ADA REPORT

Elena Martín Cantero, Marta Molina Gutiérrez

17 DE MAYO DE 2021

ESCUELA SUPERIOR DE INFORMÁTICA

Concurrent and Real-Time Programming

INDEX

COMPILATION AND EXECUTION

To compile all the exercises, we have used the command “gnatmake <ex0X.adb>”. And in order to execute them, the command “./<ex0X>”, except for the “Exercise 04” which cannot be compile nor execute.

EXERCISE 01

Explanation:

In this exercise we have to print one statement (“---- Inicio del programa main ----”) wait 5 seconds and then print a second statement (“---- Fin del programa main ----”). For waiting that time, we have used the instruction “delay”, so we need to import the package “Text_IO”.

Code:

File – ex01.adb

```
with Text_IO; use Text_IO;
procedure ex01 is
begin
   Put_Line("---- Inicio del programa main ---- ");
   delay 5.0;
   Put_Line("---- Fin del programa main ---- ");
end ex01;
```

EXERCISE 02

Explanation:

In this exercise the user enters a number. The program must check if this number is even or odd (and print it), and records the quantity of even and odds numbers that have been inserted. It has to print this information.

To do this, we have imported the packages that were given, and then, we made the file "pkg_tarea.adb", which contains the method that checks if the number is even or odd ("EsPar"), as well as, the method that shows how many odd and even numbers has been consulted ("EstadoConsultas"). To know if the number is even or odd, we have used the operation "rem" which gives us the remainder.

Finally, we have created the file "ex02.adb", which executes first the method "EstadoConsultas" and "Leer Entero". If the number inserted is 0, we abort the task and exit the program. If not, we call the method "EsPar" and the program continues as usual.

Code:

File – ex02.adb

```
with pkg_tarea; use pkg_tarea;
with pkg_procedure; use pkg_procedure;

procedure ex02 is
   T : tarea_t;
   Number : Integer;
begin
   loop
      T.EstadoConsultas;
      Leer_Entero(Number);
      if Number/=0 then
         T.EsPar(Number);
      else
         abort T;
         exit;
      end if;
   end loop;
end ex02;
```

File – pkg_tarea.adb

```
with Text_IO; use Text_IO;
with ADa.Integer_Text_IO; use Ada.Integer_Text_IO;
package body pkg_tarea is
   task body tarea_t is
      NumParConsult: Integer := 0;
      NumImparConsult: Integer := 0;
   begin
      loop
         select
```

```
            accept EsPar (N : Integer) do
               if N rem 2 = 0 then
                  Text_IO.Put("The number "); Ada.Integer_Text_IO.Put(N); Text_IO.Put(" is Even");
Text_IO.New_Line;
                  NumParConsult:=NumParConsult+1;
               else
                  Text_IO.Put("The number "); Ada.Integer_Text_IO.Put(N); Text_IO.Put(" is Odd");
Text_IO.New_Line;
                  NumImparConsult:=NumImparConsult+1;
               end if;
            end EsPar;
         or
            accept EstadoConsultas do
               Text_IO.Put("The number of even numbers consulted is ");
Ada.Integer_Text_IO.Put(NumParConsult);  Text_IO.New_Line;
               Text_IO.Put("The number of odd numbers consulted is ");
Ada.Integer_Text_IO.Put(NumImparConsult);  Text_IO.New_Line;
            end EstadoConsultas;
         end select;
      end loop;
   end tarea_t;
end pkg_tarea;
```

EXERCISE 03

Explanation:

In this exercise we have to build a task, with the parameters given. To do this we have implemented the file “pkg_tarea.adb” which contains the funcionality and variables of a task. Basically, we define the variables and we initializate them with the values given. Then we wait the “ActivationTime” and we start the loop. Wa have the variable “InVar” and then we set “InitEjecucion” to clock (the time instance). While the execution time of the task is lower than the “ExecutionTime”, we increase “InVar”. When the task ends, we print the message. Then we wait until the period is finished to start again.

