Seminario 2. Introducción a los monitores en C++11.

1. monitor_em

```
// ---------------------------------------------------------------------------
//
// Sistemas concurrentes y Distribuidos.
// Seminario 2. Introducción a los monitores en C++11.
//
// archivo: monitor_em.cpp
// Ejemplo de monitores en C++11 sin variables condición
// (solo con encapsulamiento y exclusión mutua)
//
//  -- MContador1 : sin E.M., únicamente encapsulamiento
//  -- MContador2 : con E.M. mediante clase base 'HoareMonitor' y MRef
//
// Historial:
// Julio 2017: creado
// Sept 2022 : se quita MContador3 antiguo y se adapta MContador2 para usar HoareMonitor
// ---------------------------------------------------------------------------

#include <iostream>
#include <cassert>
#include <thread>
#include <mutex>
#include <random>
#include "scd.h"

using namespace std ;
using namespace scd ;

const int num_incrementos = 10000 ;

// ****************************************************************************
// clase contador, sin exclusión mutua

class MContador1
{
private:
int cont ;

public:
```

```
MContador1( int valor_ini ) ;
void incrementa();
int leer_valor();
} ;
// ----------------------------------------------------------------------------

MContador1::MContador1( int valor_ini )
{
cont = valor_ini ;
}
// ----------------------------------------------------------------------------

void MContador1::incrementa()
{
cont ++ ;
}
// ----------------------------------------------------------------------------

int MContador1::leer_valor()
{
return cont ;
}

// *****************************************************************************
// clase contador, con exclusión mutua mediante herencia de 'HoareMonitor'

class MContador2 : public HoareMonitor
{
private:
int cont ;

public:
MContador2( int valor_ini ) ;
void incrementa();
int leer_valor();
} ;

// ----------------------------------------------------------------------------

MContador2::MContador2( int valor_ini )
{
cont = valor_ini ;
```

```
}
// -----------------------------------------------------------------------

void MContador2::incrementa()
{
cont ++ ;
}
// -----------------------------------------------------------------------

int MContador2::leer_valor()
{
return cont ;
}

// *****************************************************************************

void funcion_hebra_M1( MContador1 & monitor )
{
for( int i = 0 ; i < num_incrementos ; i++ )
monitor.incrementa();
}
// -----------------------------------------------------------------------

void test_1()
{
MContador1 monitor(0) ;

thread hebra1( funcion_hebra_M1, ref(monitor) ),
hebra2( funcion_hebra_M1, ref(monitor) );

hebra1.join();
hebra2.join();

cout  << "Monitor contador (sin exclusión mutua):" << endl
<< endl
<< "  valor esperado == " << 2*num_incrementos << endl
<< "  valor obtenido == " << monitor.leer_valor() << endl
<< endl;
}
// *****************************************************************************

void funcion_hebra_M2( MRef<MContador2>  monitor )
```

```
{
for( int i = 0 ; i < num_incrementos ; i++ )
monitor->incrementa();
}
// -----------------------------------------------------------------------------

void test_2()
{
MRef<MContador2> monitor = Create<MContador2>(0) ;

thread hebra1( funcion_hebra_M2, monitor ),
hebra2( funcion_hebra_M2, monitor );

hebra1.join();
hebra2.join();

cout  << "Monitor contador (EM usando clase derivada de HoareMonitor):" << endl
<< endl
<< "  valor esperado == " << 2*num_incrementos << endl
<< "  valor obtenido == " << monitor->leer_valor() << endl
<< endl ;
}
// *****************************************************************************

int main()
{
test_1();
test_2();
}
```

Seminario 2: Introducción a los monitores en C++11.

