Université Internationale de Rabat

1ère Année Cycle ingénieur

Module :

Système d’exploitation et programmation système

Compte rendu :

TP/TD N°3 : Processus et parallélisme

Réaliser par : EL HANAFI Maha

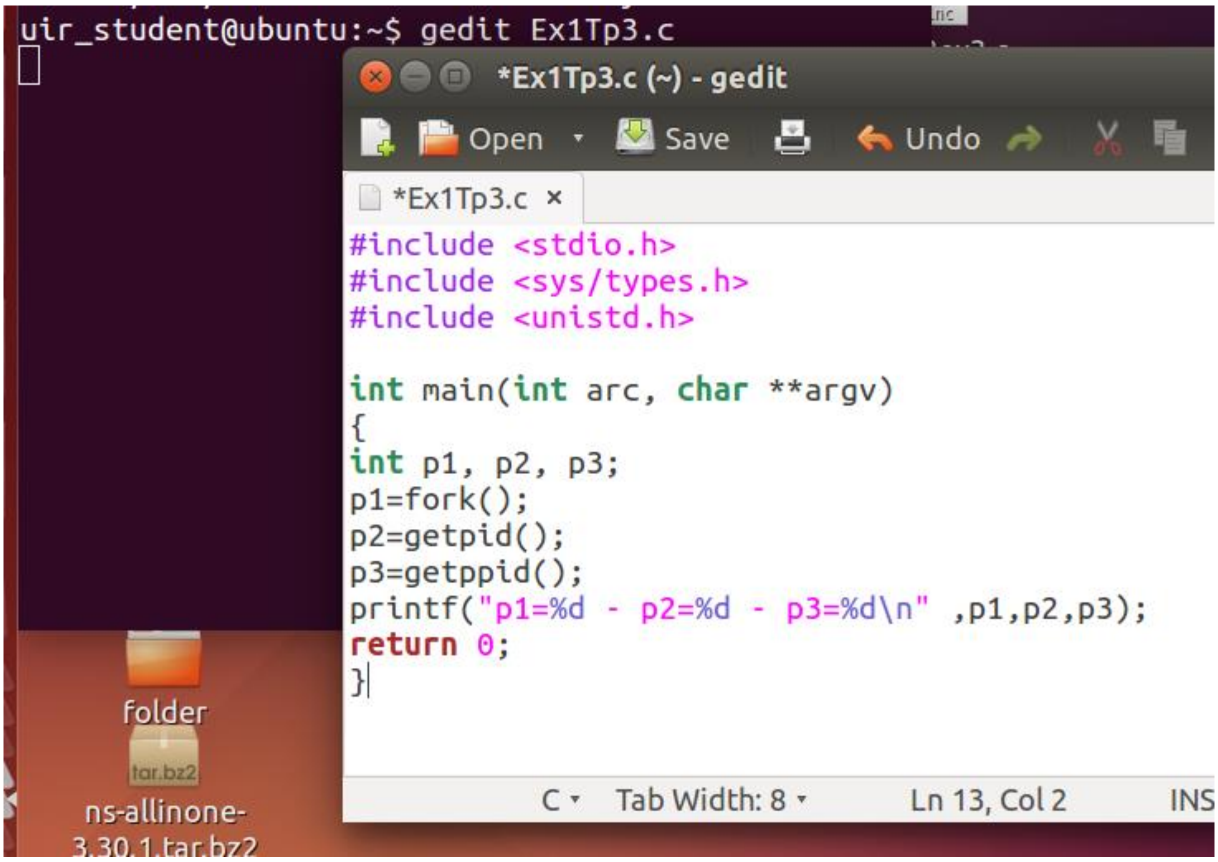
Encadré par : Abdelhak Kharbouch

Objectif :
Création et communication entre processus

Exercice1 : Création de processus

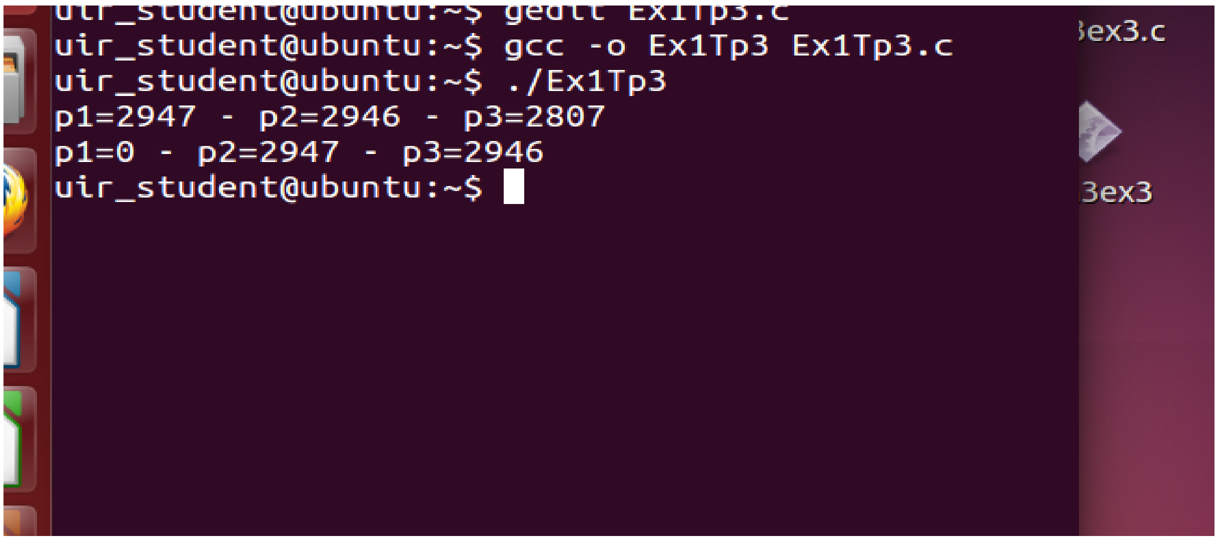
la fonction fork() retourne la valeur de type pid_t (int)
la fonction getpid() retourne le PID du processus appelant
la fonction getppid() retourne le PPID du processus appelant

1.1



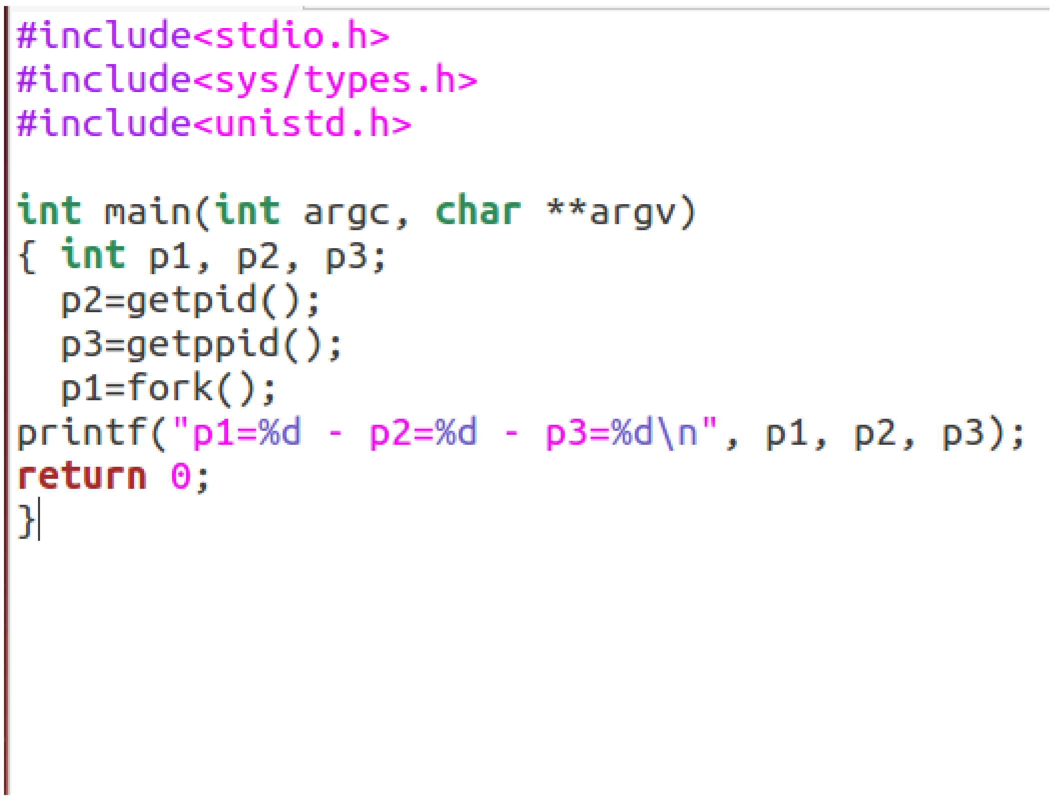
```
uir_student@ubuntu:~$ gedit Ex1Tp3.c
```

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

int main(int arc, char **argv)
{
int p1, p2, p3;
p1=fork();
p2=getpid();
p3=getppid();
printf("p1=%d - p2=%d - p3=%d\n" ,p1,p2,p3);
return 0;
}
```

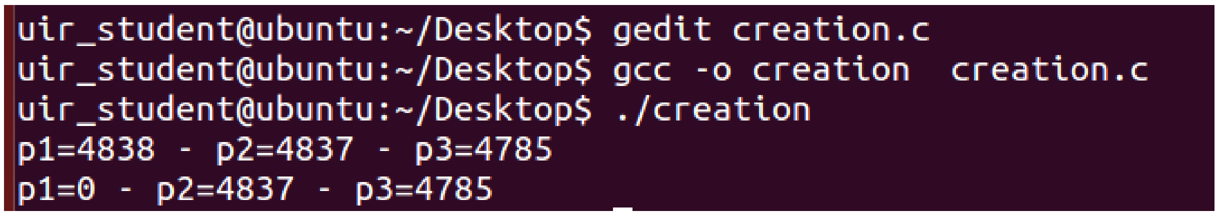


```
uir_student@ubuntu:~$ gedit Ex1Tp3.c
uir_student@ubuntu:~$ gcc -o Ex1Tp3 Ex1Tp3.c
uir_student@ubuntu:~$ ./Ex1Tp3
p1=2947 - p2=2946 - p3=2807
p1=0 - p2=2947 - p3=2946
uir_student@ubuntu:~$
```

1.2

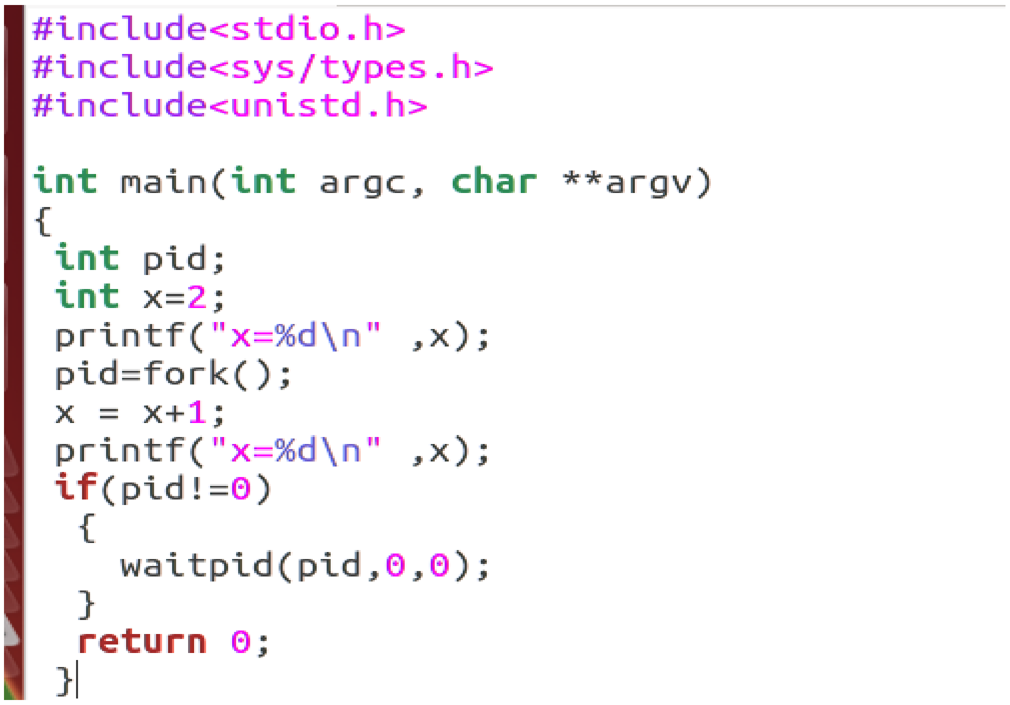


