

Ejercicio 1

Productor-consumidor en MPI

1.1 Ejercicio propuesto

Extender el programa MPI que implementa el productor-consumidor con buffer acotado para que el proceso buffer dé servicio a 5 productores y 4 consumidores. Para ello, se lanzarán 10 procesos y asumiremos que los procesos 0, ..., 4 son productores, el proceso *Buffer* el proceso 5 y el resto de procesos en el comunicador universal (6, ..., 9) son consumidores.

1.2 Solución

Se harán los siguientes cambios sobre el programa *prodcons2.cpp*:

- *Variables globales*: Las variables *Productor*, *Consumidor* y *Buffer* se usarán como etiquetas para el envío de mensajes. La variable *Buffer* se usará también para señalar el identificador del proceso buffer. Para el envío de mensajes por parte del productor o del consumidor se usará la etiqueta del proceso correspondiente.
- *Cambios en la función productor*: Se cambiará tan sólo la implementación del bucle. Se tienen que repartir las iteraciones entre los productores, por lo que el índice i se inicializará al número del identificador del proceso y se sumará el número de productores para producir el siguiente valor. El número de productores se corresponde con el identificador del proceso *Buffer* ya que es el primer proceso que no es un productor (actúa como "barrera").
- *Cambios en la función consumidor*: Al igual que en la función productor, hay que repartir las iteraciones, a cada consumidor le corresponderán $ITERS/(SIZE-Buffer-1)$, ya que $SIZE-Buffer-1$ es el número de consumidores.
- *Cambios en la función buffer*: Se esperará a recibir cualquier mensaje. Las comprobaciones se harán en función de la etiqueta del mensaje en lugar de la fuente: si la etiqueta (y no la fuente, como en la versión anterior), es *Productor*, la rama será 0, y si es *Consumidor*, será 1. Si se trata de la rama 0, se espera recibir un mensaje con el valor producido, de cualquier fuente, con la etiqueta productor. Si es la rama 1, se esperará a recibir una petición de cualquier fuente con la etiqueta consumidor, y se enviará el valor al proceso del cual ha recibido la petición. Los mensajes enviados por el Buffer tienen etiqueta 0.

Se cambiará también la inicialización de los procesos, para verificar la operación que debe ejecutarse se comprobará si el identificador del proceso es menor, igual, o mayor que el identificador del proceso Buffer y se ejecutarán, respectivamente, las funciones productor,buffer y consumidor.

1.3 Ejecución del programa

En la imagen 1.1 se puede ver la ejecución del programa descrito.

```
MacBook-Air-de-Montse:practica3 montse$ mpirun -np 10 prodcons
Productor 3 produce valor 3
Productor 0 produce valor 0
Productor 2 produce valor 2
Productor 1 produce valor 1
Productor 4 produce valor 4
Productor 0 produce valor 5
Buffer recibe 0 de Productor 0.
Buffer recibe 3 de Productor 3.
Consumidor 7 quiere leer.
Productor 3 produce valor 8
Consumidor 7 recibe valor 3 de Buffer
Buffer envía 3 a Consumidor 7.
Consumidor 9 quiere leer.
Buffer envía 0 a Consumidor 9.
Consumidor 9 recibe valor 0 de Buffer
Buffer recibe 1 de Productor 1.
Buffer recibe 4 de Productor 4.
Consumidor 8 quiere leer.
Productor 1 produce valor 6
Productor 4 produce valor 9
Consumidor 8 recibe valor 4 de Buffer
Buffer envía 4 a Consumidor 8.
Buffer recibe 2 de Productor 2.
Productor 2 produce valor 7
Consumidor 6 quiere leer.
Buffer envía 2 a Consumidor 6.
Consumidor 6 recibe valor 2 de Buffer
Productor 0 produce valor 10
Buffer recibe 5 de Productor 0.
Buffer recibe 8 de Productor 3.
Productor 3 produce valor 13
Buffer recibe 9 de Productor 4.
Buffer recibe 7 de Productor 2.
Productor 2 produce valor 12
Productor 4 produce valor 14
Consumidor 7 quiere leer.
Consumidor 7 recibe valor 7 de Buffer
Buffer envía 7 a Consumidor 7.
Buffer recibe 14 de Productor 4.
Productor 4 produce valor 19
Consumidor 6 quiere leer.
Buffer envía 14 a Consumidor 6.
Consumidor 6 recibe valor 14 de Buffer
Consumidor 9 quiere leer.
Consumidor 9 recibe valor 9 de Buffer
```