Finally, we impleted “ex03.adb”, which aborts the two tasks 8 seconds later, and 2 seconds later, shows the final message.

Code:

File – ex03.adb

```
with Text_IO; use Text_IO;
with pkg_tarea; use pkg_tarea;
with Ada.Real_Time; use Ada.Real_Time;
procedure ex03 is
   Task1 : tarea_periodica_t;
   Task2 : tarea_periodica_t;
   wait_time : Time_Span:= Milliseconds(8000);
   finish_wait_time : Time_Span:= Milliseconds(2000);
begin
   Put_line("Program main begins");
   delay To_Duration(wait_time);
   abort Task1;
   abort Task2;
   delay To_Duration(finish_wait_time);
   Put_line("Program main ends");

end ex03;
```

File – pkg_tarea.adb

```
with Text_IO; use Text_IO;
with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;
with Ada.Real_Time; use Ada.Real_Time;
with Ada.Task_Identification; use Ada.Task_Identification;
package body pkg_tarea is
   task body tarea_periodica_t is
      InitEjecucion: Time;
      ExecutionTime: Time_Span := Milliseconds(1000);
      Period: Time_Span := Milliseconds(2000);
      ActivationTime: Time_Span := Milliseconds(1000);
      InVar: Integer;
   begin
      delay To_Duration(ActivationTime);
```

```
      loop
         InVar:= 0;
         InitEjecucion := clock;
         while (clock-InitEjecucion) < ExecutionTime loop
            InVar:= InVar+1;
         end loop;
         Put_Line("Tarea("&Image(Current_Task)&"):Variable interna: " & Integer'Image(InVar));
         delay To_Duration(Period-ExecutionTime);
      end loop;
   end tarea_periodica_t;
end pkg_tarea;
```

EXERCISE 04

Explanation:

In this exercise we have to implement a binary semaphore. We have implemented the file "pkg_sem.adb" with the procedure "signal" and the procedure "wait". When we call "wait" the value of the semaphore is 0, in order to block the process, and if we call "signal" the value is 1, so the process will be unblock.

Code:

File – pkg_sem.adb

```
package body pkg_sem is
   procedure signal(sem : in out semaphore) is
   begin
      sem.signal;
   end signal;

   procedure wait(sem : in out semaphore) is
   begin
      sem.wait;
   end wait;

   protected body sem_t is
      entry wait when valor_sem = 1 is
      begin
         valor_sem := 0;
      end wait;

      procedure signal is
      begin
         valor_sem := 1;
      end signal;


   end sem_t;
end pkg_sem;
```

EXERCISE 05

Explanation:

In this exercise we have to create 5 different instances of a task. So first of all we have to define the hyperperiod, which is the m.c.m of the periods given, so it is 100. And then we have to define the secondary period, which is the minimun of the periods given, so it is 25.

In order to do this, we first implement the "pkg_tarea.adb", which initializates the tasks. We have created 5 different procedures to init each task ("startTaskX").

Then, to make the cyclic executive, we have implemented the file "ex05.adb". Here, we declare the variable "frame" and the variable "nFrames" which is the number of frames (4). In order to decide which process went in each frame, we have made:

- Tasks A and B have a period of 25, so in the hyperperiod, they will be executed 4 times.
- Tasks C and D have a period of 50, so in the hyperperiod, they will be executed 2 times.
- Task E has a period of 100, so in the hyperperiod, it will be executed 1 time.

Then we take the clock time and we begin the loop which contains a switch. In this switch, we say at which frame we are, and executes the tasks we decided for each frame (these task must fit in our frame – Σ executions times <= secondary period). Then we wait until the frame ends and we begin the loop again.

