portafolios Práctica 3 Scd

Paso de mensajes con MPI

Víctor José Rubia López

UGR ETSIIT

2ºB (B2)

[El problema de los productores-consumidores 2](#_Toc58238256)

[1. Describe qué cambios has realizado sobre el programa de partida y el propósito de dichos cambios. 2](#_Toc58238257)

[1. Código fuente 2](#_Toc58238258)

[2. Salida del programa 5](#_Toc58238259)

[El problema de los filósofos, interbloqueo 7](#_Toc58238260)

[1. Describe qué cambios has realizado sobre el programa de partida y el propósito de dichos cambios. 7](#_Toc58238261)

[3. Código fuente 7](#_Toc58238262)

[4. Salida del programa 9](#_Toc58238263)

[El problema de los filósofos, sin interbloqueo 11](#_Toc58238264)

[1. Describe qué cambios has realizado sobre el programa de partida y el propósito de dichos cambios. 11](#_Toc58238265)

[5. Código fuente 11](#_Toc58238266)

[6. Salida del programa 13](#_Toc58238267)

[El problema de los filósofos, con camarero 15](#_Toc58238268)

[1. Describe qué cambios has realizado sobre el programa de partida y el propósito de dichos cambios. 15](#_Toc58238269)

[7. Código fuente 15](#_Toc58238270)

[8. Salida del programa 19](#_Toc58238271)

El problema de los productores-consumidores

*1. Describe qué cambios has realizado sobre el programa de partida y el propósito de dichos cambios.*

Para poder tener 4 procesos productores y 5 procesos consumidores hemos declarado dos variables n\_productores y n\_consumidores. La variable num\_items es el producto de n\_productores y n\_consumidores para que sea múltiplo de ellos. La variable id\_buffer toma el valor de n\_productores, que es 4. He usado como etiqueta para los productores el 0 y como etiqueta para los consumidores el 1.

En la función producir ahora uso como valor producido el que se calcula del producto de id\_productor (que recibe ahora como argumento la función) y produccion\_individual (variable que se calcula del num\_items divido por el número de productores) sumado junto al contador.

En la función productor ahora también se tiene como parámetro id\_productor. Además, cuando hacemos el envío síncrono con MPI, usamos como tag etiq\_productores.

En cuanto a la función consumidor ocurre de forma análoga a la función productor. En los envíos y envíos de MPI usamos ahora como tag a la etiqueta del consumidor. Además, hemos cambiado el número de iteraciones del for, pues ahora hará un total de 4 iteraciones (valor almacenado en la variable consumicion\_individual).

En la función buffer hemos realizado los cambios correspondientes a las nuevas variables. He sustituido los id por las etiq. Además, en el punto 2 hemos sustituido el 0 del tag por etiq\_emisor\_aceptable. En el punto 3 en el caso de etiq\_consumidores he cambiado de igual forma la etiqueta del envío síncrono de MPI de 0 a etiq\_consumidores.

Por último en la función main ahora en la condición consideraremos que si el id\_propio es menor que 4, entonces es un productor.

