**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;

}