

Sistemas en Tiempo Real – P9

Configuración del equipo

Los PCs actuales son multicore, un aspecto que no contemplamos en esta práctica, de modo que necesitamos que Linux explote un único core. Para ello configuraremos el número de cores útiles en tiempo de ejecución en la línea de comandos. Concretamente el segundo core (van del 0 a x-1, siendo x el número de cores) se deshabilita mediante el comando:

```
$ echo 0 | sudo tee /sys/devices/system/cpu/cpu1/online
```

Realmente lo que hace el comando es escribir un cero en el fichero online correspondiente al core 1 para deshabilitarlo. Escribir un uno vuelve a habilitarlo. Deshabilitaremos todos los cores excepto el 0.

Ejercicio

El objetivo de este ejercicio es implementar lo mismo que en la práctica 8, en este caso, usando ADA. Para ello, usaremos dos tipos de tareas periódicas: una para mostrar los tiempos y otra como tarea periódica genérica (análogas a las de la práctica 8), así como una estructura que almacenará el marco temporal de cada tarea.

Marco temporal

El tipo *marco_t* se encargará de almacenar el marco temporal de cada tarea. Es idéntico al de la práctica 8 y deberá contener los siguientes campos:

- C --> Tiempo de ejecución en milisegundos (tipo entero)
- T --> Periodo de repetición en milisegundos (tipo entero)
- P --> prioridad de la hebra (tipo entero)
- id --> identificador de hebra (tipo entero)
- TCritico --> instante crítico (tipo time). Se utilizará para que todas las tareas comiencen a la vez, evitando así los retardos surgidos de la creación de tareas.

Como ADA necesita que los tipos de los parámetros de las tareas sean o bien datos básicos de longitud fija (enteros, reales, ...) o bien punteros, deberemos crear un tipo puntero al marco temporal (*Pmarco_t*). También se necesitarán un tipo vector de tamaño variable de marcos temporales (*marcos_t*, ya definido en el esqueleto de la práctica) y un tipo puntero al vector anterior (*Pmarcos_t*).

Tarea para mostrar tiempos

Tarea periódica, de periodo 10 ms que, tras esperar al instante crítico (lo recibe por parámetro), muestra el tiempo actual de ejecución. Su pseudocódigo es el siguiente:

1. Establecer su prioridad a la máxima posible (ver el anexo).
2. Calcular el siguiente instante de ejecución como la suma del instante crítico y 10 ms.
3. Esperamos por el instante crítico.

4. para i desde 0 hasta 199
 - 4.1. Mostrar el mensaje "***** x ms *****" (donde x variará de 10 en 10).
 - 4.2. Esperar al siguiente instante de ejecución.
 - 4.3. Incrementar el siguiente instante de ejecución en 10 ms.

Tarea periódica genérica

Esta tarea recibirá por parámetros su marco temporal (de tipo *Pmarco_t*) y el identificador de la tarea encargada de mostrar los tiempos (de tipo *PTareaMostrarTiempos*; usaremos este parámetro para que la tarea periódica genérica termine cuando termine la de mostrar tiempos). Esta tarea realizará las siguientes acciones:

1. Establecer su prioridad a la indicada en el marco temporal recibido (ver el anexo).
2. Calcular el siguiente instante de ejecución como la suma del instante crítico y el periodo de la tarea (ambos valores contenidos en el marco temporal de la tarea).
3. Esperar por el instante crítico.
4. Mientras la tarea de mostrar tiempos no termine
 - 4.1. Mostrar el mensaje "Tarea id con prioridad P inicia la acción periódica\n", siendo id y P los valores correspondientes a los del marco temporal de la tarea.
 - 4.2. Ejecutar la función *spin_m* usando como parámetro el campo C del marco temporal de la tarea.
 - 4.3. Mostrar el mensaje " Tarea id termina la acción periódica\n".
 - 4.4. Esperar hasta el siguiente instante de ejecución.
 - 4.5. Incrementar el siguiente instante de ejecución en T ms (campo T del marco temporal).

Procedimiento LanzarTareas

En esta práctica no hay que hacer nada en el programa principal, que se encargará de gestionar la lectura de parámetros de entrada, la lectura de los datos del fichero de entrada (tendrá el mismo formato que en la práctica 8), así como de invocar al procedimiento *LanzarTareas* (que deberá ser implementado), el cual se encargará de lanzar todas las tareas de la práctica. Este procedimiento calculará el instante crítico (hora actual más 2 segundos), asignándolo a los marcos temporales de todas las tareas, y lanzará todas las tareas de forma dinámica (ver anexo).

A la hora de ejecutar el programa, debemos hacerlo, al igual que ocurría en POSIX, desde la consola y usando el comando *sudo*, ya que la modificación de prioridades requiere permisos de administrador.


La práctica se comprimirá en un único archivo que será entregado a través del campus virtual usando la tarea creada para tal efecto.

ANEXO – Ayuda para ADA

Creación de un proyecto en ADA

- Seleccionar *Create new project from template*.
- Seleccionar *Basic→Simple Project*.
- Seleccionar el nombre del proyecto, del ejecutable y la ruta donde se creará el proyecto.

El código estará en NombreProyecto/src/NombreProcedimientoPrincipal.adb

Para compilar el código, pulsar sobre el botón  de la barra de herramientas.

Para ejecutar, pulsar sobre el botón  de la barra de herramientas.

Abrir de un proyecto existente

- Seleccionar Open existing project: e indicar la ruta del fichero .gpr que contiene el proyecto.