. Código fuente

|  |
| --- |
| // -----------------------------------------------------------------------------  //  // Sistemas concurrentes y Distribuidos.  // Práctica 3. Implementación de algoritmos distribuidos con MPI  //  // Archivo: prodcons.cpp  // Implementación del problema del productor-consumidor con  // un proceso intermedio que recibe mensajes síncronos de forma alterna.  // (versión con un único productor y un único consumidor)  //  // Historial:  // Actualizado a C++11 en Septiembre de 2017  // Compilar: mpicxx -std=c++11 prodcons-mu.cpp -o prodcons-mu  // Ejecutar: mpirun -np 10 ./prodcons-mu  // -----------------------------------------------------------------------------  #include <iostream>  #include <thread> // this\_thread::sleep\_for  #include <random> // dispositivos, generadores y distribuciones aleatorias  #include <chrono> // duraciones (duration), unidades de tiempo  #include <mpi.h>  using namespace std;  using namespace std::this\_thread ;  using namespace std::chrono ;  // ---------------------------------------------------------------------  // constantes que determinan la asignación de identificadores a roles:  const int     n\_productores           = 4,     n\_consumidores          = 5,     id\_buffer               = n\_productores, // identificador del proceso buffer     etiq\_productores        = 0, // identificador de los mensajes de los productores     etiq\_consumidores       = 1, // identificador de los mensajes de los consumidores     num\_procesos\_esperado   = n\_productores + n\_consumidores + 1, // número total de procesos esperado     num\_items               = n\_productores \* n\_consumidores, // numero de items producidos o consumidos     tam\_vector              = 10,     produccion\_individual   = num\_items / n\_productores,     consumicion\_individual  = num\_items / n\_consumidores;  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // 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 );  }  // ---------------------------------------------------------------------  // produce los numeros en secuencia (1,2,3,....)  int producir( int id\_productor)  {     static int contador = 0 ;     int valor = id\_productor\*produccion\_individual + contador;     sleep\_for( milliseconds( aleatorio<10,200>()) );     contador++ ;     cout << "Productor ha producido valor " << valor << endl << flush;     return valor ;  }  // ---------------------------------------------------------------------  void funcion\_productor(int id\_productor)  {     for ( unsigned int i= 0 ; i < produccion\_individual ; i++ )     {        // producir valor        int valor\_prod = producir(id\_productor);        // enviar valor        cout << "Productor "<< id\_productor << " va a enviar valor " << valor\_prod << endl << flush;        MPI\_Ssend( &valor\_prod, 1, MPI\_INT, id\_buffer, etiq\_productores, MPI\_COMM\_WORLD );     }  }  // ---------------------------------------------------------------------  void consumir( int valor\_cons )  {     // espera bloqueada     sleep\_for( milliseconds( aleatorio<10,200>()) );     cout << "Consumidor ha consumido valor " << valor\_cons << endl << flush ;  }  // ---------------------------------------------------------------------  void funcion\_consumidor(int id\_consumidor)  {     int         peticion,                 valor\_rec = 1 ;     MPI\_Status  estado ;     for( unsigned int i=0 ; i < consumicion\_individual; i++ )     {        MPI\_Ssend( &peticion,  1, MPI\_INT, id\_buffer, etiq\_consumidores, MPI\_COMM\_WORLD);        MPI\_Recv ( &valor\_rec, 1, MPI\_INT, id\_buffer, etiq\_consumidores, MPI\_COMM\_WORLD,&estado );        cout << "\t\tConsumidor " << id\_consumidor << " ha recibido valor " << valor\_rec << endl << flush ;        consumir( valor\_rec );     }  }  // ---------------------------------------------------------------------  void funcion\_buffer()  {     int         buffer[tam\_vector],        // buffer con celdas ocupadas y vacías                 valor ,                    // valor recibido o enviado                 primera\_libre        = 0,  // índice de primera celda libre                 primera\_ocupada      = 0,  // Índice de primera celda ocupada                 num\_celdas\_ocupadas  = 0,  // Número de celdas ocupadas                 etiq\_emisor\_aceptable;     // Identificador de emisor aceptable     MPI\_Status  estado ;                   // Metadatos del mensaje recibido     for ( unsigned int i = 0 ; i < num\_items\*2 ; i++ )     {        // 1. determinar si puede enviar solo prod., solo cons, o todos        if ( num\_celdas\_ocupadas == 0 )               // si buffer vacío           etiq\_emisor\_aceptable = etiq\_productores ;       // $~~~$ solo prod.        else if ( num\_celdas\_ocupadas == tam\_vector ) // si buffer lleno           etiq\_emisor\_aceptable = etiq\_consumidores ;      // $~~~$ solo cons.        else                                          // si no vacío ni lleno           etiq\_emisor\_aceptable = MPI\_ANY\_TAG ;     // $~~~$ cualquiera        // 2. recibir un mensaje del emisor o emisores aceptables        MPI\_Recv( &valor, 1, MPI\_INT, MPI\_ANY\_SOURCE, etiq\_emisor\_aceptable, MPI\_COMM\_WORLD, &estado );        // 3. procesar el mensaje recibido        switch( estado.MPI\_TAG ){   // leer emisor del mensaje en metadatos             case etiq\_productores:  // si ha sido el productor: insertar en buffer              buffer[primera\_libre] = valor;              primera\_libre = (primera\_libre+1) % tam\_vector;              num\_celdas\_ocupadas++;              cout << "Buffer ha recibido valor " << valor << endl << flush;              break;             case etiq\_consumidores:    // si ha sido el consumidor: extraer y enviarle              valor = buffer[primera\_ocupada];              primera\_ocupada = (primera\_ocupada+1) % tam\_vector;              num\_celdas\_ocupadas--;              cout << "\t\tBuffer va a enviar valor " << valor << endl << flush;              MPI\_Ssend( &valor, 1, MPI\_INT, estado.MPI\_SOURCE, etiq\_consumidores, MPI\_COMM\_WORLD);              break;        }     }  }  // ---------------------------------------------------------------------  int main( int argc, char \*argv[] )  {    int id\_propio, num\_procesos\_actual; // ident. propio, núm. de procesos    MPI\_Init( &argc, &argv );    MPI\_Comm\_rank( MPI\_COMM\_WORLD, &id\_propio );    MPI\_Comm\_size( MPI\_COMM\_WORLD, &num\_procesos\_actual );    if ( num\_procesos\_esperado == num\_procesos\_actual )    {      if ( id\_propio < id\_buffer )  // si mi ident. es el del productor        funcion\_productor(id\_propio);            //    ejecutar función del productor      else if ( id\_propio == id\_buffer )// si mi ident. es el del buffer        funcion\_buffer();               //    ejecutar función buffer      else                              // en otro caso, mi ident es consumidor        funcion\_consumidor(id\_propio);           //    ejecutar función consumidor    }    else if ( id\_propio == 0 ){  // si hay error, el proceso 0 informa      cerr << "error: número de procesos distinto del esperado." << endl ;      cout << "el número de procesos esperados es:    " << num\_procesos\_esperado << endl               << "el número de procesos en ejecución es: " << num\_procesos\_actual << endl               << "(programa abortado)" << endl ;    }      MPI\_Finalize( );    return 0;  }  // --------------------------------------------------------------------- |

