Mayank Lakhotia
2022PMN4226

Practical – 6

Aim: - Implement a Socket () System call for Two-Way Inter-Process Communication between:
1. Single Client and Single Server.
2. Multiple Clients and Single Server.
Consider Two independent processes for communication.
This communication must continue till a specific key is pressed or a STOP message is sent by any one of the processes.

Theory: - Interprocess Communication with Sockets:

Interprocess communication is the mechanism provided by the operating system that allows processes to communicate with each other. This communication could involve a process letting another process know that some event has occurred or transferring of data from one process to another. One of the ways to manage interprocess communication is by using sockets. They provide point-to-point, two-way communication between two processes. Sockets are an endpoint of communication and a name can be bound to them. A socket can be associated with one or more processes.

Types of Sockets: -

The different types of sockets are given as follows −

- **Sequential Packet Socket:** This type of socket provides a reliable connection for datagrams whose maximum length is fixed This connection is two-way as well as sequenced.
- **Datagram Socket:** A two-way flow of messages is supported by the datagram socket. The receiver in a datagram socket may receive messages in a different order than that in which they were sent. The operation of datagram sockets is similar to that of passing letters from the source to the destination through a mail.
- **Stream Socket:** Stream sockets operate like a telephone conversation and provide a two-way and reliable flow of data with no record boundaries. This data flow is also sequenced and unduplicated.
- **Raw Socket:** The underlying communication protocols can be accessed using the raw sockets.

Socket Creation

Sockets can be created in a specific domain and the specific type using the following declaration −

```
int socket (int domain, int type, int protocol)
```

If the protocol is not specified in the above system call, the system uses a default protocol that supports the socket type. The socket handle is returned. It is a descriptor.

The bind function call is used to bind an internet address or path to a socket. This is shown as follows −

```
int bind (int s, const struct sockaddr *name, int namelen)
```

Connecting Stream Sockets

Connecting the stream sockets is not a symmetric process. One of the processes acts as a server and the other acts as a client. The server specifies the number of connection requests that can be queued using the following declaration −

```
int listen (int s, int backlog)
```

The client initiates a connection to the server's socket by using the following declaration −

```
int connect (int s, struct sockaddr *name, int namelen)
```

A new socket descriptor which is valid for that particular connection is returned by the following declaration −

```
int accept (int s, struct sockaddr *addr, int *addrlen)
```

Stream Data Transfer

The send () and recv () functions are used to send and receive data using sockets. These are similar to the read () and write () functions but contain some extra flags. The declaration for send () and recv () are as follows −

```
int send (int s, const char *msg, int len, int flags)

int recv (int s, char *buf, int len, int flags)
```

Stream Closing

The socket is discarded or closed by calling close ()

Implementation code:

