Software Distribuït - T5 - Threads-II

Eloi Puertas i Prats

Universitat de Barcelona
Grau en Enginyeria Informàtica

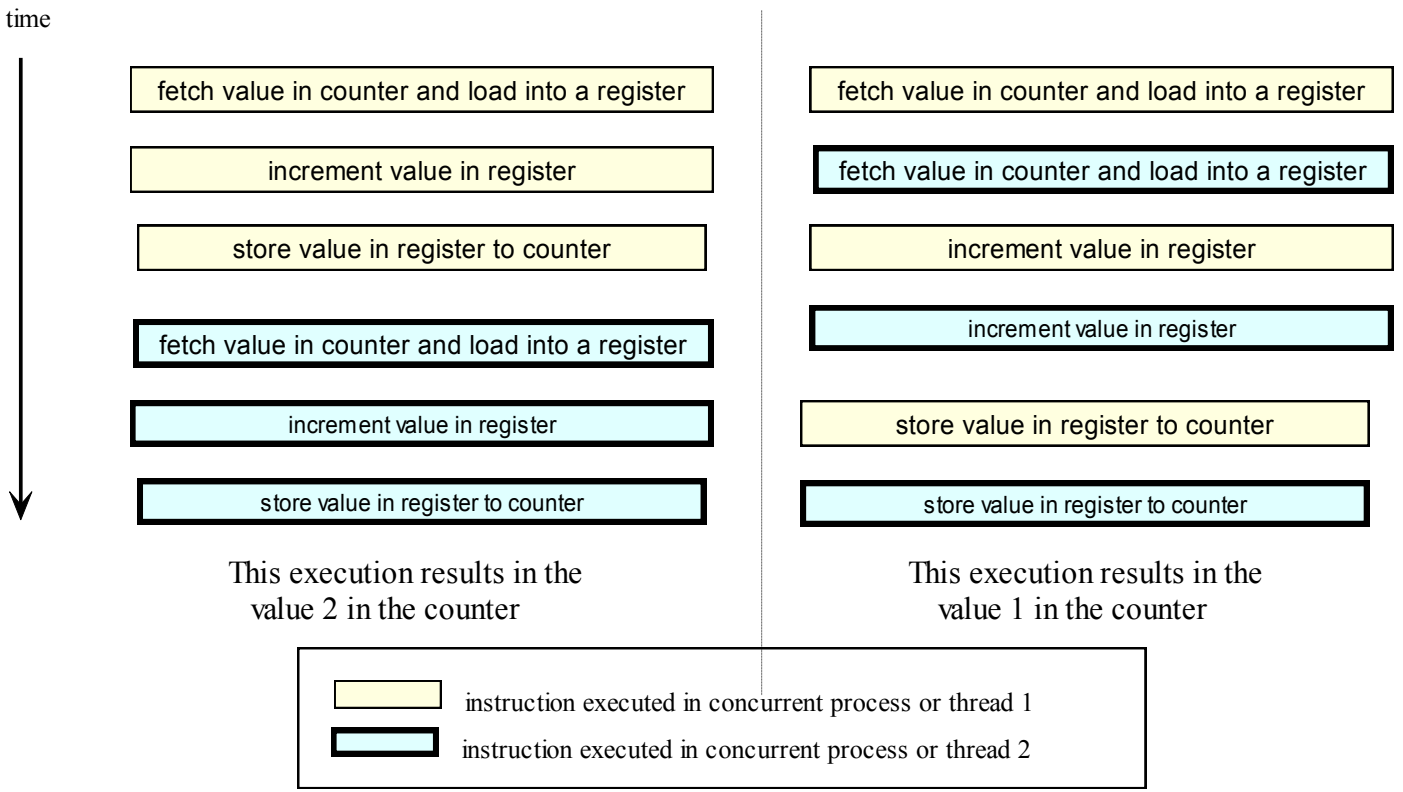
27 de març de 2014

UNIVERSITAT DE BARCELONA

Problemes dels Threads

- **Problemes de seguretat:**
  Ens hem d'assegurar que no passi res dolent degut a l'accés simultani de diversos threads a zones de memòria.
  Solució: Regions d'exclussió mútua: evitar accés simultani a instruccions que comprometin la integritat de les dades.
- **Problemas d'esperes:**
  - Deadlock: Dos o més processos es bloquegen mutuament per a poder realitzar els seus objetius.
  - Starvation: Degut a la forma (injusta) d'assignar els recursos, algún procés no pot realitzar els seus objetius.
  - Livelock: Varis processos no són capaços de realitzar els seus objetius, tot i que es mantenen actius realitzant còmput.
- **Problemes d'eficiència.**
  Hem d'assegurar que es facin el màxim comput simultani possilbe.

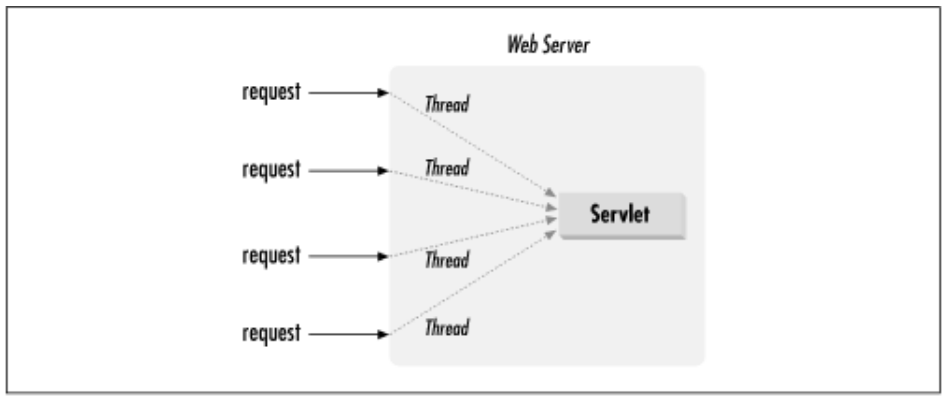
Condició de carrera



Condició de carrera + comptador compartit = problema de seguretat

Quan dos threads independents accedeixen i modifquen una mateixa dada, com un contador, la modificació ha de ser sincronitzada.

```
count++      // Thread 1
count++      // Thread 2
out.println  // Thread 1
out.println  // Thread 2
```



Exemple Contador No sincronitzat

CounterNoSincr.java

Sincronització

- Diferents threads poden córrer sobre els mateixos objectes.