. Salida del programa

|  |
| --- |
| victor@DESKTOP-B8597DM:/mnt/c/Users/victo/Google Drive/01\_SCD/Practicas/P3$ mpirun -np 10 ./prodcons-mu  Productor ha producido valor 10  Productor 2 va a enviar valor 10  Buffer ha recibido valor 10  Buffer va a enviar valor 10  Consumidor 5 ha recibido valor 10  Productor ha producido valor 0  Productor 0 va a enviar valor 0  Buffer ha recibido valor 0  Buffer va a enviar valor 0  Consumidor 7 ha recibido valor 0  Consumidor ha consumido valor 0  Consumidor ha consumido valor 10  Productor ha producido valor 5  Productor 1 va a enviar valor 5  Buffer ha recibido valor 5  Buffer va a enviar valor 5  Consumidor 8 ha recibido valor 5  Consumidor ha consumido valor 5  Productor ha producido valor 6  Productor 1 va a enviar valor 6  Buffer ha recibido valor 6  Buffer va a enviar valor 6  Consumidor 9 ha recibido valor 6  Productor ha producido valor 15  Productor 3 va a enviar valor 15  Buffer ha recibido valor 15  Buffer va a enviar valor 15  Consumidor 6 ha recibido valor 15  Productor ha producido valor 11  Productor 2 va a enviar valor 11  Buffer ha recibido valor 11  Buffer va a enviar valor 11  Consumidor 7 ha recibido valor 11  Productor ha producido valor 7  Productor 1 va a enviar valor 7  Buffer ha recibido valor 7  Buffer va a enviar valor 7  Consumidor 5 ha recibido valor 7  Productor ha producido valor 8  Productor 1 va a enviar valor 8  Buffer ha recibido valor 8  Buffer va a enviar valor 8  Consumidor 8 ha recibido valor 8  Consumidor ha consumido valor 11  Consumidor ha consumido valor 6  Productor ha producido valor 1  Productor 0 va a enviar valor 1  Buffer ha recibido valor 1  Buffer va a enviar valor 1  Consumidor 7 ha recibido valor 1  Productor ha producido valor 9  Productor 1 va a enviar valor 9  Buffer ha recibido valor 9  Buffer va a enviar valor 9  Consumidor 9 ha recibido valor 9  Productor ha producido valor 2  Productor 0 va a enviar valor 2  Buffer ha recibido valor 2  Buffer ha recibido valor 16  Productor ha producido valor 16  Productor 3 va a enviar valor 16  Buffer va a enviar valor 2  Consumidor ha consumido valor 9  Consumidor 9 ha recibido valor 2  Productor ha producido valor 3  Productor 0 va a enviar valor 3  Buffer ha recibido valor 3  Buffer va a enviar valor 16  Consumidor ha consumido valor 2  Consumidor 9 ha recibido valor 16  Buffer va a enviar valor 3  Consumidor ha consumido valor 7  Consumidor 5 ha recibido valor 3  Consumidor ha consumido valor 15  Consumidor ha consumido valor 8  Productor ha producido valor 12  Productor 2 va a enviar valor 12  Buffer ha recibido valor 12  Buffer va a enviar valor 12  Consumidor 6 ha recibido valor 12  Productor ha producido valor 13  Productor 2 va a enviar valor 13  Buffer ha recibido valor 13  Buffer va a enviar valor 13  Consumidor 8 ha recibido valor 13  Consumidor ha consumido valor 16 |

El problema de los filósofos, interbloqueo

*1. Describe qué cambios has realizado sobre el programa de partida y el propósito de dichos cambios.*

He completado donde estaban los comentarios con el uso de las funciones de envío síncrono de MPI. Además, he creado una nueva variable petición que usaré como valor a enviar.

En la función tenedores de igual forma he completado los comentarios usando la función de MPI para recibir, guardándola en valor. Posteriormente guardo el id de filósofo y por último recibo la liberación del tenedor por dicho filósofo.

