Instrumentación de Tiempo Real

UNIVERSIDAD DE CANTABRIA

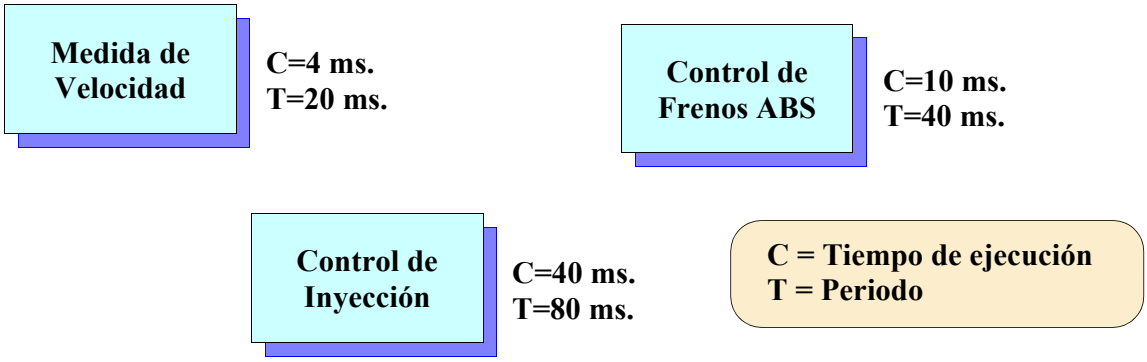
Necesidad de la concurrencia

UNIVERSIDAD DE CANTABRIA

Muchos problemas se expresan de forma natural mediante varias actividades concurrentes:

- **sistemas de control atendiendo a múltiples subsistemas y eventos**

Control de un automóvil



Solución no concurrente

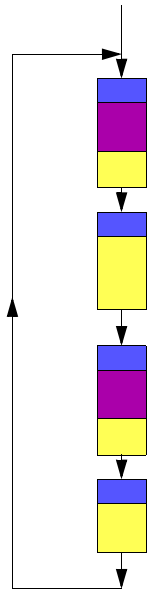
UNIVERSIDAD DE CANTABRIA

```
procedure Control_Automóvil is
begin
   loop
      Medida_De_Velocidad;
      Control_De_Frenos_ABS;
      Control_De_Inyección_1;
      Espera a que Tiempo = 20;

      Medida_De_Velocidad;
      Control_De_Inyección_2;
      Espera a que Tiempo = 40;

      Medida_De_Velocidad;
      Control_De_Frenos_ABS;
      Control_De_Inyección_3;
      Espera a que Tiempo = 60;

      Medida_De_Velocidad;
      Control_De_Inyección_4;
      Espera a que Tiempo = 80;
   end loop;
end Control_Automóvil;
```



Solución no concurrente: Problemas y limitaciones

UNIVERSIDAD DE CANTABRIA

Difícil de mantener:

- **cualquier cambio en una tarea puede trastocar la temporización**

Difícil de entender

Obliga a partir la tarea `Control_De_Inyección` en trozos

- **en ocasiones no será posible dividir una tarea**

Solución más compleja cuando los periodos de las tareas no son múltiplos

Concurrencia no soportada por el lenguaje

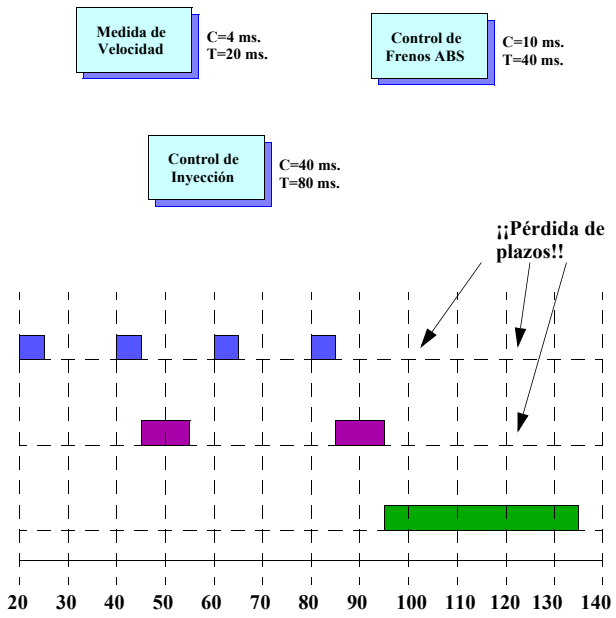
UNIVERSIDAD DE CANTABRIA

```
procedure Control_Automóvil is
  procedure Manejador_Tick is
     ...
begin
  loop
    if Han_Pasado_20ms then
      Medida_De_Velocidad;
      Han_Pasado_20ms:= False;
    end if;
    if Han_Pasado_40ms then
      Control_De_Frenos_ABS;
      Han_Pasado_40ms:= False;
    end if;
    if Han_Pasado_80ms then
      Control_De_Inyección;
      Han_Pasado_80ms:= False;
    end if;
  end loop;
end Control_Automóvil;
```


