B.M.S College of Engineering
P.O. Box No.: 1908 Bull Temple Road,
Bangalore-560 019

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

Course – UNIX SYSTEM PROGRAMMING
Course Code – 19IS4PWUSP
AY 2019-20

Initial report on mini -project work

CLIENT-SERVER ARCHITECTURE

Submitted to – Dr. Shubha Rao V

Submitted by –

Prateek Jain	Anurag Dhasmana
1BM18IS149	1BM18IS014

B.M.S College of Engineering
P.O. Box No.: 1908 Bull Temple Road, Bangalore-560 019

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

CERTIFICATE

Certified that the Project has been successfully presented at **B.M.S College Of Engineering** by **PRATEEK JAIN and ANURAG DHASMANA**, bearing USN:**1BM18IS149 and 1BM18IS014**, in partial fulfillment of the requirements for the IV Semester degree in **Bachelor of Engineering in Information Science & Engineering** of **Visvesvaraya Technological University, Belgaum** as a part of the course **UNIX SYSTEM PROGRAMMING (19IS4PWUSP)** during academic year 2019-2020.

FACULTY NAME:- Dr. Shubha Rao V

DESIGNATION:- Assistant Professor

Department of ISE, BMSCE

TABLE OF CONTENTS:-

ABSTRACT

By the design, a typical Internet service is implemented in two parts—a server that provides information and one or more clients that request information. Such client/server architecture has been gaining popularity as an approach for implementing distributed information systems. The client/server architecture typically consists of a collection of computers connected by a communication network. The functions of the information system are performed by processes (computer programs) that run on these computers and communicate through the network.

In recent years, the client/server architecture has become commonplace as the mechanism that brings the centralized corporate databases to desktop PCs on a network. In a client/server environment, one or more servers manage the centralized database and clients gain access to the server.

Client/server architecture requires clients to communicate with the servers. That’s where TCP/IP comes in—TCP/IP provides a standard way for clients and servers to exchange packets of data.

INTRODUCTION

The client/server model is an architecture (i.e a system design) that divides processing between clients and servers that can run on the same computer or, more commonly, on different computers on the same network. It is a major element of modern operating system and network design. A server is a a program, or the computer on which that program runs, that provides a specific kind of service to clients. A major feature of servers is that they can provide their services to large numbers of clients simultaneously.

The client is usually a program that provides the user interface, also referred to as the front end, typically a GUI (graphical user interface), and performs some or all of the processing on requests it makes to the server, which maintains the data and processes the requests.

TCP (Transmission control protocol)

A TCP (transmission control protocol) is a connection-oriented communication. It is an intermediate layer of the application layer and internet protocol layer in OSI model. TCP is designed to send the data packets over the network. It ensures that data is delivered to the correct destination.

- TCP creates a connection between the source and destination node before transmitting the data and keep the connection alive until the communication is active.

- In TCP before sending the data it breaks the large data into smaller packets and cares the integrity of the data at the time of reassembling at the destination node. Major Internet applications such as the World Wide Web, email, remote administration, and file transfer rely on TCP.

- TCP also offers the facility of retransmission, when a TCP client sends data to the server, it requires an acknowledgment in return. If an acknowledgment is not received, after a certain amount of time transmitted data will be the loss and TCP automatically retransmits the data.

- The communication over the network in TCP/IP model takes place in form of a client-server architecture. ie, the client begins the communication and establish a connection with a server.

