063174

THE UNIVERSITY of York

Degree Examinations

DEPARTMENT OF COMPUTER SCIENCE

Real-Time Systems and Programming Languages

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

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

Do not use red ink.

1 (25 marks)

(i) [8 marks] Explain how the facilities provided by POSIX mutexes and condition variables support the monitor communication abstraction.

(ii) [15 marks] The Pearl programming language supports a synchronization mechanism called a Bolt. A Bolt can be thought of as an object with an encapsulated state which can be "locked", "lock possible" and "lock not possible". Operations on a Bolt are performed by functions operating on the encapsulated state. Their semantics are defined in terms of two logical queues, a "reserve" queue and an "enter" queue. The following functions are available.

- `reserve` – If the state is "lock possible", then change the state to "locked" otherwise the calling thread is suspended and placed in the "reserve" queue associated with the Bolt object.
- `free` – If the state is "locked", set the state to "lock possible". Any threads waiting on the "reserve" queue are released and contend for the lock. If there are no threads waiting, any threads waiting on the "enter" queue are released and contend for the lock. If the state is not "locked", do nothing.
- `enter` – If the state of the Bolt is "locked" or there are threads waiting on the "reserve" queue, suspend the calling thread and place it on the "enter" queue. Otherwise, set the state to "lock not possible".
- `leave` – If the state is "lock not possible" and the number of completed calls to "enter" equals the number of calls to "leave", set the state to "lock possible". Any threads waiting on the reserve queue are released and contend for the lock. If there are no threads waiting, any threads waiting on the enter queue are released and contend for the lock. If the state is not "lock not possible" do nothing.

  If the completed calls to "enter" does not equal the number of calls to "leave", return from the `leave` method leaving the state unchanged.

The normal usage is that the functions are used in the following pairs: `reserve` and `free`; `enter` and `leave`.

Show how the Bolt object can be implemented in C with POSIX system calls using mutexes and condition variables. You may assume that threads are not killed or aborted.

(iii) [2 marks] Briefly describe how would you modify your solution to Part (ii) to ensure that

- a thread that calls `free` was the thread that called `reserve`, and
- a thread that calls `leave` had previously called `enter`.

2 (25 marks)

(i) [8 marks] Define the terms priority inversion, priority inheritance, immediate priority ceiling inheritance and deadline monotonic priority assignment.

(ii) [17 marks] Consider the following task set (all times are given in milliseconds):

Task	Computation Time	Period	Deadline
T1	7	40	18
T2	6	80	68
T3	6	90	72
T4	7	50	42
T5	8	25	11

Table 1: Task Characteristic in Milliseconds

There are three resources:A, B and C. The time required to access each resource is

- $A = 2$ ms
- $B = 3$ ms
- $C = 1$ ms

The tasks access the resources according to the following:

- Task $T1$ accesses A and B once each on each release.
- Task $T2$ accesses B once on each release.
- Task $T3$ accesses B and C once each on each release
- Task $T4$ accesses A and B once each on each release
- Task $T5$ accesses A and B and C once each on each release

You may assume that there are no nested resource accesses (that is, each task can only access one resource at a time).

1. Using Deadline Monotonic Priority Assignment, assign priorities to $T1$, $T2$, $T3$, $T4$ and $T5$. Use larger numbers to represent higher priorities. (3 marks)
2. Assuming simple priority inheritance, determine the blocking time of each task and compute its worst-case response time using response time analysis. Do any of the tasks miss their deadlines? (9 marks)
3. Assuming immediate priority ceiling inheritance, determine the blocking time of each task and compute its worst-case response time using response time analysis. Do any of the tasks miss their deadlines? (5 marks).

3 (25 marks)

(i) [7 marks] Illustrate how a Java method and an Ada subprogram can be coded so that a message can be printed when the method/subprogram returns. The same message should be printed irrespective of whether the return is a normal return or a return due to a propagating exception (you may ignore any asynchronous transfer of controls). Why are the approaches different?

(ii) [8 marks] Describe the facilities provided by a *recovery block* and explain how they relate to the four phases of dynamic software fault tolerance.

(iii) [10 marks] Assuming the following class is available, show how recovery blocks can be implemented in Java using exception handling.

```
class RecoveryCache {
  public RecoveryCache();
  public void save(); // save all current state
  public void restore(); // restore all current state
  public void discard(); // discard the saved state
}
```

4 (25 marks)

(i) [10 marks] Discuss the various models by which a language can represent an interrupt handler; be sure you include in your discussion the C, Ada 95, the RTSJ and Modula-1 models. How would you rank the models in terms of run-time efficiency?

