

The Australian National University
Final Examination – November 2016

Comp2310 & Comp6310

Concurrent and Distributed Systems

Study period: 15 minutes
Writing time: 3 hours (after study period)
Total marks: 100
Permitted materials: None with the only exception of an un-annotated paper-based dictionary for candidates with written approval from the school

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.

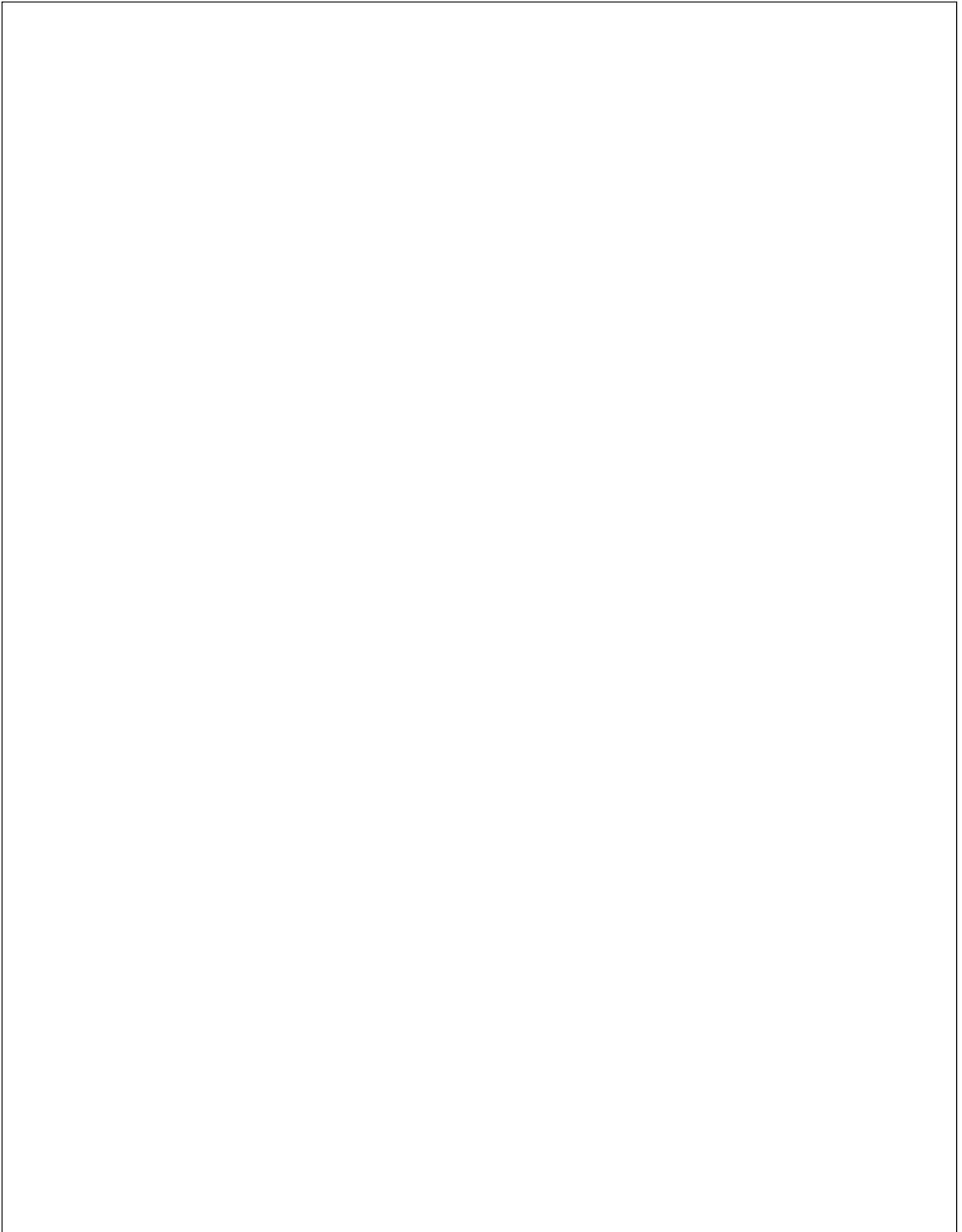
Student number:

The following are for use by the examiners

<i>Q1 mark</i>	<i>Q2 mark</i>	<i>Q3 mark</i>	<i>Q4 mark</i>	<i>Q5 mark</i>	<i>Q6 mark</i>	<i>Total mark</i>

1. [11 marks] General Concurrency

- (a) [4 marks] Define "Process", enumerate its states and depict the potential transitions between those states. Label the transitions with events which trigger those transitions.



