Chapitre 5

Programmation Modulaire
et
Synchronisation

Moniteur de Hoare

Remarques sur les sémaphores:

- **Simple, efficace mais nécessite une méthodologie**
- **Oublier un P ou un V pour, tout le système tombe en panne**
- **Contrôle et code du traitement mélangés: maintenance?**
- **Commenter les appels de P et de V est difficile**

V(S1);
::::::::::
P(S2);

P..
Intst1..
Inst2;
P..
inst3

Pro1……||……..|| Proi……|| Pn

P(mutex); **P(mutex);** **P(mutex);**
SC1; **SC2:** **SCn;**
V(mutex); ****** **V(mutex);**

Syntaxe

Moniteur <nom>
{ [<déclaration>], [<conditions>];
<opérations>
}

Module, objet passif

En exclusion mutuelle
S'exécutent dans le contexte
de l'appelant

Privé
Types, variables..

Privé
Files d'attentes (Fifo éventuellement avec priorité)
Condition c;
c.Wait(), c.signal() et c.empty()

Utilisation

nom.op(..)

Exemple1
Ressource Critique

Moniteur RC
{
Ressource_Critique R;
void Utiliser ()
{ …utiliser R… u1
}
::
::
}

Entry u2

R: ressource critique, à utiliser en exclusion mutuelle
Il suffit de la déclarer dans le moniteur
Elle sera accessible seulement à travers les opérations du moniteur

Seules les opérations du moniteurs sont visibles

U1
:::
RC.utiliser()
;
:::::

||

U2
:::
RC.utiliser();
:::::

Exemple2
BAL Simple

1 producteur
1 consommateur

p

```
:::
BAL.deposer(m);
:::::
```

c

```
:::
BAL.prelever(&m);
:::::
```

Moniteur BAL

```
{ condition oc;                          oc
  message T;                             ?
  boolean busy = false;                  False
  void deposer (message m)
        {       if (busy) oc.wait();
                T = m;
                busy = true;
                oc.signal;
        }
  void prelever (message *m)
        {       if  (!busy) oc.wait
                busy = false;
                (*m) = T;
                oc.signal();
        }
}
```

Entry

urgent

Exemple2
BAL Simple

n producteurs
m consommateurs
n et m > 1

Solution 1

p

```
:::
BAL.deposer(m);
:::::
```

c

```
:::
BAL.prelever(&m);
:::::
```

Moniteur BAL

```
{
  condition c_plein;                     C_plein
  condition c_vide;                      C_vide
  message T;                             ?
  boolean busy = false;                  False
  void deposer (message m)
        {       if (busy) c_vide.wait();
                T = m;
                busy = true;
                c_plein.signal;
        }
  void prelever (message *m)
        {       if  (!busy) c_plein.wait
                busy = false;
                (*m) = T;
                c_vide.signal();
        }
}
```

Entry

urgent

Exemple2
BAL Simple

n producteurs
m consommateurs
n et m > 1

Solution 2 avec une seule condition

p

```
:::
BAL.deposer(m);
:::::
```

c

```
:::
BAL.prelever(&m);
:::::
```

Moniteur BAL

```
{ condition oc;
  message T;
  boolean busy = false;                  ?
  void deposer (message m)               False
        {       while (busy) oc.wait();
                T = m;
                busy = true;
                oc.signal;
        }
  void prelever (message *m)
        {       while  (!busy) oc.wait
                busy = false;
                (*m) = T;
                oc.signal();
        }
}
```

Entry

urgent

Condition **C**
C est une file d'attente fifo ou avec priorité
0 est la priorité la plus forte

Un objet condition, C, est privé au moniteur
Il est utilisé par:

C.wait():
- mémorisation de l'identité de l'appelant dans **C**
- blocage de l'appelant
- libération du moniteur

C.empty():
- retourne vraie si **C** est vide sinon retourne faux

C.signal():
- réveil de la tête de **C** si elle n'est pas vide sinon cette opération est sans effet
- le processus réveillé est retiré de **C** et prend le contrôle du moniteur
- le processus qui réveille est mise en attente dans **URGENT**

Le moniteur est libéré:
- **à la fin d'une opération**
- **à l'exécution d'un wait**

→

Si URGENT n'est pas vide
  On réveille le processus en tête
Sinon
  Si Entry n'est pas vide
    On réveille le processus en tête d'Entry
  Sinon
    Le moniteur est accessible

Implémentation des moniteurs en terme de sémaphores :

```
Semaphore Entry; init(Entry, 1);
Semaphore Urgent; init(Urgent, 0);
/* Nous utilisons la fonction vide, « atomique », sur les sémaphores */
/* Une condition C*/
Semaphore C; init(C,0);
```

```
/* Fin d'une opération du moniteur */
{
  si !vide(Urgent) V(Urgent);
          sinon  V(Entry);
 /* si Entry n'est pas vide , alors faire rentrer sa tête dans le moniteur sinon le moniteur est libre*/
}
```

```
/* C.wait() */
{
  si !vide(Urgent) V(Urgent);
          sinon V(Entry);
  /* si Entry n'est pas vide , alors faire rentrer sa tête dans le moniteur sinon le moniteur est libre*/
  P(C);
}
```

```
/*C.signal()*/
{
   Si !vide (C) {
      V(C);
      P(Urgent ); }
}
```

