

```
/*
```

Problema del estancuero y los fumadores

Realizado por: Jose Santos Salvador

Grupo: B3

```
*/
```

```
/*g++ -std=c++11 -pthread -Wfatal-errors -o fumadores_su_exe santossalvadorp2-1.cpp  
HoareMonitor.cpp Semaphore.cpp*/
```

```
#include <iostream>  
#include <iomanip>  
#include <random>  
#include "HoareMonitor.h"
```

```
using namespace std ;  
using namespace HM;
```

```
constexpr int num_fumador=3;
```

```
//*****  
// 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()  
{
```

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

```
    int ingrediente = aleatorio<0,num_fumador-1>();
```

```
    // informa de que comienza a fumar
```

```
    cout << "\nEstancuero empieza a producir ingrediente "<<ingrediente << " durante " <<  
duracion_producir.count() << " milisegundos" << endl;
```

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

```
    // informa de que ha terminado de producir
```

```

    cout << "\nEstanquero termina de producir " << endl;

    return ingrediente ;
}

class Estanco : public HoareMonitor
{
    private:
        int ing_mostrador;

        CondVar estanquero, fumadores[num_fumador] ;

    public:
        Estanco();

        void obtenerIngrediente(int num_fum);
        void ponerIngrediente(int num_ingrediente);
        void esperarRecogidaIngrediente( );
};

Estanco::Estanco()
{
    ing_mostrador = -1 ;
    estanquero = newCondVar();

    for(int i=0; i<num_fumador; i++)
        fumadores[i] = newCondVar();
}

void Estanco::obtenerIngrediente(int num_fum)
{
    if(ing_mostrador != num_fum)
        fumadores[num_fum].wait();

    cout << "El ingrediente " << num_fum << " ha sido retirado" << endl;
    ing_mostrador = num_fum ;
    estanquero.signal();
}

void Estanco::ponerIngrediente(int num_ingrediente)
{
    ing_mostrador = num_ingrediente ;
    cout << "El ingrediente " << num_ingrediente << "ha sido colocado" <<endl;
    fumadores[num_ingrediente].signal();
}

void Estanco::esperarRecogidaIngrediente()
{
    if(ing_mostrador != -1)
        estanquero.wait();
}

```

```

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

void funcion_hebra_estanquero(MRef<Estanco> monitor)
{
    while( true )
    {
        int i=producir();

        monitor->ponerIngrediente(i);
        monitor->esperarRecogidaIngrediente();
    }
}

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 << "\nFumador " << 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 << "\nFumador " << num_fumador << " : termina de fumar, comienza espera de
    ingrediente." << endl;
}

//-----
// función que ejecuta la hebra del fumador
void funcion_hebra_fumador( MRef<Estanco> monitor, int num_fumador)
{
    while( true )
    {
        monitor->obtenerIngrediente(num_fumador);

        fumar(num_fumador);
    }
}

//-----

int main()
{

