ESSI 2005-2006 **Programmation Concurrente**
Deuxième année

Corrigé du TD3 : Réseaux Asynchrones – Sémaphores et Moniteurs

1. ***Problème des Producteurs - Consommateurs***

1.1 Modifier les algorithmes PC2 et PC3 dans le cas où le tampon est circulaire de taille finie.

La modification est identique pour PC2 et PC3. Nous supposons l'existence d'un tableau T de taille c, muni de deux pointeurs prem qui donne l'indice de la première case libre et dern qui pointe sur la dernière case occupée. Initialement prem et dern sont égaux à 0.

Algorithme PC3 (tampon fini)

Producteur^i	**Consommateur^j**
Répéter	**Répéter**
Produire ;	Attendre(S) ; / tampon vide /
Attendre(C) ; / tampon plein /	Attendre(E) ;
Attendre(E) ; / section critique /	**elem := T(dern) ;(dern++) mod c ;**
T(prem) := elem ; (prem ++) mod c ;	Réveiller(E) ;
Réveiller(E) ;	Réveiller(C) ;
Réveiller(S) ;	Traiter ;

1.2 Même question dans le cas où le tampon est une pile de taille finie.

La modification est identique pour PC2 et PC3. Nous supposons l'existence d'un tableau T de taille c, muni d'un pointeur haut qui donne l'indice de la première case libre. Initialement haut est égal à 0.

Algorithme PC3 (tampon fini)

Producteur^i	**Consommateur^j**
Répéter	**Répéter**
Produire ;	Attendre(S) ; / tampon vide /
Attendre(C) ; / tampon plein /	Attendre(E) ;
Attendre(E) ; / section critique /	**(haut --) mod c ; elem := T(haut) ;**
T(haut) := elem ; (prem ++) mod c ;	Réveiller(E) ;
Réveiller(E) ;	Réveiller(C) ;
Réveiller(S) ;	Traiter ;

2. ***Algorithme des Ecrivains-Lecteurs avec sémaphore***

2.1 Soit L(t) le nombre de processus en train de lire et E(t) le nombre de processus en train d'écrire. Montrer que la formule logique I suivante est toujours vraie :

$$\forall t\ ,\ I(t) = (L(t) > 0 \Rightarrow E(t) = 0) \wedge ((E(t) > 0 \Rightarrow (E(t) = 1 \wedge L(t) = 0)) = vrai$$

Le problème des écrivains et des lecteurs correspond à la situation où un seul processus (écrivain) a accès à la mémoire pour modifier des données et où plusieurs lecteurs peuvent accéder simultanément à la mémoire sans la modifier. C'est donc une généralisation de l'exclusion mutuelle dans laquelle on interdit l'accès simultané :

- à un lecteur et à un écrivain (no read-write conflict)

- à deux écrivains (no write-write conflict)

La formule logique I correspond à dire que s'il y a les lecteurs actifs alors aucun écrivain n'est actif et que s'il y a des écrivains actifs alors il n'y en a qu'un et il n'y a pas de lecteur actif. Nous pouvons réécrire I(t) en :

$$\mathbf{I(t) = (E(t) = 0 \vee L(t) = 0) \wedge (E(t) = 0 \vee (E(t) = 1 \wedge L(t) = 0))}$$
$$\mathbf{= (E(t) = 0) \vee (E(t) = 1 \wedge L(t) = 0)}$$

qui correspond à dire que soit il n'y a aucun écrivain actif, soit il y en a et aucun lecteur actif. I(0) est vraie puisque E(0) = 0. Supposons la formule vraie pour t-1 et démontrons la pour t.

Si E(t-1) = 0 et L(t-1) = 0 alors Verrou = 1 et Prem_Dern_Lect = 1. Si E(t) = 1, alors Verrou = 0. Si un lecteur a exécuté Début_lecture alors nb_lect = 1 et le lecteur est suspendu. Donc L(t) = 0.

