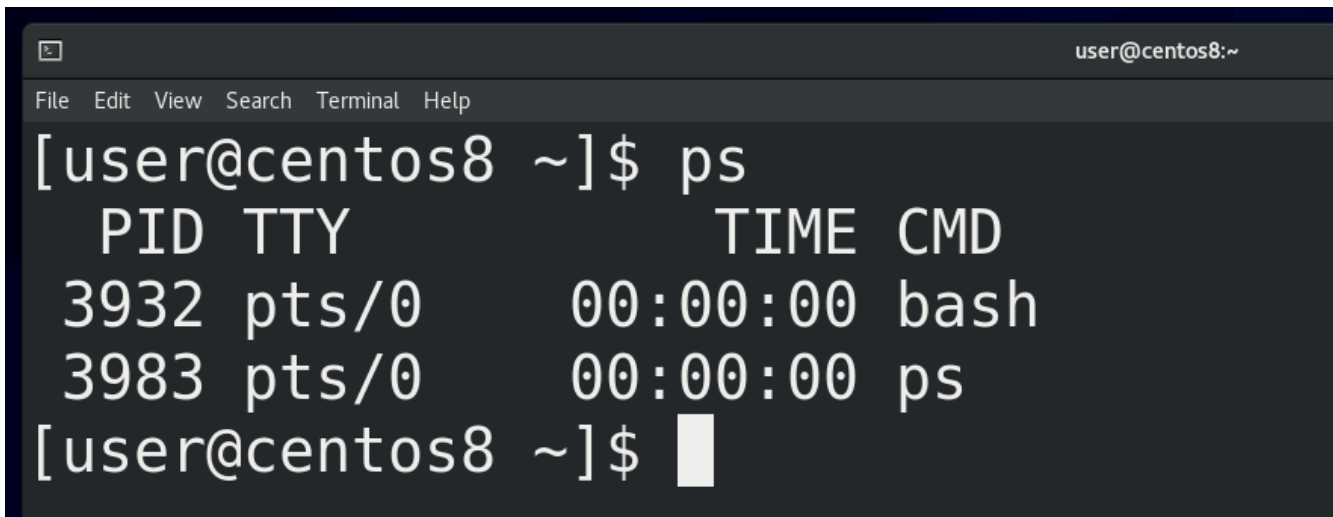


Как мы уже выяснили, программы хранятся в файловой системе на накопителе – т.е. жёстком диске или ssd. Когда мы запускаем программу, она загружается в оперативную память, так как скорость чтения с жёсткого диска или даже ssd относительно низкая, а процессор работает на больших скоростях. Как правило, большие программы загружаются в оперативную память не полностью, а по мере необходимости. При этом, для каждой программы создаётся иллюзия, что она – единственная в оперативной памяти, то есть для неё создаётся так называемая “виртуальная память”. Также программы при запуске загружают какие-то файлы, будь то файлы настроек или пользовательские файлы – как например, если мы запускаем nano file, то в память загружаются как сам /usr/bin/nano, его настройки - /etc/nanorc и ~/.nanorc, всякие библиотеки, необходимые для работы nano и сам файл, который мы открываем. Кроме этого также запускаемой программе передаются переменные окружения и ещё много всего. Ну и находясь в оперативке, эта программа делает какие-то вычисления с помощью центрального процессора, обрабатывает данные и сохраняет на диске. И совокупность всего этого называется процессом.

Иногда одной программе нужно бывает выполнить несколько операций параллельно. Представьте себе сложное математическое уравнение – есть всякие скобки, умножения и прочее. Такое уравнение можно разделить на составляющие и компьютер может разом выполнить все составляющие, а потом, используя результаты, получить простое уравнение и выполнить его. Или, допустим, веб сервер – к нему обращаются много клиентов, и каждого из них он должен обслужить и желательно параллельно. Для этого один процесс может разделяться на так называемые потоки – все они используют общую виртуальную память. У каждого процесса есть как минимум один поток.

