Programmation réseau et concurrente

Benoît Barbot

Département informatique, Université Paris-Est Créteil, M1

Mardi 23 février 2016, Cours 4 : Concurence en Java

Plan

Exclusion mutuelle avec un lock

Lock

- Interface `Lock`
- Méthode `.lock()`
- Méthode `.unlock()`

Exclusion mutuelle avec un lock

Lock

- Interface Lock
- Méthode .lock()
- Méthode .unlock()

Exemple – Demo

```
Lock l = new ReentrantLock();
long c;
...
l.lock();
try {
   c++;
} finally {
   l.unlock();
}
```

Attente active / inactive

Attente active

- await(c) := **while**(!c){}
- Le CPU teste en permanence la condition
- Rapide
- Perte de puissance de calcul

Attente active / inactive

Attente active

- await(c) := **while**(!c){}
- Le CPU teste en permanence la condition
- Rapide
- Perte de puissance de calcul

Attente inactive

- Demande au système de mettre en pause (pthread)
- Le CPU est libéré
- Plus lent en temps horloge
- Économie de la puissance de calcule

Attente active / inactive 2

Experiences

```
public void run() {
  for(long i=0; i< 1000000;i++){
    s.l.lock();             /* O */
    try {
      for(int j=0;j<100000;j++) {
        s.c++;
      }
    } finally {
      s.l.unlock();         /* O */
}
```

Attente active / inactive 2

Experiences

```
public void run() {
  for(long i=0; i< 1000000;i++){
    s.l.lock();             /* O */
    try {
      for(int j=0;j<100000;j++) {
        s.c++;
      }
    } finally {
      s.l.unlock();         /* O */
}
```

Résultat

	s.c	user	system	CPU	total
Avec Lock	200000000000	5,60s	2,56s	142%	5,723s
Sans Lock	102408776278	4,73s	0,05s	189%	2,517s
Dekker/Peterson	200000000000	8,37s	0,04s	197%	4,268s

Exclusion mutuelle – Synchronize

Synchronization

- Mot clé sychronized
- Utilisation de n'importe lequel Object comme lock
- Notation implicite avec **this**

Exclusion mutuelle – Synchronize

Synchronization

- Mot clé sychronized
- Utilisation de n'importe lequel Object comme lock
- Notation implicite avec **this**

synchronized notation explicite

```
class ObjetPartage {
  long c = 0;
  public void increment() {  c++; }
}
...
ObjetPartage s;
...
synchronized(s){
  ...
}
```

Exclusion mutuelle – Synchronize

synchronized notation implicite

```
public void synchronized increment() {
  c++;
}
```

≡

```
public void increment() {
  synchronized(this){
    c++;
  }
}
```

Exclusion mutuelle – Synchronize

synchronized notation implicite

```
public void synchronized increment() {
  c++;
}
```

≡

```
public void increment() {
  synchronized(this){
    c++;
  }
}
```

Attention à l'objet de synchronization

```
class MThread extends Thread {
  syncrhonized sectionCritique(...){...}
  ...
}
MThread T1 = new Mthread();
MThread T2 = new Mthread();
```

⇒ sectionCritique n'est pas une exclusion mutuelle entre $T1$ et $T2$

Exercise exclusion mutuelle

Implémenter une pile concurrente

- Méthode pop
- Méthode push
- Méthode isEmpty

Plan

Synchronize et Moniteur

Moniteur

- Tout objet est muni d'un *moniteur*
- Un thread *possède* le moniteur de `obj` dans un block `synchronize(obj ){...}`
- Un seul thread possède le moniteur à la fois
- Permet des synchronisations plus complexes
- (les moniteurs ne sont pas des objets)

Synchronize et Moniteur

Moniteur

- Tout objet est muni d'un *moniteur*
- Un thread *possède* le moniteur de `obj` dans un block `synchronize(obj ){...}`
- Un seul thread possède le moniteur à la fois
- Permet des synchronisations plus complexes
- (les moniteurs ne sont pas des objets)

Méthode `.wait()` et `.notify()`

- Système de signaux sur un moniteur
- `.wait()`, `.wait(long time)` attends un signal sur le moniteur.
- `.notify()` envoie un signal à un thread en attente sur le même moniteur.
- `.notifyAll()` envoie un signal à tous les threads en attente.

