Moniteurs

— notations, sémantique et implémentation —

Introduction

Motivation les sémaphores restent un mécanisme de bas-niveau :

- imbrication des opérations P et V à contrôler ;
- un sémaphore est global à un ensemble de processus ;
- utilisé pour exprimer l'exclusion mutuelle ET l'attente sur une condition.

Caractéristiques principales

- encapsulation et abstraction des données,
- exclusion mutuelle implicite dans le moniteur,
- variables spéciales du moniteur pour l'attente sur une condition.

Historique

- Dijkstra 1972 introduit la notion de "secrétaire"
- Hansen 1972 propose une langage avec "*shared class*"
- Hoare 1973 propose la nom de "*monitor*" et clarifie sa sémantique

Notations et sémantique

Une interface et une implémentation

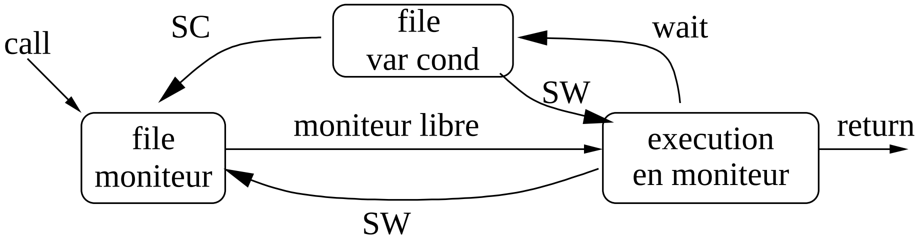
```
monitor nomM {
  déclaration variables permanentes
  initialisations
  procedure op1(formels1) { corps de op1 }
  ...
  procedure op2(formels2) { corps de op2 }
}
```

- variables permanentes = ressources contrôlées :
  - invisibles de l'extérieur ;
  - initialisées avant tout appel de procédure.
- procédures = actions sur les ressources :
  - les seules visibles de l'extérieur :
    call nomM.nomOp(arguments);
  - elles n'accèdent pas des variables extérieures
- invariant du moniteur = décrit l'état des ressources quand aucun processus ne les accèdent :
  - l'initialisation doit l'établir,
  - chaque procédure doit le maintenir.

Moyens de synchronisation

- Exclusion mutuelle : par définition, toute procédure du moniteur s'exécute en exclusion mutuelle
- Sur condition : variables spéciales appelées variables de condition, locales au moniteur :

```
cond cv; // valeur = file d'attente de processus
empty(cond); // test file vide
wait(cond); // blocage du processus dans la file
signal(cond); // réveil d'UN processus de la file
```



Politiques d'ordonnancement lors du **signal** :

- Signale et continue (SC) : le signalant reste dans le moniteur et le signalé est ordonnancé pour le moniteur
- Signale et attend (SW) : le signalant cède le moniteur et le signalé est exécuté

Exemple : implémentation d'un sémaphore

```
moniteur Semaphore {
  /* INVARIANT SEM : s > 0 */
  int s = 0;
  cond pos; /* await s > 0 */
  procedure Psem() {
    do :: s==0 -> wait(pos)
       :: else -> break
    od;
    s = s - 1
  }
  procedure Vsem() {
    s = s + 1;
    signal(pos)
  }
}
```

Si les files du moniteur sont FIFO, alors :

- politique SW : sémaphore FIFO et on peut utiliser :

  ```
  if :: s==0 -> wait(pos)
     :: else -> break
  fi
  ```

- politique SC : sémaphore non-FIFO et boucle **do** obligatoire !

Exemple : implémentation d'un sémaphore FIFO

```
moniteur FIFOSemaphore {
  /* INVARIANT SEM : s > 0 */
  int s = 0;
  cond pos; /* await s > 0 */
  procedure Psem() {
    if :: s==0 -> wait(pos)
       :: else -> s = s - 1
    fi
  }
  procedure Vsem() {
    if :: empty(po) -> s = s + 1
       :: else -> signal(pos)
    fi
  }
}
```

Si les files du moniteur sont FIFO, alors les deux politiques (SC et SW) assurent la discipline FIFO pour le sémaphore.

Choix de la politique SC d'ordonnancement

- SW est la plus ancienne, mais une sémantique trop compliquée
- SC est plus simple
- SC a été choisie par Unix, Java et Pthreads

Autres opérations sur les variables condition

- `wait(cv,rank)` insertion dans la file dans l'ordre croissant des rangs
- `signal_all(cv)` réveil de tous les processus en attente sur la file
- `minrank(cv)` valeur du rang du processus en tête de file

Exercices :

1. Implémenter avec SC `signal_all(cv)` avec **signal**.
2. Quelle est la sémantique de `signal_all(cv)` avec la politique SW ?

Implémentation des moniteurs avec des sémaphores

Pour chaque moniteur M ayant la variable conditionnelle cv déclarer comme variables globales :

```
bsem eM = 1; /* Invariant 0<= eM <=1 */
sem sCV = 0; /* Invariant 0<= sCV */
int nCV = 0; /* Invariant 0<= nCV */
```

On implémente les opérations comme suit :

```
/* entree moniteur = appel procédure */
   P(eM)
/* wait(cv) */
   nCV=nCV+1; V(e); P(sCV); P(eM)
/* signal(cv) */
   if :: nCV > 0 -> nCV=nCV-1; V(sCV)
      :: else -> skip
   fi
/* sortie moniteur = fin procédure */
   V(eM)
```

Implémentation des moniteurs dans un noyau mono-processeur

Structures de données :

- descripteur de moniteur : un lock et un file de descripteurs de processus
- tableau de descripteurs de moniteurs
- descripteur de variable conditionnelle : file de descripteurs de processus
- tableau de descripteurs de variable conditionnelles

Primitives nécessaires :

- création moniteur
- création variable conditionnelle
- initialisation moniteur
- appel de procédure = entrée moniteur; code procédure; sortie moniteur.
- **wait**, **signal**.


```
enter_monitor(int M) {
  descrM = tableau des moniteurs a l'entree M
  if (descrM.lock==1) {
    descrM.queue.put(exec_proc);
    exec_proc = 0;
  } else
    lock = 1;
  dispatcher();
}

exit_monitor(int M) {
  descrM = tableau des moniteurs a l'entree M
  if (! descrM.queue.empty()) {
    insere descrM.queue.top() a la fin de ready_proc
  } else
    lock = 0;
  dispatcher();
}
```

```
wait(int M, int CV) {
  descrCV = tableau des var. cond. a l'entree CV
  insere exec_rpoc a la fin de descrCV.queue
  exec_proc = 0;
  exit_monitor(M);
}

signal(int M, int CV) {
  descrM = tableau des moniteurs a l'entree M
  descrCV = tableau des var. cond. a l'entree CV
  if (! descrCV.queue.empty()) {
    insere descrCV.queue.top() a la
           fin de descrM.queue
  }
  dispatcher();
}
```