Lezione n.6
LPR A - INFORMATICA
THREADS: , SINCRONIZZAZIONE ESPLICITA

27/10/2008

Laura Ricci

THREADS COOPERANTI: IL MONITOR

- L'interazione esplicita tra threads avviene in un linguaggio ad oggetti come JAVA mediante l'utilizzo di oggetti condivisi
- Esempio: produttore/consumatore il produttore P produce un nuovo valore e lo comunica ad un thread consumatore C
- Il valore prodotto viene incapsulato in un oggetto condiviso da P e da C, ad esempio una coda che memorizza i messaggi scambiati tra P e C
- La mutua esclusione sull'oggetto condiviso è garantita dall'uso di metodi synchronized, ma...non è sufficiente garantire sincronizzazioni esplicite
- E' necessario introdurre costrutti per sospendere un thread T quando una condizione C non è verificata e per riattivare T quando diventa vera
- Esempio: il produttore si sospende se il buffer è pieno, si riattiva quando c'e' una posizione libera

THREAD COOPERANTI: IL MONITOR

- Monitor= Classe di oggetti utilizzabili da un insieme di threads
- Come ogni classe, il monitor contiene un insieme di campi ed un insieme di metodi
- La mutua esclusione può essere garantita dalla definizione di metodi synchronized. Un solo thread per volta si "all'interno del monitor"
- E' necessario inoltre
  - definire un insieme di condizioni sullo stato dell'oggetto condiviso
  - implementare meccanismi di sospensione/riattivazione dei threads sulla base del valore di queste condizioni
  - Implementazioni possibili:
    - definizione di variabili di condizione
    - metodi per la sospensione su queste variabili
    - definizione di code associate alle variabili in cui memorizzare i threads sospesi

THREADS COOPERANTI: IL MONITOR

JAVA

- Nella prime versioni non supporta le variabili di condizione
- assegna al programmatore il compito di gestire le condizioni mediante variabili del programma
- definisce meccanismi che consentono ad un thread
  - di sospendersi wait( ) in attesa che sia verificata una condizione
  - di segnalare notify( ) , notifyall ( ) ad un altro/ad altri threads sospesi che una certa condizione è verificata
- implementa per ogni oggetto condiviso O una coda in cui vengono memorizzati tutti i threads sospesi in attesa del verificarsi di una condizione sullo stato di O.
- Per ogni oggetto implementa due code:
  - una per i threads in attesa di acquisire la lock( )
  - per i threads in attesa del verificarsi di una condizione

THREADS COOPERANTI: IL MONITOR

- Produttore/Consumatore: due thread si scambiano dati attraverso un oggetto condiviso buffer

- Ogni thread deve acquisire la lock() sull'oggetto buffer, prima di inserire/ prelevare elementi

- Una volta acquisita la lock( )
  - il consumatore controlla se ci sono elementi nel buffer, ed eventualmente si sospende se il buffer è vuoto, altrimenti inserisce un elemento nel buffer ed eventualmente risveglia il consumatore.
  - il produttore controlla se c'è almeno una posizione libera nell buffer, in caso contrario si sospende. Quando inserisce un elemento dal buffer, controlla se vi sono eventuali consumatori in attesa

THREADS COOPERANTI: IL MONITOR

I METODI WAIT/NOTIFY

Metodi invocati sull'oggetto condiviso (se non compare il riferimento all'oggetto, l'oggetto implicito riferito è this)

- void wait( ) sospende il thread in attesa che sia verificata una condizione
- void wait (long timeout) sospende per al massimo timeout millisecondi
- void notify ( ) notifica ad un thread in attesa il verificarsi di una certa condizione su
- void notifyall ( ) notifica a tutti i threads in attesa il verificarsi di una condizione

tutti questi metodi

- fanno parte della classe Object (tutti le classi ereditano da Object,....)
- per invocare questi metodi occorre aver acquisito la lock ( ) sull'oggetto ⇒ devono essere invocati all'interno di un metodo o di un blocco sincronizzato

UN ESEMPIO: L'AUTOCARROZZERIA

- In un'autocarrozzeria si devono verniciare alcune auto.
- Per ogni auto si devono alternare un certo numero di fasi di ceratura con fasi di lucidatura
- Esistono due operai che sono specializzati, rispettivamente, nel processo di ceratura ed in quello di lucidatura
- Si deve simulare il comportamento dall'autocarrozzeria in JAVA
  - Si attivino due thread che simulino il comportamento dei due operai
  - Si definisce l'oggetto condiviso Auto, tramite cui interagiscono i due thread, che definisca i metodi per la corretta sincronizzazione tra i due thread
  - Il programma Autocarrozzeria deve creare un oggetto Auto, attivare i due thread, passando a ciascuno di essi l'oggetto condiviso, lasciarli lavorare per un certo numero di secondi, quindi interromperli

