OS-LAB-1

Members: **2017-EE-118**
2017-EE-119
2017-EE-135

1. In this question, we will understand the hardware configuration of your working machine using
the /proc filesystem.
(a) Run command more /proc/cpuinfo and explain the following terms: processor and cores. [Hint: Use lscpu to verify your definitions.]
(b) How many cores does your machine have?
(c) How many processors does your machine have?
(d) What is the frequency of each processor?
(e) How much physical memory does your system have ?
(f) How much of this memory is free ?
(g) What is total number of number of forks since the boot in the system ?
(h) How many context switches has the system performed since bootup ?

SOLUTION)

A) Run command more /proc/cpuinfo and explain the following terms: processor and cores. [Hint: Use lscpu to verify your definitions.]
CPU is an electronic circuit inside the computer that carries out instruction to perform arithmetic, logical, control and input/output operations
The **core** is an execution unit inside the **CPU** that receives and executes instructions.

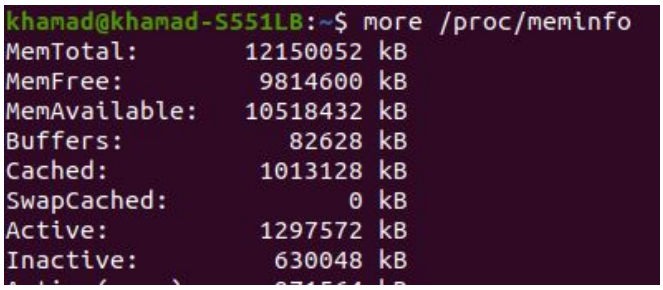
```
khamad@khamad-S551LB:~$ more /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 69
model name      : Intel(R) Core(TM) i7-4500U CPU @ 1.80GHz
stepping        : 1
microcode       : 0x26
cpu MHz         : 952.290
cache size      : 4096 KB
physical id     : 0
siblings        : 4
core id         : 0
cpu cores       : 2
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
```

B) How many cores does your machine have?
Core(s) per socket: 2

C) How many processors does your machine have?
CPU(s): 4

D) What is the frequency of each processor?
cpu MHz : 1322.489

E) How much physical memory does your system have ?



Using command more /proc/meminfo
MemTotal: 12150052 kB
MemFree: 8373276 kB
MemAvailable: 9709424 kB
Cached: 1664156 kB

G) What is the total number of forks since the boot in the system ?

Using command vmstat -f
33138 forks

H) How many context switches has the system performed since bootup ?

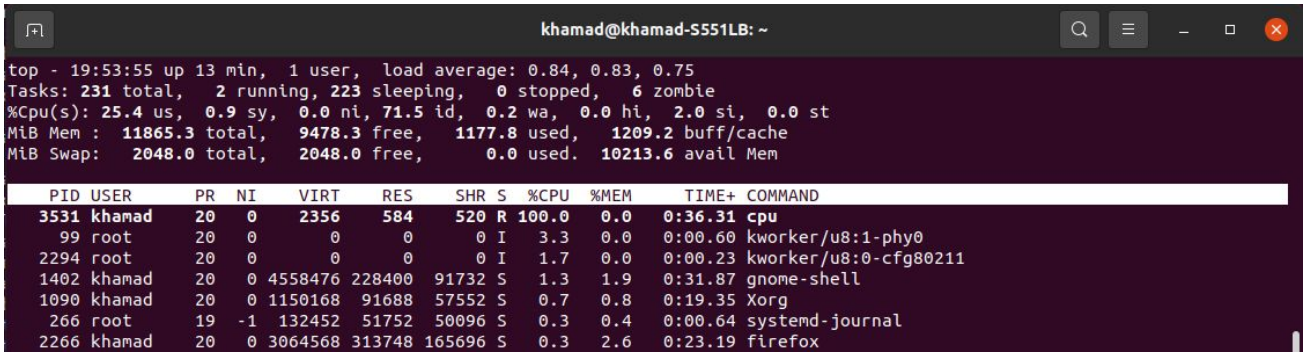
Using command grep ctxt /proc/$pid/status
voluntary_ctxt_switches: 88
nonvoluntary_ctxt_switches: 0

2. In this question, we will understand how to monitor the status of a running process using the top
command. Compile the program cpu.c given to you and execute it in the bash or any other
shell of your choice as follows.

