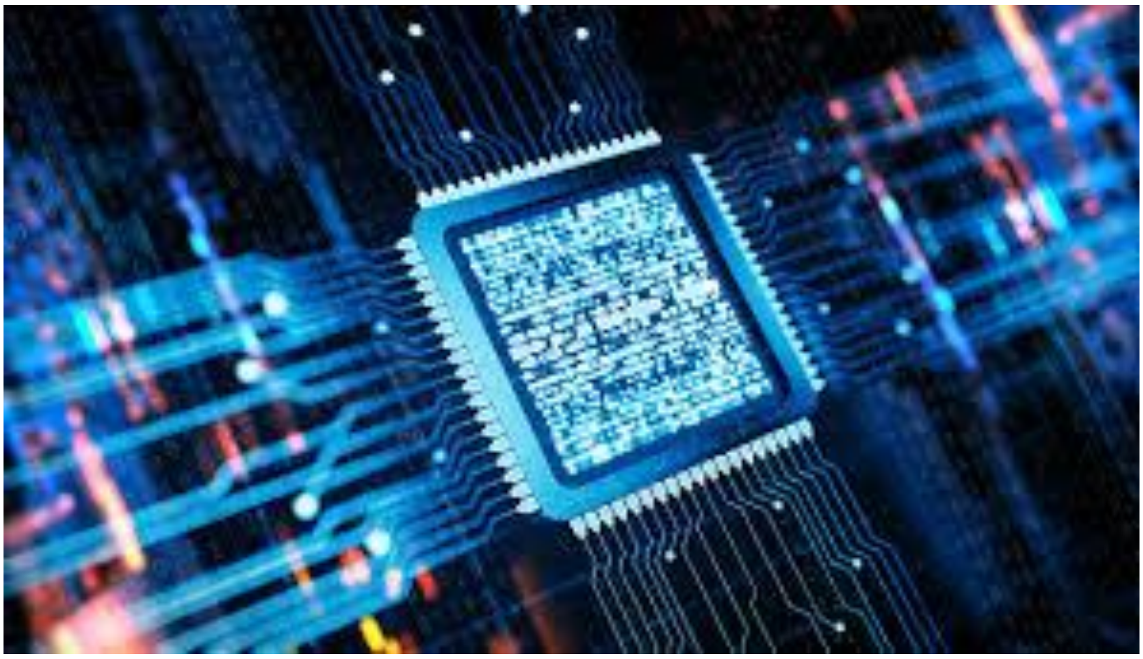


Operating System

LAB1 PROCESSES

SADE Paul & CAMARD Mathis | 17/09/2021

II Creating and Running a Process (1) - fork

1.

Read the man

2. *This question was writen with Ines MEUNIER who initially in Gr1 but change to Gr3*

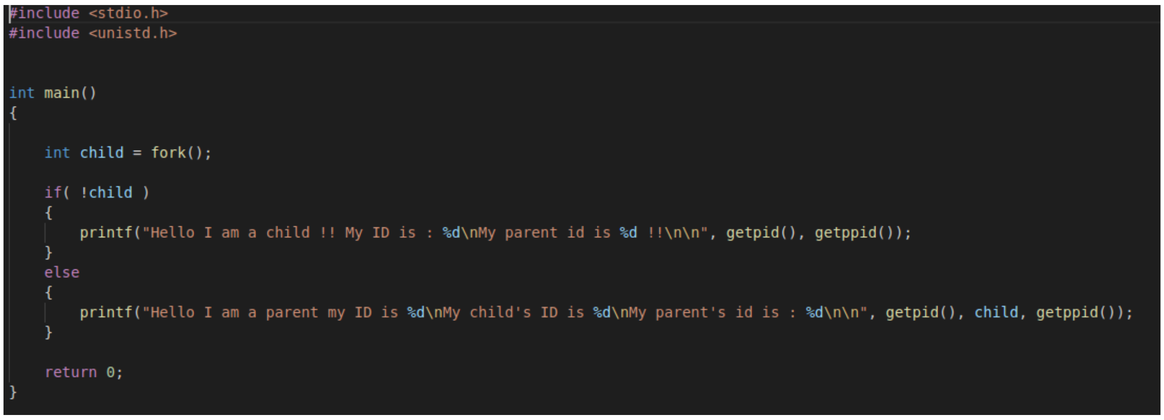
Once the fork is called the process is duplicate. The calling process is known as the parent and the duplicate one as the child. If the fork succeeds, the function returns the PID of the child to the parent and 0 to the child as a default value that corresponds to no PID of any process. If the fork fails the function return -1 to the parent and as long, there's no child and an errno (number of last error) is set to indicate an error.

