ED 8
Exclusion mutuelle par variables communes

L'objectif de l'ED est de construire une solution respectant certaines propriétés décrites ci-dessous. L'aboutissement est l'algorithme de Dekker (étendu à N tâches par Dijkstra). La construction est progressive, les premières questions aboutissent à des solutions ne satisfaisant qu'à une partie des propriétés.

Rappels

• Définitions :

- Ressource critique : une ressource partageable à un seul point d'accès (qui ne peut être attribuée qu'à une seule tâche à un instant donné) est dite ressource critique.

- Section critique : la phase d'utilisation par une tâche d'une ressource critique est dite section critique.

• Hypothèses :

- Les vitesses des tâches sont quelconques et inconnues

- Toute tâche sort de la section critique au bout d'un temps fini.

• La solution doit comporter les propriétés suivantes :

a) A tout instant une tâche au plus peut se trouver en section critique.

b) Si plusieurs tâches sont bloquées en attente de la ressource critique, alors qu'aucune tâche ne se trouve en section critique, l'une d'elles doit pouvoir y entrer au bout d'un temps fini (pas de blocage mutuel indéfini).

c) Si une tâche est bloquée hors d'une section critique, ce blocage ne doit pas empêcher l'entrée d'une autre tâche en section critique.

d) La solution doit être la même pour toutes les tâches (aucune tâche ne doit jouer de rôle privilégié).

• La programmation de l'exclusion mutuelle se décompose en trois parties :

- entrée

- section critique

- sortie

L'entrée et la sortie assurent le respect des propriétés a), b), c) et d).

Exercice 1 : Algorithme de Dekker

On se propose de programmer l'exclusion mutuelle entre deux tâches parallèles pour l'accès à une section critique : les seules opérations indivisibles sont l'affectation d'une valeur à une variable et le test de la valeur d'une variable.

Principe de la solution :

- Définir un ensemble de variables d'état, communes aux contextes des deux tâches.
- L'autorisation d'entrée en Section Critique sera définie par des tests sur ces variables, et l'attente éventuelle sera programmée comme une attente active (répétition cyclique des tests).

On supposera par la suite que les tâches sont cycliques et que leur comportement est le suivant :

```
While (true) {

      Entrée_en_section_critique

      Section_critique

      Sortie_de section_critique

      Section_non_critique

}
```

Question 1

On utilise une seule variable booléenne M telle que M = vrai si une des tâches se trouve dans sa section critique, faux sinon.

Ecrire le programme

Vérifier que l'exclusion mutuelle ne peut être ainsi programmée.

Question 2

On utilise une variable commune unique T telle que T = i si et seulement si la tâche Pi est autorisée à entrer en sa section critique (i = 0, 1).

Écrire le programme d'une tâche.

Montrer que la solution ne vérifie pas la condition c) (le blocage d'une tâche hors de sa section critique peut empêcher l'autre d'entrer en sa section critique) mais vérifie les autres conditions.

Question 3

On utilise C(i) variable booléenne attachée à la tâche Pi (i = 0, 1)

C(i) = vrai si Pi est dans sa section critique ou demande à y entrer

C(i) = faux si Pi est hors de sa section critique

Pi peut lire et modifier C(i), peut lire seulement C(j) si j différent de i.

Écrire le programme de la tâche Pi

Vérifier qu'on ne peut obtenir qu'une solution satisfaisant aux conditions a), c), d) ou b), c), d) mais non aux quatre.

Question 4

On peut obtenir une solution correcte en combinant les solutions précédentes et en introduisant une variable supplémentaire T servant à régler les conflits à l'entrée de la section critique, T n'est modifiée qu'en fin de section critique.

L'ensemble des variables est:

- C(i) avec la signification précédente (Question 3)

- T avec la signification précédente (Question 2)

S'il y a conflit (C(i) = C(j) = vrai), Pi et Pj exécutent une séquence d'attente où T a une valeur constante.

Si T= j, alors Pi annule sa demande en positionnant C(i) à faux, Pj peut alors entrer en section critique. Pi attend que T= i et refait sa demande en positionnant C(i) à vrai.

Écrire le programme de Pi. Vérifier les 4 conditions.

