Bloque II

- **Programación de tiempo real**

© Michael González, J. Javier Gutiérrez, Héctor Pérez
21/mar/12

Programación de Tiempo Real

UNIVERSIDAD DE CANTABRIA

El Ada soporta la programación de sistemas empotrados y de sistemas de tiempo real, mediante los siguientes mecanismos:

- Representación del hardware
- Interrupciones
- Gestión del tiempo
- Prioridades
- Sincronización libre de inversión de prioridad

Estos mecanismos están descritos en anexos opcionales:

- Anexo de programación de sistemas
- Anexo de sistemas de tiempo real

© Michael González, J. Javier Gutiérrez, Héctor Pérez

Representación del Hardware

UNIVERSIDAD DE CANTABRIA

En muchos sistemas los programas deben interaccionar directamente con el hardware

Muchas veces se utiliza para ello lenguaje ensamblador

El Ada proporciona mecanismos para establecer la correspondencia entre estructuras lógicas y representaciones físicas del hardware. Por ejemplo:

- asociación entre campos de registros y posiciones de bits
- posicionamiento de datos en direcciones de memoria concretas
- control de optimización

© Michael González, J. Javier Gutiérrez, Héctor Pérez
21/mar/12

Interrupciones

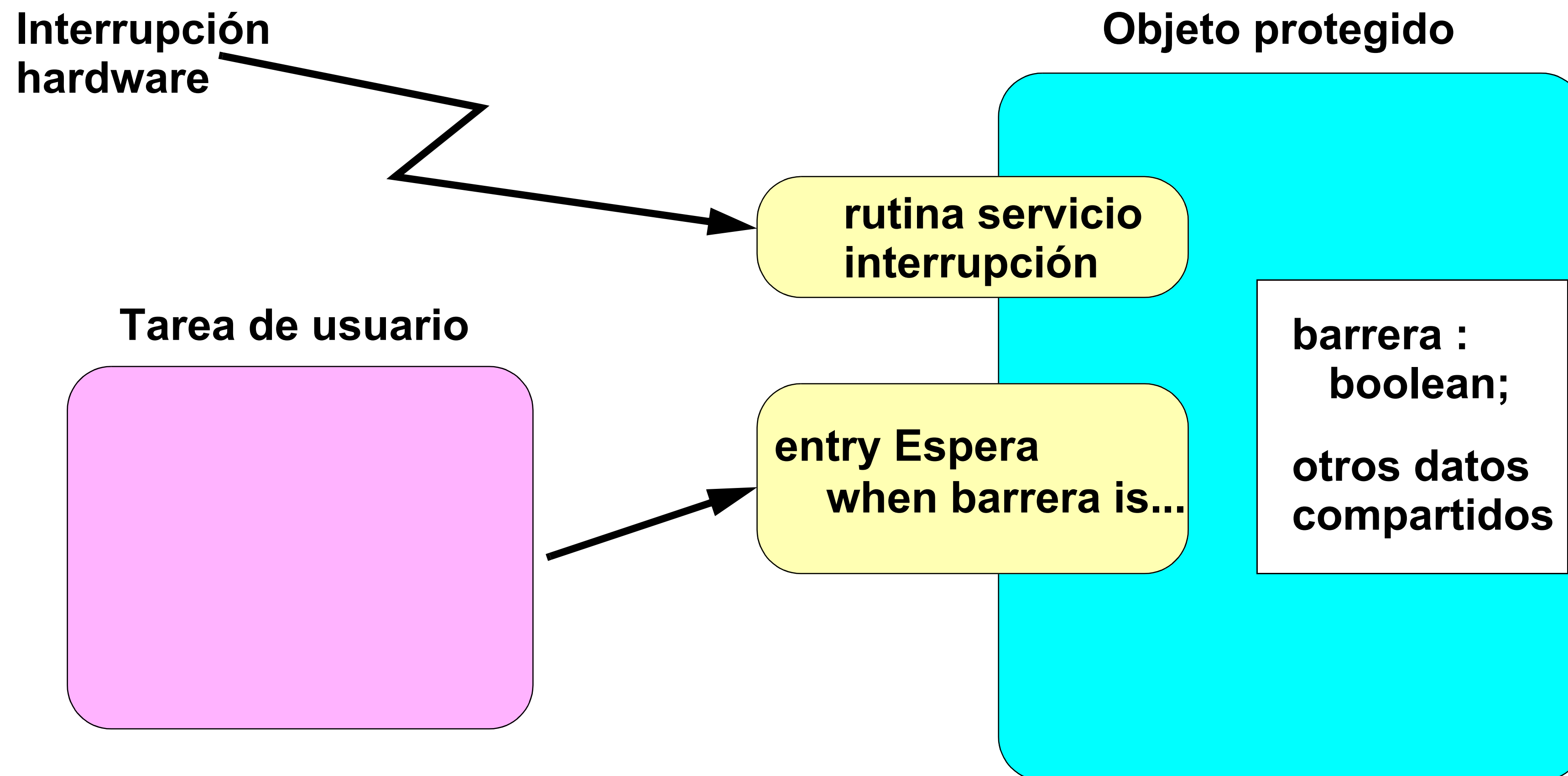
UNIVERSIDAD DE CANTABRIA

En los sistemas empotrados es preciso atender a las interrupciones provocadas por el hardware externo.