Difference between parent and child process :

- The child has its own process ID (PID) which has a different ID than its parent
- The child-parent PID is different from the parent's parent PID (same thing for child's child PID and parent's child PID)
- The child has its own memory lock (space in RAM)
- Child time process, asynchronous I/O operations, timers and its resources are set to their default value and there are no pending signals
- Child semaphore adjustments are reset, to its parent's first version
- The child does not inherit process-associated record locks from its parent
- The child is an exact duplicate of its parents with its own PID and all values set to default except the file description locks and locks that it inherits from its parent.

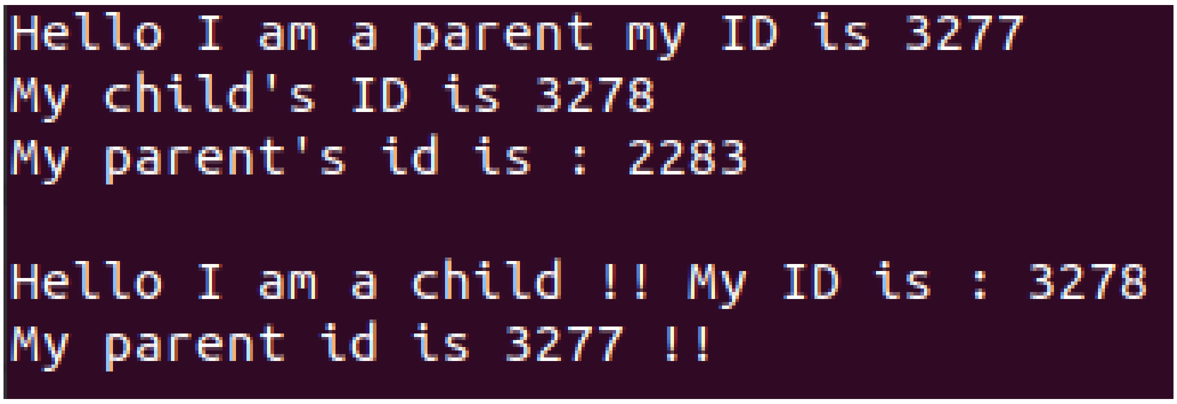
3.

Code :



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


int main()
{

    int child = fork();

    if( !child )
    {
        printf("Hello I am a child !! My ID is : %d\nMy parent id is %d !!\n\n", getpid(), getppid());
    }
    else
    {
        printf("Hello I am a parent my ID is %d\nMy child's ID is %d\nMy parent's id is : %d\n\n", getpid(), child, getppid());
    }

