Mini-projet #8 - Moniteurs avec priorités

Cours Applications Concurrentes, *Polytech-INFO4*

F. Boyer, Université Grenoble Alpes, 2022-2023

1. Introduction

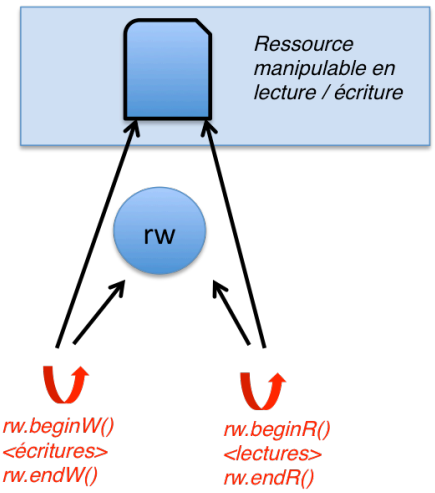
Dans cette session vous allez apprendre une méthodologie pour produire une solution avec priorités à un problème de programmation concurrente. Plus précisément, nous allons considérer un problème dans lequel :

- Des threads cherchent à accéder à des ressources selon divers modes (exclusif, partagé)
- Ces threads peuvent avoir des priorités différentes.

Nous allons appliquer cette méthodologie pour gérer des threads qui veulent accéder à une ressource partagée en lecture et/ou en écriture (classiquement appelé problème des **lecteurs-rédacteurs**). Concrètement, on souhaite programmer une classe (que l'on va appeler RW pour ReaderWriter) qui fournit l'interface suivante :

```
public class RW {
   public void beginW() ; // get a write access to the resource
   public void beginR();  // get a read access to the resource
   public void endW();    // release the write access
   public void endR();    // release the read access
}
```

Toute instance de cette classe gère les accès à une ressource donnée. L'implémentation des méthodes doit garantir **l'invariant** suivant : *à tout instant il y a soit : zéro threads, ou bien n reader, ou bien 1 writer, qui manipulent la ressource.*



Attention, la lecture ou l'écriture se font en dehors des méthodes beginR() / beginW(). Ces méthodes ne sont là que pour contrôler l'accès en lecture/écriture.

2. Solution directe

Question 2.1. Produire le tableau de gardes-actions et en déduire la solution directe. Les gardes du tableau gardes-actions doivent simplement garantir l'invariant précédemment cité.

Une fois la solution directe produite, interrogez-vous sur son efficacité, et remplacez les *notifyAll()* par *notify()* aux endroits où cela vous paraît possible.

3. Solution FIFO

Nous avons vu dans le précédent TD comment dériver une solution FIFO à partir de la solution directe.

Question 3.1. Produire une solution FIFO pour les threads *readers* et *writers*. Une telle solution doit garantir que dans le scénario où des threads readers (R_i) et Writers (W_i) demandent un accès à une ressource partagée dans l'ordre suivant : **W1 R1 R2 R3 W2 R4 R5**, alors la ressource sera allouée à W1 puis à R1//R2//R3 puis à W2 puis à R4//R5 (R4 et R5 passent après W2 même s'ils arrivent pendant que R1, R2 et R3 lisent).

4. Solution avec priorité aux readers

La démarche suivie consiste à repartir de la solution directe et à adapter les gardes pour prendre en compte les priorités. Le principe est de tester dans toute garde, que la ressource est bien libre *et qu'il n'y a pas de thread plus prioritaire en attente pour cette ressource*.

```
.. synchronized meth(..) {
   <pre-action>
    while ! <garde> || <il y a des threads plus prioritaires en
attente>
          wait() ; **
    <post-action>
  [notifyAll() ;]
```

** Par simplicité, on omet le traitement des interruptions au niveau du wait. Lors de la mise en œuvre finale, il faudra remplacer wait() par *try { wait() ; } catch (InterruptedException e) {}*).

Dans le cas présent, il faut donc que la garde pour les writers soit d'attendre qu'il n'y ait pas d'écriture ou de lecture en cours et **qu'il n'y ait pas de lecteurs en attente**.

Question 4.1. Evoluer la solution directe vers une solution avec priorité aux readers en suivant le conseil précédent.

Dans un deuxième temps, posez vous la question du regroupement des compteurs *le lecteur en cours ou en attente*.

5. Solution avec priorité aux writers

Il faut que la garde pour les readers soit d'attendre s'il y a un writer en cours **ou en attente**.

Question 6.1. En suivant la même démarche que précédemment, qui consiste à rajouter un compteur de writers en attente (nww, number of writers waiting), produisez une solution avec priorité aux writers.

Dans un deuxième temps, posez vous la question du regroupement des compteurs *nww* et *nw*. Est-ce possible ? Pourquoi cela ne fonctionne t'il pas?