинацию клавиш `Ctrl+Break`. Для завершения фонового процесса, его сначала нужно перевести на передний план.

Для завершения процесса можно воспользоваться командами `kill` и `killall`.

Команда `kill` может получать в качестве аргумента как номер процесса, так и его PID. Команда

`kill %1`

эквивалентна команде

`kill 134`

если 134 – PID процесса с номером 1.

Завершение работы процесса

С помощью команды `killall` можно прекратить выполнение нескольких процессов, имеющих одинаковое имя. Например, команда `killall mc` прекратит работу всех программ с именем `mc`, запущенных от имени пользователя.

Сигналы

Сигналы – это короткие стандартные сообщения, которыми могут обмениваться процессы с помощью системы. Сообщение-сигнал не содержит никакой информации, кроме номера сигнала (для удобства вместо номера можно использовать предопределенное системой имя). Если процессу необходимо реагировать на сигнал особым образом, он может зарегистрировать обработчик, а если обработчика нет, за него отреагирует система. Как правило, это приводит к немедленному завершению процесса, получившего сигнал. Обработчик сигнала запускается асинхронно, немедленно после получения сигнала, что бы процесс в это время не делал. Вызов подпрограммы обработки называется перехватом сигнала.

С помощью сигналов можно осуществлять такие акции управления процессами, как приостановка процесса, запуск приостановленного процесса, завершение работы процесса. Всего в Linux существует 63 разных сигнала, их перечень можно посмотреть по команде `kill -l`.

Сигналы

Сигналы принято обозначать номерами или символическими именами. Все имена сигналов начинаются с приставки SIG, но её иногда опускают: например, сигнал с номером 1 обозначают или как SIGHUP, или просто как HUP.

Можно заставить процесс *игнорировать* или *блокировать* некоторые сигналы. Игнорируемый сигнал просто отбрасывается процессом и не оказывает на него никакого влияния. Блокированный сигнал ставится в очередь, но ядро не требует от процесса никаких действий до разблокирования сигнала. После разблокирования сигнала программа его обработки вызывается только один раз, даже если в течение периода блокировки данный сигнал поступал несколько раз. Часто встречающиеся сигналы приведены в табл. 1.

Номер сигнала	Имя сигнала	Описание сигнала	Можно перехватывать	Можно блокировать	Комбинация клавиш для отправки сигнала
1	HUP	Hangup - отбой	Да	Да	
2	INT	Interrupt - в случае выполнения простых команд вызывает прекращение выполнения, в интерактивных программах — прекращение активного процесса	Да	Да	<Ctrl>+<C> или
3	QUIT	Как правило, сильнее сигнала Interrupt	Да	Да	<Ctrl>+<\>

4	ILL	Illegal Instruction - центральный процессор столкнулся с незнакомой командой (в большинстве случаев это означает, что допущена программная ошибка). Сигнал отправляется программе, в которой возникла проблема	Да	Да	
8	FPE	Floating Point Exception - вычислительная ошибка, например, деление на ноль	Да	Да	
9	KILL	Всегда прекращает выполнение процесса	Нет	Нет	
11	SEGV	Segmentation Violation - доступ к недопустимой области памяти	Да	Да	
13	PIPE	Была предпринята попытка передачи данных с помощью конвейера или очереди FIFO, однако не существует процесса, способного принять эти данные	Да	Да	
15	TERM	Software Termination - требование закончить процесс (программное завершение)	Да	Да	
17	CHLD	Изменение статуса порожденного процесса	Да	Да	
18	CONT	Продолжение выполнения приостановленного процесса	Да	Да	
19	STOP	Приостановка выполнения процесса	Нет	Нет	
20	TSTR	Сигнал останова, генерируемый клавиатурой. Переводит процесс в фоновый режим	Да	Да	<Ctrl>+<Z>

Сигналы Два сигнала – 9 (KILL) и 19 (STOP) всегда обрабатывает система. Первый из них нужен для того, чтобы уничтожить процесс наверняка (отсюда и название). Сигнал STOP приостанавливает процесс: в таком состоянии процесс не удаляется из таблицы процессов, но не выполняется до тех пор, пока не получит сигнал 18 (CONT) – после чего продолжит работу. В Linux сигнал STOP можно передать с помощью комбинации клавиш Ctrl+Z.

