TD – 3

Exercice 1

Prédire le résultat du programme C suivant :

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main() {

    fork();
    printf("Bonjour!\n");
    return 0;
}
```

Bonjour !
Bonjour !
Le processus principal après l'appel « fork » créera un seul enfant dont les deux processus Parent/enfant exécute l'instruction d'affichage.

Exercice 2

Calculer le nombre de fois le message « Bonjour ! » sera affiché sur l'écran :

```
#include <stdio.h>
#include <sys/types.h>
int main() {
    fork();
    fork();
    fork();
    printf("Bonjour !\n");
    return 0;
}
```

Bonjour !
Bonjour !
Bonjour !
Bonjour !
Bonjour !
Bonjour !
Bonjour !
Bonjour !

Le nombre de fois « bonjour » est affiché est égal au nombre de processus créés. Le nombre total de processus = 2^n, où n est le nombre d'appels système fork. Donc ici n = 3, $2^3 = 8$.

Exercice 3

Prédire le résultat du programme suivant :

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

```c
#include <unistd.h>

void forkexemple() {
    int x = 1;

    if (fork() == 0)
        printf("L'enfant a x = %d\n", ++x);
    else
        printf("Le parent a x = %d\n", --x);
}
int main()
{
    forkexemple();
    return 0;
}
```

L'enfant a x = 2
Le parent a x = 0
ou
Le parent a x = 0
L'enfant a x = 2
Le parent et l'enfant s'exécutent simultanément, deux sorties sont donc possibles.

Exercice 4

1. Si un programme exécute le code suivant :

```c
for (i = 0; i < n; i++)
    fork();
```

Que serait le nombre de processus enfants ?

$2^n - 1$

2. Ecrire un programme qui exécute une commande Linux qu'on fournit comme paramètre du programme. Exemple : « **./prog ls /home** »

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main(int argc, char* argv[]) {
  int ret;
  ret = execvp(argv[1], &argv[1]);
  if(ret == -1)
     perror("Return from execvp() not expected");
  return 0;
}
```

3. Combien de processus engendre l'exécution du code :

```c
fork() && (fork() || fork())
```

Le processus courant (appelons-le le père) engendre dans l'ensemble 3 autres processus. En effet, comme dans une instruction a && b, b n'est pas ´évaluée si l'´évaluation de 'a' donne 0, de même, dans une instruction a || b, b n'est pas ´évaluée si l'´evaluation de 'a'

ne donne pas 0. Donc, dans « fork() && b » seulement le père exécute b, et dans « fork() || b » seulement l'enfant exécute b.

4 processus:

« ./exercice4 & (sleep 1 ; ps –o "%P%p%c") »

```
#include <unistd.h>


int main(void) {

        fork() && (fork() || fork());

        sleep(10);

        return 0;

}
```

Exercice 5

Que fait le programme suivant ?

```
#include<stdio.h>
#include<unistd.h>
#define MAX 5
int main() {
char *argv[MAX];
argv[0] = "ls"; argv[1] = "-lR"; argv[2] = "/"; argv[3] = NULL;
execvp("ls", argv);
}
```

il exécute la commande shell ''ls -lR /" (ls récursif, avec affichage long)