The Law of the Distracted Spearfisherman

www.javaspecialists.eu/archive/Issue147.html

by Dr. Heinz M. Kabutz

Abstract:
Learn how to write correct concurrent code by understanding the Secrets of Concurrency. This is the second part of a series of laws that help explain how we should be writing concurrent code in Java. We look at how to debug a concurrent program by knowing what every thread in the system is doing.

Welcome to the 147th issue of

The Java(tm) Specialists' Newsletter. A few weeks ago, my laptop monitor gave up its ghost, after approximately 30000 hours of operation. I phoned Dell, full of fear that they might not have a service department on the Island of Crete. To my delight, the next morning, the parts arrived via courier and in the early evening, their engineer came to my house and swapped out the monitor.

The new laptop monitor is brighter than the original one, allowing me to sit in the shade at the beach and write this newsletter, while our children are splashing in the sea at Marathi Beach, with the fishing boats and the Lefka Ori Mountains in the background. Picture perfect. So thanks Dell, you made my day again :-)

NEW: Please see our new "Extreme Java" course, combining concurrency, a little bit of performance and Java 8. Extreme Java - Concurrency & Performance for Java 8.

The Law of the Distracted Spearfisherman

In my previous newsletter, I introduced the ten laws of concurrency that can help you make sense of some rather tricky coding. Just to remind you, here they are again. In this newsletter, we will look at the Law of the Distracted Spearfisherman.

The Law of the Distracted Spearfisherman

Focus on one thread at a time. The school of threads will blind you.

One of my hobbies as a teenager was catching fish with a speargun. Rather than sit on the rocks with a line and a baited hook, we would jump into the waves, where we could be more selective as to what exactly we wanted to catch. As I got older, the thought of being eaten by a great white shark became less appealing than the thrill of catching my own food.

The best defence for a fish is to swim right next to a bigger, better, fish. Something I learned from trying to shoot fish was that you need to stay focused, otherwise you end up with nothing. Once you zone in on a particular specimen, you need to commit to catching that one. If you let your eye wander to the next bigger fish, by the time you pull the trigger, you are too late and the fish are gone.

Let's apply this to Java Concurrency. You need to understand what *every* thread is doing in your system. Maybe you should try that now - create a thread dump of all of your threads and try to figure out what they are all doing.

Most of us know how to generate thread dumps by now, but here it is again for those who do not. The simplest way is to go to the console window that is running your Java process and then press CTRL+Break on Windows and CTRL+\ on Unix. In Unix you can also invoke kill -3 on the Java process, which dumps the thread dump to the console that is running the Java program.

If you do not have access to the console window or if you are running the process without a console, you have several other options for finding out what the threads are doing.

Since Java 5, we have a tool called jstack that ships as part of the JDK and which we can use to generate a thread dump. We can attach this to an existing running Java process to see what is going on. At the talk in Barcelona, someone mentioned that jstack does not show the deadlocks, but only the states of the threads, so I decided to create a deadlock in order to try this out.

Here is a class that should cause a deadlock after some time:

```
public class BadClass extends Thread {
 private final Object lock1;
 private final Object lock2;
 public BadClass(Object lock1, Object lock2)
{
 this.lock1 = lock1;
 this.lock2 = lock2;
 }
 public void run() {
 while(true) {
 synchronized(lock1) {
 synchronized(lock2) {
 System.out.print('.');
 System.out.flush();
 }
 }
 }
 }
 public static void main(String[] args) {
 Object lock1 = new Object();
 Object lock2 = new Object();
 BadClass bc1 = new BadClass(lock1, lock2);
 BadClass bc2 = new BadClass(lock2, lock1);
 bc1.start();
 bc2.start();
 }
}
```

When we run this in Java 5, we can see that the output of jstack is not that useful. The state of the threads is shown as BLOCKED, whereas some of them are actually in the WAITING state. It does not show the name of the threads.

```
JVM version is 1.5.0_11-b03
Thread 21068: (state = BLOCKED)
Thread 21077: (state = BLOCKED)
 - BadClass.run() @bci=13, line=13 (Compiled frame)
Thread 21076: (state = BLOCKED)
 - BadClass.run() @bci=13, line=13 (Compiled frame)
Thread 21072: (state = BLOCKED)
Thread 21071: (state = BLOCKED)
 - java.lang.Object.wait(long) @bci=0 (Interpreted frame)
 - java.lang.ref.ReferenceQueue.remove(long) @bci=44, line=116
 - java.lang.ref.ReferenceQueue.remove() @bci=2, line=132
 - Finalizer$FinalizerThread.run() @bci=3, line=159
Thread 21070: (state = BLOCKED)
 - java.lang.Object.wait(long) @bci=0 (Interpreted frame)
 - java.lang.Object.wait() @bci=2, line=474 (Interpreted
frame)
 - Reference$ReferenceHandler.run() @bci=46, line=116
```

