Solution Concurrent Programming - Exam

Given name: ____________________

Family name: ____________________

Matriculation Number: ____________________

Please read this page carefully and fill in your name right away!

Examination date:	Wednesday, December 20, 2017
Version:	1.2
Time:	60 minutes
Total points:	60 (you have about one minute per point)
Number of exercises:	5
Allowed materials:	this paper and a pen

Some exercises are split over more than one page. Please quickly skim through all exercises first, before you proceed with the exam.

If you don't know something don't waste any time, just go ahead and try to solve those questions at the end.

You are expected to answer each question concisely (brief and precise).

Exercise 1 (approx. 18 minutes)

Answer the following questions:

a) What is the difference between a sequential and a concurrent Java program? What are the similarities? What is the benefit of using concurrency? **(3 points)** **Answer:**

Difference: sequential programs use one thread of control while concurrent programs can concurrently use many threads of control.
Similarities: the initial statements (i.e., very first parts of the main method) are executed sequentially for both program types.
Benefit: parallelizable computations can benefit from massive speed improvements.

b) Which two methods do you know for creating threads in Java. What is the main benefit of one against the other? **(3 points)** **Answer:**

Extending the Thread class *and implementing the* Runnable interface. *Thread is simpler to use, but doesn't allow inheritance of other classes. Consequently, in most cases it is necessary to implement the Runnable interface.*

c) List three different synchronization techniques and summarize them in one sentence each. **(3 points)** **Answer:**

Busy-waiting: processes atomically test and set shared variables.
Semaphores: a semaphore can be illustrated as an urn that waits when no balls (permits) are left and it supports the two operations P(s) that puts balls into the urn and V(s) that takes balls from it.
Message Passing: message passing combines communication and synchronization, i.e., the sender specifies the message and a destination, and the receiver specifies message variables and a source.
Monitors: a monitor encapsulates resources together with operations that manipulate them and its operations are invoked like ordinary procedure calls.
Path Expressions: path expressions express the allowable sequence of operations as a kind of

regular expression.
Remote Procedure Calls: calls to another system which ensures synchronization itself.

d) What is a critical section? What consequences may arise if it is not properly handled? How can we properly handle critical sections? **(3 points)** **Answer:**

A "critical section" in a concurrent program is any portion of code that attempts to access a shared resource. Safety and liveness issues may arise. Synchronization mechanisms are required to properly handle critical sections.

e) List three of the six thread states a thread can be assigned to in Java (no further explanation necessary for any of the three states). **(3 points)** **Answer:**

NEW, RUNNABLE, BLOCKED, WAITING, TIMED_WAITING, TERMINATED

f) Does it make sense to use asynchronous communication patterns for a production-grade web servers client connection management? Justify your choice *and* provide a brief example how you would design the connection management. **(3 points)** **Answer:**

Yes, of course! Servers that have to handle many concurrent connections simultaneously benefit very much from additional CPU resources. Each connection to a client could be handled by a distinct thread.

Exercise 2 (approx. 9 minutes)

Suppose we have a boat with five seats that will sink in case more than five people get on board. Complete the related safety property below that effectively prevents this problem. Remember: When no travellers are on board and still one traveller tries to exit, it should lead into the ERROR state. **(9 points)**

```
property BOATCAPACITY = BOAT[0],
BOAT[i:0..5] = (
   // ToDo: implement safety conditions




).
```

Answer:

```
property BOATCAPACITY = BOAT[0],
BOAT[i:0..5] = (
    enter               -> BOAT[i+1] |
    when(i>0)   exit    -> BOAT[i-1] |
    when(i==0)  exit    -> ERROR
).
```

Exercise 3 (approx. 12 minutes)

Consider the FSP model below and answer the questions. The corresponding composed LTS graph is shown in Figure 1.

```
const N = 3
range T = 0..N
set VarAlpha = {read[T],write[T]}

Var      = Var[0],
Var[u:T] = ( read[u]    ->Var[u]
           | write[v:T]->Var[v]).

Inc = (read[v:0..N-1] -> write[v+1] -> STOP)+VarAlpha.
||ParInc = ({a,b}::Var || a:Inc || b:Inc).
```

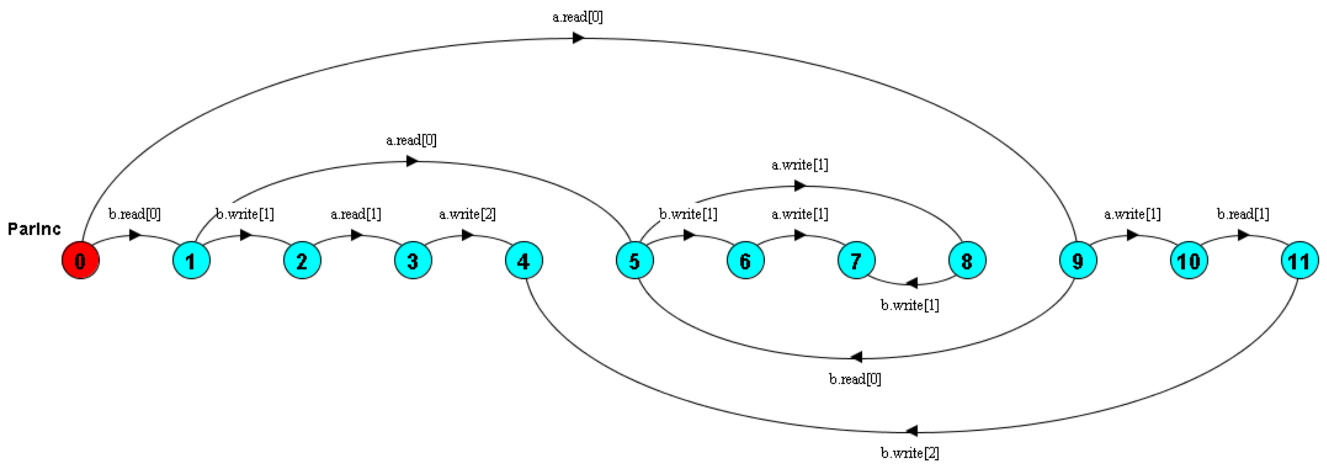

Figure 1: LTS graph for parallel increment

You will find the questions on the next page.

