



---

# MPI


---

Irvyn xicale cabrera



MATRICULA: 201963582

## EJEMPLO 1



```
from mpi4py import MPI
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
print("hello world from process ", rank)
```

```
(escuela) C:\Users\irvyn\OneDrive\Documents\programDistribuida\MPI python>mpiexec -n 4 python eje1.py
hello world from process 1
hello world from process 0
hello world from process 3
hello world from process 2
```

## Ejemplo 2

este código demuestra la comunicación entre procesos MPI, donde los procesos envían y reciben datos entre sí dependiendo de su rango.

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.rank
print("my rank is: ", rank)

if rank == 0:
    data = 10000000
    destination_process = 4
    comm.send(data, dest=destination_process)
    print("sending data %s " %data + \
          "to process %d" %destination_process)

if rank == 1:
    destination_process = 8
    data = "hello"
    comm.send(data, dest=destination_process)
    print("sending data %s " %data + \
          "to process %d" %destination_process)

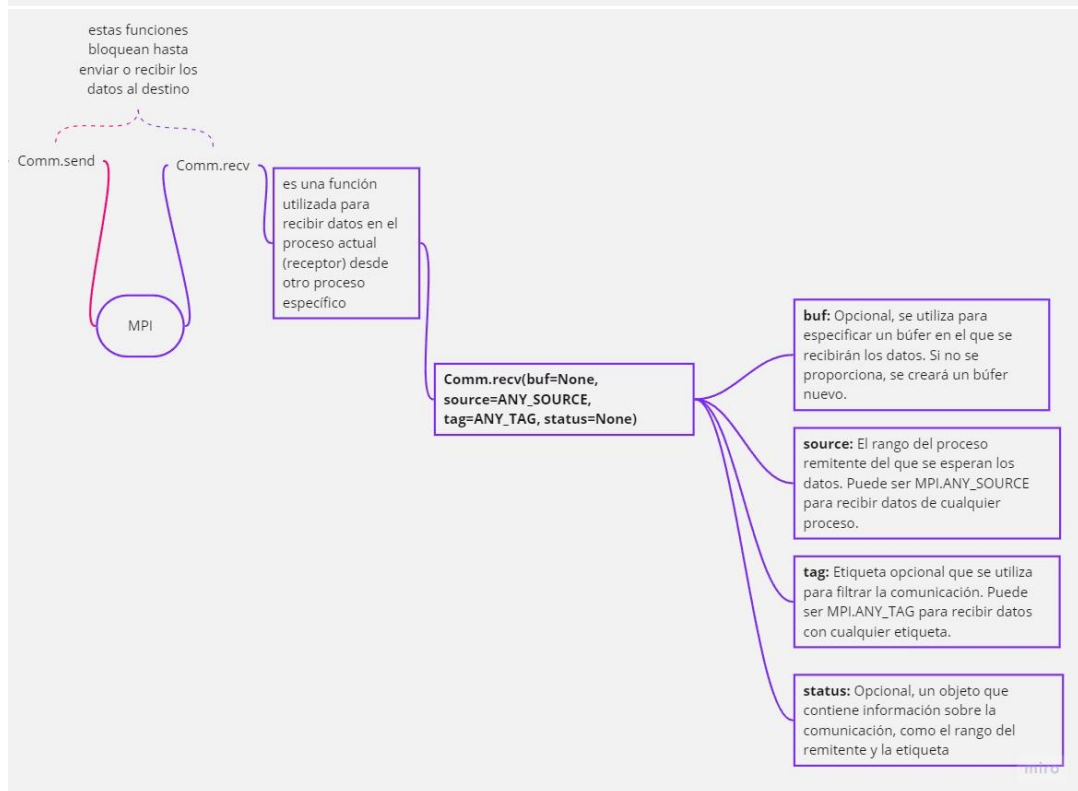
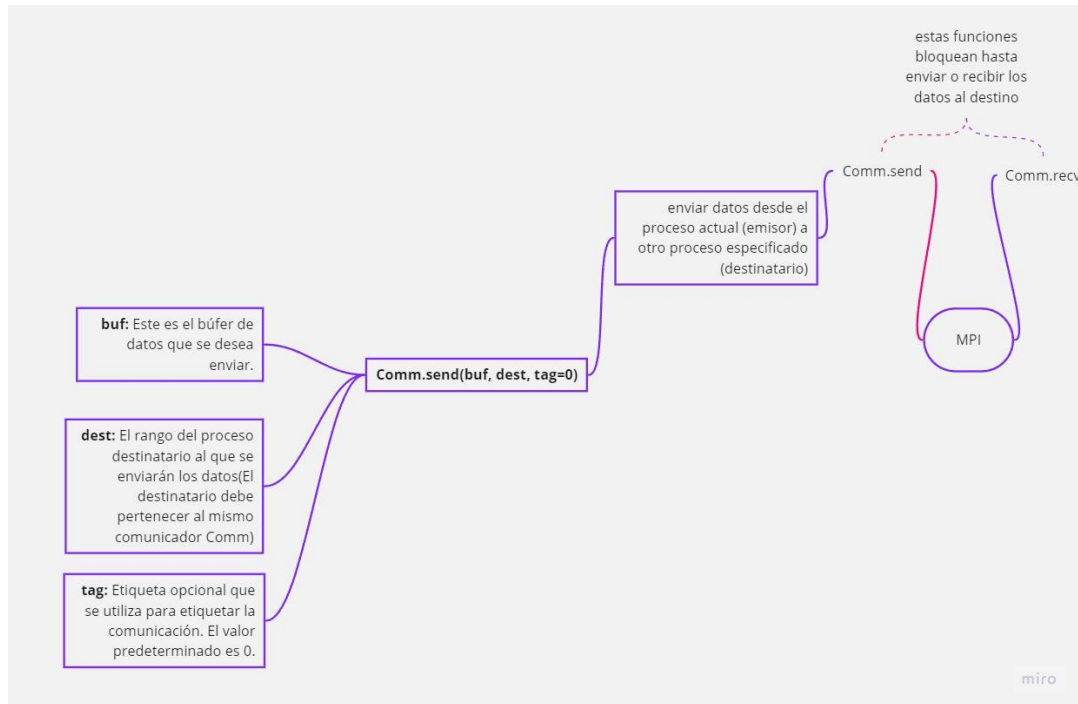
if rank == 4:
    data = comm.recv(source=0)
    print("data received is = %s" %data)