    return 0;
}
```

Result :

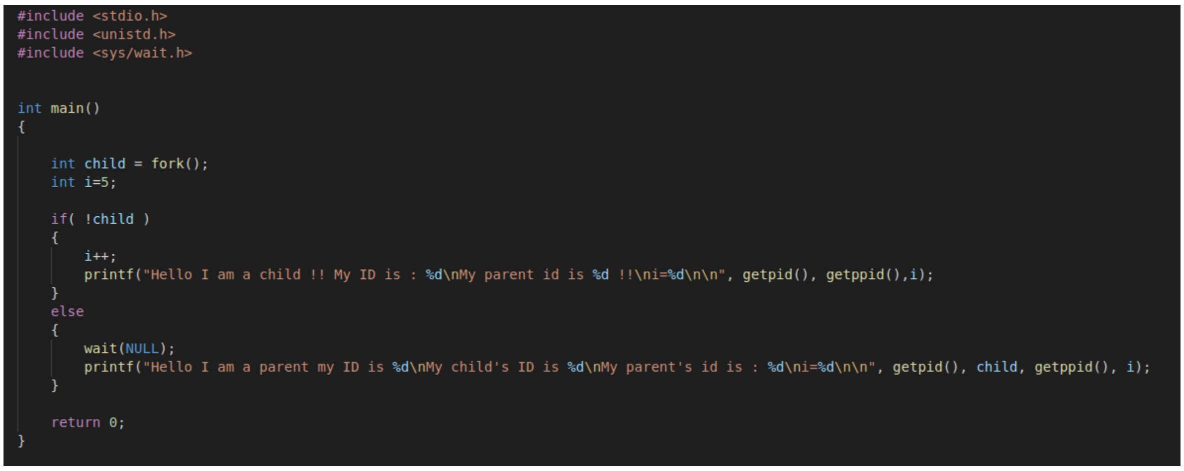


```
Hello I am a parent my ID is 3277
My child's ID is 3278
My parent's id is : 2283

Hello I am a child !! My ID is : 3278
My parent id is 3277 !!
```

In this code, we create a child process with the function fork(). After this we just implement a condition to print the child and parent ID.

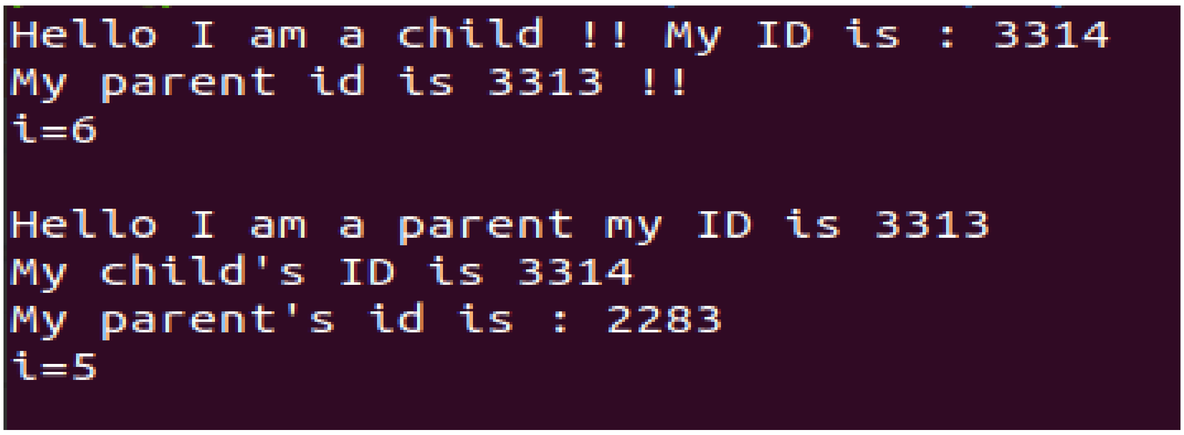
4.

Code :



```c
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>


int main()
{

    int child = fork();
    int i=5;

    if( !child )
    {
        i++;
        printf("Hello I am a child !! My ID is : %d\nMy parent id is %d !!\ni=%d\n\n", getpid(), getppid(),i);
    }
    else
    {
        wait(NULL);
        printf("Hello I am a parent my ID is %d\nMy child's ID is %d\nMy parent's id is : %d\ni=%d\n\n", getpid(), child, getppid(), i);
    }