En Ada 05 se permite especificar un procedimiento protegido como elemento de manejo de interrupciones.

- en este modelo, el hardware invoca directamente la ejecución del procedimiento protegido
- sólo funciona si el sistema operativo lo permite

Ejemplo: Tarea que espera a una interrupción



Gestión del Tiempo

UNIVERSIDAD DE CANTABRIA

El Ada permite diversas formas de manejo del tiempo integradas en el lenguaje:

- Relojes
  - Package `Ada.Calendar`
  - Package `Ada.Real_Time`
- Relojes de ejecución
- Retardos
  - Instrucciones `delay until` y `delay`

El tipo predefinido *Duration* representa intervalos de tiempo en segundos

Relojes

UNIVERSIDAD DE CANTABRIA

En Ada hay dos paquetes predefinidos que proporcionan funciones de reloj:

- Ada.Calendar
  - Define un tipo *Time* que representa la fecha y la hora
  - El reloj se supone sincronizado mediante una referencia externa
  - Los intervalos de tiempo se representan con el tipo predefinido *Duration*
- Ada.Real_Time
  - Define un tipo *Time* que representa valores de tiempo absolutos mediante el acceso a un reloj monótono no decreciente
  - Los intervalos de tiempo se representan con el tipo abstracto *Time_Span*
  - Existen funciones para convertir Time_Span a Duration

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

Paquete Ada.Calendar

UNIVERSIDAD DE CANTABRIA

```
   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;
```


© Michael González, J. Javier Gutiérrez, Héctor Pérez

Paquete Ada.Real_Time

UNIVERSIDAD DE CANTABRIA

```
package Ada.Real_Time is
   type Time is private;
   Time_First : constant Time;
   Time_Last  : constant Time;
   Time_Unit  : constant := -- real number;
   type Time_Span is private;
   Time_Span_First : constant Time_Span;
   Time_Span_Last  : 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 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;
end Ada.Real_Time;
```

Ejemplo

UNIVERSIDAD DE CANTABRIA

```
-- Imprimir en pantalla el día, mes y año actuales,
-- y luego las horas y minutos

with Ada.Calendar, Ada.Text_Io;
use Ada.Calendar; use 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;
```

Relojes de tiempo de ejecución

UNIVERSIDAD DE CANTABRIA

Son relojes que miden el tiempo de CPU utilizado por una tarea

- No incluyen esperas, bloqueos, etc.

Define funciones similares al paquete Ada.Real_Time

```
function Clock (T : Task_Id := Current_Task)
 return CPU_Time;
```

También existen temporizadores de tiempo de ejecución que pueden ser asociados a una tarea

- Al expirar el tiempo, se realizan acciones determinadas

Retrasos

UNIVERSIDAD DE CANTABRIA

Instrucciones que permiten suspender una tarea hasta que transcurra el instante o intervalo indicado

- mientras una tarea está dormida, otras pueden ejecutar

```
delay Tipo_Duration;
delay until Tipo_Time;
```

Ejemplo: Invocar procedimiento periódicamente

¿Ambas son correctas?

```
Intervalo : Duration := 5*Minutos;
loop
   delay Intervalo;
   Proceso_A_Ejecutar;
end loop;
```

```
Intervalo : Duration := 5*Minutos;
Next_Time : Time := Tiempo_Inicial;
begin
   loop
      delay until Next_Time;
      Proceso_A_Ejecutar;
      Next_Time :=
         Next_Time + Intervalo;
   end loop;
```

Tareas periódicas

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
        delay until Proximo_Periodo;
        -- acción periódica
        Proximo_Periodo := Proximo_Periodo + Periodo;
    end loop;
end Periodica;
```


© Michael González, J. Javier Gutiérrez, Héctor Pérez

Tareas periódicas (cont'd)

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
      -- realiza operaciones
      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

Prioridades Fijas

UNIVERSIDAD DE CANTABRIA

Ada puede configurar el entorno para utilizar el sistema de tareas expulsoras (o desalojantes) por prioridad fija:

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

A cada tarea se le asigna una prioridad:

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

Esta prioridad se puede modificar, usando las operaciones del paquete `Ada.Dynamic_Priorities`

- La prioridad es un subtipo definido en el paquete *System*

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

© Michael González, J. Javier Gutiérrez, Héctor Pérez
21/mar/12

Ada.Dynamic_Priorities

UNIVERSIDAD DE CANTABRIA