(b) [3 marks] What will be the output (or the possible outputs) of the following concurrent program? Give precise reasons for your answer. If you need to make assumptions about the underlying operating system, runtime environment or hardware then state those assumptions as well.

```
with Ada.Text_IO; use Ada.Text_IO;
procedure Global_Variable is
  x : Integer := 0;
  task type Worker (Iterations, Increment : Integer) is
    entry Done;
  end Worker;
  task body Worker is
  begin
    for i in 1 .. Iterations loop
      x := x + Increment;
    end loop;
    accept Done;
  end Worker;
  One : Worker (100, +1);
  Two : Worker (100, -1);
begin
  One.Done;
  Two.Done;
  Put (Integer'Image (x));
end Global_Variable;
```

(c) [4 marks] Can you implement a semaphore yourself in any programming language? Give very precise reasons.

2. [24 marks] Synchronization and Communication

- (a) [8 marks] A calculation can be split up into n concurrent sub-calculations yet the overall calculation can only continue once all n sub-calculations are completed. Sub-calculations report their successful completion or a failure. If any sub-calculation reports a failure then the overall calculation is declared failed as well.

Provide a code fragment which allows the overall calculation to be suspended until all n sub-calculations are completed. Also provide a means that the overall calculation as well as all n sub-calculations are informed about the overall success or failure upon their release from the concurrent calculations. Use a synchronization primitive and language (including pseudocode) of your choice.

(b) [16 marks] Read the following Ada program carefully. The program is syntactically correct and will compile without warnings. See questions on the following pages.

```

with Ada.Text_IO; use Ada.Text_IO;

procedure Pipeline is
  type Stage;
  task type Stage is
    entry Feed (Break_Me    : Natural;
               Last_Factor : Natural := 2);
  end Stage;

  type Link is access Stage;
  function New_Stage return Link is (new Stage);

  task body Stage is
    Number, Factor : Natural := Natural'Invalid_Value;
    Next_Stage     : Link    := null;

begin
  loop
    select
      accept Feed (Break_Me    : Natural;
                  Last_Factor : Natural := 2) do
        Number := Break_Me;
        Factor := Last_Factor;
      end Feed;
    or
      terminate;
    end select;

  declare
    Prime : Boolean := True;
  begin
    for Candidate in Factor .. Number / Factor loop
      if Number mod Candidate = 0 then
        Put ("*" & Positive'Image (Candidate));
        Next_Stage := (if Next_Stage = null then New_Stage else Next_Stage);
        Next_Stage.all.Feed (Number / Candidate, Candidate);
        Prime := False;
        exit;
      end if;
    end loop;

    if Prime then
      Put ("*" & Positive'Image (Number));
    end if;
  end;
end loop;

Pipe : constant Link := New_Stage;

begin
  Put ("1");
  Pipe.all.Feed (8);
  Pipe.all.Feed (27);
  Pipe.all.Feed (125);
end Pipeline;

```

(i) [2 marks] How many task instances are created by this program? Name them.

(ii) [4 marks] Are the tasks in any way synchronized? Explain your answer. If they are synchronized then explain by which synchronization methods and for how long exactly.

(iii) [3 marks] Is there a (potential) deadlock or raised exception in this program? Give precise reasons.

(iv) [7 marks] What is the expected output? (If the output is not deterministic then provide rules for what the output could be.)

