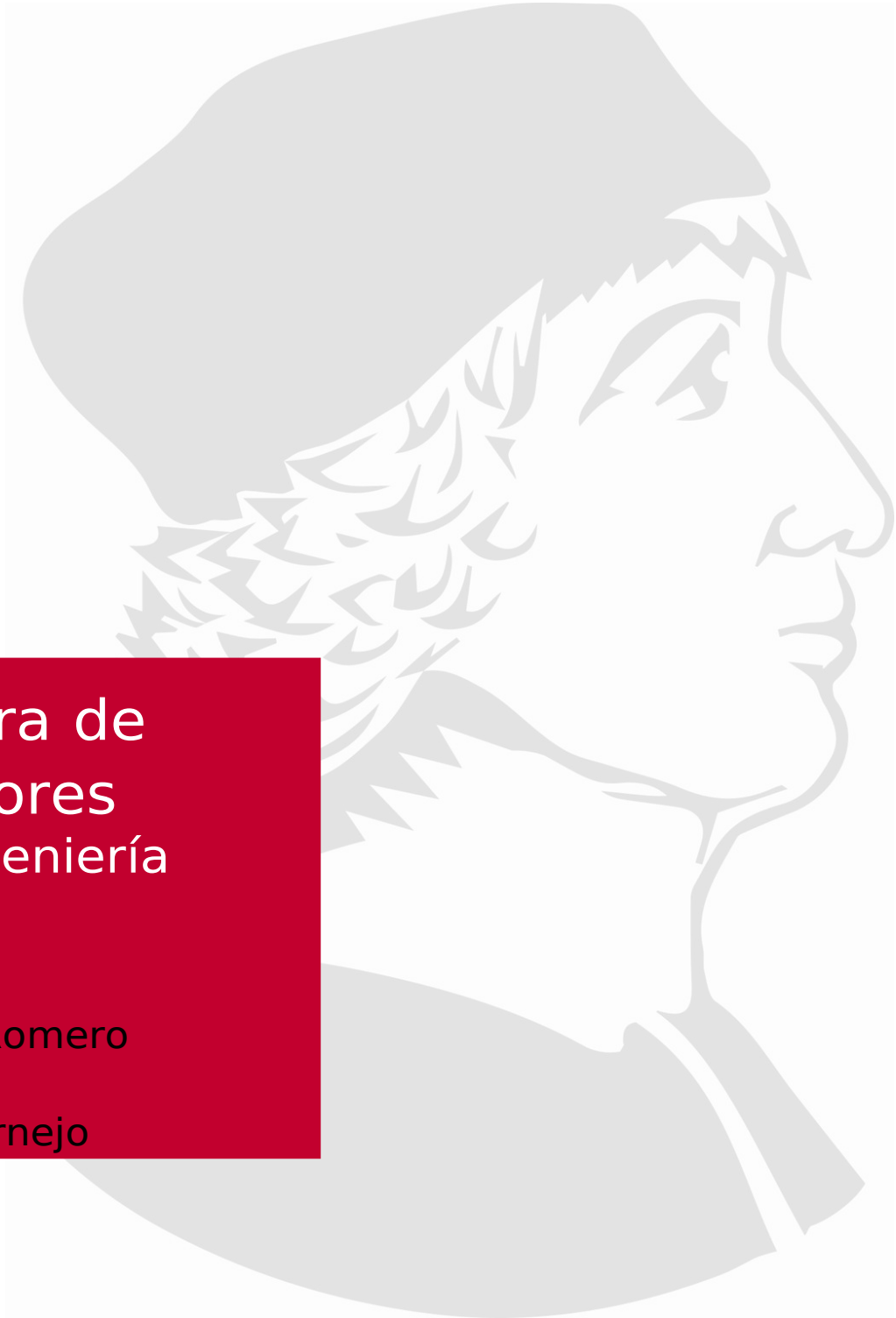


Arquitectura de Computadores

Grado en Ingeniería Informática

Práctica 2

Jose Francisco Romero Rodríguez

Sara Marcos Cornejo

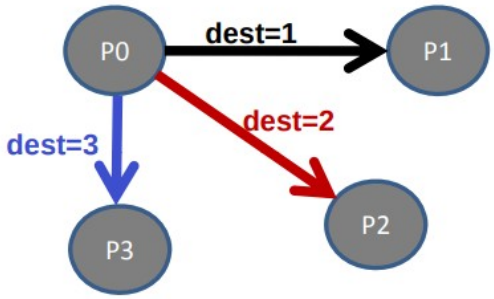
UNIVERSIDAD NEBRIJA

Indice:

