Maitrise d'Informatique
Module PC2R

Programmation Concurrente, Réactive et Répartie

Université Pierre et Marie Curie
Année 2003–2004

Examen du 28 janvier 2004

Documents autorisés - Durée 3 heures

Exercice 1 : Moniteur à base de Mutex en Java

On cherche à simuler les moniteurs Java sur les méthodes *synchronized* en utilisant un mécanisme de *mutex*. Pour cela on définira une classe `Mutex` dont les instances seront les jetons gérant les exclusions mutuelles d'un programme multi-threads. Pour cela on définit les interfaces suivantes :

```
interface Mutexable {
  public void lock() throws InterruptedException;
  public void unlock();  }
```

```
interface Synchronizable {
  protected Mutex m;
}
```

On suppose définie une classe `Mutex` qui implante l'interface `Mutexable`. Elle possède un constructeur sans paramètre.

On cherche à définir une transformation de programme qui permette de faire passer toute classe possédant au moins une méthode *synchronized* en une classe qui implante `Synchronizable`. Une méthode *synchronized* devient alors une méthode non-*synchronized* utilisant le *mutex* `m`.

1. Indiquer comment transformer la méthode `meth` suivante définie dans une classe implantant `Synchronizable` (on suppose le *mutex* `m` correctement initialisé) :

   ```
   synchronized void meth(){ /* corps */ }
   ```

   Attention si une exception se déclenche dans *corps* le mutex doit être néanmoins libéré.

2. Soit la classe `Point` suivante :

   ```
   class Point {
     int a;
     int b;
     Point(){a=0; b=0;}
     synchronized void moveto{int a, int b){this.a=a; this.b=b;}
   }
   ```

   transformer-la pour qu'elle implante `Synchronizable` et que la méthode `moveto` n'utilise plus le mot clé *synchronized*.

3. Soit la class `PointColore` suivante :

   ```
   class PointColore extends Point {
     String c;
     PointColore(){c="";}
     synchronized display{){System.out.println(a+b+c);
   }
   ```

   indiquer la transformation à effectuer sur `PointColore` pour éliminer le mot clé *synchronized* sachant que cette classe hérite d'une classe implantant déjà `Synchronizable`.

4. Décrire la transformation de programme dans le cas général. Vous pouvez définir une nouvelle classe ancêtre pour les classes issues d'une classe possédant une méthode *synchronized*.

5. Ecrire un programme principal qui crée un objet `Point`, et qui lance deux threads qui déplacent chacun le point dans deux directions opposées au moins cinquante fois chacun.

6. Indiquer le mécanisme de synchronisation que vous utiliseriez pour implanter la classe `Mutex` et l'implanter en respectant votre description. Pour construire cette classe vous pouvez utiliser des méthodes *synchronized*.

Exercice 2 : Synchronisation en RMI

Le but de cet exercice est d'implanter une synchronisation sur des objets distan ts. On illustrera cette synchronisation sur un tableau d'objets.

```
interface Valeur extends Serializable {
  String toString();
  boolean inf(Valeur v);

}
class Bad_access extends Exception {};
class Bad_value extends Exception {};

interface Vecteur {
  public Valeur get(int i) throws Bad_access;
  public void set(int i, Valeur val) throws Bad_access;
  public int length();
}
```

1. Définir l'interface `VecteurDistant` qui étend les interfaces `Remote` et `Vecteur`.

2. Ecrire une classe `VecteurD` implantant l'interface `VecteurDistant`.

3. Ecrire un serveur RMI qui enregistre sous le nom `monvect` une instance de cette classe de taille 10 et où chaque élément du vecteur contient la valeur `null`.

4. Ecrire un client qui peut être lancé comme un *thread* et qui trie, de manière simple, en ordre croissant le vecteur distant `monvect`. On supposera qu'un autre client aura préalablement rempli le tableau par des instances d'une classe implantant l'interface `Valeur`.

