The Australian National University
Final Examination – November 2019

Comp2310 & Comp6310
Systems, Networks and Concurrency

Study period:	15 minutes
Writing time:	3 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 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 (and also note inside the original answer box that your answer is continued at the end of the booklet).

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. [16 marks] General Concurrency

(a) [6 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. If you find a statement to be incorrect, then provide a corrected version of that statement in the answer box underneath by replacing only the *italics* part of the statement.

☐ All **concurrent programming languages** are capable of providing errors or warnings *with respect to synchronization operations*.

☐ **Message passing** is an operation between an active entity (task) and *a passive entity (for example a shared function)*.

☐ **Message passing** is often considered a safer alternative to shared memory based communication forms *as information is usually copied instead of shared*.

☐ **Deadlock prevention** limits *the scope of problems which can be solved*.

☐ If **all four necessary deadlock conditions are fulfilled,** then *a deadlock can possibly be avoided*.

☐ **An simple assignment statement** (for example between two integer variables) in a concurrent programming language *is atomic*.

☐ **Any code section** in a concurrent programming language *can be made to be atomic*.

☐ **Concurrent programs** require *parallel hardware*.

☐ **Hyper-threading** is implemented by *replicating a processor's arithmetic logic unit*.

☐ **Vector processing** is implemented by *replicating a processor's registers*.

☐ An **implicitly concurrent program** is executed *concurrently, and without any chance of deadlocks*.

☐ An **implicitly concurrent program** is *free of any blocking*.

(b) [6 marks] How do the following hardware architecture concepts relate to concurrent programming (if at all)?

Pipelines, Vector processors, Hyper-threading,
Out of order execution, Multiple cores, Virtual memory

Give precise reasons for your answers.

Pipelines:

Vector processors:

Hyper-threading:

Out of order execution:

Multiple cores:

Virtual memory:

(c) [4 marks] Name four concurrent programming language primitives (syntactical constructs which are understood by the compiler) which can or will lead to concurrently executing code. Explain for all four primitives, why they (potentially) result in concurrent code.

2. [16 marks] Synchronization and Communication

(a) [10 marks] Make a suggestion for a new, concurrent programming language (or an amendment to an existing programming language), which cannot express a potentially deadlocking program (while of course still providing the benefits of concurrent programming in general). Give precise reasons for your choices and why your choices will make it impossible to write a deadlocking program.

(b) [6 marks] Write a program in any programming language of your choice (including pseudo code) which implements a race condition. Yes, this is commonly considered a bad thing, so you are asked in this question to provide an example of bad programming.

3. [22 marks] Message Passing

(a) [9 marks] 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;

      loop
         select
            accept E1 do
               delay 1.0;
               Put ('X');
            end E1;
         or
            accept E2 do
               delay 1.0;
               Put ('Y');
               delay 1.0;
            end E2;
         or
            terminate;
         end select;

         delay 2.0;
         Put ('Z');
      end loop;

   end Selector;
```

Version 2:

```
task body Selector is

   begin
      accept Start;

      loop
         select
            accept E1 do
               delay 1.0;
               Put ('X');
            end E1;
         or
            delay 2.0;
            accept E2;
            Put ('Y');
            exit;
         end select;

         delay 2.0;
         Put ('Z');
      end loop;

   end Selector;
```

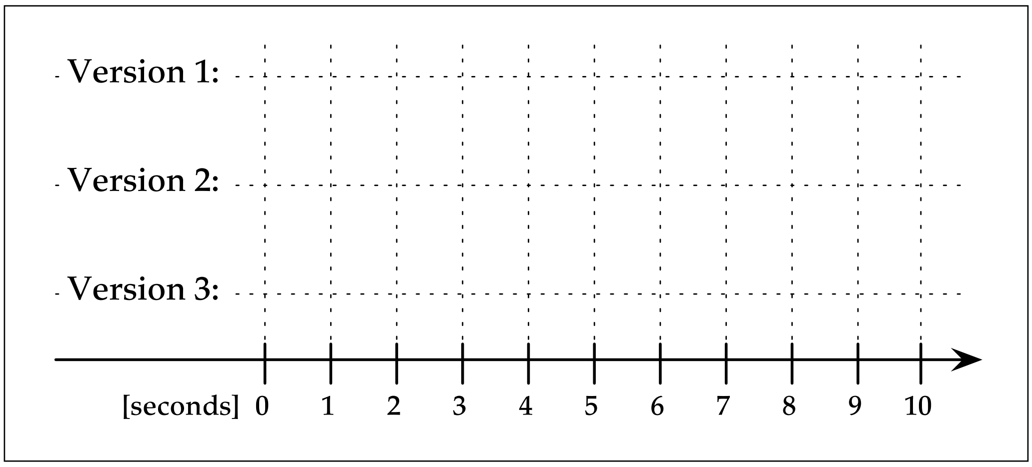
Version 3:

```
task body Selector is

   begin
      accept Start;

      loop
         delay 2.0;
         select
            accept E1 do
               Put ('X');
            end E1;
         else
            accept E2;
            Put ('Y');
            exit;
         end select;

         delay 2.0;
         Put ('Z');
      end loop;

   end Selector;
```

Called by this code section:

```
Selector.Start;
delay 1.0;
Selector.E1;
delay 1.0;
Put ('A');
delay 1.0;
select
   Selector.E2;
   Put ('B');
else
   delay 1.0;
   Put ('C');
end select;
delay 1.0;
Put ('D');
```

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)



(b) [13 marks] Read the following Ada program carefully. The whole program is syntactically correct and will compile without warnings. See questions on the following pages.

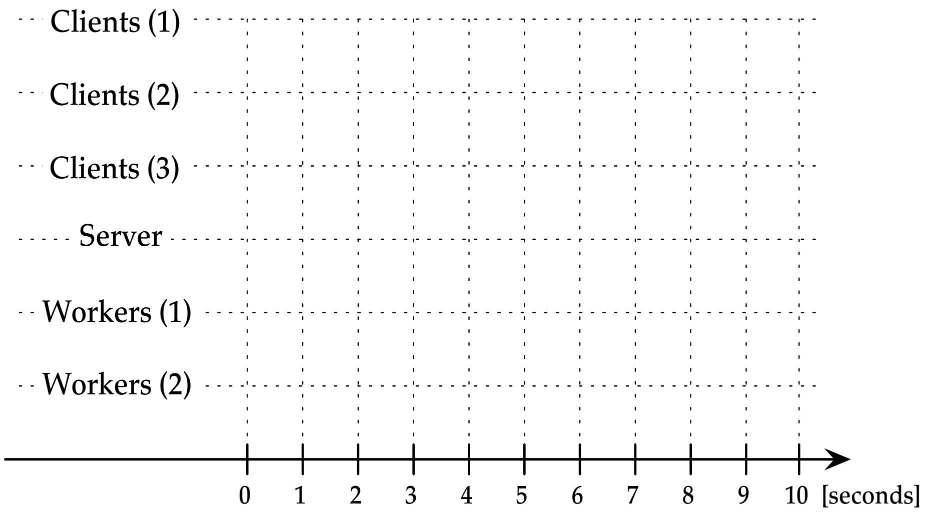
```
with Ada.Text_IO; use Ada.Text_IO;

procedure Working_Class is

   type Workers_Range is range 1 .. 2;
   type Clients_Range is range 1 .. 3;

   task type Worker is
      entry Set_Id (Provided_Id : Workers_Range);
      entry Service;
   end Worker;

  task type Client;

   Workers : array (Workers_Range) of Worker;
   Clients : array (Clients_Range) of Client; pragma Unreferenced (Clients);

   task Server is
      entry Check_In (Id : Workers_Range);
      entry Service;
   private
      entry Backlog;
   end Server;

   task body Worker is

      Id : Workers_Range := Workers_Range'Invalid_Value;

   begin
      accept Set_Id (Provided_Id : Workers_Range) do
         Id := Provided_Id;
      end Set_Id;
      loop
         select
            accept Service do
               delay 1.0;
               Put ('W'); --> Output!
               delay 1.0;
            end Service;
         or
            terminate;
         end select;
         Server.Check_In (Id);
      end loop;
   end Worker;

  task body Client is

   begin
      Server.Service;
      Put ('A'); --> Output!
      Server.Service;
      Put ('B'); --> Output!
   end Client;
                                                     --  (continued on next page ..)
   task body Server is

      type Workers_State is (Busy, Idle);

