UNIVERSIDAD
DE GRANADA

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

Carlos Ureña / Jose M. Mantas / Pedro Villar

2020-21

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
- El problema de los lectores-escritores

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 de los Lectores/Escritores.

El problema de los Lectores/Escritores.

Dos tipos de procesos acceden concurrentemente a datos compartidos:

- **Escritores**: procesos que modifican la estructura de datos (escriben en ella). El código de escritura no puede ejecutarse concurrentemente con ninguna otra escritura ni lectura (ya que modifica el estado de la estructura de datos)
- **Lectores**: procesos que leen la estructura de datos, pero no modifican su estado en absoluto. El código de lectura puede (y debe) ejecutarse concurrentemente por varios lectores de forma arbitraria, pero no puede hacerse a la vez que la escritura.

La solución de este problema usando semáforos es compleja, veremos que con monitores es sencillo.

Uso del monitor

Los procesos lectores y escritores usan el monitor de esta forma:

```
process Lector[ i:1..n ] ;
begin
   while true do begin
     Lec_Esc.ini_lectura() ;
     { código de lectura }
     Lec_Esc.fin_lectura() ;
     { resto de código }
   end
end
```

```
process Escritor[ i:1..m ] ;
begin
   while true do begin
      Lec_Esc.ini_escritura() ;
      { código de escritura }
      Lec_Esc.fin_escritura() ;
      { resto de código }
   end
end
```

- En esta implementación se ha dado prioridad a los lectores (en el momento que un escritor termina, si hay escritores y lectores esperando, pasan los lectores).
- Hay otras opciones: prioridad a escritores, prioridad al que más tiempo lleva esperando.
- El código de lectura, el de escritura y el resto del código son retrasos aleatorios.

Diseño de la solución: variables permanentes

Para poder introducir las esperas necesarias, tenemos que tener variables permanentes del monitor que nos describan el estado del recurso compartido. En este ejemplo, necesitamos dos variables:

- **`escrib`**
  variable lógica, vale **`true`** si un escritor está escribiendo, **`false`** si no hay escritores escribiendo (inicialmente **`false`**)
- **`n_lec`**
  variable entera (no negativa), es el número de lectores que están escribiendo en un momento dado (inicialmente `0`).

Los valores de estas variables reflejan correctamente el estado del monitor solo cuando no se está ejecutando código del mismo por alguna hebra (en ese caso pueden estar siendo actualizadas).

Diseño de la solución: variables condición

Las esperas se hacen en dos variables condición

- Variable condición **`lectura`**: se usa por los lectores (en **`ini_lectura`**) para esperar cuando ya hay un escritor escribiendo (es decir, cuando **`escrib==true`**).
- Variable condición **`escritura`**: se usa por los escritores (en **`ini_escritura`**) para esperar cuando ya hay otro escritor escribiendo (**`escrib==true`**) o bien hay lectores leyendo (**`n_lec>0`**).

Estas esperas aseguran que siempre se cumple:

```
(not escrib) or (n_lec == 0)
```

Cuando se termina de leer o de escribir, habrá que hacer los corresponientes **`signal`** en estas variables condición.

Vars. permanentes y procedimientos para lectores

Por todo lo dicho, el monitor se puede diseñar así:

```
monitor Lec_Esc ;

var n_lec     : integer;   { numero de lectores leyendo }
    escrib    : boolean;   { true si hay algun escritor escribiendo }
    lectura   : condition; { no hay escrit. escribiendo, lectura posible }
    escritura : condition; { no hay lect. ni escrit., escritura posible }

export ini_lectura, fin_lectura,      { invocados por lectores }
       ini_escritura, fin_escritura ; { invocados por escritores }
```

```
procedure ini_lectura()
begin
 if escrib then { si hay escritor: }
   lectura.wait(); { esperar }
 { registrar un lector más }
 n_lec := n_lec + 1 ;
 { desbloqueo en cadena de }
 { posibles lectores bloqueados }
 lectura.signal()
end
```

```
procedure fin_lectura()
begin
  { registrar un lector menos }
  n_lec:= n_lec - 1 ;
  { si es el ultimo lector:   }
  {   desbloquear un escritor }
  if n_lec == 0 then
     escritura.signal()
end
```

Procedimientos para escritores

Los procedimientos para escritores son estos dos:

```
procedure ini_escritura()
begin
 { si hay otros, esperar }
 if n_lec > 0 or escrib then
   escritura.wait()
 { registrar que hay un escritor }
 escrib := true;
end;
```

```
procedure fin_escritura()
begin
 { registrar que ya no hay escritor}
 escrib := false;
 { si hay lectores, despertar uno}
 { si no hay, despertar un escritor}
 if not lectura.empty() then
    lectura.signal();
 else
    escritura.signal() ;
end;
```

```
begin { inicializacion }
   n_lec  := 0 ;
   escrib := false ;
end
```

Fin de la presentación.