

Sistemas de Tiempo Real

Programación de aspectos de tiempo real

Capítulo 4

Programación de aspectos de tiempo real

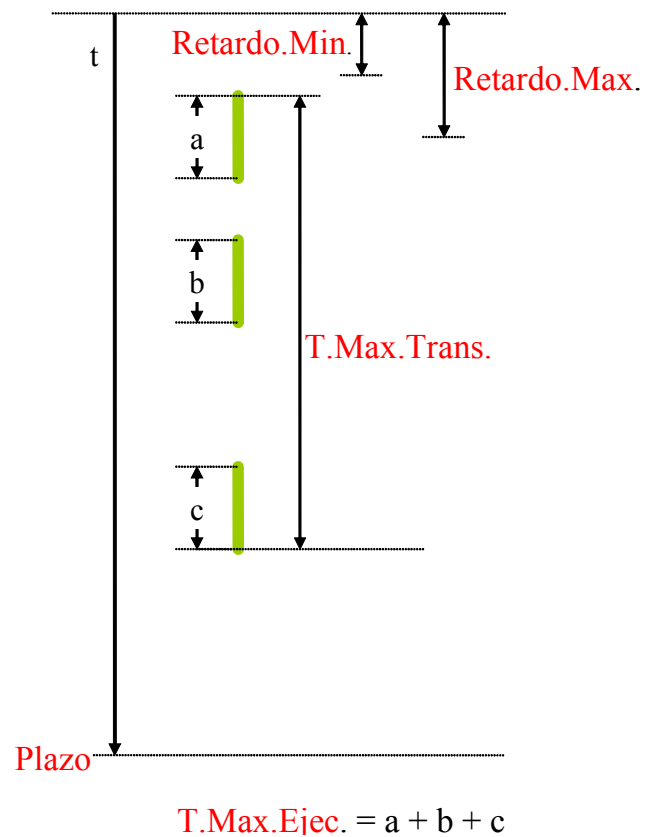
- 1. Introducción**
- 2. Interfaz con el tiempo**
- 3. Implementación de procesos periódicos**
- 4. Introducción a las prioridades de tareas**
- 5. Implementación de procesos aperiódicos. Servidores esporádicos**

4.1.- Introducción

- **Interfaz con el tiempo**
- **Programación de procesos**
 - ➔ **periódicos**
 - ➔ **aperiódicos**

4.2.- Interfaz con el tiempo

◆ Conceptos sobre tiempo



◆ Paquete Calendar de Ada

```
package Ada.Calendar is
```

```
    type Time is private;
```

```
    subtype Year_Number   is Integer range 1901 .. 2099;
```

```
    subtype Month_Number  is Integer range 1 .. 12;
```

```
    subtype Day_Number    is Integer range 1 .. 31;
```

```
    subtype Day_Duration  is Duration range 0.0 .. 86_400.0;
```

```
    function Clock return Time;
```

```
    function Year      (Date : Time) return Year_Number;
```

```
    function Month     (Date : Time) return Month_Number;
```

```
    function Day       (Date : Time) return Day_Number;
```

```
    function Seconds   (Date : Time) return Day_Duration;
```

```
procedure Split
```

```
    (Date      : Time;  
     Year      : out Year_Number;  
     Month     : out Month_Number;  
     Day       : out Day_Number;  
     Seconds   : out Day_Duration);
```

```
function Time_Of
```

```
    (Year      : Year_Number;  
     Month     : Month_Number;  
     Day       : Day_Number;  
     Seconds   : Day_Duration := 0.0)  
    return     Time;
```

```
function "+" (Left: Time;      Right: Duration) return Time;  
function "+" (Left: Duration; Right: Time)      return Time;  
function "-" (Left: Time;      Right: Duration) return Time;  
function "-" (Left: Time;      Right: Time)      return Duration;
```

```
function "<"    (Left, Right : Time) return Boolean;  
function "<="  (Left, Right : Time) return Boolean;  
function ">"    (Left, Right : Time) return Boolean;  
function ">="  (Left, Right : Time) return Boolean;  
  
Time_Error : exception;  
  
private  
.....  
  
end Ada.Calendar;
```

♦ Ejemplo. Medición del tiempo de ejecución de una tarea.

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

procedure Utiliza_Calendario is

    task Hacer_algo ;

package E_S_REALES is new Float_IO (Float);

-- vamos a medir el tiempo que tarda
-- en ejecutarse el cuerpo de una tarea
```



```
task body Hacer_algo is
-- declaraciones
    Comienzo: Time;
    Periodo : Duration;
    Tiempo : float;
begin
    Comienzo := Clock;
    --
    -- Código de la tarea
    --
    Periodo := Clock - Comienzo;
    -- escribimos el calculo
    Tiempo := float (Periodo);
    Put ("La tarea ha tardado ");
    E_S_REALES.Put(Tiempo,2,4,0); Put_Line ("");
end Hacer_algo;
```

◆ Paquete Ada.Real_Time

```
with System.Task_Primitives.Operations;  
pragma Elaborate_All (System.Task_Primitives.Operations);
```

```
package Ada.Real_Time is
```

```
    type Time is private;  
    Time_First : constant Time;  
    Time_Last  : constant Time;  
    Time_Unit  : constant := 10#1.0#E-9;
```

```
    type Time_Span is private;  
    Time_Span_First : constant Time_Span;  
    Time_Span_Last  : constant Time_Span;  
    Time_Span_Zero  : constant Time_Span;  
    Time_Span_Unit  : constant Time_Span;
```

```
    Tick : constant Time_Span;
```

```
function Clock return Time;
```

```
function "+" (Left : Time;      Right : Time_Span) return Time;  
function "+" (Left : Time_Span; Right : Time)      return Time;  
function "-" (Left : Time;      Right : Time_Span) return Time;  
function "-" (Left : Time;      Right : Time)      return Time_Span;
```

```
function "<" (Left, Right : Time) return Boolean;  
function "<=" (Left, Right : Time) return Boolean;  
function ">" (Left, Right : Time) return Boolean;  
function ">=" (Left, Right : Time) return Boolean;
```