if rank == 8:
    data1 = comm.recv(source=1)
    print("data1 received is = %s" %data1)
```

### TERMINAL

```
(escuela) C:\Users\irvyn\OneDrive\Documents\programD
ribuida\MPI python>mpiexec -n 9 python eje2.py
my rank is: 2
my rank is: 3
my rank is: 7
my rank is: 6
my rank is: 0
sending data 10000000 to process 4
my rank is: 5
my rank is: 1
sending data hello to process 8
my rank is: 8
data1 received is = hello
my rank is: 4
data received is = 10000000
```

# MAPA MENTAL



### **¿en qué se parece el funcionamiento de las instrucciones mencionadas a los descriptores de archivos utilizados en la implementación de pipes o shared memory?**

- La comunicación de datos se realiza de manera punto a punto, lo que significa que los datos se envían desde un proceso (mediante send) a otro proceso (mediante recv).
- Cuando inicia el proceso de enviar se bloquea hasta terminar la transmisión y lo mismo pasa cuando se recibe

### **¿Cómo se puede evitar el interbloqueo?**

Implementando algún sistema de tiempo y errores para evitar que procesos se que queden en interbloqueo y una vez detectados tomar acciones. Pero lo mejor sería implementar una lógica de diseño para evitar estos casos

## EJEMPLO 3

implementa una comunicación punto a punto entre dos procesos (con rango 1 y 5). Ambos procesos envían y reciben datos entre sí.

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.rank
print("my rank is %i" %(rank))

if rank == 1:
    data_send = "a"
    destination_process = 5
    source_process = 5

    data_received = comm.recv(source=source_process)
    comm.send(data_send, dest = destination_process)

    print("sending data %s " %data_send + \
          "to process %d" %destination_process)
    print("data received is = %s" %data_received)

if rank == 5:
    data_send = "b"
    destination_process = 1
    source_process = 1

    comm.send(data_send, dest = destination_process)
    data_received = comm.recv(source=source_process)

    print("sending data %s:" %data_send + \
          "to process %d" %destination_process)
    print("data received is = %s" %data_received)
```

```
(escuela) C:\Users\irvyn\OneDrive\Documents\programDist
ribuida\MPI python>mpiexec -n 6 python eje3.py
my rank is 4
my rank is 3
my rank is 2
my rank is 0
my rank is 1
sending data a to process 5
data received is = b
my rank is 5
sending data b:to process 1
data received is = a
```

