

PRACTICA 3

IMPLEMENTACIÓN DE ALGORITMOS DISTRIBUIDOS CON MPI

PRODUCTOR – CONSUMIDOR (MÚLTIPLES PRODUCTORES Y CONSUMIDORES)

Ejercicio 1: Describir qué cambios se han realizado sobre el programa de partida y el propósito de dichos cambios.

Primero he realizado cambios en las variables iniciales para incluir múltiples productores y múltiples consumidores y además he añadido la gestión por etiquetas.

```
const int
    num_productores      = 4,
    num_consumidores     = 5,
    id_buffer            = num_productores,
    etiqueta_cons        = 1,
    etiqueta_prod        = 0,
    num_procesos_esperado = num_productores + num_consumidores + 1, // +1 es el buffer
    num_items            = num_productores * num_consumidores, // multiplo de num_productores y num_consumidores
    tam_vector           = 10;
```

He establecido un número de productores y consumidores como se nos indica. La `id_buffer` coincide con el numero de productores. Las dos variables etiquetas (`etiqueta_cons` y `etiqueta_prod`) son para identificar si el proceso que envía el mensaje es un consumidor o un productor. El número de procesos esperados son los procesos consumidores (5 en este caso), procesos productores (4 en este caso) y el buffer. El número de items es el producto de `num_productores` y consumidores ya que este número debe de ser un múltimo de ambos.

El siguiente cambio lo he realizado en la función `producir` ya que hay varios procesos que la usan.

```
int producir(int id_prod)
{
    static int contador = 0 ;
    int nuevo_valor = ((num_items/num_productores) * id_prod) + contador;
    sleep_for( milliseconds( aleatorio<10,100>()) );
    contador++;
    cout << "El productor " << id_prod << " ha producido valor " << nuevo_valor << endl << flush;
    return nuevo_valor ;
}
```

Primero, a la función `producir` se le pasa la `id` del productor para poder calcular un nuevo valor que sirva para que cada productor use su contador. El nuevo valor se calcula: $((\text{num_items}/\text{num_productores}) * \text{id_prod})$, de esta manera cada productor produce un rango distinto de valores.

Tambien modificamos la funcion productor y la función consumidor.

```
void funcion_productor(int id_prod)
{
    for ( unsigned int i= 0 ; i < num_items/num_productores ; i++ )
    {
        // producir valor para el productor indicado
        int valor_prod = producir(id_prod);
        // enviar valor
        cout << "El productor " << id_prod << " va a enviar valor " << valor_prod << endl << flush;
        MPI_Ssend( &valor_prod, 1, MPI_INT, id_buffer, etiqueta_prod, MPI_COMM_WORLD );
    }
}

void funcion_consumidor(int id_cons)
{
    int          peticion,
               valor_rec = 1 ;
    MPI_Status  estado ;

    for( unsigned int i=0 ; i < num_items/num_consumidores; i++ )
    {
        MPI_Ssend( &peticion, 1, MPI_INT, id_buffer, etiqueta_cons, MPI_COMM_WORLD);
        MPI_Recv ( &valor_rec, 1, MPI_INT, id_buffer, etiqueta_cons, MPI_COMM_WORLD,&estado );
        cout << "          El consumidor " << id_cons << " ha recibido valor " << valor_rec << endl << flush ;
        consumir( valor_rec, id_cons);
    }
}
```

El rango del bucle alcanza el rango del productor y del consumidor de manera que no es la totalidad de los valores. Y ahora cuando se envía el valor a través de Ssend, se realiza a través de la etiqueta de identificación, etiqueta_prod para los productores y etiqueta_cons para los consumidores.