A screenshot of a terminal window with a dark background and light-colored text. The window title bar shows 'user@centos8:~'. The terminal has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The prompt is '[user@centos8 ~]\$'. The command 'ps' has been entered, and the output is displayed in a table format. The table has four columns: 'PID', 'TTY', 'TIME', and 'CMD'. There are two rows of data: the first row shows PID 3932, TTY pts/0, TIME 00:00:00, and CMD bash; the second row shows PID 3983, TTY pts/0, TIME 00:00:00, and CMD ps. The prompt '[user@centos8 ~]\$' is followed by a cursor.

```
[user@centos8 ~]$ ps
  PID TTY          TIME CMD
 3932 pts/0        00:00:00 bash
 3983 pts/0        00:00:00 ps
[user@centos8 ~]$
```

И так, выполняемая программа – это процесс. Начнём с того, что администратору важно видеть список процессов. Для этого есть несколько способов, начнём с утилиты ps. Если просто запустить ps, мы увидим список процессов, запущенных в этом терминале. Как вы заметили, вывелось 2 строчки – bash и ps. При том, что ps у нас выполнялся за какие-то доли секунды, он у нас всё равно виден в выводе – потому что он делает эдакий скриншот процессов именно в момент выполнения, поэтому и видит сам себя.

```
user@centos8:~  
File Edit View Search Terminal Help  
DESCRIPTION  
ps displays information about a selection of the  
active processes. If you want a repetitive update of  
the selection and the displayed information, use  
top(1) instead.  
  
This version of ps accepts several kinds of options:  
  
1    UNIX options, which may be grouped and must be  
    preceded by a dash.  
2    BSD options, which may be grouped and must not be  
    used with a dash.  
3    GNU long options, which are preceded by two  
Manual page ps(1) line 9 (press h for help or q to quit)
```

Вообще, ps работает с 3 видами ключей: юниксовыми – они обычно начинаются на один дефис (-), BSD-шные – вовсе без дефиса и GNU-шные – как правило это слова, поэтому, чтобы не счесть их за комбинацию букв, используется два дефиса. Если посмотреть документацию - man ps - можно заметить очень много дублирующихся ключей. Но документация по ps огромная, да и все ключи знать не нужно. Достаточно выучить какую-то одну комбинацию, которая подойдёт в большинстве случаев - ps -ef, а если вам понадобится что-то конкретное, то всегда можно погуглить или найти в мане.

```
user@centos8:~  
File Edit View Search Terminal Help  
UID      PID    PPID    C  STIME TTY          TIME CMD  
root      1        0    0  18:31 ?          00:00:01 /usr/lib/systemd/systemd  
root      2        0    0  18:31 ?          00:00:00 [kthreadd]  
root      3        2    0  18:31 ?          00:00:00 [rcu_gp]  
root      4        2    0  18:31 ?          00:00:00 [rcu_par_gp]  
root      5        2    0  18:31 ?          00:00:00 [kworker/0:0-events]  
root      6        2    0  18:31 ?          00:00:00 [kworker/0:0H-kblockd]  
root      8        2    0  18:31 ?          00:00:00 [mm_percpu_wq]  
root      9        2    0  18:31 ?          00:00:00 [ksoftirqd/0]
```

Как видите, вывод у ps довольно большой и не помещается на экране, поэтому урезается сбоку. Чтобы мы могли нормально прочесть, мы можем передать вывод ps команде less. Правда по умолчанию less переносит текст на новые строки, из-за чего сбиваются столбцы, поэтому less лучше использовать с ключом -S, который не переносит строки. В итоге – ps -ef | less -S.