5. Indiquer les situations possibles si deux clients de tri effectuent leur travail simultanément. On expliquera la différence si ces deux clients sont exécutés dans un même programme ou s'ils sont exécutés dans des machines virtuelles Java différentes.

6. Que faut-il ajouter au serveur pour gérer l'accès concurrent à ce vecteur distant? Implanter votre proposition.

Problème : ICFP 2002 (robots livreurs)

On se propose ici d'écrire un simulateur des robots livreurs reprenant en partie l'épreuve de programmation demandée à ICFP 2002[1]. Pour cela on commence par décrire le monde dans lequel évoluent les robots et les mouvements de ceux-ci.

Le monde Le monde est un rectangle composé de cases. On représente les cases par des caractères. Chaque case est soit vide ('.'), soit un mur ('#'), soit de l'eau ('~'), soit la base ('@'). Les robots peuvent se déplacer sur les cases vides. Leur but est d'atteindre la base. Les murs sont infranchissables. Si un robot tombe à l'eau, il se noie et disparaît du jeu. Les bords du monde sont des murs. Les coordonnées du monde sont des paires d'entiers, allant de (1,1) à (largeur,hauteur), où (1,1) est le coint sud-ouest du monde.

Les robots Un robot possède un identificateur unique (un entier positif). On se limite ici à des numéros compris entre 1 et 9.

Le joueur contrôle un robot en indiquant la direction du déplacement désiré : Nord, Sud, Est, Ouest, ainsi qu'une priorité de déplacement. A chaque étape du jeu, chaque robot exécute une commande.

[1] International Conference on Functional Programming : `http://icfpcontest.cse.ogi.edu/task.html`

Les déplacements L'arbitre du jeu détermine les déplacements des robots à chaque tour. Il détermine l'ordre de déplacement selon les priorités envoyées, décrites plus loin. Si un robot de déplace vers une case occupée par un autre robot, il le pousse si possible dans la direction choisie. Un robot poussé peut en pousser un autre. Si le robot occupant la case désirée ne peut pas être poussé (il y a un mur dans cette direction) alors aucun mouvement ne se produit. Le tableau suivant illustre les déplacements du robot `1` (noté `#1`) désirant aller à l'Est et du robot `2` (noté `#2`) désirant aller au Nord. Ces déplacements sont notés : `#1 E #2 N`

Avant	Si 1 bouge en premier	Si 2 bouge en premier
... 1.. .2.	.1. .2. ...	... .12 ...
... .1. .2.	... .21 ...	.1. .2. ...
.#. .1. .2.	.#. .21 ...	.#. .1. .2.
... .3. .1. .2.	... .3. .21 ...	.3. .1. .2. ...

L'initialisation La phase d'initialisation intervient dès qu'un client rejoint le jeu. Le protocole est le suivant :

- Client: envoie la ligne "Joueur"
- Serveur: envoie le monde et le numéro du joueur
- Serveur: quand tous les joueurs sont connectés, envoie la position initiale de tous les joueurs.

Transmission du monde

Le monde est envoyé dans le format suivant :

1. Serveur: les dimensions du tableau
2. Serveur: les n rangées du tableau (une par ligne)

Les dimensions sont deux entiers, le premier est la largeur (nombre de colonnes), le second est la hauteur (nombre de lignes) du tableau. Une fois les dimensions transmises, le serveur envoie une ligne pour chaque rangée du tableau.

Chaque rangée est une suite de caractères indiquant le type de la case correspondant à la position dans la rangée.

Voici un petit exemple de l'initialisation du monde reçue par le client correspondant au robot 3, il reçoit tout d'abord les dimensions du monde, la description de toutes les cases du monde et son numéro.

```
7 5
..@....
.......
##.~~~~
...~~~~
.......
3
```

Une fois l'ensemble des joueurs connectés, le serveur envoie la position de tous les joueurs à tous les joueurs. Les numéros des robots sont précédés du caractère `#`. Les positions sont de la forme `X posX Y posY` :

```
#2 X 2 Y 1 #1 X 1 Y 2 #3 X 1 Y 1
```

La position des joueurs est transmise en une seule ligne comme indiqué ci-dessus.