```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>

int main(int argc, char **argv)
{ int p1, p2, p3;
  p2=getpid();
  p3=getppid();
  p1=fork();
printf("p1=%d - p2=%d - p3=%d\n", p1, p2, p3);
return 0;
}
```

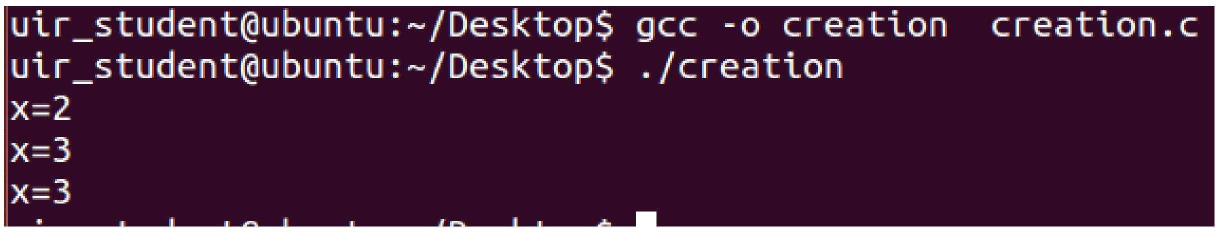


```
uir_student@ubuntu:~/Desktop$ gedit creation.c
uir_student@ubuntu:~/Desktop$ gcc -o creation  creation.c
uir_student@ubuntu:~/Desktop$ ./creation
p1=4838 - p2=4837 - p3=4785
p1=0 - p2=4837 - p3=4785
```

1.3



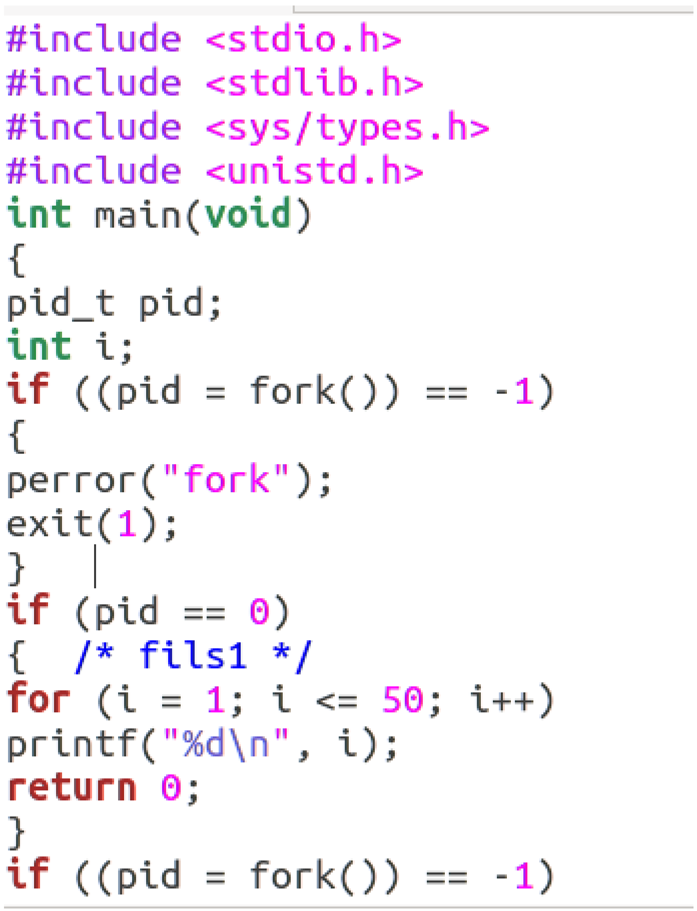
```c
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>

int main(int argc, char **argv)
{
 int pid;
 int x=2;
 printf("x=%d\n" ,x);
 pid=fork();
 x = x+1;
 printf("x=%d\n" ,x);
 if(pid!=0)
  {
    waitpid(pid,0,0);
  }
  return 0;
 }
```



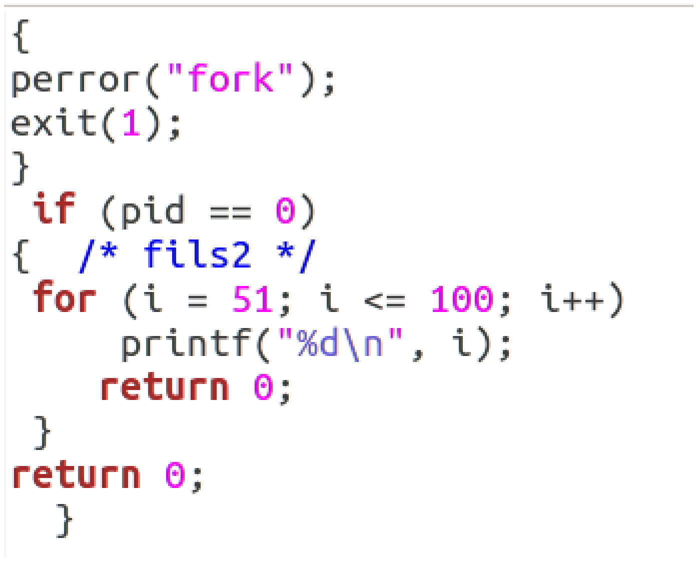
```
uir_student@ubuntu:~/Desktop$ gcc -o creation  creation.c
uir_student@ubuntu:~/Desktop$ ./creation
x=2
x=3
x=3
```

Exercice 2 : (TD n°3)

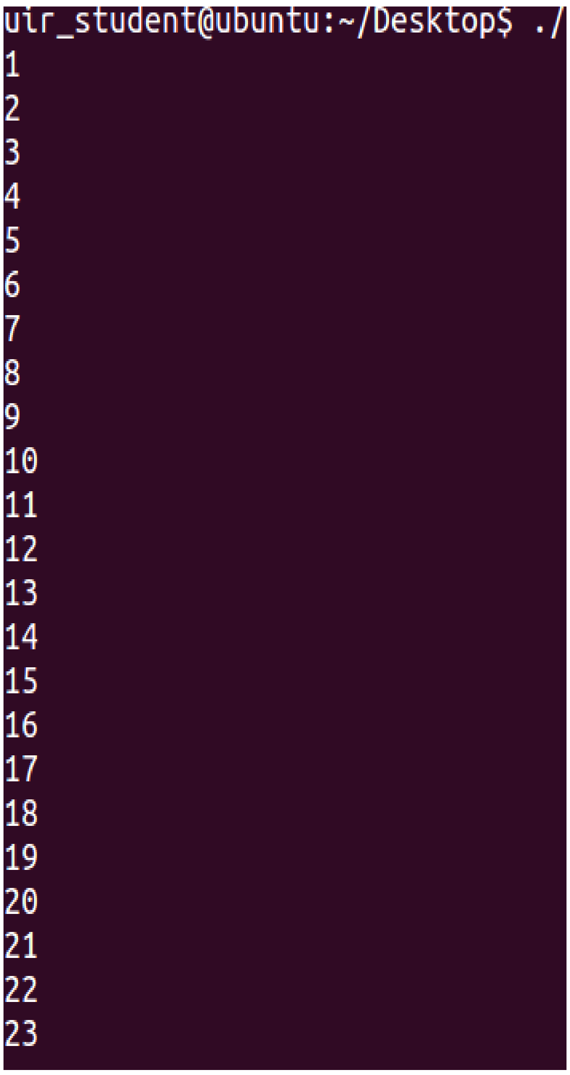
Exercice1 : Création de processus



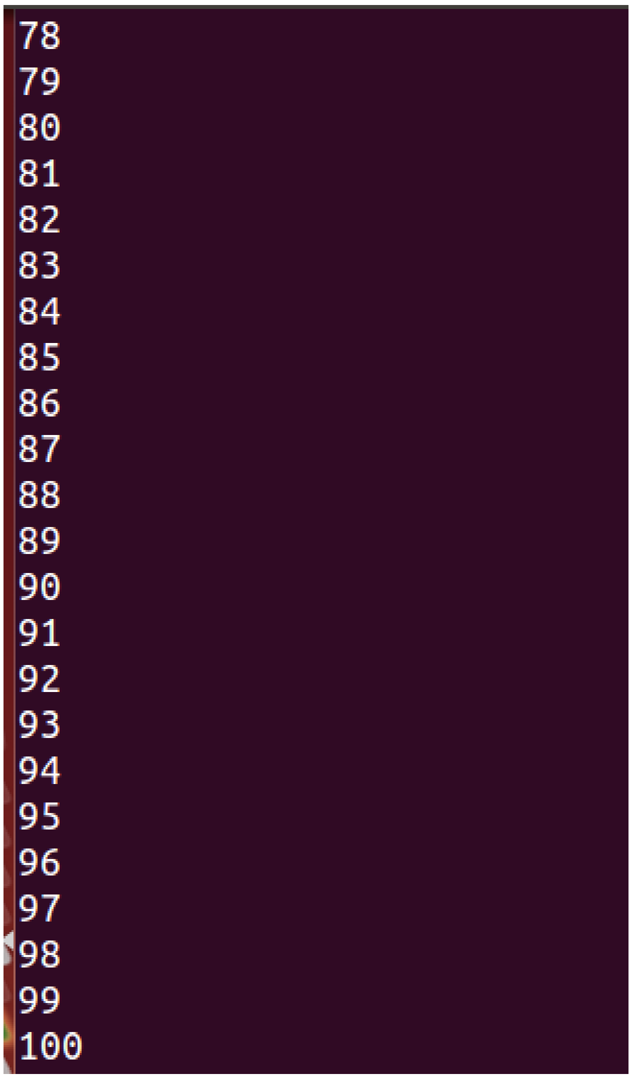
```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
int main(void)
{
pid_t pid;
int i;
if ((pid = fork()) == -1)
{
perror("fork");
exit(1);
}
if (pid == 0)
{  /* fils1 */
for (i = 1; i <= 50; i++)
printf("%d\n", i);
return 0;
}
if ((pid = fork()) == -1)
```