. Código fuente

|  |
| --- |
| // -----------------------------------------------------------------------------  //  // Sistemas concurrentes y Distribuidos.  // Práctica 3. Implementación de algoritmos distribuidos con MPI  //  // Archivo: filosofos-plantilla.cpp  // Implementación del problema de los filósofos (sin camarero).  // Plantilla para completar.  //  // Historial:  // Actualizado a C++11 en Septiembre de 2017  // -----------------------------------------------------------------------------  #include <mpi.h>  #include <thread> // this\_thread::sleep\_for  #include <random> // dispositivos, generadores y distribuciones aleatorias  #include <chrono> // duraciones (duration), unidades de tiempo  #include <iostream>  using namespace std;  using namespace std::this\_thread ;  using namespace std::chrono ;  const int     num\_filosofos = 5 ,     num\_procesos  = 2\*num\_filosofos ;  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // 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 funcion\_filosofos( int id )  {    int id\_ten\_izq = (id+1)              % num\_procesos, //id. tenedor izq.        id\_ten\_der = (id+num\_procesos-1) % num\_procesos, //id. tenedor der.        peticion;    while ( true )    {      cout <<"Filósofo " <<id << " solicita ten. izq." <<id\_ten\_izq <<endl << flush;      // ... solicitar tenedor izquierdo (completar)      MPI\_Ssend( &peticion, 1, MPI\_INT, id\_ten\_izq, 0, MPI\_COMM\_WORLD);      cout <<"Filósofo " <<id <<" solicita ten. der." <<id\_ten\_der <<endl << flush;      // ... solicitar tenedor derecho (completar)      MPI\_Ssend ( &peticion, 1, MPI\_INT, id\_ten\_der, 0, MPI\_COMM\_WORLD);      cout <<"Filósofo " <<id <<" comienza a comer" <<endl << flush ;      sleep\_for( milliseconds( aleatorio<10,100>() ) );      cout <<"Filósofo " <<id <<" suelta ten. izq. " <<id\_ten\_izq <<endl << flush;      // ... soltar el tenedor izquierdo (completar)      MPI\_Ssend( &peticion, 1, MPI\_INT, id\_ten\_izq, 0, MPI\_COMM\_WORLD);      cout<< "Filósofo " <<id <<" suelta ten. der. " <<id\_ten\_der <<endl << flush;      // ... soltar el tenedor derecho (completar)      MPI\_Ssend( &peticion, 1, MPI\_INT, id\_ten\_der, 0, MPI\_COMM\_WORLD );      cout << "Filosofo " << id << " comienza a pensar" << endl;      sleep\_for( milliseconds( aleatorio<10,100>() ) );   }  }  // ---------------------------------------------------------------------  void funcion\_tenedores( int id )  {    int valor, id\_filosofo ;  // valor recibido, identificador del filósofo    MPI\_Status estado ;       // metadatos de las dos recepciones    while ( true )    {       // ...... recibir petición de cualquier filósofo (completar)       MPI\_Recv( &valor, 1, MPI\_INT, MPI\_ANY\_SOURCE, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &estado);       // ...... guardar en 'id\_filosofo' el id. del emisor (completar)       id\_filosofo = estado.MPI\_SOURCE;       cout <<"Ten. " <<id <<" ha sido cogido por filo. " <<id\_filosofo <<endl << flush;       // ...... recibir liberación de filósofo 'id\_filosofo' (completar)       MPI\_Recv( &valor, 1, MPI\_INT, id\_filosofo, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &estado);       cout <<"Ten. "<< id<< " ha sido liberado por filo. " <<id\_filosofo <<endl << flush ;    }  }  // ---------------------------------------------------------------------  int main( int argc, char\*\* argv )  {     int id\_propio, num\_procesos\_actual ;     MPI\_Init( &argc, &argv );     MPI\_Comm\_rank( MPI\_COMM\_WORLD, &id\_propio );     MPI\_Comm\_size( MPI\_COMM\_WORLD, &num\_procesos\_actual );     if ( num\_procesos == num\_procesos\_actual )     {        // ejecutar la función correspondiente a 'id\_propio'        if ( id\_propio % 2 == 0 )          // si es par           funcion\_filosofos( id\_propio ); //   es un filósofo        else                               // si es impar           funcion\_tenedores( id\_propio ); //   es un tenedor     }     else     {        if ( id\_propio == 0 ) // solo el primero escribe error, indep. del rol        { cout << "el número de procesos esperados es:    " << num\_procesos << endl               << "el número de procesos en ejecución es: " << num\_procesos\_actual << endl               << "(programa abortado)" << endl ;        }     }     MPI\_Finalize( );     return 0;  }  // --------------------------------------------------------------------- |

. Salida del programa

|  |
| --- |
| Filósofo 0 solicita ten. izq.1  Filósofo 0 solicita ten. der.9  Filósofo 2 solicita ten. izq.3  Filósofo 2 solicita ten. der.1  Filósofo 8 solicita ten. izq.9  Filósofo 0 comienza a comer  Ten. 1 ha sido cogido por filo. 0  Ten. 3 ha sido cogido por filo. 2  Filósofo 4 solicita ten. izq.5  Ten. 9 ha sido cogido por filo. 0  Filósofo 4 solicita ten. der.3  Ten. 5 ha sido cogido por filo. 4  Filósofo 6 solicita ten. izq.7  Filósofo 6 solicita ten. der.5  Ten. 7 ha sido cogido por filo. 6  Filósofo 0 suelta ten. izq. 1  Filósofo 0 suelta ten. der. 9  Filosofo 0 comienza a pensar  Ten. 1 ha sido liberado por filo. 0  Ten. 1 ha sido cogido por filo. 2  Filósofo 2 comienza a comer  Filósofo 8 solicita ten. der.7  Ten. 9 ha sido liberado por filo. 0  Ten. 9 ha sido cogido por filo. 8  Filósofo 2 suelta ten. izq. 3  Filósofo 2 suelta ten. der. 1  Filosofo 2 comienza a pensar  Ten. 1 ha sido liberado por filo. 2  Ten. 3 ha sido liberado por filo. 2  Ten. 3 ha sido cogido por filo. 4  Filósofo 4 comienza a comer  Filósofo 0 solicita ten. izq.1  Filósofo 0 solicita ten. der.9  Ten. 1 ha sido cogido por filo. 0  Filósofo 2 solicita ten. izq.3  Filósofo 4 suelta ten. izq. 5  Filósofo 4 suelta ten. der. 3  Filosofo 4 comienza a pensar  Filósofo 2 solicita ten. der.1  Ten. 3 ha sido liberado por filo. 4  Ten. 3 ha sido cogido por filo. 2  Ten. 5 ha sido liberado por filo. 4  Ten. 5 ha sido cogido por filo. 6  Filósofo 6 comienza a comer  Filósofo 6 suelta ten. izq. 7  Filósofo 6 suelta ten. der. 5  Filosofo 6 comienza a pensar  Ten. 5 ha sido liberado por filo. 6  Ten. 7 ha sido liberado por filo. 6  Ten. 7 ha sido cogido por filo. 8  Filósofo 8 comienza a comer  Filósofo 4 solicita ten. izq.5  Filósofo 4 solicita ten. der.3  Ten. 5 ha sido cogido por filo. 4  Filósofo 6 solicita ten. izq.7  Filósofo 8 suelta ten. izq. 9  Filósofo 8 suelta ten. der. 7  Filosofo 8 comienza a pensar  Filósofo 0 comienza a comer  Filósofo 6 solicita ten. der.5  Ten. 7 ha sido liberado por filo. 8  Ten. 7 ha sido cogido por filo. 6  Ten. 9 ha sido liberado por filo. 8  Ten. 9 ha sido cogido por filo. 0  Filósofo 0 suelta ten. izq. 1  Filósofo 0 suelta ten. der. 9  Filosofo 0 comienza a pensar  Ten. 1 ha sido liberado por filo. 0  Ten. 1 ha sido cogido por filo. 2  Filósofo 2 comienza a comer  Ten. 9 ha sido liberado por filo. 0  Filósofo 2 suelta ten. izq. 3  Filósofo 2 suelta ten. der. 1  Filosofo 2 comienza a pensar  Ten. 1 ha sido liberado por filo. 2  Ten. 3 ha sido liberado por filo. 2  Ten. 3 ha sido cogido por filo. 4  Filósofo 4 comienza a comer  Filósofo 0 solicita ten. izq.1  Filósofo 0 solicita ten. der.9  Filósofo 0 comienza a comer  Ten. 1 ha sido cogido por filo. 0  Ten. 9 ha sido cogido por filo. 0  Filósofo 8 solicita ten. izq.9 |

