Laboratoire 8: Évaluation de performance

Par:
Jean-Philippe Bergeron
4656474

Le travail présenté à Gregor Bochmann dans le cadre du cours **SEG2506**

Université d'Ottawa
Le 7 avril 2008

Note:
10/10

Pénalité :

- but du devoir (assurer la bonne concurrence) manquée dans SDL : -2pt
  (Java a tous les mécanismes pour éviter ça!)

Bonus :

- SDL fonctionne! : +2 pts

Barême de correction --- en ligne avec la section anglaise.

1/1	Introduction
3/3	Modification made to the given Java program in Exercise 1,
3/3	and discussion on the design and implementation of the SDL version of the system
1/1	Results of the performance measurements
1/1	and discussion
1/1	Problems encountered and lessons learned
/0	Conclusions

Introduction

Nous avons concu 2 programmes en Java et en SDL afin d'évaleur la vitesse de chaque langage. Ces programmes sont consus d'une valeur partagée où deux processus veulent la lire et l'écrire en même temps.

On désire développer une petite application simulant les comportements concurrents d'un ensemble de producteurs et de consommateurs de messages. D'un côté, les producteurs produisent des messages qu'ils stockent dans un tampon commun; de l'autre, les consommateurs récupèrent dans ce même tampon des messages. Pour des raisons de simplicité, on suppose que la capacité du tampon est de 1 (au plus un seul message est dans le tampon à un temps donné). On doit respecter un certain nombre de contraintes et paramétrer l'application:

1. Bien sûr, un producteur ne peut produire un message que si le tampon n'est pas plein; dans cet exemple, le tampon doit être vide
2. Un consommateur ne peut lire un message que s'il y a un message dans le tampon
3. Les producteurs attendent en file dans le cas où le tampon contient déjà un message (ou quand un autre producteur met un message dans le tampon)
4. Les consommateurs attendent en file dans le cas où le tampon est vide (ou quand un autre consommateur prend le message dans le tampon)
5. Après avoir fini un producteur/consommateur réveille tous les consommateur/producteur dans la file, s'il y en a.

Partie Java

Premièrement, nous devons trouver pourquoi rien ne se produit lorsque nous exécutons le logiciel. Après avoir analysé le code, nous avons vu que dans les classes generateurEcrivain et generateurLecteur, le programme crée des Écrivains et des Lecteurs mais ne les démarre pas.

Trouvez pourquoi rien ne se produit

Nous avons modifié les fichiers Java donnés par le professeur pour les rendre fonctionnels. Premièrement, dans les classes generateurEcrivain et generateurLecteur, apres `Ecrivain e = new Ecrivain("ecrivain" + num++,V);` nous avons du rajouter `e.start();` et dans la classe lecteur, `l.start();`

Contrôlez le nombre de consommateurs et de producteurs générés

Je vais montrer ce que j'ai fait pour les écrivains, une technique similaire est utilisé du coté des lecteurs. Dans le constructeur, j'ai ajouté une variable `int num_ecrivains` qui permet de spécifier ce

nombre facilement. Ensuite, lorsque nous créons les écrivaine, j'ai mis une boucle for `for( int i=0;i<num_ecrivains;i++) {` qui crée ce nombre d'écrivains.

Un lecteur/écrivain recommence la lecture sans attendre.

Dans les classes Écrivains et Lecteurs, j'ai ajouté une boucle `while(!V.Fini()) {` qui permet d'écrire et lire à répétitions tant que la variable `V.Fini()` est fausse. Cette variable vient de la classe UneValeur et est une valeur booléenne qui dit quand le temps requis est passé.

```
public boolean Fini()
{
     return System.currentTimeMillis()>startTime+2000;
}
```

Lorsque deux secondes sont écoulés, la variable Fini devient vraie.

