ECE 5780/6780 Lab 1

Due Friday, February 23, 2017 at 11:59 pm (200 points)

1 Objectives

To gain basic understanding of the Ada programming language and become familiar with real-time programming using Ada.

2 Logistics

You are to work in teams of 1 or 2 members. You and your teammate should be able to independently describe the solutions. There is no formal report to turn in. You will, however, be required upload your well-documented, well-structured code on Canvas.

3 Installing and Using Ada

The computers in EL 255 have two flavors of GNU's Ada compiler installed on them: gnat for Windows (including the GPS IDE) and gnat for Cygwin (Linux-like environment running on top of Windows).

You may also install GNU's Ada compiler gnat on your home computer (http://libre.adacore.com/libre/ for Windows or Linux and http://www.macada.org/macada/Welcome.html for Mac OS X).

When using gnat, all files should have the suffixes .ads or .adb, for specifications and bodies, respectively. Each file should contain exactly one compilation unit. A compilation unit is, for example, a package, a procedure or a function. The main procedure of your program should be a procedure which accepts no arguments and resides in a file with the same name as the procedure. Use lower case characters for file names. For instance, a "Hello World" program could be written as follows:

```
-- File: hello_word.adb
with Text_Io;
procedure Hello_World is
begin
     Text_Io.Put_Line("Hello World!");
end Hello_World;
```

To compile your program, type

```
$ gnatmake hello_world
```

To run it (from Linux / Cygwin), type

```
$ ./hello_world
```

There are many references for using Ada such as the documentation that installs with GNAT or by searching on Google.

4 Cyclic Scheduler (50 Points)

Write a cyclic scheduler to manage three procedures (F1, F2 and F3) which have the following characteristics:

- F1 is executed every second with an execution time of 0.3s
- F2, which executes for 0.15s, starts when F1 terminates
- F3 starts executing 0.5s after F1 starts and its execution time is 0.2s

You have been given a code skeleton that implements F1. The code skeleton actually provides quite a bit of information and hints. Try to understand it first. Compile the program and run it to observe the behavior of the system, then make necessary changes. Note that there is no need to create tasks for this part of the lab.

There are some requirements.

- Each procedure should print out an informative message when it starts executing and when it finishes. For example:

  ```
  F1 has started executing. The time is now: 1.00000
  ```

- The printout messages should use virtual time (see above) and not wall clock time

Note that some amount of jitter between the start time of F1 and that of F3 cannot be avoided. However, the start times should not drift further and further away from the schedule (refer to the lecture notes on Ada to see how to avoid this problem).

Useful references:

- Calendar package (http://www.adahome.com/rm95/rm9x-09-06.html)
- Printing time duration (http://www.adahome.com/rm95/rm9x-A-10-01.html)

5 Cyclic Scheduler with a Watchdog (50 Points)

As we discussed in class, some procedures may sometimes take longer to execute than previously estimated. Modify F3 so that it occasionally takes more than 0.5s to execute and hence miss its deadline.

Create a watchdog **task** to monitor F3's execution time. When F3 misses its deadline, the watchdog task should immediately print a warning message, i.e., 0.5s after F3 starts executing, either F3 has finished or

the watchdog has printed a warning message. The watchdog should let F3 finish even if it misses its deadline. The watchdog task should start at the same time as (or just before) F3 starts executing. When F3 misses its deadline, the scheduler should re-synchronize so that F1 starts executing again at whole seconds (e.g., at time 4.0000 instead of 3.67987).

Hints:

- Implement the watchdog as a separate task.
- The select, accept and delay statements may be useful when implementing the watchdog task.
- The package Ada.Numerics.Float Random is useful for creating random numbers. (https://www2.adacore.com/gap-static/GNAT_Book/html/aarm/AA-A-5-2.html)

6 Communication among Tasks (100 Points)

Create three tasks:

- A buffer task that acts as a FIFO buffer for integers. The task should block the producer if it is trying to put an integer to a full buffer and it should block the consumer if it is trying to remove an integer from an empty buffer. The buffer size is 10.
- A producer task that puts integers in the range [0,25] into the buffer at irregular intervals.
- A consumer task that removes integers out of the buffer at irregular intervals. When the sum of the removed integers is greater than or equal to 100, the consumer task should terminate the program. The information that the buffer and the producer should terminate should be spread using synchronization (no global variables, the producer task should not count, etc...). You are not allowed to use the abort nor terminate commands.

The buffer task should not print any messages, but the producer and consumer should print messages containing the integers that are added or removed from the buffer. **Do not implement semaphores or use the Semaphore Package.** Synchronization and mutual exclusion should be implemented by the buffer task. In addition, the producer should not throw away any number even if the buffer is full at the moment.

Hints:

- The accept and when statements might be useful for the buffer task.
- The package Ada.Numerics.Discrete Random is useful for creating integer random numbers. (https://www2.adacore.com/gap-static/GNAT_Book/html/aarm/AA-A-5-2.html)
- Another way of printing discrete types or subtypes is to use the Image attribute, which returns the corresponding string. For example, if D is of type Integer we can print D as follows:

```
Text_Io.Put(Integer'Image(D));
```

8 Rubrics

The grader will use the following criteria to grade your work.

1 Cyclic Scheduler (50 Points)

Task	Points
Code is well-structured and documented	5
F2 behaves as specified	10
F3 behaves as specified	20
Procedures print out messages as requested	5
Time does not continually drift	10
Total	50

2 Cyclic Scheduler with Watchdog (50 Points)

Task	Points
Code is well-structured and documented	5
F3 occasionally takes longer to execute	10
Watchdog prints a warning message when appropriate	20
Resynchronization working	15
Total	50

3 Communication Among Tasks

Task	Points
Code is well-structured and documented	5
Buffer working	25
Producer working	25
Consumer working	25
Program quits when sum is at least 100	15
Messages printed out as specified	5
Total	100