

Network Programming

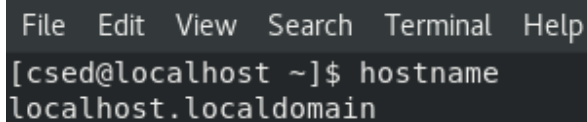
Assignment:1

- 1. Familiarity with Lab environment and understanding Client-Server model, Unix basic commands, socket programming header files, and elementary socket system calls.**
- (a) Understanding and using of commands like ifconfig, netstat, ping, arp, telnet, ftp, finger, traceroute, whois etc.**

Answers:

(a) hostname

To communicate with each and other, the computer needs a unique address. A hostname can be alphabetic or alphanumeric and contain specific symbols used specifically to define a specific node or device in the network.

A terminal window with a dark background. The menu bar at the top shows 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The command prompt is '[csed@localhost ~]\$' and the command entered is 'hostname'. The output of the command is 'localhost.localdomain'.

```
File Edit View Search Terminal Help
[csed@localhost ~]$ hostname
localhost.localdomain
```

(b) Ifconfig

Ifconfig command is used to configure the kernel-resident network interfaces. It is used at the boot time to set up the interfaces as necessary. After that, it is usually used when needed during debugging or when you need system tuning.

SMARTBOY

```
File Edit View Search Terminal Help
[csed@localhost ~]$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.2.15 netmask 255.255.255.0 broadcast 10.0.2.255
    inet6 fe80::de92:2afc:f2d4:dcc8 prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:01:78:ef txqueuelen 1000 (Ethernet)
    RX packets 46 bytes 5091 (4.9 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 83 bytes 8636 (8.4 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 4 bytes 315 (315.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 4 bytes 315 (315.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

virbr0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    inet 192.168.122.1 netmask 255.255.255.0 broadcast 192.168.122.255
    ether 52:54:00:77:5d:96 txqueuelen 1000 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

[csed@localhost ~]$ netstat -at
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 localhost.locald:domain 0.0.0.0:*               LISTEN
tcp        0      0 localhost:ipp            0.0.0.0:*               LISTEN
tcp6       0      0 localhost:ipp            [::]:*                 LISTEN
```

(c) netstat

Netstat is a Common TCP – IP networking command-line method present in most Windows, Linux, UNIX, and other operating systems. The netstat provides the statistics and information in the use of the current TCP-IP Connection network about the protocol.

```
[csed@localhost ~]$ netstat -at
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 localhost.locald:domain 0.0.0.0:*               LISTEN
tcp        0      0 localhost:ipp            0.0.0.0:*               LISTEN
tcp6       0      0 localhost:ipp            [::]:*                 LISTEN
```

SMARTBOY

```
[csed@localhost ~]$ netstat -e
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       User        Inode
Active UNIX domain sockets (w/o servers)
Proto RefCnt Flags   Type       State       I-Node      Path
unix  2      [ ]     DGRAM          31495      /run/user/1000/systemd/notify
unix  2      [ ]     DGRAM          20236      /var/run/chrony/chronyd.sock
unix  2      [ ]     DGRAM          23692      /run/user/42/systemd/notify
unix  3      [ ]     DGRAM          11940      /run/systemd/notify
unix  8      [ ]     DGRAM          11963      /run/systemd/journal/socket
unix 26      [ ]     DGRAM          11978      /run/systemd/journal/dev-log
unix  3      [ ]     STREAM        CONNECTED  33766
unix  3      [ ]     STREAM        CONNECTED  33094
unix  3      [ ]     STREAM        CONNECTED  33086
unix  3      [ ]     STREAM        CONNECTED  29130
unix  3      [ ]     STREAM        CONNECTED  28954      @/tmp/dbus-hgculXC9ZF
unix  3      [ ]     STREAM        CONNECTED  34999      /run/user/1000/bus
unix  3      [ ]     STREAM        CONNECTED  34520      @/tmp/.X11-unix/X0
unix  3      [ ]     STREAM        CONNECTED  36037      /run/systemd/journal/stdout
unix  2      [ ]     DGRAM          35808
unix  2      [ ]     DGRAM          34580
unix  3      [ ]     STREAM        CONNECTED  34167      /run/systemd/journal/stdout
unix  3      [ ]     STREAM        CONNECTED  29451
unix  3      [ ]     STREAM        CONNECTED  28902
unix  3      [ ]     STREAM        CONNECTED  32937      @/tmp/dbus-SFHgNsZRA8
unix  3      [ ]     STREAM        CONNECTED  28782      /run/systemd/journal/stdout
unix  3      [ ]     STREAM        CONNECTED  33763      /run/systemd/journal/stdout
unix  3      [ ]     STREAM        CONNECTED  33095      /run/dbus/system_bus_socket
unix  3      [ ]     STREAM        CONNECTED  33050      /run/user/1000/bus
unix  2      [ ]     DGRAM          31470
unix  3      [ ]     STREAM        CONNECTED  29137
unix  3      [ ]     STREAM        CONNECTED  28950
unix  3      [ ]     STREAM        CONNECTED  36095
unix  3      [ ]     STREAM        CONNECTED  35349
unix  3      [ ]     STREAM        CONNECTED  34549
unix  3      [ ]     STREAM        CONNECTED  21692
unix  3      [ ]     STREAM        CONNECTED  20971
unix  3      [ ]     STREAM        CONNECTED  37302
unix  3      [ ]     STREAM        CONNECTED  36333
```

(d) whois

The whois system is a listing of records that contains details about both the ownership of domains and the owners.

SMARTBOY

```
File Edit View Search Terminal Help
[csed@localhost ~]$ whois thapar.edu
[Querying whois.educause.edu]
[whois.educause.edu]
This Registry database contains ONLY .EDU domains.
The data in the EDUCAUSE Whois database is provided
by EDUCAUSE for information purposes in order to
assist in the process of obtaining information about
or related to .edu domain registration records.

The EDUCAUSE Whois database is authoritative for the
.EDU domain.

A Web interface for the .EDU EDUCAUSE Whois Server is
available at: http://whois.educause.edu

By submitting a Whois query, you agree that this information
will not be used to allow, enable, or otherwise support
the transmission of unsolicited commercial advertising or
solicitations via e-mail. The use of electronic processes to
harvest information from this server is generally prohibited
except as reasonably necessary to register or modify .edu
domain names.

-----
Domain Name: THAPAR.EDU

Registrant:
  Thapar Institute of Engineering and Technology
  Thapar University
  Thapar University
  Patiala, PUNJAB 147004
  India

Administrative Contact:
  Harcharan Jit Singh
  Thapar Institute of Engineering and Technology
  Centre of Information and Technology Management (CITM)
  Thapar Institute of Engineering and Technology
  Patiala, Punjab, 147004
  India
```

(e) arp

arp command manipulates the System's ARP cache. It also allows a complete dump of the ARP cache. ARP stands for Address Resolution Protocol. The primary function of this protocol is to resolve the IP address of a system to its mac address.

```
[csed@localhost ~]$ arp -a
_gateway (10.0.2.2) at 52:54:00:12:35:02 [ether] on enp0s3
```

(f) Ping

Ping is used to testing a network host capacity to interact with another host. Just enter the command Ping, followed by the target host's name or IP address.

