Université de Bouira
Département d'informatique — 3ième Année SI (2019/2020)
Module : système d'exploitation — Durée : 1h

Examen de Rattrapage

Exercice 1 (4 pts) :

1. Qu'est-ce qu'un PCB et de quoi est-ce composé ? (1 pt)

PCB est une structure de données particulière appelée bloc de contrôle de processus (PCB : Process Control Bloc) et dont le rôle est de reconstituer le contexte du processus.
PCB contient l'ensemble des informations dynamiques qui représente l'état d' exécution d'un processus. Il contient aussi des informations sur l'ordonnancement du processus (priorité du processus, les pointeurs sur les files d'attente)

2. Dans quel(s) état(s) peut se retrouver un processus après l'état exécution? **(1 pt)**

Après l'état exécution, un processus peut se trouver dans l'une des états suivantes: Prêt, Bloqué ou Fin

3. Quelle est la différence entre un processus lourd et un processus léger (thread) ?

Le processus lourd est créé par duplication (copie exacte du processus original) d'un processus existant.
Le deux processus lourd ne partage pas le même espace mémoire. **(1 pt)**
Le thread créateur et le thread créé partagent tous deux le même espace mémoire (**1 pt)**

Exercice 2 (8 pts) :

On considère le problème du producteur consommateur. Le processus **producteur**, délivre des messages à un processus **consommateur**. Le **producteur** produit le message dans la *ZoneP*, puis le dépose dans le buffer. Le **consommateur** prélève un message du Buffer et le place dans la ZoneC où il peut le consommer. Ce problème introduit plusieurs types de contraintes de synchronisation:

1. Un producteur ne doit pas déposer dans le buffer quand il n y a plus de place pour déposer ce qu'il a produit.
2. Un consommateur ne doit pas prélever de messages quand le buffer est vide.
3. Producteurs et consommateurs ne doivent pas accéder en même temps au buffer.

Question :

Écrire l'algorithme des processus producteurs et consommateurs en utilisant les sémaphores.

Semaphore Mutex = 1, **Plein = 0, Vide=n ;** (**2 pt)**
Message tampon[];

```
Producteur ( ){
Message m ;
Tantque Vrai faire
   P(Vide); (1 pt)
            m = creermessage() ;
            P(Mutex) ; (0.5 pt)
                  EcritureTampon(m);
            V(Mutex) ; (0.5 pt)
   V(Plein); (1 pt)
FinTantque }
```

```
Consommateur( ) {
Message m ;
Tantque Vrai faire
   P(Plein); (1 pt)
   P(Mutex) ; (0.5 pt)
     m = LectureTampon();
   V(Mutex) ; (0.5 pt)
 V(Vide); (1 pt)
Fin Tantque
}
```

Exercice 3 (8 pts) : Soit le programme suivant de synchronisation de deux threads avec l'utilisation de sémaphores.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
sem_t * my_sem1, * my_sem2;   (1 pt)
void *thread1 (void * arg) {
  while (1) {
    sem_wait(&my_sem1)          (0.5 pt)
    printf ("Je suis thread1,");
    sem_post(&my_sem2)          (0.5 pt)
      } }
```

```
void *thread2 (void * arg){
  while (1) {
sem_wait(&my_sem2)                    (0.5 pt)
   printf ("Je suis thread2,");
sem_post(&my_sem1)                    (0.5 pt)
          }}
void main (int ac, char **av){
   pthread_t th1, th2;
   sem_init (&my_sem1, 0 ,0);         (0.5 pt)
   sem_init (&my_sem2, 0 ,1);         (0.5 pt)
   pthread_create (&th1, NULL,&thread1, NULL) ;
   pthread_create (&th2, NULL,&thread2, NULL) ;
   pthread_join (&th1, NULL);
   pthread_join (&th2, NULL);
```

Question :

1. Complétez ce programme pour qu'il affiche la séquence suivante : Je suis thread1, Je suis thread2, Je suis thread1, Je suis thread2,....
2. Utiliser le programme précédent pour écrire un autre programme qui initialise une variable entière *cpt à* 1 et qui crée deux threads *p1* et *p2*. Le thread *p1* incrémente *cpt* 10000 fois et le thread *p2* décrémente *cpt* 10000 fois. Utiliser un sémaphore pour résoudre le problème de l'accès concurrent à la variable cpt.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>
sem_t semaphore; (0.5 pt)
int cpt =1; (0.5 pt)
void* p1(void* arg){
int i=0;
while (i<10000){
        sem_wait(&semaphore); (0.5 pt)
        cpt +=1;
        sem_post(&semaphore); (0.5 pt)
i++;
}
}
```

```
void* p2(void* arg){
int i=0;
while (i<10000){
        sem_wait(&semaphore); (0.5 pt)
        cpt -=1;
        sem_post(&semaphore); (0.5 pt)
i++;
}
}
int main(void)  {
  pthread_t tache1, tache2;
  sem_init(&semaphore, 0, 1); (0.5 pt)
  pthread_create (&tache1, NULL, &fonc1, NULL);
  pthread_create (&tache2, NULL, &fonc2, NULL);
  pthread_join(tache1, NULL);
  pthread_join(tache2, NULL);
  printf("cpt =%d:\n", cpt );  (0.5 pt)
  return 0;
}
```