    return 0;
}
```

Result :



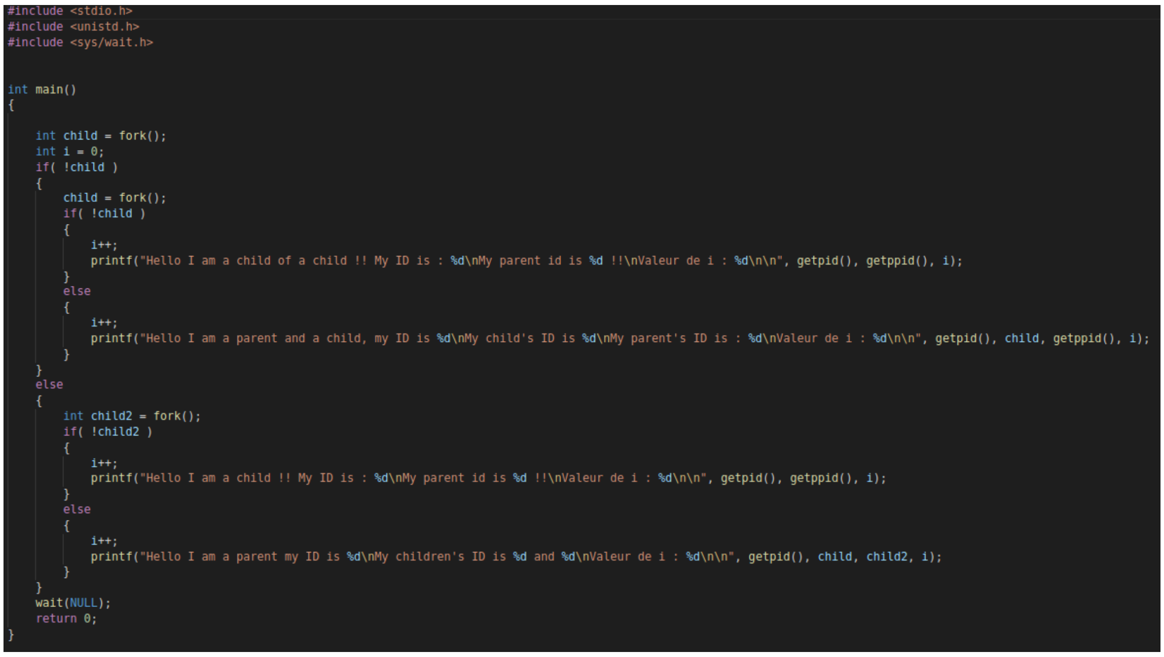
```
Hello I am a child !! My ID is : 3314
My parent id is 3313 !!
i=6

Hello I am a parent my ID is 3313
My child's ID is 3314
My parent's id is : 2283
i=5
```

We can see here that the data between the child and parent process are not shared. The incrementation of the i variable in the child process doesn’t affect the parent process.

5.

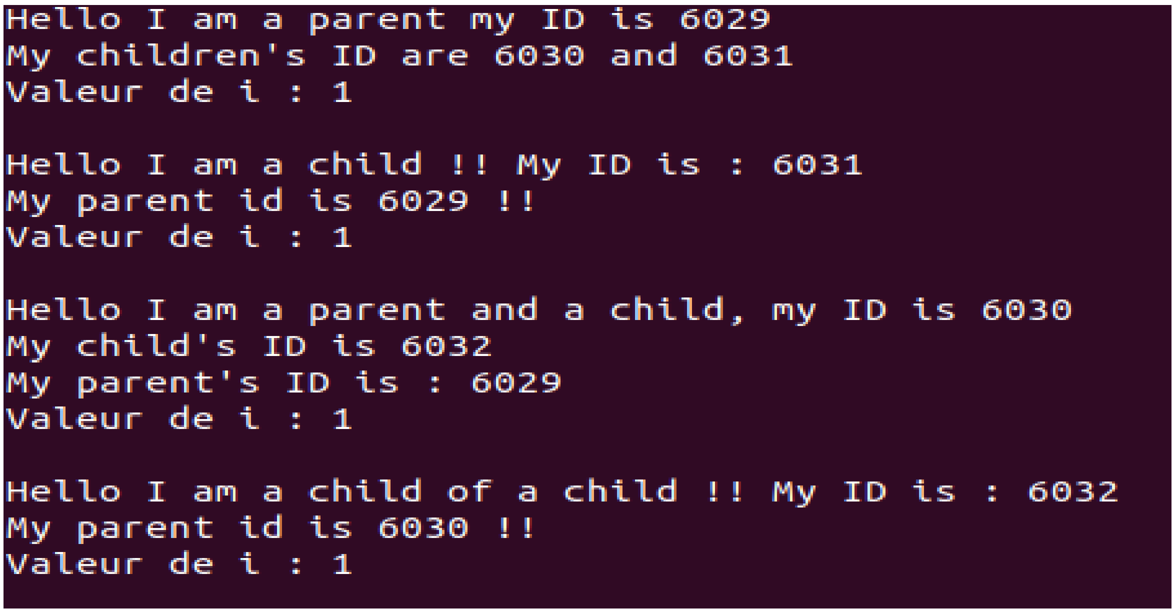
Yes it is possible to create more than one child process.

Code :