```c
{
perror("fork");
exit(1);
}
 if (pid == 0)
{  /* fils2 */
 for (i = 51; i <= 100; i++)
     printf("%d\n", i);
    return 0;
 }
return 0;
  }
```



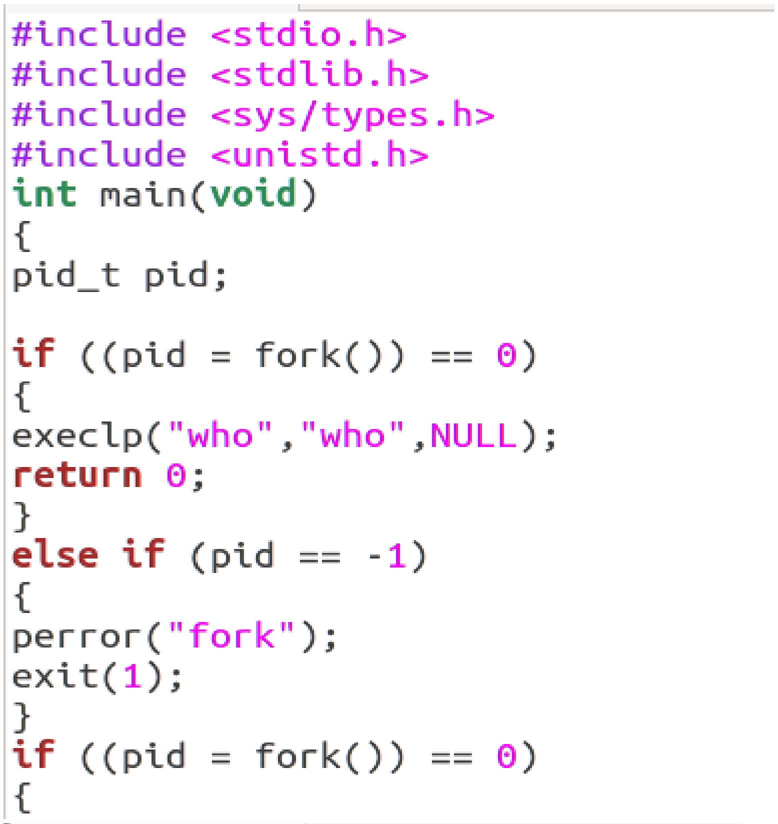
```
uir_student@ubuntu:~/Desktop$ ./
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
```



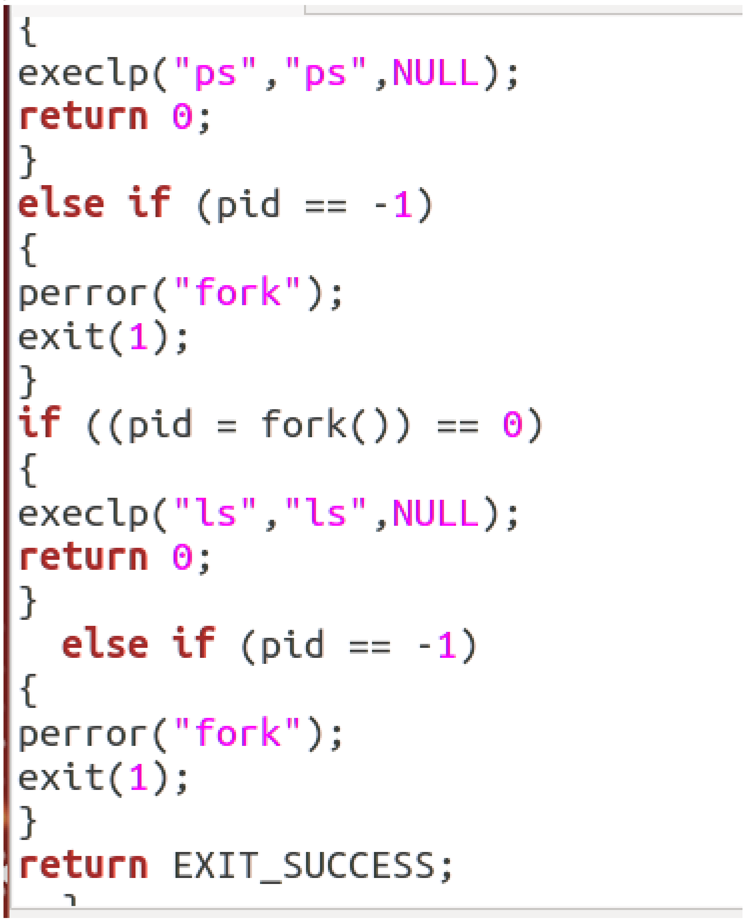
```
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

Exercice 2 : Simultanéité vs. Séquentialité

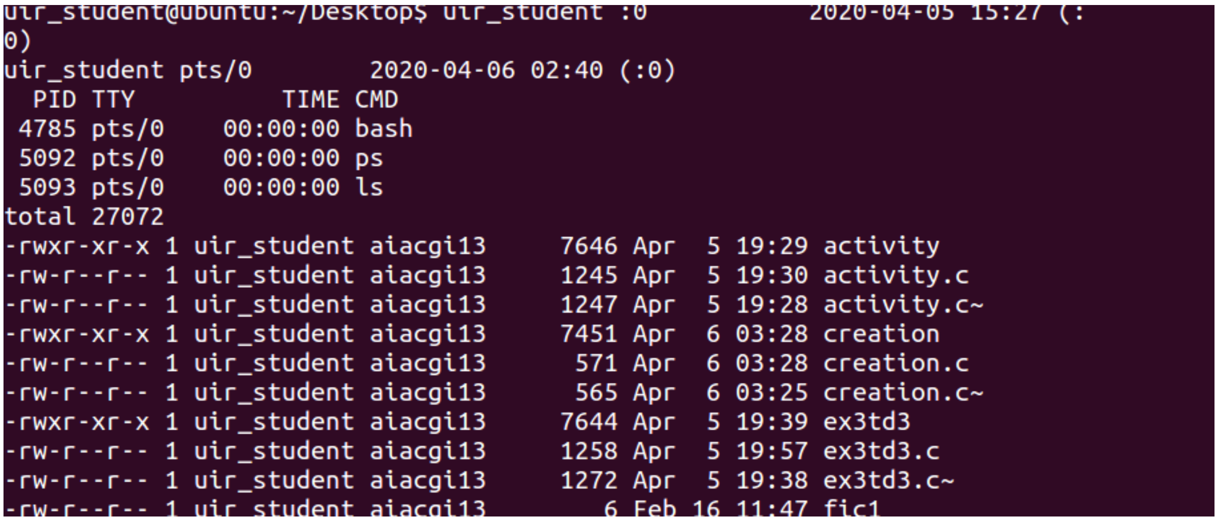
a. who & ps & ls-l



```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
int main(void)
{
pid_t pid;

if ((pid = fork()) == 0)
{
execlp("who","who",NULL);
return 0;
}
else if (pid == -1)
{
perror("fork");
exit(1);
}
if ((pid = fork()) == 0)
{
```

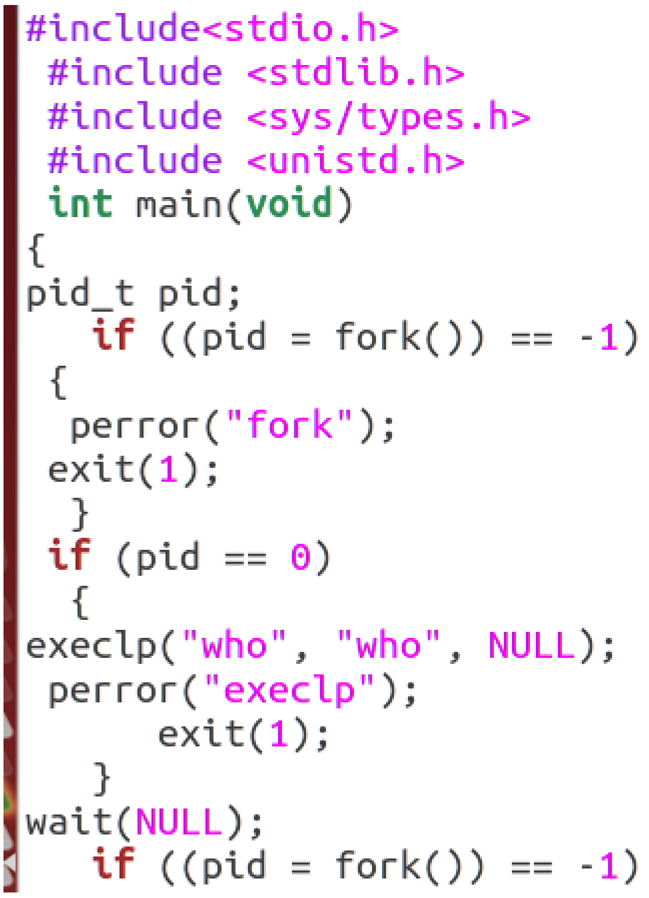


```
{
execlp("ps","ps",NULL);
return 0;
}
else if (pid == -1)
{
perror("fork");
exit(1);
}
if ((pid = fork()) == 0)
{
execlp("ls","ls",NULL);
return 0;
}
   else if (pid == -1)
{
perror("fork");
exit(1);
}
return EXIT_SUCCESS;
```

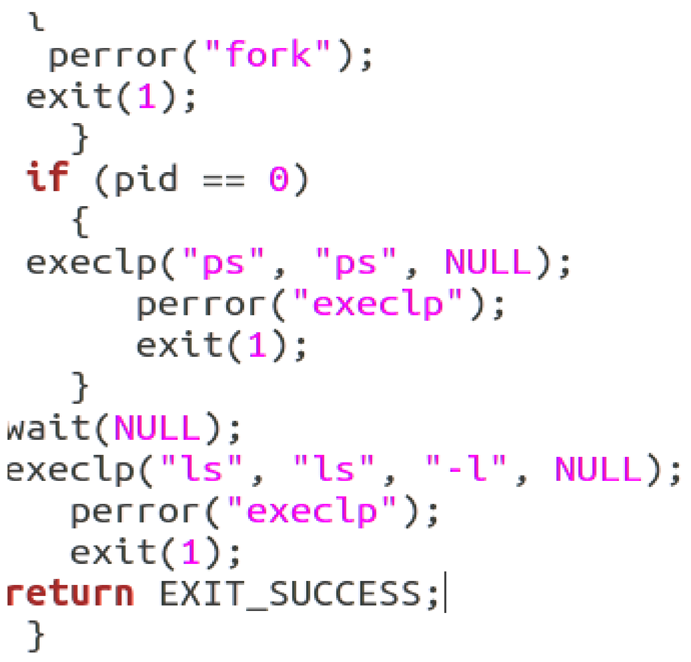


```
uir_student@ubuntu:~/Desktop$ uir_student :0          2020-04-05 15:27 (:
0)
uir_student pts/0        2020-04-06 02:40 (:0)
  PID TTY          TIME CMD
 4785 pts/0    00:00:00 bash
 5092 pts/0    00:00:00 ps
 5093 pts/0    00:00:00 ls
total 27072
-rwxr-xr-x 1 uir_student aiacgi13     7646 Apr  5 19:29 activity
-rw-r--r-- 1 uir_student aiacgi13     1245 Apr  5 19:30 activity.c
-rw-r--r-- 1 uir_student aiacgi13     1247 Apr  5 19:28 activity.c~
-rwxr-xr-x 1 uir_student aiacgi13     7451 Apr  6 03:28 creation
-rw-r--r-- 1 uir_student aiacgi13      571 Apr  6 03:28 creation.c
-rw-r--r-- 1 uir_student aiacgi13      565 Apr  6 03:25 creation.c~
-rwxr-xr-x 1 uir_student aiacgi13     7644 Apr  5 19:39 ex3td3
-rw-r--r-- 1 uir_student aiacgi13     1258 Apr  5 19:57 ex3td3.c
-rw-r--r-- 1 uir_student aiacgi13     1272 Apr  5 19:38 ex3td3.c~
-rw-r--r-- 1 uir_student aiacgi13        6 Feb 16 11:47 fic1
```