Concurrencia no soportada por el lenguaje: problemas

UNIVERSIDAD DE CANTABRIA


```
procedure Control_Automóvil is
  procedure Manejador_Tick is
     ...
begin
  loop
    if Han_Pasado_20ms then
      Medida_De_Velocidad;
      Han_Pasado_20ms:= False;
    end if;
    if Han_Pasado_40ms then
      Control_De_Frenos_ABS;
      Han_Pasado_40ms:= False;
    end if;
    if Han_Pasado_80ms then
      Control_De_Inyección;
      Han_Pasado_80ms:= False;
    end if;
  end loop;
end Control_Automóvil;
```

Concurrencia en Ada

UNIVERSIDAD DE CANTABRIA

El Ada soporta la programación de procesos concurrentes mediante las tareas ("tasks")

Las tareas constan de especificación y cuerpo:

- **en el cuerpo define una actividad que se ejecuta independientemente de las demás tareas**

```
task body Tarea is
   declaración de variables;
begin
   instrucciones (habitualmente un lazo infinito);
end Tarea;
```

Las tareas comienzan su ejecución de forma automática

Concurrencia en Ada (cont.)

UNIVERSIDAD DE CANTABRIA

```
procedure Control_del_Automóvil is
   task Medida_Velocidad;          -- especificación
   task body Medida_Velocidad is   -- cuerpo
   begin
     loop
       Acciones de Medida_de_Velocidad;
       Esperar al próximo periodo (20 ms);
     end loop;
   end Medida_Velocidad;
   task Control_ABS;           -- especificación
   task body Control_ABS is    -- cuerpo
   begin
     loop
       Acciones de Control_ABS;
       Esperar al próximo periodo (40 ms);
     end loop;
   end Control_ABS;
```

Concurrencia en Ada (cont.)

UNIVERSIDAD DE CANTABRIA

```
   task Control_Inyeccion;

   task body Control_Inyección is
   begin
      loop
        Acciones de Control de Inyección;
        Esperar al próximo periodo (80 ms);
     end loop;
   end Control_Inyección;

begin
   null; -- el programa principal es otra tarea
end Control_del_Automovil;
```


Concurrencia en Ada: Ejemplo sencillo

UNIVERSIDAD DE CANTABRIA

```
with MaRTE_OS;
with Text_IO; use Text_IO;

procedure Tareas is


   task Tarea_1;

   task body Tarea_1 is
   begin
      loop
         Put_Line ("Tarea 1");
         delay 1.0;  -- espera 1 seg
     end loop;
   end Tarea_1;
```

Concurrencia en Ada: Ejemplo sencillo (cont.)

UNIVERSIDAD DE CANTABRIA

```
   task Tarea_2;

   task body Tarea_2 is
   begin
      loop
         Put_Line ("Tarea 2");
         delay 2.0;  -- espera 2 seg
      end loop;
   end Tarea_2;


begin
   loop
      Put_Line ("Programa principal");
      delay 3.0;  -- espera 3 seg
   end loop;
end Tareas;
```


Gestión del Tiempo

UNIVERSIDAD DE CANTABRIA

El Ada permite diversas formas de manejo del tiempo:

- Package `Ada.Calendar`, con funciones para saber la hora, fecha, etc.
- Package `Ada.Real_Time` para acceder a un reloj monótono, que se incrementa siempre
- Instrucciones `delay until` y `delay`, que permiten dormir a una tarea hasta que transcurra el instante o intervalo indicado
  - mientras una tarea está dormida, otras pueden ejecutar

Paquete Ada.Calendar

UNIVERSIDAD DE CANTABRIA

```
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;
```


Paquete Ada.Calendar (cont.)

