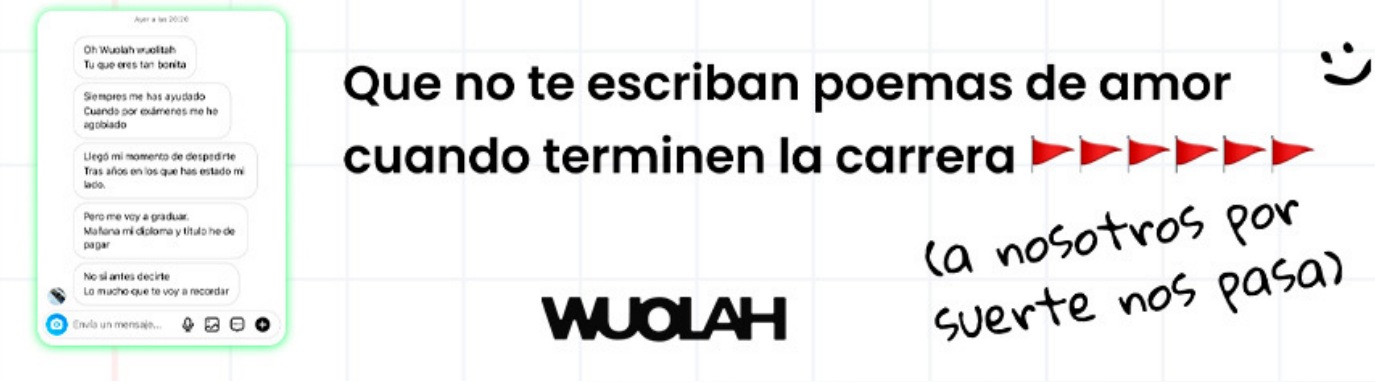
Examen-Sem-y-Monitor-resuelto.pdf

Zukii

Sistemas Concurrentes y Distribuidos

2º Grado en Ingeniería Informática

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



@Zukii on Wuolah

Examen 1 SCD Prácticas – Monitores y Semáforos - Otro

Semáforos:

Enunciado

Modificar la solución al problema de los fumadores planteado en la Práctica 1 tal como se muestra a continuación, adjuntando el archivo .cpp resultante con nombre ejercicio1.cpp:

a.Habrá dos nuevas hebras que suministrarán continuamente ingredientes al estanquero, denominadas suministradora[0] y suministradora[1]. Estas hebras serán prácticamente idénticas e irán suministrando ingredientes continuamente al estanquero mediante un vector buffer con capacidad para 3 ingredientes que seguirá una estrategia de inserción/extracción FIFO. Las dos hebras suministradoras estarán continuamente generando ingredientes (0, 1 ó 2) y escribiéndolos en el vector buffer.

b.El estanquero no producirá ingredientes por sí mismo, sino que los leerá del vector compartido con las suministradoras (vector buffer). De hecho, el estanquero no podrá poner un nuevo ingrediente en el mostrador hasta que lo lea del vector buffer.

c.Las hebras suministradoras tendrán que esperar si el vector buffer está lleno al intentar escribir en el vector, y la hebra estanquera tendrá que esperar si el buffer está vacío al intentar leer un nuevo ingrediente del vector.

d.Se requiere que las hebras suministradoras y fumadoras se describan usando una única función parametrizada en base a un índice entero para cada grupo, y que la solución use arrays de hebras y arrays de semáforos (cuando proceda).

Solución:

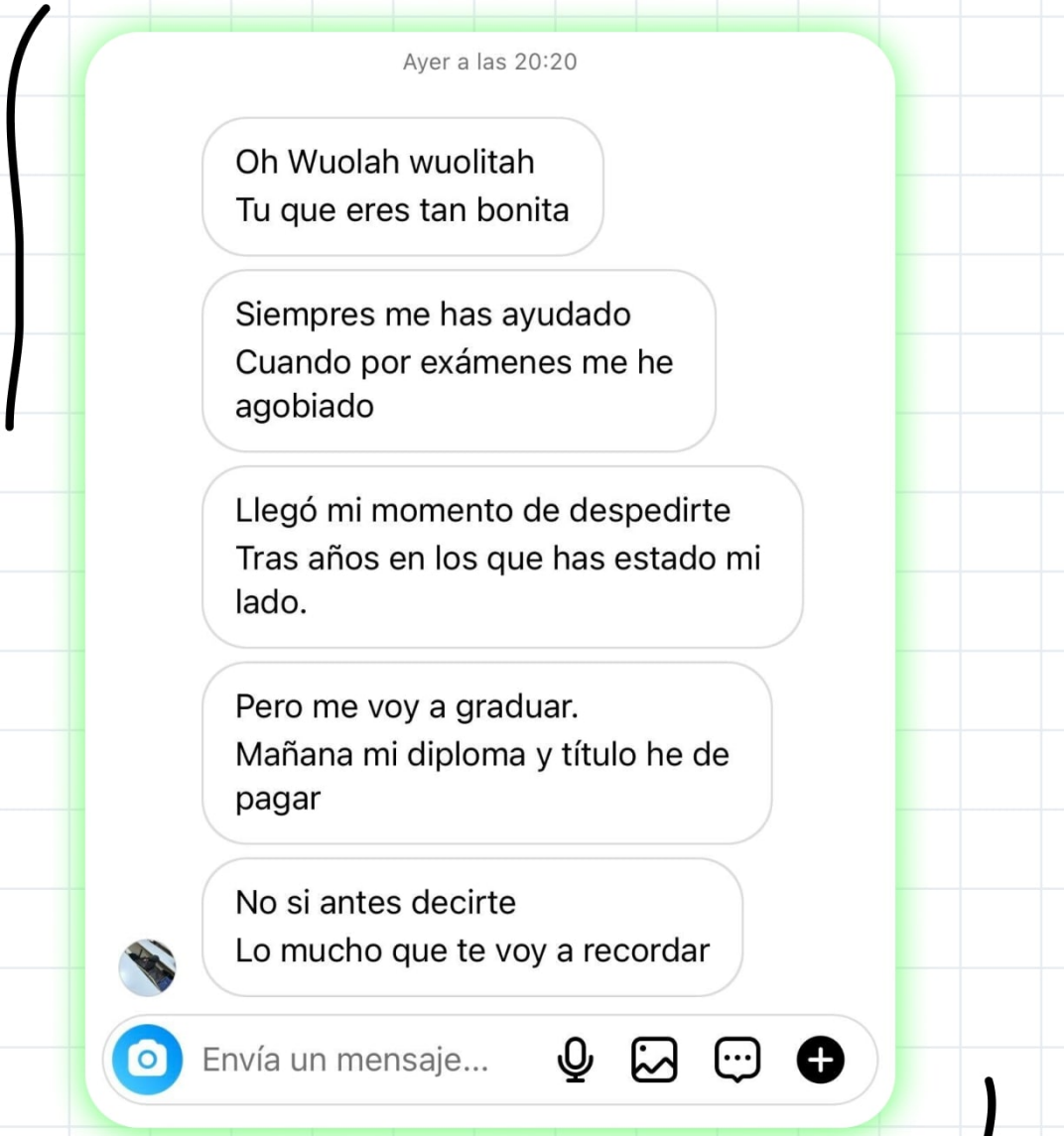
```
#include <iostream>
#include <cassert>
#include <thread>
#include <mutex>
#include <random> // dispositivos, generadores y distribuciones aleatorias
#include <chrono> // duraciones (duration), unidades de tiempo
#include "Semaphore.h"
```

@Zukii on Wuolah

```
using namespace std ;
using namespace SEM ;

//**********************************************************************
// plantilla de funciÃ³n para generar un entero aleatorio uniformemente
// distribuido entre dos valores enteros, ambos incluidos
// (ambos tienen que ser dos constantes, conocidas en tiempo de compilaciÃ³n)
//----------------------------------------------------------------------

Semaphore mostrador(1);
const int num_fumadores = 3;
const int tam_vec = 5;  // tamaÃ±o del buffer
Semaphore ingredientes[num_fumadores]={0,0,0};

const int num_suministradoras = 2;       //num hebras productoras(suministradora) - divisores de num_items
const int nc = 1;//num hebras consumidoras(estanco) - divisores de num_items

int buffer[tam_vec];     //Buffer intermedio. Vector de tam: "tam_buffer"
int       pos_escritura = 0;         //
int       pos_lectura = 0;
int elementos= 0;         //Nos dice el nÃºmero de elementos

Semaphore ocupadas(0);  // cola donde espera el consumidor (n>0)
Semaphore libres(tam_vec);                //  cola donde espera el productor (n<num_celdas_total)

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 );
```