Compter le nombre de lectures et d'écritures effectuées

Ensuite, il y avait parfois un problème où l'ordre n'était pas respecté entre les lecteurs et les écrivains. Nous avons pu remédier à ce problème en remplacant `notify(); par notifyAll();`

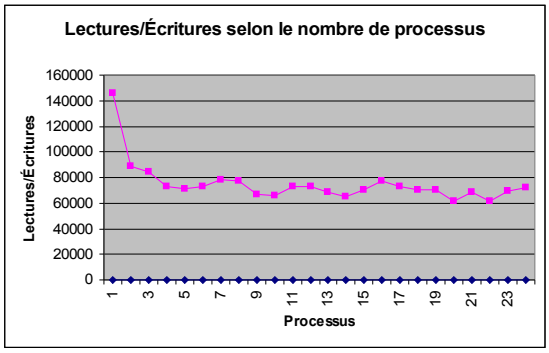
Afin de pouvoir contrôler le nombre de lectures et d'écritures, j'ai ajouté une variable `num_ecrivains` qui permet de compter le nombre d'écrivains. Lorsqu'un écrivain écrit, il met cette variable à jour dans la classe `generateurEcrivain` (Note. J'ai remarqué qu'il est possible que 2 processus accèdent cette variable en même temps et faussent les donnés. Par souci de simplicité, je n'arrangerai pas ce problème, car la différence n'est pas majeure.

Compte de lectures et écritures

Premièrement, commencons avec dix producteurs et dix consommateurs pour une période de 2 secondes. Nous obtenons les résultats suivants.

```
Lectures: 70408
Écritures: 70409
Générateur d'Écrivains 10
Générateur de Lecteurs 10

Écritures: 46351
Lectures: 46351
Générateur d'Écrivains 20
Générateur de Lecteurs 20
```

Il est à noter qu'il « manque » une lecture, ça arrive, car il a deux processus qui accèdent la variable `num_lecteurs` en même temps. Ensuite, si nous passons de nombreuses fois le test les nombres varient selon la vitesse de l'ordinateur et les autres programmes sur le système. J'ai décidé de construire un graphique avec plusieurs de ces résultats pour voir une certaine relation.



Nous pouvons voir qu'avec seulement un processus, le programme est beaucoup plus rapide, et, plus tard le nombre de processus n'importe pas trop sur la quantité de lectures et d'écritures. Personnellement, j'aurais pensé que la rapidité aurait diminuée avec le nombre de processus.

Code source Java

Classe Main

```
public static void main(String[] args)
{
    int i = 10;
    //for( int i=1;i<25;i++) { // Cette boucle m'a permis de construire le tableau
    UneValeur v = new UneValeur();
    Vector ecrivains = new Vector();
    Vector lecteurs = new Vector();
    generateurEcrivain ge = new generateurEcrivain(ecrivains, v, i);
    generateurLecteur gl = new generateurLecteur(lecteurs, v, i);

    ge.start();
    gl.start();
    try
    { // Nous attendons la fin pour écrire le nombre d'écrivains et de lecteurs
        ge.join();
        gl.join();
    }
    catch (InterruptedException e) { }
    System.out.println("Générateur d'Écrivains: " + ecrivains.size());
    System.out.println("Générateur de Lecteurs: " + lecteurs.size());
    //}
}
```

Classe generateurEcrivain

```
public class generateurEcrivain extends Thread
{
    private Vector list;
    private UneValeur V;
    public int ecritures;
    private int num_ecrivains;

    public generateurEcrivain(Vector l, UneValeur v, int num_ecrivains)
    {
        this.list = l;
        this.V = v;
        this.num_ecrivains = num_ecrivains;
    }

    public void run()
    {
        int num = 0;
        Ecrivain e[] = new Ecrivain[num_ecrivains]; // Nous créons un tableau d'écrivains
        for (int i = 0; i < num_ecrivains; i++)
        { // J'ai rajouté une boucle ici
            // generer un ecrivain
            e[i] = new Ecrivain("ecrivain" + num++, V, this);

            e[i].start(); //J'ai ajouté ceci pour exécuter l'écrivain

            list.addElement(e[i]);
        }
        // Ici, nous attendons que tous les écrivains aient fini pour imprimer le nombre d'écritures par la suite
        for (int i = 0; i < num_ecrivains; i++)
        {
            try
            {
                e[i].join();
            }
            catch (InterruptedException e1) { }
        }
        System.out.println("Écritures: " + ecritures);
    }
}
```

Classe generateurLecteur

Elle ressemble grandement à la classe Classe generateurEcrivain alors je ne la monterai pas ici.

