TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

EXAMINATIONS — 2007

END-YEAR

COMP 310

Concurrent Programming

Time Allowed: 3 Hours

Instructions: Paper foreign language dictionaries are permitted.

Non-programmable calculators without full alphabetic keys are permitted.

Electronic dictionaries and programmable calculators are not permitted.

An appendix at the end of the exam lists some useful algorithms.

Answer all of the following **six** questions:

1. Critical Sections.
2. Semaphores and Monitors.
3. The Readers/Writers Problem.
4. Channels and Data Spaces.
5. Mutual Exclusion in Distributed Systems.
6. Global Properties and Consensus.

Each question is worth 30 marks.

The exam is worth 180 marks in total.

Question 1. Critical Sections [30 marks]

(a) [5 marks]

Define the following terms with regards to the critical section problem.

(i) Mutual exclusion.

Only one process is in its critical section at any time.

(ii) Deadlock freedom.

If any process is executing the preprotocol (attempting to enter the critical section), then eventually some process enters the critical section.

(iii) Starvation freedom.

If any process is executing the preprotocol (attempting to enter the critical section), then eventually that process enters its critical section.

(b) [10 marks]

(i) What are the possible values of i after the following pseudocode has been executed.

Critical section			
int i=0			
p		**q**	
int x;		int y;	
p1:	x = i;	q1:	y = i;
p2:	i = x + 1;	q2:	i = y + 1;

1,2

(ii) Imagine that the intention of the code above was to enable both p and q to increment i. Write psudocode that correctly achieves this. Use the test-and-set atomic statement.

Use T&S to guarntee exlusive access to i.

(c) [15 marks]

(i) State which of the correctness conditions above are satisfied by the following solution to the critical section problem. Justify your answers.

Critical section	
int turn=1	
p	**q**
loop forever	loop forever
p1: non-critical section	q1: non-critical section
p2: await turn==1	q2: await turn==2
p3: critical section	q3: critical section
p4: turn := 2	q4: turn := 1

When turn=1 (resp. 2), p (resp. q) can enter the CS and q cannot. This guarantees both mutual exclusion and deadlock freedom. The solution is not starvation free. If (eg) p does not complete its NCS when turn=1, q can never enter its CS.

(ii) Prove inductively that the solution has the following invariant.

$$\neg(p3 \wedge q3)$$

Question 2. Semaphores and Monitors [30 marks]

(a) [5 marks]

A semaphore s has two components: an integer `s.value` and a set of processses `s.waitset`. Describe the effect of the `wait` and `signal` operations for a blocking semaphore `s`. You should describe the effect of each operation on `s.value` and `s.waitset`, and on the invoking processes.

> `wait(s)` *first checks whether* `s.value` *is greater than* `0`*. If so, it decrements* `s.value` *and returns. If not, it adds the invoking process into* `s.waitset` *and puts the invoked process into its* blocked *state.*
>
> `signal(s)` *first checks whether* `s.waitset` *is empty. If so, it increments* `s.value` *and returns. If not, it chooses an arbitrary process p from* `s.waitset`*, removes p from* `s.waitset` *and puts p in its* ready *state.*
>
> *Both* `wait` *and* `signal` *execute atomically.*

(b) [10 marks]

Consider the following solution to the two-process critical section problem.

Critical section	
semaphore s:={1,∅}	
p	**q**
loop forever	loop forever
p1: non-critical section	q1: non-critical section
p2: wait(s)	q2: wait(s)
p3: critical section	q3: critical section
p4: signal(s)	q4: signal(s)

(i) Is this solution deadlock free? Why?

> *Yes. If some process is attempting to enter the CS it will eventually execute wait(s). When this happens, either the other process is in the CS or not. In the first case, the process in the CS will eventually complete and execute signal(s), enabling the waiting process to enter the CS. In the second case, s.value=1, so the wait operation can complete immediately.*

(ii) Describe the difference between a weak semaphore and a strong semaphore.

> *A strong semaphore guarantees that waiting processes are signalled in the order in which they executed their wait operations. A weak semaphore does not make this guarantee.*
>
> *Saying that processes wait in a wait queue in a strong semaphore, and a wait set in a weak semaphore is also ok.*

(iii) The algorithm above could be modified to work with any number of processes (rather than just two). With what kind of semaphore would the resulting solution to the N-process critical section problem be starvation free? Why?

When the semaphore is a strong semaphore. *With a strong semaphore, the processes are unblocked in the order that they executed their* `wait` *operations (otherwise put, they wait in a wait queue). For each process, the processes ahead in that order (ahead in the queue) eventually execute signal (after their CS), so the process is eventually unblocked.*

(c) [15 marks]

(i) Describe the *signal and wait* signalling policy for monitors.

A process that is awoken by a signal operation on a condition variable executes (gains exclusive access to the monitor) before *the process that invoked the signal operation, and before any process waiting to enter the monitor. The signalling process then executes before any process waiting to enter the monitor.*

(ii) In Java, the *nested monitor problem* arises when one thread calls `wait()` on an object from within a `synchronized` block in another object. Explain why this can lead to deadlock.

