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

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

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

1. Quelle forme ont les états du diagramme d'états correspondant à ce système? Dessinez les premiers états du diagramme (il est inutile de dessiner plus de 10 états).
2. Expliquez pourquoi cet algorithme ne vérifie pas la propriété d'exclusion mutuelle.

Exercice 2*Encore un algorithme d'exclusion mutuelle*

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

1. Cet algorithme respecte-t-il la propriété d'exclusion mutuelle? Justifiez votre réponse.

Exercice 3*Premier arrivé, premier servi*

On considère l'algorithme présenté à la Figure 3 pour 3 processus partageant le tableau de taille 3 `tab` et pouvant utiliser la fonction `max`.

1. Expliquez brièvement le fonctionnement de cet algorithme.
2. Au cours de l'exécution de cet algorithme, quelle est la valeur maximale qui peut apparaître dans le tableau `tab`?
3. Cet algorithme respecte-t-il l'absence d'interblocage? Justifiez votre réponse.
4. Que cela change-t-il si on change la ligne p4 en `while(i!=j && tab[j]!=0 && tab[j]<=tab[i]){}? Y voyez-vous un souci?`

Exercice 4*Modification d'algorithme*

On considère l'algorithme d'exclusion mutuelle par passage de messages décrit aux Figures 4 et 5. 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 j 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 qu'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 destinataire. 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 clients 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 5

On considère le système suivant fait de trois types de processus P , Q et R et de deux variables partagées A et B . Les accès aux variables A et B se font en exclusion mutuelle (c'est à dire qu'il ne peut pas y avoir deux processus manipulant la variable A en même temps, ni deux processus manipulant la variable B en même temps). Pour chacun des processus, le comportement est le suivant :

1. Un processus de type P prend l'accès à la variable A , exécute sa tâche (en faisant `modify(A)` par exemple), libère la variable et recommence.
2. Un processus de type Q demande l'accès aux variables A et B , exécute sa tâche (en faisant `modify(A)` et `modify(B)` par exemple), libère les variables et recommence.
3. Un processus de type R attend à chaque fois avant de commencer qu'un processus de type P ET qu'un processus de type Q aient exécuté leur tâche pour s'exécuter, il prend ensuite l'accès à la variable B , exécute sa tâche (en faisant `modify(B)` par exemple), libère la variable et retourne dans son état initial où il attend de nouveau qu'un processus de type P et qu'un processus de type Q aient exécuté leur tâche.

Le nombre de processus de type P , Q et R n'est pas borné.

Proposez une implémentation pour les processus P , Q et R en utilisant des sémaphores pour la synchronisation entre les différents processus. Vous pouvez pour cela soit proposer un code en Java ou en C ou encore utiliser un langage abstrait comme celui vu en cours. Faites attention à la façon dont les accès aux variables A et B sont faits de façon à maximiser la concurrence.

```

        boolean B1 = false    // variables partagées
        boolean B2 = false
        int k = 1

-- Processus P
while (true)
p1: Section Non Critique
p2: B1=true;
p3: while (k!=1){
p4:   while (B2==true){}
p5:   k=1;
    }
p6: Section Critique
p7: B1=false;
}

-- Processus Q
while(true){
q1: Section Non Critique
q2: B2=true;
q3: while(k!=2){
q4:   while(B1==true){}
q5:   k=2;
    }
q6: Section Critique
q7: B2=false;
}
    
```

FIGURE 1 – Algorithme de Hyman

```

        int V1= 0 // variables partagées
        int V2= 0

-- Processus P
while(true){
p1 : Section Non Critique
p2 : V1=1;
p3 : if (V2!=0){
p4 : goto p2;}
p5 : else{V2=1;}
p6 : if (V1!=1){
p7 : if (V2!=1){
p8 : goto p2;}}
p9 : Section Critique
p10: V2=0;
}

-- Processus Q
while(true){
q1 : Section Non Critique
q2 : V1=2;
q3 : if (V2!=0){
q4 : goto q2;}
q5 : else{V2=2;}
q6 : if (V1!=2){
q7 : if (V2!=2){
q8 : goto q2;}}
q9 : Section Critique
q10: V2=0;
}

```

FIGURE 2 – Algorithme d'exclusion mutuelle

```

int []tab=[0;0;0] \variable partagée

int max(){
    int m=0;
    for(int i=0;i<3;i++){
        if(tab[i]>m){
            m=tab[i];
        }
    }
    return m;
}

-- Processus Pi (i allant de 0 à 2)
int l=0;
while(true){
p1 : l=max();
p2 : tab[i]=l+1;
p3 : for(int j=0;j<3;j++){
p4 : while(i!=j && tab[j]!=0 && tab[j]<tab[i]){}
    }
p5 : Section Critique
p6 : tab[i]=0;
}

```

FIGURE 3 – Algorithme d'exclusion mutuelle pour 3 processus

```

-- 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 4 – 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 5 – Processus Client