



LMD- Classe S5 - Informatique 2020-2021
Licence informatique

Solution TD N° 2 : primitives système de gestion des processus

Questions de cours

Q1 : Dans le système Unix, est ce que tout processus à un père ? Que se passe t-il lorsqu'un processus devient **orphelin** (mort de son père) ? Quand un processus passe à l'état **zombie** ?

Reponse :

-Oui, sauf le processus init() c'est le processus système qui est le père de tous les processus dans le système .

-Un processus orphelin : c'est un processus qui perdu son père (c'est a dire le processus père termine son exécution avant son fils.

Le processus qui a perdu son père(processus orphelin), il sera adopté et attaché au processus système init()

-Un processus zombie : c'est un processus qui a terminé son exécution avant son père

Q2 : Expliquer pourquoi dans Unix, lorsqu'un processus exécute l'appel système **exit ()**, ses ressources ne sont pas libérées tout de suite.

Le processus passe a l'état zombie reste toujours chargé dans la mémoire principale, il sera supprimé du système d'exploitation et perdre tous ses ressources une fois le père exécute wait()

Q3: quelle est la relation qui existe entre primitive système **wait** et **exit**. Comment un processus père attendre ces threads ?**c'est une relation de synchronisation**

Q4 : Indiquez pourquoi il est important de faire des appels aux fonctions **wait** dans les processus parents.

L'appel système wait permet de vider la mémoire de processus zombies qui ont terminés leur exécution

Q5 : Qui suis-je ?

- a) Je suis un processus qui s'est terminé mais son père n'a pas encore lu son code de retour. (processus zombie)
- b) Je suis un processus dont le père s'est terminé avant lui. (processus orphelin)
- c) Je suis un appel système qui permet de créer un processus. (fork())
- d) Je suis un appel système qui affiche le PID du processus père. (getppid)

Exercice N°1

Utiliser le compilateur c sous linux GCC pour compiler les différents programmes tel que la syntaxe est comme suite :

#gcc -c nom-fichier.c

#gcc -o nom-executable nom-fichier.o

#./nom-executable

L'exécution de programme est comme suite : ./nom-fichier

Pour les threads dans la phase édition ajouter l'option : lpthread

Exemple : #gcc -o nom-executable nom-fichier.o -lpthread

```

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

int main()
{

    int i;
    i=3;
    int id;

    if ((id=fork())==0)
    {
        i=10;
        printf("%d\n",i);
        exit(i);
        fork();
        i=1;
    }

    wait(NULL);
    printf("%d\n",i) ;
    i=5;
    printf("%d\n",i);
    return 0;
}

```

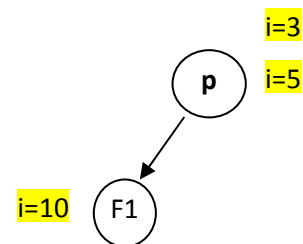
Q1 : Quel est le résultat affiché par ce programme ?

Solution

```

> gcc -c exo1.c
> gcc -o exo1 exo1.o
> ./exo1
10
3
5

```



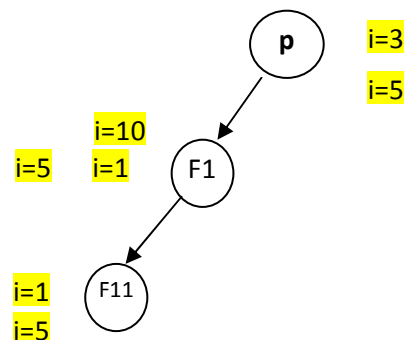
Q2 : Que se passe-t-il lorsque on supprime l'instruction `exit(i)` ; quelle est le nouveau résultat affiché par ce programme ?