A thread which calls `obj.wait()` *will block at least until* `obj.notify()` *is called on that object by second thread. In a particular application, it may be that a thread will only call* `obj.notify()` *from a* `synchronized` *block in the object on which the original thread is* `synchronized`*. If this is the case, the original thread will never complete its synchronized block, and so the notifying thread can never call* `obj.notify()`*.*

(iii) The Java language could have been designed so that when a thread calls `wait()`, it releases *all* locks on Java objects that it currently holds. What guarantee about Java's `synchronized` blocks would this break? Why would this be a bad idea?

Java monitors need to guarantee that during the interval when some thread is executing a synchronized block, no other thread can gain access to it. This guarantee is necessary so that programmers can ensure that certain operations on the object are not interleaved with other operations. The technique described above would break this guarantee.

Question 3. The Readers/Writers Problem [30 marks]

(a) [5 marks]

How can the use of mutual exclusion degrade performance on a multiprocessor?

When a process has exclusive access to some object or resource, all other processes must wait until iit yields exclusive access. When there are many processes, potentially runnning on different processors, this can result in significant waste of CPU time.

(b) [10 marks]

(i) What are the correctness conditions of the readers/writers problem?

Exclusion: Just readers, or one writer. Deadlock freedom: some process gets to read or write. Starvation freedom: every process eventually reads or writes. Concurrent reads.

(ii) State which of the correctness conditions that you listed above are satisfied by the semaphore based implementation presented in Figure 1 below. Justify your answer.

The solution satisfies the exclusion property, deadlock freedom, and concurrent reads are possible. However, writers may be starved. Concurrent reads.

(c) [15 marks]

Figure 2 presents a solution to the Readers/Writers problem that uses monitors.

(i) Complete the `if` statement in the `StartW` operation.

```
w!=0 or r!=0
```

(ii) Write pseudocode describing how readers and writers would use this monitor.

Reader:	*Writer:*
Loop:	*Loop:*
NCS	*NCS*
StartR	*StartW*
Read	*Write*
EndR	*EndW*

(iii) Readers could be starved if this monitor were run using Java's signalling policy. Explain why?

Consider the situation where there is more than one reader wiating on OKR, and one writer in the write CS, and no writer waiting. When the writer completes, it call signalC(OKR). Then one reader continues exceution of StartR. Eventually, it executes signalC(OKW), the intention being to wake another reader. However, under Java's signalling policy, it is possible for a thread that is waiting to enter the monitor to gain exclusive access before the thread that was just woken. If readers continually invoke StartR, this thread may never gain exclusive access.

Question 4. Channels and Data Spaces [30 marks]

(a) [5 marks] Classify the following types of communication channels using the terms synchronous, asynchronous, addressing and data flow. Make sure that you justify your answers.

(i) Sending a letter.

(ii) Writing graffiti on a wall.

(iii) Emailing a message.

Letter is asynchronous, asymmetric addressing (unless sender has written their address on the back), data flow is potentially two-way (same channel can carry responses back). Grafitti is asynchronous, no addressing required (just write it in a public place) and one-way. Email is asynchronous, symmetric addressing by default and data-flow is two way.

(b) [15 marks] Write an algorithm for a pipeline of processes that can reverse the order of a stream of numbers fed in one end and read out the other. Assume that there are N processes in the pipeline and the input stream of N numbers is followed by a special end of stream marker (*EOS*). For example, a pipeline composed of five processes should be able to transform an input stream of 5, 6, 1, 10, 12, *EOS* into an output stream of *EOS*, 12, 10, 1, 6, 5.

Should have N processes. Each process accepts an input number and stores it, after than it passes on the numbers it sees. When a -1 is accepted it passes its stored number followed by the -1 to the next. This means that eventually the numbers are read out in reverse. Need to describe how the processes are connected to each other.

(c) [10 marks] You are implementing an image recognition system for searching the Nevada desert for a lost plane. You have an existing program that allows *automatic recognition* of a plane in a picture but it is slow.

A friend suggested that volunteers' idle home computers could be put to use searching the entire desert. Sketch how the Master-Worker architecture could be used for this and evaluate its suitability for this purpose.

Job must be capable of being subdivided without cross-communication being required. Master creates tasks by subdividing the image into blocks. Workers contact the master and download the tasks. Workers return result to Master. Loosely coupled so no fixed limit to number of workers. Need to double up on some work if going to handle failure. Some workers may be faster so could download multiple tasks as once but whether this is more efficient depends upon bandwidth and network latency.

Question 5. Mutual Exclusion in Distributed Systems [30 marks]

(a) [5 marks] Consider the following algorithm and explain the result of its execution. Indicate what would be printed to the console and the values of the variables at different points in its execution.