(ii) [10 marks] A European 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; when 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.

Several registers are used for the interface to the detection station. The registers consists of

- a "control and status" register (CSR),
- an "input data buffer" register (IDBR), and
- an "output data buffer" register (ODBR)

The structure of the CSR register is shown in Table 2 and the register resides at octal address 8#177760#.

Bits	Meaning
0	enable device
1	when set, the value found in the IDBR is used as the current charging rate
2-5	not used
6	interrupt enable
7-11	not used
12	error bits (0 = no error)
13-15	not used

Table 2: Control register structure for road charging

Interrupts are vectored, and the address of the interrupt vector associated with the detector station is 8#60#. The hardware priority of the interrupt is 4. After an interrupt has been received, the read-only input register (IDBR, which is located at 8#177762#) contains the basic cost (in an integer number of Euros) of using the current stretch of motorway.

The computer software must respond to the interrupt and pass its ownership details to the detector station via the write-only output data buffer register located at 8#177764#. The ownership details can be read from a read-only register located at at 8#177766# (it is an integer value).

A dash board display informs the driver of the cost of using the motorway. It can be accessed by a register located at 8#177770#.

Write a Modula-1 device module to interface with the detector station. The handler should respond to interrupts from the interface device and be responsible for sending the correct vehicle details. It should also read the data register containing the current cost of the road usage and display the current **total** cost of the journey on the vehicle's dashboard.

(iii) [5 marks] During the prototyping of the system described in part (ii), the requirements were changed such that the cost of the journey had to be provided to another Modula module. Explain why it is not possible for the device module you have written for part (ii) to simply call a procedure in the other module giving the current cost. How could you make the information available?

5 (25 marks)

(i) [5 marks] Under what conditions might a system which has been predicted to meet all its timing requirements (via off-line schedulability analysis) still miss deadlines at run-time?

(ii) [8 marks] State the four conditions that must be detected in order to tolerate timing failures?

(iii) [10 marks] Evaluate the extent to which Ada provides adequate mechanisms to allow a program to detect and respond to each of the four conditions given in part (ii).

(iv) [2 marks] It is usual to assume that periodic tasks cannot be released more often than that dictated by their period (ignoring issues of release jitter). Is this assumption justifiable when periodic tasks are being programmed in Ada? Explain your answer.

6 (25 marks)

(i) [2 marks] Give two main reasons why supporting asynchronous transfer of control (ATC) in a real-time language is difficult.

(ii) [8 marks] Given your answer to part (i), explain why languages like Ada and Java (augmented by the RTSJ) provide an ATC mechanism.

(iii) [10 marks] Three tug-boats are towing a disabled tanker carrying toxic chemicals. Each tug-boat is connected to the tanker via a steel cable. A supervisor ship has an on-board computer which monitors the stress on the steel cables and sends control signals to the on-board computers on the three tugs (via wireless communication devices). If one of the cables is in danger of breaking, it is necessary to instruct all tug-boats to immediately release their cables.

The control software on the supervisor ship executes on a single processor and is written in Ada. An interrupt is generated when the stress on one of the cables is near breaking point. The Ada run-time support systems associates the name `Breaking_Limit` with this interrupt in the `Ada.Interrupt.Names` package. The control software contains the following package declaration for controlling the tug-boats:

```
package Tug_Control is
  subtype Tug_Id is Integer range 1 .. 3;
  procedure Compute_And_Send_Tug_Cable_Tension(Tid : Tug_Id);
    -- This procedure continually monitors the cable associated with
    -- tug Tid and sets the appropriate tension.
  procedure Send_Emergency_Release(Tid : Tug_Id);
    -- This procedure releases the cable associate with tug Tid.
end Tug_Control;
```

Each tug is represented in the control software by an Ada task. Show how the tug tasks can be implemented, using the above declaration, so that when the `Breaking_Limit` interrupt arrives, all three tasks immediately release their cables. Show the details of any other tasks or protected objects that are needed to implement the required functionality.

(iv) [5 marks] In some situations, having the individual tug tasks release their own cables will result in the tanker becoming unstable. How would you modify your solution so that the releasing of all three cables is done as a single operation at a high priority?