portafolios Práctica 2 Scd

Monitores

Víctor José Rubia López

UGR ETSIIT

2ºB (B2)

[El problema de los fumadores, semántica SU 2](#_Toc56105482)

[1. Pseudocódigo 2](file:///C:\Users\victo\Google%20Drive\01_SCD\Practicas\P2\Portafolio.docx#_Toc56105483)

[2. Código completo de la solución adoptada 3](#_Toc56105484)

[3. Salida del programa 5](#_Toc56105485)

[Productor-Consumidor (múltiples productores y consumidores) LIFO, semántica SU. 7](#_Toc56105486)

[4. Pseudocódigo 7](#_Toc56105487)

[5. Código completo de la solución adoptada 8](#_Toc56105488)

[6. Salida del programa 12](#_Toc56105489)

[Productor-Consumidor (múltiples productores y consumidores) FIFO, semántica SU. 14](#_Toc56105490)

[7. Pseudocódigo 14](#_Toc56105491)

[8. Código completo de la solución adoptada 15](#_Toc56105492)

[9. Salida del programa 19](#_Toc56105493)

[Lectores-Escritores, semántica SU. 21](#_Toc56105494)

[10. Pseudocódigo 21](#_Toc56105495)

[11. Código completo de la solución adoptada 22](#_Toc56105496)

[12. Salida del programa 25](#_Toc56105497)

El problema de los fumadores, semántica SU

* **Variable o variables** **permanentes**:
  + **Ingredientes**: tipo integer, puede tomar valores 0..2, se utiliza para indicar qué ingrediente se ha producido en el estanco.
* **Cola o colas condición**:
  + **Fumador[i]:** la condición de espera se produce cuando el ingrediente que se encuentra en el mostrador no es el que el fumador *i* necesita.
  + **Mostrador:** la condición de espera se produce cuando aún no se ha producido un ingrediente.

monitor Estanco ;

var fumador[i]  : condition ; { qué fumadores se encuentran fumando }

    mostrador   : condition ; { ocupación del mostrador }

    ingrediente : integer   ; { ingrediente que se produce }

process obtenerIngredienteFumador[ i : 0..2 ] ;

begin

    if i == ingrediente then { si el fumador quiere el ingrediente que el estanco ha producido}

        mostrador.signal ; { se pone el mostrador a disponible }

        fumador[i].wait() ; { se hace una espera }

        ingrediente = -1 ; { se quita el ingrediente del mostrador }

    end if

    else then

        fumador[i].wait() ; { se espera el fumador en la calle }

    end else

end

process ponerIngredientesMostrador[ i : 0..2 ];

begin

    ingrediente = i ; {si el ingrediente es igual al de i}

    fumador[ingrediente].signal() ; { el fumador que quiere el ingrediente i se le da paso al mostrador }

end

proces esperarIngredienteMostrador

begin

    if ingrediente != -1 then

        mostrador.wait() ;

    end if

end

. Pseudocódigo

. Código completo de la solución adoptada

|  |
| --- |
| #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"  using namespace std ;  using namespace HM ;  const int num\_fumadores = 3;  mutex mtx;  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // 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 );  }  class Estanco : public HoareMonitor{     private:        CondVar fumador[num\_fumadores], mostrador;        int ingrediente;     public:        Estanco(){           ingrediente=-1;           mostrador= newCondVar();           for(int i = 0; i < num\_fumadores; i++)              fumador[i]= newCondVar();        }        void obtenerIngredienteFumador(int num\_fumador){           if(num\_fumador==ingrediente){              mostrador.signal();              fumador[num\_fumador].wait();              ingrediente=-1;           }           else{              fumador[num\_fumador].wait();           }        }        void ponerIngredienteMostrador(int ingr){           ingrediente=ingr;           fumador[ingrediente].signal();        }        void esperarIngredienteMostrador(){           if(ingrediente!=-1)              mostrador.wait();        }  };  //-------------------------------------------------------------------------  // 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     mtx.lock();     cout << "Estanquero : empieza a producir ingrediente (" << duracion\_produ.count() << " milisegundos)" << endl;     mtx.unlock();     // 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     mtx.lock();     cout << "Estanquero : termina de producir ingrediente " << num\_ingrediente << endl;     mtx.unlock();     return num\_ingrediente ;  }  //----------------------------------------------------------------------  // función que ejecuta la hebra del estanquero  void funcion\_hebra\_estanquero( MRef<Estanco> monitor )  {     while(true){        const int ingr = producir\_ingrediente();        monitor->ponerIngredienteMostrador(ingr);        monitor->esperarIngredienteMostrador();     }  }  //-------------------------------------------------------------------------  // 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     mtx.lock();      cout << "Fumador " << num\_fumador << "  :"            << " empieza a fumar (" << duracion\_fumar.count() << " milisegundos)" << endl;     mtx.unlock();     // espera bloqueada un tiempo igual a ''duracion\_fumar' milisegundos     this\_thread::sleep\_for( duracion\_fumar );     // informa de que ha terminado de fumar     mtx.lock();      cout << "Fumador " << num\_fumador << "  : termina de fumar, comienza espera de ingrediente." << endl;     mtx.unlock();  }  //----------------------------------------------------------------------  // función que ejecuta la hebra del fumador  void  funcion\_hebra\_fumador( MRef<Estanco> monitor, int num\_fumador )  {     while( true )     {        monitor->obtenerIngredienteFumador(num\_fumador);        fumar(num\_fumador);     }  }  //----------------------------------------------------------------------  int main()  {     MRef<Estanco> monitorEstanco = Create<Estanco>();     thread hebra\_fumador[num\_fumadores];     thread hebra\_estanquero(funcion\_hebra\_estanquero, monitorEstanco);     for(int i=0; i < num\_fumadores; i++)        hebra\_fumador[i] = thread(funcion\_hebra\_fumador, monitorEstanco, i);     hebra\_estanquero.join();     for(int i=0; i < num\_fumadores; i++)        hebra\_fumador[i].join();  } |