Synchronize et Moniteur

Moniteur

- Tout objet est muni d'un *moniteur*
- Un thread *possède* le moniteur de `obj` dans un block `synchronize(obj ){...}`
- Un seul thread possède le moniteur à la fois
- Permet des synchronisations plus complexes
- (les moniteurs ne sont pas des objets)

Méthode `.wait()` et `.notify()`

- Système de signaux sur un moniteur
- `.wait()`, `.wait(long time)` attends un signal sur le moniteur.
- `.notify()` envoie un signal à un thread en attente sur le même moniteur.
- `.notifyAll()` envoie un signal à tous les threads en attente.

Wait et moniteur

Le moniteur est relâché pendant l'attente !

Producteur – Consommateur

```
class SharedVar {
  boolean b = false;
  int msg = 0;
  public synchronized void addMsg(int n) {
    while( b ) try { wait(); } catch (InterruptedException e) {};
    b = true;
    msg =n;
    notify();
  }
  public synchronized int readMsg() {
    while( !b ) try { wait(); } catch (InterruptedException e) {};
    b= false;
    notify();
    return msg;
  }
}
```

Utilisation des moniteurs

Structure de donnée concurentes

- Array
- Pile, File, List
- Tree, Set, Maps

Utilisation des moniteurs

Structure de donnée concurentes

- Array
- Pile, File, List
- Tree, Set, Maps

Déjà implémenté dans java.util.concurrent.*

- LinkedBlockingDeque
- ConcurrentHashMap
- ArrayBlockingQueue
- ...

Interruption de thread

InterruptedException

- Lancé par `Thread.join()`, `Object.wait()`
- Provoqué par `Tread.interrupt()`

Interruption de thread

InterruptedException

- Lancé par `Thread.join()`, `Object.wait()`
- Provoqué par `Tread.interrupt()`

Interruption en dehors d'une attente

- `bool isInterrupted()`
- `bool interrupted()` remet `isInterrupted` à **false**

Interruption de thread

InterruptedException

- Lancé par `Thread.join()`, `Object.wait()`
- Provoqué par `Tread.interrupt()`

Interruption en dehors d'une attente

- `bool isInterrupted()`
- `bool interrupted()` remet `isInterrupted` à **false**

Demo

Exercise exclusion mutuelle

Implémenter une pile concurrente et bloquante

- Méthode pop
- Méthode push
- Méthode isEmpty

Plan

Lock II

Autres méthodes

- Méthodes **boolean** .tryLock(), **boolean** .tryLock(**long** time)
- Méthode Condition .newCondition()

Lock II

Autres méthodes

- Méthodes **boolean** .tryLock(), **boolean** .tryLock(**long** time)
- Méthode Condition .newCondition()

Variable conditionné

- Obtenu avec Lock.newCondition()
- Méthode . signal équivalent de Object. notify
- Méthode . await équivalent de Object. wait
- Méthode . awaitUninterruptibly () Ignore les interruptions.

Bibliothèque Atomique

Opération atomique

- Opération non interrompue par un autre thread
- Non bloquant, très rapide
- Construit à l'aide de *Test and Set* (instruction processeur)
- getAndSet(V newValue)
  Mets à jour la valeur et renvoie l'ancienne valeur
- compareAndSet(V expect, V update)
  Mets à jour la valeur si égale à la valeur attendue

Bibliothèque Atomique

Opération atomique

- Opération non interrompue par un autre thread
- Non bloquant, très rapide
- Construit à l'aide de *Test and Set* (instruction processeur)
- getAndSet(V newValue)
  Mets à jour la valeur et renvoie l'ancienne valeur
- compareAndSet(V expect, V update)
  Mets à jour la valeur si égale à la valeur attendue

Classe disponible dans java.util.concurrent.atomic.*

- AtomicBoolean
- AtomicInteger
- AtomicReference
- AtomicIntegerArray
- ...

Exemple LinkedBlockingQueue

java.util.concurrent.LinkedBlockingQueue