Práctica 1.ª
Una aproximación a la programación concurrente en Ada

Escuela de Ingeniería y Arquitectura
Depto. de Informática e Ingeniería de Sistemas

1. Objetivos

1. Crear y manipular tareas en Ada.
2. Adquirir conocimiento empírico del funcionamiento del *scheduler* del entorno de Ada que utilizaremos.
3. Implementar un pequeño programa concurrente que requiera el uso de un algoritmo de exclusión mutua.

2. Contenidos

La práctica consta de tres bloques, de los cuales los dos primeros son de realización obligatoria, mientras que el tercero es opcional. En el primer bloque se van introduciendo conceptos básicos de programación concurrente en Ada. En el segundo se pide la implementación de un pequeño programa concurrente con una zona de exclusión mutua y en el último, de realización voluntaria, se plantea una modificación al programa planteado en el segundo bloque para utilizar el mecanismo de sincronización de citas de Ada. Es recomendable consultar las transparencias que con el título *Rápida (y parcial) introducción a la concurrencia con Ada* están disponibles en la página web de la asignatura[1].

Como resultado de la práctica se deberán entregar al profesor de prácticas los ficheros de código fuente que se van pidiendo en el enunciado. Además, la práctica pide responder

[1] `http://webdiis.unizar.es/~ezpeleta/lib/exe/fetch.php?media=misdatos:pc:introada.pdf`

o comentar algunas cuestiones. Dichas respuestas deberán entregarse en un fichero de texto denominado «`primera.txt`».

2.1. Concurrencia en Ada

Se plantean a continuación cinco pequeños ejercicios en los que trabajar con tareas en Ada, tipos de tareas, instrucción **`delay`** y sincronización por citas y comprobar cómo afecta a la ejecución concurrente de las tareas el tipo de *scheduler* del que se disponga. El código fuente que se escriba puede compilarse y ejecutarse tanto en Windows, con el compilador gnatmake en el entorno AdaGIDE, como con el compilador del clúster *hendrix* (`hendrix-ssh.cps.unizar.es`).

Parte 1: Escribe un procedimiento Ada, denominado `primera1`, cuyo fuente corresponde al fichero «`primera1.adb`», que lance tres tareas. Cada una de ellas escribirá 100 veces un caracter en la pantalla: la primera de ellas escribe el carácter 'x', la segunda, el carácter 'o', mientras que la tercera escribe el carácter '-'.

- **Cuestión 1:** Ejecuta el programa varias veces y analiza el resultado de cada ejecución. ¿Son las trazas de ejecución siempre iguales? Intenta describir el comportamiento del *scheduler*.

Parte 2: Modifica el código anterior añadiendo la siguiente declaración antes de la cláusula de contexto:
`pragma Task_Dispatching_Policy(FIFO_Within_Priorities);`
El fuente de esta versión debe estar en el fichero «`primera2.adb`». Busca en la documentación de Ada qué es un *pragma* y cuál es el significado del *pragma* `Task_Dispatching_Policy`.

- **Cuestión 2:** Ejecuta el nuevo programa varias veces. ¿Se observa algún cambio en las trazas de ejecución? ¿Las trazas de ejecución son ahora **siempre** iguales?

Parte 3: Vamos ahora a *parar* tareas temporalmente. Para ello, añade a cada una de las tareas anteriores una instrucción **`delay`**. El fuente de esta versión debe estar en el fichero «`primera3.adb`».

- **Cuestión 3:** ¿Qué ocurre si todas las tareas tienen la misma instrucción de espera (es decir, se les hace esperar el mismo tiempo)? ¿Y si estos tiempos de espera son distintos? Prueba tanto con el *pragma* como sin él.

Parte 4: En esta parte, en lugar de definir tres tareas de tres tipos diferentes, define un **tipo tarea** que admita tres discriminantes. El primero, de tipo `character`, se usará para indicar qué carácter queremos que escriba en la pantalla. El segundo se usará para indicar el tiempo que la tarea debe estar *parada* entre dos escrituras

sucesivas de su carácter. El tercero indicará cuántas veces debe escribir el carácter en la pantalla. El fuente de esta versión debe estar en el fichero «`primera4.adb`». Ejecuta dicho procedimiento, observando su comportamiento.

