064202

THE UNIVERSITY of York

Degree Examinations 2004

DEPARTMENT OF COMPUTER SCIENCE

Concurrent and Real-Time Programming in Java

Time allowed: **One and one half (1.5) hours**

Candidates should answer **two** questions only.

An appendix is provided which lists the relevant Java and RTSJ classes needed for this examination.

1 (25 marks)

(i) [10 marks] Compare and contrast the facilities provided by monitors with the corresponding facilities provided by Java.

(ii) [10 marks] The Pearl programming language supports a synchronization mechanism called a Bolt. A Bolt can be thought of as a class and an encapsulated state which can be "locked", "lock possible" and "lock not possible". Operations on a Bolt are performed by methods declared in the class. The following methods are available. Their semantics are defined in terms of two logical queues, a "reserve" queue and an "enter" queue.

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

Show how the Bolt object can be implemented in standard Java. Note that the logical queues need not directly be represented in the Java code to achieve the required semantics.

(iii) [5 marks] 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) [10 marks] Describe the semantics of `RealtimeThread` with `PeriodicParameters`. In particular, define the semantics of `RealtimeThread.waitForNextPeriod()`, `RealtimeThread.schedulePeriodic()` and `RealtimeThread.deschedulePeriodic()`. Include in your description the behaviour when deadline misses and cost overruns occur.

(ii) [5 marks] A system consists of the following periodic real-time threads (all values are given in milliseconds).

thread	Period	Deadline	Computation Time (Cost)	Priority
Thread_a	20	10	6	3
Thread_b	30	11	4	2
Thread_c	12	12	2	1

Table 1: Thread Characteristics

Assuming the RTSJ Priority scheduler schedules FIFO within priority, draw a diagram illustrating the execution of the threads during a period 0 to 59 milliseconds. You may assume that all the threads are initially released at time 0 and that they execute for exactly their cost values. You may also assume that no cost overrun or deadline miss handlers have been installed. Indicate if any deadlines are missed.

(iii) [5 marks] Consider now the same real-time thread set, but suppose that Thread_c actually consumes the following:

- 1st release - 4 milliseconds
- 2nd release - 2 milliseconds
- 3rd release - 3 milliseconds
- 4th release - 1 milliseconds
- 5th release - 1 milliseconds

Assuming cost enforcement *has been* implemented, that no cost overrun handler has been installed, and that a deadline miss handler has been installed (with a priority of 4 and a cost value of 1), illustrate the execution of the threads during a period 0 to 59 milliseconds. Indicate if any deadlines are missed. You may

assume that if a thread misses its deadline, the handler considers it as a transient overload and ignores the problem by calling the schedulePeriodic method on the real-time thread c.

(iv) [5 marks] Consider now the same real-time thread set as given in part (iii). Explain carefully what will happen if (1) the deadline miss handler has a priority of 0, and (2) the deadline miss handler does not call the schedulePeriodic method.

3 (25 marks)

(i) [8 marks] Describe the factors that determine the memory area stack of a newly created schedulable object.

(ii) [10 marks] Consider the following class:

```
import javax.realtime.*;
public class RtThread extends RealtimeThread {

  ScopedMemory A;
  ScopedMemory B;
  ScopedMemory C;
  ScopedMemory D;

  Runnable r1 = new Runnable() {
    public void run() {
      HeapMemory.instance().enter(r5);
      B.enter(r2);
    }
  };

  Runnable r2 = new Runnable() {
    public void run() {
      C.enter(r3);
      ImmortalMemory.instance().enter(r6);
    }
  };

  Runnable r3 = new Runnable() {
    public void run() {
      D.enter(r4);
      try {
        A.executeInArea(r7);
      } catch (Exception e) { };
    }
  };
```

