Portfolio Práctica 1

Jesús Sánchez de Lechina Tejada

Contents

Primer ejercicio. Productor-consumidor.

Resolución del problema del productor-consumidor haciendo uso de una cola enlazada.

Descripción de las variables utilizadas

A continuación se detalla la utilidad de las variables utilizadas:

`int vect[tam_vect]`: Vector que usaremos de buffer.

Recordamos que `tam_vect` *es una constante proporcionada en el código de ejemplo de valor 10.*

`int primera_ocupada`: Es el índice de lectura de nuestro vector buffer. Se usa por la hebra **consumidora** para saber qué posición tiene que leer.

`int primera_libre`: Es el índice de escritura de nuestro vector buffer. Se usa por la hebra **productora** para saber en qué posición tiene que escribir.

`const bool DEBUG_MODE`: Por defecto false, usada para mostrar mensajes de depuración.

Descripción de los semáforos utilizados

Hemos necesitado un total de dos semáforos para coordinar las lecturas/escrituras en nuestro buffer auxiliar:

`Semaphore ocupadas`: Semáforo que indica el número de datos que han sido producidos y que están a la espera de ser leídos. Es usado por la hebra lectora para saber si puede leer del buffer o no. A su vez es usado por la hebra productora para indicar que ha escrito en el buffer. Inicialmente tiene un valor de 0.

`Semaphore libres`: Semáforo que indica el número de datos que se pueden introducir en el buffer. Es el número de *"plazas restantes"* en el vector auxiliar. Es usado por la hebra lectora para liberar una posición del vector para que pueda ser sobreescrita. Del mismo modo es usada por la hebra productora para saber si puede escribir en el buffer o no. Inicialmente tiene un valor de 10.

Código fuente

El código fuente del programa principal es este:

```
#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
unsigned  cont_prod[num_items] = {0}, // contadores de verificación:
                    // producidos
          cont_cons[num_items] = {0}; // contadores de verificación:
                    // consumidos

int vect[tam_vec];                    // Vector buffer
Semaphore ocupadas = Semaphore(0);    // Semáforo de datos producidos
Semaphore libres = Semaphore(tam_vec);// Semáforo de plazas disponibles

int primera_ocupada = 0;
int primera_libre = 0;

const bool DEBUG_MODE = false; // Cambiar a 1 para mostrar mensajes de
               // depuración

//**********************************************************************
// 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 ;
}

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

void  funcion_hebra_productora(  )
{
   for( unsigned i = 0 ; i < num_items ; i++ )
   {
      libres.sem_wait();
      vect[primera_libre] = producir_dato() ;

      if(DEBUG_MODE)
    cout << "Introduciré " << vect[primera_libre] << " en la pos: " << primera_libre <<  endl;
      primera_libre++;
```

```
      primera_libre = (primera_libre % tam_vec);
      ocupadas.sem_signal();
   }
}

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

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

     ocupadas.sem_wait();
     consumir_dato( vect[primera_ocupada] ) ;

     if(DEBUG_MODE)
       cout << "He cogido " << vect[primera_ocupada] << " de la pos: " << primera_ocupada <<  e

     primera_ocupada++;
     primera_ocupada = (primera_ocupada % tam_vec);

     libres.sem_signal();
    }
}
//----------------------------------------------------------------------

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 (las hebras se han unido)" << endl;

   test_contadores();
}
```

Segundo ejercicio. Fumadores.

Descripción de las variables utilizadas

A continuación se detalla la utilidad de las variables utilizadas:

`int ingrediente`: Almacena el valor producido por la hebra estanquera y sirve para indicarle al fumador correspondiente que tiene su ingrediente listo para fumar.

Descripción de los semáforos utilizados

Se han necesitado 4 semáforos en total, uno para la hebra estanquera y otros tres para las hebras fumadoras:

- **Hebra estanquera:** `sem_estanco` es el semáforo que indica cuándo puede producir un ingrediente la hebra estanquera. Está inicializado a 1, pues puede producir un ingrediente al iniciar el programa, pero no podrá producir más hasta que este sea retirado. Esto se gestiona con un sem_wait a este semáforo antes de comenzar la producción y un sem_signal desde las otras hebras fumadoras a este semáforo cuando hayan retirado el ingrediente para indicar que puede reanudar la producción.
- **Hebras fumadoras:** `sem_cero`, `sem_uno` y `sem_dos`, representan a los fumadores, están todos inicializados a cero, y un flujo de control en la hebra productora le da paso al semáforo correspondiente en función del ingrediente producido. Este se comunica de vuelta con la hebra productora para indicarle que ha retirado el ingrediente.