$ gcc cpu.c -o cpu
$ ./cpu

This program runs in an infinite loop without terminating. Now open another terminal, run the
top command and answer the following questions about the cpu process.
(a) What is the PID of the process running the cpu command?
(b) How much CPU and memory does this process consume?
(c) What is the current state of the process? For example, is it running or in a blocked state or a
zombie state?

Solution)



A) What is the PID of the process running the cpu command?
PID= 32477

B) How much CPU and memory does this process consume?
%CPU=100.0 %MEM= 0.0

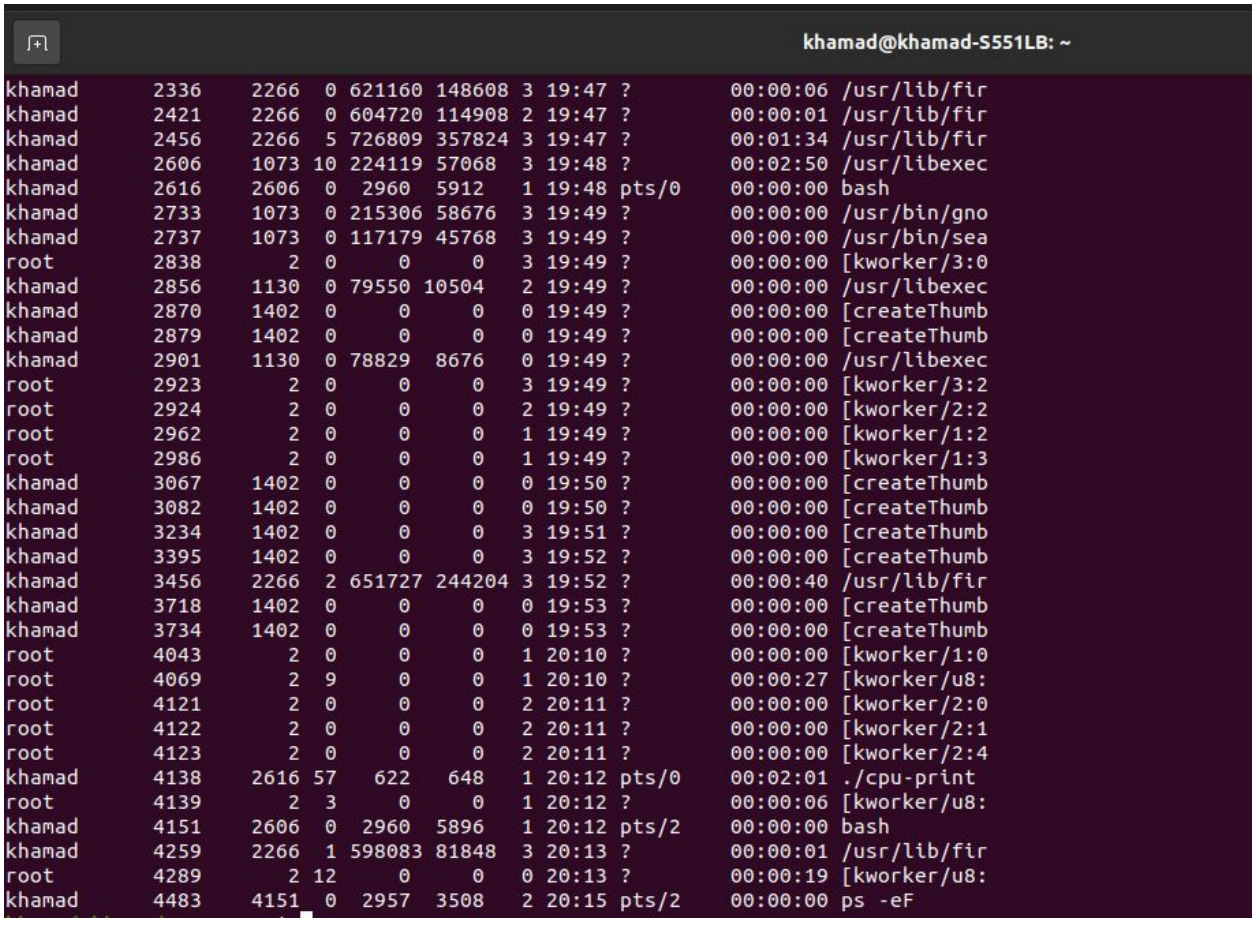
C) What is the current state of the process? For example, is it running or in a blocked state or a zombie state?
RUNNING STATE AS INDICATED IN s STATES

3. In this question, we will understand how the Linux shell (e.g., the bash shell) runs user commands
by spawning new child processes to execute the various commands.
(a) Compile the program cpu-print.c given to you and execute it in the bash or any other shell of your choice as follows.