UN ESEMPIO: L'AUTOCARROZZERIA

```
public class Car {
    private boolean dacerare =  true;

    public synchronized void cerata ()
        {dacerare = false;
        notify( );
        }

    public synchronized void possocerare( ) throws InterruptedException{
            while (dacerare == false)
            try { wait( );} catch (InterruptedException e){ }
                }
```

UN ESEMPIO: L'AUTOCARROZZERIA

```
public synchronized void lucidata()
      { dacerare = true;
       notify( );
      }
public synchronized void possolucidare () throws InterruptedException{
      { while (dacerare==true)
            try { wait();} catch (InterruptedException e) {throw e;};
      } }
      }
```

UN ESEMPIO:L'AUTOCARROZZERIA

```
import java.util.concurrent.*;
public class Ceratura implements Runnable{
      private Car car;boolean vai=true;
      public Ceratura (Car c){ car = c;}
      public void run(){
      while (vai ) {try {
            System.out.println ("Ceratura!!");
            TimeUnit.MILLISECONDS.sleep(200);
            car.cerata();
            car.possocerare();
      } catch (InterruptedException e)
            {System.out.println("Fine del Thread Ceratura"); vai=false;}}}}
```

UN ESEMPIO: L'AUTOCARROZZERIA

```
import java.util.concurrent.*;
public class Lucidatura implements Runnable{
    private Car car;boolean vai=true;
    public Lucidatura (Car c) {car=c;};
    public void run( ) {
    while (vai) {
        try {car.possolucidare();
        System.out.println("Lucidatura!!!");
        TimeUnit.MILLISECONDS.sleep(200);
        car.lucidata(); }
        catch (InterruptedException e) {System.out.println ("Fine del
          Thread Lucidatura");vai=false;}  }
         } }
```

UN ESEMPIO:L'AUTOCARROZZERIA

```
import java.util.concurrent.*;
public class Autocarrozzeria {
    public static void main (String args[ ]) throws Exception
    {Car car = new Car();
    ExecutorService exec= Executors.newFixedThreadPool(2);
    exec.execute(new Ceratura(car));
    exec.execute(new Lucidatura(car));
    TimeUnit.SECONDS.sleep(3);
    exec.shutdownNow(); } }
```

L'AUTOCARROZZERIA:OUTPUT DEL PROGRAMMA

Ceratura!!

Lucidatura!!!

Ceratura!!

Lucidatura!!!

Ceratura!!

Lucidatura!!!

Ceratura!!

Lucidatura!!!

Ceratura!!

Lucidatura!!!

Fine del Thread Lucidatura

Fine del Thread Ceratura

WAIT E NOTIFY

wait

- rilascia la lock( ) sull'oggetto prima di sospendere il thread
- in seguito ad una notifica, può riacquisire la lock()

notify( )

- risveglia uno dei thread nella coda di attesa di quell'oggetto

notifyAll( )

- risveglia tutti i threads in attesa
- i thread risvegliati competono per l'acquisizione della lock( )
- i thread verranno eseguiti uno alla volta, quando riusciranno a riacquisire la lock( ) sull'oggetto

In tutti i casi la lock() è quella dell'oggetto acceduto con il metodo

synchronized

WAIT E NOTIFY

- Il metodo wait permette di attendere un cambiamento su una condizione "fuori dal monitor", in modo passivo
- Evita il controllo ripetuto di una condizione (polling)
- A differenza di sleep() e di yield() rilascia la lock sull'oggetto
- L'invocazione di un metodo wait, notify(), notifyall() fuori da un metodo synchronized solleva l'eccezione IllegalMonitorException( )
- Infatti prima di invocare questi metodi occorre aver acquisito la lock su un oggetto condiviso
- Nota Bene: nell'esempio avrei potuto usare un if nei metodi possolucidare( ) e possocerare( ). In generale è necessario ricontrollare la condizione, dopo che si è stati svegliati da una wait( ). Vedremo in seguito perchè questo è consigliabile nella maggior parte dei casi

CONFRONTO TRA NOTIFY E NOTIFYALL

Differenze tra notify( ) e notifyall( )

- notify( ) riattiva uno dei tasks associati alla coda associata all'oggetto su cui si invoca la notify( ).
- E' errato dire, genericamente: la notifyAll riattiva tutti i tasks in attesa.
- notifyAll( ) riattiva tutti i tasks in attesa su un oggetto, l'oggetto è quello su cui è invocata la notifyall( ) (this nel caso in cui l'oggetto non è esplicitamente riferito.

