Université Toulouse III – Paul Sabatier
118 route de Narbonne
31062 Toulouse cedex 9

Travaux dirigés – n°1 à 2 – Moniteurs de Hoare

Quelques rappels

Un moniteur de Hoare est un objet évolué de synchronisation encapsulant :

- des opérations privées
- des opérations publiques qu'utiliseront des processus pour accéder à la ressource critique protégée par M
- des variables d'état, pour exprimer des prédicats notamment
- des variables « conditions » pour exprimer les blocages/déblocages des processus

Propriété fondamentale :

Les **opérations** du moniteur sont **exécutées en exclusion mutuelle.**

Si un processus exécute une opération, aucun autre processus ne peut exécuter une autre opération en même temps (premier niveau de blocage possible pour un processus : avoir l'accès à une opération du moniteur).

Sur une **variable condition** C, trois opérations sont possibles :

- **wait(C)**Le processus appelant est bloqué dans le file d'attente associée à C (deuxième niveau de blocage possible pour un processus : accéder à la ressource critique).
- **signal(C)** Le premier processus, s'il existe, de la file d'attente associée à C est débloqué (celui qui réveille perd l'accès au moniteur). Si aucun processus n'était bloqué, cette opération n'a aucun effet (pas de mémorisation comme avec l'opération V des sémaphores).
- **vide(C)**Savoir si la file d'attente associée à C est vide ou non.

Exercice 1 – Modèle des producteurs/consommateurs

On considère le modèle des producteurs-consommateurs dans lequel deux familles de processus accèdent à un buffer partagé pour y déposer (processus Producteurs) ou retirer des messages (processus Consommateurs). Le buffer est géré de manière circulaire. Les retraits s'effectuent dans l'ordre des dépôts.

On se propose d'écrire un moniteur de Hoare gérant l'accès à la ressource commune selon les différentes variantes énoncées ci-après.

La spécification du moniteur est la suivante :

```
Moniteur Prod_Conso {
  void deposer(. . . );    -- utilisée par un processus producteur
  void retirer(. . . );    -- utilisée par un processus consommateur
end Prod_Conso ;
```

Variante 1 – Buffer de N cases, messages d'un type unique

C'est la variante de base, dans laquelle la capacité du buffer est de N messages et la politique appliquée est celle décrite plus haut.

Variante 2 – Message de deux types, dépôts alternés

Dans cette variante, les messages considérés peuvent être de deux types (par exemple, noir/blanc ou recto/verso…). Un producteur dépose des messages d'un certain type. Les dépôts des messages se font de manière alternée. Les retraits se font selon la même politique que précédemment.

Variante 3 – Messages de deux types, choix du type de message retiré

Dans cette variante, on a toujours des producteurs de deux types mais les dépôts ne sont plus forcément alternés. On a aussi des consommateurs de deux types et un consommateur spécifie le type du message qu'il désire retirer. Les retraits se font toujours dans l'ordre des dépôts.

Questions

Pour chacune des variantes :

- Préciser la spécification du moniteur.
- Préciser les conditions de blocage et de réveil d'un processus producteur et d'un processus consommateur.
- En déduire les variables d'état et les variables « condition » gérées dans le moniteur.
- Donner le code du moniteur.

Université Toulouse III – Paul Sabatier
118 route de Narbonne
31062 Toulouse cedex 9

Travaux dirigés – n°3 – Moniteurs de Hoare

Modèle des lecteurs-rédacteurs

Le modèle des lecteurs-rédacteurs schématise une situation rencontrée dans la gestion de fichiers partageables ou dans l'accès à des bases de données.

Deux familles de processus accèdent à ces informations. Les lecteurs désirent seulement consulter l'information. Les rédacteurs désirent modifier cette information. Les lecteurs peuvent donc accéder à l'information en parallèle alors que les rédacteurs doivent y accéder en exclusion mutuelle.

Les comportements de ces processus sont donc les suivants :

Un lecteur	**Un rédacteur**

Boucler sur {
...
Demander à lire;
Faire la lecture;
Signaler la fin de lecture;
...
}
	Boucler sur {
	...
	Demander à écrire;
	Faire la modification;
	Signaler la fin d'écriture;
	...
	}

On se propose d'écrire un moniteur de Hoare gérant l'accès à la ressource commune selon la politique définie précédemment et selon les différentes variantes suivantes.

La spécification du moniteur se présente de la manière suivante:

```
Moniteur Lecteur_Redacteur {
  void debutLire();
  void finLire();
  void debutEcrire();
  void finEcrire();
end Lecteur_Redacteur ;
```

Variante 1

Quand aucun lecteur ne lit, les lecteurs et les rédacteurs ont la même priorité. En revanche, dès qu'un lecteur est en train de lire, tous les autres lecteurs qui le demandent peuvent également lire puisque les lectures peuvent être faites en parallèle, quel que soit le nombre de rédacteurs en attente.

Lorsqu'un rédacteur écrit, aucun autre client ne peut accéder à la ressource (ni lecteur, ni rédacteur).

Lorsque le rédacteur a terminé d'écrire, il essaye d'activer un rédacteur en priorité sur les lecteurs.

Variante 2 - Priorité des rédacteurs sur les lecteurs

On souhaite donner la priorité aux rédacteurs afin que l'information disponible ne soit pas obsolète pour le lecteur. Lorsqu'un rédacteur demande à accéder à la ressource, il doit donc l'obtenir le plus

tôt possible. Bien sûr, il ne lui est pas possible d'interrompre des lectures ou une écriture en cours. De même, il n'a pas de passe-droit vis-à-vis d'autres rédacteurs en attente (arrivés avant lui et non encore acceptés). En revanche, il est prioritaire par rapport aux lecteurs en attente.

