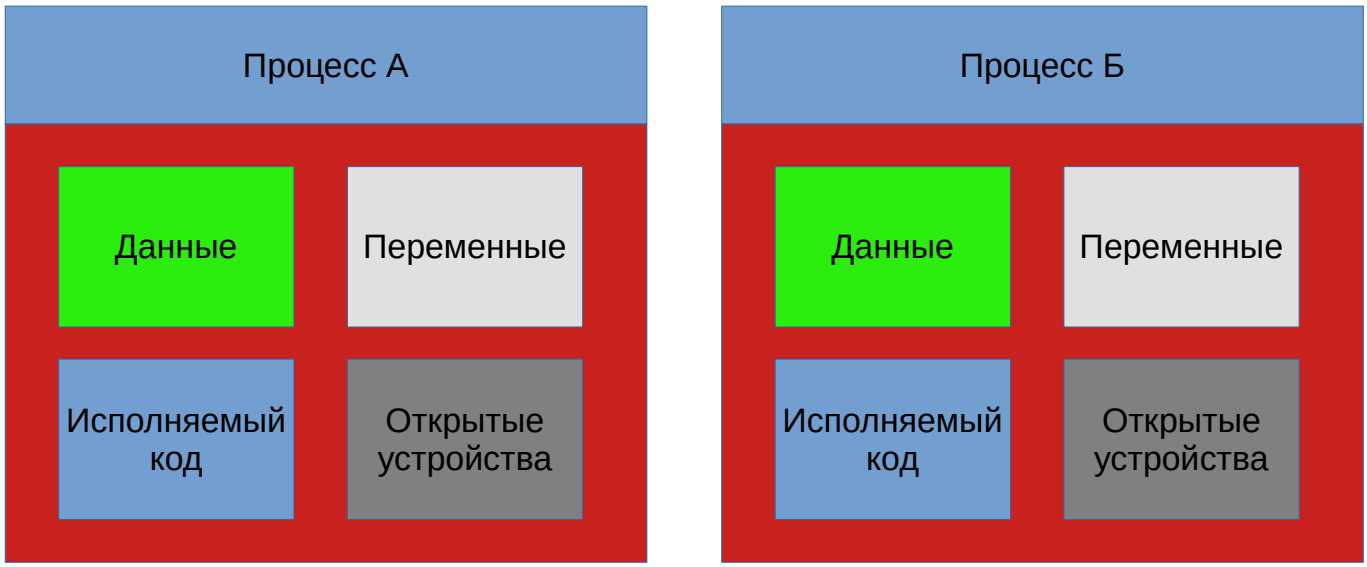
Управление процессами

Понятие процесса



Операционная система Linux является многозадачной, что, в свою очередь означает, что её ядро может управлять работой некоторого (иногда довольно большого) количества запущенных процессов.

Если говорить упрощённо, то процесс - это экземпляр программы, выполняющийся в отдельном адресном пространстве. Это, с одной стороны гарантирует безопасность данных (один процесс не может просто так получить доступ к данным другого процесса), а с другой стороны увеличивает стабильность работы (крах одного процесса никак не скажется на остальных).

Для создания нового процесса используется два системных вызова - fork и execve. fork() создает новое адресное пространство, которое полностью идентично адресному пространству основного процесса. В этот момент дочерний процесс отличается от родительского собственным идентификатором (PID) и идентификатором родительского процесса (PPID). Далее «клон» выполняет системный вызов execve с указанием на исполняемый файл и заменяет свой код - кодом исполняемого файла. Процесс готов к работе.

