PRACTICA 2

Modelo de comunicación

Punto a punto

María González Herrero

Santiago Molpeceres Díaz

UNIVERSIDAD NEBRIJA

INDICE

Ejercicio 1

En el enunciado del primer ejercicio nos pide que mediante la comunicación punto a punto entre dos mensajes que rebotan tantas veces como el límite se haya impuesto.

La comunicación punto a punto se presenta como una configuración de red de modo que los computadores son los clientes y a su vez también son los servidores.

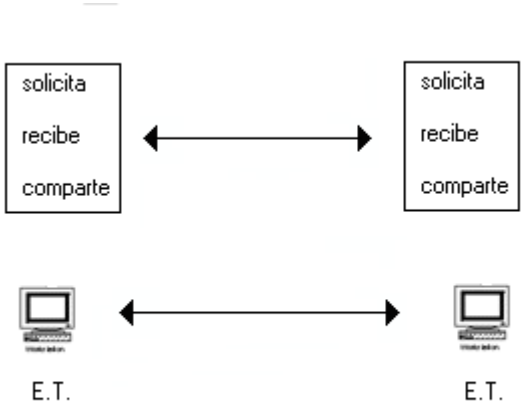

Imagen 1: Diagrama de comunicación punto a punto

Para ello hemos tomado como modelo base el ejemplo de comunicación punto a punto del enunciado de la práctica 2 dada en clase y hemos añadido pequeñas actualizaciones para cumplir con el enunciado propuesto.

Código

Primero debemos tener en cuenta que el código que ponemos es general para todos los procesos. Por eso es necesario hacer distinciones entre procesos.

Esta práctica nos pide que pasemos un mensaje en bucle entre dos procesos. Por tanto, declaramos el mensaje que vamos a pasar y el resto de las variables que van a ser necesarias para nuestro programa.

```
5   int rank,count,size;
6   char msg[20]="hello word";
```

En este caso vamos a pasar un array de chars de un máximo de 20 caracteres, con el mensaje “hello word”.

Como hemos comentado antes debemos hacer distinciones entre procesos, para este caso, vamos a poner el proceso 0 como proceso Maestro, y el resto de los procesos (el proceso 1) tomaran el rol de esclavos.

```
13  if(rank==0){
14      printf("Soy el nodo maestro y envio el mensaje : -hello word –
                  .\n\n");
15      printf("Introduce cuantas veces el nodo maestro esta dispuesto a
            escuchar\n");
16      scanf("%d", &count);
17      MPI_Send(&count, 1, MPI_INT, 1, 101, MPI_COMM_WORLD);
18      MPI_Send(&msg, 13, MPI_CHAR, 1, 100, MPI_COMM_WORLD);
19      printf("Proceso %d envia : %s  - al proceso %d\n", rank,msg,1);
20  }
```

En la línea 13, es donde empiezan las instrucciones del nodo maestro. Primero queremos hablar con el usuario para preguntarle cuantas veces el mensaje va a estar rebotando, que es lo mismo que preguntar, cuantas veces el nodo maestro está dispuesto a escuchar y la respuesta introducida por teclado la vamos a guardar en la variable “count”(línea 16), que ha sido declarada anteriormente (línea 6).

Con el SEND de la línea 17, le vamos a pasar al nodo esclavo cuantas veces debe devolvernos el mensaje, y con el SEND de la línea 18, enviaremos el mensaje.

El MPI_Send se encarga de enviar un mensaje bloqueante de un origen a un destino (bloqueante se refiere que hasta que el mensaje no se haya enviado no se puede continuar con el resto de la ejecución).

Si seguimos avanzando en el programa nos encontraremos con esta instrucción.

```
22 if(rank!=0)MPI_Recv(&count,1,MPI_INT,0,101, MPI_COMM_WORLD,&status);
```

Como estamos usando instrucciones SEND y RECIVED de bloqueo, eestas instrucciones le dicen al resto de nodos que no son maestro (el proceso 1), que deben quedarse a la espera de un mensaje que contenga un INT, para poder continuar con su ejecución.

Este mensaje es el de la línea 17. Cuando nos llegue el número de veces que debemos devolver el mensaje, empezaremos a ejecutar el resto del programa. Si la línea 17 no existiera, el nodo esclavo siempre se quedaría a la escucha y no funcionaría como debe.

En la línea 23 encontramos un While(true) quiere decir que lo que haya dentro de este while se va a ejecutar todo el rato. Y como hemos comentado antes, este programa es un programa general para ambos nodos, así que tendremos que observar sus distinciones.

```
23 while (1)
24 {
25      char msg2[20];
26      if(rank==0){
27      MPI_Recv(&msg2, 13, MPI_CHAR, 1, 100, MPI_COMM_WORLD,&status);
28      printf("Proceso %d ha recibido %s \n", rank, msg);
29          --count;
30          if(count==0){
31              break;
32          }else{
33           MPI_Send(&msg2,13, MPI_CHAR, 1, 100, MPI_COMM_WORLD);
34           printf("Proceso %d envia :%s- al proceso %d\n", rank,msg,1);
35            }
26      }else{
```

A partir de la línea 26, hasta la 36, es como debe actuar el nodo maestro. Ya que este nodo ha sido el primero en enviar el mensaje, este se queda a la espera de la recepción del mensaje del nodo esclavo (línea 27).

