Tlemcen University
Faculty of Science
Computer Science Department
1st Year Engineer

Academic year: 2023-2024
Introduction to operating systems 1

# TP n°1: Process management (1)

## 1. Objectives

At the end of this practical course based on the Linux system, the student should be able to achieve the following operational skills:

- Execute the main process management commands on Linux (ps, kill, jobs),
- Use the C language to create and manipulate processes through system calls,
- Create and manipulate several processes using the *fork()* system call.

## 2. Process commands

In this section, we present the main commands for viewing processes under UNIX.

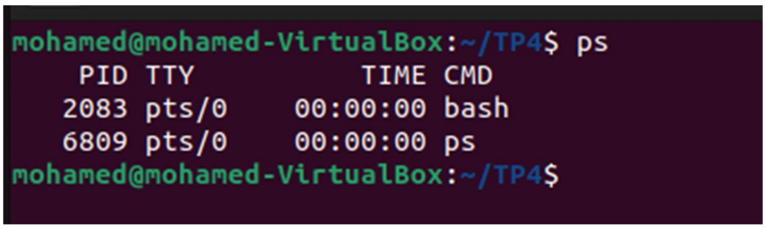
### *a) Command ps*

The command ps is used to display the list of processes associated with the current terminal. Its syntax is as follows:

```
ps [options]
```

Without any options, ps does not display the full list of processes, nor much information. It only displays processes launched from the terminal in which it was run.

Example:




**Options of command ps**

To get more details about the processes on a machine, you can use one of the options:

- e: Displays the processes currently running for all users.
- f: Displays the complete list of the format (displays additional information about the running processes).
- a: processes for all users.
- u: display user-oriented format (displays additional information about running processes).
- x: list of processes which do not belong to any TTY (terminal).
- o: This option is used to filter the column so that only the desired column is displayed. For example, to display only the ***pid*** and ***comm*** columns:

```
ps -efo pid,comm
```

The information returned on processes by the command ***ps -ef*** is as follows:

```
mohamed@mohamed-VirtualBox:~/TP4$ ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 18:15 ?        00:00:01 /sbin/init splash
root           2       0  0 18:15 ?        00:00:00 [kthreadd]
root           3       2  0 18:15 ?        00:00:00 [rcu_gp]
root           4       2  0 18:15 ?        00:00:00 [rcu_par_gp]
root           5       2  0 18:15 ?        00:00:00 [slub_flushwq]
root           6       2  0 18:15 ?        00:00:00 [netns]
root           8       2  0 18:15 ?        00:00:00 [kworker/0:0H-events_highpri]
root          10       2  0 18:15 ?        00:00:00 [mm_percpu_wq]
root          11       2  0 18:15 ?        00:00:00 [rcu_tasks_kthread]
root          12       2  0 18:15 ?        00:00:00 [rcu_tasks_rude_kthread]
root          13       2  0 18:15 ?        00:00:00 [rcu_tasks_trace_kthread]
root          14       2  0 18:15 ?        00:00:00 [ksoftirqd/0]
root          15       2  0 18:15 ?        00:00:05 [rcu_preempt]
root          16       2  0 18:15 ?        00:00:00 [migration/0]
root          17       2  0 18:15 ?        00:00:00 [idle_inject/0]
root          19       2  0 18:15 ?        00:00:00 [cpuhp/0]
root          20       2  0 18:15 ?        00:00:00 [cpuhp/1]
root          21       2  0 18:15 ?        00:00:00 [idle_inject/1]
root          22       2  0 18:15 ?        00:00:00 [migration/1]
root          23       2  0 18:15 ?        00:00:00 [ksoftirqd/1]
```

The information returned by the command ***ps -aux*** is :

- USER: the user running the process.
- %CPU: the processor utilization of the process.
- %MEM: the percentage of the size of the process's resident setting in the machine's physical memory.
- VSZ: size of the process's virtual memory in Kilobytes.
- RSS: the size of the physical memory used by the process.
- STAT: the process status code, such as Z (Zombie), S or I (Sleeping) and R (Running).

