TD 3 - ASR7 Programmation concurrente

24 avril 2017

I Lamport's bakery algorithm

En 1974, Leslie Lamport apporte une nouvelle solution au mutex (tiré de wikipédia) :

```
Initialiser toutes les cases de COMPTEUR à 0.
Initialiser toutes les cases de CHOIX à 0 (ou FAUX).

CHOIX[ID] = 1

MAX = 0
// Calcul de la valeur maximale des tickets des autres threads
POUR J = 0 à N - 1 FAIRE
  CJ = COMPTEUR[J]
  SI MAX < CJ ALORS
     MAX = CJ
  FIN SI
FIN POUR J
// Attribution du nouveau ticket
COMPTEUR [ID] = MAX + 1

// Fin de la phase d'attribution du ticket
CHOIX[ID] = 0

// Boucle sur tous les fils d'exécution
POUR J = 0 à N - 1 FAIRE
  // On attend que le fil considéré (J) ait fini de s'attribuer un ticket.
  TANT QUE CHOIX[J] == 1 FAIRE /* (1) */
     RIEN
  FIN TANT QUE
  // On attend que le fil courant (ID) devienne plus prioritaire que le fil considéré (J).
  TANT QUE COMPTEUR[J] <> 0 ET /* (2) */
     (COMPTEUR[J] < COMPTEUR[ID] OU /* (3) */
     (COMPTEUR[J] == COMPTEUR[ID] ET J < ID)) /* (4) */
  FAIRE
     RIEN
  FIN TANT QUE
FIN POUR J

Section Critique

// Sortie de section critique
COMPTEUR[ID] = 0
```

Q.I.1) - Montrez que l'algorithme garantit l'exclusion mutuelle de N fils d'exécution. Précisez sous quelles hypothèses.

II Ordonancement avec des mutex

Nous utilisons un ordonnancement préemptif avec priorité [1]. Nous allons utiliser un jeux de tâches qui mélange des tâches périodiques et des tâches ponctuelles. De plus, deux tâches partagent un mutex.

Tâche	Date(s) d'arrivée(s)	Priorité	Durée	Remarque
A	0, 6, 12, 18, 24, 30	10	1	Périodique
B	0, 10, 20, 30	8	4	Périodique, à chaque itération, la tâche doit acquérir le mutex à la fin du temps 1 et la libérer à la fin du temps 3.
C	18	5	6	Ponctuelle
D	0	1	8	Ponctuelle, à la fin du temps 6, la tâche acquière le mutex et le conserve jusqu'à la fin du temps 8.

Q.II.1) - Faire l'ordonnancement de ces tâches sur 32 unités de temps.

Q.II.2) - Quel est le temps de réponse de chaque tâche ?

Q.II.3) - Que remarque-t'on aux temps 21 à 27 ?

III Lecteur rédacteur avec tampon (CCF automne 2010)

On souhaite améliorer les performances d'une base de données. Ses tables doivent être accessibles en lecture et écriture. Pour le moment, tout est géré par des verrous de type *lecteur/rédacteur* [2]. Après différentes mesures, on s'est aperçu que l'essentiel des blocages vient des requêtes d'écriture. Les verrous ont été programmés de manière équitable, et les requêtes en écriture sont rares par rapport aux lectures, mais :

— lorsqu'une requête en écriture arrive, elle doit attendre que toutes les requêtes en lecture se terminent ce qui peut prendre du temps,
— pendant qu'elle attend, une requête en écriture bloque les nouvelles requêtes en lecture.
— si l'écriture est longue elle bloque toute autre requête pendant son exécution.

III.1 Proposition de solution

Pour y remédier, on utilisera un buffer. C'est-à-dire que la table sera dupliquée en mémoire.

— la première, la *table active* qui sert au lectures ;
— la seconde, la *table copie* qui est la seule à pouvoir être modifiée.

On utilisera à la fois des verrous de type *lecteur/rédacteur* et des verrous simples (*mutex*). Lorsqu'une requête a besoin de modifier la table :

— elle modifie la *copie* (pendant que les lectures se font sur la table *active*) ;
— elle échange les deux tables ;
— elle synchronise les modifications sur l'ancienne table *active* (pendant que les lectures utilisent l'ancienne *copie*).

Cela signifie qu'il faut faire 2 modifications au lieu d'une mais cela bloque moins les lectures, on espère donc gagner du temps.

Le principe est donc :

— Pour la lecture :

1. Plus la valeur de priorité est importance plus la tâche est prioritaire
2. Plusieurs lecteurs peuvent lire ensemble, un seul rédacteur peut écrire à la fois et il interdit la lecture en même temps.

```
lecture(reqlect) {
  int table;

  // Trouver la table active par lecture d'une variable partagée
  // de gestion des tables;
  table = ChoixTableActive();

  // Accéder en lecture à la table active.
  res = LireTable(table, reqlect);

  return res;
}
```

— Pour l'écriture :

```
modifie(reqmodif) {
  // Trouver la table copie et la table active;
  tactive = ChoixTableActive();
  tcopie = ChoixTableCopie();

  // Appliquer la requête à la copie
  ModifieCopie(tcopie, reqmodif);

  // Inverser le rôle des 2 tables c'est à dire modifier
  // la variable partagée de gestion des tables;
  InverseTable();

  // Synchroniser les deux tables c'est à dire lecture de
  // tcopie (l'ancienne copie) et écriture de tactive (l'ancienne active).
  Synchronise(tactive, tcopie);
}
```

III.2 Travail à faire

Vous disposez :

— Des mutex simples et leurs primitives :
  — Mutex M pour déclarer le mutex ;
  — Lock (M) pour bloquer le passage ;
  — Unlock(M) pour débloquer le passage.
— Des mutex lecteur/rédacteur et leurs primitives :
  — MutexLR Lr pour déclarer le mutex ;
  — DebutLecture(Lr) pour annoncer le début de la lecture ;
  — FinLecture(Lr) pour annoncer la fin de la lecture ;
  — DebutEcriture(Lr) pour annoncer le début de l'écriture ;
  — FinEcriture(Lr) pour annoncer la fin de l'écriture.
— Des fonctions :
  — table = ChoixTableActive(); fonction qui renvoie le numéro de la table active ;
  — table = ChoixTableCopie(); fonction qui renvoie le numéro de la table de copie ;
  — LireTable(table, reqlect) pour appliquer la requête en lecture reqlect à la table table ;
  — Modifie(table, reqmodif) pour appliquer la modification de la requête reqmodif à la table table ;
  — InverseTable() pour inverser le rôle des tables ;
  — Synchronise(table1, table2) pour synchroniser la table table1 à partir de la table table2.

En utilisant ces primitives, vous devez :

Q.III.1) - Donner l'algorithme du lecteur : lecture(reqlect).

Q.III.2) - Donner l'algorithme du rédacteur : modifie(reqmodif).

Q.III.3) - En supposant que la modification et la synchronisation durent $10ms$ que le choix de la table active et l'inversion des tables durent $0.1ms$, que la lecture dure $20ms$. Donnez le temps nécessaire dans le pire des cas (avant et après la modification) pour :

3(a) - une requête en écriture ;

3(b) - une requête en lecture ;

3(c) - que remarque-t'on ?