Ecole Polytechnique

Cours : "Calcul Parallèle"
Travaux dirigés
E. Goubault & T. Heindel

TD 1

22 janvier 2013

1 Démarrage de threads simples

Question Faire un programme JAVA qui démarre deux threads:

- Le premier qui attend 1 seconde avant de remplir un entier en mémoire partagée avec la valeur 42,
- Le deuxième qui examine en boucle la valeur de cet entier et attend que cette valeur devienne égale à 42.

Pour ce faire, il faut créer deux classes (une par processus) qui chacune sont des sous-classes de Thread, et une autre classe qui contient un main lançant les deux processus. L'entier en mémoire partagée sera en fait une "classe enveloppante" pour les entiers, on pourra utiliser par exemple la classe suivante:

```
public class UnEntier {
    int val;

    public UnEntier(int x) {
        val = x;
    }

    public int intValue() {
        return val;
    }

    public void setValue(int x) {
        val = x;
    }
}
```

Corrigé

```
public class Proc1 extends Thread {

    UnEntier refI;

    Proc1(UnEntier ent) {
        refI = ent;
```

```
    }

    public void run() {
try {
            System.out.print("\nP1 starts...");
    sleep(100);
    refI.setValue(42);
            System.out.print("\nP1 ends...");
} catch (InterruptedException e) { return ; }
    }
}

public class Proc2 extends Thread {

    UnEntier refI;

    Proc2(UnEntier ent) {
        refI = ent;
    }

    public void run() {
try {
            System.out.print("\nP2 starts...");
    while (refI.intValue()!=42) {
System.out.print("\nP2 waits...") ;
                sleep (5) ;
            } ;
            System.out.print("\nP2 ends...");
} catch (InterruptedException e) { return ; }
    }
}

public class Exo1 {

    public static void main(String[] args) {
        UnEntier i = new UnEntier(0);

new Proc1(i).start() ;
new Proc2(i).start() ;
    }
}
```

2 Threads récursifs: Fibonacci

Question Ecrire un calcul de la fonction f de Fibonacci, définie par la récurrence,

- $f(0) = 1$,
- $f(1) = 1$,
- $f(n+2) = f(n+1) + f(n)$.

en utilisant des threads JAVA. L'idée est de paralléliser l'algorithme récursif naturel permettant de calculer f (que l'on pourra écrire dans un premier temps si l'on ne se sent plus très à l'aise avec JAVA). Mais pour calculer $f(n+2)$, au lieu de s'appeler soi-même deux fois pour calculer $f(n+1)$ et $f(n)$, on créera un thread pour calculer $f(n+1)$ et un autre pour calculer $f(n)$. Ainsi, on définira une classe,

```
public class Fibon extends Thread {
...
    public void run() {
    ...
    }
}
```

dont la méthode run() sera en charge de calculer la fonction f. Pour ce faire il faut que la classe Fibon contienne un champ argument, et un champ résultat, ce dernier pouvant "survivre" dans la mémoire partagée au thread qui le calcule. Ainsi nous devons utiliser une "classe enveloppante" pour le résultat. On pourra utiliser pour ce faire la classe UnEntier vue précedemment.

On programmera donc,

- un constructeur Fibon(int x, UnEntier intref),
- le main de la classe: public static void main(String[] args),
- et enfin la méthode public void run() du thread.

Corrigé Pour vraiment devenir plus efficace, il faut arrêter de lancer de nouveaux «threads» à partir du moment ou on a autant de «threads» que cœurs (moins 1). Générer des nouveaux «threads» est très coûteux.

```
public class Fibon extends Thread {

    int arg;
    UnEntier res;

    Fibon(int x, UnEntier intref) {
        arg = x;
        res = intref;
    }

    public void run() {
        UnEntier res1 = new UnEntier(0);
        UnEntier res2 = new UnEntier(0);
        System.out.println("Fibo("+arg+")");
        if ((arg == 0) || (arg == 1))
            res.setValue(1);
        else {
            Thread x = new Fibon(arg-1,res1);
            Thread y = new Fibon(arg-2,res2);
            x.start();
            y.start();
            try {
                x.join();
                y.join();
            } catch(InterruptedException e) {};
            res.setValue(res1.intValue()+res2.intValue());
```

```
        }
    }

    public static void main(String[] args) {
        UnEntier R = new UnEntier(0);
        Thread t = new Fibon(Integer.parseInt(args[0]),R);
        t.start();
        try {
            t.join();
        } catch(InterruptedException e) {};
        System.out.println("Resultat="+R.intValue());
    }
}
```

