TD N°4
Synchronisation des processus

Exercice 1

Pour la définition des mécanismes ci-dessous, on se reportera au cours. On prendra soin pour chaque question de redonner les algorithmes d'entrée et de sortie de section critique.
1. Alternance stricte. Montrer que cette technique garantit bien l'exclusion mutuelle.
2. Solution de Peterson. Montrer que cette technique garantit bien l'exclusion mutuelle.
3. Sommeil/activation. Quel est l'avantage de cette technique par rapport à la précédente ?

Exercice 2

On considère l'algorithme suivant d'exécution mutuelle pour deux processus p1 et p2, où les deux booléens p1_dedans et p2_dedans sont initialisés à faux. Seul le texte de p1 est donné, celui de p2 étant tout à fait symétrique :

```
P1:     répéter
                    <section restante 1>
            Tant que p2_dedans faire rien;
            p1_dedans := vrai;
                    <section critique 1>
            p1_dedans := faux;
        Jusqu'à faux
```

1. Justifier l'absence de blocage pour cet algorithme
2. Montrer que cette solution est incorrecte.

Exercice 3

Rappeler pourquoi une simple attente active sur une variable **lock** partagé qui vaudrait vrai si un processus est en section critique et faux sinon, ne convient pas pour résoudre le problème de l'exclusion mutuelle.

Exercice 4

Dans cette exercice, nous allons apprendre à nous servir de la commande fork. Dans le premier exemple, nous faisons simplement une exécution simple d'un fork, sans faire de distinction entre le processus père et le processus fils crée.

L'appel système pid_t fork() permet de créer un processus
! **Le processus créé (fils) est un clone (copie conforme) du processus créateur (père)**
! **Le père et le fils ne se distinguent que par le résultat rendu par fork()**
" **pour le père : le numéro du fils (ou –1 si création impossible)**
" **pour le fils : 0**

A l'exécution de ce code, nous obtenons l'affichage qui suit le programme.

```
#include <stdlib.h>
#include <unistd.h>
int main(void) {
pid_t pid_fils ;
printf("Bonjour depuis processus %d\n",getpid()) ;
fork() ;
printf("Bonjour depuis processus %d\n", getpid()) ;
}
```

```
Bonjour depuis processus 3137
Bonjour depuis processus 3137
Bonjour depuis processus 3138
```