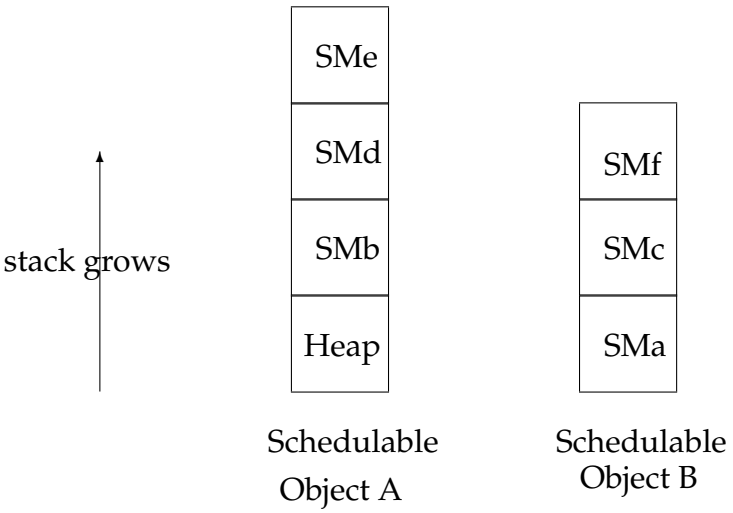
```
Runnable r4 = new Runnable() {
  public void run() {
    // Create a new Schedulable object W here with
    // immortal memory as the initial memory area.
    RealtimeThread W = new RealtimeThread(null, null, null,
                       ImmortalMemory.instance(), null, null);
  }
};

Runnable r5 = new Runnable() {
  public void run() {
    // Create a new Schedulable object X here with A
    // as the initial memory area.
    RealtimeThread X = new RealtimeThread(null, null, null,
                       A, null, null);
  }
};

Runnable r6 = new Runnable() {
  public void run() {
    // Create a new Schedulable object Y here with
    // heap as the initial memory area.
    RealtimeThread Y = new RealtimeThread(null, null, null,
                       HeapMemory.instance(), null, null);
  }
};

Runnable r7 = new Runnable() {
  public void run() {
    // Create a new Schedulable object Z here with B
    // as the initial memory area.
    RealtimeThread Z = new RealtimeThread(null, null, null,
                       B, null, null);
  }
};

public RtThread(SchedulingParameters sp, ReleaseParameters rp,
  MemoryParameters mp, MemoryArea ma, ProcessingGroupParameters pgp,
  ScopedMemory one, ScopedMemory two, ScopedMemory three,
  ScopedMemory four) {

  super(sp, rp, mp, ma, pgp, null);
  A = one;
  B = two;
  C = three;
  D = four;
}
```


Continued.

```
    public void run() {
      // Create a new schedulable object V here with the heap
      // as the initial memory area.
      RealtimeThread V = new RealtimeThread(null, null, null,
                              HeapMemory.instance(), null, null);
      A.enter(r1);
    }
  }
```

Assume that an instance of `rtThread` is created with the heap as the current memory area and with the heap as the memory parameter (that is, the initial memory area). Illustrate the memory stacks for the created real-time threads V, W, X, Y and Z.

(iii) [7 marks] Two schedulable objects A and B have the following memory area stacks (SMa - SMf are scoped memory areas).



Explain how these schedulable objects can communicate through a new scoped memory area SMg. You should consider how they manipulate their memory area stacks and what mechanisms they use once they are inside SMg to communicate.

064202

THE UNIVERSITY of York

DEPARTMENT OF COMPUTER SCIENCE

Degree Examinations 2004

Concurrent and Real-Time Programming in Java

Model Answers

Model Answers

Question 1 (25 marks)

Part (i) [10 marks]

Book work/analysis

A typical monitor provides: encapsulation of shared data, mutual exclusion and condition variables. However, there are variations on the actual locking approach (write and read locks for example) and on the semantics of the operations on the condition variable. (2 marks)

It can be argued that Java provides a simplified monitor within an object-oriented framework. (1 mark)

The similarities are (3 marks):

- encapsulation of shared data
- synchronized methods have to obtain a mutual exclusion lock
- support for nested calls

The differences are (4 marks):

- the lock can be acquired remotely from the monitor code (via the synchronized statement)
- methods which do not require the lock can be included
- no condition variable exits as such - the programmer can wait, notify, notifyAll on an anonymous condition variable. Consequently, there is no fine control over releasing threads waiting in the monitor. Threads must re-test their wait conditions.

