DUBLIN CITY UNIVERSITY

AUGUST/RESIT EXAMINATIONS 2017/2018

MODULE: CA4006 - Concurrent and Distributed Programming

PROGRAMME(S):

CASE	BSc in Computer Applications (Sft.Eng.)
CPSSD	BSc in ComputationalProblem Solv&SW Dev.
ECSAO	Study Abroad (Engineering & Computing)
ECSA	Study Abroad (Engineering & Computing)

YEAR OF STUDY: 4,O,X

EXAMINER(S):

Martin Crane	(x8974)
Prof. Brendan Tangney	(External)
Dr. Hitesh Tewari	(External)

TIME ALLOWED: 3 Hours

INSTRUCTIONS: **Answer 4 questions. All questions carry equal marks.**

PLEASE DO NOT TURN OVER THIS PAGE UNTIL YOU ARE INSTRUCTED TO DO SO.
The use of programmable or text storing calculators is expressly forbidden.
Please note that where a candidate answers more than the required number of questions, the examiner will mark all questions attempted and then select the highest scoring ones.

Requirements for this paper:
1. Log Tables

QUESTION 1 ***[TOTAL MARKS: 25]***

Q 1(a) **[12 Marks]**

Briefly describe Dekker’s Two-Process Mutual Exclusion Algorithm in words and implement it in C for two processors. Explain clearly all parts of the code.

Q 1(b) **[13 Marks]**

Peterson’s algorithm is a variation on Dekker’s algorithm whereby each processor uses two variables, `flag` and `turn`. A `flag` value of 1 indicates that the process wants to enter the critical section. The variable `turn` holds the ID of the process whose turn it is. Entrance to the critical section is granted for process P0 if P1 does not want to enter its critical section or if P1 has given priority to P0 by setting turn to 0. Implement Peterson’s algorithm in Java and briefly compare it with Dekker’s algorithm.

[End of Question1]

QUESTION 2 ***[TOTAL MARKS: 25]***

Q 2(a) **[4 Marks]**

Document fully what is meant by (i) Safety and (ii) Liveness properties.

Q 2(b) **[15 Marks]**

Briefly describe the Bakery algorithm to solve the *n* process mutual exclusion problem and implement it in C for two processors. Explain clearly all parts of the code.

Q 2(c) **[6 Marks]**

Give 2 reasons why the Bakery algorithm is not an efficient solution to the *n* process mutual exclusion problem.

[End of Question2]

QUESTION 3 ***[TOTAL MARKS: 25]***

Q 3(a) **[8 Marks]**

As part of the `java.util.concurrent` package what are `Lock` objects and how do they differ from intrinsic locks? What are `ReentrantLocks` and in what situations would one use them over `synchronized` blocks of code?

Q 3(b) **[17 Marks]**

For the code given in Figure Q3,

[7 marks]

(i) What is the meaning of the **`lock.tryLock()`** method? Explain clearly what is happening at points **A, B**. What will happen if either/both statements fail?

[3 marks]

(ii) Explain clearly what is happening at point **C**.

[7 marks]

(iii) Explain clearly what is meant by the Code at point **D**, describing briefly the possible problem that is being avoided.

```
import java.util.*;
import java.util.concurrent.*;
import java.util.concurrent.locks.*;
import static java.util.concurrent.TimeUnit.NANOSECONDS;

public class BankTransfer {
    private static Random rnd = new Random();

    public boolean transferMoney(Account fromAcct,
                                 Account toAcct,
                                 DollarAmount amount,
                                 long timeout,
                                 TimeUnit unit)
            throws InsufficientFundsException, InterruptedException {
        long fixedDelay = getFixedDelayComponentNanos(timeout, unit);
        long randMod = getRandomDelayModulusNanos(timeout, unit);
        long stopTime = System.nanoTime() + unit.toNanos(timeout);

        while (true) {
    A───▶   if (fromAcct.lock.tryLock()) {
                try {
            B───▶   if (toAcct.lock.tryLock()) {
                        try {
                            if (fromAcct.getBalance().compareTo(amount) < 0)
                                throw new InsufficientFundsException();
                            else {
              C────▶  {         fromAcct.debit(amount);
                                toAcct.credit(amount);
                                return true;
                            }
                        } finally {
                            toAcct.lock.unlock();
                        }
                    }
                } finally {
                    fromAcct.lock.unlock();
                }
            }
          ⎧ if (System.nanoTime() > stopTime)
 D ────▶ ⎨     return false;
          ⎩ NANOSECONDS.sleep(fixedDelay + rnd.nextLong() % randMod);
        }
    }

    private static final int DELAY_FIXED = 1;
    private static final int DELAY_RANDOM = 2;

    static long getFixedDelayComponentNanos(long timeout, TimeUnit unit) {
        return DELAY_FIXED;
    }

    static long getRandomDelayModulusNanos(long timeout, TimeUnit unit) {
        return DELAY_RANDOM;
    }

    static class DollarAmount implements Comparable<DollarAmount> {
        public int compareTo(DollarAmount other) {
            return 0;
        }

        DollarAmount(int dollars) {
        }
    }


    class Account {
        public Lock lock;

        void debit(DollarAmount d) {
        }

        void credit(DollarAmount d) {
        }

        DollarAmount getBalance() {
            return null;
        }
    }

    class InsufficientFundsException extends Exception {
    }
                                       }
```

