Práctica 9

**Introducción a MPI**

Jordi Blasco Lozano

Computación de alto rendimiento

Grado en Inteligencia Artificial

## Índice:

[Índice: 2](#_Toc194339654)

[1. Introducción 2](#_Toc194339655)

[2. Desarrollo 2](#_Toc194339656)

[Ejercicio 1 - Comunicación punto a punto 2](#_Toc194339657)

[¿Qué hace el programa propuesto? 2](#_Toc194339658)

[¿Qué sucede si cambiamos los tags? 3](#_Toc194339659)

[¿Cómo compilar y ejecutar con mpicc? 3](#_Toc194339660)

## Introducción

## Desarrollo

## Ejercicio 1 - Comunicación punto a punto

Este ejercicio tiene como objetivo afianzar el conocimiento en la comunicación punto a punto utilizando MPI (Message Passing Interface).

## ¿Qué hace el programa propuesto?

El programa inicia con la llamada a MPI\_Init(&argc, &argv), que configura el entorno de MPI y permite que los procesos se comuniquen. Cada proceso obtiene su identificador único (rank) mediante MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank). Esto es fundamental para asignar roles diferentes a cada proceso según su rango.

En este ejemplo se implementa una comunicación punto a punto entre dos procesos: el proceso 0 y el proceso 1. El proceso 0 es el encargado de enviar un dato (en este caso, el número 42) al proceso 1, que se encargará de recibirlo y posteriormente imprimirlo.

Cuando el proceso 0 detecta que su rank es 0, define una variable dato con el valor 42 y realiza la operación de envío mediante MPI\_Send(&dato, 1, MPI\_INT, 1, 0, MPI\_COMM\_WORLD). Aquí se especifica:

* La dirección del dato a enviar (&dato).
* El número de elementos a enviar (1).
* El tipo de dato (MPI\_INT).
* El proceso destino (1).
* Un tag (0) que actúa como etiqueta para identificar el mensaje.
* El comunicador (MPI\_COMM\_WORLD).

En el proceso 1, al comprobar que su rank es 1, se declara la variable recibida para almacenar el dato entrante. Luego se llama a MPI\_Recv(&recibido, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE), la cual espera recibir el mensaje del proceso 0. Los parámetros indican:

* La dirección donde se almacenará el dato recibido.
* El número de elementos que se esperan.
* El tipo de dato (MPI\_INT).
* El proceso fuente (0), es decir, de dónde se espera el mensaje.
* El tag (0), que debe coincidir con el tag usado en el envío.
* El comunicador.

Y se ignora el estado del mensaje recibido.

Una vez que el proceso 1 recibe el mensaje, se imprime el dato utilizando printf

## ¿Qué sucede si cambiamos los tags?

Los tags (etiquetas) son fundamentales para emparejar mensajes entre el envío y la recepción. En el programa, tanto el proceso 0 al enviar como el proceso 1 al recibir utilizan el tag 0. Si se cambiara el tag en uno de ellos, por ejemplo, enviando con tag 1 pero intentando recibir con tag 0, el mensaje no se emparejaría correctamente. Esto provocaría que el proceso que llama a MPI\_Recv se quede bloqueado esperando un mensaje con el tag correcto, o se produzca un error en la comunicación.

## ¿Qué ocurre si el proceso 1 ejecuta MPI\_Recv antes de que el proceso 0 envíe el mensaje?

MPI utiliza la comunicación bloqueante en este ejemplo, lo que significa que si el proceso 1 ejecuta MPI\_Recv antes de que el proceso 0 envíe el mensaje, el proceso 1 se bloqueará y esperará hasta que el mensaje llegue. Este comportamiento es el esperado y forma parte de la sincronización de la comunicación punto a punto.

## ¿Qué sucede si cambiamos los tags?

Si el proceso 0 intenta enviar el dato a un proceso que no existe (por ejemplo, especificando un destino con un rank que no se haya iniciado), MPI detectaría un error. Este error puede causar que el programa termine de forma inesperada o que se comporte de manera indefinida, ya que se estaría intentando comunicar con un proceso que no forma parte del comunicador. Es esencial que el rango del proceso destino exista y esté incluido en el comunicador utilizado.

## ¿Cómo compilar y ejecutar con mpicc?

* **Compilación**

Para compilar con nuestro compilador mpicc lo hacemos igual que cualquier otro archivo .c,   
mpicc -o ejercicio1 ejercicio1.c.