3. [9 marks] Selective Concurrency

Read the following Ada code carefully. The tasks and the calling code section are syntactically correct and will compile without warnings.

```
task Selector is
  entry Start;
  entry E1;
  entry E2;
end Selector;
```

with three different versions for its body (all delay values are in seconds):

Version 1:

```
task body Selector is
begin
  accept Start;
  for i in 1 .. 2 loop
    select
      accept E1 do
        delay 2.0;
        Put ('X');
      end E1;
    or
      accept E2;
      delay 2.0;
      Put ('Y');
    or
      terminate;
    end select;

    delay 1.0;
    Put ('Z');
  end loop;
end Selector;
```

Version 2:

```
task body Selector is
begin
  accept Start;
  for i in 1 .. 2 loop
    select
      accept E1 do
        Put ('X');
        delay 2.0;
      end E1;
    or
      delay 2.0;
      accept E2;
      Put ('Y');
      exit;
    end select;

    delay 1.0;
    Put ('Z');
  end loop;
end Selector;
```

Version 3:

```
task body Selector is
begin
  accept Start;
  for i in 1 .. 2 loop
    select
      accept E1 do
        delay 2.0;
        Put ('X');
      end E1;
    else
      accept E2;
      delay 2.0;
      Put ('Y');
      exit;
    end select;

    delay 1.0;
    Put ('Z');
  end loop;
end Selector;
```

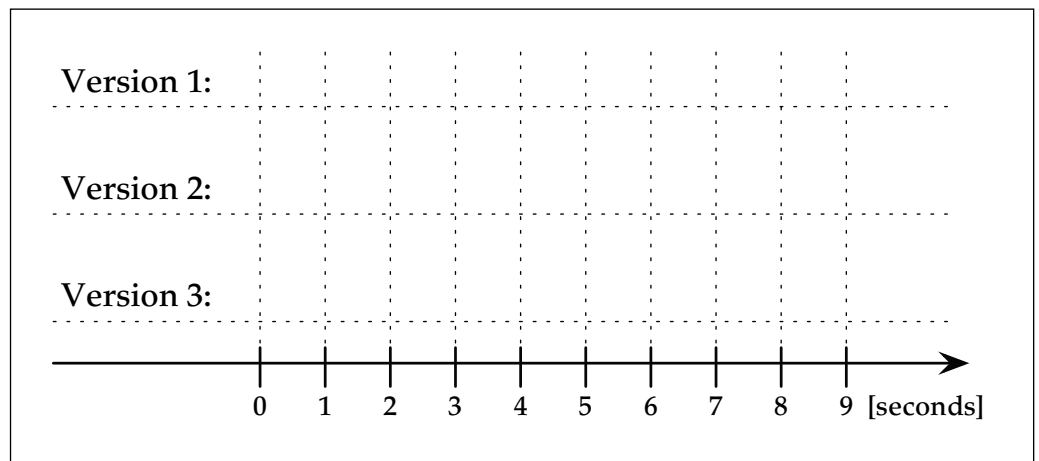
Called by this code section:

Add the outputs for all three versions to the time lines below (assume that Start is called at time zero and you have unlimited CPU capacity)

```
Selector.Start;
Put ('A');
delay 1.0;

Selector.E1;
Put ('B');
delay 1.0;

select
  Selector.E2;
  Put ('C');
or
  delay 1.0;
  Put ('D');
end select;
delay 1.0;
Put ('E');
```



4. [22 marks] Safety and Liveness

- (a) [4 marks] Suggest ways to make it impossible for each of the required deadlock conditions to occur. You should provide one suggestion for each condition.

- (b) [4 marks] Describe the differences between deadlock prevention and deadlock avoidance. Which is preferable in terms of runtime performance and why?

(c) [14 marks] Read the following Ada program carefully. The program is syntactically correct and will compile without warnings. Assume that the `Semaphores` behaves exactly like standard semaphores. See questions below.

