Programmazione Concorrente e Parallela - Serie 9

Maura Clerici

12.06.2019

Esercizio 1

L'esercizio richiedeva di stampare più frequentemente i valori delle somme. I due threads partono contemporaneamente grazie ad un countdown, dopodiché il thread1 è responsabile dell'incremento del valore e la stampa delle somme, mentre il thread2 si occupa di calcolare le somme con un semplice incremento di una variabile. Per permettere un maggior numero di stampe dell'output, è stato lanciato il metodo yield() nel thread2, all'interno del ciclo che calcola le somme (un ciclo che incrementa di un'unità ad ogni giro). In questo modo, quando il thread esegue un giro, subito dopo viene "interrotto" e rischedulato, permettendo così al thread1 di stampare l'output più frequentemente. La soluzione:

```
final Thread thread2 = new Thread(() -> {
            cdl.countDown();
            try {
                cdl.await();
            } catch (final InterruptedException e) {
                e.printStackTrace();
            }
            for (int i = 1; i <= 50000; i++) {
                S9Esercizio1.sum = i;
                Thread.yield();
            }
            S9Esercizio1.finished = true;
            System.out.println("cnt " + S9Esercizio1.cnt);
        });
```

Nella seconda versione è stato richiesto di utilizzare le priorità dei threads. Senza modificare il comportamento dei threads, quindi, sono state definite delle priorità prima di farli partire. Al thread1 è stata assegnata una priorità maggiore, mentre al thread2 una priorità minore, utilizzando le costanti Thread.MAX_PRIORITY e Thread.MIN_PRIORITY, che corrispondono ad una priorità 10 e rispettivamente 1. In questo modo, avendo la priorità maggiore, il thread1 è in grado di stampare i risultati più frequentemente. La soluzione:

```
thread1.setPriority(Thread.MAX_PRIORITY);
thread2.setPriority(Thread.MIN_PRIORITY);
thread1.start();
thread2.start();
```

Comparando entrambe le versioni, l'utilizzo dello yield() permette di ritardare maggiormente il lavoro del thread2 permettendo di stampare molte più informazioni rispetto alla soluzione con la definizione delle priorità.

Esercizio 2

Il problema di questo esercizio è dovuto dal fatto che i depots scelti random non arrivano ordinati correttamente ai blocchi synchronized. Il programma terminerebbe correttamente se i depots scelti precedentemente fossero ordinati sempre nello stesso modo, ma essendo scelti random questa situazione non potrà sempre presentarsi e quindi il programma si blocca a causa di un deadlock. Il problema è stato risolto riordinando i depots scelti random prima dei blocchi synchronized. In questo modo il lock viene sempre acquisito nell'ordine corretto evitando il deadlock. La soluzione:

```
public void run() {
        final Random random = new Random();
        int failureCounter = 0;

        Depot supplier1 = null;
        Depot supplier2 = null;
        Depot supplier3 = null;

        while (true) {

            // Choose randomly 3 different suppliers
            final List<Depot> depots = new ArrayList<>();
            int[] ordered = new int[3];
            int orderCount = -1;

            while (depots.size() != 3) {
                int temp = random.nextInt(S9Esercizio2.suppliers.length);
                final Depot randomDepot = S9Esercizio2.suppliers[temp];
                if (!depots.contains(randomDepot)) {
                    depots.add(randomDepot);
                    ordered[++orderCount] = temp;
                }
            }

            // Riordinamento dei suppliers
            if (ordered[0] > ordered[1] && ordered[0] > ordered[2]) {
```

```
    supplier1 = depots.get(0);
    if (ordered[1] > ordered[2]) {
        supplier2 = depots.get(1);
        supplier3 = depots.get(2);
    } else {
        supplier2 = depots.get(2);
        supplier3 = depots.get(1);
    }
} else {
    if (ordered[1] > ordered[0] && ordered[1] > ordered[2]) {
        supplier1 = depots.get(1);
        if (ordered[0] > ordered[2]) {
            supplier2 = depots.get(0);
            supplier3 = depots.get(2);
        } else {
            supplier2 = depots.get(2);
            supplier3 = depots.get(0);
        }
    } else {
        supplier1 = depots.get(2);
        if (ordered[0] > ordered[1]) {
            supplier2 = depots.get(0);
            supplier3 = depots.get(1);
        } else {
            supplier2 = depots.get(1);
            supplier3 = depots.get(0);
        }
    }
}
...
```