Captura de pantalla 1.1: Listado parcial de la salida del programa *prodcons2.cpp*

1.4 Código fuente

```

#include <mpi.h>
#include <iostream>
#include <math.h>
#include <time.h>           // incluye "time"
#include <unistd.h>         // incluye "usleep"
#include <stdlib.h>         // incluye "rand" y "srand"

#define Productor      0 //fuente si es el productor
#define Buffer          5 //tiene que ser el proceso 5
#define Consumidor     1 //fuente si es el Consumidor
#define ITTERS         20
#define TAM             5
#define SIZ             10 //número de procesos
/* Plantilla colores*/
#define GREEN           "\033[0;32m"
#define BLUE            "\033[0;34m"
#define CYAN            "\033[0;36m"
#define RED             "\033[0;31m"
#define PURPLE          "\033[0;35m"
#define GRAY            "\033[0;37m"
#define DARK_GRAY       "\033[0;30m"
#define YELLOW          "\033[0;33m"
#define WHITE           "\033[0;37m"
#define DEFAULT         "\033[0m"

using namespace std;

// -----

void productor(int num_productor)
{
    int value ;

    for ( unsigned int i=num_productor; i < ITTERS; i+=Buffer ){
        value = i ;
        cout << CYAN << "Productor_" << num_productor << "_produce_valor_" << value
        << endl << DEFAULT << flush;
        usleep( 1000U * (100U+(rand()%900U)) ); //bloqueo aleatorio
        //enviar valor, 1 elemento, tipo int, al buffer, con etiqueta indice, dentro
        //del comunicador universal
        MPI_Ssend( &value, 1, MPI_INT, Buffer, Productor, MPLCOMM_WORLD );
    }
}

// -----

void buffer()
{
    int          value[TAM] ,
               peticion ,
               pos  = 0,
               rama ;

    MPI_Status status ;

    for( unsigned int i=0 ; i < ITTERS*2 ; i++ )
    {
        if ( pos==0 )

```

```

        rama = 0 ;
    else if (pos==TAM)
        rama = 1 ;
    else
    {
        MPI.Probe( MPLANY_SOURCE, MPLANY_TAG, MPLCOMM_WORLD, &status );
        if ( status.MPLTAG == Productor)
            rama = 0 ;
        else if(status.MPLTAG == Consumidor)
            rama = 1 ;
    }
    switch(rama)
    {
        case 0:
            //si es el productor,...
            //sintaxis: (buf, num, datatype, source, tag, comm, status)
            MPI.Recv( &value[pos], 1, MPI.INT, MPLANY_SOURCE, Productor, MPLCOMM_WORLD,
                &status );
            cout << PURPLE << "Buffer_recibe_" << value[pos]
                << "_de_Productor_" << status.MPLSOURCE << "." << endl
                << DEFAULT << flush;
            pos++;
            break;
        case 1:
            //Si es el consumidor,...
            //recibir
            MPI.Recv( &peticion, 1, MPI.INT, MPLANY_SOURCE, Consumidor, MPLCOMM_WORLD,
                &status );
            cout << PURPLE "Consumidor_" << status.MPLSOURCE
                << "_quiere_leer_" << endl << DEFAULT << flush;
            //sintaxis: (buf, num, datatype, dest, tag, comm)
            MPI.Ssend( &value[pos-1], 1, MPI.INT, status.MPLSOURCE, 0, MPLCOMM_WORLD);
            cout << PURPLE "Buffer_envía_" << value[pos-1]
                << "_a_Consumidor_" << status.MPLSOURCE << "." << endl
                << DEFAULT << flush;
            pos--;
            break;
    }
}
}

// -----

void consumidor(int num_cons)
{
    int          value,
                peticion = 1 ;
    float        raiz ;
    MPI_Status   status ;

    for (unsigned int i=0;i<ITERS/(SIZ-Buffer-1);i++)
    {
        MPI.Ssend( &peticion, 1, MPI.INT, Buffer, Consumidor, MPLCOMM_WORLD );
        MPI.Recv ( &value, 1, MPI.INT, Buffer, 0, MPLCOMM_WORLD,&status );
        cout << YELLOW << "Consumidor_" << num_cons << "_recibe_valor_"
            << value << "_de_Buffer_" << endl << DEFAULT << flush;

        // espera bloqueado durante un intervalo de tiempo aleatorio
        // (entre una décima de segundo y un segundo)
    }
}

