SCD (GIM+GIADE) – Curso 23-24 – Prácticas.

Ejercicios adicionales de la práctica 2.

Nota: en todos los problemas incluye únicamente mensajes (con **cout**) dentro de los métodos de los monitores, de forma que esos **cout** se ejecuten en exclusión mutua sin necesidad de más sincronización que la EM del monitor.

Ejercicio P2.1 (Productores-consumidores múltiples, ver. FIFO, con impresora, con un monitor SU)

Copia tu solución FIFO al problema de los productores-consumidores con monitores SU (basado en la clase **HoareMonitor**) en un archivo nuevo llamado **p2_pc_impr.cpp**, y extiéndelo para cumplir estos requisitos adicionales a los del problema original:

(1) Asegúrate que el número total de items a producir sea múltiplo de 5 (además de múltiplo del número de productores y múltiplo del número de consumidores).

(2) Se debe crear una nueva hebra llamada *impresora*, que ejecuta un bucle. En cada iteración la hebra impresora debe llamar a un nuevo método del monitor llamado **metodo_impresora**. Ese método devuelve un valor lógico. El bucle de la hebra impresora acaba cuando ese método devuelve **false**.

(3) Si al inicio del método **metodo_impresora** la hebra impresora detecta que ya se han insertado en el buffer todos los múltiplos de 5 que se tenían que insertar, devuelve **false**, en otro caso (todavía quedan múltiplos de 5 por insertar), imprime un mensaje (se describe a continuación) y después devuelve **true** (ten en cuenta que en total se deben insertar T/5 múltiplos de 5, donde T es el numero total de items a producir por todos los productores).

(4) En el mensaje que la hebra impresora imprime (en **metodo_impresora**) se debe incluir el número total de nuevos múltiplos de 5 que se han insertado en el vector, contados desde la anterior llamada a **metodo_impresora** o desde el inicio de la simulación (si es la primera llamada). Si dicho número es cero, la hebra debe esperar bloqueada hasta que dicho número sea mayor que 0, antes de imprimir el mensaje (por tanto, nunca imprime un cero).

(5) Ten en cuenta que las hebras productoras son responsables de contabilizar la cantidad de múltiplos de 5 insertados y de desbloquear a la hebra impresora cuando sea necesario hacerlo, todo ello debe gestionarse dentro del método **escribir** del monitor.

Ejercicio P2.2 (Gasolinera, con un monitor SU)

Implementar un programa con un monitor SU (basado en la clase **HoareMonitor**) en un archivo llamado **p2_gasolinera.cpp**, con los siguientes requerimientos:

(1) El programa está formado por 10 hebras que ejecutan un bucle infinito y que representan a coches que necesitan entrar a repostar a una gasolinera un vez en cada iteración de su bucle. Hay dos tipos de hebras coche: 4 hebras de tipo *gasoil* y 6 hebras de tipo *gasolina*.

(2) Para entrar a la gasolinera un coche debe esperar a que quede libre algún surtidor del tipo que necesita. La gasolinera tiene 3 surtidores de gasolina y 2 para gasoil, inicialmente todos libres. Cada tipo de hebra ejecuta una función del programa distinta (**funcion_hebra_gasolina** y **funcion_hebra_gasoil**)

(3) El tiempo de repostaje y la parte de código de los coches que no están en la gasolinera se simulan con retardos aleatorios. Es necesario contabilizar en una variable compartida el número total de surtidores en uso en cada momento. Los coches deben imprimir mensaje cuando logran entrar en la gasolinera (tras esperar) y cuando salen, deben indicar: número y tipo de coche, si entra o sale, y numero de surtidores ocupados.

El monitor debe tener cuatro métodos públicos: **entra_coche_gasoil, sale_coche_gasoil, entra_coche_gasolina, sale_coche_gasolina** (todas tienen como parámetro el número de coche dentro de su tipo). El pseudo-código de las hebras es así.

```
Procedure HebraCocheGasoil[ n : 0..N ]
begin
  while true do begin
    Gasolinera.entrar_coche_gasoil( n )
    { retraso de duración aleatoria (repostando) }
    Gasolinera.sale_coche_gasoil( n )
    { retraso de duración aleatoria (fuera  de gasolinera) }
  end
end
```

```
Procedure HebraCocheGasolina[ m : 0..M ]
begin
  while true do begin
    Gasolinera.entra_coche_gasolina( m )
    { retraso de duración aleatoria (repostando) }
    Gasolinera.sale_coche_gasolina( m )
    { retraso de duración aleatoria (fuera de gasolinera) }
  end
end
```

Ten en cuenta que en los dos métodos **sale_coche_** , las operaciones **signal** deben ser lo último que se ejecute, y por tanto deben de ejecutarse después de cualquier **cout** (si **cout** se hace después de **signal**, el **cout** imprime información desactualizada, ya que dicho **cout** se ejecutaría cuando la hebra señaladora vuelve de la cola de urgentes y la señalada ya ha cambiado el estado del monitor).