Давайте разберём, что означают ключи и как читать вывод. Ключ -e выводит все процессы всех пользователей - ps -e. Да, процессы запускаются от имени пользователей. От этого зависит какие права будут у процесса. Допустим, если я запускаю программу папо от пользователя user, то программа сможет работать с моими файлами. А ключ f - ps -f - показывает чуть больше информации о процессе. Давайте пройдемся по столбикам - ps -ef | less -S.

```
user@centos8:~  
File Edit View Search Terminal Help  
gdm      2393  2294  0 18:32 tty1      00:00:00 /usr/libexec/ibus-engin  
gdm      2399  2239  0 18:32 tty1      00:00:00 /usr/libexec/gsd-sharin  
gdm      2406  2239  0 18:32 tty1      00:00:00 /usr/libexec/gsd-smarto  
gdm      2410  2239  0 18:32 tty1      00:00:00 /usr/libexec/gsd-sound  
gdm      2414  2239  0 18:32 tty1      00:00:00 /usr/libexec/gsd-wacom  
colord   2437    1    0 18:32 ?          00:00:00 /usr/libexec/colord  
root     2531  2171  0 18:35 ?          00:00:00 gdm-session-worker [par  
user     2541    1    0 18:35 ?          00:00:00 /usr/lib/systemd/system  
user     2547  2541  0 18:35 ?          00:00:00 (sd-pam)  
user     2557  2541  0 18:35 ?          00:00:00 /usr/bin/pulseaudio --d  
user     2562    1    0 18:35 ?          00:00:00 /usr/bin/gnome-keyring-  
user     2570  2541  0 18:35 ?          00:00:00 /usr/bin/dbus-daemon --  
user     2578  2531  0 18:35 tty2      00:00:00 /usr/libexec/gdm-waylan  
:  
:
```

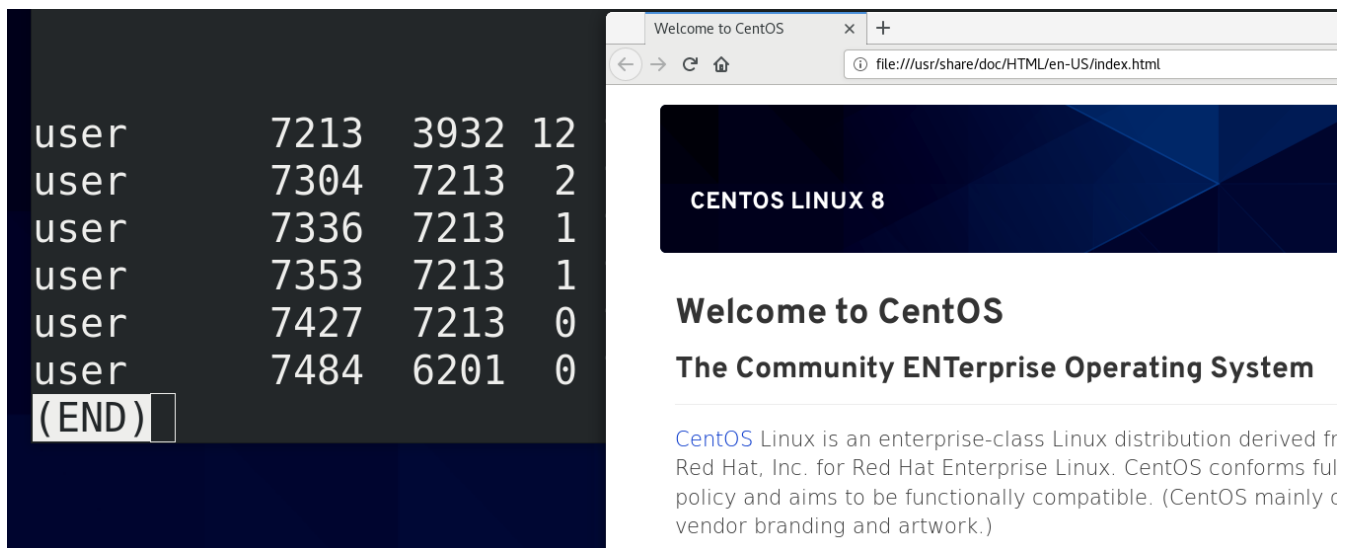
Первое – UID – user id - пользователь, который запустил процесс. Большинство процессов в системе запущены от пользователя root – его также называют суперпользователем – это юзер, у которого есть все права на систему. По возможности, люди стараются не использовать рута везде. Если у программы будет какой-то баг или уязвимость и если она запущена от рута, то программа может сильно навредить системе. Поэтому для программ, которые не требуют особых прав, обычно создают сервисных пользователей. Как правило при установке программы она сама всё это настраивает. Ну и наконец у нас тут есть программы, запущенные от нашего пользователя. Как видите, я вроде ничего кроме эмулятора терминала не запускал, а в системе уже пару сотен процессов.

