Maitrise d'Informatique
Module PC2R

Programmation Concurrente, Réactive et Répartie

Université Pierre et Marie Curie
Année 2002–2003

Examen du 20 janvier 2003

Documents autorisés - Durée 3 heures

Exercice 1 : Simulation d'entrée au stade

On cherche à simuler l'entrée d'une foule dans un stade. Pour cela on représente l'univers dans lequel la foule se déplace sous la forme d'un tableau à deux dimensions :

```
.......#....
..A....#....
.BDC...#....
..E....#####
..FG.......x
..IH...#####
.......##...
.......##...
.......##...
cote        cote
queue       stade
```

où les `.` sont des cases libres, les `#` des obstacles, la lettre minuscule `x` la position à atteindre et les lettres majuscules des personnes. Pour la simulation, chaque personne sera considérée comme un processus léger. Pour atteindre son but, chaque personne cherchera à atteindre la position `x` correspondant au passage par les guichets. Une fois cette position atteinte la personne est considérée comme passée, elle disparait du monde et son processus s'arrête. Attention il ne peut pas avoir deux personnes sur la même case au même moment.
On définit le type suivant pour les personnes: `type personne = {nom:string; mutable x : int; mutable y : int;};;`, où les champs `nom`, `x`, `y` correspondent respectivement au nom et à la position de la personne
On suppose les fonctions suivantes connues :

- `but_atteint` de type `int -> int -> bool` qui prend les coordonnées d'une position et retourne `true` si la position passée vaut la position du but à atteindre et `false` sinon.
- `pos` de type `personne -> int * int` qui retourne la position d'une personne sous la forme d'un couple d'entiers.
- `suivante` de type `int * int -> int * int` qui retourne une nouvelle position (se rapprochant du but) à partir d'une position donnée sans vérifier si la case est occupée par une autre personne.
- `libre` de type `int*int -> bool` qui retourne `true` si la position est libre dans le monde et `false` sinon.

Dans la suite des questions, on suppose que les variables globales suivantes sont définies :

- `monde` : de type `char array array`
- `largeur`: de type `int`
- `hauteur`: de type ttint
- `but` : de type `int * int`

La personne `A` correspond à l'élément `monde.(1).(2)`.
Les indices du tableau vont de `0` à `hauteur-1` pour la première composant, et de `0` à `largeur-1` pour la deuxième composante.

1. Ecrire une fonction `liste_personnes` de type `char array array -> personnes list` qui prend un monde et construit la liste des personnes y figurant.

2. On suppose que la fonction `action` de type `personne -> unit -> unit` est définie. Elle effectue les différentes étapes de calcul et de déplacement pour qu'une personne atteigne son but. Ecrire la fonction `lance` de type `unit -> unit` qui lance un thread par personne apparaissant dans le tableau `monde`.

3. Ecrire la fonction `action` qui gère l'action d'une personne en utilisant un verrou d'exclusion mutuelle sur l'ensemble du monde. A chaque déplacement d'une personne, la position de la personne change et le monde est mis à jour.

4. Modifier la fonction `action` en utilisant un verrou par case du tableau. Pour cela on construit un tableau de mutex de même dimension que le monde.

5. On ajoute maintenant la notion de groupe de personnes. Un groupe cherche à ce que tous ses éléments ne soient pas trop distants, c'est-à-dire que la distance maximale entre deux éléments d'un groupe ne dépasse pas deux fois le nombre d'éléments du groupe. On suppose connue la fonction `distance` de type `personne -> personne -> int` qui retourne la distance entre deux personnes. La synchronisation entre les membres d'un groupe s'effectue par une condition. La ou les personnes les plus proches du but attendent les retardataires. Définir une fonction `action_groupe` de type `personne list -> (unit -> unit) list` qui construit la liste des fonctions d'action pour chaque personne du groupe.

6. Ecrire le lancement des threads pour ce monde avec deux groupes "ABCD" et "FIH" et les individus "E" et "G".

7. Donner un monde où il y a interblocage entre différents groupes. Justifier votre réponse. Proposer sans l'implanter une solution pour enlever ce risque d'interblocage entre les groupes.