Передать любой сигнал из командной строки можно любым процессам с помощью команды

```
kill -сигнал PID.
```

Команда

```
kill PID
```

передает сигнал 15 (TERM).

Команды для управления процессами В табл. 2 приведены основные команды для управления процессами.

Таблица 2. Основные команды для управления процессами

Команда	Описание
at	Выполняет команды в определенное время
atq	Выводит список всех заданий, поставленных на выполнение командой atw3
atrm	Удаляет задание из очереди команды at
batch	Выполняет команды тогда, когда позволяет загрузка системы
cron	Выполняет команды по заранее заданному расписанию
crontab	Позволяет работать с файлами crontab отдельных пользователей
kill	Прекращает выполнение процесса
nice	Изменяет приоритет процесса перед его запуском
nohup	Позволяет работать процессу после выхода пользователя из системы
ps	Выводит информацию о процессах
top	Моментальный снимок процессов, запущенных в системе
renice	Изменяет приоритет работающего процесса
w	Показывает, кто в настоящее время работает в системе и с какими программами

at Команда at позволяет передавать задания демону cron для однократного выполнения в назначенное время. Выдавая задание, команда at сохраняет в отдельном файле как его текст, так и все текущие переменные среды, чего не делает команда cron. По умолчанию все результаты выполнения задания направляются пользователю в виде электронного сообщения.

at Как и в случае с `crontab` суперпользователь может контролировать, кому разрешено или запрещено выполнять команду `at`. Пользователям разрешается использовать команду `at`, если их регистрационные имена указаны в файле `/etc/at.allow`. Если этот файл не существует, проверяется файл `/etc/at.deny`, чтобы определить, не запрещен ли пользователю доступ к `at`.

Базовый формат команды `at`:

`at [-f файл] [-l -d -m] время`

- `f` файл – список заданий должен быть взят из указанного файла;
- `l` – вывод на экран списка заданий; аналог команды `atq`;
- `d` – удаление задания с указанным номером; аналог команды `atrm` (в некоторых системах заменяется опцией `-r`);
- `m` – выдача пользователю сообщения по электронной почте о завершении задания;

Время – определение времени выполнения задания. Допускается указание не только времени в формате часы:минуты, но и даты, а также многочисленных ключевых слов, таких как названия дней недели, месяцев, слов `today` (сегодня), `tomorrow` (завтра), `now` (сейчас) и др. Часто удобна запись вида `now+3 hours` (через 3 часа).

at Ниже приведены примеры корректного указания времени при вызове команды `at`:

`at 6.45am May 12` – 12 мая в 6:45 утра;

`at 11.10pm` – в 23:10;

`at now+1 hour` – через час;

`at 9am tomorrow` – завтра в 9:00 утра;

`at 15:00 May 24` – 24 мая в 15:00;

`at 4am+3 days` – через 3 дня в 4:00 утра.

Следующую команду можно вводить с терминала:

```
$ at 07:30 tomorrow
```

```
at> sort <file> outfile
```

```
at><EOT>
```

Строка `<EOT>` появляется после нажатия клавиш `Ctrl+D`.

Если необходимо с помощью команды `at` запустить файл сценария, то его имя необходимо указать после опции `-f`, например:

```
at 3:00pm tomorrow -f /apps/bin/db_table.sh
```

Для того чтобы просмотреть список запланированных заданий, используются команды `at -l` или `atq`.

В первом столбце вывода этих команд содержится идентификатор задания, за которым следуют дата и время его выполнения. В последнем столбце находится символ «a», указывающий на то, что задание получено от команды `at`. Если в последнем столбце расположен символ «b», то это указывает, что задание получено от команды `batch`, которая планирует выполнения задания на период наименьшей загрузки системы.

at Для удаления запланированного задания используется команда `atrm` (синоним команды `at -d`), имеющая следующий формат:
`atrm номер_задания`

batch В отличие от `at`, `batch` позволяет операционной системе самой решить, когда наступает подходящий момент для запуска задачи в то время, когда система наименее загружена.