. Salida del programa

|  |
| --- |
| Estanquero : empieza a producir ingrediente (34 milisegundos)  Estanquero : termina de producir ingrediente 2  Fumador 2 : empieza a fumar (113 milisegundos)  Fumador 2 : termina de fumar, comienza espera de ingrediente.  Estanquero : empieza a producir ingrediente (15 milisegundos)  Estanquero : termina de producir ingrediente 1  Fumador 1 : empieza a fumar (96 milisegundos)  Fumador 1 : termina de fumar, comienza espera de ingrediente.  Estanquero : empieza a producir ingrediente (35 milisegundos)  Estanquero : termina de producir ingrediente 1  Fumador 1 : empieza a fumar (161 milisegundos)  Estanquero : empieza a producir ingrediente (50 milisegundos)  Estanquero : termina de producir ingrediente 2  Fumador 2 : empieza a fumar (127 milisegundos)  Estanquero : empieza a producir ingrediente (64 milisegundos)  Estanquero : termina de producir ingrediente 1  Fumador 1 : termina de fumar, comienza espera de ingrediente.  Estanquero : empieza a producir ingrediente (91 milisegundos)  Fumador 2 : termina de fumar, comienza espera de ingrediente.  Estanquero : termina de producir ingrediente 1  Fumador 1 : empieza a fumar (110 milisegundos)  Estanquero : empieza a producir ingrediente (85 milisegundos)  Estanquero : termina de producir ingrediente 0  Fumador 0 : empieza a fumar (162 milisegundos)  Fumador 1 : termina de fumar, comienza espera de ingrediente.  Fumador 0 : termina de fumar, comienza espera de ingrediente.  Estanquero : empieza a producir ingrediente (32 milisegundos)  Estanquero : termina de producir ingrediente 1  Fumador 1 : empieza a fumar (80 milisegundos)  Fumador 1 : termina de fumar, comienza espera de ingrediente.  Estanquero : empieza a producir ingrediente (84 milisegundos)  Estanquero : termina de producir ingrediente 1  Fumador 1 : empieza a fumar (195 milisegundos)  Estanquero : empieza a producir ingrediente (78 milisegundos)  Estanquero : termina de producir ingrediente 0  Fumador 0 : empieza a fumar (145 milisegundos)  Estanquero : empieza a producir ingrediente (35 milisegundos)  Estanquero : termina de producir ingrediente 1  Fumador 1 : termina de fumar, comienza espera de ingrediente.  Estanquero : empieza a producir ingrediente (35 milisegundos)  Fumador 0 : termina de fumar, comienza espera de ingrediente.  Estanquero : termina de producir ingrediente 0  Fumador 0 : empieza a fumar (116 milisegundos)  Fumador 0 : termina de fumar, comienza espera de ingrediente.  Estanquero : empieza a producir ingrediente (73 milisegundos)  Estanquero : termina de producir ingrediente 0  Fumador 0 : empieza a fumar (22 milisegundos)  Estanquero : empieza a producir ingrediente (22 milisegundos)  Fumador 0 : termina de fumar, comienza espera de ingrediente.  Estanquero : termina de producir ingrediente 2  Fumador 2 : empieza a fumar (186 milisegundos)  Fumador 2 : termina de fumar, comienza espera de ingrediente.  Estanquero : empieza a producir ingrediente (69 milisegundos)  Estanquero : termina de producir ingrediente 2  Fumador 2 : empieza a fumar (124 milisegundos)  Estanquero : empieza a producir ingrediente (76 milisegundos)  Estanquero : termina de producir ingrediente 2  Fumador 2 : termina de fumar, comienza espera de ingrediente.  Estanquero : empieza a producir ingrediente (61 milisegundos)  Estanquero : termina de producir ingrediente 0  Fumador 0 : empieza a fumar (43 milisegundos)  Fumador 0 : termina de fumar, comienza espera de ingrediente.  Estanquero : empieza a producir ingrediente (63 milisegundos)  Estanquero : termina de producir ingrediente 0  Fumador 0 : empieza a fumar (173 milisegundos)  Estanquero : empieza a producir ingrediente (20 milisegundos)  Estanquero : termina de producir ingrediente 0  Fumador 0 : termina de fumar, comienza espera de ingrediente. |