Code:

File – ex05.adb

```
with Text_IO; use Text_IO;
with pkg_tarea; use pkg_tarea;
with Ada.Real_Time; use Ada.Real_Time;
with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;

procedure ex05 is
   subtype sw is Integer range 0 .. 4;
   frame : sw;
   nFrame : Integer;
   next : Time;
begin
   frame := 0;
   nFrame := 4;
   next := clock;
   loop
      frame := (frame rem nFrame) + 1;
      case frame is
         when 0 => exit;
         when 1 => startTaskA; startTaskB; startTaskC;
         when 2 => startTaskA; startTaskB; startTaskD; startTaskE;
         when 3 => startTaskA; startTaskB; startTaskC;
         when 4 => startTaskA; startTaskB; startTaskD;
      end case;
      Text_IO.Put("Marco "); Ada.Integer_Text_IO.Put(frame); Text_IO.Put(" terminado");
Text_IO.New_Line;
```

```
      next := next + Milliseconds(25000)
      delay To_Duration(next - clock);

   end loop;
end ex05;
```

File – pkg_tarea.adb

```
with Text_IO; use Text_IO;

package body pkg_tarea is
   procedure startTaskA is
   begin
                  delay 10.0;
                  Put_Line("Tarea: A terminada");
         end startTaskA;

   procedure startTaskB is
   begin
                           delay 8.0;
                           Put_Line("Tarea: B terminada");
         end startTaskB;

   procedure startTaskC is
   begin
                           delay 5.0;
                           Put_Line("Tarea: C terminada");
         end startTaskC;

   procedure startTaskD is
   begin
                           delay 4.0;
                           Put_Line("Tarea: D terminada");
         end startTaskD;

   procedure startTaskE is
   begin
                           delay 2.0;
                           Put_Line("Tarea: E terminada");
         end startTaskE;

begin
   null;
end pkg_tarea;
```

File – pkg_tarea.ads

```
with Text_IO; use Text_IO;
package pkg_tarea is
         procedure startTaskA;
```

```
   procedure startTaskB;
   procedure startTaskC;
   procedure startTaskD;
   procedure startTaskE;
end pkg_tarea;
```

EXERCISE 06

Explanation:

In this exercise we have to implement a solution for the problem of the dining savages. Here we are going to use the semaphore from “Exercise 04”. We use the semaphore as follows:

SEMAPHORE	DESCRIPTION	INIT VALUE
MUTEX	This semaphore is used in order to control that the more than one cannbial, do not access to the critical section.	1
FULL	The cooker notifies the cannibals that the pot is full.	0
EMPTY	The cannibals notify the cooker that the pot is empty.	0

To solve this problem we have implemented the file “ex06.adb”, “pkg_tarea.adb” and “pkg_tarea.ads”. The “pkg_tarea.adb” contains the function of the cannibals (eat and dance), the pot (is empty or full) and the cooker (sleeps and cooks). The “ex06.adb” file creates the tasks and control with the semaphores the syncronization.

Code:

File – ex06.adb

```
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;
with pkg_sem; use pkg_sem;
with pkg_tarea; use pkg_tarea;

procedure ex06 is
   NCan: Integer;
   NPot: Integer;
begin
   Put("Cannibals: "); Get(NCan); New_Line;
   Put("Pot's capacity: "); Get(NPot); New_Line;
   declare
      type cannibal is access task_cannibal;
      type cook is access task_cook;
      P: potType;
      Can: cannibal;
      CK: cook;
      I: Character := 'A';
      full: semaphore;
      empty: semaphore;
      mutex: semaphore;
   begin
   full:= new sem_t;
   empty:= new sem_t;
   mutex:= new sem_t(1);
   P:= new pot(NPot);
   for j in 1..NCan loop
      Can:= new task_cannibal(I, full, empty, mutex, P);
      I:=Character'Succ(I);
```

```
   end loop;
      CK:= new task_cook(full, empty, mutex, P);
   end;
end ex06;
```