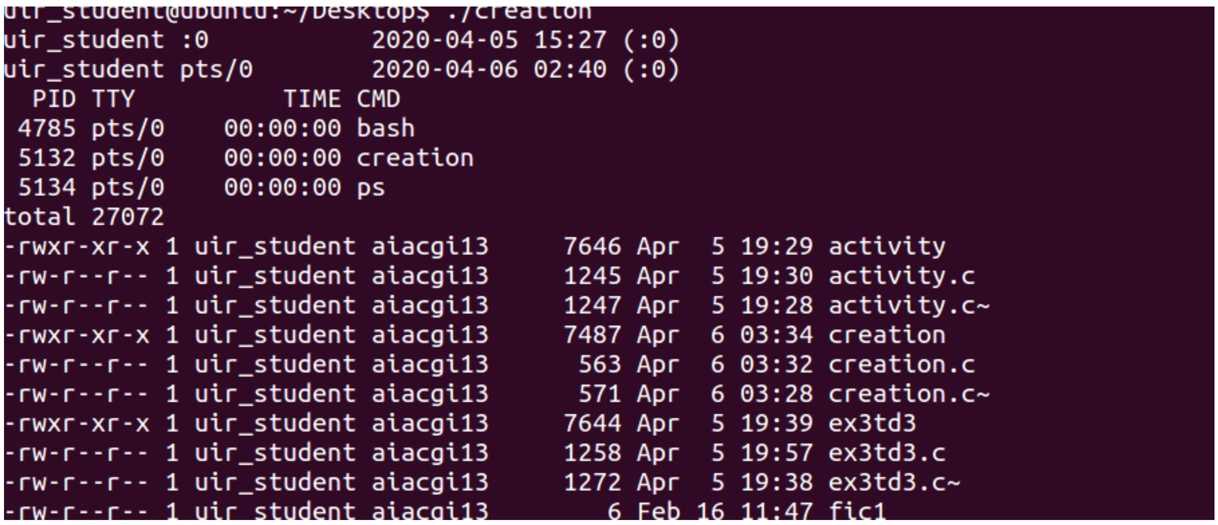
b. who ; ps ; ls -l



```
#include<stdio.h>
 #include <stdlib.h>
 #include <sys/types.h>
 #include <unistd.h>
 int main(void)
{
pid_t pid;
    if ((pid = fork()) == -1)
 {
  perror("fork");
 exit(1);
  }
 if (pid == 0)
  {
execlp("who", "who", NULL);
 perror("execlp");
      exit(1);
   }
wait(NULL);
    if ((pid = fork()) == -1)
```



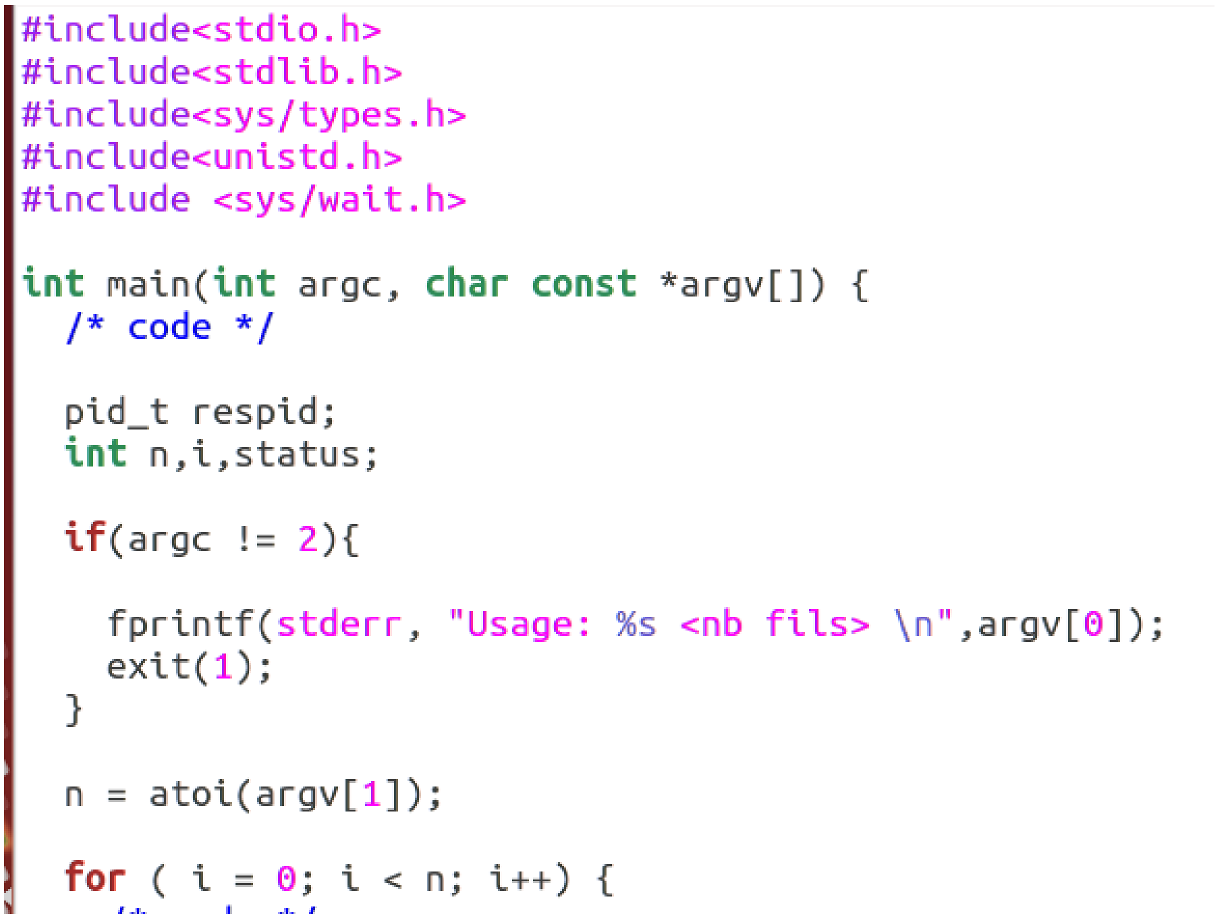
```
 {
  perror("fork");
 exit(1);
   }
 if (pid == 0)
   {
 execlp("ps", "ps", NULL);
        perror("execlp");
        exit(1);
   }
wait(NULL);
execlp("ls", "ls", "-l", NULL);
   perror("execlp");
   exit(1);
return EXIT_SUCCESS;
 }
```



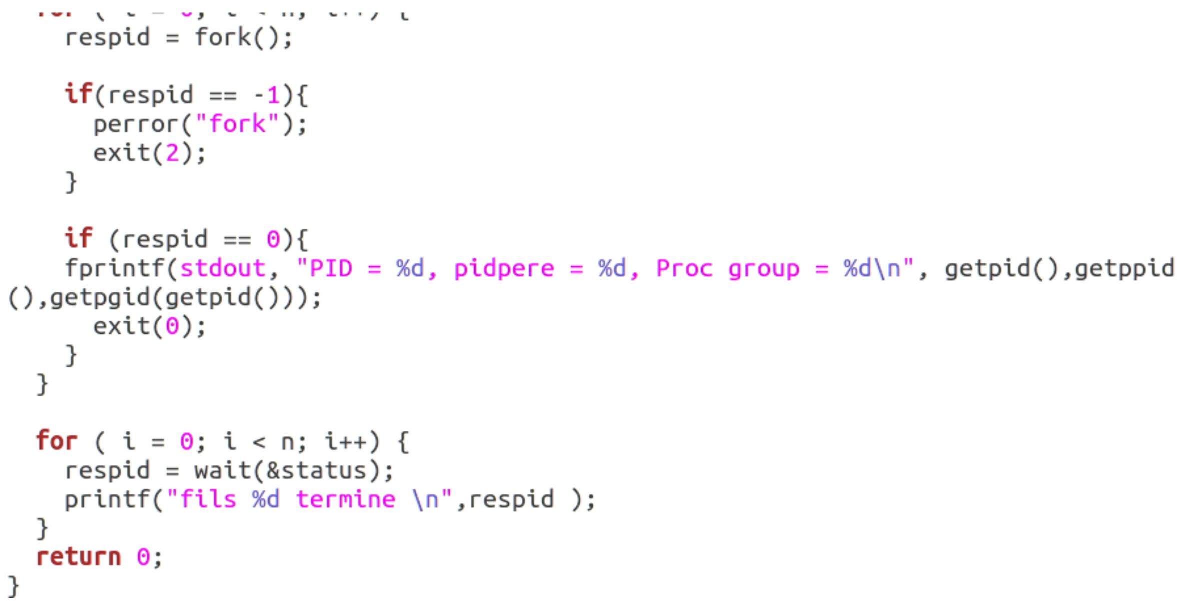
```
uir_student@ubuntu:~/Desktop$ ./creation
uir_student :0           2020-04-05 15:27 (:0)
uir_student pts/0        2020-04-06 02:40 (:0)
  PID TTY          TIME CMD
 4785 pts/0    00:00:00 bash
 5132 pts/0    00:00:00 creation
 5134 pts/0    00:00:00 ps
total 27072
-rwxr-xr-x 1 uir_student aiacgi13     7646 Apr  5 19:29 activity
-rw-r--r-- 1 uir_student aiacgi13     1245 Apr  5 19:30 activity.c
-rw-r--r-- 1 uir_student aiacgi13     1247 Apr  5 19:28 activity.c~
-rwxr-xr-x 1 uir_student aiacgi13     7487 Apr  6 03:34 creation
-rw-r--r-- 1 uir_student aiacgi13      563 Apr  6 03:32 creation.c
-rw-r--r-- 1 uir_student aiacgi13      571 Apr  6 03:28 creation.c~
-rwxr-xr-x 1 uir_student aiacgi13     7644 Apr  5 19:39 ex3td3
-rw-r--r-- 1 uir_student aiacgi13     1258 Apr  5 19:57 ex3td3.c
-rw-r--r-- 1 uir_student aiacgi13     1272 Apr  5 19:38 ex3td3.c~
-rw-r--r-- 1 uir_student aiacgi13        6 Feb 16 11:47 fic1
```

Exercice 3 : Synchronisation de processus

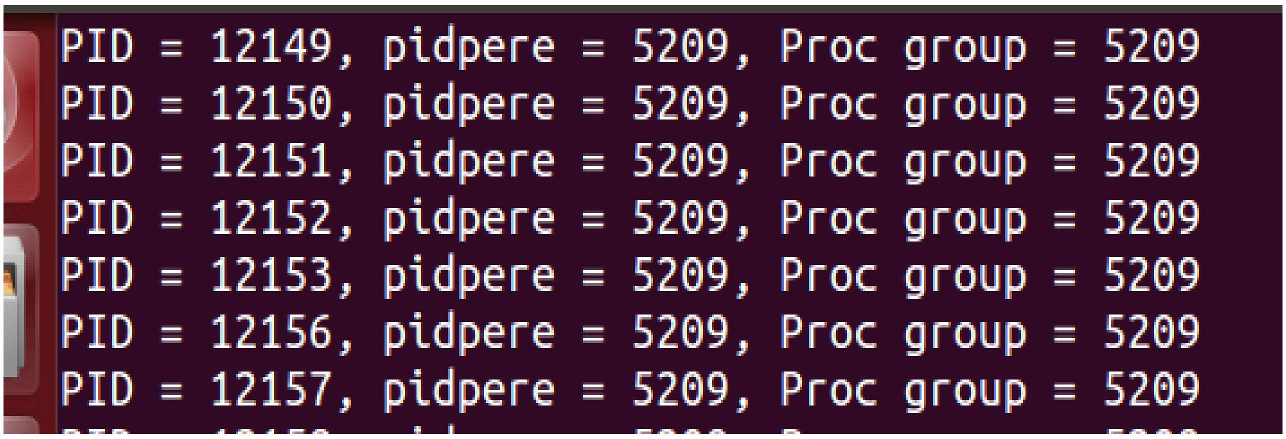
a.