***b) The command top***

The command top displays continuous information about system activity. In particular, this command can be used to monitor the resources that processes are using (amount of RAM, percentage of CPU, how long the process has been running since it was started).

The command top can also be used to find out the load average of your server or machine. For example, the first line of the top command retrieves the following information:

```
top - 16:13:34 up 540 days,  2:54,  1 user,  load average: 1.30, 1.77, 1.65
```

- top - : command reminder.
- 16 :13 :34: machine time.
- up xx days (uptime): the time since which the machine has been running without interruption. Restarting the machine resets the uptime.
- n users: number of users currently logged on to the server.
- load average 1.30, 1.77, 1.65: average load of the machine (represents the average number of processes) is divided into 3 parts: the first is calculated over the last minute, the second 5 minutes and the last 15 minutes. In this example, an average of 1.30 processes have used the processor over the last minute, 1.77 over the last 5 minutes and 1.65 over the last 15 minutes.

The second line contains the following information:

```
Tasks: 125 total,   2 running, 115 sleeping,   0 stopped,   8 zombie
```

- Number of tasks (Task 125 total) : Total number of processes,
- Number of processes (2 running): Number of active processes,
- Number of sleeping processes (115 sleeping): Number of sleeping processes, A sleeping process does nothing. It waits for a condition to become executable or active again.
- Number of stopped processes (0 stopped): Number of stopped processes. In this state, a process has been stopped, usually by receiving a signal. For example, a process that is being debugged.
- Number of zombie processes (8 zombie): These are processes that have actually finished running. It has finished executing and therefore no longer has any reason to exist, but it still has an entry in the process table. Only for various possible reasons, its parent has not been informed of this.

**Use of the processor**

```
Cpu(s):  9.4%us,  1.9%sy,  0.0%ni, 88.6%id,  0.1%wa,  0.0%hi,  0.0%si,  0.0%st
```

- xx%us (9.4%us): CPU time used by user processes.
- yy%sy (1.9%sy): CPU time used by the kernel and its processes (system processes).
- 0.0%ni: CPU time used by user processes that have been "nicated: processes launched with the command nice " (a "nicated" process gives priority to other processes).
- 88.6%id: Unsolicited CPU time.
- 0.1%wa: CPU time for waiting for I/O, if this figure is large all the time, such as 20 or more for example, this means that your computer has instructions waiting to be processed and is probably starting to struggle to do what you want it to do. This value should be close to 0 most of the time.
- 0.0%hi: CPU time used for hardware interrupts.
- 0.0%si : CPU time used for software interrupts.
- 0.0%st: CPU time "stolen" from this virtual machine for other tasks (for example using another VM). We know that when a VM is used, resources can be shared, especially CPU.

**Use of physical and virtual memory**

- Mem: Physical memory (total, used, free and buffers).
- Swap: Virtual memory (total, used, free and cached). This memory allows applications to use more RAM than the machine physically contains. This is equivalent to using a mass storage medium (hard disk) to simulate memory.

**Process details**

```
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
  313 mysql     15   0  367m 217m 5824 S    5  2.7  69500:08 mysqld
 9244 nagios    15   0     0    0    0 Z    4  0.0   0:00.11 centstora <defunct>
 2952 root      34  19     0    0    0 S    3  0.0   5729:52 kipmi0
13362 nagios    15   0  4648 1752 1104 S    0  0.0   3:02.80 ndo2db
13363 nagios    25   0 18088 4480 1252 S    0  0.1  27:09.74 nagios
    1 root      22   0  1852  572  488 S    0  0.0 112:39.53 init
    2 root      RT   0     0    0    0 S    0  0.0   2:17.27 migration/0
    3 root      34  19     0    0    0 S    0  0.0   0:07.80 ksoftirqd/0
    4 root      RT   0     0    0    0 S    0  0.0   4:07.45 migration/1
    5 root      34  19     0    0    0 S    0  0.0   0:05.28 ksoftirqd/1
```