Debemos recordar que le hemos dicho al nodo maestro cuantas veces debe escuchar y lo hemos guardado en "count". Por eso la línea 29, resta 1 a la variable "count".

Si el count es 0, que es equivalente a decir que nos han llegado dos mensajes, debemos salir del while, por eso el break de la línea 31. Por el contrario, devolveremos el mensaje que hemos almacenado en la variable "msg2".

A partir de la línea 36, hasta la 48 se define el programa que va a ejecutar el nodo esclavo.

```
36        }else{
37          MPI_Recv(&msg2, 13,MPI_CHAR, 0, 100, MPI_COMM_WORLD,&status);
38          printf("Proceso %d ha recibido %s \n", rank, msg);
39          --count;
40          if(count==0){
41           MPI_Send(&msg2, 13,MPI_CHAR, 0, 100, MPI_COMM_WORLD);
42           printf("Proceso %d envia :%s- al proceso %d\n", rank,msg,0);
43           break;
44          }else{
45        MPI_Send(&msg2, 13, MPI_CHAR, 0, 100, MPI_COMM_WORLD);
46        printf("Proceso %d envia : %s  - al proceso %d\n", rank,msg,0);
47          }
48      }
```

Como es el primero en recibir un mensaje, en la línea 37, se queda en espera de recibir un mensaje del nodo maestro, guardando el mensaje en la variable "msg2".

La línea 39 resta 1 al contador de nodo esclavo, pero aquí viene la diferencia.

El nodo esclavo va, una escucha más adelantado que el nodo maestro, por tanto, si el contador es 2, que es lo mismo que decir que hemos recibido dos mensajes, debe devolver un mensaje más, y terminar la ejecución del programa general y morir y para ello debemos salir del while, con el break de la línea 43.

Si no se han cumplido las dos recepciones implica que simplemente debemos devolverlo que es la línea 45.

```
23   while (1)
24   {
       . . .
49    }
50  printf("Proceso %d termina\n", rank);
51  MPI_Finalize();
```

La línea 51 finalizará el proceso.

Recuerda:

Las líneas 14, 15, 19, 28, 34, 38, 42, 46, 50.

Son mensajes que aparecerán por consola para un mejor seguimiento del ejercicio, no son relevantes para la práctica.

Esquema de proceso

Para un problema en el que le decimos al maestro que escuche solo 2 veces.

