The Australian National University
Mid Semester Examination – August 2019

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. [14 marks] General Concurrency

(a) [6 marks] The following code fragment shows two tasks working on shared data. Read it and proceed to the question below.

```
type Boolean_Field is array (1 .. 10) of Boolean;

All_False : constant Boolean_Field := (others => False);

Shared_Field : Boolean_Field := All_False;
```

```
task Believer;
task body Believer is

begin
   for e of Shared_Field loop
      e := True;
   end loop;
end Believer;
```

```
task Nihilist;
task body Nihilist is

begin
   Shared_Field := All_False;
end Nihilist;
```

(The expression "`(others => False)`" assigns `False` to all elements of the given array.)

After both tasks terminate, what are the values of the elements in the shared array? If different results are possible, then describe all options. Give precise reasons for your answer.

(b) [5 marks] If a process is currently in the state of being blocked, can it transition to another state ever again. What would need to happen for this process to be unblocked, and what would its next state be? Give precise reasons for all parts of your answer.

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

2. [13 marks] Contention

(a) [5 marks] Does the following pseudo-code provide effective mutual exclusion with respect to the critical sections? Give precise reasons.

```
type Critical_Section_State is (In_CS, Out_CS);
C1, C2: Critical_Section_State := Out_CS;
```

```
task body P1 is

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

```
task body P2 is

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

(b) [6 marks] The following code in a C-style programming language implements two operations called `wait` and `signal` on a shared variable `s`. Assume that `blockProcess` is a system call which will block the current process with a reference to the variable `s`.

```
int s = 1;

void wait (int s)
{
    if (s > 0)
       s = s - 1;
    else
       blockProcess (&s);
}

void signal (int s)
{
    s = s + 1;
}
```

Does this code implement a synchronization mechanism? What assumptions do you make in your answer?

(c) [2 marks] Assume that a concurrent program is executed on a single CPU (assume a single, sequential chain of machine instructions). Are mutual exclusion programming methods still required if accessing shared data? Or will the single CPU hardware enforce the correct program behavior anyway? Give precise reasons.

3. [23 marks] Synchronization

(a) [8 marks] Assume that there are two tasks: One and Ten, that repeatedly and continuously print "One" and "Ten" respectively (i.e. task One prints "One"s and task Ten prints "Ten"s). Provide code for each task, so that the total number of "One"s on the terminal is always higher than or equal to 10 times the number of "Ten"s on the terminal, while still allowing for the maximal degree of concurrency. Use pseudo-code or any programming language which you are familiar with.

(b) [8 marks] Read the following Ada program carefully. The program is syntactically correct and will compile without warnings. See comments below and questions on the following page.

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

procedure Task_Scopes is

   task Static_Outer_Task;
   task body Static_Outer_Task is

   begin
      Put_Line (“Start of Static_Outer_Task”);
      declare

         task type Dynamic_Task;
         type Dynamic_Task_Ptr is access Dynamic_Task;
         task body Dynamic_Task is

         begin
            Put_Line (“Dynamic task”);
         end Dynamic_Task;

         task Static_Inner_Task;
         task body Static_Inner_Task is

         begin
            Put_Line (“Start of Static_Inner_Task”);
            declare
               Dynamic_Instance : constant Dynamic_Task_Ptr := new Dynamic_Task;
               pragma Unreferenced (Dynamic_Instance);
            begin
               null;
            end;
           Put_Line (“End of Static_Inner_Task”);
         end Static_Inner_Task;

      begin
         null;
      end;
      Put_Line (“End of Static_Outer_Task”);
   end Static_Outer_Task;

begin
   Put_Line (“Task_Scopes”);
end Task_Scopes;
```

The `pragma Unreferenced` prevents a compiler warning which would point out that `Dynamic_Instance` is not referenced in this program.

(i) [3 marks] Explain at least one purpose of dynamic task allocations.

(ii) [5 marks] Which of the following statements are correct? Tick all correct statements – marks will be subtracted for wrongly ticked statements, so do not just tick all of them.

- ☐ "`Start of Static_Outer_Task`" will always appear before "`Task_Scopes`".
- ☐ "`Start of Static_Outer_Task`" will always appear before "`Start of Static_Inner_Task`".
- ☐ "`Start of Static_Inner_Task`" will always appear before "`Dynamic task`".
- ☐ "`Dynamic task`" will always appear before "`End of Static_Inner_Task`".
- ☐ "`Dynamic task`" will always appear before "`End of Static_Outer_Task`".
- ☐ "`Dynamic task`" will always appear before "`Task_Scopes`".
- ☐ "`Task_Scopes`" can appear anywhere in the output of this program.
- ☐ "`Dynamic task`" can appear anywhere in the output of this program.
- ☐ "`End of Static_Inner_Task`" will always appear before "`End of Static_Outer_Task`".
- ☐ "`Dynamic_Instance`" will live at least as long as the task which it refers to.

(c) [7 marks] Provide code fragments which will allow an arbitrary number of tasks to wait for a specific condition to occur and then to release them one-by-one, once this condition holds true. Each of the released tasks indicates eventually when the next task should be released. Use pseudo-code or any programming language which you are familiar with.

continuation of answer to question ☐ *part* ☐

continuation of answer to question ☐ *part* ☐

continuation of answer to question		*part*	

continuation of answer to question		*part*	