```

```
        usleep( 1000U * (100U+(rand()%900U)) );

        raiz = sqrt(value) ;
    }
}
// -----

int main(int argc, char *argv[])
{
    int rank, size;

    // inicializar MPI, leer identif. de proceso y número de procesos
    MPI_Init( &argc, &argv );
    MPI_Comm_rank( MPLCOMM_WORLD, &rank );
    MPI_Comm_size( MPLCOMM_WORLD, &size );

    // inicializa la semilla aleatoria:
    srand ( time(NULL) );

    // comprobar el número de procesos con el que el programa
    // ha sido puesto en marcha (debe ser 3)
    if ( size != SIZ )
    {
        cout<< "El_numero_de_procesos_debe_ser_" << SIZ <<endl;
        return 0;
    }

    // verificar el identificador de proceso (rank), y ejecutar la
    // operación apropiada a dicho identificador
    if ( rank < Buffer )
        productor(rank);
    else if ( rank == Buffer )
        buffer();
    else
        consumidor(rank);

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

Ejercicio 2

Cena de los filósofos en MPI

2.1 Ejercicio propuesto

Implementar una solución distribuida al problema de los filósofos de acuerdo con el esquema descrito en el guión de prácticas. Usar la operación síncrona de envío `MPI_Ssend` para realizar las peticiones y liberaciones de tenedores.

2.2 Solución

Algunos puntos acerca de la solución implementada son:

- *Variables globales:* etiquetas para el envío de mensajes : `pedir_tenedor` y `soltar_tenedor`, y el número de filósofos, `NUMFILOSOFOS`.
- *Envío de mensajes en la función Filósofo:* Se enviará un mensaje a cada uno de los dos procesos que representan la función tenedor, en el orden que corresponde, con la etiqueta `pedir_tenedor`. Después de un bloqueo aleatorio, enviarán un mensaje de liberación con la etiqueta `soltar_tenedor`.
- *Envío de mensajes en la función Tenedor:* En un bucle infinito, espera un mensaje de petición de cualquier proceso filósofo, con la etiqueta `pedir_tenedor` y otro de liberación del proceso que lo ha cogido (*Filo*), con la etiqueta `soltar_tenedor`.

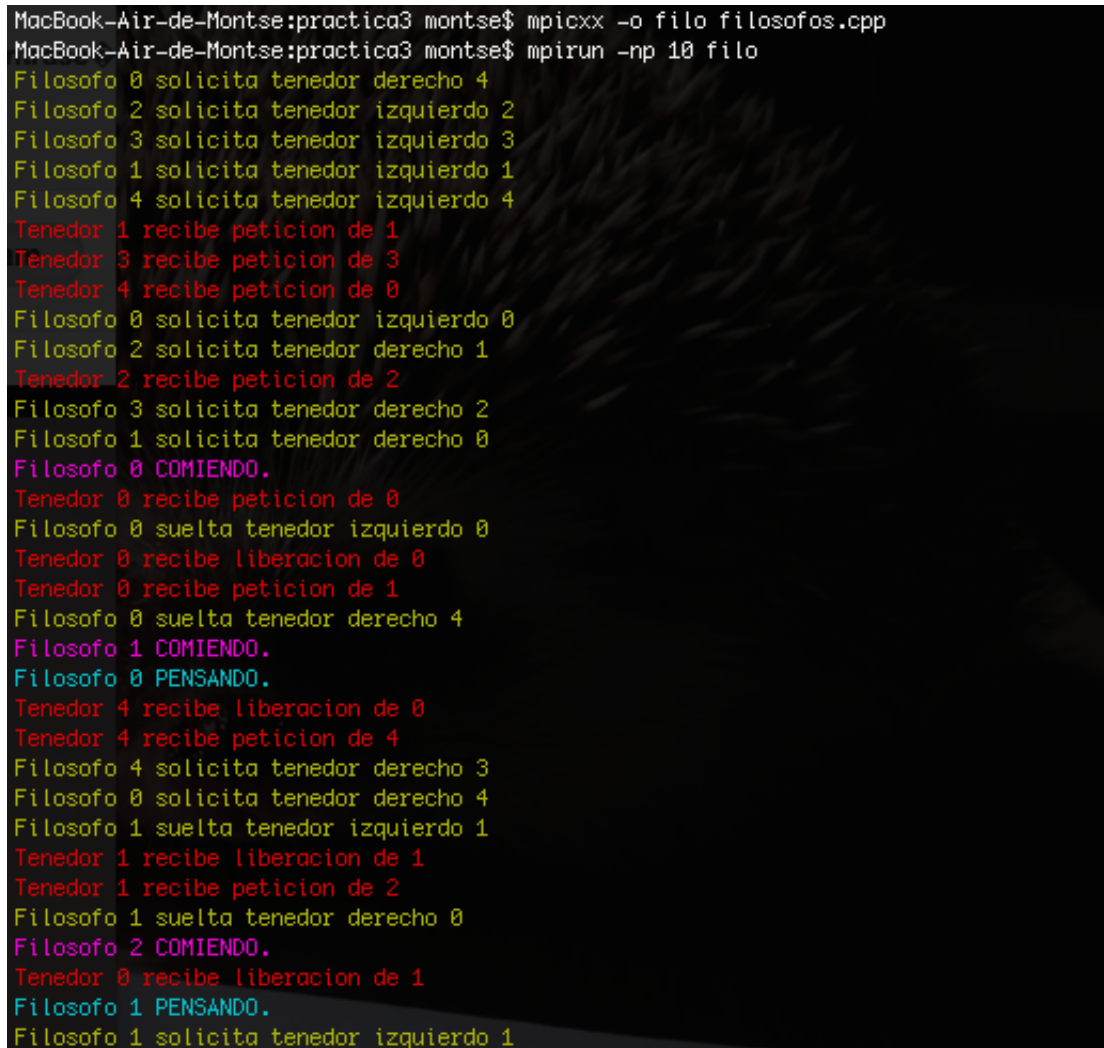
Si todos los filósofos toman simultáneamente su tenedor derecho (o todos toman el tenedor izquierdo), todos los procesos quedarán esperando a que el otro tenedor se libere, pero esto no puede ocurrir ya que ninguno podrá comer, y por tanto liberar los tenedores, si no consigue coger los dos que le corresponden, produciéndose así una situación de interbloqueo.

Para solucionarlo, tenemos que hacer que al menos uno de los filósofos pueda empezar a comer. Esto puede conseguirse haciendo que uno de los filósofos coja los tenedores en sentido contrario. Así, siempre habrá un filósofo que no podrá coger el primero de los tenedores, dejando los dos libres, y solucionando así la situación de interbloqueo. El proceso que tomará los tenedores en orden contrario será el primero ($id = 0$).

Para que las salidas por pantalla sean más claras y se correspondan con el esquema del guión de prácticas, se utilizarán algunas variables como `nf`, `td`, `tizq`,... que se usarán solo para las salidas. Para el envío de mensajes se usará el identificador del proceso en todo momento.

2.3 Ejecución del programa

En la imagen 2.1 se puede ver la ejecución del programa descrito.