Node 1		Node 3	
p1:	integer x, y	q1:	integer a, b, c, d
p2:	send(add, 3, 1, 30, 40)	q2:	receive(add, a, b, c)
p3:	receive(result, x, y)	q3:	d = add(b,c)
p4:	print result	q4:	send(result, a, d)

Assignment of parameters. Node 3 matches add messages but not subtract messages. The sending node id is sent with messages to allow replies to be sent back. The result of 70 is printed.

(b) [5 marks] Consider the permission-based Ricart-Agrawala algorithm. Imagine the *requestNum* for every node is identical due to an error in the implementation of the algorithm. What effect does this have on the fairness of the algorithm?

Granted in order of process number. Queue jumping will occur. Could lead to starvation. Example should be given to make it clear.

(c) [10 marks] Consider the token-based version of the Ricart-Agrawala algorithm. Can $requested[j]$ be less than $granted[i]$ for $i \neq j$ for node i?

Each node keeps its own requested array. The granted is passed with the token. The example could occur under the following scenario: node one makes a request, gets the token from node two but its request to node three is delayed. Node three requests the token and it is passed from node one. Because the request from node one has not arrived its requested value will be less than granted.

(d) [10 marks] Harry and Hermione are arguing over his choice of the permission-based Ricart-Agrawala algorithm instead of the token-based one.

His application has thousands of nodes, each node is connected by a slow network and the chances of multiple nodes requiring access to the critical section at the same time is low.

Give all the advantages and disadvantages of choosing one algorithm over another and recommend which algorithm should be used.

Two issues: size of the token and a slow network so message sending is expensive. Low contention means that most of the time message sending will not be needed – this should be the main factor in the decision but expect them to discuss all the tradeoffs.

Question 6. Global Properties and Consensus [30 marks]

(a) [15 marks] Consider the Dijkstra-Scholten (DS) algorithm. When drawing diagrams to answer the questions below show each node's $outDeficit$ and annotate each edge with the appropriate $inDeficit$.

(i) Using an example show how the algorithm is designed to handle a process restarting after it has declared its intention to terminate to all dependent nodes. Follow the example through from system start to system termination.

Need at most three processes to explain this. Environment node (node1) activates node 2 and node 3. Node 3 wants to terminate and signals parent. Node 2 wakes up node 3 and becomes its parent. Node 3 wants to terminate and signals Node 2. Node 2 now wants to terminate and signals its parent. Parent declares termination. Should show all outdeficits and indeficits.

(ii) Explain why the environment node would never declare termination if the parent variable was not reset after the completion of signalling.

Node 3 would still signal the environment node, node 2 would never reduce its outdeficit to zero and would never signal the environment node

(iii) Evaluate the correctness of an alternative algorithm where nodes simply inform the environment node when they wish to terminate and the environment node declare system termination when all nodes have reported their willingness to terminate.

A node might declare termination but in the meantime a message might be in transit to it. This would wake it up again. The node that sent the message may have declared termination after sending the message. More problematically, just because a node declares termination does not mean that other active nodes may not activate it.

(b) [10 marks] In the Byzantine Generals algorithm, suppose that there is exactly one traitor and that Zoe's data structures are:

Zoe's Data Structure					
general	plan	reported by			majority
		Basil	John	Leo	
Basil	R		A	R	?
John	A	R		A	?
Leo	R	R	R		?
Zoe	A				A
					?

(i) What can you know about the identity of the traitor?

You cannot tell who the traitor is, but you can tell that it isn't Leo. John and Basil must both be loyal. But then they can't disagree on sent A while Basil relayed R.

(ii) Fill in the values marked ?

The preliminary votes are A, R, R (from top to bottom). Together with final vote is , and ties are resolved in favor of R.

(iii) Construct a minimal scenario leading to this data structure.

Let John be the traitor. Leo and Basil who are loyal both choose Retreat truthfully. John sends Retreat to Leo, and Attack to both Zoe and Basil.

****

```
int nr := 0;
semaphore rw := 1; Rmutex := 1;

process Reader[i = 0 to M] {
  loop {
    Noncritical section;
    wait(Rmutex);
    nr++;
    if (nr == 1) wait(rw);
    signal(Rmutex);
    Read;
    wait(Rmutex); nr--;
    if (nr == 0) signal(rw);
    signal(Rmutex);
  }
}

process Writer[i = 0 to N] {
  loop {
    Noncritical section;
    wait(rw);
    Read and Write;
    signal(rw);
  }
 }
```

Figure 1: A Solution to the Readers/Writers problem using semaphores.

```
monitor RW
  int r := 0, w := 0
  cond OKR, OKW

  op StartR
    if w != 0 or
        !empty(OKW)
      waitC(OKR)
    r := r+1
    signalC(OKR)

  op EndR
    r := r-1
    if r = 0 signalC(OKW)

  op StartW
    if ??? // <----------- COMPLETE THIS STATEMENT
      waitC(OKW)
    w := w+1

  op EndW
    w := w-1
    if empty(OKR)
      then signalC(OKW)
      else signalC(OKR)
```

Figure 2: A solution to the Readers/Writers problem using monitors.