Synchronisation - Sémaphores

Ensimag 2A

1 Sémaphores

1.1 Présentation

Un sémaphore `s` est un objet de synchronisation avec les attributs suivants :

- un compteur entier *privé*, noté `s.c`, qui est initialisé à une valeur `c0` à la création du sémaphore ;
- une file d'attente *privée* pour gérer les threads bloqués, notée `s.f`.

Un sémaphore possède également deux méthodes *publiques* qui sont exécutées en *exclusion mutuelle* et qui vont utiliser les attributs privés :

```
// P = Prolagen, néerlandais
// → néologisme de essayer et baisser
s.P() {
   s.c--;
   if (s.c < 0) {
      // Le thread se bloque dans s.f
      s.f.wait();
   }
}

// V = Verhogen, néerlandais
// → augmenter
s.V() {
   s.c++;
   if (s.c <= 0) {
      // Débloque un thread de s.f
      s.f.signal();
   }
}
```

A partir de la valeur courante `s.c` d'un sémaphore `s`, on peut dire que :

- `c >= 0` est le nombre de ressources que peuvent prendre des threads appelant `P()` sans se bloquer ;
- `c < 0` est l'opposé du nombre de threads bloqué dans $s.f$.

Dans le cadre de ce TD on supposera que les threads sont débloqués dans un ordre FIFO.

1.2 Échauffement

Pour s'échauffer à l'utilisation des sémaphores, on va "simuler" le comportement d'un mutex avec un sémaphore. Attention néanmoins, mutex et sémaphores sont différents et pour faire de l'exclusion mutuelle il faut **toujours** utiliser un mutex et non pas un sémaphore.

Question 1 *En utilisant un sémaphore, reproduire le comportement d'un mutex pour former une exclusion mutuelle autour d'une section critique.*

On imagine maintenant un serveur auquel il est possible de se `connecter()` et de se `deconnecter()` pour y effectuer un travail. Cependant, seul 10 threads peuvent y accéder en même temps.

Question 2 *Sécurisez l'accès au serveur avec un sémaphore.*

2 Rendez-Vous

On dispose d'un ensemble de thread qui peut être divisé en deux groupes : des threads de classe A et des threads de classe B. On souhaite que ces deux classes de threads s'attendent mutuellement, c'est à dire que un thread doit être bloqué à un point s'il ne dispose pas d'un processus de l'autre classe.

Autrement dit, d'un point de vue "ressource", A veut prendre un B et B veut prendre un A.

Question 3 *En utilisant les sémaphores, écrire deux fonctions `rdv_A()` et `rdv_B()` pour satisfaire un rendez-vous entre les deux classes.*

Question 4 *Toujours en utilisant des sémaphores, faire un rendez-vous pour 3 classes de threads : `rdv_A()`, `rdv_B()` et `rdv_C()`.*

3 Barrière

N threads arrivent les uns après les autres à une barrière et attendent là jusqu'à ce que les N soient arrivés.

Question 5 *En utilisant des sémaphores, faire une barrière qui bloque tous les threads jusqu'à ce que N soient arrivés. On ne demande pas que la barrière soit réutilisable.*

4 Producteurs-Consommateurs

On dispose de deux types de threads qui s'échangent des messages par le biais d'un buffer. Les producteurs écrivent des messages dans le buffer alors que les consommateurs les suppriment. Les règles sont les suivantes :

— le buffer est de taille bornée ;
— s'il n'y a pas de messages dans le buffer, les consommateurs se mettent en attente ;
— si le buffer est plein, les producteurs se mettent en attente.

Chaque case pleine/vide peut être vue comme une ressource. Pour écrire il faut qu'il y est une case vide. Il faut faire attention à garantir l'exclusion mutuelle pour la manipulation des indices de cases.

Question 6 *Écrire* `produire(Message m)` *et* `consommer(Message *m)` *répondant au cahier des charges du Producteurs-Consommateurs.*

5 Lecteurs-Rédacteurs

On dispose de deux types de threads voulant accéder à une ressource partagée (BDD ou fichier par exemple). Les lecteurs veulent accéder à la ressource seulement pour la consulter et les rédacteurs veulent la modifier. On veut optimiser la politique d'accès tout en garantissant un fonctionnement correct en faisant respecter les règles suivantes aux threads :

— 1 seul thread à la fois écrit la ressource ;
— aucun thread ne peut lire pendant une écriture ;
— plusieurs lectures sont possibles en même temps.

