Управление процессами

В этой главе вы узнаете, как запускать процессы в фоновом режиме, а также как переключать процессы между фоном и передним планом. Также будут рассмотрены вопросы управления процессами, посылая им сигналы с помощью команды **kill**. Кроме того, будут представлены различные методы мониторинга ресурсов, которые использует процесс. Наконец, вы увидите, как контролировать приоритет процессов, чтобы влиять на то, сколько вычислительных ресурсов будут использовать процессы.

Контроль над процессом

Как упоминалось в предыдущей главе, выполнение команды приводит к тому, что называется процессом. В операционной системе Linux процессы выполняются с привилегиями пользователя, который выполняет команду. Это позволяет ограничивать процессы определенными возможностями на основе идентичности пользователя. Например, обычно обычный пользователь не может контролировать процессы другого пользователя.

Хотя существуют исключения, обычно операционная система различает пользователей в зависимости от того, являются ли они администратором, также называемым пользователем root, или нет. Пользователи без полномочий root называются обычными пользователями. Пользователи, которые вошли в корневую учетную запись, могут контролировать любые пользовательские процессы, включая остановку любого пользовательского процесса.

Команда ps может использоваться для вывода списка процессов. Помните, что команда ps поддерживает три стиля опций:

Традиционные короткие варианты в стиле UNIX, которые используют один дефис перед символом

Длинные опции в стиле GNU, которые используют два дефиса перед словом

Параметры стиля BSD, которые не используют дефисы и параметры одного символа

```
$ ps
```

```
[root@R-FW ~]# ps
  PID TTY          TIME CMD
 1122 tty1     00:00:00 bash
 1344 tty1     00:00:00 ping
 2847 tty1     00:00:00 ps
```

Команда **ps** отобразит процессы, которые выполняются в текущем терминале по умолчанию. Вывод включает в себя следующие столбцы информации:

PID: идентификатор процесса, который является уникальным для процесса. Эта информация полезна для управления процессом по его идентификационному номеру.

TTY: имя терминала или псевдотерминала, на котором запущен процесс. Эта информация полезна для различения разных процессов с одинаковыми именами.

TIME: общее количество процессорного времени, используемого процессом. Как правило, эта информация не используется обычными пользователями.

CMD: команда, которая запустила процесс.

Когда команда **ps** запускается с опцией стиля BSD, то отображается дополнительный столбец с именем **STAT**, который передает состояние процессов. Существует несколько состояний, в которых процесс может находиться: **D** (непрерывный сон), **R** (запущен), **S** (прерывистый сон), **T** (остановлен) и **Z** (зомби).

При использовании параметра стиля BSD столбец **CMD** заменяется столбцом **COMMAND**, в котором отображается не только команда, но также ее параметры и аргументы.

Чтобы увидеть все процессы текущего пользователя, используйте опцию BSD x:

```
ps x
```

```
PID TTY      STAT   TIME COMMAND
  1 ?        Ss     0:03 /usr/lib/systemd/systemd --system --deserialize 24
  2 ?        S      0:00 [kthreadd]
  3 ?        S      0:01 [ksoftirqd/0]
  5 ?        S<     0:00 [kworker/0:0H]
  6 ?        S      0:03 [kworker/u2:0]
  7 ?        S      0:00 [migration/0]
  8 ?        S      0:00 [rcu_bh]
```

Вместо просмотра только процессов, запущенных в текущем терминале, пользователи могут захотеть просмотреть все процессы, запущенные в системе. При использовании традиционных (не BSD) параметров опция **-e** будет отображать каждый процесс. Как правило, параметр **-f** также используется, поскольку он предоставляет полную информацию о команде, включая параметры и аргументы:

```
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 07:14 ?        00:00:03 /usr/lib/systemd/systemd
root         2     0  0 07:14 ?        00:00:00 [kthreadd]
root         3     2  0 07:14 ?        00:00:01 [ksoftirqd/0]
root         5     2  0 07:14 ?        00:00:00 [kworker/0:0H]
root         6     2  0 07:14 ?        00:00:03 [kworker/u2:0]
root         7     2  0 07:14 ?        00:00:00 [migration/0]
root         8     2  0 07:14 ?        00:00:00 [rcu_bh]
root         9     2  0 07:14 ?        00:00:01 [rcu_sched]
root        10     2  0 07:14 ?        00:00:00 [watchdog/0]
root        12     2  0 07:14 ?        00:00:00 [kdevtmpfs]
root        13     2  0 07:14 ?        00:00:00 [netns]
root        14     2  0 07:14 ?        00:00:00 [khungtaskd]
```

