Concurrent and Real-Time Programming

Ada Assignment

Course 2022/2023

Faculty of Computer Science
University of Castilla-La Mancha

Summary

This document contains the specification, objectives, materialization and evaluation of the theoretical/practical work to be carried out by the student and which represents a non-obligatory and recoverable activity according to the teaching guide of the subject Concurrent Programming and Real Time. This activity, related to programming in Ada, has a weight of 15% of the final grade and is divided into two parts:

1. The **implementation of a series of basic exercises** related to the use of Ada in the context of Concurrent and Real-Time Programming, together with the elaboration of a report that briefly discusses the proposed solution and integrates the commented source code related to the solution of each of the exercises. This part can be done individually or in pairs. <u>If it is done in pairs</u>, the two members must deliver the document on Campus Virtual.
2. To take an individual test that will be dealt with in the same day as the ordinary (or extraordinary, as the case may be) calls. The resolution of the proposed exercises is a necessary condition to be eligible to take the test.

Objectives

Main objective

The general objective of this activity is to acquire the basic skills to carry out the development of programs in Ada within the field of Concurrent and Real-Time Programming.

Specific objectives

The student will have to acquire the necessary skills for the creation of programs in Ada oriented to manage in an adequate way the concurrence through the high level mechanisms, as the protected objects, integrated in the own language. On the other hand, the basic handling of real time packages is also pursued.

It will be necessary to implement functional solutions for the different issues raised using the programming language Ada.

Specification

The **list of exercises to be resolved** is set out below. Build the following Ada programs using the mechanisms integrated in your own language and show the results on the standard output.

In the **appendix** you can consult the elements used to solve some of the different exercises. This content corresponds to the <u>specification files (.ads)</u> that contain the declaration of these elements.

Exercises

Exercise 1

Build an Ada program named *main* with the following functionality:

- Show this message on the output: “`---- Inicio del programa main ----`”.
- Wait 5 seconds.
- Show this message on the output: “`---- Fin del programa main ----`”.

Notas:

- To model the waiting you may use the instruction: *`delay 5.0;`*
- To print the messages you may use the library **Ada.Text_IO** and, particularly, the functions
  - *Put_Line(“mensaje1” [& “mensaje2”] [& variables])*; o,
  - *Put(“mensaje1” [& “mensaje2”] [& variables]); New_Line;*

Exercise 2

Builds in Ada, inside a package called ***pkg_task***, a **task type** with the following characteristics:

- Type name: **tarea_t**.
- The task has an entry point to communicate with it, called **EsPar**. This entry point accepts an integer and shows on screen a message saying if it is even or not, increasing a counter of queried even numbers and another one of queried odd numbers according to the case:
  - "The number [N] is Even" or "The number [N] is odd".
- The task has another entry point, **EstadoConsultas**, without parameters, that will inform us when it is queried, the total number of odd and even numbers that up to that moment have been consulted. The output should be:
  - "The number of even numbers consulted is [NumParConsult]"
  - "The odd number of numbers consulted is [NumImparConsult]"

To test the task, create an instance of the task in the main procedure from the created package (reuse the code from the previous exercise).

In the main procedure creates an infinite loop that

- First gets the status of the queries of the instantiated task.
- Then collect a number entered by keyboard. To perform this function it is provided a package, pkg_procedure, with the procedure Leer_Entero(Num: out integer), which will show us the message "Escriba un número y pulse Enter: " and will return in Num the number written as integer.
- Then, if the number is not zero, check with the task if it is even. If the number is zero, execute the exit instruction to exit the loop.

When you finish the *main* program, abort the created task.

Notes

- To exit a loop in execution use the instruction **exit**;
- To abort a task T use the abort command: **abort T**;
- Try to understand the code of the package containing the procedure so that you can reuse it if you need it later.

Ejercicio 3

Build using Ada a task type which is periodic with the following characteristics:

- **Activation time:** 1 second.
- **Period:** 2000 milliseconds.
- **Execution time:** 1000 milliseconds.

For the modeling of the execution time you can use an **active waiting** that increases an integer variable during the indicated time. Initialize to zero the value of this variable in each cycle. To face this problem you must use the *Ada.Real_Time* library. Also, it will be necessary to declare this type inside a package.