UNIVERSIDAD DE CANTABRIA

```
   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;
   ...
end Ada.Calendar;
```

Paquete Ada.Calendar
Ejemplo: mide intervalo

UNIVERSIDAD DE CANTABRIA

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

procedure Mide_Dif_Tiempo is
   T_Ini, T_Fin : Time;
   Diferencia : Duration;

begin
   T_Ini := Clock;  --  lee hora de inicio
   ...  --  Código bajo medida
   T_Fin := Clock;  --  lee hora de finalización

   Diferencia := T_Fin - T_Ini; -- calcula la diferencia

   Put_Line("Diferencia :" & Duration'Image (Diferencia));
end Mide_Dif_Tiempo;
```


Paquete Ada.Calendar
Ejemplo: lee fecha y hora

UNIVERSIDAD DE CANTABRIA

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

procedure Muestra_Dia_Y_Hora is
   Instante : Time := Clock;
   Hora     : Integer := Integer (Seconds (Instante)) /
                                  3600;
   Minuto   : Integer := (Integer (Seconds (Instante)) -
                                   Hora*3600) / 60;
begin
   Put_Line("Hoy es " & Integer'Image(Day(Instante)) &
            " del " & Integer'Image(Month(Instante)) &
            " de " & Integer'Image(Year(Instante)));
   Put_Line("La hora es : " & Integer'Image(Hora) &
            ":" & Integer'Image(Minuto));
end Muestra_Dia_Y_Hora;
```

Tareas periódicas
Instrucción "delay until"

UNIVERSIDAD DE CANTABRIA

Con la instrucción `delay until` es posible hacer tareas periódicas:

```
task body Periodica is
    Periodo : constant Duration := 0.050; -- en segundos
    Proximo_Periodo : Time := Clock;
begin
    loop
        -- hace cosas
        ...
        Proximo_Periodo := Proximo_Periodo + Periodo;
        delay until Proximo_Periodo;
    end loop;
end Periodica;
```


Tareas periódicas
Instrucción "delay"

UNIVERSIDAD DE CANTABRIA

También se puede hacer lo mismo (*pero mal*) con la orden delay:

```
task body Periodica is
    Periodo : constant Duration := 0.050; -- en segundos
    Proximo_Periodo : Time := Clock;
begin
    loop
        -- hace cosas
        Proximo_Periodo := Proximo_Periodo + Periodo;
        delay Proximo_Periodo - Clock;
    end loop;
end Periodica;
```

En este caso la tarea puede ser interrumpida por otra entre la lectura del reloj (con `Clock`) y la ejecución del `delay`.

- **Esto provocaría un retraso mayor que el deseado.**

Política de planificación

UNIVERSIDAD DE CANTABRIA

El Ada utiliza el sistema de tareas expulsoras por prioridad fija con orden FIFO dentro de la misma prioridad:

- cuando una tarea de alta prioridad se activa, expulsa a otra tarea de menor prioridad que se esté ejecutando

A cada tarea se le asigna una prioridad:

```
pragma Priority(13);
```

Mediante la planificación de tareas por prioridades fijas, es posible garantizar la respuesta en tiempo real de las tareas

- La teoría RMA permite realizar el análisis del sistema
- Asignación óptima de prioridades: inversamente proporcional al periodo de las tareas (a menor periodo, mayor prioridad)

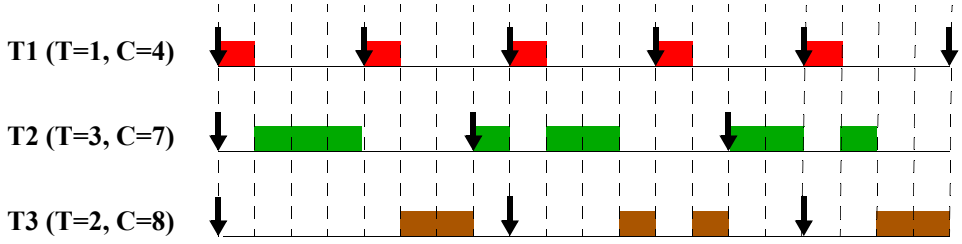
Política de planificación (cont.)

