Les réponses aux questions de du TP Exclusion mutuelle

Guohao DAI, Groupe L-2
Deuxième année - Département Sciences du Numérique
2021-2022

Exclusion mutuelle

1. La classe *Peterson* fournie dans l'archive propose une implémentation du protocole d'exclusion mutuelle de Peterson vu en cours. Vérifiez et complétez éventuellement cette implémentation.

Solution : Veuillez-vous référer à l'implémentation spécifique dans le fichier de retour *Peterson.java*

2. L'ordre des deux premières affectations de la méthode *entrer()* (lignes 29 et 30 : *Peterson.demande[id] = true;* et *Peterson.tour = di ;*) est-il important ? Pourquoi ?

Solution : L'ordre des deux premières affectations est important. Le premier affectation signifie "le thread actuel veut accéder à la ressource". Le second signifie "laisser les ressources du thread actuel vers l'autre thread". Si la commande est échangée, le thread qui veut actuellement accéder à la ressource attend longtemps car il donne la ressource à l'autre processus

3. La classe *java.util.concurrent.atomic.AtomicBoolean* propose une implantation logicielle de primitives de type *TestAndSet*, *CompareAndSwap...*
   a) Implanter le protocole d'exclusion mutuelle pour N processus utilisant la primitive *TestAndSet* présentée en cours (planche 23)
   b) Réaliser une version vivace du protocole, garantissant que toute demande d'entrée en section critique finira par être servie.
   c) Comparer les performances des deux versions, entre elles et par rapport à une version utilisant un bloc *synchronized* pour assurer l'exclusion mutuelle.
   d) Comparer, pour 2 processus, ces versions à une version utilisant le protocole de Peterson

Solution : Dans la version rendu, j'utilise *AtomicBoolean* pour rendre le booléen atomique, et j'utilise des blocs synchronisés pour modifier *entrer()* et *sortir()* pour assurer l'exclusion mutuelle.

Schéma producteurs consommateurs

4. Compléter la classe TamponBorné fournie, qui ne comporte aucune synchronisation, afin de gérer convenablement les accès concurrents.

Solution : Dans ce fichier *ProdConso.jave*, j'ai modifié le contenu de la sortie d'affichage pour rendre le programme plus lisible lorsqu'il est en cours d'exécution.
Pour les producteurs, lorsque *nbOccupé* >= capacité tampon, le processus du producteur ne peut pas continuer, mais une instruction *this.wait()* doit être exécutée :

```
1. while (nbOccupé >= taille) {
2.       try {
3.           System.out.println("Productor: The buffer is full, please
   wait...");
4.           this.wait();
```

```
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
    }
```

Pour les consommateurs, lorsque *nbOccupé* <= 0, le processus du consommateur ne peut pas continuer, mais une instruction *this.wait()* doit être exécutée :

```
while (nbOccupé <= 0) {
      try {
                System.out.println("Customer: The buffer is empty, please
   wait...");
                this.wait();
      } catch (InterruptedException e) {
          e.printStackTrace();
      }
}
```

Selon l'affichage dans le terminal, nous pouvons voir :

(1) Initialement, les consommateurs doivent attendre, car le tampon est vide (comme le montre la figure 1).

```
java ProdConso <nbProd> <nbConso> <nbCases>
-> choix par défaut : 5/10/10
nbProd (arg1) : 5 /nbConso (arg2) : 10 /nbCases) (arg3) : 10
Customer: nbOccupé (before) = 0
Customer: The buffer is empty, please wait...

Customer: nbOccupé (before) = 0
Customer: The buffer is empty, please wait...

Customer: nbOccupé (before) = 0
Customer: The buffer is empty, please wait...
```

Figure 1

(2) Les consommateurs retirent des entiers différents, dans l'ordre croissant, sans trous dans la numérotation (comme le montre la figure 2).

```
Customer: nbOccupé (before) = 5
Customer: Tampon[5] = 75
Costumer: nbOccupé (after) = 4

Customer: nbOccupé (before) = 4
Customer: Tampon[6] = 76
Costumer: nbOccupé (after) = 3

Customer: nbOccupé (before) = 3
Customer: Tampon[7] = 77
Costumer: nbOccupé (after) = 2

Customer: nbOccupé (before) = 2
Customer: Tampon[8] = 78
Costumer: nbOccupé (after) = 1
```

Figure 2