Figure Q3

[End of Question3]

QUESTION 4 ***[TOTAL MARKS: 25]***

Q 4(a) **[9 Marks]**

In the context of Distributed Mutual Exclusion, describe the following using a diagram: A simple, centralized algorithm and a distributed token ring algorithm. Compare these algorithms with respect to the number of messages which must be exchanged and the delay before entry (in terms of message times). What is the main problem with each?

Q 4(b) **[7 Marks]**

Clock Synchronization algorithms are essential for Mutual Exclusive access to shared resources in Distributed Systems. Figure Q4b shows three processes, each with its own clock, each of which runs at different rates. Explain, using a diagram, the operation of Lamport's Algorithm for synchronizing the logical clocks and ensuring partial causal ordering of the messages sent from process to process in Figure Q4b. What is the principal problem with Lamport's algorithm?

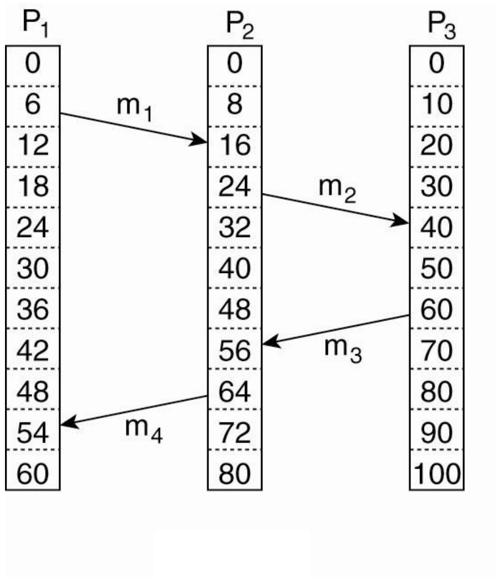

Figure Q4b.

Q 4(c) **[9 Marks]**

In many distributed algorithms, it is necessary for one process to take the role of coordinator or initiator. Dynamically choosing such a process typically happens through an election process. Figure Q4c shows a distributed system comprising eight processes. The previous coordinator of the distributed system was process 7 which has now crashed. Using the example shown in Figure 4c, demonstrate the operation of the Bully Algorithm. You may assume that the processes are labelled according to their priority. Would such an algorithm scale up to large scale systems?

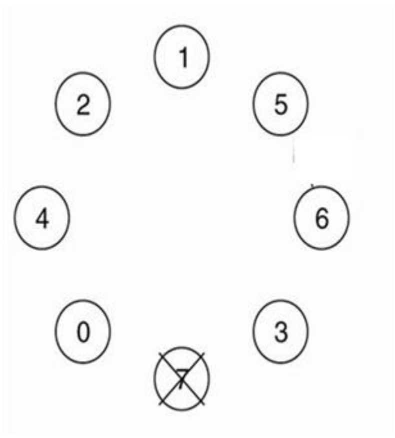

Figure Q4c

[End of Question4]

QUESTION 5 ***[TOTAL MARKS: 25]***

Q 5(a) **[10 Marks]**

Name and briefly describe the three approaches used to define Java Artifacts in SOAP web services, indicating, giving reasons, under which conditions each approach is preferable.

Q 5(b) **[15 Marks]**

Part of a Java Interface `LongTerm.java` to return the product and sum of two `long`s is shown in Figure Q5. Using Java Web Services, give the implementation of the following components of this interface: the Service Endpoint Interface (SEI), the Service Implementation Bean (SIB) and the Endpoint Publisher. You should fully comment your code.

```
public long mult(in long a, in long b);
  // implementation omitted
   };

public long add(in long a, in long b);
  // implementation omitted
   };
```

Figure Q5 Interface `LongTerm.java`

[End of Question5]

[END OF EXAM]