Añadiendo un Soporte Eficiente para Sistemas Multiprocesadores al lenguaje Ada 2012

S. Sáez J. Real A. Crespo

Instituto de Automática e Informática Industrial
Universidad Politécnica de Valencia

Jornada Técnica de AdaSpain 2013, Madrid

Index

1. Introducción
2. Requisitos para la planificación en multiprocesadores
3. Propuesta 1: Cambio atómico y/o diferido de atributos en Ada
4. Propuesta 2: Timing Events con afinidad

Índice

1 Introducción

2 Requisitos para la planificación en multiprocesadores

3 Propuesta 1: Cambio atómico y/o diferido de atributos en Ada

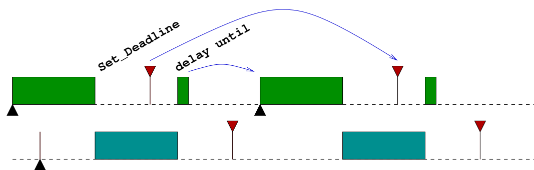
4 Propuesta 2: Timing Events con afinidad

Atributos de Planificación de una Tarea en Ada 2012

- Los atributos principales que afectan a la planificación de una tarea de tiempo real en Ada 2012 son la prioridad, el plazo de entrega y la CPU.
- Los valores iniciales se establecen mediante aspectos: Priority , Relative_Deadline y CPU.
- Su valor se puede cambiar en tiempo de ejecución mediante los procedimientos: Ada.Dynamic_Priorities.Set_Priority , Ada.Dispatching.EDF.Set_Deadline y System.Multiprocessors.Dispatching_Domains.Set_CPU
- Todos estos procedimientos son *puntos de planificación* en las políticas expulsivas correspondientes si se ejecutan fuera de una acción protegida.

Artefactos en la planificación de tareas

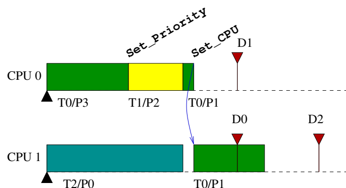
- Si se cambian cualquiera de dichos parámetros antes de una sentencia **delay** se pueden producir artefactos en la planificación por la perdida de prioridad relativa en la CPU actual o en la de destino.

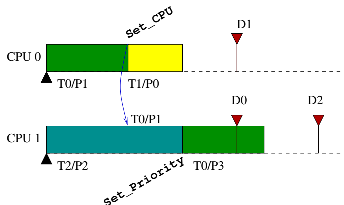


- Para evitarlo Ada 2012 ofrece procedimientos para cambiar el plazo de entrega o la CPU a la vez que se lleva a cabo una operación de **delay until**.
  - Delay_Until_And_Set_Deadline(Next_Release, Rel_Deadline).
  - Delay_Until_And_Set_CPU(Next_Release, CPU).

Artefactos en la planificación de tareas (cont.)

- Sin embargo, no hay soporte para:
  - Cambiar la prioridad conjuntamente con un **delay until**.
  - Cambiar más de un atributo conjuntamente con un **delay until**.
  - Cambiar más de un atributo de forma atómica.
- Estos cambios son habituales en los esquemas de planificación para multiprocesadores *job partitioning* y *task spliting*.





Índice

1 Introducción

2 Requisitos para la planificación en multiprocesadores

3 Propuesta 1: Cambio atómico y/o diferido de atributos en Ada

4 Propuesta 2: Timing Events con afinidad

Requisitos para la planificación sobre multiprocesadores

Permitir el cambio de varios atributos de la tarea ...

- ... de forma atómica desde el punto de vista de la planificación de la misma,
- ... en el instante actual.
- ... en un instante en el futuro.

Ejemplos:

Job partitioning
: En la siguiente activación de una tarea.

Task spliting o *Dual priority*
: En un instante de tiempo programado o tras un cierto tiempo de ejecución, y al principio de la siguiente activación.