Código fuente

El código fuente del programa principal es el siguiente:

```
#include <iostream>
#include <cassert>
#include <thread>
#include <mutex>
#include <random> // dispositivos, generadores y distribuciones aleatorias
#include <chrono> // duraciones (duration), unidades de tiempo
#include "Semaphore.h"

using namespace std ;
using namespace SEM ;


Semaphore sem_cero = Semaphore(0),
  sem_uno = Semaphore(0),
  sem_dos = Semaphore(0),
  sem_estanco = Semaphore(1);

int ingrediente;
```

```
//**********************************************************************
// 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 );
}

int Producir()
{
  cout << "La hebra estanquera está produciendo un ingrediente" << endl;
  chrono::milliseconds duracion_producir( aleatorio<20,200>() );
  this_thread::sleep_for( duracion_producir );
  return aleatorio<0,2>();
}

//----------------------------------------------------------------------
// función que ejecuta la hebra del estanquero

void funcion_hebra_estanquero(  )
{

  while(true)
    {
      sem_estanco.sem_wait();
      ingrediente = Producir();
      cout << "Producido " << ingrediente << endl;
      switch(ingrediente)
    {
    case 0:
      sem_cero.sem_signal();
      break;
    case 1:
      sem_uno.sem_signal();
      break;
    case 2:
      sem_dos.sem_signal();
      break;
    default:
      cout << "Suceso imposible" << endl;
```

```
      break;
    }
    }
}

//----------------------------------------------------------------------
// Función que simula la acción de fumar, como un retardo aleatoria de la hebra

void fumar( int num_fumador )
{

   // calcular milisegundos aleatorios de duración de la acción de fumar)
  chrono::milliseconds duracion_fumar( aleatorio<20,200>() );//

   // informa de que comienza a fumar

    cout << "Fumador " << num_fumador << "  :"
         << " empieza a fumar (" << duracion_fumar.count() << " milisegundos)" << endl;

   // espera bloqueada un tiempo igual a ''duracion_fumar' milisegundos
   this_thread::sleep_for( duracion_fumar );

   // informa de que ha terminado de fumar

    cout << "Fumador " << num_fumador << "  : termina de fumar, comienza espera de ingrediente

}

//----------------------------------------------------------------------
// función que ejecuta la hebra del fumador
void  funcion_hebra_fumador( int num_fumador )
{
   while( true )
   {
     switch(num_fumador)
       {
       case 0:
     sem_cero.sem_wait();
     cout << "Hebra cero coge su ingrediente: 0" << endl;
     sem_estanco.sem_signal();
     fumar(num_fumador);
     break;
       case 1:
     sem_uno.sem_wait();
     cout << "Hebra uno coge su ingrediente: 1" << endl;
     sem_estanco.sem_signal();
     fumar(num_fumador);
     break;
```

```
        case 2:
      sem_dos.sem_wait();
      cout << "Hebra dos coge su ingrediente: 2" << endl;
      sem_estanco.sem_signal();
      fumar(num_fumador);
      break;
        default:
      cout << "Suceso imposible" << endl;
      break;
    }
    }

}

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

int main()
{
  cout << "Problema de los fumadores" << endl;

  thread hebra_estanquera(funcion_hebra_estanquero),
    hebra_fumadora_cero(funcion_hebra_fumador, 0),
    hebra_fumadora_uno(funcion_hebra_fumador,1),
    hebra_fumadora_dos(funcion_hebra_fumador, 2);


  // Al tratarse de bucles infinitos, este bloque nunca se ejecutará
  hebra_estanquera.join();
  hebra_fumadora_cero.join();
  hebra_fumadora_uno.join();
  hebra_fumadora_dos.join();


  cout << "Fin (las hebras se han unido)" << endl;

}
```