# Sistemas en Tiempo Real – Práctica ADA 3

En esta práctica implementaremos un sistema similar al de la Práctica Posix 3: control de la señal SIGINT (Ctrl+C) y muestra de información de forma periódica. Se deberán realizar las siguientes tareas:

- MostrarHora:
  - Cada 5 segundos mostrará por pantalla la hora del reloj calendario de ADA en formato *día/mes/año ---- hora:minutos:segundos*.
- ContarCtrlC:
  - Cuenta el número de veces que ocurre la señal SIGINT (Ctrl+C) (Ver el anexo para saber cómo capturar la señal SIGINT en ADA).
  - En caso de que pasen 6 segundos desde la última vez que se pulsó Ctrl+C, deberá aparecer por pantalla el mensaje "No se ha pulsado Ctrl+C en 6 segundos".
- MostrarCtrlC:
  - Cada 2 segundos mostrará por pantalla el número de veces que ha ocurrido la señal SIGINT (Ctrl+C).

Todas las tareas recibirán un parámetro con la dirección de memoria del contador (de esta forma todas las tareas comparten el valor del contador) y otro con el número de señales a esperar, de forma que los bucles de todas las tareas terminarán cuando el contador llegue al número de señales esperadas (usar una constante, por ejemplo 5). Todos los retardos de las tareas se realizarán mediante retardos absolutos y usando el reloj de tiempo real de ADA. Como se van a usar los dos relojes, hay que diferenciar entre los tipos/funciones que pertenezcan a ambos paquetes y se llamen igual usando el nombre del paquete.

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

## 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* 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 … end;--Cuerpo de la tarea
Tarea: TareaTipo(Variable'Access);--Creación de la tarea pasando
                     --por parámetro la dirección de la variable
```

## Captura de SIGINT en ADA

Para capturar la señal SigInt en ADA debemos crear un manejador de evento. Para ello debemos crear un objeto protegido (sin la cláusula *type*) con dos métodos, un procedimiento, que será el manejador de la señal, y una entrada, que usaremos para esperar a la ocurrencia de la señal en la tarea esporádica.

Dentro de la definición del objeto protegido hay que incluir los siguientes pragmas (deben incluirse tras definir el procedimiento para que no de error de compilación por no encontrarlo):

```
-- Pragma para indicar que ProcedimientoControlador es un
-- manejador de interrupciones
pragma Interrupt_Handler(ProcedimientoControlador);

-- Pragma para indicar que ProcedimientoControlador va a
-- controlar la señal Sigint
pragma Attach_Handler(ProcedimientoControlador, Sigint);

-- Pragma necesario para que el programa pueda manejar las
-- señales
pragma Unreserve_All_Interrupts;
```

Para poder utilizar señales, debemos incluir los paquetes `Ada.Interrupts` y `Ada.Interrupts.Names`.

## Algunas funciones de utilidad

- Put (paquete GNAT.IO): Muestra por pantalla una cadena de caracteres o un entero.
- New_Line (paquete GNAT.IO): Imprime un salto de línea.
- Put_Line (paquete GNAT.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.
- Conversión de tipos (supongamos i una variable entera y f una variable float): f=float(i). Es importante la conversión de tipos ya que ADA no permite la operación entre tipos diferentes.
- &: Operador de concatenación.