# PRÁCTICA 1

Por: Jaime Tejedor y Luis Díaz del Río

## Introducción:

En esta práctica se nos pide realizar 3 ejercicios con el objetivo de familiarizarnos con el entorno de desarrollo de las prácticas así como la librería MPI:

## Ejercicio 1:

Implementar un programa usando MPI, que imprima por salida estándar:

"Hola mundo, soy el proceso X de un total de Y." cuando el número total de tareas es Y=50 y X un rango de 0 a 49.

Calcular el tiempo de ejecución de cada proceso y realizar una gráfica explicando los resultados a medida que aumenta el número de procesos.

```c
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv){
    int size;
    int rank;
    double start, finish, time;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    MPI_Barrier(MPI_COMM_WORLD);
    start = MPI_Wtime();
    printf("Hola Mundo! Soy el proceso %d de un total de %d.\n", rank, size);
    MPI_Barrier(MPI_COMM_WORLD);
    finish = MPI_Wtime();
    time = finish - start;
    printf("El proceso %d ha tardado %f. \n", rank, time);
    MPI_Finalize();
    return 0;
}
```

Siguiendo las instrucciones dadas en el documento introductorio, declaramos las variables antes de utilizar la función MPI_Init(). Size y rank son para el número de procesos y el proceso en el que estamos actualmente respectivamente. Start, finish y time es para sacar los tiempos que se piden en el enunciado de la práctica.

Tras la declaración de variables iniciamos con MPI_Init() donde le pasamos como argumentos las variables que entran por la función main. Mediante MPI_Comm_size(), escribimos en la variable size el número de procesos, seguidamente utilizamos MPI_Comm_rank() para escribir en la variable rank el proceso en el que estamos actualmente.

Para calcular los tiempos hacemos uso de las funciones MPI_Barrier() y MPI_Wtime() para calcular el tiempo que tarda cada proceso. Dentro de la primera "barrera" guardamos el tiempo antes de sacar por pantalla el texto que se pide en el enunciado de la práctica, las cerrar dicha "barrera" cogemos el tiempo una vez el mensaje ha salido por pantalla y hacemos la resta del tiempo al finalizar y al iniciar para calcular el tiempo total que ha tardado cada proceso.

Seguidamente sacamos por pantalla el tiempo de cada proceso por pantalla. Una vez han terminado todos los procesos paramos la ejecución mediante la función MPI_Finalize()

Resultados:

Para 50 procesos:

```
noone@noone-VirtualBox:~/Escritorio/ArquitecturaOrdenadores/practica1/Ejercicio1$ mpirun -np 50 ./ejercicio1
Hola Mundo! Soy el proceso 0 de un total de 50.
Hola Mundo! Soy el proceso 4 de un total de 50.
Hola Mundo! Soy el proceso 8 de un total de 50.
Hola Mundo! Soy el proceso 16 de un total de 50.
Hola Mundo! Soy el proceso 1 de un total de 50.
Hola Mundo! Soy el proceso 24 de un total de 50.
Hola Mundo! Soy el proceso 9 de un total de 50.
Hola Mundo! Soy el proceso 26 de un total de 50.
Hola Mundo! Soy el proceso 10 de un total de 50.
Hola Mundo! Soy el proceso 32 de un total de 50.
Hola Mundo! Soy el proceso 33 de un total de 50.
Hola Mundo! Soy el proceso 12 de un total de 50.
Hola Mundo! Soy el proceso 25 de un total de 50.
Hola Mundo! Soy el proceso 14 de un total de 50.
Hola Mundo! Soy el proceso 6 de un total de 50.
Hola Mundo! Soy el proceso 2 de un total de 50.
Hola Mundo! Soy el proceso 17 de un total de 50.
Hola Mundo! Soy el proceso 5 de un total de 50.
Hola Mundo! Soy el proceso 40 de un total de 50.
Hola Mundo! Soy el proceso 42 de un total de 50.
Hola Mundo! Soy el proceso 43 de un total de 50.
Hola Mundo! Soy el proceso 13 de un total de 50.
Hola Mundo! Soy el proceso 20 de un total de 50.
Hola Mundo! Soy el proceso 48 de un total de 50.
Hola Mundo! Soy el proceso 18 de un total de 50.
Hola Mundo! Soy el proceso 41 de un total de 50.
Hola Mundo! Soy el proceso 19 de un total de 50.
Hola Mundo! Soy el proceso 3 de un total de 50.
Hola Mundo! Soy el proceso 27 de un total de 50.
Hola Mundo! Soy el proceso 22 de un total de 50.
Hola Mundo! Soy el proceso 36 de un total de 50.
Hola Mundo! Soy el proceso 11 de un total de 50.
Hola Mundo! Soy el proceso 28 de un total de 50.
Hola Mundo! Soy el proceso 15 de un total de 50.
Hola Mundo! Soy el proceso 21 de un total de 50.
Hola Mundo! Soy el proceso 34 de un total de 50.
Hola Mundo! Soy el proceso 29 de un total de 50.
Hola Mundo! Soy el proceso 37 de un total de 50.
Hola Mundo! Soy el proceso 7 de un total de 50.
Hola Mundo! Soy el proceso 30 de un total de 50.
Hola Mundo! Soy el proceso 44 de un total de 50.
Hola Mundo! Soy el proceso 31 de un total de 50.
Hola Mundo! Soy el proceso 23 de un total de 50.
Hola Mundo! Soy el proceso 49 de un total de 50.
Hola Mundo! Soy el proceso 38 de un total de 50.
Hola Mundo! Soy el proceso 35 de un total de 50.
Hola Mundo! Soy el proceso 46 de un total de 50.
Hola Mundo! Soy el proceso 39 de un total de 50.
Hola Mundo! Soy el proceso 47 de un total de 50.
Hola Mundo! Soy el proceso 45 de un total de 50.
```

```
Hola Mundo! Soy el proceso 45 de un total de 50.
El proceso 0 ha tardado 1.092124.
El proceso 8 ha tardado 1.007627.
El proceso 12 ha tardado 0.958392.
El proceso 10 ha tardado 0.988766.
El proceso 2 ha tardado 0.975991.
El proceso 32 ha tardado 1.033010.
El proceso 9 ha tardado 1.071966.
El proceso 13 ha tardado 0.980134.
El proceso 34 ha tardado 0.855650.
El proceso 1 ha tardado 1.112299.
El proceso 40 ha tardado 1.020436.
El proceso 48 ha tardado 1.012993.
El proceso 4 ha tardado 1.200539.
El proceso 42 ha tardado 1.054940.
El proceso 11 ha tardado 0.959983.
El proceso 16 ha tardado 1.196149.
El proceso 41 ha tardado 1.050167.
El proceso 14 ha tardado 1.131013.
El proceso 5 ha tardado 1.112231.
El proceso 43 ha tardado 1.109603.
El proceso 36 ha tardado 0.999748.
El proceso 37 ha tardado 0.972001.
El proceso 33 ha tardado 1.184956.
El proceso 6 ha tardado 1.155440.
El proceso 20 ha tardado 1.120430.
El proceso 7 ha tardado 0.992005.
El proceso 35 ha tardado 0.931273.
El proceso 3 ha tardado 1.078499.
El proceso 49 ha tardado 0.946956.
El proceso 38 ha tardado 1.007157.
El proceso 18 ha tardado 1.208476.
El proceso 17 ha tardado 1.260372.
El proceso 44 ha tardado 1.075996.
El proceso 22 ha tardado 1.144819.
El proceso 19 ha tardado 1.195093.
El proceso 15 ha tardado 1.132031.
El proceso 24 ha tardado 1.358912.
El proceso 39 ha tardado 0.984042.
El proceso 26 ha tardado 1.385117.
El proceso 28 ha tardado 1.194250.
El proceso 45 ha tardado 0.975733.
El proceso 21 ha tardado 1.227899.
El proceso 27 ha tardado 1.271220.
El proceso 29 ha tardado 1.231985.
El proceso 23 ha tardado 1.190198.
El proceso 46 ha tardado 1.131024.
El proceso 47 ha tardado 1.047919.
El proceso 25 ha tardado 1.471729.
El proceso 30 ha tardado 1.323383.
El proceso 31 ha tardado 1.351933.
```