Из сказанного выше следует, что процессы имеют иерархическую структуру:

у каждого процесса может быть несколько дочерних процессов и один родительский процесс. Исключение составляют процессы, порождённые самим ядром (у них PPID равен нулю)

Идентификаторы процессов имеют непредсказуемое значение и меняются от сеанса к сеансу. Т.е. если мы сейчас перезагрузим систему, идентификаторы процессов, представленных на рисунке 1. изменятся. Исключение составляет процесс systemd, идентификатор которого всегда равен единице.
Этот процесс имеет ещё одну необычную особенность: его нельзя завершить с помощью команды kill, которая будет рассматриваться далее.

```
[demo@localhost ~]$ pstree -p | head -n 25
systemd(1)-+-ModemManager(708)-+-{ModemManager}(740)
           |                   `-{ModemManager}(743)
           |-NetworkManager(784)-+-dhclient(3159)
           |                     |-{NetworkManager}(805)
           |                     `-{NetworkManager}(807)
           |-VBoxClient(1912)---VBoxClient(1914)---{VBoxClient}(1919)            |-
    VBoxClient(1925)---VBoxClient(1927)
           |-VBoxClient(1932)---VBoxClient(1935)---{VBoxClient}(1938)
           |-VBoxClient(1940)---VBoxClient(1942)-+-{VBoxClient}(1944)
           |                                     `-{VBoxClient}(1946)
           |-VBoxService(791)-+-{VBoxService}(793)
           |                  |-{VBoxService}(794)
           |                  |-{VBoxService}(795)
           |                  |-{VBoxService}(796)
           |                  |-{VBoxService}(797)
           |                  |-{VBoxService}(798)
           |                  `-{VBoxService}(800)
           |-abrt-watch-log(704)
           |-abrt-watch-log(706)
           |-abrtd(703)
           |-accounts-daemon(687)-+-{accounts-daemon}(692)
           |                      `-{accounts-daemon}(694)
           |-alsactl(674)
           |-anacron(3725)
           |-at-spi-bus-laun(1970)-+-dbus-daemon(1975)---{dbus-daemon}(1976)
```

Для наблюдения за работой процессов в системе предусмотрен набор специальных утилит. Одну из них мы только что использовали - это команда pstree, показывающая процессы в виде дерева, хотя чаще используют другой инструмент - команду ps, которая выводит данные в виде таблицы. Если запустить команду без каких-либо параметров, то можно обнаружить, что вывод она генерирует довольно скромный (рис. 2) Это происходит потому, что по-умолчанию ps показывает только процессы, запущенные в текущем сеансе оболочки.

```
[demo@localhost ~]$ ps
  PID TTY          TIME CMD
 4383 pts/0    00:00:00 bash
 4427 pts/0    00:00:00 ps
```

Команда ps

Опция	Действие
-e	Показать все процессы (Синоним: опция -A)
-u <пользователь>	Показать все процессы, принадлежащие указанному пользователю, например: ps -u postfix
-t <имя терминала>	Показать все процессы, запущенные в указанном терминале, например: ps -u tty2
-f	Показать больше информации. Опция может комбинироваться с другими. Скажем, вместе с -L она заставляет команду ps показывать количество имеющихся у процесса потоков (колонка NLWP)
-o	"Output". Опция позволяет указать, какие именно поля мы хотим видеть в выводе. (Синоним: опция --format). Например: ps -e -o pid,comm, user (Показать только идентификатор процесса, команду, которой он был запущен и владельца)

Пример использования команды ps:

```
[demo@localhost ~]$ ps -e
  PID TTY          TIME CMD
    1 ?        00:00:02 systemd
    2 ?        00:00:00 kthreadd
    3 ?        00:00:16 ksoftirqd/0
    5 ?        00:00:00 kworker/0:0H
    7 ?        00:00:00 migration/0
    8 ?        00:00:00 rcu_bh
    9 ?        00:00:04 rcu_sched

[demo@localhost ~]$ ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0     0 12:23 ?        00:00:02 /usr/lib/systemd/systemd --switched-root
root         2     0     0 12:23 ?        00:00:00 [kthreadd]
root         3     2     0 12:23 ?        00:00:16 [ksoftirqd/0]
root         5     2     0 12:23 ?        00:00:00 [kworker/0:0H]
root         7     2     0 12:23 ?        00:00:00 [migration/0]
root         8     2     0 12:23 ?        00:00:00 [rcu_bh]
root         9     2     0 12:23 ?        00:00:04 [rcu_sched]
root        10     2     0 12:23 ?        00:00:01 [watchdog/0]
```

В случае, если ps не удаётся определить команду запуска процесса вместе с аргументами, значение выводится в квадратных скобках. Обычно это является свидетельством того, что процесс относится к служебным процессам ядра.

