

```
#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>

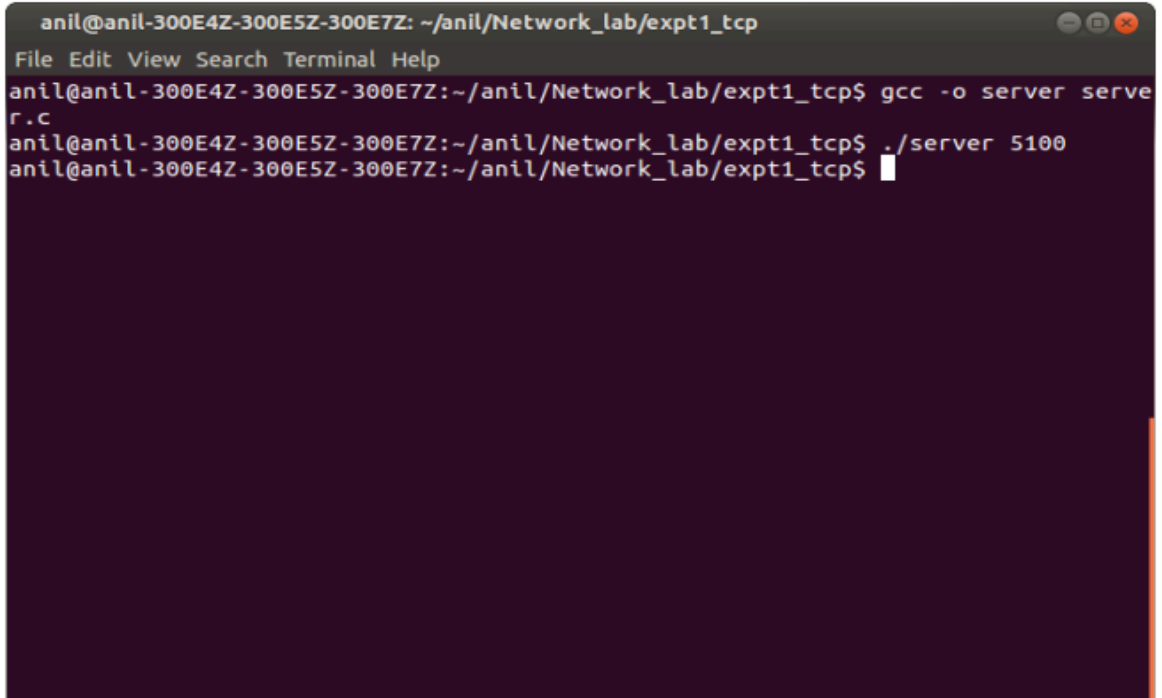
int main( int argc, char *argv[])
{
    struct sockaddr_in server, cli;
    int cli_len;
    int sd, n, i, len;
    int data, temp;
    char buffer[100];
    if((sd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        perror("Socket failed:");
        exit(1);
    }

    // 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);
}
```

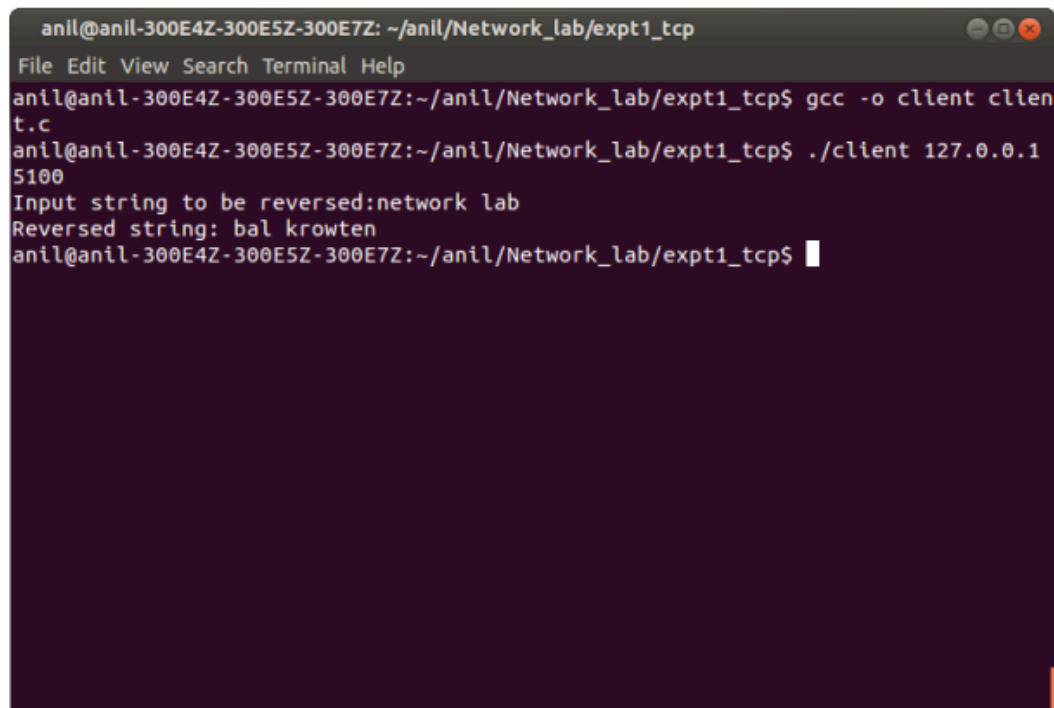
Output

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,

the domain **AF_INET** is used. The next field type has the value **SOCK_DGRAM**. It supports datagrams (connectionless, unreliable messages of a fixed maximum length). The protocol field specifies the protocol used. We always use 0. If the socket function call is successful, a socket descriptor is returned. Otherwise -1 is returned. The header files necessary for this function call 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 is used to store the IP address of the server machine and is of type `struct in_addr`

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

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 a port to the socket in the case of server