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

int main(int argc, char const *argv[]) {
  /* code */

  pid_t respid;
  int n,i,status;

  if(argc != 2){

    fprintf(stderr, "Usage: %s <nb fils> \n",argv[0]);
    exit(1);
  }

  n = atoi(argv[1]);

  for ( i = 0; i < n; i++) {
```

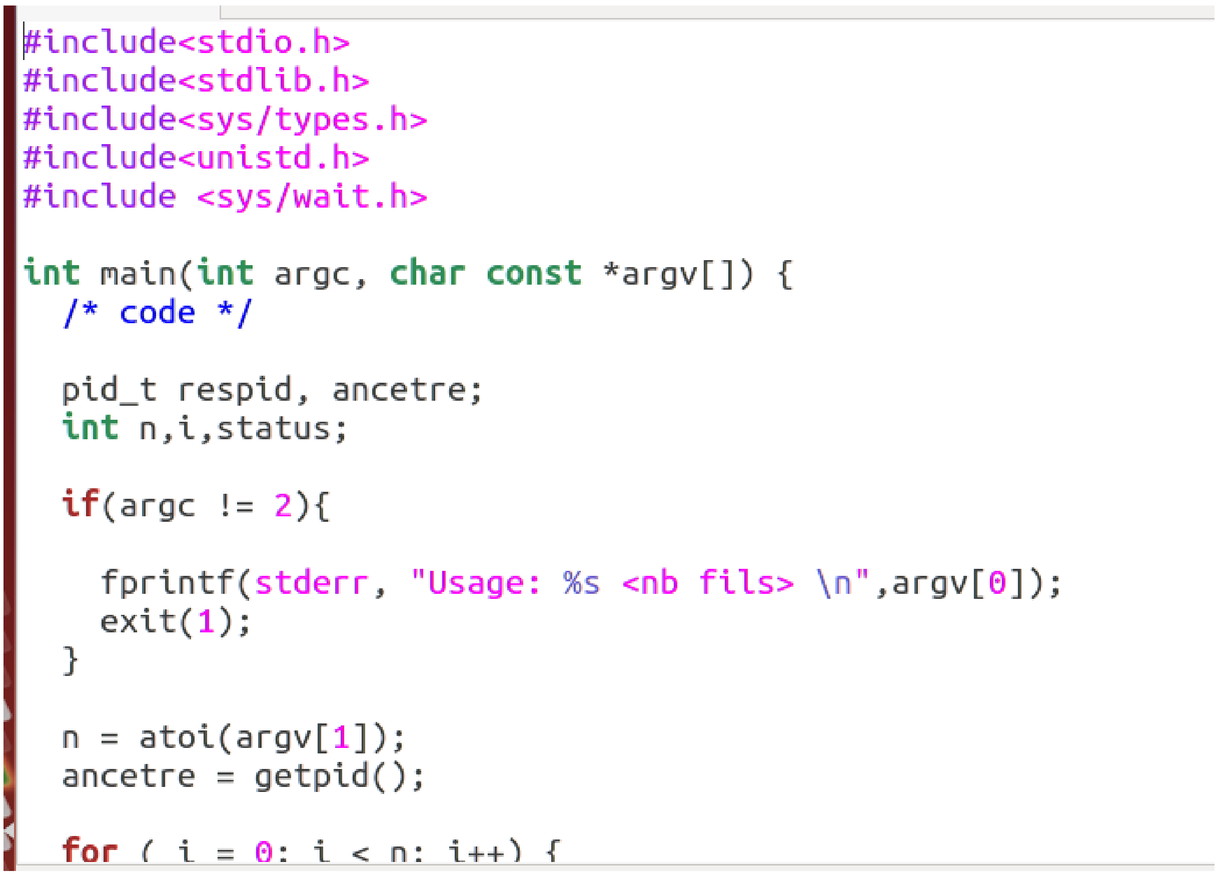


```c
    respid = fork();

    if(respid == -1){
      perror("fork");
      exit(2);
    }

    if (respid == 0){
    fprintf(stdout, "PID = %d, pidpere = %d, Proc group = %d\n", getpid(),getppid
(),getpgid(getpid()));
      exit(0);
    }
  }

  for ( i = 0; i < n; i++) {
    respid = wait(&status);
    printf("fils %d termine \n",respid );
  }
  return 0;
}
```

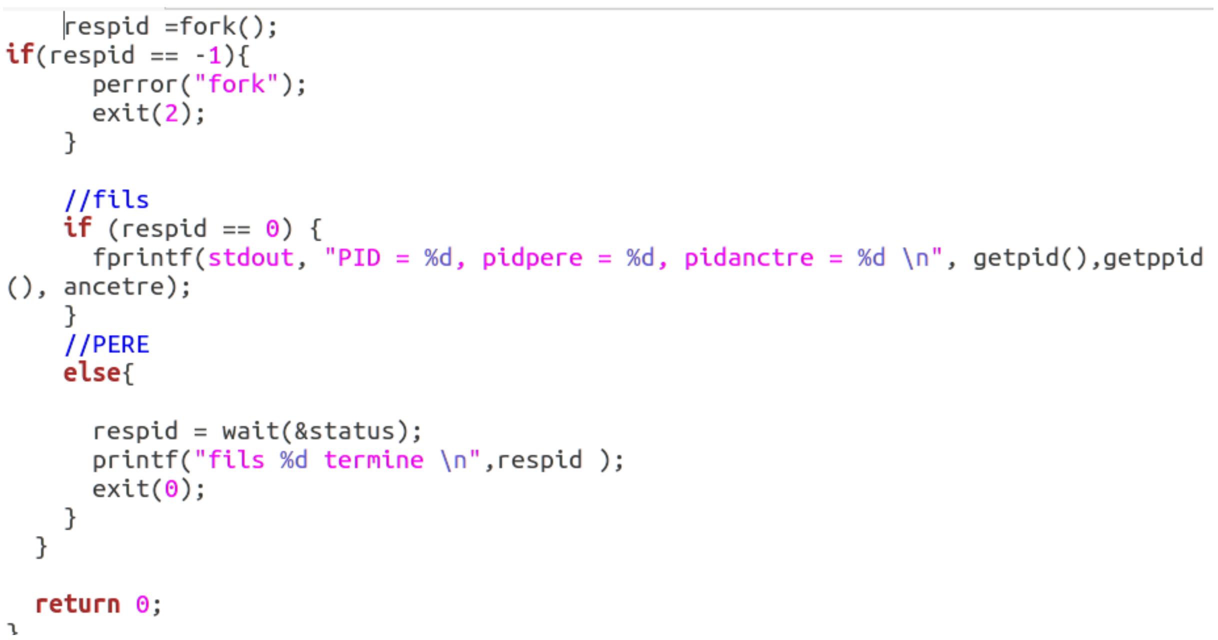


```
PID = 12149, pidpere = 5209, Proc group = 5209
PID = 12150, pidpere = 5209, Proc group = 5209
PID = 12151, pidpere = 5209, Proc group = 5209
PID = 12152, pidpere = 5209, Proc group = 5209
PID = 12153, pidpere = 5209, Proc group = 5209
PID = 12156, pidpere = 5209, Proc group = 5209
PID = 12157, pidpere = 5209, Proc group = 5209
```

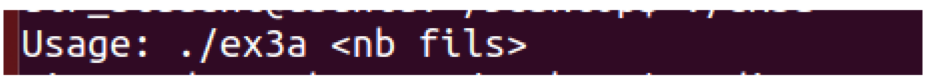
b.



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

int main(int argc, char const *argv[]) {
  /* code */

  pid_t respid, ancetre;
  int n,i,status;

  if(argc != 2){

    fprintf(stderr, "Usage: %s <nb fils> \n",argv[0]);
    exit(1);
  }

  n = atoi(argv[1]);
  ancetre = getpid();

  for ( i = 0; i < n; i++) {
```

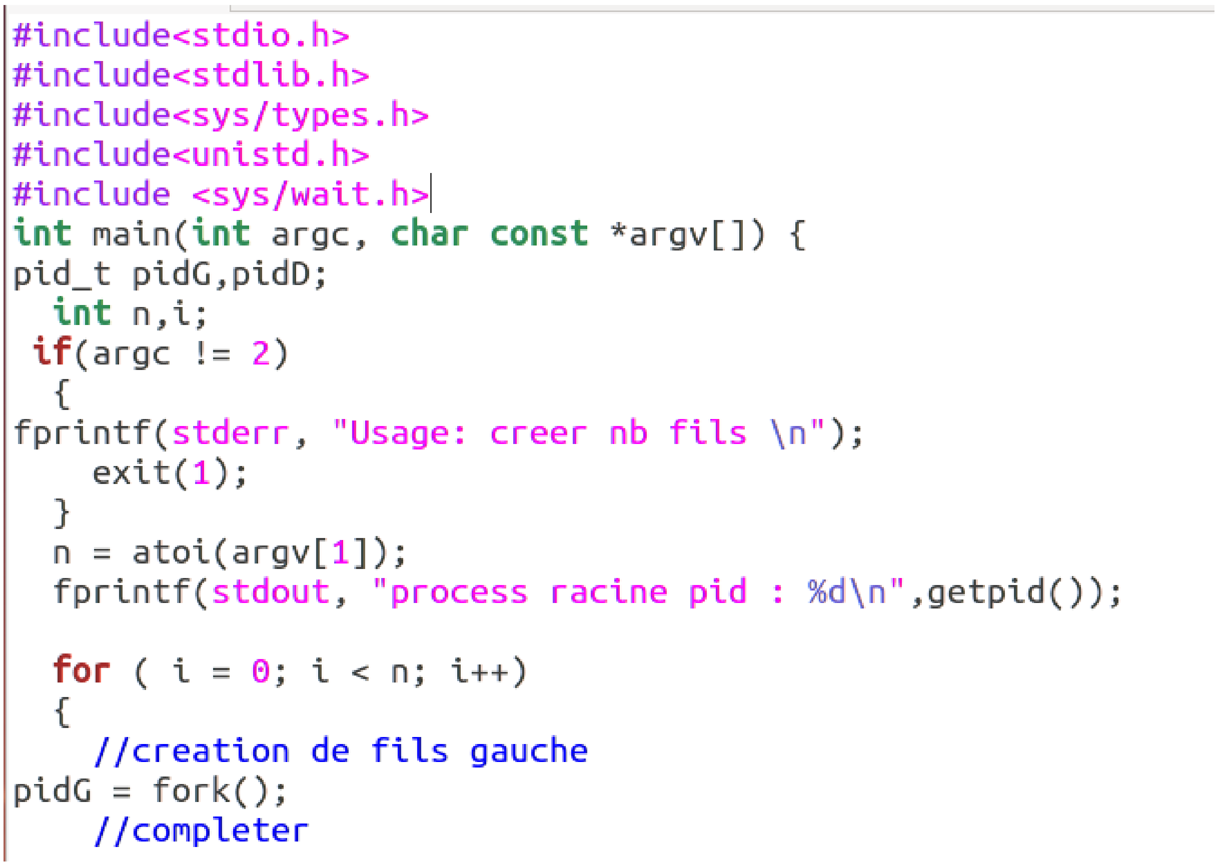


```
    respid =fork();
if(respid == -1){
      perror("fork");
      exit(2);
    }

    //fils
    if (respid == 0) {
      fprintf(stdout, "PID = %d, pidpere = %d, pidanctre = %d \n", getpid(),getppid
(), ancetre);
    }
    //PERE
    else{

      respid = wait(&status);
      printf("fils %d termine \n",respid );
      exit(0);
    }
  }

  return 0;
}
```