```
MacBook-Air-de-Montse:practica3 montse$ mpicxx -o filo filosofos.cpp
MacBook-Air-de-Montse:practica3 montse$ mpirun -np 10 filo
Filosofo 0 solicita tenedor derecho 4
Filosofo 2 solicita tenedor izquierdo 2
Filosofo 3 solicita tenedor izquierdo 3
Filosofo 1 solicita tenedor izquierdo 1
Filosofo 4 solicita tenedor izquierdo 4
Tenedor 1 recibe peticion de 1
Tenedor 3 recibe peticion de 3
Tenedor 4 recibe peticion de 0
Filosofo 0 solicita tenedor izquierdo 0
Filosofo 2 solicita tenedor derecho 1
Tenedor 2 recibe peticion de 2
Filosofo 3 solicita tenedor derecho 2
Filosofo 1 solicita tenedor derecho 0
Filosofo 0 COMIENDO.
Tenedor 0 recibe peticion de 0
Filosofo 0 suelta tenedor izquierdo 0
Tenedor 0 recibe liberacion de 0
Tenedor 0 recibe peticion de 1
Filosofo 0 suelta tenedor derecho 4
Filosofo 1 COMIENDO.
Filosofo 0 PENSANDO.
Tenedor 4 recibe liberacion de 0
Tenedor 4 recibe peticion de 4
Filosofo 4 solicita tenedor derecho 3
Filosofo 0 solicita tenedor derecho 4
Filosofo 1 suelta tenedor izquierdo 1
Tenedor 1 recibe liberacion de 1
Tenedor 1 recibe peticion de 2
Filosofo 1 suelta tenedor derecho 0
Filosofo 2 COMIENDO.
Tenedor 0 recibe liberacion de 1
Filosofo 1 PENSANDO.
Filosofo 1 solicita tenedor izquierdo 1
```

Captura de pantalla 2.1: Listado parcial de la salida del programa *filosofos.cpp*

2.4 Código fuente