Ahora modificamos la función de buffer.

```
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
    etiqueta_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
            etiqueta_aceptable = etiqueta_prod ; // $$$ solo prod.
        else if ( num_celdas_ocupadas == tam_vector ) // si buffer lleno
            etiqueta_aceptable = etiqueta_cons ; // $$$ solo cons.
        else // si no vacío ni lleno
            etiqueta_aceptable = MPI_ANY_TAG ; // $$$ cualquiera

        // 2. recibir un mensaje del emisor o emisores aceptables

        MPI_Recv( &valor, 1, MPI_INT, MPI_ANY_SOURCE, etiqueta_aceptable, MPI_COMM_WORLD, &estado );

        // 3. procesar el mensaje recibido

        switch( estado.MPI_TAG ) // leer emisor del mensaje en metadatos
        {
            case etiqueta_prod: // 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 ;
                break;

            case etiqueta_cons: // si ha sido el consumidor: extraer y enviarle
                valor = buffer[primera_ocupada] ;
                primera_ocupada = (primera_ocupada+1) % tam_vector ;
                num_celdas_ocupadas-- ;
                cout << "Buffer va a enviar valor " << valor << endl ;
                MPI_Ssend( &valor, 1, MPI_INT, estado.MPI_SOURCE, etiqueta_cons, MPI_COMM_WORLD);
                break;
        }
    }
}
```

Primero hemos borrado de las variables iniciales el `id_emisor_aceptable` y hemos puesto `etiqueta_aceptable` ya que al haber varios productores y varios consumidores, se maneja mejor a través de etiquetas.

La variable `etiqueta_aceptable` contendrá `etiqueta_prod` en el caso que el buffer se encuentre vacío, `etiqueta_cons` en el caso que el buffer se encuentre completo y `MPI_ANY_TAG` para recibir mensajes de productores o consumidores.

Una vez realizado esto, si la etiqueta que se ha enviado pertenece a un productor se guardará en el buffer el valor enviado y si la etiqueta pertenece a un consumidor se le envía un valor obtenido del buffer.

Ejercicio 2: Incluir el código fuente completo de la solución adoptada.

```
#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 ;

const int
    num_productores    = 4,
    num_consumidores    = 5,
    id_buffer          = num_productores,
    etiqueta_cons       = 1,
    etiqueta_prod       = 0,
    num_procesos_esperado = num_productores + num_consumidores + 1 , // +1 es el
buffer
    num_items           = num_productores * num_consumidores, // multiplo de
num_productores y num_consumidores
    tam_vector          = 10;

//*****
// 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 );
}
// -----
// producimos un numero aleatorio para cada productor utilizando cada uno un rango
// -----
int producir(int id_prod)
{
    static int contador = 0 ;
    int nuevo_valor = ((num_items/num_productores) * id_prod) + contador;
    sleep_for( milliseconds( aleatorio<10,100>() ) );
    contador++;
    cout << "El productor " << id_prod << " ha producido valor " << nuevo_valor << endl <<
flush;
    return nuevo_valor ;
}
// -----
```

```

void funcion_productor(int id_prod)
{
    for ( unsigned int i= 0 ; i < num_items/num_productores ; i++ )
    {
        // producir valor para el productor indicado
        int valor_prod = producir(id_prod);
        // enviar valor
        cout << "El productor " << id_prod << " va a enviar valor " << valor_prod << endl <<
flush;
        MPI_Ssend(  &valor_prod,  1,  MPI_INT,  id_buffer,  etiqueta_prod,
MPI_COMM_WORLD );
    }
}
// -----

void consumir( int valor_cons, int id_cons)
{
    // espera bloqueada
    sleep_for( milliseconds( aleatorio<110,200>()) );
    cout << "          El consumidor " << id_cons << " ha consumido valor " << valor_cons
<< endl << flush ;
}
// -----

void funcion_consumidor(int id_cons)
{
    int      peticion,
            valor_rec = 1 ;
    MPI_Status estado ;

    for( unsigned int i=0 ; i < num_items/num_consumidores; i++ )
    {
        MPI_Ssend( &peticion, 1, MPI_INT, id_buffer, etiqueta_cons, MPI_COMM_WORLD);
        MPI_Recv (  &valor_rec,  1,  MPI_INT,  id_buffer,  etiqueta_cons,
MPI_COMM_WORLD,&estado );
        cout << "          El consumidor " << id_cons << " ha recibido valor " << valor_rec <<
endl << flush ;
        consumir( valor_rec, id_cons);
    }
}
// -----

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
            etiqueta_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
        etiqueta_aceptable = etiqueta_prod ;    // $~~~$ solo prod.
    else if ( num_celdas_ocupadas == tam_vector ) // si buffer lleno
        etiqueta_aceptable = etiqueta_cons ;    // $~~~$ solo cons.
    else                                     // si no vacío ni lleno
        etiqueta_aceptable = MPI_ANY_TAG ;    // $~~~$ cualquiera

    // 2. recibir un mensaje del emisor o emisores aceptables

    MPI_Recv( &valor, 1, MPI_INT, MPI_ANY_SOURCE, etiqueta_aceptable,
MPI_COMM_WORLD, &estado );

    // 3. procesar el mensaje recibido

    switch( estado.MPI_TAG ) // leer emisor del mensaje en metadatos
    {
        case etiqueta_prod: // 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 ;
            break;

            case etiqueta_cons: // si ha sido el consumidor: extraer y enviarle
                valor = buffer[primera_ocupada] ;
                primera_ocupada = (primera_ocupada+1) % tam_vector ;
                num_celdas_ocupadas-- ;
                cout << "Buffer va a enviar valor " << valor << endl ;
                MPI_Ssend( &valor, 1, MPI_INT, estado.MPI_SOURCE, etiqueta_cons,
MPI_COMM_WORLD);
                break;
    }
}

// -----

int main( int argc, char *argv[] )
{
    int id_propio, num_procesos_actual;

    // inicializar MPI, leer identif. de proceso y número 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 )
    {