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

(a nosotros por suerte nos pasa)



WUOLAH

@Zukii on Wuolah

```
}

// ------------------------------------------------------------------
// funciÃ³n llamada por el consumidor para extraer un dato

int leer( )
{
  assert( 0 < elementos );
  elementos--;
  const int valor = buffer[pos_lectura];

  pos_lectura = (pos_lectura + 1) % tam_vec;

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

void escribir( int valor )
{
  //cout << "escribir: ocup == " << num_celdas_ocupadas << ", total == " << num_celdas_total
<< endl;
  assert( elementos < tam_vec );

  // hacer la operaciÃ³n de inserciÃ³n, actualizando estado del monitor
  buffer[pos_escritura] = valor;
  elementos++;

        pos_escritura = (pos_escritura + 1) % tam_vec;
}
```

```
//-------------------------------------------------------------
// FunciÃ³n que simula la acciÃ³n de producir un ingrediente, como un retardo
// aleatorio de la hebra (devuelve nÃºmero de ingrediente producido)

int producir_ingrediente()
{
  // calcular milisegundos aleatorios de duraciÃ³n de la acciÃ³n 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 ;
}

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

void funcion_hebra_estanquero( )
{
        int ingrediente;
```

@Zukii on Wuolah

```
        while(true){
                while ( elementos == 0)
                sem_wait(ocupadas);

                ingrediente = leer();

                // seÃ±alar al productor que hay un hueco libre, por si estÃ¡ esperando
                sem_signal(libres);

                sem_wait(mostrador);
                cout << "Se ha puesto en el mostrador el igrediente: " << ingrediente << endl;
                sem_signal(ingredientes[ingrediente]);
        }
}

//----------------------------------------------------------------------
// FunciÃ³n que simula la acciÃ³n de fumar, como un retardo aleatoria de la hebra

void fumar( int num_fumador )
{

  // calcular milisegundos aleatorios de duraciÃ³n de la acciÃ³n 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;

}

//----------------------------------------------------------------------
// funciÃ³n que ejecuta la hebra del fumador
void  funcion_hebra_fumador( int num_fumador )
{
  while( true )
  {
                sem_wait(ingredientes[num_fumador]);
                cout << "Se ha retirado del mostrador el ingrediente: " << num_fumador << endl;
                cout << "Num elementos en buffer: " << elementos << endl;
                sem_signal(mostrador);

                fumar(num_fumador);
  }
}

void funcion_hebra_suministradora(int num_hebra)
{
        while(true)
        {
                int ingrediente = producir_ingrediente();

                while ( elementos ==  tam_vec)
```

```
                sem_wait(libres);

                escribir(ingrediente);
                cout << "Se ha puesto en el BUFFER el ingrediente: " << ingrediente << endl;
                cout << "Num elementos en buffer: " << elementos << endl;


                // seÃ±alar al consumidor que ya hay una celda ocupada (por si esta
esperando)
                sem_signal(ocupadas);
        }
}


//----------------------------------------------------------------------

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

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

        for(int i = 0;  i < num_suministradoras ; i++)
                hebras_suministradoras[i]= thread(funcion_hebra_suministradora,i);

  hebra_estanquero.join();

        for(int i = 0; i < num_fumadores ; i++)
```

```
        hebras_fumadores[i].join();


    for(int i = 0;  i < num_suministradoras ; i++)
        hebras_suministradoras[i].join();
}
```

Enunciado

Modifica tu solución al problema de los Lectores-Escritores de la práctica 2, tal como se indica a continuación, adjuntando el archivo .cpp resultante con nombre ejercicio2.cpp:

a. Se lanzarán 4 hebras lectoras y 3 hebras escritoras.

b. Existirá una nueva hebra, denominada "revisora", que accederá periódicamente a la misma estructura de datos, al igual que lectores y escritores, usando operaciones similares para el acceso a dicha estructura, llamadas "ini_revison" y "fin_revision".

c. La hebra revisora solo podrá acceder a la estructura cuando ya haya un escritor escribiendo en la estructura. Por lo tanto, en esta versión, un escritor no tiene acceso exclusivo a la estructura, ya que la revisora podría estar accediendo a la estructura concurrentemente con dicho escritor.

d. La hebra revisora no podrá salir de su proceso de revisión mientras un escritor permanezca dentro. Por tanto, si la hebra revisora quiere salir y un escritor aún no ha salido, la revisora deberá esperar a que esté saliendo o haya salido dicho escritor.

e. Un lector podrá acceder a la estructura aunque la hebra revisora aún no haya salido de la estructura.