```
function "+" (Left, Right : Time_Span) return Time_Span;  
function "-" (Left, Right : Time_Span) return Time_Span;  
function "-" (Right : Time_Span)      return Time_Span;  
function "*" (Left : Time_Span; Right : Integer) return Time_Span;  
function "*" (Left : Integer;   Right : Time_Span) return Time_Span;  
function "/" (Left, Right : Time_Span) return Integer;  
function "/" (Left : Time_Span; Right : Integer) return Time_Span;
```

```
function "abs" (Right : Time_Span) return Time_Span;

function "<" (Left, Right : Time_Span) return Boolean;
function "<=" (Left, Right : Time_Span) return Boolean;
function ">" (Left, Right : Time_Span) return Boolean;
function ">=" (Left, Right : Time_Span) return Boolean;

function To_Duration (TS : Time_Span) return Duration;
function To_Time_Span (D : Duration) return Time_Span;

function Nanoseconds (NS : Integer) return Time_Span;
function Microseconds (US : Integer) return Time_Span;
function Milliseconds (MS : Integer) return Time_Span;

type Seconds_Count is new Integer range -Integer'Last..Integer'Last;

procedure Split (T: Time; SC: out Seconds_Count; TS: out Time_Span);
function Time_Of (SC: Seconds_Count; TS: Time_Span) return Time;
```

```
private
  type Time is new Duration;
  Time_First : constant Time := Time'First;
  Time_Last  : constant Time := Time'Last;
  type Time_Span is new Duration;
  Time_Span_First : constant Time_Span := Time_Span'First;
  Time_Span_Last  : constant Time_Span := Time_Span'Last;
  Time_Span_Zero   : constant Time_Span := 0.0;
  Time_Span_Unit   : constant Time_Span := 10#1.0#E-9;
  Tick : constant Time_Span :=
    Time_Span (System.Task_Primitives.Operations.RT_Resolution);
  -- Time and Time_Span are represented in 64-bit Duration value in
  -- in nanoseconds. For example, 1 second and 1 nanosecond is
  -- represented as the stored integer 1_000_000_001.
  pragma Import (Intrinsic, "<");
  pragma Import (Intrinsic, "<=");
  pragma Import (Intrinsic, ">");
  pragma Import (Intrinsic, ">=");
  pragma Import (Intrinsic, "abs");
end Ada.Real_Time;
```

Un valor I de tipo Time representa un instante absoluto durante un intervalo de tiempo

$$E + I * Time_Unit .. E + (I + 1) * Time_Unit$$

Time_Unit: menor cantidad de tiempo representable por el sistema

E: época (instante de origen de tiempo)

Time_Span: cantidad de tiempo (intervalo o duración)

To_Duration convierte **Time_Span** a **Duration**

Tick: intervalo durante el cual la hora del sistema permanece constante

♦ Programación de retardos

Retardos relativos

delay ***n***

donde **n** es de tipo **Duration**, número de **segundos de espera**

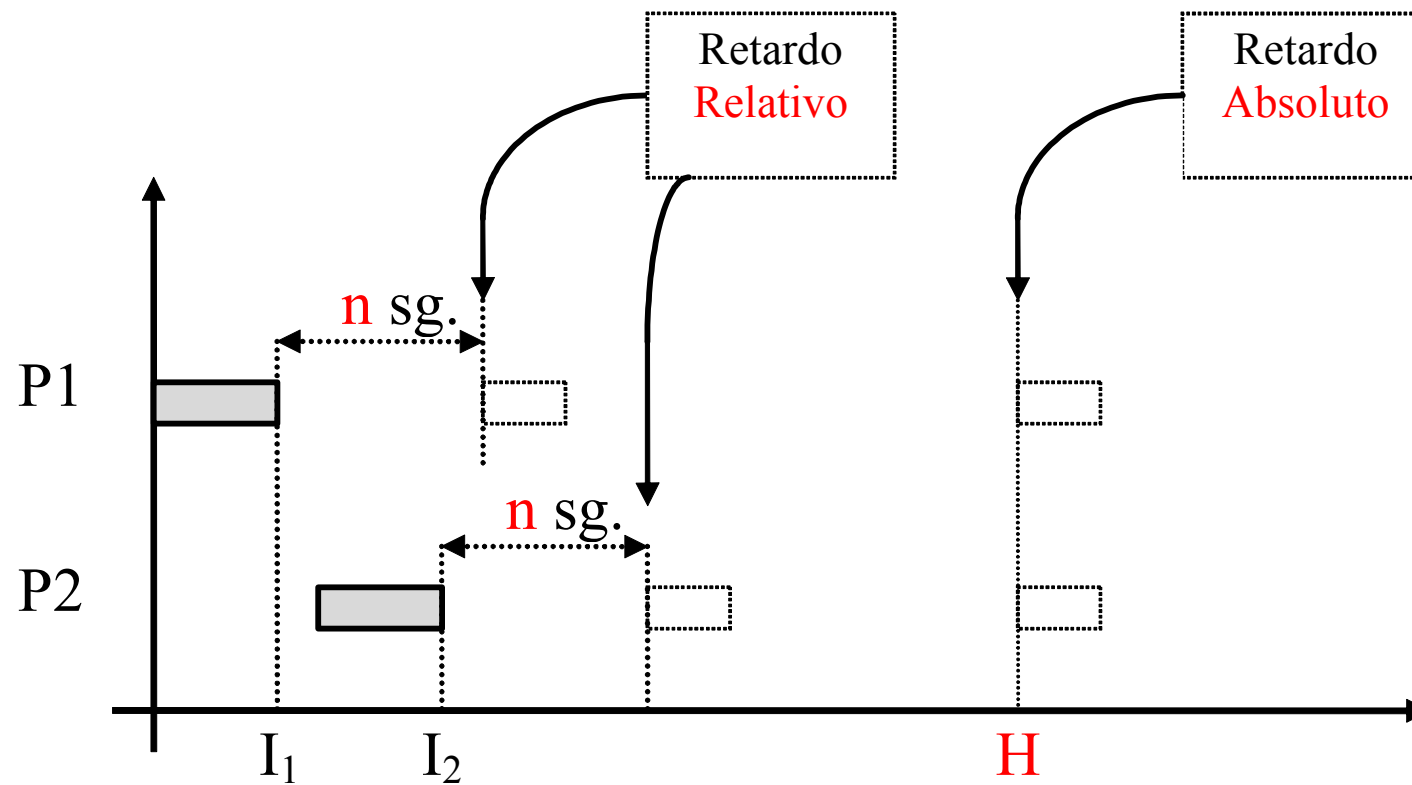
El proceso vuelve a estar “ejecutable” después de **n** sgs.

Retardos absolutos

delay until ***HH***

donde **HH** es de tipo **Time**, indica la **hora de activación** de la tarea

El proceso vuelve a estar “ejecutable” en el instante **HH**.



Time-Outs en Paso de Mensajes (*Rendez vous*)

