Solution: TP n°1: Process management (1)

Exercise n°1

1) What is the process of *pid* 1? Justify

The PID=1 process with the name init is the parent of all the other processes (since process 0 no longer performs fork()), it is this process which takes in all the processes which have no parent (in order to collect information when each process dies).

2) What is the difference between the commands *ps* and *top*?

`ps` and `top` are both commands used to display information about running processes on a Linux system. However, there are some differences between them.

`ps -ef` displays all processes running on the system, while `top` displays real-time information about processes that are currently running.

3) What does the command *pstree* do?

pstree shows the running processes as a tree which is a more convenient way to display the processes hierarchy and makes the output more visually appealing. The root of the tree is either init or the process with the given pid. pstree can also be installed in other Unix systems.

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?

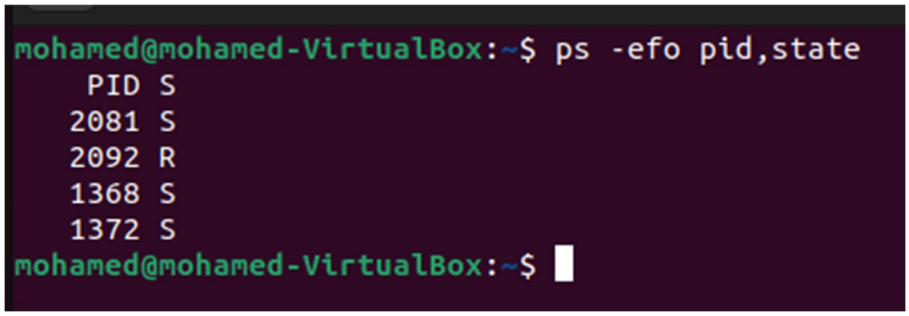
We use the following command:

```
ps -efo pid,state
```

The different states of a process

R	**Running**
I	**Sleepy (> 20 s)**
S	**Sleepy (< 20 s)**
D	**Waiting for a disk operation**
T	**Interrupted**
Z	**Zombie**

Example



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?

   This is not possible because the shell does not allow the user to enter another command. In this case, either start a new shell session via a terminal or run gedit in the background.

2) Stop the process "gedit". Go back to the window "gedit" and try to enter some text. What happens? Why is this?

   We can't type because the gedit process is in the suspended state (it is not in the running state).

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?

   The process is in the suspended state (stopped).

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?

   The first two are in the suspended state, while the third is in the background execution state.

5) Kill the first "gedit" with the command *kill*.

   jobs -l , // to retrieve the pid

   kill -9 pid

6) Run the second "gedit" in the background. Check the status of your two "gedit" with the command *jobs*.

   ```
   jobs -l
   bg num-job
   ```

7) Kill the two remaining "gedit" processes with the command kill.

   ```
   jobs -l      // to retrieve the pid
   kill -9  pid
   ```

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?

Just look at the values returned by getpid()

2) Open another terminal and use the appropriate command to check the pid of the parent and child?

Simply use the following command :

ps -f -U USER | grep prog-name (username your session name)

3) Run the process in the background.

./program-name &

4) Return to the foreground

First run the jobs command to retrieve the job number, then the fg command followed by the job number:

```
jobs
fg job-number
```

5) Suspend execution of this process (stop the process).

```
CTRL-z
```

6) Stop the execution of this process in the second terminal in one of two ways.

Either retrieve its pid (you can use the pidof command) and then inject it into the kill command as an argument, or use the jobs command to retrieve its job number and then inject it into the kill command as an argument or its pid.

```
kill -9 pid
kill -9 $(pidof prog-name)
kill -9 'pgrep prog-name'          (one of three)
Or
jobs -l    to retrieve it pid
kill -9   pid
```

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).

Example :

```
#include <stdio.h>
int i;
main()
{
 for(i=1 ;i <=1000000000 ;i++)
 printf("i = %d ",i);
}
```

2) Run “count”. Stop it using Ctrl-z. Check its status with jobs.

   It is in a suspended state

3) Switch “count” back to the foreground. Does the program display the numbers from 1? Why is this? Kill “count” with Ctrl-C.

   Because it will continue from the point of its suspension

   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.

   *Simply type the command **yes** followed by the string of your choice*

   ***yes** displays the string passed as an argument without stopping*

   $yes string

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?

   $ ./count > /dev/null &

   Type the command top and see the column NI which represents the priority level of the process.

   The NI of the count process is 0.

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"?

   $nice -n 10 yes > /dev/null &

   The priority level of yes is lower than that of the count process (yes's NI is 10 whereas count's NI is 0).

   The lowest value of NI means that the process has priority.

   NI varies between -20 and +19

   -20 is the highest priority

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?

   $nice -n 19 yes > /dev/null &

   The process “count” is the highest priority with NI = 0

   The process representing the yes in question 6) is second with a priority of 10

   The last is the process “yes” which represents the yes of question 7) with the lowest priority which is 19.

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?

   We retrieve the pid of the "count" process

   $renice -n 10 pid-of-count

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?

   It is not possible to increase the priority of a process as a single user. You need to be a superuser to increase the priority.

   $sudo renice -n -15 pid-du-second-yes