# Memoria Práctica 3

Sistemas en Tiempo Real

## Programación de Tareas periódicas en ADA

**Almudena García Jurado-Centurión**

# Índice

## 0. Enunciado

Desarrollar una aplicación que ejecute cuatro tareas periódicas con las siguientes características, haciendo uso de tareas ADA:

- Las tareas escriben su nombre (p.ej A, B, C y D) y la hora
- Los periodos de las tareas son, 1, 2, 3 y 4s.
- Los plazos de respuesta son iguales a los periodos.
- Se debe detectar y tratar el incuplimiento de los plazos de respuesta.

## 1. Planteamiento

Una tarea periódica consiste en una rutina que se ejecuta cada cierto intervalo de tiempo, el cual puede estar definido de forma estática. Además, puede tener asociada una prioridad, que indique el orden de ejecución en caso de ejecutarse junto a varias tareas más.

En una situación ideal, la ejecución de la tarea periódica estaría únicamente determinado por su *deadline* (el intervalo máximo de tiempo que puede tardar una tarea en responder), y su prioridad.

Pero para llegar a esa situación necesitaríamos un planificador de tiempo real duro (o *hard real-time)*, el cual no está disponible en la mayoría de sistemas operativos ni de compiladores habituales.
Por esta razón, en nuestro caso, al no disponer de un planificador de tiempo real duro, el plazo de ejecución y la prioridad no siempre van a cumplirse.

## 2. Estructura de la aplicación

La aplicación esta compuesta por 2 paquetes, un procedimiento llamado Write_Time y un procedimiento principal:

- El paquete Global (tomado del enunciado), alberga las definiciones de tipo de ejecución, tarea periódica

- El paquete P_Periodic_Task, que alberga la definición de tarea periódica y su comportamiento

- El procedimiento Write_Time, que se encarga de mostrar la fecha y hora actual

- El procedimiento principal, Main, que lanza la ejecución de los 4 hilos.

## 3. Implementación

### 3. 1. Paquete P_Periodic_Task

Para implementar la tarea periódica, haremos uso del tipo *task,* que nos permite definir una tarea programada, con las acciones y la temporización que nosotros definamos.

La tarea la implementaremos en el paquete *P_Periodic_Task*

El paquete estará definido de la siguiente manera:

```
with System;
with Ada.Real_Time;

package P_Periodic_Task is
   task type Periodic_Task(Period: Integer; --Period
                           Phase: Integer; --Phase
                           Priority: System.Priority; --Priority
                           DeadLine: Integer; --DeadLine
                           id: Integer); --Name
end P_Periodic_Task;
```

La tarea estará definida como una estructura de tipo *task,* con los siguientes campos:

- **Period**: Correspondiente al intervalo a esperar entre una ejecución y la siguiente
- **Phase:** Duración de una tarea
- **Priority:** Nivel de prioridad
- **DeadLine:** Plazo máximo de respuesta
- **id:** Identificador de la tarea

Una vez definidos los campos, debemos definir el comportamiento que esperamos obtener. Para ello, implementamos los procedimientos *Handle_Failure* y *Periodic_Activity.* El primero implementará el comportamiento en caso de superar el deadline, y el segundo el comportamiento normal en otra situación.

- Si la respuesta se obtiene antes del deadline, la tarea mostrará un mensaje diciendo el identificador de la tarea y la hora actual.

- Si la respuesta se obtiene después del deadline, la tarea mostrará un mensaje indicando que se ha alcanzado el deadline, junto a la hora actual

La implementación de estas dos funciones es la siguiente:

```
procedure Handle_Failure (Fail : Global.Execution_Type) is
begin
   Ada.Text_IO.Put("Deadline reached. Task with delay: ");
   Write_Time(Ada.Calendar.Clock);
end Handle_Failure;

procedure Periodic_Activity(id: Integer) is
begin
   Ada.Text_IO.Put("I'm the task " & id'img & " the time is ");
   Write_Time(Ada.Calendar.Clock);
end Periodic_Activity;
```

*Handle_Failure*, como dijimos previamente, es el procedimiento que muestra el mensaje en caso de superar el deadline; mientras que *Periodic_Activity* muestra el mensaje en el resto de casos.

Para mostrar la hora usaremos el procedimiento Write_Time, ya incluido en el enunciado, que nos mostrará la fecha y hora actual.

En ambos casos, se mostrará un mensaje seguido de la fecha y hora. En el caso de *Periodic_Activity* se mostrará, además, el identificador de la tarea.

## 3.2. Bloques task

Una vez definidos las funciones asociadas a cada caso, pasamos a definir la propia tarea.

