IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

TD6 – Lecteurs-Rédacteurs

Modélisation et Programmation Concurrente – ELU 512

Objectifs

À la fin de l'activité, les élèves devront être capables d' :
- expliquer la notion de synchronisation, de condition d'attente et d'invariant de sûreté ;
- identifier le besoin de synchronisation de *threads* ;
- appliquer une méthode systématique de réalisation de la synchronisation de *threads*. La méthode implique la modification d'une seule classe ;
- utiliser les moniteurs Java pour mettre en œuvre le bloquage/déblocage de *threads* dans un programme concurrent.

1 Le problème des lecteurs-rédacteurs

Il s'agit d'un problème classique en informatique concurrente. Dans ce problème, une base de données est accédée par des lecteurs et par des rédacteurs. Les lecteurs ne modifiant pas les données de la base, plusieurs lecteurs peuvent y accéder en même temps. Mais, un seul rédacteur peut écrire à la fois.

Exercice 1 (*Un modèle sans synchronisation*)

Un extrait d'une conception objet de la solution au problème des lecteurs-rédacteurs est présenté figure 1. L'accès à la base de données (ressource partagée) est géré par un objet de type `Synchronizer`. Les lecteurs et rédacteurs seront des *travailleurs*, ils héritent de la classe `Worker`. Ils seront des tâches de leur propre fil d'activité. Les méthodes `rest` et `work` de la classe `Worker` bloquent l'activité du *thread* respectivement `restTime` et `workTime`.

▷ **Question 1.1 :**
Un lecteur (resp. rédacteur) attend un laps de temps de durée `restTime` avant de lire (resp. écrire). Une fois ce temps écoulé, le lecteur lit (resp. rédacteur écrit) pendant le temps de lecture (resp. le temps d'écriture). Ces deux durées seront stockées dans