1. Single Client and Single Server:

Source Code: - C Language

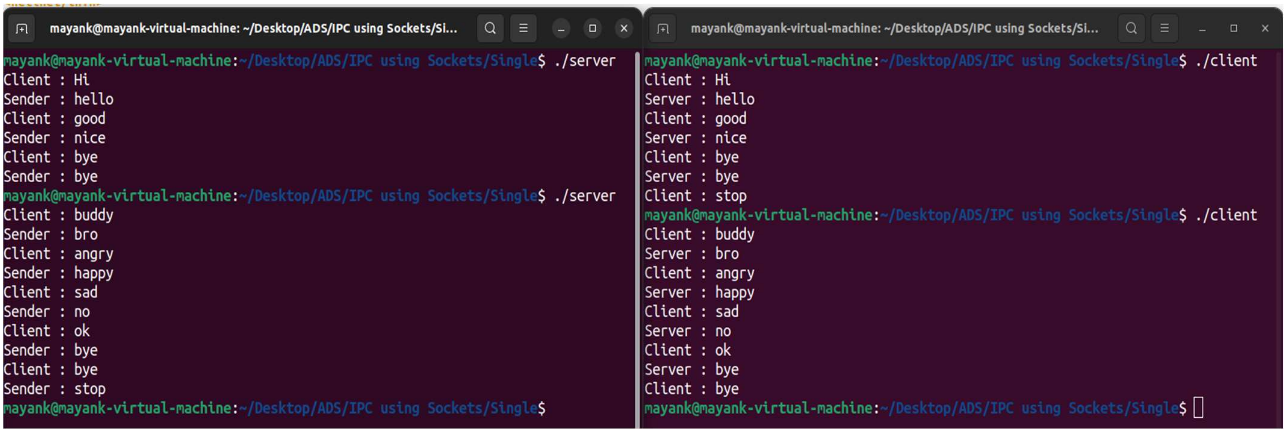
server.c

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<arpa/inet.h>
#include<netinet/in.h>
#include<netinet/ip.h>
#include<unistd.h>
int main()
{
    int recevfd=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    if(recevfd==-1)
    {
        perror("S error"); //Socket Error
        exit(0);
    }
    struct sockaddr_in server,client;
    server.sin_family = AF_INET;
    server.sin_port=htons(5000);
    server.sin_addr.s_addr=INADDR_ANY;
    int b=bind(recevfd, (struct sockaddr *)&server, sizeof(server));
    if(b==-1)
    {
        perror("b error"); //Bind Error
        exit(0);
    }
    char rcv[20], snd[20];
    while(1)
    {
    int size=sizeof(server);
    int len=recvfrom(recevfd, rcv, sizeof(rcv),0,(struct sockaddr *)
&client,&size);
    rcv[len]='\0';
    if(strcmp(rcv,"stop")==0)
    break;
    printf("Client : %s\n", rcv);
    printf("Sender : ");
    scanf("%s",snd);
    sendto(recevfd,snd,strlen(snd),0,(struct sockaddr *)&client,sizeof(client));
    if(strcmp(snd,"stop")==0) // To stop the conversation stop key use
    break;
    }
    close(recevfd);
}
```

client.c

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<arpa/inet.h>
#include<netinet/in.h>
#include<netinet/ip.h>
#include<unistd.h>
int main()
{
    int sendfd=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    if(sendfd==-1)
    {
        perror("S error"); //Socket Error
        exit(0);
    }
    struct sockaddr_in server,client;
    server.sin_family = AF_INET;
    server.sin_port=htons(5000);
    server.sin_addr.s_addr=INADDR_ANY;
    char snd[20], rcv[20];
    while(1)
    {
    printf("Client : ");
    scanf("%s", snd);

    int size=sizeof(client);

    sendto(sendfd, snd, strlen(snd), 0, (struct sockaddr *)&server ,
sizeof(server));

    if(strcmp(snd,"stop")==0) // To stop the conversation stop key use

    break;

    int len=recvfrom(sendfd, rcv,sizeof(rcv), 0,(struct sockaddr
*)&client,&size);

    rcv[len]='\0';

    if(strcmp(rcv,"stop")==0) // To stop the conversation stop key use

    break;

    printf("Server : %s\n", rcv);

    }

    close(sendfd);

}
```

Output: -



mayank@mayank-virtual-machine: ~/Desktop/ADS/IPC using Sockets/Si...
mayank@mayank-virtual-machine: ~/Desktop/ADS/IPC using Sockets/Si...

```
mayank@mayank-virtual-machine:~/Desktop/ADS/IPC using Sockets/Single$ ./server
Client : Hi
Sender : hello
Client : good
Sender : nice
Client : bye
Sender : bye
mayank@mayank-virtual-machine:~/Desktop/ADS/IPC using Sockets/Single$ ./server
Client : buddy
Sender : bro
Client : angry
Sender : happy
Client : sad
Sender : no
Client : ok
Sender : bye
Client : bye
Sender : stop
mayank@mayank-virtual-machine:~/Desktop/ADS/IPC using Sockets/Single$
```

```
mayank@mayank-virtual-machine:~/Desktop/ADS/IPC using Sockets/Single$ ./client
Client : Hi
Server : hello
Client : good
Server : nice
Client : bye
Server : bye
Client : stop
mayank@mayank-virtual-machine:~/Desktop/ADS/IPC using Sockets/Single$ ./client
Client : buddy
Server : bro
Client : angry
Server : happy
Client : sad
Server : no
Client : ok
Server : bye
Client : bye
mayank@mayank-virtual-machine:~/Desktop/ADS/IPC using Sockets/Single$
```