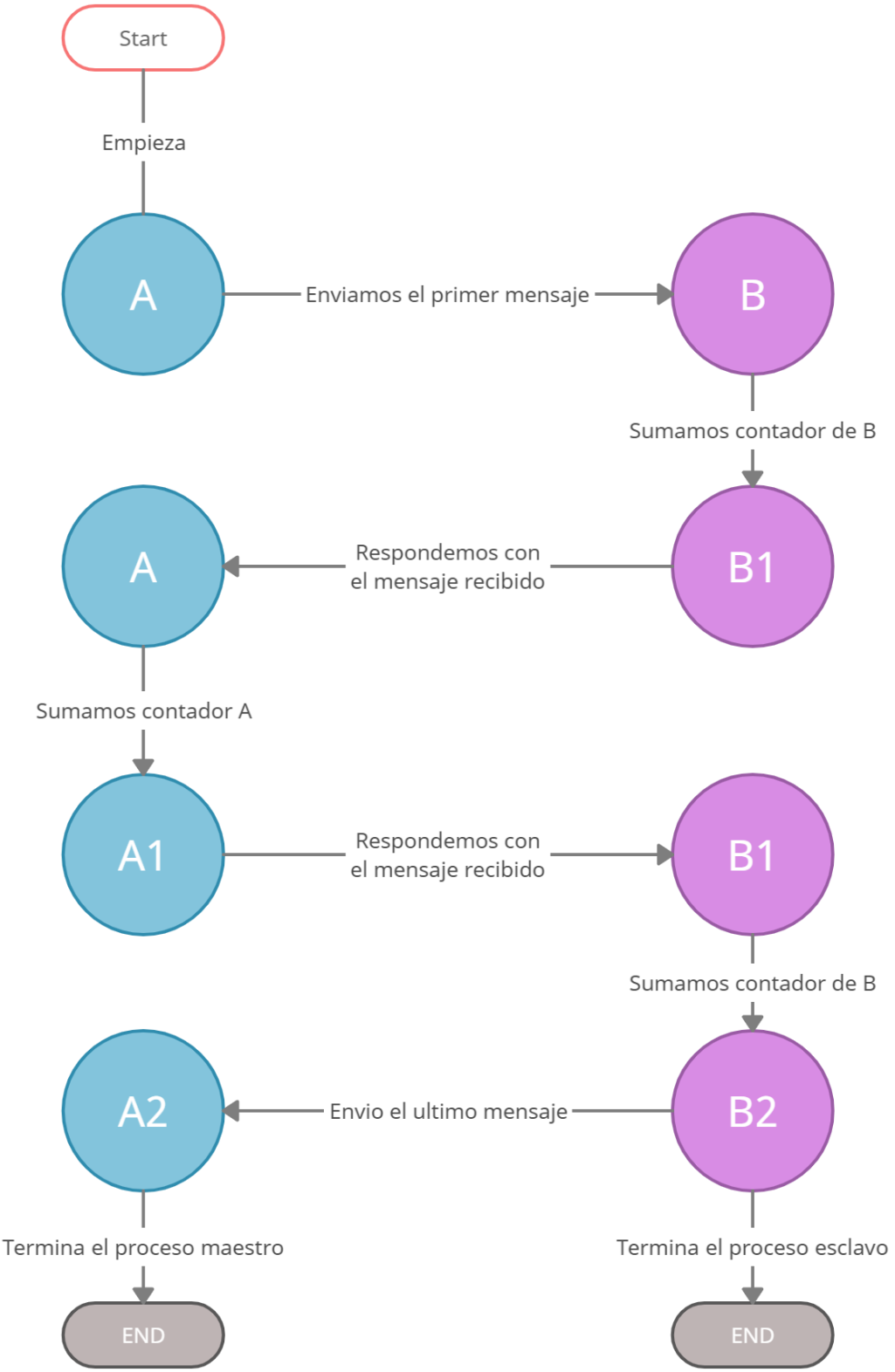

Imagen 2: Diagrama de esquema de proceso

Diagrama de flujo

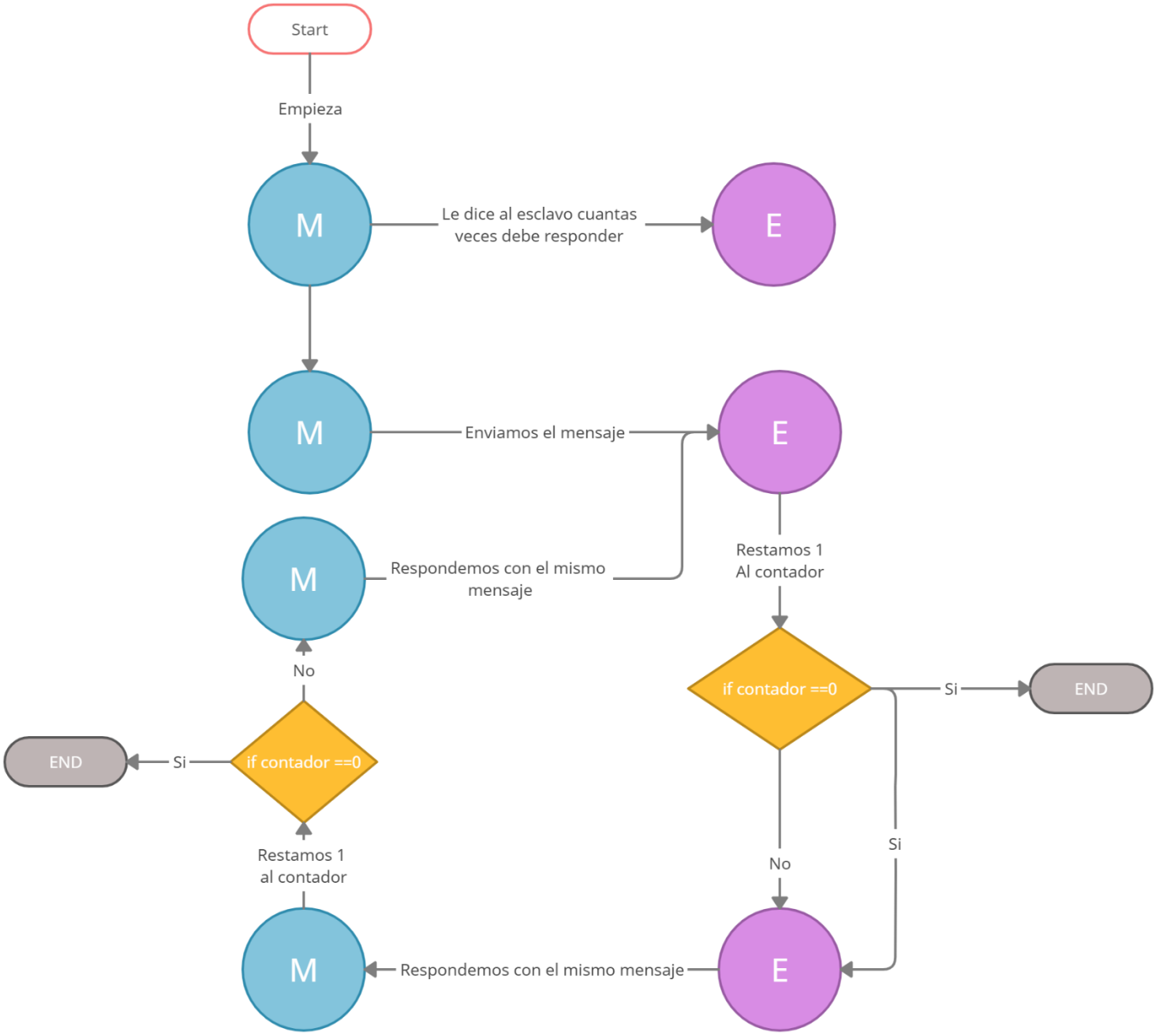

Imagen 3: Diagrama de flujo

Salida por consola

Ajustado para un fácil seguimiento del programa.

```
Soy el nodo maestro y envio el mensaje : -hello word -.

Introduce cuantas veces el nodo maestro esta dispuesto a escuchar
2
Proceso 0 envia : hello word  - al proceso 1
Proceso 1 ha recibido hello word
Proceso 1 envia : hello word  - al proceso 0
Proceso 0 ha recibido hello word
Proceso 0 envia : hello word  - al proceso 1
Proceso 1 ha recibido hello word
Proceso 1 envia : hello word  - al proceso 0
Proceso 1 termina
Proceso 0 ha recibido hello word
Proceso 0 termina
```

Imagen 4: Salida por consola del ejercicio 1

Ejercicio 2

Para este segundo ejercicio seguimos con la comunicación punto a punto definida antes en el ejercicio anterior. Pero esta vez añadimos unos floats de tamaño fijo (10000). Haciendo que todas las posiciones de un array (A) se inicializan con el rango respectivo para cada proceso. Siendo el array A el búfer para las operaciones de SEND y el B para la de RECEIVE.

Tal y como se muestra en el esquema dado en la práctica hemos creado los procesos.

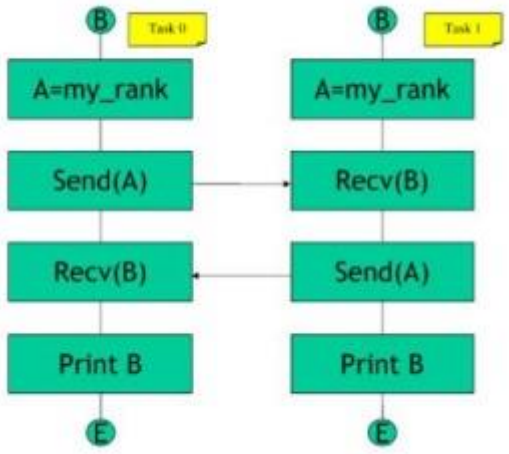

Imagen 5: Diagrama de 2 procesos usando comunicación punto a punto

Código

IMPORTANTE: Este código está hecho a un límite de 5 elementos para que sea más visual.
En el caso de que quiera usar un número distinto, como 10000:

- la variable Lenght debe obtener ese valor
- Los array deben tener ese valor

Primero vamos a declarar las variables que vamos a utilizar.