Variante 3 - Gestion plus équitable des accès

À quelles situations erronées peuvent conduire les variantes 1 et 2 ?

Quelle solution proposer pour corriger de telles situations ?

Dans cette variante, un rédacteur qui termine d'écrire doit laisser l'accès en priorité à tous les lecteurs en attente à cet instant, et non au rédacteur suivant comme il est spécifié dans la variante 1. En revanche, les lecteurs qui arriveront ultérieurement (après cette demande d'écriture) devront respecter la règle de priorité des rédacteurs sur les lecteurs telle qu'elle est exprimée dans la variante 2.

Variante 4 - Gestion FIFO des accès

On suppose maintenant que l'accès aux données est effectué suivant une politique FIFO ; les requêtes sont traitées dans l'ordre de leurs arrivées.

Pour ordonner globalement les lecteurs et les rédacteurs en attente, tous les processus seront bloqués sur une même condition. En effet, il n'est pas possible de savoir si le 3^e rédacteur est arrivé avant ou après le 4^e lecteur lorsque lecteurs et rédacteurs sont rangés dans des files distinctes. On notera que, lors du réveil, il n'est pas possible (pour le « signaleur ») de distinguer un lecteur d'un rédacteur. Aussi, on peut être amené à réveiller un processus (lecteur ou rédacteur), puis le rebloquer par la suite si le déblocage s'avère impossible dans la situation actuelle.

Questions

Pour chacune des variantes :

- ☞ Préciser la spécification du moniteur.
- ☞ Préciser les conditions de blocage et de réveil d'un processus lecteur et d'un processus rédacteur.
- ☞ En déduire les variables d'état et les variables « condition » gérées dans le moniteur.
- ☞ Donner le code du moniteur.

Université Toulouse III – Paul Sabatier
118 route de Narbonne
31062 Toulouse cedex 9

Travaux dirigés – n°4 – Moniteurs de Hoare

Traitement de commandes

A l'occasion des fêtes de Noël, votre magasin préféré a décidé de mettre à la disposition de ses fidèles clients, un certain nombre (NB_GUICHETS) de guichets spéciaux pour prendre en charge leurs commandes. Le principe de fonctionnement de ces guichets est le suivant :

- Un client attend un guichet libre, y dépose sa commande, puis attend que celle-ci soit prête.
- Le traitement d'une commande comporte un certain nombre (NB_ETAPES) d'étapes successives et des employés spécialisés sont affectés à chacune de ces étapes. L'étape i du traitement d'une commande ne peut débuter que lorsque l'employé chargé de l'étape précédente (i-1) a terminé sa tâche.
- Lorsque sa commande a été traitée, le client quitte le guichet, pleinement satisfait de ce nouveau service.

Par exemple, en pratique, le magasin met trois guichets à disposition de ses clients et le traitement d'une commande consiste en quatre étapes : (1) établir la liste des articles commandés, (2) aller chercher les articles commandés, (3) les emballer avec du papier cadeau et (4) faire payer le client.

Problème

On considère deux types de processus : Client et Employe.

- Un processus Client dépose une commande à un guichet libre et attend que cette commande soit traitée avant de quitter ce guichet. Au plus NB_GUICHETS commandes peuvent être traitées en parallèle.
- Un processus Employé, spécialisé dans l'étape i, prend en charge la première commande en attente de cette étape, accomplit sa part de travail avant de rapporter la commande traitée à son guichet d'origine. Il peut alors prendre en charge une nouvelle commande. Lorsque la dernière étape a été exécutée sur une commande, le processus Client ayant formulé cette commande peut reprendre son exécution.

On suppose définis les types :

- NumeroEtape : qui désigne un entier compris entre 0 et NB_ETAPES
- NumeroGuichet : qui désigne un entier compris entre 1 et NB_GUICHETS
- Commande : qui désigne la commande établie par un client

On suppose aussi donné, le sous-programme :

- void appliquerEtape (NumeroEtape numEtape, Commande *uneCommande) ;

qui peut être utilisé par un processus Employe afin d'accomplir sa part de travail (i.e. l'étape numEtape) sur une commande donnée et ainsi lui apporter une modification.

On se propose de synchroniser, en utilisant un moniteur de Hoare nommé GererCmdes, des processus Client et des processus Employé pour que les commandes soient traitées le plus efficacement possible.

La spécification de ce moniteur est la suivante :

```
Moniteur GererCmdes {
    void commander (void) ;              // Utilisé par un Client
                                         // Utilisés par un Employe
    void commencerEtape (NumeroEtape etape, Commande *cmde, NumeroGuichet *guichet) ;
    void terminerEtape (Commande cmde, NumeroGuichet guichet) ;
}
```

Le comportement des processus Client et Employe est donc le suivant :

```
Processus Client {
    // Se rendre au magasin
    GererCmdes.commander() ;
    // Rentrer chez soi, satisfait
}
```

```
Processus Employe (NumeroEtape etapeAppliquee) {
  while (1) {
    GererCmdes.commencerEtape(etapeAppliquee,
                  &cmdeATraiter,  &guichetOrigine) ;
    appliquerEtape(etapeAppliquee, cmdeATraiter) ;
    GererCmdes.terminerEtape(cmdeATraiter,
                                  guichetOrigine) ;
  }
}
```

Questions

- ☞ Préciser les conditions de blocage et de réveil d'un processus Client et d'un processus Employe.
- ☞ En déduire les variables d'état et les variables « condition » gérées dans le moniteur.
- ☞ Donner le code du moniteur.