$ gcc cpu-print.c -o cpu-print
$ ./cpu-print

This program runs in an infinite loop printing output to the screen. Now, open another terminal and use the ps command with suitable options to find out the pid of the process spawned by the shell to run the cpu-print executable. You may want to explore the ps command thoroughly to understand the various output fields it shows.
(b) Find the PID of the parent of the cpu-print process, i.e., the shell process. Next, find the PIDs of all the ancestors, going back at least 5 generations (or until you reach the init process).

4138-> 2616-> 2606-> 1073->1 (init)



(c) We will now understand how the shell performs output redirection. Run the following com-
Mand.

$ ./cpu-print > /tmp/tmp.txt &

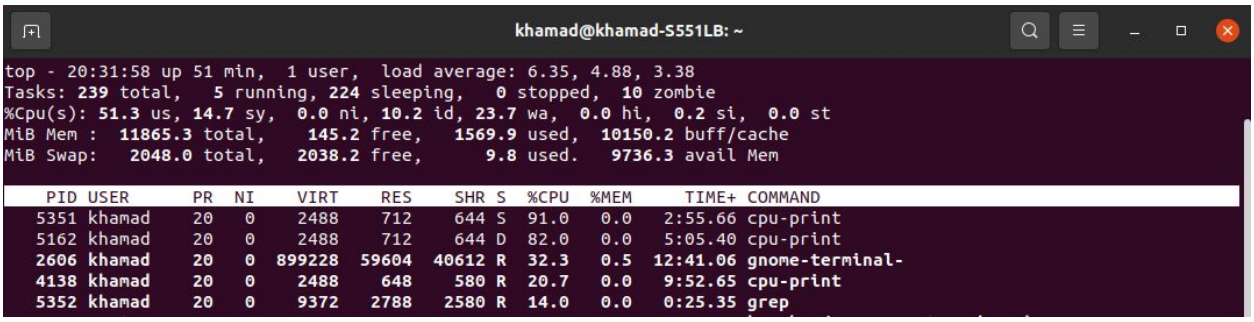
Look at the proc file system information of the newly spawned process. Pay particular atten-
tion to where its file descriptors 0, 1, and 2 (standard input, output, and error) are pointing
to. Using this information, can you describe how I/O redirection is being implemented by the shell?

```
khamad      4151    2606  0  3122  6604   2 20:12 pts/2    00:00:00 bash
root        4289       2 11     0     0   0 20:13 ?        00:01:26 [kworker/u8:5-events_unbound]
root        4493       2  0     0     0   2 20:16 ?        00:00:00 [kworker/2:0-mm_percpu_wq]
root        4495       2  0     0     0   0 20:16 ?        00:00:00 [kworker/0:0-events]
khamad      4522    2266  4 755904 422608 2 20:17 ?        00:00:22 /usr/lib/firefox/firefox -contentproc -childID 8
root        4617       2  0     0     0   3 20:19 ?        00:00:00 [kworker/3:1-events]
khamad      4635    1402  0     0     0   1 20:19 ?        00:00:00 [createThumbnail] <defunct>
root        4661       2  0     0     0   1 20:19 ?        00:00:00 [kworker/1:1-events]
root        4666       2  0     0     0   1 20:19 ?        00:00:00 [kworker/1:2-cgroup_destroy]
root        4687       2 13     0     0   2 20:19 ?        00:00:49 [kworker/u8:4-events_unbound]
root        4758       2  0     0     0   2 20:20 ?        00:00:00 [kworker/2:2-events]
khamad      4858    1402  0     0     0   3 20:22 ?        00:00:00 [createThumbnail] <defunct>
root        4979       2  0     0     0   3 20:22 ?        00:00:00 [kworker/3:3-events]
khamad      5081    2266  0 598083 83656  0 20:24 ?        00:00:00 /usr/lib/firefox/firefox -contentproc -childID 9
root        5158       2 11     0     0   2 20:25 ?        00:00:05 [kworker/u8:0-events_unbound]
khamad      5162    4151 95   622   712   1 20:25 pts/2    00:00:07 ./cpu-print
khamad      5163    4151  0  2957  3356   3 20:26 pts/2    00:00:00 ps -eF
```

(d) Next, we will understand how the shell implements pipes. Run the following command.

