

Session: Concurrency and Parallelism in Java

.....

0. Hardware und Compiler:

Stalking the Lost Write: Memory Visibility in Concurrent Java

Jeff Berkowitz, New Relic

December 2013

(im gleichen Verzeichnis).

Schauen Sie sich die sog. „Executions“ der beiden Threads nochmal an. Erkennen Sie, dass es nur die zufällige Ablaufreihenfolge der Threads ist, die die verschiedenen Ergebnisse erzeugt? Das sind die berüchtigten „race conditions“...

Tun Sie mir den Gefallen und lassen Sie das Programm Mycounter in den se2_examples selber ein paar Mal ablaufen. Tritt die Race Condition auf? Wenn Sie den Lost Update noch nicht verstehen: Bitte einfach nochmal fragen!

Tipp: Wir haben ja gesehen, dass `i++` von zwei Threads gleichzeitig ausführt increments verliert (lost update). Wir haben eine Lösung gesehen in MyCounter: das Ganze in einer Methode verpacken und die Methode „synchronized“ machen. Also die „Ampellösung“ mit locks dahinter. Das ist teuer und macht unseren Gewinn durch mehr Threads kaputt.

Hm, könnte man `i++` ATOMAR updaten ohne Locks??

Suchen Sie mal nach „AtomicInteger“ im concurrent package von Java!

1. Arbeitet Ihr Gehirn concurrent oder parallel (-:-)?

It works parallel

2. Multi-threading hell: helfen „private, protected, public“ dagegen?

Ist Ihnen klar warum die OO-Kapselung durch „private“ Felder nicht hilft bei Multithreading?

The OO-Capsule doesn't help, since access modifiers are not important for threads. They have access without restrictions.

3. Muss der schreibende Zugriff von mehreren Threads auf folgende Datentypen synchronisiert werden?

- statische Variablen - yes

- Felder/Attribute in Objekten - yes
- Stackvariablen – no
- Heapvariablen (mit new() allokierte..) - no

4. Machen Sie die Klasse threadsafe (sicher bei gleichzeitigem Aufruf). Gibt's Probleme? Schauen Sie ganz besonders genau auf die letzte Methode getList() hin.

```
class Foo {
    private List aL = new CopyOnWriteArrayList();
    public int elements;
    private String firstName;
    private String lastName;

    public Foo () {};

    public void synchronized setfirstName(String fname) {
        firstName = fname;
        elements++;
    }
    public void synchronized setlastName(String lname) {
        lastName = lname;
        elements++;
    }
    public synchronized ArrayList getList () {
        return aL;
    }
}
```

5. kill_thread(thread_id) wurde verboten: Wieso ist das Töten eines Threads problematisch? Wie geht es richtig?

One receives the error that the method is deprecated. It is also unsafe for program. The right option would be to use the interrupt() method,.

6. Sie lassen eine Applikation auf einer Single-Core Maschine laufen und anschliessend auf einer Multi-Core Maschine. Wo läuft sie schneller?

Probably in a multi-core machine, since it would probably faster. It is, however, also important how to application is set to use the machine.

7. Was verursacht Non-determinismus bei multithreaded Programmen? (Sie wollen in einem Thread auf ein Konto 100 Euro einzahlen und in einem zweiten Thread den Kontostand verzehnfachen)

If the threads aren't ordered, one cannot know which one goes first.

8. Welches Problem könnte auftreten wenn ein Thread produce() und einer consume() aufruft? Wie sähen Lösungsmöglichkeiten aus? Wenn es mehr als einen Producer und/oder Consumer gibt? Wo könnte/sollte „volatile“ hin? Wo eventuell „synchronized“?

```
1 public class MyQueue {
2
3     private Object store;
4     int flag = false; // empty
5
6     public void produce(Object in) {
7         while (flag == true) ; //full
8         store = in;
9         flag = true; //full
10    }
11    public Object consume() {
12        Object ret;
13        while (flag == false) ; //empty
14        ret = store;
15        flag = false; //empty
16        return ret;
17    }
18 }
```

The while-loops are rigid. —> synchronised

1. 9. Fun with Threads:
2. 1. Was passiert mit dem Programm?
3. 2. Was kann auf der Konsole stehen?

```
public class C1 {
public synchronized void doX (C2 c2) {
    c2.doX();
}
public synchronized void doY () {
    System.out.println("Doing Y");
}

public static void main (String[] args) {
    C1 c1 = new C1();
}
```

```
C2 c2 = new C2();  
Thread t1 = new Thread() -> { while (true) c1.doX(c2);});  
Thread t2 = new Thread() -> { while (true) c2.doY(c1);});  
t1.start();  
t2.start();  
}  
};
```

```
public class C2 {  
  
    public synchronized void doX () {  
        System.out.println("Doing X");  
    }  
    public synchronized void doY ( C1 c1) {  
        c1.doY();  
    }  
}
```

1. Deadlock
2. Doing Y and Doing X are printed out, one after the other