Examen système d'exploitation **3ième année SI** **2019/2020**
Documentation non autorisée durée: 1h

Exercice 1 (2 pts) :

1. Quelles sont les opérations d'un sémaphore qui s'exécutent d'une manière indivisible ?
   Le test de la valeur du sémaphore, le changement de sa valeur et la mise en attente éventuelle sont effectués en une seule opération atomique indivisible. C'est-à-dire les opérations P(S) et V(S). 1pts
2. L'appel système fork() est le moyen pour créer des processus, par duplication d'un processus existant. Expliquer comment distinguer entre le processus père et le processus fils.
   Pour distinguer le processus père du processus fils on regarde la valeur de retour de fork(), qui peut être:1pts

- **La valeur 0 dans le processus fils.**
- **Positive pour le processus père et qui correspond au PID du processus fils**
- **Négative si la création de processus a échoué ;**

Exercice 2 (8 pts) : On considère deux processus P1 et P2 concurrents.
Les codes des processus P1 et P2 sont les suivants:

```
P1 ( ) {                          P2 ( ) {
while(TRUE)                         while(TRUE)
 printf("je suis le processus 1\n");   printf("je suis le processus 2\n");
        }                             }
```

1- Synchroniser les deux processus à l'aide des sémaphores afin de montrer à l'écran la sortie suivante :

je suis le processus 2
je suis le processus 1
je suis le processus 2
je suis le processus 1

sem_t s1,s2 **1pts**
sem_init(s1,0,1) ; /*pour permettre au processus P2 de commencer*/ *0.5pt*
sem_init(s2,0,0) ***/*pour empêcher le processus P1 de commencer jusqu'à ce que P2 le réveil */*** *0.5pt*

```
P1 ( ) {                                  P2 ( ) {
while(TRUE)   {                           while(TRUE)   {
sim_wait(&s2) ; //   0,5 pt                   sim_wait(&s1)  0,5
printf("je suis le processus 1\n");           printf("je suis le processus 2\n");
sim_post(&s1) ;        0,5                    sim_post(&s2);  0,5
        }                                   }
}                                         }
```

1- Synchroniser les deux processus à l'aide des sémaphores afin de montrer à l'écran la sortie suivante:

je suis le processus 2
je suis le processus 1
je suis le processus 1
je suis le processus 2

sem_t s1, s2
sem_init(s1,0,2) ; /*pour permettre au processus P2 de commencer*/ *0.5pt*
sem_init(s2,0,0) ***/*pour empêcher le processus P1 de commencer jusqu'à ce que P2 le réveil */*** *0.5pt*

```
P1( ) {
While (TRUE){
sim_wait(&s2) ; //              (0,5 pt)
        printf("je suis le processus 2\n");  ;
sim_post(&s1) ;  //             (0,5 pt)
}
}
```

```
P2 ( ) {
While (TRUE){
sim_wait(&s1) ;sim_wait(&s1);    (1 pt)
printf("je suis le processus 2\n");  ;
sim_post(&s2); sim_post(&s2) ;   (1 pt)
}
}
```

Exercice 3 (5 pts) ::

Écrire un programme utilisant 3 threads afin d'afficher la séquence suivante:

A, B,C, A,B, C...,

Où, le thread 1 affiche la lettre A, le thread 2 affiche la lettre B et le thread 3 affiche la lettre C

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
sem_t sem1, sem2, sem3;
void *f1(){
   while(1){
   sem_wait(&sem1);
   printf("A, ");
   sem_post(&sem2);
  }
}
```

```
void *f2(){
    while(1) {
    sem_wait(&sem2);
    printf("B, ");
    sem_post(&sem3);
  }
}
```

```
void *f3(){
  while(1) {
    sem_wait(&sem3);
    printf("C, ");
    sem_post(&sem1);
}
 }
```

```
int main(){
  pthread_t p1, p2, p3;
  sem_init(&sem1, 0, 1);
  sem_init(&sem2, 0, 0);
  sem_init(&sem3, 0, 0);
 pthread_create(&p1, NULL, f1,  (void*)NULL);
 pthread_create(&p2, NULL, f2,  (void*)NULL);
 pthread_create(&p3, NULL, f3,  (void*)NULL);
 pthread_join(p1, NULL);
 pthread_join(p2, NULL);
 pthread_join(p3, NULL);
 return 0;
}
```

Exercice4 (5 pts) :

Écrire un programme en C qui :

- Initialise un tableau aléatoirement,
- Crée un processus fils,
- Calcule la somme des éléments du tableau dans une variable S,
- Attend la terminaison de son fils et affiche S.

Le processus fils calcule et affiche le maximum du tableau.

```
# define N 100
int main () {
float   t[N], S=0,max=0;
int i;
pid_t P
for (i=0; i < N; i++)
        t[i] = rand ();
P = fork ();
```

```
if (P == 0) {
  for(i=0 ; i<N ; i++)
          if (max < t[i])
              max= t[i];
 printf ("Max=%f \n", max);
}
```

```
else {
   for(i=1; i<N; i++)
        S+= t[i];
   waitpid (P ,NULL ,0);
   printf ("S=%f \n", S);
}
return 0;
}
```