Exercice 2 : Algorithme de Peterson

On vous propose les deux algorithmes suivants, A et B, pour traiter l'exclusion mutuelle entre deux tâches. Pouvez-vous dire s'ils sont corrects et les analyser sous les deux points de vue suivants :

a- Respect de l'exclusion mutuelle

b- Absence d'interblocage

Pour donner une réponse plus claire, ne pas hésiter à numéroter les instructions des tâches.

Algorithme A :

```
//Déclaration et initialisation des variables globales
      int CANDIDAT[2] ; //peut prendre la valeur 0 ou 1
      int PRIORITE=1 ;
      for (i=0 ;i<2 ;++i) CANDIDAT[i]=0 ; //false
TÂCHE0 :
      while (1) {
      CANDIDAT[0]= 1;
      while ((CANDIDAT[1]) and (PRIORITE == 1)) ;
      Section Critique() ;
      PRIORITE = 1;
      CANDIDAT[0]= 0;
      section_non_critique() ;
}
TÂCHE1 :
      while (1) {
      CANDIDAT(1)= 1;
      while ((CANDIDAT[0] and (PRIORITE == 0)) ;
      Section Critique() ;
      PRIORITE = 0;
      CANDIDAT[1]=0;
      Section_non_critique() ;
}
```

Algorithme B

```
//Déclaration et initialisation des variables globales
      int CANDIDAT[2] ; //peut prendre la valeur 0 ou 1
      int PRIORITE=1 ;
      for (i=0 ;i<2 ;++i) CANDIDAT[i]=0 ; //false
      TÂCHE0 :
      while (1) {
      CANDIDAT[0]= 1;
      PRIORITE = 1;
      while ((CANDIDAT[1]) and (PRIORITE == 1));
      Section Critique() ;
      CANDIDAT[0]= 0 ;
      Section_non_critique() ;
}
TÂCHE1 :
            while (1) {
            CANDIDAT[1]= 1;
            PRIORITE = 0;
            while ((CANDIDAT[0] and (PRIORITE == 0)) ;
            Section Critique() ;
            CANDIDAT[1]= 0;
            Section_non_critique() ;}
```

ED 8 Corrigé indicatif
Exclusion mutuelle par variables communes

Les booléens sont implantés sous forme d'entiers prenant la valeur 0 (faux) ou 1(vrai)

Question 1

On utilise une seule variable booléenne M telle que M = vrai si un des processus se trouve dans sa section critique, faux sinon.

Ecrire le programme

Vérifier que l'exclusion mutuelle ne peut être ainsi programmée.

On suppose qu'on attribue à P0 et P1, les identifiants 0 et 1

```
/*déclarations et initialisation des variables globales*/
int M = 0 ;//M est a faux
/* etat d'occupation de la ressource*/
Tâche P0
While (1) {
      /*entree_SC */
      while (M );/* attente active*/
      M=1;//M est a vrai
      Section_critique();
      /*sortie_SC*/
      M=0;
      Section_non_critique() ;
}
Tâche P1
While (1) {
      /*entree_SC */
      while (M );/* attente active*/
      M=1;
      Section_critique();
      /*sortie_SC*/
      M=0;
      Section_non_critique() ;

}
```

Une exécution « entrelacée » de l'entrée en Section critique, par les deux tâches entraîne le non-respect de l'exclusion mutuelle, `entree_SC` n'est pas atomique

Question 2

On utilise une variable commune unique T telle que T = i si et seulement si la tâche Pi est autorisée à entrer en sa section critique (i = 0, 1).

Écrire le programme d'une tâche.

Montrer que la solution ne vérifie pas la condition c) (le blocage d'un processus hors de sa section critique peut empêcher l'autre d'entrer en sa section critique) mais vérifie les autres conditions.

```
/*déclarations et initialisation des variables globales*/
int T = 0 ;
/* T vaut 0 ou 1 tâche autorisée à entrer en section critique*/
Tâche P0
While (1) {
      /*entree_SC */
      while (T==1 );/* attente active*/
      Section_critique();
      /*sortie_SC*/
      T=1;
      Section_non_critique() ;
}
Tâche P1
While (1) {
      /*entree_SC */
      while (T==0 );/* attente active*/
      Section_critique();
      /*sortie_SC*/
      T=0 ;
      Section_non_critique() ;

}
```

Cette solution règle le problème de l'exclusion mutuelle, une seule tâche peut entrer en SC(valeur de T). C'est un fonctionnement à l'alternat donc si un processus tombe en panne hors section critique, l'autre processus sera bloqué. La condition c) n'est pas respectée.