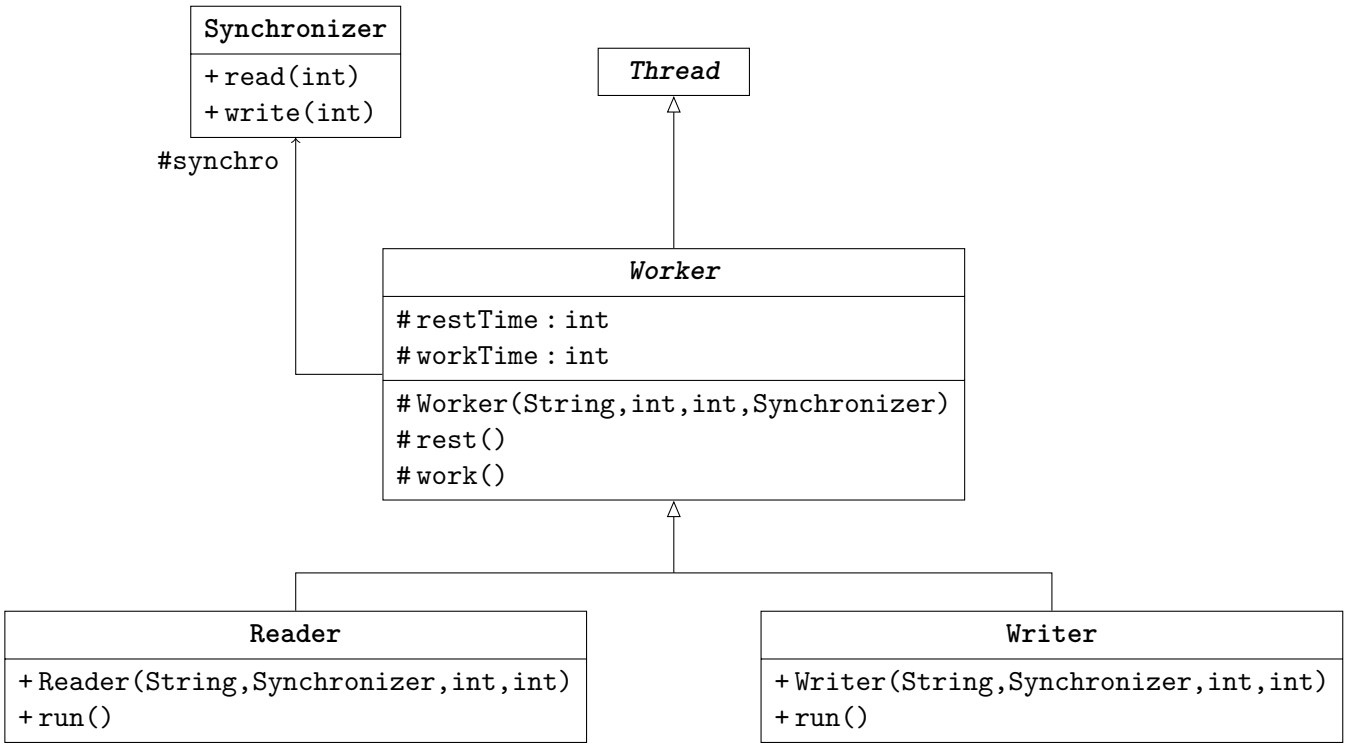

FIGURE 1 – Extrait du diagramme de classes de l'application

l'attribut `workTime` de la classe `Worker`. Donner le code de la *vie* d'un lecteur et d'un rédacteur.

▷ **Question 1.2 :**
Quelles garanties d'accès à la base de données aurait-on si les méthodes `read` et `write` de la classe `Synchronizer` étaient synchronisées par le mot clé `synchronized` ?

Exercice 2 (*Les conditions d'attente pour la synchronisation*)

Pour garantir les spécifications données dans la description du problème, il faut s'assurer que la ressource partagée est dans un « bon » état avant que les lecteurs (rédacteurs) y accèdent. Il faut donc pouvoir influencer l'ordre des accès à la base de données.

De manière générale, lorsque des *threads* coopèrent pour accéder à une ressource partagée il faut influencer l'ordre des accès à celle-ci. On dit alors qu'il faut *synchroniser* les *threads*.

Le schéma général est le suivant :

1. un *thread* prend le verrou, teste l'état de la ressource et doit attendre si elle n'est pas dans un état qui lui convient (par exemple pour un lecteur, un rédacteur écrit dans la base) ;
2. dans ce cas, le *thread* relâche le verrou en espérant qu'un autre *thread* modifiera la ressource (par exemple le rédacteur a fini d'écrire) ;
3. après avoir relâché le verrou, il se bloque en attente sur une condition ;
4. lorsqu'un *thread* modifie l'état de la ressource (par exemple le rédacteur a fini d'écrire), une condition devient vraie. Il signale la modification aux *threads* bloqués sur cette condition ;
5. le(s) *thread(s)* relâché(s) essaye(nt) de reprendre le verrou.

Pour synchroniser les accès à une ressource partagée, il faut donc identifier les conditions d'attente sur lesquelles les *threads* qui accèdent doivent se bloquer si l'état de la ressource ne leur convient pas.

▷ **Question 2.1 :**
Exprimer en français la condition d'attente des lecteurs et celle des rédacteurs ?

Exercice 3 (*Une méthode pour réaliser la synchronisation*)

Dans cet exercice, nous allons illustrer la méthode systématique de résolution du problème de synchronisation de *threads* vue en cours. Pour cela, nous allons l'appliquer dans le cadre du problème des lecteurs-rédacteurs décrit précédemment.

Cette méthode repose sur l'*invariant de sûreté* du système. L'invariant de sûreté est une condition qui doit être satisfaite à tout instant de l'exécution du système : si c'est le cas, rien de « mauvais » ne peut alors arriver.

▷ **Question 3.1 :**
Quel est l'invariant de sûreté pour le problème des lecteurs-rédacteurs ? Exprimer l'invariant en français.

