The Australian National University
Final Examination – November 2020

Comp2310 & Comp6310
Systems, Networks and Concurrency

Study period:	15 minutes
Time allowed:	3.5 hours (after study period)
Total marks:	100
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 additional 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*	*Q4 mark*	*Q5 mark*	*Q6 mark*	*Total mark*

1. [15 marks] General Concurrency

(a) [9 marks] Concurrent programming languages will always provide some form of concurrent entities. Their names vary widely (in Ada for instance they are called *tasks*), or they may not be mentioned at all as part of the syntax. Yet in all cases, the compiler will need to provide an implementation of those concurrent entities.

(i) [3 marks] What are the options for the compiler designer to implement the concurrent entities of a concurrent programming language? Describe each option briefly.

(ii) [6 marks] What are the advantages and disadvantages of each option?

(b) [6 marks] Which forms of hardware are supportive or required for the concurrent execution of your code? Enumerate them and discuss their impact on the performance of your programs briefly. Also state what needs to be done so that they can perform (optimally).

2. [31 marks] Message Passing

(a) [23 marks] In an assignment, students were asked to provide a solution for a message buffer task, which provides the following properties: The buffer task shall …

a. … read messages from this entry:

```
type Notes is range 1 .. 10;

task Buffer is
   entry Receive (Note : Notes);
end Buffer;
```

b. … deliver messages to a provided receiver task via this entry:

```
task Receiver is
   entry Transmit (Note : Notes);
end Receiver;
```

c. … allow the sender (the task calling `Receive`) to send messages even if the `Receiver` is currently not available (as long as the buffer still has storage capacity).

d. … be able to store more than 1 message.

e. … deliver messages in the same order in which they have been received.

f. … always be responsive to `Receive`, as long as the buffer still has storage capacity.

g. … send out stored messages to the `Receiver`, as soon as the `Receiver` is available.

h. … not use CPU time, when no messages can be passed.

i. … terminate when an invalid value is received.

The submissions of 3 students are below (all compile without warnings of course).

Student A:

```
task body Buffer is

  Store : Notes := Notes'Invalid_Value;

begin
   loop
      accept Receive (Note : Notes) do
         Store := Note;
      end Receive;
      exit when not Store'Valid; -- terminate Buffer on invalid value received
      Receiver.Transmit (Store);
   end loop;
end Buffer;
```

Student B:

```
task body Buffer is
   Store     : Notes   := Notes'Invalid_Value;
   Shop_Open : Boolean := True;
begin
   while Shop_Open loop
      select
         when not Store'Valid => accept Receive (Note : Notes) do
            Store := Note;
            Shop_Open := Note'Valid;
         end Receive;
      else
         if Store'Valid then
            Receiver.Transmit (Store);
            Store := Notes'Invalid_Value;
         end if;
      end select;
   end loop;
end Buffer;
```

Student C:

```
task body Buffer is
   type Index is mod 5;
   package Storage is new Queue_Pack_Protected_Generic (Notes, Index);
   use Storage;
   Store : Protected_Queue;
begin
   loop
      select
         accept Receive (Note : Notes) do
            Store.Enqueue (Note); -- can also Enqueue invalid values
         end Receive;
      else
         if not Store.Is_Empty then
            declare
               Note : Notes := Notes'Invalid_Value;
            begin
               Store.Dequeue (Note);
                                  -- raises Constraint_Error on invalid return value
               Receiver.Transmit (Note);
            end;
         end if;
      end select;
   end loop;
exception
   when Constraint_Error => null; -- terminate Buffer on invalid value received
end Buffer;
```


See questions on the following pages.

(i) [15 marks] Evaluate all three student submissions with the form below. Tick all properties (by writing "x") which have been successfully implemented and provide detailed feedback for improvements if the solution falls short of expectations.

Property	Student A	Student B	Student C
a.	☐	☐	☐
b.	☐	☐	☐
c.	☐	☐	☐
d.	☐	☐	☐
e.	☐	☐	☐
f.	☐	☐	☐
g.	☐	☐	☐
h.	☐	☐	☐
i.	☐	☐	☐
Feedback:			

(ii) [8 marks] If you find all three student submissions lacking in some regard, provide your own solution below. Use any programming language of your choice (including pseudo code) to implement a buffer, which fulfils all required properties, and is based on synchronous message passing. If you find one of the student submissions already fulfils all requirements, then nominate that one for full marks.

(b) [8 marks] Assume that a concurrent programming language does not provide you with a synchronous message passing, but with some form of asynchronous message passing. This could be a synchronous queue or a maybe a completely asynchronous message passing system.

(i) [8 marks] Construct a synchronous message passing system. Use any programming language of your choice (including pseudo code). You should provide two methods, function, procedures, subroutines, processes, threads, or tasks (depending on the language of your choice and your design) to provide a synchronous sending and a synchronous receiving operation respectively. State your assumptions about the asynchronous message passing system which you use as a foundation.

3. [20 marks] Data Parallelism

(a) [12 marks] Some programming languages provide "implicit concurrency". While it is good to automate things, it is also good to understand what is going on.