3 Crible d'Erathostène

Question Ecrire un programme JAVA "par passage de messages" affichant la suite des nombres premiers en utilisant la méthode du crible : un processus est chargé de générer les entiers naturels, dont on élimine d'abord les multiples de 2, puis 3, 5, etc. au moyen de processus filtrants successifs. On utilisera les deux classes suivantes:

```
import java.util.*;

public class MsgQueue {
    Vector queue = new Vector();

    public synchronized void send(Object obj) {
        queue.addElement(obj);
    }

    public synchronized Object recv() {
        if (queue.size() == 0)
            return null;
        Object obj = queue.firstElement();
        queue.removeElementAt(0);
        return obj;
    }
}
```

Et:

```
public class Process {
    MsgQueue In;
    MsgQueue Out;

    public Process(MsgQueue i, MsgQueue o) {
        In = i;
        Out = o;
    }

    public void send(int x) {
        Out.send(new Integer(x));
```

```
// System.out.println(Thread.currentThread().getName()+": send("+x+")");
    }

    public int recv() {
        Object x = In.recv();
        while (x == null)
            x = In.recv();
        int res = ((Integer) x).intValue();
// System.out.println(Thread.currentThread().getName()+": recv "+res);
        return res;
    }
}
```

Corrigé

```
import java.util.*;


class MsgQueue {
    Vector queue = new Vector() ;

    public synchronized void send(Object obj) {
        queue.addElement(obj);
    }

    public synchronized Object recv() {
        if (queue.size()==0)
            return null;
        Object obj = queue.firstElement();
        queue.removeElementAt(0);
        return obj;
    }

    public synchronized int size() {
        return queue.size();
    }
}

class Process {
    MsgQueue In;
    MsgQueue Out;

    public Process(MsgQueue i,MsgQueue o) {
        In=i;
        Out=o;
    }

    public void send(int x) {
        Out.send(new Integer(x));
    }

    public int recv() {
```

```
        Object x = In.recv();
        while (x==null)
    x=In.recv();
        int res = ((Integer) x).intValue();
        return res;
    }
}

class Generateur extends Thread {
    int n;
    Process p;

    Generateur(int max,MsgQueue mq) {
n=max;
p=new Process(null,mq);
    }

    public void run() {
for(int i=2;i<=n;i++) {
    p.send(i);
};
p.send(-1);
    }
}

class Filtre extends Thread {
    int filtre;
    Process p;

    Filtre(MsgQueue mq) {
        p=new Process(mq,new MsgQueue());
    }

    public void run() {
        int i;

filtre = p.recv();
if (filtre != -1) {
    System.out.println(filtre);
    new Filtre(p.Out).start();
            i=p.recv();
            while (i!=-1) {
if (i % filtre != 0) {
    p.send(i);
};
                i=p.recv();
    };
    p.send(-1);
}
    }
}
```

```
class Eratosthene {

    public static void main(String[] args){
if (args.length<1) {
    System.out.println("Usage : java Eratosthene n");
    return;
}
MsgQueue mq = new MsgQueue();
new Generateur(Integer.parseInt(args[0]),mq).start();
new Filtre(mq).run();
    }
}
```

4 Calcul de π en parallèle

Implémenter en utilisant des threads JAVA le calcul de π en parallèle par la formule suivante,

$$\begin{aligned} \pi &= \int_0^1 \frac{4}{1+x^2} dx \\ &\cong \sum_{i=1}^{n} \frac{1}{n} \frac{4}{1+\left((i-\frac{1}{2})\frac{1}{n}\right)^2} \end{aligned}$$

Une solution est le paradigme **Maître/Esclave**: Un maître va lancer N esclaves chargés de calculer les sommes partielles,

$$P_k = \sum_{i=k*n/N+1}^{(k+1)*n/N} \frac{1}{n} \frac{4}{1+\left((i-\frac{1}{2})\frac{1}{n}\right)^2}$$

pour $k = 0, \ldots, N-1$.