```

#include <iostream>
#include <time.h>           // incluye "time"
#include <unistd.h>         // incluye "usleep"
#include <stdlib.h>         // incluye "rand" y "srand"
#include <mpi.h>

#define NUM_FILOSOFOS    5
// etiquetas
#define pedir_tenedor    0
#define soltar_tenedor   1

/* Plantilla colores */
#define GREEN              "\033[0;32m"
#define BLUE              "\033[0;34m"
#define CYAN              "\033[0;36m"
#define RED               "\033[0;31m"
#define PURPLE            "\033[0;35m"
#define DARK_GRAY         "\033[0;30m"
#define YELLOW            "\033[0;33m"
#define WHITE             "\033[0;37m"
#define DEFAULT           "\033[0m"

using namespace std;

void Filosofo( int id, int nprocesos);
void Tenedor ( int id, int nprocesos);

// -----

int main( int argc, char** argv )
{
    int rank, size;

    srand(time(0));
    MPI_Init( &argc, &argv );
    MPI_Comm_rank( MPLCOMM_WORLD, &rank );
    MPI_Comm_size( MPLCOMM_WORLD, &size );

    if( size!=(NUM_FILOSOFOS*2))
    {
        if( rank == 0)
            cout<<RED << "El número de procesos debe ser " << NUM_FILOSOFOS*2
            << endl << DEFAULT << flush ;
        MPI_Finalize( );
        return 0;
    }

    if ((rank \% 2) == 0)
        Filosofo(rank,size); // Los pares son Filósofos
    else
        Tenedor(rank,size);  // Los impares son Tenedores

    MPI_Finalize( );
    return 0;
}
// -----

```

```

void Filosofo( int id, int nprocesos )
{
    int izq = (id+1) \% nprocesos;
    int der = ((id+nprocesos)-1) \% nprocesos;
    int mensaje = 0;
    //esto es solo para que se corresponda con el esquema
    int nf = id/2;
    int td = der/2;
    int tizq = izq/2;

    while(1)
    {
        if(id == 0){ //el primero cogerá el tenedor en otro orden
            // Solicita tenedor derecho
            cout << YELLOW << " Filosofo_"<< nf << "_solicita_tenedor_derecho_"
            << td << endl << DEFAULT << flush;
            MPI_Ssend(&mensaje, 1, MPLINT, der, pedir_tenedor, MPLCOMM_WORLD);
            // Solicita tenedor izquierdo
            cout << YELLOW << " Filosofo_"<< nf << "_solicita_tenedor_izquierdo_"
            << tizq << endl << DEFAULT << flush;
            MPI_Ssend(&mensaje, 1, MPLINT, izq, pedir_tenedor, MPLCOMM_WORLD);
        }
        else{
            // Solicita tenedor izquierdo
            cout << YELLOW << " Filosofo_"<< nf << "_solicita_tenedor_izquierdo_"
            << tizq << endl << DEFAULT << flush;
            MPI_Ssend(&mensaje, 1, MPLINT, izq, pedir_tenedor, MPLCOMM_WORLD);

            // Solicita tenedor derecho
            cout << YELLOW << " Filosofo_"<< nf << "_solicita_tenedor_derecho_"
            << td << endl << DEFAULT << flush;
            MPI_Ssend(&mensaje, 1, MPLINT, der, pedir_tenedor, MPLCOMM_WORLD);
        }

        cout<< PURPLE << " Filosofo_"<< nf << "_COMIENDO."
        << endl << flush;
        sleep((rand() \% 3)+1); //comiendo

        // Suelta el tenedor izquierdo
        cout << YELLOW << " Filosofo_"<< nf << "_suelta_tenedor_izquierdo_"
        << tizq << endl << DEFAULT << flush;
        MPI_Ssend(&mensaje, 1,MPLINT, izq, soltar_tenedor, MPLCOMM_WORLD);

        // Suelta el tenedor derecho
        cout << YELLOW << " Filosofo_"<< nf << "_suelta_tenedor_derecho_" << td
        << endl << DEFAULT << flush;
        MPI_Ssend(&mensaje, 1,MPLINT, der, soltar_tenedor, MPLCOMM_WORLD);

        // Piensa (espera bloqueada aleatorio del proceso)
        cout << CYAN << " Filosofo_" << nf << "_PENSANDO."
        << endl << DEFAULT << flush;