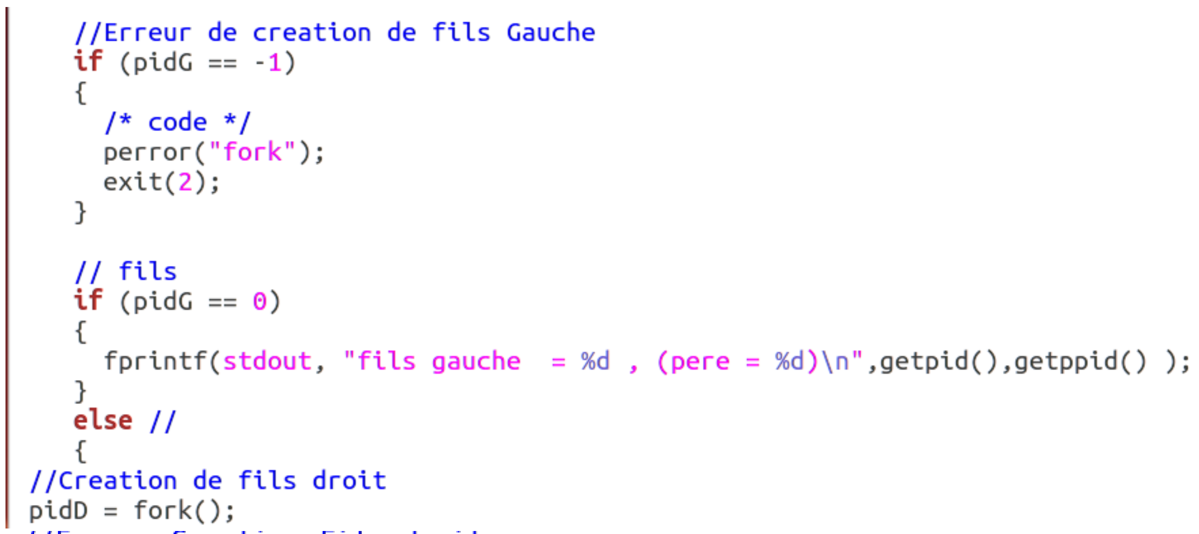


```
Usage: ./ex3a <nb fils>
```

c.

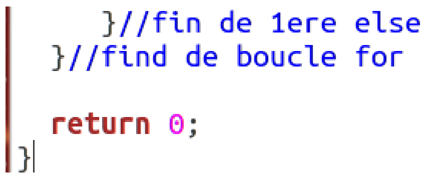


```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<unistd.h>
#include <sys/wait.h>
int main(int argc, char const *argv[]) {
pid_t pidG,pidD;
  int n,i;
 if(argc != 2)
  {
fprintf(stderr, "Usage: creer nb fils \n");
    exit(1);
  }
  n = atoi(argv[1]);
  fprintf(stdout, "process racine pid : %d\n",getpid());

  for ( i = 0; i < n; i++)
  {
    //creation de fils gauche
pidG = fork();
    //completer
```

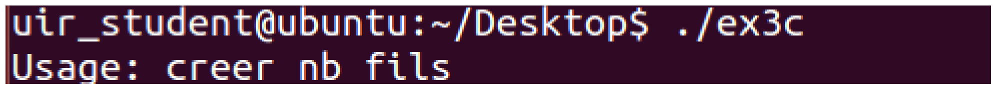


```
    //Erreur de creation de fils Gauche
    if (pidG == -1)
    {
      /* code */
      perror("fork");
      exit(2);
    }

    // fils
    if (pidG == 0)
    {
      fprintf(stdout, "fils gauche  = %d , (pere = %d)\n",getpid(),getppid() );
    }
    else //
    {
 //Creation de fils droit
 pidD = fork();
```

```
 //Erreur Creation Fils droit
 if (pidD == -1)
 {
perror("fork");
exit(2);
 }
 if (pidD == 0)
 {
fprintf(stdout, "fils droit   = %d , (pere = %d)\n", getpid(),getppid());
 }
 //pere
 else
 {
//completer
waitpid(pidG, NULL, 0);
waitpid(pidD, NULL, 0);
exit(0);//parent exits when his children are done
//Code executer par le processus père

 }
```

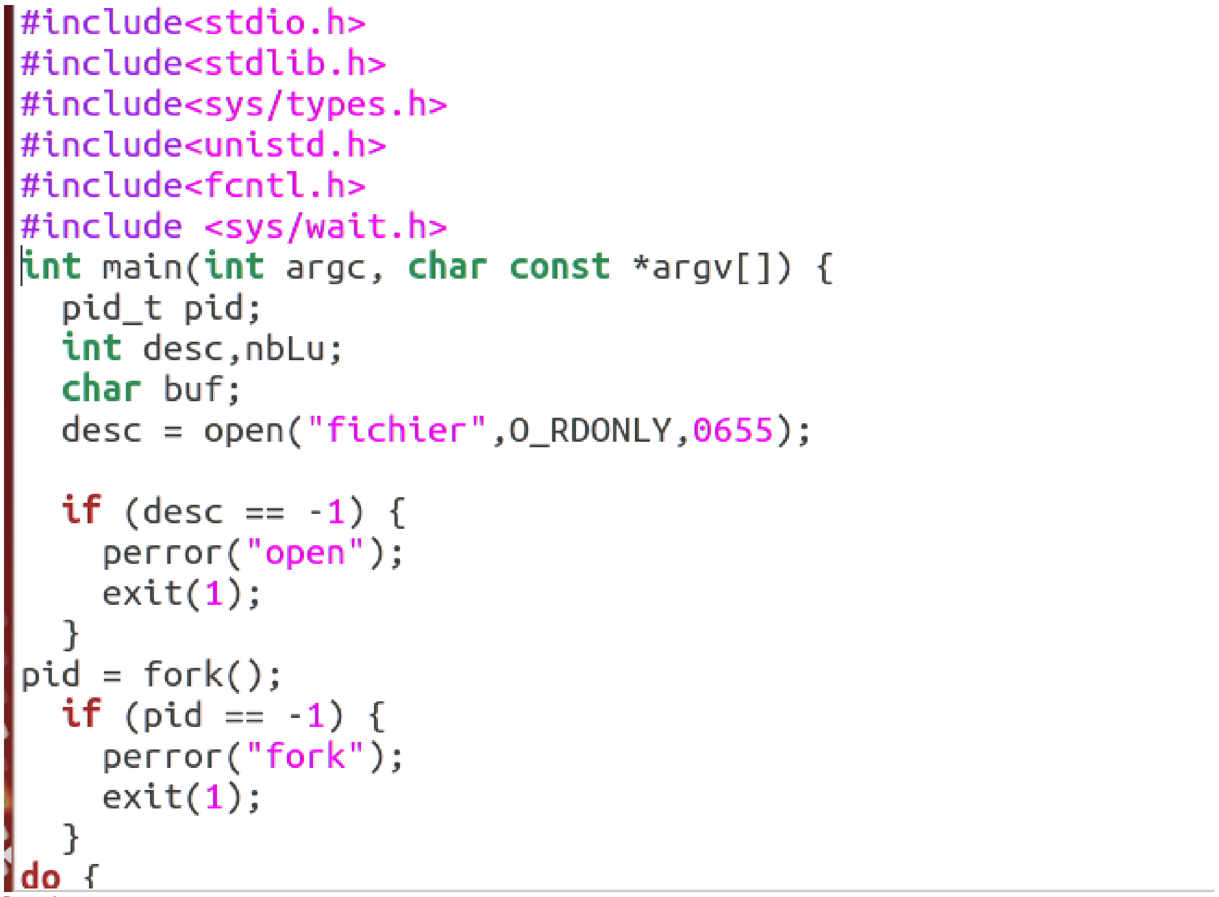


```
      }//fin de 1ere else
  }//find de boucle for

  return 0;
}
```

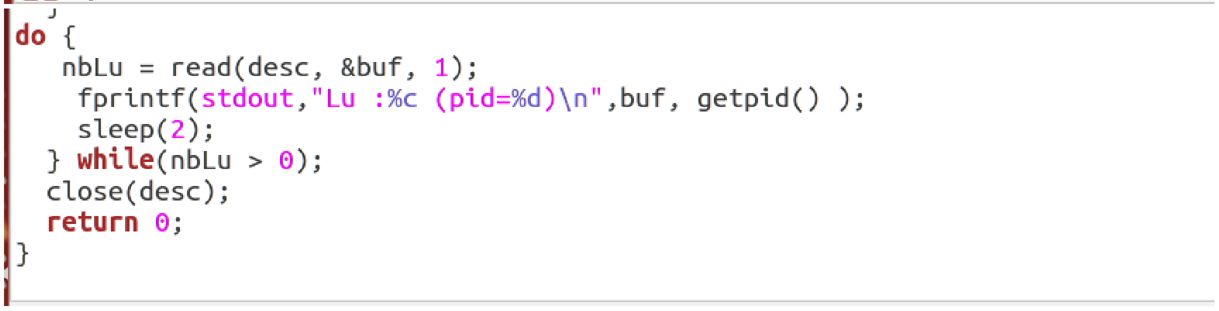


```
uir_student@ubuntu:~/Desktop$ ./ex3c
Usage: creer nb fils
```

