THE AUSTRALIAN NATIONAL UNIVERSITY

Second Semester Examination, November 2013

COMP2310 / COMP6310
(Concurrent and Distributed Systems)

Writing Period: 3 hours duration

Study Period: 15 minutes duration

Permitted Materials: One A4 page with handwritten notes on both sides.
NO calculator permitted.

Questions are NOT equally weighted. Marks total 100.
This exam will contribute 50% to your final assessment.
Write your answers in blue or black pen only. Answers written in pencil will not be marked.

The questions are followed by labelled, framed blank panels into which your answers are to be written. Additional answer panels are provided (at the end of the paper) should you wish to use more space for an answer than is provided in the associated labelled panels. If you use an additional panel, be sure to indicate clearly the question and part to which it refers to.

The marking scheme will put a high value on clarity so, as a general guide, it is better to give fewer answers in a clear manner than to outline a greater number in a sketchy, half-answered fashion.

Please write clearly – if we cannot read your writing you may lose marks!

Student Number (please write clearly):	Enrollment (circle one):
	COMP2310
	COMP6310

Official use only:

QUESTION 1 ?? General Concurrency

(a) A concurrent language must provide some concept of a task/thread or other concurrent entity. List two other elements that relate to concurrency and might be expected to be provided by a concurrent language.

QUESTION 1(a) **[2 marks]**

(b) Explain how deadlock may be modeled in a language such as FSP. Give a simple example of an FSP parallel composition which deadlocks where neither of the component process deadlock.

QUESTION 1(b) **[3 marks]**

(c) What is the main similarity between the terms *concurrent system* and *distributed system*? What is the main difference (in terms of style of communication)?

QUESTION 1(c) **[2 marks]**

(d) Define the term *semaphore*. Does the implementation of a semaphore require special hardware support or can a semaphore be implemented in any high-level programming language? Give precise reasons.

QUESTION 1(d) **[3 marks]**

(e) **COMP2310 students**, *answer the following:* Describe a methodology going from requirement to model and model to implementation for concurrent systems.

COMP6310 students, *answer the following:* When composing a concurrent system out of component sub-systems, the properties of the components may either be *compositional* or not. For both safety and progress properties, state and briefly explain whether they are compositional.

QUESTION 1(e) **[3 marks]**

Question 1 (continued)

(f) In a voting system, *N* voters may **cast** a vote for a referendum; the vote may have a value of **1** (accept) or **0** (reject). They then wait until all others have voted (exactly once) for the outcome, which may either be **accept** or **reject** based on which value was in the majority. The whole process then repeats itself. Model this as a concurrent system in FSP.

QUESTION 1(f) **[5 marks]**

Student Number: .....

QUESTION 2 ?? Mutual Exclusion and Synchronization

A central computer with two terminals is used to reserve seats in a concert hall. A seat, initially unreserved, can be reserved (only once) and responds to the respective query action corresponding to its state. At each terminal, a clerk can choose any of the existing `S` seats and then makes a **query** on whether it is reserved; if it is not reserved, they can then **reserve** it. To avoid double-bookings, the clerk **acquire**s a lock before the query on the seat and **release**s it after either reserving it or finding out it is already reserved. The whole system may be expressed in FSP as follows:

```
SEAT = SEAT[0],
SEAT[r: 0..1] = ( query[res] -> SEAT[res]
                | when (r==0) reserve -> SEAT[1]
                | when (r==1) reserve -> ERROR ).
const S = 2
||SEATS = ( seat[1..S]:SEAT ).
TERMINAL = acquire ->
             ( seat[s: 1..S].query[0] -> seat[s].reserve ->
               release -> TERMINAL
             | seat[s: 1..S].query[1] -> release -> TERMINAL ).
LOCK = ( acquire -> release -> LOCK ).
set Terminals = {a, b}
||CONCERT = (Terminals:TERMINAL || Terminals::SEATS || Terminals::LOCK).
```

(a) Write pseudo-code to implement the system, with `LOCK` and `SEAT` implemented as a monitors, and each `TERMINAL` implemented as an active thread. Include the code for the thread creation. Code related to defensive programming and for producing traces may be omitted.

QUESTION 2(a) **[12 marks]**

Question 2 (continued)

QUESTION 2(a) continued.

(b) If it was desired to implement the tracing of actions (in terms of print statements to the standard output), where would you place the corresponding code? Briefly explain why.

QUESTION 2(b) **[2 marks]**

(c) The above system, as expressed in FSP, lacks the desirable property of allowing concurrent allocation of different seats by the clerks. Describe how you would modify the FSP description and your implementation to rectify this.

QUESTION 2(c) **[4 marks]**

QUESTION 3 ?? Safety and Liveness

(a) Describe the term 'wait-for cycle', and illustrate this with an specific example of a deadlocked concurrent system, such as the Dining Philosophers (you may reduce the number of philosophers).

QUESTION 3(a) **[3 marks]**

(b) Many spinlock implementations (e.g. ones that use test-and-set or compare-exchange operations) tend not to be fair. Explain what fairness means in this context. Explain why a spinlock implemented using test-and-set is not fair.

QUESTION 3(b) **[2 marks]**

(c) Consider the readers-writers problem, where either multiple readers or a single writer may **acquire** mutually exclusive resource before the **release**. What is the potential problem of *fairness* that exists for the writers, and how would you design the system to avoid this?

QUESTION 3(c) **[3 marks]**

(d) Provide pseudo-code for two concurrent threads for deadlock-free and starvation-free mutual exclusion of a critical section.

