CS-E4110 Concurrent Programming
Autumn 2016 - Tutorials
Race Detection

2016-10-31

In the tutorial answers to the tasks below will be discussed. Additionally the usage of race detection tools will be demonstrated.

Task 1

Examine the attached Java program which has a data race. Answer the following questions:

- Where is the data race?
- Would a happens-before based race detection tool find the race in all execution traces? If not provide a counterexample.
- Is the program guaranteed to terminate?

```
1  public class PollStop {
2      static int count = 0;
3      static boolean stop = false;
4
5      public static void main(String[] args) throws InterruptedException {
6          a.start();
7          while(true) {
8              synchronized(a) {
9                  if (count > 100) {
10                     stop = true;
11                     break;
12                 }
13             }
14         }
15     }
16     static Thread a = new Thread() {
17         public void run() {
18             while (!stop) {
19                 synchronized(this) {
20                     ++count;
21                     System.out.print(count + "␣");
22                 }
23             }
24             System.out.println();
25         }};
26 }
```

Attachment: PollStop.java

Task 2

Examine the attached Java program. Answer the following questions:

- Is the program data race free (DRF)?
- If so, would a lockset based race detection tool verify this in all execution traces? If not provide a counterexample.

```
1  public class MessagePass {
2      static class Message {
3          String question;
4          int answer;
5      }
6
7      static boolean passed = false;
8      static Message msg;
9
10     public static void main(String[] args) throws InterruptedException {
11         a.start(); b.start();
12     }
13     static Thread a = new Thread() {
14         public void run() {
15             msg = new Message();
16             msg.question = "Six␣by␣nine?";
17             msg.answer = 42;
18             synchronized (MessagePass.class) {
19                 passed = true;
20             }
21         }};
22     static Thread b = new Thread() {
23         public void run() {
24             while (true) {
25                 synchronized (MessagePass.class) {
26                     if (passed) break;
27                 }
28             }
29             System.out.println(msg.question + "␣" + msg.answer);
30         }};
31 }
```

Attachment: MessagePass.java

Task 3

Examine the attached Java program.

Hint: If you wish to test different behaviors of the program you can edit dividend to larger or smaller values before running it. The value 50000 happened to produce both results roughly equally on the assistant's computer.

Answer the following questions:

- Is the program data race free (DRF)?
- If so, would a lockset based race detection tool verify this in all execution traces? If not provide a counterexample.

```
1  public class Division {
2      static int dividend, divisor, result;
3
4      public static void main(String[] args) throws InterruptedException {
5          dividend = 50000;
6          divisor = 7;
7          a.start();
8          Thread.sleep(1);
9          int remainder;
10         synchronized (a) {
11             remainder = dividend;
12         }
13         if (remainder < divisor) {
14             System.out.println(result);
15         } else {
16             System.out.println("No result.");
17         }
18     }
19     static Thread a = new Thread() {
20         public void run() {
21             while (true) {
22                 result += 1;
23                 synchronized (this) {
24                     dividend -= divisor;
25                     if (dividend < divisor)
26                         break;
27                 }
28             }
29         }};
30 }
```

Attachment: Division.java

Task 4

Examine the attached program. Answer the following questions:

- Is the program data race free (DRF)?
- If so, would a lockset based race detection tool verify this in all execution traces? If not provide a counterexample.

```
1  public class Staggered {
2      static int x;
3
4      public static void main(String[] args) throws InterruptedException {
5          a.start();
6          b.start();
7          c.start();
8          a.join();
9          b.join();
10         c.join();
11         System.out.println(x);
12     }
13     static Thread a = new Thread() {
14         public void run() {
15             synchronized (b) {
16                 synchronized (c) {
17                     x = 1;
18                 }
19             }
20         }};
21     static Thread b = new Thread() {
22         public void run() {
23             synchronized (a) {
24                 synchronized (c) {
25                     x = 2;
26                 }
27             }
28         }};
29     static Thread c = new Thread() {
30         public void run() {
31             synchronized (a) {
32                 synchronized (b) {
33                     x = 3;
34                 }
35             }
36         }};
37 }
```

Attachment: Staggered.java

Task 5

Examine the attached program. Answer the following questions:

- Is the program data race free (DRF)?
- What is the program's intent? In other words, describe on a high level what the program does.
- Is the program safe? If not, what could go wrong?

```
1  import java.util.Random;
2  public class OptimisticBuffer {
3      // Implementation
4      public final int capacity = 2048;
5      private long[] data = new long[capacity];
6      private int begin, end;
7      public synchronized int getSize() {
8          return end − begin;
9      }
10     public synchronized void put(long x) {
11         data[end++] = x;
12     }
13     public synchronized long get() {
14         return data[begin++];
15     }
16
17     // Usage
18     static OptimisticBuffer buffer = new OptimisticBuffer();;
19     public static void main(String[] args) throws InterruptedException {
20         new Thread(producer).start();
21         new Thread(producer).start();
22         new Thread(consumer).start();
23         Thread.sleep(100);
24         new Thread(consumer).start();
25     }
26     static Runnable producer = new Runnable() {
27         public void run() {
28             Random rand = new Random();
29             for (int i = 0; i < buffer.capacity/2; i++) {
30                 buffer.put(rand.nextLong());
31             }
32         }};
33     static Runnable consumer = new Runnable() {
34         public void run() {
35             while (buffer.getSize() > 0) {
36                 long x = buffer.get();
37                 // use x...
38             }
39         }};
40 }
```

Attachment: OptimisticBuffer.java