```
    int rank,count=0,size,lenght=5,max;
    float a[5]={};
```

- Rank: Recogerá el valor del proceso.
- Count: Contador que llevará el registro de los envíos.
- Size: recoge el valor del numero de nodos en el programa.
- Lenght: Recogerá la dimensión del array.
- Max: Indica hasta donde debe llegar count.
- A: Array que vamos a pasar.

Antes de decirle al proceso que hacer, el enunciado nos pide que los elementos del array "A" deben ser el valor del rango, para ello, después de iniciar el proceso debemos rellenar el array "A" de su valor.

```
  for(int i=0;i<lenght;i++){
      a[i]=(float)rank;
  }
```

El primer comportamiento del nodo maestro (0), es saber cuantos mensajes debe enviar. Para ello le pedimos al usuario que nos indique cuantas veces quiere que el proceso se repita, guardando la respuesta en la variable "max".

```
 printf("Introduce cuantas veces el nodo maestro esta dispuesto
      enviar\n");
 scanf("%d", &max);
```

Este máximo también debe saberlo el nodo esclavo por eso, le pasamos el valor de max al siguiente nodo.

```
MPI_Send(&max, 1, MPI_INT, 1, 101, MPI_COMM_WORLD);
```

Y después el mensaje a enviar.

```
MPI_Send(&a, lenght, MPI_FLOAT, 1, 100, MPI_COMM_WORLD);
```

Como ya hemos hecho un envío, debemos subir el contador.

```
count++;
```

En la línea 24, nos volvemos a encontrar con esta línea de código.

```
if(rank!=0)MPI_Recv(&max, 1, MPI_INT, 0, 101, MPI_COMM_WORLD,&status);
```

Lo que indica que el nodo esclavo va a estar bloqueado hasta la recepción de un int, el cual es el máximo de recepciones que va a tener.