```

```

// ejecutar la operación apropiada a 'id_propio'
if ( id_propio < num_productores)
    funcion_productor(id_propio);
else if ( id_propio == id_buffer )
    funcion_buffer();
else
    funcion_consumidor(id_propio);
}
else
{
    if ( id_propio == 0 ) // solo el primero escribe error, indep. del rol
    { 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 ;
    }
}

// al terminar el proceso, finalizar MPI
MPI_Finalize( );
return 0;
}

```

Ejercicio 3: Incluir un listado parcial de la salida del programa.

```

El productor 1 ha producido valor 5
El productor 1 va a enviar valor 5
Buffer ha recibido valor 5
Buffer va a enviar valor 5
    El consumidor 5 ha recibido valor 5
El productor 2 ha producido valor 10
El productor 2 va a enviar valor 10
Buffer ha recibido valor 10
Buffer va a enviar valor 10
    El consumidor 6 ha recibido valor 10
El productor 1 ha producido valor 6
El productor 1 va a enviar valor 6
Buffer ha recibido valor 6
Buffer va a enviar valor 6
    El consumidor 7 ha recibido valor 6
El productor 0 ha producido valor 0
El productor 0 va a enviar valor 0
Buffer ha recibido valor 0
Buffer va a enviar valor 0
    El consumidor 8 ha recibido valor 0
El productor 2 ha producido valor 11
El productor 2 va a enviar valor 11
Buffer ha recibido valor 11
Buffer va a enviar valor 11
    El consumidor 9 ha recibido valor 11
El productor 3 ha producido valor 15

