Lab1: Programmer's view, creating processes

subject

- Display the "manual" of a command : `man`
- Create a new file : `touch myfile.c`
- Editing a file : `nano myfile.c`
- Running a file : `gcc -o execName myfile.c` then `./execName` to display results.

Creating and running a process - fork

2- What happens after a `fork` call ?

A `fork()` is a system call that takes no arguments. It creates a child process which is a copy of the actual process. After creation of the child process, both processes start execution from the next instruction.

How are parent and child differentiated ?

`fork()` can return 3 things :

- The PID of the child process (> 0), into the parent process.
- 0, into the child process.
- An error (< 0), into the parent process when the child process can't be created.

The PID (process id) of the child is different than it's PPID (parent process id). We can find them using `getpid()` and `getppid()`.

3- Small program that makes the difference between child and parent

```
#include <stdio.h>
#include <unistd.h>

int main() {
  int p = fork();
  if (p == 0) {
    printf("I'm the child.\n");
  } else {
    printf("I'm the parent.\n");
  }
  printf("Process id = %d\n", getpid());
  printf("Parent process id = %d\n", getppid());
}
```

Output :

```
I'm the parent.
I'm the child.
Process id = 4797
Process id = 4798
Parent process id =110
Parent process id = 4797
```

It seems that both processes are running at the same time. Let's try something else :

```
int p = fork();
if (p == 0) {
  printf("I'm the child.\n
          Process id = %d\n
          Parent process id = %d\n"
          , getpid(), getppid());
} else {
  printf("I'm the parent.\n
          Process id = %d\n
          Parent process id = %d\n"
          , getpid(), getppid());
}
```

Output :

```
I'm the parent.
I'm the child.
Process id = 4816
Process id = 4817
Parent process id =110
Parent process id = 4816
```

The output is the same (except for the values). We can conclude that both processes are running exactly at the same time.

However, it is still possible to make 3 observations :

- One *Process id* is the same as one *Parent process id* = one is the parent of the other.
- Both *Process id* are very close = they are created almost at the same time.
- One *Parent process id* is very low = we can think that it is the parent of the parent (in addition, the id outputted (110) is the same for both version of code).

With these observations, the most logical association would be :

```
Parent :
Process id = 4816
Parent process id =110

Child :
Process id = 4817
Parent process id = 4816
```

4- Is data shared between parent and child ?

```
int i = 5;
if (fork() == 0) {
  // I’m the child
  i++;
} else {
  // I’m the parent
  sleep(3); // sleep for 3 seconds
  printf("%d\n", i);
}
```

Output :

```
5
```

This result means that processes aren't sharing data. If it was the case, the output value would have been `6`.

5- Is it possible to create more than one child process ?

Yes, we can use several `fork()` in the same program.

```
#include <stdio.h>
#include <unistd.h>

int main() {
  fork();
  fork();
  printf("I am a process!\n");
}
```

Output :

```
I am a process!
I am a process!
I am a process!
I am a process!
```

Creating and Running a Process - exec

2- Run *vim* using on of the `exec` functions. Is the process id of the new running application different from the original one ?

```
#include <stdio.h>
#include <unistd.h>

int main() {
  // display the process id
  printf("Process id: %d", getpid());

  // simply use any exec call
  execl("/usr/bin/vim", "vim", NULL);

  return 0;
}
```

This program outputs `Process id: 7133` and runs *vim*.

Observations :

- If we do `CTRL + C` on the terminal, *vim* closes.
- If we close *vim*, the terminal output `Exiting due to channel error`.

When the program was running, executing `pidof vim` in an other terminal outputed `7133` which is the same pid as the script that we launch in the first terminal. The process id of the new running application is the same that the original one. This result explains our observations.

We choose to use *vim* instead of *firefox* because we wanted to use the `pidof` function to know the pid of a running application. As *firefox* runs multiple processes at the same time, `pidof` is easier to use with *vim*.

3- Is data shared by the parent and child processes and to what extent ?

As the pid is the same for both applications, they are sharing the same data. Also, we can directly pass data to the second application through exec's args.

4- Explain what happens in the following program. What is the main difference with the previous version ?

```
int i = 5;

if (fork() == 0) {
  execl("usr/bin/vim", "vim", NULL);
  i++;
  printf("%d\n", i);
} else {
  printf("%d, %d\n", getpid(), i);
}
```

Output

```
8512, 5
6
```

Conclusions :

- Data aren't shared between parent and child processes.
- exec functions doesn't create child processes.