Второй столбик – PID – process id – идентификатор процесса. Он уникальный для каждого процесса, но совпадает для потоков одного процесса. Когда программа завершается, она освобождает номер и через какое-то время другая программа может использовать этот номер. С помощью этих номеров мы можем управлять процессами.

```
user@centos8 ~]$ ps -ef | grep nano
user      6151   3932    0 18:58 pts/0    00:00:00 nano
user      6272   6201    0 18:59 pts/1    00:00:00 grep --color=auto nano
[user@centos8 ~]$ ps -f 3932
UID        PID  PPID  C  STIME TTY          STAT      TIME CMD
user        3932   3884    0 18:40 pts/0    Ss         0:00  bash
[user@centos8 ~]$ ps -f 3884
UID        PID  PPID  C  STIME TTY          STAT      TIME CMD
user        3884   2541    0 18:40 ?        Ssl        0:04  /usr/libexec/gnome
[user@centos8 ~]$ ps -f 2541
UID        PID  PPID  C  STIME TTY          STAT      TIME CMD
user        2541    1     0 18:35 ?        Ss         0:00  /usr/lib/systemd/sy
[user@centos8 ~]$
```

```
GNU nano 2.9.8                               New Buffer
```

Третий столбик – PPID – parent process id – идентификатор родительского процесса. Почти все процессы в системе были запущены каким-то другим процессом. Допустим, когда мы запускаем эмулятор терминала, а в нём nano – то родительским процессом для nano является bash, который запущен в этом эмуляторе терминала - `ps -ef | grep nano`. Родительским процессом для этого bash - `ps -f ppid` - является gnome - процесс рабочего окружения. Родительским процессом для него является systemd - первый процесс. О systemd мы ещё поговорим.



Четвёртый столбик – C – использование процессора данным процессом. Много где у нас нули, но давайте запустим какое-нибудь тяжёлое приложение, допустим, firefox, найдём этот процесс - `ps -ef | grep firefox | less -S` - и увидим, что для него это значение отличается от нуля.

Пятый столбик - STIME – время, когда процесс был запущен.

```
user@centos8:~  
File Edit View Search Terminal Help  
gdm      2399   2239   0 18:32 tty1      00:00:00 /usr/libexec/gsd-sharing  
gdm      2406   2239   0 18:32 tty1      00:00:00 /usr/libexec/gsd-smartcard  
gdm      2410   2239   0 18:32 tty1      00:00:00 /usr/libexec/gsd-sound  
gdm      2414   2239   0 18:32 tty1      00:00:00 /usr/libexec/gsd-wacom  
colord    2437     1    0 18:32 ?          00:00:00 /usr/libexec/colord  
root     2531   2171   0 18:35 ?          00:00:00 gdm-session-worker [pam/g>  
user     2541     1    0 18:35 ?          00:00:00 /usr/lib/systemd/systemd >  
user     2547   2541   0 18:35 ?          00:00:00 (sd-pam)  
user     2557   2541   0 18:35 ?          00:00:00 /usr/bin/pulseaudio --dae>  
user     2562     1    0 18:35 ?          00:00:00 /usr/bin/gnome-keyring-da>  
user     2570   2541   0 18:35 ?          00:00:00 /usr/bin/dbus-daemon --se>  
user     2578   2531   0 18:35 tty2      00:00:00 /usr/libexec/gdm-wayland->  
user     2581   2578   0 18:35 tty2      00:00:00 /usr/libexec/gnome-sessio>  
:  
:
```

