

# Examenmonitorespracticassolucion...



Anónimo



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

```
#include <iostream>
#include <iomanip>
#include <cassert>
#include <ranom>
#include <thread>
#include "scd.h"
```

```
using namespace std ;
using namespace scd ;
```

```
constexpr int
    num_items = 30 ; // número de items a producir/consumir
int
    siguiente_dato = 0 ; // siguiente valor a devolver en 'producir_dato'
```

```
constexpr int
    min_ms    = 5, // tiempo mínimo de espera en sleep_for
    max_ms    = 20 ; // tiempo máximo de espera en sleep_for
```

```
mutex      mtx ; // mutex de escritura en pantalla
```

```
const int num_consumidoras=3, // Número de hebras consumidora
          num_productoras=2; // Número de hebras productoras
```

```
/**
// funciones comunes a las dos soluciones (fifo y lifo)
//-----
```

```
/**
```

```
unsigned producir_dato(const int num_hebra, int j)
{
    this_thread::sleep_for( chrono::milliseconds( aleatorio<min_ms,max_ms>() ));
    const int dato=num_hebra*num_items/num_productoras+j;
    mtx.lock();
    cout << "producido: " << dato << endl << flush ;
    mtx.unlock();
    return dato ;
}
```

```
//-----
void consumir_dato( const int num_hebra, unsigned dato )
```

sin ánimo  
de lucro,  
chequea esto:



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

WUOLAH

```

{

this_thread::sleep_for( chrono::milliseconds( aleatorio<min_ms,max_ms>() ));
mtx.lock();
cout << "          hebra consumidora " << num_hebra << " consume: " << dato << endl << flush ;
mtx.unlock();
}

//-----

// *****
// clase para monitor buffer, version FIFO, semántica SU, multiples prod/cons
class Buffer : public HoareMonitor
{
private:
static const int      // constantes ('static' ya que no dependen de la instancia)
    num_celdas_total = 5; // núm. de entradas del buffer
int                // variables permanentes
    buffer[num_celdas_total], // buffer de tamaño fijo, con los datos
    primera_libre ,
    primera_ocupada,
    n;                // Contador de items disponibles para leer

CondVar            // colas condicion:
    ocupadas,        // cola donde espera el consumidor (n>0)
    libres ;          // cola donde espera el productor (n<num_celdas_total)

public:              // constructor y métodos públicos
    Buffer() ;         // constructor
    int leer();         // extraer un valor (sentencia L) (consumidor)
    void escribir( int valor ); // insertar un valor (sentencia E) (productor)
} ;
// -----

Buffer::Buffer( )
{
    primera_libre = 0 ;
    primera_ocupada = 0;
    n = 0;
    ocupadas = newCondVar();
    libres = newCondVar();
}
// -----
// función llamada por el consumidor para extraer un dato

```



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 Buffer::leer( )
{
    // esperar bloqueado hasta que 0 < primera_libre
    //Cambiamos la condicion para el funcionamiento correcto
    if ( n == 0 )
        ocupadas.wait();

    //cout << "leer: ocup == " << primera_libre << ", total == " << num_celdas_total << endl ;
    //assert( 0 < primera_libre );

    // hacer la operación de lectura, actualizando estado del monitor
    const int valor = buffer[primera_ocupada] ;
    primera_ocupada = (primera_ocupada + 1) % num_celdas_total;;
    n--;

    // señalar al productor que hay un hueco libre, por si está esperando
    libres.signal();

    // devolver valor
    return valor ;
}
// -----

```

```

void Buffer::escribir( int valor )
{
    // esperar bloqueado hasta que primera_libre < num_celdas_total
    //Cambiamos la condicion para el funcionamiento correcto
    if( n == num_items)
        libres.wait();

    // Insercion, actualizando estado del monitor
    buffer[primera_libre] = valor ;
    primera_libre = (primera_libre + 1) % num_celdas_total;
    n++;

    // Avisar de que ya hay una celda ocupada (por si esta esperando)
    ocupadas.signal();
}

```

// clase Calculadora

```

class Calculadora : public HoareMonitor
{
private:

```



este es el único banco  
del que te tienes  
que preocupar este mes

WUOLAH

si no entiendes nada...



```
int n, escri, lecturas ;  
CondVar condicion;
```

```
public:  
Calculadora() ;  
void count(int num_consumidor);  
int get_value();  
} ;  
// -----
```

```
Calculadora::Calculadora()  
{  
    n = 0 ;  
    escri = 0;  
    lecturas = 0;  
    espera = newCondVar();  
}  
// -----
```

```
void Calculadora::count(int num_consumidor)  
{  
    if(num_consumidor == 0)  
        escri++;  
    n++;  
    if(lecturas < increm){  
        condicion.signal();  
    }  
}  
// -----
```

```
int Calculadora::get_value()  
{  
    if(lecturas >= escri){  
        condicion.wait();  
    }  
    lecturas++;  
    return n ;  
}
```

```
// funciones de hebras
```

```
// *****
```

```
// -----  
void funcion_hebra_productora( MRef<Buffer> monitor, const int num )  
{  
    for( unsigned j = 0 ; j < num_items/num_productoras ; j++ )  
    {  
        int dato = producir_dato(num,j) ;
```



Este  
noviembre  
lo entenderás

WUOLAH

WUOLAH

```

        monitor->escribir( dato );
    }
}

// -----
void funcion_hebra_consumidora( MRef<Buffer> monitor, const int num, MRef<Calculadora>
calculadora )
{
    for( unsigned j = 0 ; j < num_items/num_consumidoras ; j++ )
    {
        int valor = monitor->leer();
        calculadora->count(num);
        consumir_dato( num,valor ) ;
    }
}

// -----
void visualizadora( MRef<Calculadora> calculadora )
{
    for( unsigned j = 0 ; j < num_items/num_consumidoras ; j++ )
    {
        chrono::milliseconds duracion(aleatorio<10,25>());
        int lecturas = calculadora->get_value();
        this_thread::sleep_for( duracion );
        cout << ".....HEMOS LEÍDO " << lecturas << " ITEMS EN TOTAL " <<
endl;
    }
}

// -----
int main()
{
    cout << "-----" << endl
        << "  Problema de productores-consumidores (Monitor SU, buffer FIFO).  " << endl
        << "-----" << endl
        << flush ;

    MRef<Buffer> monitor = Create<Buffer>() ;
    MRef<Calculadora> calculadora = Create<Calculadora>() ;
    thread hebra_prod[num_productoras], hebra_cons[num_consumidoras], hebra_visu;

    for( unsigned i = 0 ; i < num_productoras ; i++ )
        hebra_prod[i]=thread ( funcion_hebra_productora, monitor,i );

```

```
for( unsigned i = 0 ; i < num_consumidoras ; i++ )
    hebra_cons[i]=thread (funcion_hebra_consumidora, monitor, i, calculadora );

    hebra_visu=thread ( visualizadora, calculadora );

// esperar a que terminen las hebras
for( unsigned i = 0 ; i < num_productoras ; i++ )
    hebra_prod[i].join();
for( unsigned i = 0 ; i < num_consumidoras ; i++ )
    hebra_cons[i].join();

    hebra_visu.join();

}
```