Productor-Consumidor (múltiples productores y consumidores) LIFO, semántica SU.

* **Variable o variables** **permanentes**: para cada una describe el tipo, nombre, valores posibles y significado de la variable.
  + **Buffer[10]**: vector de tipo integer que se utiliza para que almacene los datos insertados pendientes de extraer. El tamaño viene definido por otra variable que llamaré *ntotal\_celdas*.
  + **Primera\_libre**: variable de tipo integer que se usa para tener el control de la ocupación del vector de buffer y para poder acceder a él.
* **Cola o colas condición**: para cada una, escribe el nombre y la condición de espera asociada (una expresión lógica de las variables permanentes).
  + **Ocupadas:** representa la cola de espera de las ocupadas. Su condición es n > 0). La hebra consumidora espera cuando n=0.
  + **Libres:** representa la cola de espera de las libres (hasta n < 10 en mi caso). La hebra productora espera cuando n = 10.

. Pseudocódigo

|  |
| --- |
| monitor ProdConsSULIFO;  var ntotal\_celdas = 10    : integer ;      buffer[ntotal\_celdas] : array[0..num\_celdas\_total] of integer ;      primera\_libre            : integer ;      ocupadas                 : condition ;      libres                   : condition ;  process leer ;  begin      while primera\_libre == 0 then          ocupadas.wait() ;      end      assert(0 < primera\_libre) ;      var valor = buffer[primera\_libre - 1] : integer ;      primera\_libre-- ;      libres.signal() ;      return valor  end  process escribir ;  begin      while primera\_libre == num\_celdas\_total then          libres.wait() ;      assert(primera\_libre < num\_celdas\_total);      buffer[primera\_libre] = valor ;      primera\_libre++ ;      ocupadas.signal() ;  end |

