****

Alumno: Pablo M. Moreno Mancebo
Grupo : D1

****

||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

****

Actividades del Seminario

****

1.
I Compila y ejecuta monitor_em.cpp. Veras que ejecuta las
funciones test_1,test_2 y test_3, cada una de ellas usa
uno de los tres monitores descritos.
-
II Verifica que el valor obtenido es distinto del esperado en el
caso del monitor sin exclusiÃ³n mutua. Verifica que los otros dos
monitores (con EM), el valor obtenido coincide con el esperado.
-
III Prueba a quitar el unlock de MContador2::incrementa.
Describe razonadamente que ocurre.
- Mi terminal se ha quedado sin mostrar nada,
mientras se ejecutaba:Interbloqueo

2.
Describe razondamente en tu portafolio el motivo de cada uno de
estos tres hechos:

1. La hebra que entra la Ãºltima al mÃ©todo cita (la hebra
seÃ±aladora) es siempre la primera en salir de dicho mÃ©todo.

-En el for de MBarreraSC::cita(int num_hebra)
el for es desde 0 hasta num_hebras-1 de manera que cada hebra
debe de esperar a que todas las
demÃ¡s invoquen a la cita, excepto la Ãºltima en llegar, que debe
despertar a las demÃ¡s.

de manera que no le afecta : cola.notify_one()

2. El orden en el que las hebras seÃ±aladas logran entrar de nuevo
al monitor no siempre coincide con el orden de salida de wait
(se observa porque los nÃºmeros de orden de entrada no
aparecen ordenados a la salida).

-Es aleatorio ya que depende de un tiempo aleatorio

3. El constructor de la clase no necesita ejecutarse en exclusiÃ³n mutua.

-El constructor solo se invoca una vez y es al crear el monitor al principio del programa fuera de ninguna hebra si no en el mismo main

Prueba a usar notify_all en lugar de notify_one. Describe razonadamente en tu portafolio si se observa o no algÃºn cambio importante en la traza del programa.

-Al ejecutarlo no veo ningun cambio

3. Varios productos y consumidores

```
// prodcons2_sc_lifo.cpp
//
//con multiples productores y consumidores
// Opcion LIFO
//
// -----------------------------------------------------------------------------


#include <iostream>
#include <iomanip>
#include <cassert>
#include <thread>
#include <mutex>
#include <condition_variable>
#include <random>
#include "HoareMonitor.h"

using namespace std ;
using namespace HM ;

constexpr int num_items  = 40 ;     // nÃºmero de items
constexpr int num_hebras_productoras = 10;
constexpr int num_hebras_consumidoras = 10;
const int datos_por_productor = num_items / num_hebras_productoras;
const int datos_por_consumidor = num_items / num_hebras_consumidoras;

mutex mtx;

// contadores de verificaciÃ³n:
unsigned cont_prod[num_items],
        cont_cons[num_items],
```

```
  cont_productores[num_hebras_productoras] = {0};




//aleatorio

template< int min, int max > int aleatorio()
{
  static default_random_engine generador( (random_device())() );
  static uniform_int_distribution<int> distribucion_uniforme( min, max ) ;
  return distribucion_uniforme( generador );
}

//**********************************************************************
// funciones comunes a las dos soluciones (fifo y lifo)
//----------------------------------------------------------------------

int producir_dato(int i)
{
 int dato = i * datos_por_productor + cont_productores[i];

   this_thread::sleep_for( chrono::milliseconds( aleatorio<20,100>() ));

 mtx.lock();
 cout << "producido: " << dato << endl << flush ;
 mtx.unlock();

   cont_prod[dato] ++ ;
 cont_productores[i]++;
   return dato ;
}


void consumir_dato( unsigned dato )
{
   if ( num_items <= dato )
   {
      cout << " dato === " << dato << ", num_items == " << num_items << endl ;
      assert( dato < num_items );
   }
   cont_cons[dato] ++ ;
   this_thread::sleep_for( chrono::milliseconds( aleatorio<20,100>() ));
 mtx.lock();
   cout << "                  consumido: " << dato << endl ;
 mtx.unlock();
}


void ini_contadores()
{
   for( unsigned i = 0 ; i < num_items ; i++ )
```

```
   {  cont_prod[i] = 0 ;
      cont_cons[i] = 0 ;
   }
}

void test_contadores()
{
   bool ok = true ;
   cout << "comprobando contadores ...." << flush ;

   for( unsigned i = 0 ; i < num_items ; i++ )
   {
      if ( cont_prod[i] != 1 )
      {
         cout << "error: valor " << i << " producido " << cont_prod[i] << " veces." << endl ;
         ok = false ;
      }
      if ( cont_cons[i] != 1 )
      {
         cout << "error: valor " << i << " consumido " << cont_cons[i] << " veces" << endl ;
         ok = false ;
      }
   }
   if (ok)
      cout << endl << flush << "soluciÃ³n (aparentemente) correcta." << endl << flush ;
}



class MultProdConsSU : public HoareMonitor
{
 private:
// constantes:
 static const int num_celdas_total = 10;   //  nÃºm. de entradas del buffer

// variables permanentes:
 int buffer[num_celdas_total],             //  buffer de tamaÃ±o fijo, con los datos
     primera_libre ;               //  indice de celda de la prÃ³xima inserciÃ³n

// colas condicion:
 CondVar ocupadas,            //  cola donde esperan los consumidores
     libres ;             //  cola donde esperan los productores


// constructor y mÃ©todos pÃºblicos
 public:
   MultProdConsSU(  ) ;           // constructor
   int  leer();                // extraer un valor (sentencia L) (consumidor)
   void escribir( int valor ); // insertar un valor (sentencia E) (productor)
} ;
// -----------------------------------------------------------------------------
```

```
MultProdConsSU::MultProdConsSU(  )
{
  primera_libre = 0 ;
 ocupadas = newCondVar();
 libres = newCondVar();
}
// -----------------------------------------------------------------------
// funciÃ³n llamada por el consumidor para extraer un dato

int MultProdConsSU::leer(  )
{

  // si no hay ninguna celda ocupada esperamos
  if ( primera_libre == 0 )
    ocupadas.wait();

  // hacer la operaciÃ³n de lectura, actualizando estado del monitor
  assert( 0 < primera_libre  );
  primera_libre-- ;
  const int valor = buffer[primera_libre] ;

  // seÃ±alar al productor que hay un hueco libre, por si estÃ¡ esperando
  libres.signal();

  // devolver valor
  return valor ;
}
// -----------------------------------------------------------------------

void MultProdConsSU::escribir( int valor )
{

  // si todas las celdas estÃ¡n ocupadas esperamos
  if ( primera_libre == num_celdas_total )
    libres.wait();

  //cout << "escribir: ocup == " << num_celdas_ocupadas << ", total == " << num_celdas_total << endl ;
  assert( primera_libre < num_celdas_total );

  // hacer la operaciÃ³n de inserciÃ³n, actualizando estado del monitor
  buffer[primera_libre] = valor ;
  primera_libre++ ;

  // seÃ±alar al consumidor que ya hay una celda ocupada (por si esta esperando)
  ocupadas.signal();
}
// *****************************************************************************
// funciones de hebras

void funcion_hebra_productora( MRef<MultProdConsSU> monitor, int num_hebra )
{
  for(unsigned i = 0; i < datos_por_productor; i++){
```

```
    int valor = producir_dato(num_hebra) ;
    monitor->escribir( valor );
  }
}
// -----------------------------------------------------------------------

void funcion_hebra_consumidora( MRef<MultProdConsSU> monitor )
{
  for( unsigned i = 0 ; i < datos_por_consumidor ; i++ )
  {
    int valor = monitor->leer();
    consumir_dato( valor ) ;
  }
}
// -----------------------------------------------------------------------

int main()
{
  cout << "-----------------------------------------------------------------------------" << endl
       << "Problema de los productores-consumidores (multiples prod/cons, Monitor SC, buffer LIFO). " << endl
       << "-----------------------------------------------------------------------------" << endl
       << flush ;

  MRef<MultProdConsSU> monitor = Create<MultProdConsSU>();

 // Lanzamos todas las hebras
 thread hebras_productoras[num_hebras_productoras];
 for (int i = 0; i < num_hebras_productoras; i++)
  hebras_productoras[i] = thread(funcion_hebra_productora, monitor, i);

 thread hebras_consumidoras[num_hebras_consumidoras];
 for (int i = 0; i < num_hebras_consumidoras; i++)
   hebras_consumidoras[i] = thread(funcion_hebra_consumidora, monitor);

 // Esperamos a todas las hebras
 for (int i = 0; i < num_hebras_productoras; i++)
  hebras_productoras[i].join();

 for (int i = 0; i < num_hebras_consumidoras; i++)
   hebras_consumidoras[i].join();


  // comprobar que cada item se ha producido y consumido exactamente una vez
  test_contadores() ;
}


*****************************************************************************

|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
```