- PID: The process PID (the processes running simultaneously on a machine).
- USER: The user running this process.
- PR: The priority of the process (the smaller the number, the higher the priority).
- NI: The nice of the process.
- VIRT: Virtual size of a process, i.e. the amount of memory it actually uses in memory (not just RAM).
- RES: Amount of physical memory occupied by the process.
- SHR: Indicates how much of the VIRT column is actually shared.
- S: Process status. S (sleeping), D (uninterruptible sleep), R (running), Z (zombie), or T (stopped or traced).
- %CPU: CPU load.
- %MEM: Memory load.
- TIME+: Total processor usage time since the process was launched.
- COMMAND: Process name.

To switch top to background mode, type ***CTRL+Z***, and it will display :

[job-number]+ stopped being the PID of the top process, if you want to restart it in the background simply type the command :

```
bg job-number
```

To end the command ***top***, simply press the "q" key.

***c) The command pstree***

This command is used to list the relationships between processes by means of a graphical representation of the filiation relationships between processes. You can use the command “pstree”. The “-p” option also displays the process pid.

```
pstree -p
```

***d) The command jobs***

The command ***jobs*** is used to display the list of tasks (processes) in the current Shell (suspended or running in the background) with their job numbers, PIDs and process states.

Example :

```
mohamed@mohamed-VirtualBox:~/TP4$ time ls -lR / > list.ls 2>/dev/null
^Z
[1]+  Arrêté                 ls --color=auto -lR / > list.ls 2> /dev/null

real    0m1,025s
user    0m0,000s
sys     0m0,000s
mohamed@mohamed-VirtualBox:~/TP4$ time ls -lR / > list.ls 2>/dev/null &
[2] 11298
mohamed@mohamed-VirtualBox:~/TP4$ jobs
[1]+  Arrêté                 ls --color=auto -lR / > list.ls 2> /dev/null
[2]-  En cours d'exécution    time ls --color=auto -lR / > list.ls 2> /dev/null &
mohamed@mohamed-VirtualBox:~/TP4$
real    0m7,734s
user    0m2,794s
sys     0m4,635s
```

The information in square brackets represents the job number. It is followed by the PID, the state of the process and its name (command).

### *e) The command fg (foreground)*

This command is used to restart the execution of a process in the background as a process in the foreground using the following syntax:

***fg %n***

n is the job number

### *f) The command bg (background)*

The bg command is used to restart the execution of a suspended process as a background process. Its syntax is :

***bg %n***

n is the job number.

### *g) The command kill/killall*

The command kill is used to send a signal to a process with its PID number. To terminate a process, its PID is first discovered, then the PID is passed to the kill command as argument.

If a process does not respond to a TERM signal, the KILL signal can be used. You can use kill with the -9 option using the following syntax:

***kill -9 pid***

The command killall is used to kill several processes at once. The syntax is the same, simply indicate the name of the process. For example, if you want to kill all the gedit processes, use the following command:

***kill -9 gedit***

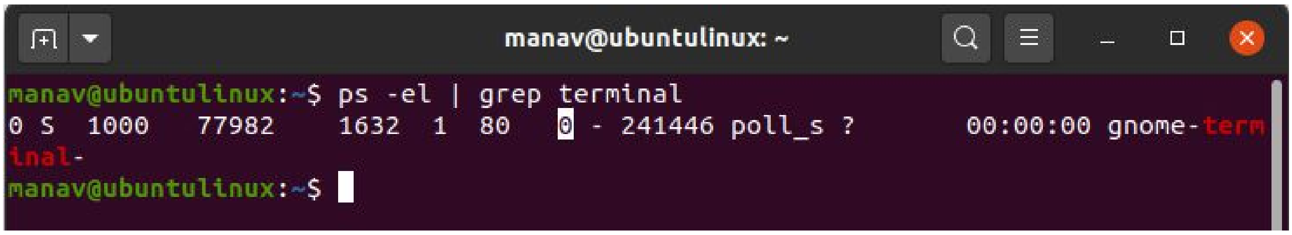
### ***h) The commands nice and renice***

**The command nice** helps in execution of a process with modified scheduling priority. It launches a process with a user-defined scheduling priority. In this, if we give a process a higher priority, then Kernel will allocate more CPU time to that process.

