Université Aboubekr BELKAID Tlemcen
Faculté des Sciences - Tidjani HAddam
Département d'informatique

Classe LMD S5 Informatique
Année Universitaire : 2020- 2021

Solution TD N° 3 : Exclusion Mutuelle

Exercice N° 1

Dans un système multi-processus, possède une instruction **swap (reg[i], m)** dans l'effet de permuter le contenu du registre reg[i] relatif au processus Pi avec le contenu de l'espace mémoire m en bloquant l'accès a cette emplacement durant l'opération de permutation.

Vous avez le programme suivant :

```
Contexte commun  var m :0..1  ;
                 m := 0 ; (initialisation)

processus Pi
        debut
                reg [i] :=1 ;
                    répéter debut
                                swap (reg[i], m);

                                jusqu'a  (reg[i]  =  0)
                            fin

                <section critique>

                swap  (reg[i],  m);
        fin
```

Q1 : vérifier si la contrainte d'exclusion mutuelle et l'absence de famine sont satisfaites dans ce programme.

1

Exercice N° 2

Contexte commun var m :0..1 ;
m := 0 ; (initialisation)

Processus Pi
Debut

Repéter debut
Add (reg [i], m)

Jusqu'à (reg [i] = -1)
Fin

<section critique>
m:= 0;
fin

L'instruction Add (reg[i], m) à pour effet décrémenter de 1 le contenu du mot mémoire m puis de copier le résultat dans le registre reg [i]. la fonction Add est définie comme suite :

Add (reg[i], m)
debut
m := m-1 ;
* ←**Interruption**
reg [i] := m;
fin

Q1: Vérifier si l'exclusion mutuelle et l'absence de l'interblocage sont satisfaites dans ce programme
Q2 : Que se passe t-il si l'instruction Add n'est pas exécutée de manière indivisible (avec interruption au niveau *) ?
Q3 : Est ce qu'il y a un problème de famine ?

Solution TD N°3 exclusion mutuelle :

Solution exercice N°1

La fonction swap (reg[i], m)

{

z=reg[i];

reg[i] =m;

m=z;

}

m=0 initialisation

Vérification l'exclusion mutuelle ?

1) P0 arrive, p0 met reg [0]=1

P0 test la boucle répéter, p0 exécute swap (reg [0]=1, m=0)

⇨ Reg[0]=0 et m=1

P0 test la condition jusqu'a, p0 trouve reg [0]=0 vrai

⇨ p0 dort de la boucle répéter, p0 entre dans la section critique.

2) pendant que p0 dans la section critique

P1 arrive, p1 met reg [1]=1

P1 exécute la boucle répéter

P1 exécute la fonction swap (reg [1]=1, m=1)

⇨ reg [1]=1 et m=1

P1 test la condition jusqu'à, p1 trouve reg [1]=1≠0 condition fausse => p1 boucle par l'attente active.

3) p0 sort de la section critique

P0 execute swap (reg[0]=1,m=1)=> reg[0]=1, m=0

P1 entraine exécute en boucle infinie la boucle répéter,

P1 exécute swap (reg [1]=1, m=0) => reg [1]=0 et m=1

P1 trouve la condition jusqu'à reg [1]=0 => p1 entre dans la section critique

⇨ exclusion mutuelle est satisfaite

2) vérification l'absence de famine ?

1) p0 arrive, p0 entre dans la section critique, car reg [0]=0 et m=1

2) P1 arrive, p1 boucle car reg[1]=1≠0

Swap (reg[i], m)

{

z=reg[i] **(int1)**

reg[i]=m (**int2)**

m=z **(int3)**

}

3) Nous avons 3 positions pour l'interruption de processus p1

P0 sort de la section critique, p0 exécute swap (reg [0]=0, m=1)

P0 met reg [0]=1 et m=0

4) p1 trouve m=0, p1 exécute swap (reg [1], m)

P1 peut entrer dans la section critique, mais il est interrompu par p0

P0 redemande une deuxième fois d'entrer dans la section critique, si p0 entre une deuxième fois dans la section avant p1 alors on dit qu'il ya un problème de famine, c'est-à-dire p0 monopolise l'utilisation de la section critique.

Cas1 : interruption de p1 au niveau int1

Swap (reg [1]=1, m=0) {z=reg [1]=1 interruption de p1 a int1

P0 demande une deuxième fois d'entrer dans la section critique, p0 arrive, p0 met reg [0]=1

P0 exécute la boucle répéter, p0 exécute swap (reg [0]=1, m=0)

=>reg [0]=0 et m=1

P0 trouve reg [0]=0=> p0 entre dans la section critique

Donc p1 reprend son exécution âpres int 1

P1 met reg [1]=1 et m=1

- ⇨ p1 trouve reg [1]=1≠0 => p1 boucle
- ⇨ et p0 dans la section critique une deuxième fois=> il y a un problème de famine s'il y a une interruption en int1.