- Cada objecte Java té associat un lock.
- El lock pot ser reclamat per qualsevol dels threads que corren en la màquina virtual.
- Si un thread t1 té el lock i un altre thread t2 el reclama, t2 es bloqueja esperant fins que t1 retorni el lock.

Sincronització explícita i implícita

Hi ha dues formes d'aconseguir el lock d'un objecte:

Explícitament Definint un bloc de codi sincronitzat:

Implícitament Cada vegada que invoquem un mètode de l'objecte que hagi estat definit com "synchronized", ens veurem obligats a obtenir el lock d'aquest mateix objecte abans de l'execució:

Sincronització explícita i implícita

```
//Explicit
public class Main{
    Persona p;
    public void method(){
        // codi no te lock
        synchronized (p) {
            //  codi te el lock de l'objecte p
        }
    }
}
//Implicit
public class Persona{
   synchronized void metode() {
      // codi te el lock de l'objecte this
   }
}
```

Exemple Contador sincronitzat

CounterSincr.java

Sincronitzant el contador. Alternativa 1

```
public synchronized void doGet(HttpServletRequest req, HttpServletResponse res) {
    PrintWriter out = res.getWriter();
     count++;
     out.println("Since loading, this servlet has been accessed " +
                 count + " times.");
}
```

Sincronitzant el contador. Alternativa 2

```
PrintWriter out = res.getWriter();
synchronized(this) {
  count++;
  out.println("Since loading, this servlet has been accessed " +
              count + " times.");
}
```