Состояние процесса

Статус	Описание
R (Runing)	Исполняющийся (либо стоящий в очереди на исполнение) процесс
S (Sleeping)	Спящий процесс. В этом состоянии пребывают процессы, ожидающие какого-либо события (определенного времени, действия пользователя, уведомления об освободившимся ресурсе, сигнала от другого процесса и т.п.)
T (Traced or sTopped)	Приостановленный (трассируемый) процесс
Z (Zombie)	Зомби-процесс. Термин является яркой метафорой того, что процесс уже «умер», но ещё не «погребён». При завершении процесс освобождает все свои ресурсы (за исключением PID — идентификатора процесса) и становится «зомби» — пустой записью в таблице процессов, хранящей код завершения для родительского процесса. Система уведомляет родительский процесс о завершении дочернего с помощью сигнала SIGCHLD. Предполагается, что после получения SIGCHLD он считает код возврата, после чего запись зомби будет удалена из списка процессов. Если родительский процесс по каким-то причинам игнорирует SIGCHLD, то зомби остаются до завершения родительского процесса.
D (Dead or Direct)	Процесс, находящийся в непрерывном сне. Т.е. процесс ожидает определенного («прямого») сигнала от аппаратной части и не реагирует на другие сигналы. Довольно частой причиной попадание в это состояние является неготовность подсистемы ввода-вывода обработать запрос процесса.

Отобразить информациюв реальном времени может команда top.
Вверху отображается сводная информация о системе, ниже в табличном виде представлены данные о работающих процессах. Команда top является интерактивной, на генерируемый ей вывод можно влиять следующим образом:

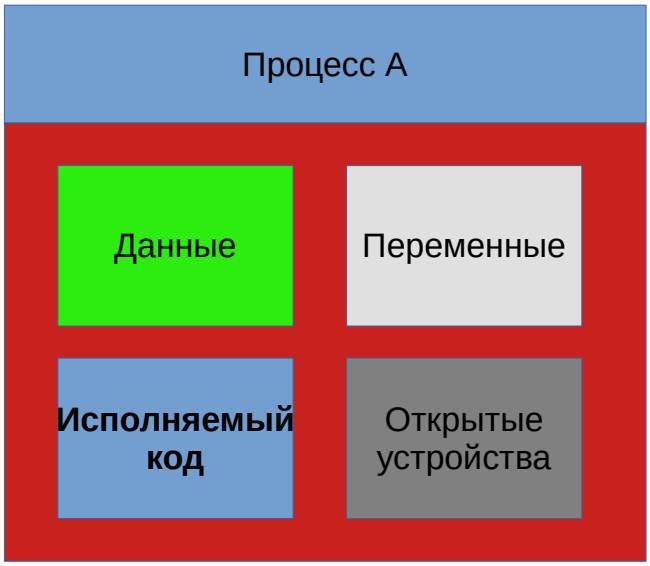
- P - Отсортировать процессы по значению нагрузки, которую они создают на ЦП;
- M - Отсортировать процессы по объёму используемой оперативной памяти;
- L (Locate) - Отыскать процесс с указанным именем;
- U - Показать все процессы, принадлежащие определённому пользователю;
- V - включить древовидное отображение;
- H - включить показ потоков (threads)
- k (kill) - Завершить процесс;
- ПРОБЕЛ - Обновить вывод. По-умолчанию top обновляет вывод автоматически раз в две секунды. Это значение можно переопределить с помощью опции -d или нажав клавишу "d" (delay).
- Alt-h - сдвинуть список влево (помогает, если он не помещается)
- Alt-j - сдвинуть список вниз
- Alt-k - сдвинуть список вверх
- Alt-l - сдвинуть список вправо
- q - Выйти из top

Помимо того, что команда top отображает по умолчанию, к её выводу можно добавлять нужные нам поля. Делается это через команду "f" (field). При нажатии кнопки f top показывает, что именно он может отобразить при необходимости. Уже добавленные в вывод колонки отображаются жирным шрифтом. Переключение производится клавишей "Пробел".

