Systèmes Concurrents

1h45 heures, documents autorisés

janvier 2011

Les exercices sont indépendants.

1 Questions de cours

1. Dans les moniteurs, à quoi sert l'exclusion mutuelle, pourquoi ne pas se contenter uniquement des variables conditions ?
2. Peut-on résoudre tout problème de synchronisation avec un moniteur qui n'a droit qu'à une seule variable condition ?
3. Soit la solution au problème des lecteurs/rédacteurs, telle que présentée en cours :

variable d'état : nblect (init=0)
sémahores : accès (init=1), lect (init=1)

DL *(demander_ lecture)*	TL *(terminer_ lecture)*
lect.P() si nblect = 0 alors accès.P() finsi nblect++ lect.V()	lect.P() nblect−− si nbLect = 0 alors accès.V() finsi lect.V()
DE *(demander_ écriture)*	TE *(terminer_ écriture)*
accès.P()	accès.V()

On considère l'arrivée de 7 processus appelant (dans cet ordre) :
DE, DL, DL, DL, DE, DL, DE.

- Le premier processus effectue ensuite TE. Quelle(s) est(sont) la(les) requête(s) débloquée(s) ?
- Même question si la stratégie était FIFO ?

⟶

2 Sémaphore

On considère une usine divisée en N ateliers. Les ouvriers rentrent et sortent de l'usine en suivant le code suivant, où `j` est un numéro d'atelier :

```
demander_accès(j);
aller_dans_l_atelier(j); travailler(); quitter_l_atelier_et_l_usine(j);
indiquer_sortie(j);
```

Les opérations `demander_accès` et `indiquer_sortie` sont les opérations de synchronisation, les autres représentent les actions effectives et ne concernent pas la synchronisation.

Le système doit assurer :

- il n'y a jamais (strictement) plus de 100 ouvriers dans l'usine ;
- il n'y a jamais (strictement) plus de 20 ouvriers dans un atelier donné ;
- N est arbitraire et peut aussi bien être inférieur que supérieur à 5 ;
- hors les deux premières contraintes, on souhaite empêcher le moins possible l'entrée d'un ouvrier.

On propose la solution suivante à base de sémaphores :

```
Sémaphore usine = 100; // valeur initiale du sémaphore
Sémaphore[N] atelier = { 20, ..., 20 } // tableau de N sémaphores initialisés à 20

procédure demander_accès(j)
   usine.P();
   atelier[j].P();

procédure indiquer_sortie(j)
   usine.V();
   atelier[j].V();
```

Questions :

1. Cette solution est-elle juste (justifier !) ?
2. Peut-elle empêcher à tort l'entrée d'un ouvrier ?
3. Présente-t-elle un risque d'interblocage ?
4. Présente-t-elle un risque de famine ?

3 Moniteur

Appliquer la méthodologie de développement pour construire un moniteur résolvant le problème ci-dessus, en déroulant clairement les sept étapes.

4 Sémaphore

On souhaite implanter une barrière pour N processus. Une barrière est un point d'attente qui est bloquant tant que les N processus ne l'ont pas atteint et qui devient passant quand les N sont là.

1. Barrière non réinitialisée. La barrière ne sert qu'une seule fois. Une fois levée, elle le reste donc. Le code d'un processus utilisateur est :

   ```
   processus travail(i) // i ∈ [1,N] = l'identification du processus
      code applicatif
      code de synchronisation pour franchir la barrière
      code applicatif
   ```

   L'implantation se fait en utilisant un processus coordinateur auxiliaire, dont le rôle est de débloquer les N processus quand ils sont tous là :

   ```
   processus coordinateur
     ...
   ```

   Donner le code de synchronisation nécessaire (dans `travail` et dans `coordinateur`) en utilisant des sémaphores. Penser à préciser la valeur initiale du/des sémaphore(s). Indication : le nombre de sémaphores nécessaires ne dépend pas de N.

2. Barrière réinitialisable : la barrière peut servir plusieurs fois, elle est rabaissée après le déblocage des N processus. Le code d'un processus utilisateur est (noter la boucle) :

   ```
   processus travail(i) // i ∈ [1,N] = l'identification du processus
      boucle
         code applicatif
         code de synchronisation pour franchir la barrière
         code applicatif
      finboucle
   ```

   Comme précédemment, on utilise un processus coordinateur.

   (a) Le code réalisant la barrière non réinitialisée fonctionne-t-il ? (justifier !)

   (b) Si non, donner le code de synchronisation nécessaire (dans `travail` et dans `coordinateur`) en utilisant des sémaphores. Penser à préciser la valeur initiale des sémaphores. Indication : le nombre de sémaphores nécessaires dépend de N.

→

5 Transactions

Soit quatre transactions portant sur trois variables (x, y, z) et ainsi entrelacées :

Temps	T1	T2	T3	T4
1	début			
2		début		
3			début	
4				début
5	écrire x			
6		lire y		
7			lire x	
8			lire y	
9		écrire x		
10				écrire y
11				lire z
12	écrire z			
13			fin	
14		fin		
15	fin			
16				fin

1. Construire le graphe de dépendance ;
2. En déduire si l'exécution précédente est sérialisable ;
3. On utilise l'algorithme de certification présenté dans le cours. Quelle(s) transaction(s) valide(nt) effectivement et quelle(s) transaction(s) abandonne(nt) ?

Barème envisagé : 1 : 3pt ; 2 : 4pt ; 3 : 5pt ; 4 : 4pt ; 5 : 4pt.