Índice

1 Introducción

2 Requisitos para la planificación en multiprocesadores

3 Propuesta 1: Cambio atómico y/o diferido de atributos en Ada

4 Propuesta 2: Timing Events con afinidad

Scheduling Parameters

Un nuevo *tagged type*, Sched_Params, que

- almacene los atributos de planificación que se le deben cambiar a una tarea.
- que permita que se le apliquen de forma atómica ...
- ... inmediatamente,

```
Apply_Sched_Params(Sched_Params, Task_Id)
```

- ... ode forma diferida.

```
Delay_Until_And_Apply_Sched_Params (Sched_Params,
                                    Delay_Until_Time)
```

Scheduling Parameters para prioridades fijas

```
package Ada.Real_Time.Scheduling_Parameters is
   type Sched_Params is tagged private;

   procedure Set_Priority (SP : in out Sched_Params; Prio: Any_Priority );
   function Get_Priority (SP : Sched_Params) return Any_Priority;
   procedure Set_CPU (SP : in out Sched_Params; CPU_Nr: CPU_Range);
   function Get_CPU (SP : Sched_Params) return CPU_Range;

   procedure Apply_Sched_Params (SP : Sched_Params;
                     T_Id : Task_Id := Current_Task);
   procedure Retrieve_Sched_Params (SP : in out Sched_Params;
                     T_Id : Task_Id := Current_Task);
   procedure Delay_Until_And_Apply_Sched_Params (SP : Sched_Params;
                     Delay_Until_Time : Time);
private
   type Sched_Params is
      record
         Prio : Any_Priority := Default_Priority ;
         CPU_Nr : CPU_Range := Not_A_Specific_CPU;
      end record;
end Ada.Real_Time.Scheduling_Parameters;
```

Scheduling Parameters con plazo relativo

⇒ El plazo se establece con respecto al instante de activación.

- Para tareas periódicas con *job partitioning*.
  El plazo y el resto de los atributos se cambiarán en el instante de su siguiente activación.

```
Delay_Until_And_Apply_Sched_Params (Sched_Params,
                                    Delay_Until_Time)
```

- Para tareas esporádicas.
  El plazo y el resto de los atributos se cambiarán en el instante en que algún evento active la tarea, v.g., mediante un *Suspension Object*.

```
Apply_Sched_Params(Sched_Params, Task_Id)
```

Scheduling Parameters con plazo relativo (cont.)

```
package Ada.Real_Time.Scheduling_Parameters.EDF is
   type Sched_Params_With_Rel_Deadline is new Sched_Params with private;

   procedure Set_Deadline (SP : in out Sched_Params_With_Rel_Deadline;
                           D : Time_Span);
   function  Get_Deadline (SP : Sched_Params_With_Rel_Deadline)
                           return Time_Span;

   procedure Apply_Sched_Params (SP : Sched_Params_With_Rel_Deadline;
                           T_Id: Task_Id := Current_Task);
   procedure Retrieve_Sched_Params (SP : in out Sched_Params_With_Rel_Deadline;
                           T_Id: Task_Id := Current_Task);
   procedure Delay_Until_And_Apply_Sched_Params (
                           SP : Sched_Params_With_Rel_Deadline;
                           Delay_Until_Time : Time);
private
   type Sched_Params_With_Rel_Deadline is new Sched_Params with
      record
         Relative_Deadline : Time_Span := Time_Span_Last;
      end record;
end Ada.Real_Time.Scheduling_Parameters.EDF;
```

Scheduling Parameters con plazo absoluto

⇒ El plazo se establece con respecto a un instante de tiempo dado.

- Para tareas periódicas con *job partitioning*.
  El plazo y el resto de los atributos se cambiarán en el instante de su siguiente activación.

```
Delay_Until_And_Apply_Sched_Params (Sched_Params,
                                    Delay_Until_Time)
```

- Para tareas esporádicas.
  En el caso del que el plazo de entrega no dependa de cuando llegue el evento de activación.

```
Apply_Sched_Params(Sched_Params, Task_Id)
```

⇒ Se deben actualizar los parámetros cada vez que se quieren aplicar.