La tarea se compondrá de dos bloques *task: task_view,* que se encargará de realizar el *rendezvouz,* y llamará al procedimiento indicado en cada caso; y *Periodic_Task,* el bloque principal, que se encargará de marcar los tiempos de la tarea, e invocar a las entradas declaradas en *task_view.*

La implementación de *view_task* es la siguiente:

```
task body view_task is
begin
   loop
      select
         --Task without delay
         accept ex_Periodic_Activity(id: Integer) do
            Periodic_Activity(id);
         end ex_Periodic_Activity;
      or
         accept ex_Handle_Failure do
            --DeadLine reached
            Handle_Failure;
         end ex_Handle_Failure;
      end select;
   end loop;
end ;
```

El bloque tiene dos eventos: *ex_Periodic_Activity,* que invoca a *Periodic_Activity;* y *ex_Handle_Failure,* que invoca a *Handle_Failure*

Estas dos entradas son controladas mediante un bloque *select,* que recibe la señal y selecciona la entrada indicada.

La implementación de *Periodic_Task* es la siguiente:

```
task body Periodic_Task is
   use type Ada.Real_Time.Time;

   time: Ada.Calendar.Time := Ada.Calendar.Clock;
   Next : Ada.Real_Time.Time := Global.Start_Time + Ada.Real_Time.Milliseconds(Phase);
   End_deadline : Ada.Real_Time.Time := Global.Start_Time + Ada.Real_Time.Milliseconds(Deadline);

   begin
   -- Periodic_Task
   loop
      delay until Next;
      select
         view_task.ex_Handle_Failure;
      else
         if Ada.Real_Time.Clock > Next then
            view_task.ex_Periodic_Activity(id);
          end if;
      end select;
      Next := Next + Ada.Real_Time.Milliseconds(Period);
   end loop;


end Periodic_Task;
```

Este bloque utiliza tres variables: *time,* que indica la hora actual; *Next,* que almacena la hora correspondiente con el siguiente periodo de ejecución; y *End_deadline,* que almacena la hora correspondiente al plazo límite de respuesta.

Este bloque se mantiene en espera hasta llegar la hora prevista para su siguiente ciclo, momento en el cual comprueba si se ha superado el *deadline*. Si se ha superado, llama al evento *ex_Handle_Failure* de *view_task*; si no, llama a *ex_Periodic_Activity.*

Una vez invocado, incrementa la variable *Next,* y espera hasta el siguiente ciclo.

## 3.3. Procedimiento main

Para lanzar los hilos, usaremos el procedimiento Main, ubicado en el fichero Main.adb

El contenido del Main sera el siguiente:

```
with P_Periodic_Task;

procedure Main is
   type Periodic is new P_Periodic_Task.Periodic_Task;

   T1: Periodic(1000,0,1,1000, 1); --Task 1
   T2: Periodic(2000,0,1,2000, 2); --Task 2
   T3: Periodic(3000,0,1,3000, 3); --Task 3
   T4: Periodic(4000,0,1,4000, 4); --Task 4
begin
   null;
end Main;
```

Para lanzar los hilos, creamos un nuevo tipo con el tipo del paquete, y lanzamos 4 objetos con ese tipo.

## 4. Código completo

- **Global.ads**

```
with System;
with Ada.Real_Time;

package Global is
   type Execution_Type is (Correct, Internal_Error, External_Error);

   type Periodic_Profile_Type is record
      Period : Ada.Real_Time.Time_Span;
      Phase : Ada.Real_Time.Time;
      Priority : System.Priority;
      Deadline : Ada.Real_Time.Time_Span;
   end record;


   Start_Time : Ada.Real_Time.Time := Ada.Real_Time.Clock;
end Global;
```

Nota: se han eliminado aquellos tipos y procedimientos no necesarios para la práctica

- **p_periodic_task.ads**

```
with System;
with Ada.Real_Time;

package P_Periodic_Task is
   task type Periodic_Task(Period: Integer; --Period
                           Phase: Integer; --Phase
                           Priority: System.Priority; --Priority
                           DeadLine: Integer; --DeadLine
                           id: Integer); --Name
end P_Periodic_Task;
```