```

```
cout << "-----" << endl
<< "Problema de los fumadores-estanquero con semaforos semantica SU." << endl
<< "-----" << endl;
```

```
MRef<Estanco> monitor = Create<Estanco>();
```

```
thread hebra_estanquero (funcion_hebra_estanquero,monitor);
thread hebra_fumador[num_fumador];
for(int i=0; i < num_fumador; i++){
    hebra_fumador[i]= thread (funcion_hebra_fumador,monitor,i);
}
```

```
hebra_estanquero.join();
for(int i=0; i < num_fumador; i++){
    hebra_fumador[i].join();
}
hebra_fumador[num_fumador].join();
```

```
}
```

```

jue, 22 de nov, 23:04
juse@juse-OMEN-by-HP-Laptop: ~/Descargas/practica2

-----
Problema de los fumadores-estanquero con semaforos semantica SU.
-----

Estanquero empieza a producir ingrediente El ingrediente 12 ha sido retirado durante
79 milisegundos

Fumador 2 : empieza a fumar (99 milisegundos)
El ingrediente 1 ha sido retirado

Fumador 1 : empieza a fumar (101 milisegundos)

Estanquero termina de producir
El ingrediente 1ha sido colocado

Fumador 2 : termina de fumar, comienza espera de ingrediente.
El ingrediente 2 ha sido retirado

Estanquero empieza a producir ingrediente 0 durante 53 milisegundos

Fumador 2 : empieza a fumar (144 milisegundos)

Fumador 1 : termina de fumar, comienza espera de ingrediente.
El ingrediente 1 ha sido retirado

Fumador 1 : empieza a fumar (33 milisegundos)

Fumador 1 : termina de fumar, comienza espera de ingrediente.
El ingrediente 1 ha sido retirado

Fumador 1 : empieza a fumar (133 milisegundos)

Estanquero termina de producir
El ingrediente 0ha sido colocado
El ingrediente 0 ha sido retirado

Fumador 0 : empieza a fumar (44 milisegundos)

Estanquero empieza a producir ingrediente 2 durante 121 milisegundos

Fumador 0 : termina de fumar, comienza espera de ingrediente.

Fumador 2 : termina de fumar, comienza espera de ingrediente.
El ingrediente 2 ha sido retirado

Fumador 2 : empieza a fumar (75 milisegundos)

Fumador 1 : termina de fumar, comienza espera de ingrediente.
El ingrediente 1 ha sido retirado

Fumador 1 : empieza a fumar (71 milisegundos)

Estanquero termina de producir
El ingrediente 2ha sido colocado

```

```

/*
Problema Barberia y cliente

Realizado por: Jose Santos Salvador
Grupo: B3

*/

/*g++ -std=c++11 -pthread -Wfatal-errors -o barberia_su_exe santosalvadorp2-2.cpp
HoareMonitor.cpp Semaphore.cpp*/
#include <iostream>
#include <iomanip>
#include <random>
#include "HoareMonitor.h"

using namespace std ;
using namespace HM;

constexpr int numClientes = 3; // constante número de clientes de la barbería

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

// -----

void EsperarFueraBarberia( int num_cliente )
{
    chrono::milliseconds duracion_esperar_fuera( aleatorio<20,200>() );

    cout << "\tCliente " << num_cliente << ": espera fuera de la barberia ("
        << duracion_esperar_fuera.count() << " milisegundos)" << endl;

    this_thread::sleep_for( duracion_esperar_fuera );

    cout << "\tCliente " << num_cliente << ": ha terminado de esperar y se marcha." << endl;
}

void CortarPeloACliente(){

    chrono::milliseconds duracion_cortar( aleatorio<20,200>() );

```

```

    this_thread::sleep_for( duracion_cortar );
}

class Barberia : public HoareMonitor
{

private:
    int cliente;

    CondVar barbero, sala_espera, cliente_en_silla;

public:
    Barberia();
    void finCliente();
    void cortarPelo(int num_cliente);
    void siguienteCliente();
};

Barberia::Barberia()
{
    sala_espera = newCondVar();
    cliente_en_silla = newCondVar();
    barbero = newCondVar();
}

void Barberia::cortarPelo(int num_cliente)
{
    if(barbero.get_nwt() == 0)
        sala_espera.wait();
    else
    {
        cout<<"\nEl cliente numero " << num_cliente << " despierta al babero" << endl;
        barbero.signal();
    }

    cout << "El babero le está cortando el pelo al cliente numero " << num_cliente << endl;
    cliente = num_cliente;
    cliente_en_silla.wait();
}

void Barberia::siguienteCliente()
{
    if(sala_espera.get_nwt() == 0)
    {
        cout << "No hay clientes en la sala de espera" << endl;
        barbero.wait();
    }
    else
    {
        cout << "Pasa un nuevo cliente" <<endl;
        sala_espera.signal();
    }
}