El problema de los filósofos, sin interbloqueo

1. Describe qué cambios has realizado sobre el programa de partida y el propósito de dichos cambios.

En este caso, debemos evitar que todos los filósofos cojan su tenedor en el mismo orden ya que ocurriría un interbloqueo. Para evitar esto lo que haremos es identificar la secuencia de acciones que han de ocurrir. Una vez identificadas, hemos hecho un if-else en el que distinguimos, dependiendo de la id del filósofo, si se coge el tenedor derecho o izquierdo.

. Código fuente

|  |
| --- |
| // -----------------------------------------------------------------------------  //  // Sistemas concurrentes y Distribuidos.  // Práctica 3. Implementación de algoritmos distribuidos con MPI  //  // Archivo: filosofos-plantilla.cpp  // Implementación del problema de los filósofos (sin camarero).  // Plantilla para completar.  //  // Historial:  // Actualizado a C++11 en Septiembre de 2017  // -----------------------------------------------------------------------------  #include <mpi.h>  #include <thread> // this\_thread::sleep\_for  #include <random> // dispositivos, generadores y distribuciones aleatorias  #include <chrono> // duraciones (duration), unidades de tiempo  #include <iostream>  using namespace std;  using namespace std::this\_thread ;  using namespace std::chrono ;  const int     num\_filosofos = 5 ,     num\_procesos  = 2\*num\_filosofos ;  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // 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 funcion\_filosofos( int id )  {    int id\_ten\_1 , //id. tenedor izq.        id\_ten\_2, //id. tenedor der.        peticion;    while ( true )    {       if(id < num\_filosofos){          id\_ten\_1 = (id+1) % num\_procesos;          id\_ten\_2 = (id+num\_procesos-1) % num\_procesos;          cout << "Tengo id menor que 5 y soy fil. num. " << id << endl << flush;       }       else{          id\_ten\_2 = (id + 1) % num\_procesos;          id\_ten\_1 = (id + num\_procesos-1) % num\_procesos;          cout << "Tengo id mayor que 5 y soy fil. num. " << id << endl << flush;       }      cout <<"Filósofo " <<id << " solicita ten. 1. con id " <<id\_ten\_1 <<endl;      // ... solicitar tenedor izquierdo (completar)      MPI\_Ssend( &peticion, 1, MPI\_INT, id\_ten\_1, 0, MPI\_COMM\_WORLD );      cout <<"Filósofo " <<id <<" solicita ten. 2. con id " <<id\_ten\_2 <<endl;      // ... solicitar tenedor derecho (completar)      MPI\_Ssend( &peticion, 1, MPI\_INT, id\_ten\_2, 0, MPI\_COMM\_WORLD );      cout <<"Filósofo " <<id <<" comienza a comer" <<endl ;      sleep\_for( milliseconds( aleatorio<10,100>() ) );      cout <<"Filósofo " <<id <<" suelta ten. 1. con id " <<id\_ten\_1 <<endl;      // ... soltar el tenedor izquierdo (completar)      MPI\_Ssend( &peticion, 1, MPI\_INT, id\_ten\_1, 0, MPI\_COMM\_WORLD );      cout<< "Filósofo " <<id <<" suelta ten. 2. con id " <<id\_ten\_2 <<endl;      // ... soltar el tenedor derecho (completar)      MPI\_Ssend( &peticion, 1, MPI\_INT, id\_ten\_2, 0, MPI\_COMM\_WORLD );      cout << "Filosofo " << id << " comienza a pensar" << endl;      sleep\_for( milliseconds( aleatorio<10,100>() ) );   }  }  // ---------------------------------------------------------------------  void funcion\_tenedores( int id )  {    int valor, id\_filosofo ;  // valor recibido, identificador del filósofo    MPI\_Status estado ;       // metadatos de las dos recepciones    while ( true )    {       // ...... recibir petición de cualquier filósofo (completar)         MPI\_Recv( &valor, 1, MPI\_INT, MPI\_ANY\_SOURCE, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &estado);       // ...... guardar en 'id\_filosofo' el id. del emisor (completar)         id\_filosofo = estado.MPI\_SOURCE;       cout <<"Ten. " <<id <<" ha sido cogido por filo. " <<id\_filosofo <<endl;       // ...... recibir liberación de filósofo 'id\_filosofo' (completar)         MPI\_Recv( &valor, 1, MPI\_INT, id\_filosofo, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &estado);       cout <<"Ten. "<< id<< " ha sido liberado por filo. " <<id\_filosofo <<endl ;    }  }  // ---------------------------------------------------------------------  int main( int argc, char\*\* argv )  {     int id\_propio, num\_procesos\_actual ;     MPI\_Init( &argc, &argv );     MPI\_Comm\_rank( MPI\_COMM\_WORLD, &id\_propio );     MPI\_Comm\_size( MPI\_COMM\_WORLD, &num\_procesos\_actual );     if ( num\_procesos == num\_procesos\_actual )     {        // ejecutar la función correspondiente a 'id\_propio'        if ( id\_propio % 2 == 0 )          // si es par           funcion\_filosofos( id\_propio ); //   es un filósofo        else                               // si es impar           funcion\_tenedores( id\_propio ); //   es un tenedor     }     else     {        if ( id\_propio == 0 ) // solo el primero escribe error, indep. del rol        { cout << "el número de procesos esperados es:    " << num\_procesos << endl               << "el número de procesos en ejecución es: " << num\_procesos\_actual << endl               << "(programa abortado)" << endl ;        }     }     MPI\_Finalize( );     return 0;  }  // --------------------------------------------------------------------- |