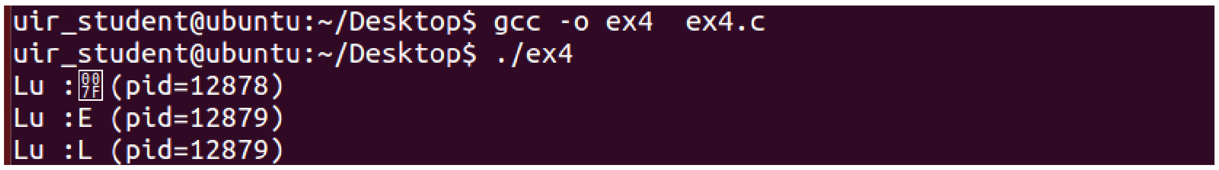
Exercice 4 :



```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<unistd.h>
#include<fcntl.h>
#include <sys/wait.h>
int main(int argc, char const *argv[]) {
  pid_t pid;
  int desc,nbLu;
  char buf;
  desc = open("fichier",O_RDONLY,0655);

  if (desc == -1) {
    perror("open");
    exit(1);
  }
pid = fork();
  if (pid == -1) {
    perror("fork");
    exit(1);
  }
do {
```

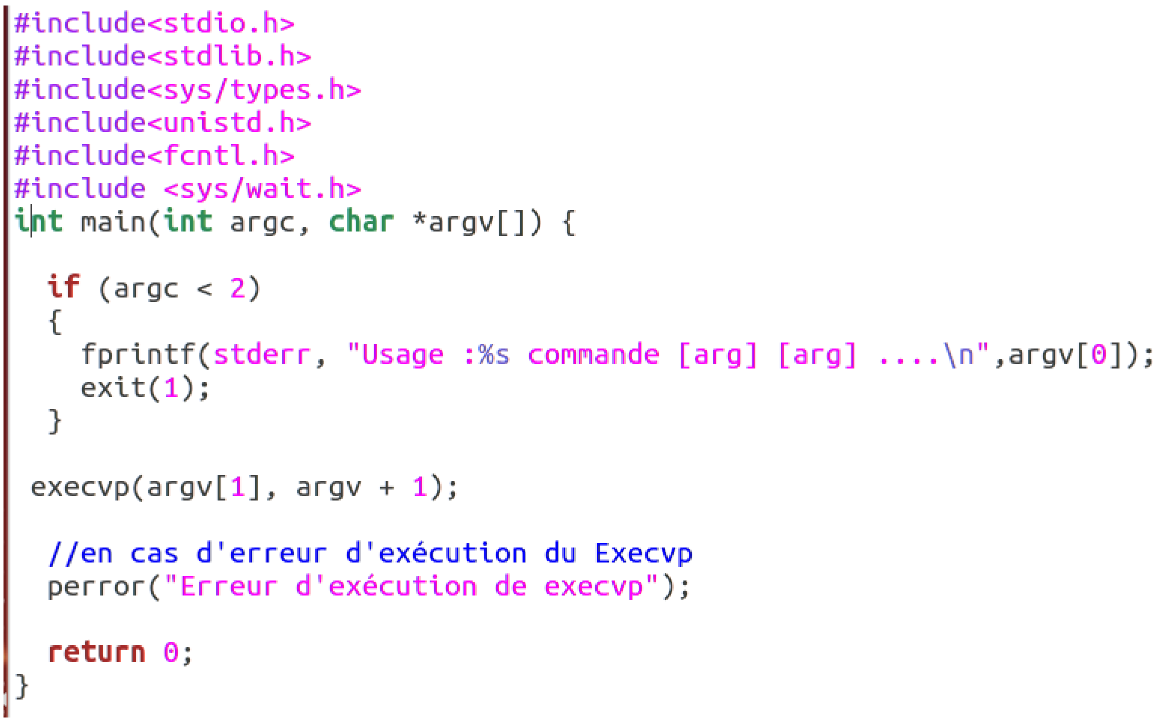


```
do {
    nbLu = read(desc, &buf, 1);
     fprintf(stdout,"Lu :%c (pid=%d)\n",buf, getpid() );
     sleep(2);
  } while(nbLu > 0);
  close(desc);
  return 0;
}
```

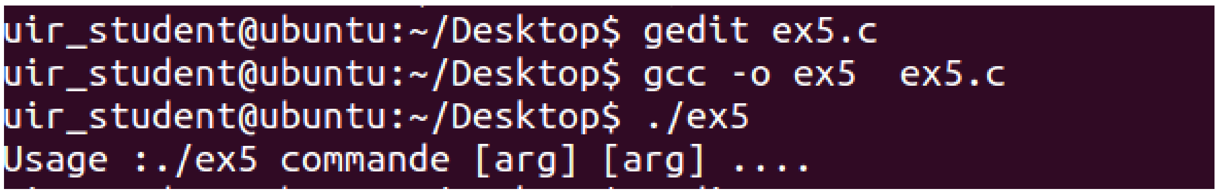


```
uir_student@ubuntu:~/Desktop$ gcc -o ex4  ex4.c
uir_student@ubuntu:~/Desktop$ ./ex4
Lu :(pid=12878)
Lu :E (pid=12879)
Lu :L (pid=12879)
```

Exercice5 : la commande execvp



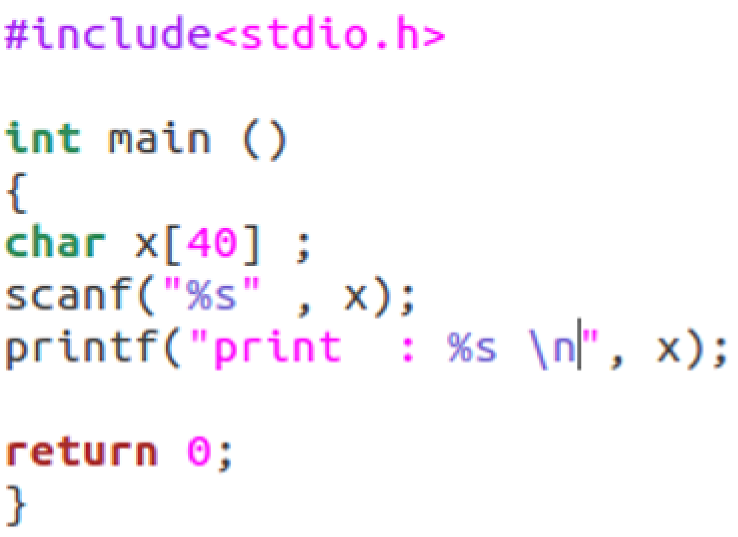
```c
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<unistd.h>
#include<fcntl.h>
#include <sys/wait.h>
int main(int argc, char *argv[]) {

  if (argc < 2)
  {
    fprintf(stderr, "Usage :%s commande [arg] [arg] ....\n",argv[0]);
    exit(1);
  }

 execvp(argv[1], argv + 1);

  //en cas d'erreur d'exécution du Execvp
  perror("Erreur d'exécution de execvp");

  return 0;
}
```



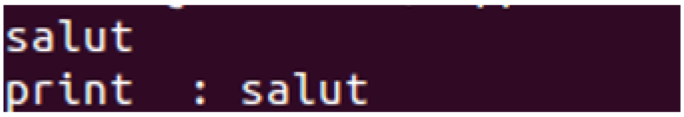
```
uir_student@ubuntu:~/Desktop$ gedit ex5.c
uir_student@ubuntu:~/Desktop$ gcc -o ex5  ex5.c
uir_student@ubuntu:~/Desktop$ ./ex5
Usage :./ex5 commande [arg] [arg] ....
```

Exercice 3 : La fonction execl()

3.1



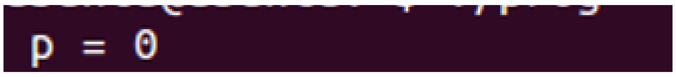
```c
#include<stdio.h>

int main ()
{
char x[40] ;
scanf("%s" , x);
printf("print  : %s \n", x);

return 0;
}
```



```
salut
print  : salut
```

3.2

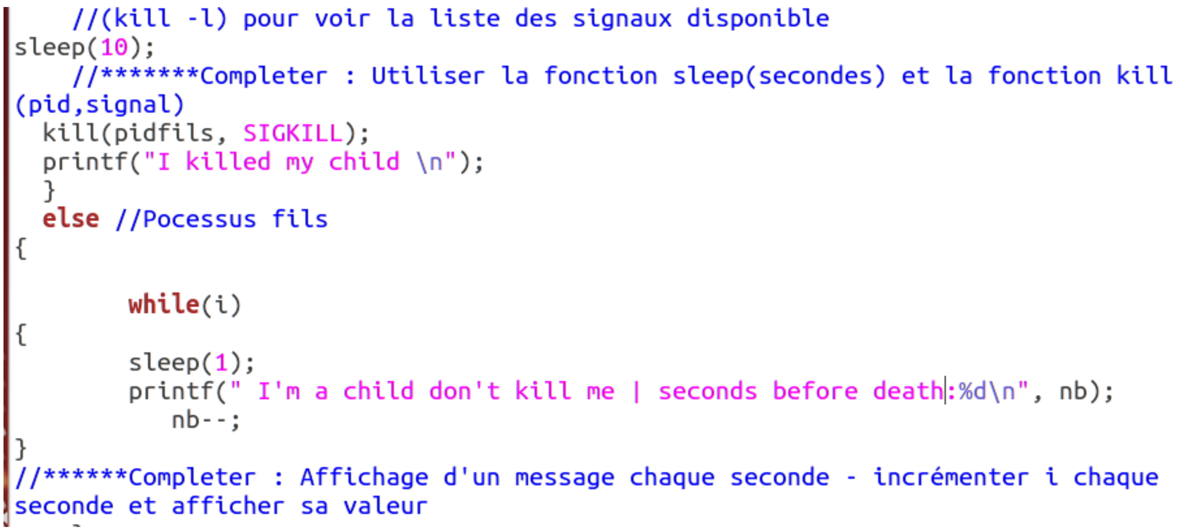
```
#include<unistd.h>
#include<stdio.h>

int main() {
int p;
p = fork();

if(p == 0){
execl("/ubuntu/print","print","salut",(char *) NULL);
printf(" p = %d \n" , p);
}
return 0;
}
```



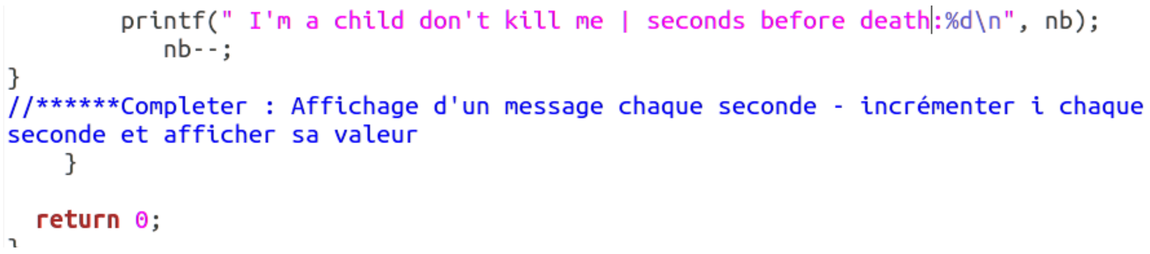
```
 p = 0
```

Exercice 4 : la fonction kill()