Дополнительные состояния процесса

Статус	Описание
N	Низкоприоритетный процесс
<	Высокоприоритетный процесс
s	Процесс - лидер сессии. Для своих внутренних целей ядро группирует
	процессы и эти группы носят название - сессия.
l	multi-threaded процесс
+	Процесс находится в foreground process group (Это группа процессов, которая получает в данный момент сигналы от терминала).

Потоки



Разговор о процессах будет неполным, если не упомянуть такое понятие, как поток. Дело в том, что создание нового процесса - это довольно ресурсоёмкая задача для ядра. Поэтому, для распараллеливания задач некоторые процессы создают потоки. Поток обычно занимается выполнением какой-то логически обособленной задачи, но в отличие от процесса, не имеет собственного PID. Потоки (в оригинале threads) еще называют легковесными процессами. К недостаткам потоков можно отнести то, что один проблемный поток может повредить остальные, т.к. потоки делят общее адресное пространство. Посмотреть информацию о потоках можно двумя способами: нажав "H" во время работы команды top или с помощью ps: "ps -eLf"

Задание 1

1) Для каждой из описанных ниже ситуаций укажите статус (R,S,T и т.д.)

Процесс временно приостановлен

Процесс ожидает ответа от аппаратного устройства

Процесс ожидает нажатия клавиши на клавиатуре

Процесс находится в очереди на выполнение

Процесс выполняется

2) Запустите команду top. Нажмите "V" для переключения к древовидному виду. Далее, нажмите "H" для включения отображения потоков. Обратите внимание, как изменился список процессов. Параметр "Task" во второй строчке вывода сменился на "Threads". Возросло или уменьшилось число отображаемых объектов?

Управление процессами

Сигнал	Описание	Клавиши
1 (SIGHUP)	Hang up «Отбой». Изначально был предназначен для того, чтобы информировать процесс о потере связи с управляющим терминалом. Многие процессы-демоны, получив этот сигнал, перечитывают свои конфигурационные файлы.	
2 (SIGINT)	Прервать процесс (может быть перехвачен)	Ctrl - c
9 (SIGKILL)	Немедленное завершение процесса. Не может быть перехвачен. Рекомендуется использовать только в случае, когда завершение процесса другими способами невозможно.	
15 (SIGTERM)	Используется по умолчанию. Корректный способ завершить процесс. В отличие от сигнала SIGKILL может быть перехвачен.	
17 (SIGCHLD)	Сигнал, посылаемый при изменении статуса дочернего процесса (завершен, приостановлен или возобновлен).	
18 (SIGCONT)	Возобновить выполнение процесса, если он ранее был приостановлен	
19 (SIGSTOP)	Приостановить процесс. Не может быть проигнорирован процессом.	
20 (SIGTSTP)	Приостановить процесс.	Ctrl - z

Для размещения данных, описывающих работу процессов, ядро использует специальную файловую систему, смонтированную в директорию /proc. Если посмотреть содержимое этой директории, то можно обнаружить там множество поддиректорий, имена которых представляют собой десятичные числа:

```
[demo@localhost ~]$ ls /proc
1        1801  2074  281   404   647   buddyinfo    modules
10       1806  2132  283   4046  649   bus          mounts
1115  1893  2142  284   405   651   cgroups      mtrr
1117  1895  2158  285   406   672   cmdline      net
1119  19      2160  288   4062  673   consoles     pagetypeinfo
1127  1906  2171  289   407   674   cpuinfo      partitions
   1129 1908  2172  290   4071  675   crypto       sched_debug
   1130 1913  2182  291   408   677   devices      schedstat
1133  1916  2183  292   409   685   diskstats    scsi
1149  1921  2188  293   410   688   dma          self
   12 1923  2198  294   411   689   driver       slabinfo
   13 1932  2201  295   412   690   execdomains  softirqs
   14 1951  2207  296   413   691   fb           stat
1429  1956  2216  297   414   692   filesystems  swaps
```

Эти числа - идентификаторы процессов. Каждая поддиректория, в свою очередь содержит различную информацию, описывающую поведение процесса в системе. Например, /proc/<PID>/stat - хранит статус процесса, /proc/<PID>/comm - имя команды, которой процесс был запущен и т.д.

