**Jose Miguel Hernández García 2ºC C3**

**Productor - Consumidor con LIFO**

**Semáforos utilizados:**

* Libres: Se inicializa a tam\_vec, es decir, es el número de entradas libres del buffer

(k + #L - #E). La función productor le aplica sem\_wait y la función consumidor le

aplica sem\_signal.

* Ocupadas: Se inicializa a 0 (#L - #E). La función productor le aplica sem\_signal y la función consumidor le aplica sem\_wait.

Donde #E es el total de valores insertados en el buffer desde el inicio y #L es el total de valores extraídos desde el inicio, con la condición de que no pueden haberse extraído más valores de los que se han insertado.

**Variables utilizadas:**

* Buffer[tam\_vec]: Es el buffer que almacena los valores.
* Primera\_libres: Se inicializa a 0, es el índice del vector de la primera celda libre.

**Código:**

#include <iostream>

#include <cassert>

#include <thread>

#include <mutex>

#include <random>

#include "Semaphore.h"

using namespace std ;

using namespace SEM ;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// variables compartidas

const int num\_items = 40 , // número de items

tam\_vec = 10 ; // tamaño del buffer

int buffer[tam\_vec]; // buffer que almacena los valores

int primera\_libre = 0; // índice del vector de la primera celda libre

unsigned cont\_prod[num\_items] = {0}, // contadores de verificación: producidos

cont\_cons[num\_items] = {0}; // contadores de verificación: consumidos

Semaphore libres = tam\_vec,

ocupadas = 0;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// plantilla de función para generar un entero aleatorio uniformemente

// distribuido entre dos valores enteros, ambos incluidos

// (ambos tienen que ser dos constantes, conocidas en tiempo de compilación)

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

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()

{

static int contador = 0 ;

this\_thread::sleep\_for( chrono::milliseconds( aleatorio<20,100>() ));

cout << "producido: " << contador << endl << flush ;

cont\_prod[contador] ++ ;

return contador++ ;

}

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

void consumir\_dato( unsigned dato )

{

assert( dato < num\_items );

cont\_cons[dato] ++ ;

this\_thread::sleep\_for( chrono::milliseconds( aleatorio<20,100>() ));

cout << " consumido: " << dato << endl ;

}

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

void test\_contadores()

{

bool ok = true ;

cout << "comprobando contadores ...." ;

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 ;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Funcion de la hebra productora para una pila acotada (LIFO)

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

void funcion\_hebra\_productora( )

{

for( unsigned i = 0 ; i < num\_items ; i++ ){

int dato = producir\_dato();

sem\_wait(libres);

// Introducimos el dato en el buffer intermedio

cout << "Introducimos valor en el buffer: " << dato << endl;

buffer[primera\_libre] = dato;

primera\_libre++;

sem\_signal(ocupadas);

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Funcion de la hebra consumidora para una pila acotada (FIFO)

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

void funcion\_hebra\_consumidora( )

{

for( unsigned i = 0 ; i < num\_items ; i++ ){

sem\_wait(ocupadas);

// Leemos el dato desde el buffer intermedio

primera\_libre--;

int dato = buffer[primera\_libre];

cout << "Extraemos valor del buffer: " << dato << endl;

sem\_signal(libres);

consumir\_dato(dato);

}

}

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

int main()

{

cout << "--------------------------------------------------------" << endl

<< "Problema de los productores-consumidores (solución LIFO)." << endl

<< "--------------------------------------------------------" << endl

<< flush ;

thread hebra\_productora ( funcion\_hebra\_productora ),

hebra\_consumidora( funcion\_hebra\_consumidora );

hebra\_productora.join() ;

hebra\_consumidora.join() ;

cout << "\*\*\*\*\*\*FIN DEL PROGRAMA \*\*\*\*\*\*" << endl;

test\_contadores();

return 0;

}

**Productor - Consumidor con FIFO**

**Semáforos utilizados:**

* Libres: Se inicializa a tam\_vec, es decir, es el número de entradas libres del buffer

(k + #L - #E). La función productor le aplica sem\_wait y la función consumidor le

aplica sem\_signal.