Introducción:

MPI define dos tipos de comunicación:

- Punto a punto: involucra a 2 procesos (emisor y receptor).



  Esta comunicación puede ser:
  - Bloqueante: Mantiene el proceso bloqueado hasta que la operación finalice(MPI_Send/MPI_Recv).
  - No bloqueante: Inicia la operación y recupera el control inmediatamente: el proceso debe encargarse más adelante de averiguar si la operación ha finalizado o no.(MPI_Isend/MPI_Irecv)
- Colectiva: Involucra a todos los procesos de un comunicador.

Envío y recepción de mensajes bloqueante:

```
int MPI_Send(void* buf, int count, MPI_Datatype datatype,
             int dest, int tag, MPI_Comm comm);
```

IN	buf	initial address of send buffer
IN	count	number of entries to send
IN	datatype	datatype of each entry
IN	dest	rank of destination
IN	tag	message tag
IN	comm	communicator

```
int MPI_Recv(void* buf, int count, MPI_Datatype datatype,
     int source, int tag, MPI_Comm comm, MPI_Status *status);
```

OUT	buf	initial address of receive buffer
IN	count	max # of entries to receive
IN	datatype	datatype of each entry
IN	source	rank of source
IN	tag	message tag
IN	comm	communicator
OUT	status	return status (inf. acerca del mensaje recibido)

Objetivos:

En esta práctica se aprenderá el concepto de comunicación punto a punto entre procesos de

MPI.

Los objetivos fijados son los siguientes:

- Compilación de programas MPI.
- Ejecución de programas en varios procesos de forma paralela.
- Estructura de un programa MPI.
- Iniciar y finalizar el entorno MPI con MPI_Init y MPI_Finalize.
- Identificador (rango) del proceso con MPI_Comm_rank.
- Consultar el número de procesos lanzados con MPI_Comm_size.
- Primitivas send y recieve.

Ejercicio 1 (1 punto) 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.

Una conexión punto a punto se refiere a una conexión de comunicaciones entre dos puntos finales o nodos de comunicación.

Características:

- Se da entre pares de procesos
- La selección de los mensajes es mediante ranking dentro de un grupo y tag (contexto).
- El ranking y tag estan asociados a un comunicador

Rutinas Bloqueantes. Sólo retornan una vez que la comunicación se completa o cuando el buffer del usuario puede ser usado o reusado. (MPI_Send/Recv)

Rutinas No Bloqueantes. El proceso no espera a la finalización de la operación de comunicación y puede continuar solapando cómputo y comunicación. Debe comprobar más adelante si ha terminado la operación. (MPI _Isend /Irecv)

Primitivas:

SEND	Bloqueante	No Bloqueante
Standard	Mpi_send	Mpi_isend
Ready	Mpi_rsend	Mpi_irsend
Sincrónico	Mpi_ssend	Mpi_issend
Con Buffer	Mpi_bsend	Mpi_ibsend

RECEIVE	Bloqueante	No Bloqueante
Estandar	Mpi_recv	Mpi_irecv

- **Ventajas** comunicación punto a punto:
  - Barata (no se necesita dispositivos de red ni servidores dedicados).
  - Fácil de configurar y mantener (menos complejidad)
  - Permite compartir datos y recursos
- **Desventajas** comunicación punto a punto:
  - No son escalables
  - La administración de la red debe hacerse en cada máquina (no centralizada)
  - No son muy seguras
  - Todos los dispositivos pueden actuar como cliente y como servidor, pudiendo ralentizar su funcionamiento.
  - Reducen su rendimiento

