Moniteurs de Hoare

1 Les Moniteurs

Dans les langages avec ***moniteurs***, il y a deux sortes d'entités :

- Les processus, au sens habituel : exécution de programme séquentiel,
- Les moniteurs : ce sont des modules à état, partagés entre les processus.

Un moniteur comporte :

- Un bloc de ***variables d'état*** : elles constituent la chose partagée.
- Une ***procédure d'initialisation*** (`init`), exécutée à la création du moniteur.
- Des ***procédures d'accès*** (`entrée`), permettant l'accès aux variables d'état du moniteur.

```
moniteur CMPT
   var cmpt: ent ; % variables d'état %
   init cmpt:=0 finit % initialisation %
   entrée incrémenter (incrément: ent) % procédure d'accès %
          cmpt:= cmpt+incrément
   fentrée
   entrée consulter: ent % procédure d'accès %

          résul cmpt
   fentrée
fmoniteur
```

Un moniteur garantit l'***exclusion*** entre les exécutions de ses procédures d'accès : un appel de moniteur est retardé tant qu'un autre processus se trouve dans le moniteur.
Un moniteur offre de plus des objets spéciaux appelés ***conditions*** qui permettent de programmer des attentes. Une condition est une file d'attente de processus. À l'intérieur d'un moniteur, un processus peut exécuter :

`attendre(c)` : provoque la mise en attente du processus dans la file `c` et relâche l'exclusion d'entrée dans le moniteur.

`reprendre(c)` : si au moins un processus est en attente dans la file `c`, un de ces processus est choisi et relancé, avec son contexte (l'état de ses variables locales) à partir de l'endroit où il a exécuté `attendre`. Le processus qui exécute `reprendre` est lui-même suspendu. Si aucun processus n'est en attente sur `c`, `reprendre(c)` n'a aucun effet.

A la ***sortie*** d'une procédure d'accès, s'il existe des processus signaleurs suspendus, un de ces processus est réactivé. Sinon il y a simplement relâche de l'exclusion d'entrée du moniteur.

Exemple d'utilisation des conditions : un moniteur tampon à une place.

```
moniteur TAMPON1
     var tampon: ent;
     var étatTampon: {vide,plein};
     var TamponVide: condition; var TamponPlein: condition;

   init étatTampon:=vide finit

   entrée prod (v:ent)
     si étatTampon=plein alors attendre(TamponVide) fsi;
     tampon:=v; étatTampon:=plein; reprendre(TamponPlein)
   fentrée

   entrée cons : ent
     si étatTampon=vide alors attendre(TamponPlein) fsi;
     résul tampon; étatTampon:=vide; reprendre(TamponVide)
   fentrée

fmoniteur
```

Les processus communiquent en appelant les entrées de moniteurs, par exemple :

```
processus P: ... TAMPON1.prod(53); ... y:= TAMPON1.cons; ...
```

Remarques : Il n'y a pas de nommage explicite des processus. Les processus n'interagissent que par l'intermédiaire d'un moniteur. Un moniteur non plus ne connaît pas les processus qui l'appellent. Les moniteurs sont des entités passives appelées par les processus.

2 Méthodologie

Le mécanisme des conditions permet de faire attendre explicitement un processus. C'est un moyen puissant et très général pour exprimer des synchronisations, mais dangereux à utiliser sans précaution. Une méthodologie d'utilisation des conditions est la suivante : si une certaine condition logique, exprimée par un prédicat ***P*** concernant l'état doit être satisfaite pour qu'une action puisse être poursuivie, on associe une condition c_p à ce prédicat. Aux endroits du programme où **P** doit être satisfait, on écrit :

`...` **si non P alors** `attendre(`c_p`) fsi;` {***P*** est vrai} `...`

et aux endroits où on sait que ***P*** devient vrai, on écrit :

`...` {***P*** est vrai ici} `reprendre(`c_p`)...`

Si ceci est partout respecté, c'est-à-dire si ***P*** est vérifié devant tout `reprendre(`c_p`)`, l'assertion ***P*** est bien évidemment vérifiée derrière tout `attendre(`c_p`)` car le langage assure le passage du contrôle exclusif au processus réveillé.

Exercice 1
Dans la programmation du tampon à une place, indiquer les prédicats associés aux conditions. Programmer, un tampon producteur consommateur à N places.

Exercice 2
Programmer, à l'aide d'un moniteur, un allocateur de N imprimantes. Ses entrées sont :
`demande : ent` rend en résultat le numéro de l'imprimante attribuée
`rend(impr: ent)` restitue l'imprimante (supposée attribuée) de numéro `impr`.

