Les moniteurs

Les moniteurs spécifiés par Hoare et Brinch Hansen reposent sur les principes suivants :
-Exclusion mutuelle implicite entre les méthodes d'accès
=>file d'attente au module
-Conditions d'accès reposant sur des tests de variables d'état
=>file d'attente par condition d'accès
Qu'est-ce qu'un moniteur?
Un moniteur est un objet encapsulant des procédures dont une seule peut s'exécuter à un moment donné. La structure de moniteur permet d'éviter certains phénomènes d'interblocages qui peuvent apparaître avec les sémaphores. Ces interblocages sont généralement la conséquence d'erreurs de programmation vicieuses et difficiles à détecter. C'est pourquoi, on considère que l'écriture de sémaphores est délicate. La déclaration d'un moniteur est beaucoup plus facile par contre.

```
monitor MonMoniteur{
        char tampon[100];
        void écrivain(char *chaîne) {}
        char * lecteur() {}
}
```

Les moniteurs proposent une solution de "haut-niveau" pour la protection de données partagées (Hoare 1974)
• Ils simplifient la mise en place de sections critiques
• Ils sont définis par
– des données internes (appelées aussi variables d'état)
– des primitives d'accès aux moniteurs (points d'entrée)
– des primitives internes (uniquement accessibles depuis l'intérieur du moniteur)
– une ou plusieurs files d'attentes

```
Moniteur Nom_moniteur ;
 Début
Déclaration des variables locales (ressources partagées);
Déclaration et corps des procédures du moniteur
 (points d'entrée);
Initialisation des variables locales;
Fin
```

Seul un processus (ou tâche ou thread) peut être actif à un moment donné à l'intérieur du moniteur

• La demande d'entrée dans un moniteur (ou d'exécution d'une primitive du moniteur) sera bloquante tant qu'il y aura un processus actif à l'intérieur du moniteur
⇒ L'accès à un moniteur construit donc implicitement une exclusion mutuelle
Lorsqu'un processus actif au sein d'un moniteur ne peut progresser dans son travail (une certaine condition est fausse), il libère l'accès au moniteur avant de se bloquer.

•Lorsque des variables internes du moniteur ont changé, le moniteur doit pouvoir « réveiller » un processus bloqué.

•Pour cela, il existe deux types de primitives :
– wait : qui libère l'accès au moniteur, puis bloque le processus appelant sur une condition
– signal : qui réveille sur une condition un des processus en attente à l'intérieur du moniteur (un processus qui a
Une variable condition : est une variable
- qui est définie à l'aide du type condition;
- qui a un identificateur mais,
- qui n'a pas de valeur (contrairement à un sémaphore).
•Une condition :
- ne doit pas être initialisée
- ne peut être manipulée que par les primitives Wait et Signal.
- est représentée par une file d'attente de processus bloqués sur la même cause;
- est donc assimilée à sa file d'attente.

•La primitive Wait bloque systématiquement le processus qui l'exécute
•La primitive Signal réveille un processus de la file d'attente de la condition spécifiée, si cette file d'attente n'est pas vide; sinon elle ne fait absolument rien.

Les variables condition
•Syntaxe :

```
cond.Wait;
cond.Signal;
```

/* cond est la variable de type condition déclarée comme variable locale */
•Autre syntaxe :

```
Wait(cond) ;
Signal(cond);
```

•Un processus réveillé par Signal continue son exécution à l'instruction qui suit le Wait qui l'a bloqué.

Exemple 1 :

```
Moniteur ProducteurConsommateur ;
 {variables locales }
Var Compte : entier ; Plein, Vide : condition ;
 {procédures accessibles aux programmes utilisateurs }
Procedure Entry Déposer(message M) ;
Début
        si Compte=N alors Plein.Wait ;
        dépôt(M);
        Compte=Compte+1;
        si Compte==1 alors Vide.Signal;
Fin
Procedure Entry Retirer(message M) ;
Début
```

```
        si Compte=0 alors Vide.Wait ;
        retrait(M);
        Compte=Compte-1;
        si Compte==N-1 alors Plein.Signal;
Fin
Début
        {Initialisations }Compte= 0;
Fin.

Processus Producteur
message M;
Début
        tant que vrai faire
        Produire(M);
        ProducteurConsommateur.déposer(M)
Fin
Processus Consommateur
message M;
Début
        tant que vrai faire
        ProducteurConsommateur.retirer(M);
        Consommer(M);
Fin
```

Exemple 2 :

```
Moniteur RepasPhilosophes ;
type Statut =(pense, faim, mange);
Var Statut état[5];
Condition fourchettes[5];
 {procédures accessibles aux programmes utilisateurs }
Procedure Entry Prendre(entier i);
Début
        état[i]=faim;
        si (état[(i+1) mod 5]≠mange et état[(i-1) mod 5]≠mange)
        alors état[i]=mange;
        sinon fourchettes[i].Wait; finsi;
Fin

Procedure Entry Rendre(entier i);
Début
        état [i]=pense;
        {réveil éventuel d'un voisin}
        si (état[(i+1) mod 5]=faim et état[(i+2) mod 5]≠mange)
        alors état[(i +1) mod 5]=mange; fourchettes[(i+1) mod 5].Signal;
        sinon
        si (état[(i-1) mod 5]=faim et état[(i-2) mod 5]≠mange)
```

```
        alors état[(i-1) mod 5]=mange; fourchettes[(i-1) mod 5].Signal ;finsi;
         finsi;
Fin
Début {Initialisations }
        Pour i de 0 à 4 faire état[i]=pense;
Fin

Processus Philosophe i
entier i= numero du processus ;
Début
        tant que vrai faire
        penser;
        RepasPhilosophes.prendre(i);
        manger;
        RepasPhilosophes.rendre(i);
Fin
```