TD n°5 : Communication et synchronisation

Objectif : Gérer les conflits d'accès aux ressources partagées

Exercice 1 – Notions de cours

1) Pourquoi le partage de ressources pose des problèmes dans un système multiprogrammé en temps partagé ?

2) Le système UNIX permet-il de contrôler les accès aux ressources partagées ?

3) Qu'est-ce qu'une section critique ?

4) Laquelle de ces affirmations sur les sémaphores est exacte ?

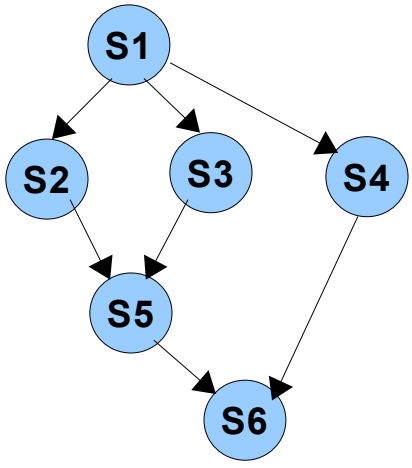
   A. Un processus peut se bloquer lorsqu'il libère un sémaphore « plein ».
   B. Un processus peut se bloquer lorsqu'il demande un sémaphore « vide ».
   C. Un processus peut se bloquer lorsqu'il crée un sémaphore.
   D. Un processus peut se bloquer lorsqu'il libère un sémaphore sur lequel un autre processus est en attente.

Exercice 2 – Sémaphores de synchronisation

On considère 6 blocs d'instructions, S1 à S6. Un graphe de préséance présente les contraintes sur l'ordre d'exécution des instructions. Par exemple, la flèche allant de S1 vers S2 indique que l'instruction S2 doit nécessairement être exécutée après S1. On place chaque bloc d'instruction dans un processus distinct P1 à P6.

Utilisez des sémaphores pour synchroniser les processus de manière à respecter les contraintes du graphe de préséance ci-dessous :

```
Process P1 { S1; }

Process P2 { S2; }

Process P3 { S3; }

Process P4 { S4; }

Process P5 { S5; }

Process P6 { S6; }
```



Exercice 3 – Sémaphores d'exclusion mutuelle

Soient trois processus concurrents P1, P2 et P3 qui partagent les variables `n` et `out`. Pour contrôler les accès aux variables partagées, un programmeur propose les codes suivants :

```
Semaphore mutex1 = 1 ;
Semaphore mutex2 = 1 ;
```

Code du processus P1 :

```
mutex1.P() ;
mutex2.P() ;
      out=out+1 ;
      n=n-1 ;
mutex2.V() ;
mutex1.V() ;
```

Code du processus P2 :

```
mutex2.P() ;
      out=out-1 ;
mutex2.V() ;
```

Code du processus P3 :

```
mutex1.P() ;
      n=n+1 ;
mutex1.V() ;
```

1) Cette proposition est-elle correcte ? Si non, pourquoi ?

2) Proposer une solution correcte.

Exercice 4 – Lecteurs/rédacteurs 

Deux villes A et B sont reliées par une seule voie de chemin de fer. Les trains peuvent circuler dans le même sens de A vers B ou de B vers A. Cependant, ils ne peuvent pas circuler dans les sens opposés.

On considère deux classes de processus : les trains allant de A vers B (*Train AversB*) et les trains allant de B vers A (*Train BversA*). Leur comportement se définit comme suit :

Train AversB :

Demande d'accès à la voie par A;
Circulation sur la voie de A vers B;
Sortie de la voie par B;

Train BversA :

Demande d'accès à la voie par B ;
Circulation sur la voie de B vers A;
Sortie de la voie par A;

1) Parmi les modèles étudiés en cours (producteur/consommateur, lecteur/rédacteur, les philosophes), à quel modèle ce problème correspond-il ?

2) En utilisant les sémaphores (opérations P et V), traduire les demandes d'accès et de sorties, de façon à ce que les processus respectent les règles de circulation sur la voie unique.

Exercice 5 – Producteurs/consommateurs

Considérons le problème producteur/consommateur vu en cours. *n* producteurs produisent des messages et les déposent dans un tampon de taille illimitée. *n* consommateurs peuvent récupérer les messages déposés dans le tampon.

```
Semaphore Mutex = 1 ;
Message tampon[];

Producteur ( )
{
      Message m ;

      Tantque Vrai faire
            m = creermessage() ;
            Mutex.P() ;
                  EcritureTampon(m);
            Mutex.V() ;
      FinTantque
}

Consommateur( )
{
      Message m ;

      Tantque Vrai faire
            Mutex.P() ;
                  m = LectureTampon();
            Mutex.V() ;
      Fin Tantque
}
```

1)Adaptez cette solution pour que l'échange des message soit synchronisé (consommateurs bloqués si le tampon est vide).

2)On considère à présent que un tampon de taille *Max* (celui-ci peut contenir au plus *Max* messages). Ajouter les modifications nécessaires pour prendre en compte cette nouvelle contrainte (producteurs bloqués si le tampon est plein).