```
Task body La_que_acepta is
begin
  ...
  Select
    accept E (...) do
      ... (I0)
    end accept ;
    ... (I1)
  or
    delay 3.0 ;
    ... (I2)
  end select ;
  ... (I3)
end La_que_acepta ;
```

```
Task body La_que_llama is
begin
  ...
  Select
    T.E (...) ;
    ... (I1)
  or
    delay 3.0 ;
    ... (I2)
  end select ;
  ... (I3)
end La_que_llama ;
```

♦ Time-Out en Objetos Protegidos

Igual que en el caso de la tarea “llamante” del apartado anterior.

♦ Transferencia asíncrona de control (ATC)

Llamada a entrada protegida

```
...  
Select  
    Nombre_OP_T.Entrada (...)  
    ... (I1)  
then abort  
    ... (I2)  
end select;  
    ... (I3)
```

♦ Transferencia asíncrona de control (ATC)

Retardo relativo o absoluto

```
...  
Select  
    delay n;  -- o bien delay until  
    ... (I1)  
then abort  
    ... (I2)  
end select ;  
... (I3)
```

Abort deferred (terminación diferida o aplazada)

La suspensión se **aplaza hasta que se haya completado** dicha operación

Operaciones con terminación aplazada:

- Una acción protegida
- Estar esperando a que se complete una llamada a una entrada
- Estar esperando por la terminación de una tarea hijo
- Inicialización, finalización y asignación de un *objeto controlado*.

4.3.- Implementación de procesos periódicos

Repetir

Accion

Esperar la hora de la **siguiente activación**
hasta Fin

o bien

Repetir

Esperar la hora de la **siguiente activación**

Accion

hasta Fin

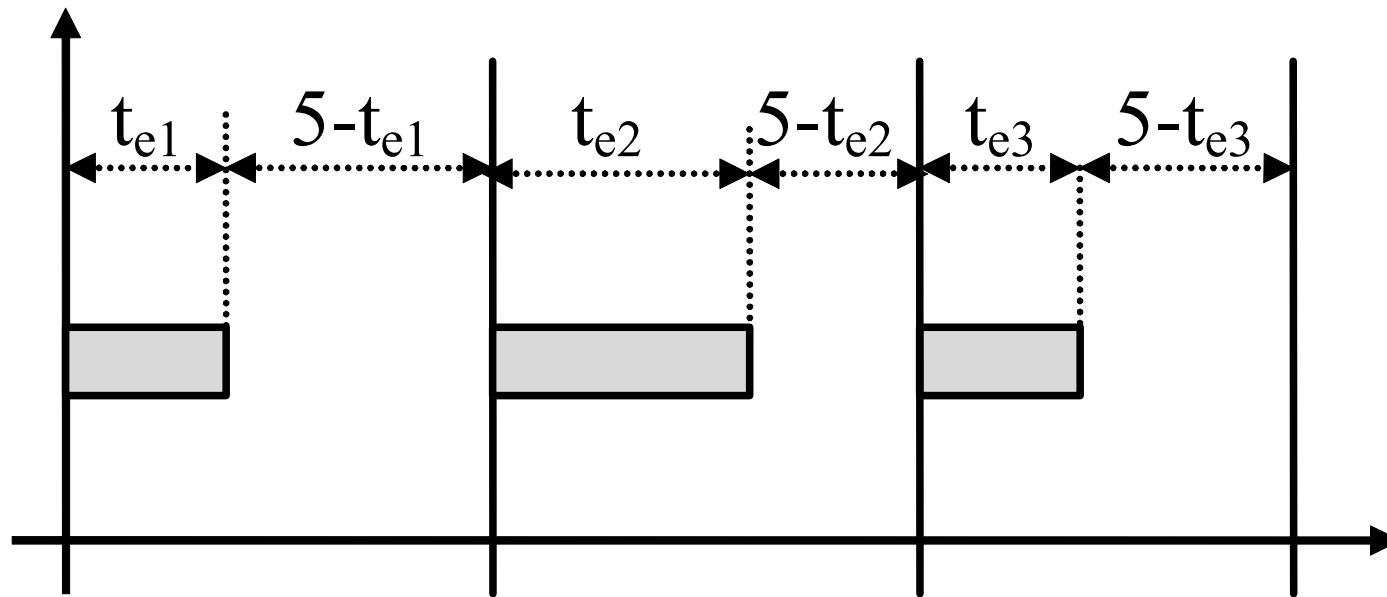
En lenguaje Ada ...

