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

1 (25 marks)

(i) [15 marks] Describe the support that Real-Time POSIX provides for priority-based scheduling. You should consider the following:

- the various scheduling schemes including sporadic servers (10 marks)
- the role of priority in pthreads, mutexes and message queues (5 marks)

(ii) [10 marks] A POSIX application consists of three threads with the following characteristics:

Thread	Period (ms)	Computation Time (ms)	Priority
A	4	1	5
B	10	2	3
C	20	2	2

All three tasks share a critical instant at time 0.

To handle aperiodic activity, the program has a sporadic server thread with the following characteristics:

- Replenishment period = 5 milliseconds
- Budget = 2 milliseconds
- High priority = 4
- Low priority = 1

Assuming that

- aperiodic events arrive at times, 3, 4, 5 and 6 milliseconds after a critical instant, and
- to handle each event requires 2 milliseconds of execution time from the sporadic server,

illustrate the execution of the threads between time 0 and time 20 milliseconds. When showing the sporadic server executing, indicate at what priority level it is running.

2 (25 marks)

(i) [13 marks] Describe fully the protected type mechanism provided by the Ada language. You should include in your description the Ada requeue facility.

(ii) [12 marks] Events are bivalued state variables (*up* or *down*) which can be implemented in Ada. Ada tasks can *set* (assign to *up*), *reset* (assign to *down*), or *toggle* an event. Any tasks *waiting* for the event to become *up* (or *down*) are released by a call of *set* (or *reset*); *toggle* can also release *waiting* tasks.

The following package specification illustrates an interface to events:

```
package Events is

  type Event_State is (Up, Down);

  protected type Event(Initial: Event_State := Down) is
    procedure|function|entry Set;
       -- you decide if it is a procedure, function or entry
    procedure|function|entry Reset;
    procedure|function|entry Toggle;
    procedure|function|entry State return Event_State;
    procedure|function|entry Wait(S : Event_State);
  private
    -- you decide
  end Event;

end Events;
```

Events may be created using the protected type `Event`. An event is, by default, initially `Down`. Show how events can be implemented by

- indicating whether the visible operations should be procedures, functions or entries;
- giving details of the private part of the `Event` protected type specification; and
- sketching the body of the `Event` protected type.

3 (25 marks)

(i) [8 marks] Explain the thread interruption mechanism in standard Java.

(ii) [17 marks] The following implements a readers/writers protocol in Java

```
public class ReadersWriters
{
  private int readers = 0;
  private int waitingWriters = 0;
  private boolean writing = false;

  public synchronized void StartWrite()
        throws InterruptedException
  {
    while(readers > 0 || writing)
    {
      waitingWriters++;
      wait();
      waitingWriters--;
    }
    writing = true;
  }

  public synchronized void StopWrite()
  {
    writing = false;
    notifyAll();
  }

  public synchronized void StartRead()
        throws InterruptedException
  {
    while(writing || waitingWriters > 0) wait();
    readers++;
  }

  public synchronized void StopRead()
  {
    readers--;
    if(readers == 0) notifyAll();
  }
}
```

Explain carefully how this algorithm works (5 marks) and why it is NOT robust in the face of thread interruption (3 marks).

Modify the algorithm so that it is robust (9 marks).

4 (25 marks)

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

(ii) [15 marks] An embedded computer controls a pedestrian crossing. The crossing has traffic lights for controlling the movement of cars, a button for pedestrians to press when they wish to cross the road, an illuminated pedestrian figure and a bleeper for signaling to pedestrians that it is safe to cross. The required control algorithm is as follows:

- when the system is initialised, the traffic lights are set to green; the illuminated figure is set to red, the bleeper is off, and the button is enabled;
- when the button is pressed, the button is disabled;
- after 20 seconds, the traffic lights are turned to amber;
- after a further 10 seconds, the traffic lights are turned to red, the illuminated figure is turned to green and the bleeper is turned on;
- after a further 30 seconds, the bleeper is turned off, the illuminated figure is set to flashing green, and the traffic lights are set to flashing amber;
- after a further 10 seconds, the illuminated figure is set to red, the traffic lights are set to green and the button is enabled.

The embedded computer has the following 16-bit memory-mapped I/O registers:

- Button Control Register at address Octal 167444 of which bit 6, when set to 1, enables interrupts (an interrupt signaling that the button has been pressed). When bit 6 is cleared (set to 0), all but the last button press is ignored. The last button press is remembered and an interrupt is generated when bit 6 is again set to 1.
- Bleeper Control Register at address Octal 167446 of which bit 8, when set to 1, turns on the beeper. When bit 8 is cleared, the bleeper is turned off.
- Traffic Lights Data Buffer Register at address Octal 167450. The status of the traffic lights is determined by the following bit patterns:

Continued.

  - octal 0000 - all lights off
  - octal 0010 - traffic lights are red
  - octal 0014 - traffic lights are amber
  - octal 0020 - traffic lights are green
  - octal 0024 - traffic lights are flashing amber

- Illuminated Pedestrian Data Buffer Register at Octal location 167452. The status of the illuminated person is determined by the following bits patterns:
  - octal 0000 - the figure is not illuminated
  - octal 0010 - the figure is illuminated red
  - octal 0014 - the figure is illuminated green
  - octal 0020 - the figure is illuminated green and flashing

- Interrupts from button presses are vectored and are identified by `Ada.Interrupts.Names.Button_Interrupt` and occur at hardware priority 6.