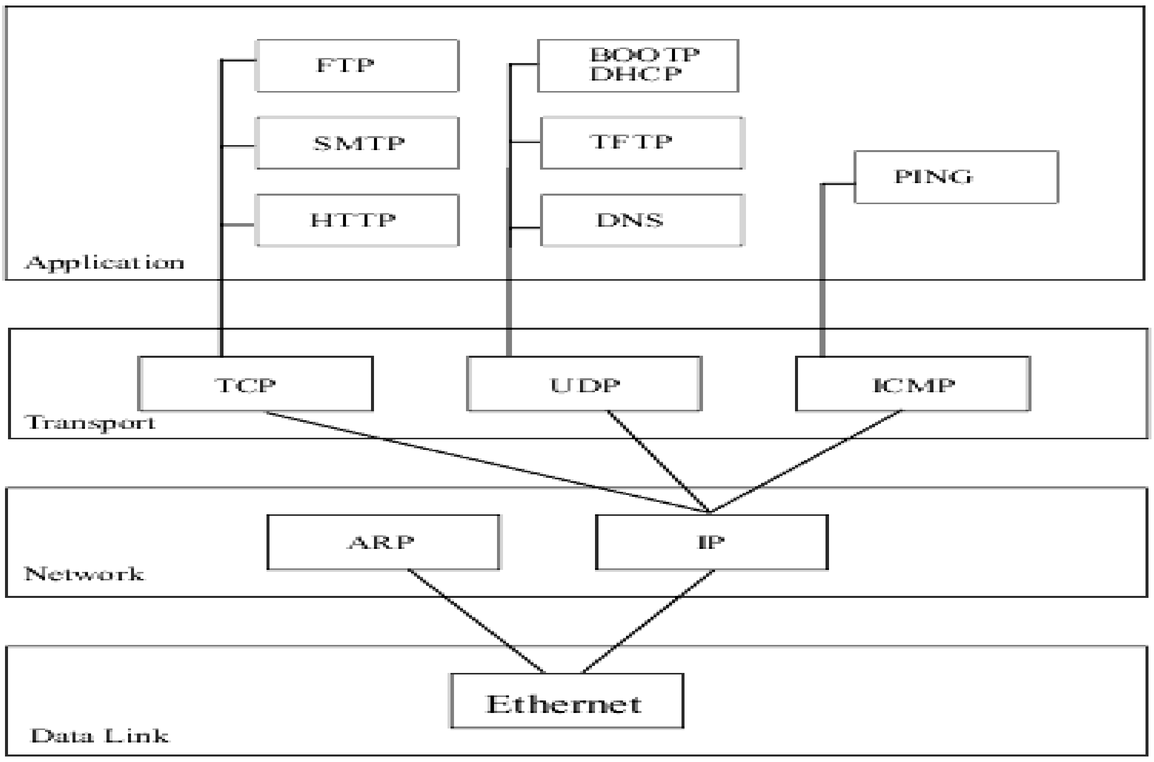


FIG:-TCP/IP MODEL AND ITS FLOW

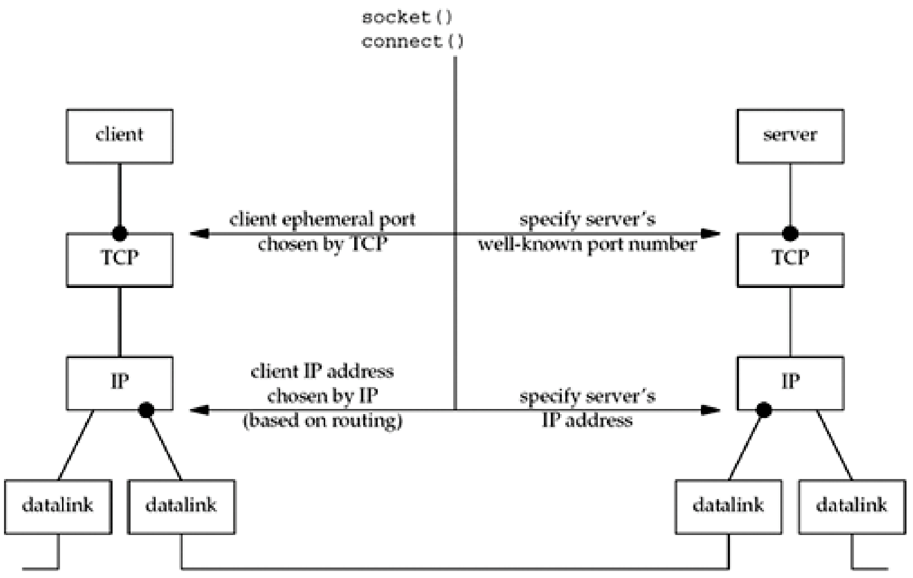
SOCKET PROGRAMMNG: Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server.

PROBLEM STATEMENT:-

- In the Client-Server architecture there exists a different leverage of cases for peer to peer communication.
- So it is quite difficult to use UDP technique for the connection-less communication and also if we perform the client server communication through internet,it might lead to bottleneck and also high cost for the connection setup.
- We need to implement more logic for the peer to peer communication which might be cumbersome.
- The only advantage that we can achieve in UDP is that we doesn’t require any dedicated socket(such as specifying the port number).