        // espera bloqueado durante un intervalo de tiempo aleatorio
        // (entre una décima de segundo y un segundo)
        usleep( 1000U * (100U+(rand()\%900U)) );
    }
}
//

```

```
void Tenedor(int id, int nprocesos)
{
    int buf;
    MPI_Status status;
    int Filo;
    int nt = id/2; //para que se corresponda con el esquema

    while(1)
    {
        MPI_Recv(&buf, 1, MPI_INT, MPLANY_SOURCE, pedir_tenedor, MPLCOMM_WORLD, &status);
        Filo = status.MPLSOURCE; //proceso

        //para que se corresponda con el esquema
        int nfilo = Filo/2;

        cout << RED << "Tenedor_" << nt << "_recibe_peticion_de_" << nfilo
        << endl << DEFAULT << flush;
        //espera que el filosofo que lo ha cogido lo libere
        MPI_Recv(&buf, 1, MPI_INT, Filo, soltar_tenedor, MPLCOMM_WORLD, &status);
        cout << RED << "Tenedor_" << nt << "_recibe_liberacion_de_" << nfilo
        << endl << DEFAULT << flush;
    }
}
// _____
```


Ejercicio 3

Cena de los filósofos con camarero central en MPI

3.1 Ejercicio propuesto

Otra forma de evitar la posibilidad del interbloqueo consiste en impedir que todos los filósofos intenten ejecutar la acción de "tomar tenedor" al mismo tiempo. Para ello podemos usar un proceso *camarero central* que permita sentarse a la mesa como máximo a 4 filósofos. Podemos suponer que tenemos 11 procesos y que el camarero es el proceso 10.

3.2 Solución

Se trabaja sobre la implementación anterior del programa, el programa *filosofos.cpp*. Algunos de los puntos acerca de la nueva implementación del programa son:

- *Variables globales:* Añadimos la variable *CAMARERO* que establece que el proceso camarero tiene el identificador 10. Se añaden también las etiquetas *sentarse*, *levantarse*, que servirán para el envío de mensajes entre el camarero central y los filósofos.
- *Implementación de la función camarero:* Se ejecutarán las siguientes sentencias en un bucle infinito: si hay suficiente espacio en la mesa, se pueden recibir peticiones tanto para sentarse como para levantarse. Se espera a recibir un mensaje de cualquier proceso, y de cualquier tipo. Si es un mensaje con la etiqueta *sentarse*, se recibe el mensaje correspondiente y se envía un mensaje al filósofo del que lo haya recibido indicando que puede sentarse a la mesa. Si el filósofo quiere levantarse, simplemente decrementamos el contador *sentados*. Si no hay espacio suficiente en la mesa, se esperará a recibir un mensaje de algún proceso filósofo que quiera levantarse y se decrementará el contador.
- *Modificaciones en la función filósofo:* Antes de solicitar los tenedores, el proceso filósofo mandará un mensaje al proceso camarero, con la etiqueta *sentarse*, y esperará a que el proceso camarero envíe un mensaje con la misma etiqueta para poder sentarse. Después de comer, el proceso mandará un mensaje al camarero indicando que va a levantarse con la etiqueta correspondiente.
- *Modificaciones en la función tenedor:* No hay cambios.

3.3 Ejecución del programa

En la imagen 3.1 se puede ver la ejecución del programa descrito.

3.4 Código fuente

```

#include <iostream>
#include <time.h>           // incluye "time"
#include <unistd.h>         // incluye "usleep"
#include <stdlib.h>         // incluye "rand" y "srand"
#include <mpi.h>

#define NUM_FILOSOFOS      5
#define CAMARERO           10
// etiquetas
#define pedir_tenedor      0
#define soltar_tenedor     1
#define sentarse           2
#define levantarse         3