CONFRONTO NOTIFY E NOTIFYALL

```
public class sincronizza {

    // definizione dell'oggetto condiviso; per semplicità la omettiamo
    synchronized void attendi(  )throws InterruptedException{
    try{ //testa una condizione sull'oggetto; per semplicità la omettiamo
        wait( );
        System.out.println(Thread.currentThread()+" ");
        } catch(InterruptedException e){throw e;};}
    synchronized void risveglia( )      {notify();     }
    synchronized void risvegliatutti( )     {notifyAll();}
}
```

CONFRONTO TRA NOTIFY E NOTIFYALL

```
public class ThreadBlocco implements Runnable {
        sincronizza mys;boolean vai=true;
        public ThreadBlocco (sincronizza s)
                {mys=s;}
        public void run( ){
        while (vai) {
              try{
              mys.attendi( );
        }catch (InterruptedException e){System.out.println("sono interrotto"+
                                                Thread.currentThread( ));
           }
         }}}
```

CONFRONTO TRA NOTIFY E NOTIFYALL

```
import java.util.*;
import java.util.concurrent.*;
public class provanotify {
public static void main (String args[ ])  throws Exception
    {sincronizza mys1 = new sincronizza();
    sincronizza mys2 = new sincronizza();
    ExecutorService exec = Executors.newCachedThreadPool();
    for (int i=0; i<4; i++)
        exec.execute(new ThreadBlocco(mys1));
    exec.execute(new ThreadBlocco(mys2));
```

CONFRONTO TRA NOTIFY E NOTIFYALL

```
  boolean turno=true;
for (int i=0; i<4; i++)
{ if (turno){
            System.out.println("notify");
            mys1.risveglia();
            turno=false;
            Thread.sleep(500);}
       else { System.out.println("notifyall");
                mys1.risvegliatutti();
                turno=true;
               Thread.sleep(500);} }
System.out.println("ora risveglio l'ultimo thread");
mys2.risvegliatutti();   exec.shutdownNow( ); } }
```

CONFRONTO TRA NOTIFY E NOTIFYALL

notify

Thread[pool-1-thread-1,5,main]

notifyall

Thread[pool-1-thread-2,5,main]

Thread[pool-1-thread-3,5,main]

Thread[pool-1-thread-4,5,main]

Thread[pool-1-thread-1,5,main]

notify

Thread[pool-1-thread-2,5,main]

notifyall

Thread[pool-1-thread-4,5,main]

Thread[pool-1-thread-3,5,main]

Thread[pool-1-thread-1,5,main]

Thread[pool-1-thread-2,5,main]

CONFRONTO TRA NOTIFY E NOTIFYALL

ora risveglio l'ultimo task

Thread[pool-1-thread-5,5,main]

sono stato interrotto Thread[pool-1-thread-5,5,main]

sono stato interrotto Thread[pool-1-thread-4,5,main]

sono stato interrotto Thread[pool-1-thread-3,5,main]

sono stato interrotto Thread[pool-1-thread-2,5,main]

sono stato interrotto Thread[pool-1-thread-1,5,main]

- a tutti i threads viene inviata una interruzione dalla ShutdownNow(), mentre sono sospesi sulla wait
- L'interruzione viene intercettata come una InterruptedException()

ESERCIZIO

Il laboratorio di Informatica del Polo Marzotto è utilizzato da tre tipi di utenti, studenti, tesisti e professori ed ogni utente deve fare una richiesta al tutor per accedere al laboratorio. I computers del laboratorio sono numerati da 1 a 20. Le richieste di accesso sono diverse a seconda del tipo dell'utente:

a) i professori accedono in modo esclusivo a tutto il laboratorio, poichè hanno necessità di utilizzare tutti i computers per effettuare prove in rete.

b) i tesisti richiedono l'uso esclusivo di un solo computer, identificato dall'indice i, poichè su quel computer è istallato un particolare software necessario per lo sviluppo della tesi.

c) gli studenti richiedono l'uso esclusivo di un qualsiasi computer.

I professori hanno priorità su tutti nell'accesso al laboratorio, i tesisti hanno priorità sugli studenti. (prosegue nella pagina successiva)

ESERCIZIO

Scrivere un programma JAVA che simuli il comportamento degli utenti e del tutor. Il programma riceve in ingresso il numero di studenti, tesisti e professori che utilizzano il laboratorio ed attiva un thread per ogni utente. Ogni utente accede k volte al laboratorio, con k generato casualmente. Simulare l'intervallo di tempo che intercorre tra un accesso ed il successivo e l'intervallo di permanenza in laboratorio mediante il metodo sleep. Il tutor deve coordinare gli accessi al laboratorio. Il programma deve terminare quando tutti gli utenti hanno completato i loro accessi al laboratorio.