Comunicación por mensajes

José Luis Quiroz Fabián

1 Objetivos

Los objetivos buscados en esta práctica son los siguientes:

- Estudiar la comunicación de procesos.
- Aplicar el concepto de la comunicación de procesos usando *sockets* y MPI.

2 Sockets

Los sockets nos permiten comunicar procesos en una misma computadora o bien en computadoras en red. Usando *sockets* podemos establecer un canal de datos fiable (garantizando que todo dato que un emisor envía le llegará el receptor) y no fiable (no hay garantía que los datos lleguen o bien lleguen repetidos).

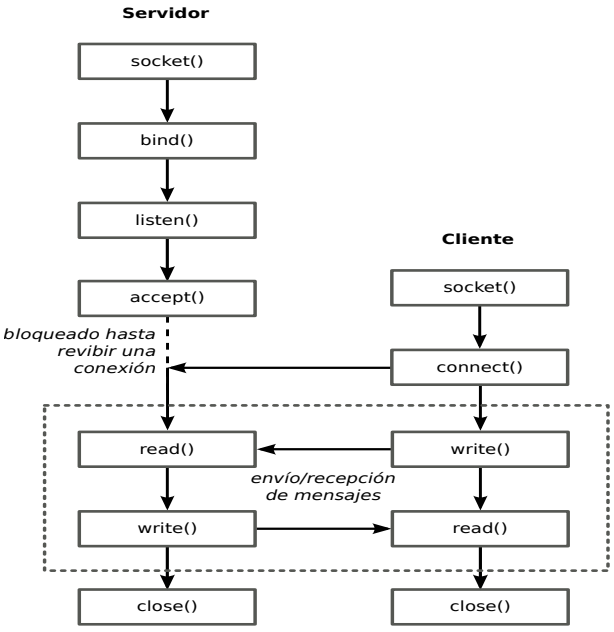

Figure 1: Modelo de comunicación

En la Figura 1 se observa el modelo de comunicación fiable mediante Sockets en C. Como se observa en la figura, el servidor debe especificar el puerto por

donde escuchará y queda en espera de peticiones. Cuando recibe una petición crea un Socket para comunicase con el cliente que le hizo la petición.

2.1 Ejemplo

En este ejemplo se tiene un cliente y un servidor los cuales se comunican por medio del puerto 80 (en el servidor). En este ejemplo el servidor únicamente espera un cliente.

2.1.1 Servidor

```
#include<stdio.h>
#include<string.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<unistd.h>

int main(int argc , char *argv[])
{
int socket_desc , socket_cliente , c , read_size;
struct sockaddr_in server , client;
char mensaje[1000];


        //SE CREA EL SOCKET
socket_desc = socket(AF_INET , SOCK_STREAM , 0);
if (socket_desc == -1)
{
printf("Could not create socket");
}


        //INFORMACIÓN DEL SOCKET
server.sin_family = AF_INET;
server.sin_addr.s_addr = INADDR_ANY;
server.sin_port = htons( 3030 );

//SE ENLAZA EL SOCKET
if( bind(socket_desc,(struct sockaddr *)&server , sizeof(server)) < 0)
{
//print the error message
perror("bind failed. Error");
return 1;
}

//VAMOS ESPERAR CONEXIONES DE CLIENTES
listen(socket_desc , 3);



//ESPERANDO POR CONEXIONES
c = sizeof(struct sockaddr_in);

socket_cliente = accept(socket_desc, (struct sockaddr *)&client, (socklen_t*)&c);
if (socket_cliente < 0)
{
perror("FALLO AL ACEPTAR LA CONEXION");
return 1;
}
```

```
//RECIBIENDO MENSAJE DEL CLIENTE
read_size = recv(socket_cliente , mensaje , 1000 , 0);;
        mensaje[read_size]='\0';
        printf("Recibi tu mensaje: %s \n",mensaje);


return 0;
}
```