```
Task body Periodica is
  periodo : Duration := (5.0 - 2.0) ;
begin
  loop
    Accion;
    delay periodo;
  end loop;
end Periodica;
```

... **PERO** el periodo real depende del tiempo de ejecución de la acción.

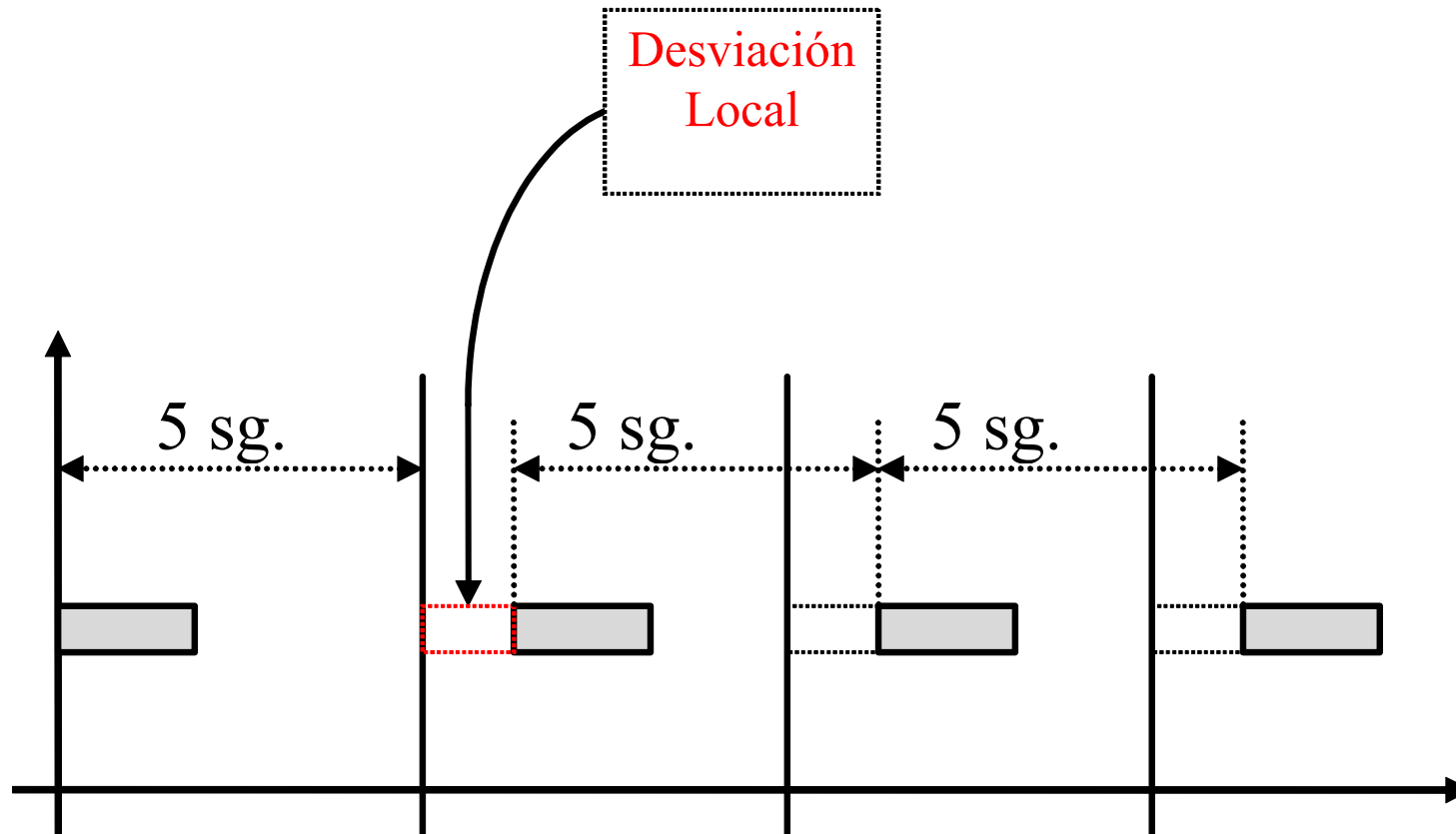
Siguiente solución

