The Australian National University
Mid Term Examination – September 2011

COMP 2310 / COMP 6310
Concurrent and Distributed Systems

Study period:	15 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.

Student number:

The following are for use by the examiners

Q1 mark	*Q2 mark*	*Q3 mark*	*Total mark*

1. [13 marks] General concurrency

(a) [2 marks] Define "Concurrency" as a technical term.

(b) [3 marks] Can you always assume that sequential programs will be executed sequentially on hardware level? Give precise reasons.

(c) [3 marks] Can you always assume that concurrent programs will be executed concurrently on hardware level? Give precise reasons.

(d) [2 marks] Under which specific circumstances will a set of sequential programs become a concurrent system?

(e) [3 marks] State at least three basic facilities which are expected to be provided by a concurrent programming language and describe those facilities briefly.

2. [12 marks] Contention

(a) [8 marks] Does the following pseudo-code provide mutual exclusion with respect to the critical sections? Is this code free from potential dead-locks and/or starvation? Give precise reasons. Simple yes/no answers will get no marks.

```
type Critical_Section_State is (In_CS, Out_CS);

C1, C2: Critical_Section_State := Out_CS;
```

```
task body P1 is                       task body P2 is

begin                                 begin
  loop                                  loop
    -- non_critical_section_1              -- non_critical_section_2
    loop                                  C2 := In_CS;
      exit when C2 = Out_CS;              loop
    end loop;                               exit when C1 = Out_CS;
    C1 := In_CS;                          end loop;
      -- critical_section_1                 -- critical_section_2
    C1 := Out_CS;                         C2 := Out_CS;
  end loop;                             end loop;
end P1;                               end P2;
```

(b) [4 marks] Provide pseudo-code for two concurrent processes for deadlock-free and starvation-free mutual exclusion, based on an atomic operation or language primitive of your choice.

3. [25 marks] Synchronization

(a) [7 marks] Assume that there are three threads or processes, X, Y, and Z, that repeatedly and continuously print "X", "Y", and "Z" respectively. Use a synchronization primitive of your choice to coordinate the printing such that the number of "X"s printed is always less than or equal to the sum of "Y"s and "Z"s printed. Use pseudo-code or any programming language which you are familiar with.

(b) [3 marks] What is the "`requeue`" facility (as in Ada) and why is it required? Name an example of a concurrent system which requires "`requeue`" and explain why it is needed there.

(c) **[15 marks] Read the following Ada program carefully. The program is syntactically correct and will compile without warnings.**

```
procedure Synchronize_It is

   task Stack is
      entry Push;
      entry Pop;
   end Stack;

   protected Storage is
      entry Deposit;
      entry Retrieve;
   private
      Filled : Boolean := False;
   end Storage;

   task body Stack is

      Count : Natural := 0;

   begin
      loop
         select
            accept Push do
               Count := Count + 1;
            end;
         or
            when Count > 0 =>
               accept Pop do
                  Count := Count - 1;
               end;
         or
            terminate;
         end select;
      end loop;
   end Stack;

   protected body Storage is

      entry Deposit when not Filled is begin
         Filled := True;
      end Deposit;

      entry Retrieve when Filled is begin
         Filled := False;
      end Retrieve;

   end Storage;
```

(Code continued in the next column)

```
   task Forwards;
   task Backwards;

   task body Forwards is
   begin
      Stack.Push;
      Storage.Deposit;
      Storage.Retrieve;
      Stack.Pop;
   end Forwards;

   task body Backwards is

   begin
      Storage.Retrieve;
      Stack.Pop;
      Stack.Push;
      Storage.Deposit;
   end Backwards;

begin
   null;
end Synchronize_It;
```

Questions on the following pages

(i) [2 marks] How many task queues are implemented in this program? Name them.

(ii) [2 marks] Which of the four entries in this program are (potentially) blocking?

(iii) [4 marks] Is this a deterministic program? Give precise reasons.

(iv) [7 marks] Does this program always terminate, deadlock, or livelock? If you answered 'no' in the previous question, then you might need to give multiple alternatives here. Give precise reasons.

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	