File – pkg_tarea.adb

```
with pkg_sem; use pkg_sem;
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Numerics.Discrete_Random;
package body pkg_tarea is

         type range_dance is range 1 .. 20;
         package random_dance_time is new Ada.Numerics.Discrete_Random(range_dance);
         gen: random_dance_time.Generator;

   protected body pot is

      function Read return Integer is
      begin
         return food;
      end Read;

      procedure Decrease is
      begin
         food := food - 1;
      end Decrease;

      procedure Fill is
      begin
         food := InitialValue;
      end Fill;

   end pot;

   task body task_cannibal is
      procedure Eat is
      begin
         mutex.wait;
         if Pot.Read = 0 then
            empty.signal;
            full.wait;
            Pot.Fill;
         end if;
         Put_Line("Cannibal "& ID &" is eating");
         Pot.Decrease;
         mutex.signal;
         delay 1.5;
```

```
      end Eat;

      procedure Dance is
         begin
            Put_Line("Cannibal "& ID &" is dancing");
            delay Duration (random_dance_time.Random(gen));
        end Dance;

   begin
      loop
         Dance; Eat;
      end loop;
   end task_cannibal;

   task body task_cook is
      procedure Sleep is
      begin
      Put_Line("Cook is sleeping");
      empty.wait;
      Put_Line("Cook wakes up");
      end Sleep;

      procedure Cook is
      begin
      Put_Line("Cook is cooking");
      delay 3.0;
      full.signal;
      Put_Line("Cook has finished cooking");
      end Cook;
   begin
      loop
         Sleep; Cook;
      end loop;
   end task_cook;

end pkg_tarea;
```

File – pkg_tarea.ads

```
with pkg_sem; use pkg_sem;
package pkg_tarea is
   protected type pot (InitialValue:Integer) is
      function Read return Integer;
      procedure Decrease;
      procedure Fill;
   private
      food: Integer := InitialValue;
   end pot;
```

```
   type potType is access pot;

  task type task_cannibal(ID: Character; full: semaphore; empty: semaphore; mutex:
semaphore; Pot: potType);
   task type task_cook(full: semaphore; empty: semaphore; mutex: semaphore; Pot: potType);
end pkg_tarea;
```

EXERCISE 07

Explanation:

In this exercise we have to simulate a bank. First we have implemented the package "pkg_bank_account.adb" which contains the procedures that can be perfomed in our system. In the file "ex07.adb" we have clients (people) which are implented as tasks, and these tasks will simulate a performance in our system.

As the bank account is a protected object, we have to implement some methods that call the protected procedures. The money that get inserted or extracted is chosen randomly (with Ada.Numerics.Discrete_Random).

We have also implemented the file "pkg_bank_account.ads" which defines the "pkg_bank_account.adb" package.

Code:

File – ex07.adb

```
with Ada.Text_IO; use Ada.Text_IO;
with pkg_bank_account; use pkg_bank_account;
with Ada.Numerics.Discrete_Random;
with Ada.Task_Identification; use Ada.Task_Identification;

procedure ex07 is


   type P_Account is access bank_account;

   Bank_Accounts: array (1 .. 5) of P_Account;
   subtype Amount_Range is  Integer range 1 .. 1000;
   package Amount_Random is new Ada.Numerics.Discrete_Random (Amount_Range);
   money : Amount_Random.Generator;

   subtype Num_Range is  Integer range 1 .. 5;
   package Num_Random is new Ada.Numerics.Discrete_Random (Num_Range);
   num : Num_Random.Generator;

   task type TPerson (MyAcc : P_Account);

   task body TPerson is
                  procedure balance_query is
                  begin
         balance_query(MyAcc.all);
                  end balance_query;

                  procedure withdraw is
                  begin

                           withdrawal(MyAcc.all, Amount_Random.Random(money));
```

```
                end withdraw;

        procedure deposit is
                begin

                        deposit(MyAcc.all, Amount_Random.Random(money));

                end deposit;

        procedure transfer (bank : P_Account) is
        begin
          transfer(MyAcc.all, bank.all, Amount_Random.Random(money));
        end transfer;

        begin
          Put_Line("[Person " & Image(Current_Task) & "] does: ");
                        balance_query;
                        delay 2.0;
                        withdraw;
                        delay 3.0;
                        deposit;
                        delay 1.0;
          transfer (Bank_Accounts(Num_Random.Random(num)));

    end TPerson;
begin
    declare
        type Person is access TPerson;

                P1: Person;
        P2: Person;
        P3: Person;
        Amount: access Integer := new Integer'(1000);
    begin

        for i in 1 .. 5 loop
                        Bank_Accounts(i) := new bank_account(Amount, i);
        end loop;
        P1 := new TPerson (Bank_Accounts (2));
        delay 2.0;
        P2 := new TPerson (Bank_Accounts (4));
        delay 2.0;
        P3 := new TPerson (Bank_Accounts (1));
    end;
end ex07;
```

