Les Moniteurs

Le reproche que l'on peut adresser aux régions critiques concerne d'une part, le coût excessif relatif aux réveils systématiques des processus bloqués, en particulier dans des systèmes monoprocesseur, et d'autre part la dissémination des différentes instructions (au niveau des points de synchronisation) dans le code d'un processus. Ainsi, pour localiser tous les points de synchronisation afin d'examiner toutes les formes d'utilisation des ressources, il est impératif d'étudier la totalité des processus concurrents. Cette façon de faire est quelque peu contraignante et influe grandement sur le rendement du programmeur.
Les moniteurs apportent une solution appréciable à ces problèmes en soulageant le programmeur de ces contraintes.
Ainsi, les moniteurs constituent la première approche qui tente une "séparation" entre la partie traitement et la partie contrôle correspondante d'un module fonctionnel dans un environnement concurrentiel. Autrement dit, un processus concurrent exécute normalement son algorithme ou sa partie traitement, et à la rencontre d'un point de synchronisation, il fait appel au moniteur pour obtenir l'accord du franchissement de ce point. le franchissement d'un point de synchronisation sera donc contrôlé au niveau du moniteur associé qui décidera de la poursuite ou du blocage de ce processus.

Fonctions

L'exclusion mutuelle des procédures est assurée par le mécanisme d'exécution du moniteur. A chaque procédure peut être attachée une condition à laquelle est associée une file d'attente. Fonctionnellement, le moniteur encapsule dans une même entité les données et les procédures qui les utilisent. L'accès au moniteur est restreint à des appels de procédures externes (des procédures internes au moniteur peuvent exister et leur utilité peut être limitée à faire passer le moniteur dans un état bien déterminé). Tout moniteur doit comporter une section d'initialisation exécutée avant tout accès. Cette section initialisera les variables globales du moniteur.

Les procédures du moniteur ordonnent leurs actions au moyen de primitives de haut niveau: ***wait*** et ***signal***.

Primitives du moniteur

Wait : primitive de blocage.
La primitive Wait(cond), où *cond* représente la condition d'attente, met le processus appelant dans la file d'attente associée à cond, et libère l'exclusion mutuelle du moniteur pour permettre l'accès aux processus demandeurs externes.

Signal : Primitive de réveil.
La primitive ***signal***(cond) opère comme suit:

S'il n'existe aucun processus bloqué dans la file associée à la condition cond, alors le processus invocateur, soit P, continue en séquence. Dans le cas contraire, le processus P est temporairement bloqué et un processus en attente dans la file associée à cond est réactivé. Le processus invocateur du signal sera réveillé lorsque le moniteur devient libre; c'est-à-dire: le processus réveillé par signal vient de quitter le moniteur ou se bloque une nouvelle fois. De plus, le processus bloqué par signal devient prioritaire sur les processus externes qui tentent d'appeler le moniteur.

A remarquer qu'un événement signal n'est pas mémorisé contrairement aux sémaphores, autrement dit, un signal non attendu est purement perdu.

Forme syntaxique du moniteur

```
nom_du_moniteur.Monitor
  % déclaration des variables %
Var    x:  Integer ,
       y: Boolean;


Condition c;


 % déclaration des procédures du moniteur %

procedure xx(yy);
begin


end;
procedure zz(tt);
begin


end;
 % initialisation  % la séquence d'instructions de la partie initialisation du moniteur %
 begin            %  est exécutée avant tout appel de l'extérieur aux procédures du   %
                  % moniteur. Elle a pour but de mettre ce dernier dans un état       %
                  %  initial adéquat.                                                 %

 end;
endmonitor
```

Variante de Kessels [Kes77]

La critique que l'on peut faire au mécanisme des moniteurs de Hoare concerne la primitive signal; en effet, le rôle de cette primitive consiste à réveiller, lorsque la condition de franchissement d'un point de synchronisation devient vraie, un processus bloqué en ce point.

Kessels proposa la variante suivante dans laquelle certains problèmes engendrés par la primitive signal sont évités par l'élimination de la primitive elle-même. On ne dispose donc que de l'unique primitive de blocage wait, et le réveil est à la charge du mécanisme d'exécution du moniteur qui le réalise de manière automatique. On aura donc:

Wait*(cond)*: où cond est maintenant une expression booléenne représentant de manière explicite la condition d'attente. Cond peut être composée de variables d'état du moniteur et éventuellement de constantes, elle peut être déclarée au début du moniteur. Cette primitive (wait(cond)) provoque le blocage du processus appelant P tantque la condition d'attente est vraie. Quand un processus quitte le moniteur ou se met en attente, les conditions associées aux primitives wait sont réévaluées. Si une des conditions examinées par le réévaluateur devient fausse, alors un des processus bloqué par cette condition est réveillé. Le réévaluateur est un processus du mécanisme d'exécution du moniteur qui se charge de la réévaluation des conditions de blocage.