DUBLIN CITY UNIVERSITY

SEMESTER ONE RESIT EXAMINATIONS 2012

MODULE TITLE:	Concurrent Programming
MODULE CODE:	CA463/CA463D
COURSE:	BSc. in Computer Applications (Software Engineering Stream), Study Abroad
YEAR:	4/X
EXAMINERS:	Dr. J. Power Dr. F. Bannister Dr. Martin Crane Ext: 8974
TIME ALLOWED:	2 Hours
INSTRUCTIONS:	Please answer **any 3** questions:

Requirements for this paper
Please tick (X) as appropriate

- ☐ ***Log Table***
- ☐ ***Graph Paper***
- ☐ ***Attached Answer Sheet***
- ☐ ***Statistical Tables***
- ☐ ***Floppy Disk***
- ☐ ***Actuarial Tables***

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.

PLEASE DO NOT TURN OVER THIS PAGE UNTIL YOU ARE INSTRUCTED TO DO SO

Question 1 **[Total marks: 33]**

1(a) [12 marks]

Define Dekker's Algorithm in words and implement it in SR for two processors. Explain clearly all parts of the code.

1(b)

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.

[11 marks]

(i) Implement Peterson's algorithm in Java and briefly compare it with Dekker's algorithm.

[10 marks]

(ii) Show that, in Peterson's algorithm in (i), (a) Mutual Exclusion is satisfied and (b) No Starvation occurs.

--[End of Question 1]--

Question 2 **[Total marks: 33]**

2(a) [10 marks]

Show, using SR code, how semaphores may be used to implement monitors and how monitors may be used to implement semaphores.

2(b) [11 marks]

Write code in Java to implement a `Semaphore` class. Your implementation should give a default constructor, a constructor which initialises with value `i`, a `release` method (for the `P` operation) and an `acquire` method (for the `V` operation).

2(c) [12 marks]

Using the `Semaphore` class in (b) above, implement the Bounded Buffer Class in Java. The methods `void deposit(Object value)` and `Object fetch()` should be implemented.

--[End of Question 2]—

Question 3 **[Total marks: 33]**

3(a) [23 marks]

Write code for the Reader-Preference Solution to the Readers-Writers Problem in SR. Explain clearly all parts of the code. Explain briefly (without code) the method of "Passing the Baton". What mechanisms does SR provide that make the implementation of "Passing the Baton" possible?

3(b) [10 marks]

Implement in SR and describe fully Ballhausen's solution to the Readers-Writers Problem. Discuss the efficiency of this solution vis-à-vis the Readers-Preference solution in part (a) above.

--[End of Question 3]—

Question 4 **[Total marks: 33]**

4(a) [9 marks]

Explain the difference between Low Level Concurrency Objects and High Level Concurrency Objects in Java. Why would you use the latter rather than the former?

4(b) [10 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?

4(c) [14 marks]

For the code given in Figure Q4,

[6 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?

[2 marks]

(ii) Explain clearly what is meant by the Code at point **C**.

[6 marks]

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

Figure Q4

```
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) {
            if (fromAcct.lock.tryLock()) {
                try {
                    if (toAcct.lock.tryLock()) {
                        try {
                            if (fromAcct.getBalance().compareTo(amount) < 0)
                                throw new InsufficientFundsException();
                            else {
                                fromAcct.debit(amount);
                                toAcct.credit(amount);
                                return true;
                            }
                        } finally {
                            toAcct.lock.unlock();
                        }
                    }
                } finally {
                    fromAcct.lock.unlock();
                }
            }
            if (System.nanoTime() < stopTime)
                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) {
        }
    }
```

A

B

C

D

```
    class Account {
        public Lock lock;

        void debit(DollarAmount d) {
        }

        void credit(DollarAmount d) {
        }

        DollarAmount getBalance() {
            return null;
        }
    }

    class InsufficientFundsException extends Exception {
    }
}
```

--[End of Question 4]—

Question 5 **[Total marks: 33]**

5(a) [10 marks]

Sketch Cassavant and Kuhl's taxonomy of load balancing algorithms. Define clearly the heading for each section. State clearly the difference between:

1. Algorithms with One-time Assignment and Dynamic Reassignment.
2. Adaptive and non-Adaptive Algorithms.

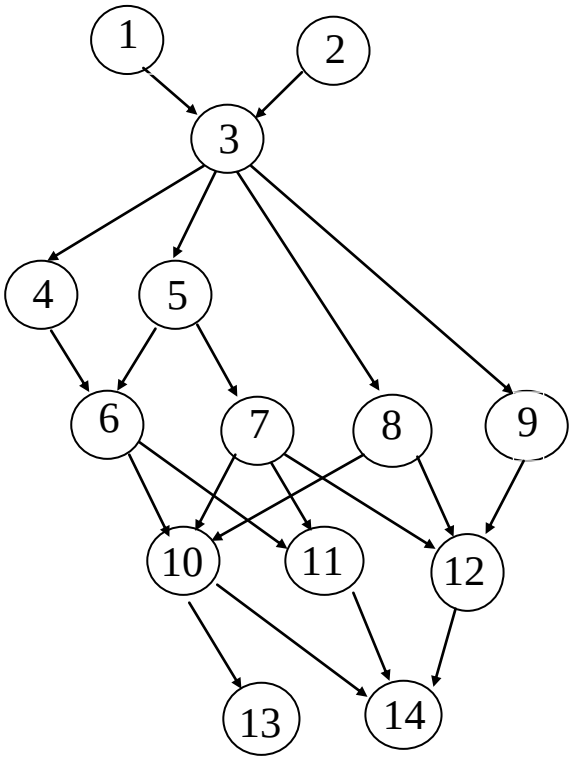
5(b) [12 marks]

Describe Coffman's load balancing algorithm. How would you classify this algorithm? What assumptions is it based on?

5(c) [11 marks]

Use Coffman's algorithm to schedule the task graph in Figure Q5 on to a four processor system.

Figure Q5



--[End of Question 5]—