Université Paris Diderot — Année 2014–2015

Examen Théorie et Pratique de la Concurrence

Mercredi 21 Mai 2014. Durée : 2 heures.
Documents autorisés : deux feuilles manuscrites A4 recto-verso

Ce sujet comporte quatre exercices.

Exercice 1 — *Algorithme d'exclusion mutuelle pour deux processus*

```
integer NP := 0      // variables partagées
integer NQ := 0

-- Processus P                          -- Processus Q
loop forever :                          loop forever :
p1: Section Non Critique                q1: Section Non Critique
p2: NP := NQ+1                          q2: NQ:=NP+1
p3: while(NQ!=0 and NP>NQ){}            q3: while(NP!=0 and NQ>=NP){}
p4: Section Critique                    q4: Section Critique
p5: NP:=0                               q5: NQ:=0
```

FIGURE 1 – Algorithme d'exclusion mutuelle

On considère l'algorithme d'exclusion mutuelle pour deux processus donné à la Figure 1.

1. Quelle forme ont les états du diagramme d'états de cet algorithme ? Dessinez le début de ce diagramme d'états (ne dessinez pas plus que 10 états).
2. Les valeurs prises par les variables entières NP et NQ sont-elles bornées ? Quelle conséquence cela a-t-il sur le diagramme d'états ?
3. Cet algorithme respecte-t-il la propriété d'exclusion mutuelle ? Justifiez votre réponse.
4. Cet algorithme respecte-t-il la propriété d'absence d'interblocage ? Justifiez votre réponse.

Exercice 2 — *Algorithme du Filtre*

On considère l'algorithme du Filtre donné à la Figure 2 pour assurer l'exclusion mutuelle entre n processus. Dans cet algorithme l'instruction `while((there exists k!=i such that Level[k] >=i) and victim[j]==i){ }` signifie que si il existe un indice k différent de i tel que la valeur contenue dans `Level[k]` est plus grande ou égale à j et si `Victim[j]==i` alors on continue la boucle.

1. Donnez une expression $Nblevel(j)$ pour tout j entre 1 et n caractérisant à tout instant de l'exécution du programme le nombre de Processus Pi tel que $Level[i] = j$.
2. Déduire de la question précédente que l'algorithme du Filtre respecte la propriété d'exclusion mutuelle.
3. L'algorithme du Filtre respecte-t-il la propriété d'absence d'interblocage ? Justifiez votre réponse.

4. On suprime de l'algorithme l'instruction s4 et on remplace l'instruction de boucle s5 par `while(there exists k !=i such that level[k] >=j)`.
   (a) L'algorithme ainsi obtenu respecte-t-il toujours la propriété d'exclusion mutuelle ? Justifiez votre réponse.
   (b) Qu'en est-il pour l'absence d'interblocage ? Justifiez votre réponse.

```
--Variables partagees
boolean Level[1..n] := [0;...;0]
boolean Victim[1..n]:= [0;...;0]

-- Processus Pi (pour i allant de 1 a n)
loop forever :
s1: Section Non Critique
s2: for(int j=1;j <=n; j++){
s3:   Level[i]=j;
s4:   Victim[j]=i;
s5:   while((there exists k !=i such that Level[k] >=j) and Victim[j]==i){}
s6: }
s7: Section Critique
s8: level[i]=0;
```

FIGURE 2 – Algorithme du Filtre

Exercice 3 *Modification d'algorithme*

On considère l'algorithme d'exclusion mutuelle par passage de message décrit aux Figures 3 et 4. Dans cet algorithme, on trouve un contrôleur et n processus clients. Le contrôleur dispose d'un ensemble d'entiers Q comme structure données sur lequel il peut faire les opérations suivantes : insérer un nouvel entier dans l'ensemble (avec la méthode `insert(Q,j)`) et retirer un entier de l'ensemble lorsque celui-ci est non vide (avec la méthode `takeOut(Q)` qui renvoie l'entier retiré). Nous supposons que aucun message n'est perdu, que les messages arrivent dans l'ordre dans lequel ils sont envoyés, et que tout message émis finit par arriver à son destinateur. Pour rappel, l'opération `Send(ok) -> i` où i est un entier signifie "envoie le message ok au processus i" et si i n'est pas un entier mais est le contrôleur, le message est envoyé au contrôleur.