**The renice command** allows the user to change and modify the scheduling priority of an already running process. Linux Kernel schedules the process and allocates CPU time accordingly for each of them.

1- To check the nice value of a process, you can use the following command:

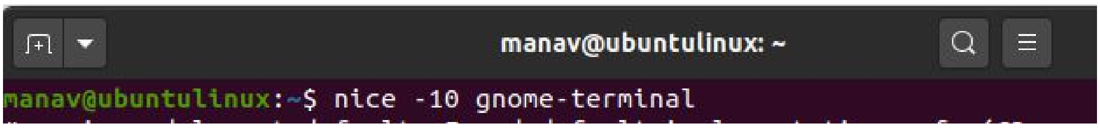
`ps -el | grep terminal`    (example of a process whose name contains the character string “terminal”)




The eight highlighted value (column) is the nice value of the process.

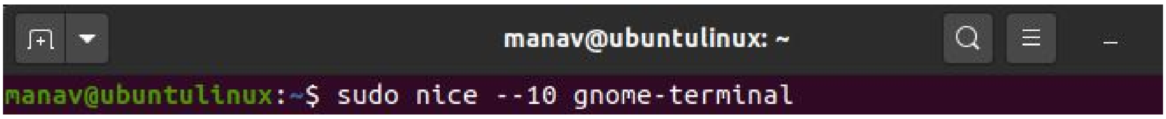
2- To set the priority of a process

Example : `nice -10 gnome-terminal`



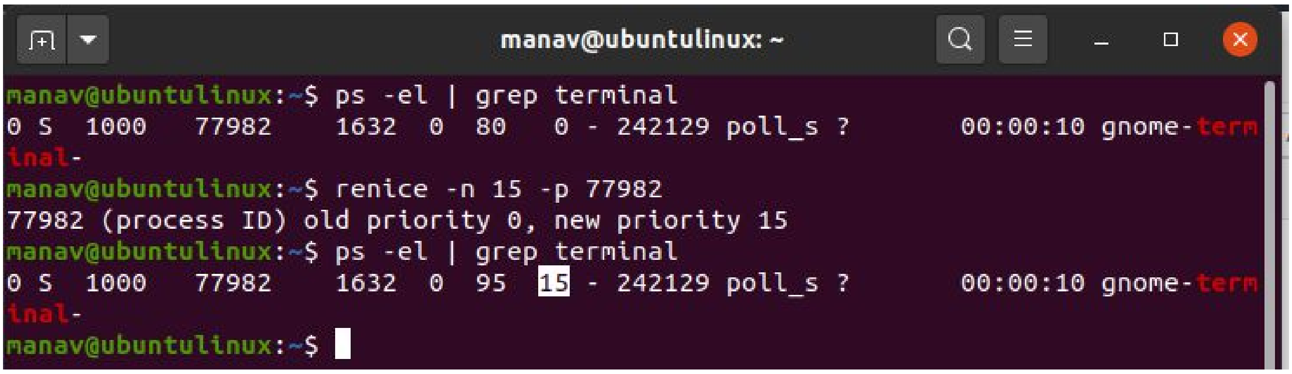

3- To set the negative priority for a process

**Example :** `nice --10 gnome-terminal`




4- Changing priority of the running process.

