063174

THE UNIVERSITY of York

Degree Examinations 2003

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) [13 marks] Describe in detail the facilities provided by standard Java for communication and synchronization between threads. You may ignore any deprecated facility.

(ii) [12 marks] Consider the following class

```
public class LaunchClock extends youDecide
{
   public LaunchClock(int seconds);
   public synchronized void startCountDown();
   public synchronized void pauseCountDown();
   public synchronized void waitLaunchTime();
   // any other methods, fields etc that you need
}
```

An instance of the `LaunchClock` class is created with the number of seconds to the Space Shuttle launch. The count-down process is started by a call to the `startCountDown` method. The `LaunchClock` then counts down in units of one seconds. Threads, which wish to wait for the count down to reach zero, call the `waitLaunchTime` method. They are held in the method until the count down completes, at which point they are all released. At any time, before the countdown reaches zero, it can be paused by calling the `pauseCountDown` method. This results in the count down being stopped until it is started again (by a further call to the `startCountDown` method).

Show how the `LaunchClock` class can be implemented.

2 (25 marks)

(i) [12 marks] **Compare and contrast** the Ada and the Real-Time Java asynchronous transfer of control facilities.

(ii) [13 marks] Consider the following Ada example which uses asynchronous transfer of control:

```
E1, E2 : exception;
task Server is
  entry Atc_Event;
end Server;

task To_Be_Interrupted;

task body Server is
begin
  ...
  accept Atc_Event do
    Seq2; -- including raise E2;
  end Atc_Event;
  ...
end Server;

task body To_Be_Interrupted is
begin
  ...
  select
    Server.Atc_Event;
    Seq3;
  then abort
    Seq1; -- including raise E1;
  end select;
exception
  when E1 =>
    Seq4;
  when E2 =>
    Seq5;
  when others =>
    Seq6;
end To_Be_Interrupted;
```

where `Seq1` .. `Seq6` are sequences of statements.

Explain the possible outcomes of executing the above code.

3 (25 marks)

(i) [12 marks] Explain the facilities provided by Ada's select statement. Focus on the support provided for a server rather than a client task.

(ii) [13 marks] Consider the following Ada task specification:

```
task Server is
   entry Service_A;
   entry Service_B;
   entry Service_C;
 end Server;
```

Sketch the body of this task so that it implements the following algorithm.

- The task will accept calls to `Service_A` in preference to calls to `Service_B`.
- The task will accept calls to `Service_A` in preference to calls to `Service_C`.
- The task will accept calls to `Service_B` in preference to calls to `Service_C`.
- The task terminates when it has not received a call to `Service_A` or `Service_B` or `Service_C` within 10 seconds of the last time it serviced any call, and as long as it has serviced an equal number (non zero) of calls to `Service_A` and `Service_B` and `Service_C`.
- The task does NOT busy-wait.

4 (25 marks)

(i) [5 marks] Define the terms **local drift** and **cumulative drift**.

(ii) [20 marks] A real-time operating system provides the following interface

```
struct time_t
{
  long milli;
  int nanos;
};

void wallClock(struct time_t *now);
  // returns, in now, the number of milliseconds and nanoseconds
  // since midnight 1/1/1970

void sleep(const struct time_t forTime);
  // sleep for a minimumm of forTime milli and nanoseconds.

void sleepUntil(const struct time_t toTime);
  // sleep until toTime milli and nanoseconds past midnight 1/1/1970
```

The operating system guarantees that a thread issuing a `sleep` or `sleepUntil` system call will not be rescheduled before the time (given in the parameter) has passed.

1. (8 marks) A thread issues the following system call

   ```
   struct time_t sleepTime;

   sleepTime.milli = 53;
   sleepTime.nanos = 13;
   sleep(sleepTime);
   ```

   Assuming no thread interruption mechanism, explain the factors which determine when the thread is next executing (that is, actually running on the processor).

2. (12 marks) Using the same operating system interface, illustrate how the programmer can write a periodic thread. Ensure that you avoid cumulative drift. You may assume that the thread's period is measured in milliseconds.

5 (25 marks)

(i) [10 marks] **Compare and contrast** pre-emptive priority-based scheduling using Deadline Monotonic priority assignment with Earliest-Deadline-First scheduling.

(ii) [15 marks] A real-time operating system supports

- pre-emptive priority based dispatching of threads;
- priorities in the range 1 to 3 (3 being the highest priority);
- a facility for dynamically changing a thread's priority.

An applications programmer has a set of periodic independent non-blocking threads each of which starts immediately it is created and has a deadline. Illustrate how the programmer can use the dynamic priority facilities to achieve Earliest-Deadline-First scheduling. You may use the following system call interface:

```
typedef int threadId;

void setThreadPriority(threadId thread, int priority);
  // set the priority of a thread

void setPriority(int priority);
  // set the priority of the calling thread

threadId currentThread();
  // returns the thread Id of the calling thread

void waitForNextPeriod();
  // block the current thread until its next period is due.

long getDeadline(threadId id);
  // get the next absolute deadline of the current thread
```

You may assume that the operating system knows the details of a thread's period (and deadline) and that a thread does not receive an `InterruptedException` whilst waiting for its next period to arrive.

6 (25 marks)

(i) [12 marks] Both Ada and POSIX provide a monitor-like communication and synchronization mechanism. **Compare and contrast their facilities**.

(ii) [13 marks] Events are bivalued state variables (*up* or *down*) which can be implemented using POSIX mutexes and condition variables. POSIX threads can *set* (assign to *up*), *reset* (assign to *down*), or *toggle* an event. Any threads *waiting* for the event to become *up* (or *down*) are released by a call of *set* (or *reset*); *toggle* can also release *waiting* threads.

A particular Ada development environment supports only sequential Ada and provides a binding from Ada to the POSIX mutexes and condition variable facilities. You may assume the following package is available.

```
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);
    function Mutex_Trylock(M: in out Mutex_T) return Boolean;
    procedure Mutex_Unlock(M: in out Mutex_T);

    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
   ... -- not relevant for this question
end Mutexes;
```

When the initialisation procedures are called, the associated mutex or condition variable is initialised with default attributes (and are of no concern for this question).

The following Ada package implements the event abstraction:

```
with Mutexes; use Mutexes;
package Events is

  type Event_State is (Up, Down);
  type Event is limited private;

  procedure Initialise(E: in out Event; S:  Event_State);
  procedure Set(E: in out Event);
  procedure Reset(E: in out Event);
  procedure Toggle(E: in out Event);
  function State(E: Event) return Event_State;
  procedure Wait(E: in out Event; S : Event_State);

private
  type Event is You_Decide;
end Events;
```

Events may be created using the type `Event`. An event must be initialised before it can be used. Show how events can be implemented by

- completing the above package specification and
- sketching the body of the `Events` package.

Your solution should NOT use any of the Ada concurrency facilities (that is no Ada tasks, no protected objects etc.) or any POSIX processes/threads.