. Código completo de la solución adoptada

|  |
| --- |
| #include <iostream>  #include <iomanip>  #include <cassert>  #include <thread>  #include <random>  #include "HoareMonitor.h"  using namespace std ;  using namespace HM ;  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // variables compartidas  constexpr int num\_items  = 40, productores = 4, consumidores = 4;  mutex mtx ;  unsigned cont\_prod[num\_items] = {0},           cont\_cons[num\_items] = {0},           producidos[productores];  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // 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 );  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // funciones comunes a las dos soluciones (fifo y lifo)  //----------------------------------------------------------------------  int producir\_dato( int i)  {        static int contador = 0 ;        this\_thread::sleep\_for( chrono::milliseconds( aleatorio<20,100>() ));        mtx.lock();        cout << "producido: " << contador << endl << flush ;        producidos[i]++;        mtx.unlock();        cont\_prod[contador] ++ ;        return contador++;  }  //-----------------------------------------------------------------------------  void consumir\_dato( unsigned dato )  {     assert( dato < num\_items );     cont\_cons[dato] ++ ;     this\_thread::sleep\_for( chrono::milliseconds( aleatorio<20,100>() ));     mtx.lock();     cout << "                  consumido: " << dato << endl ;     mtx.unlock();  }  //-----------------------------------------------------------------------------  void test\_contadores()  {     bool ok = true ;     cout << "comprobando contadores ...." << flush ;     for( unsigned i = 0 ; i < num\_items ; i++ )     {        if ( cont\_prod[i] != 1 )        {           cout << "error: valor " << i << " producido " << cont\_prod[i] << " veces." << endl ;           ok = false ;        }        if ( cont\_cons[i] != 1 )        {           cout << "error: valor " << i << " consumido " << cont\_cons[i] << " veces" << endl ;           ok = false ;        }     }     if (ok)        cout << endl << flush << "solución (aparentemente) correcta." << endl << flush ;  }  //-----------------------------------------------------------------------------  class ProdConsSULIFO : public HoareMonitor  {  private:     static const int ntotal\_celdas = 10 ;     int buffer[ntotal\_celdas], primera\_libre ;     CondVar ocupadas, libres ;  public:     ProdConsSULIFO(){        primera\_libre = 0 ;        ocupadas = newCondVar();        libres = newCondVar();     }     int  leer(){        while (primera\_libre == 0)           ocupadas.wait();        assert(0 < primera\_libre);        const int valor = buffer[primera\_libre - 1];        primera\_libre--;        libres.signal();        return valor;     }     void escribir(int valor){        while (primera\_libre == ntotal\_celdas)           libres.wait();        assert(primera\_libre < ntotal\_celdas);        buffer[primera\_libre] = valor;        primera\_libre++;        ocupadas.signal();     }  } ;  //-----------------------------------------------------------------------------  void funcion\_hebra\_productora( MRef<ProdConsSULIFO> monitor, int num\_hebra )  {     for( unsigned i = 0 ; i < num\_items/productores ; i++ )     {        int valor = producir\_dato(num\_hebra) ;        monitor->escribir( valor );     }  }  //-----------------------------------------------------------------------------  void funcion\_hebra\_consumidora( MRef<ProdConsSULIFO> monitor, int num\_hebra )  {     for( unsigned i = 0 ; i < num\_items/consumidores ; i++ )     {      int valor = monitor->leer();      consumir\_dato( valor );     }  }  //-----------------------------------------------------------------------------  int main()  {     cout << "-------------------------------------------------------------------------------" << endl          << " Productores consumidores con  (" << productores          << " productores y " << consumidores << " consumidores, versión SU y LIFO). " << endl          << "-------------------------------------------------------------------------------" << endl          << flush ;     MRef<ProdConsSULIFO> monitor = Create<ProdConsSULIFO>( );     thread hebraprod[productores], hebracons[consumidores];     for(int k = 0; k < productores; k++){        producidos[k] = 0;     }     for(int i = 0; i < productores; i++){        hebraprod[i] = thread ( funcion\_hebra\_productora, monitor, i );     }     for(int j = 0; j < consumidores; j++){        hebracons[j] = thread ( funcion\_hebra\_consumidora, monitor, j );     }     for(int i = 0; i < productores; i++){        hebraprod[i].join();     }     for(int j = 0; j < consumidores; j++){        hebracons[j].join();     }     for(int l = 0; l < productores; l++){        cout << "Hebra " <<  l << " ha producido " << producidos[l] << " valores" << endl;     }     test\_contadores() ;  } |

. Salida del programa

|  |
| --- |
| -------------------------------------------------------------------------------  Productores consumidores con (4 productores y 4 consumidores, versión SU y LIFO).  -------------------------------------------------------------------------------  producido: 0  producido: 1  producido: 2  consumido: 1  producido: 3  producido: 4  consumido: 2  consumido: 0  producido: 5  producido: 6  producido: 7  consumido: 4  consumido: 3  producido: 8  consumido: 5  producido: 9  consumido: 7  consumido: 9  consumido: 6  consumido: 8  producido: 10  producido: 11  producido: 12  producido: 13  producido: 14  producido: 15  producido: 16  producido: 17  consumido: 11  consumido: 10  producido: 18  consumido: 12  producido: 19  producido: 20  consumido: 13  producido: 21  consumido: 17  producido: 22  producido: 23  consumido: 16  consumido: 18  producido: 24  producido: 25  consumido: 20  consumido: 21  producido: 26  producido: 27  consumido: 23  producido: 28  consumido: 24  producido: 29  consumido: 25  consumido: 22  producido: 30  producido: 31  consumido: 29  producido: 32  consumido: 27  consumido: 28  producido: 33  consumido: 26  producido: 34  producido: 35  consumido: 30  consumido: 31  consumido: 33  producido: 36  producido: 37  consumido: 32  consumido: 19  consumido: 34  consumido: 35  consumido: 36  consumido: 15  producido: 38  producido: 39  consumido: 37  consumido: 14  consumido: 38  consumido: 39  Hebra 0 ha producido 10 valores  Hebra 1 ha producido 10 valores  Hebra 2 ha producido 10 valores  Hebra 3 ha producido 10 valores  comprobando contadores ....  solución (aparentemente) correcta. |

Productor-Consumidor (múltiples productores y consumidores) FIFO, semántica SU.