UNIVERSIDAD DE CANTABRIA

```
task T1 is
 pragma Priority(3);
end T1;
task body T1 is
begin
 loop
   ejecuta 1 seg;
   Prox := Prox + 4;
   delay until Prox;
 end loop;
end T1;
```

```
task T2 is
 pragma Priority(2);
end T2;
task body T2 is
begin
 loop
   ejecuta 3 seg;
   Prox := Prox + 7;
   delay until Prox;
 end loop;
end T2;
```

```
task T3 is
 pragma Priority(1);
end T3;
task body T3 is
begin
 loop
   ejecuta 2 seg;
   Prox := Prox + 8;
   delay until Prox;
 end loop;
end T3;
```



Ejemplo completo de tareas periódicas

UNIVERSIDAD DE CANTABRIA

```
with MaRTE_OS;
with Ada.Calendar, Text_IO, Execution_Load;
use  Ada.Calendar, Text_IO;
procedure Simple_Tasks_Delay_Until_Calendar is
   pragma Priority (1); -- Prioridad del procedimiento principal
   task T1 is
      pragma Priority (3);
   end T1;
   task body T1 is
      Prox_Activacion : Time := Clock;
   begin
      loop
         Put_Line ("      <- Tarea con prioridad 3 empieza");
         Execution_Load.Eat (1.0);
         Prox_Activacion := Prox_Activacion + 4.0;
         Put_Line ("      -- Tarea con prioridad 3 acaba");
         delay until Prox_Activacion;
      end loop;
   end T1;
```


Ejemplo completo de tareas periódicas (cont.)

UNIVERSIDAD DE CANTABRIA

```
   task T2 is
      pragma Priority (2);
   end T2;
   task body T2 is
      Prox_Activacion : Time := Clock;
   begin
      loop
         Put_Line ("   <- Tarea con prioridad 2 empieza");
         Execution_Load.Eat (2.0);
         Prox_Activacion := Prox_Activacion + 7.0;
         Put_Line ("   -- Tarea con prioridad 2 acaba");
         delay until Prox_Activacion;
      end loop;
   end T2;
```

Ejemplo completo de tareas periódicas (cont.)

UNIVERSIDAD DE CANTABRIA

```
   Prox_Activacion : Time := Clock;
begin
   loop
      Put_Line ("<- Principal con prioridad 1 empieza");
      Execution_Load.Eat (6.0);
      Prox_Activacion := Prox_Activacion + 11.0;
      Put_Line ("-- Principal con prioridad 1 acaba");
      delay until Prox_Activacion;
   end loop;
end Simple_Tasks_Delay_Until_Calendar;
```


Sincronización de datos

UNIVERSIDAD DE CANTABRIA

Normalmente el acceso a un conjunto de datos relacionados entre sí debe realizarse de manera mutuamente exclusiva:

- **exclusión mutua: dos tareas no deben acceder al conjunto de datos simultáneamente, mientras una accede, la otra tiene que esperar**

Exclusión mutua en Ada: *objetos protegidos*

- **permiten definir de forma conjunta:**
  - **los datos protegidos**
  - **las funciones y procedimientos para acceder a los datos**
- **mientras una tarea está ejecutando un procedimiento las demás que tratan de acceder a los datos deben esperar**
  - **varias funciones (lecturas) pueden ejecutarse a la vez**

Sincronización de datos: Objetos Protegidos

UNIVERSIDAD DE CANTABRIA

Especificación

```
protected Datos is
  pragma Priority (10);
  procedure Proc1 (...);
  function Funct1 (...)
     return ...;
private
  Datos protegidos;
end Datos;
```

Funciones y procedimientos para acceder a los datos protegidos

Datos para los que se desea imponer el acceso mutuamente exclusivo por parte de las tareas

Cuerpo

```
protected body Datos is
  procedure Proc1 (...) is
  begin
    ...
  end Proc1;
  function Funct1 (...)
     return ... is
  ...
end Datos;
```

Cuerpo de las funciones y procedimientos definidos en la especificación

Problemas si no se usa Objeto Protegido

UNIVERSIDAD DE CANTABRIA

```
with MaRTE_OS;
with Calendar, Text_IO;
use Calendar, Text_IO;
procedure Error_Sincro_No_PO is
  A, B, C : Integer := 0;  --  Datos compartidos

  task Muestra_Datos is
    pragma Priority (4);
  end Muestra_Datos;
  task body Muestra_Datos is
    Proxima_Activacion : Time := Clock;
    Periodo : Duration := 1.0;
  begin
    loop
      Put_Line ("A:" & Integer'Image (A) & "  B:" &
                Integer'Image (B) & "  C:" & Integer'Image (C));
      Proxima_Activacion := Proxima_Activacion + Periodo;
      delay until Proxima_Activacion;
    end loop;
  end Muestra_Datos;
```

Problemas si no se usa Objeto Protegido (cont.)