Question 3

On utilise c(i) variable booléenne attachée au processus Pi (i = 1, 2)
C(i) = vrai si Pi est dans sa section critique ou demande à y entrer
C(i) = faux si Pi est hors de sa section critique
Pi peut lire et modifier c(i), peut lire seulement C(j) si j différent de i.

Écrire le programme du processus Pi. Montrer qu'on ne peut obtenir qu'une solution satisfaisant aux conditions a), c), d) ou b), c), d) mais non aux quatre.

```
/*déclarations et initialisation des variables globales*/
int C[2]={0,0};
Tâche P0
While (1) {
      /*entree_SC */
      while (C[1]);/* attente active*/
      C[0]=1 ;
      Section_critique();
      /*sortie_SC*/
      C[0]=0;
      Section_non_critique() ;
}
Tâche P1
While (1) {
```

```
      /*entree_SC */
      while (C[0]);/* attente active*/
      C[1]=1 ;
      Section_critique();
      /*sortie_SC*/
      C[1]=0;
      Section_non_critique() ;

}
```

L'exclusion mutuelle n'est pas garantie, chaque tâche peut trouver la variable C de l'autre tâche à faux et décider son entrée en SC
Une autre solution pourrait être que chaque processus positionne sa variable à true, puis teste la variable de l'autre processus, dans ce cas on risque un blocage mutuel indéfini.

Question 4

On peut obtenir une solution correcte en combinant les solutions précédentes et en introduisant une variable supplémentaire T servant à régler les conflits à l'entrée de la section critique, T n'est modifiée qu'en fin de section critique.

L'ensemble des variables est:

- C(i) avec la signification précédente (Question 3)

- T avec la signification précédente (Question 2)

S'il y a conflit (C(i) = C(j) = vrai), Pi et Pj exécutent une séquence d'attente où T a une valeur constante.

Si T = j, alors Pi annule sa demande en faisant C(i) = faux, Pj peut alors entrer en section critique. Pi attend que t = i et refait sa demande par C(i) = vrai

Écrire le programme de Pi. Vérifier les 4 conditions.

```
/*déclarations et initialisation des variables globales*/
int C[2]={0,0} ;
int T=0

Tâche P0
while (1) {
      /*entree_SC */
      C[0]=1 ;
      while (C[1]) { /* P1 est en SC ou a demande a entrer en SC*/
            while (T==1) C[0]=0 ;/*P1 est prioritaire, retrait de P0*/
            C[0]=1 ;/*T=0 P0 recandidate*/
      }
      Section_critique();
      /*sortie_SC*/
      C[0]=0;
      T=1 ;
      Section_non_critique() ;
}

Tâche P1
While (1) {
      /*entree_SC */
```

```
      C[1]=1 ;

      while (C[0]) { /* P0 est en SC ou a demande a entrer en SC*/
            while (T==0) C[1]=0 ;/*P0 est prioritaire, retrait de P1*/
            C[1]=1 ;/*T=1 P1 recandidate*/
      }
      Section_critique();
      /*sortie_SC*/
      C[1]=0;
      T=0 ;
      Section_non_critique() ;
}
```

Exercice 2 : Algorithme de Peterson

a) Respect de l'exclusion mutuelle

Algorithme A

L'un peut entrer en SC parce que l'autre n'y est pas, puis l'autre peut entrer en SC parce qu'il est prioritaire, d'où non respect de l'exclusion mutuelle.

Algorithme B

Si l'un est en SC parce que l'autre n'y est pas, l'autre ne peut entrer en SC puisqu'il a positionné PRIORITE à la valeur de celui qui est en SC ; il y a donc respect de l'exclusion mutuelle.

b) Absence de l'interblocage

Les algorithmes A et B sont exempts d'interblocage, l'arbitrage est réalisé par la valeur de la variable PRIORITE.