Programmation Multi-Threading en Java

Christophe Gravier, Frédérique Laforest, Julien Subercaze

Télécom Saint-Étienne
Université Jean Monnet

{prénom.nom}@univ-st-etienne.fr

FI2_INFO4 2012–2013

Plan

Objectifs

Multi-Threading

Les solutions de synchronisation

Objectifs pédagogiques

Obligatoires

- Principe de la programmation Multi-Thread
- Mécanismes de synchronisation

Optionnel

- Réalisation d'une application lecteurs/rédacteurs

Plan

Objectifs

Multi-Threading

Les solutions de synchronisation

Processus et Threads

Définition : processus

Un processus possède son propre environnement d'éxécution, en particulier un espace mémoire réservé. Généralement processus = application.

Définition : Thread

Les threads ou *processus légers* existent au sein d'un processus (au moins 1). Les threads partagent les ressources du processus.

Dans ce cours

- Nous verrons seulements les threads

Utilisation

Les applications multi-thread permettent une gestion plus efficace des ressources mais sont en contrepartie plus difficiles à mettre en œuvre pour des raisons de synchronisation.

Utilisation

Les applications multi-thread permettent une gestion plus efficace des ressources mais sont en contrepartie plus difficiles à mettre en œuvre pour des raisons de synchronisation.

Quand utiliser le multi-threading

- Calcul intensif qui peut être réparti sur plusieurs processeurs ou cœurs.
- Ralentissement dû aux temps d'attente I/O, typiquement réseau
- Eviter le *freeze* sur des machines monocœur.
- GUI

Utilisation

Les applications multi-thread permettent une gestion plus efficace des ressources mais sont en contrepartie plus difficiles à mettre en œuvre pour des raisons de synchronisation.

Quand utiliser le multi-threading

- Calcul intensif qui peut être réparti sur plusieurs processeurs ou cœurs.
- Ralentissement dû aux temps d'attente I/O, typiquement réseau
- Eviter le *freeze* sur des machines monocœur.
- GUI

Quand ne pas utiliser

- Faible temps de calcul $\rightarrow$ ralentissement dus aux *switchs*
- Machine à un seul processeur, un seul cœur.

A programmer has one problem

A programmer has one problem

He thought, "I know, I'll use threads."

A programmer has one problem

He thought, "I know, I'll use threads."

Now the programmer has two problems

l'objet Thread

Thread en Java

Deux objets possibles :

- `Thread`
- `Runnable`

```java
public class HelloRunnable implements Runnable {

    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        (new Thread(new HelloRunnable())).start();
    }

}
```

Exemple

```java
package fr.tse.info.tp7.exemple;

public class Exemple1Thread1 implements Runnable {
   int numero;
   public Exemple1Thread1(int numero) {
      super();
      this.numero = numero;
   }
   @Override
   public void run() {
      int i = 0;
      while (i < 5) {
         System.out.println("Thread " + numero);
         i++;
      }
   }

}
```

Exemple - 2

```
1  package fr.tse.info.tp7.exemple;
2
3  public class Exemple1 {
4
5    public static void main(String[] args) {
6      Exemple1Thread1 thread1 = new Exemple1Thread1(1);
7      Exemple1Thread1 thread2 = new Exemple1Thread1(2);
8      new Thread(thread1).start();
9      new Thread(thread2).start();
10   }
11
12 }
```

Démonstration

Démonstration

Résultat

Le résultat n'est pas identique à chaque exécution !

Démonstration

Résultat

Le résultat n'est pas identique à chaque exécution !

Race condition - Situation de concurrence

C'est le nom donné à ce type de situation, ou le résultat est dépendant d'évenèments incontrollables par le programme. Ici c'est l'allocation du processeur par l'OS qui n'est pas contrôlé.

Difficulté du multi-threading - II

Deadlock - Interblocage

Deux (ou +) threads sont bloqués indéfiniment.

1. Alphonse et Gaston sont très polis
2. Quand on salue un ami, on se doit de rester courber jusqu'à ce que son ami se soit relevé.
3. Si A. et G. se courbent en même temps, ils restent bloqués

Difficulté du multi-threading - II

Deadlock - Interblocage

Deux (ou +) threads sont bloqués indéfiniment.

1. Alphonse et Gaston sont très polis
2. Quand on salue un ami, on se doit de rester courber jusqu'à ce que son ami se soit relevé.
3. Si A. et G. se courbent en même temps, ils restent bloqués

Starvation - Famine

Situation dans laquelle un thread ne peut avoir accès à une ressource partagée et ne peut progresser dans son traitement. Si un thread *glouton* appelle une méthode synchronisée très souvent, les autres threads peuvent se retrouver bloqués.

Difficulté du multi-threading - II

Deadlock - Interblocage

Deux (ou +) threads sont bloqués indéfiniment.

1. Alphonse et Gaston sont très polis
2. Quand on salue un ami, on se doit de rester courber jusqu'à ce que son ami se soit relevé.
3. Si A. et G. se courbent en même temps, ils restent bloqués

Starvation - Famine

Situation dans laquelle un thread ne peut avoir accès à une ressource partagée et ne peut progresser dans son traitement. Si un thread *glouton* appelle une méthode synchronisée très souvent, les autres threads peuvent se retrouver bloqués.

Livelock

Similaire au deadlock, mais changement constant d'état.

- A. et G. se croisent dans un corridor
- Chacun veut éviter l'autre et se retrouve sans cesse bloqué

Plan

Objectifs

Multi-Threading

Les solutions de synchronisation

Inventaire des solutions

Synchronisation de threads

- **Join**
- Wait
- Notify

Inventaire des solutions

Synchronisation de threads

- **Join**
- Wait
- Notify

Synchronisation des données

- volatile
- Immutabilité
- **Objets Atomiques**

Inventaire des solutions

Synchronisation de threads

- **Join**
- Wait
- Notify

Synchronisation des données

- volatile
- Immutabilité
- **Objets Atomiques**

Synchronisation du code

- **synchronized**
- **Locks**

Synchronisation de threads

join

permet d'attendre la fin d'éxécution d'un thread

```
1  package fr.tse.info.tp7.exemple;
2  public class Exemple1 {
3
4    public static void main(String[] args) {
5      Thread thread1 = new Thread(new Exemple1Thread1(1)
6      Thread thread2  new Thread(new Exemple1Thread1(2)
7      thread1.start();
8      thread1.join();
9      thread2.start();
10     thread2.join();
11     System.out.println("Fin du programme");
12   }
13
14 }
```

Synchronisation de threads

join

permet d'attendre la fin d'éxécution d'un thread

```
1  package fr.tse.info.tp7.exemple;
2  public class Exemple1 {
3
4    public static void main(String[] args) {
5      Thread thread1 = new Thread(new Exemple1Thread1(1)
6      Thread thread2  new Thread(new Exemple1Thread1(2)
7      thread1.start();
8      thread1.join();
9      thread2.start();
10     thread2.join();
11     System.out.println("Fin du programme");
12   }
13
14 }
```

Résultat ?

Objets Atomtiques

Fournis par le JDK

Le package `java.util.concurrent.atomic` contient de nombreuses d'objets qui peuvent être accédés et modifiés de manière *thread-safe*.

`AtomicBoolean, AtomicLong, ...`

Méthodes - Exemple sur `AtomicLong`

- `long get()`
- `long incrementAndGet()`
- ...

The End