```
Task body Periodica is
    periodo : Duration := (5.0);
    inicio : Time;
    t_ejecucion : Duration;
begin
    loop
        inicio := clock;
        Accion ;
        t_ejecucion := clock - inicio;
        delay periodo - t_ejecucion;
    end loop;
end Periodica;
```



... **PERO** puede aparecer una desviación en la activación de una tarea.

Desviación local



Solución con retardo absoluto:

Task body Periodica is

 Siguiente_Instante : Time;

 Intervalo : Duration := 5.0 ;

 -- por ejemplo 5 segundos

begin

 Siguiente_Instante := Clock + Intervalo;

 loop

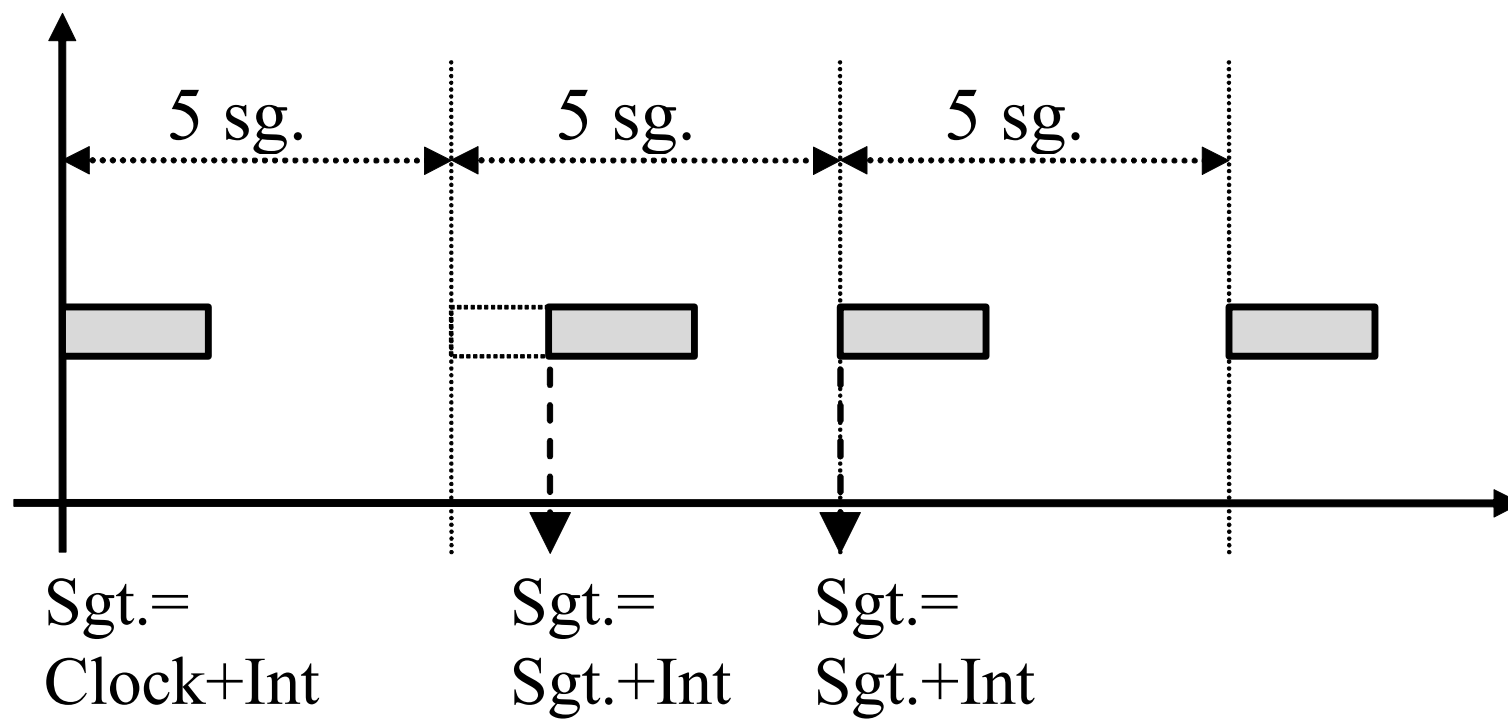
 Accion ;

 delay until Siguiente_Instante;

 Siguiente_Instante := Siguiente_Instante
 + Intervalo;

 end loop;

end Periodica;



4.4.- Introducción a las prioridades de tareas

♦ Paquete System

package System is

.....

-- Priority-related Declarations (RM D.1)

subtype **Any_Priority** is Integer

range 0 .. Standard'Max_Interrupt_Priority;

subtype **Priority** is Any_Priority

range 0 .. Standard'Max_Priority;

subtype **Interrupt_Priority** is Any_Priority

range Standard'Max_Priority + 1 ..

Standard'Max_Interrupt_Priority;

Default_Priority : constant Priority :=

(Priority'First + Priority'Last) / 2;

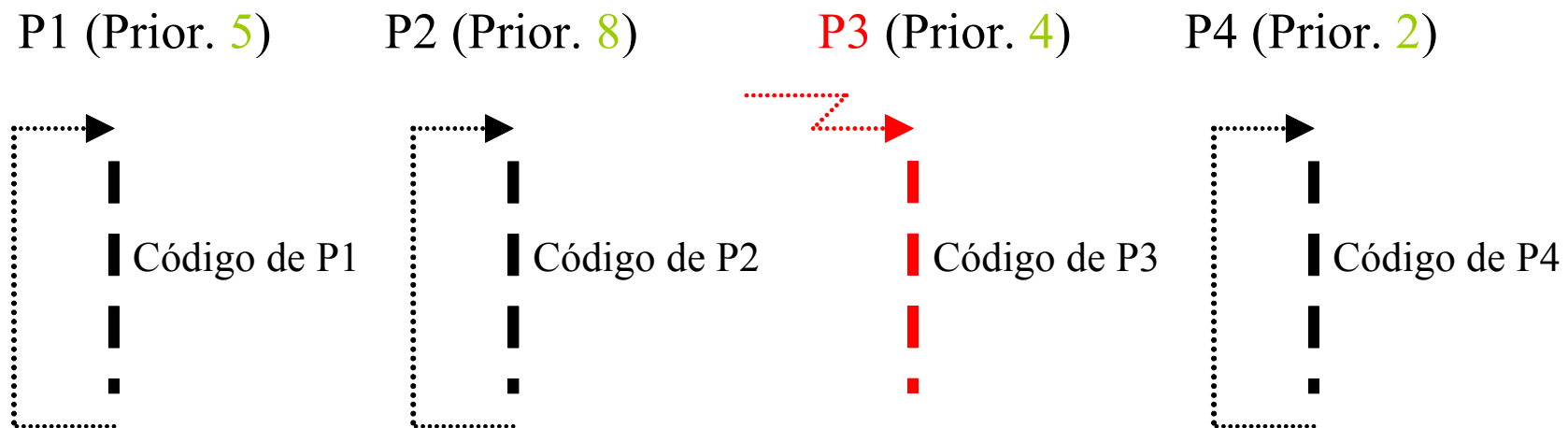
◆ Asignación de una prioridad a una tarea

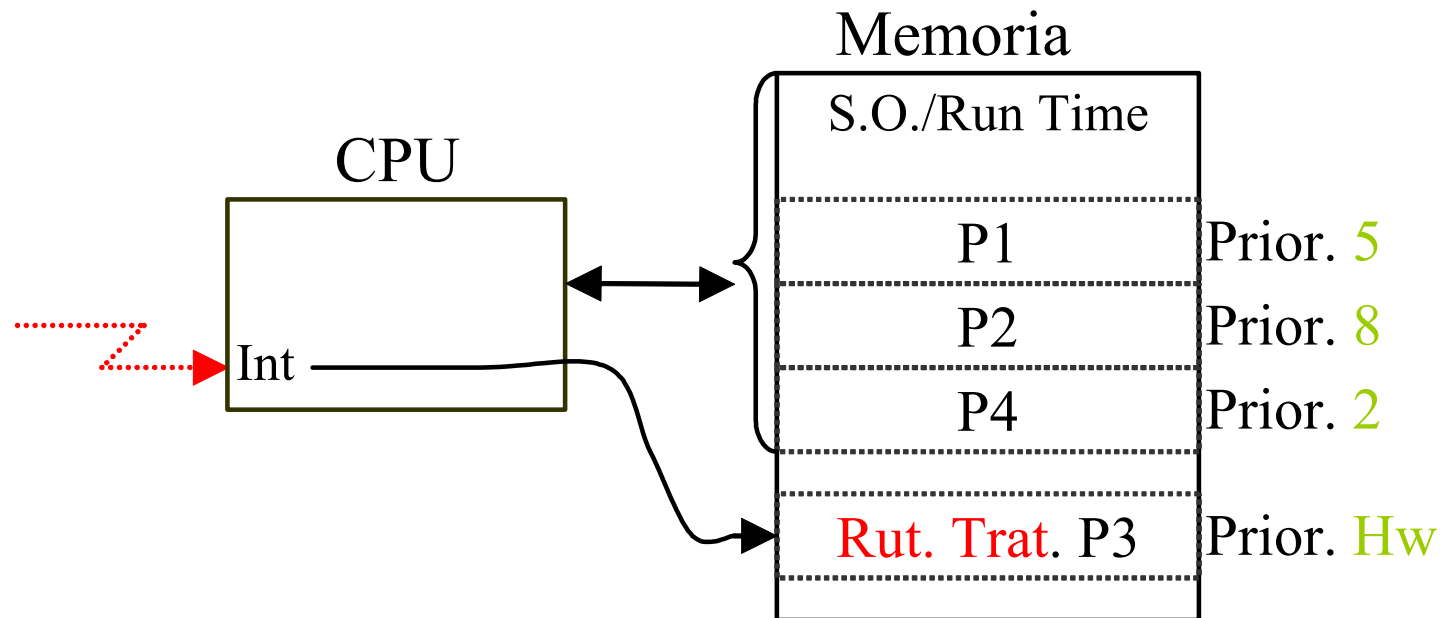
```
Task A is  
    pragma Priority (15) ;  
end A ;
```