2. Multiple Clients and Single Server:

Source Code: - C Language

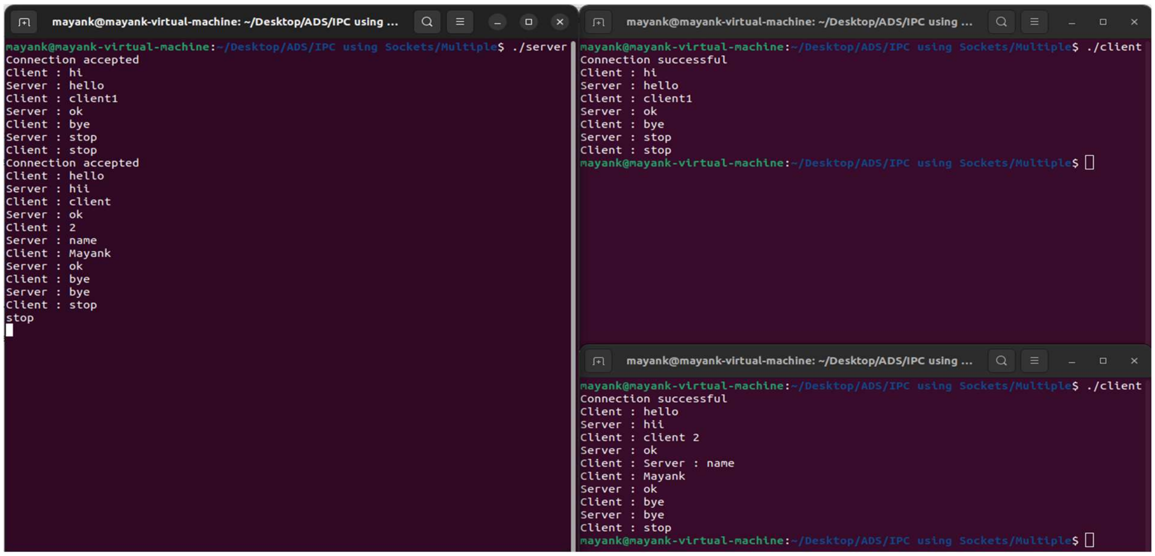
server.c

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <stdlib.h>
int main()
{
    int serverfd=socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if(serverfd==-1)
    {
        perror("S error\n");// Socket error
        exit(0);
    }
    struct sockaddr_in server,client;
    server.sin_family = AF_INET;
    server.sin_port = htons(5000);
    server.sin_addr.s_addr = inet_addr("127.0.0.1");
    int b=bind(serverfd, (struct sockaddr *)&server, sizeof(server));
    if(b==-1)
    {
        perror("B error\n");// Binding error
        exit(0);
    }
    listen(serverfd,5);
    int size=sizeof(struct sockaddr);
    char snd[20],rcv[20];
    while(1)
    {
    int clientfd=accept(serverfd, (struct sockaddr *)&client,&size);
    if(clientfd==-1)
    {
        perror("a error\n");// Accept error
        exit(0);
    }
    printf("Connection accepted\n");
    for(;;)
    {
int r=recv(clientfd, rcv, sizeof(rcv),0);
rcv[r]='\0';
printf("Client : %s\n",rcv);
if(strcmp(rcv,"stop")==0)// To stop the conversation stop key use
break;
printf("Server : ");
scanf("%s",snd);
send(clientfd,snd,strlen(snd),0);
    }
    }
}
```

client.c

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <stdlib.h>
int main()
{
    int clientfd=socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if(clientfd==-1)
    {
        perror("S error\n");//Socket Error
        exit(0);
    }
    struct sockaddr_in server,client;
    server.sin_family = AF_INET;
    server.sin_port = htons(5000);
    server.sin_addr.s_addr = inet_addr("127.0.0.1");
    int c=connect(clientfd, (struct sockaddr *)&server, sizeof(server));
    if(c==-1)
    {
        perror("C error\n");//Connection error
        exit(0);
    }
    printf("Connection successful\n");
    char snd[20], rcv[20];
    while(1)
    {
    printf("Client : ");
    scanf("%s",snd);
    send(clientfd,snd,strlen(snd),0);
    if(strcmp(snd,"stop")==0)// To stop the conversation stop key use
    break;
    int r=recv(clientfd, rcv, sizeof(rcv),0);
    rcv[r]='\0';
    printf("Server : %s\n",rcv);
    }
}/* when one client is with server other have to wait when one client leave
other can join and complete its work with server. */
```

Output: -


mayank@mayank-virtual-machine: ~/Desktop/ADS/IPC using ...
mayank@mayank-virtual-machine: ~/Desktop/ADS/IPC using ...
mayank@mayank-virtual-machine: ~/Desktop/ADS/IPC using ...

```
mayank@mayank-virtual-machine:~/Desktop/ADS/IPC using Sockets/Multiple$ ./server
Connection accepted
Client : hi
Server : hello
Client : client1
Server : ok
Client : bye
Server : stop
Client : stop
Connection accepted
Client : hello
Server : hii
Client : client
Server : ok
Client : 2
Server : name
Client : Mayank
Server : ok
Client : bye
Server : bye
Client : stop
stop
```

```
mayank@mayank-virtual-machine:~/Desktop/ADS/IPC using Sockets/Multiple$ ./client
Connection successful
Client : hi
Server : hello
Client : client1
Server : ok
Client : bye
Server : stop
Client : stop
mayank@mayank-virtual-machine:~/Desktop/ADS/IPC using Sockets/Multiple$
```

```
mayank@mayank-virtual-machine:~/Desktop/ADS/IPC using Sockets/Multiple$ ./client
Connection successful
Client : hello
Server : hii
Client : client 2
Server : ok
Client : Server : name
Client : Mayank
Server : ok
Client : bye
Server : bye
Client : stop
mayank@mayank-virtual-machine:~/Desktop/ADS/IPC using Sockets/Multiple$
```