KLIPFFEL Tararaina
BLONDET Thomas

# Programmation concurrente

**Objectifs atteints et temps de réalisation :**

Objectif 1 : 7h
Objectif 2 : 1h
Objectif 3 : 5h
Objectif 4 : 4h

**Explication & Tests :**

**Objectif 1&2 :**

Nous avons mis en place un jeu de test pour vérifier le bon comportement de notre programme à la version v1. Chaque test est paramétré dans un des fichiers option_v1*.xml.

Test 1 : *Option_v11.xml*

Dans ce premier test nous avons voulu vérifier qu'il pouvait y avoir autant de lecteur qu'on veut sur la ressource disponible. On choisi alors de mettre 10 lecteurs en paramètre et 0 rédacteur. On obtient le résultat suivant :

```
Reader-0 Require : Resource 0
Reader-0 Acquire : Resource 0
Reader-4 Require : Resource 0
Reader-4 Acquire : Resource 0
Reader-8 Require : Resource 0
Reader-8 Acquire : Resource 0
Reader-5 Require : Resource 0
Reader-5 Acquire : Resource 0
Reader-3 Require : Resource 0
Reader-3 Acquire : Resource 0
Reader-2 Require : Resource 0
Reader-2 Acquire : Resource 0
Reader-0 Release : Resource 0
Reader-6 Require : Resource 0
Reader-6 Acquire : Resource 0
Reader-4 Release : Resource 0
Reader-7 Require : Resource 0
Reader-7 Acquire : Resource 0
Reader-8 Release : Resource 0
Reader-9 Require : Resource 0
Reader-9 Acquire : Resource 0
Reader-2 Release : Resource 0
Reader-3 Release : Resource 0
Reader-5 Release : Resource 0
Reader-1 Require : Resource 0
Reader-1 Acquire : Resource 0
Reader-7 Release : Resource 0
Reader-6 Release : Resource 0
```

…

On se rend compte que lors qu'un lecteur demande l'accès il l'obtient instantanément, il n'y a pas de blocage et que le release ne pose pas de problème dans le déroulement du programme non plus (ex : Reader - 4 ).

Test 2 : *Option_v12.xml*

Ce second test nous permettra de montrer les effets de blocage entre rédacteur et lecteur. Nous avons donc exécuté le programme avec un rédacteur au milieu de 3 lecteurs.
Nous obtenons ces résultats d'exécution :

```
Reader-3 Require : Resource 0
Reader-3 Acquire : Resource 0
Writer-0 Require : Resource 0
Reader-2 Require : Resource 0
Reader-1 Require : Resource 0
Reader-3 Release : Resource 0
Writer-0 Acquire : Resource 0
Writer-0 Release : Resource 0
Reader-2 Acquire : Resource 0
Reader-1 Acquire : Resource 0
Reader-1 Release : Resource 0
Reader-2 Release : Resource 0
Reader-3 Require : Resource 0
Reader-3 Acquire : Resource 0
Writer-0 Require : Resource 0
Reader-3 Release : Resource 0
Writer-0 Acquire : Resource 0
Reader-1 Require : Resource 0
Reader-2 Require : Resource 0
Writer-0 Release : Resource 0
Reader-1 Acquire : Resource 0
Reader-2 Acquire : Resource 0
Reader-2 Release : Resource 0
Reader-1 Release : Resource 0
                ...
```

On constate plusieurs choses intéressantes :

- le rédacteur est en attente que tout les lecteurs aient libéré leurs accès à la ressource pour obtenir son droit d'accès ;
- les lecteurs sont eux aussi bloqués si un rédacteur est bloqué ;
- si un rédacteur a accès à la ressource, il bloque les lecteurs qui en demande aussi l'accès jusqu'à ce qu'il rende son accès.

Ces caractéristiques font parti des propriétés attendues du programme à la version v1.

Test 3 : *Option_v13*

Ce dernier test nous permet de vérifier le bon fonctionnement des rédacteurs entre eux. Pour cela nous exécutons le programme avec seulement 4 rédacteurs.

```
Writer-1 Require : Resource 0
Writer-1 Acquire : Resource 0
Writer-0 Require : Resource 0
Writer-2 Require : Resource 0
Writer-1 Release : Resource 0
Writer-0 Acquire : Resource 0
```

```
Writer-0 Release : Resource 0
Writer-2 Acquire : Resource 0
Writer-1 Require : Resource 0
Writer-0 Require : Resource 0
Writer-2 Release : Resource 0
Writer-1 Acquire : Resource 0
Writer-2 Require : Resource 0
Writer-1 Release : Resource 0
            …
```

On remarque que même entre eux les rédacteurs se bloquent, ce test garanti qu'il ne peut pas y avoir plusieurs rédacteurs en même temps sur le même ressource.