Si E(t-1) = 0 et L(t-1) > 0 alors Verrou = 0. Aucun écrivain ne pourra débuter une écriture. Donc E(t) = 0.

Si E(t-1) = 1 et L(t-1) = 0 alors Verrou = 0. Aucun nouvel écrivain ne pourra débuter une écriture et donc E(t) = 0. Ce qui démontre la formule I.

2.2 Montrer que si Verrou n'est pas fort, alors le processus Ecrivain peut être en état de famine.

La gestion de la file d'attente ne garantissant pas un traitement équitable des processus en attente. Certains peuvent rester bloqués indéfiniment : les lecteurs peuvent donc faire une conjuration pour bloquer l'écrivain.

2.3 Montrer que l'algorithme Ecrivains-Lecteurs avec un Verrou fort résout le problème de des écrivains et des lecteurs, sans interblocage, sans famine et de manière équitable.

Pour démontrer la viabilité de cette solution, il faut montrer qu'un lecteur (écrivain) qui désire lire (écrire) finira par y être autorisé. Un tel lecteur L va exécuter Début_lecture. Supposons d'abord qu'il n'y a aucun écrivain actif. Si Prem_Dern_Lect = 1 alors il n'y a aucun lecteur actif. Donc L progresse, place le verrou à 1 et termine son protocole Début_lecture. Il est donc autorisé à lire. Si Prem_Dern_Lect = 0 alors L va être placé dans la FIFO correspondante. Aucun écrivain ne pourra placer le verrou et donc les lecteurs précédents progressent et finiront par réveiller L. S'il y a un écrivain actif, si Prem_Dern_Lect = 1 alors L va attendre sur le verrou correspondant à l'écrivain qui va progresser et finira par réveiller L. Si Prem_Dern_Lect = 0 alors L va être placé dans la FIFO correspondante. L'écrivain finira par réveiller un lecteur précédent.

Pour démontrer la progression des écrivains, on utilise un raisonnement similaire.

2.4 Modifier l'algorithme pour limiter le nombre de lecteurs simultanés.

La solution avec des sémaphores est la suivante. Pour l'écriture, l'accès à la base doit se faire en exclusion mutuelle : un sémaphore binaire **Verrou** initialisé à 1. Les protocoles sont donc :

```
Début_écriture ;                    Fin_écriture ;
        Attendre(Verrou) ;                  Réveiller(Verrou) ;
```

Pour la lecture, nous devons d'abord garantir que le nombre de lecteurs est inférieur au nombre limite de lecteurs autorisés, noté **lect_max**. Puis nous devons garantir qu'aucun écrivain ne peut accéder à la base. Donc le premier lecteur doit verrouiller la base en utilisant le Verrou. Par contre, les autres lecteurs peuvent accéder sans problème. Pour la fin de lecture, le dernier lecteur doit lever le Verrou. Nous devons donc utiliser un compteur **nb_lect**, initialisé à 0, permettant de connaître le nombre de lecteurs accédant simultanément à la base, un sémaphore **Lect_Aut**, initialisé à lect_max, garantissant le respect de la limitation du nombre de lecteurs simultanés et un deuxième sémaphore binaire **Prem_Dern_Lect**, initialisé à 1, garantissant que les opérations sur nb_lect s'exécutent en exclusion mutuelle.

Début_lecture ;
    Attendre(Lect_Aut) ;
    Attendre(Prem_Dern_Lect) ;
    nb_lect++ ;
    Si nb_lect = 1 **alors** Attendre(Verrou) ;
    Réveiller(Prem_Dern_Lect) ;

Fin_lecture ;
    Réveiller(Lect_Aut) ;
    Attendre(Prem_Dern_Lect) ;
    nb_lect-- ;
    Si nb_lect = 0 **alors** Réveiller(Verrou) ;
    Réveiller(Prem_Dern_Lect) ;

*3. **Algorithme des Ecrivains-Lecteurs avec moniteur***
3.1 Proposer un algorithme en utilisant un moniteur.