```c
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>


int main()
{

    int child = fork();
    int i = 0;
    if( !child )
    {
        child = fork();
        if( !child )
        {
            i++;
            printf("Hello I am a child of a child !! My ID is : %d\nMy parent id is %d !!\nValeur de i : %d\n\n", getpid(), getppid(), i);
        }
        else
        {
            i++;
            printf("Hello I am a parent and a child, my ID is %d\nMy child's ID is %d\nMy parent's ID is : %d\nValeur de i : %d\n\n", getpid(), child, getppid(), i);
        }
    }
    else
    {
        int child2 = fork();
        if( !child2 )
        {
            i++;
            printf("Hello I am a child !! My ID is : %d\nMy parent id is %d !!\nValeur de i : %d\n\n", getpid(), getppid(), i);
        }
        else
        {
            i++;
            printf("Hello I am a parent my ID is %d\nMy children's ID is %d and %d\nValeur de i : %d\n\n", getpid(), child, child2, i);
        }
    }
    wait(NULL);
    return 0;
}
```

Result :



```
Hello I am a parent my ID is 6029
My children's ID are 6030 and 6031
Valeur de i : 1

Hello I am a child !! My ID is : 6031
My parent id is 6029 !!
Valeur de i : 1

Hello I am a parent and a child, my ID is 6030
My child's ID is 6032
My parent's ID is : 6029
Valeur de i : 1

Hello I am a child of a child !! My ID is : 6032
My parent id is 6030 !!
Valeur de i : 1
```

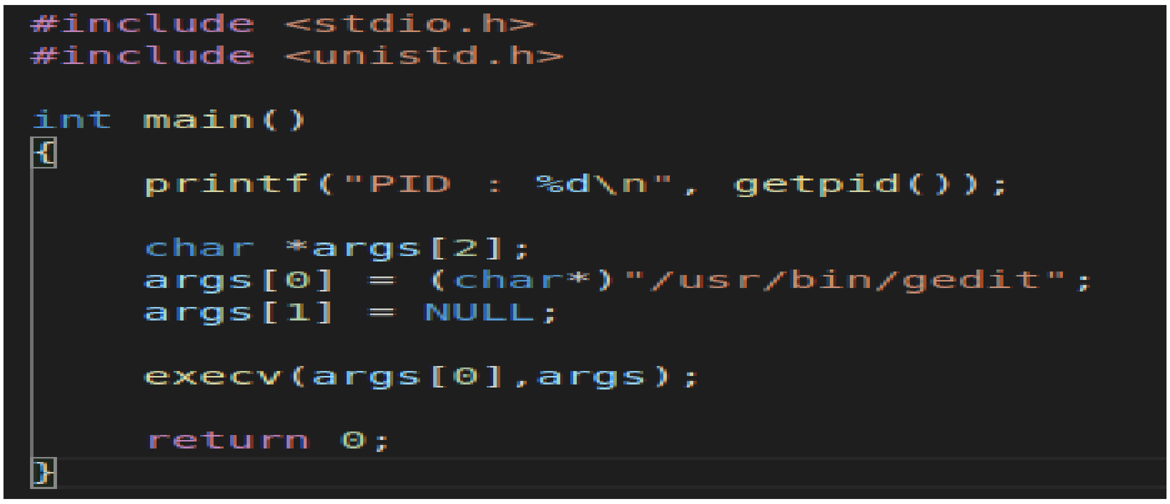
In this code we create two childs of the same parent. We can see that for the same parent ID we have to child with two différents ID.

III Creating and Running a Process

A. read man

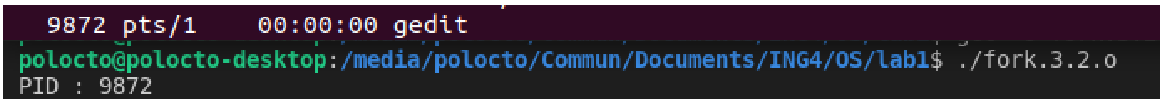
B.

Code :



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

int main()
{
    printf("PID : %d\n", getpid());

    char *args[2];
    args[0] = (char*)"/usr/bin/gedit";
    args[1] = NULL;

    execv(args[0],args);

    return 0;
}
```

Result :