Solución:

```
#include <iostream>
#include <iomanip>
#include <cassert>
#include <thread>
#include <condition_variable>
#include <random>
```

@Zukii on Wuolah

```
#include <mutex>
#include "HoareMonitor.h"

using namespace std ;
using namespace HM;

const int num_lectores = 4;
const int num_escritores = 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 );
}

//----------------------------------------------------------------------
// FunciÃ³n que hace que una hebra haga una espera aleatoria@i

void EsperaAleatorio()
{
        chrono::milliseconds duracion( aleatorio<100,500>() );
        this_thread::sleep_for( duracion );
}

//----------------------------------------------------------------------
// FunciÃ³n que simula la acciÃ³n de escribir, como un retardo
// aleatorio de la hebra

void escribir(int escritor)
{
```

```
  // calcular milisegundos aleatorios de duraciÃ³n de la acciÃ³n de fumar)
  chrono::milliseconds duracion_escrib( aleatorio<10,100>() );

  // informa de que comienza a producir
  cout << "Se estÃ¡ escribiendo. Escritor: "<< escritor << "          " <<
          duracion_escrib.count() << " milisegundos)" << endl;

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

  // informa de que ha terminado de producir
  cout << "Escritor " << escritor << " ha terminado" << endl;
}

//----------------------------------------------------------------------
// FunciÃ³n que simula la acciÃ³n de leer, como un retardo aleatoria de la hebra

void leer(int lector)
{

  // calcular milisegundos aleatorios de duraciÃ³n de la acciÃ³n de fumar)
  chrono::milliseconds duracion_leer( aleatorio<20,200>() );

  // informa de que comienza a fumar

   cout << "Lector " << lector << "  :"
       << " empieza a leer (" << duracion_leer.count()<< " milisegundos)" << endl;

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

```
  // informa de que ha terminado de fumar

   cout << "Lector " << lector << "  : termina de leer." << endl;
}

void revisar()
{

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

   cout << "Revisor empieza a revisar (" << duracion_leer.count() << " milisegundos)" << endl;

  this_thread::sleep_for( duracion_leer );

   cout << "Revisor termina de revisar.(Espera para salir)" << endl;
}

//
**********************************************************************
//
**********************************************************************
//
**********************************************************************
//
**********************************************************************
// clase para monitor estanco, semÃ¡ntica SU.

class Lec_Esc : public HoareMonitor
{
  private:
          int       n_lec;
          bool escrib;
```

```
        bool revisando;

        CondVar lectura;
        CondVar escritura;
        CondVar revision;
        CondVar terminar_rev;

  public:
  Lec_Esc() ; // constructor
  void ini_lectura();
  void fin_lectura();
  void ini_escritura();
  void fin_escritura();
        void ini_revision();
  void fin_revision();

};

// -----------------------------------------------------------
Lec_Esc::Lec_Esc()
{
  n_lec = 0;
  escrib = false;

        terminar_rev = newCondVar();
        revision = newCondVar();
  lectura = newCondVar();
  escritura = newCondVar();
}
// -----------------------------------------------------------
void Lec_Esc :: ini_lectura()
```

@Zukii on Wuolah

```
{
        if (escrib)
                lectura.wait();

        n_lec++;

        lectura.signal();
}

// -----------------------------------------------------------------
void Lec_Esc:: fin_lectura()
{
        n_lec--;

        if (n_lec== 0 && !revisando)
                escritura.signal();
}

// -----------------------------------------------------------------
void Lec_Esc:: ini_escritura()
{
        if (n_lec> 0 || escrib || revisando)      //SI aun se sigue revisando, esperar a que
termine
                escritura.wait();

        escrib = true;

        if(!revisando)
                revision.signal();          //Llamar a un revisor si no se esta revision
}
```

@Zukii on Wuolah

```
//-------------------------------------------------------------
 void Lec_Esc :: fin_escritura()
{
        escrib = false;

        if(revisando)
                terminar_rev.signal(); //Signal para indicar que revision puede terminar