2. Productor-Consumidor con monitor SU y buffer FIFO

```
// -----------------------------------------------------------------------------
//
// Sistemas concurrentes y Distribuidos.
// Seminario 2. Introducción a los monitores en C++11.
//
// Archivo: prodcons1_su.cpp
//
// Ejemplo de un monitor en C++11 con semántica SU, para el problema
// del productor/consumidor, con productor y consumidor únicos.
// Opcion LIFO
//
// Historial:
// Creado el 30 Sept de 2022. (adaptado de prodcons2_su.cpp)
// 20 oct 22 --> paso este archivo de FIFO a LIFO, para que se corresponda con lo que dicen las transparencias
// -----------------------------------------------------------------------------


// Se realiza mediante FIFO(primero en entrar es el primero en salir).
// -----------------------------------------------------------------------------


#include <iostream>
#include <iomanip>
#include <cassert>
#include <random>
#include <thread>
#include "scd.h"

using namespace std ;
using namespace scd ;

constexpr int
num_items = 15 ;   // número de items a producir/consumir
int
siguiente_dato = 0 ; // siguiente valor a devolver en 'producir_dato'

constexpr int
```

```
min_ms    = 5,     // tiempo minimo de espera en sleep_for
max_ms    = 20 ;   // tiempo máximo de espera en sleep_for


mutex
mtx ;                 // mutex de escritura en pantalla
unsigned
cont_prod[num_items] = {0}, // contadores de verificación: producidos
cont_cons[num_items] = {0}; // contadores de verificación: consumidos

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

int producir_dato(  )
{

this_thread::sleep_for( chrono::milliseconds( aleatorio<min_ms,max_ms>() ));
const int valor_producido = siguiente_dato ;
siguiente_dato ++ ;
mtx.lock();
cout << "hebra productora, produce " << valor_producido << endl << flush ;
mtx.unlock();
cont_prod[valor_producido]++ ;
return valor_producido ;
}
//----------------------------------------------------------------------

void consumir_dato( unsigned valor_consumir )
{
if ( num_items <= valor_consumir )
{
cout << " valor a consumir === " << valor_consumir << ", num_items == " << num_items <<
endl ;
assert( valor_consumir < num_items );
}
cont_cons[valor_consumir] ++ ;
this_thread::sleep_for( chrono::milliseconds( aleatorio<min_ms,max_ms>() ));
mtx.lock();
cout << "                hebra consumidora, consume: " << valor_consumir << endl ;
mtx.unlock();
}
```

```
//----------------------------------------------------------------------

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

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 << "solución (aparentemente) correcta." << endl << flush ;
}

// *****************************************************************************
// clase para monitor buffer, version FIFO, semántica SC, multiples prod/cons

class ProdConsSU1 : public HoareMonitor
{
private:
static const int           // constantes ('static' ya que no dependen de la instancia)
num_celdas_total = 10;   //   núm. de entradas del buffer
int                        // variables permanentes
buffer[num_celdas_total],//   buffer de tamaño fijo, con los datos
primera_libre           ,//   indice de celda de la próxima inserción
primera_ocupada         ,//   índice de celda de la próxima extracción
n                      ;//   número de celdas ocupadas.

CondVar                    // colas condicion:
ocupadas,                  //  cola donde espera el consumidor (n>0)
libres ;                   //  cola donde espera el productor  (n<num_celdas_total)
```

```
public:                 // constructor y métodos públicos
ProdConsSU1() ;         // constructor
int  leer();            // extraer un valor (sentencia L) (consumidor)
void escribir( int valor ); // insertar un valor (sentencia E) (productor)
} ;
// -----------------------------------------------------------------------

ProdConsSU1::ProdConsSU1(  )
{
primera_libre    = 0 ;
primera_ocupada  = 0;
n              = 0;
ocupadas       = newCondVar();
libres         = newCondVar();
}
// -----------------------------------------------------------------------
// función llamada por el consumidor para extraer un dato

int ProdConsSU1::leer(  )
{
// esperar bloqueado hasta que 0 < primera_libre
if ( n == 0 )    // Si el número de celdas ocupadas es 0 .....
ocupadas.wait();

cout << "leer: ocup == " << n << ", total == " << num_celdas_total << endl ;
assert( 0 < n  );

// hacer la operación de lectura, actualizando estado del monitor
const int valor = buffer[primera_ocupada] ;
n--;
primera_ocupada = (primera_ocupada + 1) % num_celdas_total;

// señalar al productor que hay un hueco libre, por si está esperando
libres.signal();

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

void ProdConsSU1::escribir( int valor )
{
```

```
// esperar bloqueado hasta que primera_libre < num_celdas_total
if ( primera_libre == num_celdas_total )
libres.wait();

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

// hacer la operación de inserción, actualizando estado del monitor
buffer[primera_libre] = valor ;
primera_libre = (primera_libre + 1) % num_celdas_total ;
n++;

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

void funcion_hebra_productora( MRef<ProdConsSU1> monitor )
{
for( unsigned i = 0 ; i < num_items ; i++ )
{
int valor = producir_dato(  ) ;
monitor->escribir( valor );
}
}
// -----------------------------------------------------------------------------

void funcion_hebra_consumidora( MRef<ProdConsSU1>  monitor )
{
for( unsigned i = 0 ; i < num_items ; i++ )
{
int valor = monitor->leer();
consumir_dato( valor ) ;
}
}
// -----------------------------------------------------------------------------

int main()
{
cout << "------------------------------------------------------------" << endl
<< "Problema del productor-consumidor únicos (Monitor SU, buffer LIFO). " << endl
```

```
<< "-----------------------------------------------------------------" << endl
<< flush ;

// crear monitor  ('monitor' es una referencia al mismo, de tipo MRef<...>)
MRef<ProdConsSU1> monitor = Create<ProdConsSU1>() ;

// crear y lanzar las hebras
thread hebra_prod( funcion_hebra_productora, monitor ),
hebra_cons( funcion_hebra_consumidora, monitor );

// esperar a que terminen las hebras
hebra_prod.join();
hebra_cons.join();

test_contadores() ;
}
```