Повлиять на работающий процесс администратор (или ядро) может посредством специальных вызовов, называемых сигналами. Для отправки сигнала используется команда kill.

Список сигналов, которые можно отправить процессу командой kill, можно увидеть, вызвав её с опцией -l.

```
[demo@localhost ~]$ 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
…
```

Как видно из вывода, сигналов довольно много. Не все они предназначены для того, чтобы их использовал системный администратор. Некоторые сигналы использует ядро. Например, сигнал с именем SIGSEGV (Segment Violation) ядро отправляет процессу при попытке обратиться к несуществующей области памяти либо при обращении с нарушением прав доступа (памяти чужого процесса).

Для отправки сигнала команда kill требует следующие значения: что за сигнал мы отправляем (может быть задан номером или именем, причём имя можно использовать без приставки "SIG") и кому (процессу с каким идентификатором). Например: "kill -19 1234"

Узнать идентификатор процесса или процессов можно командами pidof или pgrep (ps+grep):

```
[demo@localhost ~]$ pgrep firefox
4397
[demo@localhost ~]$ pidof firefox
4397
```

Команда pgrep обладает рядом полезных опций, позволяющих отбирать процессы более гранулировано:

-u - отобрать процессы, принадлежащие определённому пользователю (пользователям)

-P - отобрать процессы, являющиеся дочерними для указанного

-t - отобрать процессы, связанные с указанным терминалом (например, pgrep -t tty2).

Для группового завершения процессов используется команда killall. Поумолчанию она также, как и kill отправляет сигнал с номером 15:

```
[demo@localhost ~]$ sudo killall xclock
[1]-  Terminated              xclock -update 1
[2]+  Terminated              xclock -update 1
```

Еще одна команда завершения - pkill. Смысл её параметров такой же, как у команды pgrep. Например:

```
[demo@localhost ~]$ echo $$
3379
[demo@localhost ~]$ sleep 300 &
   [1] 5426
[demo@localhost ~]$ sleep 400 &
   [2] 5433
[demo@localhost ~]$ sleep 500 &
   [3] 5448
[demo@localhost ~]$ pgrep -P 3379
5426
5433
5448
```

Запуск процесса в фоновом режиме

command &

jobs

bg

fg

Некоторые команды, запустившись, выводят какую-то информацию на экран, после чего завершаются, возвращая нам доступ к командной строке (например, pwd, id и т.п.). Однако, если вы запустите задачу, которая не завершается мгновенно, пользоваться командной строкой будет невозможно до тех пор, пока задача не завершится. Проверить это можно с помощью любой программы, обладающей оконным интерфейсом, скажем, gnome-calculator. Изменить ситуацию можно, запустив программу в фоновом режиме. Это делается путём добавления символа «&» к концу строки, например:

```
[demo@localhost ~]$ gnome-calculator &
[1] 5711
```

В этом случае на экране отображается номер задачи (Job ID) для созданного процесса (он уникален в пределах текущей командной оболочки) и PID (он уникален в пределах всей системы). Увидеть список всех задач (jobs), работающих в текущем сеансе командной оболочке можно следующим образом:

```
[demo@localhost ~]$ jobs
[1]    Running                 gnome-calculator &
[1]+  Running                 gnome-calculator &
```

Команда jobs имеет параметр "-l", который позволяет показывать задачи вместе с их PID.

Если процесс был запущен в фоновом режиме, его можно перевести в обычный режим командой fg (foreground). В качестве параметра команда fg принимает Job ID процесса. Также предусмотрена обратная команда – bg (background), предназначенная для перевода процесса из обычного режима в фоновый.

