TP N° 3 : Exclusion Mutuelle par attente active

Exercice 1 : Problème des accès concurrents

Ecrire un programme qui initialise une variable entière V à 0 et crée deux thread th1 et th2. Le thread th1 incrément V 10000 fois et le thread th2 décrémente V 10000 fois.
Quelle devrait être la valeur finale de V ? Que remarquez-vous après plusieurs exécutions du programme ? Expliquez.

Exercice 2

Ecrire un programme qui initialise une variable entière V à 0 et crée deux thread th1 et th2. Le thread th1 et le thread 2 incrément V 10000 fois.
Quelle devrait être la valeur finale de V ? Que remarquez-vous après plusieurs exécutions du programme ? Expliquez.

Exercice 3 : algorithme de Peterson pour l'exclusion mutuelle

Réécrire le programme exercice 2 en y intégrant la solution de Peterson pour résoudre le problème de l'accès concurrent à la même variable.

Algorithme de PETERSON

Initialisation : var c : tableau [1..2] : entier ;
tour : 0…1 ;
tour := 0 ; c [0] := 0 ; c[1] := 0 ;

Processus Pi

debut
c [i] : = 1 ;
tour :=1-i;
tant que ((c[1-i]==1) et (tour==1-i)) faire rien;

<section critique>

c [i] : = 0 ;
fin

Exercice 4

Ecrire un programme qui crée une matrice 3*3. Initialisé les valeurs des éléments diagonale par des 0.

Crée deux threads th1 et th2. Le thread th1 incrément 100 fois la valeur des éléments diagonale de cette matrice et le thread th2 décrémente 100 fois la valeur de mêmes éléments diagonale de cette matrice.

Q1 : Quelle devrait être la valeur finale de cette matrice? Que remarquez-vous après plusieurs exécutions du programme ?

Q2 : Résoudre le problème de l'accès concurrent aux éléments de cette matrice en utilisant l'algorithme suivant :

Algorithme d'exclusion mutuelle

Var désire : Array [1..2] of boolean init [false,false]
Tour : 1..2 init 1 (ou 2)

$$desire[i] = \begin{cases} vrai & Pi\ veut\ entrer\ à\ la\ SC\ ou\ il\ est\ déjà\ dedans \\ faux & Pi\ est\ hors\ de\ sa\ SC\ et\ il\ n'a\ pas\ l'envie\ d'y\ accéder \end{cases}$$

$$Tour = \begin{cases} 1 & le\ tour\ est\ au\ P1 \\ 2 & le\ tour\ est\ au\ P2 \end{cases}$$

Pi :

```
Désire[i]  := true                                  // Pi veut entrer
Si (Desire[3-i] ) alors                             //Pi suppose que l'autre process veut rentrer ou il est dans la SC
    Si (tour = 3-i) alors                           // Situation de conflit : Est-ce que P3-i à le tour d'y accéder
        Désire[i]  := false ;                       // oui, Pi renonce
        Tant que (tour <> i) faire                  // Pi attend son tour
        FTq
        Désire[i]  := true ;                        // Pi reprend sa décision d'accéder
    FSi
    Tant que Desire[3-i] faire                      // P3-i est il sorti de sa SC ? ou désire t- il encore d'y accéder ?
    FTq
Fsi
```

<SC>

```
Tour := 3-i ;
Desire[i] := faux ;
```

Exercice 5

Pour calculer la somme des éléments d'un tableau on utilise deux threads th1 et th2. Le premier parcourt les éléments d'indice impair et l'autre les éléments d'indice pair comme suit :
const M = ……//taille du tableau
var T : tableau [1…M] entier ;
somme : entier initialisé a 0

thread1 ()	**thread2 ()**	**Programme principale**
var i :entier	var j :entier	
debut	debut	debut
i :=1 ;	j :=2 ;	
tant que (i≤M) faire	tant que (j≤M) faire	thread1() ; thread2()
somme:= somme+ T[i];	somme:= somme+ T[j];	
i:=i+2;	j:=j+2;	Fin
fin tant que	fin tant que	
fin	fin	

1- Quel est le problème posé par cette solution ? Justifier.
2- Proposer une solution afin de résoudre ce problème.

3- Réécrire ce programme en y intégrant la solution de l'algorithme dekker pour résoudre le problème d'accès concurrent (problème d'exclusion mutuelle) à une variable partagé.

```
#include <stdio.h>
#include <pthread.h>
#include <stdbool.h>


Int somme ;
        Const int  M =10;
        Int T[M] ;

        Void Fct1(){
        Int i ;
        I=1;
        While( i <M){
        Somme = somme + T[i] ;
        I=i+2 ;
        }
        }
```

```
Void Fct2(){
Int j ;
J=2;
While( j <M){
Somme = somme + T[j] ;
j=j+2 ;
}
}
```

```
int main()
{
   pthread_t p1, p2;
   printf ("remplir le tableau tab :\n");
   for (k=0; k<M;k++){
   printf("tab[%d] =",k);
   scanf("%d",&tab[k]);
}
    // creation deux threads executent les deux la même fonction
    pthread_create(&p1, NULL, fct1, (void*)0);
    //sleep(2);
    pthread_create(&p2, NULL, fct2, (void*)1);
    // attendre la fin dÂ'execution des deux thread
    pthread_join(p1, NULL);
    pthread_join(p2, NULL);
    printf("la somme est : %d\n",somme);


    return 0;
}
```

Algorithme de DEKKER

Vous avez le programme suivant :

Contexte commun : var c : tableau [1..2] : booléen ;

tour : 0.. 1 ;

tour := 0 ; c [1] := 0 ; c[2] := 0 ;

Processus Pi

debut

c [i] : = 1 ;

tant que c[1-i] faire si tour = (1-i) alors debut

c[i] : = 0 ;

tant que tour = (1-i) faire boucle

c [i] : = 1 ;

fin

<section critique>

tour : = 1-i;

c [i] : = 0;

fin

4- Une autre variante de la solution précédente consiste à utiliser deux variables globales **somme1** et **somme2**. La première variable **somme1** sera utilisé uniquement par th1 pour calculer la somme des éléments d'indice impair et **somme2** sera utilisé uniquement par th2 pour calculer la somme des éléments d'indice pair. Le résultat final (**somme1+somme2**) sera calculé par un nouveau processus th3.

Le programme principal proposé sera :

```
Debut
thread1t
hread2th
read3
Fin
```

4- Donner les modifications necessaries pour th1 et th2 ainsi le code de th3.

thread1 ()	**thread2 ()**	**thread3 ()**
Var i :entier	Var j :entier	
Debut	Debut	Debut
i :=1 ;	j :=2 ;	
tant que (i≤M) faire	tant que (j≤M) faire	Somme3 =somme1 +somme2
somme1:= somme1+ T[i];	somme2:= somme2+ T[j];	Fin
i:=i+2;	j:=j+2;	
fin tant que	fin tant que	
fin	fin	

TP N 3 : Exclusion Mutuelle par attente active