/* Plantilla colores */
#define GREEN               "\033[0;32m"
#define BLUE               "\033[0;34m"
#define CYAN               "\033[0;36m"
#define RED                "\033[0;31m"
#define PURPLE             "\033[0;35m"
#define GRAY               "\033[0;37m"
#define DARK_GRAY          "\033[0;30m"
#define YELLOW             "\033[0;33m"
#define WHITE              "\033[0;37m"
#define DEFAULT            "\033[0m"

using namespace std;

void Filosofo( int id, int nprocesos);
void Tenedor ( int id, int nprocesos);
void Camarero ();
// -----

int main( int argc, char** argv )
{
    int rank, size;

    srand(time(0));
    MPI_Init( &argc, &argv );
    MPI_Comm_rank( MPLCOMM_WORLD, &rank );
    MPI_Comm_size( MPLCOMM_WORLD, &size );

    if( size!=(NUM_FILOSOFOS*2 + 1))
    {
        if(rank == 0)
            cout<<RED << "El número de procesos debe ser " << NUM_FILOSOFOS*2 + 1
              << endl << DEFAULT << flush ;
        MPI_Finalize( );
        return 0;
    }

    if ((rank%2) == 0 && rank != CAMARERO){
        Filosofo(rank,size); // Los pares son Filósofos
    }
    else if(rank == CAMARERO){
        Camarero();
    }
}

```

```

else
    Tenedor(rank, size); // Los impares son Tenedores

MPI_Finalize( );
return 0;
}
// -----

void Filosofo( int id, int nprocesos )
{
    int izq = (id+1) \% (nprocesos-1);
    int der = ((id+nprocesos-1)-1) \% (nprocesos-1);
    int mensaje = 0;
    //esto es solo para que se corresponda con el esquema
    int nf = id/2;
    int td = der/2;
    int tizq = izq/2;
    //para recibir mensajes
    int buf;
    MPI_Status status;

    while(1)
    {
        //SENTARSE
        cout << YELLOW << " Filosofo_" << nf << "_solicita_sentarse_a_la_mesa."
        << endl << DEFAULT << flush;
        MPI_Ssend(&mensaje, 1, MPI_INT, CAMARERO, sentarse, MPLCOMM_WORLD);

        //COGER TENEDORES
        // Solicita tenedor izquierdo
        cout << YELLOW << " Filosofo_" << nf << "_solicita_tenedor_izquierdo_"
        << tizq << endl << DEFAULT << flush;
        MPI_Ssend(&mensaje, 1, MPI_INT, izq, pedir_tenedor, MPLCOMM_WORLD);

        // Solicita tenedor derecho
        cout << YELLOW << " Filosofo_" << nf << "_solicita_tenedor_derecho_"
        << td << endl << DEFAULT << flush;
        MPI_Ssend(&mensaje, 1, MPI_INT, der, pedir_tenedor, MPLCOMM_WORLD);

        //COMER

        cout << PURPLE << " Filosofo_" << nf << "_COMIENDO."
        << endl << flush;
        sleep((rand() \% 3)+1); //comiendo

        //SOLTAR TENEDORES
        // Suelta el tenedor izquierdo
        cout << YELLOW << " Filosofo_" << nf << "_suelta_tenedor_izquierdo_"
        << tizq << endl << DEFAULT << flush;
        MPI_Ssend(&mensaje, 1, MPI_INT, izq, soltar_tenedor, MPLCOMM_WORLD);

        // Suelta el tenedor derecho
        cout << YELLOW << " Filosofo_" << nf << "_suelta_tenedor_derecho_" << td
        << endl << DEFAULT << flush;
        MPI_Ssend(&mensaje, 1, MPI_INT, der, soltar_tenedor, MPLCOMM_WORLD);

        //LEVANTARSE
        MPI_Ssend(&mensaje, 1, MPI_INT, CAMARERO, levantarse, MPLCOMM_WORLD);
    }
}

```



```

    // PENSAR
    cout << CYAN << "Filosofo_" << nf << "_FILOSOFANDO."
    << endl << DEFAULT << flush;

    // espera bloqueado durante un intervalo de tiempo aleatorio
    // (entre una décima de segundo y un segundo)
    usleep( 1000U * (100U+(rand())\%900U) );
}
}
// -----

void Tenedor(int id, int nprocesos)
{
    int buf;
    MPI_Status status;
    int Filo;
    int nt = id/2; //para que se corresponda con el esquema

    while(1)
    {
        MPI_Recv(&buf, 1, MPI_INT, MPLANY_SOURCE, pedir_tenedor, MPLCOMM_WORLD, &status);
        Filo = status.MPLSOURCE; //proceso
        //para que se corresponda con el esquema
        int nfilo = Filo/2;

        cout << RED << "Tenedor_" << nt << "_recibe_peticion_de_" << nfilo
        << endl << DEFAULT << flush;
        //espera que el filosofo que lo ha cogido lo libere
        MPI_Recv(&buf, 1, MPI_INT, Filo, soltar_tenedor, MPLCOMM_WORLD, &status);
        cout << RED << "Tenedor_" << nt << "_recibe_liberacion_de_" << nfilo
        << endl << DEFAULT << flush;
    }
}
// -----

void Camarero()
{
    int buf;
    MPI_Status status;
    int Filo, nfilo;
    int sentados = 0;
    int mensaje = 0;
    while(1){
        if(sentados < 4){ //si hay espacio, puede recibir ambas peticiones
            MPI_Probe(MPLANY_SOURCE, MPLANY_TAG, MPLCOMM_WORLD, &status);
        }
        else{
            MPI_Probe(MPLANY_SOURCE, levantarse, MPLCOMM_WORLD, &status);
        }