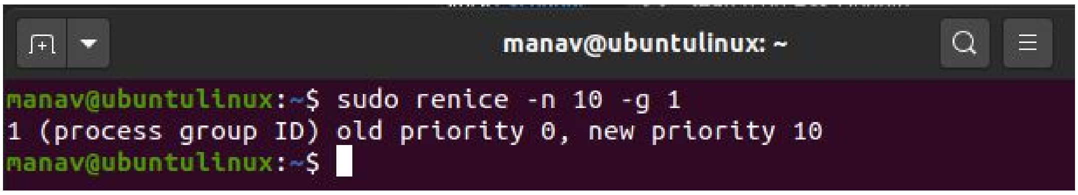
**Example :** `sudo renice -n 15 -p 77982`




This will change the priority of the process with pid 77982.

5- To change the priority of all programs of a specific group.

**Example :** `renice -n 10 -g 4`




This command will set all the processes of gid 4 priority to 10.

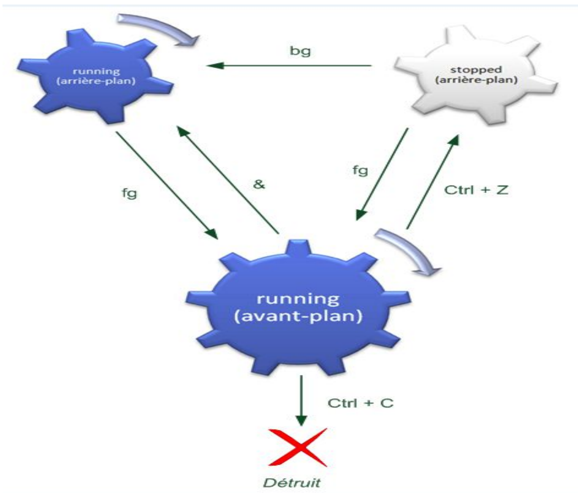
6- To change the priority of all programs of a specific user.

**Example :** `sudo renice -n 10 -u 2`

This will set all the processes of user 2 to 10.

## 3. Switching between the different states of a process

Switching between the different states of a process is illustrated in the figure below:




## 4. Creation of processes (primitive fork())

A process can create a new process using the following function:

```
int fork(void)
```

The new process runs concurrently with the process that created it.

A call to fork() by a process, called the parent process, instructs UNIX to create a new process called the child process, which is an exact copy of the current process in most of its attributes.

This function returns:

- -1 on failure (the process child is not created),
- 0 in the child process,
- The number of the child process (PID) in the parent.

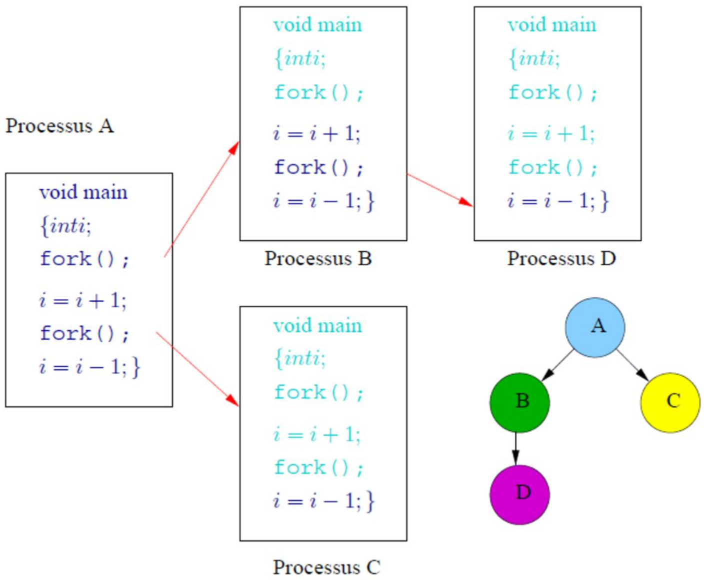
The two processes are exactly the same and run the same program on two separate copies of the data. The data spaces are completely separate: changing a variable in one is invisible in the other.