Solution

```

> gcc -o exo12 exo1.c
> ./exo12
10
1
5
1
5
3
5

```



Exercice N°2

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <stdlib.h>
int i =2;
int main()
{
    pid_t p1,p2;
    if ((p1=fork())==0) f1();
    if (p1==0) f2();
    else if ((p2= fork())==0) f3();
        if (p1==0) f4();
        if (p2==0) f4();
    pere();
}

void pere(void)
{
    sleep(3);
    while ( wait(NULL)> 0);
    i= i+1;
    printf("je suis pere i =%d\n",i);
}

void f1(void)
{
    sleep(6);
    i= i+1;
    printf("Fils1 i=%d\n",i);
    exit(0);
}

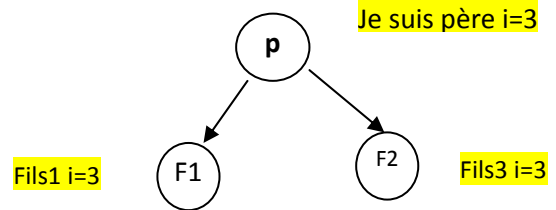
void f2(void)
{ sleep(4);
  i = i+1;
  printf("Fils2 i =%d\n",i);
}

void f3(void)
{ sleep(2);
  i=i+1;
  printf("Fils3 i =%d\n",i);
  exit(0);
}

void f4(void)
{ sleep(1);
  i=i+1;
  printf("Fils4 i =%d\n",i);
}
```

Q1 : indiquez le nombre de processus créés dans ce programme (dessiner l'arborescence)

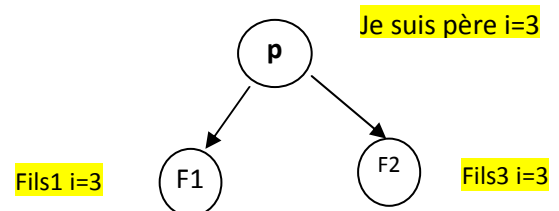
```
./exo21
Fils3 i =3
Fils1 i=3
je_suis pere i =3
```



Q2 : quelle est le nouveau résultat affiché par ce programme si on supprime **else** souligné ?

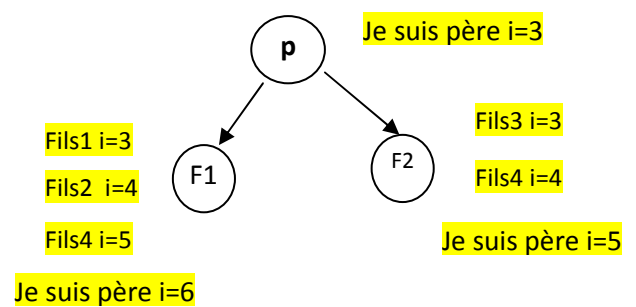
Même résultat que la question 1

```
Fils3 i =3
Fils1 i=3
je suis pere i =3
ubuntu@ubuntu-Lenovo-G500:~/Bureau$
```



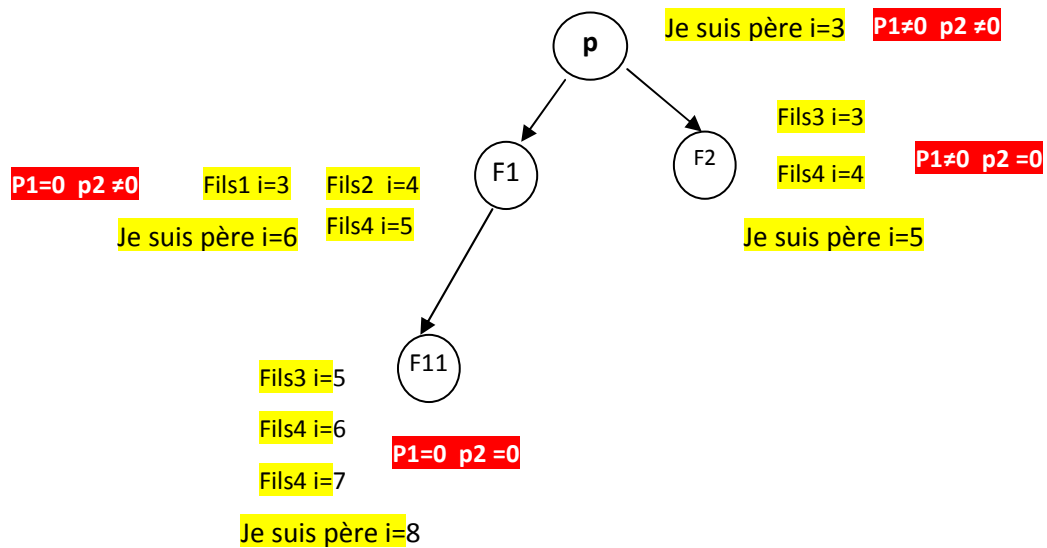
Q3 : si on supprime `exit(0)` a la fin de la fonction `f1()` et `f3()` , quelle est le résultat affiché :