SMARTBOY

```
[csed@localhost ~]$ ping www.google.com
PING www.google.com (142.250.194.196) 56(84) bytes of data.
64 bytes from dell2s07-in-f4.1e100.net (142.250.194.196): icmp_seq=1 ttl=116 time=16.5 ms
64 bytes from dell2s07-in-f4.1e100.net (142.250.194.196): icmp_seq=2 ttl=116 time=12.2 ms
64 bytes from dell2s07-in-f4.1e100.net (142.250.194.196): icmp_seq=3 ttl=116 time=12.0 ms
64 bytes from dell2s07-in-f4.1e100.net (142.250.194.196): icmp_seq=4 ttl=116 time=12.5 ms
64 bytes from dell2s07-in-f4.1e100.net (142.250.194.196): icmp_seq=5 ttl=116 time=11.9 ms
64 bytes from dell2s07-in-f4.1e100.net (142.250.194.196): icmp_seq=6 ttl=116 time=11.9 ms
64 bytes from dell2s07-in-f4.1e100.net (142.250.194.196): icmp_seq=7 ttl=116 time=12.0 ms
64 bytes from dell2s07-in-f4.1e100.net (142.250.194.196): icmp_seq=8 ttl=116 time=11.9 ms
64 bytes from dell2s07-in-f4.1e100.net (142.250.194.196): icmp_seq=9 ttl=116 time=12.0 ms
64 bytes from dell2s07-in-f4.1e100.net (142.250.194.196): icmp_seq=10 ttl=116 time=12.0 ms
64 bytes from dell2s07-in-f4.1e100.net (142.250.194.196): icmp_seq=11 ttl=116 time=11.9 ms
64 bytes from dell2s07-in-f4.1e100.net (142.250.194.196): icmp_seq=12 ttl=116 time=12.0 ms
64 bytes from dell2s07-in-f4.1e100.net (142.250.194.196): icmp_seq=13 ttl=116 time=12.2 ms
64 bytes from dell2s07-in-f4.1e100.net (142.250.194.196): icmp_seq=14 ttl=116 time=11.9 ms
64 bytes from dell2s07-in-f4.1e100.net (142.250.194.196): icmp_seq=15 ttl=116 time=11.9 ms
64 bytes from dell2s07-in-f4.1e100.net (142.250.194.196): icmp_seq=16 ttl=116 time=11.8 ms
64 bytes from dell2s07-in-f4.1e100.net (142.250.194.196): icmp_seq=17 ttl=116 time=11.8 ms
64 bytes from dell2s07-in-f4.1e100.net (142.250.194.196): icmp_seq=18 ttl=116 time=12.2 ms
64 bytes from dell2s07-in-f4.1e100.net (142.250.194.196): icmp_seq=19 ttl=116 time=12.0 ms
64 bytes from dell2s07-in-f4.1e100.net (142.250.194.196): icmp_seq=20 ttl=116 time=12.2 ms
64 bytes from dell2s07-in-f4.1e100.net (142.250.194.196): icmp_seq=21 ttl=116 time=11.9 ms
64 bytes from dell2s07-in-f4.1e100.net (142.250.194.196): icmp_seq=22 ttl=116 time=12.1 ms
64 bytes from dell2s07-in-f4.1e100.net (142.250.194.196): icmp_seq=23 ttl=116 time=12.2 ms
64 bytes from dell2s07-in-f4.1e100.net (142.250.194.196): icmp_seq=24 ttl=116 time=12.2 ms
64 bytes from dell2s07-in-f4.1e100.net (142.250.194.196): icmp_seq=25 ttl=116 time=13.2 ms
64 bytes from dell2s07-in-f4.1e100.net (142.250.194.196): icmp_seq=26 ttl=116 time=12.0 ms
64 bytes from dell2s07-in-f4.1e100.net (142.250.194.196): icmp_seq=27 ttl=116 time=12.3 ms
64 bytes from dell2s07-in-f4.1e100.net (142.250.194.196): icmp_seq=28 ttl=116 time=11.9 ms
64 bytes from dell2s07-in-f4.1e100.net (142.250.194.196): icmp_seq=29 ttl=116 time=12.2 ms
64 bytes from dell2s07-in-f4.1e100.net (142.250.194.196): icmp_seq=30 ttl=116 time=12.2 ms
64 bytes from dell2s07-in-f4.1e100.net (142.250.194.196): icmp_seq=31 ttl=116 time=12.2 ms
64 bytes from dell2s07-in-f4.1e100.net (142.250.194.196): icmp_seq=32 ttl=116 time=11.9 ms
64 bytes from dell2s07-in-f4.1e100.net (142.250.194.196): icmp_seq=33 ttl=116 time=12.0 ms
64 bytes from dell2s07-in-f4.1e100.net (142.250.194.196): icmp_seq=34 ttl=116 time=12.1 ms
64 bytes from dell2s07-in-f4.1e100.net (142.250.194.196): icmp_seq=35 ttl=116 time=11.9 ms
```

(g) finger

Finger command is a user information lookup command which gives details of all the users logged in. This tool is generally used by system administrators. It provides details like login name, user name, idle time, login time, and in some cases their email address even.

```
[csed@localhost ~]$ finger csed
Login: csed                      Name: CSED
Directory: /home/csed           Shell: /bin/bash
On since Mon Aug  9 14:55 (IST) on tty2 from tty2
    22 minutes 28 seconds idle
No mail.
No Plan.
```


(h) telnet

telnet uses the Telnet protocol (part of the TCP/IP protocol suite) to connect to a remote computer over a network. You can execute telnet from the Command (DOS) prompt. Syntax: telnet [RemoteServer] [Port] [RemoteServer]: specify the name of the server to which you want to connect.

```
[csed@localhost ~]$ arp -a
_gateway (10.0.2.2) at 52:54:00:12:35:02 [ether] on enp0s3
[csed@localhost ~]$ telnet 10.0.2.2 13531
Trying 10.0.2.2...
telnet: connect to address 10.0.2.2: Connection refused
```

(i) traceroute

The tracert command is a command which is used to get the network packet being sent and received and the number of hops required for that packet to reach to target. This command can also be referred to as a traceroute.

```
[csed@localhost ~]$ traceroute goole.com

traceroute to goole.com (217.160.0.201), 30 hops max, 60 byte packets
 1 _gateway (10.0.2.2)  0.782 ms  0.816 ms  0.798 ms
 2 * * *
 3 * * *
```

(j) ftp

FTP (File Transfer Protocol) is a standard network protocol used to transfer files to and from a remote network.

Establishing an FTP Connection

To open an ftp connection to a remote system, invoke the ftp command followed by the remote server IP address or domain name. For example, to connect to an FTP server at “192.168.42.77” you would type: [ftp 192.168.42.77](ftp://192.168.42.77)

If the connection is established, a confirmation message will be displayed, and you will be prompted to enter your FTP username.

Once you enter the username you will be prompted to type your password.

If the password is correct, the remote server will display a confirmation message and the ftp> prompt.

Below are some of the most common FTP commands

SMARTBOY

(k) help or ? - list all available FTP commands.

(l) cd - change directory on the remote machine.

(m) lcd - change directory on the local machine.

(n) ls - list the names of the files and directories in the current remote directory.

(o) mkdir - create a new directory within the current remote directory.

(p) pwd - print the current working directory on the remote machine.

(q) delete - remove a file in the current remote directory.

(r) rmdir - remove a directory in the current remote directory.

(s) get - copy one file from the remote to the local machine.

(t) mget - copy multiple files from the remote to the local machine.

(u) put - copy one file from the local to the remote machine.

(v) mput - copy multiple files from the local to the remote machine

(b) Socket header files contain data definitions, structures, constants, macros, and options used by socket subroutines. An application program must include the appropriate header file to make use of structures or other information a particular socket subroutine requires.

Commonly used socket header files are:

- **/usr/include/netinet/in.h**: Defines Internet constants and structures.
- **/usr/include/netdb.h** Contains data definitions for socket subroutines.
- **/usr/include/sys/socket.h** Contains data definitions and socket structures.
- **/usr/include/sys/types.h** Contains data type definitions.
- **/usr/include/arpa.h** Contains definitions for internet operations.
- **/usr/include/sys/errno.h** Defines the errno values that are returned by drivers and other kernel-level code.

