Synchronisation par sémaphores

Etudier les problèmes suivants extrait du livre « Synchronisation par sémaphores » de Mohammed Saïd HABET.

Exercice 1 Exclusion mutuelle

1.1 Exclusion mutuelle

Considérons n processus $P_1, P_2, \ldots\ldots Pn$ s'exécutant en parallèle et utilisant une ressource critique R comme suit :

```
Processus Pi    /* i=1,n */
début
  <section non critique>
  <utilisation de R>
  <section restante>
fin
```

On doit garantir que l'utilisation de R se fasse en exclusion mutuelle : la section <utilisation de R> doit être une section critique (S.C).

Les conditions de l'exclusion mutuelle sont les suivantes :

a) à tout instant un processus au plus peut se trouver en section critique

b) un processus qui exécute du code hors section critique ne doit pas empêcher d'autres processus d'accéder en section critique

c) si plusieurs processus sont bloqués en attente d'entrer en section critique alors qu'aucun autre ne s'y trouve , l'un d'eux doit y accéder au bout d'un temps fini.

d) toute demande d'entrée en section critique doit être satisfaite.

Pour ce faire on peut utiliser un sémaphore `mutex` et exprimer les algorithmes des processus `Pi` ainsi :

```
var mutex : sémaphore init 1;
     /* mutex est une variable globale de type sémaphore */
     /* dont la valeur est initialisée à 1 */
Processus Pi    /* i=1,n */
début
  <section non critique>
  P(mutex)
  <utilisation de R>  /* section critique */
  V(mutex)
  <section restante>
fin
```

Questions : Modéliser et vérifier que le programme garanti la propriété d'exclusion mutuelle.

Exercice 4 :

Soit P_0 et P_1 deux processus parallèles se partageant deux ressources R_1 et R_2. Les algorithmes de ces deux processus sont écrits comme suit :

```
var s1,s2 : sémaphore init 1,1 ;
Processus P0                          Processus P1
Début                                 Début
 a0: (1) P(s1)                         a1: (1') P(s2)
     (2) utiliser R1                       (2') utiliser R2
     (3) P(s2)                             (3') P(s1)
         utiliser R1 et R2                     utiliser R1 et R2
         V(s1)                                 V(s2)
         V(s2)                                 V(s1)
         Aller à a0                            Aller à a1
Fin                                   Fin
```

1) à quelle situation anormale peut conduire l'exécution de ces deux processus ?

2) donner une solution à ce problème.

Question : Faire l'exercice et vérifier votre solution.

Exercice 2 Précédence de tache

1.2 Précédence des tâches

Dans quelques applications , notamment les applications temps réel , certaines tâches (processus) doivent avoir une relation de précédence : une tâche ne peut entamer son exécution que lorsque d'autres tâches ont terminé leurs exécution.

Exemple :

Considérons un système comportant deux tâches T_1 et T_2 et où T_1 doit précéder T_2

La synchronisation peut être réalisée simplement comme suit :

```
var s : sémaphore init 0;
     /* s est une variable globale de type sémaphore */
     /* dont la valeur est initialisée à 0 */
Processus T1              Processus T2
début                     début
  <Exécution>               P(s)
  V(s)                      <Exécution>
fin                       fin
```

Question : Vérifier la propriété de précédence par l'utilisation d'un testeur (système modélisant la propriété)

Exercice 3 :

On considère un ensemble de six tâches séquentielles {A, B, C, D, E ,F}.

La tâche A doit précéder les tâches B, C, D. Les tâches B et C doivent précéder la tâche E.

Les tâches D et E doivent précéder la tâche F.

Réaliser la synchronisation de ces tâches en utilisant les sémaphores.

Question : Faire l'exercice et vérifier votre solution.

Exercice 3. Lecteurs/Rédacteurs

1.3 Lecteurs / Rédacteurs

On considère un objet (un fichier par exemple) qui n'est accessible que par deux catégories d'opérations : les lectures et les écritures. Plusieurs lectures (consultations) peuvent avoir lieu simultanément ; par contre les écritures (mises à jour) doivent se faire en exclusion mutuelle. On appellera « lecteur » un processus faisant des lectures et « rédacteur » un processus faisant des écritures.

Il s'agit donc de réaliser la synchronisation entre lecteurs et rédacteurs en respectant les contraintes suivantes :

- exclusion mutuelle entre lecteurs et rédacteurs : si un lecteur demande à lire et qu'il y a une écriture en cours , la demande est mise en attente. De même que si un rédacteur demande à écrire et qu'il y a au moins une lecture en cours , la demande est mise en attente.
- exclusion mutuelle entre rédacteurs : si un rédacteur demande à écrire et qu'il y a une écriture en cours , la demande est mise en attente.

L'attente d'un processus lecteur / rédacteur peut être assimilée au blocage du processus dans une file de sémaphore.

Pour satisfaire les contraintes ci-dessus , on peut procéder comme suit :

```
var s,mutex : sémaphore init 1,1 ;
    nl : entier init 0 ;
```

```
Processus Lecteur
Début
  .
  .
  .
  P(mutex)
  nl := nl + 1
  si nl=1 alors P(s) finsi
  V(mutex)
  Lecture
  P(mutex)
  nl := nl - 1
  si nl=0 alors V(s) finsi
  V(mutex)
  .
```

```
Processus Rédacteur
Début
  .
  .
  .
  P(s)
  Ecriture
  V(s)
  .
  .
  .
Fin
```

Exercice 5. Producteurs / Consommateurs

1.4 Producteurs / Consommateurs

On considère deux classes de processus :

- les producteurs : produisent des informations ,
- les consommateurs : consomment les informations produites par les producteurs.

Pour que les producteurs et consommateurs puissent s'exécuter en parallèle , ils partagent un tampon dans lequel seront stockées les informations (messages) produites et en attente d'être consommées.

La synchronisation peut s'exprimer comme suit :

```
var S1,S2,mutex : sémaphore init N,0,1 ; /* N : taille du tampon */
Processus Producteur                    Processus Consommateur
Début                                   Début
  Cycle /* répeter  indéfiniment */       Cycle
   <Produire un message>                   P(S2)
   P(S1)                                   P(mutex)
   P(mutex)                                <Prelever un message>
   <Déposer le message>                    V(mutex)
   v(mutex)                                V(S1)
   V(S2)                                   <Consommer le message>
  FinCycle                                FinCycle
Fin                                     Fin
```

Exercice 5. Les philosophes

1.5 Philosophes

Cinq philosophes sont assis sur des chaises autour d'une table ronde pour philosopher et manger des spaghettis.Sur la table sont disposées cinq assiettes , cinq fourchettes et un plat de spaghettis qui est toujours plein.

Chaque philosophe passe son temps à penser puis manger. Pour manger il doit utiliser les deux fourchettes situées de par et d'autres de son assiette. Après avoir mangé, le philosophe repose les deux fourchettes sur la table et se remet à penser. Et ainsi de suite.

C'est-à-dire que le schéma d'un philosophe est :

```
Philosophe i (i=0,4)
Début
  Cycle
     Penser
     Manger   /* nécessite deux fourchettes */
  FinCycle
Fin
```

Questions :

1. Proposer un modèle où chaque philosophe voulant manger prend la fourchette gauche puis la fourchette droite
2. Même question sauf que un des philosophes commence par celle de droite
3. Même question sauf que nous interdisons « par sémaphore » que tous les philosophes souhaitent manger en même temps.