Задание 2

1) Откройте окно терминала и выполните в нем следующие команды:

(while true ; do echo -n "stone " >> /tmp/outfile.txt ; sleep 1 ; done) &

(while true ; do echo -n "scissors " >> /tmp/outfile.txt ; sleep 1 ; done) &

(while true ; do echo -n "paper " >> /tmp/outfile.txt ; sleep 1 ; done) &

2) Таким образом, мы создали три процесса, которые последовательно записывают разные слова в файл /tmp/outfile.txt

3) Откройте еще одно терминала и выполните команду

tail -f /tmp/outfile.txt

Она позволит отследить изменения в файле outfile.txt.

1) В первом окне выполните команду "jobs -l" чтобы увидеть список работающих процессов вместе с их PID и "Job ID".

2) Проверьте, как изменилось состояние процессов, повторно вызвав команду jobs.

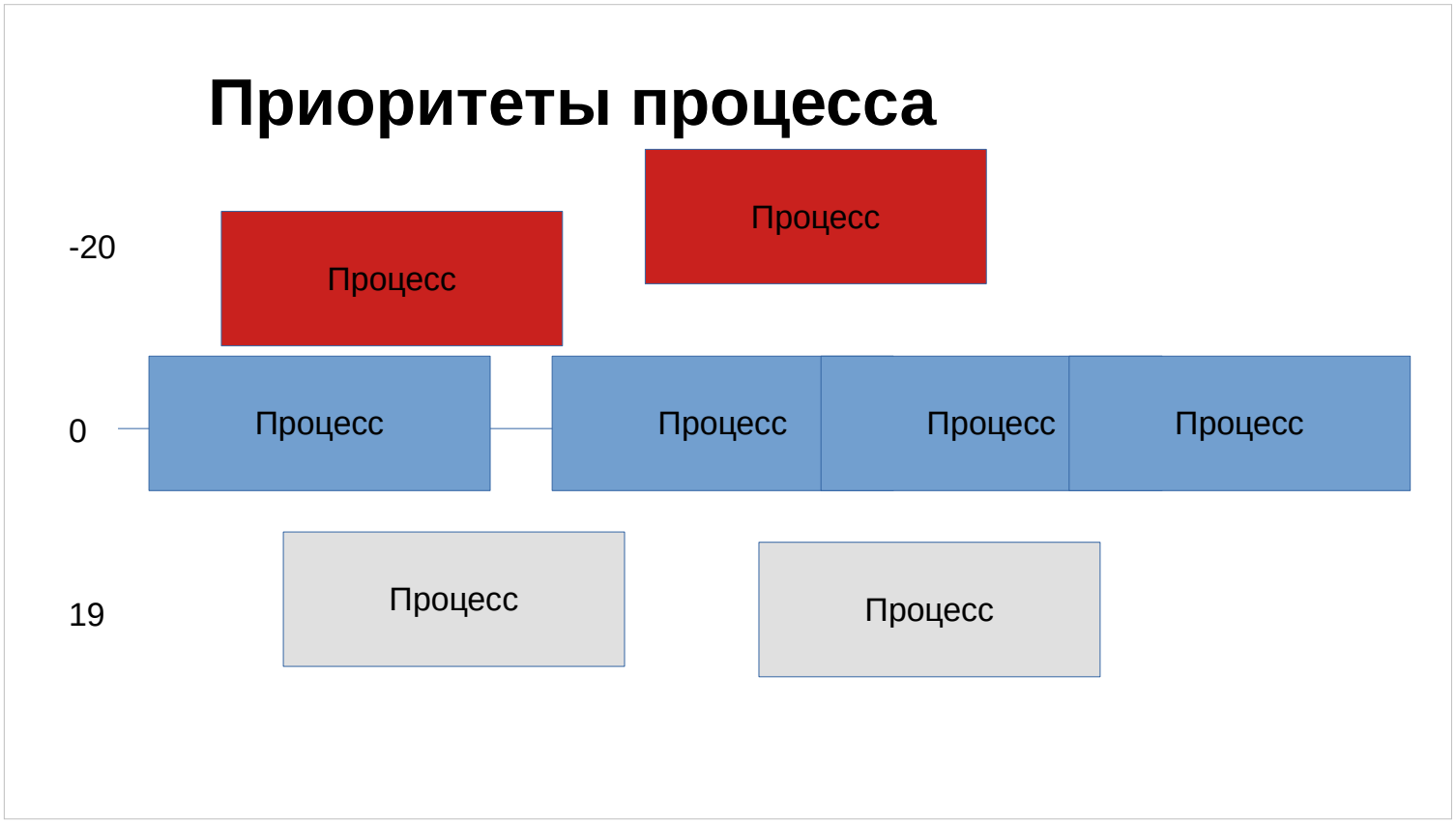
3) Далее, с помощью сигнала 19 (SIGSTOP) остановите первый и третий процессы из списка.