. Salida del programa

|  |
| --- |
| Tengo id menor que 5 y soy fil. num. 2  Filósofo 2 solicita ten. 1. con id 3  Tengo id mayor que 5 y soy fil. num. 8  Filósofo 8 solicita ten. 1. con id 7  Filósofo 8 solicita ten. 2. con id 9  Ten. 7 ha sido cogido por filo. 8  Tengo id menor que 5 y soy fil. num. 0  Filósofo 0 solicita ten. 1. con id 1  Filósofo 2 solicita ten. 2. con id 1  Ten. 3 ha sido cogido por filo. 2  Filósofo 0 solicita ten. 2. con id 9  Tengo id mayor que 5 y soy fil. num. 6  Filósofo 6 solicita ten. 1. con id 5  Ten. 1 ha sido cogido por filo. 0  Filósofo 0 comienza a comer  Ten. 9 ha sido cogido por filo. 0  Ten. 5 ha sido cogido por filo. 6  Filósofo 6 solicita ten. 2. con id 7  Tengo id menor que 5 y soy fil. num. 4  Filósofo 4 solicita ten. 1. con id 5  Filósofo 0 suelta ten. 1. con id 1  Filósofo 0 suelta ten. 2. con id 9  Filosofo 0 comienza a pensar  Ten. 1 ha sido liberado por filo. 0  Ten. 1 ha sido cogido por filo. 2  Filósofo 2 comienza a comer  Filósofo 8 comienza a comer  Ten. 9 ha sido liberado por filo. 0  Ten. 9 ha sido cogido por filo. 8  Filósofo 8 suelta ten. 1. con id 7  Filósofo 8 suelta ten. 2. con id 9  Filosofo 8 comienza a pensar  Filósofo 6 comienza a comer  Ten. 7 ha sido liberado por filo. 8  Ten. 7 ha sido cogido por filo. 6  Ten. 9 ha sido liberado por filo. 8  Tengo id mayor que 5 y soy fil. num. 8  Filósofo 8 solicita ten. 1. con id 7  Filósofo 2 suelta ten. 1. con id 3  Filósofo 2 suelta ten. 2. con id 1  Filosofo 2 comienza a pensar  Ten. 1 ha sido liberado por filo. 2  Ten. 3 ha sido liberado por filo. 2  Filósofo 6 suelta ten. 1. con id 5  Filósofo 6 suelta ten. 2. con id 7  Filosofo 6 comienza a pensar |

El problema de los filósofos, con camarero

1. Describe qué cambios has realizado sobre el programa de partida y el propósito de dichos cambios.

Para solucionar este problema, se agrega un proceso camarero con id=10. Este proceso irá indicando a los filósofos que esperan a que haya un sitio libre, que pueden sentarse para comer. Una vez hayan terminado, le enviarán un mensaje al camarero para indicárselo.

Los filósofos realizan las siguientes acciones: piden sentarse al camarero, esperan a que le digan que puede sentarse, se sienta, solicita el tenedor izquierdo, solicita el tenedor derecho y se levanta avisando al camarero. Además calculan su posición inmediata a derecha e izquierda para

El tenedor espera una petición desde cualquier filósofo vecino, recibe la petición del filósofo y espera a que el filósofo suelte el tenedor.

El camarero si no hay sitios libres los filósofos solo podrían levantarse. Si por el contrario hubiese, los filósofos pueden tanto sentarse como levantarse.

