Université Bordeaux I
U.F.R. Mathématiques et Informatique

IUP MIAGe 3^{eme} année
Année 1999-2000

Mercredi 26 avril, 10 heures 15 - 12 heures 15
Tous documents autorisés.

Programmation Parallèle

Alain Griffault

Numéro d'anonymat :

1 Contrôle de variables partagées (5 points)

Soit le source suivant.

```
class Mouvement extends Thread {
    static StringBuffer a = new StringBuffer(); // attribut de classe
    static StringBuffer b = new StringBuffer(); // attribut de classe
    public void run() {
        a.append("a");
        b.append("b");
        System.out.println(a.toString()+b.toString());
    }
    public static void main(String []argv) {
        Mouvement zero = new Mouvement();
        Mouvement un = new Mouvement();
        zero.start();
        un.start();
    }
}
```

Question 1.1 (3 points) *Compte tenu des différents ordonnancements possibles, indiquer dans le tableau suivant les autres valeurs possibles des sorties lors d'une exécution.*

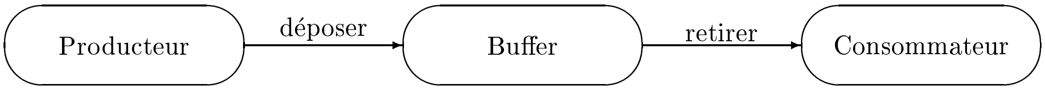
Thread zero	ab	aabb						
Thread un	aabb	ab						

Question 1.2 (2 points) *Modifiez simplement le source afin que ces autres résultats soient impossibles.*

```
class Mouvement extends Thread {
    static StringBuffer a = new StringBuffer(); // attribut de classe
    static StringBuffer b = new StringBuffer(); // attribut de classe
    public void run() {
        a.append("a");
        b.append("b");
        System.out.println(a.toString()+b.toString());
    }
    public static void main(String []argv) {
        Mouvement zero = new Mouvement();
        Mouvement un = new Mouvement();
        zero.start();
        un.start();
    }
}
```

2 Le problème du producteur-consommateur (7 points)

Dans ce problème, un producteur dépose des objets dans un buffer, qu'un consommateur retire ensuite.



Vous disposez de l'interface suivante qui décrit les actions possibles sur le buffer.

```
public interface Tampon{
    public void deposer(int nb);
    public int retirer();
}
```

du source de l'application.

```
import java.util.*;
class ProdCons {
    Producteur pr;
    Consommateur co;
    ProdCons(Tampon media) {
        pr = new Producteur(media);
        co = new Consommateur(media);
        pr.start();
        co.start();
    }
    public static void main(String []args) {
        Tampon media;
        if (args.length==0)
            media = new TamponV1();
        else
            media = new TamponV2(Integer.parseInt(args[0]));
        new ProdCons(media);
    }
    private class Producteur extends Thread {
        Random alea = new Random(); // permet de générer aléatoirement une suite d'entiers
        Tampon media;
        Producteur(Tampon buf) {
            media = buf;
        }
        public void run() {
            while (true)
                media.deposer(Math.abs(alea.nextInt()%10));
        }
    }
    private class Consommateur extends Thread {
        Tampon media;
        Consommateur(Tampon buf) {
            media = buf;
        }
        public void run() {
            int nb;
            while (true)
                nb = media.retirer();
        }
    }
}
```

et d'une première implémentation de l'interface.

```
import java.util.*;

public class TamponV1 implements Tampon{

    private Vector buf;

    TamponV1() {
        buf = new Vector();
    }

    synchronized public void deposer(int nb) {
        buf.addElement(new Integer(nb));
        System.out.print("+"+nb);
        notify();
    }

    synchronized public int retirer() {
        while (buf.isEmpty()) {
            try {
                wait();
            }
            catch (InterruptedException e){}
        }
        int nb = ((Integer)buf.remove(0)).intValue();
        System.out.print("-"+nb);
        return nb;
    }
}
```

Question 2.1 (1 point) *Justifiez le fait que les méthodes soient déclarées* `synchronized`.

Question 2.2 (1 point) *Justifiez l'utilisation de l'instruction* `wait();` *dans la méthode* `retirer()`.

Question 2.3 (1 point) *Justifiez l'utilisation de l'instruction* `notify();` *dans la méthode* `deposer(int nb)`.

Question 2.4 (4 points) *On désire maintenant limiter la taille du buffer. Donnez une nouvelle implémentation de l'interface* `Tampon.java` *qui respecte les contraintes :*

- *Le producteur ne peut pas déposer un message lorsque le buffer est plein.*
- *Le consommateur ne peut pas retirer un message lorsque le buffer est vide.*

Vous pourrez utiliser la méthode suivante :

```
Class Vector
    public int size()
        Returns the number of components in this vector.
```

et complétez en respectant précisément la syntaxe JAVA le source suivant.

```
import java.util.*;

public class TamponV2 implements Tampon{

    private Vector buf;
    int taille;

    TamponV2(int nb) {
        buf = new Vector();
        taille = nb;
    }

    synchronized public void deposer(int nb) {















    }

    synchronized public int retirer() {















        return nb;
    }
}
```

3 Le non-déterminisme (3 points)

Soit le programme CSP suivant :

```
Repetition ::

// déclaration et initialisation des variables
  var fini : boolean init false;

// une structure répétitive qui contient les commandes gardées.
  do [
     (not fini) --> fini = true;
        []
     (not fini) --> fini = false;
  ] od
```

Question 3.1 (1,5 points) *Complétez pour ce programme le tableau suivant :*

	Repetition	Justification
Le programme échoue t-il ?		
Le programme termine t-il ?		
Valeur finale de `fini`		

Soit cet autre programme CSP :

```
Alternance ::

// déclaration et initialisation des variables
  var fini : boolean init false;

// une structure alternative qui contient les commandes gardées.
  if [
     (not fini) --> fini = true;
        []
     (not fini) --> fini = false;
  ] fi
```

Question 3.2 (1,5 points) *Complétez pour ce programme le tableau suivant :*

	Alternance	Justification
Le programme échoue t-il ?		
Le programme termine t-il ?		
Valeur finale de `fini`		

4 Ordonnancement et client-serveur (5 points)

Soit un ensemble $\{T_1, \ldots, T_n\}$ de tâches contraintes par un graphe d'ordonnancement G.

Soient également des ouvriers polyvalent capables d'effectuer toutes les tâches, et un patron, incapable d'effectuer la moindre d'entre elles, mais organisateur hors pair.

Le patron et les ouvriers pouvant être sur des sites distincts, le travail est régit de la façon suivante :

1. Un ouvrier désirant travailler demande au patron une tâche a effectuer.
   (a) Si plusieurs tâches sont possibles, le patron demande à l'ouvrier d'en effectuer une dans un certain délai.
   (b) Si aucune tâche n'est possible, le patron demande à l'ouvrier de revenir un autre jour.
2. Lorsqu'un ouvrier a terminé son ouvrage, il avertit le patron, qui le paye pour le service effectué.
3. Lorsque le temps imparti à un ouvrier est écoulé, et que celui-ci n'a pas rendu son ouvrage, il est congédié sans dédommagement par le patron.

Question 4.1 (2,5 points) *En vous référant à des exercices vus en travaux dirigés, proposez une architecture pour ce problème.*

Question 4.2 (2,5 points) *Listez, expliquez et justifiez les mécanismes de synchronisations mis en œuvre.*