Td concurrence

02 et 03 décembre 2024

1

On s'intéresse à un bus touristique pouvant contenir C passagers.
On suppose que l'on a $n > C$ passagers qui attendent leur tour, puis se remettent en attente à l'arrêt du bus dès qu'ils ont fini pour le revoir.
Le bus ne démarre que lorsqu'il est plein.
Pour formaliser ce problème, on utilise des fonctions fictives :

- `board` et `unboard` permettent au passager de monter et de descendre ;
- le bus doit appeler les fonctions `load` lorsqu'il se remplit, `run` lorsqu'il démarre son tour et `unload` lorsqu'il se vide.

Attention, les passagers ne peuvent pas descendre avant que le bus ait ouvert ses portes pour une fin de tour avec `unload` et ne peuvent pas monter avant que le bus ait ouvert ses portes pour un nouveau tour avec `load`.

2.1 Écrire les fonctions en pseudo-code correspondant au bus et aux passagers **sans** prendre en compte les problèmes de synchronisation dans un premier temps.

2.2 Proposer une solution en pseudo-code utilisant deux compteurs protégés par des mutexs et quatre sémaphores.

2.3 Votre solution peut-elle être utilisée dans le cas où l'on a plusieurs bus ? Justifier.

2.4 Proposer une nouvelle solution pour le pseudo-code du bus dans le cas où il y a m bus numérotés en respectant les règles suivantes :

- un seul bus peut ouvrir ses portes aux passagers à la fois ;
- plusieurs bus peuvent faire un tour en même temps ;
- les bus ne peuvent pas se doubler donc ils se chargent et se déchargent toujours dans le même ordre ;
- un bus doit avoir fini de décharger avant qu'un autre bus vienne décharger.

2 Dans une garderie, la loi impose qu'il y ait toujours au moins un adulte pour trois enfants. On va écrire une fonction utilisée par les thread "adultes" et une pour les threads "enfants".

Une première solution serait la suivante :

On utilise une sémaphore partagé :

```
multiplex = Semaphore(0)
```

Fonction adulte :

```
multiplex.signal()
multiplex.signal()
multiplex.signal()

# critical section

multiplex.wait()
multiplex.wait()
multiplex.wait()
```

Fonction enfant :

```
multiplex.wait()

 # critical section

 multiplex.signal()
```

Expliquer pourquoi cette solution ne permet pas toujours aux adultes de partir s'ils le peuvent.

Proposer une solution à ce problème.

Expliquer pourquoi, même avec la solution proposée, on peut se retrouver avec par exemple deux adultes présents, quatre enfants et pourtant plus aucun enfant qui ne peut venir.

Proposer une solution qui permet à tout moment d'accepter un total de $3n$ enfants si n adultes sont encore présents. On pourra utiliser 4 compteurs permettant de compter le nombre d'enfants, le nombre d'adultes ainsi que les adultes sur le départ et les enfants en attente, un mutex et deux sémaphores.

3 On considère ici les fonctions `f1` et `f2`. On va lancer un thread pour chacune de ces deux fonctions. En supposant que les opérations $x = x + 1$, $y = y + 1$ et $y = y + 2$ son atomiques donner les valeurs finales possibles pour x et y.

```
int x=0;
int y=0;
void* f1(void* arg){
   if(x==0){
      x=x+1;
      y=y+1;
   }
}

void* f2(void* arg){
   if(x==0){
      x=x+1;
      y=y+2;
   }
}
```

4 La boutique du coiffeur est composée d'une salle d'attente contenant `NB_CHAISES` chaises et du salon où se trouve la chaise du coiffeur. Lorsque le coiffeur a fini de couper les cheveux d'un client, il fait entrer le client suivant dans le salon. Si la salle d'attente est vide, le coiffeur s'y installe pour dormir. Si un client trouve le coiffeur endormi, il le réveille. Sinon, il s'installe dans la salle d'attente s'il reste de la place (et rentre chez lui sinon).

Dans cet exercice vous utiliserez des sémaphores dont vous préciserez les valeurs initiales et vous pourrez vous contenter de noter `wait(s)` et `post(s)` pour les opérations d'acquisition et de restitution d'un sémaphore noté `s`.

Initialisation : `places = NB_CHAISES %représente le nombre de places disponibles`

Protocole du coiffeur :

```
Tant que (true) :
      si (places<NB_CHAISES)
           places ++
           coupe cheveux
```

Protocole de chaque client :

```
si (places>0)
   places --
   se faire couper les cheveux
sinon
   rentrer chez soi les cheveux longs
```

1. Il s'agit maintenant d'ajouter les synchronisations nécessaires au programme ci-dessus. Le premier problème à résoudre est une condition de compétition entre les clients lorsqu'ils rentrent dans la salle d'attente. Corrigez ce problème.
2. Assurez-vous ensuite que le coiffeur ne commence pas à couper les cheveux tant que le client n'est pas prêt.
3. Enfin, assurez-vous que le client ne s'assoit pas sur le siège tant que le coiffeur n'est pas prêt.