Scheduling Parameters con plazo absoluto (cont.)

```
package Ada.Real_Time.Scheduling_Parameters.EDF is
   type Sched_Params_With_Abs_Deadline is new Sched_Params with private;

   procedure Set_Deadline  (SP : in  out Sched_Params_With_Abs_Deadline;
                            D :  Time);
   function  Get_Deadline  (SP : Sched_Params_With_Abs_Deadline)
                            return  Time;

   procedure Apply_Sched_Params (SP : Sched_Params_With_Abs_Deadline;
                            T_Id:  Task_Id  := Current_Task);
   procedure Retrieve_Sched_Params (SP : in  out Sched_Params_With_Abs_Deadline;
                            T_Id:  Task_Id  := Current_Task);
   procedure Delay_Until_And_Apply_Sched_Params (
                            SP : Sched_Params_With_Abs_Deadline;
                            Delay_Until_Time  :  Time);
private
   type Sched_Params_With_Abs_Deadline is new Sched_Params with
      record
         Absolute_Deadline  :  Time := Time_Last;
      end record;
end Ada.Real_Time.Scheduling_Parameters.EDF;
```

Ejemplo de tarea periódica con *job partitioning*

```
task body Periodic_With_Job_Partitioning is
   type List_Range is mod N;
   Params_List   : array (List_Range) of Sched_Params := (...); -- Decided at design tin
   Params_Iter   : List_Range := List_Range'First ;
   ...
begin
   Task_Initialize ;
   Next_Release := Ada.Real_Time.Clock;
   Next_Params := Param_List(Param_Iter); -- First job parameters
   Next_Params.Apply_Sched_Params(); -- Scheduling parameters for the first activation
   loop
      Task_Main_Loop;
      -- Next job preparation
      Params_Iter := Params_Iter'Succ;
      Next_Params := Params_List(Params_Iter);
      Next_Release := Next_Release + Period;
      -- Suspends the task until the next job activation
      Delay_Until_And_Apply_Sched_Params(Next_Params, Next_Release);
      -- Next job will wake up with the next scheduling parameters applied
   end loop;
end Periodic_With_Job_Partitioning ;
```

Synchronous Task Control para tareas esporádicas

- Se añade un nuevo Suspension_Object que permite establecer nuevos atributos que se aplicarán al salir del Suspension Object.

```
with Ada.Real_Time.Scheduling_Parameters; use Ada.Real_Time.Scheduling_Parameters;
package Ada.Real_Time.Synchronous_Task_Control.Scheduling_Parameters is
   procedure Set_True (
                    S : in out Suspension_Object_With_Sched_Params);
   procedure Set_True_And_Apply_Sched_Params (
                    S : in out Suspension_Object_With_Sched_Params;
                    SP: access all Sched_Params'Class);
   procedure Set_False (
                    S : in out Suspension_Object_With_Sched_Params);
   function Current_State (
                    S : Suspension_Object_With_Sched_Params) return Boolean;
   procedure Suspend_Until_True (
                    S : in out Suspension_Object_With_Sched_Params);
   procedure Suspend_Until_True_And_Apply_Sched_Params (
                    S : in out Suspension_Object_With_Sched_Params;
                    SP: access all Sched_Params'Class);
end Ada.Real_Time.Synchronous_Task_Control.Scheduling_Parameters;
```

Implementación: ¿Qué se necesita en el SOTR?

- Una estructura de datos que represente los parámetros de planificación en el espacio de usuario.
- Un API que permita aplicar/recuperar los parámetros a/desde un *hilo* activo, de forma atómica o cuando el hilo se despierte.

POSIX thread attributes object (TAO)

POSIX current API

```
pthread_attr_{init|destroy}               // initialize and destroy TAO
pthread_attr_{set|get}detachstate         // set/get detach state attribute in TAO
pthread_attr_{set|get}inheritsched        // set/get inherit scheduler attribute in TAO
pthread_attr_{set|get}schedparam          // set/get scheduling parameter attributes in TAO
pthread_attr_{set|get}schedpolicy         // set/get scheduling policy attribute in TAO
```

POSIX non-portable extensions (from Linux)

```
pthread_attr_{set|get} affinity_np        // set/get CPU affinity attribute in TAO
pthread_getattr_np                        // get attributes of created thread
```

Extensiones propuestas para POSIX

⇒ Extender el API para soportar el establecimiento de atributos.

POSIX extensions

```
#include <pthread.h>

// Para implementar Apply_Sched_Params
int pthread_setattr_np (pthread_t thread, pthread_attr_t *attr);
// Para implementar Delay_Until_And_Apply_Sched_Params
int pthread_setattr_on_suspend_np ( pthread_attr_t *attr);
```

- El cambio sólo afectaría a los atributos de planificación. El resto permanecerían inmutables.
- pthread_setattr_np Establecería los atributos de planificación de forma atómica del hilo especificado.
- pthread_setattr_on_suspend_np Establece los siguientes atributos de planificación para la tarea invocante cuando se suspenda.