(i) [4 marks] Will implicit concurrency always, sometimes or never add synchronization mechanisms (like locks) to your code? Give precise reasons for your answer.

(ii) [4 marks] Can or will implicit concurrency take full advantage (in terms of maximal performance for your program) of all existing hardware? Give precise reasons for your answer.

(iii) [4 marks] Can implicit concurrency lead to unsafe code (for example: race conditions)? Give precise reasons for your answer. Which assumptions did you make to answer this question?

(b) [8 marks] Write a program to implement the discrete cross-correlation function (as a discrete array) between two cyclic, discrete functions (which are themselves represented by discrete arrays) which optimizes for performance on an 8-core CPU with vector processing units (processing 8 16-bit integer numbers per vector operation):

$$Cross\_Correlation(A,B)_k = \sum_i (A_i \cdot B_{i+k})$$

Sequentially such a function could be implemented like this:

```
subtype Input_Range  is Integer range -(2**15) .. +(2**15 - 1);
subtype Output_Range is Integer range -(2**31) .. +(2**31 - 1);

type Samples is mod 2**16;

type Input_Function  is array (Samples) of Input_Range;
type Output_Function is array (Samples) of Output_Range;

function Cross_Correlation (A, B : Input_Function) return Output_Function is

   CC : Output_Function := (others => 0);

begin
   for k in Samples loop
      for i in Samples loop
         CC (k) := CC (k) + A (i) * B (i + k);
      end loop;
   end loop;
   return CC;
end Cross_Correlation;
```


Use any programming language of your choice (including pseudocode). State what you assume about your compiler.

4. [11 marks] Scheduling

(a) [11 marks] The CPU scheduler on the computer which you are using right now has been carefully designed and optimized over decades. Let's have a closer look:

(i) [5 marks] What can your operating system know about the processes running on your computer?

(ii) [3 marks] What information could your scheduler use to make a good decision of what to schedule next?

(iii) [3 marks] What do you assume your scheduler will optimize for specifically, and how will it do that?

5. [11 marks] Distributed Systems

(a) [11 marks] Mutual exclusion in distributed systems via Token Ring structures have been discussed in the lectures, yet we did not provide a concrete implementation to illustrate the method. Read the following Ada code carefully. It is syntactically correct and will compile without warnings.

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

procedure Token_Ring_Mutual_Exclusion is

   Current_Token_Task : Task_Id := Current_Task; -- shared variable

   type Node;
   type Node_Access is access all Node;

   task type Node is
      entry Link (Next : Node_Access);
      entry Token;
   end Node;

   task body Node is
      Next_Node : Node_Access;
   begin
      accept Link (Next : Node_Access) do
         Next_Node := Next;
      end Link;
      loop

         Put_Line ("Previous token owner: " & Image (Current_Token_Task));
         accept Token do
            Current_Token_Task := Current_Task;
            Put_Line ("Current token owner:  " & Image (Current_Token_Task));
            Next_Node.all.Token;
         end Token;

         Put_Line (Image (Current_Task) & " runs concurrently");

      end loop;
   end Node;

   type No_Of_Nodes is mod 10;
   Nodes : array (No_Of_Nodes) of aliased Node;

begin
   for N in No_Of_Nodes loop
      Nodes (N).Link (Nodes (N + 1)'Access);
   end loop;

   Nodes (Nodes'First).Token;
end Token_Ring_Mutual_Exclusion;
```

The provided code is intended to illustrates mutually exclusive access (reading and writing) to a shared variable in an ongoing sequence of accesses from all tasks (the program is not designed to terminate). `Current_Task` (from `Ada.Task_Identification`) provides the `Task_Id` of the currently running task, i.e. the task calling `Current_Task`.

(i) [2 marks] Will this program provide ongoing output? Or will it crash, dead-lock, or live-lock? Give detailed reasons for your answer.

(ii) [3 marks] Does this code actually provide mutually exclusive access to the shared variable while also keeping all tasks running concurrently while they are not accessing this shared variable? Give detailed reasons for your answer.

(iii) [6 marks] If you find the provided code lacking in some respect, please provide an alternative implementation below. Use any programming language of your choice (including pseudo code), that is based on synchronous message passing. If you find the provided implementation to be perfect, than nominate it for full marks.

6. [12 marks] Networks & Time

(a) [6 marks] What OSI network layer 1 (physical layer) interfaces do you find in your computer? Enumerate them and name the network protocols which are utilizing those interfaces (for some you cannot know the associated protocols, so leave that open). Hint: not all of those interfaces will be accessible to you directly. We do not expect you to actually know all of them, but we expect you to be able to take an educated guess of what needs to be or will most likely be found in your computer.

(b) [6 marks] Logical time (or Lamport time) is often preferred in computer applications over wall-clock time.

(i) [3 marks] What are the advantages and disadvantages of logical time?

(ii) [1 marks] What can you conclude if a process receives a message which has a logical time stamp which is larger then the logical time of the current process?

(iii) [2 marks] What can you conclude if the logical time stamps of two events are identical?

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* [ ]