TD nº 5 - Communication à l'aide de boîtes aux lettres

Le but de ce TD est de mettre en place un système permettant à des processus différents de communiquer. La solution proposée ici est d'utiliser une *boîte aux lettres* dans laquelle les processus peuvent déposer des messages ou lire les messages déposés. Pour raison de simplicité, des "processus" communicants seront des `threads` d'un programme Java.

Exercice 1. *Le cas simple*

Dans un premier temps, on s'intéresse à la réalisation d'une boîte aux lettres ne pouvant contenir qu'un seul message à la fois. Cette boîte aux lettres est utilisée entre un client et un serveur : le client dépose un message pour le serveur (une requête) puis le serveur lit le message. Dans ce premier exemple, le serveur va se contenter d'afficher à l'écran le message reçu. Il n'est donc pas nécessaire de prévoir de mécanisme pour envoyer des messages du serveur vers le client.

On utilise deux sémaphores : un sémaphore `sDepot` pour indiquer aux clients que la boîte aux lettres est libre, et un sémaphore `sRetrait` pour indiquer au serveur qu'un message a été déposé dans la boîte aux lettres.

On va créer une classe `BalSimple` comportant une méthode `deposeRequete` qui attend que la boîte aux lettres soit libre puis y dépose le message passé en argument, et une méthode `retireRequete` qui attend que la boîte aux lettres contienne un message puis vide la boîte aux lettres et renvoie le message qui s'y trouvait.

1. Écrivez la classe `BalSimple` avec son constructeur et les deux méthodes `deposeRequete` et `retireRequete` :

```
class BalSimple {
  String message;
  Semaphore sDepot, sRetrait;

  public BalSimple() {
    // initialisation des s'emaphores
  }
  public void deposeRequete(String mess) {
    /* - attendre que la BAL soit vide
       - d'eposer le message dans la BAL
       - indiquer que la BAL est pleine
    */
  }
  public String retireRequete(){
    /* - attendre que la BAL soit pleine
       - lire le message
       - indiquer que la BAL est vide
       - renvoyer le message
    */
  }
}
```

Par la suite, les sémaphores `sDepot` et `sRetrait` ne doivent être modifiés que par l'intermédiaire des méthodes `deposeRequete` et `retireRequete`.

2. Ecrivez la classe `Client` (`Thread` ou `Runnable` selon votre choix) dont la tâche dépose un unique message dans la boîte aux lettres.

Les clients seront créés avec un nom identifiant comme dans le TD précédent et le message envoyé contiendra le nom du client.

Indication : Utilisez la méthode `deposeRequete` de la boîte aux lettres.

3. Ecrivez la classe `Serveur` dont la tâche attend qu'un message soit déposé dans la boîte aux lettres et affiche ce message.

Les serveurs sont également identifiés et l'affichage du message doit faire apparaître le nom du serveur qui a reçu le message.

Indication : Utilisez la méthode `retireRequete` de la boîte aux lettres.

4. Ecrivez la classe principale de votre application qui exécutera un `Client` et un `Serveur` fonctionnant sur la même boîte aux lettres (sinon ce n'est pas très intéressant...). Vérifiez que le `Serveur` reçoit et affiche le message envoyé par le `Client`.

5. Modifiez la classe `Client` pour que chaque client dépose maintenant 6 requêtes successives dans la boîte aux lettres, en ajoutant un délai aléatoire entre deux requêtes.

Indication : Pour ajouter un délai de durée aléatoire [1] entre 50 et 150 ms, vous pouvez utiliser le bloc de code suivant :

```
try {
  Thread.sleep(random.nextInt(100) + 50);
} catch (InterruptedException) {
  e.printStackTrace();
}
```

6. Modifiez la classe `Serveur` pour que cette tâche exécute une boucle infinie de lectures de messages postés dans la boîte aux lettres. Pour simuler le temps de traitement d'une requête, ajoutez un délai aléatoire compris entre 100 et 300 ms après avoir récupéré un message dans la boîte aux lettres.

7. Modifiez le programme principal pour qu'il lance une tâche `Serveur` et trois tâches `Client`.

8. Quand le programme termine son exécution ?

Bonus : Réfléchissez aux différentes manières d'implémenter une méthode `stop()` dans la classe `Serveur` pour interrompre le fonctionnement du serveur. Écrivez cette méthode et modifiez votre programme principal pour qu'il termine le fonctionnement du `Serveur` après avoir envoyé toutes les requêtes.

Exercice 2. *Question, réponse*

On considère toujours une boîte aux lettres unique ne pouvant contenir qu'un seul message mais cette fois-ci on veut que le serveur envoie une réponse au client après avoir traité sa requête.