```

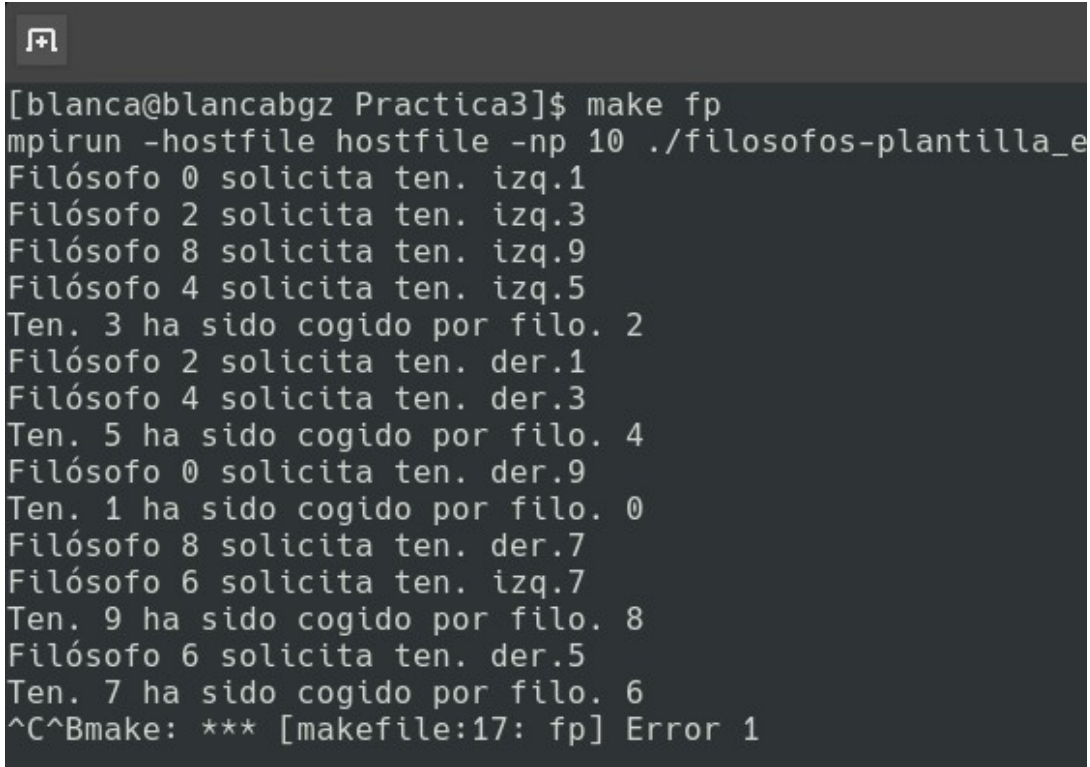

El productor 3 va a enviar valor 15
Buffer ha recibido valor 15
El productor 1 ha producido valor 7
El productor 1 va a enviar valor 7
Buffer ha recibido valor 7
El productor 0 ha producido valor 1
El productor 0 va a enviar valor 1
Buffer ha recibido valor 1
El productor 2 ha producido valor 12
El productor 2 va a enviar valor 12
Buffer ha recibido valor 12
El productor 2 ha producido valor 13
El productor 2 va a enviar valor 13
Buffer ha recibido valor 13
El productor 3 ha producido valor 16
El productor 3 va a enviar valor 16
Buffer ha recibido valor 16
El productor 0 ha producido valor 2
El productor 0 va a enviar valor 2
Buffer ha recibido valor 2
 El consumidor 5 ha consumido valor 5
 El consumidor 5 ha recibido valor 15
Buffer va a enviar valor 15
El productor 1 ha producido valor 8
El productor 1 va a enviar valor 8
Buffer ha recibido valor 8
El productor 2 ha producido valor 14
El productor 2 va a enviar valor 14
Buffer ha recibido valor 14
 El consumidor 9 ha consumido valor 11
 El consumidor 9 ha recibido valor 7
Buffer va a enviar valor 7
 El consumidor 6 ha consumido valor 10
 El consumidor 6 ha recibido valor 1
Buffer va a enviar valor 1
El productor 0 ha producido valor 3
El productor 0 va a enviar valor 3
Buffer ha recibido valor 3
El productor 3 ha producido valor 17
El productor 3 va a enviar valor 17
Buffer ha recibido valor 17
 El consumidor 7 ha consumido valor 6
 El consumidor 7 ha recibido valor 12
Buffer va a enviar valor 12
 El consumidor 8 ha consumido valor 0
 El consumidor 8 ha recibido valor 13
Buffer va a enviar valor 13
El productor 1 ha producido valor 9
El productor 1 va a enviar valor 9
Buffer ha recibido valor 9
El productor 0 ha producido valor 4
El productor 0 va a enviar valor 4