Paso de parámetros de tipo puntero a tareas

Para poder pasar parámetros de tipo puntero a las tareas debemos usar la palabra reservada *access* delante del tipo (sería similar al * en C++). Para evitar posibles problemas, se definirá un tipo puntero al tipo original, que será el usado en la tarea.

Sin embargo, como lo que queremos es pasar la dirección de una variable estática (lo que en C++ sería el &), usaremos *access all* en lugar de sólo *access* en la definición del tipo de los parámetros de la tarea.

Además, cuando definamos la variable que queremos pasar por declaración, usaremos la palabra *aliased* delante del nombre del tipo de la variable (esto permite acceder a la dirección de la misma) y, posteriormente usaremos el atributo *'Access* o *'Unchecked_Access* (para variables locales) para acceder a la dirección de la variable.

Por último, en el caso de necesitar usar luego ese parámetro como una variable estática (no como puntero), por ejemplo a la hora de pasar parámetros a métodos que reciban el tipo estático (sin puntero), usaremos el campo *.all* que tienen todos los punteros. Esto hace referencia al contenido del puntero, no a la dirección.

Resumiendo:

```
Variable: aliased TipoNormal;--Variable de tipo no puntero
TipoPuntero is access all TipoNormal;--Nuevo tipo puntero
task type TareaTipo(Parametro: TipoPuntero);--Parámetro de la
                                     --tarea de tipo puntero
task body TareaTipo is begin --Cuerpo de la tarea
    ...
end;
Tarea: TareaTipo(Variable'Access);--Creación de la tarea pasando
                                     --por parámetro la dirección de la variable
```

Algunas funciones de utilidad

- `New_Line` (paquete `Ada.Text_IO`): Imprime un salto de línea.
- `Put_Line` (paquete `Ada.Text_IO`): Imprime por pantalla una cadena de caracteres y la termina con un salto de línea.
- `integer'Image(Dato)`: Devuelve el valor del entero *Dato* en forma de cadena de caracteres.
- `integer'Value(Dato)`: Devuelve el valor de la cadena de caracteres *Dato* en forma de entero.
- `&`: Operador de concatenación.
 - o Por ejemplo, para imprimir una cadena del tipo “el valor de la variable *var* es *x*”, siendo *x* el valor de la variable *var*, usaríamos la instrucción:
`Put_Line("el valor de la variable var es "&integer'image(var))`

Retardos en ADA

Para realizar retardos en ADA utilizamos la palabra reservada *delay* seguida de la cantidad de tiempo que deseamos retardar la tarea. Si entre *delay* y el intervalo colocamos la palabra reservada *until*, el retardo será absoluto.

Las funciones *Milliseconds*, *Microseconds* y *Nanoseconds* nos permiten convertir valores tipo *integer* a valores tipo *time_spam*.

La función *Clock* nos devuelve la hora actual en formato *time*.

La sobrecarga del operador '+' nos permite realizar la suma de tipos *time* + *time_spam* devolviendo el resultado como tipo *time*.

Estos tipos/funciones están definidos en el paquete *ada.real_time*;

Establecimiento de la política en ADA

Para establecer la política de planificación en ADA se usa el siguiente *pragma* justo al principio del código del programa principal (antes incluso que la inclusión paquetes):

```
pragma Task_Dispatching_Policy(Politica);
```

Para esta práctica, usaremos la política *FIFO_Within_Priorities*.

Asignación de prioridades estáticas en ADA

La asignación de la prioridad estática de una tarea (es la que vamos a utilizar en esta práctica) se realiza incluyendo el siguiente *pragma* en la definición de la tarea (no en el cuerpo).

```
pragma priority(Prioridad);
```

Prioridad puede ser un valor constante o un parámetro de la tarea (en el caso de esta práctica, el valor nos lo dará el campo *P* del marco temporal que pasamos por parámetro a la tarea).

Por otra parte, los atributos *Priority'first* y *Priority'last* nos dan los valores de prioridades mínimo y máximo respectivamente. Para la asignación de prioridades, se requiere la inclusión del paquete *system*.

Ejecución dinámica de tareas

Como hemos visto hasta ahora, cuando definimos tareas estáticas, al iniciarse el *begin* del bloque en el que están definidas, se lanzan de forma automática. Para esta práctica, esto no es deseable, ya que necesitamos establecer una serie de valores antes de lanzar las tareas. Para ello, debemos definir, para cada tipo tarea, un tipo puntero a dicha tarea. A continuación, definimos las tareas de forma similar a como hacemos cuando usamos tipos de tareas estáticas, pero usando el tipo puntero a tarea que acabamos de crear (éstas variables puntero a tarea no se lanzarán hasta que se creen mediante *new*). Por último, cuando queramos lanzar la tarea correspondiente bastará con usar *new*. El siguiente pseudocódigo ilustra cómo realizar esta labor:

```
-- Zona de declaración de un bloque
task type TipoTarea(parametros);
task body TipoTarea is
    ...
begin
    ...
end;
type PunteroTarea is access TipoTarea; --Tipo puntero a Tarea
PT: PunteroTarea; --Variable de tipo puntero a Tarea
begin --La tarea PT no se lanza al iniciarse el bloque
    ... -- Código anterior al lanzamiento de la tarea PT
    PT=new TipoTarea(parámetros); --Creamos una nueva tarea de
        -- tipo TipoTarea (no PunteroTarea). En este
        -- momento se lanza la tarea PT
    ... -- Código posterior al lanzamiento de la tarea PT
end;
```