                                        //Si hay lectores, tienen prioridad sobre otro escritor
        if (lectura.get_nwt() != 0 || revisando)  //SI el numero de procesos esperando es != 0

        //Y aun se esta revisando
                lectura.signal();
        else
                escritura.signal();
}

// -------------------------------------------------------------
void Lec_Esc :: ini_revision()
{
        if(!escrib || n_lec > 0)   //Si hay lectores o no se esta escribiendo espera
                revision.wait(); //Espera mientras no haya un escritor

        revisando = true;
}

//-------------------------------------------------------------
 void Lec_Esc :: fin_revision()
{
        if(escrib)          //Espera mientras se escribe
                terminar_rev.wait();
```

```
        revisando = false;
}

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

void funcion_hebra_lector(MRef<Lec_Esc> monitor, int num_lector)
{
        while(true)
        {
                EsperaAleatorio();
                monitor->ini_lectura();
                leer(num_lector);
                monitor->fin_lectura();
        }
}

//-------------------------------------------------------------
void  funcion_hebra_escritor(MRef<Lec_Esc> monitor, int num_escritor)
{
  while( true )
  {
                EsperaAleatorio();
                monitor->ini_escritura();
                escribir(num_escritor);
                monitor->fin_escritura();
  }
}

void funcion_hebra_revisora(MRef<Lec_Esc> monitor)
```

```
{
        while(true)
        {
                EsperaAleatorio();
                monitor->ini_revision();
                revisar();
                monitor->fin_revision();
        }
}

// -----------------------------------------------------------------

int main()
{
  cout << "------------------------------------------------------------------" << endl
      << "Problema de los lectores y escritores. Monitor SU " << endl
      << "------------------------------------------------------------------" << endl
      << flush ;

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

  thread hebras_lectoras[num_lectores],
       hebras_escritoras[num_escritores];

  thread hebra_revisora(funcion_hebra_revisora, monitor);

        for(int i = 0; i < num_lectores ; i++)
                hebras_lectoras[i] = thread(funcion_hebra_lector, monitor, i);

        for(int i = 0; i < num_escritores ; i++)
                hebras_escritoras[i] = thread(funcion_hebra_escritor, monitor, i);
```

@Zukii on Wuolah

```
        for(int i = 0; i < num_lectores ; i++)
                hebras_lectoras[i].join();

        for(int i = 0; i < num_escritores ; i++)
                hebras_escritoras[i].join();

        hebra_revisora.join();#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_lectores = 4;
const int num_escritores = 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 );
}

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

@Zukii on Wuolah

```
// FunciÃ³n que hace que una hebra haga una espera aleatoria@i

void EsperaAleatorio()
{
        chrono::milliseconds duracion( aleatorio<100,500>() );
        this_thread::sleep_for( duracion );
}

//----------------------------------------------------------------------
// FunciÃ³n que simula la acciÃ³n de escribir, como un retardo
// aleatorio de la hebra

void escribir(int escritor)
{
  // calcular milisegundos aleatorios de duraciÃ³n de la acciÃ³n de fumar)
  chrono::milliseconds duracion_escrib( aleatorio<10,100>() );

  // informa de que comienza a producir
  cout << "Se estÃ¡ escribiendo. Escritor: "<< escritor << "          " <<
        duracion_escrib.count() << " milisegundos)" << endl;

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

  // informa de que ha terminado de producir
  cout << "Escritor " << escritor << " ha terminado" << endl;
}

//----------------------------------------------------------------------
// FunciÃ³n que simula la acciÃ³n de leer, como un retardo aleatoria de la hebra
```

@Zukii on Wuolah

```
void leer(int lector)
{

  // calcular milisegundos aleatorios de duraciÃ³n de la acciÃ³n de fumar)
  chrono::milliseconds duracion_leer( aleatorio<20,200>() );

  // informa de que comienza a fumar

   cout << "Lector " << lector << "  :"
        << " empieza a leer (" << duracion_leer.count()<< " milisegundos)" << endl;

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

  // informa de que ha terminado de fumar

   cout << "Lector " << lector << "  : termina de leer." << endl;
}

void revisar()
{

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

   cout << "Revisor empieza a revisar (" << duracion_leer.count()<< " milisegundos)" << endl;

  this_thread::sleep_for( duracion_leer );

   cout << "Revisor termina de revisar.(Espera para salir)" << endl;
}
```