Buffer ha recibido valor 4
El productor 3 ha producido valor 18
El productor 3 va a enviar valor 18
Buffer ha recibido valor 18
El productor 3 ha producido valor 19
El productor 3 va a enviar valor 19
Buffer ha recibido valor 19
 El consumidor 5 ha consumido valor 15
 El consumidor 5 ha recibido valor 16
Buffer va a enviar valor 16
 El consumidor 6 ha consumido valor 1
 El consumidor 6 ha recibido valor 2
Buffer va a enviar valor 2
 El consumidor 8 ha consumido valor 13
 El consumidor 8 ha recibido valor 8
Buffer va a enviar valor 8
 El consumidor 7 ha consumido valor 12
 El consumidor 7 ha recibido valor 14
Buffer va a enviar valor 14
 El consumidor 9 ha consumido valor 7
 El consumidor 9 ha recibido valor 3
Buffer va a enviar valor 3
 El consumidor 5 ha consumido valor 16
 El consumidor 5 ha recibido valor 17
Buffer va a enviar valor 17
 El consumidor 8 ha consumido valor 8
 El consumidor 8 ha recibido valor 9
Buffer va a enviar valor 9
 El consumidor 6 ha consumido valor 2
 El consumidor 6 ha recibido valor 4
Buffer va a enviar valor 4
 El consumidor 7 ha consumido valor 14
 El consumidor 7 ha recibido valor 18
Buffer va a enviar valor 18
 El consumidor 9 ha consumido valor 3
 El consumidor 9 ha recibido valor 19
Buffer va a enviar valor 19
 El consumidor 8 ha consumido valor 9
 El consumidor 5 ha consumido valor 17
 El consumidor 9 ha consumido valor 19
 El consumidor 6 ha consumido valor 4
 El consumidor 7 ha consumido valor 18

FILÓSOFOS: SOLUCIONES CON INTERBLOQUEO Y SIN INTERBLOQUEO

Ejercicio 1: Describe los aspectos más destacados de tu solución al problema de los filósofos, la situación que conduce al interbloqueo y tu solución al problema del interbloqueo.

Se produce interbloqueo ya que si todos los filósofos intentan coger el tenedor de la izquierda, todos los filósofos conseguirían tener los tenedores sin problema. Pero a la hora de coger el tenedor de la derecha, quedarían todos bloqueado ya que no hay tenedores disponibles.

A terminal window with a dark background and light text. It shows the execution of a program with 10 philosophers. The output shows philosophers 0, 2, 4, 6, and 8 successfully picking up their left forks. However, philosophers 1, 3, 5, 7, and 9 are stuck because their right forks (which are the left forks of the previous philosopher) are already held by someone. The terminal ends with a 'make: *** [makefile:17: fp] Error 1' message, indicating a deadlock.

```
[blanca@blancabgz Practica3]$ make fp
mpirun -hostfile hostfile -np 10 ./filosofos-plantilla_e
Filósofo 0 solicita ten. izq.1
Filósofo 2 solicita ten. izq.3
Filósofo 8 solicita ten. izq.9
Filósofo 4 solicita ten. izq.5
Ten. 3 ha sido cogido por filo. 2
Filósofo 2 solicita ten. der.1
Filósofo 4 solicita ten. der.3
Ten. 5 ha sido cogido por filo. 4
Filósofo 0 solicita ten. der.9
Ten. 1 ha sido cogido por filo. 0
Filósofo 8 solicita ten. der.7
Filósofo 6 solicita ten. izq.7
Ten. 9 ha sido cogido por filo. 8
Filósofo 6 solicita ten. der.5
Ten. 7 ha sido cogido por filo. 6
^C^Bmake: *** [makefile:17: fp] Error 1
```