UNIVERSIDAD DE CANTABRIA

```
  task Incrementa is
    pragma Priority (3);
  end Incrementa;

  task body Incrementa is
    Proxima_Activacion : Time := Clock;
    Periodo : Duration := 1.3;
  begin
    loop
      A := A + 1;
      B := B + 1;
      C := C + 1;

      Proxima_Activacion := Proxima_Activacion + Periodo;
      delay until Proxima_Activacion;
    end loop;
  end Incrementa;
begin
   null;
end Error_Sincro_No_PO;
```


Problemas si no se usa Objeto Protegido (cont.)

UNIVERSIDAD DE CANTABRIA

Valores mostrados por la tarea `Muestra_Datos`:

```
A: 0  B: 0  C: 0
A: 1  B: 1  C: 1
A: 2  B: 2  C: 2
A: 3  B: 3  C: 3
A: 4  B: 3  C: 3
A: 4  B: 4  C: 4
A: 5  B: 5  C: 5
...
```

¡¡Resultado no deseado!!

Comportamiento correcto utilizando Objeto Protegido

UNIVERSIDAD DE CANTABRIA

```
with MaRTE_OS;
with Calendar; use Calendar;
with Text_IO; use Text_IO;

procedure No_Error_Sincro_Con_PO is

   --  Objeto Protegido "Datos compartidos"
   protected Datos_Compartidos is
      pragma Priority (4);

      procedure Escribe_Datos
                  (Dato_A, Dato_B, Dato_C : in Integer);
      procedure Lee_Datos
                  (Dato_A, Dato_B, Dato_C : out Integer);
   private
      A, B, C : Integer := 0;
   end Datos_Compartidos;
```


Comportamiento correcto utilizando Objeto Protegido

UNIVERSIDAD DE CANTABRIA

```
   protected body Datos_Compartidos is

      procedure Escribe_Datos
                  (Dato_A, Dato_B, Dato_C : in Integer) is
      begin
         A := Dato_A;
         B := Dato_B;
         C := Dato_C;
      end Escribe_Datos;


      procedure Lee_Datos
                  (Dato_A, Dato_B, Dato_C : out Integer) is
      begin
         Dato_A := A;
         Dato_B := B;
         Dato_C := C;
      end Lee_Datos;

   end Datos_Compartidos;
```

Comportamiento correcto utilizando Objeto Protegido (cont.)

UNIVERSIDAD DE CANTABRIA

```
task Muestra_Datos is
  pragma Priority (4);
end Muestra_Datos;

task body Muestra_Datos is
  Proxima_Activacion : Time := Clock;
  Periodo : Duration := 1.0;
  A, B, C : Integer;
begin
  loop
    Datos_Compartidos.Lee_Datos (A, B, C);
    Put_Line ("A:" & Integer'Image (A) & "  B:" &
              Integer'Image (B) & "  C:" & Integer'Image (C));

    Proxima_Activacion := Proxima_Activacion + Periodo;
    delay until Proxima_Activacion;
  end loop;
end Muestra_Datos;
```


Comportamiento correcto utilizando Objeto Protegido

UNIVERSIDAD DE CANTABRIA

```
task Incrementa is
  pragma Priority (3);
end Incrementa;

task body Incrementa is
  Proxima_Activacion : Time := Clock;
  Periodo : Duration := 1.3;
  A, B, C : Integer;
begin
  loop
    Datos_Compartidos.Lee_Datos (A, B, C);
    A := A + 1;     B := B + 1;     C := C + 1;
    Datos_Compartidos.Escribe_Datos (A, B, C);

    Proxima_Activacion := Proxima_Activacion + Periodo;
    delay until Proxima_Activacion;
  end loop;
end Incrementa;
begin
   null;
end No_Error_Sincro_Con_PO;
```

Inversión de prioridad

UNIVERSIDAD DE CANTABRIA

En la sincronización de datos puede aparecer la inversión de prioridad:

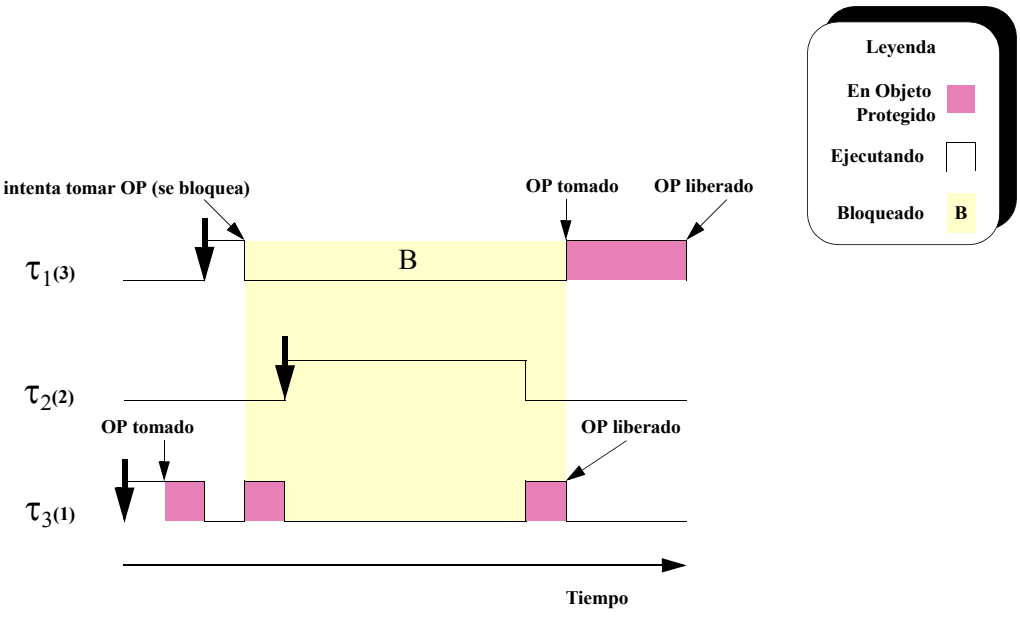
- una tarea de alta prioridad puede verse forzada a esperar a que una de menor prioridad termine
- esto provoca retrasos enormes, inaceptables en sistemas de tiempo real

Para evitarla existe el protocolo de techo de prioridad, asociado a los objetos protegidos:

- a cada objeto protegido se le asigna un techo de prioridad
- con el `pragma Priority`
- debe ser igual o superior a las prioridades de las tareas que usan ese objeto protegido

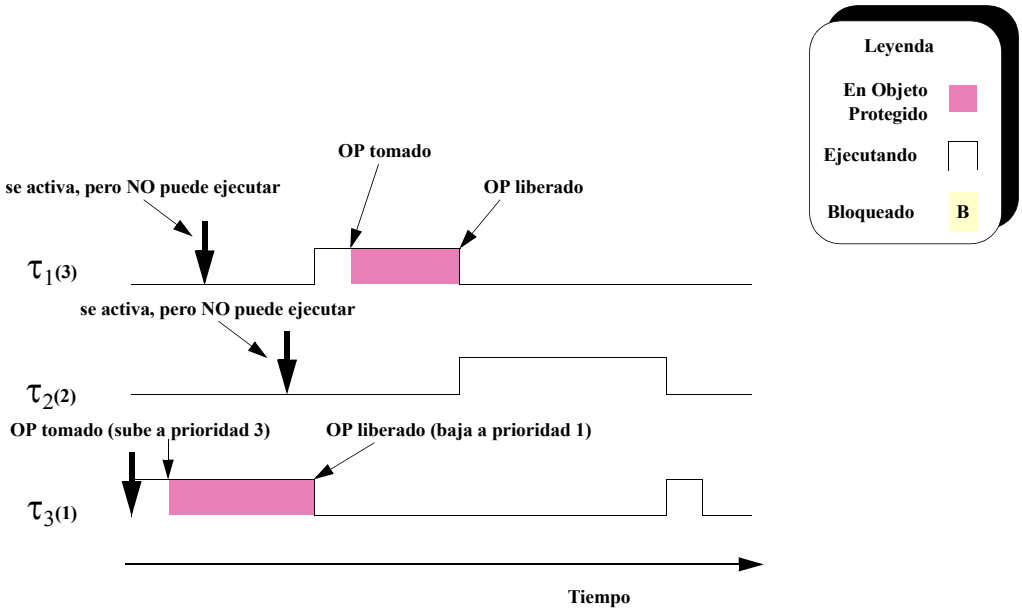
Inversión de prioridad (cont.)