- **p_periodic_task.adb**

```
pragma Task_Dispatching_Policy (FIFO_Within_Priorities);
pragma Queuing_Policy (Priority_Queuing);
pragma Locking_Policy (Ceiling_Locking);

with Global;
with Write_Time;
with Ada.Real_Time;
with Ada.Calendar;
with Ada.Text_IO;

package body P_Periodic_Task is

   task view_task is
      entry ex_Handle_Failure;
      entry ex_Periodic_Activity(id: Integer);
   end view_task;


   procedure Handle_Failure is
   begin
      Ada.Text_IO.Put("Deadline reached. Task with delay: ");
      Write_Time(Ada.Calendar.Clock);
   end Handle_Failure;

 procedure Periodic_Activity(id: Integer) is
 begin
    Ada.Text_IO.Put("I'm the task " & id'img & " the time is ");
    Write_Time(Ada.Calendar.Clock);
 end Periodic_Activity;


 task body view_task is
 begin
    loop
       select
          --Task without delay
          accept ex_Periodic_Activity(id: Integer) do
             Periodic_Activity(id);
          end ex_Periodic_Activity;
       or
          accept ex_Handle_Failure do
             --DeadLine reached
             Handle_Failure;
          end ex_Handle_Failure;
       end select;
    end loop;
 end ;
```

En este fichero, además de definir la tarea, se aplicarán las directivas pragma para forzar el esquema de planificación de las tareas. En este caso, el algoritmo de planificación será un FIFO con prioridades.

```
   task body Periodic_Task is
      use type Ada.Real_Time.Time;

      time: Ada.Calendar.Time := Ada.Calendar.Clock;
      Next : Ada.Real_Time.Time := Global.Start_Time + Ada.Real_Time.Milliseconds(Phase);
      End_deadline : Ada.Real_Time.Time := Global.Start_Time + Ada.Real_Time.Milliseconds(Deadline);

      begin
      -- Periodic_Task
      loop
         delay until Next;
         select
            view_task.ex_Handle_Failure;
         else
            if Ada.Real_Time.Clock > Next then
               view_task.ex_Periodic_Activity(id);
            end if;
          end select;
          Next := Next + Ada.Real_Time.Milliseconds(Period);
      end loop;


   end Periodic_Task;
end P_Periodic_Task;
```

Nota: el bloque *exception* se ha considerado innecesario, y se ha eliminado

- **write_time.adb**

```
with Ada.Real_Time;
with Ada.Dynamic_Priorities;
with Ada.Text_IO; with Ada.Integer_Text_IO;
with Ada.Float_Text_IO;
with Calendar;

procedure Write_Time (Actual_Time : Calendar.Time) is
begin
   Ada.Integer_Text_IO.Put(Integer(Calendar.Year(Actual_Time)),4);
   Ada.Text_IO.Put(":");
   Ada.Integer_Text_IO.Put(Integer(Calendar.Month(Actual_Time)),2);
   Ada.Text_IO.Put(":");
   Ada.Integer_Text_IO.Put(Integer(Calendar.Day(Actual_Time)),2);
   Ada.Text_IO.Put(":");
   Ada.Float_Text_IO.Put(Float(Calendar.Seconds(Actual_Time)));
   Ada.Text_IO.New_Line;
end Write_Time;
```

- **Main.adb**

```
with P_Periodic_Task;

procedure Main is
   type Periodic is new P_Periodic_Task.Periodic_Task;

   T1: Periodic(1000,0,1,1000, 1); --Task 1
   T2: Periodic(2000,0,1,2000, 2); --Task 2
   T3: Periodic(3000,0,1,3000, 3); --Task 3
   T4: Periodic(4000,0,1,4000, 4); --Task 4
begin
   null;
end Main;
```

Código completo en
https://github.com/AlmuHS/Practicas_STR/tree/master/Practica3_v2

## 5. Ejecución

Tras ejecutar la aplicación, obtenemos el siguiente resultado:

```
Deadline reached. Task with delay: 2018: 2: 6: 8.48436E+04
I'm the task  2 the time is 2018: 2: 6: 8.48436E+04
Deadline reached. Task with delay: 2018: 2: 6: 8.48446E+04
I'm the task  1 the time is 2018: 2: 6: 8.48446E+04
Deadline reached. Task with delay: 2018: 2: 6: 8.48456E+04
I'm the task  4 the time is 2018: 2: 6: 8.48456E+04
I'm the task  1 the time is 2018: 2: 6: 8.48456E+04
Deadline reached. Task with delay: 2018: 2: 6: 8.48466E+04
Deadline reached. Task with delay: 2018: 2: 6: 8.48476E+04
I'm the task  1 the time is 2018: 2: 6: 8.48476E+04
I'm the task  2 the time is 2018: 2: 6: 8.48476E+04
Deadline reached. Task with delay: 2018: 2: 6: 8.48486E+04
```

Vemos que el deadline se alcanza en muchísimas ocasiones, a razón de una vez por cada 4 tareas.

También vemos que la ejecución no sigue ninguna secuencia determinada.

Pese a todo, los tiempos son bastante regulares, y no se aprecian grandes diferencias de tiempo entre la ejecución de una tarea y la de otra