TD

L3–UPJV

Septembre 2025

Dans le dernier TD, nous avons étudié des méthodes logicielles d'exclusion mutuelle. Les solutions que nous avons élaborées étaient restreintes à l'exclusion de deux threads (ou processus) uniquement. Ici, nous allons dériver une solution qui généralise l'exclusion mutuelle à un nombre quelconque de threads.

Exercices 1 *Synchronisation par ticket*

Une idée simple est d'utiliser le principe du ticket numéroté employé par les administrations pour gérer les files d'attente : quand un utilisateur arrive, il prend un ticket numéroté à un distributeur, les numéros vont croisant, et il attend que son numéro s'affiche sur un écran pour accéder au service. Le code de la figure 1 implémente ce mécanisme pour les threads.

1. *Expliquer comment fonctionne le code.*
2. *Cette solution n'est pas correcte, expliquer pourquoi (donner un exemple d'exécution qui ne fonctionne pas).*

Exercices 2 *Synchronisation par ticket sans compteur commun*

À partir du code précédent, dérivez une solution où on n'utilise plus les variables globales `next` *et* `number` *(qui indique qui est le prochain thread à rentrer en section critique), mais où chaque thread compare la valeur de son ticket par rapport à la valeur des autres avant de décider de rentrer en section critique.*

Exercices 3 *Barrière de synchronisation*

Une barrière de synchronisation permet de synchroniser tous les threads ou processus à un endroit précis de leur programme. Lorqu'un thread atteint ce point, il attend que tous les autres threads aient atteint le point de synchronisation avant de continuer.

En utilisant les variables de conditions, concevez la structure de données et les deux fonctions (initialisation et barrière) qui implémentent cette fonctionnalité. La structure de donnée se nommera `barrier_t`*, la fonction d'initialisation s'appellera* `barrier_init(barrier_t *b,int n)` *où* `n` *est le nombre de threads qui participent à la barrière, et la fonction barrière se nommera* `barrier(barrier_t *b)`*, à l'appel de cette fonction, un thread attend que les* `n-1` *autres threads aient aussi appelé cette fonction avant de continuer.*

```
int ticket[N]; // Ticket pour chaque thread, il y a N threads
int number=1; // Distributeur de tickets
int next=1; // numéro du prochain ticket qui peut entrer en SC

void *foo(void *arg) {
  int k=atoi((char *)arg); // k est le numéro du thread
  while (1) {
    ticket[k]=number; number=number+1; // Je prend mon ticket
    while (ticket[k]!=next); // J'attend mon tour
    ... // On est dans la section critique (SC)
    next=next+1;
  }
}
```

FIGURE 1 – Exclusion mutuelle par ticket