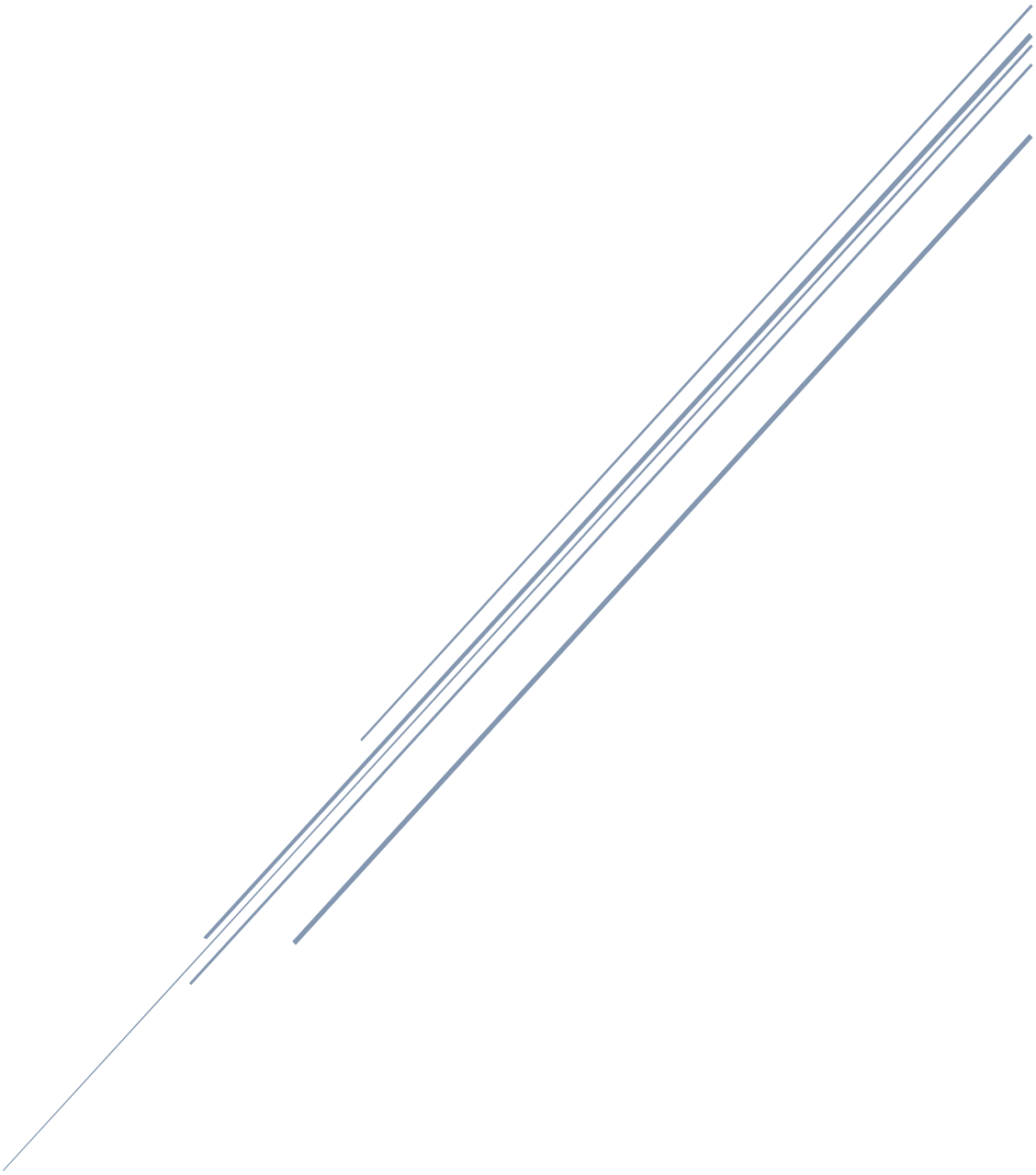
PORTAFOLIOS PRÁCTICA 3 SCD

Paso de mensajes con MPI

Víctor José Rubia López



UGR ETSIIT
2ºB (B2)

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.

1. 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 e
sperado
```

```
   num_items               = n_productores * n_consumidores, // numero de items producidos o c
onsumidos
   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 << fl
ush;
      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, &es
tado );

      // 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_WORL
D);
            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;
}
// ---------------------------------------------------------------------
```

2. 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.

3. 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, &es
tado);
     // ..... 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 << flu
sh;

     // ..... recibir liberación de filósofo 'id_filosofo' (completar)
     MPI_Recv( &valor, 1, MPI_INT, id_filosofo, MPI_ANY_TAG, MPI_COMM_WORLD, &estad
o);
     cout <<"Ten. "<< id<< " ha sido liberado por filo. " <<id_filosofo <<endl << f
lush ;
  }
}
// ---------------------------------------------------------------------

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;
}

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

4. 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.

5. 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, &est
ado);
     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;
}

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

6. 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.

7. 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 << flu
sh;
     // ..... recibir liberación de filósofo 'id_filosofo' (completar)
       MPI_Recv( &valor, 1, MPI_INT, id_filosofo, MPI_ANY_TAG, MPI_COMM_WORLD, &est
ado);
     cout <<"Ten. "<< id<< " ha sido liberado por filo. " <<id_filosofo <<endl << f
lush ;
  }
}
// ----------------------------------------------------------------------

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, &e
stado);

      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;
}

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

8. 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
```