Concurrency:
State Models & Design Patterns

Practical Session

Week 02

Assignment 01

Discussion

A01 - Exercise 1

a) Do recent central processing units (CPUs) of desktop PCs support concurrency? Why became concurrency for many software applications very important these days?

Yes, they do. In today’s connected world, large-scale data analysis requesting huge compute clusters exploiting massive concurrency became very popular in machine learning or trend prediction.

b) Why do we need synchronization mechanisms in concurrent programs?

Synchronization is needed to ensure safety and liveness in concurrent programs.

A01 - Exercise 1

c) What is safety? Give one concrete example of a safety violation.

Safety is a property of concurrent programs and it claims that the consistency of resources is ensured, i.e., data cannot be corrupted by concurrent processes. For example, a safety violation occurs when two bank transactions happen at the very same time, and the bank account ends up with the wrong balance.

d) What is liveness? Give a concrete example of a liveness violation.

Liveness is a property of concurrent programs and states that processes are eventually (i.e., guaranteed at some time in the future) able to do useful work. A liveness violation occurs, if someone makes a bank transaction and the transaction never finishes.

A01 - Exercise 1

e) Can a binary semaphore lead to a deadlock? Why? Can it lead to starvation? Why?

The binary semaphore cannot lead to a deadlock, because it does not allow circular dependencies. However, depending on the implementation, it can lead to starvation due to processes that are not receiving any signallings.

f) How do monitors differ from semaphores? Please provide a precise answer.

They differ in terms of flexibility, usage, and programming style. Semaphores can allow several concurrent processes to enter the mutual exclusion (mutex, also known as “critical section”), whereas with monitors it is just one.

A01 - Exercise 1

g) How are monitors and message passing similar? And how are they different?

Monitors and message passing are similar in the sense that they both encapsulate data and operations. They are different in the sense that monitors rely on shared data and processes need to sync using locks, whereas message passing uses no shared state and messages are used to achieve synchronization.

A01 - Exercise 2

```
x := 1
Thread 1 -> x := x + 7.
Thread 2 -> x := x * 5.
```

Considering the code above: Give all possible values of x at the end of the execution of both threads together with their corresponding execution traces.

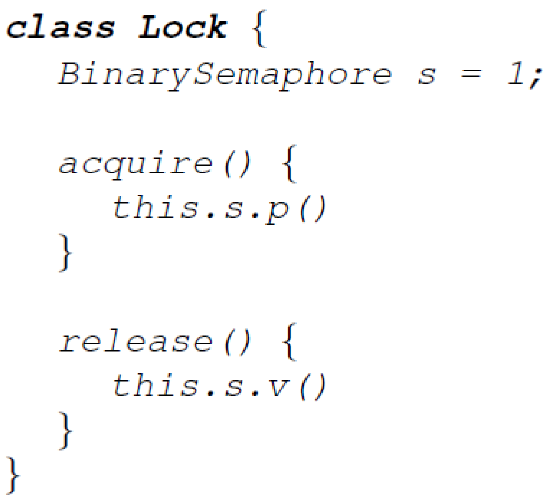
```
r1(x = 1), w1(x = 8), r2(x = 8), w2(x = 40)   --> 40
r1(x = 1), r2(x = 1), w1(x = 8), w2(x = 5)    --> 5
r1(x = 1), r2(x = 1), w2(x = 5), w1(x = 8)    --> 8
r2(x = 1), r1(x = 1), w2(x = 5), w1(x = 8)    --> 8
r2(x = 1), r1(x = 1), w1(x = 8), w2(x = 5)    --> 5
r2(x = 1), w2(x = 5), r1(x = 5), w1(x = 12)   --> 12
```

A01 - Exercise 3

Implement a monitor using a binary semaphore. Use pseudo-code and comment it.

The general strategy is to implement a lock based on a binary semaphore and then implement a monitor which includes that lock.

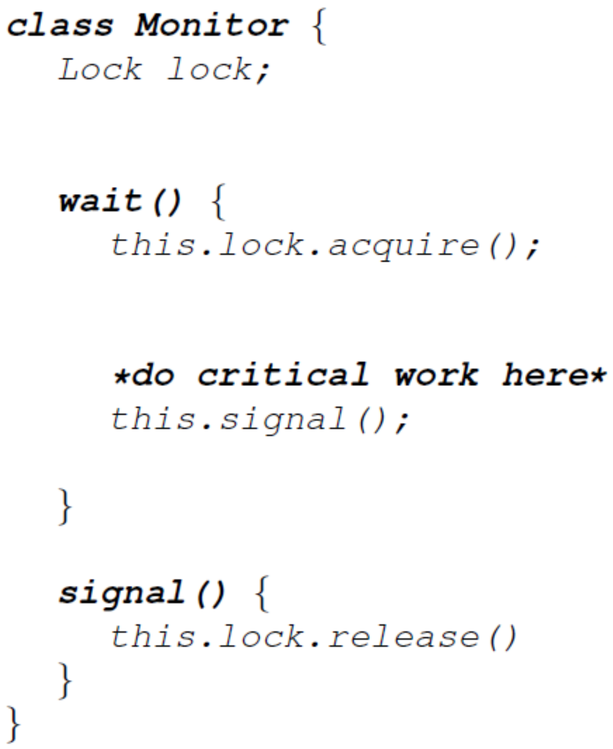
(...)



```
class Lock {
   BinarySemaphore s = 1;

   acquire() {
      this.s.p()
   }

   release() {
      this.s.v()
   }
}
```



```
class Monitor {
   Lock lock;


   wait() {
      this.lock.acquire();


      *do critical work here*
      this.signal();

   }

   signal() {
      this.lock.release()
   }
}
```

Assignment 02

Preview

A02 - Exercise 1 (2 pts)

Answer the following questions:

a) What states can a Java thread be in?

b) How can you turn a Java class into a monitor?

c) What is the Runnable interface good for?

d) Specify an FSP that repeatedly performs hello, but may stop at any time.

A02 - Exercise 2 (2 pts)

Consider the following Java implementation of a Singleton within a single-threaded application:

```
public class Singleton {
        private static Singleton instance = null;

        private Singleton() {}

        public static Singleton getInstance() {
                if(instance == null) {
                        instance = new Singleton();
                }
                return instance;
        }
}
```

a) What happens if the application is multithreaded?

b) How to implement a thread-safe singleton in Java?

c) Suppose there arrive 1000 requests/second from different threads at this Singleton. Does your implementation introduce a bottleneck? If yes, how can you improve it?

A02 - Exercise 3 (3.5 pts)

Download LTSA from http://www.doc.ic.ac.uk/˜jnm/book/ltsa/download.html. For each of the following processes shown in Figure 1, provide the Finite State Process (FSP) description of the corresponding Labeled Transition System (LTS) graph. You may verify the FSP descriptions by generating the state machines and using the “draw” functionality of the tool.

A02 - Exercise 4 (2.5 pts)

Consider the full Race5K FSP from the lecture.

a) **How many states and how many possible traces does it have if the number of steps is 5 (as in the lecture)?**

b) **What is the number of states and traces in the general case (i.e. for n steps)? We assume 2 processes.**

c) **Check your solution using the LTSA tool.**

*You have to <u>attend the lecture</u> to reveal such slides.**

:-)

**Disclaimer:*

The content that has been shown on this slide is irrelevant for the exam.