a) List and briefly explain two out of the four necessary and sufficient conditions for deadlock occurrence. **(4 points)** **Answer:**

Serially reusable resources: the deadlocked processes share resources under mutual exclusion.
Incremental acquisition: processes hold on to acquired resources while waiting to obtain additional ones.
No pre-emption: once acquired by a process, resources cannot be pre-empted but only released voluntarily.
Wait-for cycle: a cycle of processes exists in which each process holds a resource which its successor in the cycle is waiting to acquire.

b) Explain the characteristics of deadlock states in LTS graphs. In other words, what is an indicator for deadlocking states in an LTS graph? Why? **(2 points)** **Answer:**

Deadlocking states in the graph have no outgoing edges. Once arrived in such a state there is no way out: a deadlock just happened.

c) Does the system suffer from liveness issues (i.e. deadlocks)? If yes, in which states? **(3 points)** **Answer:**

Yes, it does. States 4 and 7 are affected.

d) Does the system suffer from safety issues (i.e. corruptions)? If yes, in which states? **(3 points)**
<u>**Answer:**</u>

Yes, it does. For states 4 and 7 both threads write concurrently at the same resource.

Exercise 4 (approx. 15 minutes)

Please answer the following questions regarding the Petri net in figure 2:

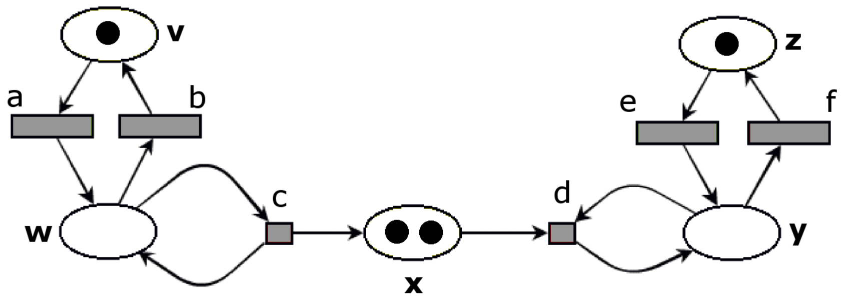

Figure 2: Auto-firing Petri net

a) Complete the definition (i.e., input functions, output functions, marking) of the Petri net. **(3 points)**

P = {v, w, x, y, z}

T = {a, b, c, d, e, f}

I(v) = {__________}, **O(v)** = {__________}

I(w) = {__________}, **O(w)** = {__________}

I(x) = {__________}, **O(x)** = {__________}

I(y) = {__________}, **O(y)** = {__________}

I(z) = {__________}, **O(z)** = {__________}

m = {__________}

Answer:

$I(v) = \{b\}$, $O(v) = \{a\}$

$I(w) = \{a, c\}$, $O(w) = \{b, c\}$

$I(x) = \{c\}$, $O(x) = \{d\}$

$I(y) = \{d, e\}$, $O(y) = \{d, f\}$

$I(z) = \{f\}$, $O(z) = \{e\}$

$m = \{v, x, x, z\}$

b) Is the Petri net bounded? Why? **(3 points)** **Answer:**

No, transition c is able to produce an infinite amount of tokens that will be forwarded into place x.

c) Is the Petri net safe? Why? **(3 points)** **Answer:**

No, it has already more than one token in place x in the initial marking.

d) Is the Petri net conservative? Why? **(3 points)** **Answer:**

No, it's not conservative because the number of tokens is not constant (see answer a).

e) Are all the transitions live in the Petri net? Why? **(3 points)** **Answer:**

All transition are live because the left part (transition c) is able to continuously produce tokens which keep all the transitions alive. No deadlocks can occur.

Exercise 5 (approx. 6 minutes)

Consider the code in listing 1 and answer the following questions:

1. Which asynchronous invocation pattern do you recognize? How does it briefly work? **(2 points)** **Answer:**

   Futures: It requests the result from the server, but instead of the final result it receives a reference to a value object in which the server will store later the delayed result.

2. What is the main advantage of using this pattern in comparison to an implementation based on *Direct Invocations*? **(2 points)** **Answer:**

   The client can still progress with his own execution even though the server has not yet finished the requested calculations.

3. How many *hours* does the client wait for the result of the server before he aborts? **(2 points)** **Answer:**

   48 hours

```java
protected static void initiateCalculations(final Server server, final int n){
  new Thread() {
    public void run() {
      try {
        System.out.println("CALLING fibonacci(" + n + ")");
        FutureTask<Integer> future = server.service(n);
        System.out.println("GOT future(fibonacci(" + n + "))");
        int val = future.get(2, TimeUnit.DAYS).intValue();
```

```
            System.out.println("GOT fibonacci(" + n + ") = " + val);
          } catch(InterruptedException e) {
          } catch(ExecutionException e) {
          } catch (TimeoutException e) {
          }
        }
      }.start();
    }
```

Listing 1: Sample Java code that can be executed concurrently