Формат команды `batch` представляет собой список команд для выполнения, следующих в строках за командой. Заканчивается список нажатием комбинации клавиш `Ctrl+D`. Можно поместить список команд в файл и перенаправить его на стандартный ввод команды `batch`

Следующую команду можно вводить с терминала:

```
$ batch  
sort <file> outfile  
<EOT>
```

cron Команда **cron** (/usr/sbin/cron) запускает процесс, выполняющий команды в указанные дни и время. Регулярно выполняемые команды задаются инструкциями в файлах **crontab** в каталоге **/var/spool/cron/crontabs**. Пользователи могут задавать собственные файлы **crontab** с помощью команды **crontab**.

Однако часто суперпользователь запрещает использование **cron** обычными пользователями. В этом случае суперпользователь создает вспомогательные файлы **cron.deny** и **cron.allow**, содержащие списки пользователей, которым соответственно запрещено и разрешено использовать команду **crontab**.

Каждая строка в файле **crontab** содержит шаблон времени и команду. Команда выполняется тогда, когда текущее время соответствует приведенному шаблону. Шаблон состоит из пяти частей, разделенных пробелами или символами табуляции, и имеет вид:

минуты часы день_месяца месяц день_недели задание
Эти пять полей должны обязательно присутствовать в файле. Для того, чтобы **cron** игнорировал поле шаблона времени, необходимо поставить в нем символ «*». Например, шаблон 10 01 01 * * означает, что программа должна быть запущена в десять минут второго каждого первого числа любого (*) месяца, каким бы днем недели оно ни было. В таблице 3 приведены поля таблицы задания **cron**.

Поле	Описание
минуты	Минуты в течение часа. Значения от 0 до 59
часы	Час запуска задания. Значения от 0 до 23, где 0 – полночь
день_месяца	День месяца, в который должна выполняться команда
месяц	Месяц, в который необходимо запустить задание. Значение лежит в пределах от 1 до 12
день_недели	День недели в виде цифр от 0 до 7 (0 и 7 означают воскресенье) или первых трех букв, например Mon
задание	Командная строка для запуска задания

Демон **cron** проверяет файл **crontab** и файлы команд **at** только при инициализации и при выполнении команд **crontab** или **at**. Это сокращает накладные расходы на проверку новых или измененных файлов по сравнению с регулярной проверкой.

Демон **cron** перехватывает данные, выдаваемые заданием в стандартный выходной поток и в стандартный поток ошибок, и, если туда что-нибудь выдано, посылает результат пользователю по электронной почте. Если задание не выдало никаких результатов в эти потоки, сообщение пользователю не посылается (если только речь не идет о задании **at**, при посылке которого была указана опция **-m**).

cron *Установка стандартных значений cron*

Для регистрации в журнале всех действий, выполненных демоном cron, необходимо указать параметр **CRONLOG=YES** (задан по умолчанию) в файле `/etc/default/cron`. Если задано значение **CRONLOG=NO**, регистрация не выполняется. Поддержку регистрации можно включать и отключать, поскольку обычно демон cron создает огромные журнальные файлы. Значение переменной **PATH** для пользовательских заданий cron можно установить с помощью присваивания **PATH=** в файле `/etc/default/cron`. Значение **PATH** для заданий пользователя root можно установить с помощью присваивания **SUPATH=** в файле `/etc/default/cron`. Следует продумать последствия использования соответствующих значений **PATH** и **SUPATH** для защиты системы.

Пример файла `/etc/default/cron`:

```
CRONLOG=YES
```

```
PATH=/usr/bin:/usr/ucb:
```

В этом примере установлено журналирование и задано стандартное значение **PATH** для заданий непривилегированных пользователей, `/usr/bin:/usr/ucb:`. Для заданий root по-прежнему будет использоваться путь поиска выполняемых файлов `/usr/sbin:/usr/bin`.

Сценарий `/etc/cron.d/logchecker` проверяет, не превышает ли размер файла журнала предел, установленный в системе. Если превышает, то журнал перемещается в файл `/var/cron/olog`.

cron **Используемые файлы**

/etc/cron.d - основной каталог **cron**;

/etc/cron.d/FIFO - используется в качестве файла блокировки;

/etc/default/cron - содержит стандартные значения для демона **cron**;

/var/cron/log - журнал демона **cron**;

/var/spool/cron - область сброса;

/etc/cron.d/logchecker - перемещает файл журнала в **/var/cron/olog**, если его размер превышает установленный в системе предел.

/etc/cron.d/queuedefs - файл описания очередей для команд **at**, **batch** и **cron**.