До сих пор команды выполнялись на так называемом переднем (***foreground)*** плане. Для простых команд, которые выполняются быстро и затем выходят, целесообразно использовать приоритетное выполнение. Процесс переднего (***foreground***) плана - это процесс, который не позволяет пользователю использовать оболочку, пока процесс не будет завершен.

Когда один процесс запускает другой, первый процесс называется родительским процессом, а новый процесс называется дочерним процессом. Таким образом, другой способ мышления процесса переднего плана состоит в том, что при запуске на переднем плане дочерний процесс не позволяет выполнять какие-либо дополнительные команды в родительском процессе до тех пор, пока дочерний процесс не завершится.

Команда по-умоллчанию запускается в ***foreground*** режиме.

Обычно пользователи вводят только одну команду в командной строке, но с помощью символа «**;**» (точка с запятой) в качестве разделителя между командами пользователь может вводить несколько команд в одной командной строке. Редко необходимо запускать две или более команд из одной командной строки, но иногда это может быть полезно.

Когда команды разделяются точкой с запятой, выполняется команда слева от точки с запятой; когда он заканчивается, выполняется команда справа от точки с запятой. Другой способ описать это - сказать, что команды выполняются последовательно. Например:

```
echo Hello;sleep 5; echo World
```

Примечание: команда **sleep** выполнит на 5 секунд, прежде чем перейти к следующей команде.

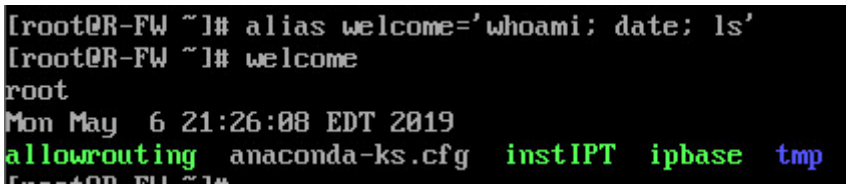
Напомним, что **alias** - это функция, которая позволяет пользователю создавать псевдонимы для команд или командных строк. Наличие псевдонима, который запускает несколько команд, может быть очень полезным, и для этого пользователь должен

использовать **;** между каждой из команд при создании псевдонима. Например, чтобы создать псевдоним с именем **welcome**, который выводит текущего пользователя, дату и текущий список каталогов, выполните следующую команду:

```
alias welcome = "whoami; date; ls"
```

Теперь псевдоним welcome выполнит серию whoami, date и ls следующим образом:

```
welcome
```



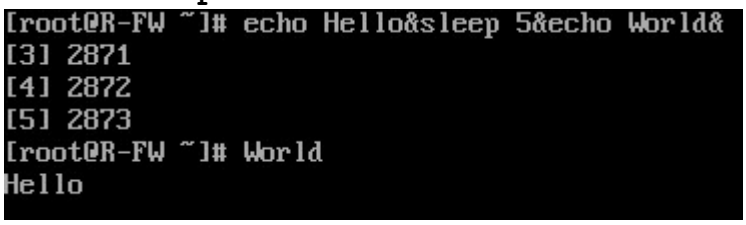
Реальный сценарий, включающий несколько команд, возникает, когда администратору необходимо удаленно отключить и перезапустить сетевое соединение. Если пользователь наберет одну команду, чтобы отключить сеть, а затем выполнит ее, то сетевое соединение будет разорвано, и пользователь не сможет снова восстановить сеть. Вместо этого пользователь может ввести команду для отключения сети, затем точку с запятой, а затем команду для подключения к сети. После нажатия клавиши Enter обе команды будут выполняться, одна за другой.

