Programación Concurrente
Prácticas 1, 2 y 3

Dpto. LSIIS — Unidad de Programación Convocatoria de Junio — curso 2008/2009

Normas

- La fecha límite de entrega de las prácticas 1, 2 y 3 es el **lunes 22 de junio de 2009** a las 21:00.
- Planeamos publicar los días **1 de junio** y **15 de junio** (ambas fechas aproximadas) el estado de las prácticas recibidas hasta ese momento.
- Deberá mencionarse explícitamente el uso de recursos (código, algoritmos específicos, esquemas de implementación, etc.) que no hayan sido desarrollados por el alumno o proporcionados como parte de asignaturas de la carrera. La falta de esta mención podrá impedir aprobar las prácticas.
- Os recordamos que **todas** las prácticas entregadas pasan por un proceso automático de detección de copias. Los involucrados en la copia de una práctica corren el riesgo de suspenderlas para todo el año académico en curso, independientemente de la calidad o adecuación del código.

1. ¡Organización!

Ordenar elementos (en un vector, en una lista, etc.) es un problema recurrente en la informática y se han diseñado muchos modos de llevarlo a cabo, incluyendo variaciones concurrentes. En esta práctica vamos a implementar un método de ordenación concurrente. No es el más eficiente[1] ni, es en general, el más adecuado (para vectores grandes necesita más tareas de las que en general sería recomendable tener), pero es extremadamente sencillo.

La idea fundamental se basa en que si un vector no está ordenado, entonces necesariamente habrá dos elementos adyacentes en él que no estén ordenados. Cada una de las tareas concurrentes del sistema examinará dos elementos contiguos, asignados a la misma, y los intercambiará si no están ordenados: en la figura 2, la i-ésima tarea intenta, repetidamente, asegurar que los elementos en la posición i e $i + 1$ están ordenados.

La especificación del recurso que realiza la comunicación y sincronización entre tareas se encuentra en la figura 3.

2. Trabajo a realizar

2.1. Primera práctica

La entrega de la **primera práctica** constará de una implementación del recurso en Ada 95 usando objetos protegidos, basada en la especificación dada en el enunciado. La implementación a realizar debe estar contenida en un fichero llamado `ordenacion.adb` (ver el apartado 3).

[1] De hecho es **muy poco** eficiente.

```
generador:
loop
    <adquirir vector V>
    Depositar_Vector (M, V, Tam);
    Retirar_Ordenado (M, V);
end loop;
```

Figura 1: Tarea con vectores a ordenar.

```
 M : Mezclador;

ordenador (I):
loop
    Intercambiar (M, I);
end loop;
```

Figura 2: Esquema de tarea ordenadora.

C-TADSOL Mezclador
OPERACIONES
ACCIÓN Depositar_Vector: *Mezclador[io]* $\times$ $\mathbb{N}$*[i]* $\times$ *Tipo_Vector[i]*
ACCIÓN Retirar_Ordenado: *Mezclador[io]* $\times$ *Tipo_Vector[io]*
ACCIÓN Intercambiar: *Mezclador[io]* $\times$ *Tipo_Indice[i]*
SEMÁNTICA
DOMINIO:
TIPO: *Mezclador*= *(nelem:* $\mathbb{N}\times$ *v: Tipo_Vector)*
Tipo_Vector = Secuencia*(Tipo_Dato)*
INVARIANTE: $\forall m \in Mezclador \bullet m.nelem \leq \mathsf{Longitud}(m.v)$

INICIAL(m**):** $true$

PRE: true
CPRE: true
Depositar_Vector(m, l, v)
***POST*:** *Dejamos un vector en el recurso para que las tareas concurrentes lo vayan ordenando.*
POST: $m^{sal} = (l, v)$

PRE: true
***CPRE*:** *El vector en el recurso está completamente ordenado.*
CPRE: $\forall i, 1 \leq i \leq m.nelem \bullet m.v(i) \leq m.v(i+1)$
Retirar_Ordenado(m, o)
***POST*:** *Devolvemos vector ordenado a la tarea llamante.*
POST: $m^{sal} = m^{ent} \wedge o^{sal} = m^{ent}.v$

PRE: true
CPRE: $i < m.nelem \wedge m(i) > m(i+1)$
***CPRE*:** *Hay dos elementos desordenados en la parte "activa" del vector guardado en el recurso.*
Intercambiar(m, i)
***POST*:** *Intercambiamos los elementos no ordenados de vector en el recurso.*
POST: $m^{sal}.nelem = m^{ent}.nelem \wedge m^{sal}.v =$
$m^{ent}.v(1..i-1) + \langle m^{ent}.v(i+1), m^{ent}.v(i)\rangle + m^{ent}.v(i+2..m^{ent}.nelem)$

Figura 3: Especificación del recurso "ordenador" de vectores.

