The Australian National University
Second Semester Midterm Examination – September 2008

COMP2310
Concurrent and Distributed Systems

Study period:	10 minutes
Time allowed:	1.5 hours
Total marks:	50
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*	*Total mark*

1. [10 marks] General Concurrency

(a) [2 marks] What can you conclude from the fact that different parts of a system are executed **concurrently**? Give a precise answer.

(b) [4 marks] In what sense does the notion of program correctness change when concurrency is employed? Name and explain two properties which are frequently used in correctness proofs for concurrent system.

(c) [4 marks] Sketch all possible process states (as seen by the scheduler/dispatcher) and their transitions (including secondary memory states). Why is the state 'created' different from the state 'ready'?

2. [20 marks] Synchronization

(a) [8 marks] You saw simple forms of two-process deadlocks in multiple places in the lecture. The exclusive usage of which of the following primitives can possibly create such a simple deadlock situation: semaphores, monitors, synchronous message passing, asynchronous message passing? For each of the four primitives give either an example for a deadlock situation or a reason why deadlocks cannot occur.

(b) [4 marks] Name and describe an example for a synchronization hardware primitive which enables you to construct mutually exclusive access to critical sections. Give code which you would insert before and after every critical section based on your chosen hardware support primitive (not considering fairness or starvation conditions).

(c) [8 marks] Consider the following Ada95 program which compiles without warning (and find the questions on the next page).

```
with Ada.Text_IO        ; use Ada.Text_IO;
with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;

procedure Global is

   protected Harmonizer is

      entry     Add (Value : in Natural);
      function  Sum return Natural;
      procedure Open_Add;

   private
      Add_Open        : Boolean  := False;
      Protected_Value : Natural  := 0;
   end Harmonizer;

   protected body Harmonizer is

      entry Add (Value : in Natural) when Add_Open is
      begin
         Protected_Value := Protected_Value + Value;
      end Add;
      function Sum return Natural is
      begin
         return Protected_Value;
      end Sum;
      procedure Open_Add is
      begin
         Add_Open := True;
      end Open_Add;
   end Harmonizer;

   task type Child_Task (Increment : Natural);
   task body Child_Task is
   begin
      Harmonizer.Add (Increment);
      Harmonizer.Open_Add;
      Harmonizer.Add (Increment);
   end Child_Task;

   Child_1 : Child_Task (10);
   Child_2 : Child_Task (20);
   Child_3 : Child_Task (30);

begin
   select
      Harmonizer.Add (40);
   or
      delay 0.1; -- seconds
      Harmonizer.Open_Add;
   end select;
   Put (" First sum  :"); Put (Harmonizer.Sum); New_Line;
   Harmonizer.Open_Add;
   Harmonizer.Add (40);
   Put (" Second sum :"); Put (Harmonizer.Sum); New_Line;
end Global;
```

(i) [4 marks] Is this a deterministic program? (i.e. will it terminate and always provide the same output?) What is the sequence of calls as they are accepted and processed by the protected object? If you find the program non-deterministic then give multiple possible sequences.

(ii) [2 marks] Which output do you expect to find on the terminal?

(iii) [2 marks] The function `Sum` can potentially be called and executed by multiple tasks simultaneously. Why does this not pose a problem and is even explicitly supported by Ada95? Give a precise, technical answer.

