UNIVERSIDAD
DE GRANADA

Sistemas Concurrentes y Distribuidos: Práctica 2. Casos prácticos de monitores en C++11.

Carlos Ureña / Jose M. Mantas / Pedro Villar

2017-18

Grado en Ingeniería Informática / Grado en Ingeniería Informática y Matemáticas.
Dpt. Lenguajes y Sistemas Informáticos
ETSI Informática y de Telecomunicación
Universidad de Granada

Práctica 2. Casos prácticos de monitores en C++11.
Índice.

Objetivos

Los objetivos de esta práctica son ilustrar el proceso de diseño e implementación de los monitores SU no triviales, usando para ello dos casos prácticos:

- El problema de los fumadores, que ya hemos visto resuelto con semáforos, pero ahora con monitores
- Un problema nuevo, el problema de la barbería.

Al igual que en los problemas anteriores, veremos el proceso de diseño (creación del pseudo-código), seguido de la implementación (traducción a monitores SU u monitores Hoare)

Sección 1.
El problema de los fumadores.

El problema de los fumadores

En este ejercicio consideraremos de nuevo el mismo problema de los fumadores y el estanquero cuya solución con semáforos ya vimos en la práctica 1. Queremos diseñar e implementar **un monitor SU** (**monitor Hoare**) que cumpla los requerimientos:

- Se mantienen las tres hebras de fumadores y la hebra de estanquero.
- Se mantienen exactamente igual todas las condiciones de sincronización entre esas hebras.
- El diseño de la solución incluye un monitor (de nombre `Estanco`) y las variables condición necesarias.
- Hay que tener en cuenta que ahora no disponemos de los valores de los semáforos para conseguir la sincronización.

A continuación haremos un diseño del monitor, y se deja como actividad la implementación de dicho diseño.

Hebras de fumadores

Los **fumadores**, en cada iteración de su bucle infinito:

- LLaman al procedimiento del monitor **`obtenerIngrediente(i)`**, donde `i` es el número de fumador (o el número del ingrediente que esperan). En este procedimiento el fumador espera bloqueado a que su ingrediente esté disponible, y luego lo retira del mostrador.
- Fuman, esto es una llamada a la función **Fumar**, que es una espera aleatoria.

```
process Fumador[ i : 0..2 ] ;
begin
  while true do begin
    Estanco.obtenerIngrediente( i );
    Fumar( i ) ;
  end
end
```

Hebra estanquero

El **estanquero**, en cada iteración de su bucle infinito:

- Produce un ingrediente aleatorio (llama a una función **`ProducirIngrediente()`**, que hace una espera de duración aleatoria y devuelve un número de ingrediente aleatorio).
- Llama al procedimiento del monitor **`ponerIngrediente(i)`**, (se pone el ingrediente `i` en el mostrador) y después a **`esperarRecogidaIngrediente()`** (espera bloqueado hasta que el mostrador está libre).

```
process Estanquero ;
  var ingre : integer ;
begin
  while true do begin
    ingre := ProducirIngrediente();
    Estanco.ponerIngrediente( ingre );
    Estanco.esperarRecogidaIngrediente();
  end
end
```

Variables permanentes y condiciones

Las variables permanentes necesarias se deducen de las esperas que deben hacer las hebras:

- Cada fumador debe esperar a que el mostrador tenga un ingrediente y que ese ingrediente coincida con su número de ingrediente o fumador.
- El estanquero debe esperar a que el mostrador esté vacío (no tenga ningún ingrediente)

Como actividad, debes de escribir (en tu portafolio):

- **Variable o variables permanentes**: para cada una describe el tipo, nombre, valores posibles y significado de la variable.
- **Cola o colas condición**: para cada una, escribe el nombre y la condición de espera asociada (una expresión lógica de las variables permenentes).
- **Pseudo-código** de los tres procedimientos del monitor.

Actividad: implementación del programa

Escribe y prueba un programa que haga la simulación del problema de los fumadores, y que incluya el monitor SU descrito en pseudo-código. En tu portafolio:

- Incluye el código fuente completo de la solución adoptada.
- Incluye un listado de la salida del programa durante una ejecución.

Sección 2.
El problema del barbero durmiente..

Barbería: tipos de hebras.