2.1.2 Cliente

```
#include<stdio.h>
#include<string.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include <time.h>

int main(int argc , char *argv[]){
    int sock;
    struct sockaddr_in server;
    char message[1000] , server_reply[2000];
    time_t rawtime;
    struct tm * timeinfo;


    sock = socket(AF_INET , SOCK_STREAM , 0);
    if (sock == -1)
    {
        printf("NO SE PUEDE CREAR EL SOCKET");
        exit(0);
    }

    server.sin_addr.s_addr = inet_addr("52.87.193.127");
    server.sin_family = AF_INET;
    server.sin_port = htons( 3030 );

    //CONECTANDO AL SERVIDOR
    if (connect(sock , (struct sockaddr *)&server , sizeof(server)) < 0)
    {
        perror("FALLO EN LA CONEXION");
        return 1;
    }

    time ( &rawtime );
    timeinfo = localtime ( &rawtime );
    strcpy(message,asctime (timeinfo));

    if( send(sock , message , strlen(message) , 0) < 0){
            puts("FALLO AL ENVIAR");
            return 1;
    }

    close(sock);
    return 0;
}
```

Compilar y ejecutar el ejemplo anterior cambiando la dirección IP.

3 MPI

MPI (Interfaz de Paso de Mensajes) es un estándar que define la sintaxis y la semántica de las funciones contenidas en una biblioteca de paso de mensajes diseñada para ser usada en programas que exploten la existencia de múltiples procesadores. El paso de mensajes es una técnica empleada en programación concurrente para aportar sincronización entre procesos y permitir la exclusión mutua, de manera similar a como se hace con los semáforos, monitores, etc. Los elementos principales que intervienen en el paso de mensajes son el proceso que envía, el que recibe y el mensaje.

3.1 Ejemplo simple MPI

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



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

    int mi_id, numprocs;

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&mi_id);

    printf("Proceso %d  de un total de  %d\n",numprocs);

    MPI_Finalize();
}
```

3.2 Ejemplo mensajes MPI

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

#define ETIQUETA 100


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

    int mi_id, numprocs;
    int buffer;
    MPI_Status estado;

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&mi_id);
    if(mi_id == 0){
      buffer=20;
      MPI_Send(&buffer,1,MPI_INT,1,ETIQUETA,MPI_COMM_WORLD);
      printf("Proceso %d  Dato enviado  %d\n",mi_id,buffer);
    }else{
        MPI_Recv(&buffer,1,MPI_INT,0,ETIQUETA,MPI_COMM_WORLD,&estado);
        printf("Proceso %d  Dato recibido %d\n",mi_id,buffer);
    }
    MPI_Finalize();
```

```
}
```

3.3 Ejercicio

La suma global de un conjunto de N números se pueden obtener mediante el uso de un pipeline circular. Cada proceso de inicio genera un valor y después lo envía a su siguiente en el pipeline. Al final de un número de iteraciones, donde cada proceso reenvía cada valor que recibe, los procesos tiene la suma global (ver el ejemplo de abajo para $N = 4$). Realice un programa MPI para N procesos, donde $N > 3$. Los valores de inicio de cada proceso se puede suponer que se generan de forma aleatoria.

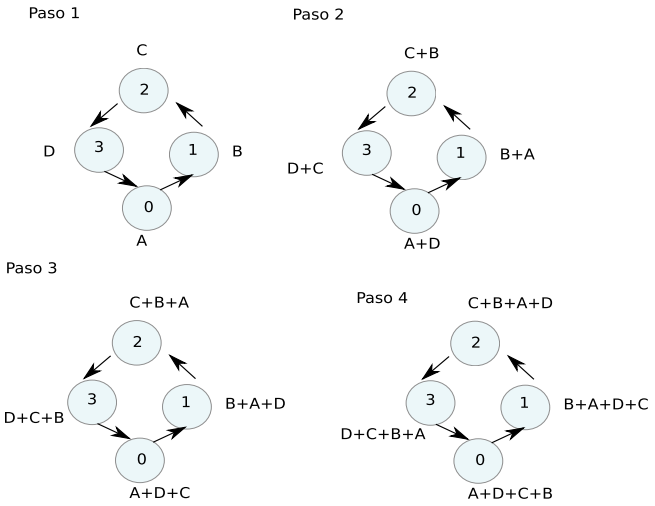

Figure 2: Suma global