The Australian National University
Second Semester Examination – November 2006

COMP2310
Concurrent and Distributed Systems

Study period:	15 minutes
Time allowed:	3 hours
Total marks:	100
Permitted materials:	None

Questions are **not** equally weighted – sizes of answer boxes do **not** necessarily relate to the number of marks given for this question.

All your answers must be written in the boxes provided in this booklet. You will be provided with scrap paper for working, but only those answers written in this booklet will be marked. Do not remove this booklet from the examination room. There is additional space at the end of the booklet in case the boxes provided are insufficient. Label any answer you write at the end of the booklet with the number of the question it refers to.

Greater marks will be awarded for answers that are simple, short and concrete than for answers of a sketchy and rambling nature. Marks will be lost for giving information that is irrelevant to a question.

Name (family name first):

Student number:

The following are for use by the examiners

Q1 mark	*Q2 mark*	*Q3 mark*	*Q4 mark*	*Q5 mark*	*Q6 mark*	*Total mark*

1. [4 marks] General Concurrency

(a) [4 marks] Name the two basic forms of information exchange between processes in concurrent systems. Under which circumstances would you prefer one of these forms over the other (and visa versa)? (give examples)

2. [6 marks] Synchronization

(a) [3 marks] Define 'mutual exclusion' for multiple processes. Which additional properties do you expect to find in an implementation of a mutual exclusion system?

(b) [3 marks] Can you construct any form of shared memory based synchronization if semaphores are the only supplied synchronization primitive? Detail your answer.

3. [6 marks] Message Passing

(a) [6 marks] Can you emulate asynchronous message passing by means of synchronous message passing? If no, explain why not. If yes, give and explain a solution. If not all features of asynchronous message passing can be emulated, explain the limitations of your solution.

4. [16 marks] Scheduling

(a) [8 marks] In a single CPU system a large set of tasks needs to be scheduled. In case of *unknown computation times* in your task set, which scheduling strategy would apply in order to minimize:

(i) [4 marks] the average turnaround time

(ii) [4 marks] the maximal turnaround time.

Answer (i) and (ii) for the two cases that the task-switching delays are very long / very short.

(b) [4 marks] Now assume that every task provides with every scheduling request the *expected amount of CPU time* which this task will require. Is it useful to change the schedulers which you suggest in part (a) and to incorporate this new information? (You still need to minimize the same values.) If so: in which ways, if not: why not?

(c) [4 marks] What is an optimal fixed priority scheduling scheme?

5. [28 marks] Safety and Liveness

(a) [4 marks] Fairness as a means to avoid starvation is a classical liveness property. Explain the difference between 'linear waiting' and 'first-in, first-out' fairness concepts. Which of these two concepts would you more likely find implemented in a distributed system and why?

(b) [4 marks] Explain the difference between deadlock prevention and deadlock avoidance. Give a concrete example (i.e. a possible way to implement such a scheme) for both cases.

(c) [20 marks] The following Ada program is syntactically correct and will compile without warnings. Read it carefully. You will notice that two lines are commented out. Consider case 1 first.

```
procedure Synced_Processes is

   NoOfClients           : constant Positive := 10;
   NoOfExistingResources : constant Positive :=  5;
   NoOfExistingInstances : constant Positive :=  2;

   type    Resource_Ix         is          range 1..NoOfExistingResources;
   subtype Instances_Available is Natural  range 0..NoOfExistingInstances;
   type    Resources_Available is array (Resource_Ix'Range) of Instances_Available;

   protected Resource_Controller is
      entry     Get_Resource             (Resource_Ix);
      procedure Release_Resource (Ix : in Resource_Ix);
   private
      Resources : Resources_Available := (others => NoOfExistingInstances);
   end Resource_Controller;

   protected body Resource_Controller is

      entry Get_Resource (for Ix in Resource_Ix) when Resources (Ix) > 0 is
      begin
         Resources (Ix) := Resources (Ix) - 1;
      end Get_Resource;

      procedure Release_Resource (Ix : in Resource_Ix) is
      begin
         Resources (Ix) := Resources (Ix) + 1;
      end Release_Resource;

   end Resource_Controller;

   task type Client;

   task body Client is

      NoOfClaimedInstances : constant Positive := 1;                          -- case 1
---   NoOfClaimedInstances : constant Positive := NoOfExistingInstances;      -- case 2
---   NoOfClaimedInstances : constant Positive := NoOfExistingInstances + 1;  -- case 3

   begin
      for Ix in Resource_Ix'Range loop
         for Instance in 1..NoOfClaimedInstances loop
            delay 0.0; Resource_Controller.Get_Resource (Ix);
         end loop;
      end loop;

      for Ix in Resource_Ix'Range loop
         for Instance in 1..NoOfClaimedInstances loop
            delay 0.0; Resource_Controller.Release_Resource (Ix);
         end loop;
      end loop;
   end Client;

   Clients : array (1..NoOfClients) of Client;

begin
   null;
end Synced_Processes;
```

(i) [1 mark] You will find 'active' and 'passive' entities in this program. Name all active entities and the passive entities which are used by those.

(ii) [3 marks] How many task queues are implemented by this program? List and describe them.