Avec ces 3 tests, on peut garantir que la version v1 du programme s'exécute en adéquation avec les consignes quelque soit le nombre de lecteurs et de rédacteurs.

**Objectif 3 :**

A cet objectif nous avons atteint la version v2 de notre programme. Nous avons testé notre programme en prenant les paramètres dans le fichier *option_v21.xml* et nous obtenons le résultat suivant :

```
Writer-0 Require : Resource 0
Writer-0 Acquire : Resource 0
Writer-1 Require : Resource 0
Reader-3 Require : Resource 0
Reader-2 Require : Resource 0
Writer-0 Release : Resource 0
Reader-2 Acquire : Resource 0
Reader-3 Acquire : Resource 0
Reader-2 Release : Resource 0
Reader-3 Release : Resource 0
Writer-1 Acquire : Resource 0
Writer-0 Require : Resource 0
Writer-1 Release : Resource 0
Reader-3 Require : Resource 0
Reader-3 Acquire : Resource 0
Reader-2 Require : Resource 0
Reader-2 Acquire : Resource 0
Reader-2 Release : Resource 0
Reader-3 Release : Resource 0
Writer-0 Acquire : Resource 0
            …
```

On se rend compte que le rédacteur ne reprend la main qu'après 2 cycles de lecteurs pour les 2 cycles du test. Cela nous confirme le bon fonctionnement de la version v2 du programme.

De plus, dans cette version les tâches en attentes sont réveillées par la fonction *NotifyAll* ce qui ne garde pas l'ordre d'arrivée. Ce n'est pas une file à priorité. On s'en rend compte facilement lorsqu'il y a plusieurs lecteurs disponibles. Ce test est mis en place dans le fichier *option_v22.xml.*

```
Writer-0 Require : Resource 0
Writer-0 Acquire : Resource 0
Reader-6 Require : Resource 0
Reader-2 Require : Resource 0
Reader-3 Require : Resource 0
Reader-1 Require : Resource 0
```

```
Writer-0 Release : Resource 0
Reader-3 Acquire : Resource 0
Reader-2 Acquire : Resource 0
Reader-6 Acquire : Resource 0
Reader-1 Acquire : Resource 0
Reader-4 Require : Resource 0
Reader-4 Acquire : Resource 0
Reader-5 Require : Resource 0
Reader-5 Acquire : Resource 0
```

On voit que l'ordre d'arrivé des lecteurs est 6 – 2 – 3 – 1 et que l'ordre dans lequel ils ont eu accès à la ressource est 3 – 2 – 6 – 1. On voit que le brassage après le *release* de l'écrivain 0 a bien été effectué.

**Objectif 4 :**

Dans cet objectif nous nous placerons dans les mêmes conditions que la version v1, en prenant en compte deux niveaux de priorité {LOW_WRITER;HIGH_WRITER}.

Une condition est intrinsèquement lié à une serrure. Pour obtenir une instance de condition sur une serrure particulière on utilise la méthode newCondition ().

```
private Condition writer = lock.newCondition();
private Condition reader = lock.newCondition();
```

Voici une exécution avec l'option LOW_WRITE :

```
Reader-4 Require : Resource 0
Reader-4 Acquire : Resource 0
Reader-5 Require : Resource 0
Reader-5 Acquire : Resource 0
Writer-0 Require : Resource 0
Reader-6 Require : Resource 0
Reader-6 Acquire : Resource 0
Writer-1 Require : Resource 0
Reader-4 Release : Resource 0
Reader-2 Require : Resource 0
Reader-2 Acquire : Resource 0
Reader-3 Require : Resource 0
Reader-3 Acquire : Resource 0
Reader-5 Release : Resource 0
Reader-2 Release : Resource 0
Reader-6 Release : Resource 0
Reader-3 Release : Resource 0
Writer-0 Acquire : Resource 0
Reader-2 Require : Resource 0
Writer-0 Release : Resource 0
Writer-1 Acquire : Resource 0
                ...
```

Et avec HIGH_WRITE :

```
Writer-1 Require : Resource 0
Writer-1 Acquire : Resource 0
Writer-0 Require : Resource 0
Reader-4 Require : Resource 0
Reader-3 Require : Resource 0
Reader-2 Require : Resource 0
Reader-6 Require : Resource 0
Reader-5 Require : Resource 0
Writer-1 Release : Resource 0
Writer-0 Acquire : Resource 0
Writer-0 Release : Resource 0
Reader-4 Acquire : Resource 0
Writer-1 Require : Resource 0
Reader-4 Release : Resource 0
Writer-1 Acquire : Resource 0
Writer-0 Require : Resource 0
Writer-1 Release : Resource 0
Writer-0 Acquire : Resource 0
Writer-0 Release : Resource 0
Reader-3 Acquire : Resource 0
```