Implementación en el Run-Time de Ada

```
procedure Apply_Sched_Params
  (SP:          : Sched_Params;
   T_Id         : Task_Id : Current_Task)
is
   Attributes  : aliased  pthread_attr_t ;
   Result      : Interfaces .C. int ;
begin
   -- Retrieve the current thread attributes
   Get_Task_Attributes ( Attributes 'Access, T_Id);
   -- Modify the task attributes
   Set_Attr_Priority ( Attributes 'Access, SP.Prio);
   Set_Attr_CPU(Attributes'Access, SP.CPU_Nr);

   -- Set the new thread attributes immediately
   Result := pthread_setattr_np (T_Id.Common.LL.Thread, Attributes'Access);
   pragma Assert (Result = 0);
end Apply_Sched_Params;
```

Implementación en el Run-Time de Ada (cont.)

```
procedure Delay_Until_And_Apply_Sched_Params
  (SP:                : Sched_Params;
   Delay_Until_Time   : Ada.Real_Time.Time;
   T_Id               : Task_Id : Current_Task)
is
   Attributes  : aliased  pthread_attr_t ;
   Result      : Interfaces .C. int ;
begin
   -- Retrieve the current thread attributes
   Get_Task_Attributes ( Attributes 'Access, T_Id );
   -- Modify the task attributes
   Set_Attr_Priority ( Attributes 'Access, SP.Prio );
   Set_Attr_CPU(Attributes'Access, SP.CPU_Nr);

   -- Take note of the new thread attributes to be applied upon thread wake up
   Result := pthread_setattr_on_suspend_np ( Attributes 'Access);
   pragma Assert (Result = 0);

   delay until Delay_Until_Time; -- New scheduling attributes take effect on wake up
end Delay_Until_And_Apply_Sched_Params;
```

Implementación en el Run-Time de Ada (cont.)

```
type Suspension_Object_With_Sched_Params is record
   State : Boolean;
   pragma Atomic (State);
   -- Boolean that indicates whether the object is open.

   Waiting : Boolean;
   -- Flag showing if there is a task already suspended on this object
   L : aliased System.OS_Interface.pthread_mutex_t;
   -- Protection for ensuring mutual exclusion on the Suspension_Object
   CV : aliased System.OS_Interface.pthread_cond_t;
   -- Condition variable used to queue threads until condition is signaled

   -- Scheduling parameters
   SP : access all Sched_Params'Class;
   -- Scheduling Parameters to be applied to the suspended task
   T_Id : Task_Id;
   -- Task suspended within the Suspension Object
end record;
```

Implementación en el Run-Time de Ada (cont.)

```
procedure Suspend_Until_True_And_Apply_Sched_Params
   (S  : in out Suspension_Object_With_Sched_Params;
    SP : access all Sched_Params'Class) is
begin
   if S.Waiting then
      ...  -- Error
   else
      if S.State then
         S.State := False;
         SP.Apply_Sched_Params; -- Change your sched parameters immediately
      else
         S.Waiting := True;
         S.T_Id := Current_Task; -- Program your sched parameters to be changed
         S.SP := SP;             -- by the task that wakes you up
         loop
            Result := pthread_cond_wait (S.CV'Access, S.L'Access);
            exit when not S.Waiting;
         end loop;
      end if;
      ...
   end if;
end Suspend_Until_True;
```

Implementación en el Run-Time de Ada (cont.)

```
procedure Set_True
   (S : in out Suspension_Object_With_Sched_Params) is
begin
     ...
    if S.Waiting then
       S.Waiting := False;
       S.State := False;

       if S.SP /= null then
           S.SP.Apply_Sched_Params(S.T_Id);
       end if;

       Result := pthread_cond_signal (S.CV'Access);
    else
       S.State := True;
    end if;
     ...
end Set_True;
```