Фоновые процессы (background)

Когда выполнение команды может занять некоторое время, может быть лучше, чтобы эта команда выполнялась в «фоновом режиме» (**background**). При выполнении в фоновом режиме дочерний процесс немедленно возвращает управление родительскому процессу (в данном случае оболочке), позволяя пользователю выполнять другие команды. Чтобы команда выполнялась как фоновый процесс, добавьте **&** (символ амперсанда) после команды

Чтобы несколько команд выполнялись в фоновом режиме в одной командной строке, поместите амперсанд после каждой команды в этой командной строке. В следующем примере все три команды запускаются почти одновременно и возвращают управление обратно в оболочку, поэтому пользователю не нужно ждать завершения какой-либо из команд перед выполнением другой команды (хотя пользователю может потребоваться снова нажать Enter, чтобы получить подсказка):

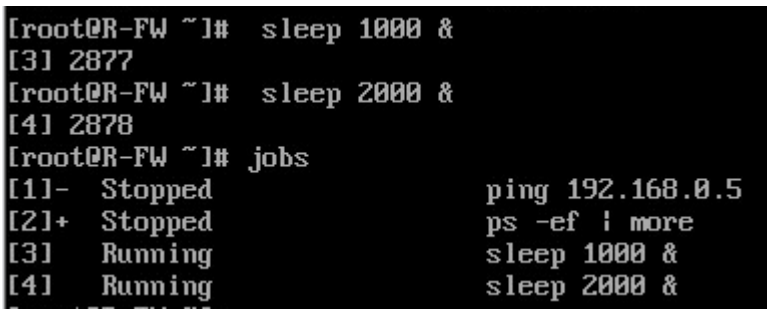
```
echo Hello&sleep 5&echo World&
```



Существует дополнительная разница в выполнении команд в фоновом режиме. После того, как каждая команда начинает выполняться, она выводит [**номер задания**], затем пробел, а затем идентификационный номер процесса (**PID**). Эти числа используются для управления процессом, как вы увидите позже.

После завершения выполнения каждой фоновой команды отображается номер задания, иногда за которым следует символ «-» или «+», слово «**Done**», а затем сама командная строка.

Хотя в терминале по-прежнему выполняются фоновые процессы, их можно отобразить с помощью команды **jobs**. Важно отметить, что команда **jobs** покажет только фоновые процессы в текущем терминале. Если фоновые процессы выполняются в другом терминале, они **не будут** показаны при запуске команды **jobs** в **текущем** терминале. Ниже приведен пример использования команды **jobs**:



Перемещение процессов

Если команда sleep с предыдущей страницы выполняется без амперсанда, терминал не будет доступен в течение 1000 секунд:

```
sleep 1000
```

Чтобы снова сделать терминал доступным, администратор должен будет использовать **CTRL + Z**:

Теперь терминал вернулся, но команда **sleep** приостановлена. Чтобы поместить приостановленную команду в фоновый режим, выполните команду **bg**. Команда **bg** возобновляет работу, не выводя ее на передний план.

```
bg
[1] + sleep 1000 &
```

Команда, которая была приостановлена или отправлена в фоновый режим, затем может быть возвращена на передний план с помощью команды **fg**. Чтобы вернуть команду сна на передний план, снова заблокировав терминал, используйте команду **fg**:

```
fg
sleep 1000
```

Предположим, у нас есть два приостановленных процесса:

```
sleep 1000
^Z
[1]+  Stopped                 sleep 1000
sleep 2000
^Z
[2]+  Stopped                 sleep 2000
jobs
[1]-  Stopped                 sleep 1000
[2]+  Stopped                 sleep 2000
```

И **bg**, и **fg** могут принять номер задания в качестве аргумента, чтобы указать, какой процесс должен быть возобновлен. Следующие команды возобновят sleep 1000 в фоновом режиме и возобновят sleep 2000 на переднем плане соответственно:

```
bg 1
[1]- sleep 1000 &
fg 2
sleep 2000
```

Также возможно использовать имя имени команды в качестве аргумента:

```
sleep 1000
^Z
[1]+  Stopped                 sleep 1000
bg sleep
[1]+ sleep 1000 &
```

Учитывая несколько задач и только один терминал для использования с ними, команды **fg** и **bg** предоставляют администратору возможность вручную выполнять несколько задач.

Отправка сигнала

Сигнал - это сообщение, которое отправляется процессу, чтобы сообщить процессу о каком-либо действии, таком как остановка, перезапуск или пауза. Сигналы очень полезны для управления действиями процесса.

