## 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;
}
```