Примеры команд, запускаемых cron

01 * * * * /usr/bin/script – команда **/usr/bin/script** запускается в первую минуту каждого часа;

20 8 * * * /usr/bin/script - команда **/usr/bin/script** запускается каждый день в 8 часов 20 минут;

00 6 * * 0 /usr/bin/script - команда **/usr/bin/script** запускается в 6 часов каждое воскресенье;

40 7 1 * * /usr/bin/script - команда **/usr/bin/script** запускается каждый в 7 часов 40 минут каждое первое число месяца.

crontab

crontab - поддержка файлов **crontab** для отдельных пользователей. Используется для добавления, удаления или выведения списка таблиц, используемых для управления демоном **cron**. Каждый пользователь может иметь свои собственные файлы **crontab**, и, хотя эти файлы доступны в `/var/cron/tabs` они не предназначены для редактирования напрямую. Если файл `/var/cron/cron.allow` существует, пользователь должен быть указан в нем, чтобы использовать **crontab**.

Если файл `/var/cron/cron.allow` не существует, но имеется файл `/var/cron/cron.deny`, тогда пользователь не должен быть указан в `/var/cron/cron.deny` чтобы использовать **crontab**. Если оба этих файла отсутствуют, только администратору разрешается использовать **crontab**.

Если файлы `/var/cron/cron.allow` и `/var/cron/cron.deny` существуют, они должны быть доступны группе **crontab** для чтения. Если группа не сможет прочесть эти файлы, пользователям не будет разрешено использовать **crontab**.

Синтаксис команды:

crontab [-u пользователь] [-l | -r | -e]

Доступны следующие опции:

-u пользователь - указывает пользователя, чей **crontab** должен быть отредактирован. Если эта опция не указана, использует **crontab** пользователя, выполняющего команду. Обратите внимание, что **su** может запутать и при работе через **su** всегда следует использовать опцию -u для безопасности.

crontab

- **l** – отображает текущий файл crontab;
- **g** – удаляет текущий файл crontab;
- **e** – изменяет файл crontab редактором, указанным в переменной окружения EDITOR.

При работе с собственным файлом crontab опцию **-u** можно не указывать.

Еще до того как файл crontab будет помещен в очередь заданий программы cron, необходимо установить переменную окружения EDITOR. Чтобы установить vi в качестве редактора, откройте файл .profile, находящейся в домашнем каталоге и поместите в него команды

```
EDITOR=vi; export EDITOR
```

Далее создайте файл <имя_пользователя>cron (например, ivancron), где <имя_пользователя> - ваше регистрационное имя, в котором укажите команды для cron.

Чтобы поместить в очередь планировщика заданий свой файл crontab, выполните команду

```
crontab <имя_пользователя>cron,
```

например:

```
crontab ivancron
```

Копия файла будет помещена в каталог /var/spool/cron, а имя копии будет совпадать с регистрационным именем (например, ivan).

Для вывода на экран содержимого cron-файла используется команда `crontab -l`.

Чтобы отредактировать cron-файл используется команда `crontab -l`. Для редактирования будет использоваться редактор, указанный в переменной окружения EDITOR. При сохранении файла cron проверяет значения полей и информирует

crontab об обнаруженных ошибках. При наличии ошибок файл не будет принят.
Для удаления своего cron-файла используется команда `crontab -r`.
Если cron-файл удален случайно, то установите заново исходный файл с помощью команды `crontab <имя_пользователя>cron`.
Используемые файлы
`/var/cron/cron.allow` - список пользователей, которым разрешено использовать `crontab`;
`/var/cron/cron.deny` - список пользователей, которым запрещено использовать `crontab`;
`/var/cron/tabs` - каталог с индивидуальными `crontab` файлами.

nohup Задание, выполняющееся в фоновом режиме, уничтожается, когда запустивший его пользователь выходит из системы. Команда `nohup` позволяет фоновому процессу продолжать свою работу даже тогда, когда пользователь отключился от терминала. Команда `&` сделать этого не позволяет. Формат команды:
`nohup выполняемая_фоновая_команда &`

ps Команда предназначена для получения информации о существующих в операционной системе процессах. GNU-версия этой программы, входящая в состав Linux, поддерживает опции в стиле трех разных типов UNIX. Опции в стиле Unix98 состоят из одного или нескольких символов, перед которыми должен стоять дефис. Опции в стиле BSD имеют аналогичный вид, только используются без дефиса. Опции, характерные только для GNU-версии представляют собой слово, перед которым должно стоять два дефиса. Их нельзя объединять, как однобуквенные опции двух предшествующих типов. Таким образом, существует три равноправных формата задания этой команды:
`ps [-опции]`
`ps [опции]`
`ps [-- длинное_имя_опции [-- длинное_имя_опции] ...]`