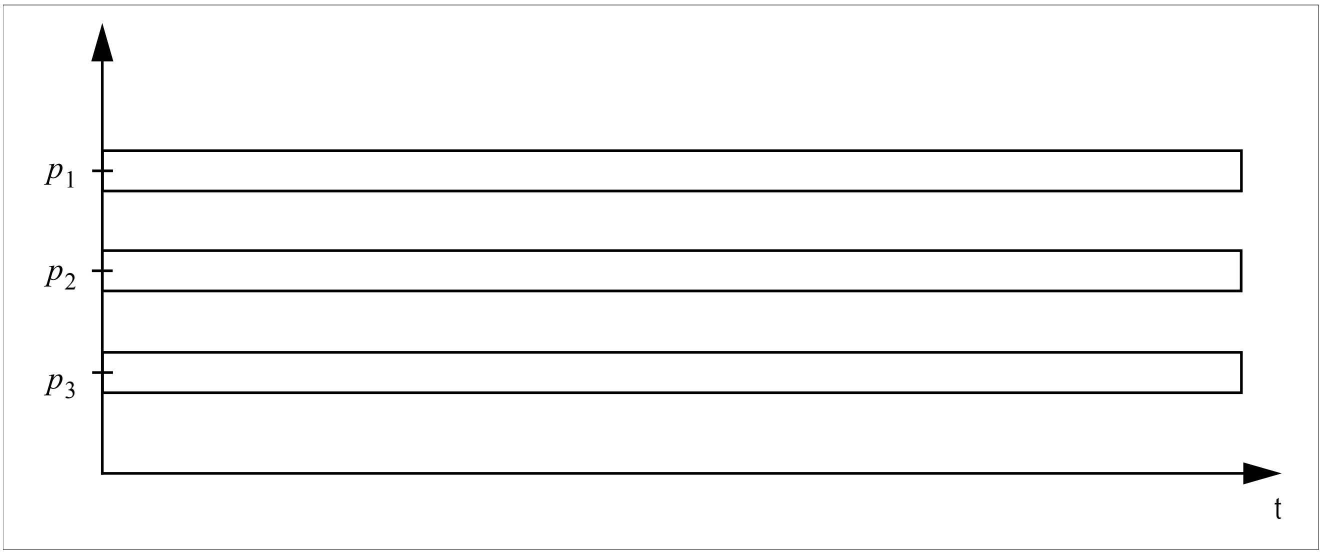
3. [20 marks] Message Passing

(a) [4 marks] While setting up a message passing system between two networked computers, you experience that the message received is different from the message sent - even though you used the same programming language on both sides. Which are the possible problems here? Distinguish between a reproducible and a stochastic difference between the messages. Also distinguish between receipt of a complete (same length) and an incomplete (different length) message.

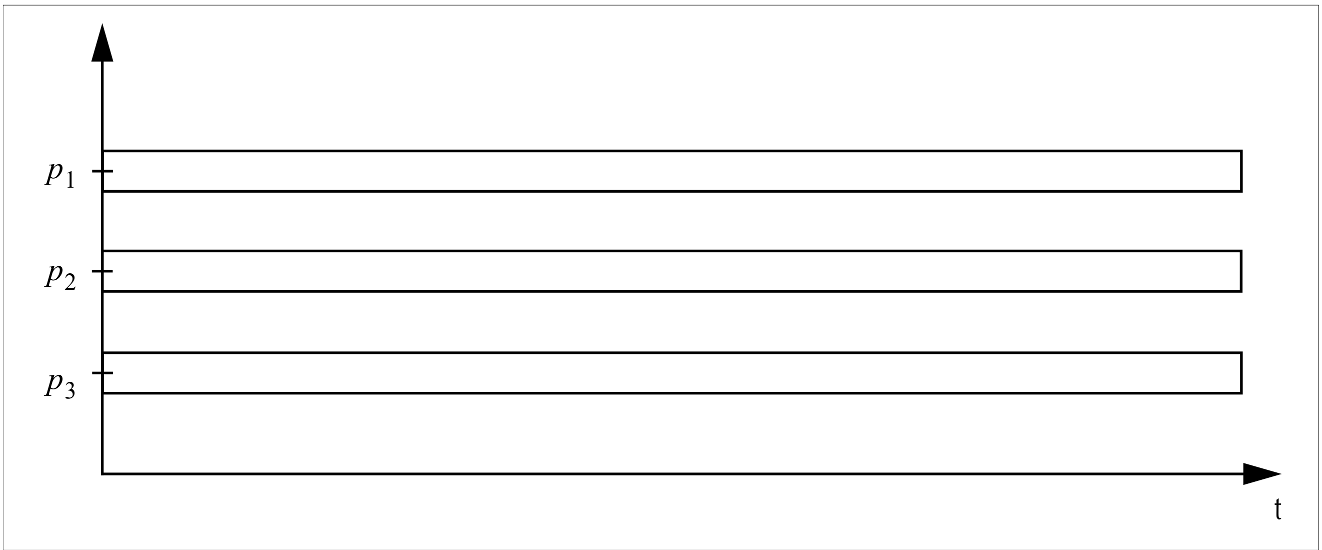
(b) [8 marks] Consider three processes `p1`, `p2`, `p3`, that will communicate with each other using send and receive message passing calls. The following series of events are supposed to take place concurrently:

```
 Process p1           |  Process p2           |  Process p3
-----------------------------------------------------------------
    A                 |     D                 |     G
 sendto (p3)          |  sendto (p1)          |  receivefrom (p2)
    B                 |     E                 |     H
 receivefrom (p2)     |  sendto (p3)          |  receivefrom (p1)
    C                 |     F                 |     I
```

