Managing running processes

By now we learned that Linux is a multi-user and multi-tasking Operating System.
In this chapter we will take a look at running programs, better called processes. Linux has the tools to show a list of the running processes, to monitor the system usage and to stop or kill processes. These commands that show information about processes, extract their info from the /proc folder. Each process will save their info in a subdirectory of /proc and will be named a number, more specific the process ID (PID). For example /proc/1/status, will show the status of the process with process ID 1. An example of a process is to start the nano-editor by using the nano-command that is saved on your Linux. If 15 users would do this at the same time, not one but 15 processes would start. Every process could be identified by its process ID. This ID is a unique number for each started nano instance on the current system. No process can have the same PID as another process, as long as this process is running. If a process ends, another process can reuse it’s PID. Another attribute of a process is that every process is associated with a user account and group account. This association will determine what system resources the process is able to use. For example: a process that runs as root will have more access to the filesystem than a process a normal user would have started. A system manager is able to manage the processes. Some processes might influence the performance of the system. Searching for these processes, based on memory- and CPU-usage, will be looked at in the chapter.

Listing processes with ps

ps is the oldest and most used command to list the running processes.

```bash
student@linux-ess:~$ ps
    PID TTY          TIME CMD
   2556 pts/0    00:00:00 bash
```

```
  11680 pts/0    00:00:00 ps
```

By adding the option u to the ps command some additional information is shown. These being: The usernames, start time of the process, CPU- and memory usage, from where the process is started for example tty1 or pts/0. The concept of these terminals descends from the time people exclusively worked from terminals. There was one person on one terminal. Nowadays multiple terminals can be opened on the same screen by opening multiple virtual terminals.

```bash
student@linux-ess:~$ ps u
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAN
student     1389  0.0  0.1   9120  5748 pts/0    Ss   06:37   0:00 -bash
student     7593  0.0  0.0  10068  1564 pts/0    R+   07:20   0:00 ps u
```

In this example we can see that:

- student started process 7593, which is command ps u
- pts/0 is used
- STAT shows the status of the process, 'r' for running or 's' for sleeping
- USER is the name of the user as which the process runs
- PID is the unique number of the process. This number will later be used to kill or send signals to the process
- %CPU and %MEM are the CPU and memory time the process is using
- VSZ, the virtual set size, shows the image size of the process in kilobytes (size of memory given to the process)
- RSS, the resident set size, shows the size of the process in memory (actual size in use by the process)
- START is the time the process started
- TIME is the cumulative system time that has been used

A lot of processes running on your system are not associated with a terminal, these are mostly processes that run in the background. For example: logging of system activities, listening to incoming data from the network. These processes often start when Linux starts and stop when you shut down. When starting a graphical environment (like Ubuntu Desktop) a lot of background process start as well, look at audio, authentication, …)

To show all running processes for your current user use:

bash

```
student@linux-ess:~$ ps ux | less
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAN
student     1807  0.0  0.3  20956 13480 ?        Ss   13:41   0:00 /lib/s
student     1808  0.0  0.1 105604  5328 ?        S    13:41   0:00 (sd-pa
student     1814  0.0  0.1  48228  6440 ?        S<sl 13:41   0:00 /usr/b
student     1815  0.0  0.1  32116  6468 ?        Ssl  13:41   0:00 /usr/b
student     1817  0.1  0.7 1508148 30068 ?       S<sl 13:41   0:05 /usr/b
student     1825  0.0  0.1 249548  7592 ?        Sl   13:41   0:00 /usr/b
student     1834  0.0  0.3  15716 12076 ?        Ss   13:41   0:00 /usr/b
student     1842  0.0  0.2 249300  8384 ?        Ssl  13:41   0:00 /usr/l
student     1846  0.0  0.1 619288  7516 ?        Ssl  13:41   0:00 /usr/l
student     1848  0.0  0.1 380884  6332 ?        Sl   13:41   0:00 /usr/l
student     1852  0.0  0.1 244796  5416 ?        Ssl  13:41   0:00 /usr/l
student     1895  0.0  0.6 715924 27288 ?        SNsl 13:41   0:00 /usr/l
student     1906  0.0  0.1 171040  6104 tty2     Ssl+ 13:41   0:00 /usr/l
student     1907  0.0  0.2 398428 10412 ?        Ssl  13:41   0:00 /usr/l
student     1915  0.0  0.3 231688 15420 tty2     Sl+  13:41   0:00 /usr/l
student     1943  0.0  0.1 245296  6640 ?        Ssl  13:41   0:00 /usr/l
student     1948  0.0  0.9 643640 39360 ?        Sl   13:41   0:00 /usr/l
:
```