Si queremos compilar con un compilador de c++ podemos usar el de mpic++,   
mpic++ -o ejercicio1 practica1.cpp

* **Ejecución**

Para la ejecución de los programas lo haremos de la siguiente forma, tanto si es de c como si es de c++ mpirun -np 3 ./ejercicio1, el número que sigue a -np es el número de procesos que se ejecuta en paralelo

## Código para la tarea propuesta

La tarea consiste en modificar el ejemplo para que el proceso 0 envíe un número al proceso 2, y este último lo multiplique por 3 antes de imprimirlo.

#include <mpi.h>

#include <stdio.h>

**int** main(**int** argc, **char**\*\* argv) {

MPI\_Init(&argc, &argv);

**int** rank;

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

if (rank == 0) {

**int** dato = 42; *// Valor a enviar*

MPI\_Send(&dato, 1, MPI\_INT, 2, 0, MPI\_COMM\_WORLD);

} else if (rank == 2) {

**int** recibido;

*// Recibimos el dato enviado por el proceso 0*

MPI\_Recv(&recibido, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD MPI\_STATUS\_IGNORE);

*// Multiplicamos el dato por 3*

**int** resultado = recibido \* 3;

printf("Proceso 2 recibió %d y al multiplicarlo por 3 da: %d\n", recibido,   
 resultado);

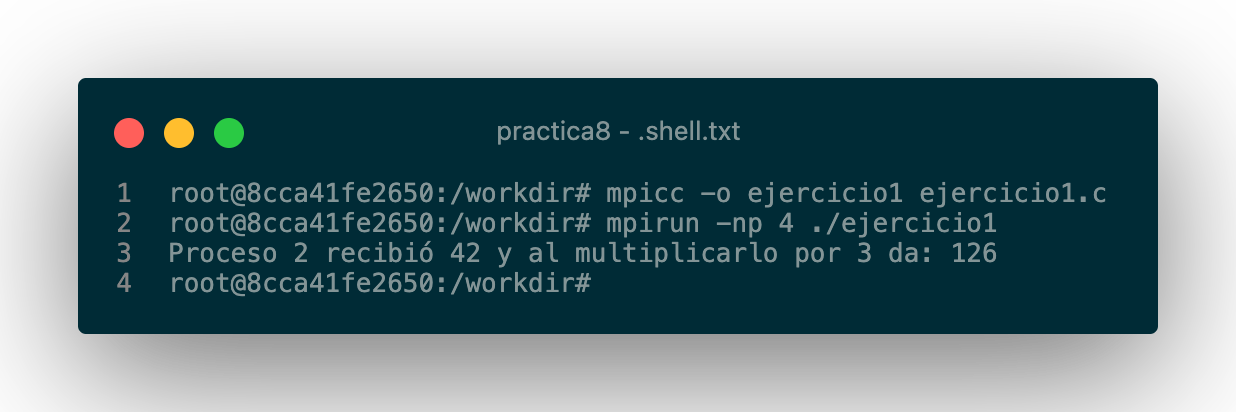
}

MPI\_Finalize();

return 0;

}

## Salida



## ¿Cómo funciona?

**Proceso 0:** Envía un número (en este ejemplo, 42) al proceso 2 utilizando MPI\_Send.

**Proceso 2:** Recibe el mensaje con MPI\_Recv, multiplica el valor recibido por 3 y muestra el resultado por pantalla.

**MPI\_Send y MPI\_Recv**

Estas funciones permiten la comunicación bloqueante entre procesos. El proceso emisor envía datos a un proceso receptor especificando, entre otros parámetros, el identificador del proceso destino y un tag para identificar el mensaje. El receptor, a su vez, llama a MPI\_Recv esperando recibir el mensaje con el tag correspondiente.

**Manejo de Procesos**

La estructura condicional (if y else if) en el código permite asignar tareas específicas según el rango del proceso. Esto garantiza que solo el proceso 0 envíe datos y solo el proceso 2 los reciba y procese.

## Ejercicio 2 - Comunicación en cadena

## ¿Cómo funciona?

El programa propuesto utiliza la comunicación en cadena para que el dato viaje de un proceso al siguiente hasta alcanzar el último.

**Proceso 0**:

Inicia con un dato (0 en el ejemplo original).

Envía este dato al proceso 1 usando MPI\_Send.

Procesos intermedios (1 hasta size-2):

Cada proceso recibe el dato del proceso anterior mediante MPI\_Recv.

Incrementa el valor recibido (con recibido++).

Envía el nuevo valor al siguiente proceso con MPI\_Send.

Proceso final (rank == size - 1):

Recibe el dato, lo incrementa y, al no tener a quién enviarlo (ya que es el último), imprime el valor final.