You generally want to run separate tasks in the parent and child processes. The value returned by fork() is therefore very important in differentiating the parent process from the child process. A test can be used to execute different parts of the code in the parent and child processes.

```
int code_retour ;
code_retour = fork ();
switch (code_retour ) {
 case -1 :
   printf ("Process creation failed \n");
   break;
 case 0 :
   printf ("I am the process child \n");
   break;
 default :
   printf ("I am the process parent \n");
   printf ("I have just created the child process whose pid is %d \n",code_retour);
   break;
}
```

A process has a single parent and can have 0 or more children.

To illustrate how the fork system call works, the details of which will be seen later, we offer the following example:




The first fork system call by process A creates process B and the second call creates process C, while the fork system call by process B creates process D. Each of the processes executes the parts that are highlighted, as shown in the figure above.

The generated tree shows the parent-child relationship between the processes.

# Exercises

**Exercise n°1**

1) What is the process of *pid* 1? justify
2) What is the difference between the commands *ps* and *top*?
3) What does the command *pstree* do?
4) How can the command *ps* be used to obtain the list of processes in the first column and their state in the 2nd column? What are the possible states?

**Exercise n°2**

1) Run the text editor “gedit” from a terminal (just by typing gedit) and enter a sentence. Return to the terminal and try to run the command *ls*. What happens and why?
2) Stop the process “gedit”. Go back to the window “gedit” and try to enter some text. What happens? Why is this?
3) Run “gedit” again and then stop it. Display the current processes in your terminal using the command *jobs*. In what state are these processes?
4) Run a third “gedit”, this time in the background. Use the command *jobs* to display the processes in your terminal. What is the difference between the first two “gedit” and the third? How do you explain this?
5) Kill the first “gedit” with the command *kill*.
6) Run the second “gedit” in the background. Check the status of your two “gedit” with the command *jobs*.
7) Kill the two remaining “gedit” processes with the command kill.

**Exercise n°3**

Use the compiler “c” under Linux “gcc” to compile the following program.

#gcc -c filename.c

#gcc -o exe-name filename.o

#./exe-name

```
#include <stdio.h>
#include <unistd.h>
void main(void){
int pid;
pid = fork();
if (pid == -1)
printf("Creation Error \n");
else if (pid == 0){
      printf("I am the child: pid = %d and my parent is ppid = %d \n",
      getpid(),getppid());
      sleep(22);
      printf("I am the child: pid = %d and my parent is ppid = %d \n",
      getpid(),getppid());
      sleep(20);
      }
     else{
      printf("I am the parent: pid = %d\n", getpid());
     sleep(20);
     }
}
```

1) What is the pid of the parent and child?
2) Open another terminal and use the appropriate command to check the pid of the parent and child?
3) Run the process in the background.
4) Return to the foreground
5) Suspend execution of this process (stop the process).
6) Stop the execution of this process in the second terminal in one of two ways.

## Exercise n° 4

1) Create a small program in C that you call “count” and that displays numbers from 1 to infinity (a large number).
2) Run “count”. Stop it using Ctrl-z. Check its status with jobs.
3) Switch “count” back to the foreground. Does the program display the numbers from 1? Why is this? Kill “count” with Ctrl-c.
4) Read the manual page for the command ***yes***. Display a sentence of your choice on the screen using this command. Kill the command *yes* with Ctrl-C.
5) Run your program “count” in the background without any output appearing on the screen (use /dev/null). Check that the program “count” is actually running with the command *top*. What is its *nice* level?
6) Run ***yes*** with no output on the screen and in the background with a nice level of +10. Run *top* again. What is the priority level of the command ***yes*** in relation to the program "count"?
7) Run another ***yes*** with no output on the screen and in the background with a nice level of +19. Check it with the command *top*. What are the priority levels of the three processes?
8) Change the nice level from “count” to +10 and run the command *top*. Look closely at the two processes with the same nice level (at +10). Does one have higher priority than the other?
9) Try to change the nice level of the second ***yes*** (the one you ran at +19) to -15. What happens? Why or why not?