Programmazione Concorrente e Parallela - Serie 7

Maura Clerici

11.04.2019

Esercizio 1

In questo esercizio, le istanze di TestWorker rappresentano i thread in esecuzione. Durante quest'ultima, essi accedono in lettura e scrittura ad una sharedMap e questo comporta problemi di race condition. La Map è stata definita come final, ma gli oggetti all'interno della HashMap possono comunque cambiare.

In una prima versione, è stato utilizzato un blocco synchronized:

```
synchronized (this) {
            if (counter == 0) {
                if (sharedMap.containsKey(key)
                        && sharedMap.get(key).equals(int1)) {
                    sharedMap.remove(key);
                    log("{" + key + "} remove 1");
                }
            } else if (counter == 1) {
                if (!sharedMap.containsKey(key)) {
                    sharedMap.put(key, int1);
                    log("{" + key + "} put 1");
                }
            } else if (counter == 5) {
                if (sharedMap.containsKey(key) && sharedMap.get(key).equals(10)) {
                    final Integer prev = sharedMap.put(key, int5);
                    log("{" + key + "} replace " + prev.intValue() + " with 5");
                }
            } else if (counter == 10) {
                if (sharedMap.containsKey(key)) {
                    final Integer prev = sharedMap.put(key, int10);
                    log("{" + key + "} replace " + prev.intValue() + " with 10");
                }
            }
        }
```

Nella seconda versione è stata definita la sharedMap come ConcurrentHashMap. È stato utilizzato l'idioma del CAS in quanto le operazioni eseguite sulla mappa sono operazioni sia di lettura che di scrittura e devono essere atomiche:

```
if (counter == 0) {
                sharedMap.remove(key, int1);
                log("{" + key + "} remove 1");
            } else if (counter == 1) {
                sharedMap.putIfAbsent(key, int1);
                log("{" + key + "} put 1");

            } else if (counter == 5) {
                if (sharedMap.replace(key, 10, int5)) {
                    final Integer prev = sharedMap.put(key, int5);
                    log("{" + key + "} replace " + prev.intValue() + " with 5");
                }
            } else if (counter == 10) {
                final Integer prev = sharedMap.replace(key, int10);
                if (prev != null) {
                    log("{" + key + "} replace " + prev.intValue() + " with 10");
                }
            }
```

Esercizio 2

Il problema di questo esercizio è che i thread, ReadWorker, accedono in lettura e scrittura ad una sharedPhrase, variabile statica condivisa. Anche all'interno del main vengono effettuati degli accessi alla stessa variabile, dopo che i thread sono partiti e prima del join.

In una prima versione sono stati usati dei synchronized block per risolvere il problema. All'interno del run():

```
    synchronized (S7Esercizio2.sharedPhrase) {
                // Build phrase string from shares words
                final Iterator<String> iterator = S7Esercizio2.sharedPhrase.iterator();
                while (iterator.hasNext()) {
                    sb.append(iterator.next());
                    sb.append(" ");
                }
            }
```

All'interno del main, dopo la partenza dei thread:

```
    for (int i = 0; i < 10; i++) {
```

```
            synchronized (sharedPhrase) {
                S7Esercizio2.sharedPhrase.add(getWord());
            }
            try {
                Thread.sleep(1000);
            } catch (final InterruptedException e) {
                e.printStackTrace();
            }
        }
```

In una seconda versione è stata utilizzata una synchronized collection. Nel main, la sharedPhrase è stata definita come una synchronizedList:

```
S7Esercizio2.sharedPhraseCollections.synchronizedList(list);
```

Sempre nel main, dopo la partenza dei thread l'accesso è protetto:

```
for (int i = 0; i < 10; i++) {
            S7Esercizio2.sharedPhrase.add(getWord());
            try {
                Thread.sleep(1000);
            } catch (final InterruptedException e) {
                e.printStackTrace();
            }
        }
```

All'interno del run l'accesso va invece protetto con:

```
synchronized (S7Esercizio2.sharedPhrase) {
                final Iterator<String> iterator = S7Esercizio2.sharedPhrase.iterator();
                while (iterator.hasNext()) {
                    sb.append(iterator.next());
                    sb.append(" ");
                }
            }
```

In una terza versione è stata usata, come concurrent collection, una CopyOnWriteArrayList. La sharedPhrase è stata definita:

```
static CopyOnWriteArrayList<String> sharedPhrase;
```

Non è stato necessario apportare ulteriori modifiche per proteggere i diversi accessi.

Eseguendo le tre diverse versioni, utilizzando i synchronized blocks e le synchronized collections il programma impiega circa 10 secondi, mentre utilizzando le concurrent collections impiega 11 secondi.