Índice

1 Introducción

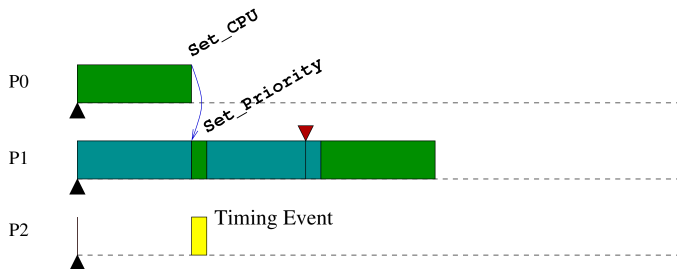
2 Requisitos para la planificación en multiprocesadores

3 Propuesta 1: Cambio atómico y/o diferido de atributos en Ada

4 Propuesta 2: Timing Events con afinidad

Timing Events en Ada 2012

- No se pueden aplicar para resolver el problema eficientemente dado que no sé puede especificar en que CPU se va a ejecutar.
- Aparecerían artecfactos en la ejecución, pero estarían acotados, dado que el *Handler* se invoca desde el mecanismo de la interrupción del reloj de tiempo real[1]



[1] Habitualmente es no interrumpible.

Añadir afinidad a los *Timing Events*

- Permitir que se pueda establer la CPU en la que debe ejecutar el Timing_Event.

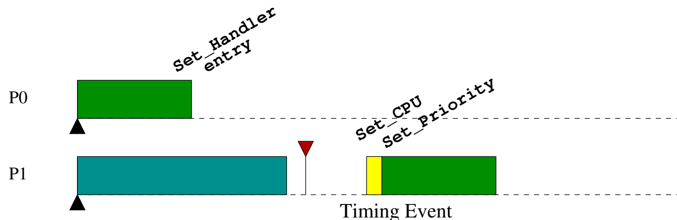
```
with System.Multiprocessors; use System.Multiprocessors;
package Ada.Real_Time.Timing_Events is
   ...
  procedure Set_Scheduling_Domain(TM : in out Timing_Event;
                                  SD: access all Scheduling_Domain);
  function Get_Scheduling_Domain(TM : Timing_Event) return Scheduling_Domain;

  procedure Set_CPU(TM : in out Timing_Event; CPU_Nr: CPU_Range);
  function Get_CPU(TM : Timing_Event) return CPU_Range;
end Ada.Real_Time.Timing_Events;
```

Cambio atómico de atributos con *Timing Events*

Combinado por ejemplo con un objeto protegido.
Delay_Until_And_Apply_Sched_Params

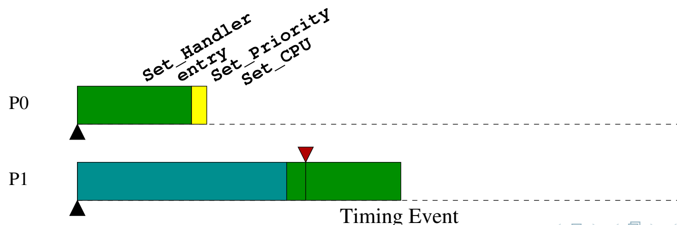
- Solicitar la ejecución del Timing_Event en la CPU de destino de la tarea que se debe despertar.
- La tarea se bloquea en un *entry*.
- Cuando el *handler* se ejecuta le cambia la CPU, el resto de los parámetros y se abre la condición de la *entry*.



Cambio atómico de atributos con *Timing Events* (cont.)

Apply_Sched_Params

- Solicitar la ejecución de un Timing_Event con tiempo cero en la CPU de origen de la tarea.
- Cuando se ejecuta el *handler* la tarea no tiene la CPU.
- El *handler* le cambia la prioridad y/o el plazo de entrega y finalmente la CPU. **Ojo!**
- Si lo invoca directamente la tarea debería bloquearse para garantizar que no avanza antes de que se ejecute el *handler*.



Ventajas e incovenientes

Ventajas

- No require modificar el sistema operativo.
- Añadirle afinidad a los Timing_Events es relativamente sencillo.
- Puede utilizarse internamente para implementar el tipo Sched_Params que son más sencillos de utilizar.

Inconvenientes

- En los sistemas operativo en que la aplicación se ejecuta en espacio de usuario, v.g. GNU/Linux, la ejecución de los Timing_Events es más ineficiente ya que se hace en el contexto de una tarea de la máxima prioridad y no en el contexto de un manejador de reloj.