Concurrency:
State Models & Design Patterns

Practical Session

Week 03

Exercises 02

Discussion

Exercise 02 - Task 1

a) What states can a Java thread be in?

There exist six states in the Java runtime environment: NEW, RUNNABLE, BLOCKED, WAITING, TIMED_WAITING and TERMINATED.

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

It is as simple as declaring all public methods synchronized.

Exercise 02 - Task 1

c) What is the Runnable interface good for?

You can create a running thread based on a Runnable object. This is very useful since Java up to release 7 does not support the concept of multiple inheritance, so inheriting from the Thread class is not always an available option.

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

HELLO = (hello -> HELLO | hello -> STOP).

Exercise 02 - Task 2

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?

There exists the possibility that many instances of the Singleton class are created, potentially overwriting previously assigned variable values.

Exercise 02 - Task 2

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

The `getInstance()` method needs to be synchronized as shown below:

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

Exercise 02 - Task 2

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?

Yes, a more efficient and fine-grained solution is as follows ():

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

Exercise 02 - Task 3

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.

```
APPOINTMENT = (hello -> converse -> goodbye -> STOP).

TREBLE = (in[i:1..3] -> out[i*3] -> TREBLE).

FIVETICK (N=5) = FIVETICK[1],
FIVETICK[i:1..N] = ( when(i<N) tick -> FIVETICK[i+1]
                   | when(i==N) tick -> STOP).

...
```

Exercise 02 - Task 4

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)?**

   36 states (cartesian product of individual state spaces) fostering 252 traces (see next question)

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

   - Number of states: $(n + 1)^2$
   - Number of traces: (2*n)! / (n!*n!), from n steps, respectively transitions

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

   :-)

Exercises 03

Preview

Exercise 03 - Task 1

Answer the following questions:

a) What are safety properties? How are they modeled in FSP?

b) Is the busy-wait mutex protocol fair? Deadlock-free? Justify your answer.

c) Can you ensure safety in concurrent programs without using locks?

d) The Java language designers decided to implement concurrency based on monitors. What is the main reason behind this decision? What other options except monitors could have been chosen?

Exercise 03 - Task 2

Consider the following process definitions. Are T1 and T2 equivalent? Why?

```
R = (a->c->R).
S = (b->c->S).
||T1 = (R || S).
T2 = (a->b->c->T2|b->a->c->T2).
```

Exercise 03 - Task 3

A miniature portable FM radio has three controls. An on/off switch turns the device on and off. Tuning is controlled by two buttons, scan and reset, which operate as follows. (...)

Exercise 03 - Task 4

Consider the following problem specification:

a) We have a stack that has two slots.

b) The stack operations pop and push should be atomic. In other words, the stack is locked while being popped or pushed.

c) The input space is 0..1 (You can only push 0s and 1s).

d) There is one producer process that pushes 0s and 1s repeatedly.

e) There is one consumer that pops the top of the stack.

f) Throw an error for pushing to a full stack or popping from an empty stack.

Write an FSP description for this problem and verify your model using LTSA.