Ahora vamos a ver cómo trabaja cada proceso.

```
        float b[5]={};
        int envio_recibo;
```

Los dos procesos van a compartir una declaración en la que B, es una array en la que vamos a guardar los datos que nos envían; esta es declarada en cada ciclo para que sea totalmente nueva y no haya confusiones de si realmente el programa esta o no guardando los datos.

También vamos a encontrar la variable “envio_recibo”, que dado que solo hay dos nodos, si estamos en el nodo maestro , enviamos y recibimos del nodo esclavo 1 y si estamos en el nodo esclavo , solo recibimos y enviamos al nodo maestro 0.

Para el proceso maestro.

```
29      if(rank==0){
30          //Analizo los envíos
31          envio_recibo=1;
32          if(count<max){
33        //como ya he enviado 1, espero la proxima recepcion
34 MPI_Recv(&b,lenght, MPI_INT, envio_recibo,
      100,MPI_COMM_WORLD,&status);

35 printf("El proceso %d ha recibido\n",rank);

36 int j=0;while (j<lenght){printf("Iteracion %i-
      %0.1f\n",j,b[j]);j=j+1;};

37 MPI_Send(&a, lenght, MPI_FLOAT, envio_recibo, 100, MPI_COMM_WORLD);

38 printf("Proceso %d envia : A - al proceso %d\n", rank,envio_recibo);

39     count++;

40     }else{

41  MPI_Recv(&b,lenght, MPI_INT, envio_recibo, 100,
      MPI_COMM_WORLD,&status);

42 printf("El proceso %d ha recibido\n",rank);
43 int j=0;while (j<lenght){printf("Iteracion %i-
      %0.1f\n",j,b[j]);j=j+1;};
44     break;
45     }
47   }else if(--){
```

Primero valoramos si hemos enviado ya todos los mensajes que debemos.

Si no es el caso:

Como ha sido el primero en enviar mensaje, con la línea 34, nos quedamos esperando a la recepcion de valores que deben introducirse en el array de B.

Se nos pide que imprimamos los datos recibidos, por tanto, recorremos el array B con la línea 36, y mostramos lo que contiene y volvemos a enviar nuestro array A al nodo esclavo (1).

Si no es el caso:

Esperamos la última recepcion, imprimimos los datos que se nos pasa y cerramos el proceso.

Para el proceso esclavo.

```
47          }else if(rank==1){
48              //Analizo las recepciones
49            envio_recibo=0;
50                MPI_Recv(&b,lenght, MPI_INT,envio_recibo,100,
      MPI_COMM_WORLD,&status);
51                printf("El proceso %d ha recibido\n",rank);
52               int j=0;while (j<lenght){printf("Iteracion %i –
      %0.1f\n",j,b[j]);j=j+1;};
53                count++;
54                MPI_Send(&a, lenght, MPI_FLOAT, envio_recibo, 100,
      MPI_COMM_WORLD);
55                printf("Proceso %d envia : A - al proceso %d\n",
      rank,envio_recibo);
                  if(count==max){
                      break;
                  }
          };
```

Es un proceso en bucle ya que siempre debe cumplir con su propósito, pero únicamente mira si debe parar de trabajar.

Con la línea 37, nos quedamos a la espera del mensaje, guardando los datos recibidos en el array b; la línea 52 imprime el array b y para finalizar enviamos el array "A" perteneciente al nodo esclavo. Siempre se va a cumplir este proceso. Para saber si tiene que seguir "vivo" el proceso, debemos saber cuántas recepciones hemos tenido; entonces si el contador es igual al máximo de recepciones, el proceso esclavo muere.

<u>Recuerda:</u>

Las líneas 21, 38, 55.

Son mensajes que aparecerán por consola para un mejor seguimiento del ejercicio, no son relevantes para la práctica.

Esquema de proceso

Para un problema en el que le decimos al maestro envíe solo 2 veces.