. Código fuente

|  |
| --- |
| // -----------------------------------------------------------------------------  //  // Sistemas concurrentes y Distribuidos.  // Práctica 3. Implementación de algoritmos distribuidos con MPI  //  // Archivo: filosofos-plantilla.cpp  // Implementación del problema de los filósofos (sin camarero).  // Plantilla para completar.  //  // Historial:  // Actualizado a C++11 en Septiembre de 2017  // -----------------------------------------------------------------------------  #include <mpi.h>  #include <thread> // this\_thread::sleep\_for  #include <random> // dispositivos, generadores y distribuciones aleatorias  #include <chrono> // duraciones (duration), unidades de tiempo  #include <iostream>  using namespace std;  using namespace std::this\_thread ;  using namespace std::chrono ;  const int     num\_filosofos = 5 ,     num\_procesos  = 2\*num\_filosofos +1,      etiq\_levantarse = 2,      etiq\_sentarse = 1;  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // 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 funcion\_filosofos( int id )  {    int id\_ten\_izq = (id + 1) % (num\_procesos-1) , //id. tenedor izq.        id\_ten\_der = (id + num\_procesos - 2) % (num\_procesos - 1), //id. tenedor der.           id\_camarero = num\_procesos - 1,        peticion;    while ( true )    {      cout << "Filosofo " << id << " solicita sentarse" << endl << flush;      MPI\_Ssend( &peticion, 1, MPI\_INT, id\_camarero, etiq\_sentarse, MPI\_COMM\_WORLD);      cout <<"Filósofo " <<id << " solicita ten. izq." <<id\_ten\_izq <<endl << flush;      // ... solicitar tenedor izquierdo (completar)      MPI\_Ssend( &peticion, 1, MPI\_INT, id\_ten\_izq, 0, MPI\_COMM\_WORLD );      cout <<"Filósofo " <<id <<" solicita ten. der." <<id\_ten\_der <<endl << flush;      // ... solicitar tenedor derecho (completar)      MPI\_Ssend( &peticion, 1, MPI\_INT, id\_ten\_der, 0, MPI\_COMM\_WORLD );      cout <<"Filósofo " <<id <<" comienza a comer" <<endl << flush ;      sleep\_for( milliseconds( aleatorio<10,100>() ) );      cout <<"Filósofo " <<id <<" suelta ten. izq. " <<id\_ten\_izq <<endl << flush;      // ... soltar el tenedor izquierdo (completar)      MPI\_Ssend( &peticion, 1, MPI\_INT, id\_ten\_izq, 0, MPI\_COMM\_WORLD );      cout<< "Filósofo " <<id <<" suelta ten. der. " <<id\_ten\_der <<endl << flush;      // ... soltar el tenedor derecho (completar)      MPI\_Ssend( &peticion, 1, MPI\_INT, id\_ten\_der, 0, MPI\_COMM\_WORLD );      cout << "Filosofo " << id << " solicita levantarse" << endl << flush;      MPI\_Ssend( &peticion, 1, MPI\_INT, id\_camarero, etiq\_levantarse, MPI\_COMM\_WORLD);      cout << "Filosofo " << id << " comienza a pensar" << endl << flush;      sleep\_for( milliseconds( aleatorio<10,100>() ) );   }  }  // ---------------------------------------------------------------------  void funcion\_tenedores( int id )  {    int valor, id\_filosofo ;  // valor recibido, identificador del filósofo    MPI\_Status estado ;       // metadatos de las dos recepciones    while ( true )    {       // ...... recibir petición de cualquier filósofo (completar)         MPI\_Recv( &valor, 1, MPI\_INT, MPI\_ANY\_SOURCE, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &estado);       // ...... guardar en 'id\_filosofo' el id. del emisor (completar)         id\_filosofo = estado.MPI\_SOURCE;       cout <<"Ten. " <<id <<" ha sido cogido por filo. " <<id\_filosofo <<endl << flush;       // ...... recibir liberación de filósofo 'id\_filosofo' (completar)         MPI\_Recv( &valor, 1, MPI\_INT, id\_filosofo, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &estado);       cout <<"Ten. "<< id<< " ha sido liberado por filo. " <<id\_filosofo <<endl << flush ;    }  }  // ---------------------------------------------------------------------  void funcion\_camarero(){     int filosofos\_sentados = 0,         etiq\_actual,         valor; // valor recibido     MPI\_Status estado; // metadatos de las dos recepciones     while(true){        if( filosofos\_sentados >= num\_filosofos - 1)           etiq\_actual = etiq\_levantarse;        else           etiq\_actual = MPI\_ANY\_TAG;          MPI\_Recv( &valor, 1, MPI\_INT, MPI\_ANY\_SOURCE, etiq\_actual, MPI\_COMM\_WORLD, &estado);        switch (estado.MPI\_TAG)        {        case etiq\_sentarse:           filosofos\_sentados++;           cout << "El filosofo " << estado.MPI\_SOURCE << " se sienta en la mesa. " << endl << flush;           break;          case etiq\_levantarse:           filosofos\_sentados--;           cout << "El filosofo " << estado.MPI\_SOURCE << " se levanta de la mesa. " << endl << flush;           break;        }     }  }  int main( int argc, char\*\* argv )  {     int id\_propio, num\_procesos\_actual ;     MPI\_Init( &argc, &argv );     MPI\_Comm\_rank( MPI\_COMM\_WORLD, &id\_propio );     MPI\_Comm\_size( MPI\_COMM\_WORLD, &num\_procesos\_actual );     if ( num\_procesos == num\_procesos\_actual )     {        // ejecutar la función correspondiente a 'id\_propio'        if( id\_propio == num\_procesos - 1)           funcion\_camarero();        else if ( id\_propio % 2 == 0 )          // si es par           funcion\_filosofos( id\_propio ); //   es un filósofo        else                               // si es impar           funcion\_tenedores( id\_propio ); //   es un tenedor     }     else     {        if ( id\_propio == 0 ) // solo el primero escribe error, indep. del rol        { cout << "el número de procesos esperados es:    " << num\_procesos << endl << flush               << "el número de procesos en ejecución es: " << num\_procesos\_actual << endl << flush               << "(programa abortado)" << endl << flush ;        }     }     MPI\_Finalize( );     return 0;  }  // --------------------------------------------------------------------- |