Ejercicio 2 (4 puntos) 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. Ejemplo de la salida por consola con 4 procesos:

Cada proceso imprime: “Soy el proceso X y he inicializado la variable con el dato X”

El proceso 0 imprime tantas veces como procesos haya: “Soy el proceso 0 y he recibido el valor X del proceso X”

Por último, el proceso 0 imprime una vez: “Soy el proceso 0 y ya he recibido los datos de los X procesos: datoProceso0 datoProceso1 datoProceso2 datoProceso3 ”

```c
#include <stdio.h>
#include "mpi.h"
#include <stdlib.h>
#include <time.h>

int main(int argc, char **argv)
{
    int rank,size,num;


    MPI_Status estado;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    int datos[size];
    srand(time(NULL)+rank);
    num = rand() % 100;

    printf("Soy el proceso %d y he inicializado la variable con el dato %d\n", rank, num);
    if(rank == 0)
    {
        datos[0]=num;
        for(int i = 1; i < size; i++)
        {
            MPI_Recv(&num, i, MPI_INT, i, 0, MPI_COMM_WORLD, &estado);
            datos[i] = num;
            printf("Soy el proceso %d y he recibido el valor %d del proceso %d\n", rank, num, i);
        }

        wait(datos);
        printf("Soy el proceso %d y ya he recibido los datos de los %d procesos: ", rank, size);
        for(i = 0; i < size; i++)
        {
            printf("dato proceso %d-> %d    ", i, datos[i]);
        }
        printf("\n");
    }
    else
    {
        MPI_Send(&num, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
    }
    MPI_Finalize();

    return 0;
}
```

Para realizar este ejercicio se inicializa cada variable con un valor aleatorio a través de srand:

- srand (time(NULL)); → Instrucción que inicializa el generador de números aleatorios

- num = rand() %100; → Asigna a la variable num un número aleatorio entre 0 y 99

Cuando entra en el primer for se realiza esta asignación de números aleatorios a cada proceso, almacenando estos valores en un buffer.

Todos ellos envían este valor al proceso 0.

Por último, una vez que el proceso 0 ha recibido los datos de todos los procesos creados, se realiza un for para mostrar por pantalla todos los valores de los datos almacenados en el buffer.

La salida por consola sería la siguiente:

```
sara@sara-vivobook:~/Downloads$ mpicc pract2ej2.c -o ejecutable_ej2
pract2ej2.c: In function ‘main’:
pract2ej2.c:31:9: warning: implicit declaration of function ‘wait’ [-Wimplicit-function-declaration]
   31 |         wait(datos);
      |         ^~~~
sara@sara-vivobook:~/Downloads$ mpirun -np 4 ./ejecutable_ej2
hwloc/linux: Ignoring PCI device with non-16bit domain.
Pass --enable-32bits-pci-domain to configure to support such devices
(warning: it would break the library ABI, don't enable unless really needed).
hwloc/linux: Ignoring PCI device with non-16bit domain.
Pass --enable-32bits-pci-domain to configure to support such devices
(warning: it would break the library ABI, don't enable unless really needed).
hwloc/linux: Ignoring PCI device with non-16bit domain.
Pass --enable-32bits-pci-domain to configure to support such devices
(warning: it would break the library ABI, don't enable unless really needed).
hwloc/linux: Ignoring PCI device with non-16bit domain.
Pass --enable-32bits-pci-domain to configure to support such devices
(warning: it would break the library ABI, don't enable unless really needed).
Soy el proceso 0 y he inicializado la variable con el dato 72
Soy el proceso 1 y he inicializado la variable con el dato 90
Soy el proceso 2 y he inicializado la variable con el dato 26
Soy el proceso 3 y he inicializado la variable con el dato 47
Soy el proceso 0 y he recibido el valor 90 del proceso 1
Soy el proceso 0 y he recibido el valor 26 del proceso 2
Soy el proceso 0 y he recibido el valor 47 del proceso 3
Soy el proceso 0 y ya he recibido los datos de los 4 procesos: dato proceso 0-> 72    dato proceso 1-> 90
sara@sara-vivobook:~/Downloads$
```

Ejercicio 3 (4 puntos) 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.