## EJEMPLO 4

este código divide la tarea de calcular la suma de números en múltiples procesos MPI, cada uno de los cuales calcula su propia suma parcial. Luego, las sumas parciales se envían al proceso raíz, donde se suman para obtener el resultado final de la suma.

```
from mpi4py import MPI
import numpy as np

if (__name__ == '__main__'):
    comm = MPI.COMM_WORLD
    myrank = comm.Get_rank()
    nproc = comm.Get_size()
    N = 1000
    startval = int(N * myrank / nproc + 1)
    endval = int(N + (myrank + 1) / nproc)
    partial_sum = np.array(0, dtype = 'i')
    for i in range(startval, endval + 1):
        partial_sum += i
    if (myrank != 0):
        comm.Send([partial_sum, 1, MPI.INT], dest = 0, tag = 7)
    else:
        tmp_sum = np.array(0, dtype = 'i')
        for i in range(1, nproc):
            comm.Recv([tmp_sum, 1, MPI.INT], source = i, tag = 7)
            partial_sum += tmp_sum
        print("The sum is {0}\n".format(partial_sum))
```

```
(escuela) C:\Users\irvyn\OneDrive\Documents\programDistribuida\MPI python>mpiexec -n 10 python eje4.py  
The sum is 3578751
```

## EJEMPLO 5

El proceso raíz inicializa la variable y luego la comparte con todos los demás procesos, lo que les permite acceder al mismo valor compartido.

```
from mpi4py import MPI  
  
comm = MPI.COMM_WORLD  
rank = comm.Get_rank()  
  
if rank == 0:  
    variable_to_share = 100  
else:  
    variable_to_share = None  
  
variable_to_share = comm.bcast(variable_to_share, root=0)  
print("process = %d" %rank + " variable share = %d" %variable_to_share)
```



```
(escuela) C:\Users\irvyn\OneDrive\Documents\programDist  
ribuida\MPI python>mpiexec -n 10 python eje5.py  
process = 0 variable share = 100  
process = 8 variable share = 100  
process = 9 variable share = 100  
process = 2 variable share = 100  
process = 3 variable share = 100  
process = 1 variable share = 100  
process = 4 variable share = 100  
process = 6 variable share = 100  
process = 5 variable share = 100  
process = 7 variable share = 100
```

## EJEMPLO 6

Cada proceso recibe un elemento de la lista y lo almacena en su propia variable recvbuf.

```
from mpi4py import MPI  
  
comm = MPI.COMM_WORLD  
rank = comm.Get_rank()  
  
if rank == 0:  
    array_to_share = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
else:  
    array_to_share = None  
  
recvbuf = comm.scatter(array_to_share, root = 0)  
print("process = %d" %rank + " variable shared = %d" %recvbuf)
```

```
(escuela) C:\Users\irvyn\OneDrive\Documents\programDist  
ribuida\MPI python>mpiexec -n 10 python eje6.py  
process = 1 variable shared = 2  
process = 2 variable shared = 3  
process = 8 variable shared = 9  
process = 0 variable shared = 1  
process = 9 variable shared = 10  
process = 3 variable shared = 4  
process = 4 variable shared = 5  
process = 6 variable shared = 7  
process = 5 variable shared = 6  
process = 7 variable shared = 8
```

## **Comm.scatter(sendbuf, recvbuf, root)**

se utiliza para distribuir datos desde el proceso raíz (especificado por root) a todos los demás procesos en el comunicador. Es útil cuando deseas dividir una tarea en partes iguales entre los procesos.