To check that the task works correctly, show a message on the screen with the value of the integer variable at the end of each execution. In addition, the message must indicate which task is showing the message. To do this, use the package Ada.Task_Identification. This package allows us to use the *Current_Task* function, which returns an identifier of the task. To be able to represent this identifier on screen we must pass it to String by means of the *Image* function. An example could look like this:

```
Put_Line("Tarea("&Image(Current_Task)&"):Variable interna: "
                  & Integer'Image(variable_entera));
```

To test the task, create two separate instances of the task type designed in the *main* procedure from the package you created (reuse the code from the previous exercise). When you finish running the *main* procedure, which will simply wait 8 seconds, abort the two created taks. Also, wait another 2 seconds before showing the message at the end of the *main* procedure.

Notes

- To model the activation time wait a time equal to the activation time just before starting the task loop: ***delay To_Duration(ActivationTime);***
- To model the active waiting use a *while* loop. Just before the while loop, take the time instant using the *clock* function of the *Ada.Real_Time* package and store it in a variable (*InitEjecucion*). Then, in each iteration of the while loop compare the current instant (we use *clock* again) subtracting the *while* loop start time (this subtraction would give us the time that the loop has been running). The loop must be repeated as long as the time is less than the time that has been established as execution time (*ExecutionTime*).

Ejercicio 4

Model a **binary semaphore** in Ada using protected objects. Encapsulate the code in a package and define an interface in the package to use this semaphore, which can only be employed through this interface.

Ejercicio 5

Design, in Ada, a cyclyc executive for the tasks shown in the next table:

Task	T	D	C
A	25	25	10
B	25	25	8
C	50	50	5
D	50	50	4
E	100	100	2

It is necessary to define the system hyperperiod as well as the secondary period. Model the tasks in a package and use this package in the cyclic executive model. A message should be shown on the screen at the end of the execution of each of the tasks and another message indicating the end of each of the frames.

Notes

- Remember that, in the cyclic executive, tasks do not behave as processes but as sequentially executed procedures.

Exercise 6

Write in ADA a program that simulates the behavior of the solution of the problem of the **dining savages** discussed in class. The context of this problem lies in the scenario where savages from a tribe of cannibals spend all their time eating and dancing. To eat they have to collect their food ration from the same pot, which can hold a maximum of N (define it as a constant value) food rations. When a cannibal wants to eat, an exclusive ration of the common pot has to be served.

In case the pot is empty and does not contain any ration, the tribal cook should be notified and wait until he fills the pot with N new portions. The activity of dancing is an activity that takes a random amount of time. The cook, when he does not have to fill the pot, spends his time sleeping and waiting for new notifications.

For the implementation, each cannibal will be a task and the cook will be another task. Finally, the pot must be implemented as a shared object. You must assess the need to use some kind of semaphore and, if necessary, you can reuse the code from exercise 5 in which the implementation of a semaphore through protected objects was requested.

***Note**: You can use as a reference the implementation given in this web.

Exercise 7

Consider a **banking context** in which several customers can hold an account. Each account has a certain balance and one or more clients can operate on it simultaneously through one of the following operations: a) balance query, b) money withdrawal, c) deposit and d) transfer to another bank account. The balance query does not imply any modification of the balance of an account, which means that several of the holders may invoke the said operation without the need for mutual exclusion.

On the other hand, extraction and deposit operations do imply a modification of the balance and, therefore, must be carried out exclusively. In other words, when any of these operations are carried out, the rest must wait in order to avoid possible inconsistencies. In addition, in the extraction of money the restriction must be satisfied that the balance must be equal to or greater than the amount you wish to obtain. To this end, the use of conditional barriers is recommended.

Finally, a transfer between two bank accounts involves changing the balance in both the source and target accounts. In the case of the source account, the balance is decreased according to the amount transferred. On the other hand, this same amount will produce an increase in the destination account.

The coding of a program is requested in the programming language Ada, where the TAD *bank account* is modelled by means of a protected object that encapsulates the operations mentioned above. The program must simulate several clients (implemented as tasks) operating simultaneously on one or several accounts, such that the execution result shows that the restrictions established for each of the operations are satisfied. Finally, the bank transfer operation involves the interaction between two protected objects; in your solution, assess the possibility of either managing the transfer from the bank account itself, or designing an additional management entity that will be in charge of dealing with both accounts without direct contact between the two.

Delivery

The report associated with this activity must be delivered no later than **May 22, 2023 at 7.30 p.m.** through the corresponding task that will be enabled on Campus Virtual. Reports delivered after the deadline will not be taken into account. Reports delivered by e-mail will not be accepted. The report must be delivered exclusively in electronic format (it is not necessary to print it for delivery).

The practice report should include a brief discussion of the solution proposed for each exercise (no more than 1 page), with special emphasis on those parts of the source code that are more relevant. Also, the report should include all the source code and a section briefly indicating how to compile and execute.