Il nous faut donc définir les quatre protocoles garantissant les 2 conditions précédentes. Nous allons résoudre ce problème en utilisant un moniteur Lecture-Ecriture.

Algorithme Moniteur Lecture-Ecriture
Variable entière nb_lect **initialisée à** 0
Variable booléenne ecrit_en_cours **initialisée à** faux
Variable conditionnelle OK_lect, OK_ecrit

Début_écriture
    Si (nb_lect > 0) ∨ (ecrit_en_cours) **alors**
        Attendre(OK_ecrit) ;
    ecrit_en_cours := vrai ;

Début_lecture
    Si (ecrit_en_cours) **alors**
        Attendre(OK_lect) ;
    nb_lect ++ ;
    Réveiller(OK_lect) ;

Fin_écriture
    ecrit_en_cours := faux ;
    Réveiller(OK_lect) ;
    Réveiller(OK_ecrit) ;

Fin_lecture
    nblect -- ;
    Si nb_lect = 0 **alors Réveiller**(OK_ecrit) ;

3.2 Utiliser l'algorithme de simulation d'un sémaphore par un moniteur pour transformer l'algorithme du cours en un algorithme utilisant un moniteur. Comparer les deux algorithmes.

La solution avec des sémaphores est la suivante.

Début_écriture ;
    Attendre(Verrou) ;

Fin_écriture ;
    Réveiller(Verrou) ;

Début_lecture ;
    Attendre(Prem_Dern_Lect) ;

Fin_lecture ;
    Attendre(Prem_Dern_Lect) ;

```
nb_lect++ ;                          nb_lect-- ;
Si nb_lect = 1 alors Attendre(Verrou) ;   Si nb_lect = 0 alors Réveiller(Verrou) ;
Réveiller(Prem_Dern_Lect) ;          Réveiller(Prem_Dern_Lect) ;
```

Nous avons vu que l'on pouvait simuler un sémaphore binaire par un moniteur :

```
Algorithme Moniteur Simulateur_Sémaphore
    Variable booléenne occupé initialisée à faux
    Variable conditionnelle Libre
    Sém_Attendre                          Sém_Réveiller
        Si occupé alors Attendre(Libre) ;     occupé := faux ;
        occupé := vrai ;                      Réveiller(Libre) ;
```

Effectuons les substitutions des sémaphores Verrou et Prem_Dern_Lect par leurs simulations.

```
Algorithme Moniteur Lecture-Ecriture_Sim
Variable entière nb_lect initialisée à 0
Variable booléenne ecrit_occup, lect_imp initialisée à faux
Variable conditionnelle OK_lect, OK_ecrit

Début_écriture ;                     Fin_écriture ;
    Si ecrit_occup alors Attendre(OK_ecrit) ;   ecrit_occup:= faux ;
    ecrit_occup := vrai ;            Réveiller(OK_ecrit) ;

Début_lecture ;                      Fin_lecture ;
    Si lect_imp alors Attendre(OK_lect) ;   Si lect_imp alors Attendre(OK_lect) ;
    lect_imp := vrai ;               lect_imp := vrai ;
    nb_lect++ ;                      nb_lect-- ;
    Si nb_lect = 1 alors             Si nb_lect = 0 alors
      Si ecrit_occup alors Attendre(OK_ecrit) ;          ecrit_occup:= faux ;
                    ecrit_occup := vrai ;                Réveiller(OK_ecrit) ;
    lect_imp := faux ;               lect_imp := faux ;
    Réveiller(OK_lect) ;             Réveiller(OK_lect) ;
```

En comparant les deux solutions, on constate :

- que ecrit_occup et ecrit_en_cours jouent le même rôle
- que lect_imp est remplacé par des tests sur nb_lect.