- ps** При этом опции разных типов нельзя употреблять в одной команде.
- Первая группа опций регулирует вывод команды. Независимо от наличия опций этой группы команда ps выдает для каждого процесса отдельную строку, но содержимое этой строки может быть разным. В зависимости от заданных опций могут присутствовать следующие поля:
- USER – имя владельца процесса;
 - PID – идентификатор процесса в системе;
 - PPID – идентификатор родительского процесса;
 - %CPU – доля времени центрального процессора (в процентах), выделенного данному процессу;
 - %MEM – доля реальной памяти (в процентах), используемая данным процессом;
 - VSZ – виртуальный размер процесса (в килобайтах);
 - RSS – размер резидентного набора (количество 1К-страниц в памяти);
 - STIME – время старта процесса;
 - TTY – указание на терминал, с которого запущен процесс;
 - S или STAT – статус процесса;
 - PRI – приоритет планирования;
 - NI – значение nice;
 - TIME – сколько времени центрального процессора занял данный процесс;
 - CMD или COMMAND – командная строка запуска программы, выполняемой данным процессом;
 - другие поля, полный список которых приведен на map-странице, посвященной команде ps.

ps В поле Статус процесса могут располагаться следующие значения:

- R – выполнимый процесс, ожидающий только момента, когда планировщик задач выделит ему очередной квант времени;
- S – процесс «спит»;
- D – процесс находится в состоянии подкачки на диске;
- T – остановленный процесс;
- Z – процесс-зомби.

Рядом с указателем статуса могут стоять дополнительные символы из следующего набора:

- W – процесс не имеет резидентных страниц;
- < – высоко-приоритетный процесс;
- N – низко-приоритетный процесс;
- L – процесс имеет страницы, заблокированные в памяти.

Вторая группа опций определяет, какие процессы включаются в вывод команды. Чтобы получить список всех процессов необходимо использовать команду ps с опциями ax или -A. Вывод в этих двух случаях отличается только в поле CMD: в первом случае выдается полная командная строка запуска программы, а во втором – только имя запущенной программы. Описание всех опций программы ps привести невозможно. Поэтому приведем только несколько примеров ее применения, которые покажут, как пользоваться этой командой в типичных ситуациях.

Для того чтобы увидеть все процессы в системе, используя стандартную форму вывода:

ps -e

ps Можно к команде добавить опцию `-o`, после которой указать через запятую, какие именно поля вы хотите видеть в выводе команды:

```
ps -eo pid,user,cmd
```

Для того, чтобы увидеть все процессы в системе, используя форму вывода BSD-систем:

```
ps ax
```

Для того, чтобы увидеть все процессы в системе, с применением графического отображения отношения «предок-потомок»:

```
ps -ef
```

Впрочем, для того, чтобы увидеть «лес» деревьев «предок-потомок», лучше воспользоваться очень интересным аналогом команды `ps -ef` — командой `pstree`.

Для того, чтобы увидеть, сколько % ЦПУ и памяти занимают запущенные вами процессы:

```
ps -u
```

Чтобы узнать приоритет процесса и значение `nice`, воспользуйтесь опцией `-l`:

```
ps -l
```


