Sistemas Concurrentes y Distribuidos. Curso 23-24.

Ejercicios adicionales de la práctica 1.

Ejercicio P1.1. (Productor-consumidor FIFO con impresora, con semáforos)

Copia tu solución FIFO al problema de un productor y un consumidor con vector intermedio (de la práctica 1) en un archivo llamado exactamente **p1_pc_impr.cpp**, y extiéndelo para cumplir estos requisitos adicionales a los del problema original (usando semáforos)

(1) Se creará una nueva variable compartida que contendrá el número de celdas ocupadas en el buffer en cada momento (es decir, elementos producidos y añadidos al vector pero todavía no consumidos). Esta variable se debe actualizar por el productor y por el consumidor, teniendo en cuenta que dicha actualización forma parte del proceso de inserción o extracción de valores del buffer, de forma que el valor de la variable siempre se corresponda con el estado de dicho buffer (es decir, ninguna hebra en ningún momento leerá en esa variable un valor distinto del real según las inserciones y extracciones completadas hasta ese momento)**.**

(2) Se debe crear una nueva hebra llamada *impresora*, que ejecuta un bucle finito. En cada iteración debe bloquearse hasta que sea desbloqueada por otra hebra, y después, tras ser desbloqueada, debe imprimir por pantalla el número de celdas ocupadas que hay en el vector en ese momento

(3) Cuando el productor produzca un número múltiplo de 5, inmediatamente después de insertar dicho número al vector, debe desbloquear a la hebra impresora y luego debe bloquearse hasta ser desbloqueado por la hebra impresora. Hay que tener en cuenta que la hebra impresora debe imprimir el contador de casillas ocupadas resultante de esta última inserción del productor (no un valor posterior distinto, resultado de otras operaciones posteriores distintas a esta última).

(4) La hebra impresora, una vez que haya escrito el mensaje por pantalla, desbloqueará al productor y volverá al principio de su ciclo. El número de iteraciones de la hebra impresora es igual al número de múltiplos de 5 que hay entre 0 y N-1 (ambos incluidos), es decir, será N/5 (división entera), donde N es el número total de valores producidos.

Ejercicio P1.2. (Gasolinera, con semáforos)

Implementar un programa con semáforos (en un archivo llamado **p1_gasolinera.cpp**), con los requerimientos que se indican aquí abajo. Se recomienda hacer una versión cumpliendo (1) a (3) y luego otra cumpliendo esos y además (4).

(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: A hebras de tipo *diésel* y B hebras de tipo *gasolina*. Declara dos constantes para A y B y elige sus valores. Dos o más coches no pueden repostar a la vez en el mismo surtidor, pero sí pueden hacerlo en dos o más surtidores distintos.

(2) Para entrar a la gasolinera un coche debe esperar a que quede libre algún surtidor del tipo que necesita. La gasolinera tiene C surtidores de diésel y D de gasolina, inicialmente todos libres. Cada tipo de hebra ejecuta una función del programa distinta (**funcion_hebra_gasolina** y **funcion_hebra_gasoil**). Declara dos constantes para C y D y elige sus valores, pero asegurate de que se cumple C<A y D<B.

(3) La parte de código de los coches que no están en la gasolinera se simula con retardos aleatorios. El repostaje se simula con una llamada a la función **repostar.** En ella se hace un retraso de duración aleatoria. En esa función, cada hebra coche debe imprimir un mensaje al inicio y otro al final, estos mensajes serán de la forma "*Coche número N de diesel (o gasolina) comienza a (o termina de) repostar*".

(4) Es necesario contabilizar en una variable compartida el número total de surtidores en uso en cada momento. Esa variable se debe incrementar antes de llamar a **repostar** y decrementar después. Inmediatamente después de modificarla (incrementarla o decrementarla) las hebras deben de imprimir el valor de esta variable (el mensaje será de la forma "*El número de surtidores en uso ahora es N*"), asegurándose de que el valor impreso es el valor resultado de ese incremento o decremento, y no se imprime después de otros incrementos o decrementos posteriores.

El pseudo-código de cada tipo de hebra puede ser como se indica a continuación:

```
Procedure HebraTipoX[ n : 0..k ]
begin
  while true do begin
    { esperar hasta que haya un surtidor adecuado libre }
    { incrementar el número de surtidores en uso }
    { imprimir el número de surtidores en uso }
    repostar( n )
    { decrementar el número de surtidores en uso }
    { imprimir el numero de surtidores en uso }
    { señalar que se ha dejado el surtidor libre }
    { retraso de duración aleatoria (fuera de gasolinera) }
  end
end
```