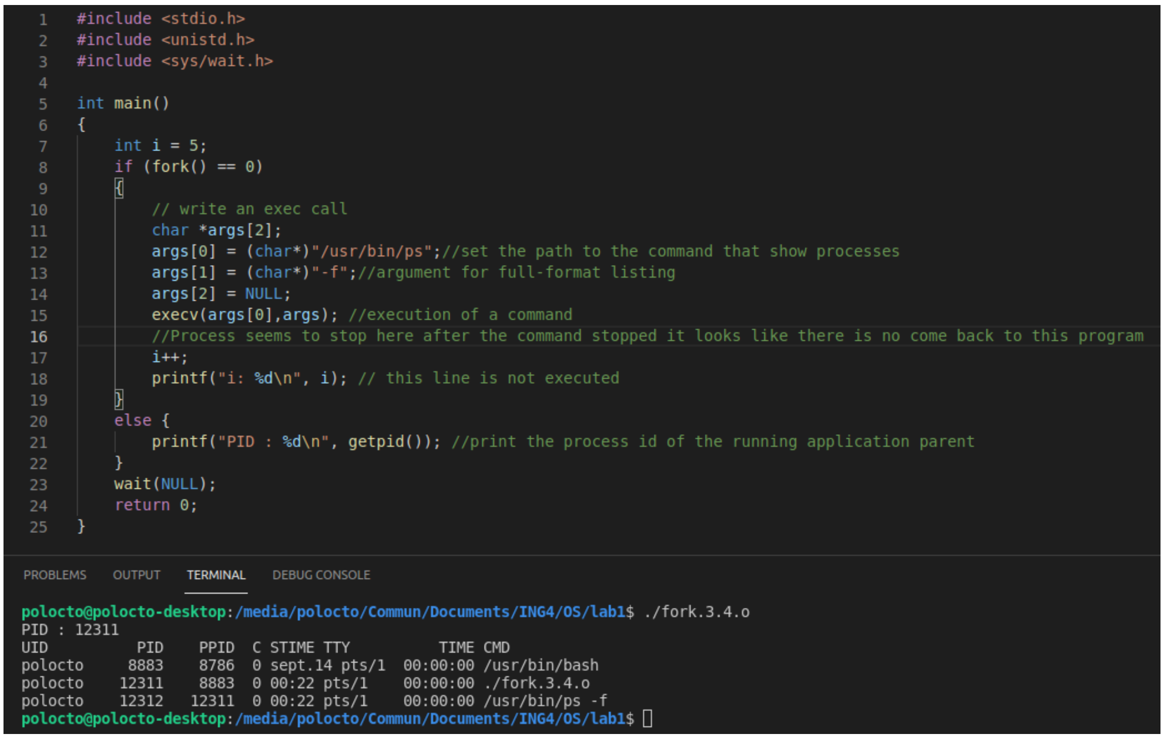
```
 9872 pts/1    00:00:00 gedit
polocto@polocto-desktop:/media/polocto/Commun/Documents/ING4/OS/lab1$ ./fork.3.2.o
PID : 9872
```

With the following command, we obtain a list of processes and their id: ps -e. The first number in the terminal represents the PID of the process. We can conclude that the process id of the running application is the same as the original one since they share the same PID.

C.