To show all running processes of all users use:

bash

```
student@linux-ess:~$ ps aux | less
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAN
root           1  0.0  0.3 102468 13252 ?        Ss   13:41   0:03 /sbin/
root           2  0.0  0.0      0     0 ?        S    13:41   0:00 [kthre
root           3  0.0  0.0      0     0 ?        I<   13:41   0:00 [rcu_g
root           4  0.0  0.0      0     0 ?        I<   13:41   0:00 [rcu_p
root           5  0.0  0.0      0     0 ?        I<   13:41   0:00 [netns
root           7  0.0  0.0      0     0 ?        I<   13:41   0:00 [kwork
root        10  0.0  0.0      0     0 ?        I<   13:41   0:00 [mm_pe
root        11  0.0  0.0      0     0 ?        S    13:41   0:00 [rcu_t
root        12  0.0  0.0      0     0 ?        S    13:41   0:00 [rcu_t
root        13  0.0  0.0      0     0 ?        S    13:41   0:00 [ksoft
root        14  0.0  0.0      0     0 ?        I    13:41   0:01 [rcu_s
root        15  0.0  0.0      0     0 ?        S    13:41   0:00 [migra
root        16  0.0  0.0      0     0 ?        S    13:41   0:00 [idle_
root        17  0.0  0.0      0     0 ?        S    13:41   0:00 [cpuhp
:
```

In the following example the option -e is used to show all running processes, the option -o is given when specific data is wanted. We chose for the username, process ID, memory usage, virtual set size, resident set size, tty , status, start time and the command.

bash

```
student@linux-ess:~$ ps -eo user,pid,%mem,vsz,rss,tty,stat,start,comm | l
USER         PID %MEM    VSZ   RSS TT       STAT  STARTED COMMAND
root           1  0.2 166152 11288 ?        Ss   12:47:06 systemd
root           2  0.0      0     0 ?        S    12:47:06 kthreadd
root           3  0.0      0     0 ?        I<   12:47:06 rcu_gp
root           4  0.0      0     0 ?        I<   12:47:06 rcu_par_gp
root           5  0.0      0     0 ?        I<   12:47:06 netns
root           7  0.0      0     0 ?        I<   12:47:06 kworker/0:0H-ev
root           9  0.0      0     0 ?        I<   12:47:06 kworker/0:1H-ev
root          10  0.0      0     0 ?        I<   12:47:06 mm_percpu_wq
root          11  0.0      0     0 ?        S    12:47:06 rcu_tasks_rude_
root          12  0.0      0     0 ?        S    12:47:06 rcu_tasks_trace
root          13  0.0      0     0 ?        S    12:47:06 ksoftirqd/0
root          14  0.0      0     0 ?        I    12:47:06 rcu_sched
root          15  0.0      0     0 ?        S    12:47:06 migration/0
root          16  0.0      0     0 ?        S    12:47:06 idle_inject/0
root          18  0.0      0     0 ?        S    12:47:06 cpuhp/0
:
```

The previous command could also be done with the BSD style via the command: ps axo user,pid,%mem,vsz,rss,tty,stat,start,comm

We can also add the option –sort= to the command and choose a parameter to sort our list. We chose the rss values from large to small by placing another – sign right before it.