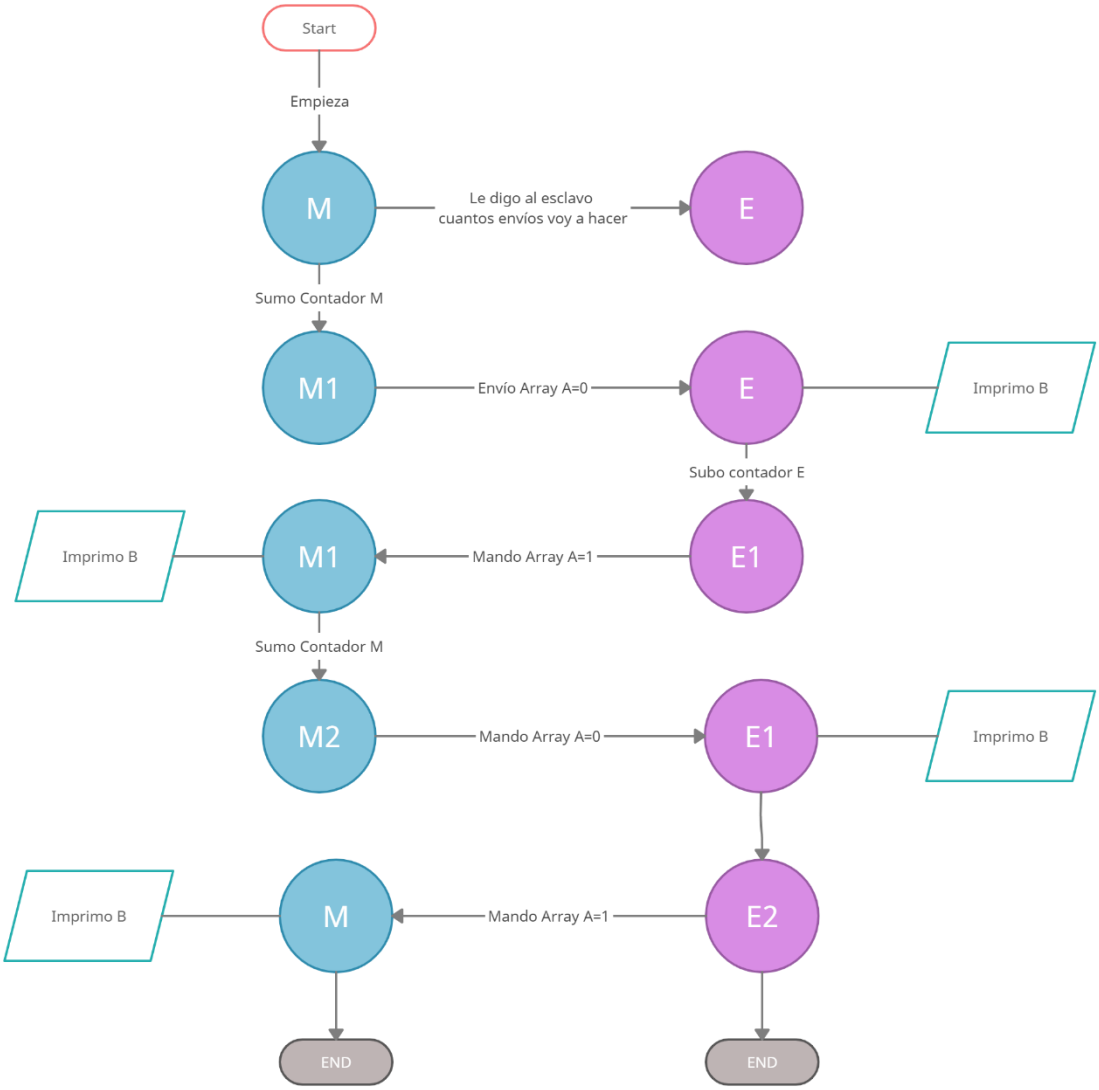

Imagen 6: Diagrama de un esquema de proceso para el ejercicio 2.

