Arquitectura de Computadores

- Pablo Rayón Zapater
- Fernando Pérez Ballesteros
- José María Fernández

Práctica 2

UNIVERSIDAD NEBRIJA

Índice

Ejercicio 1

Explicar en qué consiste la comunicación punto a punto, los tipos de primitivas que hay en MPI y las ventajas y desventajas de este tipo de comunicación.

La comunicación punto a punto es un sistema basado en el envío y recepción de mensajes o datos por medio de enlaces individuales, esto quiere decir que únicamente cada uno de los sistemas terminales es capaz de comunicarse con otro de manera simultánea.
En MPI se puede implementar esto con las funciones SEND Y RECV o LSEND Y LRECV.
La primera pareja de instrucciones es bloqueante, esto quiere decir que el proceso se queda pausado o bloqueado hasta que estas funciones finalicen, el otro par de funciones sin embargo permite que el flujo del proceso siga en ejecución una vez son llamadas estas funciones, por lo que es posible que otro proceso reciba o envíe antes de que la instrucción anterior se complete totalmente.
Esto se aprecia en el ejercicio 2, donde en un mismo proceso con varias iteraciones no es posible avanzar a la siguiente iteración con la llamada a recepción de mensajes hasta que esa misma instrucción de la iteración previa finaliza totalmente, por eso aparecen en orden las recepciones del mensaje. Po otro lado tenemos el envió de mensajes, pese a que también es una llamada bloqueante, al ser ejecutada por diferentes procesos no influye en el orden de ejecución de estos, por lo que cada proceso puede enviar al tiempo que sea, por esta razón observamos que n hay ningún orden a la hora de ejecutarse los envíos.

Ejercicio 2

Implementar un código donde cada proceso del comunicador inicializa una variable con un valor aleatorio (distinto para cada proceso) y se la envía al proceso 0 utilizando las primitivas bloqueantes de comunicación punto a punto. Una vez recibidas todas, el proceso 0 imprime por pantalla los valores ordenados por número de proceso.

```
Soy el proceso 2 y he inicializado la variable en : 9
Soy el proceso 1 y he inicializado la variable en : 6
Soy el proceso 3 y he inicializado la variable en : 9
Soy el proceso 0 y he recibido 6 del proceso 1
Soy el proceso 0 y he recibido 9 del proceso 2
Soy el proceso 0 y he recibido 9 del proceso 3
```

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

void main(int argc, char **argv)
{
    int rank, count, rcv;

    MPI_Status status;
    MPI_Init(&argc, &argv);

    MPI_Comm_size(MPI_COMM_WORLD, &count);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if (rank == 0)
    {
        for (int i = 1; i < count; i++)
        {
            MPI_Recv(&rcv, 1, MPI_INT, i, 100, MPI_COMM_WORLD, &status);
            printf("Soy el proceso 0 y he recibido %d del proceso %d\n", rcv, i);
        }
    }
    else
    {
        srand(time(0) + rank);
        rcv = rand() % 10;
        MPI_Send(&rcv, 1, MPI_INT, 0, 100, MPI_COMM_WORLD);
        printf("Soy el proceso %d y he inicializado la variable en : %d\n", rank, rcv);
    }

    MPI_Finalize();
}
```

Como se puede apreciar en las imágenes, el programa consigue inicializar variables aleatorias dependientes del tiempo y del propio rango del proceso que las ejecuta para así conseguir un numero diferente por cada proceso, estas son inicializadas en la primera instrucción del else, este else perteneciendo a la sección del código que dicta como se comportaran los procesos que no sean el 0, los cuales son los únicos que envían. Por otro lado, la sección superior de código, que se ve envuelta en un condicional que comprueba el rango del proceso a ejecutar, se encarga de recibir tantos mensajes como numero de procesos haya, sin incluirse a el mismo. Por lo que se realiza una comunicación punto a punto, en la que cada proceso envía a un solo receptor (Proceso 0) y este solo recibe comunicaciones individuales de los otros.

Ejercicio 3

Implementar un código donde utilizando comunicación punto a punto tres procesos rebotan continuamente los mensajes entre sí, hasta que deciden detenerse una vez alcanzado límite autoimpuesto.