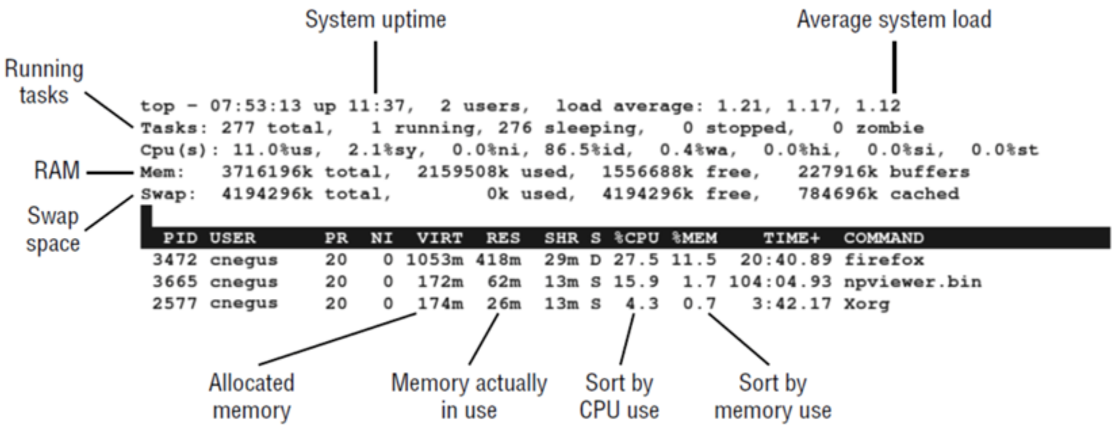
bash

```
student@linux-ess:~$ ps -eo user,pid,%mem,vsz,rss,tty,stat,start,comm --s
USER         PID %MEM    VSZ   RSS TT       STAT  STARTED COMMAND
root        1303  1.0 874820 42356 ?        Ssl  06:37:14 snapd
root         498  0.9  72440 37876 ?        S<s  06:31:35 systemd-journal
root         542  0.6 354880 27228 ?        SLsl 06:31:35 multipathd
root         905  0.5 109744 21560 ?        Ssl  06:31:44 unattended-upgr
root         845  0.4  32648 19100 ?        Ss   06:31:43 networkd-dispat
root         854  0.3 392568 12944 ?        Ssl  06:31:43 udisksd
root           1  0.3 102068 12804 ?        Ss   06:31:16 systemd
systemd+     824  0.3  25392 12604 ?        Ss   06:31:41 systemd-resolve
root         735  0.2  51124 11892 ?        Ss   06:31:39 VGAuthService
root         895  0.2 243276 11724 ?        Ssl  06:31:44 ModemManager
root        8289  0.2  17164 10932 ?        Ss   09:50:51 sshd
student     1277  0.2  17052  9840 ?        Ss   06:37:13 systemd
root         901  0.2  15420  9500 ?        Ss   06:31:44 sshd
student     8367  0.2  18272  9136 ?        S    09:50:54 sshd
root         736  0.2 314864  8944 ?        Ssl  06:31:39 vmtoolsd
:
```

The previous command could also be done with the BSD style via the command: ps axo user,pid,%mem,vsz,rss,tty,stat,start,comm k -rss

Listing processes with top

Another command that can be used is top, this command is more screen oriented than ps and provides the possibility to change the state of processes by using the kill command to end a process or renice command to change the priority. If you want to edit all processes you'll need to start top as root.



```
top - 07:53:13 up 11:37,  2 users,  load average: 1.21, 1.17, 1.12
Tasks: 277 total,   1 running, 276 sleeping,   0 stopped,   0 zombie
Cpu(s): 11.0%us,  2.1%sy,  0.0%ni, 86.5%id,  0.4%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   3716196k total,  2159508k used,  1556688k free,   227916k buffers
Swap:  4194296k total,        0k used,  4194296k free,   784696k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 3472 cnegus    20   0 1053m 418m  29m D 27.5 11.5  20:40.89 firefox
 3665 cnegus    20   0  172m  62m  13m S 15.9  1.7 104:04.93 npviewer.bin
 2577 cnegus    20   0  174m  26m  13m S  4.3  0.7   3:42.17 Xorg
```

In the top window there are a few possible command to use:

- h for help
- q to go back
- M to sort based on memory usage
- P to sort based on CPU usage
- 1 to switch between CPU's, if multiple CPU's are available
- R to inverse sort the output shown
- u, then insert a username to show all processes of this user
- r to renice, after r, you will need the PID of the process and the value between -20 and 19
- k to kill, followed by the PID of the process, thereafter 15 to kill clean or 9 to kill abruptly