1. Expliquez en quelques lignes pourquoi cet algorithme respecte la propriété d'exclusion mutuelle.
2. Est-ce-que cet algorithme respecte la propriété d'absence de famine si l'ensemble Q est quelconque, c'est-à-dire si il n'y a aucune règle spécifique sur l'ordre dont les entiers sont insérés et retirés de cet ensemble ? Si la réponse est négative, quelles restrictions peuvent être imposées sur cet ensemble de façon à ce que l'absence de famine soit respectée ?
3. Modifiez cet algorithme de façon à autoriser non pas un seul processus mais p (avec $p \leq n$) en section critique.

4. On souhaite maintenant distinguer deux types de client. Les client de type `R` et les clients de type `W` avec les propriétés suivantes. Il ne peut pas y avoir un client de type `R` et un client de type `W` en même temps en section critique ; au même instant il y a au plus un client de type `W` en section critique et au plus p clients de type `R`. Modifiez l'algorithme précédent par passage de message de façon à obtenir un algorithme pour ce nouveau problème (vous pouvez bien entendu utiliser des nouveaux types de message et des nouvelles variables).

Exercice 4 *Conception d'algorithme concurrent*

Dans une ville , un coiffeur possède un salon ayant une porte d'entrée, une porte de sortie, un fauteuil de coiffure et N chaises. Les clients arrivent par la porte d'entrée et sortent par la porte de sortie après avoir eu leur coupe de cheveux. Comme le salon est petit, uniquement le client sur le fauteuil de coiffure peut-être servi à un moment donné par le coiffeur. Le coiffeur passe sa vie entre dormir et servir ses clients. Quand il n'a aucun client, le coiffeur dort. Quand un client arrive et le fauteuil est libre, il s'assoit dans le fauteuil et il réveille le coiffeur. Si le fauteuil n'est pas libre, le client occupe une chaise s'il y a des chaises de libre ou il attend qu'une chaise se libère sinon. Un client sur une chaise attend que le fauteuil se libère. Après avoir fini une coupe, le coiffeur fait sortir le client servi et s'endort.

Modélisez ce problème en utilisant des sémaphores pour la synchronisation entre le coiffeur et ses clients. Vous pouvez pour cela soit proposé un code en `Java` ou en `C` ou encore utilisé un langage abstrait comme celui vu en cours. Les informations suivantes doivent être visibles dans votre réponse :

1. un processus `Client` dans lequel sont identifiés clairement ses états : en attente d'une chaise, sur une chaise en attente du fauteuil, sur le fauteuil en attente de la fin de sa coupe et servi ;
2. un processus `Coiffeur` avec les états endormi et en service ;
3. les déclarations de sémaphore et leur valeur initiale.

```
-- Processus Controleur
boolean InCS := False
integer Waiting :=0
integer Current:=0
set Q:=empty

loop forever :
c1: while(Waiting==0 and InCS==True){}
c2: Current:=TakeOut(Q);
c3: InCS:=True;
c4: Waiting:=Waiting-1;
c4: Send(ok)->Current;
```

```
-- Gestion des messages de Controleur
case (request,j)  :
  Insert(Q,j);
  Waiting=Waiting+1;

case (out,j) :
  InCS:=False;
```

FIGURE 3 – Controleur

```
-- Processus client Pi (pour i allant de 1 a n)
boolean CS:=False

loop forever :
s1: Section Non Critique
s2: send(request,i) -> Controleur;
s3: while(CS==False){}
s4: Section Critique
s5: send(out,i) -> Controleur;
s6: CS:=False;
```

```
--Gestion de messages de Pi
case (ok)  :
  CS:=True;
```

FIGURE 4 – Processus Client