Exemple3
Blocage avec priorité

Il s'agit d'un moniteur qui gère l'horloge d'un système. Le moniteur permet :
- **d'initialiser l'horloge**
- **d' incrémenter l'horloge (à chaque top ou unité de temps)**
- **de mettre en sommeil un appelant pendant un nombre d'unités précisé par ce dernier.**

```
Reveil_Moi

Moniteur {
 int H;
 condition c;
 void init(int m) { H = m;}

/* exécutée à chaque interruption horloge*/
void incr() {
  H++;
 c.signal;
 /* réveiller, éventuellement, un processus. Ce dernier va tester sa condition de réveil (voir boucle dans W_M) */
}

void W_M (int x) {
   int MH;
   MH = H+x;
   while (H > MH) c.wait(MH);
      c.signal;
                  }
```

H variable du moniteur partagée entre les processus à travers les opérations du moniteur

Priorité: entier naturel
Priorité la plus forte = 0

```
/* mon heure de réveil*/
/* se bloquer ou se rebloquer si mon heure n'est pas arrivée */
/* réveil systématique du suivant qui est la tête de c si c n'est pas vide*/
/* on procède à un réveil en cascade des processus qui possèdent */
/* la même heure de réveil */
/* cette cascade s'arrête, soit parce que c est devenue vide*/
/* soit l'heure du réveillé n'est pas la bonne, il sera rebloqué (voir la boucle */
```

Variable locale à W_M (appartient au contexte du processus) qui exprime l'heure de réveil du processus suspendu

La priorité, permettra :
- d'avoir les processus qui ont la même heure de réveil d'être en séquence dans la condition
- de minimiser le nombre de réveils qui reconduisent à un blocage

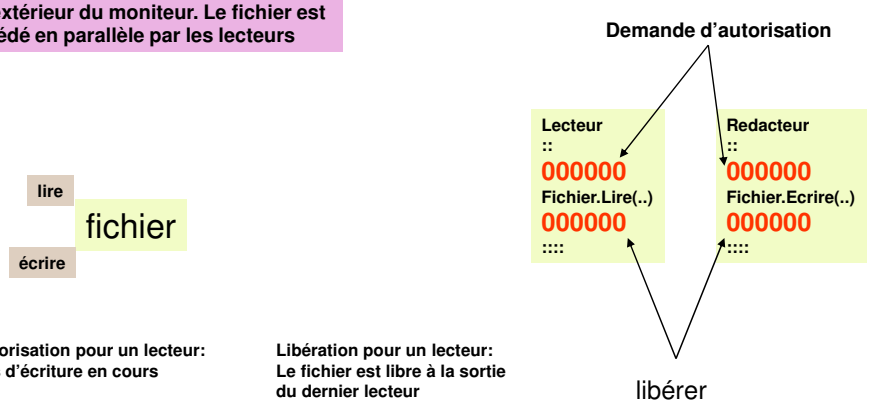
Le moniteur est un contrôleur d'accès à une ressource

Si la ressource est critique, il suffit de la déclarer dans le moniteur. L'utilisation obligatoire de cette ressource à travers les opérations du moniteur assure son utilisation en exclusion mutuelle

Si la ressource pouvait être accédée en parallèle, alors elle doit être déclarée à l'extérieur du moniteur. Le contrôle d'accès sera programmé dans le moniteur.

Exemple4: Lecteurs/Rédacteurs
Nous souhaitons programmer le contrôle d'accès à un fichier
- **Les lecteurs peuvent accéder au fichier en parallèle**
- **Une écriture doit être en exclusion mutuelle avec une lecture ou avec une autre écriture**

1. On spécifie d'abord l'objet utilisé à l'extérieur du moniteur. Le fichier est accédé en parallèle par les lecteurs



Autorisation pour un lecteur:
Pas d'écriture en cours

Libération pour un lecteur:
Le fichier est libre à la sortie du dernier lecteur

Autorisation pour un rédacteur:
Pas d'écriture, ni lecture en cours

Libération pour un rédacteur:
Le fichier est libre à la sortie du rédacteur en cours

2. On spécifie le moniteur d'accès
Les opérations lire et écrire sont enveloppées par les opérations du moniteur qui permettent d'obtenir:

- **une autorisation d'accès**
- **une sortie du fichier**

```
Lecteur
::
L_R.Deb_Lire();
Fichier.Lire(..)
L_R.Fin_Lire();
::::
```

```
Redacteur
::
L_R.Deb_Lire();
Fichier.Lire(..)
L_R.Fin_Lire();
::::
```

3. Développer le moniteur

```
Moniteur L_R /* différentes variantes à développer en TD */
{
  ::::::::
  void Deb_Lire()  :::
  void Fin_Lire()  :::
  void Deb_Ecrire()  :::
  void Fin_Ecrire()  :::
}
```