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

procedure Router_Family is
  type Router_Ix is mod 5;
  Ports : array (Router_Ix) of Semaphore (Initial => 1);
  task type Router is
    entry Provide_Id (Id : Router_Ix);
  end Router;
  task body Router is
    Router_Nr : Router_Ix := Router_Ix'Invalid_Value;

begin
  accept Provide_Id (Id : in Router_Ix) do
    Router_Nr := Id;
  end Provide_Id;

  Put_Line ("Router" & Router_Ix'Image (Router_Nr) & " acquires its port A");
  Ports (Router_Nr).Wait;
  Put_Line ("Router" & Router_Ix'Image (Router_Nr) & " acquires its port B");
  Ports (Router_Nr + 1).Wait;

  Put_Line ("Router" & Router_Ix'Image (Router_Nr) & " transfers data");
  -- Code omitted which copies data between port A and port B

  Ports (Router_Nr).Signal;
  Ports (Router_Nr + 1).Signal;
  Put_Line ("Router" & Router_Ix'Image (Router_Nr) & " released access to both ports.");
end Router;

Routers : array (Router_Ix) of Router;

begin
  for Id in Router_Ix loop
    Routers (Id).Provide_Id (Id);
  end loop;
end Router_Family;
```

(i) [2 marks] How many tasks are created by this program, how many semaphores are available in total and how many semaphores have to be acquired by each task in order to complete?

(ii) [4 marks] Will this program never/certainly/potentially deadlock? Provide precise reasons.

(iii) [8 marks] If you answered with “certainly or potentially deadlocks” in the previous question then suggest changes to the program such that it never deadlocks. If you answered with “never deadlocks” then suggest changes to the program such that it will potentially deadlock. Which of the required deadlock conditions are you adding or removing with your suggestion?

5. [11 marks] Data Parallelism

(a) [11 marks] Read this syntactically correct Chapel expression and then proceed to the questions below:

```
sqrt (+ reduce ((Vector_1 - Vector_2)**2))
```

where you should assume the following declarations for `Vector_1` and `Vector_2`:

```
const Index = {1 .. 1000};  
var Vector_1, Vector_2 : [Index] real;
```

(i) [1 mark] What is the type of this expression?

(ii) [6 marks] Enumerate and explain the data parallel operations which are implemented by this Chapel expression.

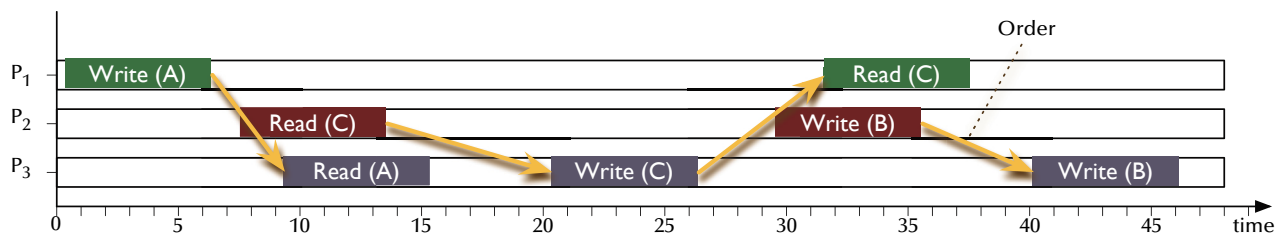
(iii) [4 marks] How many concurrent entities (tasks, processes, threads or alike) are potentially created by this expression? How many of those could potentially be executing as hardware concurrent entities? Give precise reasons for your answer.

6. [23 marks] Distributed Systems

(a) [6 marks] Three transactions:

- 1: Write (A) – Read (C)
- 2: Read (C) – Write (B)
- 3: Read (A) – Write (C) – Write (B)

are executed by three different processes resulting in the following conflicting pairs of operations (indicated by arrows):



Are any two of those transactions serializable? Is the whole set of transactions serializable? Explain your answers precisely.

(b) [5 marks] What can you conclude about the events a and b (including whether they happened on the same or on different processors) if the relations between the logical times $C(a)$ and $C(b)$ associated with these events are:

(i) [1 mark] $C(a) < C(b)$

(ii) [1 mark] $C(a) = C(b)$

(iii) [1 mark] $C(a) \neq C(b)$

(iv) [2 marks] Is it true that if $C(a) > C(b)$ then there always exists an event c , such that: $C(a) > C(c) > C(b)$? Will your answer change if you measure time in calendar (or “real”) time instead of logical time? Give precise reasons for your answers.

(c) [12 marks] Read the following Ada program carefully. The program is syntactically correct and will compile without warnings. See questions on the following pages.

