TP n°1: Process management (1)

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?