* **Variable o variables** **permanentes**: para cada una describe el tipo, nombre, valores posibles y significado de la variable.
  + **Buffer[10]**: vector de tipo integer que se utiliza para que almacene los datos insertados pendientes de extraer. El tamaño viene definido por otra variable que llamaré *ntotal\_celdas*.
  + **Primera\_libre y primera\_ocupada**: variables de tipo integer que se usan para tener acceso al buffer (usamos la variable **nceldas\_ocupadas**, de tipo entero, para almacenar el número de celdas ocupadas).
* **Cola o colas condición**: para cada una, escribe el nombre y la condición de espera asociada (una expresión lógica de las variables permanentes).
  + **Ocupadas:** representa la cola de espera de las ocupadas. Su condición es n > 0). La hebra consumidora espera cuando n=0.
  + **Libres:** representa la cola de espera de las libres (hasta n < 10 en mi caso). La hebra productora espera cuando n = 10.

. Pseudocódigo

|  |
| --- |
| monitor ProdConsSUFIFO;  var ntotal\_celdas = 10       : integer ;      buffer[ntotal\_celdas]    : integer ;      primera\_libre            : integer ;      primera\_ocupada          : integer ;      nceldas\_ocupadas         : integer ;      ocupadas                 : condition ;      libres                   : condition ;  process leer ;  begin      while n\_celdasocupadas == 0 then          ocupadas.wait() ;      end      assert(0 < n\_celdasocupadas) ;      var valor = buffer[primera\_ocupada] : integer ;      primera\_ocupada = (primera\_ocupada + 1) % ntotal\_celdas ;      n\_celdasocupadas-- ;      libres.signal() ;      return valor  end  process escribir ;  begin      while n\_celdasocupadas == ntotal\_celdas then          libres.wait() ;      assert(n\_celdasocupadas < ntotal\_celdas);      buffer[primera\_libre] = valor ;      primera\_libre = (primera\_libre + 1) % ntotal\_celdas ;      n\_celdasocupadas++ ;      ocupadas.signal() ;  end |