Diagrama de Flujo

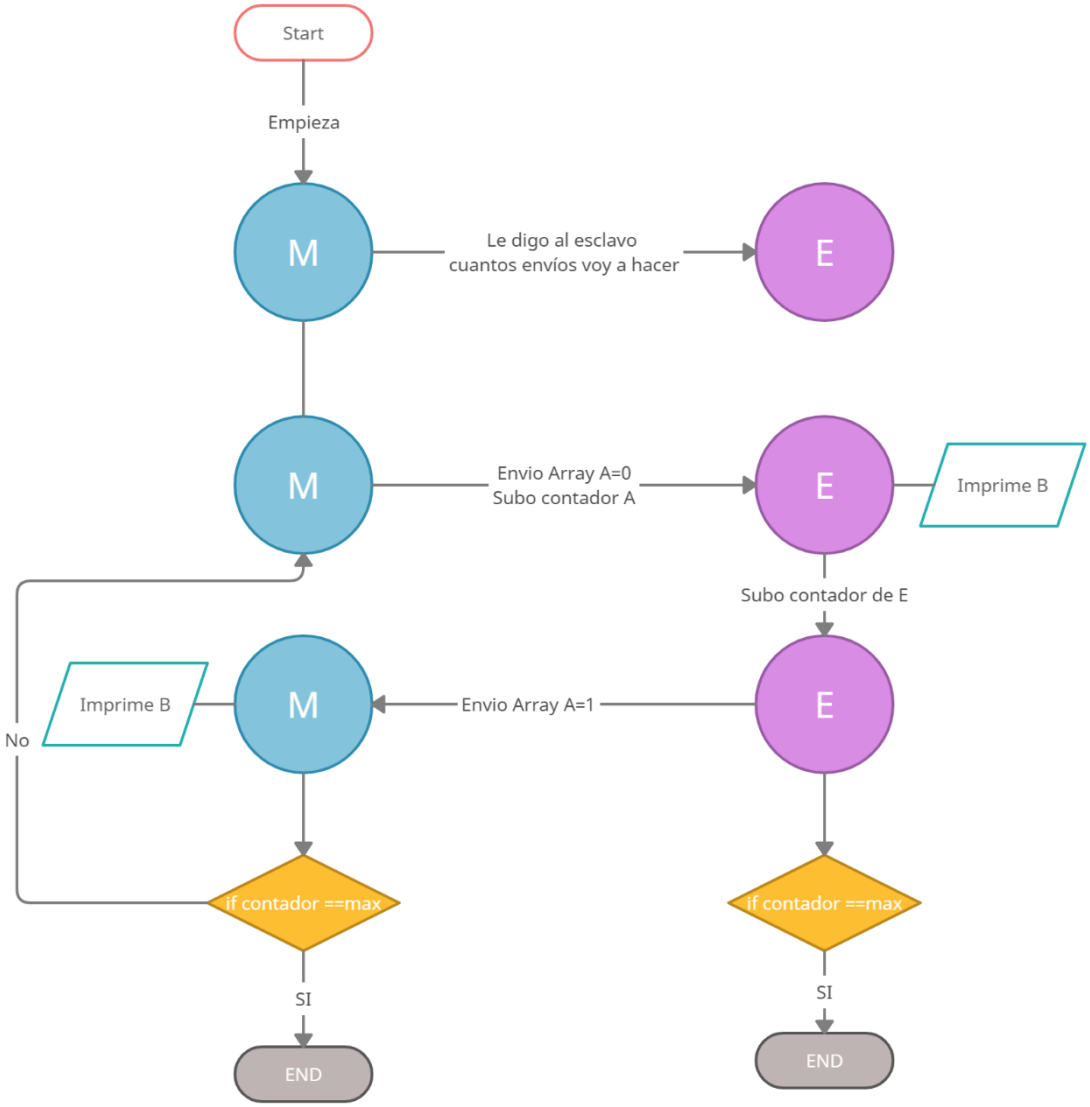

Imagen 7: Diagrama de flujo para el ejercicio 2.

Salida por consola

Ajustado para un fácil seguimiento del programa.

```
Introduce cuantas veces el nodo maestro esta dispuesto a enviar
2
Proceso 0 envia : A - al proceso 1
El proceso 1 ha recibido
Iteracion 0 - 0.0
Iteracion 1 - 0.0
Iteracion 2 - 0.0
Iteracion 3 - 0.0
Iteracion 4 - 0.0
Proceso 1 envia : A - al proceso 0
El proceso 0 ha recibido
Iteracion 0 - 1.0
Iteracion 1 - 1.0
Iteracion 2 - 1.0
Iteracion 3 - 1.0
Iteracion 4 - 1.0
Proceso 0 envia : A - al proceso 1
El proceso 1 ha recibido
Iteracion 0 - 0.0
Iteracion 1 - 0.0
Iteracion 2 - 0.0
Iteracion 3 - 0.0
Iteracion 4 - 0.0
Proceso 1 envia : A - al proceso 0
Proceso 1 termina
El proceso 0 ha recibido
Iteracion 0 - 1.0
Iteracion 1 - 1.0
Iteracion 2 - 1.0
Iteracion 3 - 1.0
Iteracion 4 - 1.0
Proceso 0 termina
```

Imagen 8: Salida de por consola del ejercicio 2.

Ejercicio 3

Por último, debemos de implementar un código de una comunicación punto a punto, que lleva a cabo una operación de send/receive circular, la cual se nos ha proporcionado en el enunciado de la práctica.

Diagrama de flujo

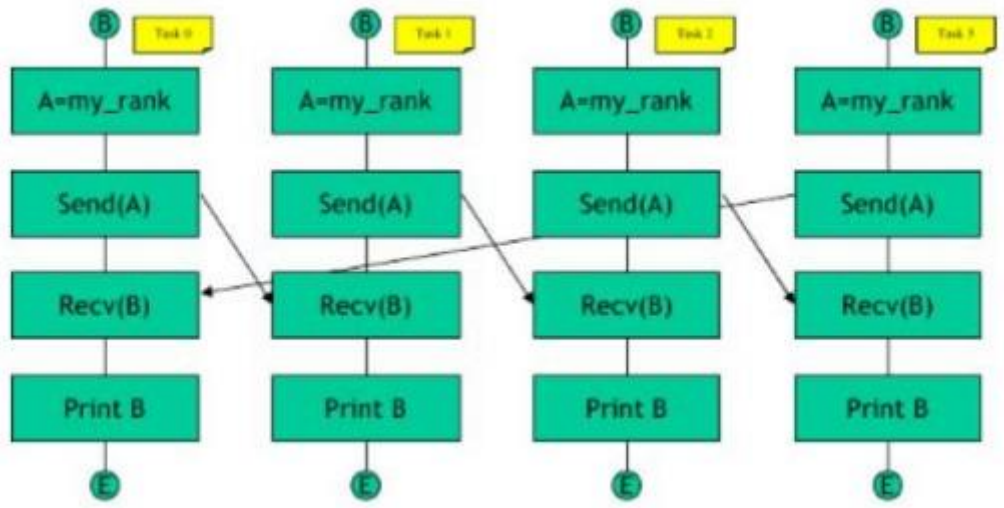

Imagen 9: Diagrama de una comunicación punto a punto con operaciones circulares

Código

IMPORTANTE: Este código este hecho a un límite de 5 elementos para que sea más visual.
En el caso de que quiera usar un número distinto, como 10000:

- la variable lenght debe obtener ese valor
- Los array deben tener ese valor

Para este código también debemos hacer distinciones entre nodos.

Primero rellenamos el array A con el valor del proceso.

```
for(int i=0;i<lenght;i++){
    a[i]=(float)rank;
}
```

Para el nodo maestro (0).

```
19 MPI_Send(&a, lenght, MPI_FLOAT, (rank + 1) % size,
      TAG_MESSAGE,MPI_COMM_WORLD);
20 printf("Proceso %d envia : A - al proceso %d\n", rank,(rank + 1) %
      size);
21 MPI_Recv(&b, lenght, MPI_FLOAT, (rank + size - 1) % size, TAG_MESSAGE,
      MPI_COMM_WORLD,&status);
22 printf("Proces %d received\n", rank);
23 int j=0;while (j<lenght){printf("Iteracion %i –
      %0.1f\n",j,b[j]);j=j+1;};
```

Enviamos el Array A al nodo siguiente (línea 19). Y nos quedamos esperando la respuesta del último nodo (línea 21). Una vez recibido el array del último nodo, imprimimos el array (línea 23).

Para el resto de los nodos.

```
25 printf("Proceso %d envia : A - al proceso %d\n", rank,(rank + 1)
       %size);
26 MPI_Send(&a, lenght, MPI_FLOAT, (rank + 1) % size, TAG_MESSAGE,
      MPI_COMM_WORLD);
27 MPI_Recv(&b, lenght, MPI_FLOAT, (rank + size - 1) % size, TAG_MESSAGE,
      MPI_COMM_WORLD,&status);
26 printf("Proces %d received\n", rank);
27 int j=0;while (j<lenght){printf("Iteracion %i –
      %0.1f\n",j,b[j]);j=j+1;};
```

Enviamos el array A de cada proceso al siguiente (línea 25).
Luego, nos quedamos a la espera de que nos llegue el array de “A” del nodo N-1(línea 26), y finalmente lo imprimimos (línea 26 y 27).

Salida por consola

Ajustado para un mejor seguimiento del programa

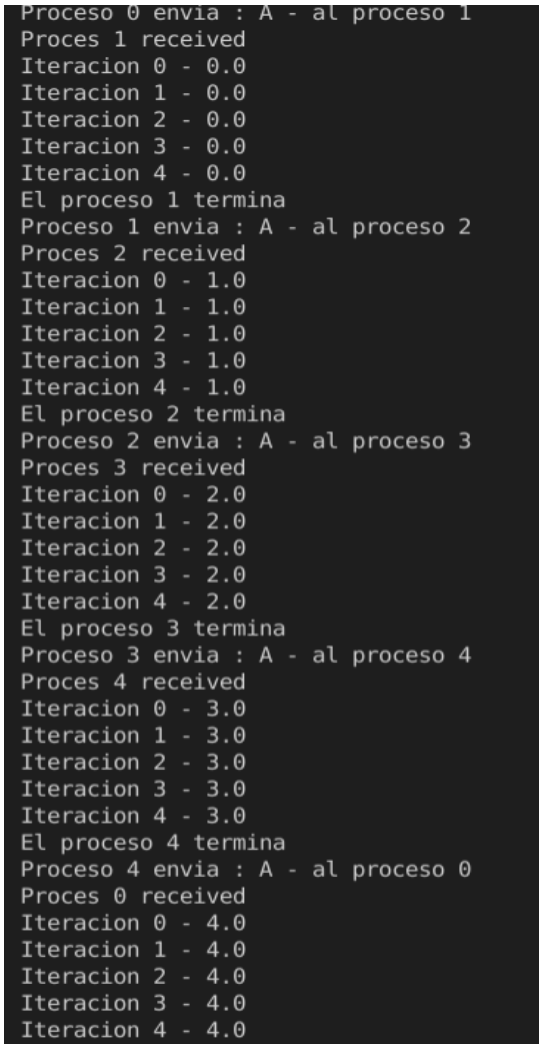

Imagen 10: Salida por consola del ejercicio 3

WEBGRAFÍA

https://puntoapuntohalfduplex.blogspot.com/2012/01/comunicacion-punto-punto.html

https://lsi2.ugr.es/jmantas/ppr/ayuda/mpi_ayuda.php?ayuda=MPI_Send