CO2017 — Concurrency Surgery, 2017–18

Dr Gilbert Laycock (gtl1)

2018–02–04; R1103

Questions

This example taken more-or-less verbatim from the 2016 exam paper.

Interleaving

Suppose there are two threads P and Q. Each consists of sequential atomic steps as shown:

P: $p_1; p_2; p_3; p_4$
Q: $q_1; q_2;$

1. Calculate how many ***interleavings*** are possible if threads P and Q are executed concurrently.
2. Give an example of an ***valid*** interleaving of threads P and Q.
3. Give an example of an ***invalid*** interleaving of threads P and Q.

Race conditions, failure to enforce mutual exclusion

Consider the (outline) code in Figure 1.

The code is supposed to be a very simple simulation of a seat block booking service. There are 1000 seats initially available for an event, controlled by the `Bookings` class. Blocks of seats can be `booked`, or `unbooked`.

Four threads, using classes `MakeBookings` and `CancelBookings`, will concurrently make and cancel bookings.

The `Test` class simply initialises the system and starts the threads; it displays the number of available seats at regular intervals.

You may ignore the possibility of `Exceptions` for this question.

1. In the context of this example, identify the ***shared resource*** and briefly explain what this means.
2. This example does not enforce ***mutual exclusion***. Give an example of the kind of undesirable output that might result from this, and suggest how you would modify the code to prevent it happening.

```
1  public class Bookings {
2    private int available;
3    public Bookings(int initial) { available = initial; }
4    public int getAvailable()     { return available;     }
5    public void book(int n)     { available=available-n; }
6    public void unbook(int n)     { available=available+n; }
7  }
8
9  public class MakeBookings extends Thread {
10   private Bookings bk;
11   public MakeBookings(Bookings b) { bk=b; }
12   public void run() {
13     for (int i=20; i<1100; i+=400) {
14       bk.book(i);
15       Thread.sleep(1000);
16     }
17   }
18 }
19
20 public class CancelBookings extends Thread {
21   private Bookings bk;
22   public CancelBookings(Bookings b) { bk=b; }
23   public void run() {
24     for (int i=1000; i>0; i-=100) {
25       bk.unbook(i);
26       Thread.sleep(1500);
27     }
28   }
29 }
30
31 public class Test {
32   public static void main(String[] args) {
33     Bookings          b1 = new Bookings(1000);
34     MakeBookings    mb1 = new MakeBookings(b1);
35     MakeBookings    mb2 = new MakeBookings(b1);
36     CancelBookings cb1 = new CancelBookings(b1);
37     CancelBookings cb2 = new CancelBookings(b1);
38     mb1.start(); mb2.start();
39     cb1.start(); cb2.start();
40     while (true) {
41       System.out.println("Available: "+b1.getAvailable());
42       Thread.sleep(2000);
43     }
44   }
45 }
```

Figure 1: Outline simple bulk seat booking service

3. This example does not enforce the implicit requirements on the `Bookings` class that:

   (a) there can never be more than the initial number of seats available; and

   (b) there can never be fewer than zero seats available.

   Give an example of the kind of undesirable output that might result because these implicit requirements are not enforced.

   In addition explain how you would modify the system, using ***monitors*** and ***guards*** so that users of the `Bookings` class are forced to ***wait*** if they make a request that would violate the requirements.