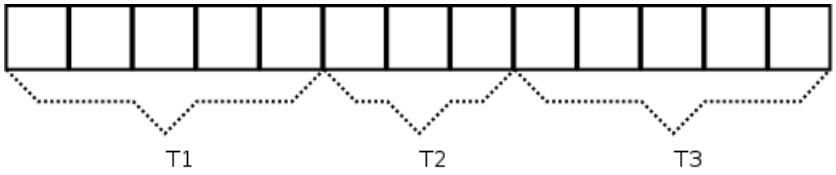

Figura 4: Cada tarea se ocupa de un segmento del vector.

2.2. Segunda práctica

La entrega de la **segunda práctica** constará de una implementación del recurso compartido en Ada 95 mediante *rendez-vous* y/o paso de mensajes síncrono y basada en la especificación entregada en el enunciado. La implementación deberá estar contenida en un fichero llamado `ordenacion.adb` (ver el apartado 3).

2.3. Tercera práctica

Usar tantas tareas como elementos en el vector (menos uno) es impracticable para vectores de tamaño incluso reducido. Posiblemente una mejor manera sea hacer que cada tarea pueda ocuparse de mantener la uniformidad de un segmento de vector que sea mayor que 2 elementos — en general, un segmento comprendido entre un elemento inicial y uno final (ver figura 4) — de modo que la cabecera del procedimiento intercambiar sea

ACCIÓN Intercambiar: *Mezclador[io]* $\times$ *Tipo_Indice[i]* $\times$ *Tipo_Indice[i]*

Intercambiar(m, min, max)

Se os pide cambiar la definición de los procesos y / o el recurso (pueden ser ambos) para que el sistema funcione de acuerdo con esta idea. Suponed que disponemos de un predicado $ordenado(s1, s2)$ que es cierto cuando la secuencia $s2$ tiene todos los elementos de la secuencia $s1$ y sólo esos, incluidos repeticiones, y además $s2$ está ordenada.

La entrega de esta **tercera práctica** constará de una memoria con un nuevo diseño que debe cumplir en lo posible los mismos requisitos que se tenían para el problema inicial. En esta entrega deben incluirse especificación formal del recurso necesario para la sincronización de los procesos y el código de los procesos en función de la interfaz del recurso especificado. La entrega de la práctica se hará de forma **electrónica** en formato PDF.

3. Información general

El texto de estas prácticas se encuentra en

`http://moodle.upm.es/`

La entrega del código y la memoria se realizará **vía WWW** en la dirección

`http://lml.ls.fi.upm.es/entrega/`

El código que finalmente entreguéis (tanto para objetos protegidos como para paso de mensajes) no debe realizar **ninguna** operación de entrada / salida.

Para facilitar la realización de la práctica están disponibles en `http://moodle.upm.es/` varias unidades de compilación:

- `ordenacion.ads` y `ordenacion.adb`: dan, respectivamente, el interfaz y un esqueleto para facilitar la implementación del recurso. Podéis tomar este esqueleto como punto de partida para implementar tanto el servidor de paso de mensajes como el objeto protegido. El **único** fichero que podéis entregar como prácticas 1 y 2 es el correspondiente a la implementación (el `.adb`) del recurso compartido.
- `red_ordenadora.adb`: programa principal que crea el recurso compartido, arranca las tareas de control y simula el funcionamiento del sistema.
- `parametros.ads`: paquete que contiene definiciones de tipos y constantes necesarias en la implementación. Podéis variarlas con objeto de cambiar el comportamiento del sistema.

Por supuesto durante el desarrollo podéis cambiar el código que os entreguemos para hacer diferentes pruebas, depuración, etc., pero el código que entreguéis debe poder compilarse y ejecutar sin errores junto con el resto de los paquetes entregados **sin modificar estos últimos**. Podéis utilizar las librerías auxiliares que estén disponibles en `http://moodle.upm.es/` en la asignatura de Programación Concurrente. Si os veis en la necesidad de usar algún otro paquete adicional que no venga con GNAT y que no os estemos proporcionando, hacédnoslo saber con antelación suficiente para evaluar la petición.

El programa de recepción de prácticas podrá rechazar entregas que:

- Tengan errores de compilación.
- Utilicen otras librerías o paquetes aparte de los estándar de Ada y los que se han mencionado anteriormente.
- No estén suficientemente comentadas. Alrededor de un tercio de las líneas deben ser comentarios **significativos**. No se tomarán en consideración para su evaluación prácticas que tengan comentarios ficticios con el único propósito de rellenar espacio.
- Estén escritas con un estilo incorrecto: los programas se compilarán con el compilador GNAT y las opciones `-gnaty -gnata`. Recomendamos, por tanto, usar estas opciones para desarrollar las prácticas.
- No superen unas pruebas mínimas de ejecución, similares a las que tenéis en el programa de simulación que os entregamos.