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

//
*********************************************************************************

//
*********************************************************************************

//
*********************************************************************************

// clase para monitor estanco, semÃ¡ntica SU.


class Lec_Esc : public HoareMonitor

{

  private:

        int       n_lec;

        bool escrib;

        bool revisando;


        CondVar lectura;

        CondVar escritura;

        CondVar revision;

        CondVar terminar_rev;


  public:

  Lec_Esc() ; // constructor

  void ini_lectura();

  void fin_lectura();

  void ini_escritura();

  void fin_escritura();

        void ini_revision();

  void fin_revision();


};
```

@Zukii on Wuolah

```
// ---------------------------------------------------------------------
Lec_Esc::Lec_Esc()
{
  n_lec = 0;
  escrib = false;

        terminar_rev = newCondVar();
        revision = newCondVar();
  lectura = newCondVar();
  escritura = newCondVar();
}
// ---------------------------------------------------------------------
void Lec_Esc :: ini_lectura()
{
        if (escrib)
                lectura.wait();

        n_lec++;

        lectura.signal();
}

// ---------------------------------------------------------------------
void Lec_Esc :: fin_lectura()
{
        n_lec--;

        if (n_lec == 0 && !revisando)
                escritura.signal();
}
```

```
// -------------------------------------------------------------------
void Lec_Esc :: ini_escritura()
{
	if (n_lec > 0 || escrib || revisando)	//SI aun se sigue revisando, esperar a que termine
		escritura.wait();

	escrib = true;

	if(!revisando)
		revision.signal();		//Llamar a un revisor si no se esta revision
}

//-------------------------------------------------------------------
 void Lec_Esc :: fin_escritura()
{
	escrib = false;

	if(revisando)
		terminar_rev.signal(); //Signal para indicar que revision puede terminar

					//Si hay lectores, tienen prioridad sobre otro escritor
	if (lectura.get_nwt() != 0 || revisando)  //SI el numero de procesos esperando es != 0

	//Y aun se esta revisando
		lectura.signal();
	else
		escritura.signal();
}

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

```
void Lec_Esc :: ini_revision()
{
	if(!escrib || n_lec > 0)    //Si hay lectores o no se esta escribiendo espera
		revision.wait(); //Espera mientras no haya un escritor

	revisando = true;
}

//-------------------------------------------------------------
 void Lec_Esc :: fin_revision()
{
	if(escrib)          //Espera mientras se escribe
		terminar_rev.wait();

	revisando = false;
}

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

void funcion_hebra_lector(MRef<Lec_Esc> monitor, int num_lector)
{
	while(true)
	{
		EsperaAleatorio();
		monitor->ini_lectura();
		leer(num_lector);
		monitor->fin_lectura();
	}
}
```

@Zukii on Wuolah

```
//-------------------------------------------------------------
void  funcion_hebra_escritor(MRef<Lec_Esc> monitor, int num_escritor)
{
  while( true )
  {
                EsperaAleatorio();
                monitor->ini_escritura();
                escribir(num_escritor);
                monitor->fin_escritura();
  }
}

void funcion_hebra_revisora(MRef<Lec_Esc> monitor)
{
        while(true)
        {
                EsperaAleatorio();
                monitor->ini_revision();
                revisar();
                monitor->fin_revision();
        }
}

// ---------------------------------------------------------------

int main()
{
  cout << "------------------------------------------------------------------" << endl
      << "Problema de los lectores y escritores. Monitor SU " << endl
      << "------------------------------------------------------------------" << endl
      << flush ;
```

@Zukii on Wuolah

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

thread hebras_lectoras[num_lectores],
    hebras_escritoras[num_escritores];

thread hebra_revisora(funcion_hebra_revisora, monitor);

    for(int i = 0; i < num_lectores ; i++)
        hebras_lectoras[i] = thread(funcion_hebra_lector, monitor, i);

    for(int i = 0; i < num_escritores ; i++)
        hebras_escritoras[i] = thread(funcion_hebra_escritor, monitor, i);


    for(int i = 0; i < num_lectores ; i++)
        hebras_lectoras[i].join();

    for(int i = 0; i < num_escritores ; i++)
        hebras_escritoras[i].join();

    hebra_revisora.join();

}
}
```

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.