Transmission de la configuration Le jeu dure un nombre de tours illimités. A chaque tour se déroule de la manière suivante :

- Client : envoie une commande pour le robot
- Serveur : attend les commandes de tous les clients, puis effectue les déplacements de chacun et envoie à tous les clients les différents déplacements effectués pendant ce tour.

Commandes des robots Une commande d'un robot consiste en deux parties : la priorité et l'action. La priorité est une entier strictement positif inférieur à 100 et l'action une direction : 'N' 'S' 'E' 'O'. Voici des exemples de commandes : `1 N` ou `90 S`, toute commande mal formée tue le robot.

Réponse du serveur A la fin de chaque tour le serveur enverra à chaque joueur une réponse montrant la résolution des déplacements durant le tour. Tous les joueurs reçoivent la même information. Le format d'une réponse est une liste des déplacements des robots. Un robot est identifié par un numéro précédé du caractère `#`. Son déplacement est une des quatre directions possibles.
Exemple :
`#2 E #3 #1 N` (Le robot 2 a bougé vers l'Est, le robot 1 vers le Nord et le robot 2 n'a pas bougé.)
`#2 E #1 W E` (Le robot 2 est allé vers l'Est et le robot 1 a bougé vers l'Ouest puis vers l'Est (il a été poussé)).

Résolution des déplacements On déplace les robots selon l'ordre des priorités. Le robot de plus haute priorité effectue son action, puis on passe au robot de priorité suivante. En cas de même priorité on déplace en premier le robot de plus grand numéro. Attention le déplacement d'un robot peut pousser un ou plusieurs robots.

But du jeu : atteindre la base Le jeu s'arrête quand soit il n'y a plus de robots (il n'y a pas de gagnant) ou bien quand un robot est arrivé à la base, dans ce cas ce dernier est déclaré vainqueur. A la fin du jeu le serveur affiche le résultat localement et relance un nouveau jeu pour le même monde.

Découpage du problème On demande d'écrire un serveur de ce jeu en O'Caml et un client en Java dans l'ordre que vous préférez.

Indications Tant pour la partie Java que pour la partie O'Caml vous pouvez d'une part utiliser les clients et serveurs décrits dans les polycopiés en indiquant les modifications que vous y apportez et d'autre part vous pouvez supposer connues les fonctions d'analyse syntaxiques des messages du protocole décrit. Indiquer néanmoins leur nom et leur type pour les fonctions O'Caml et pour les méthodes et classes Java.

Serveur O'Caml On suppose que la description du monde, le nombre de tours, le nombre de clients et le port de communication sont donnés. Indiquez quand vous vous en server leur nom et leur type.

1) Ecrire un serveur qui attend la connexion de n clients et leur envoie les informations d'initialisation.
2) Ajouter la partie attente des actions d'un tour.
3) Ajouter la partie résolution des mouvements et détection de la fin de la partie
4) Ajouter la partie envoi des informations.

Client Java On suppose que le nom et le port de communication du serveur sont connus.

5 Ecrire la partie initialisation du client avec réception du monde et de la position initiale;

6 Ecrire la partie affichage du monde (en format texte); les robots sont représentés par leur numéro sur un caractère.

7 Ajouter une boucle d'interaction qui lit au clavier la direction et la priorité du mouvement de votre robot, envoie ces informations au serveur puis affiche le nouveau monde transmis par le serveur.

8 Proposer et implanter une stratégie tenant compte des risques de chute pour atteindre la base pour un client robot. Vous pouvez supposer connue la matrice des distances (en nombre de mouvements) de chaque case à la base.