Некоторые сигналы могут быть отправлены процессам с помощью простых комбинаций клавиш. Например, чтобы приостановить процесс переднего плана, отправьте сигнал остановки терминала, нажав **CTRL + Z**. Останов Терминала приостанавливает программу, но не полностью останавливает программу. Чтобы полностью остановить процесс переднего плана, отправьте сигнал прерывания, нажав **CTRL + C**.

Существует много разных сигналов, каждый из которых имеет символическое имя и числовой идентификатор. Например, **CTRL + C** назначается символическое имя SIGINT и числовой идентификатор 2.

Чтобы увидеть список всех сигналов, доступных для вашей системы, выполните команду kill -l:

```
[root@R-FW ~]# kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

Эти сигналы могут иметь уникальные значения, характерные для определенной команды (поскольку программист, создавший команду, может регулировать поведение программы), но, как правило, они позволяют останавливать и возобновлять процессы, переконфигурировать процессы или завершать их. процесс. Все сигналы с номером больше 31 предназначены для управления процессами в реальном времени, и эта тема выходит за рамки данного курса. Некоторые из наиболее распространенных сигналов приведены в следующей таблице:

N	ИМЯ	ОПИСАНИЕ	МОЖНО ПЕРЕХВАТЫВАТЬ	МОЖНО БЛОКИРОВАТЬ	КОМБИНАЦИЯ КЛАВИШ
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 ‹Del›
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 недозволенной 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›

Есть несколько команд, которые позволят вам указать сигнал для отправки в процесс; команда **kill** используется чаще всего. Команда **kill** принимает три разных способа указать сигнал:

- Номер сигнала, используемого в качестве опции: **-2**
- Краткое название сигнала, используемого в качестве опции: **-INT**
- Полное имя сигнала, используемого в качестве опции: **-SIGINT**

Все три опции, показанные выше, указывают на сигнал прерывания.

Если пользователь не указывает сигнал с параметром, команда kill отправляет сигнал **Terminate SIGTERM**.

При отправке сигнала укажите один или несколько процессов для отправки сигнала. Существует множество методов для определения процесса или процессов. Более распространенные методы включают в себя:

- Указание идентификатора процесса (**PID**)
- Использование префикса **%** (знак процента) к номеру задания
- Использование опции **-p** вместе с идентификатором процесса (**PID**)

Например, сначала представьте сценарий, в котором пользователь запускает некоторый процесс в фоновом режиме, и этот пользователь хочет отправить сигнал процессу. Для этой демонстрации команда sleep запускается в фоновом режиме:

```
sleep 5000 &
[1] 2901
```

Несколько вещей заслуживают внимания из результатов запуска этого процесса в фоновом режиме. Сначала отметьте номер задания в квадратных скобках: [1]. Во-вторых, обратите внимание на идентификатор процесса (PID), который равен 2901. Чтобы отправить сигнал завершения в этот процесс, вы можете использовать любую из следующих команд:

```
kill 2901
kill %1
kill -p 2901
```

Как указывалось ранее, сигнал завершения обычно завершает процесс. Иногда процесс прерывает сигнал завершения, поэтому он не может завершить этот процесс. Перехват происходит, когда процесс ведет себя иначе, чем норма, когда он получает сигнал; это может быть результатом того, как программист создал код для команды.

Пользователь может попытаться использовать другие сигналы, такие как SIGQUIT или SIGINT, чтобы попытаться завершить процесс, но эти сигналы также могут быть захвачены. Единственный сигнал, который завершит процесс и не может быть пойман в ловушку, - это **SIGKILL**. Поэтому, если другие сигналы не смогли завершить процесс, используйте сигнал **SIGKILL**, чтобы принудительно завершить процесс.

Пользователи не должны обычно использовать сигнал **SIGKILL** в качестве начальной попытки попытаться завершить процесс, потому что это вынуждает процесс завершаться немедленно и не дает процессу возможность «очиститься» после себя. Процессы часто выполняют критические операции, такие как удаление временных файлов, когда они естественным образом завершаются.

В следующих примерах показано, как отправить сигнал «принудительное уничтожение» процессу:

```
kill -9 2901
kill -KILL %1
kill -SIGKILL -p 2901
```

Существуют и другие команды, отправляющие сигналы процессов, такие как команды killall и pkill, которые полезны для одновременной остановки многих процессов; Обычно команда **kill** является хорошим выбором для отправки сигналов одному процессу.

Как и команда **kill**, обе команды **killall** и **pkill** принимают три способа указания конкретного сигнала. В отличие от команды kill, эти другие команды могут использоваться для завершения всех процессов определенного пользователя с параметром -u. Например, **killall -u bob** остановит весь процесс, принадлежащий пользователю "bob".

Команды **killall** и **pkill** также принимают имя процесса вместо идентификатора процесса или задания. Просто будьте осторожны, так как это может в конечном итоге остановить больше процессов, чем вы ожидали. Пример остановки процесса с использованием имени процесса:

```
kill sleep
-bash: kill: sleep: arguments must be process or job IDs
killall sleep
[1]+  Terminated              sleep 5000
```

HUP Сигнал

Когда пользователь выходит из системы, всем процессам, которые принадлежат этому пользователю, автоматически отправляется сигнал зависания **SIGHUP**. Как правило, этот сигнал приводит к завершению этих процессов.

В некоторых случаях пользователь может захотеть выполнить команду, которая не будет автоматически завершаться при отправке сигнала HUP. Чтобы процесс игнорировал сигнал зависания, запустите процесс с помощью команды **nohup**.

Например, рассмотрим сценарий, в котором у пользователя есть скрипт с именем **myjob.sh**, который должен продолжать работать всю ночь. Пользователь должен запустить этот скрипт в фоновом режиме терминала, выполнив:

```
nohup myjob.sh &
```

После выполнения сценария пользователь может перейти к выходу из системы. В следующий раз, когда пользователь войдет в систему, выходные данные скрипта, если они есть, будут находиться в файле **nohup.out** в домашнем каталоге этого пользователя.

Приоритет процесса

Когда процесс выполняется, он должен иметь доступ к процессору для выполнения действий. Не все процессы имеют одинаковый доступ к процессору. Например, системный процесс обычно имеет более высокий приоритет при обращении к ЦП.

Ядро Linux динамически регулирует приоритет процессов, чтобы попытаться заставить операционную систему казаться отзывчивой пользователю и эффективной при выполнении задач. Пользователь может влиять на приоритет, который будет назначен процессу, устанавливая значение, называемое *niceness*

Чем выше вы устанавливаете значение *nice*, тем ниже приоритет, который будет назначен процессу. Значение по умолчанию для процессов равно нулю; большинство пользовательских процессов работают с этим хорошим значением. Только пользователь с административным (root) доступом может установить отрицательные значения niceness или изменить допустимость существующего процесса, чтобы он был более низким значением niceness.

Чтобы установить начальную niceness команды, используйте команду nice в качестве префикса к команде, которую нужно выполнить. Например, чтобы выполнить команду cat /dev/zero> /dev/null с наименьшим возможным приоритетом, выполните следующую команду:

```
nice -n 19 cat / dev / zero> / dev / null
```

Если пользователь входит в систему как пользователь root, он также может выполнить команду **cat /dev/zero>/dev/null** с максимально возможным приоритетом, выполнив следующую команду:

```
nice -n -20 cat / dev / zero> / dev / null
```

Чтобы настроить точность существующего процесса, используйте команду **renice**. Это может быть полезно, когда система становится менее отзывчивой после выполнения команды, интенсивно использующей процессор. Пользователь может сделать систему более отзывчивой, сделав этот процесс более «приятным».

Для этого пользователю нужно будет найти идентификатор процесса для этого процесса с помощью команды **ps**, а затем использовать **renice**, чтобы вернуть приоритет в нормальное состояние. Например:

```
$ su -
Password:
nice -n -20 cat /dev/zero > /dev/null &
[1] 121
ps
  PID TTY          TIME CMD
    1 ?        00:00:00 init
   70 ?        00:00:00 login
  108 ?        00:00:00 su
  109 ?        00:00:00 bash
  121 ?        00:00:04 cat
  122 ?        00:00:00 ps
root@localhost:~# renice -n 0 -p 121
121 (process ID) old priority -20, new priority 0
```

Примечание. Показанная выше команда su позволяет обычному пользователю временно стать пользователем root. Пользователь должен ввести пароль учетной записи **root**, когда это будет предложено.

Мониторинг процессов

Хотя команда **ps** может отображать активные процессы, верхняя команда позволяет отслеживать процессы в режиме реального времени, а также управлять процессами. Например, на следующем рисунке показано использование команды **top** для мониторинга запущенных в настоящее время процессов, включая три команды **cat:**

