PROCESSUS

Dans ce TD on va utiliser les fonctions suivantes

int fork()

– L'appel système fork duplique le processus.
– Après l 'appel, l'exécution continue dans les 2 processus (fils et père)
– Pour savoir dans quel processus (père/fils) on se retrouve, il faut regarder la valeur retournée par la fonction :

- PID (fils) pour le processus père
- 0 pour le processus fils
- -1 en cas d'échec (manque de mémoire, trop de processus, …)

getpid()

Cette fonction retourne le numéro du processus courant

Exemple :

```
void main() {
int pid, i ;
        pid=fork() ;
        if (pid == -1)
                printf("erreur : echec du fork()\n");
        else if (pid == 0)
                printf("je suis le processus fils\n ");
          else
               printf("je suis le processus père\n ")
}
```

1. Quel est le résultat du programme suivant ?

```
void main() {
        int valeur ;
        valeur = fork() ;
        printf("je suis le processus numero %d\n", (int) getpid() ) ;
}
```

2. Soit un processus qui fait :

```
main() {
        printf ("Bonjour");
        if (fork ())
                printf (" Monsieur\n");
        else
                printf (" Madame\n");
}
```

Que se passe-t-il ? Pourquoi ? Comment y remédier ?

3. Combien de processus sont créés par le programme suivant ?

```
void main() {
        fork() ;
        fork() ;
        fork()
}
```

4. Ecrire un programme qui crée deux fils, l'un affiche les entiers de 1 a 20, l'autre de 21 a 40

5. L'appel système **waitpid(pid)** permet à un processus d'attendre la terminaison d'un autre. Ecrire un petit programme qui lance un processus et qui attend sa fin.

6. Le système d'exploitation met à notre disposition trois appels système permettant aux processus d'interagir au moyen de signaux. L'appel **kill(pid,signal)** envoie un signal à un processus tandis que l'appel **sigaction(signal,procédure)** permet à un processus d'accomplir une action spéciale en cas de réception d'un signal. Le dernier appel **pause()** endort un processus jusqu'à ce qu'il reçoive un signal. Ecrire un petit programme qui lance un processus et qui attend le signal SIGUSR1 de sa part. Sur réception de ce signal chacun des deux processus devra réexpédier ce signal à l'autre et se mettre en attente d'un nouveau signal. Une sorte de ping-pong ultra rapide ou le signal SIGUSR1 remplace la balle donc.

7. Pour calmer le jeu du ping-pong et faire qu'il soit observable par un être humain, nous souhaitons que le processus recevant le signal SIGUSR1 affiche ping ou pong à l'écran (suivant qu'il s'agit du père ou du fils) et attende deux secondes avant de renvoyer le signal. Pour faire attendre un processus, le système nous fournit l'appel système **alarm(secondes)** qui programme l'envoie du signal SIGALRM dans le nombre de secondes indiqué à partir de l'instant de l'appel. Ainsi si un processus exécute alarm(5) il recevra le signal SIGALRM dans cinq secondes quoi qu'il fasse entre-temps. Ecrire un premier programme qui attend pendant cinq secondes et qui se termine. Réécrire ensuite le programme du ping-pong comme nous le souhaitons (avec les messages et les temporisations).

Le processus de départ (bidon1) va exécuter 3 fork donc créer 3 processus (bidon2, bidon3, bidon4).

Chacun de ces 3 fils possède le même segment de code que le père, et va donc s'exécuter là où bidon1 en était :

- *bidon2 va donc exécuter 2 fork (bidon5 et bidon6)*
- *bidon5 va reprendre là où en était bidon2 et exécuter 1 fork (bidon7)*
- *bidon6 créé par le 3ème fork ne va rien faire*
- *bidon3 va exécuter 1 fork (bidon8)*
- *bidon8 créé par le 3ème fork ne va rien faire*

En tout, il y aura 8 processus bidon en parallèle.

Ex 3. Envoi de signaux

Que fait le programme suivant ?

```
/* ================================================== demo-signal.c */
#include <signal.h>

void handler(int sig)
{
                    printf("Signal SIGUSR1 recu\n") ;
                    signal(sig, handler) ;
}

void main()

{
                   signal (SIGUSR1,handler) ;
                    for (;;) {}
}
```

Ce programme se contente de boucler sur lui-même… mais avant cela, il précise qu'il faudra exécuter la fonction `handler` *à chaque fois qu'il recevra le signal* `SIGUSR1`*.*