```
Task body A is  
    ...  
begin  
    ...  
end A ;
```

4.5.- Implementación de procesos aperiódicos. Servidores esporádicos.

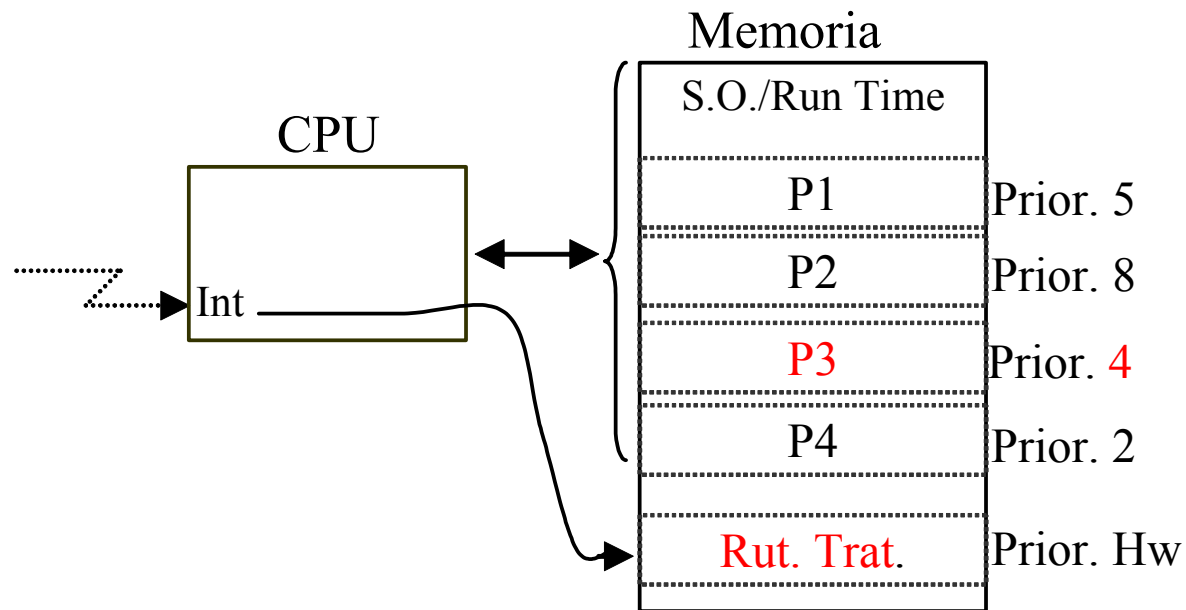
Implementamos el proceso P3 (esporádico) mediante una rutina de tratamiento de interrupción ...





- ➔ prioridades software (gestionadas por el Run Time System)
- ➔ prioridades hardware (gestionadas por el procesador)

Separamos el **código del proceso** y el **código de la rutina de tratamiento** de interrupción



La **rutina de interrupción** se limitará a **desbloquear** al proceso, que tendrá su prioridad software.

Proceso aperiódico

Rutina de Tratamiento de Interrupción

Envia señal de llegada del evento

Fin de Interrupción

Proceso Esporádico P3 (**Prior. 4**)

loop

Bloqueado en espera de señal de llegada del evento

Código de P3

Da por servida la interrupción

end loop ;

Limitar su frecuencia máxima de activación

Rutina de Tratamiento de Interrupción

Envia señal de llegada del evento

Instante := Clock

Fin de Interrupción

Proceso Esporádico P3 (Prior. 4)

loop

Bloqueado en **espera de señal** de llegada del evento

Código de P3

Delay until Instante + Periodo_Minimo

Da por servida la interrupción

end loop ;

♦ Programación de las interrupciones en Ada95

Paquete Ada.Interrupts

```
with System;
with Interfaces.C.System_Constants;
package Ada.Interrupts is

    type Interrupt_ID is new integer
        range 0 .. Interfaces.C.System_Constants.NSIG;
    type Parameterless_Handler is
        access protected procedure;
    function Is_Reserved (Interrupt : Interrupt_ID)
        return Boolean;
    function Is_Attached (Interrupt : Interrupt_ID)
        return Boolean;
```

```
function Current_Handler (Interrupt : Interrupt_ID)
    return Parameterless_Handler;
procedure Attach_Handler
    (New_Handler : Parameterless_Handler;
     Interrupt    : Interrupt_ID);
procedure Exchange_Handler
    (Old_Handler : out Parameterless_Handler;
     New_Handler : Parameterless_Handler;
     Interrupt    : Interrupt_ID);
procedure Detach_Handler
    (Interrupt : Interrupt_ID);
function Reference (Interrupt : Interrupt_ID)
    return System.Address;

end Ada.Interrupts;
```

En la asociación dinámica, para el procedimiento protegido que vaya a ser un manejador de una interrupción:

pragma Interrupt_Handler (Handler_Name) ;

Para la asociación estática de un procedimiento protegido a una interrupción:

pragma Attach_Handler (Handler_Name, Expression) ;

◆ Ejemplo de programación de interrupciones. Asignación dinámica.