- **Cuestión 4:** ¿Qué se puede decir del orden en que se activan las tareas? ¿Dice el manual de referencia de Ada 95 algo al respecto?

Parte 5: En esta parte se trata de que el programa controle el momento de activación de las tareas. Para ello hay que escribir en «`primera5.adb`» un programa que cumpla los siguientes requisitos:

- Al iniciar su ejecución pide, para cada una de sus tres tareas, el primero de sus discriminantes
- La ejecución debe ser tal que si los datos de entrada fuesen 'A', 'B', 'C' la ejecución debe dar como resultado

```
ABCABCABCABC...
```

- Mientras que si los datos introducidos fueran 'x', '-', 'o' la ejecución dará como resultado

```
x-ox-ox-ox-o...
```

- **Cuestión 5:** ¿Qué diferencia fundamental plantea esta última cuestión respecto a la anterior? ¿Se te ocurre alguna solución alternativa a la que has propuesto?
- **Pistas:** Mira información sobre sincronización por citas

2.2. Simulación de entradas a un parque

La figura 1 muestra, de una manera esquemática, un parque con dos entradas (que identificaremos como A y B). Por estas entradas, los visitantes pueden acceder al mismo y los responsables quieren saber en todo momento cuántas personas han entrado (no importan las salidas, o se puede suponer que nadie de los que entra sale posteriormente).

Se pide escribir un fichero denominado «`primera6.adb`» con el código fuente de un simulador para el sistema considerado. Se trata de escribir un programa que simule el comportamiento del sistema durante un periodo determinado de ejecución hasta que se alcancen 20 entradas por cada una de las dos puertas. El programa debe indicar, para cada entrada, el instante de tiempo en que se produce dicha entrada en el parque y la puerta por la que entra. Por otro lado debe informar del tiempo medio transcurrido desde que cada visitante entró en el parque. Para llevar a cabo la simulación, deben lanzarse dos tareas, una por cada entrada. Debe tenerse presente que:

Figura 1: Representación esquemática de un parque con dos entradas.

- Las posibles sincronizaciones necesarias para el programa se han de implementar mediante exclusiones mutuas, usando el algoritmo de Peterson (una simplificación del algoritmo de Dekker para dos procesos)[2].
- Para manejo del tiempo se puede usar el paquete `Ada.Calendar`.
- Si las dos tareas que simulan las entradas se programan utilizando un mismo tipo de tarea, la realización de la parte voluntaria de esta práctica, que se explica en la siguiente sección, se puede realizar con más facilidad.

Una posible traza de la ejecución de una simulación puede ser la siguiente:

```
Entrada a parque por puerta B
--> Personas en el parque: 1, tiempo medio de estancia: 0.0000
-----> Por puerta A:  0
-----> Por puerta B:  1

Entrada a parque por puerta A
--> Personas en el parque: 2, tiempo medio de estancia: 0.6717
-----> Por puerta A:  1
-----> Por puerta B:  1

Entrada a parque por puerta A
--> Personas en el parque: 3, tiempo medio de estancia: 6.0349
-----> Por puerta A:  2
-----> Por puerta B:  1

Entrada a parque por puerta B
--> Personas en el parque: 4, tiempo medio de estancia: 6.1678
-----> Por puerta A:  2
-----> Por puerta B:  2
       ⋮
Entrada a parque por puerta B
--> Personas en el parque: 40, tiempo medio de estancia: 60.9987
-----> Por puerta A: 20
-----> Por puerta B: 20
```

[2]Puedes encontrar el algoritmo de Peterson en las transparencias de la asignatura correspondientes al curso pasado: `http://webdiis.unizar.es/~ezpeleta/lib/exe/fetch.php?media=misdatos:pc:07.pdf`

```
Han entrado 40 personas, 20 por cada puerta.
Tiempo medio en el parque: 59.4738
```

2.3. Generalización del sistema de simulación de entradas al parque (voluntario)

En esta sección, se plantea como ejercicio generalizar el programa de simulación del apartado anterior de forma que el número de puertas sea un entero N mayor que 2, identificadas por letras mayúsculas consecutivas a partir de la A. La figura 2 muestra, de una manera esquemática, el parque con cuatro entradas y cinco entradas.