Managing front- and background processes

Our bash shell, unlike a Graphical User Interface, does not have the possibility to run and show different programs simultaneously in the shell, but we are able to run programs in the front- or background. Knowing that it is possible to run multiple programs at the same time in our shell there must be a way to choose what is running at the fore- and background at what time. For starters there are different methods of running programs in the background. You can add a & after the command to start it in the background

```bash
student@ubuntu-server:~$ sleep 300 &
[1] 1462
```

```
student@ubuntu-server:~$ sleep 250 &
[2] 1463
student@ubuntu-server:~$ sleep 200 &
[3] 1464
student@ubuntu-server:~$ sleep 150 &
[4] 1465
student@ubuntu-server:~$ find /usr > /tmp/alluserfiles &
[5] 1466
student@ubuntu-server:~$ jobs
[1]   Running                 sleep 300 &
[2]   Running                 sleep 250 &
[3]   Running                 sleep 200 &
[4]-  Running                 sleep 150 &
[5]+  Done                    find /usr > /tmp/alluserfiles
```

The + shows the last (=most recent) process added to the background
The – shows the second to last process added to the background
To pause a process and put it in the background use ctrl + Z

bash

```
student@linux-ess:~$ sleep 50
^Z
[1]   Done                    sleep 300
[2]   Done                    sleep 250
[3]   Done                    sleep 200
[4]   Done                    sleep 150
[5]+  Stopped                 sleep 50
```

To bring the command back to the front, use the fg command.

bash

```
student@linux-ess:~$ fg
sleep 50
```

The fg command can be used in different ways, following are some options:

- fg %<jobnumber>
- fg %string : command needs to start with string
- fg %?string : job has string in the commandline
- fg %+ or fg : last job send to background
- fg %- : second to last program send to background

With the bg command you can resume a paused process that is located in the background

bash

```
student@linux-ess:~$ sleep 50
^Z
[1]+  Stopped                 sleep 50
student@linux-ess:~$ jobs
[1]+  Stopped                 sleep 50
student@linux-ess:~$ bg %1
[1]+ sleep 50 &
student@linux-ess:~$ jobs
[1]+  Running                 sleep 50 &
```

A process running in the background can still show its output, even when another process is running. For example, when working with nano, an output can come up in your screen. Press ctrl + L to renew the window.

ℹ Use 2> /dev/null to send all errors to the void so they won’t show up

Killing or renicing (changing the priority) is also possible with these processes

Sending signals to a process with kill:

bash

```
student@linux-ess:~$ sleep 500 &
[1] 11977
student@linux-ess:~$ kill 11977
[1]+  Terminated              sleep 500
student@linux-ess:~$ sleep 500 &
[1] 11981
```

```
student@linux-ess:~$ kill -9 11981
student@linux-ess:~$ sleep 500 &
[1] 11982
student@linux-ess:~$ kill -15 11982
student@linux-ess:~$ sleep 500 &
[1] 11983
student@linux-ess:~$ kill -SIGKILL 11983
```

With the kill or killall command, there are more possibilities than stopping a process. You can reload configuration files, pause, continue, ... To do this, signals, numbers and/or names are used. A few examples, use the kill -l command to show all options:

- SIGKILL (9): abruptly and immediately stop a process
- SIGTERM (15): stand way of stopping a process, cleanly shutdown of the process
- SIGHUP (1): tell a process to reload its configuration files
- SIGSTOP (19): pause a process
- SIGCONT (18): resume a process
- ...

> ℹ Killall can be used when you want to kill multiple processes with commands with the same name. `killall -9 vim` would kill all processes which run the command vim.

> ℹ Processes are unable to ignore the signals SIGKILL and SIGSTOP. For even more info about the signals use man 7 signal. When multiple signal numbers are listed, use the middle one.

With the nice command, a process can start with a given nice-value or priority. This value gives the process priority to use the CPU. -20 is the best or highest nice-value and 19 the worst or lowest. A normal user can only use a positive value from 0 till 19.

```bash
student@linux-ess:~$ nice -n 10 sleep 100 &
[4] 11986
student@linux-ess:~$ ps -ao user,ni,comm | grep sleep
student   10 sleep
student@linux-ess:~$ nice -n -15 sleep 100 &
[6] 11995
student@linux-ess:~$ nice: cannot set niceness: Permission denied
student@linux-ess:~$ ps -ao user,ni,comm | grep sleep
student    10 sleep
student    0  sleep
```

Use the renice command to change the priority of a running process. Only root can set the priority higher.

```bash
student@linux-ess:~$ nice -n 0 sleep 100 &
[1] 12011
student@linux-ess:~$ sudo renice -n -5 12011
12011 (process ID) old priority 0, new priority -5
student@linux-ess:~$ ps ao user,ni,comm | grep sleep
student   -5 sleep
```

‹ Previous
10 Automation

Next ›
Lab