

# TD4 : Synchronisation de processus

---

## 1 Exclusion mutuelle sur multi-processeur

### 1.1 L'instruction DMLR

Soit l'instruction DMLR (*Decrease Memory and Load Register*) définie de la manière suivante ( $R_i$  est un registre du processeur  $i$  et  $m$  est une case mémoire partagée) :

```
instruction DMLR  $R_i, m$   
| <bloquer l'accès à la case  $mem[m]$ >  
|  $mem[m] := mem[m] - 1;$   
|  $R_i := mem[m];$   
| <libérer l'accès à la case  $mem[m]$ >
```

Programmez l'exclusion mutuelle à l'aide de cette instruction, c'est-à-dire les trois sections de code <init>, <prologue> et <épilogue>. Quelles critiques peut-on faire de cette solution ?

### 1.2 L'instruction XRM

Soit l'instruction XRM (*eXchange Register with Memory*) définie ci-dessous. Programmez l'exclusion mutuelle à l'aide de cette instruction.

```
instruction XRM  $R_i, m$   
| <bloquer l'accès à la case  $mem[m]$ >  
| échange les valeurs de  $R_i$  et  $mem[m];$   
| <libérer l'accès à la case  $mem[m]$ >
```

## 2 Solution programmée de Peterson (1981)

On considère deux processus ( $P_0$  et  $P_1$ ) désirant entrer en section critique. Le code de synchronisation du processus  $P_i$  est le suivant :

```
init           demande := {faux, faux};  
  
prologue      demande[i] := vrai;  
              tour := (1 - i);  
              répéter  
              jusqu'à (demande[1 - i] = faux) ou (tour = i);  
  
              < section critique >  
  
épilogue      demande[i] := faux
```

Montrez que l'exclusion mutuelle est vérifiée et que les deux processus ne peuvent être bloqués sans raison.

### 3 Synchronisation de l'accès aux ressources

On dispose de  $n$  ressources d'une même classe. L'état de ces ressources est donné par le tableau :

libre : **tableau** [ 1 .. n ] de **booléens**

On vous demande d'écrire les trois sections de code  $\langle \text{init} \rangle$ ,  $\langle \text{allocation} \rangle$  et  $\langle \text{libération} \rangle$  qui respectent les points suivants :

- au début les ressources sont toutes libres,
- le code  $\langle \text{allocation} \rangle$  bloque le demandeur si le nombre de ressources libres est égal à zéro ; ou choisit une des ressources libres dans les autres cas ;
- le code  $\langle \text{libération} \rangle$  réveille les demandeurs éventuellement endormis.

$\langle \text{allocation de la ressource de n}^\circ r \rangle$   
 $\langle \text{utilisation de la ressource de n}^\circ r \rangle$   
 $\langle \text{libération de la ressource de n}^\circ r \rangle$

Bien entendu il est fortement conseillé d'utiliser un ou plusieurs sémaphores (à compteur) pour programmer cette synchronisation.

### 4 Machine parallélisée

On dispose d'une machine capable d'exécuter des calculs arithmétiques en parallèle. Par les mots **Cobegin** et **Coend**, le programmeur peut spécifier que les instructions d'un bloc sont sans lien entre elles, et peuvent donc être traitées en parallèle. Utiliser **Cobegin** et **Coend** pour expliciter le parallélisme dans les calculs suivants :

```
x := a + b + c + d ;  
y := a + b * c + d ;  
z := a * ( b * c * d + e ) ;
```

*Règle du Never Wait.* Soit la séquence :

```
a := b * c ;  
si ( a = 9 ) alors  
    d := 10 ;  
fin si  
e := d * f ;
```

Celle-ci a donc une durée de quatre unités de temps au maximum (le test et les opérations arithmétiques comptent pour une unité). En utilisant **Cobegin** et **Coend**, il est possible d'écrire un programme dont la durée soit de trois unités de temps au maximum ! Comment ?