Other issues that might obtain marks are: the order in which competing threads acquire access to the lock, real-time models such as priority inheritance, order of wait queues.

Part (ii) [10 marks]

Unseen problem.

```
public class Bolt {

  public Bolt() {
  }

  public synchronized void reserve() {
      while (state != LOCK_POSSIBLE) {
        waitingReserve++;
        try {
          wait();
        } catch(InterruptedException ie) {
        } finally {
          waitingReserve--;
        }
      };
      state = LOCKED;
  }

  public synchronized void free() {
       if(state == LOCKED) {
         state = LOCK_POSSIBLE;
         notifyAll();
      }
  }

  public synchronized void enter() {
    while (state == LOCKED | waitingReserve > 0) {
      try {
        wait();
      } catch (InterruptedException ie) {};
    }
    state = LOCK_NOT_POSSIBLE;
    entered++;
  }

  public synchronized void leave() {
    if(state == LOCK_NOT_POSSIBLE) {
      if (--entered == 0) {
        state = LOCK_POSSIBLE;
        notify();
      }
    }
  }
  private int entered = 0;
  private int waitingReserve = 0;
```

```
  private final int LOCKED = 1;
  private final int LOCK_POSSIBLE = 2;
  private final int LOCK_NOT_POSSIBLE = 3;
  private int state = LOCK_POSSIBLE;
}
```

Part (iii) [5 marks]

Analysis of unseen problem

Essentially, it is necessary to save a reference to the thread requesting calling `reserve` and `enter`. The reference can be found by calling `currentThread` in the `Thread` class. A check can then be made and an appropriate exception thrown.

Alternatively, ThreadLocal data could be used.

Question 2 (25 marks)

Part (i) [10 marks]

Book work on a intricate part of the specification.

The waitForNextPeriod (wFNP) method has the following semantics:

- When the deadlineMiss count is greater than zero and the previous call to wFNP returned true, wFNP decrements the deadlineMiss count and returns false immediately. This situation indicates that the current release has missed its deadline. At this point, the current release is still active (this means the current budget is unaltered).
- When the deadlineMiss count is greater than zero and the previous call to wFNP returned false, wFNP decrements the deadlineMiss count and returns false immediately. This situation indicates that the next release time has already passed and the next deadline has already been missed. At this point, the current release has completed and the next release is active (and a new budget of cost is allocated).
- When a deadline miss handler has been released and the deadlineMiss count equals zero and no call to the schedulePeriodic method has occurred since the deadline miss handler was released, wFNP deschedules the real-time thread until a call to the schedulePeriodic method occurs; wfNP then returns true at the point of the next release after the call to SchedulePeriodic. At this point, the next release is active (and a new budget of cost is allocated).
- When the deadlineMiss count equals zero and no deadline has been missed on the current release and the time for the next release has passed, wFNP returns true immediately. At this point, the next release is active (and a new budget of cost is allocated).
- When the deadlineMiss count equals zero and no deadline has been missed on the current release and the time for the next release has not passed, wFNP returns true at the next release time. At which point, the next release is active (and a new budget of cost is allocated).

A call to the schedulePeriodic method sets the deadlineMiss count to zero Note, the fireCount for the deadlineMiss handler indicates the number of deadline overruns that have occurred.

Model Answers

Part (ii) [5 marks]

Unseen problem

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
a	a	a	a	a	a	b	b	b	b	c	c	c	c						

20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
a	a	a	a	a	a	c	c			b	b	b	b			c	c		

40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
a	a	a	a	a	a			c	c										

No thread misses a deadline.

