The Australian National University
Mid Semester Examination – August 2017

Comp2310 & Comp6310 Systems, Networks and Concurrency

Study period:	15 minutes
Time allowed:	1.5 hours (after study period)
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. [15 marks] General concurrency

(a) [2 marks] What is the difference (if any) between concurrent systems and parallel systems? Explain precisely.

(b) [3 marks] What happens when a process executes a `delay` statement? Explain as precisely as you can.

(c) [5 marks] Name and explain all possible state changes of a process. Your explanation need to include what causes those state changes and which software or hardware components implement them.

(d) [5 marks] Can you implement a binary and/or general semaphore in any programming language? Give precise reasons or pseudo-code in both cases. If you provide pseudo-code and you make any assumptions about how that code is supposed to be executed then explain those assumptions clearly. Give precise reasons if you believe that it cannot be done.

2. [27 marks] Synchronization

(a) [14 marks] A large number of concurrent entities operating on a single entity of shared data is a common scenario in concurrent systems.

(i) [2 marks] Explain exactly why there is a need for a coordination mechanism in such a case.

(ii) [4 marks] Write a code section in a programming language of your choice (including pseudocode) which makes *writing* to such shared data safe while preserving a maximal level of concurrency. You should also guarantee that any concurrent entity which wants to write to the shared data will eventually get its turn.

(iii) [4 marks] Write a code section in a programming language of your choice (including pseudocode) which makes *reading* as well as *writing* to such shared data safe while preserving a maximal level of concurrency. (You can reference your previous answer if you answer this question by expanding on your previous answer.)

(iv) [4 marks] Write a code section in a programming language of your choice (including pseudocode) which makes *reading* as well as *writing* to such shared data safe and provide a method which allows a concurrent entity to bypass a queue of currently waiting concurrent entities to *write* on the shared data while preserving a maximal level of concurrency. (You can reference your previous answer if you answer this question by expanding on your previous answer.)

(b) [6 marks] Write a program in a programming language of your choice (including pseudocode) which implement n tasks which each print out m lines of"`Phase` x" (with $x \in [1...m]$, i.e. x is in the discrete range between 1 and m). No task is supposed to enter phase x (and print out"`Phase` x") before all other tasks have complete their phase $x - 1$. Other than this restriction all tasks are required to be running concurrently.

(c) [7 marks] **A fellow student has written the following Ada code fragment. Read it carefully and consider especially the marked lines. See questions on the following page.**

```
protected First_Protector is
   entry Call_Me;
end First_Protector;

protected Second_Protector is
   procedure Call_Available;
   entry Call_Me;
private
   entry Process_Calls;
   Available : Boolean := False;
end Second_Protector;

protected body First_Protector is

   entry Call_Me when True is

   begin
      Second_Protector.Call_Me;                 -- ❶
      Second_Protector.Process_Calls;           -- ❷
      requeue Second_Protector.Call_Me;         -- ❸
   end Call_Me;

end First_Protector;

protected body Second_Protector is

   procedure Call_Available is

   begin
      Available := True;
   end Call_Available;

   entry Call_Me when Available is

   begin
      if Call_Me'Count > 0 then
         requeue Process_Calls;                 -- ❹
      else
         requeue Call_Me;                       -- ❺
         Available := False;
         Call_Me;                               -- ❻
      end if;
   end Call_Me;

   entry Process_Calls when True is

   begin
      null;
   end Process_Calls;

end Second_Protector;
```

(i) [3 marks] For each of the specifically marked lines on the previous page, tick one or multiple of the following options. Consider especially whether there could be situations where this code could lock up indefinitely or lead to infinite loops.

Marked line	ok	compiler warning(s) expected	compiler error(s) expected	questionable runtime behaviour expected
❶				
❷				
❸				
❹				
❺				
❻				

(ii) [4 marks] For those lines which you have *not* marked as "ok" in the previous question, elaborate why they will lead to a warning/error or why you would consider them questionable.

3. [8 marks] Message passing

(a) [8 marks] Seven male gangsters are preparing for their next robbery. In order to keep things on a need-to-know basis they do not reveal their names, but refer to each other by colours. Of course everybody wants to be Mr. Black and nobody wants to be Mr. Pink. Here is where the boss comes in and hands out names on a first-come-first-served basis, so the first gangster who asks to be Mr. Black will indeed become Mr. Black.
The Ada code below shows the activities for each `Gangster` task.
Add the `Boss` entity such that no requested name is ever confirmed twice and therefore the output on the screen shows seven different names. After all names are confirmed the boss will also need to send the gangsters on the job (by calling `Start_Job`). Do not focus on syntax details, but on the logical structure of your code section.
(In slight adaptation of a famous movie.)

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

procedure Shootout is

   type Colours is (Black, White, Blue, Blonde, Orange, Brown, Pink);

   task type Gangster is
      entry Start_Job;
   end Gangster;

   Gangsters : array (Colours) of Gangster;

-- Declaration for Boss is missing here.

   task body Gangster is

      Id      : Colours := Colours'Invalid_Value;
      Success : Boolean := Boolean'Invalid_Value;

   begin
      for Colour in Colours loop
         Boss.Can_I_be_Mr (Colour, Success);
         if Success then
            Id := Colour;
            Put_Line ("I am Mr. " & Colours'Image (Id));
            exit;
         end if;
      end loop;

      accept Start_Job;
      Put_Line ("Boss told me (" &  Colours'Image (Id) & ") to go gangster");
   end Gangster;

begin
   null;
end Shootout;
```

(answer the question on the following page)

continuation of answer to question | | *part* | |

continuation of answer to question | | *part* | |

continuation of answer to question | | *part* | |

continuation of answer to question | | *part* | |