. Salida del programa

|  |
| --- |
| Filosofo 0 solicita sentarse  Filosofo 2 solicita sentarse  Filósofo 2 solicita ten. izq.3  Filósofo 2 solicita ten. der.1  Filósofo 2 comienza a comer  Filosofo 4 solicita sentarse  Filósofo 4 solicita ten. izq.5  Filósofo 4 solicita ten. der.3  Filosofo 6 solicita sentarse  Filósofo 6 solicita ten. izq.7  Filósofo 6 solicita ten. der.5  Filosofo 8 solicita sentarse  Filósofo 8 solicita ten. izq.9  Filósofo 8 solicita ten. der.7  Ten. 1 ha sido cogido por filo. 2  Ten. 3 ha sido cogido por filo. 2  Ten. 5 ha sido cogido por filo. 4  Ten. 7 ha sido cogido por filo. 6  Ten. 9 ha sido cogido por filo. 8  El filosofo 2 se sienta en la mesa.  El filosofo 4 se sienta en la mesa.  El filosofo 6 se sienta en la mesa.  El filosofo 8 se sienta en la mesa.  Filósofo 2 suelta ten. izq. 3  Filósofo 2 suelta ten. der. 1  Filosofo 2 solicita levantarse  Filosofo Ten. 1 ha sido liberado por filo. 2  2 comienza a pensar  Ten. 3 ha sido liberado por filo. 2  Ten. 3 ha sido cogido por filo. 4  Filósofo 4 comienza a comer  El filosofo 2 se levanta de la mesa.  El filosofo 0 se sienta en la mesa.  Filósofo 0 solicita ten. izq.1  Filósofo 0 solicita ten. der.9  Ten. 1 ha sido cogido por filo. 0  Filosofo 2 solicita sentarse  Filósofo 4 suelta ten. izq. 5  Filósofo 4 suelta ten. der. 3  Ten. 3 ha sido liberado por filo. 4  Filosofo 4 solicita levantarse  Filosofo 4 comienza a pensar  Ten. 5 ha sido liberado por filo. 4  Ten. 5 ha sido cogido por filo. 6  Filósofo 6 comienza a comer  El filosofo 4 se levanta de la mesa.  El filosofo 2 se sienta en la mesa.  Filósofo 2 solicita ten. izq.3  Filósofo 2 solicita ten. der.1  Ten. 3 ha sido cogido por filo. 2  Filosofo 4 solicita sentarse  Filósofo 6 suelta ten. izq. 7  Filósofo 6 suelta ten. der. 5  Ten. 5 ha sido liberado por filo. 6  Filosofo 6 solicita levantarse  Filosofo 6 comienza a pensar  Ten. 7 ha sido liberado por filo. 6  Ten. 7 ha sido cogido por filo. 8  Filósofo 8 comienza a comer  El filosofo 6 se levanta de la mesa.  El filosofo 4 se sienta en la mesa.  Filósofo 4 solicita ten. izq.5  Filósofo 4 solicita ten. der.3  Ten. 5 ha sido cogido por filo. 4  Filósofo 8 suelta ten. izq. 9  Filósofo 8 suelta ten. der. 7  Filosofo 8 solicita levantarse  Filósofo 0 comienza a comer  Ten. 7 ha sido liberado por filo. 8  Filosofo 8 comienza a pensar  Ten. 9 ha sido liberado por filo. 8  Ten. 9 ha sido cogido por filo. 0  El filosofo 8 se levanta de la mesa.  Filosofo 6 solicita sentarse  Filósofo 6 solicita ten. izq.7  Filósofo 6 solicita ten. der.5  Ten. 7 ha sido cogido por filo. 6  El filosofo 6 se sienta en la mesa.  Filósofo 0 suelta ten. izq. 1  Filósofo 0 suelta ten. der. 9  Filosofo 0 solicita levantarse  Filosofo 0 comienza a pensar  Ten. 1 ha sido liberado por filo. 0  Ten. 1 ha sido cogido por filo. 2  Filósofo 2 comienza a comer  Ten. 9 ha sido liberado por filo. 0  El filosofo 0 se levanta de la mesa.  Filosofo 8 solicita sentarse  Filósofo 8 solicita ten. izq.9  Filósofo 8 solicita ten. der.7  El filosofo 8 se sienta en la mesa.  Ten. 9 ha sido cogido por filo. 8  Filosofo 0 solicita sentarse  Filósofo 2 suelta ten. izq. 3  Filósofo 2 suelta ten. der. 1  Filosofo 2 solicita levantarse  Ten. 1 ha sido liberado por filo. 2  Filosofo 2 comienza a pensar  Ten. 3 ha sido liberado por filo. 2  Ten. 3 ha sido cogido por filo. 4  Filósofo 4 comienza a comer  El filosofo 2 se levanta de la mesa.  El filosofo 0 se sienta en la mesa.  Filósofo 0 solicita ten. izq.1  Filósofo 0 solicita ten. der.9  Ten. 1 ha sido cogido por filo. 0  Filosofo 2 solicita sentarse  Filósofo 4 suelta ten. izq. 5  Filósofo 4 suelta ten. der. 3  Filosofo 4 solicita levantarse  Filosofo 4 comienza a pensar  Filósofo 2 solicita ten. izq.3  Filósofo 2 solicita ten. der.1  Ten. 3 ha sido liberado por filo. 4  Ten. 3 ha sido cogido por filo. 2  Ten. 5 ha sido liberado por filo. 4  Ten. 5 ha sido cogido por filo. 6  Filósofo 6 comienza a comer  El filosofo 4 se levanta de la mesa.  El filosofo 2 se sienta en la mesa.  Filósofo 6 suelta ten. izq. 7  Filósofo 6 suelta ten. der. 5  Filosofo 6 solicita levantarse |