

Examen-Monitores-y-Semaforos-res...



Zukii



Sistemas Concurrentes y Distribuidos



2º Grado en Ingeniería Informática



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

ZERO AZÚCAR
**#ZERO
PALABRAS**

DEMASIADO BUENO PARA
EXPLICARLO CON PALABRAS

Coca-Cola
Real Magic™

REAL MAGIC, COCA-COLA ZERO son marcas registradas de The Coca-Cola Company.



quieres trabajar
en Wuolah??

TE BUSCAMOS

@Zukii on Wuolah

Examen 1 SCD Prácticas – Monitores y Semáforos

Semáforos:

Enunciado

Modifica tu solución al problema de los fumadores de la práctica 1 tal como se indica a continuación (mediante el uso de funciones para manejo de hebras y semáforos de C++11), guardando el resultado en un archivo denominado "fumadores2.cpp":

Después que un fumador fume las dos primeras veces, en la tercera iteración, decide no fumar y prefiere echarse a dormir. Para ello:

-Cuando dicho fumador deba retirar su ingrediente en esa tercera iteración, espera primero a que el estancoero ponga su ingrediente en el mostrador, después "retira" su ingrediente (para permitir al estancoero poner más ingredientes) y después se duerme (esperará en un nuevo semáforo asociado a dicho fumador).

-Cuando el estancoero vaya a colocar el cuarto ingrediente de un fumador (el estancoero también debe llevar la cuenta de cuántos ingredientes ha puesto de cada tipo), debe antes avisar a dicho fumador para que se despierte de su dulce sueño y pueda así recoger su ingrediente del mostrador y entrar de nuevo en su funcionamiento normal.

Solución:

```
#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 ;
```

sin ánimo
de lucro,
chequea esto:



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

WUOLAH

@Zukii on Wuolah

```
//*****  
  
// 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)  
//-----  
  
Semaphore mostrador(1);  
const int num_fumadores = 3;  
Semaphore ingredientes[num_fumadores]={0,0,0};  
Semaphore siesta[num_fumadores]={0,0,0};  
int ingredientes_dados[num_fumadores] = {0,0,0};  
const int pausa_fumar = 3;  
  
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  
// 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
```




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.**

@Zukii on Wuolah

```
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_fumadores-1>();

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

return num_ingrediente ;
}

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

void funcion_hebra_estanquero( )
{
    int ingrediente;

    while(true){
        ingrediente = producir_ingrediente();
        cout << ingredientes_dados[0] << "      " << ingredientes_dados[1] << "" <<
ingredientes_dados[2] << endl;
        if(ingredientes_dados[ingrediente] == (pausa_fumar))
        {
            sem_signal(siesta[ingrediente]);
        }

        sem_wait(mostrador);
        cout << "Se ha puesto en el mostrador el ingrediente: " << ingrediente << endl;
```

```
        sem_signal(ingredientes[ingrediente]);
    }
}

//-----
// 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 << 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 << num_fumador << " : para de fumar " << endl;

}

// función que ejecuta la hebra del fumador
void funcion_hebra_fumador( int num_fumador )
{
```



@Zukii on Wuolah

```
while( true )
{
    sem_wait(ingredientes[num_fumador]);

    cout << "Se ha retirado del mostrador el ingrediente:" << num_fumador <<
endl;

    ingredientes_dados[num_fumador]++;

    sem_signal(mostrador);

    if(ingredientes_dados[num_fumador] == pausa_fumar){
        cout << num_fumador << "se duerme " << endl;
        sem_wait(siesta[num_fumador]);
        cout << num_fumador << "se despierta " << endl;
    }

    else{
        fumar(num_fumador);
    }
}

int main()
{
    thread hebra_estanquero(funcion_hebra_estanquero);
    thread hebras_fumadores[num_fumadores];

    for(int i = 0; i < num_fumadores; i++)
        hebras_fumadores[i] = thread(funcion_hebra_fumador,i);

    hebra_estanquero.join();
}
```

es el
momento de
presentarte
como tributo



Enunciado

Modifica tu solución al problema de los fumadores de la práctica 2, tal como se indica a continuación (mediante el uso de funciones para manejo de hebras y monitores de C++11), guardando el resultado en un archivo denominado "fumadores-p2.cpp" que será adjuntado.

Después que un fumador fume las dos primeras veces, en la tercera iteración, decide no fumar y prefiere echarse a dormir. Para ello:

- Cuando dicho fumador deba retirar su ingrediente en esa tercera iteración, espera primero a que el estancero ponga su ingrediente en el mostrador, después "retira" su ingrediente (para permitir al estancero poner más ingredientes) y después se duerme incondicionalmente (deberá invocar un nuevo método "dormir" del monitor Estanco).
- Cuando el estancero vaya a colocar el cuarto ingrediente de un fumador (dentro del monitor estanco también se debe llevar la cuenta de cuántos ingredientes se han puesto para cada tipo de ingrediente), debe antes avisar a dicho fumador para que se despierte de su dulce sueño. Esto puede implementarse, modificando el método poner_Ingrediente del monitor Estanco.
- Una vez el fumador es despertado, podrá recoger su cuarto ingrediente del mostrador y entrar de nuevo en su modo de funcionamiento normal.