```
Fils3 i =3
Fils4 i =4
Fils1 i=3
je suis pere i =5
Fils2 i =4
Fils4 i =5
je suis pere i =6
je suis pere i =3
ubuntu@ubuntu-Lenovo-G500:~/Bureau$
```



Q4 : si on supprime `exit(0)` a la fin de la fonction `f1()` et `f3()` , quelle est le résultat affiché et aussi else :

```
Fils3 i =3
Fils4 i =4
Fils1 i=3
je suis pere i =5
Fils2 i =4
Fils4 i =5
Fils3 i =5
Fils4 i =6
Fils4 i =7
je suis pere i =8
je suis pere i =6
je suis pere i =3
ubuntu@ubuntu:~$
```



Exercice N° 3

Vous avez le programme suivant en c :

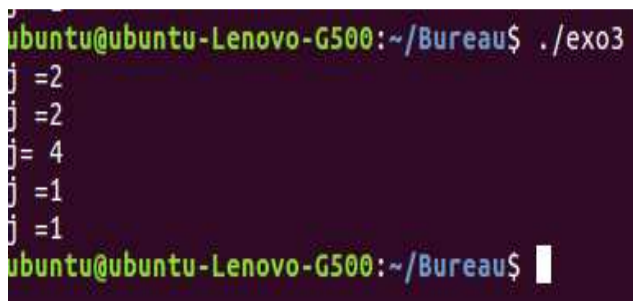
```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.h>
#include <stdlib.h>
Void main()

{

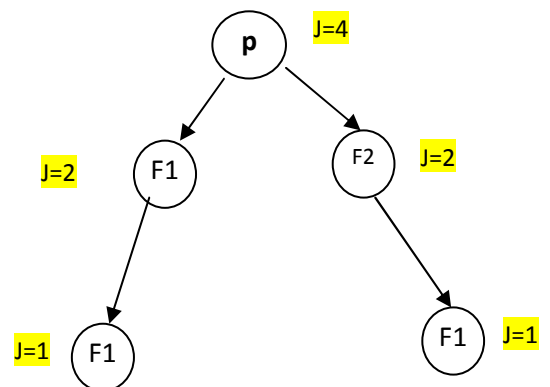
int j =3, id, i ;
for (i=0 ; i<2; i++)
{
    if ((id=fork())==0)) {
        j=j-1;
        printf("j =%d\n",j);
        if ((id=fork())==0)) { j = j-1;
            printf("j =%d\n",j);
        }

        exit(0);
    }
}
wait(0);
j = j +1;
printf("j=    %d\n",j);
}
```

Q1 : En supposant que fork() ne renvoie jamais -1, combien de processus vont être créés ? Justifier votre réponse.



```
ubuntu@ubuntu-Lenovo-G500:~/Bureau$ ./exo3
j =2
j =2
j= 4
j =1
j =1
ubuntu@ubuntu-Lenovo-G500:~/Bureau$
```



Q2 : Quelle est la valeur de j affichée par chacun des processus créés ? Justifier votre réponse.

```
ubuntu@ubuntu-Lenovo-G500:~/Bureau$ ./exo3
j =2
j =1
j =2
j =1
j =1
j= 2
j =0
j= 3
j =0
j= 1
j= 4
j= 2
j =-1
j= 0
j= 1
j= 2
ubuntu@ubuntu-Lenovo-G500:~/Bureau$ j= 3
```