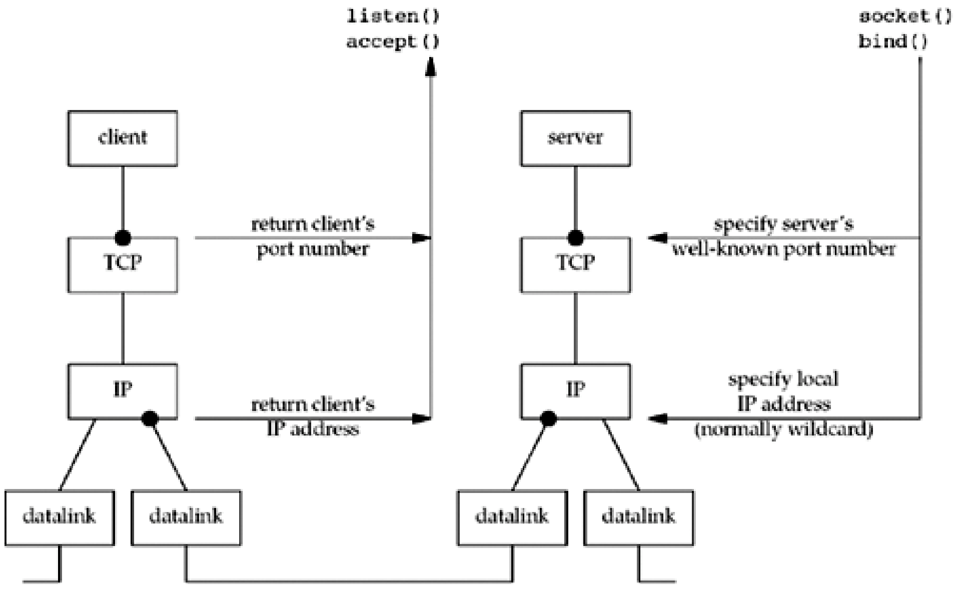
THE FOLLOWING FIGURE SHOWS THE CLIENT AND SERVER OPERATION IN TCP/IP MODEL:-

CLIENT perspective:-



socket()
connect()
client
server
client ephemeral port chosen by TCP
specify server's well-known port number
TCP
TCP
IP
client IP address chosen by IP (based on routing)
specify server's IP address
IP
datalink
datalink
datalink
datalink

SERVER perspective



listen()
accept()
socket()
bind()
client
server
return client's port number
specify server's well-known port number
TCP
TCP
IP
return client's IP address
IP
specify local IP address (normally wildcard)
datalink
datalink
datalink
datalink

THE STEPS INVOLVED IN ESTABLISHING A SOCKET ON THE SERVER SIDE ARE AS FOLLOWS:

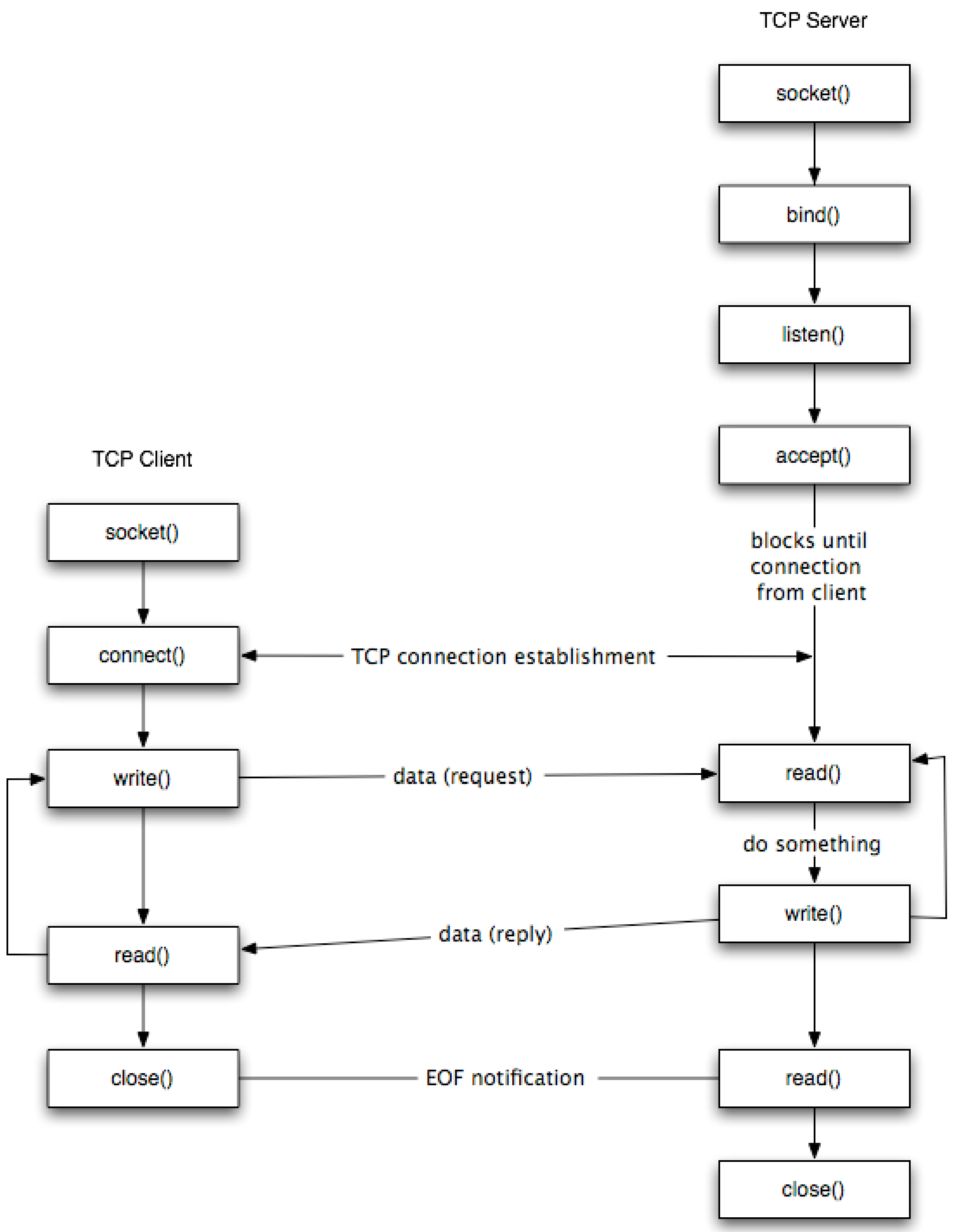
1. Create a socket with the socket() system call.
2. Bind the socket to an address using the bind() system call.
3. Listen for connections with the listen() system call.
4. Accept a connection with the accept() system call.
5. Send and receive data.

The client/server model has some important advantages that have resulted in it becoming the dominant type of network architecture. One is that it is highly efficient in that it allows many users at dispersed locations to share resources, such as a web site, a database, files or a printer. Another is that it is highly (and very easily) scalable, from a single computer to thousands of computers.

THE SOCKET API's USED IN THE PROJECT ARE AS FOLLOWS:-

- Socket
- Bind
- Listen
- Accept
- Connect
- Read
- Write
- Send

THE FOLLOWING FIGURE SHOW THE DETAILED WORKING OF SYSTEM CALLS:-



EXPLANATION ABOUT THE SOCKET API's:-

- **Socket():-**The socket system call creates a new socket by assigning a new descriptor. The new descriptor is returned to the calling process. Any subsequent system calls are identified with the created socket. The socket system call also assigns the protocol to the created socket descriptor.
- **Bind():-**The bind system call associates a local network transport address with a socket. For a client process, it is not mandatory to issue a bind call. The kernel takes care of doing an implicit binding when the client process issues the connect system call. It is often necessary for a server process to issue an explicit bind request before it can accept connections or start communication with clients
- **Connect():-**The connect system call is normally called by the client process to connect to the server process. If the client process has not explicitly issued a bind system call before initiating the connection, implicit binding on the local socket is taken care of by the stack..
- **Listen():-**The listen call indicates to the protocol that the server process is ready to accept any new incoming connections on the socket. There is a limit on the number of connections that can be queued up, after which any further connection requests are ignored.
- **Accept():-**The accept system call is a blocking call that waits for incoming connections. Once a connection request is processed, a new socket descriptor is returned by accept. This new socket is connected to the client and the other socket s remains in LISTEN state to accept further connections.
- **Write():-** A successful write() updates the change and modification times for the file. If fs refers to a socket, write() is equivalent to send() with no flags set. Behavior for sockets: The write() function writes data from a buffer on a socket with descriptor fs. The socket must be a connected socket.
- **Read():-**The read() call reads data on a socket with descriptor fs and stores it in a buffer. The read() all applies only to connected sockets. This call returns up to N bytes of data. If there are fewer bytes available than requested, the call returns the number currently available.
- **Close:-**The close function shuts down the socket associated with socket descriptor s, and frees resources allocated to the socket. If s refers to an open TCP connection, the connection is closed.