Sincronitzant el contador. Alternativa 3

```
PrintWriter out = res.getWriter();
int local_count;
synchronized(this) {
  local_count = ++count;
}
out.println("Since loading, this servlet has been accessed " +
            local_count + " times.");
```

Deadlock (problema de bloqueig)

- Dos o més threads estan en Deadlock quan tots els threads en aquest conjunt estan esperant un esdeveniment que només pot ser causat per un altre thread en el conjunt.
- Els esdeveniments als quals ens referim són concernents amb l'assignació i alliberament de recursos principalment.
- Una forma senzilla d'evitar Deadlocks és ordenar els recursos que desitgem obtenir de forma que tots els threads es sol.liciten en el mateix ordre.

Exemple Deadlock

DeadLock.java

Evitar els Deadlock

```
class T1 extends Thread {
  void run(){
  synchronized(obj1) {
    synchronized(obj2) {
      // Hacer algo con los dos
    }
  }
 }
}
class T2 extends Thread {
  void run(){
  synchronized(obj2) {
     synchronized(obj1) {
    // Hacer algo con los dos
     }
  }
}
class Main {
  void main(String []) {
    String obj1,obj2; T1 t1; T2 t2;
    t1.start();
    t2.start()
  }
}
```

Finalització de Threads

Els threads s'han d'acabar de forma natural, retornant del mètode run.
Un thread no pot finalitzar un altre thread.
Pot enviar-li una senyal d'interrupció, però és el thread que s'està executant qui ha d'atendre a aquesta senyal i fer-li cas.

Coordinació entre Threads

- Monitors en Java, mecanisme wait/notify
- Esperes entre processos en Java. Thread.join();
- Utilitats de concurrència de JAVA

Mecanisme wait/notify

- Java ens ofereix un mecanisme d'esperes i notificacions per facilitar la programació multithread.
- Cada objecte té associada una llista de notificació, en la qual poden col.locar els threads.
- Si un thread guanya el lock d'un objecte, pot decidir en qualsevol moment quedar-se esperant a la llista de notificació d'aquest objecte. A partir d'aquest moment el thread es trobarà en estat waiting i no ocuparà CPU fins que un altre thread li notifiqui que ha de despertar-se.
- Es necessita tenir el lock de l'objecte per fer una operació de wait o notify, per poder evitar situacions de carrera entre threads que fan wait i notify. Però és responsabilitat nostre garantir que les condicions que van propociar l'espera es mantenen. És a dir que no ócorre la notificació abans de l'espera.

Mecanisme wait/notify

Exemples utilitat:

- cues bloquejants. Quan estan buides o plenes s'esperen.
- protocols productors-consumidors.
- supressió d'esperes actives.

Wait / Notify

```
public final void wait() throws InterruptedException;
// Bloqueja el thread actual i l'envia a la llista d'espera de l'objecte sobre el que es fa.
public final void wait(long timeout) throws InterruptedException;
// Bloqueja el thread actual i l'envia a la llista d'espera de l'objecte sobre el que es fa
// amb un timeout per si ningu el desperta.

public final void notify()
//Desperta un dels threads (seleccionat aleatoriament) que esperen a la llista de notificacio de l'objec
public final void notifyAll()
//Desperta a tots els threads que esperen a la llista de notificacio de l'objecte.
```

UNIVERSITAT DE BARCELONA

Join de Threads

```
public final void join() throws InterruptedException
// Bloqueja el thread actual fins que el thread sobre el qual s'invoca el metode,
// hagi acabat (o el thread actual rebi un interrupt ()).

// Es creen i es posen en marxa uns quants fills
Thread hijos[NUM_HIJOS] = creaHijos();
// El pare espera que tots els fills acabin per continuar
for (int i = 0 ; i < NUM_HIJOS ; i++) {
hijos[i].join();
}
```

UNIVERSITAT DE BARCELONA

Utilitats de concurrència de JAVA

- Mecanismes de sincronització.
- Coleccions concurrents.
- Planificació i execució de tasques.

API java.util.concurrent