Para solucionar este interbloqueo, vamos a hacer que un número de filósofos cojan primero el tenedor de la izquierda y el resto coja el tenedor de la derecha.

```

void funcion_filosofos( int id )
{
    int id_ten_izq, //id. tenedor izq.
        id_ten_der, //id. tenedor der
        peticion;

    if(id > num_filosofos){
        id_ten_izq = (id+1) % num_procesos, //id. tenedor izq.
        id_ten_der = (id+num_procesos-1) % num_procesos; //id. tenedor der
    }else{
        id_ten_izq = (id+num_procesos-1) % num_procesos,
        id_ten_der = (id+1) % num_procesos;
    }
}

```

El criterio que utilizo para seleccionar a los filósofos es que si el id es menor que el numero total de filósofos, solicita primero el tenedor de la izquierda. En caso contrario solicita primero el tenedor de la derecha.

Ejercicio 2: Incluye el código fuente completo de la solución adoptada para evitar la posibilidad de interbloqueo.

```

#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. tenedor izq.
        id_ten_der, //id. tenedor der
        peticion;

    if(id > num_filosofos){
        id_ten_izq = (id+1) % num_procesos, //id. tenedor izq.
        id_ten_der = (id+num_procesos-1) % num_procesos; //id. tenedor der
    }else{
        id_ten_izq = (id+num_procesos-1) % num_procesos,
        id_ten_der = (id+1) % num_procesos;
    }

    while ( true )
    {
        cout <<"Filósofo " <<id << " solicita ten. izq." <<id_ten_izq <<endl;
        MPI_Ssend(&peticion,1,MPI_INT,id_ten_izq,0,MPI_COMM_WORLD);

        cout <<"Filósofo " <<id <<" solicita ten. der." <<id_ten_der <<endl;
        MPI_Ssend(&peticion,1,MPI_INT,id_ten_der,0,MPI_COMM_WORLD);

        cout <<"Filósofo " <<id <<" comienza a comer" <<endl ;
        sleep_for( milliseconds( aleatorio<10,100>() ) );

        cout <<"Filósofo " <<id <<" suelta ten. izq. " <<id_ten_izq <<endl;
        MPI_Ssend(&peticion,1,MPI_INT,id_ten_izq,0,MPI_COMM_WORLD);

        cout<< "Filósofo " <<id <<" suelta ten. der. " <<id_ten_der <<endl;
        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 )
    {
        MPI_Recv(&valor,1,MPI_INT,MPI_ANY_SOURCE,0,MPI_COMM_WORLD,&estado); //
        recibe peticion filosofoso
        id_filosofo = estado.MPI_SOURCE; // guarda el id_filosofo
        cout <<"Ten. " <<id <<" ha sido cogido por filo. " <<id_filosofo <<endl;

        MPI_Recv(&valor,1,MPI_INT,id_filosofo,0,MPI_COMM_WORLD,&estado); // recibir
        liberación de filósofo 'id_filosofo'
        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;
}

// -----

```

Ejercicio 3: Incluye un listado parcial de la salida de este programa.

```

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

```

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

FILÓSOFOS: CENA DE LOS FILÓSOFOS CON CAMARERO

Ejercicio 1: Describir la solución al problema de los filósofos con camarero central.

