

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 ausgeführt 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.

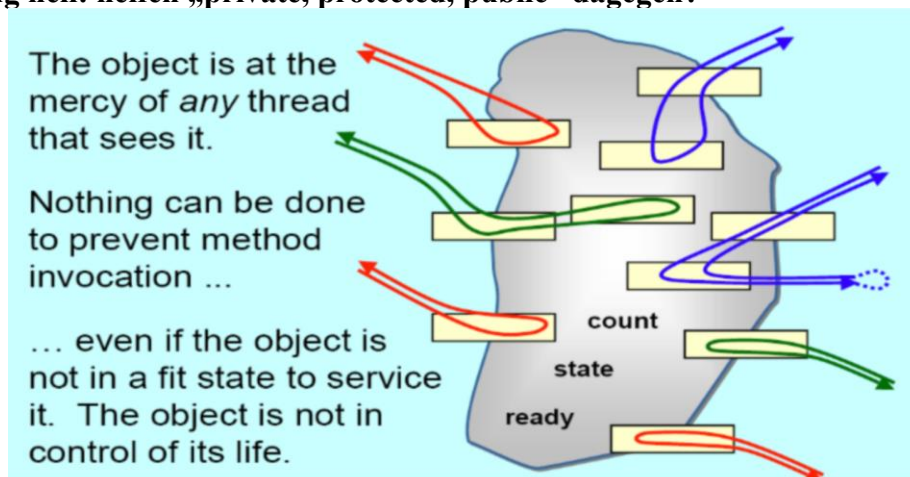
Hm, könnte man `i++` ATOMAR updaten ohne Locks??

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

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

Das Gehirn macht eine Mischung aus den beiden.

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



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

Die Membervariablen helfen nicht, da selbst bei private keine neue Klasse beim Multithreading erstellt wird und somit auch private variablen von mehreren threads verändert werden können.

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

- statische Variablen Ja
- Felder/Attribute in Objekten Ja
- Stackvariablen Ja
- Heapvariablen (mit new() allokierte..) Nein

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 ArrayList aL = new ArrayList();  
    public int elements;  
    private String firstName;  
    private String lastName;  
  
    public Foo () {};  
  
    public void synchronized setfirstName(String fname) {  
        firstName = fname;  
        elements++;  
    }  
    public 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?

Beim Threadkill kann ich nicht sichergehen in welchem Moment der Thread gestoppt wird. Dadurch können Fehler beim setzen von variablen geben. Entweder man interrupted den thread oder man hat eine boolean in der while schleife, welcher auf false gesetzt wird durch eine „requeststop()“ Methode.

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

Wenn das Programm mit mehreren Kernen arbeiten kann, ist es natürlich schneller mit einer Multi-Core Maschine.

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)

Es kann sein das der zweite Thread als erstes Ausgeführt wird und dadurch der Kontostand verzehnfacht wird bevor die 100 Euro addiert wurde. Dadurch würde jetzt ein 1000€ Verlust entstehen.

8. Welches Problem könnte auftreten wenn ein Thread produce() und einer consume() aufruft? Wie sähen Lösungsmöglichkeiten aus?

```
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 }
```

9. Fun with Threads:

1. Was passiert mit dem Programm?
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 class C2 {  
  
  public synchronized void doX () {  
    System.out.println("Doing X");  
  }  
  public synchronized void doY ( C1 c1) {  
    c1.doY();  
  }  
}  
  
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();  
}  
};
```

Das Programm erstellt erstmal ein C1 und C2 Objekt. Dann werden 2 Threads erstellt die jeweils eine Methode aufrufen. Da aber alles synchronized ist, wird nur der erste Thread laufen, da dort eine while(true) schleife ist. Der Konsolenoutput ist die ganze Zeit „Doing X“