Système T.D 3

Exercice 1 (Le problème des lecteurs / rédacteurs)
Ce problème modélise les accès à une base de données. Plusieurs processus(threads) tentent constamment d'accéder à la base de données soit pour écrire, soit pour lire des informations.

Pour assurer une certaine cohérence des données de la base, il faut interdire l'accès (en lecture et en écriture) à tous les processus, si un autre processus est en train de modifier la base (mode écriture). Par contre, plusieurs processus peuvent accéder à la base, en même temps, en mode lecture.

Les rédacteurs représentent les processus qui demandent des accès en écriture à la base de données. Les lecteurs représentent les processus qui demandent des accès en lecture.

Question 0.1 Il est demandé d'implanter ce problème avec des Threads en Java avec des sémaphores.

Pour contrôler les accès à la base, on connait le nombre de lecteurs qui sont en train de lire (NbL). Le compteur NbL est un objet partagé par tous les lecteurs. L'accès à ce compteur doit être exclusif (sémaphore Mutex).

Un lecteur peut accéder à la base, s'il y a déjà un lecteur qui accède à la base (NbL>0) ou aucun rédacteur n'est enrain d'utiliser la base. Un rédacteur peut accéder à la base, si elle n'est pas utilisée par les autres (un accès exclusif à la base). Pour assurer cet accès exclusif, on utilise une autre sémaphore (Redact).

Question 0.2 Modifier le code de manière à empécher les lecteurs d'entrer en lecture de la base si un au moins des rédacteurs est en attente.

en java hasQueuedThreads() permet de savoir si des threads sont en attente sur une sémaphore.

Exercice 2 (le diner de philosophe chinois avec des Moniteurs Java)
Le problème est le suivant : 5 philosophes sont autour d'une table pour se restaurer. Chacun possède une baguette et une seule!!! Le philosophe étant bien élevé ne mange qu'avec deux baguettes, il doit donc en emprunter une à un voisin. On vous demande de proposer une solution simulant la vie de philosophe cad penser, manger et ainsi de suite avec des moniteurs Java.

On va supposer que l'on a 5 philosophes 0 à 4 et 5 baguettes 0 à 4, la $i$ème étant à gauche du phylosophe i.

On considère que l'on a la classe :

```
class Table {
private boolean[] usedFork;

public Table( int numberForks) {
usedFork= new boolean[ numberForks];

for (int i =0;i<usedFork.length;i++)
```

```
usedFork[i]=false;
}

private int left( int i) {
return i;
}

private int right ( int i) {
if (i+1< usedFork.length)
return i+1;
else return 0; }

public synchronized void takeForks (int place)
...

public synchronized void putBackForks (int place)
```

et que les threads sont démarrés par :

```
public class Philosophers{
private static final int numberPhilisophers=5;

public static void main(String[] args) {
Table table =newTable(numberForks);

for (int i=0;i<numberPhilisophers;i++)
new Philosophe(table,i); //i=place
}}
```

Question 0.3 Implanter la classe Philosophe et complétez la classe Table

Exercice 3 (Exclusion mutuelle par jeton sur anneau)
Cet exercice a pour but de réaliser un mécanisme d'exclusion mutuelle par jeton circulaire, entre les sections critiques de K threads.

On réalise ici l'exclusion mutuelle par un mécanisme de prise de jeton : un jeton circule de thread en thread. Lorsqu'un thread veut exécuter une section critique, il attend le jeton, le prend et le conserve jusqu'à la fin de celle-ci puis le passe au thread suivant.

Question 0.4 Comment implanter une telle solution ?

Question 0.5 Quels sont les avantages inconvénients ?

Question 0.6 Implantez la solution en Java.