```
with System;
with Ada.Task_Identification;
package Ada.Dynamic_Priorities is
   procedure Set_Priority
    (Priority : in System.Any_Priority;
     T        : in Ada.Task_Identification.Task_Id :=
                     Ada.Task_Identification.Current_Task);
   function Get_Priority
    (T : Ada.Task_Identification.Task_Id :=
          Ada.Task_Identification.Current_Task)
   return System.Any_Priority;
end Ada.Dynamic_Priorities;
```

Tareas periódicas (modelo)

UNIVERSIDAD DE CANTABRIA

Parámetros discretos o tipos acceso

```
task type Periodica (Prioridad    : System.Priority;
                     Periodo      : access Duration) is
     pragma Priority (Prioridad);
end Periodica;

task body Periodica is
    Proximo_Periodo : Time := Clock;
begin
    loop
        delay until Proximo_Periodo;
        -- acción periódica
        Proximo_Periodo := Proximo_Periodo + Periodo;
    end loop;
```

Ada.Real_Time
o
Ada.Calendar

Políticas de planificación

UNIVERSIDAD DE CANTABRIA

Políticas definidas en Ada 2005

- FIFO_Within_Priorities
  - Prioridades fijas expulsora y orden FIFO para el mismo nivel de priorifad
- Non_Preemptive_FIFO_Within_Priorities
  - Prioridades fijas no expulsora y orden FIFO para el mismo nivel de priorifad
- Round_Robin_Within_Priorities
  - Prioridades fijas expulsora y se realizan turnos circulares para el mismo nivel de priorifad
- EDF_Across_Priorities:
  - Basada en plazos temporales

Políticas de planificación (cont'd)

UNIVERSIDAD DE CANTABRIA

Se define la política de planificación para todo el sistema mediante:

```
pragma Task_Dispatching_Policy (Policy);
```

Se pueden mezclar distintas políticas de planificación (planificación jerárquica) según intervalos de prioridad

```
pragma Priority_Specific_Dispatching
 (Policy, Low_Priority, High_Priority);
```

Estos pragmas son directivas de compilador y nos permiten configurar explícitamente nuestra aplicación

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
- debe ser igual o superior a las prioridades de las tareas que usan ese objeto protegido

© Michael González, J. Javier Gutiérrez, Héctor Pérez
21/mar/12

Techos de prioridad y gestión de colas

UNIVERSIDAD DE CANTABRIA

Se define el uso del protocolo de techo de prioridad en un objeto protegido mediante:

```
pragma Locking_Policy (Ceiling_Locking);
```

El techo de prioridad de un objeto o un tipo protegido se especifica mediante un pragma *Priority*

```
protected type OP (Techo : System.Priority) is
   pragma Priority (Techo);
   ...
end OP;
```

El orden de entrada a los *entries* se controla con el pragma Queuing_Policy

```
pragma Queuing_Policy (Priority_Queuing);
pragma Queuing_Policy (FIFO_Queuing);
```

Ejemplo de tareas concurrentes con planificación por prioridades fijas

```
--  Fichero gnat.adc
pragma Locking_Policy (Ceiling_Locking);  -- Politica de sincronizacion
pragma Queuing_Policy (Priority_Queuing); -- Politica de encolado
pragma Task_Dispatching_Policy (FIFO_Within_Priorities); --  Politica de planif.

--  Fichero test_rt_app.adb
with System;
procedure Test_RT_App is

   task type Worker (The_Priority : System.Priority) is
      pragma Priority (The_Priority); --  Prioridad de la tarea
   end Worker;

   task body Worker is
   begin
      ...
   end Worker;

   Tarea_1 : Worker (The_Priority => 10);
   Tarea_2 : Worker (The_Priority => 15);
begin
   ...
end Test_RT_App;
```

Bibliografía

UNIVERSIDAD DE CANTABRIA

[1] J. BARNES. "Programming in Ada 2005", first edition. Addison-Wesley, 2006.

[2] J. BARNES. "Programming In Ada 95", first edition. Addison-Wesley, 1995.

[3] S. Tucker Taft, Robert A. Duff, Randall L. Brukardt, Erhard Ploedereder, and Pascal Leroy (Eds.). “Ada 2005 Reference Manual. Language and Standard Libraries. International Standard ISO/IEC 8652:1995(E) with Technical Corrigendum 1 and Amendment 1”. LNCS 4348, Springer, 2006.

Productos libres de AdaCore: https://libre.adacore.com/