Jose Miguel Hernández García 2ºC C3

Productor - Consumidor con LIFO

Semáforos utilizados:

- Mostr_vacio: Se inicializa a 1 ya que el mostrador empieza estando libre y se cambia a 0 cuando hay algo sobre él. La función del estanquero le aplica sem_wait y la función del fumador le aplica sem_signal.

- Ingr_disp[3]: Se trata de un array de semáforos y el valor de cada uno de ellos es #Pi - #Ri. La función estanquero aplica sem_signal al semáforo correspondiente al ingrediente que se ha generado, es decir, si el ingrediente generado es el número 0, se aplica sem_signal al semáforo número 0, por otro lado, la función del fumador le aplica sem_wait al semáforo correspondiente con el num_fumador que se pasa como argumento.

Variables utilizadas:

No se han utilizado variables globales para la resolución de este problema

Código:

```
#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 mostr_vacio  = 1,          // Semáforo del estanquero
          ingr_disp[3] = {0, 0, 0};  // Array de semáforos para los fumadores

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

//----------------------------------------------------------------------
// función que produce un ingrediente aleatorio entre 0 y 2
```

```
int producir()
{
    return aleatorio<0, 2>();
}

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

void funcion_hebra_estanquero(  )
{
    while( true ){
        int ingr = producir();
        sem_wait(mostr_vacio);
        cout << "Puesto el ingrediente numero " << ingr << endl << endl;
        sem_signal(ingr_disp[ingr]);
    }
}

//----------------------------------------------------------------------
// 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." << endl;
}

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

void funcion_hebra_fumador( int num_fumador )
{
    while( true )
    {
        sem_wait(ingr_disp[num_fumador]);
        cout << "                              Retirado el ingrediente numero "
             << num_fumador << endl << endl;
        sem_signal(mostr_vacio);
        fumar(num_fumador);
    }
}

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

int main()
{
    // Declaramos e iniciamos las hebras
    thread hebra_estanquero(funcion_hebra_estanquero),
           hebra_fum1(funcion_hebra_fumador, 0),
```

```
            hebra_fum2(funcion_hebra_fumador, 1),
            hebra_fum3(funcion_hebra_fumador, 2);

    // unimos las hebras
    hebra_estanquero.join();
    hebra_fum1.join();
    hebra_fum2.join();
    hebra_fum3.join();

    return 0;
}
```