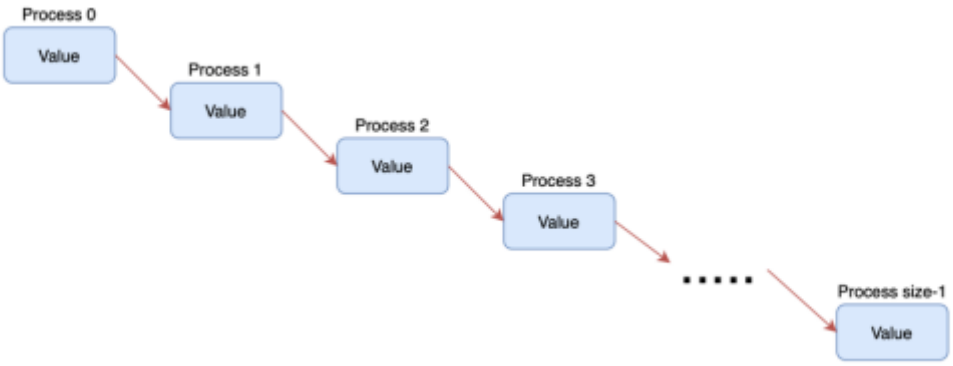
Para 5 procesos:

```
noone@noone-VirtualBox:~/Escritorio/ArquitecturaOrdenadores/practica1/Ejercicio1$ mpirun -np 5 ./ejercicio1
Hola Mundo! Soy el proceso 0 de un total de 5.
Hola Mundo! Soy el proceso 1 de un total de 5.
Hola Mundo! Soy el proceso 4 de un total de 5.
Hola Mundo! Soy el proceso 2 de un total de 5.
Hola Mundo! Soy el proceso 3 de un total de 5.
El proceso 0 ha tardado 0.052088.
El proceso 2 ha tardado 0.049179.
El proceso 3 ha tardado 0.039305.
El proceso 4 ha tardado 0.057330.
El proceso 1 ha tardado 0.072224.
```

Como podemos ver, a mayor número de procesos, mayor es el tiempo para resolver dichos procesos.

## Ejercicio2:

Implementar un programa usando MPI, donde el proceso 0 toma un dato del usuario y lo envía al resto de nodos en anillo. Esto es, el proceso i recibe de i-1 y transmite el dato a i+1, hasta que el dato alcanza el último nodo:




Asumir que el dato que se transmite es un entero y que el proceso cero lee el dato del usuario.

```c
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv){
    int userValue;
    int rank;
    int size;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    if(rank == 0){
        printf("Add a number: ");
        scanf("%d", &userValue);
        printf("Al proceso %d le ha entrado el valor %d y lo va a mandar al proceso %d \n", rank, userValue, rank+1);
        MPI_Send(&userValue, 1, MPI_INT, rank+1, 0, MPI_COMM_WORLD);
    }else{
        MPI_Recv(&userValue, 1, MPI_INT, rank-1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        if(rank+1 < size){
            MPI_Send(&userValue, 1, MPI_INT, rank+1, 0, MPI_COMM_WORLD);
        }
        printf("El proceso %d ha recibido el valor %d del proceso %d \n", rank, userValue, rank-1);
    }
    MPI_Finalize();
    return 0;
}
```

La primera parte del código no cambia respecto al ejercicio anterior salvo por la variable userValue en la que se almacenará número elegido por el usuario y que a su vez hará de buffer del mensaje.

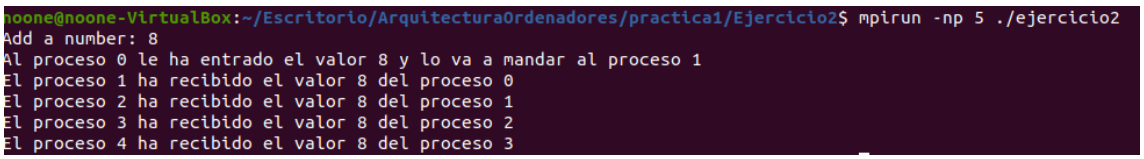
Ya después de las funciones iniciales de MPI, y mediante un if, comprobamos el proceso en el que estamos. En el caso de que sea “0”, pedimos al usuario que añada un número por consola, sacamos por pantalla el proceso de origen, el valor que vamos a mandar y el proceso al que va dirigido.

Mediante la función MPI_Send() mandamos un único mensaje almacenado en el buffer userValue, de tipo MPI_INT al siguiente proceso, en este caso rank+1. El tag lo dejamos a “0” y el Comm a MPI_COMM_WORLD.

En el caso de que nuestro proceso fuese distinto de cero, primero recibimos el mensaje mediante la función MPI_Recv(), que recibe un único mensaje almacenado el userValue, de tipo MPI_INT, de rank-1 (proceso anterior). Tanto el tag como el Comm son iguales que en MPI_Send(). Además hay que pasarle un estado, no sabíamos muy bien que significaba exactamente, pero viendo información online lo más común es utilizar MPI_STATUS_IGNORE por lo que hemos decidido ponerlo así.