Exercice 2 : Vectors en RMI

On cherche à définir une structure linéaire dynamique distante regroupant des objets respectant l'interface `ElementInt` suivante :

```
public interface ElementInt extends java.io.Serializable   {
 public String toString();
 public boolean egal(ElementInt e);
 public boolean inferieur(ElementInt e);
}
```

On définit d'autre part l'interface de ces "Vector"s.

```
import java.rmi.*;
public interface VectorDInt extends Remote {
  public void add(ElementInt e) throws RemoteException;
  public void remove(ElementInt e) throws RemoteException;
  public ElementInt elementAt(int i) throws RemoteException;
  public int size() throws RemoteException;
}
```

En voici une implantation

```
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
import java.util.*;

public class VectorD extends UnicastRemoteObject
                     implements VectorDInt {
  Vector v;
  VectorD() throws RemoteException {v = new Vector();}

  public void add(ElementInt e) throws RemoteException { v.add(e); }
```

```
  public void remove(ElementInt e) throws RemoteException { v.remove(e);}
  public ElementInt elementAt(int i)  throws RemoteException {return (ElementInt)v.elementAt(i);}
  public int size() {return v.size();}
}
```

1. Soit la classe `Creation` suivante :

```
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;

public class Creation {

  public static void main (String args[]) {

    if (System.getSecurityManager() == null) {
      System.setSecurityManager(new RMISecurityManager());
    }
         ...
  }
}
```

   Complétez cette classe pour rendre visibles deux instances de `VectorD` respectivement aux URL `rmi://exam.jussieu.fr/vecteur1` et `vecteur2`.

2. Soit la classe `Pair` suivante qui respecte l'interface `ElementInt` :

```
public class Pair implements ElementInt {
  String cle;
  int value;
  Pair (String k, int v) {cle=k; value=v;}
  public String toString() {return cle+":"+value;}
  public boolean egal(ElementInt e) {Pair p = (Pair)e;
    return this.egal(p);}
  public boolean egal(Pair p) {return cle.equals(p.cle);}
  public boolean inferieur(ElementInt e) {Pair p = (Pair)e;
    return this.inferieur(p); }
  public boolean inferieur(Pair p) { return value < p.value;}
}
```

   et le programme `Client.java` suivant :

```
import java.rmi.*;
public class Client {
  public static void main( String argv[]) {
    String machine = argv[0];
    String port = argv[1];
    String url1="rmi://"+machine+":"+port+"/vecteur1";
    String url2="rmi://"+machine+":"+port+"/vecteur2";
     try {

       VectorDInt v1 = (VectorDInt)Naming.lookup(url1);
       VectorDInt v2 = (VectorDInt)Naming.lookup(url2);

       v1.add(new Pair("XX",22));
       v1.add(new Pair("YY",23));
       v1.add(new Pair("ZZ",12));
```

```
   System.out.println(v1.size());
   for (int i=0; i< v1.size(); i++) {
     System.out.println(v1.elementAt(i).toString());
  }
}
catch (Exception e) {
  System.err.println("exception : " +
 e.getMessage());
  e.printStackTrace(); } } }
```

qui est lancé de la manière suivante sur la machine `cl1exam` :

```
java Client cl1exam.jussieu.fr 1099
```

indiquez ce que fait ce programme et précisez ce qu'il affiche et sur quelle machine.

3. On relance le serveur `Creation` sur la machine `exam.jussieu.fr` ainsi que deux clients `Client` sur deux machines différentes : `cl1exam` et `cl2exam`. Décrivez les différents scénarios possibles de remplissage du vecteur `vecteur1` et les conséquences sur les affichages.

4. Ecrire une sous-classe `VectorUD` de `VectorD` qui évite l'ajout de deux objets égaux (respectant l'interface `ElementInt`) au sens de la méthode `egal`. Effectuez une synchronisation sur le vecteur distant qui empêche les ajouts simultanés d'éléments égaux. Que se passe-t-il dans le cas où après avoir relancé le serveur on lance les deux clients précédents sur deux machines différentes. Indiquez les différents cas.

5. On cherche à ne plus être bloquant quand on ajoute un élément dans un vecteur distant. Pour cela définissez une sous-classe de `VectorUMD` qui lance un thread pour l'ajout d'un élément tout en évitant l'ajout d'éléments égaux.

6. On définit l'interface `VectorMaxDInt` qui étend l'interface `VectorDInt` de la manière suivante :

```
interface VectorMaxDInt extends VectorDInt {
  public ElementInt max();
}
```

   Ecrire une classe `VectorMaxD` qui implante cette interface. Pendant le calcul du maximum le vecteur ne doit pas être modifé par d'autres processus.

7. On désire aussi ne plus être bloquant pour le calcul du max. Pour cela on utilise le mécanisme de rappel de RMI.

   (a) Décrivez et implantez les différentes interfaces et classes pour définir de nouveaux vecteurs distants acceptant le calcul de l'élément maximum de manière non bloquante;

   (b) On suppose qu'un nouveau serveur est lancé sur la machine `exam` et expose 10 de ces nouveaux vecteurs sous les noms `vecteur1`, ..., `vecteur10`. Décrivez et implantez un nouveau client qui puisse lancer ce calcul de maximum sur ces 10 vecteurs distants et choisit le maximum de ces 10 résultats quand ils lui sont tous tranmis.