(iii) [8 marks] Will the program in the presented ‘case 1’ behave deterministically and will it terminate, deadlock, or livelock? Give precise reasons for all your answers.

(iv) [8 marks] Now consider cases 2 and 3 (imagine the corresponding line un-commented and the other two cases commented out). Will the program now terminate, deadlock, or livelock? Give precise reasons for both cases.

6. [30 marks] Distributed Systems

(a) [4 marks] This question addresses issues associated with virtual (logical) times in distributed systems. If you find two logical times *C(a)* and *C(b)* attached to events *a* and *b* in different processes, then what can you conclude if:

(i) $C(a) \neq C(b)$

(ii) $C(a) < C(b)$

Alternatively, if you know something about the relation between the events *a* and *b* in a distributed system, what can you conclude about their logical times *C(a)* and *C(b)* if:

(iii) *a* happened concurrently with *b*

(iv) *a* refers to the sending event and *b* refers to the receiving event of the same message

(b) [4 marks] Suggest a practical distributed system where you would implement a two-phase locking transaction scheduler, and one practical distributed system where you would implement a time-stamp ordering transaction scheduler. Give reasons for your decisions.

(c) [22 marks] Consider the Ada code below (which compiles without warning). The program implements a complete token ring structure including token recovery in case of a token loss. All tasks in the ring are identical besides their unique id. It is assumed that a network based message passing system is employed for all entry calls between nodes on the ring. It is further assumed that the maximal message passing delay for a complete cycle along the ring is known (and expressed in `Longer_Than_Any_Cycle_Can_Take`).

```
with Ada.Real_Time; use Ada.Real_Time;

procedure Token_Ring is

   type Ring_Range is mod 10;

   task type Chain_Link is
      entry Set_Link_Id (Link_Id   : in Ring_Range);
      entry Token;
      entry Recovery    (Initiator : in Ring_Range);
   end Chain_Link;

   Chain_Links : array (Ring_Range) of Chain_Link;

   task body Chain_Link is

      Id : Ring_Range;

   begin
      accept Set_Link_Id (Link_Id : in Ring_Range) do
         Id := Link_Id;
      end Set_Link_Id;

      declare
         Longer_Than_Any_Cycle_Can_Take : constant Time_Span := milliseconds (1000);
         Ring_Broken_Time_Out           : constant Time_Span := milliseconds (3000);

         Next_Link         : constant Ring_Range := Ring_Range'succ (Id);
         Initiator_Id      : Ring_Range;
         Cycle_Start_Time,
         Token_Sighting    : Time                := Clock;

         task Initiate is
            entry Recovery;
         end Initiate;

         task body Initiate is

         begin
            loop
               select
                  accept Recovery;
               or
                  terminate;
               end select;
               Chain_Links (Next_Link).Recovery (Id);
            end loop;
         end Initiate;
```

(continued on next page)

```
      begin
         loop
            select
               accept Token;

               Token_Sighting    := Clock;
               Cycle_Start_Time := Clock;
               --Chain_Links (Next_Link).Token;

            or accept Recovery (Initiator : in Ring_Range) do
                  Initiator_Id := Initiator;
               end Recovery;

               if Initiator_Id = Id then
                  Chain_Links (Next_Link).Token;
                  Cycle_Start_Time := Clock;
               elsif Initiator_Id > Id then
                  Chain_Links (Next_Link).Recovery (Initiator_Id);
               end if;

            or delay until Cycle_Start_Time + Longer_Than_Any_Cycle_Can_Take;

               Initiate.Recovery;
               Cycle_Start_Time := Clock;

            or delay until Token_Sighting + Ring_Broken_Time_Out;

               exit;

            end select;
         end loop;

      exception
         when Tasking_Error => null; -- Next_Link not callable
      end;
   end Chain_Link;

begin
   for i in Chain_Links'Range loop
      Chain_Links (i).Set_Link_Id (i);
   end loop;
end Token_Ring;
```

(i) [2 marks] You will find a `select-or-delay` statement with two delay alternatives in the main loop of the `Chain_Link` tasks. Describe the meaning of those two alternatives.

(ii) [2 marks] Absolute delays (‘`delay until`’) are employed in this example. Would it be possible to use relative delays instead for the same purpose? If yes: how? if no: why not?

(iii) [2 marks] What is the purpose of the task `Initiate`? Specifically: why is the only call which `Initiate` makes not simply done directly inside the task `Chain_Link` itself?

(iv) [8 marks] When (approximately in seconds after program start) is a ‘token’ provided and by whom? Explain your answer. How many messages have been exchanged (minimally and maximally) before this happens?

(v) [8 marks] Will the program deadlock, livelock, run infinitely, or terminate (assuming a fully reliable message passing system)? What is the probability of termination in case of a 1% probability of a message loss between the chain links (it corresponds here to a 1% probability that an entry call to another Chain_Link task are omitted without any error message or exception). Calculate this probability only for the case that a token is already in circulation (i.e. ignore the first start-up phase in your calculation). Give reasons for your answers.

continuation of answer to question part

continuation of answer to question part

continuation of answer to question [ ] part [ ]

continuation of answer to question [ ] part [ ] continuation of answer to question [ ] part [ ]

continuation of answer to question [ ] part [ ]