Además comprobamos si el siguiente proceso existe (en este caso que no sea mayor que 4) para evitar mandar un mensaje a un proceso inexistente. Si el proceso existe, mandamos el mensaje almacenado en userValue al siguiente proceso.

Resultado:



## Ejercicio3:

Modificar la implementación del ejercicio 2 para que el dato introducido por el usuario dé tantas vueltas como este indique en el anillo.

¿Qué desventaja se aprecia en este tipo de comunicaciones punto a punto a medida que aumentan el número de procesos requeridos? Razonar la respuesta.

¿Cómo podría mejorar el sistema y su implementación? Razonar la respuesta.

```c
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv){
    int pNum;
    int size;
    int userValue = 7;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &pNum);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    for(int i=0; i < 3; i++){
        if(pNum == 0){
            if(i == 0){
                printf("Al proceso %d le ha entrado el valor %d y lo va a mandar al proceso %d \n", pNum, userValue, pNum+1);
            }else{
                printf("El proceso %d ha recibido el valor %d del proceso %d \n", pNum, userValue, size-1);
            }
            MPI_Send(&userValue, 1, MPI_INT, pNum+1, 0, MPI_COMM_WORLD);
            MPI_Recv(&userValue, 1, MPI_INT, size-1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        }else{
            MPI_Recv(&userValue, 1, MPI_INT, pNum-1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            if(pNum+1 < size){
                MPI_Send(&userValue, 1, MPI_INT, pNum+1, 0, MPI_COMM_WORLD);
            }
            if(pNum == size-1){
                MPI_Send(&userValue, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
            }
            printf("El proceso %d ha recibido el valor %d del proceso %d \n", pNum, userValue, pNum-1);
        }
    }
    MPI_Finalize();
    return 0;
}
```

Para este ejercicio hemos modificado parte del código del ejercicio anterior. He de comentar que la variable pNum es la equivalente a la variable rank del ejercicio anterior. Además, en este ejercicio el usuario no mete el número por pantalla, si no que lo define dentro de la variable userValue.

La primera modificación es meter el código dentro de un bucle for. Mediante este bucle el mensaje iterará en el anillo tanto como el usuario especifique. En nuestro caso 3 veces.

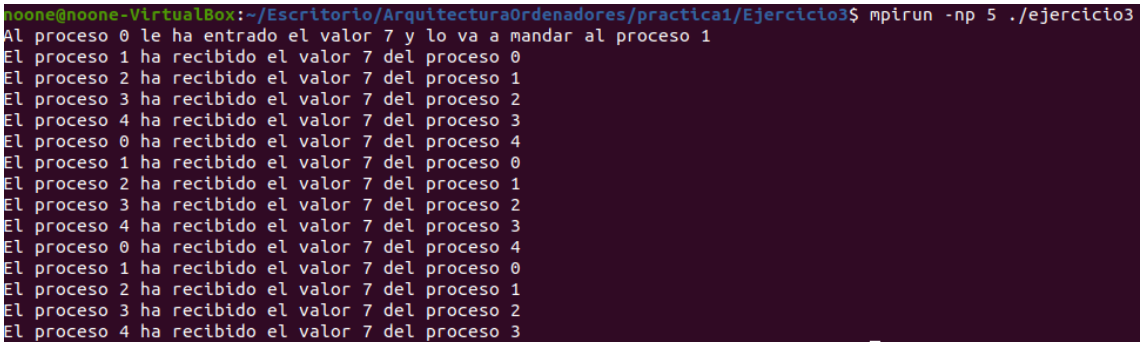
Dentro del primer if, que tiene la misma función que en el ejercicio anterior, comprobamos el número de iteración para marcar cuando está entrando el dato metido por el usuario y cuando está recibiendo el mensaje de otro proceso.

Además, no solo mandamos el mensaje al siguiente proceso, si no que recibimos un mensaje del último proceso, en nuestro caso el proceso 4.

En cuanto al código en el interior del else, encontramos una única modificación. Mediante un if, comprobamos si el proceso actual es el último proceso, en caso de ser así, devolvemos el mensaje al proceso inicial (el proceso 0).

No sabríamos explicar, ni demostrar, las cuestiones planteadas en el ejercicio 3.

Resultados:




## Conclusión:

Estamos contentos con el trabajo realizado, consideramos que no solo hemos adquirido los conocimientos necesarios para la realización de la práctica, si no que además se han cumplido los objetivos de manera óptima.

## Bibliografía:

Documentos dados por la docente que imparte las prácticas.

https://mpitutorial.com/tutorials/mpi-send-and-receive/