Each student should upload a single compressed file whose name will consist of <surname_name>.tar.gz (example: perez_luis.tar.gz). If the work is done in pairs, each of the members has to make the delivery, specifying on the cover page the name of the two members. When decompressing the file, the following structure of directories/archives must be obtained:

README.txt	It will contain the students data
report	Report in PDF
src	Source code of the solutions (one subdir per exercise).

Assessment

Specifically, attention is drawn to the need for the work to be original and for no evidence of plagiarism to be detected, something that will be checked with the tools available at UCLM. In the event that it is detected that this premise has been infringed, article 8 of the "UCLM Student Assessment Regulations" will be applied and, as stated in point 6, "will lead to a numerical grade of zero (0) both in the tests and in the subject in which it has been detected, regardless of the rest of the grades that the student has obtained. This consequence must be understood as independent of the disciplinary responsibilities that the student may incur".

The report provided and the degree to which the skills have been acquired by taking the test will be assessed, and the questions will be related to the practices carried out previously, on the date of the regular and/or extraordinary calls.

In this test, **carried out individually**, it will not be possible to use the report with the solutions of the previously resolved exercises.

Bibliography

- [BURNS03] Burns, A., Wellings, A., Sistemas en Tiempo Real y Lenguajes de Programación (3ª Edición), Addison-Wesley 2003.

Appendix – Files .ads / .adb

Exercise 2 – pkg_tarea.ads

```
package pkg_tarea is
   task type tarea_t is
       entry EsPar(N: integer);
       entry EstadoConsultas;
    end tarea_t;
end pkg_tarea;
```

Exercise 2 – pkg_procedure.ads

```
package pkg_procedure is
      procedure Leer_Entero(Num : out Integer);
end pkg_procedure;
```

Exercise 2 – pkg_procedure.adb

```
with Ada.Text_IO;
with Ada.Characters.Handling;
package body pkg_procedure is
      procedure Leer_Entero(Num : out Integer) is
      -- Lee de entrada un número entero no dejando al usuario escribir
      -- caracteres que no sean dígitos.
      -- Para Linux cambiar por ASCII.LF sino ASCII.CR
      Intro : constant Character := ASCII.LF;
      -- Para Linux cambiar por ASCII.Del sino ASCII.BS
      Back : constant Character := ASCII.Del;

      Char : Character;
      Fin : Boolean := False;
      Numero : Natural := 0;

     -- Cadena para leer el número carácter a carácter
     -- El máximo de caracteres es Integer'Width-1 porque no leemos signo
      Cadena_Numero : String (1 .. Integer'Width – 1);
```

```
        begin
        Ada.Text_IO.New_line;
        Ada.Text_IO.Put ("Escriba un número y pulse Enter: ");
        while not Fin loop
              Ada.Text_IO.Get_Immediate (Char);
              if Ada.Characters.Handling.Is_Digit (Char) then
                 Numero := Numero + 1;
                 Cadena_Numero(Numero) := Char;
                 Ada.Text_IO.Put (Char);
              elsif Char = Intro then
                 Fin := True;
              elsif Numero>0 and Char = Back then
                 -- Si el usuario ha pulsado la tecla backspace
                 -- borra el dígito escrito anteriormente
                 Ada.Text_IO.Put (ASCII.BS & ' ' & ASCII.BS);
                 Numero:=Numero-1;
              end if;
        end loop;
        Numero := Integer'Value (Cadena_Numero (1 .. Numero));
        Ada.Text_IO.New_line;
        Num := Numero;
        exception
          when Constraint_Error =>
            Ada.Text_IO.New_line;
            Ada.Text_IO.Put_Line ("Lo siento: " & Cadena_Numero
                          & " es demasiado largo para almacenarse");
            Num := 0;
        end Leer_Entero;
  end pkg_procedure;
```

Exercise 3 – pkg_tarea.ads

```
package pkg_tarea is
   task type tarea_periodica_t;
end pkg_tarea;
```

Exercise 4 – pkg_sem.ads

```
package pkg_sem is
   type sem_t is limited private;
   procedure signal(sem : in out sem_t);
   procedure wait(sem : in out sem_t);
private
   protected type sem_t(valor_inicial : integer := 0) is
      entry wait;
      procedure signal;
   private
      valor_sem : integer := valor_inicial;
   end sem_t;
end pkg_sem;
```

Exercise 5 – pkg_tarea.ads

```
with Ada.Real_Time; use Ada.Real_Time;
with Ada.Text_IO; use Ada.Text_IO;
package tareas is
    procedure A;
    procedure B;
    procedure C;
    procedure D;
    procedure E;
end tareas;
```