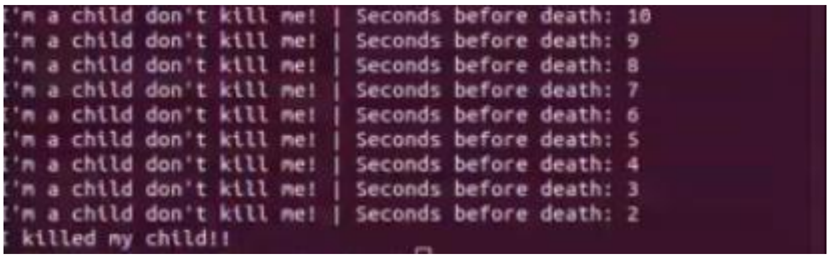
```
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <stdlib.h>
int main(int argc, char const *argv[]) {
   int i = 0; // pour calculer le temps ecoulé
  int pidfils = fork();
int nb=10;
  // en cas d'erreur de création de processus
  if(pidfils < 0)
  {
    perror("Erreur de fork");
    exit(-1);
  }

  if (pidfils > 0)
  {
    /* Processus père */
    //dormir 10 secondes et tuer le processus fils de pid = pidfils avec le signal
SIGKILL : 9
```



```
    //(kill -l) pour voir la liste des signaux disponible
sleep(10);
    //*******Completer : Utiliser la fonction sleep(secondes) et la fonction kill
(pid,signal)
  kill(pidfils, SIGKILL);
  printf("I killed my child \n");
  }
  else //Pocessus fils
{

        while(i)
{
        sleep(1);
        printf(" I'm a child don't kill me | seconds before death:%d\n", nb);
           nb--;
}
//******Completer : Affichage d'un message chaque seconde - incrémenter i chaque
seconde et afficher sa valeur
```



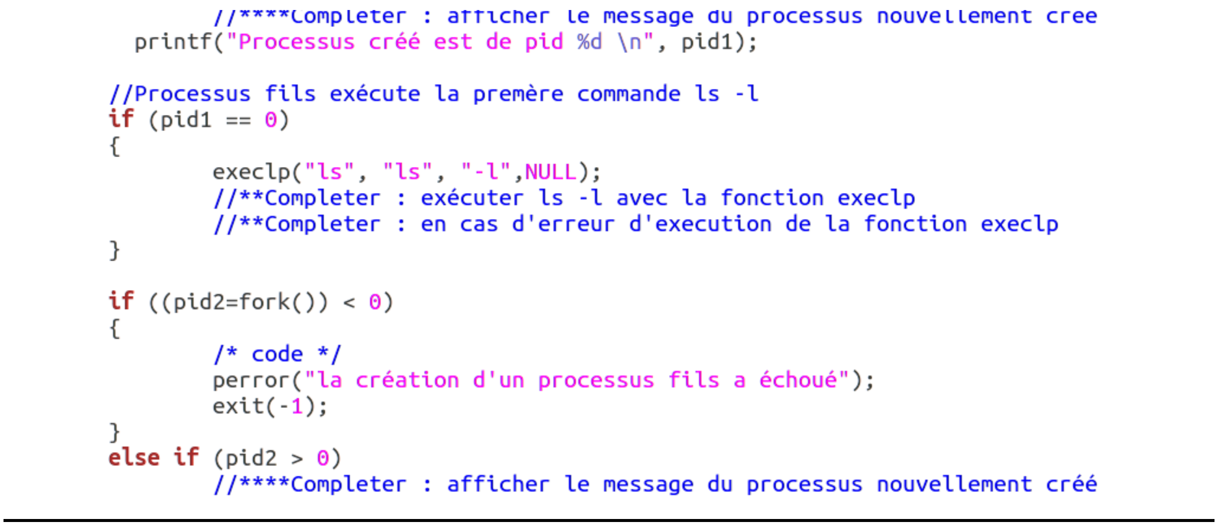
```
        printf(" I'm a child don't kill me | seconds before death:%d\n", nb);
           nb--;
}
//******Completer : Affichage d'un message chaque seconde - incrémenter i chaque
seconde et afficher sa valeur
    }

  return 0;
```



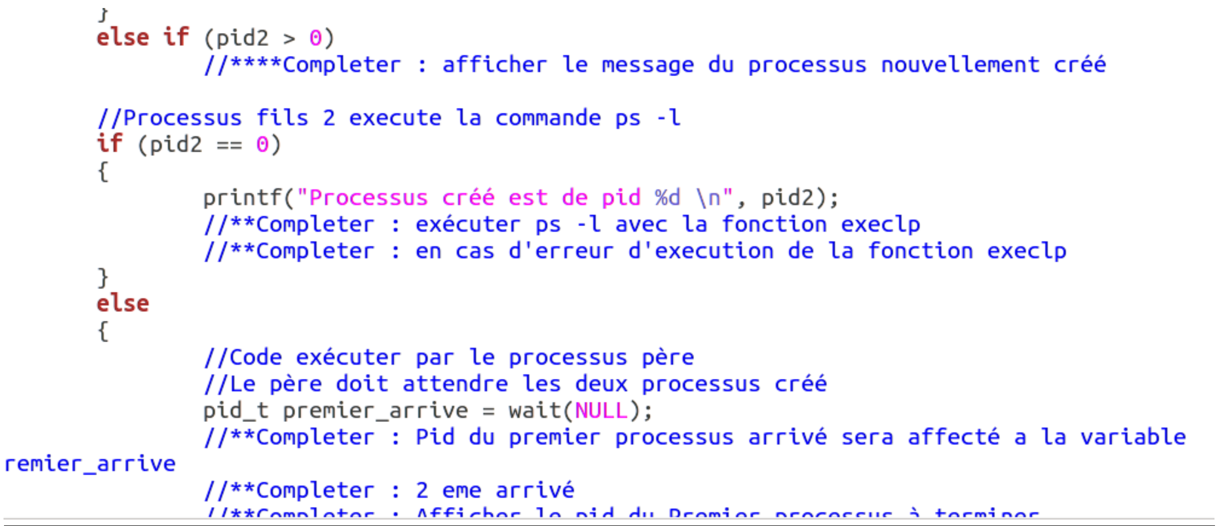
```
'm a child don't kill me! | Seconds before death: 10
'm a child don't kill me! | Seconds before death: 9
'm a child don't kill me! | Seconds before death: 8
'm a child don't kill me! | Seconds before death: 7
'm a child don't kill me! | Seconds before death: 6
'm a child don't kill me! | Seconds before death: 5
'm a child don't kill me! | Seconds before death: 4
'm a child don't kill me! | Seconds before death: 3
'm a child don't kill me! | Seconds before death: 2
 killed my child!!
```

Exercice 5 : la fonction wait()

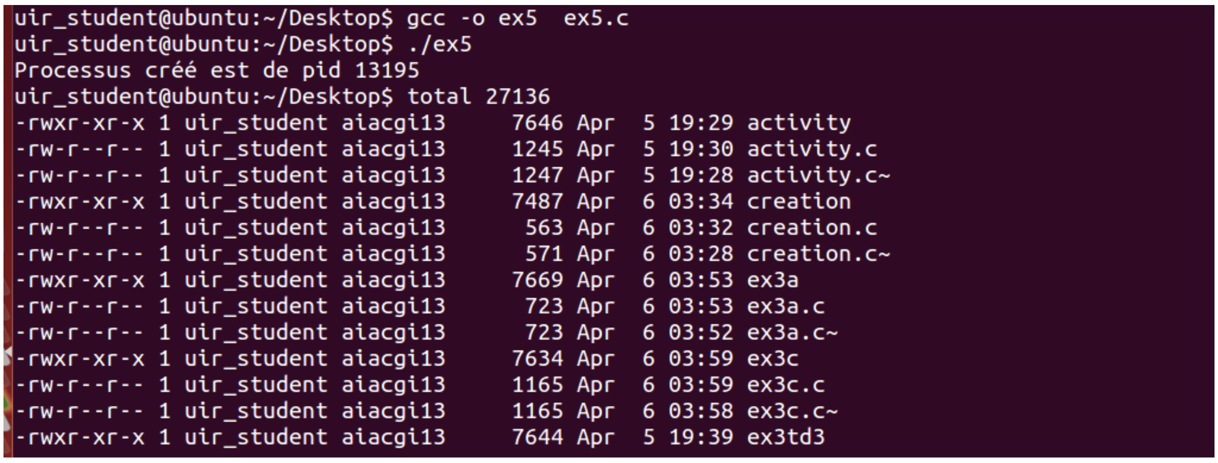
```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<unistd.h>
#include <sys/wait.h>
int main(int argc, char const *argv[]) {

        pid_t pid1,pid2;
        int status;

        /* Création du premier Processus */
        if ((pid1=fork()) < 0)
        {
                /* En cas d'erreur de création du processus */
                perror("la création d'un processus fils a échoué");
                exit(1);
        }
        /*Processus père affiche le pid de son fils nouvellement créé*/
        else if(pid1 > 0)
                //****Completer : afficher le message du processus nouvellement créé
```



```
                //****Completer : afficher le message du processus nouvellement cree
        printf("Processus créé est de pid %d \n", pid1);

        //Processus fils exécute la premère commande ls -l
        if (pid1 == 0)
        {
                execlp("ls", "ls", "-l",NULL);
                //**Completer : exécuter ls -l avec la fonction execlp
                //**Completer : en cas d'erreur d'execution de la fonction execlp
        }

        if ((pid2=fork()) < 0)
        {
                /* code */
                perror("la création d'un processus fils a échoué");
                exit(-1);
        }
        else if (pid2 > 0)
                //****Completer : afficher le message du processus nouvellement créé
```



```
        }
        else if (pid2 > 0)
                //****Completer : afficher le message du processus nouvellement créé

        //Processus fils 2 execute la commande ps -l
        if (pid2 == 0)
        {
                printf("Processus créé est de pid %d \n", pid2);
                //**Completer : exécuter ps -l avec la fonction execlp
                //**Completer : en cas d'erreur d'execution de la fonction execlp
        }
        else
        {
                //Code exécuter par le processus père
                //Le père doit attendre les deux processus créé
                pid_t premier_arrive = wait(NULL);
                //**Completer : Pid du premier processus arrivé sera affecté a la variable
remier_arrive
                //**Completer : 2 eme arrivé
                //**Completer : Afficher le pid du Premier processus à terminer
```



```
uir_student@ubuntu:~/Desktop$ gcc -o ex5  ex5.c
uir_student@ubuntu:~/Desktop$ ./ex5
Processus créé est de pid 13195
uir_student@ubuntu:~/Desktop$ total 27136
-rwxr-xr-x 1 uir_student aiacgi13    7646 Apr  5 19:29 activity
-rw-r--r-- 1 uir_student aiacgi13    1245 Apr  5 19:30 activity.c
-rw-r--r-- 1 uir_student aiacgi13    1247 Apr  5 19:28 activity.c~
-rwxr-xr-x 1 uir_student aiacgi13    7487 Apr  6 03:34 creation
-rw-r--r-- 1 uir_student aiacgi13     563 Apr  6 03:32 creation.c
-rw-r--r-- 1 uir_student aiacgi13     571 Apr  6 03:28 creation.c~
-rwxr-xr-x 1 uir_student aiacgi13    7669 Apr  6 03:53 ex3a
-rw-r--r-- 1 uir_student aiacgi13     723 Apr  6 03:53 ex3a.c
-rw-r--r-- 1 uir_student aiacgi13     723 Apr  6 03:52 ex3a.c~
-rwxr-xr-x 1 uir_student aiacgi13    7634 Apr  6 03:59 ex3c
-rw-r--r-- 1 uir_student aiacgi13    1165 Apr  6 03:59 ex3c.c
-rw-r--r-- 1 uir_student aiacgi13    1165 Apr  6 03:58 ex3c.c~
-rwxr-xr-x 1 uir_student aiacgi13    7644 Apr  5 19:39 ex3td3
```