. Código completo de la solución adoptada

|  |
| --- |
| #include <iostream>  #include <iomanip>  #include <cassert>  #include <thread>  #include <random>  #include "HoareMonitor.h"  using namespace std ;  using namespace HM ;  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // variables compartidas  constexpr int num\_items  = 40, productores = 4, consumidores = 2;  mutex mtx ;  unsigned cont\_prod[num\_items] = {0},           cont\_cons[num\_items] = {0},           producidos[productores];  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // 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 );  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // funciones comunes a las dos soluciones (fifo y lifo)  //----------------------------------------------------------------------  int producir\_dato( int i)  {     if(producidos[i] < (num\_items / productores)){        static int contador = 0 ;        this\_thread::sleep\_for( chrono::milliseconds( aleatorio<20,100>() ));        mtx.lock();        cout << "producido: " << contador << endl << flush ;        producidos[i]++;        mtx.unlock();        cont\_prod[contador] ++ ;        return contador++;     }  }  //-----------------------------------------------------------------------------  void consumir\_dato( unsigned dato )  {     if ( num\_items <= dato )     {        cout << " dato === " << dato << ", num\_items == " << num\_items << endl ;        assert( dato < num\_items );     }     cont\_cons[dato] ++ ;     this\_thread::sleep\_for( chrono::milliseconds( aleatorio<20,100>() ));     mtx.lock();     cout << "                  consumido: " << dato << endl ;     mtx.unlock();  }  //-----------------------------------------------------------------------------  void test\_contadores()  {     bool ok = true ;     cout << "comprobando contadores ...." << flush ;     for( unsigned i = 0 ; i < num\_items ; i++ )     {        if ( cont\_prod[i] != 1 )        {           cout << "error: valor " << i << " producido " << cont\_prod[i] << " veces." << endl ;           ok = false ;        }        if ( cont\_cons[i] != 1 )        {           cout << "error: valor " << i << " consumido " << cont\_cons[i] << " veces" << endl ;           ok = false ;        }     }     if (ok)        cout << endl << flush << "solución (aparentemente) correcta." << endl << flush ;  }  //-----------------------------------------------------------------------------  class ProdConsSUFIFO : public HoareMonitor  {  private:     static const int ntotal\_celdas = 10 ;     int buffer[ntotal\_celdas], primera\_libre, primera\_ocupada, n\_celdasocupadas ;     CondVar ocupadas, libres ;  public:     ProdConsSUFIFO(){        primera\_libre = 0;        primera\_ocupada = 0;        n\_celdasocupadas = 0;        ocupadas = newCondVar();        libres = newCondVar();     }     int leer(){        while (n\_celdasocupadas == 0)           ocupadas.wait();        assert(0 < n\_celdasocupadas);        const int valor = buffer[primera\_ocupada];        primera\_ocupada = (primera\_ocupada + 1) % ntotal\_celdas;        n\_celdasocupadas--;        libres.signal();        return valor;     }     void escribir(int valor){        while (n\_celdasocupadas == ntotal\_celdas)           libres.wait();        assert(n\_celdasocupadas < ntotal\_celdas);        buffer[primera\_libre] = valor;        primera\_libre = (primera\_libre + 1) % ntotal\_celdas;        n\_celdasocupadas++;        ocupadas.signal();     }  } ;  //-----------------------------------------------------------------------------  void funcion\_hebra\_productora( MRef<ProdConsSUFIFO> monitor, int num\_hebra )  {     for( unsigned i = 0 ; i < num\_items/productores ; i++ )     {        int valor = producir\_dato(num\_hebra) ;        monitor->escribir( valor );     }  }  //-----------------------------------------------------------------------------  void funcion\_hebra\_consumidora( MRef<ProdConsSUFIFO> monitor, int num\_hebra )  {      for( unsigned i = 0 ; i < num\_items/consumidores ; i++ )     {      int valor = monitor->leer();      consumir\_dato( valor );     }  }  //-----------------------------------------------------------------------------  int main()  {     cout << "-------------------------------------------------------------------------------" << endl          << " Productores consumidores con  (" << productores          << " productores y " << consumidores << " consumidores, versión SU y FIFO). " << endl          << "-------------------------------------------------------------------------------" << endl          << flush ;     MRef<ProdConsSUFIFO> monitor = Create<ProdConsSUFIFO>( );     thread hebraprod[productores], hebracons[consumidores];     for(int k = 0; k < productores; k++){        producidos[k] = 0;     }     for(int i = 0; i < productores; i++){        hebraprod[i] = thread ( funcion\_hebra\_productora, monitor, i );     }     for(int j = 0; j < consumidores; j++){        hebracons[j] = thread ( funcion\_hebra\_consumidora, monitor, j );     }     for(int i = 0; i < productores; i++){        hebraprod[i].join();     }     for(int j = 0; j < consumidores; j++){        hebracons[j].join();     }     for(int l = 0; l < productores; l++){        cout << "Hebra " <<  l << " ha producido " << producidos[l] << " valores" << endl;     }     test\_contadores() ;  } |

. Salida del programa

|  |
| --- |
| -------------------------------------------------------------------------------  Productores consumidores con (4 productores y 2 consumidores, versión SU y FIFO).  -------------------------------------------------------------------------------  producido: 0  producido: 1  producido: 2  producido: 3  producido: 4  producido: 5  producido: 6  consumido: 1  producido: 7  producido: 8  consumido: 0  producido: 9  producido: 10  consumido: 3  producido: 11  consumido: 2  producido: 12  producido: 13  consumido: 5  producido: 14  consumido: 4  producido: 15  consumido: 6  producido: 16  producido: 17  consumido: 7  producido: 18  producido: 19  producido: 20  producido: 21  producido: 22  consumido: 8  producido: 23  consumido: 9  consumido: 10  consumido: 11  producido: 24  producido: 25  consumido: 13  consumido: 12  producido: 26  producido: 27  consumido: 15  producido: 28  producido: 29  consumido: 14  consumido: 16  producido: 30  producido: 31  producido: 32  consumido: 17  consumido: 18  consumido: 19  producido: 33  producido: 34  producido: 35  consumido: 20  consumido: 22  consumido: 21  consumido: 23  producido: 36  consumido: 24  consumido: 25  producido: 37  producido: 38  consumido: 27  consumido: 26  consumido: 29  consumido: 28  consumido: 30  producido: 39  consumido: 31  consumido: 33  consumido: 32  consumido: 34  consumido: 35  consumido: 37  consumido: 36  consumido: 38  consumido: 39  Hebra 0 ha producido 10 valores  Hebra 1 ha producido 10 valores  Hebra 2 ha producido 10 valores  Hebra 3 ha producido 10 valores  comprobando contadores ....  solución (aparentemente) correcta. |

Lectores-Escritores, semántica SU.