(c) Elementary socket system calls:

- **socket() System Call:** Creates an end point for communication and returns a descriptor:
int socket (int AddressFamily, int Type, int Protocol);

SMARTBOY

- **Bind() System call:** Binds a name to a socket. The bind subroutine assigns a Name parameter to an unnamed socket. It assigns a local protocol address to a socket.
`int bind (int sockfd, struct sockaddr *myaddr, int addrlen);`
- **connect() System call:** The connect function is used by a TCP client to establish a connection with a TCP server.
`int connect(int sockfd, struct sockaddr *servaddr, int addrlen);`
- **listen() System call:** This system call is used by a connection-oriented server to indicate that it is willing to receive connections.
`int listen (int sockfd, int backlog);`
- **accept() System call:** The actual connection from some client process is waited for by having the server execute the accept system call.
`int accept (int sockfd, struct sockaddr *cliaddr, int *addrlen);`
- **send(), sendto(), recv() and recvfrom() system calls:** These system calls are similar to the standard read and write functions.
- **close() system call:** The normal Unix close function is also used to close a socket and terminate a TCP connection.
`int close (int sockfd);`

Assignment:2

- 1. WAP to create a Client/Server model (chat server) using TCP socket programming in connection- oriented Scenario.**

Server Side:

```
#include <stdio.h>

#include <netdb.h>

#include <netinet/in.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <sys/types.h>

#define MAX 80

#define PORT 8080

#define SA struct sockaddr

void func(int sockfd)
{
    char buff[MAX];
    int n;
    for (;;) {
        bzero(buff, MAX);

        read(sockfd, buff, sizeof(buff));
        printf("From client: %s\t To client : ", buff);
        bzero(buff, MAX);
        n = 0;
        while ((buff[n++] = getchar()) != '\n')
```

SMARTBOY

```
    ;

    write(sockfd, buff, sizeof(buff));
    if (strncmp("exit", buff, 4) == 0) {
        printf("Server Exit...\n");
        break;
    }
}
}

int main()
{
    int sockfd, connfd, len;
    struct sockaddr_in servaddr, cli;
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        printf("socket creation failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully created..\n");
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(PORT);
    if ((bind(sockfd, (SA*)&servaddr, sizeof(servaddr))) != 0) {
        printf("socket bind failed...\n");
```

SMARTBOY

```
        exit(0);
    }
    else
        printf("Socket successfully binded..\n");
    if ((listen(sockfd, 5)) != 0) {
        printf("Listen failed...\n");
        exit(0);
    }
    else
        printf("Server listening..\n");
    len = sizeof(cli);
    connfd = accept(sockfd, (SA*)&cli, &len);
    if (connfd < 0) {
        printf("server accept failed...\n");
        exit(0);
    }
    else
        printf("server accept the client...\n");
    func(connfd);
    close(sockfd);
}
```

Client Side:

```
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

SMARTBOY

```
#include <sys/socket.h>

#define MAX 80

#define PORT 8080

#define SA struct sockaddr

void func(int sockfd)
{
    char buff[MAX];
    int n;
    for (;;) {
        bzero(buff, sizeof(buff));
        printf("Enter the string : ");
        n = 0;
        while ((buff[n++] = getchar()) != '\n')
            ;
        write(sockfd, buff, sizeof(buff));
        bzero(buff, sizeof(buff));
        read(sockfd, buff, sizeof(buff));
        printf("From Server : %s", buff);
        if ((strcmp(buff, "exit", 4)) == 0) {
            printf("Client Exit...\n");
            break;
        }
    }
}

int main()
{
    int sockfd, connfd;
```

SMARTBOY

```
struct sockaddr_in servaddr, cli;

sockfd = socket(AF_INET, SOCK_STREAM, 0);

if (sockfd == -1) {
    printf("socket creation failed...\n");
    exit(0);
}

else
    printf("Socket successfully created..\n");

bzero(&servaddr, sizeof(servaddr));

servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
servaddr.sin_port = htons(PORT);

if (connect(sockfd, (SA*)&servaddr, sizeof(servaddr)) != 0) {
    printf("connection with the server failed...\n");
    exit(0);
}

else
    printf("connected to the server..\n");

func(sockfd);

close(sockfd);
}
```

SMARTBOY

```
manik@manik:~$ ./ass2_TCP_server
```

```
Socket successfully created..
```

```
Socket successfully binded..
```

```
Server listening..
```

```
server accept the client...
```

```
From client: Hello Manik
```

```
To client: Hello Dear
```

```
0
```

```
file: ass2_TCP_client.c  
socket successfully created..
```


Assignment:3**1. WAP to create a Client/Server model (Chat Server) using UDP socket programming in Connection- less Scenario.****Server Side:**

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#define PORT 8080
#define MAXLINE 1024
int main() {
    int sockfd;
    char buffer[MAXLINE];
    char *hello = "Hello Manik";
    struct sockaddr_in servaddr, cliaddr;
    if ( (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ) {
        perror("socket creation failed");
        exit(EXIT_FAILURE); }
    memset(&servaddr, 0, sizeof(servaddr));
    memset(&cliaddr, 0, sizeof(cliaddr));
    servaddr.sin_family = AF_INET; // IPv4
    servaddr.sin_addr.s_addr = INADDR_ANY;
    servaddr.sin_port = htons(PORT);
    if ( bind(sockfd, (const struct sockaddr *)&servaddr,
        sizeof(servaddr)) < 0 )
    {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }
    int len, n;
    len = sizeof(cliaddr);
    n = recvfrom(sockfd, (char *)buffer, MAXLINE,
        MSG_WAITALL, ( struct sockaddr *) &cliaddr,
        &len);
    buffer[n] = '\0';
```

SMARTBOY

```
printf("Client : %s\n", buffer);
sendto(sockfd, (const char *)hello, strlen(hello),
        MSG_CONFIRM, (const struct sockaddr *) &cliaddr,
        len);
printf("Hello message sent.\n");

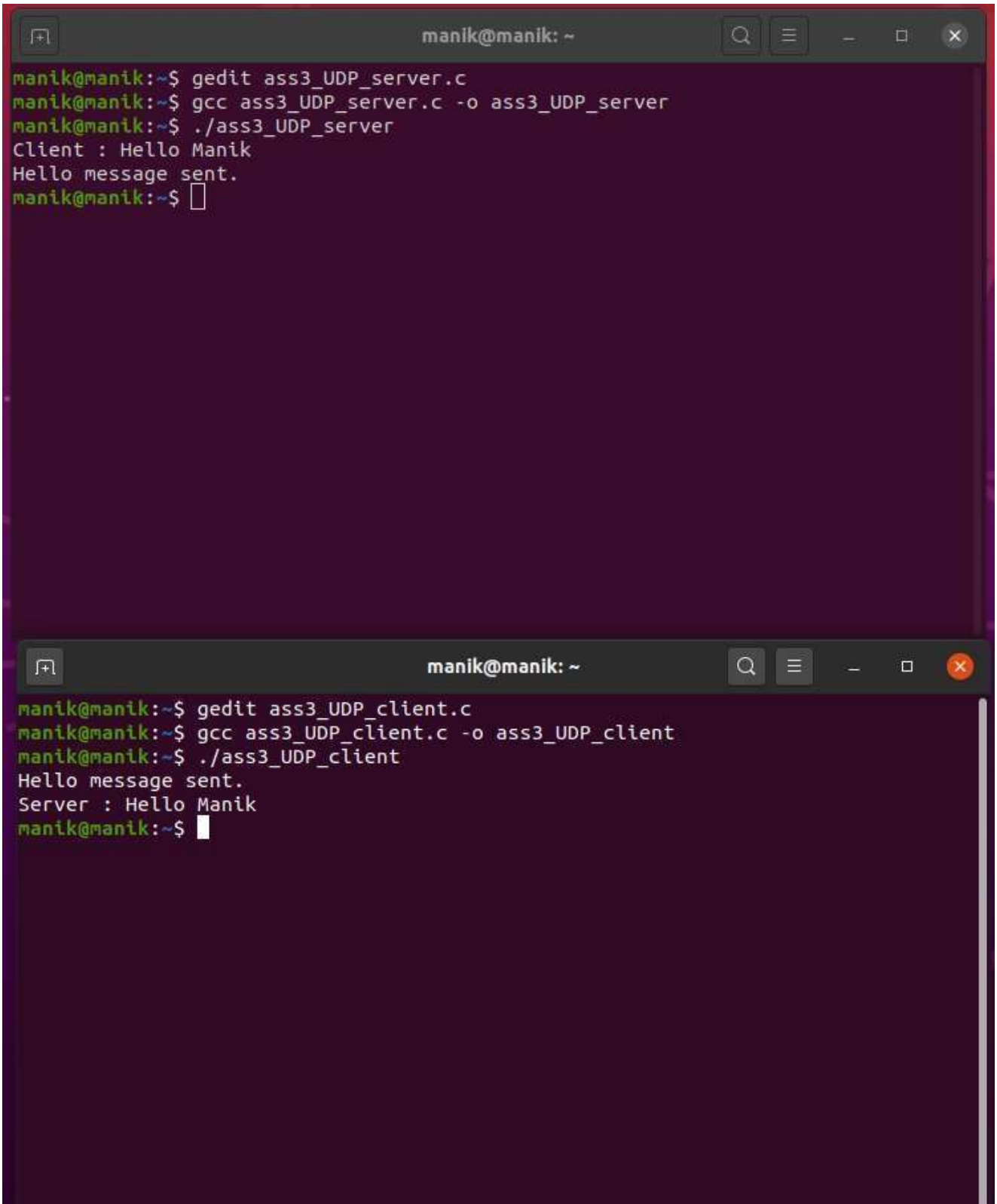
return 0;
}
```