Solución:

```
#include <iostream>
#include <iomanip>
#include <cassert>
#include <thread>
#include <condition_variable>
#include <random>
#include <mutex>
#include "HoareMonitor.h"
```

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

```
const int num_fumadores = 3; //num de fumadores
```


@Zukii on Wuolah

```
const int pausa_dormir = 3;
```

```
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
```

```
// 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_fumadores-1>();
```

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

```
cout << "Estanquero : termina de producir ingrediente " << num_ingrediente << endl;
```

```
return num_ingrediente ;
```

@Zukii on Wuolah

```
}
```

```
//-----
```

```
// Funci3n que simula la acci3n de fumar, como un retardo aleatoria de la hebra
```

```
void fumar( int num_fumador )
```

```
{
```

```
    // calcular milisegundos aleatorios de duraci3n de la acci3n 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;
```

```
}
```

```
//
```

```
*****
```

```
//
```

```
*****
```

```
//
```

```
*****
```

```
//
```

```
*****
```

Que no te escriban poemas de amor
cuando terminen la carrera ▶▶▶▶▶▶▶▶▶▶



WUOLAH

(a nosotros por suerte nos pasa)

No si antes decirte
Lo mucho que te voy a recordar

Pero me voy a graduar.
Mañana mi diploma y título he de
pagar

Llegó mi momento de despedirte
Tras años en los que has estado mi
lado.

Siempre me has ayudado
Cuando por exámenes me he
agobiado

Oh Wuolah wuolilah
Tu que eres tan bonita

@Zukii on Wuolah

// clase para monitor estanco, semántica SU.

class Estanco : public HoareMonitor

{

private:

int mostrador;

int ingrediente_recibidos[num_fumadores];

CondVar mostrador_vacio;

CondVar esta_mi_ig[num_fumadores];

CondVar siesta[num_fumadores];

public:

Estanco(); // constructor

void ponerIngrediente(int ponerIngrediente);

void esperarRecogidaIngrediente();

void obtenerIngrediente(int ig);

void dormir(int fumador);

int num_ig_recibidos(int fumador);

void ig_recibido(int fumador);

};

// -----

Estanco::Estanco()

{

mostrador = -1; //Mostrador vacio inicialmente

mostrador_vacio = newCondVar();

WUOLAH

@Zukii on Wuolah

```
        for(int i = 0 ; i < num_fumadores ; i++)
        {
            esta_mi_ig[i] = newCondVar();
            siesta[i] = newCondVar();
        }

    }

    // -----
void Estanco :: ponerIngrediente(int ig)
{
    mostrador = ig; //Colocamos el ingrediente en el mostrador

    if(num_ig_recibidos(ig) == pausa_dormir)
    {
        siesta[ig].signal();
        cout << ig << " se despierta" << endl;
    }

    esta_mi_ig[ig].signal();//Llamamos al fumador ig para que lo recoja
}

// -----
void Estanco :: esperarRecogidaIngrediente()
{
    if(mostrador != -1) //Si estÃ¡ vacÃ­o el mostrador
        mostrador_vacio.wait();

}

// -----
```

@Zukii on Wuolah

```
void Estanco :: obtenerIngrediente(int ig)
{
    if(mostrador != ig)          //Si no estÃ¡ el ingrediente, esperar a que estÃ©
        esta_mi_ig[ig].wait();

    mostrador = -1; //El mostrador se encuentra vacÃ­o

    mostrador_vacio.signal();    //SeÃ±alar que esta vacio el mostrador
}

void Estanco :: dormir(int fumador)
{
    cout << fumador << " se duerme" << endl;
    siesta[fumador].wait();
}

//-----

int Estanco :: num_ig_recibidos(int fumador)
{
    return(ingrediente_recibidos[fumador]);
}

void Estanco :: ig_recibido(int fumador)
{
    ingrediente_recibidos[fumador]++;
}

//-----

// funciÃ³n que ejecuta la hebra del estanquero
```



```
void funcion_hebra_estanquero(MRef<Estanco> monitor)
{
    int ingrediente;

    while(true){
        ingrediente = producir_ingrediente();
        monitor->ponerIngrediente(ingrediente);
        cout << "Se ha puesto en el mostrador el ingrediente: " << ingrediente << endl;
        monitor->esperarRecogidaIngrediente();

        /*
        cout << endl << "Contador Ingredientes: " << endl;

        for(int i = 0 ; i < num_fumadores ; i++)
            cout << monitor->num_ig_recibidos(i) << "    ";

        cout << endl << endl;
        */
    }

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

            monitor->obtenerIngrediente(num_fumador);
            cout << "Se ha retirado del mostrador el ingrediente: " << num_fumador << endl;
```

WUOLAH

@Zukii on Wuolah

}

Consejo: Ten hechas las sesiones y ejercicios propuestos correctamente ya que los ejercicios suelen ser unas leves modificaciones que si cambiar de FIFO a LIFO y cosas del estilo

Anotación: La nota del examen fue de 9 sobre 10.

Zukii

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

si lees esto me debes un besito