Sketch the implementation of an Ada package for implementing the required control algorithm. You may assume the default Bit_Ordering is that bit X in the Ada program corresponds to Bit X in the hardware.

5 (25 marks)

(i) [5 marks] With fixed priority scheduling, deadline monotonic priority ordering (DMPO) is optimal if a set of constraints are met. Define these constraints.

(ii) [10 marks] Give the proof that DMPO is optimal.

(iii) [10 marks] The following table gives some of the parameters for three processes that satisfy the constraints of part (i). All time values are in milliseconds.

- Have these processes been scheduled in DMPO? (2 marks).
- Calculate what the missed computation time parameters are (C_A, C_B, C_C) (8 marks).

Process	Period	Dealine	Computation Time	Response Time
τ_A	20	20	C_A	16
τ_B	25	12	C_B	10
τ_C	50	50	C_C	38

Table 1: Process Set

6 (25 marks)

(i) [5 marks] Real-Time Java provides the following classes to support asynchronous events.

```
public class AsyncEvent
{
  public AsyncEvent();

  public synchronized void addHandler(AsyncEventHandler handler);
  public synchronized void removeHandler(AsyncEventHandler handler);
  public void setHandler(AsyncEventHandler handler);

  public void bindTo(java.lang.String happening);

  public ReleaseParameters createReleaseParameters();

  public synchronized void fire();
  public boolean handledBy(AsyncEventHandler target);
}

public abstract class AsyncEventHandler implements Schedulable
{
  public AsyncEventHandler();

  public AsyncEventHandler(SchedulingParameters scheduling,
         ReleaseParameters release, MemoryParameters memory,
         MemoryArea area, ProcessingGroupParameters group);
  ... // other constructors available

  ... // methods which implement the Schedulable interface

  protected final synchronized int getAndClearPendingFireCount();
  protected synchronized int getAndDecrementPendingFireCount();
  protected synchronized int getAndIncrementPendingFireCount();

  public abstract void handleAsyncEvent();

  public final void run();
}
```

```
public abstract class BoundAsyncEventHandler
       extends AsyncEventHandler
{
  public BoundAsyncEventHandler();
  ... // other constructors
}
```

Explain the semantic model behind these classes and how they can be used.

(ii) [10 marks] Real-Time Java provides the following classes and interface to support asynchronous transfer of control.

```
public class AsynchronouslyInterruptedException extends
              java.lang.InterruptedException
{
  public  AsynchronouslyInterruptedException();

  public synchronized boolean disable();
  public boolean doInterruptible (Interruptible logic);

  public synchronized boolean enable();
  public synchronized boolean fire();

  public boolean happened (boolean propagate);

  public static AsynchronouslyInterruptedException getGeneric();

  public boolean isEnabled();

  public void propagate();
}

public interface Interruptible
{

  public void interruptAction (
              AsynchronouslyInterruptedException exception);

  public void run (AsynchronouslyInterruptedException exception)
         throws AsynchronouslyInterruptedException;

}
```

Explain the semantics of the Real-Time Java model and illustrate how the class can be used.

(iii) [10 marks] Three tug-boats are towing a disabled tanker carrying toxic chemicals. Each tug-boat is connected to the tanker via an intelligent 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 Real-Time Java. An interrupt is generated when the stress on one of the cables is near breaking point. The Java Virtual Machine associates the string "BreakingLimit" with this interrupt. The control software contains the following class declaration for controlling the tug-boats:

```
import javax.realtime.*;
public class TugControl implements Interruptible
{
  public TugControl(int tid)
  {
     // tid is a Tug identifier
  }

  private void computeAndSendTugCableTension()
            throws AsynchronouslyInterruptedException
  {
    // computationally complex calculations
    // send communication to appropriate tug's on-board computer
  }

  private void sendEmergencyRelease()
  {
    // send a emergency command to the Tug to release the cable
  }

  public void run(AsynchronouslyInterruptedException exception)
            throws AsynchronouslyInterruptedException
  {
     computeAndSendTugCableTension();
  }

  public void interruptAction(
            AsynchronouslyInterruptedException exception)
  {
    sendEmergencyRelease();
  }
}
```

Each tug is represented in the control software by a Real-time Thread

```java
import javax.realtime.*;
public class TugThread extends RealtimeThread
{
  TugControl myTug;
  AsynchronouslyInterruptedException myAIE;

  public TugThread(TugControl tc,
         AsynchronouslyInterruptedException ae)
  {
    super();
    myTug = tc;
    myAIE = ae;
  }

  public void run()
  {
     // you fill in
  }
}
```

Show how the above declaration can be used so that when the interrupt arrives, all TugControl software immediately release their cables. As well as completing the run method in `TugThread`, you will need to sketch the implementation of an asynchronous event handler to handle the "Breaking Limit" interrupt, and show how this can be linked to the `TugThread` and `TugControl` software by a main method. You may ignore all issues of thread scheduling in your answer.