Classe Ecrivain

```
public class Ecrivain extends Thread
{
    private UneValeur V;
    private String Nom;
    private generateurEcrivain ecr;


    public Ecrivain(String Nom, UneValeur V, generateurEcrivain ecr)
    {
```

```
        this.Nom = Nom;
        this.V = V;
        this.ecr = ecr;
    }

    public void run()
    {
        while (!V.Fini()) // On essaie d'écrire tant que le temps n'est pas fini
        {
            long val = Math.round(Math.random() * 100);
            V.ecrire(val, Nom);
            ecr.ecritures++;
        }
    }
}
```

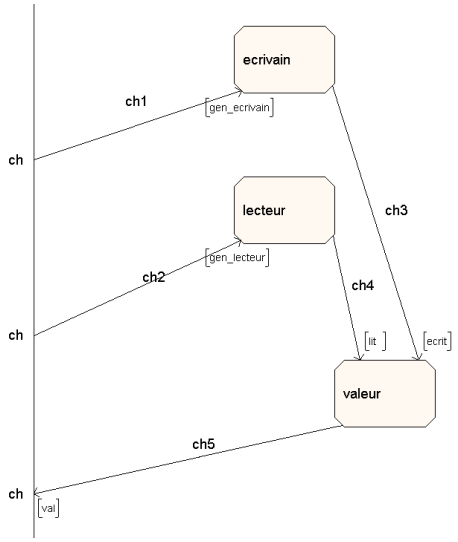
Classe UneValeur

```
public class UneValeur
{
    private long val = 0;
    private boolean val_Presente = false;
    private long startTime;

    public boolean Fini()
    {
        // Nous regardons si le temps passé dépasse 2 secondes
        return System.currentTimeMillis() > startTime + 2000;
    }

    public UneValeur()
    {
        // Dans le constructeur, nous initialisons le temps
        startTime = System.currentTimeMillis();
    }

    public synchronized long lire(String Nom)
    { // J'ai ajouté la variable nom pour afficher le lecteur à l'écran
        while (!val_Presente)
        {
            try
            {
                wait();
            }
            catch (InterruptedException e) { }
        }
        val_Presente = false;
        notifyAll(); //
        //System.out.println("lecteur " + Nom + " lit " + val);
        // J'ai mis l'écriture à l'écran dans cette partie du code pour que ca écrive au bon moment et non avant d'entrer dans la
boucle d'attente
        // J'ai également mis cette ligne en commentaire pour faire les test sans avoir a attendre l'écriture à l'aécran
        return val;
    }
    public synchronized void ecrire(long x, String Nom)
    { // J'ai ajouté la variable nom pour afficher l'écrivain à l'écran
        while (val_Presente)
        {
            try
            {
                wait();
            }
            catch (InterruptedException e) { }
        }
        val = x;
        //System.out.println("ecrivain " + Nom + " ecrit " + val); // Même chose pour cette ligne
        val_Presente = true;
        notifyAll(); //
    }
}
```

Partie SDL

Je dois avouer que j'ai eu de la difficulté avec la partie SDL. J'ai essayé de créer le programme le plus conforme possible, mais c'est impossible de calculer le temps en SDL. Vous pouvez voir le code dans la répertoire SDL.

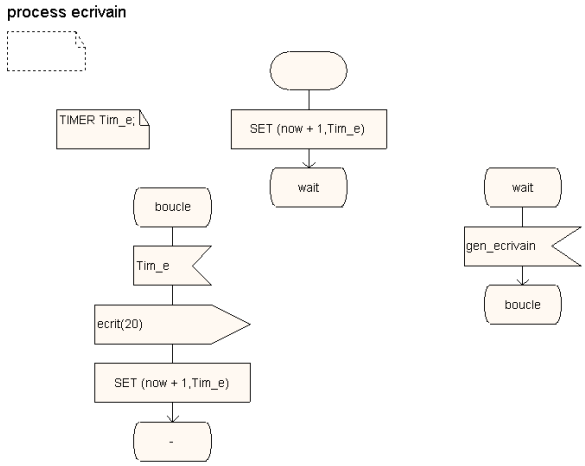
Main



J'ai commencé en faisant la structure principale. Elle contient un écrivain, un lecteur et une valeur. Au début, nous devons envoyer les signaux gen_ecrivain et gen_lecteur. Ces signaux activent 2 Timer qui génèrent des lecteurs de des écrivains. En fait, ils envoyent des signaux lit et ecrit.