### **Parámetros:**

sendbuf: El arreglo de datos que se encuentra en el proceso raíz y se distribuirá a otros procesos.

recvbuf: El arreglo donde se almacenarán los datos distribuidos en cada proceso. En el proceso raíz, recvbuf puede ser None.

root: El rango del proceso raíz que posee los datos a distribuir. Los demás procesos deben especificar el mismo valor para root.

## **Comm.scatterv(sendbuf, recvbuf, root)**

Permite distribuir segmentos de diferentes tamaños desde un arreglo de datos (sendbuf) en el proceso raíz a los demás procesos en el comunicador. Cada proceso especifica cuántos elementos debe recibir, lo que permite distribuciones no uniformes. Esto puede

ser útil en situaciones en las que las tareas distribuidas tienen requisitos no uniformes de datos.

**Parámetros:**


sendbuf: El arreglo de datos que se encuentra en el proceso raíz y se distribuirá a otros procesos.

recvbuf: El arreglo donde se almacenarán los datos distribuidos en cada proceso. En el proceso raíz, recvbuf puede ser None.

root: El rango del proceso raíz que posee los datos a distribuir. Los demás procesos deben especificar el mismo valor para root.

**EJEMPLO 7**

Recopila los datos de los procesos y los agrega al array del proceso raíz 0



```

from mpi4py import MPI

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()
data = (rank + 1)**2

data = comm.gather(data, root=0)
if rank == 0:
    print("rank = %s" %rank +\
          "... received data to other process")
    for i in range(1,size):
        value = data[i]
        print(" process %s received %s from process %s" \
              %(rank, value, i))

```

```

(escuela) C:\Users\irvyn\OneDrive\Documents\programDist
ribuida\MPI python>mpiexec -n 10 python eje7.py
rank = 0... received data to other process
process 0 received 4 from process 1
process 0 received 9 from process 2
process 0 received 16 from process 3
process 0 received 25 from process 4
process 0 received 36 from process 5
process 0 received 49 from process 6
process 0 received 64 from process 7
process 0 received 81 from process 8
process 0 received 100 from process 9

```

**comm.Gather(sendbuf, root=0)**

se utiliza para recopilar datos desde todos los procesos y reunirlos en un solo proceso, el proceso raíz (especificado por `root`). Cada proceso proporciona un dato que se almacena en un arreglo en el proceso raíz.

### **Parámetros:**

`sendbuf`: El dato que cada proceso desea recopilar. En el proceso raíz, este argumento puede ser `None`.

`root` (opcional): El rango del proceso raíz que recopilará los datos. Si no se especifica, se asume que el proceso raíz es el proceso con rango 0.

## **`comm.Gatherv(sendbuf, recvbuf, root=0)`**

es similar a `Gather`, pero permite a cada proceso enviar diferentes cantidades de datos al proceso raíz. Cada proceso especifica cuántos elementos desea enviar, lo que permite una recopilación no uniforme.

### **Parámetros:**

`sendbuf`: El arreglo de datos que cada proceso desea recopilar. En el proceso raíz, este argumento puede ser `None`.

`recvbuf`: El arreglo donde se almacenarán los datos recopilados en el proceso raíz. En los procesos no raíz, este argumento debe ser `None`.

`root` (opcional): El rango del proceso raíz que recopilará los datos. Si no se especifica, se asume que el proceso raíz es el proceso con rango 0.

## EJEMPLO 8

Enviamos mediante Alltoall la data del proceso y recibimos la data de los otros procesos.

```
from mpi4py import MPI
import numpy

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()

senddata = (rank + 1) * numpy.arange(size, dtype = int)
recvdata = numpy.empty(size, dtype = int)
comm.Alltoall(senddata, recvdata)

print(" process %s sending %s receiving %s" \
      %(rank, senddata, recvdata))
```

```
(escuela) C:\Users\irvyn\OneDrive\Documents\programDistribuida\MPI python>mpiexec -n 5 python eje8.py
process 0 sending [0 1 2 3 4] receiving [0 0 0 0 0]
process 1 sending [0 2 4 6 8] receiving [1 2 3 4 5]
process 3 sending [ 0 4 8 12 16] receiving [ 3 6 9 12 15]
process 2 sending [ 0 3 6 9 12] receiving [ 2 4 6 8 10]
process 4 sending [ 0 5 10 15 20] receiving [ 4 8 12 16 20]
```

