

This function call creates a socket and returns a socket descriptor. The domain parameter specifies a communication domain; this selects the protocol family which will be used for communication. These families are defined in `<sys/socket.h>`. In this program, the domain `AF_INET` is used. The socket has the indicated type, which specifies the communication semantics. `SOCK_STREAM` type provides sequenced, reliable, two-way, connection based byte streams. The protocol field specifies the protocol used. We always use 0. If the system call is a failure, a -1 is returned. The header files used are `sys/types.h` and `sys/socket.h`.

2. Filling the fields of the server address structure.

The socket address structure is of type `struct sockaddr_in`.

```
struct sockaddr_in {  
  
    u_short sin_family;  
    u_short sin_port;  
    struct in_addr sin_addr;  
    char sin_zero[8]; /*unused, always zero*/  
};  
struct in_addr {  
  
    u_long s_addr;  
  
};
```

The fields of the socket address structure are
sin_family which in our case is `AF_INET`
sin_port which is the port number where socket binds
sin_addr which is the IP address of the server machine

The header file that is to be used is **netinet/in.h**

```
struct sockaddr_in servaddr;  
servaddr.sin_family = AF_INET;  
servaddr.sin_port = htons(port_number);
```

Why `htons` is used ?. Numbers on different machines may be represented differently (big-endian machines and little-endian machines). In a little-endian machine the low order byte of an integer appears at the lower address; in a big-endian machine instead the low order byte appears at the higher address. Network order, the order in which numbers are sent on the internet is big-endian.

It is necessary to ensure that the right representation is used on each machine. Functions are used to convert from host to network form before transmission- htons for short integers and htonl for long integers.

The value for servaddr.sin_addr is assigned using the following function

```
inet_pton(AF_INET, "IP_Address", &servaddr.sin_addr);
```

The binary value of the dotted decimal IP address is stored in the field when the function returns.

3. Binding of the client socket to a local port

This is optional in the case of client and we usually do not use the bind function on the client side.

4. Connection of client to the server

A server is identified by an IP address and a port number. The connection operation is used on the client side to identify and start the connection to the server.

```
int connect(int sd, struct sockaddr * addr, int addrlen);
```

sd – file descriptor of local socket

addr – pointer to protocol address of other socket

addrlen – length in bytes of address structure

The header files to be used are sys/types.h and sys/socket.h

It returns 0 on success and -1 in case of failure.

5. Reading from socket

In the case of TCP connection reading from a socket can be done using the read system call

```
int read(int sd, char * buf, int length);
```

6. writing to a socket

In the case of TCP connection writing to a socket can be done using the write system call

```
int write( int sd, char * buf, int length);
```

7. closing the connection

The connection can be closed using the close system call

```
int close( int sd);
```

Steps for TCP Connection for server

1. Creating a listening socket

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

This system call creates a socket and returns a socket descriptor. The domain field used is **AF_INET**. The socket type is **SOCK_STREAM**. The protocol field is 0. If the system is a failure, a -1 is returned. Header files used are sys/types.h and sys/socket.h.

2. Binding to a local port

```
int bind(int sd, struct sockaddr * addr, int addrlen);
```

This call is used to specify for a socket the protocol port number where it will wait for messages. A call to bind is optional on the client side, but required on the server side. The first field is the socket descriptor of the local socket. Second is a pointer to the protocol address structure of this socket. The third is the length in bytes of the structure referenced by **addr**. This system call returns an integer. It is 0 for success and -1 for failure. The header files are **sys/types.h** and **sys/socket.h**.

3. Listening on the port

The listen function is used on the server in connection oriented communication to prepare a **socket** to accept messages from clients.

```
int listen(int fd, int qlen);
```

fd – file descriptor of a socket that has already been bound

qlen – specifies the maximum number of messages that can wait to be processed by the server while the server is busy servicing another request. Usually it is taken as 5. The header files used are sys/types.h and sys/socket.h. This function returns 0 on success and -1 on failure.

4. Accepting a connection from the client

The accept function is used on the server in the case of connection oriented communication to accept a connection request from a client.

int accept(int fd, struct sockaddr * addressp, int * addrlen);

The first field is the descriptor of the server socket that is listening. The second parameter **addressp** points to a socket address structure that will be filled by the address of calling client when the function returns. The third parameter **addrlen** is an integer that will contain the actual length of address structure of the client. It returns an integer that is a descriptor of a new socket called the connection socket. Server sockets send data and read data from this socket. The header files used are sys/types.h and sys/socket.h.