UNIVERSIDAD DE CANTABRIA



Protocolo de techo de prioridad

UNIVERSIDAD DE CANTABRIA



Técnicas básicas de entrada/salida
Interacción computador/dispositivo

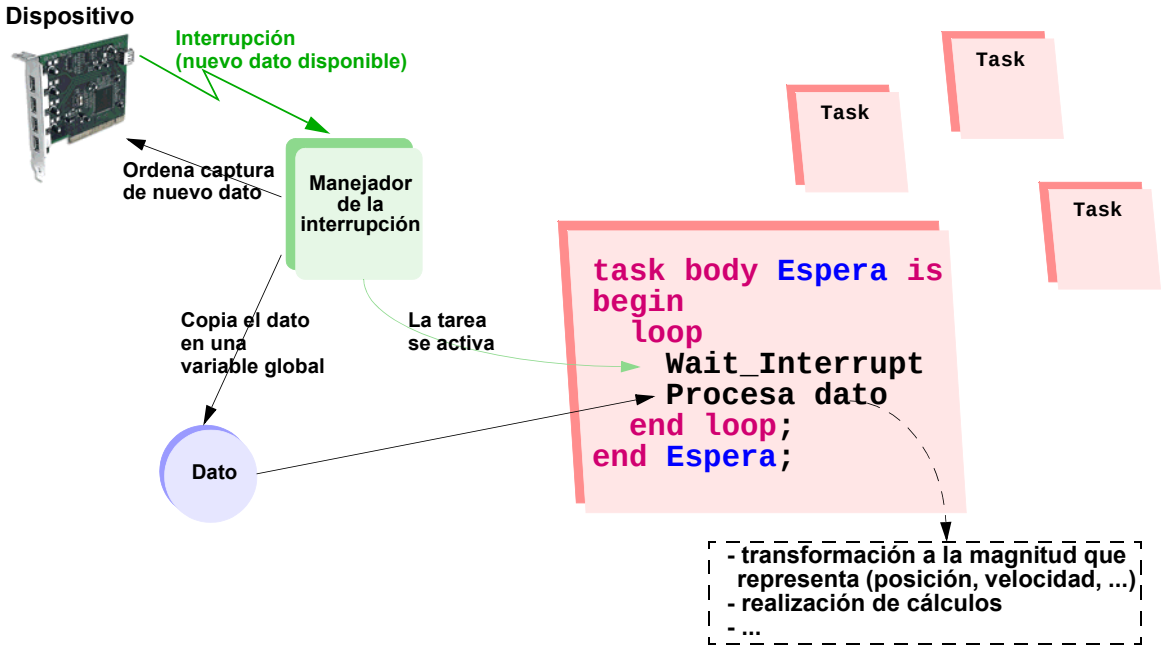
UNIVERSIDAD DE CANTABRIA

Puede realizarse siguiendo dos esquemas básicos:

- **Interacción controlada por el programa**
  - **el programa debe estar constantemente comprobando si se ha finalizado cada operación de Entrada/Salida**
  - **se denomina "espera activa" o "encuesta"**
- **Interacción basada en interrupciones**
  - **la interacción con el dispositivo la realiza el manejador de la interrupción**
  - **una tarea se suspende a la espera de la interrupción**
    - **se activa cuando finaliza la ejecución del manejador**
  - **mientras la tarea está suspendida las demás pueden seguir realizando su trabajo**