* Ocupadas: Se inicializa a 0 (#L - #E). La función productor le aplica sem\_signal y la función consumidor le aplica sem\_wait.

Donde #E es el total de valores insertados en el buffer desde el inicio y #L es el total de valores extraídos desde el inicio, con la condición de que no pueden haberse extraído más valores de los que se han insertado.

**Variables utilizadas:**

* Buffer[tam\_vec]: Es el buffer que almacena los valores.
* Primera\_libres: Se inicializa a 0, es el índice del vector de la primera celda libre.
* Primera\_ocupada: Se inicializa a 0, es el índice del vector de la primera celda ocupada

**Código:**

#include <iostream>

#include <cassert>

#include <thread>

#include <mutex>

#include <random>

#include "Semaphore.h"

using namespace std ;

using namespace SEM ;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// variables compartidas

const int num\_items = 40, // número de items

tam\_vec = 10; // tamaño del buffer

int buffer[tam\_vec]; // buffer que almacena los valores

unsigned primera\_ocupada = 0, // índice del vector de la primera celda ocupada

primera\_libre = 0; // índice del vector de la primera celda libre

unsigned cont\_prod[num\_items] = {0}, // contadores de verificación: producidos

cont\_cons[num\_items] = {0}; // contadores de verificación: consumidos

Semaphore libres = tam\_vec,

ocupadas = 0;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// plantilla de función para generar un entero aleatorio uniformemente

// distribuido entre dos valores enteros, ambos incluidos

// (ambos tienen que ser dos constantes, conocidas en tiempo de compilación)

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

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()

{

static int contador = 0 ;

this\_thread::sleep\_for( chrono::milliseconds( aleatorio<20,100>() ));

cout << "producido: " << contador << endl << flush ;

cont\_prod[contador] ++ ;

return contador++ ;

}

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

void consumir\_dato( unsigned dato )

{

assert( dato < num\_items );

cont\_cons[dato] ++ ;

this\_thread::sleep\_for( chrono::milliseconds( aleatorio<20,100>() ));

cout << " consumido: " << dato << endl ;

}

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

void test\_contadores()

{

bool ok = true ;

cout << "comprobando contadores ...." ;

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 ;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Funcion de la hebra productora para una cola circular (FIFO)

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

void funcion\_hebra\_productora( )

{

for( unsigned i = 0 ; i < num\_items ; i++ ){

int dato = producir\_dato();

sem\_wait(libres);

// Introducimos el valor en el buffer intermedio

cout << "Introducimos el valor en el buffer: " << dato << endl;

buffer[primera\_libre] = dato;

primera\_libre++;

sem\_signal(ocupadas);

primera\_libre %= tam\_vec;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Funcion de la hebra consumidora para una cola circular (FIFO)

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

void funcion\_hebra\_consumidora( )

{

for( unsigned i = 0 ; i < num\_items ; i++ ){

sem\_wait(ocupadas);

// Leemos el dato desde el buffer intermedio

int dato = buffer[primera\_ocupada];

primera\_ocupada++;

cout << "Extraemos el valor del buffer: " << dato << endl;

sem\_signal(libres);

consumir\_dato(dato);

primera\_ocupada %= tam\_vec;

}

}

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

int main()

{

cout << "--------------------------------------------------------" << endl

<< "Problema de los productores-consumidores (solución LIFO)." << endl

<< "--------------------------------------------------------" << endl

<< flush ;

thread hebra\_productora ( funcion\_hebra\_productora ),

hebra\_consumidora( funcion\_hebra\_consumidora );

hebra\_productora.join() ;

hebra\_consumidora.join() ;

cout << "\*\*\*\*\*\*FIN DEL PROGRAMA \*\*\*\*\*\*" << endl;

test\_contadores();

return 0;

}