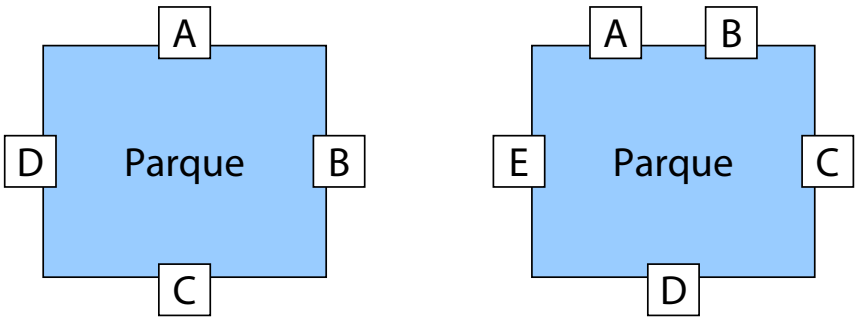

Figura 2: Representación esquemática de un parque con cuatro y cinco entradas.

Para llevar a cabo la simulación, deben lanzarse $N + 1$ tareas, una por cada entrada y otra adicional para gestionar la simulación. El valor del número N de entradas se declarará como una constante en el código Ada.

Como consecuencia del incremento del número de entradas, la sincronización para el acceso a la sección crítica va a resultar más eficiente si se utiliza el mecanismo de citas de Ada (y también simplificará su programación). Para llevar a cabo la simulación, deben lanzarse $N+1$ tareas, una por cada entrada y otra adicional para gestionar la simulación.

Se pide, partiendo del código fuente del programa `primera6`, escribir un fichero denominado «`primera7.adb`» con las modificaciones planteadas.

Una posible traza de la ejecución de una simulación, con $N = 4$, puede ser la siguiente:

```
Entrada a parque por puerta A
--> Personas en el parque:  1, tiempo medio de estancia: 0.0000
-----> Por puerta A:  1
-----> Por puerta B:  0
-----> Por puerta C:  0
-----> Por puerta D:  0

Entrada a parque por puerta A
--> Personas en el parque:  2, tiempo medio de estancia: 3.2197
```

```
-----> Por puerta A:  2
-----> Por puerta B:  0
-----> Por puerta C:  0
-----> Por puerta D:  0

Entrada a parque por puerta C
--> Personas en el parque:  3, tiempo medio de estancia: 3.0923
-----> Por puerta A:  2
-----> Por puerta B:  0
-----> Por puerta C:  1
-----> Por puerta D:  0

Entrada a parque por puerta D
--> Personas en el parque:  4, tiempo medio de estancia: 2.6406
-----> Por puerta A:  2
-----> Por puerta B:  0
-----> Por puerta C:  1
-----> Por puerta D:  1
       ⋮
Entrada a parque por puerta D
--> Personas en el parque: 80, tiempo medio de estancia: 63.3035
-----> Por puerta A: 20
-----> Por puerta B: 20
-----> Por puerta C: 20
-----> Por puerta D: 20
Han entrado 80 personas, 20 por cada puerta.
Tiempo medio en el parque: 62.0260
```

3. Entrega de material

- Como resultado de esta práctica hay que enviar por correo electrónico al profesor de prácticas (`latre@unizar.es`) un fichero denominado «`pract1.zip`» que contenga los ficheros solicitados: «`primera.txt`», «`primera1.adb`», «`primera2.adb`», «`primera3.adb`», «`primera4.adb`», «`primera5.adb`», «`primera6.adb`» y, si se ha realizado la parte opcional, «`primera7.adb`», además de todos aquellos ficheros que puedan ser necesarios para la compilación de los programas solicitados.
- Fecha límite de entrega: hasta el miércoles 21 de marzo de 2012, fecha inmediatamente anterior a la de la próxima sesión de prácticas.