Écrivain

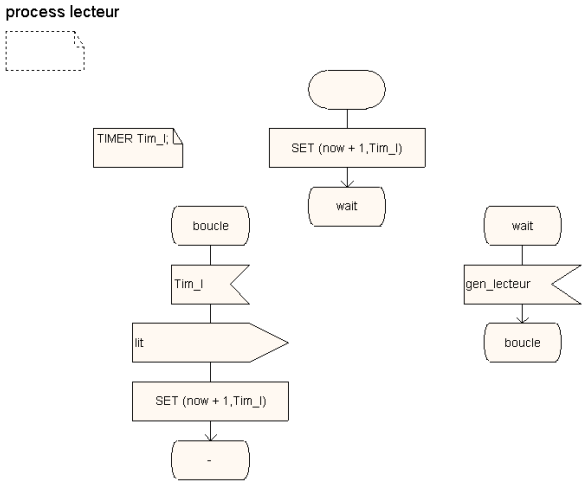
Nous pouons voir le processus écrivain ci-dessous



Il contient une boucle et à chaque unité de temps, il écrit la valeur 20.

Lecteur

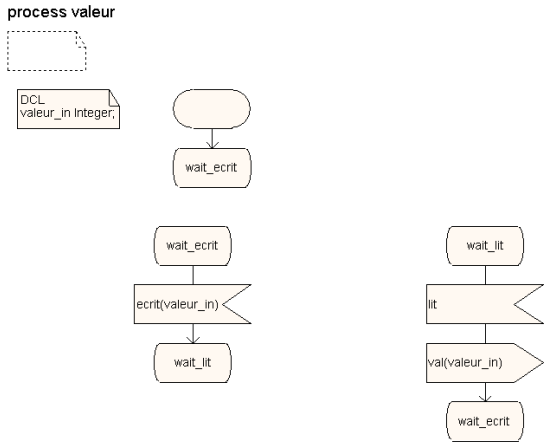
Ci-dessous est le processus lecteur.



À chaque unité de temps, il essaie de lire la valeur.

Valeur

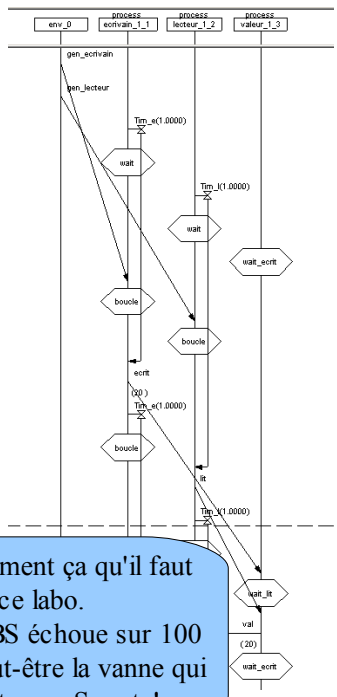
Maintenant, voyons le processus le plus intéressant : le processus valeur.



Au début, il entre dans l'état wait_ecrit où il attent qu'un processuis envoie le signal écrit. Ensuite, il emmagasine cette valur dans la variable valeur_in. Et ensuite se retrouve dans l'état wait_lit où il attend un signal de lecture. Lorsqu'il le recoit, il envoie le signal val avec l'argument valeur_in.

Résultats

Voyons comment ce programme fonctionne. Nous avons à droite, le fichier msc contenant la trace d'exécution. Au début, nous fournissons les signaux gen_ecrivain et gen_lecteur. Ensuite, nous allons dans une boucle où les écrivains et lecteurs lisent et écrivent à répétition.



Pourquoi? C'est justement ça qu'il faut éviter dans ce labo.
Si un seul système ABS échoue sur 100 000 voitures, c'est peut-être la vanne qui transporte 8 enfants aux Scouts!

Discussion

Je vais réexpliquer les problèmes rencontrés dans le rapport. Lorsque nous essayons de mettre une valeur à jour par plusieurs thread, il se peut que deux le fassent en même temps et que la valeur soit écrasée. Dans 100000 exécutions, il m'est arrivé de voir que la variable a été augmentée 99997 fois seulement. Je suis conscient que ce problème existe lorsque nous comptons le nombre de transactions effectués, mais l'erreur est tellement minime que ce n'est pas un grand trouble pour cette utilisation.

J'ai appris à utiliser les Thread dans ce rapport. Sur mon ordinateur à deux processeurs, l'utilisation de ceux-ci augmente beaucoup la rapidité d'exécution. C'est plus difficile à utiliser par contre, car il a des risques lors de l'utilisation de variables communes.

Contents