```

```

    }
}

void Barberia::finCliente()
{
    cout << "El cliente " << cliente << " ha finalizado su corte de pelo" << endl;
    cliente_en_silla.signal();
}

void funcion_hebra_cliente( MRef<Barberia> monitor, int num_cliente )
{

    while(true)
    {
        monitor->cortarPelo(num_cliente);
        EsperarFueraBarberia(num_cliente);
    }

}
// -----

void funcion_hebra_barbero( MRef<Barberia> monitor )
{

    while(true)
    {
        monitor->siguienteCliente();
        CortarPeloACliente();
        monitor->finCliente();
    }

}

int main()
{

    cout << "-----" << endl
        << "PROBLEMA DEL BARBERO Y CLIENTE CON MONITOR" << endl
        << "-----" << endl;

    MRef<Barberia> monitor = Create<Barberia>();

    thread barbero(funcion_hebra_barbero, monitor);
    thread clientes[numClientes];

    for (int i = 0; i < numClientes; i++)
        clientes[i] = thread (funcion_hebra_cliente, monitor, i);

    barbero.join() ;

    for (int i = 0; i < numClientes; i++)
        clientes[i].join();
}

```

```
_clientes[numClientes].join();  
}
```

```
Actividades Terminal jue, 22 de nov, 23:04  
juse@juse-OMEN-by-HP-Laptop: ~/Descargas/practica2  
-----  
PROBLEMA DEL BARBERO Y CLIENTE CON MONITOR  
-----  
No hay clientes en la sala de espera  
  
El cliente numero 0 despierta al babero  
El babero le está cortando el pelo al cliente numero 0  
El cliente 0 ha finalizado su corte de pelo  
  Cliente 0: espera fuera de la barberia (148 milisegundos)  
Pasa un nuevo cliente  
El babero le está cortando el pelo al cliente numero 1  
  Cliente 0: ha terminado de esperar y se marcha.  
El cliente 1 ha finalizado su corte de pelo  
Pasa un nuevo cliente  
El babero le está cortando el pelo al cliente numero 2  
  Cliente 1: espera fuera de la barberia (70 milisegundos)  
El cliente 2 ha finalizado su corte de pelo  
Pasa un nuevo cliente  
El babero le está cortando el pelo al cliente numero 0  
  Cliente 2: espera fuera de la barberia (80 milisegundos)  
  Cliente 1: ha terminado de esperar y se marcha.  
El cliente 0 ha finalizado su corte de pelo  
  Cliente 0: espera fuera de la barberia (171 milisegundos)  
Pasa un nuevo cliente  
El babero le está cortando el pelo al cliente numero 1  
  Cliente 2: ha terminado de esperar y se marcha.  
El cliente 1 ha finalizado su corte de pelo  
  Cliente 1: espera fuera de la barberia (91 milisegundos)  
Pasa un nuevo cliente  
El babero le está cortando el pelo al cliente numero 2  
  Cliente 0: ha terminado de esperar y se marcha.  
  Cliente 1: ha terminado de esperar y se marcha.  
El cliente 2 ha finalizado su corte de pelo  
  Cliente 2: espera fuera de la barberia (41 milisegundos)  
Pasa un nuevo cliente  
El babero le está cortando el pelo al cliente numero 0  
  Cliente 2: ha terminado de esperar y se marcha.  
El cliente 0 ha finalizado su corte de pelo  
  Cliente 0: espera fuera de la barberia (130 milisegundos)  
Pasa un nuevo cliente  
El babero le está cortando el pelo al cliente numero 1  
El cliente 1 ha finalizado su corte de pelo  
  Cliente 1: espera fuera de la barberia (69 milisegundos)  
Pasa un nuevo cliente  
El babero le está cortando el pelo al cliente numero 2  
  Cliente 1: ha terminado de esperar y se marcha.  
  Cliente 0: ha terminado de esperar y se marcha.  
El cliente 2 ha finalizado su corte de pelo  
  Cliente 2: espera fuera de la barberia (124 milisegundos)  
Pasa un nuevo cliente  
El babero le está cortando el pelo al cliente numero 1  
  Cliente 2: ha terminado de esperar y se marcha.  
El cliente 1 ha finalizado su corte de pelo  
  Cliente 1: espera fuera de la barberia (115 milisegundos)  
Pasa un nuevo cliente  
El babero le está cortando el pelo al cliente numero 0
```