```

with Ada.Text_IO;    use Ada.Text_IO;

procedure Distributed_Workers is
  type Workers_Range is range 1 .. 2;
  type Clients_Range is range 1 .. 4;
  task type Client;
  task Server is
    entry Service;
    entry Report;
  private
    entry Hold;
  end Server;
  task type Worker is
    entry Ready;
    entry Service;
  end Worker;
  task body Client is
  begin
    Server.Service;
    Put ("C");
  end Client;
  Workers : array (Workers_Range) of Worker;
  Clients : array (Clients_Range) of Client;
           pragma Unreferenced (Clients);

  task body Server is
    Free_Workers : Natural := Workers'Length;
  begin
    loop
      select
        accept Service do
          delay 1.0;
          Put ("S");
          for i in Workers_Range loop
            select
              Workers (i).Ready;
              Free_Workers :=
                Free_Workers - 1;
              requeue
                Workers (i).Service;
            else
              null;
            end select;
          end loop;
          requeue Hold;
        end Service;
      or
        accept Report;
        Free_Workers := Free_Workers + 1;
      or
        when Free_Workers > 0 =>
          accept Hold do
            Put ("H");
            requeue Service;
          end Hold;
      or
        terminate;
      end select;
    end loop;
  end Server;

  task body Worker is
  begin
    loop
      select
        accept Ready;
      or
        accept Service do
          delay 2.0;
          Put ("W");
          Server.Report;
        end Service;
      or
        terminate;
      end select;
    end loop;
  end Worker;
begin
  null;
end Distributed_Workers;

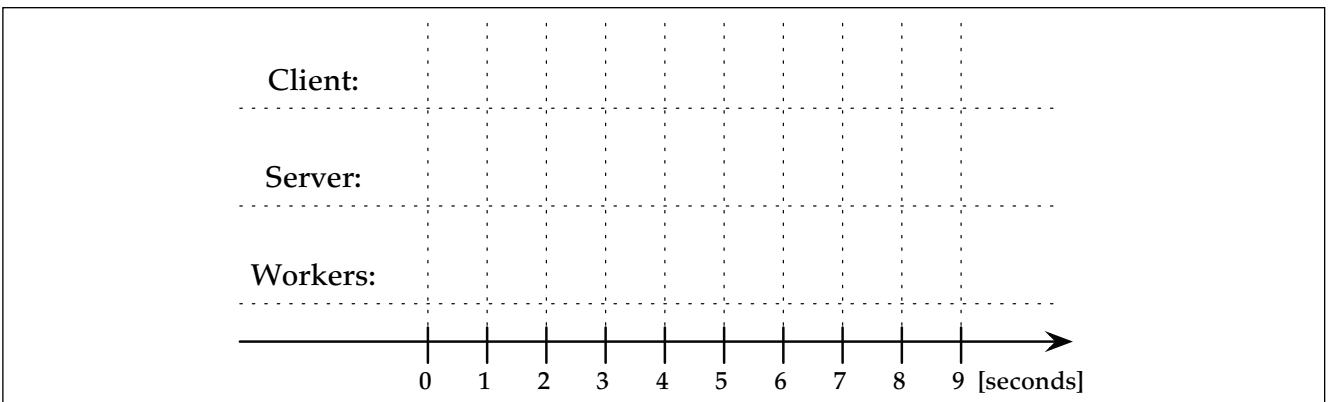
```

(continued in right column)

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

(ii) [4 marks] Considering the program structure, which of the entries in this program would you consider to be potentially blocking for a non-trivial amount of time? Assume that your underlying hardware supports to run all concurrent entities in this program in parallel.

(iii) [6 marks] Develop the sequence of outputs from this program on a time-line. If multiple outputs are possible then show them all.



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

continuation of answer to question

part

continuation of answer to question

part