Le déroulement est donc le suivant :

— lorsque la boîte aux lettres est disponible, un client peut y déposer une requête. Il attend alors que la boîte aux lettres contienne une réponse du serveur ;
— lorsque la boîte aux lettres contient une requête, le serveur peut lire ce message. Tant que le serveur n'a pas répondu, la boîte aux lettres est en attente de réponse ;
— lorsque le serveur a fini de préparer la réponse à la requête du client, il dépose cette réponse dans la boîte aux lettres ;
— lorsque la boîte aux lettres contient une réponse, le client qui avait placé la requête peut lire la réponse, ce qui a pour effet de libérer la boîte aux lettres.

1. Combien d'états différents utilise-t-on pour la boîte aux lettres ? Combien de sémaphores va-t-on devoir utiliser ?

Comme dans l'exercice précédent, les sémaphores seront gérés par les méthodes de la boîte aux lettres.

2. Modifiez la classe `BalSimple` pour en faire une classe `BalQR` disposant des méthodes suivantes :
— `deposeRequete(String mess)` qui permet à un client de déposer un message ;
— `retireRequete()` qui permet à un serveur de lire le message déposé par un client. Cette méthode bloque la boîte aux lettres jusqu'à ce qu'elle reçoive la réponse du serveur ;

1. La fonction `random.nextInt(int n)` renvoie un entier aléatoire uniformément choisi entre 0 (inclus) et n (exclu). Pour l'utiliser, vous devez inclure le module `java.util.Random`.

— `deposeReponse(String mess)` qui permet au serveur qui avait reçu la requête du client de déposer la réponse à cette requête ;
— `retireReponse()` qui permet au client qui avait envoyé la requête de lire la réponse, et libère la boîte aux lettres.

3. Écrivez la classe `ClientQR` dont la fonction `run()` dépose successivement 6 requêtes dans la boîte aux lettres. Après chaque dépot d'une requête, il faut récupérer et afficher la réponse du serveur. Ajoutez un délai entre la réception d'une réponse et l'envoi de la requête suivante.

4. Écrivez la classe `ServeurQR` qui exécute une boucle infinie dont chaque itération consiste à lire une requête postée dans la boîte aux lettres, attendre un délai aléatoire pour simuler le temps de traitement, puis déposer la réponse à la requête. Dans cet exercice, la réponse à un message est simplement le message préfixé de la chaîne « `Serveur i, Réponse:` » où i est le numéro du serveur.

5. Testez votre programme en ne lançant initialement qu'un seul `ClientQR`, et un seul `ServeurQR`. Puis lancez deux `ServeurQR` et trois `ClientQR`.

Exercice 3. *Emplacements multiples*

On revient à la situation de l'exercice 1 (pas de réponse du serveur), mais on veut ajouter des emplacements à la boîte aux lettres pour qu'elle puisse contenir plusieurs messages à la fois. On va utiliser la classe suivante :

```
class BalTable {
  int indiceDepot,  indiceRetrait, nb;
  String message [];
  Semaphore sDepot, sRetrait;

  public Bal (int nbcases) {
    this.nb = nbcases;
    this.message = new String[this.nb]; // tableau de nb cha\^{\i}nes
    this.indiceDepot = 0; // initialisation des indices
    this.indiceRetrait = 0;
  }

  public void deposeRequete(String mess) {
    // \'a compl\'eter
  }

  public String retireRequete() {
    // \'a compl\'eter
  }
}
```

Afin que les requêtes déposées dans la boîte aux lettres soient traitées dans leur ordre d'arrivée (FIFO), on utilise deux indices :
— `indiceDepot` désigne la case du tableau dans laquelle on placera la prochaine requête ;
— `indiceRetrait` désigne la position dans le tableau où se trouve la requête la plus ancienne à traiter.

Ces deux indices sont initialisés à 0 et incrémentés à chaque fois que l'opération à laquelle ils correspondent est effectuée. Le tableau est considéré comme circulaire c'est-à-dire que l'incrémentation est faite *modulo* le nombre de cases du tableau :

```
    this.indiceDepot = (this.indiceDepot + 1) % this.nb
```

Les sémaphores `sDepot` et `sRetrait` ont une signification légèrement plus complexe qu'avant puisqu'ils représentent respectivement le nombre de cases vides de la boîte aux lettres et le nombre de requêtes en attente de traitement.

1. Écrivez les méthodes `deposeRequete` et `retireRequete` de la classe `BalTable` en utilisant les sémaphores et les indices.

2. Faut-il modifier les classes `Client` et `Serveur` écrites dans l'exercice 1 ?

3. Modifiez la classe principale de votre application puis testez le fonctionnement avec 3 clients et un seul serveur.

4. Identifiez les problèmes de synchronisation qui peuvent se produire si deux requêtes sont déposées simultanément dans la boîte aux lettres. Résolvez ces problèmes en ajoutant des verrous là où c'est nécessaire.