(i) [2 marks] Detail a possible time-line of events assuming that the message passing facility is *asynchronous* (indicate the events `A-I`, and the messages by send-events, receive-events, and connecting arrows).



(ii) [2 marks] Detail a possible time-line of events assuming that the message passing facility is *synchronous* (indicate the events `A-I`, and the messages by send-events, receive-events, and connecting arrows).



(iii) [4 marks] Now consider that all three processes repeat the events given above in infinite loops utilizing synchronous and asynchronous communication as indicated:

```
 Process p1           |  Process p2           |  Process p3
-----------------------------------------------------------------
 loop                 |  loop                 |  loop
    A                 |     D                 |     G
    SendSync (p2)     |     SendAsync (p3)    |     ReceiveSync (p1)
    B                 |     E                 |     H
    SendSync (p3)     |     ReceiveSync (p1)  |     ReceiveAsync (p2)
    C                 |     F                 |     I
 end loop             |  end loop             |  end loop
```

Detail what will happen to the progress of the two remaining processes if one of the processes dies unexpectedly (or is explicitly terminated). Differentiate the cases involving termination of processes `p1`, `p2`, or `p3` (i.e. only one process will terminate in every test-run). Explain what assumptions about the communication systems you made in your answer.

(c) [8 marks] Consider the following Ada95 program. The program is syntactically correct and compiles without warning. (see question on the next page.)

```
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;

procedure Ring is

   type Ring_Range is mod 5;

   task type Task_Type  is
      entry Receive_Task_Id (Task_Id : in Ring_Range);
      entry Ring (Count_In : in Natural);
   end Task_Type;

   Task_Array : array (Ring_Range) of Task_Type;

   task body Task_Type is
      Counter : Natural := 0;
      Towhom  : Ring_Range;
      Id      : Ring_Range;
   begin
      accept Receive_Task_Id (Task_Id : in Ring_Range) do
         Id := Task_Id;
      end Receive_Task_Id;
      Towhom := Id + 1;

      if Id = Task_Array'First then
         Task_Array (Towhom).Ring (Counter);
         accept Ring (Count_In : Natural) do
            Counter := Count_In + 1;
         end Ring;
      else
         accept Ring (Count_In : Natural) do
            Counter := Count_In + 1;
         end Ring;
         Task_Array (Towhom).Ring (Counter);
      end if;
      Put ("Task "); Put (Integer (Id), 2); Put (" counts "); Put (Counter, 2);
New_Line;
   end Task_Type;

begin
   for i in Task_Array'Range loop
      Task_Array (i).Receive_Task_Id (i);
   end loop;
end Ring;
```

(i) [4 marks] Does this program always terminate? If not the all parts of the program terminate, enumerate which parts are not terminating. If the program always terminates, explain which sequence(s) of calls happen(s) before the program terminates.

(ii) [2 marks] What is the smallest and the largest `Counter` number which appears on the terminal? What is the produced terminal output? Is the program (and its terminal output) deterministic?

(iii) [2 marks] Why does the first task in the task array need to behave differently from all other tasks? Explain what would happen if the first task would behave like all other tasks.

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 [ ]