top

PID	USER	PRI	NI	SIZE	RSS	SHARE	STAT	%CPU	%MEM	TIME	COMMAND
1102	kos	17	0	83148	75M	69052	R	94,8	40,6	52:06	vmware
937	root	9	0	21868	14M	8520	S	2,1	7,7	2:59	X
1099	kos	9	0	3876	3340	3136	S	1,3	1,7	0:02	kdeinit
1284	root	15	0	1164	1164	920	R	1,1	0,6	0:00	top
1092	kos	9	0	4652	4176	3612	S	0,5	2,1	0:06	kdeinit
1286	kos	9	0	720	720	608	S	0,3	0,3	0:00	xwd
1	root	8	0	132	80	80	S	0,0	0,0	0:03	init
2	root	9	0	0	0	0	SW	0,0	0,0	0:00	keventd
3	root	9	0	0	0	0	SW	0,0	0,0	0:34	kpm-idled
4	root	9	0	0	0	0	SW	0,0	0,0	0:00	kswapd
5	root	9	0	0	0	0	SW	0,0	0,0	0:00	kreclamd
6	root	9	0	0	0	0	SW	0,0	0,0	0:00	bdflush
7	root	9	0	0	0	0	SW	0,0	0,0	0:01	kupdated
591	rpc	9	0	172	124	124	S	0,0	0,0	0:00	portmap
623	root	9	0	288	244	244	S	0,0	0,1	0:00	syslogd
631	klogd	9	0	804	168	168	S	0,0	0,0	0:00	klogd

Рисунок 1. Вывод команды top

Команда ps позволяет сделать моментальный снимок процессов, запущенных в системе. В отличие от ps команда top отображает состояние процессов и их активность в реальном времени. На рис. 1 показано окно, в котором запущена программа top.

В верхней части вывода отображается астрономическое 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.

Далее отображается таблица, описывающая отдельные процессы.

Графы таблицы аналогичны поля вывода команды ps.

Содержимое окна обновляется каждые 5 секунд. Список процессов может быть отсортирован по используемому времени ЦПУ (по умолчанию), по использованию памяти, по PID, по времени исполнения. Переключать режимы отображения можно с помощью команд, которые программа top воспринимает. Это следующие команды:

- <Shift>+<N> – сортировка по PID;
- <Shift>+<A> – сортировать процессы по возрасту;
- <Shift>+<P> – сортировать процессы по использованию ЦПУ;
- <Shift>+<M> – сортировать процессы по использованию памяти;
- <Shift>+<T> – сортировка по времени выполнения.

top Кроме команд, определяющих режим сортировки, команда **top** воспринимает еще ряд команд, которые позволяют управлять процессами в интерактивном режиме. С помощью команды <K> можно завершить некоторый процесс (его PID будет запрошен), а с помощью команды <R> можно переопределить значение `nice` для некоторого процесса. Таким образом, эти две команды аналогичны командам **kill** и **renice**.

kill **kill** - завершить процессы или послать им сигнал. Утилита **kill** посылает сигнал процессу или процессам, заданным операндами **pid**. Процесс, которому посылается сигнал, должен принадлежать текущему пользователю, но суперпользователь (`root`) может посылать сигналы любым процессам.

`/usr/bin/kill -s сигнал pid...`

[Разрыв обтекания текста] `/usr/bin/kill -l [статус_выхода]`

[Разрыв обтекания текста] `/usr/bin/kill [-сигнал] pid ...`

Утилита **kill** посылает сигнал процессу или процессам, заданным операндами **pid**.

Поддерживаются следующие опции:

-l - выдать все значения сигналов, поддерживаемые в данной реализации, если операнды не указаны. Если указан операнд `статус_выхода`, соответствующий значению специального параметра `? командного интерпретатора и вызову wait` для процесса, работа которого прекращена сигналом, выдать имя сигнала, которым была прекращена работа процесса. Если указан операнд `статус_выхода`, представляющий собой целое число без знака - номер сигнала, - выдать имя

kill соответствующего сигнала. В противном случае, результат не определен.

-s сигнал – задает сигнал, который надо послать, используя одно из символьных имен. Значение сигнала распознается независимо от регистра символов. При этом префикс **SIG** указывать не надо. Кроме того, распознается символьное имя 0, представляющее сигнал со значением ноль. Указанный сигнал будет посылаться вместо стандартного **SIGTERM**. Поддерживаются следующие операнды:

pid - одно из следующих значений:

- 1) десятичное целое, задающее процесс или группу процессов, которым надо послать сигнал. Какие процессы выбираются при указании положительного, отрицательного числа или нуля в качестве значения **pid** см. в описании функции **kill**. Если указан процесс номер 0, посылается сигнал всем процессам в соответствующей группе процессов. Если первый операнд **pid** - отрицательный, перед ним надо указать два дефиса (--), чтобы он не интерпретировался как опция;
- 2) идентификатор задания системы управления заданиями, определяющий фоновую группу процессов, которой надо послать сигнал. Идентификаторы заданий можно указывать только для вызовов **kill** в среде выполнения текущего командного интерпретатора.