```
with Ada.Interrupts; use Ada.Interrupts;
with RutInt;
procedure Demo is
    Timer      : constant := 8; -- interrupción del timer (55 ms.)
    original: Parameterless_Handler;
    mio: Parameterless_Handler := RutInt.Agent.Handler'Access;
begin
    -- Cogemos el puntero a la rutina actual
    original := Current_Handler (Timer);
    -- Asignamos el nuevo puntero
    Attach_Handler (mio, Timer);
    -- Acciones del programa principal
    ...
    -- Recuperamos el puntero original
    Exchange_Handler (mio, original, Timer);
end Demo;
```

```
-----  
--  Especific. del paquete que contiene la Rut. de Tratamiento  
-----  
package RutInt is  
  protected Agent is  
    procedure Handler;  
    pragma Interrupt_Handler (Handler);  
  private  
  end Agent;  
end RutInt;
```

```
-----  
-- Defin. del paquete que contiene la Rut.de Tratamiento  
-----  
with Ada.Interrupts; use Ada.Interrupts;  
with System, Interfaces; use System, Interfaces;  
with System.Storage_Elements; use System.Storage_Elements;  
package body RutInt is  
  protected body Agent is  
    procedure Handler is          -- Rutina de Tratamiento  
    begin  
      -- Cuerpo de la Rutina de Tratamiento  
      ...  
      -- Fin de la Rutina de Tratamiento  
    end Handler;  
  end Agent;  
begin  
  -- Inicializacion de las interrupciones:  
  ...      -- por ejemplo, permitir interr.  
end RutInt;  
-----
```


◆ Ejemplo de programación de interrupciones. Asignación estática.

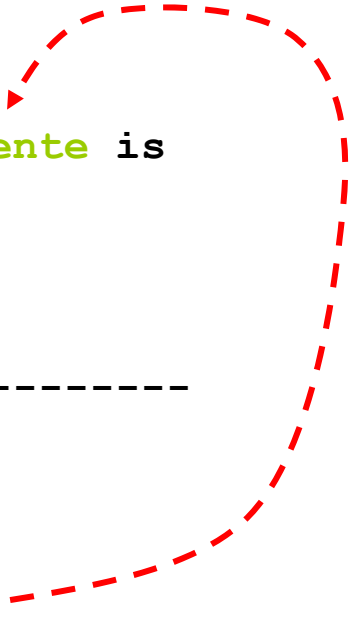
```
protected Objeto_Interrupcion is
    pragma Priority (Prioridad_Interrupcion);
    procedure Rutina_Tratamiento;
    pragma Attach_Handler (Rutina_Tratamiento,
                           Ada.Interrupts.Names.Id_Interrupcion);
end Objeto_Interrupcion;

protected body Objeto_Interrupcion is
    procedure Rutina_Tratamiento is
    begin
        ...
    end Rutina_Tratamiento;
end Objeto_Interrupcion;
```

◆ Implementación en Ada de un servidor esporádico

```
-----  
----- Declaracion de la tarea esporadica  
task Esporadica is  
    pragma priority(Prioridad_Esporadica) ;  
end Esporadica;  
-----  
----- Objeto protegido  
Protected Controlador _Eventos is  
    pragma Priority (Prioridad_Interrupcion) ;  
    procedure Interrupcion ;  
    pragma Attach_Handler (Interrupcion, Id_Int) ;  
    -- Id_Int esta declarado como Interrupt_Id  
    -- e inicializado según el sistema  
    entry Esperar_Evento ;  
private  
    Llamada_Pendiente : Boolean := False; -- barrera  
end Controlador _Eventos;
```

```
-----  
----- Cuerpo del Objeto protegido  
Protected body Controlador_Eventos is  
  procedure Interrupcion is  
  begin  
    Llamada_Pendiente := True;  
  end Interrupcion;  
  entry Esperar_Evento when Llamada_Pendiente is  
  begin  
    Llamada_Pendiente := False;  
  end Llamada_Pendiente;  
end Controlador_Eventos;  
-----  
----- Tarea esporadica  
Task body Esporadica is  
begin  
  loop  
    Controlador.Esperar_Evento;  
    Accion;  
    Dar_por_servida_la_interrupción;  
  end loop;  
end Esporadica;  
-----
```



```

-----
----- Cuerpo del Objeto protegido
Protected body Controlador_Eventos is
Z → procedure Interrupcion is
  begin
    Llamada_Pendiente := True;
  end Interrupcion;
  entry Esperar_Evento when Llamada_Pendiente is
  begin
    Llamada_Pendiente := False;
  end Llamada_Pendiente;
end Controlador_Eventos;
-----

----- Tarea esporadica
Task body Esporadica is
begin
  loop
    Controlador.Esperar_Evento;
    Accion;
    Dar_por_servida_la_interrupción;
  end loop;
end Esporadica;
-----

```

```

-----
----- Cuerpo del Objeto protegido
Protected body Controlador_Eventos is
  procedure Interrupcion is
  begin
    Llamada_Pendiente := True;
  end Interrupcion;
  entry Esperar_Evento when Llamada_Pendiente is
  begin
    Llamada_Pendiente := False;
  end Llamada_Pendiente;
end Controlador_Eventos;
-----

----- Tarea esporadica
Task body Esporadica is
begin
  loop
    Controlador.Esperar_Evento;
    Accion;
    Dar_por_servida_la_interrupción;
  end loop;
end Esporadica;
-----

```

```
-----  
----- Cuerpo del Objeto protegido  
Protected body Controlador_Eventos is  
  procedure Interrupcion is  
  begin  
    Llamada_Pendiente := True;  
  end Interrupcion;  
  entry Esperar_Evento when Llamada_Pendiente is  
  begin  
    Llamada_Pendiente := False;  
  end Llamada_Pendiente;  
end Controlador_Eventos;  
-----
```

```
-----  
----- Tarea esporadica  
Task body Esporadica is  
begin  
  loop  
    Controlador.Esperar_Evento;  
    Accion;  
    Dar_por_servida_la_interrupción;  
  end loop;  
end Esporadica;  
-----
```

Paquete Ada.Interrupts.Names

```
-- This is an ORK version for ERC32 targets of this package.
with Ada.Interrupts;
-- used for Interrupt_ID
with System.OS_Interface;
-- used for names of interrupts

package Ada.Interrupts.Names is
    -----
    -- External Interrupts --
    -----

    -- This interrupt has a Interrupt Level equal 14.
    -- i.e. System.Interrupt_Priority'Last - 1 or
    -- System.Interrupt_Priority'First + 13
    External_Interrupt_4      : constant Interrupt_ID :=
        System.OS_Interface.External_Interrupt_4;