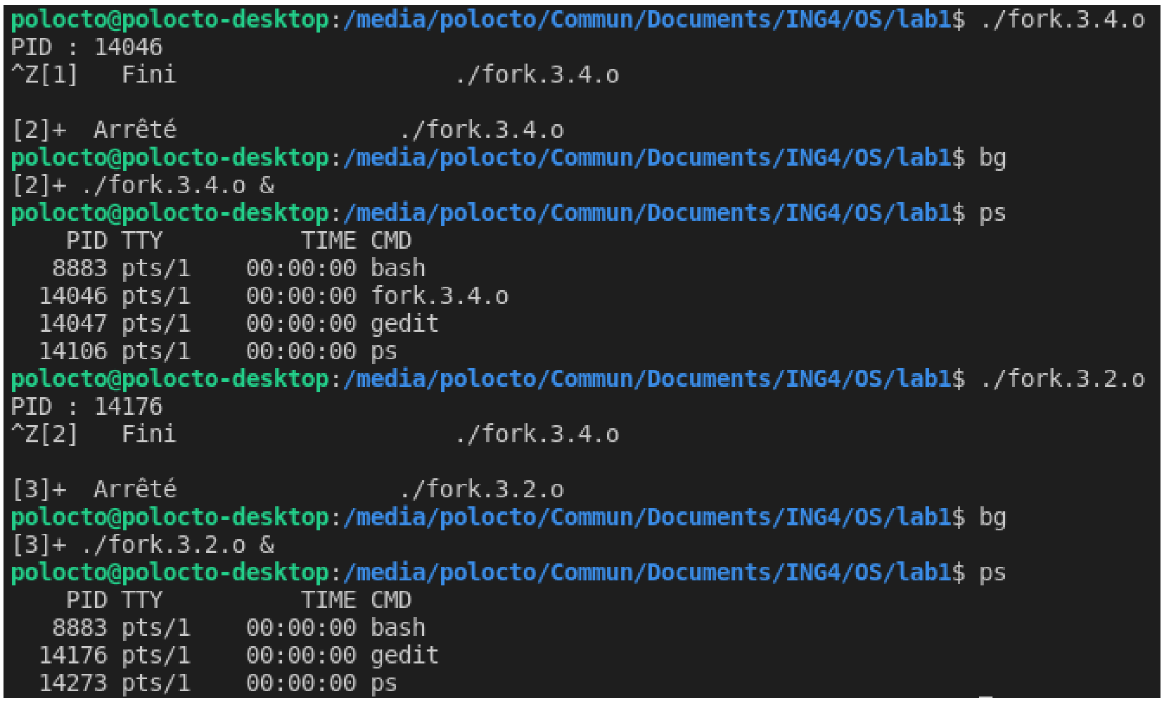
The data between parent and child processes can be shared using shared memory. This can work when we are using the fork function. If we run a program from another process, without using the function fork() all datas from the initial process will be erase. To prevent this we use the function fork() to be able to keep the datas between parent's process and his child.

D.



```
 1  #include <stdio.h>
 2  #include <unistd.h>
 3  #include <sys/wait.h>
 4
 5  int main()
 6  {
 7      int i = 5;
 8      if (fork() == 0)
 9      {
10          // write an exec call
11          char *args[2];
12          args[0] = (char*)"/usr/bin/ps";//set the path to the command that show processes
13          args[1] = (char*)"-f";//argument for full-format listing
14          args[2] = NULL;
15          execv(args[0],args); //execution of a command
16          //Process seems to stop here after the command stopped it looks like there is no come back to this program
17          i++;
18          printf("i: %d\n", i); // this line is not executed
19      }
20      else {
21          printf("PID : %d\n", getpid()); //print the process id of the running application parent
22      }
23      wait(NULL);
24      return 0;
25  }
```