## EJEMPLO 9

Cada proceso contribuye con datos específicos, que se suman en el proceso raíz para obtener un resultado final. La suma se realiza con la operación de reducción MPI.SUM. Esto es útil para calcular sumas globales de datos distribuidos en un entorno paralelo.

```
import numpy
from mpi4py import MPI

comm = MPI.COMM_WORLD
size = comm.size
rank = comm.rank

array_size = 10
recvdata = numpy.zeros(array_size, dtype = int)
senddata = (rank + 1) * numpy.arange(array_size, dtype = int)

print(" process %s sending %s " %(rank, senddata))

comm.Reduce(senddata, recvdata, root=0, op=MPI.SUM)
print('on task',rank,'after Reduce:  data = ',recvdata)
```

```
process 1 sending [ 0  2  4  6  8 10 12 14 16 18]
on task 1 after Reduce:  data = [0 0 0 0 0 0 0 0 0 0]
process 4 sending [ 0  5 10 15 20 25 30 35 40 45]
on task 4 after Reduce:  data = [0 0 0 0 0 0 0 0 0 0]
process 2 sending [ 0  3  6  9 12 15 18 21 24 27]
on task 2 after Reduce:  data = [0 0 0 0 0 0 0 0 0 0]
process 0 sending [0 1 2 3 4 5 6 7 8 9]
on task 0 after Reduce:  data = [ 0 15 30 45 60 75 90 105 120 135]

(escuela) C:\Users\irvyn\OneDrive\Documents\programDistribuida\MPI python>
```

- **MPI.SUM:** Realiza una suma de todos los valores distribuidos entre los procesos y devuelve la suma total.
- **MPI.PROD:** Calcula el producto de todos los valores distribuidos entre los procesos y devuelve el producto total.

- **MPI.MAX:** Encuentra el valor máximo entre todos los valores distribuidos y devuelve el valor máximo.
- **MPI.MIN:** Encuentra el valor mínimo entre todos los valores distribuidos y devuelve el valor mínimo.
- **MPI.LAND:** Realiza una operación lógica "AND" entre todos los valores booleanos distribuidos y devuelve "VERDADERO" si todos los valores son verdaderos, de lo contrario, devuelve "FALSO".
- **MPI.LOR:** Realiza una operación lógica "OR" entre todos los valores booleanos distribuidos y devuelve "VERDADERO" si al menos un valor es verdadero, de lo contrario, devuelve "FALSO".
- **MPI.LXOR:** Realiza una operación lógica "XOR" entre todos los valores booleanos distribuidos y devuelve "VERDADERO" si un número impar de valores son verdaderos, de lo contrario, devuelve "FALSO".
- **MPI.BAND:** Realiza una operación "AND" a nivel de bits entre todos los valores enteros distribuidos y devuelve el resultado.
- **MPI.BOR:** Realiza una operación "OR" a nivel de bits entre todos los valores enteros distribuidos y devuelve el resultado.
- **MPI.BXOR:** Realiza una operación "XOR" a nivel de bits entre todos los valores enteros distribuidos y devuelve el resultado.
- **MPI.MINLOC:** Encuentra el valor mínimo entre todos los valores distribuidos y también devuelve la ubicación (rango) del proceso que contiene el valor mínimo.
- **MPI.MAXLOC:** Encuentra el valor máximo entre todos los valores distribuidos y también devuelve la ubicación (rango) del proceso que contiene el valor máximo.

## Ventajas de la unificación de long e int en Python

- La unificación de long e int en Python 3 elimina la necesidad de preocuparse por la diferencia entre estos tipos de datos.
- La unificación permite realizar operaciones aritméticas con enteros grandes de manera más sencilla y coherente, ya que no es necesario cambiar entre tipos int y long.
- hace que el código sea más claro y fácil de entender, ya que no es necesario considerar las diferencias de tipo en la mayoría de los casos.