

ejercicio3.pdf



Clarasdfgh



Sistemas Concurrentes y Distribuidos



2º Grado en Ingeniería Informática



Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

quieres trabajar en Wuolah??

tú puedes ayudarnos a llevar **WUOLAH**
al siguiente nivel (o alguien que conozcas)

**TE
BUSCAMOS**



sin ánimo de lucro, chequea esto:

quieres trabajar
en Wuolah??

TE BUSCAMOS

sin ánimo
de lucro,
chequea esto:



tú puedes
ayudarnos a
llevar
WUOLAH
al siguiente
nivel
(o alguien que
conozcas)

Examen prácticas 1, 2 -- SCD 2020

Ejercicio 3:

Modifica tu solución al problema de los fumadores de la práctica 2, tal como se indica a continuación, adjuntando el archivo .cpp resultante con nombre ejercicio3.cpp:

Habrà una nueva hebra denominada suministradora que irá suministrando ingredientes continuamente al estanquero mediante un monitor buffer con capacidad para cinco ingredientes que seguirá una estrategia de gestión FIFO. La hebra suministradora estará continuamente generando ingredientes (0, 1, ó 2) y escribiéndolos en el buffer.

El estanquero no producirá ingredientes por sí mismo sino que los leerá del buffer compartido con la suministradora. De hecho, el estanquero no podrá poner un nuevo ingrediente en el mostrador hasta que lo lea del buffer.

La hebra suministradora tendrá que esperar si el buffer está lleno al intentar escribir y la hebra estanquera tendrá que esperar si el buffer está vacío al intentar leer un ingrediente.

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

using namespace std ;
using namespace HM;

//*****
// variables compartidas

const int num_items = 3;

mutex    mtx; //Mutex para la salida por pantalla

//*****
// 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 simula la acción de producir un ingrediente, como un retardo
```

WUOLAH

```

// aleatorio de la hebra (devuelve número de ingrediente producido)

int producir_ingrediente()
{
    // calcular milisegundos aleatorios de duración de la acción de fumar)
    chrono::milliseconds duracion_produ( aleatorio<10,100>() );

    // informa de que comienza a producir
    cout << "Estanquero : empieza a producir ingrediente (" <<
duracion_produ.count() << " milisegundos)" << endl;

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

    const int num_ingrediente = aleatorio<0,num_items-1>() ;

    // informa de que ha terminado de producir
    cout << "Estanquero : termina de producir ingrediente " << num_ingrediente <<
endl;

    return num_ingrediente ;
}

//*****
// Monitor hoare Suministrador
//-----

class Suministrador : public HoareMonitor
{
private:
    static const int tam_buffer = 5;
    int      buffer[tam_buffer],
           pos;

    CondVar produce, consume;

public:
    Suministrador( );
    void producirIngrediente( );
    int consumirIngrediente( );
};

// -----
Suministrador::Suministrador( ){
    for (int i = 0; i < tam_buffer; i++){
        buffer[i] = 0;
    }
    produce = newCondVar();
    consume = newCondVar();
    pos     = 0;
}

// -----
void Suministrador::producirIngrediente( ){
    if(pos == tam_buffer){
        produce.wait();
    }
}

```



Cerveceros de España recomienda el consumo responsable.

Cuando disfrutas de tu gente y de la cerveza,
con cabeza, disfrutas el doble.



**UNA GRAN CERVEZA.
UNA GRAN RESPONSABILIDAD.**


```

    int ingrediente = producir_ingrediente();
    buffer[pos] = ingrediente;

    pos++;

    consume.signal();
}

// -----
int Suministrador::consumirIngrediente( ){
    if(pos == 0){
        consume.wait();
    }
    int ingrediente = buffer[0];

    pos--;

    for(int i = 0; i < pos; i++){
        buffer[i] = buffer[i+1];
    }

    produce.signal();

    return ingrediente;
}

//*****
// Monitor hoare Estanco
//-----

class Estanco : public HoareMonitor
{
private:
    int    mostrador;

    CondVar estanquero,
            fumadores[num_items];

public:
    Estanco( );
    void obtenerIngrediente( int ing );
    void ponerIngrediente( int ing );
    void esperarRecogidaIngrediente( );
};

// -----
Estanco::Estanco( ){
    mostrador = -1;    //mostrador vacío

    for (int i = 0; i < num_items; i++){
        fumadores[i] = newCondVar();
    }
    estanquero = newCondVar();
}

```



LOS JUEGOS DEL CUATRI

es el
momento de
presentarte
como tributo



```
// -----  
void Estanco::obtenerIngrediente( int ing ){  
    while(mostrador != ing){  
        fumadores[ing].wait();  
    }  
  
    mtx.lock();  
    cout << "Fumador " << ing << " : toma ingrediente del mostrador" << endl;  
    mtx.unlock();  
  
    mostrador = -1;  
    estanquero.signal();  
}  
  
// -----  
void Estanco::ponerIngrediente( int ing ){  
    esperarRecogidaIngrediente();  
  
    mtx.lock();  
    cout << "Estanquero : ingrediente " << ing << " puesto en el mostrador" <<  
endl;  
    mtx.unlock();  
  
    mostrador = ing;  
    fumadores[ing].signal();  
}  
  
// -----  
void Estanco::esperarRecogidaIngrediente( ){  
    while(mostrador != -1){  
        estanquero.wait();  
    }  
}  
  
//-----  
// Función que ejecuta la hebra del suministrador  
void funcion_hebra_suministradora( MRef<Suministrador> monitor)  
{  
    while( true )  
    {  
        monitor->producirIngrediente();  
    }  
}  
  
//-----  
// Función que ejecuta la hebra del estanquero  
  
void funcion_hebra_estanquero( MRef<Estanco> estanco, MRef<Suministrador>  
suministrador )  
{  
    while( true ){  
        int ingrediente = suministrador->consumirIngrediente();  
  
        estanco->ponerIngrediente(ingrediente);  
        estanco->esperarRecogidaIngrediente();  
    }  
}
```

```

}

//-----
// 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( MRef<Estanco> monitor, int num_fumador )
{
    while( true )
    {
        monitor->obtenerIngrediente(num_fumador);
        fumar(num_fumador);
    }
}

//-----

int main()
{
    cout << "-----" <<
endl
         << " Problema de los fumadores con monitor SU -- examen, ejercicio 3" <<
endl
         << "-----" <<
endl
         << flush ;

    MRef<Estanco> estanco = Create<Estanco>( );
    MRef<Suministrador> suministrador = Create<Suministrador>( );

    thread suministradora;
    thread estanquero;
    thread fumadores[num_items];

```

```

for (int i = 0; i < num_items; i++){
    fumadores[i] = thread (funcion_hebra_fumador, estanco, i);
}
estanquero = thread (funcion_hebra_estanquero, estanco, suministrador);
suministradora = thread (funcion_hebra_suministradora, suministrador);

for (int i = 0; i < num_items; i++){
    fumadores[i].join();
}
estanquero.join();
suministradora.join();

cout << endl << endl;
}

```