Exercice 3 - Lecteurs et rédacteurs

On s'intéresse au problème classique des lectures et des rédactions concurrentes sur un fichier. Les contraintes sont les suivantes :

(1) Une seule écriture à un instant donné.

(2) Une lecture ne peut être effectuée en même temps qu'une écriture.

Par contre, plusieurs lectures peuvent avoir lieu simultanément.

On utilise un moniteur dont la structure est donné ci-dessous :

```
moniteur LecRedac
  var nbLec: ent; var nbEcr: ent;
   ... conditions ...
  init nbEcr:= 0; nbLec:= 0 finit
  entrée debLec ...
  entrée debEcr ...
  entrée finLec ...
  entrée finEcr ...
fin moniteur
```

Les processus utilisateurs accèdent au fichier sous contrôle du moniteur. Toute session d'écriture est encadrée par **debEcr** et **finEcr** et toute session de lecture est encadrée par **debLec** et **finLec**.

1 - Sachant que **nbLec** désigne le nombre de lectures en cours et **nbEcr** le nombre d'écritures en cours, indiquer un prédicat **P** auquel doit satisfaire en permanence l'état du moniteur.

2 - Définir deux conditions permettant de programmer ce moniteur et caractériser chaque condition par un prédicat garanti vrai lorsqu'un processus sort de la file d'attente de cette condition. Programmer le moniteur.

Exercice 4 - Lecteurs et rédacteurs avec stratégie équitable

Avec la solution précédente, on court le risque suivant :
une population de lecteurs peut monopoliser l'usage du fichier et empêcher les écritures. Il est donc intéressant d'adopter une stratégie qui assure la prise en compte des écritures aussi bien que des lectures en un temps fini. Pour cela on adopte les règles suivantes :

(1) S'il y a des écritures en attente, les nouvelles lectures attendent la terminaison d'une écriture (celle en cours si effectivement c'est une écriture qui est en cours, la première en attente sinon).

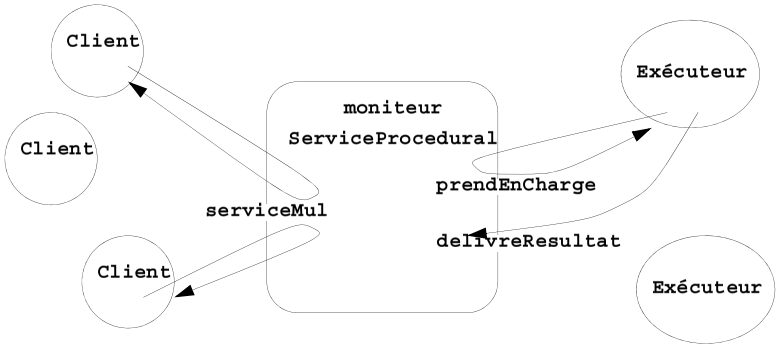
(2) A la fin d'une écriture, toutes les lectures en attente passent.

Programmer un moniteur qui réalise cette stratégie.

Exercice 5

On se propose de programmer un moniteur qui assure un «service procédural» à une collection de processus clients. Il s'agit d'un moniteur qui offre un service (une multiplication d'entiers dans notre exemple). Il y a par ailleurs plusieurs processus exécuteurs suceptibles de réaliser ce service (dans une application réelle, ces processus sont supposés être exécutés sur d'autres machines, éventuellement spécialisées, éventuellement distantes...).

Une demande de service est prise en charge par un exécuteur qui récupère les paramètres du service, exécute le service et délivre le résultat. Le client du service est mis en attente tant que le résultat n'est pas disponible. Ainsi, pour un client, tout se passe comme s'il avait simplement appelé une procédure.



Principe de réalisation :

Lorsqu'un client fait appel à **serviceMul**, il faut mémoriser les paramètres **x** et **y** du service et mettre en attente le client de l'obtention du résultat. Pour cela on utilise deux tableaux pour mémoriser les paramètres, **paramX** et **paramY** et un tableau de conditions **resultatPret** pour mettre les clients en attente. Lorsqu'un travail est pris en charge, l'exécuteur récupère les paramètres du travail et l'indice **iTrav** de la condition sur laquelle attend le client. C'est cet indice qui est rappelé lors de la fourniture du résultat afin de délivrer ce résultat au bon client.

Compléter le squelette de ce moniteur donné ci-après.