IMPLEMENTED CODE:-

CLIENT SIDE SCRIPTING:-

```
#include<stdio.h>

#include<sys/types.h>

#include<sys/socket.h>

#include<netinet/in.h>

#include<netdb.h>

void error(char *msg)

{

   perror(msg);

   exit(0);

}

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

{

int sockfd,portno,n;

struct sockaddr_in serv_addr;

struct hostent *server;

char buffer[256];

if (argc < 3)

{

fprintf(stderr,"usage %s hostname port\n", argv[0]);

exit(0);

}

portno = atoi(argv[2]);

sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

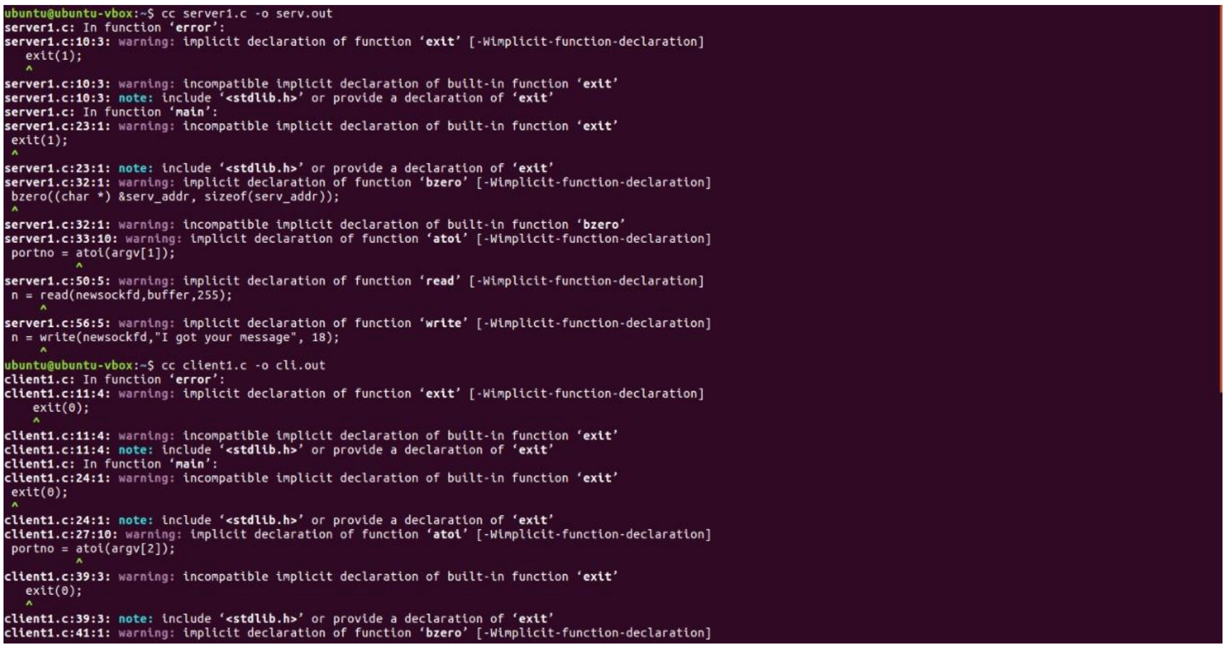
```
if (sockfd < 0)

{

 error("Error opening socket");

}

server = gethostbyname(argv[1]);

if(server == NULL)

{

 fprintf(stderr,"Error,no such host\n");

  exit(0);

}

bzero((char *) &serv_addr,sizeof(serv_addr));

serv_addr.sin_family = AF_INET;

bcopy((char *)server->h_addr,(char *)&serv_addr.sin_addr.s

addr,server->h_length);

serv_addr.sin_port=htons(portno);

if(connect(sockfd,&serv_addr,sizeof(serv_addr)) < 0)

{

    error("Error connecting");

}


printf("Enter The Message");

bzero(buffer,256);

fgets(buffer,255,stdin);

n = write(sockfd,buffer,strlen(buffer));

if(n<0)