Part (iii) [5 marks]

Unseen problem.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
a	a	a	a	a	a	b	b	b	b	c1	c1	hc	c1	c1					

20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
a	a	a	a	hc	a	a	c2	c2		b	b	b	b			hc	c3	c3	

40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
a	a	a	a	a	a			hc	c4	c5									

Thread a misses its deadline on the first, second, third and fourth release.

Part (iv) [5 marks]

Analysis

1) Essentially, handler will not execute until there are no runnable threads.

2) Thread_C will not be rescheduled after its first release.

Question 3 (25 marks)

Part (i) [8 marks]

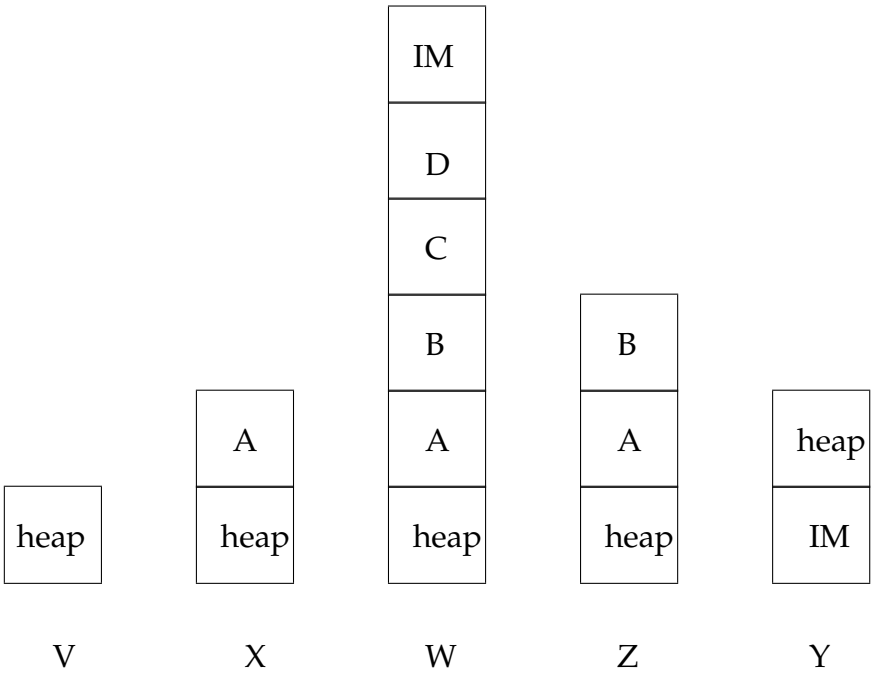
Book work

The stack of a created schedulable object is determined by the value of the initial memory area (the area parameter) and the currently active memory area

- If current area is the heap and
  - the initial memory area is the heap, the new stack contains only the heap memory area
  - the initial memory area is not the heap, the new stack contains the heap and the initial memory area
- If current area is the immortal memory area and
  - the initial memory area is the immortal memory area, the new stack contains only the immortal memory area
  - the initial memory area is not the immortal memory area, the new stack contains the immortal memory area and the initial memory area
- If the current area is a scoped memory area and the initial memory area is the currently active memory area (or null), the new stack is the parent's stack up to and including the current memory area
- If the current area is a scoped memory area and the initial memory area is not the currently active memory area, the new stack is the parent's stack up to and including the current memory area, plus the initial memory area.

Part (ii) [10 marks]

unseen problem



IM = immortal area

Part (iii) [7 marks]

Unseen problem

First the schedulable objects have to enter into SMg. However, this is not straightforward as if they both just enter they violate the single parent rule. Consequently, they should, call Heap.instance().executeInArea() with a runnable to get a new memory stack. (4 marks)

Once inside they will have to use the RTSJ portal facility. One will have to set the portal object and the other will have to read from it. Again, care is needed as the threads will need to synchronize (possibly on the memory area itself). (3 marks)