Номера (идентификаторы) процессов можно найти с помощью команды **ps**.

kill Примеры использования команды kill

Команды:

```
kill -9 100 -165
```

```
kill -s kill 100 -165
```

```
kill -s KILL 100 -165
```

посылают сигнал **SIGKILL** процессу с идентификатором 100 и всем процессам, идентификатор группы процессов которых равен 165, если, конечно, посылающий процесс имеет право отправки сигнала этим процессам, и процессы с соответствующими идентификаторами существуют.

Во избежание двусмысленности, если первый аргумент - отрицательное число, задающее номер сигнала или группы процессов, всегда предполагается, что это номер сигнала.

Поэтому, чтобы послать стандартный сигнал группе процессов (например, 123), надо использовать команды следующего вида:

```
kill -TERM -123
```

```
kill -- -123
```

nice nice - запускает программу с заданием приоритета.

Приоритет процесса определяется так называемым «значением nice», которое лежит в пределах от +20 (наименьший приоритет, процесс выполняется только тогда, когда ничто другое не занимает процессор), до -20 (наивысший приоритет).

nice Значение `nice` устанавливается для каждого процесса в момент порождения этого процесса и при обычном запуске команд или программ принимается равным приоритету родительского процесса. Но существует специальная команда `nice`, которая позволяет изменять значение `nice` при запуске программы. Формат использования этой программы:
`nice [- adnice] command [args]`
 где `adnice` – значение (от `-20` до `+19`), добавляемое к значению `nice` процесса-родителя. Полученная сумма и будет значением `nice` для запускаемого процесса. Отрицательные `adnice` значения может устанавливать только суперпользователь. Если опция `- adnice` не задана, то по умолчанию для процесса-потомка устанавливается значение `nice`, увеличенное на 10 по сравнению со значением `nice` родительского процесса.

Пример использования

```
$ nice yes > /dev/null &
[1] 5199
```

```
$ ps -l
```

```

F S  UID  PID  PPID  C PRI NI ADDR SZ WCHAN  TTY      TIME
CMD
0 S  1000  3383  3380  0  80   0 - 6445 wait  pts/0   00:00:00 bash
0 R  1000  5199  3383  99  90  10 - 1757 -   pts/0   00:00:07 yes
0 R  1000  5200  3383   0  80   0 - 2399 -   pts/0   00:00:00 ps

```

nice Чтобы запустить процесс со значением nice, отличным от 10, можно использовать ключ -n.

```
$ nice -n 15 yes > /dev/null &
```

или

```
$ nice -15 yes > /dev/null &
```

```
[1] 5270
```

```
$ ps -l
```

```
F S  UID  PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY      TIME
CMD
```

```
0 S  1000  3383  3380  0  80   0 - 6447 wait  pts/0   00:00:00 bash
```

```
0 R  1000  5270  3383  99  95  15 - 1757 -   pts/0   00:00:02 yes
```

```
0 R  1000  5271  3383  0  80   0 - 2399 -   pts/0   00:00:00 ps
```

Чтобы установить значение nice ниже нуля, требуются права суперпользователя. В противном случае будет установлено значение 0. Ниже мы пробуем задать значение nice -1 без прав root:

```
$ $ nice -n -1 yes > /dev/null &
```

```
[1] 5285
```

```
nice: cannot set niceness: Permission denied
```

```
$ ps -l
```

```
F S  UID  PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY      TIME
CMD
```

```
0 S  1000  3383  3380  0  80   0 - 6447 wait  pts/0   00:00:00 bash
```

```
0 R  1000  5285  3383  95  80   0 - 1757 -   pts/0   00:00:07 yes
```

```
0 R  1000  5295  3383  0  80   0 - 2399 -   pts/0   00:00:00 ps
```

nice Поэтому, чтобы задать значение nice меньше 0, необходимо запускать программу как root, или использовать sudo.

```
# nice -n -1 yes > /dev/null &
[1] 5537
```

```
# ps -l
```

```
F S  UID  PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY      TIME
CMD
4 S   0  5428  3383  0  80   0 - 14430 wait  pts/0   00:00:00 su
0 S   0  5436  5428  1  80   0 - 7351 wait  pts/0   00:00:00 bash
4 R   0  5537  5436  87  79  -1 - 1757 -   pts/0   00:00:04 yes
4 R   0  5538  5436  0  80   0 - 2399 -   pts/0   00:00:00 ps
```

renice Команда, renice, служит для изменения значения nice для уже выполняющихся процессов. Формат команды:

```
renice priority [[-p] PID] [[-g] grp] [[-u] user]
```