In Java 6, they have improved jstack to also show detected deadlocks. You can now see exactly what is going on.

```
Full thread dump Java HotSpot(TM) Client VM (1.6.0_01-b06):
"Attach Listener" daemon prio=10 tid=0x08078400 nid=0x60ee
 runnable [0x00000000..0x00000000]
 java.lang.Thread.State: RUNNABLE
"DestroyJavaVM" prio=10 tid=0x08058400 nid=0x6084
 waiting on condition [0x00000000..0xb7d90080]
 java.lang.Thread.State: RUNNABLE
"Thread-1" prio=10 tid=0x080a2800 nid=0x608d
 waiting for monitor entry [0xb5842000..0xb5842fb0]
 java.lang.Thread.State: BLOCKED (on object monitor)
 at BadClass.run(BadClass.java:13)
 - waiting to lock <0x8c509ee0> (a java.lang.Object)
 - locked <0x8c509ee8> (a java.lang.Object)
"Thread-0" prio=10 tid=0x080a1400 nid=0x608c
 waiting for monitor entry [0xb5893000..0xb5893f30]
 java.lang.Thread.State: BLOCKED (on object monitor)
 at BadClass.run(BadClass.java:13)
 - waiting to lock <0x8c509ee8> (a java.lang.Object)
 - locked <0x8c509ee0> (a java.lang.Object)
"Low Memory Detector" daemon prio=10 tid=0x0808d400
nid=0x608a
 runnable [0x00000000..0x00000000]
 java.lang.Thread.State: RUNNABLE
"CompilerThread0" daemon prio=10 tid=0x0808bc00 nid=0x6089
 waiting on condition [0x00000000..0xb59e6838]
 java.lang.Thread.State: RUNNABLE
"Signal Dispatcher" daemon prio=10 tid=0x0808a800 nid=0x6088
 runnable [0x00000000..0xb5a37e10]
 java.lang.Thread.State: RUNNABLE
"Finalizer" daemon prio=10 tid=0x08081800 nid=0x6087
 in Object.wait() [0xb5aca000..0xb5acb0b0]
 java.lang.Thread.State: WAITING (on object monitor)
```

```
 at java.lang.Object.wait(Native Method)
 - waiting on <0x8c50a128> (a ReferenceQueue$Lock)
 at java.lang.ref.ReferenceQueue.remove
 - locked <0x8c50a128> (a ReferenceQueue$Lock)
 at java.lang.ref.ReferenceQueue.remove
 at java.lang.ref.Finalizer$FinalizerThread.run
"Reference Handler" daemon prio=10 tid=0x08080400 nid=0x6086
 in Object.wait() [0xb5b1b000..0xb5b1c030]
 java.lang.Thread.State: WAITING (on object monitor)
 at java.lang.Object.wait(Native Method)
 - waiting on <0x8c509eb8> (a Reference$Lock)
 at java.lang.Object.wait(Object.java:485)
 at java.lang.ref.Reference$ReferenceHandler.run
 - locked <0x8c509eb8> (a Reference$Lock)
"VM Thread" prio=10 tid=0x08077000 nid=0x6085 runnable
"VM Periodic Task Thread" prio=10 tid=0x0808f000 nid=0x608b
 waiting on condition
JNI global references: 623
Found one Java-level deadlock:
=============================
"Thread-1":
 waiting to lock monitor 0x080829d8
 (object 0x8c509ee0, a java.lang.Object),
 which is held by "Thread-0"
"Thread-0":
 waiting to lock monitor 0x08082974
 (object 0x8c509ee8, a java.lang.Object),
 which is held by "Thread-1"
Java stack information for the threads listed above:
===================================================
"Thread-1":
 at BadClass.run(BadClass.java:13)
 - waiting to lock <0x8c509ee0> (a java.lang.Object)
 - locked <0x8c509ee8> (a java.lang.Object)
"Thread-0":
 at BadClass.run(BadClass.java:13)
 - waiting to lock <0x8c509ee8> (a java.lang.Object)
 - locked <0x8c509ee0> (a java.lang.Object)
Found 1 deadlock.
```

Now take some time to go over that thread dump again and make sure that you know exactly what every thread is doing. Don't go to the next one until you have figured out the one you are looking at. Remember the Law of the Distracted Spearfisherman! If you jump around between threads, without knowing exactly what each one is doing, you will end up with nothing at all!

Javva the Hutt tried out the advice in this newsletter and found lots of threads that did not need to be there. In his case, the quantity of threads was not an issue, but he concedes that it was a good exercise to know what is going on in the application. Read the article here.

One of the challenges of using tools like jstack and CTRL+Break is that the threads come back unsorted. This makes it difficult to compare two thread dumps of a live system, a technique that can help to see the progress of thread snapshots.

In newsletter 132, I show how to generate nicely sorted thread dumps using JSP. You can include this on your server in an administrator page so that you can compare thread dumps taken at different times. Since the threads are sorted by name, you also have a better grouping of the

threads, making it much easier to understand.

The JSP thread dumper does not show deadlocks either, but for that you can use my Deadlock Detector newsletter.

You might think that you have too many threads in your system and that you do not need to know what they are all up to. That is asking for trouble if you are writing multi-threaded code. A deadlock cannot be resolved in Java since you cannot force another thread to release its locks.

So remember the Law of the Distracted Spearfisherman. Zone in one thread at a time when you are trying to debug the system and make sure that you understand what it is doing before going any further. *The best defence for a fish is to swim next to a bigger, better fish!*

Kind regards from Marathi Beach :-)

Heinz

Concurrency Articles Related Java Course