QUESTION 3(d) **[8 marks]**

QUESTION 4 ?? Message Passing

(a) Can you emulate synchronous message passing with asynchronous message passing? Explain why and how. What, if any, difference would you expect to observe?

QUESTION 4(a) **[3 marks]**

(b) Message passing is an alternative implementation for concurrent systems such as the Concert Hall example of Question 2. Describe how the synchronous message passing implementation of such a system differs in this case from that of the monitor based implementation. You may express your answer in terms of your implementation from Question 2 or a monitor implementation of a system of similar complexity that has been studied in this course. You may also express your answer in terms of the following classes which implement synchronous message passing.

```
public class Selectable {
    public void guard(boolean g);
}
public class Channel extends Selectable {
    public synchronized void send(int  v);
    public synchronized int receive();
}
public class Select {
    public void add(Selectable s);
    public synchronized int choose();
}
```

QUESTION 4(b) **[8 marks]**

QUESTION 4(b) continued.

QUESTION 5 ?? Architectures

(a) *An operating system is an example of an inherently concurrent system.* Make an argument to support this assertion.

QUESTION 5(a) **[2 marks]**

(b) Ada provides a `select` statement; a `Select` class (see Q4) may be defined in Java and Posix has a `select()` system call. Discuss the similarities and differences (in terms of semantics and ease of use) of each.

QUESTION 5(b) **[4 marks]**

(c) Compare and contrast the behavior of Unix (heavyweight) processes and Java (lightweight) threads.

QUESTION 5(c) **[3 marks]**

(d) Write a C-based pseudo-code to implement the compound shell command `ls | grep 2310`, by using **`fork()`** and each process executing one of the sub-commands. It may be assumed that the `ls` and `grep` executables are in the `/bin` directory.

Hint: you may find the following system calls useful:

```
int execl(const char *path, const char *arg, ...);
// execute the file path, with command line parameters arg ...
int pipe(int pipefd[2]); //create pipe, store f.d.s in pipefd
int dup2(int oldfd, int newfd); //make newfd a copy of oldfd
// standard input has an fd of 0, standard output an fd of 1
```

QUESTION 5(d) **[5 marks]**

QUESTION 6 ?? Distributed Systems

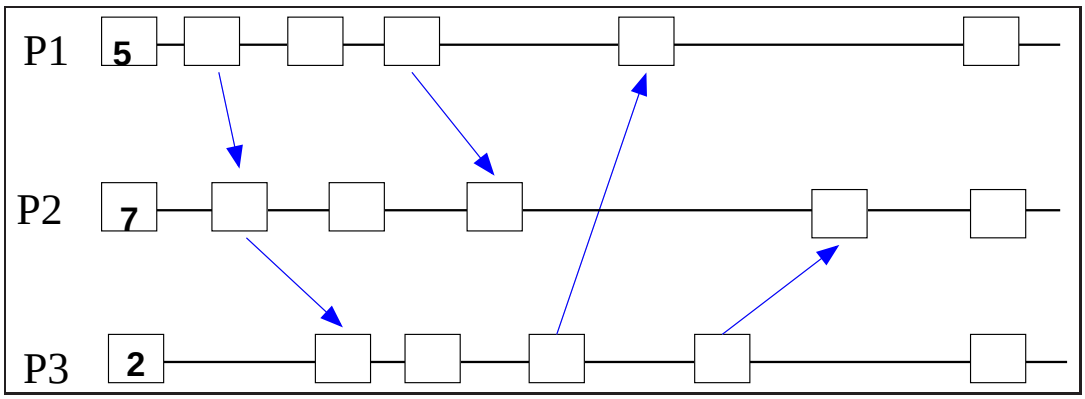
(a) In the context of distributed systems, define the terms *scalability* and *transparency*.

QUESTION 6(a) **[2 marks]**

(b) In the TCP/IP protocol, packets belonging to a single message may arrive out-of-order or sometimes not arrive at their destination. How is this handled in order to make it a reliable protocol?

QUESTION 6(b) **[3 marks]**

(c) The following diagram shows a system of three processes using Lamport's algorithm for logical time. The initial value of the logical time in each process is shown in the first box in each line. Timing events are indicated by subsequent boxes; these may or may not be associated with message passing (indicated by arrows). Within each square box add the value of the logical clock. **[3 marks]**



(d) Define the term *multicast* over a group *g* of processes. Describe an algorithm for a *multicast* of messages over a group *g* of processes that is *reliable*, i.e. can still succeed if an individual message between 2 of the processes is lost.

QUESTION 6(d) **[5 marks]**

(e) Under what condition(s) is a transaction history serializable for two transactions?

QUESTION 6(e) **[2 marks]**

Question 6 (continued)

(f) Discuss which of the individual ACID (atomicity, consistency, isolation and durability) properties are particularly hard to fulfil in a large scale distributed system? Describe a situation where it might be desirable to relax one of these conditions, specifying which of the conditions this is.

QUESTION 6(f) **[3 marks]**

(g) Briefly describe two ways an election algorithm is needed in order to deliver distributed transactions with replicated data. State under what circumstances would the algorithm be initiated. Describe also where an agreement algorithm would be needed in this context

QUESTION 6(g) **[5 marks]**

QUESTION 6(g) continued.

Additional answers to QUESTION __(__)[__]

Additional answers to QUESTION __(__)[__]

Additional answers to QUESTION ___(___)[___]

Additional answers to QUESTION ___(___)[___]

Additional answers to QUESTION ___(___)[___]

Additional answers to QUESTION ___(___)[___]

Additional answers to QUESTION ___(___)[___]

Additional answers to QUESTION __(__)[__]

Additional answers to QUESTION __(__)[__]