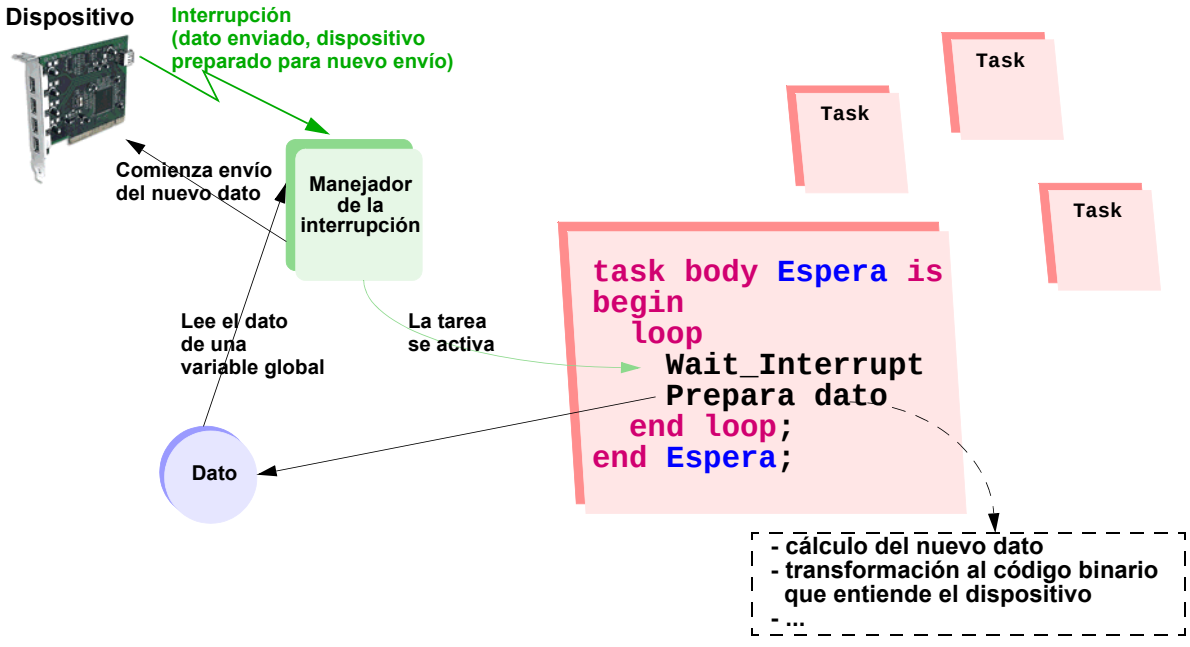
Interacción basada en interrupciones

UNIVERSIDAD DE CANTABRIA



Interacción basada en interrupciones (cont.)

UNIVERSIDAD DE CANTABRIA



Operación de espera a una interrupción

UNIVERSIDAD DE CANTABRIA

El paquete `MaRTE_Hardware_Interrupts_Wait` proporciona el procedimiento:

```
procedure Wait_Associated_Interrupt;
```

- permite a una tarea bloquearse a la espera de que se ejecute un manejador de interrupción
- previamente la tarea debe haberse asociado con la interrupción instalando un manejador con:
  `MaRTE_Hardware_Interrupts.Associate`
- El manejador puede elegir NO activar la tarea a la espera retornando `POSIX_INTR_HANDLED_DO_NOT_NOTIFY`
  - en lugar de retornar `POSIX_INTR_HANDLED_DO_NOTIFY` como hemos hecho hasta ahora

Ejemplo: espera de la interrupción del puerto paralelo

UNIVERSIDAD DE CANTABRIA

```
with MaRTE_OS;
with MaRTE_Hardware_Interrupts, MaRTE_Hardware_Interrupts_Wait;
use  MaRTE_Hardware_Interrupts, MaRTE_Hardware_Interrupts_Wait;
with IO_Interface; use IO_Interface;
with Basic_Integer_Types; use Basic_Integer_Types;
with System;
with Basic_Console_IO, Text_IO, Ada.Integer_Text_IO;
use Text_IO, Ada.Integer_Text_IO;

procedure Espera_Int_Puerto_Paralelo is

   -- Registros del puerto paralelo
   PP_BASE        : constant IO_Port := 16#378#; -- Puerto 1
   PP_DATA_REG    : constant IO_Port := PP_BASE + 0;
   PP_STATUS_REG  : constant IO_Port := PP_BASE + 1;
   PP_CONTROL_REG : constant IO_Port := PP_BASE + 2;

   -- Byte leido del puerto
   Data_Read : Unsigned_8;
   pragma Volatile (Data_Read);
```

Ejemplo: espera de la interrupción del puerto paralelo (cont.)

UNIVERSIDAD DE CANTABRIA

```
   -- Manejador de interrupción del puerto paralelo
   function PP_Handler (Area : in System.Address;
                        Intr : in Hardware_Interrupt)
                       return Handler_Return_Code is
   begin
      Data_Read := IO_Interface.Inb (PP_DATA_REG);
      return POSIX_INTR_HANDLED_NOTIFY;
   end PP_Handler;

begin
   -- Instala manejador de interrupción
   if Associate (PARALLEL1_INTERRUPT,
                 PP_Handler'Unrestricted_Access,
                 System.Null_Address, 0) /= 0 then
      Put_Line ("Error: Associate");
   end if;

   -- Configura como puerto de entrada y habilita las
   -- interrupciones del puerto paralelo
   IO_Interface.Outb (PP_CONTROL_REG, 2#00_1_1_0000#);
```


Ejemplo: espera de la interrupción del puerto paralelo (cont.)

UNIVERSIDAD DE CANTABRIA

```
   -- Habilita la interrupción en el PIC
   if Unlock (PARALLEL1_INTERRUPT) /= 0 then
      Put_Line ("Error: Unlock");
   end if;

   -- Espera a que ocurra la interrupción y muestra
   -- el valor leido
   loop
      MaRTE_Hardware_Interrupts_Wait.Wait_Associated_Interrupt;
      Put ("Byte leido:");
      Put (Integer (Data_Read), Base => 16);
      New_Line;
   end loop;

end Espera_Int_Puerto_Paralelo;
```