Client Side:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#define PORT 8080
#define MAXLINE 1024
int main() {
    int sockfd;
    char buffer[MAXLINE];
    char *hello = "Hello Manik";
    struct sockaddr_in servaddr;
    if ( (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ) {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }
    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(PORT);
    servaddr.sin_addr.s_addr = INADDR_ANY;
    int n, len;
    sendto(sockfd, (const char *)hello, strlen(hello),
            MSG_CONFIRM, (const struct sockaddr *) &servaddr,
            sizeof(servaddr));
    printf("Hello message sent.\n");
    n = recvfrom(sockfd, (char *)buffer, MAXLINE,
                 MSG_WAITALL, (struct sockaddr *) &servaddr,
                 &len);
```

SMARTBOY

```
buffer[n] = '\0';  
printf("Server : %s\n", buffer);  
close(sockfd);  
return 0;  
}
```



The image shows two terminal windows from a user named 'manik' on a machine named 'manik'. The top window shows the compilation and execution of a UDP server program. The bottom window shows the compilation and execution of a UDP client program. The client sends a message, and the server responds.

```
manik@manik: ~  
manik@manik:~$ gedit ass3_UDP_server.c  
manik@manik:~$ gcc ass3_UDP_server.c -o ass3_UDP_server  
manik@manik:~$ ./ass3_UDP_server  
Client : Hello Manik  
Hello message sent.  
manik@manik:~$  
  
manik@manik:~$ gedit ass3_UDP_client.c  
manik@manik:~$ gcc ass3_UDP_client.c -o ass3_UDP_client  
manik@manik:~$ ./ass3_UDP_client  
Hello message sent.  
Server : Hello Manik  
manik@manik:~$
```

Assignment:4

- 1. WAP to implement a string reversal client-server user-level application using TCP socket API in C. You need to setup client-server connection, once connection setups, server accepts strings from clients and replies with reverse strings. For example, when client sends "TIETP", Server replies with "PTEIT". Both server and client have to output both sending & receiving strings on the terminal. The server and client processes should be run on different machines.**

Server Side:

```
#include <netinet/in.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <unistd.h>

#define PORT 8090

int main()
{
    int server_fd, new_socket, valread;
    struct sockaddr_in address;
    char str[100];
    int addrlen = sizeof(address);
    char buffer[1024] = { 0 };
    char* hello = "Hello from server";
    if ((server_fd = socket(AF_INET,
                          SOCK_STREAM, 0)) == 0) {
        perror("socket failed");
        exit(EXIT_FAILURE);
    }
```

SMARTBOY

```
address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons(PORT);
if (bind(server_fd, (struct sockaddr*)&address,
        sizeof(address)) < 0) {
    perror("bind failed");
    exit(EXIT_FAILURE);
}
if (listen(server_fd, 3) < 0) {
    perror("listen");
    exit(EXIT_FAILURE);
}
if ((new_socket = accept(server_fd,
        (struct sockaddr*)&address,
        (socklen_t*)&addrlen)) < 0) {
    perror("accept");
    exit(EXIT_FAILURE);
}
valread = read(new_socket, str,
        sizeof(str));
int i, j, temp;
int l = strlen(str);
printf("\nString sent by client:%s\n", str);
for (i = 0, j = l - 1; i < j; i++, j--) {
    temp = str[i];
    str[i] = str[j];
    str[j] = temp;
}
```

SMARTBOY

```
}  
  
send(new_socket, str, sizeof(str), 0);  
printf("\nModified string sent to client\n");  
return 0;  
}
```

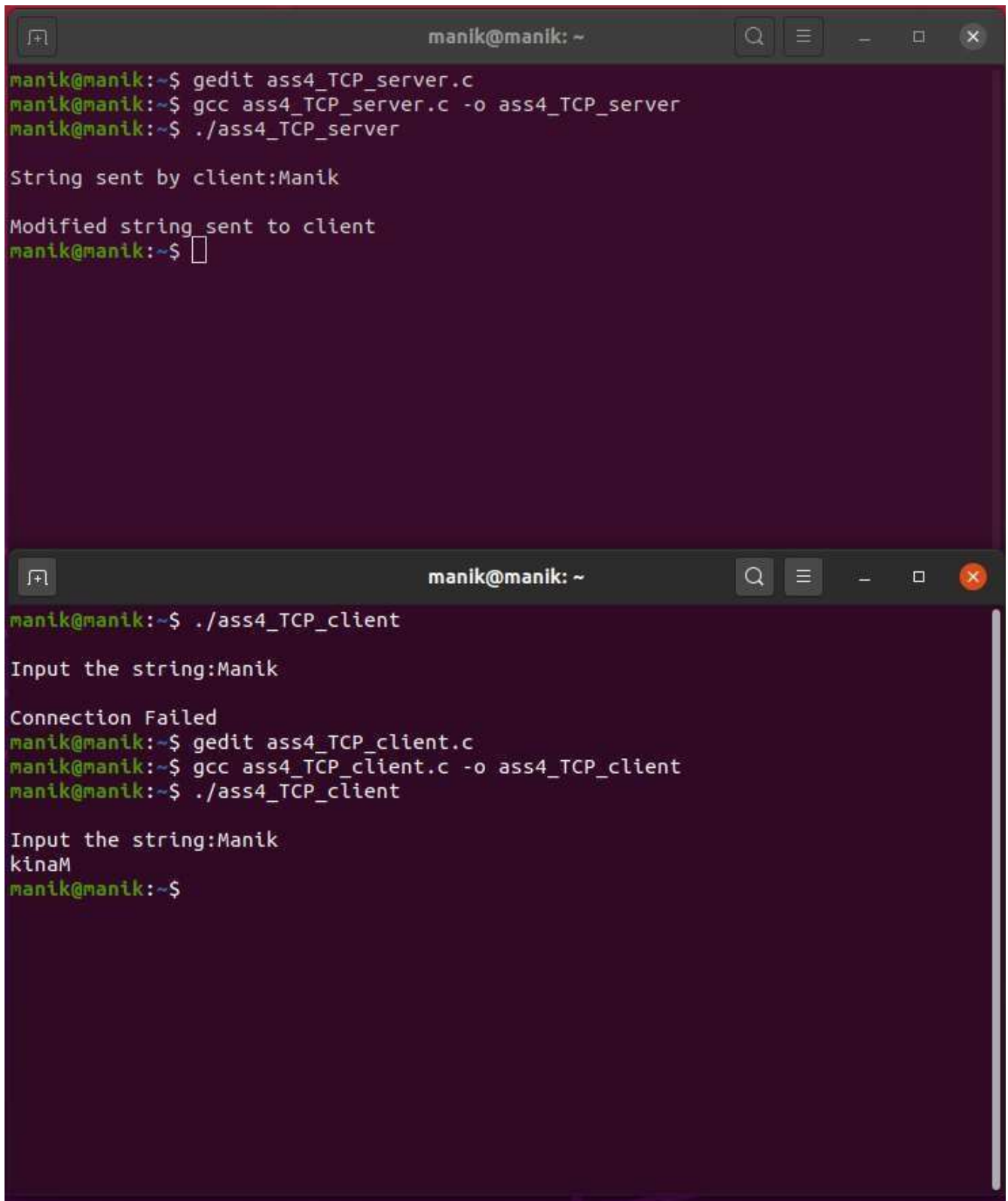
Client Side:

```
#include <arpa/inet.h>  
#include <netinet/in.h>  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <sys/socket.h>  
#include <unistd.h>  
#define PORT 8090  
int main()  
{  
    struct sockaddr_in address;  
    int sock = 0, valread;  
    struct sockaddr_in serv_addr;  
    char str[100];  
    printf("\nInput the string:");  
    scanf("%[^\n]s", str);  
  
    char buffer[1024] = { 0 };  
    if ((sock = socket(AF_INET,  
                        SOCK_STREAM, 0))
```


SMARTBOY

```
< 0) {  
    printf("\n Socket creation error \n");  
    return -1;  
}  
memset(&serv_addr, '0', sizeof(serv_addr));  
serv_addr.sin_family = AF_INET;  
serv_addr.sin_port = htons(PORT);  
if (inet_pton(AF_INET, "127.0.0.1",  
             &serv_addr.sin_addr)  
    <= 0) {  
    printf("\nAddress not supported \n");  
    return -1;  
}  
if (connect(sock, (struct sockaddr*)&serv_addr,  
           sizeof(serv_addr))  
    < 0) {  
    printf("\nConnection Failed \n");  
    return -1;  
}  
int l = strlen(str);  
send(sock, str, sizeof(str), 0);  
valread = read(sock, str, l);  
printf("%s\n", str);  
return 0;  
}
```

SMARTBOY



```
manik@manik: ~  
manik@manik:~$ gedit ass4_TCP_server.c  
manik@manik:~$ gcc ass4_TCP_server.c -o ass4_TCP_server  
manik@manik:~$ ./ass4_TCP_server  
  
String sent by client:Manik  
  
Modified string sent to client  
manik@manik:~$  
  
manik@manik:~$ ./ass4_TCP_client  
  
Input the string:Manik  
  
Connection Failed  
manik@manik:~$ gedit ass4_TCP_client.c  
manik@manik:~$ gcc ass4_TCP_client.c -o ass4_TCP_client  
manik@manik:~$ ./ass4_TCP_client  
  
Input the string:Manik  
kinaM  
manik@manik:~$
```

2. Design the same string reversal client-server application as mentioned above except that here you will be using UDP socket API for implementing the client-server application.

Server Side:

SMARTBOY

```
#include <sys/socket.h>
```

```
#include <netinet/in.h>
```

```
#include <arpa/inet.h>
```

```
#include <netdb.h>
```

```
#include <string.h>
```

```
#include <stdlib.h>
```

```
#include <unistd.h>
```

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

```
#define S_PORT 43454
```

```
#define C_PORT 43455
```

```
#define ERROR -1
```

```
#define IP_STR "127.0.0.1"
```

```
void strrev(char *str, int len) {
```

```
    int i, j;
```

```
    char temp;
```

```
    for (i = 0, j = len - 1; i < j; ++i, --j) {
```

```
        temp = str[i];
```

```
        str[i] = str[j];
```

```
        str[j] = temp;
```

```
    }
```

```
}
```

```
int main(int argc, char const *argv[]) {
```

```
    int sfd, len;
```

```
    char *str_buf;
```

SMARTBOY

```
struct sockaddr_in servaddr, clientaddr;

sfd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);

if (sfd == ERROR) {
    perror("Could not open a socket");
    return 1;
}

memset((char *) &servaddr, 0, sizeof(servaddr));

servaddr.sin_family=AF_INET;
servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
servaddr.sin_port=htons(S_PORT);


memset((char *) &clientaddr, 0, sizeof(clientaddr));

clientaddr.sin_family=AF_INET;
clientaddr.sin_addr.s_addr=inet_addr(IP_STR);
clientaddr.sin_port=htons(C_PORT);


if((bind(sfd,(struct sockaddr *)&servaddr,sizeof(servaddr)))!=0) {
    perror("Could not bind socket");
    return 2;
}


printf("Server is running on %s:%d\n", IP_STR, S_PORT);

while(1) {
    recvfrom(sfd, &len, sizeof(len), 0, (struct sockaddr *)&clientaddr,
(socklen_t *)&clientaddr);

    str_buf = (char *) malloc(len*sizeof(char));
```

SMARTBOY

```
        recvfrom(sfd, str_buf, len, 0, (struct sockaddr *)&clientaddr, (socklen_t
*)&clientaddr);

        printf("Client at %s:%d said: %s\t", inet_ntoa(clientaddr.sin_addr),
ntohs(clientaddr.sin_port), str_buf);

        strrev(str_buf, len);

        sendto(sfd, &len, sizeof(len), 0, (struct sockaddr *)&clientaddr,
sizeof(clientaddr));

        sendto(sfd, str_buf, len, 0, (struct sockaddr *)&clientaddr,
sizeof(clientaddr));

        printf("The reverse is: %s\n", str_buf);

        free(str_buf);
    }

    return 0;
}
```

Client Side:

```
#include <sys/socket.h>

#include <netdb.h>

#include <string.h>

#include <stdlib.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <stdio.h>

#include <string.h>

#define S_PORT 43454
```

SMARTBOY

```
#define C_PORT 43455
```

```
#define ERROR -1
```

```
#define IP_STR "127.0.0.1"
```

```
int main(int argc, char const *argv[]) {  
    int sfd, len;  
    char str_buf[2048];  
    struct sockaddr_in servaddr, clientaddr;  
    socklen_t addrlen;  
    sfd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);  
    if (sfd == ERROR) {  
        perror("Could not open a socket");  
        return 1;  
    }  
    memset((char *) &servaddr, 0, sizeof(servaddr));  
    servaddr.sin_family=AF_INET;  
    servaddr.sin_addr.s_addr=inet_addr(IP_STR);  
    servaddr.sin_port=htons(S_PORT);  
  
    memset((char *) &clientaddr, 0, sizeof(clientaddr));  
    clientaddr.sin_family=AF_INET;  
    clientaddr.sin_addr.s_addr=inet_addr(IP_STR);  
    clientaddr.sin_port=htons(C_PORT);  
  
    if((bind(sfd,(struct sockaddr *)&clientaddr,sizeof(clientaddr)))!=0) {  
        perror("Could not bind socket");  
        return 2;
```


SMARTBOY

```
}

printf("Client is running on %s:%d\n", IP_STR, C_PORT);
printf("Enter a string: ");
scanf("%[^\n]*c",str_buf);
len = strlen(str_buf);
sendto(sfd, &len, sizeof(len), 0, (struct sockaddr *)&servaddr, sizeof(servaddr));
sendto(sfd, str_buf, len, 0, (struct sockaddr *)&servaddr, sizeof(servaddr));
addrlen = sizeof(clientaddr);
recvfrom(sfd, &len, sizeof(len), 0, (struct sockaddr *)&clientaddr, &addrlen);
recvfrom(sfd, str_buf, len, 0, (struct sockaddr *)&clientaddr, &addrlen);
printf("Server Replied: %s\n", str_buf);

return 0;
}
```

SMARTBOY

```
manik@manik:~$ gedit ass4_UDP_server.c
manik@manik:~$ gcc ass4_UDP_server.c -o ass4_UDP_server
manik@manik:~$ ./ass4_UDP_server
Server is running on 127.0.0.1:43454
client at 127.0.0.1:43455 said: 14anOk The reverse is: kznani
```

```
h g edit as s 4 GDP c lident . c
fi g edit as s 4 GDP client . c
5 gcc ass4_UDP_client.c -o ass4_UDPclient
a . ass4_UDPclient
Enter a string: 14anTk
be ve Re 1 Ted kTnañ
```

Assignment:5

1. TCP socket creation and data transmission in C for client server communication for string manipulations

(a) String sorting

(b) String comparison

(c) Sting copy

Server Side:

```
#include <unistd.h>
```

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

```
#include <sys/socket.h>
```

```
#include <stdlib.h>
```

```
#include <netinet/in.h>
```

```
#include <string.h>
```

```
#define PORT 8080
```

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

```
{
```

```
    int serFileDescriptor, newSocket, returnValue;
```

```
    struct sockaddr_in address;
```

```
    int addrlen = sizeof(address);
```

```
    char buffer[1024] = {0};
```

```
    char r[1000];
```

```
    int opt = 1;
```

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

```
    //Here 0 parameter in socket System call defines the Protocol Type
```

```
    if(serFileDescriptor == 0)
```

```
    {
```

```
        perror("socket failed");
```

SMARTBOY

```
    exit(EXIT_FAILURE);
}
// Forcefully attaching socket to the port 8080
if (setsockopt(serFileDescriptor, SOL_SOCKET, SO_REUSEADDR |
SO_REUSEPORT, &opt, sizeof(opt)))
{
    perror("setsockopt");
    exit(EXIT_FAILURE);
}
address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons( PORT ); //host to network byte ordering - little ending / big
ending
//The htons function takes a 16-bit number in host byte order and returns a
//16-bit number in network byte order used in TCP/IP networks (the AF_INET or
AF_INET6 address family).

int temp = bind(serFileDescriptor, (struct sockaddr *)&address, sizeof(address));
if(temp < 0)
{
    perror("bind failed");
    exit(EXIT_FAILURE);
}
if (listen(serFileDescriptor, 3) < 0)
//Here 3 defines the Queue length of server waiting for Clients
{
    perror("listen");
    exit(EXIT_FAILURE);
}
```

SMARTBOY

```
if ((newSocket = accept(serverFileDescriptor, (struct sockaddr *)&address,
(socklen_t*)&addrlen))<0)
{
    perror("accept");
    exit(EXIT_FAILURE);
}
char inputString[1024];
char selected[1024];
bzero(inputString, sizeof(inputString));
returnValue = read(newSocket, inputString, 1024);
printf("Client's Message -");
printf("%s", inputString);

char operation[] = {"Specify the Operation -\n 1: Sorting\n 2: Comparison\n 3: Copy\n
4: Insertion\n 5: Deletion"};
send(newSocket, operation, strlen(operation), 0);
printf("Message Sent Successfully\n");

bzero(selected, sizeof(selected));
returnValue = read(newSocket, selected, 1024);
printf("Client's Message -");
printf("%s", selected);

//Sorting Operation
if (strncmp("1", selected, 1) == 0)
{
    printf("Operation Selected - Sorting\n");
    char temp;
    int i, j;
```

SMARTBOY

```
int n = strlen(inputString);
for (i = 0; i < n-1; i++) {
    for (j = i+1; j < n; j++) {
        if (inputString[i] > inputString[j]) {
            temp = inputString[i];
            inputString[i] = inputString[j];
            inputString[j] = temp;
        }
    }
}

send(newSocket, inputString, strlen(inputString), 0);
printf("Message Sent to Client after Sorting Operation- %s\n", inputString);
} // Comparison Operation
else if (strcmp("2", selected, 1) == 0)
{
    printf("Operation Selected - Comparison\n");
    int value;
    char stringComp[1024];
    bzero(stringComp, sizeof(stringComp));
    returnValue = read( newSocket, stringComp, 1024);
    printf("Second String -");
    printf("%s", stringComp);
    value = strcmp(inputString, stringComp);
    if (value == 0)
    {
        printf("Strings are Same\n");
        char result[] = {"Both Strings are Same"};
        send(newSocket, result, strlen(result), 0);
    }
}
```


SMARTBOY

```
    printf("Message Sent to Client after Comparison Operation %s\n -", result);
}
else
{
    printf("Strings are different\n");
    char result[] = {"Both Strings are Different"};
    send(newSocket, result, strlen(result), 0);
    printf("Message Sent to Client after Comparison Operation %s\n -", result);
}
} //Copy Operation
else if(strncmp("3", selected, 1) == 0)
{
    printf("Operation Selected - Copy\n");
    char *copiedString;
    copiedString = (char*)malloc(100);
    strcpy(copiedString, inputString);
    printf("String Copied \n");
    char result[] = {"String Copied"};
    send(newSocket, result, strlen(result), 0);
    printf("Copied String is: %s\n", copiedString);
} //Exit if No Operation Selected
else
{
    printf("Connection Terminated\n");
}
close(newSocket);
close(serverFileDescriptor);
return 0; }
```

Client Side:

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

```
#include <sys/socket.h>
```

```
#include <arpa/inet.h>
```

```
#include <unistd.h>
```

```
#include <string.h>
```

```
#define PORT 8080
```

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

```
{
```

```
    int socketR = 0, returnValue;
```

```
    struct sockaddr_in serv_addr;
```

```
    char buffer[1024] = {0};
```

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

```
    if(socketR < 0)
```

```
    {
```

```
        printf("\n Socket creation error \n");
```

```
        return -1;
```

```
    }
```

```
    serv_addr.sin_family = AF_INET;
```

```
    serv_addr.sin_port = htons(PORT);
```

```
//The inet_pton() function converts an Internet address in its standard text
```

SMARTBOY

//format into its numeric binary form. The argument af specifies the family of the address.

```
if(inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr)<=0)
{
    printf("\nInvalid address/ Address not supported \n");
    return -1;
}
```

```
if (connect(socketR, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
{
    printf("\nConnection Failed \n");
    return -1;
}
```

