Programmazione Concorrente e Parallela - Serie 5

Maura Clerici

29.03.2019

Esercizio 1

Il programma eseguito con i lock espliciti impiega molto tempo a terminare. Per eliminarli, possiamo permetterci di utilizzare, come tecnica One writer many readers, una variabile volatile per la risorsa condivisa sharedValue, in quanto i thread eseguono operazioni di lettura e la variabile viene modificata unicamente da un'entità (il main). Il programma, dopo la modifica, risulta molto più veloce.

```
public class S5Esercizio1 {

final static AtomicBoolean isRunning = new AtomicBoolean(true);
static volatile int sharedValue = 0;
... }
```

Esercizio 2

Il problema che si presenta proviene dall'oggetto di EventListener, il quale durante la costruzione esegue il metodo registerListener, ma 'possibile che quest'operazione possa generare dei conflitti al momento della costruzione dell'oggetto.

Per risolvere questo problema è stato riscritto un nuovo costruttore per EventListener:

```
public EventListener(final int id) {
    this.id = id;
}
```

Nel main, all'interno del ciclo for che crea e registra il listener alla sorgente, è stato tolto:

```
allListeners.add(new EventListener(i, eventSource));
```

... ed è stato utilizzato:

```
    final EventListener listener = new EventListener(i);
    eventSource.registerListener(i, listener);
    allListeners.add(listener);
```

In questo modo, mantenendo separati i metodi per la costruzione dell'oggetto e la registrazione dello stesso, il programma non presenta più problemi.

Esercizio 3

Inizialmente, modificando il valore della variabile THREADSAFE_SHARE a true, il comportamento del programma non cambia. Il problema è che la variabile sharedState non è protetta, dichiarandola come variabile volatile, mantenendo THREADSAFE_SHARE uguale a true, il programma funziona, ma reimpostando THREADSAFE_SHARE a false il programma termina con evidenti problemi di sincronizzazione. Per questo, sono stati definiti dei metodi synchronized anche per la classe SharedState:

```
    final class SharedState implements IState {
    private int value = 0;

    @Override
    public synchronized void increment() {
        value++;
    }

    @Override
    public synchronized int getValue() {
        return value;
    }
}
```