programme testfork.c

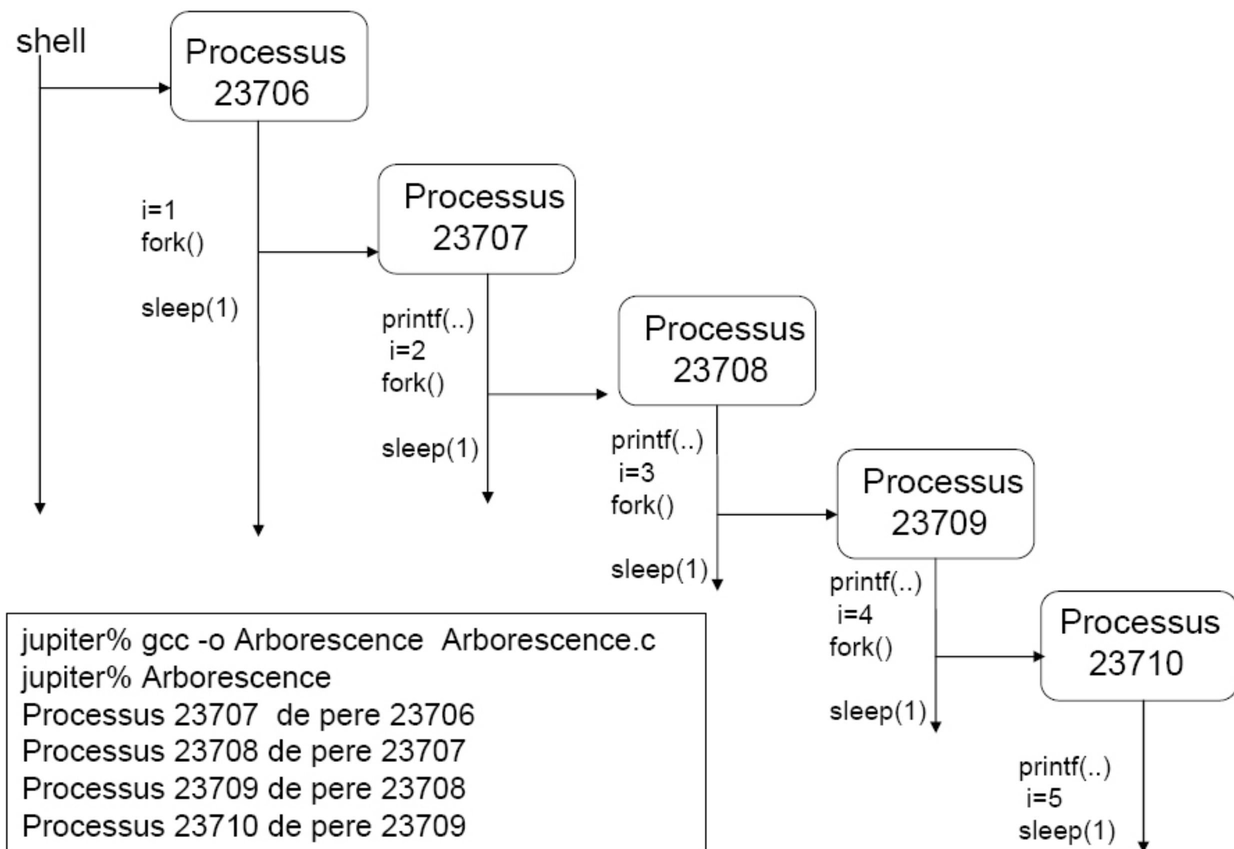
```
int main() {
  if (fork() != 0) {
    printf("je suis le père, mon PID est %d\n", getpid());
    sleep(10)  /* blocage pendant 10 secondes */
    exit(0);
  } else {
    printf("je suis le fils, mon PID est %d\n", getpid());
    sleep(10)  /* blocage pendant 10 secondes */
    exit(0);
  }
}
```

```
<unix> gcc -o testfork testfork.c
<unix> ./testfork & ps
je suis le fils, mon PID est 3148
je suis le père, mon PID est 3147
[2] 3147
 PID TTY          TIME CMD
2989 pts/0   00:00:00 tcsh
3147 pts/0   00:00:00 testfork
3148 pts/0   00:00:00 testfork
3149 pts/0   00:00:00 ps
<unix>
```

Exemple simple d'exécution

Exercice 5

```
// programme Arborescence.c : appel système fork()
#include <sys/types.h> // pour le type pid_t
#include <unistd.h> // pour fork
#include <stdio.h> // pour printf
int main ( )
{ pid_t p;
int i, n=5 ;
for (i=1; i<n; i++){
    p = fork();
    if (p > 0 )
    break ;
    printf(" Processus %d de père %d. \n", getpid(), getppid()) ;
}
sleep(1); // dormir pendant 1 seconde
return 0 ;
}
```



Exercice 6

Sous Linux, la commande fork() retourne une valeur de type pid_t qui vaut 0 si on se trouve dans le processus fils, 1 en cas d'erreur et le PID du fils si on se trouve dans le processus père. Ainsi, nous pouvons faire exécuter un code différent en fonction du PID.

1. Créer un programme qui crée un processus et qui exécute les affichages suivant :

-- "Bonjour depuis le fils, de PID xxx" par le processus fils
-- "Bonjour depuis le père, de PID xxx" par le processus père
-- "Erreur" en cas d'erreur lors de l'appel ‘a la fonction fork().

2. Que fait le programme suivant :

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

int main(void) {
int compt=0 ;
pid_t pid_fils1, pid_fils2 ;
printf("PID du père : %d\n",getpid()) ;
pid_fils1 = fork() ;
if (pid_fils1 == -1) {
      fprintf(stderr," Erreur numéro %d\n",errno) ;
      return(1) ;
}
if (pid_fils1 == 0) {
      compt++ ;
      printf("Fils1 de PID %d : Compteur %d\n",getpid().compt) ;
      pid_fils2 = fork() ;
      if (pid_fils2 == -1) {
            fprintf(stderr," Erreur numéro %d\n",errno) ;
            return(1) ;
      }
      if (pid_fils2 == 0) {
            compt++ ;
            printf("Fils2 de PID %d : Compteur %d\n",getpid(),compt) ;
      }
}
else {
      compt++ ;
      printf("Père de PID %d : Compteur %d\n",getpid(),compt) ;
}
}
```