    External_Interrupt_4_Priority: constant System.Interrupt_Priority :=
        System.Interrupt_Priority'First + 13;
```



```
-- This interrupt has a Interrupt Level equal 11.
-- i.e. System.Interrupt_Priority'Last - 4 or
-- System.Interrupt_Priority'First + 10
External_Interrupt_3      : constant Interrupt_ID :=
    System.OS_Interface.External_Interrupt_3;
External_Interrupt_3_Priority: constant System.Interrupt_Priority :=
    System.Interrupt_Priority'First + 10;

-- This interrupt has a Interrupt Level equal 10.
-- i.e. System.Interrupt_Priority'Last - 5 or
-- System.Interrupt_Priority'First + 9
External_Interrupt_2      : constant Interrupt_ID :=
    System.OS_Interface.External_Interrupt_2;
External_Interrupt_2_Priority: constant System.Interrupt_Priority :=
    System.Interrupt_Priority'First + 9;
```

```
-- This interrupt has a Interrupt Level equal 3.
-- i.e. System.Interrupt_Priority'Last - 12 or
-- System.Interrupt_Priority'First + 2
External_Interrupt_1      : constant Interrupt_ID :=
    System.OS_Interface.External_Interrupt_1;
External_Interrupt_1_Priority: constant System.Interrupt_Priority :=
    System.Interrupt_Priority'First + 2;

-- This interrupt has a Interrupt Level equal 2.
-- i.e. System.Interrupt_Priority'Last - 13 or
-- System.Interrupt_Priority'First + 1
External_Interrupt_0      : constant Interrupt_ID :=
    System.OS_Interface.External_Interrupt_0;
External_Interrupt_0_Priority: constant System.Interrupt_Priority :=
    System.Interrupt_Priority'First + 1;
```

```
-----  
--  Timers Interrupts  --  
-----  
  
--  This interrupt has a Interrupt Level equal 15.  
--  i.e. System.Interrupt_Priority'Last or  
--  System.Interrupt_Priority'First + 14  
Watch_Dog_Time_Out          : constant Interrupt_ID :=  
    System.OS_Interface.Watch_Dog_Time_Out;  
Watch_Dog_Time_Out_Priority : constant System.Interrupt_Priority :=  
    System.Interrupt_Priority'First + 14;  
  
--  This interrupt has a Interrupt Level equal 13.  
--  i.e. System.Interrupt_Priority'Last - 2 or  
--  System.Interrupt_Priority'First + 12  
Real_Time_Clock             : constant Interrupt_ID :=  
    System.OS_Interface.Real_Time_Clock;  
Real_Time_Clock_Priority    : constant System.Interrupt_Priority :=  
    System.Interrupt_Priority'First + 12;
```

```
-- This interrupt has a Interrupt Level equal 12.
-- i.e. System.Interrupt_Priority'Last - 3 or
-- System.Interrupt_Priority'First + 11
General_Purpose_Timer          : constant Interrupt_ID :=
    System.OS_Interface.General_Purpose_Timer;
General_Purpose_Timer_Priority: constant System.Interrupt_Priority
    := System.Interrupt_Priority'First + 11;

-----
-- DMA Interrupts --
-----

-- This interrupt has a Interrupt Level equal 9.
-- i.e. System.Interrupt_Priority'Last - 6 or
-- System.Interrupt_Priority'First + 8
DMA_Time_Out                 : constant Interrupt_ID :=
    System.OS_Interface.DMA_Time_Out;
DMA_Time_Out_Priority : constant System.Interrupt_Priority :=
    System.Interrupt_Priority'First + 8;
```

```
-- This interrupt has a Interrupt Level equal 8.
-- i.e. System.Interrupt_Priority'Last - 7 or
-- System.Interrupt_Priority'First + 7
DMA_Access_Error          : constant Interrupt_ID :=
    System.OS_Interface.DMA_Access_Error;
DMA_Access_Error_Priority : constant System.Interrupt_Priority :=
    System.Interrupt_Priority'First + 7;

-----
--  UART Interrupts  --
-----

-- This interrupt has a Interrupt Level equal 7.
-- i.e. System.Interrupt_Priority'Last - 8 or
-- System.Interrupt_Priority'First + 6
UART_Error                : constant Interrupt_ID :=
    System.OS_Interface.UART_Error;
UART_Error_Priority       : constant System.Interrupt_Priority :=
    System.Interrupt_Priority'First + 6;
```

```
-- This interrupt has a Interrupt Level equal 5.
-- i.e. System.Interrupt_Priority'Last - 10 or
-- System.Interrupt_Priority'First + 4
UART_B_Ready          : constant Interrupt_ID :=
    System.OS_Interface.UART_B_Ready;
UART_B_Ready_Priority : constant System.Interrupt_Priority :=
    System.Interrupt_Priority'First + 4;

-- This interrupt has a Interrupt Level equal 4.
-- i.e. System.Interrupt_Priority'Last - 11 or
-- System.Interrupt_Priority'First + 3
UART_A_Ready          : constant Interrupt_ID :=
    System.OS_Interface.UART_A_Ready;

UART_A_Ready_Priority : constant System.Interrupt_Priority :=
    System.Interrupt_Priority'First + 3;
```

```
-----
--  Miscellaneous Interrupts  --
-----

--  This interrupt has a Interrupt Level equal 6.
--  i.e. System.Interrupt_Priority'Last - 9 or
--  System.Interrupt_Priority'First + 5
Correctable_Error_In_Memory : constant Interrupt_ID :=
    System.OS_Interface.Correctable_Error_In_Memory;
Correctable_Error_In_Memory_Priority : constant
    System.Interrupt_Priority :=
    System.Interrupt_Priority'First + 5;
--  This interrupt has a Interrupt Level equal 1.
--  i.e. System.Interrupt_Priority'Last - 14 or
--  System.Interrupt_Priority'First
Masked_Hardware_Errors      : constant Interrupt_ID :=
    System.OS_Interface.Masked_Hardware_Errors;
Masked_Hardware_Errors_Priority : constant System.Interrupt_Priority
    := System.Interrupt_Priority'First;

end Ada.Interrupts.Names;
```