Licence d'Informatique, IUP 3

Devoir de systèmes d'exploitation

e-bourse[1]

À faire en binôme.
Date de remise : vendredi 22 décembre 2000.

Le but de ce devoir est d'implanter en Java un service très simplifié de bourse en ligne. Ce projet vise à vous initier à la programmation concurrente et à l'utilisation des sémaphores et ne nécessite aucune virtuosité dans la maîtrise de Java.

Dans le document que vous rendrez en accompagnement de votre code, vous pourrez rédiger les programmes répondant aux questions dans le langage de votre choix, y compris dans un pseudo-langage tel que celui utilisé en TD. Vous y justifierez l'utilisation des différents sémaphores et variables. Vous y présenterez également les jeux de tests et leurs résultats, que vous commenterez. Les informations concernant le codage sont données à la fin de ce sujet.

1 La bourse, la banque et le courtier

Le modèle simple que nous nous proposons d'implanter ici comprend trois acteurs : une bourse, une banque et un courtier. Chacun de ces acteurs détient un certain nombre d'informations et doit pouvoir répondre à des requêtes, qui sont les suivantes :

La bourse : Elle connaît pour chaque action son nombre d'unités disponibles et sa valeur courante et elle doit répondre aux requêtes suivantes :

```
double valeur(action a)
int nombre(action a)
double acheter(action a, int n)
double vendre(action a, int n)
```

`valeur` donne la valeur courante d'une action et `nombre` le nombre d'unités disponibles de cette action. `acheter` et `vendre` permettent respectivement d'acheter et de vendre `n` unités de l'action `a`. L'entier retourné est la valeur d'achat ou de vente (qui est la valeur de l'action au moment de l'opération). Si le nombre d'unités demandées n'est pas disponible alors l'achat ne peut être réalisé ; vous choisirez alors une convention concernant la valeur retournée à cet effet.

[1] Prononcer « i-bourse » pour être dans le vent.

La banque : Elle connaît pour chaque individu le solde de son compte en banque et doit répondre aux requêtes suivantes :

```
boolean transfert(individu d, individu a, double s)
```

Une telle requête transfert la somme `s` du compte de l'individu `d` au compte de l'individu `a` et retourne un booléen indiquant si le transfert a été fait.

Le courtier : Il connaît pour chaque individu la nature et le nombre des actions qu'il possède, et il doit répondre aux requêtes suivantes :

```
achat(individu i, action a, int n)
vente(individu i, action a, int n)
```

qui permette à un individu `i` d'acheter ou de vendre `n` unités de l'action `a`.

Votre implantation devra respecter les contraintes suivantes :

1. La bourse, la banque et le courtier doivent être capables de traiter plusieurs requêtes en parallèle (lorsque cela est possible). En particulier, le courtier doit accepter plusieurs requêtes en parallèle d'un même individu.
2. Le courtier n'autorise pas plus de 5 requêtes (d'achat ou de vente) simultanées, tous individus confondus.
3. Un virement n'est effectué que si la somme est disponible sur le compte de départ.
4. Le nombre total de chaque action est limité à 1000. Lors d'une vente de n unités, la valeur d'une action perd $\frac{n}{15}$ %. Lors d'un achat de n unités, elle gagne $\frac{n}{15}$ %. (Vous pouvez modifier ces lois à votre guise pour plus de réalisme.)
5. La bourse et le courtier ont chacun un compte à la banque.
6. La bourse est toujours solvable.
7. Le courtier touche 10% de commission sur toute opération d'achat ou de vente.
8. C'est le courtier qui effectue toutes les requêtes auprès de la banque et de la bourse et qui se porte garant de leur validité ; en particulier, vous pouvez considérer que la bourse lui fait confiance (la banque, en revanche, n'a besoin de faire confiance à personne, car elle se contente de transférer de l'argent d'un compte à un autre). Mais notez que le courtier n'a pas accès aux soldes des comptes ; sa seule action vis-à-vis de la banque se limite à ordonner des virements et à en recevoir le résultat (succès ou échec).

Afin que le parallélisme puisse être observé, vous simulerez des délais artificiels (de l'ordre de quelques secondes) à divers endroits de votre programme, par exemple au moment d'effectuer un virement bancaire (simulant par là un délai d'autorisation).

Interface. On suppose connus statiquement la liste des individus et la liste des actions. Vous choisirez ces listes comme vous le souhaitez ; elles doivent être à la fois suffisamment grandes pour permettre d'illustrer les différentes situations possibles et raisonnablement petites. Ces deux listes seront données sur la ligne de commande sous forme de deux fichiers obéissant à la syntaxe suivante :

- pour la bourse

```
150 actions Eurotunnel à 40.5 F
1200 actions Microsoft à 122.0 F
...
```

– pour la banque

```
Pierre a 10000 F
Paul a 5200 F
...
Le courtier a 120000 F
```

Le compte de la bourse n'apparaîtra pas dans cette liste.

L'interface de votre programme se réduira à l'interprétation de commandes pour le courtier, données sur l'entrée standard sous la forme suivante :

`actions`
: pour obtenir les valeurs et nombres de toutes les actions ;

`Pierre achète 20 Eurotunnel`
: pour déclencher une requête d'achat ;

`Paul vend 57 Microsoft`
: pour déclencher une requête de vente ;

`quitte`
: pour terminer le programme.

Les résultats des requêtes seront affichés sur la sortie standard. Rien ne vous interdit d'écrire *aussi* une interface plus sophistiquée, mais l'interface en mode texte demandée sera utilisée pour tester votre programme. Lorsque votre programme termine il devra afficher le solde de tous les comptes, les actions détenues par chaque individu ainsi que les nombres et valeurs de toutes les actions en bourse.

Codage

Les divers processus seront implantés par des *threads* (processus légers). Ces threads pourront partager des informations globales (des sémaphores, bien sûr, mais également d'autres variables globales).

En Java, on crée un thread en invoquant la méthode `start` d'un objet d'une classe héritant de la classe pré-définie `Thread`. Cette méthode `start` appelle la méthode `run`, qui par défaut ne fait rien, et que vous redéfinirez pour y mettre le code de votre processus. Une utilisation typique est donc la suivante :

```
public class MonProcessus extends Thread {
  public void run() {
    /* son code ici */
  }
```

avec dans le programme principal :

```
public class MonProgramme {
  public static void main (String[] args) {
```

```
    /* ... */
    new MonProcessus().start();
    /* ... */
  }
}
```

La classe `Thread` contient une méthode `sleep` permettant de mettre le thread en sommeil un temps donné. Elle prend en argument un nombre entier de milli-secondes.

Une classe `Semaphore` implantant les sémaphores vous est fournie. Elle a la spécification suivante :

Constructeur	`Semaphore(int n)`
Méthodes	`void P()` `void V()`

Vous trouverez des informations générales sur Java et ses threads à l'adresse suivante : `http ://java.sun.com/docs/books/tutorial/`