```
MPI_Status status;
MPI_Init(&argc, &argv);

MPI_Comm_size(MPI_COMM_WORLD, &count);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
for (int i = 1; i <= iter; i++)
{
    if (rank == 0)
    {

        if (way == 1)
            MPI_Send(msg, 20, MPI_CHAR, 1, 100, MPI_COMM_WORLD);
        if (way == 0)
            MPI_Recv(&msg, 20, MPI_CHAR, 1, 100, MPI_COMM_WORLD, &status);
        printf("Soy el proceso 0 y he recibido %s del proceso 1 en iteracion %d \n", msg, i);
    }

    else if (rank == count - 1)
    {
        if (way == 0)
            MPI_Recv(&msg, 20, MPI_CHAR, rank - 1, 100, MPI_COMM_WORLD, &status);
        if (way == 1)
            MPI_Send(msg, 20, MPI_CHAR, rank - 1, 100, MPI_COMM_WORLD);
        printf("Soy el proceso %d y he recibido %s del proceso %d en iteracion %d\n", rank, msg, rank - 1, i);
    }
    else
    {
        if (way == 1)
        {
            MPI_Recv(&msg, 20, MPI_CHAR, rank - 1, 100, MPI_COMM_WORLD, &status);
            MPI_Send(msg, 20, MPI_CHAR, rank + 1, 100, MPI_COMM_WORLD);
            printf("Soy el proceso %d y he recibido %s del proceso %d en iteracion %d\n", rank, msg, rank - 1, i);
            way = 0;
        }
        else if (way == 0)
        {
            MPI_Recv(&msg, 20, MPI_CHAR, rank + 1, 100, MPI_COMM_WORLD, &status);
            MPI_Send(msg, 20, MPI_CHAR, rank - 1, 100, MPI_COMM_WORLD);
            printf("Soy el proceso %d y he recibido %s del proceso %d en iteracion %d\n", rank, msg, rank + 1, i);
            way = 1;
        }
    }
}

MPI_Finalize();
```

```
Soy el proceso 0 y he recibido  del proceso 1 en iteracion 1
Soy el proceso 0 y he recibido  del proceso 1 en iteracion 2
Soy el proceso 0 y he recibido  del proceso 1 en iteracion 3
Soy el proceso 0 y he recibido  del proceso 1 en iteracion 4
Soy el proceso 0 y he recibido  del proceso 1 en iteracion 5
Soy el proceso 1 y he recibido  del proceso 0 en iteracion 1
Soy el proceso 1 y he recibido  del proceso 2 en iteracion 2
Soy el proceso 1 y he recibido  del proceso 0 en iteracion 3
Soy el proceso 1 y he recibido  del proceso 2 en iteracion 4
Soy el proceso 1 y he recibido  del proceso 0 en iteracion 5
Soy el proceso 2 y he recibido  del proceso 1 en iteracion 1
Soy el proceso 2 y he recibido  del proceso 1 en iteracion 2
Soy el proceso 2 y he recibido  del proceso 1 en iteracion 3
Soy el proceso 2 y he recibido  del proceso 1 en iteracion 4
Soy el proceso 2 y he recibido  del proceso 1 en iteracion 5
```

En este apartado se ha logrado satisfactoriamente ejecutar un sistema de rebotes en los que cada proceso envía al siguiente un mensaje hasta llegar al final, donde se alterna el sentido y ahora cada proceso lo enviará al anterior, hasta que se llegue al comienzo de esta cadena, cada vez que se regresa al proceso inicial se considera un rebote. En este ejemplo hemos implementado 5 rebotes. Para comprobar el sentido de envío y recepción de mensajes se ha empleado una variable de dos estados, cada uno de ellos representando un sentido de envío, a la derecha o a la izquierda.
Las otras dos excepciones vienen dadas por el primero y último proceso, los cuales reciben y envían solo del/al segundo proceso o del/al penúltimo respectivamente.