Дальше - TTY – от слова телетайп. На хабре есть неплохая [статья](#), объясняющая разницу между телетайпом, консолью, терминалом и т.п. А ps в этом столбике говорит, с каким терминалом ассоциируется данный процесс. Обычно, процесс, запущенный системой и не требующий графики, вывода информации на экран, не связан ни с каким терминалом. Процессы, требующие графики, завязаны на каком-нибудь виртуальном терминале – о них мы говорили ранее. Можно заметить, что тут указано tty1 и tty2 – если нажать правый ctrl + f1 или ctrl+f2, можно увидеть, что именно здесь у нас запущен графический интерфейс. При переходе на ctrl+f3 и далее открывается виртуальный терминал. А для эмуляторов терминала здесь могут быть значения pts/0, pts/1, pts/2 и т.п.

```
user      8443   3932   8 19:14 pts/0      00:00:06 /usr/lib64/firefox/firefox  
user      8522   8443   0 19:14 pts/0      00:00:00 /usr/lib64/firefox/firefox  
user      8556   8443   0 19:14 pts/0      00:00:00 /usr/lib64/firefox/firefox  
user      8642   8443  11 19:14 pts/0      00:00:08 /usr/lib64/firefox/firefox  
user      8690   8443   0 19:15 pts/0      00:00:00 /usr/lib64/firefox/firefox  
user      8928   8779   0 19:16 pts/1      00:00:00 grep --color=auto firefox
```

Ещё одно поле – TIME – это сколько времени процессор потратил на работу с данным процессом. Вы можете заметить, что здесь сплошные нули – потому что большинство этих процессов не требуют и секунды процессорного времени. Но если немного поработать с тем же браузером, то это время будет расти - ps -ef | grep firefox. Кстати, чтобы мне не приходилось постоянно запускать эту команду, я могу использовать команду watch – watch “ps -ef | grep firefox”. Эта команда будет каждые 2 секунды запускать указанную команду. И так мы видим, что параметр TIME для нашего браузера постоянно увеличивается.

```

rpc      972      1 0 18:32 ?      00:00:00 /usr/bin/rpcbind -w -f
root     976      2 0 18:32 ?      00:00:00 [rpciod]
root     977      2 0 18:32 ?      00:00:00 [kworker/u3:0]
root     978      2 0 18:32 ?      00:00:00 [xprtiod]
root     981      1 0 18:32 ?      00:00:00 /sbin/auditd
root     983     981 0 18:32 ?      00:00:00 /usr/sbin/sedispatch
rtkit    1005      1 0 18:32 ?      00:00:00 /usr/libexec/rtkit-daemon
dbus     1007      1 0 18:32 ?      00:00:01 /usr/bin/dbus-daemon --system ->
root     1008      1 0 18:32 ?      00:00:00 /usr/sbin/alsactl -s -n 19 -c ->
libstor+ 1009      1 0 18:32 ?      00:00:00 /usr/bin/lsmc -d
root     1010      1 0 18:32 ?      00:00:00 /usr/lib/systemd/systemd-machin>
root     1011      1 0 18:32 ?      00:00:00 /sbin/rngd -f

```

Ну и последнее – CMD – это команда, которая запустила процесс. Некоторые значения в квадратных скобках – для таких процессов ps не смог найти аргументов – обычно это процессы самого ядра.

Ладно, с выводом ps разобрались. Теперь мы знаем, где найти информацию о процессах. Но, помните, я говорил, что в Unix подобных системах придерживаются идеи “Всё есть файл”? И даже процессы у нас представлены в виде файлов. Но хранить информацию о процессах на жёстком диске нецелесообразно – какие-то процессы существуют доли секунд, какие-то появляются и удаляются сотнями – жёсткий диск не подходит для такого. А вот в оперативной памяти информацию о процессах можно спокойно хранить и представлять в виде файлов. Но раз уж речь идёт о файлах, то нам нужна файловая система. И вот ядро действительно создаёт так называемую виртуальную файловую систему, которая существует только в оперативной памяти.

```

user@centos8 /proc
[user@centos8 ~]$ cd /proc/
[user@centos8 proc]$ ls
1      1171  2211  2393  2724  2878  4      638      buddyinfo  meminfo
10     1175  2212  2399  2726  2880  4080   639      bus         misc
1005   12    2220  24    2733  29    437    640      cgroups    modules
1007   13    2239  2406  2738  2922  442    641      cmdline    mounts
1008   140   2247  2410  2752  2934  450    642      consoles   mtrr
1009   141   2276  2414  2759  2935  457    6489     cpuinfo    net
1010   142   2280  2437  2763  2951  459    742      crypto     pagetypeinfo
1011   143   2285  25    2767  2961  466    7584     devices    partitions