{
      error("Error writing to socket");
}
bzero(buffer,256);
n = read(sockfd,buffer,255);
if(n < 0)
{
    error("Error reading");
}
printf("%s\n",buffer);
return 0;
}
```

SERVER SIDE SCRIPTING:-

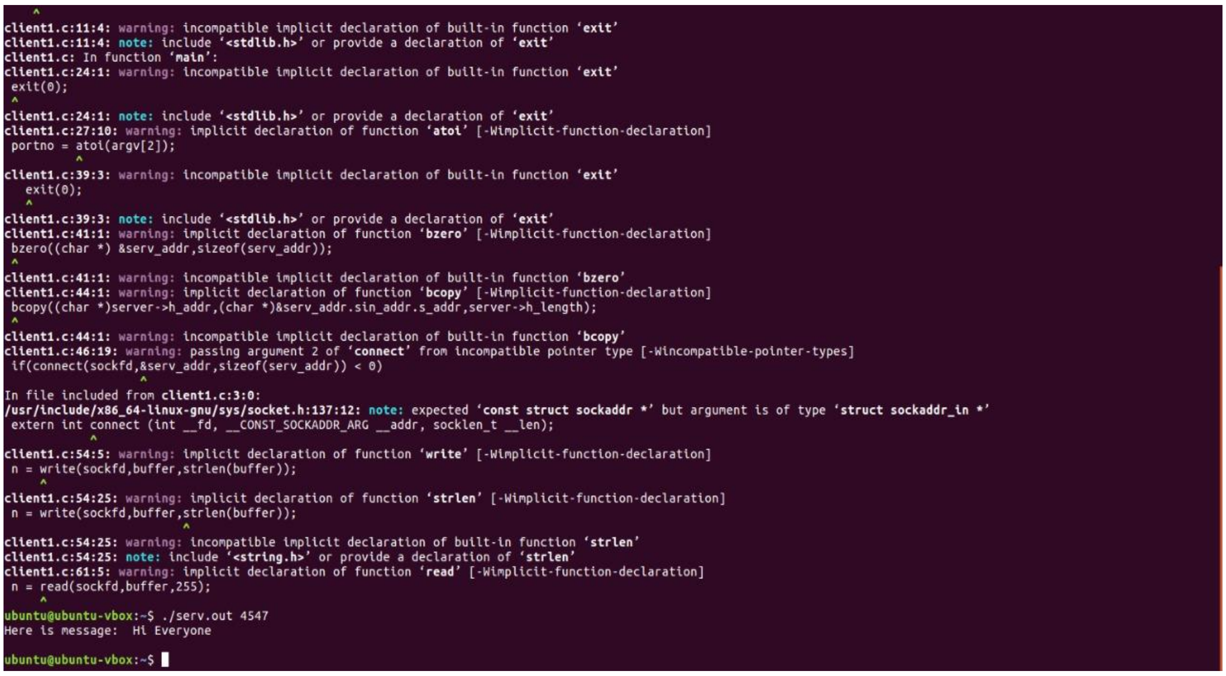
```
#include<stdio.h>

#include<sys/types.h>

#include<sys/socket.h>

#include<netinet/in.h>

void error(char *msg)

{

  perror(msg);

  exit(1);

}

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

{

int sockfd,newsockfd,portno,clilen,n;

struct sockaddr_in serv_addr;

struct sockaddr_in cli_addr;

char buffer[256];

if (argc < 2)

{

fprintf(stderr, "ERROR, no port provided\n");

exit(1);

}

sockfd=socket(AF_INET, SOCK_STREAM, 0);

if (sockfd < 0)

{

  error("Error Opening socket");
}
bzero((char *) &serv_addr, sizeof(serv_addr));
portno = atoi(argv[1]);
serv_addr.sin_family=AF_INET;
serv_addr.sin_addr.s_addr=INADDR_ANY;
serv_addr.sin_port=htons(portno);
if (bind(sockfd,(struct sockaddr *) &serv_addr,sizeof(serv_addr))<0)
{
error("error binding");
}
listen(sockfd,5);
clilen = sizeof(cli_addr);
newsockfd = accept(sockfd,(struct sockaddr *) &cli_addr, &clilen);
if(newsockfd < 0)
{
  error("Error on accept");
}
bzero(buffer,256);
n = read(newsockfd,buffer,255);
if(n<0)
{
    error("Reading from socket");
}
printf("Here is message: %s\n",buffer);
n = write(newsockfd,"I got your message", 18);
if(n<0)
```

