063174

THE UNIVERSITY *of York*

BSc, BEng and MEng

DEPARTMENT OF COMPUTER SCIENCE

Part III(B)

Real-Time Systems and Programming Languages

Time allowed: **Three (3) hours**

Candidates should answer not more than **four** questions

1 (25 marks)

(i) [6 marks] Define what is meant by avoidance synchronisation and synchronisation based on conditional waiting.

(ii) [6 marks] Although the Ada 83 select and accept statements were based on the concept of avoidance synchronisation, there was no requeue mechanism. Explain the main limitation of the Ada 83 approach to avoidance synchronisation and discuss the extent to which the Ada 95 requeue mechanism removes this limitation. Briefly discuss another way of removing the limitation and yet still keeping the avoidance model.

(iii) [8 marks] Show how a counting semaphore can be implemented using an Ada 95 protected object.

(iv) [5 marks] Given that Ada 95 supports protected objects, should the Ada 95 rendezvous facilities be removed from the language? Explain your answer.

2 (25 marks)

(i) [18 marks] Consider a data item which is to be shared between a single producer and a single consumer. The producer is a periodic task which reads a sensor and writes the value to the shared data item. The consumer is a sporadic task which wishes to take the latest value placed in the shared data item by the producer.

The following package claims to provide a generic algorithm with which the producer and consumer can communicate safely WITHOUT the need for mutual exclusion or busy waiting.

```
generic
  type Data is private;
  Initial_Value : Data;
package Simpsons_Algorithm is
  procedure Write(Item: Data); -- non-blocking
  procedure Read (Item : out Data); -- non-blocking
end Simpsons_Algorithm;

package body Simpsons_Algorithm is

  type Slot is (First, Second);

  Four_Slot : array (Slot, Slot) of Data :=
             (First => (Initial_Value,Initial_Value),
              Second => (Initial_Value,Initial_Value));

  Next_Slot : array(Slot) of Slot := (First, First);

  Latest : Slot := First;
  Reading : Slot := First;

  procedure Write(Item : Data) is
    Pair, Index : Slot;
  begin
    if Reading = First then
      Pair := Second;
    else
      Pair := First;
    end if;
    if Latest = First then
      Index := Second;
    else
      Index := First;
    end if;
```

```
    Four_Slot(Pair, Index) := Item;
    Next_Slot(Pair) := Index;
    Latest := Pair;
  end Write;

  procedure Read(Item : out Data) is
    Pair, Index : Slot;
  begin
    Pair := Latest;
    Reading := Pair;
    Index := Next_Slot(Pair);
    Item := Four_Slot(Pair, Index);
  end Read;
end Simpsons_Algorithm;
```

Describe carefully how this algorithm works by explaining what happens when:

1. a `Write` is followed by a `Read`
2. a `Read` is preempted by a `Write`
3. a `Write` is preempted by a `Read`
4. a `Read` is preempted by more than one `Write`
5. a `Write` is preempted by more than one `Read`

Comment how the algorithm trades-off safe communication against data freshness.

(ii) [7 marks] Many compilers optimise code so that frequently accessed variables are held in registers local to a task. In this context, comment on whether the above algorithm's claim of safe communication is justifiable.

Are there any changes to the algorithm needed to make it work for all implementations of Ada 95?

3 (25 marks)

(i) [8 marks] Define the equation for response-time analysis for pre-emptive priority-based systems. You should take into account blocking but ignore process jitter. You can assume that the deadline of a process is less than its period and that all processes are released at time 0.

(ii) [10 marks] A real-time systems designer wishes to run a mixture of safety-critical, mission-critical and non-critical periodic and sporadic Ada 95 tasks on the same processor. He/she is using pre-emptive priority-based scheduling and has used the response-time analysis equation to predict that all tasks meet their deadlines. Give reasons why the system might nevertheless fail to meet its deadlines at run-time.

(iii) [7 marks] What enhancements could be provided to the Ada 95 run-time support systems to help eliminate the problems identified in Part (ii).

4 (25 marks)

(i) [10 marks] Describe Anderson's and Lee's ideal fault-tolerant component for sequential systems.

(ii) [7 marks] How would you express an ideal fault-tolerant component for sequential systems in Ada 95?

(iii) [8 marks] A subset of Ada 95 has been defined for use in safety critical embedded systems. The language has formal semantics and a set of tools that allow static analysis techniques to be performed. It has no exception handling facility. The designers argue that exceptions cannot occur if the program has been proved correct. What arguments would you put forward for the introduction of exception-handling facilities into the language?

5 (25 marks)

(i) [10 marks] Describe the facilities provided by Modula-1 for the programming of device drivers.

Continued.

(ii) [15 marks] The British government is considering introducing charges for the use of motorways. One possible mechanism is that detector stations can be set up at regular intervals along all motorways; as a vehicle passes the detector station, its details are logged and a road charge is recorded. At the end of each month, the vehicle owner can be sent a bill for his/her motorway usage.

Each vehicle will require an interface device which interrupts an on-board computer when a detector station requests its details. The computer has a word size of 16 bits and has memory mapped I/O, with all I/O registers 16 bits in length. Interrupts are vectored, and the address of the interrupt vector associated with the detector station is 60B. After an interrupt has been received, a read-only input register located at 177750B contains the basic cost of using the current stretch of motorway (in pennies). The hardware priority of the interrupt is 4.