* **Variable o variables** **permanentes**:
  + **Num\_lec**: vector de tipo integer que nos indica el número de lectores que están escribiendo en un momento dado (inicialmente a 0)
  + **escribiendo**: variable de tipo lógica (booleana) que se usa para tener el control de si un escritor está escribiendo (true) o si no hay escritores escribiendo (false).
* **Cola o colas condición**:
  + **lectura:** usada por los lectores para esperar cuando ya no hay un escritor escribiendo (cuando escribiendo==true).
  + **escritura:** usada por los escritores para esperar cuando ya no hay otro escritor escribiendo (cuando escribiendo==true) o bien cuando hay lectores leyendo (num\_lec > 0).

. Pseudocódigo

|  |
| --- |
| monitor LectoresEscritores;  var num\_lec                  : integer ; {nº de lectores leyendo}      escribiendo              : boolean ; { true si hay algún escritor escribiendo}      lectura                  : condition ; { no hay escrit. escribiendo, se puede leer }      escritura                : condition ; { no hay lect. ni escrit., se puede escribir }  process iniLec ;  begin      if escribiendo then { si hay escritor: }          lectura.wait() ; { esperar }      end if      num\_lec++ ; { registrar un lector más }      lectura.signal() ; { desbloqueo en cadena de posibles lectores bloqueados }  end  process finLec ;  begin      num\_lec-- ; { registrar un lector menos }      if num\_lec == 0 then { si es el último lector }          escritura.signal () ; { desbloqueamos un escritor }      end if  end  process iniEsc ;  begin      if num\_lec > 0 OR escribiendo then { si hay otros, esperamos }          escritura.wait() ;      end if      escribiendo = true ; { registramos que hay un escritor }  end  process finEsc ;  begin      escribiendo = false ; { registramos que ya no hay escritor }      if lectura.get\_nwt() != 0 then { si el número de procesos esperando es dto de 0}          lectura.signal() ; { si no hay lectores, despertamos uno }      end if      else then          escritura.signal() ; {si no hay, despertamos un escritor }      end else  end |

. Código completo de la solución adoptada

|  |
| --- |
| #include <iostream>  #include <iomanip>  #include <cassert>  #include <thread>  #include <mutex>  #include <condition\_variable>  #include <chrono> // duraciones (duration), unidades de tiempo  #include <random> // dispositivos, generadores y distribuciones aleatorias  #include "HoareMonitor.h"  using namespace std;  using namespace HM;  const int num\_lectores = 3;  const int num\_escritores = 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 );  }  void escribir(int escritor){      chrono::milliseconds dur\_escritura(aleatorio<20,200>());      //  Se empieza a escribir      cout << "El escritor " << escritor << " comienza a escribir ( " << dur\_escritura.count() << " milisegundos)" << endl;      //  Espera bloqueada de dur\_escritura milisegundos      this\_thread::sleep\_for(dur\_escritura);      //  Informa que ha terminado de escribir      cout << "El escritor " << escritor << " ha terminado de escribir" << endl;  }  void Espera(){      chrono::milliseconds tiempo(aleatorio<20,200>());      this\_thread::sleep\_for(tiempo);  }  void leer(int lector){      chrono::milliseconds dur\_lectura(aleatorio<20,200>());      //  Se empieza a leer      cout << "El lector " << lector << " comienza a leer ( " << dur\_lectura.count() << " milisegundos)" << endl;      //  Espera bloqueada de dur\_lectura milisegundos      this\_thread::sleep\_for(dur\_lectura);      //  Informa que ha terminado de leer      cout << "El lector " << lector << " ha terminado de leer" << endl;  }  class LectoresEscritores: public HoareMonitor{      private:          int num\_lec;          bool escribiendo;          CondVar lectura, escritura;      public:          LectoresEscritores(){              num\_lec = 0;              escribiendo = false;              lectura = newCondVar();              escritura = newCondVar();          }          void iniLec(){              if(escribiendo)                  lectura.wait();              num\_lec++;              lectura.signal();          }          void finLec(){              num\_lec--;              if(num\_lec == 0)                  escritura.signal();          }          void iniEsc(){              if(num\_lec > 0 || escribiendo)                  escritura.wait();              escribiendo = true;          }          void finEsc(){              escribiendo = false;              if(lectura.get\_nwt() != 0)                  lectura.signal();              else                  escritura.signal();          }  };  void funcion\_hebra\_lector(MRef<LectoresEscritores> monitor, int numLectores){      while(true){          Espera();          monitor->iniLec();          leer(numLectores);          monitor->finLec();      }  }  void funcion\_hebra\_escritor(MRef<LectoresEscritores> monitor, int numEscritores){      while(true){          Espera();          monitor->iniEsc();          escribir(numEscritores);          monitor->finEsc();      }  }  int main(){      cout << "-------------------------------------------------------" << endl <<              "- Problema de los lectores y escritores. Monitor SU. --" << endl <<              "-------------------------------------------------------" << endl << flush;      MRef<LectoresEscritores> monitor = Create<LectoresEscritores>();      thread hebras\_lectoras[num\_lectores], hebras\_escritoras[num\_escritores];      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();     } |

