EXAMEN — Théorie et Pratique de la Concurrence — 24/05/2017 12^{00}-14^{30}

Les seuls documents autorisés sont deux feuilles manuscrites A4 recto-verso.
Ce sujet comporte quatre exercices.

Exercice 1 *Algorithme d'exclusion mutuelle pour trois processus*

```
            int A=0;      // variables partagées

-- Processus P
    while(true){
p1:  Section Non Critique
p2:  while(A!=0){}
p3:  Section Critique
p4:  A=1;
    }

-- Processus Q
    while(true){
q1:  Section Non Critique
q2:  while(A!=1){}
q3:  Section Critique
q4:  A=2;
    }

-- Processus R
    while(true){
r1:  Section Non Critique
r2:  while(A!=2){}
r3:  Section Critique
r4:  A=0;
    }
```

FIGURE 1 – Algorithme d'exclusion mutuelle

On considère l'algorithme d'exclusion mutuelle pour trois processus proposé à la Figure 1 avec les même hypothèses que celles du cours à savoir la section critique termine toujours et il y a équité entre les processus.

1. Dessinez le début du diagramme d'états correspondant à cet algorithme (maximum 10 états).
2. Cet algorithme respecte-t-il l'exclusion mutuelle ? Justifiez votre réponse.
3. Cet algorithme respecte-t-il l'absence d'interblocage ? Justifiez votre réponse.
4. Cet algorithme respecte-t-il l'absence de famine ? Justifiez votre réponse.
5. Qu'en est-il des réponses aux questions précédentes si on suppose de plus que les section non critiques terminent ?

Exercice 2 *Encore un algorithme d'exclusion mutuelle*

On considère l'algorithme d'exclusion mutuelle pour deux processus présenté à la Figure 2.

1. Cet algorithme garantit-il l'exclusion mutuelle ? Justifiez votre réponse.
2. Cet algorithme garantit-il l'absence d'interblocage ? Justifiez votre réponse.
3. Que se passe-t-il par rapport aux deux propriétés précédentes si l'on remplace dans la function `myTurn` la ligne
   `ret=((req[i] <req[1-i]) || req[1-i]==0);`
   par la ligne `ret=((req[i] <= req[1-i]) ||req[1-i]==0);` ?

```
int d=1; // Variables Partagées
int req[2]={0;0};

boolean myTurn(int i){
  boolean ret=true;
  ret=((req[i] <req[1-i]) || req[1-i]==0);
  return ret;
}
```

```
-- Processus P0
    while(true){
p1:  req[0]=d;
p2:  d=d+1;
p3:  while(myTurn(0)!=true){}
p4:  Section Critique
p5:  req[0]=0;
    }
```

```
-- Processus P1
    while(true){
q1:  req[1]=d;
q2:  d=d+1;
q3:  while(myTurn(1)!=true){}
q4:  Section Critique
q5:  req[1]=0;
    }
```

FIGURE 2 – Algorithme d'exclusion mutuelle

4. Comment modifier la function `myTurn` pour garantir à la fois l'exclusion mutuelle et l'absence d'interblocage ? Justifiez votre réponse.

Exercice 3 *Le problème de la discothèque*

On considère une discothèque passant deux types de musique, par exemple rock et hip-hop. Afin d'éviter les conflits entre clients, cette discothèque adopte le fonctionnement suivant en boucle. Elle commence par passer du rock et ne laisse rentrer que les adeptes de rock, lorsqu'elle est remplie, elle ne laisse plus rentrer personne, elle change de style musical, elle attend de se vider et laisse rentrer ensuite les adeptes de hip-hop. Lorsqu'elle est pleine, elle ne laisse plus rentrer personne, elle change de style musical, elle attend de se vider et recommence avec du rock. Les adeptes de rock et de hip-hop, quant à eux, font en boucle les actions suivantes : attendre de pouvoir rentrer dans la discothèque, rentrer dans la discothèque, sortir de la discothèque. On ne veut pas qu'il y aient d'adeptes de rock et de hip-hop en même temps dans la discothèque. On supposera que la capacité de la discothèque est 10. Proposez en Java, en C ou en pseudo-code, une modélisation de ce système en prenant en compte que l'on souhaite un processus par adepte de rock, un processus par adepte de hip-hop et un processus pour la discothèque. Vous pouvez utiliser les variables partagées que vous souhaitez mais il vous est demandé de faire attention à leur manipulation.

Exercice 4 *Algorithme par passage de messages*

On considère l'algorithme d'exclusion mutuelle pour trois processus proposé à la Figure 3.

1. Donnez une brève explication du fonctionnement de cet algorithme.
2. Cet algorithme respecte-t-il l'exclusion mutuelle ? Justifiez votre réponse.
3. Cet algorithme respecte-t-il l'absence d'interblocage ? Justifiez votre réponse.
4. Proposez des solutions pour éviter (même de façon partielle) les problèmes évoqués précédemment. On ne cherchera pas forcément à résoudre les problèmes de famine.

```
-- Processus Pi pour i valant 0, 1 ou 2
    boolean access=False;
    int nack:=0;

    while(true){
p1:   Section Non Critique
p2:   access=True
p3:   send(req,i)->P(i+1 mod 3);
p4:   send(req,i)->P(i+2 mod 3);
p5:   while(nack!=2){}
p6:   Section Critique
p7:   access=false;
p8:   nack=0;
p9:   send(out,i)->P(i+1 mod 3);
p10:  send(out,i)->P(i+2 mod 3);
    }

-- Gestion des messages de Pi pour i valant 0,1 ou 2
case (req,j) :
    if(access==False){
       send(ok)->P(j);
    }
case (ok) :
    nack=nack+1;
case (out,j) :
   if(access==True){
       send(req,i)->P(j);
    }
```

FIGURE 3 – Code du processus i