```

Вообще, этих виртуальных файловых систем несколько, они используются для разных задач, мы о них поговорим в другой раз. Сейчас нас интересует файловая система procfs. Она примонтирована в директорию /proc - cd /proc. Если посмотреть содержимое этой директории -ls - мы увидим кучу директорий и файлов. Директории вам ничего не напоминают? Именно, это номера процессов, т.е. pid-ы. Ядро операционной системы генерирует эту информацию налету, стоит нам посмотреть – мы увидим актуальную информацию.


```
user@centos8:/proc
[user@centos8 proc]$ cat version
Linux version 4.18.0-147.8.1.el8_1.x86_64 (mockbuild@kbuilder.bsys.centos.org) (
gcc version 8.3.1 20190507 (Red Hat 8.3.1-4) (GCC)) #1 SMP Thu Apr 9 13:49:54 UT
C 2020
[user@centos8 proc]$ cat uptime
3195.22 2893.17
[user@centos8 proc]$ uptime
 19:25:41 up 53 min,  1 user,  load average: 0.13, 0.17, 0.19
```

В этой директории кроме директорий процессов есть много других файлов – допустим, `version` - `cat version` – показывает нам информацию о версии ядра или `uptime` - `cat uptime` – информацию о том, сколько секунд включена система. Ну в секундах непонятно, поэтому легче использовать утилиту `uptime`. Кстати, постарайтесь самостоятельно найти, что означает второе значение в файле `/proc/uptime` и напишите в комментариях так, чтобы было понятно всем.

```
user@centos8:/proc
[user@centos8 proc]$ ps -ef | grep firefox
user      10402    2647 17 19:27 tty2      00:00:02 /usr/lib64/firefox/firefox

user@centos8:/proc/10402
[user@centos8 proc]$ cd 10402
[user@centos8 10402]$ ls
attr          exe           mounts        projid_map    status
autogroup     fd            mountstats    root          syscall
auxv          fdinfo        net           sched         task
cgroup        gid_map      ns            schedstat     timers
clear_refs    io            numa_maps     sessionid     timerslack_ns
cmdline       limits       oom_adj       setgroups     uid_map
comm          loginuid     oom_score     smaps         wchan
coredump_filter map_files     oom_score_adj smaps_rollup
cpuset        maps          pagemap       stack
cwd           mem           patch_state   stat
environ       mountinfo    personality    statm
```

Ну и давайте посмотрим, что же такого в директориях процессов. Найдём `pid` процесса, допустим того-же `firefox` - `ps -ef | grep firefox` - и зайдём в эту директорию - `cd pid; ls`. Тут у нас тоже куча файлов, которые относятся к нашему процессу. Эти файлы нужны не столько для людей, сколько для программ.

```
user@centos8/proc/10402
File Edit View Search Terminal Help
[user@centos8 10402]$ cat cmdline
/usr/lib64/firefox/firefox[user@centos8 10402]$
[user@centos8 10402]$ cat environ
LD_LIBRARY_PATH=/usr/lib64/firefox:/usr/lib64/firefox/plugins:/usr/lib64/firefox
FONTCONFIG_PATH=/etc/fonts:/usr/lib64/firefox/res/XftXDG_MENU_PREFIX=gnome-LANG=

user@centos8/proc/10402
File Edit View Search Terminal Help
[user@centos8 10402]$ cat status
Name:   firefox
Umask:  0002
State:  S (sleeping)
Tgid:   10402
Ngid:   0
Pid:    10402
PPid:   2647
```

Какие-то из этих файлов и мы можем прочесть. Например, cmdline - cat cmdline. Тут отображена команда, которая запустила процесс. Или environ - cat environ - те переменные, которые передались процессу при запуске. Или status - cat status . Какие-то из этих строчек понятны, а какие-то без гугла не разберёшь. Знать всё это не нужно, но, со временем, углубляясь в теорию или сталкиваясь с какими-то проблемами, вы начнёте разбираться в этих файлах.