La méthode de réalisation de la synchronisation de *threads* basée sur l'invariant de sûreté est la suivante :

1. Identifier les variables qui permettent d'exprimer l'invariant de sûreté du système : l'état du système qui doit être garanti à tout instant de son exécution.
2. Identifier les actions « critiques » (qui modifient l'invariant de sûreté) : ces actions sont réalisées par des méthodes dans le cas d'une solution de synchronisation avec des moniteurs.
3. Les actions identifiées doivent être appelées en exclusion mutuelle puisqu'elles modifient l'invariant et donc les variables qui le représentent : les méthodes correspondant aux actions seront *marquées* `synchronized` dans une solution de synchronisation avec des moniteurs.
4. Identifier, pour chaque action critique, la condition d'attente pour la synchronisation (pour assurer que l'invariant de sûreté est satisfait) : chaque condition est réalisée par une méthode dont la valeur de retour est un `boolean`. Le corps de cette méthode doit suivre le patron suivant :
   — on *simule* l'exécution de l'action (donc les valeurs de variables correspondantes sont modifiées) ;
   — on évalue si l'invariant de sûreté reste satisfait avec ces nouvelles valeurs ;
   — si l'invariant n'est plus satisfait, on restaure l'ancienne valeur des variables (on annule l'action) et on bloque le *thread*.

▷ **Question 3.2 :**
Identifier les variables qui permettent d'exprimer l'invariant de sûreté pour le problème des lecteurs-rédacteurs.

▷ **Question 3.3 :**
À l'aide des variables identifiées, exprimer l'invariant de sûreté.

▷ **Question 3.4 :**
Quelles sont les actions « critiques » que peut effectuer un lecteur ? Quelles sont celles d'un rédacteur ? Dans quelle(s) classe(s) ces actions doivent être ajoutées ?

▷ **Question 3.5 :**
Modifier maintenant le comportement du lecteur et du rédacteur pour qu'ils utilisent les nouvelles méthodes ajoutées.

▷ **Question 3.6 :**
Donner le pseudo-code des actions critiques identifiées.

▷ **Question 3.7 :**
Selon la méthode de construction d'une solution de synchronisation donnée plus haut, chaque condition d'attente est réalisée par une méthode. Donner le code de ces méthodes.

Exercice 4 (*Bloquer et débloquer des* threads *avec les moniteurs Java*)

Nous allons nous intéresser dans cet exercice à l'utilisation des moniteurs Java pour bloquer et débloquer effectivement les *threads* quand ceci est nécessaire.

En Java, la classe `Object` offre trois méthodes permettant de faire cela. Elles peuvent être appelées par un *thread* qui détient le verrou de l'objet, c'est-à-dire qu'il est possible de les invoquer dans une méthode `synchronized` uniquement [1] :

- `o.wait()` : le *thread* appelant libère le verrou de `o` et se bloque ;
- `o.notify()` : un *thread* au hasard parmi ceux bloqués dans un `wait` sur `o` est débloqué ;
- `o.notifyAll()` : tous les *threads* bloqués dans un `wait` sur `o` sont débloqués.

▷ **Question 4.1 :**
Utiliser les moniteurs Java pour traduire le pseudo code de la question 3.6.

Exercice 5 (*Et si on ajoutait des priorités ?*)

On souhaite ajouter une nouvelle contrainte d'ordonnancement à notre système : les rédacteurs doivent être prioritaires. Autrement dit un lecteur obtient le droit d'accès uniquement si aucun rédacteur n'a envie d'y accéder. Il n'y a cependant pas préemption de la ressource par les rédacteurs lorsque celle-ci est accédée par des lecteurs.

Ce genre d'exigence ne peut pas s'exprimer simplement sous la forme d'un invariant. Il faut renforcer la précondition d'accès.

▷ **Question 5.1 :**
Quelle condition faudrait-il ajouter pour l'accès des lecteurs à la base ? Exprimer cette condition en français.

Réaliser ces contraintes peut nécessiter d'ajouter des nouvelles variables auxiliaires.

▷ **Question 5.2 :**
Quelle(s) variables sont nécessaires pour exprimer la nouvelle condition d'accès ?

▷ **Question 5.3 :**
Modifier le code pour prendre en compte la nouvelle contrainte.

1. On peut aussi à l'intérieur d'un bloc `synchronized` utilisant `this` comme verrou.