Para esta solución primero se le mandarán los mensajes a el camarero para poder sentarse y levantarse de la mesa.

```
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 <<"Filósofo " <<id << " solicita sentarse" << endl;
        MPI_Ssend(&peticion,1,MPI_INT,id_camarero,etiq_sentarse,MPI_COMM_WORLD);

        cout <<"Filósofo " <<id << " solicita ten. izq." <<id_ten_izq <<endl;
        MPI_Ssend(&peticion,1,MPI_INT,id_ten_izq,0,MPI_COMM_WORLD);

        cout <<"Filósofo " <<id <<" solicita ten. der." <<id_ten_der <<endl;
        MPI_Ssend(&peticion,1,MPI_INT,id_ten_der,0,MPI_COMM_WORLD);

        cout <<"Filósofo " <<id <<" comienza a comer" <<endl ;
        sleep_for( milliseconds( aleatorio<10,100>() ) );

        cout <<"Filósofo " <<id <<" suelta ten. izq. " <<id_ten_izq <<endl;
        MPI_Ssend(&peticion,1,MPI_INT,id_ten_izq,0,MPI_COMM_WORLD);

        cout<< "Filósofo " <<id <<" suelta ten. der. " <<id_ten_der <<endl;
        MPI_Ssend(&peticion,1,MPI_INT,id_ten_der,0,MPI_COMM_WORLD);

        cout <<"Filósofo " <<id << " solicita levantarse" << endl;
        MPI_Ssend(&peticion,1,MPI_INT,id_camarero,etiq_levantarse,MPI_COMM_WORLD);

        cout << "Filosofo " << id << " comienza a pensar" << endl;
        sleep_for( milliseconds( aleatorio<10,100>() ) );

    }
}
```

Se añaden dos mensajes más:

1. El filósofo quiere sentarse a comer en la mesa, envía un mensaje al camarero con la etiqueta `etiq_sentarse`.
2. El filósofo quiere levantarse de la mesa después de haber liberado sus dos tenedores, envía un mensaje al camarero con la etiqueta `etiq_levantarse`.

Ahora vamos a incluir una función llamada `funcion_camarero`.

```
void funcion_camarero(){
    MPI_Status estado;
    int valor,
        etiqueta_aceptable,
        mesa = 0;
    while(true){
        if(mesa == 4){
            etiqueta_aceptable = etiq_levantarse;
        }else if(mesa == 0){
            etiqueta_aceptable = etiq_sentarse;
        }else{
            etiqueta_aceptable = MPI_ANY_TAG;
        }

        MPI_Recv(&valor,1,MPI_INT,MPI_ANY_SOURCE,etiqueta_aceptable,MPI_COMM_WORLD,&estado);

        switch( estado.MPI_TAG ){
            case etiq_sentarse:
                mesa++;
                cout << "El filosofo " << estado.MPI_SOURCE << " se ha sentado a la mesa " << endl;
                break;
            case etiq_levantarse:
                mesa--;
                cout << "El filosofo " << estado.MPI_SOURCE << " se ha levantado de la mesa " << endl;
                break;
        }
        cout << mesa << endl;
    }
}
```

El camarero controla que en la mesa no se pueden sentar mas de 4 filósofos.

Entonces se pueden dar estas situaciones:

1. Si la mesa esta completa, solo recibe los mensajes donde los filósofos piden levantarse.
2. Si la mesa esta vacía, solo recibe los mensajes de los filósofos que solicitan sentarse en la mesa.
3. Si no se da ninguna de estas dos condiciones, se aceptan mensajes de levantarse y sentarse.

