TD N°2 : Process management

Course reminder

The primitive ***fork()*** creates a child process of the calling process (the parent), with the same program as the latter. The value returned by *fork()* is :

- -1: In the event of failure, for example if the process table is full.
- 0 : the value returned to the child, and its *pid* and the *pid* of its parent can be found by calling the functions *getpid()* and getppid()respectively.
- A positive integer: the value returned to the parent, which represents the *pid* of its created child.

A process can end its execution by calling the primitive ***exit()***:

```
#include <stdlib.h>
void exit(int status);
```

The process passes its return code as an argument to *exit()*: an integer between 0 and 255. In the C language, executing "***return n"*** from the function *main()* is equivalent to calling ***exit(n)***. A process can also be terminated by a signal, sent by another process or by the operating system itself in the event of abnormal termination.

A parent process can obtain information about the termination of a child using the call ***wait()***:

```
#include <sys/wait.h>
pid_t wait(int * pointer_to_status);
```

When *wait()* is called, the parent process goes to sleep waiting for one of its children to die. When a child dies, *wait()* returns the PID of the dead child.

Exercise n°1

1) What does the following program display?

```
int main() {
   pid_t pid;
   int x = 1;

   pid = fork();
   if (pid == 0) {
       printf("Dans fils : x=%d\n", ++x);
       exit(0);
   }
   printf("Dans pere : x=%d\n", --x);
   exit(0):
}
```

2) Illustrate the execution of the following programs through a tree structure, explaining the display of each process and its filiation.

Program 1

```
int main() {

    fork();
    fork();
    printf("hello!\n");
    exit(0);
}
```

Program 2

```
int main() {

    fork();
    fork();
    fork();
    printf("hello!\n");
    exit(0);
}
```

3) How many lines does "TLEMCEN !" print for each of the following programs?

Program 1

```
int main() {
    int i;
    for(i=0; i<3; i++)
    fork();
    printf("TLEMCEN!\n");
    exit(0);
}
```

Program 2

```
int main() {
    if (fork())
    fork();
    printf("TLEMCEN!\n");
    exit(0);
}
```

Program 3

```
int main() {
    if (fork()==0) {
        if (fork()) {
          printf("TLEMCEN!\n");
          }
    }
}
```

Exercise n°2

Let's consider the following two programs:

Program 1

```
int main(void) {
    if(fork() && (fork() || fork()));
    sleep(5);
    exit(0);
}
```

Program 2

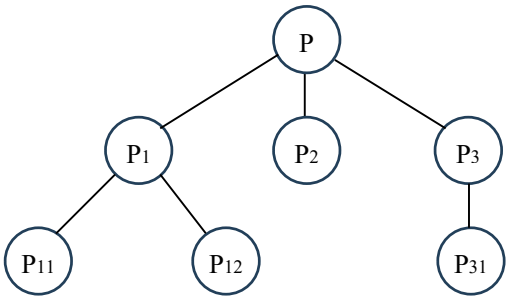
```
int main ( void ) {
    int i;
    for (i =0; i <3; i ++)
    if (fork ()) i ++;
    sleep (5);
    return i ;
}
```

Assume that any call to the *fork()* primitive does not return an error code.

1) How many processes are created by each of these two programs (not counting the parent)?
2) Draw the tree structure of the parent and the processes created (the family tree) for each program.
3) Suggest a shell command line to check your answers.

Exercise n°3

Write a program in C language to create the following tree structure and display the *pid* of each process and that of its parent.



Exercise n°4

1) Write a program that creates 10 child processes, each of which will display its sequence number between "0" and "9" 1000 times, i.e. the first child created has sequence number "0" and will have to display the number "0" 1000 times, and the second child created has sequence number "1" and will have to display the number "1" 1000 times, and so on for the other processes created.
2) Check that this program displays 10000 characters.
3) How will the display appear if system calls are not used to synchronize the processes created?
4) Suggest a mechanism for synchronizing the processes created so that the display shows 1000 "0" one after the other, then 1000 "1" one after the other and so on, i.e. a process created will not start the display until its predecessor has finished.