```
char buff[1024];
int num=0;
bzero(buff, sizeof(buff));
printf("Enter the String : ");
while ((buff[num++] = getchar()) != '\n');
send(socketR , buff , strlen(buff) , 0);
printf("Message sent to Server Succesfully - \n");
// printf("%s", buff);
```

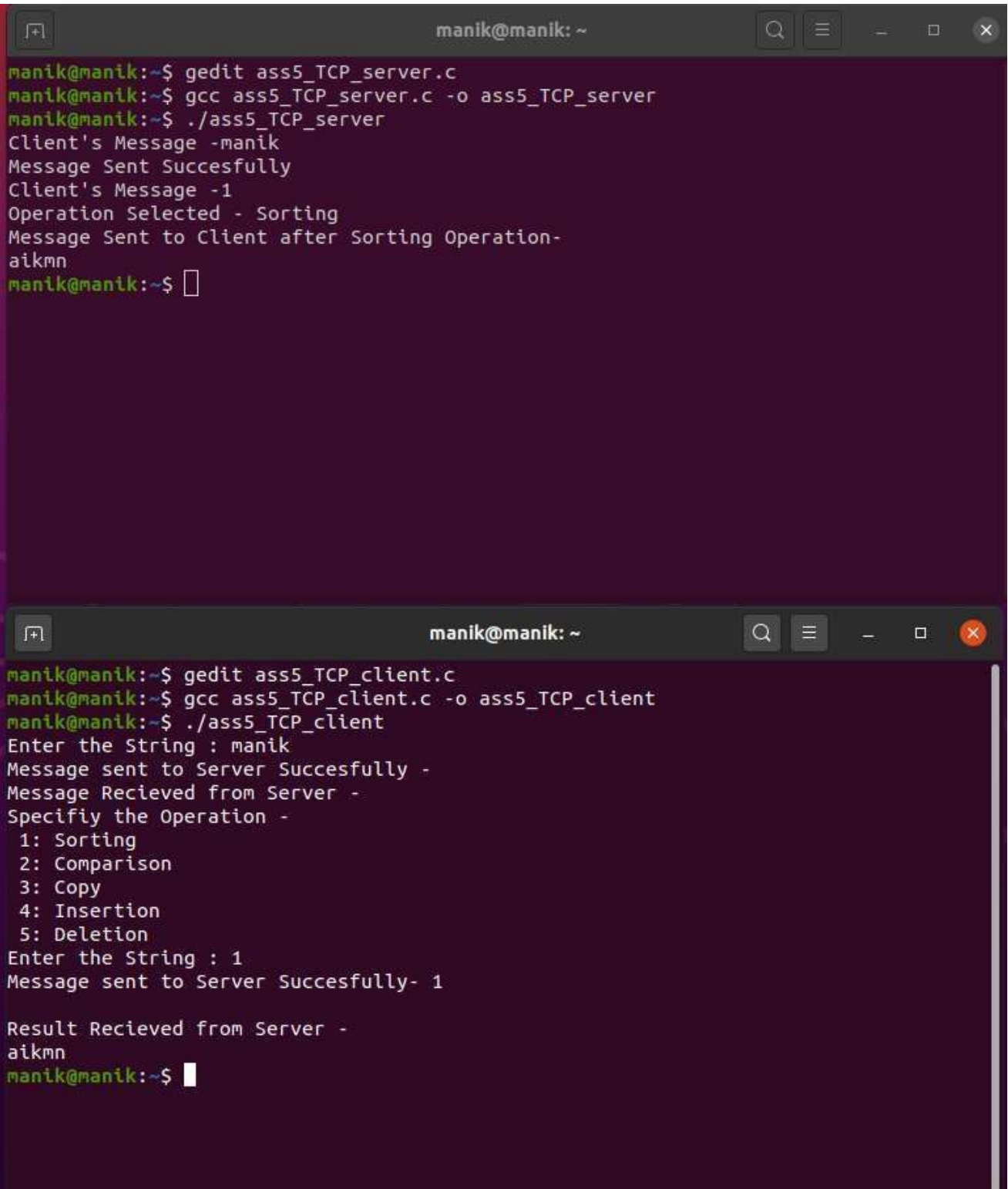
```
returnValue = read( socketR , buffer, 1024);
printf("Message Recieved from Server - \n");
printf("%s\n",buffer );
num = 0;
```

SMARTBOY

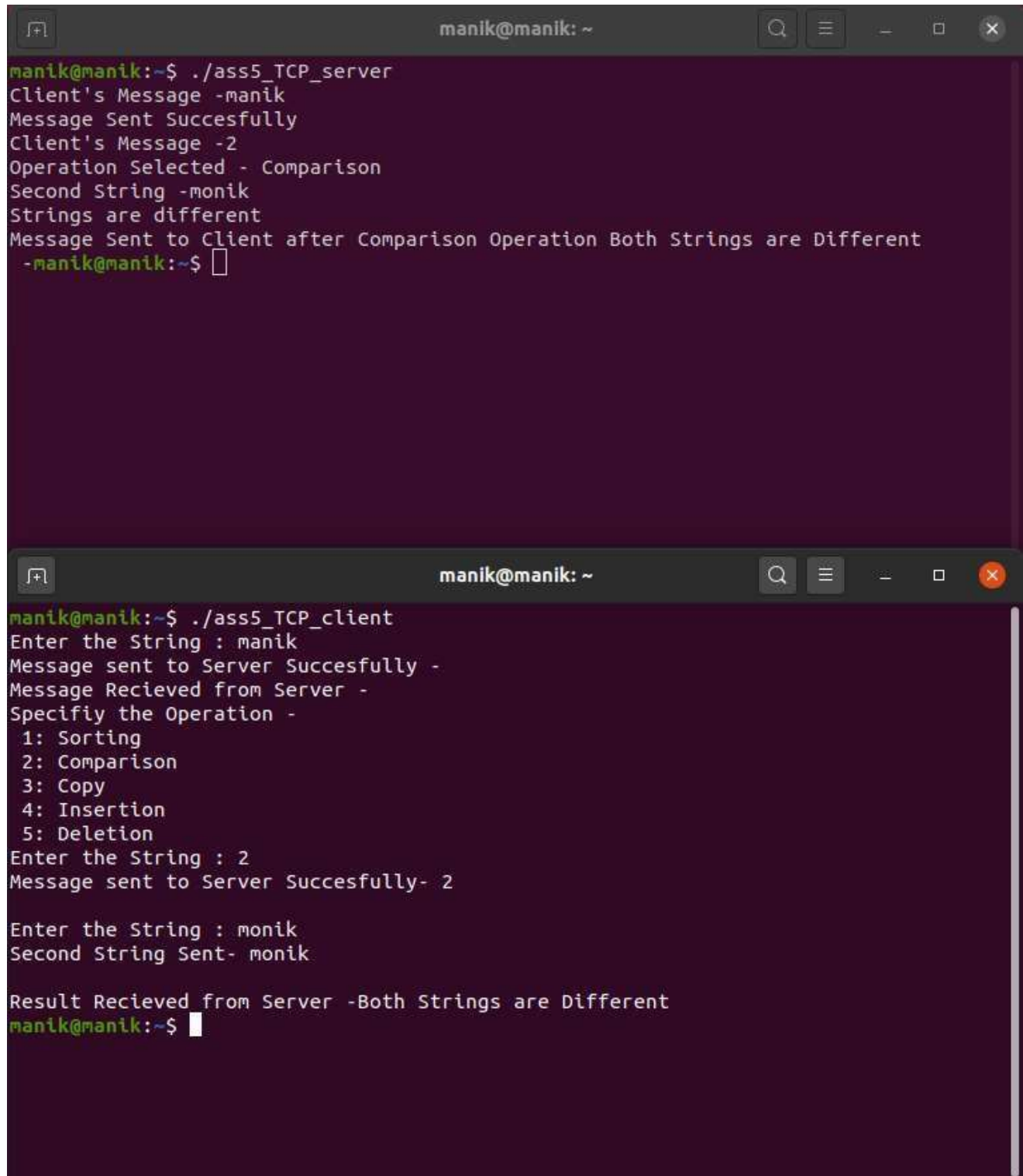
```
char buff1[1024];
bzero(buff1, sizeof(buff1));
printf("Enter the String : ");
while ((buff1[num++] = getchar()) != '\n');
send(socketR , buff1 , strlen(buff1) , 0);
printf("Message sent to Server Succesfully- %s\n", buff1);
// printf("%s", buff1);