Ejercicio 2: Incluir el código fuente completo de la solución adoptada.

```
#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_sentarse = 1,
    etiq_levantarse = 2;

//*****
// 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 <<"Filósofo " <<id << " solicita sentarse" << endl;
        MPI_Ssend(&peticion,1,MPI_INT,id_camarero,etiq_sentarse,MPI_COMM_WORLD);

        cout <<"Filósofo " <<id << " solicita ten. izq." <<id_ten_izq <<endl;
        MPI_Ssend(&peticion,1,MPI_INT,id_ten_izq,0,MPI_COMM_WORLD);

        cout <<"Filósofo " <<id <<" solicita ten. der." <<id_ten_der <<endl;
```

```

MPI_Ssend(&peticion,1,MPI_INT,id_ten_der,0,MPI_COMM_WORLD);

cout <<"Filósofo " <<id <<" comienza a comer" <<endl ;
sleep_for( milliseconds( aleatorio<10,100>() ) );

cout <<"Filósofo " <<id <<" suelta ten. izq. " <<id_ten_izq <<endl;
MPI_Ssend(&peticion,1,MPI_INT,id_ten_izq,0,MPI_COMM_WORLD);

cout<< "Filósofo " <<id <<" suelta ten. der. " <<id_ten_der <<endl;
MPI_Ssend(&peticion,1,MPI_INT,id_ten_der,0,MPI_COMM_WORLD);

cout <<"Filósofo " <<id << " solicita levantarse" << endl;
MPI_Ssend(&peticion,1,MPI_INT,id_camarero,etiq_levantarse,MPI_COMM_WORLD);

cout << "Filosofo " << id << " comienza a pensar" << endl;
sleep_for( milliseconds( aleatorio<10,100>() ) );
}
}

void funcion_camarero(){
    MPI_Status estado;
    int valor,
        etiqueta_aceptable,
        mesa = 0;
    while(true){
        if(mesa == 4){
            etiqueta_aceptable = etiq_levantarse;
        }else if(mesa == 0){
            etiqueta_aceptable = etiq_sentarse;
        }else{
            etiqueta_aceptable = MPI_ANY_TAG;
        }

        MPI_Recv(&valor,1,MPI_INT,MPI_ANY_SOURCE,etiqueta_aceptable,MPI_COMM_WORLD,&estado);

        switch( estado.MPI_TAG ){
            case etiq_sentarse:
                mesa++;
                cout << "El filosofo " << estado.MPI_SOURCE << " se ha sentado a la mesa " <<
endl;
                break;
            case etiq_levantarse:
                mesa--;
                cout << "El filosofo " << estado.MPI_SOURCE << " se ha levantado de la mesa " <<
endl;
                break;
        }
        cout << mesa << endl;
    }
}
// -----

```

```

void funcion_tenedores( int id )
{
    int valor, id_filosofo ; // valor recibido, identificador del filósofo
    MPI_Status estado ;      // metadatos de las dos recepciones

    while ( true )
    {
        MPI_Recv(&valor,1,MPI_INT,MPI_ANY_SOURCE,0,MPI_COMM_WORLD,&estado); //
recibe petición filósofo
        id_filosofo = estado.MPI_SOURCE; // guarda el id_filosofo
        cout <<"Ten. " <<id <<" ha sido cogido por filo. " <<id_filosofo <<endl;

        MPI_Recv(&valor,1,MPI_INT,id_filosofo,0,MPI_COMM_WORLD,&estado); // recibir
liberación de filósofo 'id_filosofo'
        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 && id_propio != num_procesos -1) // si es par
            funcion_filosofos( id_propio ); // es un filósofo
        else if(id_propio %2 != 0) // si es impar
            funcion_tenedores( id_propio ); // es un tenedor
        else
            funcion_camarero();
    }
    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;
}

```

Ejercicio 3: Incluir un listado parcial de la salida del programa.

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

Filosofo 6 comienza a pensar
Ten. 5 ha sido liberado por filo. 6
El filosofo 6 se ha levantado de la mesa
3
El filosofo 4 se ha sentado a la mesa
4
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
Ten. 9 ha sido liberado por filo. 8
Ten. 9 ha sido cogido por filo. 0
Ten. 7 ha sido liberado por filo. 8
Filósofo 8 suelta ten. der. 7
Filósofo 8 solicita levantarse
Filósofo 0 comienza a comer
El filosofo 8 se ha levantado de la mesa
3
Filosofo 8 comienza a pensar