. Salida del programa

|  |
| --- |
| -------------------------------------------------------  - Problema de los lectores y escritores. Monitor SU. --  -------------------------------------------------------  El lector 1 comienza a leer ( 20 milisegundos)  El lector 1 ha terminado de leer  El escritor 0 comienza a escribir ( 198 milisegundos)  El escritor 0 ha terminado de escribir  El lector 2 comienza a leer ( 59 milisegundos)  El lector 0 comienza a leer ( 31 milisegundos)  El lector 1 comienza a leer ( 62 milisegundos)  El lector 0 ha terminado de leer  El lector 2 ha terminado de leer  El lector 1 ha terminado de leer  El escritor 2 comienza a escribir ( 124 milisegundos)  El escritor 2 ha terminado de escribir  El escritor 1 comienza a escribir ( 128 milisegundos)  El escritor 1 ha terminado de escribir  El lector 1 comienza a leer ( 163 milisegundos)  El lector 2 comienza a leer ( 108 milisegundos)  El lector 0 comienza a leer ( 167 milisegundos)  El lector 2 ha terminado de leer  El lector 1 ha terminado de leer  El lector 0 ha terminado de leer  El escritor 0 comienza a escribir ( 160 milisegundos)  El escritor 0 ha terminado de escribir  El lector 1 comienza a leer ( 130 milisegundos)  El lector 2 comienza a leer ( 95 milisegundos)  El lector 0 comienza a leer ( 67 milisegundos)  El lector 0 ha terminado de leer  El lector 2 ha terminado de leer  El lector 1 ha terminado de leer  El escritor 2 comienza a escribir ( 55 milisegundos)  El escritor 2 ha terminado de escribir  El lector 2 comienza a leer ( 47 milisegundos)  El lector 0 comienza a leer ( 177 milisegundos)  El lector 2 ha terminado de leer  El lector 1 comienza a leer ( 117 milisegundos)  El lector 2 comienza a leer ( 149 milisegundos)  El lector 0 ha terminado de leer  El lector 1 ha terminado de leer  El lector 1 comienza a leer ( 159 milisegundos)  El lector 2 ha terminado de leer  El lector 0 comienza a leer ( 181 milisegundos)  El lector 2 comienza a leer ( 69 milisegundos)  El lector 2 ha terminado de leer  El lector 1 ha terminado de leer  El lector 1 comienza a leer ( 115 milisegundos)  El lector 0 ha terminado de leer  El lector 2 comienza a leer ( 169 milisegundos)  El lector 0 comienza a leer ( 24 milisegundos)  El lector 1 ha terminado de leer  El lector 0 ha terminado de leer  El lector 1 comienza a leer ( 93 milisegundos)  El lector 0 comienza a leer ( 91 milisegundos)  El lector 2 ha terminado de leer  El lector 1 ha terminado de leer  El lector 0 ha terminado de leer  El escritor 1 comienza a escribir ( 108 milisegundos)  El escritor 1 ha terminado de escribir  El escritor 0 comienza a escribir ( 109 milisegundos)  El escritor 0 ha terminado de escribir  El lector 0 comienza a leer ( 193 milisegundos)  El lector 2 comienza a leer ( 32 milisegundos)  El lector 1 comienza a leer ( 140 milisegundos)  El lector 2 ha terminado de leer  El lector 1 ha terminado de leer  El lector 2 comienza a leer ( 151 milisegundos)  El lector 0 ha terminado de leer  El lector 0 comienza a leer ( 191 milisegundos)  El lector 1 comienza a leer ( 34 milisegundos)  El lector 1 ha terminado de leer  El lector 2 ha terminado de leer  El lector 2 comienza a leer ( 169 milisegundos) |