        Filo = status.MPLSOURCE;
        nfilo = Filo/2;

        if(status.MPLTAG == sentarse){
            MPI_Recv(&mensaje, 1, MPI_INT, Filo, sentarse, MPLCOMM_WORLD, &status);
            sentados++;
            cout << GREEN << "Camarero_sienta_a_filosofo_" << nfilo << "_a_la_mesa."
            << endl << DEFAULT << flush;
        }
        else{

```

```
MPI_Recv(&mensaje,1,MPI_INT, Filo,levantarse, MPLCOMM_WORLD, &status);
cout << GREEN << "Camarero_levanta_a_filosofo_"<< nfilo << "_de_la_mesa."
<< endl << DEFAULT << flush;
sentados--;
    }
}
// _____
```

```
MacBook-Air-de-Montse:Desktop montse$ mpirun -np 11 filo
Filosofo 0 solicita sentarse a la mesa.
Filosofo 1 solicita sentarse a la mesa.
Camarero sienta a filosofo 0 a la mesa.
Camarero sienta a filosofo 1 a la mesa.
Filosofo 2 solicita sentarse a la mesa.
Filosofo 0 solicita tenedor izquierdo 0
Filosofo 1 solicita tenedor izquierdo 1
Tenedor 1 recibe peticion de 1
Filosofo 4 solicita sentarse a la mesa.
Camarero sienta a filosofo 2 a la mesa.
Camarero sienta a filosofo 4 a la mesa.
Tenedor 0 recibe peticion de 0
Filosofo 2 solicita tenedor izquierdo 2
Filosofo 3 solicita sentarse a la mesa.
Filosofo 0 solicita tenedor derecho 4
Filosofo 1 solicita tenedor derecho 0
Filosofo 4 solicita tenedor izquierdo 4
Tenedor 4 recibe peticion de 0
Filosofo 2 solicita tenedor derecho 1
Tenedor 2 recibe peticion de 2
Filosofo 0 COMIENDO.
Filosofo 0 suelta tenedor izquierdo 0
Tenedor 0 recibe liberacion de 0
Filosofo 0 suelta tenedor derecho 4
Tenedor 0 recibe peticion de 1
Filosofo 1 COMIENDO.
Tenedor 4 recibe liberacion de 0
Tenedor 4 recibe peticion de 4
Camarero levanta a filosofo 0 de la mesa.
Filosofo 0 FILOSOFANDO.
Camarero sienta a filosofo 3 a la mesa.
Filosofo 4 solicita tenedor derecho 3
Filosofo 3 solicita tenedor izquierdo 3
Filosofo 4 COMIENDO.
Tenedor 3 recibe peticion de 4
Filosofo 0 solicita sentarse a la mesa.
Filosofo 1 suelta tenedor izquierdo 1
Filosofo 4 suelta tenedor izquierdo 4
Tenedor 4 recibe liberacion de 4
Filosofo 1 suelta tenedor derecho 0
Tenedor 1 recibe liberacion de 1
Filosofo 4 suelta tenedor derecho 3
Tenedor 1 recibe peticion de 2
Filosofo 2 COMIENDO.
Tenedor 0 recibe liberacion de 1
Tenedor 3 recibe liberacion de 4
Tenedor 3 recibe peticion de 3
Camarero levanta a filosofo 1 de la mesa.
Filosofo 1 FILOSOFANDO.
Camarero levanta a filosofo 4 de la mesa.
Camarero sienta a filosofo 0 a la mesa.
Filosofo 3 solicita tenedor derecho 2
Filosofo 4 FILOSOFANDO.
Filosofo 0 solicita tenedor izquierdo 0
Tenedor 0 recibe peticion de 0
Filosofo 0 solicita tenedor derecho 4
Tenedor 4 recibe peticion de 0
```

Captura de pantalla 3.1: Listado parcial de la salida del programa *filosofos_camarero.cpp*