Université El-Oued
Faculté des sciences exactes- Département d'informatique
Module : SEII Niveau : 3ème LMD

TP3 : Les Sémaphores

➢ **Objectifs du TP**

Ce TP montre l'usage des sémaphores en Java. Nous présentons d'abord la classe Semaphore, puis faisons l'exercice d'appliquer ce concept sur les applications traditionnelles de gestion de la synchronisation vues en TD.

I. L'exclusion mutuelle

Pour traiter les problèmes d'exclusion mutuelle, Java propose la définition de sections critiques exprimées à l'aide du mot clé `synchronized`.

Tout objet Java est équipé d'un verrou d'exclusion mutuelle. Ainsi, pour assurer qu'une seule activité accède à un objet `unObj` d'une classe quelconque, on définit les actions sur l'objet dans une région critique par la syntaxe :

```
synchronized (unObj) { <
    Région critique >
}
```

Une méthode peut aussi être qualifiée de `synchronized` :

```
synchronized T uneMethode(…) { … }
```

Ceci est équivalent à :

```
T uneMethode(…) {
      synchronized (this) { … }
}
```

Il y a donc exclusion d'accès de l'objet sur lequel on applique la méthode, pas de la méthode elle-même, qui peut être exécutée concurremment sur des objets différents.

Exemple :

```
public class TestSync extends Thread {
private int solde ;
public void run (){
for ( int i =0; i <50; i++){
      incrementer ();
System .out. println ("le solde est de : " + solde );
}
}
public synchronized void incrementer (){
int i = solde ;
solde = i +1;
   }
}

public class TestTestSync {
public static void main ( String [] args ){
TestSync a = new TestSync();
TestSync b = new TestSync();
a. start ();
b. start ();
}}
```

II. La classe Semaphore

La classe java.util.concurrent.Semaphore représente un sémaphore de comptage. Elle est accessible principalement grâce aux méthodes suivantes:

- Constructeur avec un argument entier: pour l'initialisation du sémaphore.
- La méthode acquire() : équivalente au down() ou à P(), permet de demander l'accès à la section critique, en bloquant le processus en cours si la SC est déjà prise.
- La méthode release() : équivalente au up() ou V(), permet de libérer la SC, en débloquant potentiellement un processus qui était bloqué.

III. Barrière de synchronisation

Soient N processus parallèles ayant un point de rendez-vous. Un processus arrivant au point de rendez-vous se met en attente s'il existe au moins un autre processus qui n'y est pas arrivé. Le dernier arrivé réveillera les processus bloqués. Traduire l'algorithme suivant en Java en utilisant les sémaphores.

```
Contexte commun:
   sémaphore mutex = 1, s= 0;
   entier NbArrivés = 0; /* nbre de processus arrivés au rendez-vous */
Procédure RDV
Début
 down(mutex);
 NbArrivés = NbArrivés + 1;
 Si (NBArrivés < N) Alors      /* non tous arrivés */
   up(mutex);        /* on libère mutex et */
   down(s);        /* on se bloque */
 Sinon
   up(mutex);        /* le dernier arrivé libère mutex*/
   Pour i = 1 à N-1 Faire up(s); /* et réveille les N-1 bloqués */
 Finsi
Fin
```

IV. Producteur/Consommateur

Nous désirons simuler le problème du producteur/consommateur en utilisant les sémaphores Java. Le buffer partagé sera représenté par un objet de type Vector.

La classe Vector implémente un tableau extensible d'objets. Comme un tableau, un vecteur contient des composants accessibles grâce à un indexe entier. Cependant, la taille du vecteur peut augmenter ou diminuer pour accommoder l'ajout et la suppression des données après la création du vecteur.

Pour un vecteur de chaînes de caractères, par exemple, l'initialisation se fait ainsi:

Vector<String> v = new Vector<String>();

- Pour ajouter un objet à la fin du vecteur : v.add(object);
- Pour lire l'élément d'indexe i du vecteur : elem = v.get(i);
- Pour supprimer un élément à l'indexe i: v.remove(i);
- Pour saisir le premier élément d'un vecteur : elem = v.firstElement() ;

V. Homework

Réaliser le problème du coiffeur endormi que vous avez vu en TD en utilisant les sémaphores.