El problema del barbero durmiente es representativo de cierto tipo de problemas reales: ilustra perfectamente la relación de tipo cliente-servidor que a menudo aparece entre los procesos.

- El problema trata sobre una barbería en la cual hay dos tipos de actores o hebras ejecutándose concurrentemente:
  - Una única hebra llamada *barbero*
  - Varias hebras llamadas *clientes* (un número fijo, n).
- En la barbería hay:
  - Una única silla de cortar el pelo, y
  - Una sala de espera para los clientes, con una cantidad de sillas al menos igual al número de clientes

Estructura del proceso barbero

Respecto al **barbero** ejecuta un bucle infinito, en cada iteración da estos tres pasos:

1. Si no hay clientes en la sala de espera, espera dormido a que llege un cliente a la barbería, o bien, si hay clientes en dicha sala, llama a uno de ellos.
2. Pela al cliente durante un intervalo de tiempo, cuya duración exacta la determina el barbero.
3. Avisa al cliente de que ha terminado de pelarlo.

Estructura de los procesos de clientes

Cada **cliente** ejecuta un bucle infinito, en cada iteración da estos tres pasos:

1. Entra a la barbería. Si el barbero está ocupado pelando, esperará en la sala de espera hasta que el barbero lo llame, o bien, si el barbero está dormido, el cliente despierta al barbero.
2. Espera en la silla de corte a que el barbero lo pele, hasta que el barbero le avisa de que ha terminado. Sale de la barbería.
3. Espera fuera de la barbería durante un intervalo de tiempo, cuya duración exacta la determina el cliente.

Esperas bloqueadas

Las hebras no consumen CPU (es decir, hacen una espera bloqueada) en estos casos:

- Barbero:
  - Cuando no hay clientes que atender (decimos que *duerme*), hasta que lo despierta un cliente.
  - Cuando le está cortando el pelo a un cliente, durante un tiempo aleatorio.
- Cliente:
  - Cuando está en la sala de espera, hasta que lo despierta el barbero.
  - Cuando el barbero le está cortando el pelo, hasta que lo despierta el barbero.
  - Cuando está fuera de la barbería, durante un tiempo aleatorio.

Monitor SU

El objetivo es implementar un monitor SU en C++11 para la sincronización de las hebras, que llamaremos **`Barberia`**. Dicho monitor tiene tres procedimientos exportados, que se describen aquí:

- El cliente número `i` llaman a **`cortarPelo(i)`** para obtener servicio del barbero, despertándolo o esperando a que termine con el cliente anterior (pasos 1 y 2 de cada iteración del cliente)
- El barbero llama a **`siguienteCliente`** para esperar la llegada de un nuevo cliente y servirlo (paso 1 de cada iteración del barbero)
- Cuando el barbero termina de pelar al cliente actual (paso 2) llama a **`finCliente`**, indicándole que puede salir de la barbería y esperando a que lo haga para pasar al siguiente cliente (paso 3).

Procesos cliente y barbero.

El pseudo-código de las hebras, por tanto, es el que se indica aquí:

```
process Cliente[ i : 0..n ] ;
begin
   while true do begin
      { pasos 1 y 2 }
      Barberia.cortarPelo(i);
      { paso 3 }
      EsperarFueraBarberia(i);
   end
end
```

```
process Barbero ;
begin
   while true do begin
      { paso 1 }
      Barberia.siguienteCliente();
      { paso 2 }
      CortarPeloACliente();
      { paso 3 }
      Barberia.finCliente();
   end
end
```

Los procedimientos **`EsperarFueraBarberia(i)`** y **`CortarPeloACliente`** son simples esperas bloqueadas de duración aleatoria.

Actividad: implementación del monitor

Para realizar la actividad, debes de dar estos pasos, reflejando los resultados en tu portafolio:

1. Decide que variables condición y variables permanentes son necesarias. Ten en cuenta que a veces las condiciones que esperan las hebras en una cola pueden incluir condiciones sobre el estado de las otras colas.
2. Para cada variable condición, describe la condición asociada, las hebras que la usan y los puntos donde se hace **`wait`**/**`signal`**.
3. Escribe el pseudo-código del monitor.
4. Escribe la implementación en C+11 del programa completo de simulación de la barbería. Comprueba, viendo la traza, que se ejecuta sin interbloqueos y cumpliendo las condiciones.

Fin de la presentación.