1 Interrupts

In the zip-archive
http://www.iste.uni-stuttgart.de/fileadmin/user_upload/iste/ps/Lehre/intern/RTP2012/a7/a7.zip
accessible through the internal university network you will find the following interface of
the package Interrupts and a test harness. Your task is to implement the subprograms
of the protected type Interrupt_Interface in Interrupts.

with Ada.Interrupts.Names;
package Interrupts is

protected type Interrupt_Interface (Int_ID : Ada.Interrupts.Interrupt_ID) is
  procedure Poll_Signal (Signal_Has_Arrived : out Boolean);
  -- Checks whether the interrupt Int_ID has occurred. The check is
  -- non-blocking. The interrupt is transient, meaning that Poll_Signal
  -- consumes the interrupt and a following Poll_Signal will not get
  -- any interrupt unless a new interrupt has occurred in the meantime.
  --
  -- Postconditions:
  -- (1) Signal_Has_Arrived = true iff the interrupt has occurred.
  -- (2) Interrupt_Occurred = false.

private
  procedure Handler;
  -- The interrupt handler that catches Int_ID.
  pragma Attach_Handler (Handler, Int_ID);

  Interrupt_Occurred : Boolean := False;
  -- True iff the interrupt has occurred.
end Interrupt_Interface;

end Interrupts;
Nota Bene: In the whole set of packages that come with the archive file you can obtain at the above address, the only package you have to look at is the package specification of **Interrupts**. You may also look at the other packages but this is not necessary to solve this exercise (you may learn, however, how to generate an interrupt on a Unix(-like) platform).

IMPORTANT NOTE: The code in the archive file will only run on a Unix(-like) system (it was tested on Solaris and Linux) where signalling via the command 'kill' is available. The code will not compile on a Windows machine. It may even depend on your compiler, as compilers are free to reserve interrupts (the example was successfully tested with GNAT Pro 6.2.2). However, even if you cannot compile the whole program, you should still be able to implement the body of **Interrupts** and provide your own test harness.

## 2 Questions

Are the following statements true or false?

a. Under transient overload of a system, hard deadlines are allowed to be missed occasionally under WCET assumptions as long as they are met under ACET (average CET) assumptions.

b. Any non-determinism of timing characteristics of hard real-time applications is to be avoided at all costs.

c. When a thread terminates with an unexpected and unhandled exception, this is a “fail uncontrolled” situation.

d. Local drift is the effect that a series of delays of equal duration in a task will eventually add up to much more than the sum of the delays, because the task cannot always be scheduled immediately to run when the delay expires.

e. In Ada, parent and guardian of a task are always identical.

f. To prove that a task set scheduled with rate-monotonic priority assignment meets all its deadlines, it is sufficient to show this property for the longest period only.

## 3 Basic Synchronisation in Java

The following code fragments show alternative implementations of coordinates in Java. As multiple threads may concurrently access objects of type `Coordinate`, synchronisation is needed to prevent race conditions. In Java, methods may be declared `synchronized`. If one thread invokes a synchronised method on an object, the lock of that object is first acquired, the method body is executed, and then the lock is released. Another thread invoking a synchronised method on that same object will be blocked until the lock is released.
a) Analyse each alternative and identify potential problems. Explain what could go wrong at runtime. Describe potential scenarios under which an error occurs.

b) Implement a thread-safe class for coordinates that solves those problems.
4 Condition Synchronisation in Java

The following code fragment shows a class `PrintBuffer` that stores documents to be printed. Concurrent users may add new documents to the buffer which are processed by several printers. If a printer wants to process a new document, it will fetch it via the `get` method. Insert the missing statements that guarantee that multiple concurrent users and printers can safely access `PrintBuffers` and that the conditions hold.

```java
class PrintBuffer {
    public void add(Document d) {
        // condition: next < N
        ...
        a[next] = d;
        next++;
        ...
    }

    public Document get() {
        Document d;
        // condition: next > 0
        ...
        next--;
        d = a[next];
        ...
        return d;
    }

    private static final int N = 10;
    private int next = 0;
    private Document[] a[] = new Document[N];
}
```

Hint: The `wait`-statement in Java interrupts the current thread until `notify` or `notifyAll` is invoked; `notify` awakens at most one thread, `notifyAll` notifies all waiting threads.