Programmation Concurrente

Exclusion mutuelle par attente active

Exercice 1

Est-ce que l'algorithme ci-dessous garanti l'exclusion mutuelle (un seul thread en section critique, pas d'interblocage, pas de famine) entre les threads T0 et T1 ? Si ce n'est pas la cas, justifiez votre réponse en décrivant une séquence possible d'opérations qui prouve le contraire.

```
bool inside[2] = {false,false};

// id is the thread number (0 or 1)
void prelude(int id) {
     while (inside[1-id]) {}
     inside[id] = true;
}

void postlude(int id) {
     inside[id] = false;
}
```

```
T0:
  while (inside[1-id]) {}
  [sort du while]
  SWITCH
T1:
  while (inside[1-id]) {}
  [sort du while]
  inside[1] = true;
  [entre en SC]
  SWITCH
T0:
  inside[0] = true;
  [entre aussi en SC !]
```

→ **EXCLUSION MUTUELLE NON SATISFAITE !**

Exercice 2

Voir code de la correction.

Exercice 3

Implémentez l'algorithme de Peterson pour l'exclusion mutuelle par attente active dans le cadre de 2 threads.

Elaborez ensuite un scénario afin de tester que votre implémentation fonctionne correctement.

- Que remarquez-vous ?

L'algorithme ne fonctionne pas.

- Si le résultat que vous obtenez n'est pas celui escompté, essayez de trouver un moyen pour corriger le problème.

L'implémentation historique illustre le problème d'incohérence mémoire lié aux architectures multi-processeurs modernes.

L'algorithme doit être modifié afin d'y ajouter une barrière mémoire (voir code de la correction).

Une autre possibilité est de ne l'exécuter que sur 1 CPU avec la fonction sched_setaffinity ou encore avec la commande taskset 0x00000001 prog. Cependant, même s'il fonctionne dans ce cas, l'ordonnanceur produit une famine de durée variable.

```
bool intention[2] = {false,false};
int turn = 0; // ou 1

void *T0(void *arg) {
    while (true) {
        intention[0] = true;
        while (turn != 0) {
            while (intention[1]) {}
            turn = 0;
        }
        // section critique
        intention[0] = false;
        // section non-critique
    }
}

void *T1(void *arg) {
    while (true) {
        intention[1] = true;
        while (turn != 1) {
            while (intention[0]) {}
            turn = 1;
        }
        // section critique
        intention[1] = false;
        // section non-critique
    }
}
```

```
intention[0]=intention[1]=false
turn = 0

T0:
  intention[0] = true
  SWITCH
T1:
  intention[1] = true
  while (turn != 1) {
      [bloqué dans la boucle]
      while (intention[0]) {}
  SWITCH
T0:
  entre en SC
  intention[0] = false;
  intention[0] = true;
  SWITCH
T1:
  [toujours bloqué dans la boucle]
  while (intention[0]) {}
  SWITCH
T0:
  entre en SC
  intention[0] = false;
  intention[0] = true;
  SWITCH
T1:
  [toujours bloqué dans la boucle]
  while (intention[0]) {}
  SWITCH
... etc.

→ ATTENTE NON BORNÉE (FAMINE) !
```