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

int main(int argc, char *argv[])
{

    bool estado = false;
    int rank, vueltas, size, siguiente;

    MPI_Init(&argc, &argv);
    MPI_Status status;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    if(rank == 0)
    {
        printf("Introduzca el numero de veces que desea que reboten los mensajes entre sí: ");
        scanf("%d", &vueltas);
        siguiente = 1;
        printf("Proceso %d envia %d al proceso %d\n", rank, vueltas, siguiente);
        MPI_Send(&vueltas, 1, MPI_INT, siguiente, 0, MPI_COMM_WORLD);
    }
```

```c
    do {
        if(rank == 1) {
            if(estado==true){
                MPI_Recv(&vueltas, 1, MPI_INT, 2, 0, MPI_COMM_WORLD, &status);
            }
            else{
                MPI_Recv(&vueltas, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
            }
        }
        else{
            MPI_Recv(&vueltas, 1, MPI_INT, 1, 0, MPI_COMM_WORLD, &status);
        }
        printf("Proceso %d ha recibido %d\n", rank, vueltas);

        if(rank == 0){
            --vueltas; //disminuir num vueltas
         }
        if(vueltas > 0){
            if(rank==1){
                if(estado==true){
                    siguiente = 0;
                }else{
                    siguiente = 2;
                }
                estado = !estado; // cambio de estado
            }
            else{
                siguiente = 1;
            }
            MPI_Send(&vueltas, 1, MPI_INT, siguiente, 0, MPI_COMM_WORLD);
            printf("Proceso %d envia %d al proceso %d\n", rank, vueltas, siguiente);
        }

    } while (vueltas > 0);
    printf("Proceso %d finaliza \n", rank);

    MPI_Finalize();
    return 0;
}
```

Para este ejercicio se envía como dato el número de vueltas que el usuario introduce por pantalla. Este dato se va a ir enviando entre los procesos rebotando entre sí.

Mientras el número de vueltas sea mayor que 0 seguirá enviando los procesos mediante una lógica de diagrama de estados en función de qué proceso se trate y el estado, generando como salida: siguiente (el proceso al que se envía).

Cada vez que se termine un ciclo entero se desconatrá una unidad del total de vueltas restantes.

La salida por consola sería la siguiente:

```
Pass --enable-32bits-pci-domain to configure to support such devices
(warning: it would break the library ABI, don't enable unless really needed).
hwloc/linux: Ignoring PCI device with non-16bit domain.
Pass --enable-32bits-pci-domain to configure to support such devices
(warning: it would break the library ABI, don't enable unless really needed).
hwloc/linux: Ignoring PCI device with non-16bit domain.
Pass --enable-32bits-pci-domain to configure to support such devices
(warning: it would break the library ABI, don't enable unless really needed).
hwloc/linux: Ignoring PCI device with non-16bit domain.
Pass --enable-32bits-pci-domain to configure to support such devices
(warning: it would break the library ABI, don't enable unless really needed).
hwloc/linux: Ignoring PCI device with non-16bit domain.
Pass --enable-32bits-pci-domain to configure to support such devices
(warning: it would break the library ABI, don't enable unless really needed).
Introduzca el numero de veces que desea que reboten los mensajes entre sí: 2
Proceso 0 envia 2 al proceso 1
Proceso 1 ha recibido 2
Proceso 1 envia 2 al proceso 2
Proceso 2 ha recibido 2
Proceso 2 envia 2 al proceso 1
Proceso 1 ha recibido 2
Proceso 1 envia 2 al proceso 0
Proceso 0 ha recibido 2
Proceso 0 envia 1 al proceso 1
Proceso 1 ha recibido 1
Proceso 1 envia 1 al proceso 2
Proceso 2 ha recibido 1
Proceso 2 envia 1 al proceso 1
Proceso 1 ha recibido 1
Proceso 1 envia 1 al proceso 0
Proceso 0 ha recibido 1
```

Bibliografía:

https://es.wikipedia.org/wiki/Red_punto_a_punto#Ventajas_de_las_redes_punto_a_punto