Corrigé

```
package tds;

public class PiParallele extends Thread {
static int nminuscule = 10000 ; // valeur defaut
static int nmajuscule = 1000; // valeur par défaut

double resultat;
int debut;
int fin;

PiParallele(int d, int f){
resultat = 0;
debut = d;
fin = f;
}
public void run(){
```

```
int i = fin;
for (; i>= debut;i--){
    double delta =  4/(1+((i-0.5)*1/(double) nminuscule )*((i-0.5)*1/(double) nminuscule));
resultat += delta;
//System.out.println(" "+delta);
}
System.out.println( "La somme partielle de "+ debut+" à "+ i + " est " + resultat+".");
}
double getResult(){
return resultat ;
}

public static void main (String[] args){
if (args.length > 1){
nminuscule = Integer.parseInt(args[0]);
nmajuscule = Integer.parseInt(args[1]);
}
if (nminuscule % nmajuscule !=0){
System.out.println("$N$ ne divise pas $n$! Le calcul ne sera pas correcte.");
}
PiParallele[] p = new PiParallele[nmajuscule]; // champ d'*objets*
Thread[] t = new Thread[nmajuscule]; // champ de *threads*
for (int k = nmajuscule-1; k >=0 ; k--){
p[k] = new PiParallele((k*nminuscule)/nmajuscule +1,
((k+1)*nminuscule)/nmajuscule);
t[k] = new Thread(p[k]);
t[k].start();
}
double mastersum = 0;
for (int k = nmajuscule-1; k >=0 ; k--){
try {
t[k].join();
} catch (InterruptedException e){
System.out.println("Master interrupted!");
}
mastersum += p[k].getResult();
}
System.out.println("pi en approximation: " + mastersum/nminuscule);
}
}
```

5 Tri de Hoare

Question Programmer le *quicksort* d'une liste d'entiers en essayant de paralléliser l'algorithme séquentiel. Que gagne-t'on en complexité (moyenne, pire cas ?)

Corrigé

```
package tds;
```

```
import java.util.Random;


public class TriHoare extends Thread {
volatile int[] lechamp;
int iInf;
int iSup;

protected void swap (int i, int j){
int tmp = lechamp[j];
lechamp[j] = lechamp[i];
lechamp[i] = tmp;
}

TriHoare(int[] c, int petit, int grand){
lechamp = c;
iInf = petit;
iSup = grand;
}

public void run(){
if (iSup <= iInf){
// rien à faire
} else {
// recursion

int l = iInf;
int r = iSup -1;
int valeur = lechamp[iSup];// choix de pivot triviale/bête


while (l < r){
while (lechamp[l] <= valeur && l < iSup){
l++;
}
while (lechamp[r] >= valeur && r > iInf){
r--;
}
if (l < r){
swap (l , r);
}
}
if (lechamp[l] > valeur){
swap(l,iSup);
}

Thread t1, t2;
t1 = new Thread (new TriHoare(lechamp,iInf,l-1));
t2 = new Thread (new TriHoare(lechamp,l+1,iSup));
t1.start();
t2.start();
```

```
try{
t1.join();
t2.join();
} catch (InterruptedException e){}
}
}

public static void main (String[] args){
int n = 5000;
if (args.length >0){
n = Integer.parseInt(args[0]);
}
int[] champ = new int[n];
Random rand = new Random();
for (int i = 0; i<n; i++){
champ[i] = rand.nextInt(n*100);
}
Thread t;
t = new Thread (new TriHoare(champ,0,n-1));
t.start();
try{
t.join();
} catch (InterruptedException e){}

for (int i = 0; i<n-1; i++){
System.out.print(champ[i]+", ");
}
}
}
```