Ca2 : interruption au niveau int2

Swap (reg [1]=1, m=0)

{z= reg [1]

Reg [1]=m interruptions int 2***

P0 interrompe le p1 au niveau int2

P1 met z= reg [1]=1

Reg [1]=m=0 (m a été changé par p0 lorsqu'il est sortie de la section critique) interruption de p1 au niveau int2

P0 demande une deuxième fois d'entrer dans la section critique

P0 met reg [0] =1

P0 exécute répéter

P0 exécute swap (reg [0]=1, m=0) , p0 met reg[0]=0 et m=1

- ⇨ p0 trouve reg [0]=0 vrai => p0 entre dans la section critique

p1 reprend son exécution a partir de int2

p1 met m=1 ; p1test la condition jusqu'à : p1 trouve reg [1]=0 (condition vrai)

p1 entre dans la section critique avec p0 => il ya un problème d'exclusion mutuelle, p0 et p1 les deux en même temps dans la section s'il y a une interruption au niveau int2

ca3 : interruption au niveau int3

p0 sort de la section critique

p0 execute swap(reg[0]=0,m=1)

p0 met reg [0]=1et m=0

p1 execute repeter, p1 execute swap(reg[1]=1,m=0)=>

p1 met z=reg [1]=1, reg [1]=m=0 et m=z=1 interruption int3

p0 demande une deuxième fois d'entrer dans la section critique

p0 met reg [0]=1

p0 test la boucle répéter, p0 exécute swap (reg[0]=1,m=1)=> reg[0]=1 et m=1

⇨ p0 trouve reg [0]=1=> p0 boucle

p1 reprend son exécution a partir int3

p1 trouve reg [1]=0 (vrai)

p1 entre dans la section critique et p0 boucle

⇨ il n y a pas un problème de famine s'il y a une interruption au niveau int3

solution exercice N°2

m=0

add(reg[i],m)

{

M=m-1

Reg[i]=m

}

Pi

{

Repeter {

Add (reg[i],m)

Jusqu'a (reg[i]= -1)

}

<**Section critique >**

m=0

}

Vérification de l'exclusion mutuelle ?

1) P0 arrive, p0 test la boucle repeter

P0 execute add (reg[0],m=0)

P0 met m=m-1=0-1=-1 et reg [0]=-1

P0 trouve reg [0]=-1 vrai=> p0 entre dans la section critique

2) pendant que p0 dans la section critique, p1 arrive, p1 test la boucle répéter,

p1 exécute add(reg[1],m=1), p1 met m=m-1=-1-1=-2 et reg[1]=-2

p1 trouve reg [1]=-2≠-1=> p1 boucle

3) p0 sort de la section critique, p0 met m=0

P1 exécute add(reg[1],m=0) p1 met m=m-1=0-1=-1 et reg[1]=-1

P1 trouve reg [1]=-1(vrai) => p1 entre dans la section critique

⇨ exclusion mutuelle est satisfaite

Q2 : vérification de l'absence d'interblocage

Nous utilisons la méthode interruption par interruption

1)p0 arrive, p0 test la boucle répéter
p0 exécute add(reg[0],m=0), p0 met m=0-1=-1 interruption de p0
2)p1 arrive, p1 test la boucle répète
p1 exécute add(reg[1],m=1), p1 met m=-1-1=-2 interruption de p1
3)p0 reprend son exécution au niveau interruption int
p0 met reg [0]=-2 interruption de p0
4)p1 reprend son exécution au niveau int
p1 met reg [1]=-2 interruption de p1
5)p0 reprend son exécution, p0 test la condition jusqu'à, p0 trouve reg[0]=-2≠-1=> p0 boucle
6)p1 reprend son exécution, p1 test la condition jusqu'à, p1 trouve la condition jusqu'à reg[1]=-2≠-1 => p1 boucle par l'attente active.

Donc il y a un problème d'interblocage p0 et p1 bouclent en même temps et la section critique libre

Q2 : si l'instruction add exécuté avec interruption => il y a un problème d'interblocage , p0 et p1 bouclent en parallèle

Q3 : verification l'absence de famine ?

Add (reg[i],m){

M=m-1 (int1)

Reg[i]=m (int2)}

Cas1: verification s'il y a un problème de famine au niveau interruption int1

P0 arrive, p0 entre dans la section critique car m=-1 et reg [0]=-1

P1 arrive, p1 boucle car m=-2 et reg [1]=-2

3) p0 sort de la section critique, p0 met m=0

P1 peut entrer dans la section critique mais il est interrompu par p0 au niveau int1

C'est-à-dire p1 exécute add(reg[1],m=0){m=0-1=-1 interruption int1}

P0 demande une deuxième fois d'entrer dans la section critique

P0 exécute la boucle répéter, p0 exécute add(reg[0],m=-1){m=-1-1=-2 et reg[0]=-2)

P0 trouve reg [0]=-2≠-1 => p0 boucle

P1 reprend son exécution, p1 met reg[1]=-2 => p1 trouve reg[1]=-2≠-1 => p1 boucle

⇨ p0 et p1bouclent en parallèle => il y a un problème d'interblocage au niveau int1

cas : interruption au niveau int2

1)p0 arrive, p0 entre dans la section critique car m=-1 et reg [0]=-1
2)p1 arrive, p1 boucle car m=-2 et reg [1]=-2
3)p0 sort de la section critique, p0 met m=0
4)p1 peut entrer dans la section critique mais il est interrompu par p0 au niveau int2
c'est-à-dire p1 exécute add(reg[1],m=0){m=0-1=-1,reg[1]=-1 interruption int2}

5)p0 demande une deuxième fois d'entrer dans la section critique, p0 exécute
add(reg[0],m=-1){ m=-1-1=-2, reg[0]=-2) => p0 trouve reg[0]=-2≠-1 => p0 boucle
6)p1 reprend son exécution, p1 trouve reg [1]=-1(condition vrai)
⇨ p1 entre dans la section critique et p0 boucle
⇨ il n y a pas un problème de famine s'il y a une interruption au niveau int2