Например, команда

```
renice -1 987 -u daemon -p 32
```

увеличивает на 1 приоритет процессов с PID 987 и 32, а также всех процессов пользователя daemon.

Суперпользователь может изменить приоритет любого процесса в системе. Другие пользователи могут изменять значение приоритета только для тех процессов, для которых данный пользователь является владельцем. При этом обычный пользователь может только уменьшить значение приоритета (увеличить значение nice), но не может увеличить приоритет, даже для возврата значения nice к значению,

renice устанавливаемому по умолчанию. Поэтому процессы с низким приоритетом не могут породить высокоприоритетных потомков.

Пример использования

Есть работающая программа `yes` со значением `nice` 10:

```
$ ps -l
```

```
F S UID PID PPID C PRI NI ADDR SZ WCHAN TTY TIME  
CMD
```

```
0 S 1000 3383 3380 0 80 0 - 6447 wait pts/0 00:00:00 bash
```

```
0 R 1000 5645 3383 99 90 10 - 1757 - pts/0 00:00:04 yes
```

```
0 R 1000 5646 3383 0 80 0 - 2399 - pts/0 00:00:00 ps
```

Чтобы изменить его значение, мы можем использовать команду `renice` со значением `nice` и PID процесса. Давайте изменим значение `nice` на 15:

```
$ renice -n 15 -p 5645
```

```
5645 (process ID) old priority 10, new priority 15
```

```
$ ps -l
```

```
F S UID PID PPID C PRI NI ADDR SZ WCHAN TTY TIME  
CMD
```

```
0 S 1000 3383 3380 0 80 0 - 6447 wait pts/0 00:00:00 bash
```

```
0 R 1000 5645 3383 99 95 15 - 1757 - pts/0 00:00:31 yes
```

```
0 R 1000 5656 3383 0 80 0 - 2399 - pts/0 00:00:00 ps
```


renice Обычный пользователь может только увеличивать значение nice (уменьшать приоритет) любого процесса. Если попробовать изменить значение nice с 15 до 10, получим следующее сообщение об ошибке:

```
$ renice -n 10 -p 5645
```

```
renice: failed to set priority for 5645 (process ID): Permission denied
```

Также, команда **renice** позволяет суперпользователю изменять значение nice процессов любого пользователя. Это делается с помощью ключа **-u**. Следующая команда изменяет значение приоритета всех процессов пользователя на **-19**:

```
# renice -n -19 -u lubos
```

```
1000 (user ID) old priority 0, new priority -19
```

w **w** - выводит информацию о работающих в данный момент на машине пользователях и об их процессах.

w - [husfv] [user]

Заголовок показывает в следующем порядке: текущее время, сколько времени работает система, сколько пользователей в данный момент работают и среднее время загрузки системы за последние 1, 5 и 15 минут.

Для каждого пользователя выводятся следующие записи: регистрационное имя, название терминала (tty), удалённая машина, время регистрации в системе, время простоя, JCPU, PCPU и командную строку его текущего процесса.

w Время JCPU - это время, использованное всеми процессами, закреплёнными за tty. Оно не включает завершённые фоновые задания, но включает фоновые задания, выполняющиеся в данный момент.

Время PCPU - это время, использованное текущим процессом, указанным в поле «what» («что»).

Доступные опции

- **h** – не выводить заголовков;
- **u** – игнорировать имена пользователей при определении времени текущего процесса и cpu;
- **s** – использовать короткий формат. Не выводит время регистрации, время JCPU и PCPU;
- **f** – включить или выключить вывод поля from (имя удалённой машины);
- **V** – вывести информацию о версии;
- user – показать информацию только об указанном пользователе.

Используемые файлы

/var/run/utmp информация о пользователях, зарегистрированных в данный момент;

/proc - информация о процессах.

Основы системного администрирования и сетевых технологий
