CS-E4110 Concurrent Programming
Autumn 2016 - Tutorials
Java Programming with Threads

2016-09-21

Task 1

Write a short program which launches two threads, ThreadA and ThreadB. ThreadA must print the string "`Hello `" to `System.out` and ThreadB must print the string "`world!`" to `System.out`.

What are the possible outputs of the program?

Download links
PrintHelloWorld.java Program skeleton

Task 2

Your task is to upgrade the program `SerialArrayProcessing` to use threads to speed up data processing.

```
1  import java.util.Arrays;
2
3  public class SerialArrayProcessing {
4
5      public static void main(String[] args) {
6          final long timestamp = System.currentTimeMillis();
7
8          //The initial data
9          final int[] data = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
10
11
12         //Iterate over the array
13         for(int i = 0; i < data.length; i++) {
14             data[i] = verySlowFunction(data[i]);
15         }
16
17         //Print the modified data
18         System.out.println(Arrays.toString(data));
19         System.out.println("Time (ms): " + (System.currentTimeMillis()-timestamp));
20     }
21
22     public static int verySlowFunction(int argument) {
23         //Let's pretend that this will take a long time...
24         try {
25             Thread.sleep((long)(Math.random() * 1000));
26         } catch (InterruptedException e) { }
27
28         return argument * argument;
29     }
30
31 }
```

Write an improved version, called `ParallelArrayProcessing`. Launch a single thread per array element. Each thread must read an array element, give the

element to the `verySlowFunction` as an argument and store the return value back in the source cell.

Download links

SerialArrayProcessing.java	Original program which needs an upgrade
ParallelArrayProcessing.java	Program skeleton

Task 3

Write a concurrently correct implementation for the following interface:

```
1  public interface AtomicInt {
2
3      /**
4       * Atomically increments the current value by one and
5       * return the new value.
6       * @return the updated value
7       */
8      int incrementAndGet();
9
10     /**
11      * Atomically set the current value of the AtomicInt.
12      * @param newValue
13      * @return the updated value
14      */
15     int set(int newValue);
16
17     /**
18      * Return the current value of the AtomicInt.
19      * @return current value
20      */
21     int get();
22 }
```

Download links

AtomicInt.java	Required interface

Task 4

Read the Java Language Specification (JLS) chapter 17: `https://docs.oracle.com/javase/specs/jls/se8/html/jls-17.html`

Use the definitions from the JLS to explain why your implementation of `AtomicInt` from Task 3 is correct.

Task 5

Study the given Java implementation of the Manna-Pnueli critical section algorithm. The program will run in an infinite loop unless it detects the condition "more than one thread in the critical section". Modify the implementation according to the following instructions. Perform each modification separately.

Does the algorithm still work? If not, explain how and why the changes break the program.

1. Remove the keyword `volatile` from the declaration of `inCS`
2. Remove the `synchronized` -block from around the `if-else` statements on lines 17-20 and 40-43
3. Change both `synchronized(lock)` statements to `synchronized(this)`

```
1  /* Manna-Pnueli mutual exclusion algorithm */
2  public class MannaPnueli {
3      //Number of processes currently in critical section
4      static volatile int inCS = 0;
5      //Process p wants to enter critical section
6      static volatile int wantp = 0;
7      //Process q wants to enter critical section
8      static volatile int wantq = 0;
9      //Object used as a mutual exclusion lock
10     static final Object lock = new Object();
11
12     static class P extends Thread {
13         @Override
14         public void run() {
15             while (true) {
16                 synchronized(lock) {
17                     if (wantq == -1)
18                         wantp = -1;
19                     else
20                         wantp = 1;
21                 }
22                 while (wantq == wantp)
23                     Thread.yield();
24                 inCS++;
25                 Thread.yield();
26                 //Critical section
27                 if (inCS != 1)
28                     System.exit(1);
29                 inCS--;
30                 wantp = 0;
31             }
32         }
33     }
34
35     static class Q extends Thread {
```

```
36          @Override
37          public void run() {
38              while (true) {
39                  synchronized(lock) {
40                      if (wantp == −1)
41                          wantq = 1;
42                      else
43                          wantq = −1;
44                  }
45                  while (wantp == −wantq)
46                      Thread.yield();
47                  inCS++;
48                  Thread.yield();
49                  //Critical section
50                  if (inCS != 1)
51                      System.exit(1);
52                  inCS−−;
53                  wantq = 0;
54              }
55          }
56      }
57
58      public static void main(String[] args) {
59          Thread threadP = new P();
60          Thread threadQ = new Q();
61          threadP.start();
62          threadQ.start();
63      }
64  }
```

Download links

MannaPnueli.java Java implementation of Manna-Pnueli