CONCURRENT AND DISTRIBUTED SYSTEMS

Examples Sheet

This course is split into two halves. The first eight lectures appear in Michaelmas Term, and the second eight lectures appear in the Lent Term. The Course Lecturer recommends that a total of four supervisions are offered for this course, two for each half.

Relevant exam questions for this course can be found in many different locations, including: *Concurrent Systems*, *Concurrent Systems and Applications*, *Concurrent and Distributed Systems* and *Operating Systems Foundations*. Not all of these questions cover topics which are part of the current syllabus. In particular, the Concurrent Systems and Applications course also contains a lot of Java-related material which is now taught separately in the *Further Java* course.

A list of relevant past exam questions is listed in Appendix A.

Part One

Supervision 1

Q1. What's the difference between a process and a thread in an operating system?

Q2. Describe three types of concurrency which can occur on a single CPU.

Q3. Why is writing a concurrent program hard?

Q4. In the user-level threading model, why does pre-emption by the operating system (e.g. due to a system call) necessarily halt the execution of all threads?

Q5. The precise error demonstrated in Slide 17 of the notes can be solved by adding the following additional check:

```
beer = checkFridge();
if (!beer) {
 if (!note) {
  if (!beer) {
   note = 1;
   buyBeer();
   note = 0;
  }
 }
}
```

Describe why this solves the problematic context switch suggested on Slide 17. Give an example of a new location for a context switch for the above example which results in concurrent purchase of beer by two flatmates.

Q6. What does it mean for an operation to be atomic? Which operations in Java are guaranteed to be atomic (Hint: search for Java Language Spec and read the relevant section of the book you come across)?

Q7. Describe how you would use the semaphore mechanism to control workers in a beer factory. Worker 1 makes the bottles and places them on a table with a capacity to hold up to 10 bottles. Worker 2 takes bottles from the table and fills them with beer.

Q8. What's the difference between a mutual exclusion lock and a semaphore which is initially set to one?

Q9. Does the solution presented in Slide 31 work?

Q10. Describe a solution to the Producer-Consumer Problem which allows concurrent access to the buffer by one producer and one consumer when you have multiple producers and multiple consumers vying for access.

E1. Write answers to the following exam questions:

- 2005 Paper 5 Question 4
- 2004 Paper 4 Question 8

Supervision 2

Q11. A concurrent program has six threads and seven resources. Each resource has a lock associated with it to ensure only one thread can access it at a time. Threads either hold the lock for a resource (`L`), are blocked waiting to acquire a resource (`B`) or are currently not interested in the resource (`X`). The current state of execution of the program is described in the table below, where each column represents a resource, and each row represents a thread:

T1	L	B	X	X	X	X	L
T2	X	L	X	X	X	X	B
T3	X	X	X	X	X	X	B
T4	X	X	L	X	L	X	X
T5	X	X	X	B	X	L	X
T6	X	X	B	L	X	X	X

Write down the threads which are in deadlock and describe why this is the case. What is the minimum number of threads you need to terminate in order to ensure the program is no longer deadlocked?

Q12. A student has implemented a demo bank system in Java as shown in Appendix B.

a) Run his program.

b) Explain why, despite the fact he has added locking to the `BankAccount` class, his program still produces erroneous results.

c) Copy his source code into a new class called `ConsumerWithLocks` and, by using a global lock on the consumer object, ensure his program runs correctly.

d) Copy his source code into a new class called `ConsumerWith2PL` and, by taking out locks on the `BankAccount` objects, implement Strict Two-Phase Locking.

e) Describe why implementing a solution based on Time Stamp Ordering is difficult write in Java.

E2. Write answers to the following exam questions:

- 2001 Paper 3 Question 1
- 2000 Paper 4 Question 1

Additional Questions

Q13. It is possible to build mutual exclusion using event counts and sequencers (see Handout 2, slide 4) - is it possible to build a traditional (counting) semaphore? How?

Q14. Sketch a solution to the multiple-reader single-writer (MRSW) using Java monitors.

Q15. What are the principal differences between traditional transactions (as implemented by databases) and *software* transactional memory?

Part Two

Supervision 3

TODO

Supervision 4

TODO

Appendix A

A list of exam questions which cover topics still on the current course syllabus are listed below.

- 1998 Paper 11 Question 6
- 1999 Paper 3 Question 1 [ignore "scheduler activations"]
- 2001 Paper 3 Question 1
- 2000 Paper 3 Question 1
- 2000 Paper 4 Question 1
- 2005 Paper 5 Question 4
- 2004 Paper 4 Question 8
- 2008 Paper 11 Question 9
- 2010 Paper 5 Question 4

TODO: this list is rather incomplete

Appendix B

```
public class Customer {

   class BankAccount {
      private int balance;

      BankAccount(int initialBalance) {
         balance = initialBalance;
      }

      synchronized int getBalance() {
         return balance;
      }

      synchronized void credit(int amount) {
         balance = balance + amount;
      }

      synchronized void debit(int amount) {
         balance = balance - amount;
      }
   }

   final BankAccount savings = new BankAccount(1000);
   final BankAccount current = new BankAccount(100);

   int getTotalBalance() {
      return savings.getBalance() + current.getBalance();
   }

   void transferFromSavings(int value) {
      current.credit(value);
      savings.debit(value);
   }

   void transferToSavings(int value) {
      savings.credit(value);
      current.debit(value);
   }

   public static void main(String[] args) {

      final Customer customer = new Customer();

      new Thread(){
         public void run() {
            while(true) {
               int assets = customer.getTotalBalance();
               if (assets != 1100) {
                  System.out.println("Assets are:" + assets);
               }
            }
         }
      }.start();

      while(true) {
         customer.transferFromSavings(100);
         customer.transferToSavings(100);
      }
   }
}
```