```
top - 22:04:35 up 14:50,  2 users,  load average: 0.08, 0.04, 0.05
Tasks:  92 total,   1 running,  89 sleeping,   2 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.3 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :   500476 total,    45976 free,    79832 used,   374668 buff/cache
KiB Swap:   421884 total,   417420 free,     4464 used.   379440 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 2884 root      20   0       0      0      0 S   0.3   0.0   0:00.46 kworker/0:1
 2898 root      20   0  161880   2212   1568 R   0.3   0.4   0:00.06 top
    1 root      20   0  125480   3584   2236 S   0.0   0.7   0:03.49 systemd
    2 root      20   0       0      0      0 S   0.0   0.0   0:00.02 kthreadd
    3 root      20   0       0      0      0 S   0.0   0.0   0:01.47 ksoftirqd/0
    5 root       0 -20       0      0      0 S   0.0   0.0   0:00.00 kworker/0:0H
    6 root      20   0       0      0      0 S   0.0   0.0   0:03.48 kworker/u2:0
    7 root      rt   0       0      0      0 S   0.0   0.0   0:00.00 migration/0
    8 root      20   0       0      0      0 S   0.0   0.0   0:00.00 rcu_bh
    9 root      20   0       0      0      0 S   0.0   0.0   0:01.91 rcu_sched
   10 root      rt   0       0      0      0 S   0.0   0.0   0:00.60 watchdog/0
```

Команда top имеет множество функций; например, им можно манипулировать в интерактивном режиме. Нажатие клавиши **h** во время выполнения верхней команды приведет к отображению экрана «справки»:

Клавиши k и r используются для управления задачами или процессами в программе **top**.

Нажатие клавиши k позволит пользователю «убить» или отправить сигнал процессу. После нажатия k, top запросит PID, а затем сигнал для отправки этому процессу.

Нажатие клавиши r позволит пользователю «переименовать» процесс, запрашивая PID, а затем новое значение добротности.

Мониторинг системы

Есть также пара команд, которые можно использовать для мониторинга общего состояния системы: команды **uptime** и **free.**

Команда **uptime** показывает текущее время и время, в течение которого система работала, а также количество пользователей, которые в данный момент вошли в систему, и средняя нагрузка за последние один, пять и пятнадцать минут.

Числа, указанные для средних значений нагрузки, основаны на количестве доступных ядер ЦП. Думайте о каждом процессоре как о наличии 100% доступных ресурсов (процессорного времени). Один процессор = 100%, четыре процессора = 400%. Команда **uptime** сообщает об объеме используемых ресурсов, но делится на 100. Таким образом, 1.00 фактически составляет 100% используемого процессорного времени, 2.00 - 200% и так далее.

Если в системе только одно ядро ЦП, значение 1,00 означает, что система полностью загружена задачами. Если система имеет два ядра ЦП, то значение 1,00 будет означать 50% нагрузки (1,00 / 2,00). Чтение времени безотказной работы 1,00 (или 100%) использования на 4-ядерном ЦП означало бы, что используются 1,00 / 4,00 (или 1/4 (или 25%)) общих вычислительных ресурсов ЦП.

Чтобы получить представление о том, насколько занята система, используйте команду uptime:

```
uptime
18:24:58 up 5 days, 10:43,  1 user,  load average: 0.08, 0.03, 0.05
```

Чтобы получить представление о том, как ваша система использует память, полезна команда free. Эта команда отображает не только значения для оперативной памяти (RAM), но также для подкачки, которая представляет собой пространство на жестком диске, которое используется в качестве памяти для незанятых процессов, когда оперативной памяти становится недостаточно.

Команда free сообщает об общем объеме памяти, а также о том, сколько в настоящее время используется и сколько свободно использовать. Вывод также разбивает использование памяти для буферов и кешей:

```
free
```