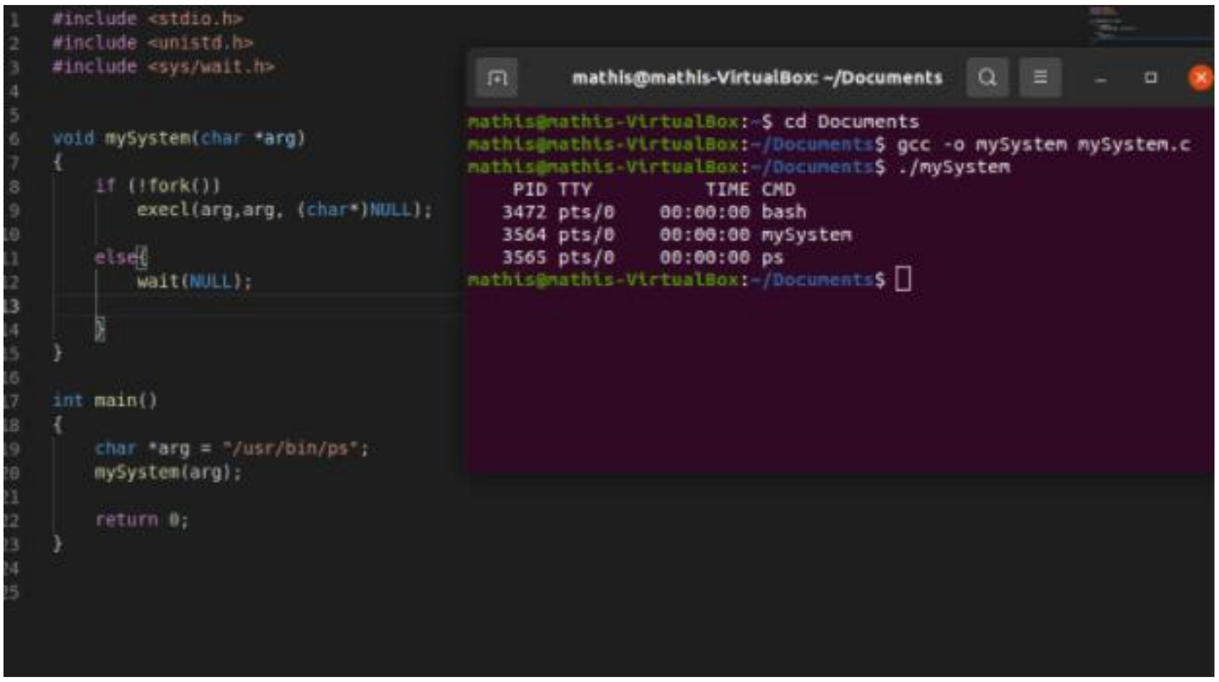
```
polocto@polocto-desktop:/media/polocto/Commun/Documents/ING4/OS/lab1$ ./fork.3.4.o
PID : 12311
UID          PID    PPID  C STIME TTY          TIME CMD
polocto     8883    8786  0 sept.14 pts/1  00:00:00 /usr/bin/bash
polocto    12311    8883  0 00:22 pts/1    00:00:00 ./fork.3.4.o
polocto    12312   12311  0 00:22 pts/1    00:00:00 /usr/bin/ps -f
polocto@polocto-desktop:/media/polocto/Commun/Documents/ING4/OS/lab1$ 
```



```
polocto@polocto-desktop:/media/polocto/Commun/Documents/ING4/OS/lab1$ ./fork.3.4.o
PID : 14046
^Z[1]   Fini                    ./fork.3.4.o

[2]+  Arrêté                  ./fork.3.4.o
polocto@polocto-desktop:/media/polocto/Commun/Documents/ING4/OS/lab1$ bg
[2]+ ./fork.3.4.o &
polocto@polocto-desktop:/media/polocto/Commun/Documents/ING4/OS/lab1$ ps
    PID TTY          TIME CMD
   8883 pts/1    00:00:00 bash
  14046 pts/1    00:00:00 fork.3.4.o
  14047 pts/1    00:00:00 gedit
  14106 pts/1    00:00:00 ps
polocto@polocto-desktop:/media/polocto/Commun/Documents/ING4/OS/lab1$ ./fork.3.2.o
PID : 14176
^Z[2]   Fini                    ./fork.3.4.o

[3]+  Arrêté                  ./fork.3.2.o
polocto@polocto-desktop:/media/polocto/Commun/Documents/ING4/OS/lab1$ bg
[3]+ ./fork.3.2.o &
polocto@polocto-desktop:/media/polocto/Commun/Documents/ING4/OS/lab1$ ps
    PID TTY          TIME CMD
   8883 pts/1    00:00:00 bash
  14176 pts/1    00:00:00 gedit
  14273 pts/1    00:00:00 ps
```

First w ewe call the fork function in order to create a child process. In the program of the child process we called an exec function using the path of the used command. When the command is executed, the process stop and doesn't continue to run the program after the exec function (l17-18).

IV Last question : lookup the system function in the manual & implement your own function (void mySystem(char *cmd)).



```c
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>


void mySystem(char *arg)
{
    if (!fork())
        execl(arg,arg, (char*)NULL);

    else{
        wait(NULL);

    }
}

int main()
{
    char *arg = "/usr/bin/ps";
    mySystem(arg);

    return 0;
}
```

```
mathis@mathis-VirtualBox:~$ cd Documents
mathis@mathis-VirtualBox:~/Documents$ gcc -o mySystem mySystem.c
mathis@mathis-VirtualBox:~/Documents$ ./mySystem
    PID TTY          TIME CMD
   3472 pts/0    00:00:00 bash
   3564 pts/0    00:00:00 mySystem
   3565 pts/0    00:00:00 ps
mathis@mathis-VirtualBox:~/Documents$
```