****

Fumadores con Semaforo

****

Archivo cpp : fumadores.cpp
Como actividad, debes de escribir (en tu portafolio):
I Variable o variables permanentes: para cada una describe el
tipo, nombre, valores posibles y significado de la variable.

```
const int NUMERO_DE_FUMADORES = 3;
private:

    int mostrador; //mostrador donde se coloca el ingrediente i, si tiene valor -1 estÃ¡ vacÃ
    CondVar estanquero, fumador[NUMERO_DE_FUMADORES];
```

II Cola o colas condiciÃ³n: para cada una, escribe el nombre y la
condiciÃ³n de espera asociada (una expresiÃ³n lÃ³gica de las
variables permenentes).

CondVar estanquero : para hacer la similitud con los Semaforos
Vector de CondVar de fumadores : fumador[NUMERO_DE_FUMADORES]

III Pseudo-cÃ³digo de los tres procedimientos del monitor.

```
process  ObtenerIngrediente [ i : 0 , 2];
begin
   if(mostrador != i) // Comprueba si su ingrediente esta en mostrador
      fumador[i].wait();//Espera si entra en el if
   mostrador = -1; // Vacia el mostrador y muestra que ha usado el ingrediente
   estanquero.signal();// se avisa al estanquero que el mostrador estÃ¡ vacÃ
end

process Estanco::PonerIngrediente [i];
begin
   mostrador = i; // se pone el ingrediente en el mostrador
   fumador[i].signal(); // se avisa al fumador i que su ingrediente estÃ¡ disponible
end

process Estanco::EsperarRecogida();
begin
   if(mostrador != -1) // Si el mostrador no esta vacio espera bloqueada
      estanquero.wait();
end
```

||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

****

Problema del Barbero

****

Esta casi todos los aspectos comentados en el cpp : barbero.cpp

Aclaracion:

*) Variables CondVar : 3
- silla -> donde cortar el pelo a un cliente
- saladeespera -> donde esperan los clientes a pelarse
- cama -> donde duerme el barbero cuando no hay nadie en la saladeespera