The computer software must respond to the interrupt within a period of 5 seconds, and pass its ownership details to the detector station via the interface device. It does this by writing to a bank of 9 control and status registers located at 177752B. The structure of these registers is shown in Table 1.

Register	Bits	Meaning
1	0-7	Vehicle Registration Character 1
2	0-7	Vehicle Registration Character 2
3	0-7	Vehicle Registration Character 3
4	0-7	Vehicle Registration Character 4
5	0-7	Vehicle Registration Character 5
6	0-7	Vehicle Registration Character 6
7	0-7	Vehicle Registration Character 5
8	0-7	Vehicle Registration Character 8
9	0	set to 1 to transmit data
9	1-4	travel details
		1 = business
		2 = pleasure
		3 = overseas tourist
		4 = police
		5 = military
		6 = emergency services
9	5-15	security code (0-2047)

Table 1: Register structure for road charging

The bank of CSR registers is write only.

Sketch a Modula-1 device module to interface with the detector station. The module should handle the interrupts from the interface device and be responsible for sending the correct vehicle details. It should also read the data register containing the cost of the road usage, and pass the current total cost of the journey to a task which will output it on a visual display unit on the vehicles dashboard. You may assume the following is available:

```
MODULE journey_details;

  DEFINE
    business, pleasure, overseas_tourist,
    police, military, emergency_service, travel_details
    registration_number, security_code,
    current_journey, current_security;
  CONST
    business  = 1;
    pleasure  = 2;
    overseas_tourist = 3;
    police = 4;
    military = 5;
    emergency_service = 6;
  TYPE
    string = array[1..8];
    security_code = 0 .. 2047;
    travel_details = 1 ..6; (* 1 = business, etc *)
  VAR
    registration_number : string := "........";

  FUNCTION current_journey : travel_details;
  BEGIN
    ...
  END;

  FUNCTION current_security : security_code;
  BEGIN
    ...
  END;

END journey_details;
```


Continued.

```
MODULE display_interface;
  DEFINE display_driver;

  PROCEDURE display_driver(c : integer);
  BEGIN
    (* prints the cost on the dashboard display *)
    ...
  END;
END display_interface;

MODULE bit_manipulation;
  DEFINE set_bits;

  PROCEDURE set_bits(VAR x: BITS; pos : integer; value : integer);
  BEGIN
    (* sets the bits in x starting at bit position pos  *)
    (* to value *)
    ...
  END;
END bit_manipulation;
```

You may assume that the compiler represents the type `Character` as an 8-bit value and the type `Integer` as a 16-bit value. You may also assume that the device and interrupts are always enabled.

6 (25 marks)

(i) [10 marks] Describe in detail the facilities provided by POSIX which give the same functionality as monitors.

(ii) [15 marks] Consider the scheduling of a queue of requests to transfer data to/from a disk. In order to minimise the disk head movement, all the requests for a particular cylinder are serviced in one go. The scheduler sweeps up the disk, servicing requests for each cylinder in turn. When it reaches the last cylinder, it returns to the first one immediately and starts the sweep again.

Write an Ada 95 package body for (Disk_Head_Scheduler) which provides the necessary synchronization using the POSIX facilities you have described in Part (i). DO NOT USE THE ADA SYNCHRONISATION FACILITIES. Assume that there are two interface procedures: one for requesting access to a particular cylinder, the other for releasing the cylinder.

```
package Disk_Head_Scheduler is
  Size_Of_Disk : constant := ...;
  type Disk_Address is range 1 .. Size_Of_Disk;
  type Cylinder_Address is mod 1 .. 20;
  procedure Request(Dest: Cylinder_Address);
  procedure Release(Dest: Cylinder_Address);
end Disk_Head_Scheduler
```

Users call procedures in the following disk driver package:

```
with Disk_Head_Scheduler; use Disk_Head_Scheduler;
package Disk_Driver is
  procedure Read(Data: out  Data_T; Addr : Disk_Address);
  procedure Write(Data: in  Data_T; Addr : Disk_Address);
end Disk_Driver;

package body Disk_Driver is

  procedure Read(Data: out  Data_T; Addr : Disk_Address) is
      Cylinder : Cylinder_Address;
  begin
      -- work out cylinder address
      Disk_Head_Scheduler.Request(Cylinder);
      -- read data
      Disk_Head_Scheduler.Release(Cylinder);
  end Read;

   -- similarly for Write

end  Disk_Driver;
```

Your solution should NOT contain any tasks or POSIX processes/threads.

You may use the following package which provides a simplified Ada 95 binding to POSIX Mutexes and Condition Variables. All Mutexes and Condition Variables are initialised with default attributes.

```
package Mutexes is
    type Mutex_T is limited private;
    type Cond_T is limited private;

    procedure Mutex_Initialise (M: in out Mutex_T);
    procedure Mutex_Lock(M: in out Mutex_T);
    procedure Mutex_Trylock(M: in out Mutex_T);
    procedure Mutex_Unlock(M: in out Mutex_T);
```


Continued.

```
    procedure Cond_Initialise(C: in out Cond_T);
    procedure Cond_Wait(C: in out Cond_T;
                          M : in out Mutex_T);

    procedure Cond_Signal(C: in out Cond_T);
    procedure Cond_Broadcast(C: in out Cond_T);
private
   ...
end Mutexes;
```