On peut représenter le système par le code suivant :

```
lecteur() {
  debut_lire();
  ... // lecture ressource
  fin_lire();
}
redacteur() {
  debut_redac();
  ... // modification ressource
  fin_redac();
}
```

La ressource à manipuler ici est unique. Donc un seul sémaphore binaire suffit pour savoir si elle est affecté aux lecteurs ou aux rédacteurs. Cependant, il va falloir que ce soit le premier lecteur qui la prenne et le dernier lecteur qui la rende. Il va donc falloir compter les lecteurs et le faire en exclusion mutuelle.

Question 7 *Écrire* `debut_lire()`, `fin_lire()`, `debut_redac()` *et* `fin_redac()` *répondant au cahier des charges du Lecteurs-Rédacteurs. Tant qu'il y a des lectures en cours, on donnera la priorité aux lecteurs.*

On veut maintenant que les processus soient ordonnés par groupe de lecteurs FIFO avec les rédacteurs, c'est à dire que de nouveaux lecteurs ne rentrent pas si un rédacteur attend d'entrer. Il suffit de bloquer les processus dans un ordre FIFO pour bloquer les lecteurs qui arrivent après un écrivain. Un seul sémaphore servant de "sas" suffit.

Question 8 *Écrire* `debut_lire()`, `fin_lire()`, `debut_redac()` *et* `fin_redac()` *répondant au cahier des charges du Lecteur-Rédacteur sans priorité aux lecteurs.*

On veut maintenant que les threads soient ordonnés par ordre d'arrivée indépendamment de leur type. Autrement dit, de nouveaux lecteurs ne rentrent pas si un rédacteur attend d'entrer. Pour ça, il suffit de bloquer les threads dans un ordre FIFO pour bloquer les lecteurs qui arrivent après un écrivain. Comme les sémaphores de ce TD débloquent les threads dans un ordre FIFO, on utilisera un deuxième sémaphore binaire servant de sas d'entrée.

Question 9 *Écrire* `debut_lire()`, `fin_lire()`, `debut_redac()` *et* `fin_redac()` *avec ordonnancement par ordre d'arrivée.*

6 Les Philosophes

Un groupe de N philosophes sont attablés autour d'une table ronde et ils vont manger des pâtes. Pour manger des pâtes, chaque philosophe a besoin de deux fourchettes. Malheureusement ils n'ont qu'une fourchette par personne. Chaque fourchette est donc posée entre deux philosophes. Le but va être de synchroniser les philosophes avec deux fonctions `prendre_fourchettes` et `poser_fourchettes`.

Le philosophe i exécute le code suivant :

```
while(1) {
  penser();
  prendre_fourchettes(i);
  manger();
  poser_fourchettes(i);
}
```

La première intuition est de considéré chaque fourchette comme une ressource à gérer. Les fourchettes sont placées et numérotées. Le philosophe numéro i prend la fourchette i et $(i+1)\%N$.

Question 10 *Effectuer la synchronisation avec un sémaphore par fourchette.*

Question 11 *Pourquoi cette solution ne fonctionne pas ?*

Par la suite, on va faire en sorte de synchroniser les philosophes de façon à ce qu'ils prennent les deux fourchettes en même temps, ou aucune. Pour cela un philosophe aura son propre sémaphore (dit sémaphore privé) pour se bloquer/être débloqué individuellement.

Question 12 *Un philosophe peut avoir plusieurs états logiques. Combien ? Lesquels ?*

Question 13 *Quelles sont les conditions pour qu'un philosophe entre dans l'état MANGE ?*

Question 14 *Écrire une fonction* `test_mange(i)` *qui débloque le philosophe i si les conditions de la question précédente sont validées.*

Question 15 *Écrire les fonctions* `prendre_fourchettes(i)` *et* `poser_fourchettes(i)` *en utilisant la fonction* `test_mange(i)`.

Une autre "solution" consiste à introduire une différence entre les philosophes. On va considérer que tous les philosophes sont gauchers (ils prennent la fourchette de gauche en premier) sauf le dernier qui est droitier.

Question 16 *En utilisant un sémaphore par fourchette, faire la synchronisation en introduisant un droitier parmi les philosophes.*

Une dernière solution consiste à limiter le nombre de philosophes pouvant s'asseoir à la table (qui pourrait en accueillir N) à $N-1$. Pour cela il faut filtrer l'entrée de la salle à manger pour en limiter l'accès à au plus $N-1$ philosophes simultanément, et laisser le dernier penser au sens de la vie le ventre vide. Il pourra cependant entrer si un autre lui laisse la place, mais c'est une autre histoire.

Question 17 *En utilisant au minimum un sémaphore par fourchette, faire la synchronisation en limitant le nombre de philosophes prenant des fourchettes à $N-1$.*