File – pkg_bank_account.adb

```
with Text_IO; use Text_IO;
with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;

package body pkg_bank_account is
   result : Integer := 0;
   procedure balance_query (B: in out bank_account) is
   begin
      B.balance_query;
   end balance_query;

   procedure deposit (B: in out bank_account; money: in Integer) is
   begin

      B.deposit(money);
   end deposit;

   procedure withdrawal (B : in out bank_account; money: in Integer) is
   begin
      B.withdrawal(money, result);
   end withdrawal;

   procedure transfer (BSrc: in out bank_account; BDest : in out bank_account; money: in
Integer) is
   begin
      Text_IO.Put_Line("---- Init transfer ----");
      BSrc.withdrawal(money, result);
      if result /= -1 then
         BDest.deposit(money);
      end if;
      Text_IO.Put_Line("---- End transfer ----");
   end transfer;

   protected body bank_account is
      procedure balance_query is
      begin
         Text_IO.Put("ID: "); Ada.Integer_Text_IO.Put(Id); Text_IO.New_Line;
         Text_IO.Put("Current amount: "); Ada.Integer_Text_IO.Put(Amount); Text_IO.Put("€");
Text_IO.New_Line;
      end balance_query;

      procedure deposit (money : in Integer) is
      begin
         Ada.Integer_Text_IO.Put(money); Text_IO.Put("€ are going to be inserted in account ");
Ada.Integer_Text_IO.Put(Id); Text_IO.New_Line;
         Amount := Amount + money;
      end deposit;

      procedure withdrawal (money : in Integer; result : out Integer) is
```

```
      begin
        Ada.Integer_Text_IO.Put(money); Text_IO.Put("€ are going to be extracted from
account "); Ada.Integer_Text_IO.Put(Id); Text_IO.New_Line;
        if money > Amount then
          Text_IO.Put("You cannot extract "); Ada.Integer_Text_IO.Put(money); Text_IO.Put("€
from this account"); Text_IO.New_Line;
          result := -1;
        else
          Amount := Amount - money;
          result := money;
        end if;
      end withdrawal;
   end bank_account;
   begin
      null;
end pkg_bank_account;
```

File – pkg_bank_account.ads

```
with Ada.Text_IO; use Ada.Text_IO;

package pkg_bank_account is
   protected type bank_account(B_Amount: access Integer; B_Id: Integer) is
                        procedure balance_query;
                        procedure deposit(money: in Integer);
                        procedure withdrawal(money: in Integer; result: out Integer);
private
      Amount: Integer := B_Amount.all;
             Id : Integer := B_Id;
   end bank_account;

   procedure balance_query(B: in out bank_account);
          procedure deposit(B: in out bank_account; money: in Integer);
          procedure withdrawal(B: in out bank_account; money: in Integer);
          procedure transfer(BSrc: in out Bank_Account; BDest: in out Bank_Account; money: in
Integer);
end pkg_bank_account;
```

Conclusion

In summary, doing this work, we have decided that for us, Ada may not be the best lenguage to develop programs as we are used to in other subjects, but it is so complete when we want to develop concurrent and real time programs.