      States : array (Workers_Range) of Workers_State := (others => Idle);

   begin
      loop
         select
            accept Check_In (Id : Workers_Range) do
               States (Id) := Idle;
            end Check_In;
         or
            accept Service do
               for i in Workers_Range loop
                  if States (i) = Idle then
                     States (i) := Busy;
                     requeue Workers (i).Service;
                  end if;
               end loop;
               Put ('X'); --> Output!
               requeue Backlog;
            end Service;
         or when (for some s of States => s = Idle) =>
               accept Backlog  do
                  Put ('Y'); --> Output!
                  requeue Service;
               end Backlog;
         or
            terminate;
         end select;
      end loop;
   end Server;

begin
   for w in Workers_Range loop
      Workers (w).Set_Id (w);
   end loop;
end Working_Class;
```

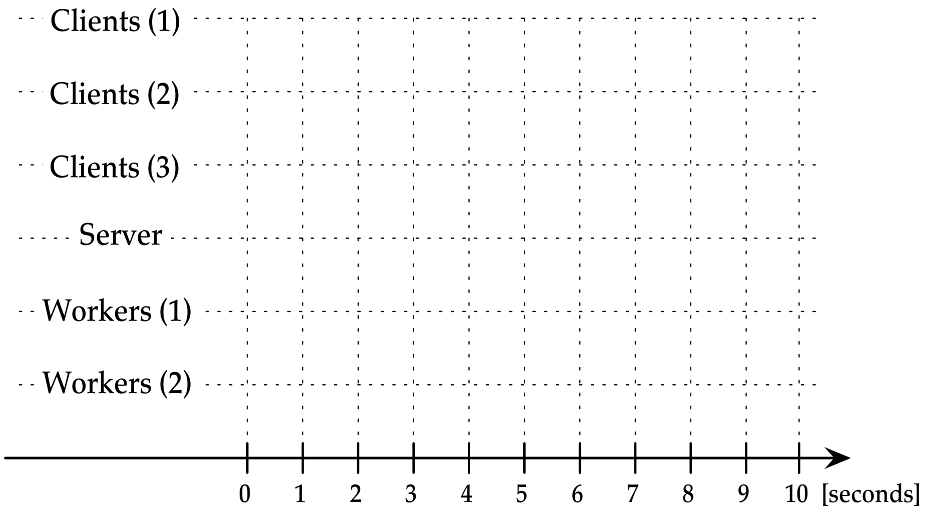
The `pragma Unreferenced` prevents a compiler warning which would point out that `Clients` is not referenced in this program.

(i) [3 marks] In the program above mark all message sending statements, all requeuing statements and all extended rendezvous blocks. Provide a legend in the answer box below to indicate how you marked the three different kinds of code sections.

(ii) [4 marks] Will the tasks in this program always, sometimes or never terminate? Give precise reasons for your answer. This could for example be a specific case where some tasks are blocked forever, or an explanation why every message will eventually be accepted and every extended rendezvous will eventually be completed.

(iii) [6 marks] On the following time-line(s), provide the output which you expect from each task in this program. If the output is non-deterministic, then also describe a second, structurally different possibility (not just a permutation of indices). Consider zero seconds to be the start-time of the program.





4. [9 marks] Scheduling

(a) [3 marks] What is preemptive scheduling and for what reasons is it commonly used?

(b) [3 marks] Which scheduling algorithm would you suggest in order to minimize the maximum turnaround time for a task set of unknown characteristics (especially: you do not know the computation times). Give precise reasons.

(c) [3 marks] If you know the exact computation times of all tasks in a task set, would you change to a different scheduling algorithm in question (b) (while still minimizing the maximum turnaround time)? If so: to which other scheduling algorithm? Give precise reasons.

5. [8 marks] Data Parallelism

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.

6. [29 marks] Distributed Systems & Architectures

(a) [8 marks] Transactions

(i) [3 marks] Transactions are said to fulfil the ACID properties. One of those properties is often not strictly followed, when the overall performance of a system is important. Which property would that be and why does its violation allow for a potentially higher performing system?

(ii) [5 marks] Executing transactions concurrently, does require some analysis of their potential interferences. Suggest at least one way how one can guarantee that the concurrent execution of transactions will not leave a system in an inconsistent state. Give precise reasons for your answer.

(b) [6 marks] Enumerate and describe the OSI network layers which need to be implemented in a network router.

(c) [3 marks] Explain why it is practically impossible to record a global snapshot of most distributed system at a specific global time. Give precise reasons.

(d) [4 marks] What kind of global snapshot is practically achievable for most distributed systems? Explain briefly how you can acquire such a snapshot.

(e) [8 marks] Write a program in any programming language of your choice (including pseudo code) which implements distributed mutual exclusion in an effective and efficient way.

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* ☐