Algorithm

Client

1. Create socket
2. Connect the socket to the server
3. Read the string to be reversed from the standard input and send it to the server
Read the matrices from the standard input and send it to server using socket
4. Read the reversed string from the socket and display it on the standard output
Read product matrix from the socket and display it on the standard output
5. Close the socket

Server

1. Create listening socket
2. bind IP address and port number to the socket
3. listen for incoming requests on the listening socket
4. accept the incoming request
5. connection socket is created when accept returns
6. Read the string using the connection socket from the client
7. Reverse the string
8. Send the string to the client using the connection socket
9. close the connection socket
10. close the listening socket

Client Program

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

```
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<fcntl.h>
#include<string.h>
#include<stdlib.h>
#include<unistd.h>

int main( int argc, char *argv[])
{
    struct sockaddr_in server;
    int sd ;
    char buffer[200];
    if((sd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        perror("Socket failed.");
        exit(1);
    }
    // server socket address structure initialisation
    bzero(&server, sizeof(server) );
    server.sin_family = AF_INET;
    server.sin_port = htons(atoi(argv[2]));
    inet_pton(AF_INET, argv[1], &server.sin_addr);
    if(connect(sd, (struct sockaddr *)&server, sizeof(server))< 0)
    {
        perror("Connection failed.");
        exit(1);
    }
    fgets(buffer, sizeof(buffer), stdin);
    buffer[strlen(buffer) - 1] = '\0';
    write (sd,buffer, sizeof(buffer));
    read(sd,buffer, sizeof(buffer));
    printf("%s\n", buffer);
    close(fd);
}
```

Server Program

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

```
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<fcntl.h>
#include<string.h>
#include<stdlib.h>
#include<unistd.h>

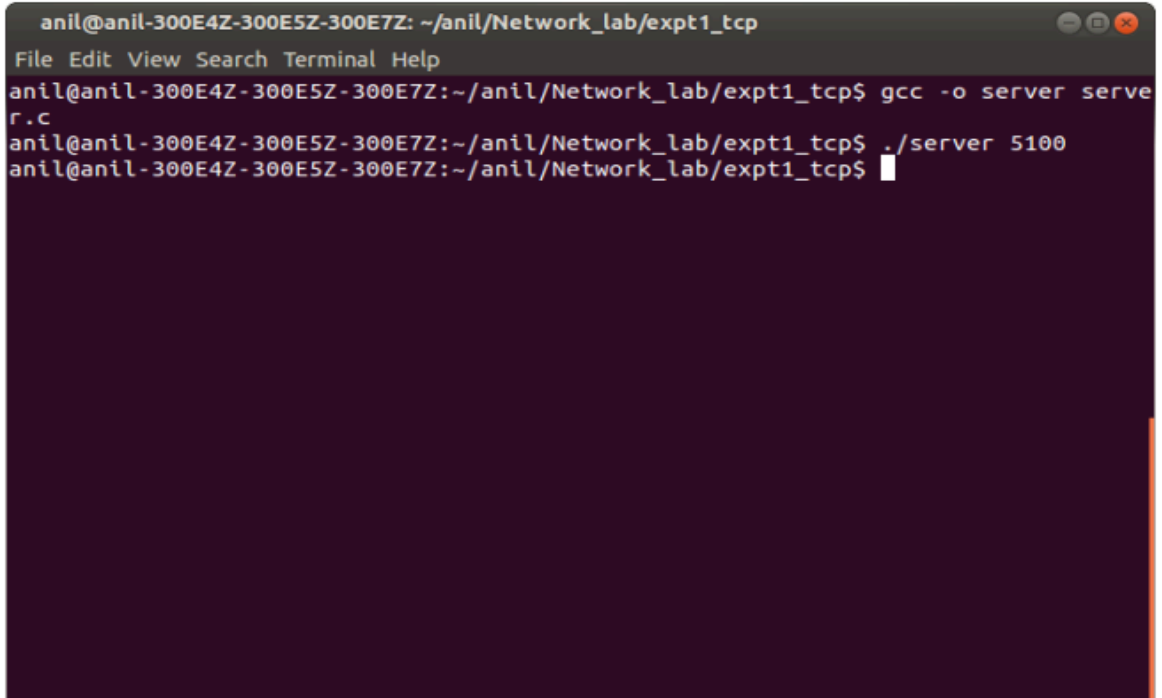
int main( int argc, char *argv[])
{
    struct sockaddr_in server, cli;
    int cli_len;
    int sd, n, i, len;
    int data, temp;
    char buffer[100];
    if((sd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        perror("Socket failed:");
        exit(1);
    }

    // server socket address structure initialisation
    bzero(&server, sizeof(server) );
    server.sin_family = AF_INET;
    server.sin_port = htons(atoi(argv[1]));
    server.sin_addr.s_addr = htonl(INADDR_ANY);
    if(bind(sd, (struct sockaddr*)&server, sizeof(server)) < 0)
    {
        perror("bind failed:");
        exit(1);
    }
    listen(sd,5);
    if((data = accept(sd , (struct sockaddr *) &cli, &cli_len)) < 0)
    {
        perror("accept failed:");
        exit(1);
    }
    read(data,buffer, sizeof(buffer));
    len = strlen(buffer);
```

```
for( i =0; i<= len/2; i++)  
{  
temp = buffer[i];  
buffer[i] = buffer[len - 1-i];  
buffer[len-1-i] = temp;  
}  
write (data,buffer, sizeof(buffer));  
close(data);  
close(sd);  
}
```