4) Во втором окне убедитесь, что теперь в файле outfile.txt появляется только слово "scissors".

5) Возобновите остановленные процессы с помощью сигнала 18.

6) Проверьте, осуществляют ли они запись в файл?

7) Завершите все три процесса.



Большое количество работающих в системе процессов приводит к тому, что работающие процессы вполне могут начать конкурировать друг с другом за процессорное время, и ядру системы необходим механизм, с помощью которого оно сможет обеспечивать процессам возможность выполняться совместно друг с другом. Этот механизм называется планировщиком или диспетчером процессов. По умолчанию диспетчер старается делать так, чтобы процессы получали эквивалентные кванты процессорного времени, однако в реальных обстоятельствах может возникнуть ситуация, когда администратору потребуется сделать так, чтобы какой-то процесс (или процессы) выполнялись в первую очередь. Или наоборот, указать системе, что тот или иной процесс имеет низкую значимость и может выполняться по «остаточному принципу».

Для этой цели у процесса предусмотрен специальный атрибут, называемый niceness («любезность»). Его значение может лежать в диапазоне от -20 до 19. Чем выше значение этого атрибута, тем процесс «любезнее» (легче отдаёт своё процессорное время другим процессам). По умолчанию процесс запускается со значением niceness, унаследованным от родительского процесса и чаще всего это значение равно нулю (т.е. приоритет нейтральный).

Таким образом, если нам нужно повысить приоритет процесса, ему необходимо выставить отрицательное значение niceness и наоборот. Например:

запустить процесс с обычным приоритетом:

```
[demo@localhost ~]$  /usr/local/bin/executable
```

запустить процесс с повышенным приоритетом:

```
[demo@localhost ~]$  nice  -n  -10  /usr/local/bin/executable
```

запустить процесс с пониженным приоритетом:

```
[demo@localhost ~]$  nice  -n 10  /usr/local/bin/executable
```

Если процесс уже был запущен ранее, изменить его приоритет можно с помощью команды renice. Для этого нам понадобиться узнать PID процесса и указать этот PID команде renice в качестве параметра:

```
[demo@localhost ~]$   renice -n 10 <PID>
```

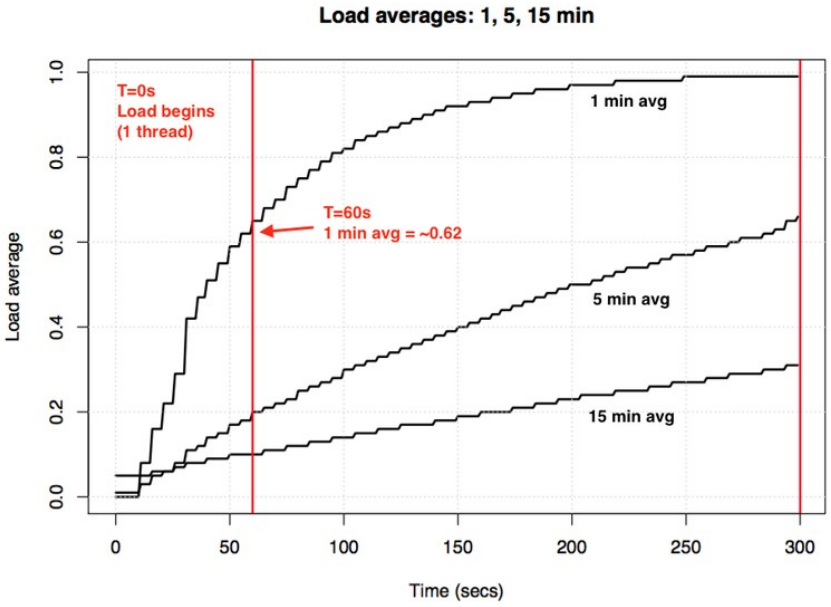
При использовании команд nice и renice действуют несколько правил:

- Отрицательные значения nice может указывать только пользователь root. Непривилегированные пользователи могут задавать только положительные значения.
- Непривилегированные пользователи могут изменять приоритет только тех процессов, владельцами которых они являются. root имеет права изменять приоритет всех процессов
- Непривилегированный пользователь может только понижать приоритет процесса (даже собственного). Увеличивать приоритет может только root

Узнать текущее значение niceness процесса можно с помощью команды top (колонка “NI”), либо с помощью команды ps:

```
[demo@localhost ~]$ ps -e -o pid,comm,nice --sort=nice | less
  PID COMMAND         NI
    5 kworker/0:0H    -20
   13 netns           -20
   15 writeback       -20
   16 kintegrityd     -20
   17 bioset          -20
   18 kblockd         -20
   19 md              -20
   28 crypto          -20
   36 kthrotld        -20
…
```

Информация о нагрузке на систему



Достаточно часто перед администратором встает задача определить, высока ли загрузка работающей системы? Каков запас "прочности" в плане нагрузки, можно ли её повысить или уже достаточно? Для ответа на этот вопрос в качестве отправной точки используют команду uptime:

```
[demo@localhost ~]$ uptime
 07:22:49 up  1:51,  2 users,  load average: 2.50, 4.50, 4.89
```

Команда показывает время, прошедшее с момента загрузки системы, количество залогиненых в данный момент пользователей и среднюю нагрузку на систему за последние минуту, пять и пятнадцать минут. О чем говорят эти цифры? Если загрузка за минуту больше остальных двух цифр, то это означает, что нагрузка на систему растёт и наоборот.

Каждая из цифр - это длина очереди процессов (и потоков), ожидающих возможности выполниться. Т.е. первое, на что стоит посмотреть - это нагрузка, которую работающие процессы создают на процессор(ы):

```
[demo@localhost ~]$ mpstat
Linux 3.10.0-693.21.1.el7.x86_64 (localhost.localdomain)  05/28/2018  _x86_64_   (1
    CPU)

07:39:48 AM  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest
    %gnice   %idle
07:39:48 AM  all      14.33    0.10    1.97    0.11       0.00    0.16    0.00       0.00       0.00
      83.32
[demo@localhost ~]$
```

Здесь стоит обратить внимание на показатель "%idle", который позволяет оценить простой процессоров.

На показатель Load Average также влияют процессы, находящиеся в состоянии "D". Если необходимо оценить, насколько интенсивно используется подсистема ввода-вывода, можно использовать команды iostat или, что ещё лучше - iotop. По аналогии с командой top, которая выводит данные о работающих процессах, команда iotop выводит информацию об использовании устройств ввода-вывода. Полезным параметром команды является опция "-o", позволяющая выводить информацию не обо всех процессах, а только о тех, кто действительно использует ввод-вывод.

Задание 3

Давайте добавим в систему некий "тяжеловесный" процесс, создающий чрезмерную нагрузку на процессор. Имя процесса будет выбрано случайным образом.

1) RNDNAME=$(mktemp --dry-run) cp /bin/md5sum $RNDNAME $RNDNAME /dev/urandom & clear
2) С помощью команды top определите имя этого процесса и завершите его
3) Проверьте, завершен ли процесс с помощью команд ps и grep ps -e | grep $RNDNAME