$ ./cpu-print | grep hello

Once again, identify the newly spawned processes, and find out where their standard in-put/output/error file descriptors are pointing to. Use this information to explain how pipes are implemented by the shell.

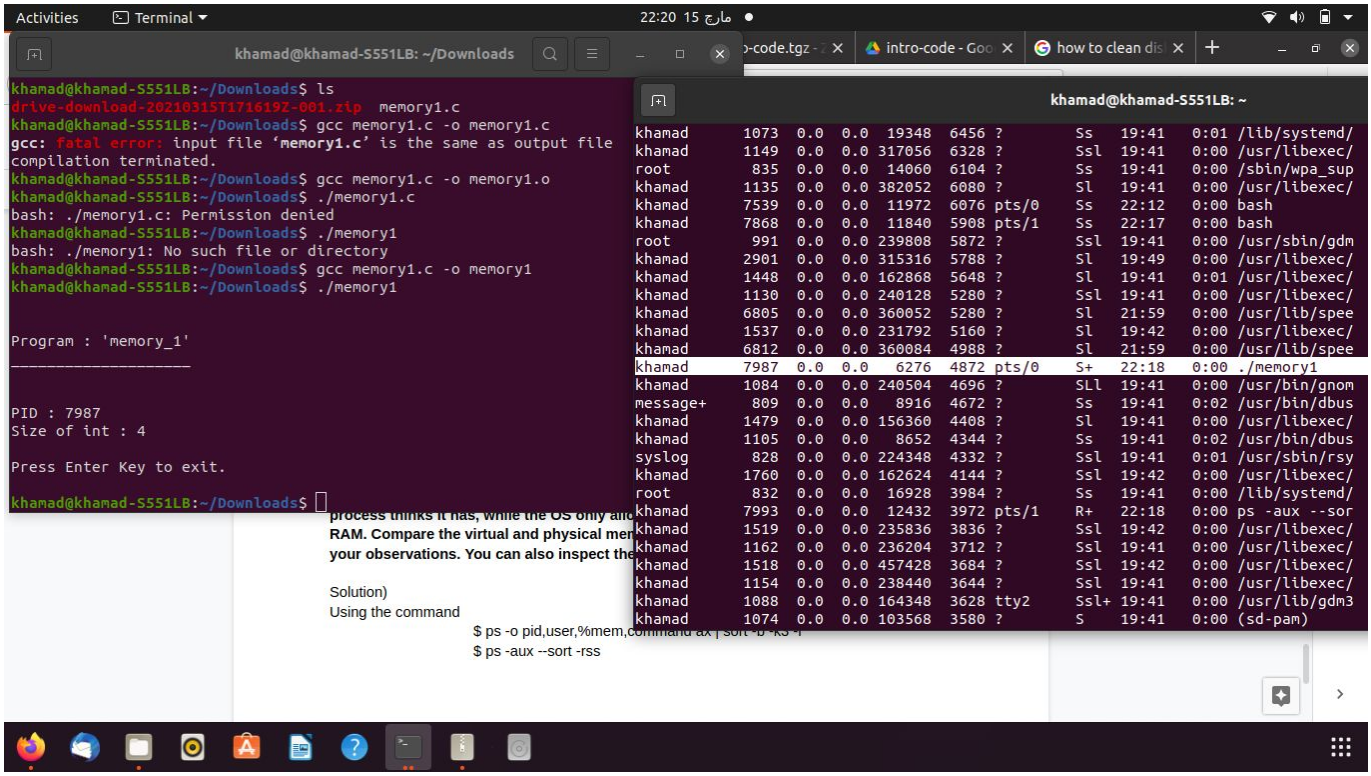


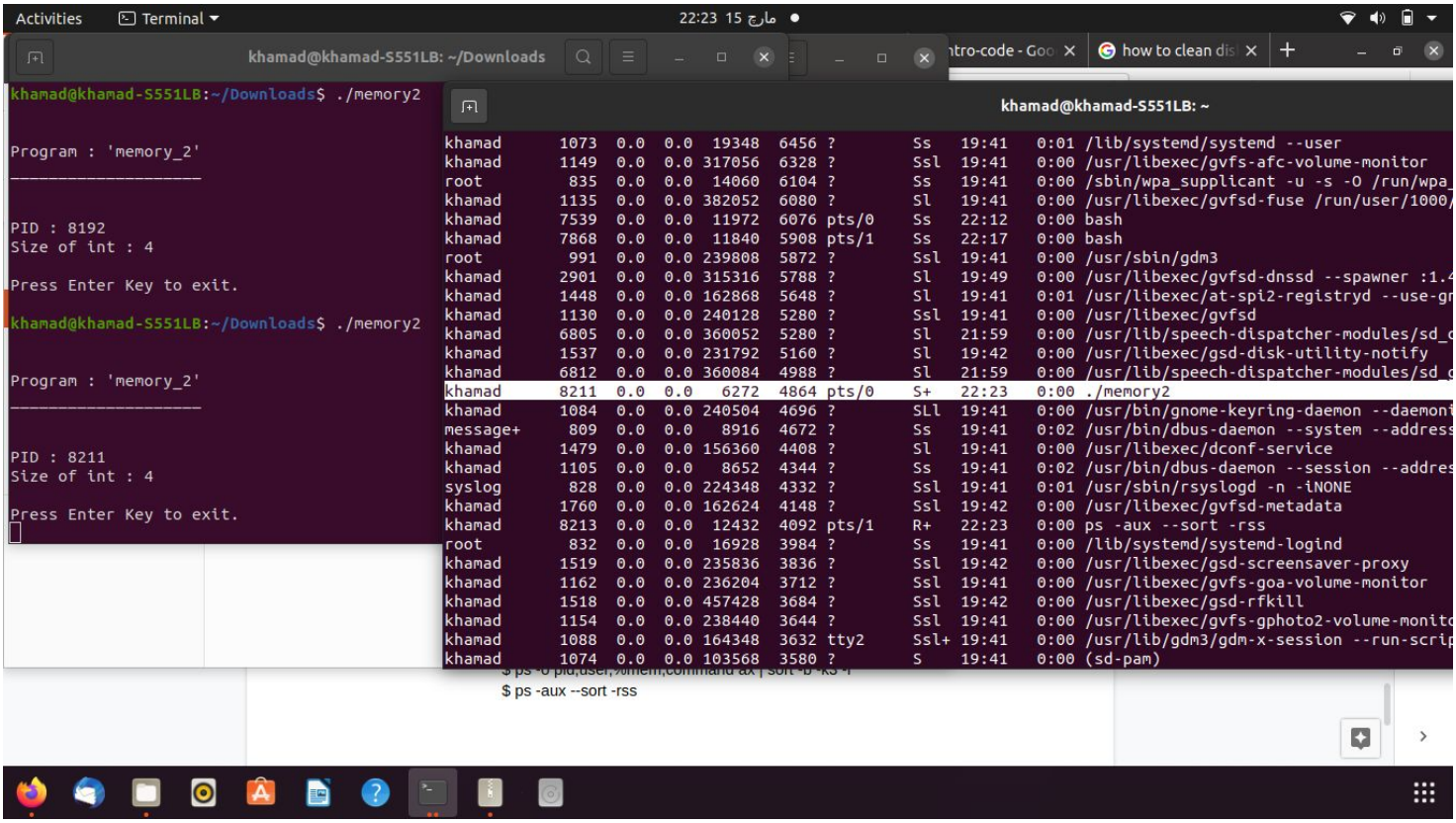
(e) When you type in a command into the shell, the shell does one of two things. For some com-mands, executables that perform that functionality already come built into the Linux kernel.For such commands, the shell simply invokes the executable like it runs the executables of your own programs. For other commands where the executable does not exist, the shell im-plements the command itself within its code. Consider the following commands that you can type in the bash shell: cd, ls, history, ps. Which of these commands already exist as built-in executables in the Linux kernel that are then simply executed by the bash shell, and which are implemented by the bash code itself?

Q4) Consider the two programs memory1.c and memory2.c given to you. Compile and run them one after the other. Both programs allocate a large array in memory. One of them accesses the array and the other doesn't. Both programs pause before exiting to let you inspect their memory usage. You can inspect the memory used by a process with the ps command. In particular, the output will tell you what the total size of the "virtual" memory of the process is, and how much of this is actually physically resident in memory. You will learn later that the virtual memory of the process is the memory the process thinks it has, while the OS only allocates a subset of this memory physically in RAM. Compare the virtual and physical memory usage of both programs, and explain your observations. You can also inspect the code to understand your observations.

