The Australian National University
Mid Semester Examination – August 2020

Comp2310 & Comp6310
Systems, Networks and Concurrency

Study period:	15 minutes
Time allowed:	2 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 exam form. You can use scrap paper for working, but only those answers written in this form will be marked. Do not upload your exam anywhere but the prescribed exam submission system. There is additional space at the end of the booklet in case the answer boxes provided are insufficient. Label any answer you write at the end of the exam form with the number of the question it refers to and note at the question itself, that you provided addition material at the end.

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) [8 marks] In most concurrent programming languages, any concurrent entity (for example a `task` in Ada) can create new concurrent entities. What are the possible dependencies between the original and the newly created concurrent entity? Explain in terms of access to local variables as well as lifetimes.

(b) [7 marks] A concurrent entity (for example a `task` in Ada) has some influence on where and how often its statements will be interleaved with other concurrent entities. Explain precisely how a concurrent entity can control or moderate those interleavings. Can a concurrent entity completely prevent being interleaved? If so: how exactly?

2. [13 marks] Contention

(a) [13 marks] In most concurrent programming languages, the following simple vector addition will not be atomic.

```
type Vector is array (Coordinates) of Real;

function "+" (V_Left, V_Right : Vector) return Vector is

   V_Result : Vector;

begin
   for Axis in Vector'Range loop
      V_Result (Axis) := V_Left (Axis) + V_Right (Axis);
   end loop;
   return V_Result;
end "+";

Vector_1 : constant Vector := (others => 2.0);
Vector_2 : constant Vector := (others => 3.0);
Result   : constant Vector := Vector_1 + Vector_2;
```

Assume that `Coordinates` is a scalar type and `Real` is a floating point type.

(i) [6 marks] Can you make vector addition an atomic operation? If not, then explain why not. If you can, then provide code to make it atomic. Use pseudo-code or any programming language which you are familiar with. The syntax of your atomic vector addition (if you can make one) does not need to follow the syntax above.

(ii) [7 marks] Make vector addition concurrent. Use pseudo-code or any programming language which you are familiar with.

3. [22 marks] Synchronization

(a) [6 marks] There are multiple entry and exit doors at a concert venue and each entry and exit is equipped with an automatic detection system for when a person leaves or enters (assume each door system to be implemented by individual tasks).

Provide code to calculate the total number of people inside the venue, and allow multiple display units (also assume them to be individual tasks) to show this number on boards across the venue.

In times of viruses all around, we also need to strictly limit the number of people inside the venue at any time. Provide a mechanism so that entry door tasks will not be able to progress any further if the total number of people reaches 100. Use pseudo-code or any programming language which you are familiar with.

(b) **[16 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 pages.**

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

procedure Task_Processing is

   No_Of_Clients : constant Positive := 3;
   type Client_Range is mod No_Of_Clients;

   package Dispense_Client_Ids is new Id_Dispenser (Element => Client_Range);
   use Dispense_Client_Ids;

   protected Release is

      entry Free_1 (Id : Client_Range);
      entry Free_2 (Client_Range);

   private

      Max_Id : Client_Range := Client_Range'Invalid_Value;
      Clients_Left_1, Clients_Left_2 : Client_Range := Client_Range'Last;

   end Release;

   protected body Release is

      entry Free_1 (Id : Client_Range)
         when Free_1'Count = Natural (Clients_Left_1) + 1 is

      begin
         if Max_Id'Valid then
            if Id = Max_Id then
               Max_Id := Client_Range'Invalid_Value;
               Clients_Left_1 := Clients_Left_1 - 1;
               Put_Line (Client_Range'Image (Id) & " is released 1");
            else
               Max_Id := Client_Range'Max (Max_Id, Id);
               requeue Free_1;
            end if;
         else
            Max_Id := Id;
            requeue Free_1;
         end if;
      end Free_1;

      entry Free_2 (for Id in Client_Range) when Id = Clients_Left_2 is

      begin
         Clients_Left_2 := Clients_Left_2 - 1;
         Put_Line (Client_Range'Image (Id) & " is released 2");
      end Free_2;

   end Release;
```

(continued on next page)

```
   task type Client;
   task body Client is
      Id : constant Client_Range := Draw_Id;
   begin
      Release.Free_1 (Id);
      Put_Line (Client_Range'Image (Id) & " is free 1");
      Release.Free_2 (Id);
      Put_Line (Client_Range'Image (Id) & " is free 2");
   end Client;
   Clients : array (Client_Range) of Client; pragma Unreferenced (Clients);
begin
   null;
end Task_Processing;
```

The `pragma Unreferenced` prevents a compiler warning which would point out that Clients is not referenced in this program. Id_Dispenser and `Draw_Id` do exactly what you think they do (they provide unique Id's).

See questions on the following pages.

(i) [8 marks] Explain what is happening in this program. Are `Free_1` and `Free_2` equivalent? If you find differences in their behaviour, then describe them precisely. Which one would you prefer in your own code and why exactly?

(ii) [3 marks] What is the terminal output of this program. If you find that there are multiple options, then describe those options as precisely as you can.

(iii) [5 marks] Use message passing to achieve the same synchronization effect between the Client tasks. 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* [ ]