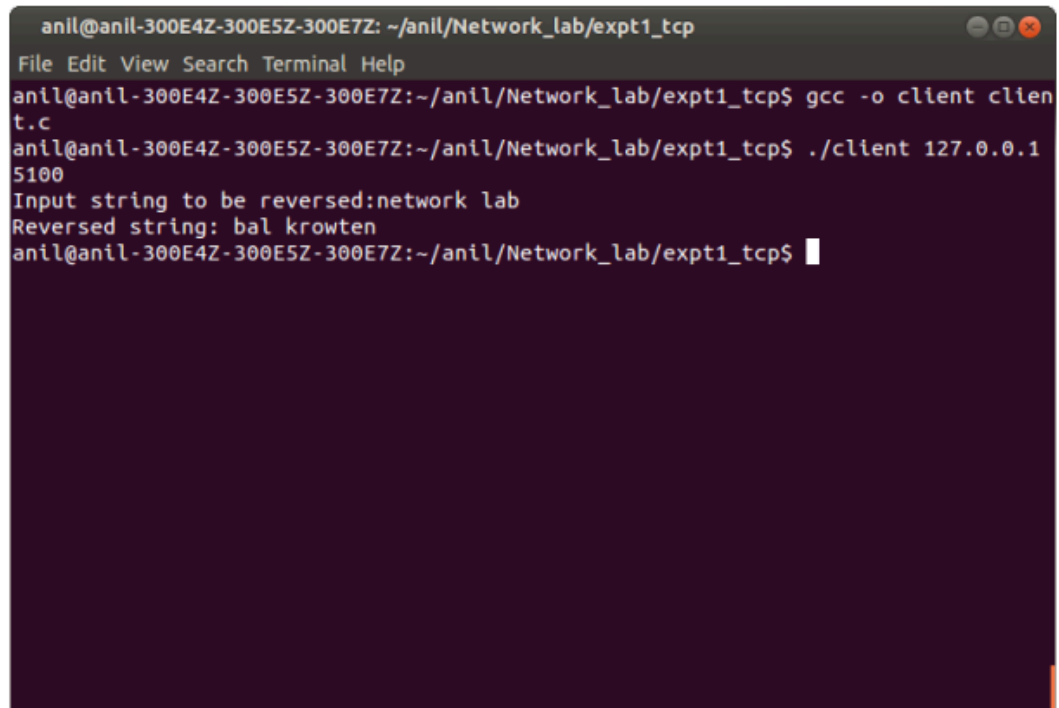
Output

Open with ▼

Server

The screenshot shows a terminal window titled "anil@anil-300E4Z-300E5Z-300E7Z: ~/anil/Network_lab/expt1_tcp". The terminal has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The user enters the command `gcc -o server server.c` to compile the program. The next command is `./server 5100`, which starts the server. The prompt returns to `anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt1_tcp$` with a cursor at the end.

```
anil@anil-300E4Z-300E5Z-300E7Z: ~/anil/Network_lab/expt1_tcp  
File Edit View Search Terminal Help  
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt1_tcp$ gcc -o server server.c  
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt1_tcp$ ./server 5100  
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt1_tcp$
```

Client

```
anil@anil-300E4Z-300E5Z-300E7Z: ~/anil/Network_lab/expt1_tcp
File Edit View Search Terminal Help
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt1_tcp$ gcc -o client client.c
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt1_tcp$ ./client 127.0.0.1 5100
Input string to be reversed:network lab
Reversed string: bal krowten
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt1_tcp$
```

Experiment 8

Implementation of Client-Server communication using Socket Programming and UDP as transport layer protocol

Aim: Client sends two matrices to the server using udp protocol. The server multiplies the matrices and sends the product to the client, which then displays the product matrix.

Description:

Steps for transfer of data using UDP

1. Creation of UDP socket

The function call for creating a UDP socket is

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

The domain parameter specifies a communication domain; this selects the protocol family which will be used for communication. These families are defined in <sys/socket.h>. In this program,