Solution)
Using the command

$ ps -o pid,user,%mem,command ax | sort -b -k3 -r
$ ps -aux --sort -rss




Activities
Terminal
22:23 15 مارچ
khamad@khamad-S551LB: ~/Downloads
how to clean dis
khamad@khamad-S551LB:~/Downloads$ ./memory2
Program : 'memory_2'
PID : 8192
Size of int : 4
Press Enter Key to exit.
khamad@khamad-S551LB:~/Downloads$ ./memory2
Program : 'memory_2'
PID : 8211
Size of int : 4
Press Enter Key to exit.
khamad@khamad-S551LB: ~
khamad 1073 0.0 0.0 19348 6456 ? Ss 19:41 0:01 /lib/systemd/systemd --user
khamad 1149 0.0 0.0 317056 6328 ? Ssl 19:41 0:00 /usr/libexec/gvfs-afc-volume-monitor
root 835 0.0 0.0 14060 6104 ? Ss 19:41 0:00 /sbin/wpa_supplicant -u -s -O /run/wpa_
khamad 1135 0.0 0.0 382052 6080 ? Sl 19:41 0:00 /usr/libexec/gvfsd-fuse /run/user/1000/
khamad 7539 0.0 0.0 11972 6076 pts/0 Ss 22:12 0:00 bash
khamad 7868 0.0 0.0 11840 5908 pts/1 Ss 22:17 0:00 bash
root 991 0.0 0.0 239808 5872 ? Ssl 19:41 0:00 /usr/sbin/gdm3
khamad 2901 0.0 0.0 315316 5788 ? Sl 19:49 0:00 /usr/libexec/gvfsd-dnssd --spawner :1.4
khamad 1448 0.0 0.0 162868 5648 ? Sl 19:41 0:01 /usr/libexec/at-spi2-registryd --use-gn
khamad 1130 0.0 0.0 240128 5280 ? Ssl 19:41 0:00 /usr/libexec/gvfsd
khamad 6805 0.0 0.0 360052 5280 ? Sl 21:59 0:00 /usr/lib/speech-dispatcher-modules/sd_d
khamad 1537 0.0 0.0 231792 5160 ? Sl 19:42 0:00 /usr/libexec/gsd-disk-utility-notify
khamad 6812 0.0 0.0 360084 4988 ? Sl 21:59 0:00 /usr/lib/speech-dispatcher-modules/sd_g
khamad 8211 0.0 0.0 6272 4864 pts/0 S+ 22:23 0:00 ./memory2
khamad 1084 0.0 0.0 240504 4696 ? SLl 19:41 0:00 /usr/bin/gnome-keyring-daemon --daemoni
message+ 809 0.0 0.0 8916 4672 ? Ss 19:41 0:02 /usr/bin/dbus-daemon --system --address
khamad 1479 0.0 0.0 156360 4408 ? Sl 19:41 0:00 /usr/libexec/dconf-service
khamad 1105 0.0 0.0 8652 4344 ? Ss 19:41 0:02 /usr/bin/dbus-daemon --session --addres
syslog 828 0.0 0.0 224348 4332 ? Ssl 19:41 0:01 /usr/sbin/rsyslogd -n -iNONE
khamad 1760 0.0 0.0 162624 4148 ? Ssl 19:42 0:00 /usr/libexec/gvfsd-metadata
khamad 8213 0.0 0.0 12432 4092 pts/1 R+ 22:23 0:00 ps -aux --sort -rss
root 832 0.0 0.0 16928 3984 ? Ss 19:41 0:00 /lib/systemd/systemd-logind
khamad 1519 0.0 0.0 235836 3836 ? Ssl 19:42 0:00 /usr/libexec/gsd-screensaver-proxy
khamad 1162 0.0 0.0 236204 3712 ? Ssl 19:41 0:00 /usr/libexec/gvfs-goa-volume-monitor
khamad 1518 0.0 0.0 457428 3684 ? Ssl 19:42 0:00 /usr/libexec/gsd-rfkill
khamad 1154 0.0 0.0 238440 3644 ? Ssl 19:41 0:00 /usr/libexec/gvfs-gphoto2-volume-monito
khamad 1088 0.0 0.0 164348 3632 tty2 Ssl+ 19:41 0:00 /usr/lib/gdm3/gdm-x-session --run-scrip
khamad 1074 0.0 0.0 103568 3580 ? S 19:41 0:00 (sd-pam)
$ ps -aux --sort -rss