```
{
error("Error writing");
}
return 0;
}
```

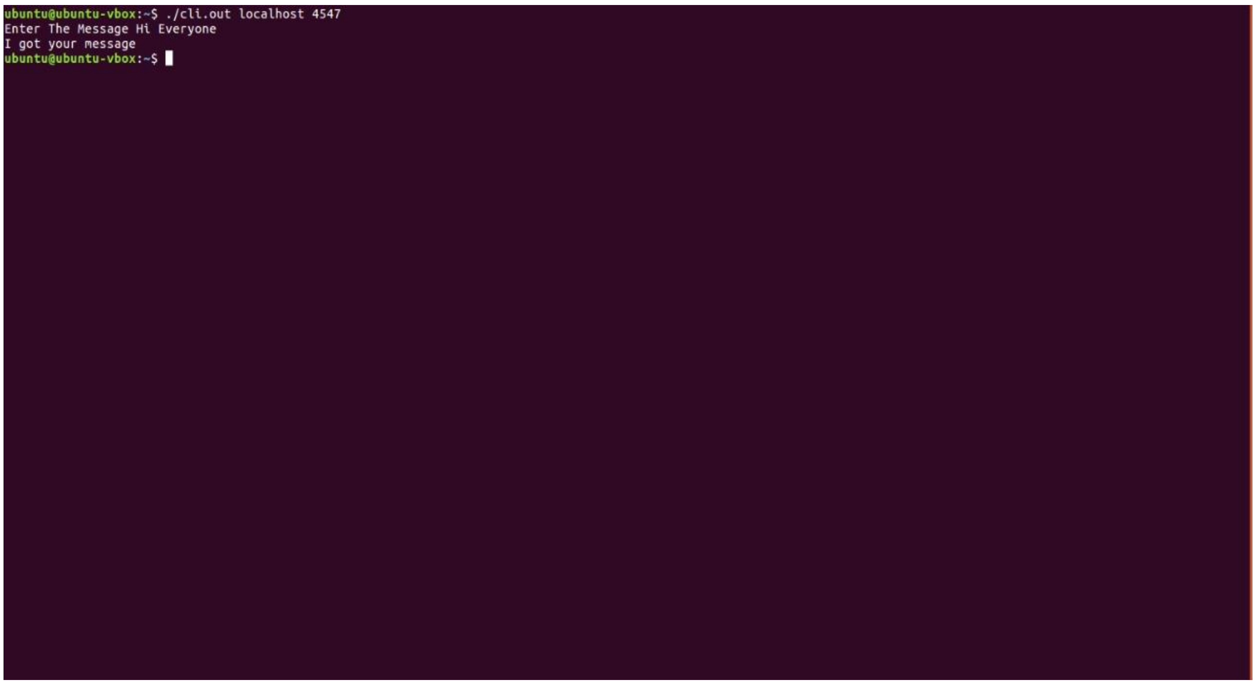
OUTPUT/SNAPSHOTS:-



```
ubuntu@ubuntu-vbox:~$ cc server1.c -o serv.out
server1.c: In function ‘error’:
server1.c:10:3: warning: implicit declaration of function ‘exit’ [-Wimplicit-function-declaration]
   exit(1);
   ^
server1.c:10:3: warning: incompatible implicit declaration of built-in function ‘exit’
server1.c:10:3: note: include ‘<stdlib.h>’ or provide a declaration of ‘exit’
server1.c: In function ‘main’:
server1.c:23:1: warning: incompatible implicit declaration of built-in function ‘exit’
 exit(1);
 ^
server1.c:23:1: note: include ‘<stdlib.h>’ or provide a declaration of ‘exit’
server1.c:32:1: warning: implicit declaration of function ‘bzero’ [-Wimplicit-function-declaration]
 bzero((char *) &serv_addr, sizeof(serv_addr));
 ^
server1.c:32:1: warning: incompatible implicit declaration of built-in function ‘bzero’
server1.c:33:10: warning: implicit declaration of function ‘atoi’ [-Wimplicit-function-declaration]
 portno = atoi(argv[1]);
          ^
server1.c:50:5: warning: implicit declaration of function ‘read’ [-Wimplicit-function-declaration]
 n = read(newsockfd,buffer,255);
     ^
server1.c:56:5: warning: implicit declaration of function ‘write’ [-Wimplicit-function-declaration]
 n = write(newsockfd,"I got your message", 18);
     ^
ubuntu@ubuntu-vbox:~$ cc client1.c -o cli.out
client1.c: In function ‘error’:
client1.c:11:4: warning: implicit declaration of function ‘exit’ [-Wimplicit-function-declaration]
    exit(0);
    ^
client1.c:11:4: warning: incompatible implicit declaration of built-in function ‘exit’
client1.c:11:4: note: include ‘<stdlib.h>’ or provide a declaration of ‘exit’
client1.c: In function ‘main’:
client1.c:24:1: warning: incompatible implicit declaration of built-in function ‘exit’
 exit(0);
 ^
client1.c:24:1: note: include ‘<stdlib.h>’ or provide a declaration of ‘exit’
client1.c:27:10: warning: implicit declaration of function ‘atoi’ [-Wimplicit-function-declaration]
 portno = atoi(argv[2]);
          ^
client1.c:39:3: warning: incompatible implicit declaration of built-in function ‘exit’
   exit(0);
   ^
client1.c:39:3: note: include ‘<stdlib.h>’ or provide a declaration of ‘exit’
client1.c:41:1: warning: implicit declaration of function ‘bzero’ [-Wimplicit-function-declaration]
```



```
    ^
client1.c:11:4: warning: incompatible implicit declaration of built-in function ‘exit’
client1.c:11:4: note: include ‘<stdlib.h>’ or provide a declaration of ‘exit’
client1.c: In function ‘main’:
client1.c:24:1: warning: incompatible implicit declaration of built-in function ‘exit’
 exit(0);
 ^
client1.c:24:1: note: include ‘<stdlib.h>’ or provide a declaration of ‘exit’
client1.c:27:10: warning: implicit declaration of function ‘atoi’ [-Wimplicit-function-declaration]
 portno = atoi(argv[2]);
          ^
client1.c:39:3: warning: incompatible implicit declaration of built-in function ‘exit’
   exit(0);
   ^
client1.c:39:3: note: include ‘<stdlib.h>’ or provide a declaration of ‘exit’
client1.c:41:1: warning: implicit declaration of function ‘bzero’ [-Wimplicit-function-declaration]
 bzero((char *) &serv_addr,sizeof(serv_addr));
 ^
client1.c:41:1: warning: incompatible implicit declaration of built-in function ‘bzero’
client1.c:44:1: warning: implicit declaration of function ‘bcopy’ [-Wimplicit-function-declaration]
 bcopy((char *)server->h_addr,(char *)&serv_addr.sin_addr.s_addr,server->h_length);
 ^
client1.c:44:1: warning: incompatible implicit declaration of built-in function ‘bcopy’
client1.c:46:19: warning: passing argument 2 of ‘connect’ from incompatible pointer type [-Wincompatible-pointer-types]
 if(connect(sockfd,&serv_addr,sizeof(serv_addr)) < 0)
                   ^
In file included from client1.c:3:0:
/usr/include/x86_64-linux-gnu/sys/socket.h:137:12: note: expected ‘const struct sockaddr *’ but argument is of type ‘struct sockaddr_in *’
 extern int connect (int __fd, __CONST_SOCKADDR_ARG __addr, socklen_t __len);
            ^
client1.c:54:5: warning: implicit declaration of function ‘write’ [-Wimplicit-function-declaration]
 n = write(sockfd,buffer,strlen(buffer));
     ^
client1.c:54:25: warning: implicit declaration of function ‘strlen’ [-Wimplicit-function-declaration]
 n = write(sockfd,buffer,strlen(buffer));
                         ^
client1.c:54:25: warning: incompatible implicit declaration of built-in function ‘strlen’
client1.c:54:25: note: include ‘<string.h>’ or provide a declaration of ‘strlen’
client1.c:61:5: warning: implicit declaration of function ‘read’ [-Wimplicit-function-declaration]
 n = read(sockfd,buffer,255);
     ^
ubuntu@ubuntu-vbox:~$ ./serv.out 4547
Here is message:  Hi Everyone

ubuntu@ubuntu-vbox:~$ 
```



```
ubuntu@ubuntu-vbox:~$ ./cli.out localhost 4547
Enter The Message Hi Everyone
I got your message
ubuntu@ubuntu-vbox:~$
```

CONCLUSION

- The main goal of this project is to develop a client server architecture which is done by using tcp/ip network model and also socket programming concept by which the peer to peer communication takes place by programming a port (such as specifying the port with a dedicated number)
- It might be useful in saving the high cost for the connection setup such as UDP protocal and also it will not be cumbersome to develop a socket programming as it is easy and well designated steps by using some services is API's.