if(strncmp("2", buff1, 1)==0)
{
    int i = 0;
    char secondString[1024];
    bzero(secondString, sizeof(secondString));
    printf("Enter the String : ");
    while ((secondString[i++] = getchar()) != '\n');
    send(socketR , secondString , strlen(secondString) , 0);
    printf("Second String Sent- %s\n", secondString);
}
char buff2[1024];
bzero(buff2, sizeof(buff2));
returnValue = read( socketR , buff2, 1024);
printf("Result Recieved from Server -");
printf("%s\n",buff2 );
close(socketR);
return 0;
}
```

(i) Sorting operation

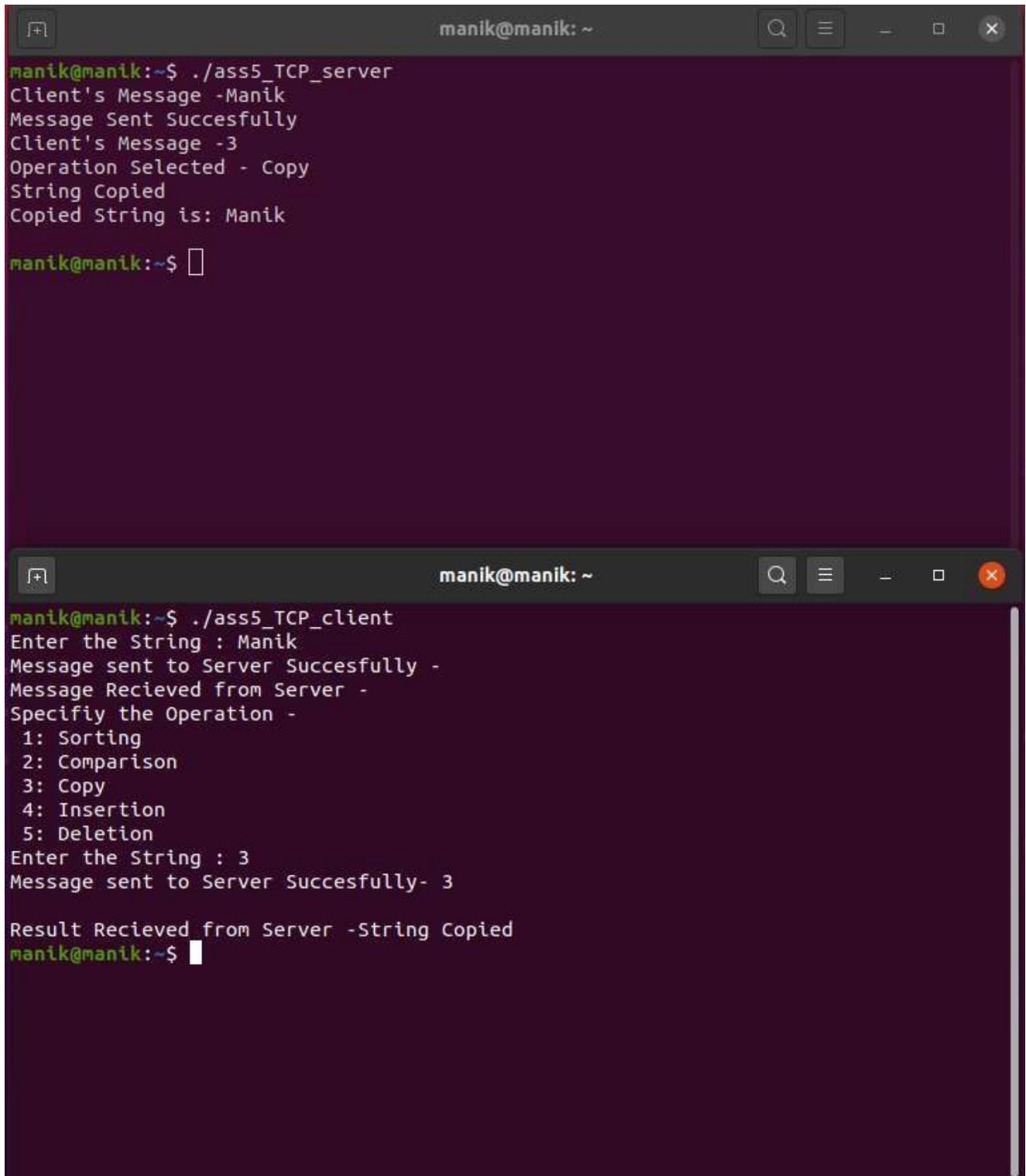


```
manik@manik: ~  
manik@manik:~$ gedit ass5_TCP_server.c  
manik@manik:~$ gcc ass5_TCP_server.c -o ass5_TCP_server  
manik@manik:~$ ./ass5_TCP_server  
Client's Message -manik  
Message Sent Succesfully  
Client's Message -1  
Operation Selected - Sorting  
Message Sent to Client after Sorting Operation-  
aikmn  
manik@manik:~$  
  
manik@manik:~$ gedit ass5_TCP_client.c  
manik@manik:~$ gcc ass5_TCP_client.c -o ass5_TCP_client  
manik@manik:~$ ./ass5_TCP_client  
Enter the String : manik  
Message sent to Server Succesfully -  
Message Recieved from Server -  
Specify the Operation -  
1: Sorting  
2: Comparison  
3: Copy  
4: Insertion  
5: Deletion  
Enter the String : 1  
Message sent to Server Succesfully- 1  
  
Result Recieved from Server -  
aikmn  
manik@manik:~$
```

(ii) Comparison

```
manik@manik: ~  
manik@manik:~$ ./ass5_TCP_server  
Client's Message -manik  
Message Sent Succesfully  
Client's Message -2  
Operation Selected - Comparison  
Second String -monik  
Strings are different  
Message Sent to Client after Comparison Operation Both Strings are Different  
-manik@manik:~$  
  
manik@manik: ~  
manik@manik:~$ ./ass5_TCP_client  
Enter the String : manik  
Message sent to Server Succesfully -  
Message Recieved from Server -  
Specifiy the Operation -  
1: Sorting  
2: Comparison  
3: Copy  
4: Insertion  
5: Deletion  
Enter the String : 2  
Message sent to Server Succesfully- 2  
  
Enter the String : monik  
Second String Sent- monik  
  
Result Recieved from Server -Both Strings are Different  
manik@manik:~$
```

(iii) Copy operation



```
manik@manik: ~  
manik@manik:~$ ./ass5_TCP_server  
Client's Message -Manik  
Message Sent Succesfully  
Client's Message -3  
Operation Selected - Copy  
String Copied  
Copied String is: Manik  
manik@manik:~$  
  
manik@manik:~$ ./ass5_TCP_client  
Enter the String : Manik  
Message sent to Server Succesfully -  
Message Recieved from Server -  
Specify the Operation -  
1: Sorting  
2: Comparison  
3: Copy  
4: Insertion  
5: Deletion  
Enter the String : 3  
Message sent to Server Succesfully- 3  
Result Recieved from Server -String Copied  
manik@manik:~$
```

Assignment: 6

- 1. WAP for the implementation of handling multiple clients on single server using TCP socket programming in connection-oriented Scenario. Use select() to handle multiple clients (without using fork()).**

Server Side:

```
#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <errno.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <sys/time.h>

#define TRUE 1

#define FALSE 0

#define PORT 8888


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

{

    int opt = TRUE;

    int master_socket , addrlen , new_socket , client_socket[30] ,

        max_clients = 30 , activity, i , valread , sd;

    int max_sd;

    struct sockaddr_in address;
```


SMARTBOY

```
char buffer[1025];

fd_set readfds;

//a message
char *message = "ECHO Daemon v1.0 \r\n";

for (i = 0; i < max_clients; i++)
{
    client_socket[i] = 0;
}

if( (master_socket = socket(AF_INET , SOCK_STREAM , 0)) == 0)
{
    perror("socket failed");
    exit(EXIT_FAILURE);
}

if( setsockopt(master_socket, SOL_SOCKET, SO_REUSEADDR, (char *)&opt,
    sizeof(opt)) < 0 )
{
    perror("setsockopt");
    exit(EXIT_FAILURE);
}

address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons( PORT );

if (bind(master_socket, (struct sockaddr *)&address, sizeof(address))<0)
{
    perror("bind failed");
    exit(EXIT_FAILURE);
}
```

SMARTBOY

```
}

printf("Listener on port %d \n", PORT);
if (listen(master_socket, 3) < 0)
{
    perror("listen");
    exit(EXIT_FAILURE);
}

addrlen = sizeof(address);
puts("Waiting for connections ...");

while(TRUE)
{
    FD_ZERO(&readfds);
    FD_SET(master_socket, &readfds);
    max_sd = master_socket;
    for ( i = 0 ; i < max_clients ; i++)
    {
        //socket descriptor
        sd = client_socket[i];
        if(sd > 0)
            FD_SET( sd , &readfds);
        if(sd > max_sd)
            max_sd = sd;
    }
    activity = select( max_sd + 1 , &readfds , NULL , NULL , NULL);

    if ((activity < 0) && (errno!=EINTR))
```

SMARTBOY

```
{
    printf("select error");
}

if (FD_ISSET(master_socket, &readfds))
{
    if ((new_socket = accept(master_socket,
        (struct sockaddr *)&address, (socklen_t *)&addrlen)) < 0)
    {
        perror("accept");
        exit(EXIT_FAILURE);
    }

    printf("New connection , socket fd is %d , ip is : %s , port : %d \n" , new_socket ,
inet_ntoa(address.sin_addr) , ntohs
        (address.sin_port));

    if( send(new_socket, message, strlen(message), 0) != strlen(message) )
    {
        perror("send");
    }

    puts("Welcome message sent successfully");

    for (i = 0; i < max_clients; i++)
    {
        if( client_socket[i] == 0 )
        {
            client_socket[i] = new_socket;
```

SMARTBOY

```
printf("Adding to list of sockets as %d\n" , i);

    break;
}
}
}
for (i = 0; i < max_clients; i++)
{
    sd = client_socket[i];

    if (FD_ISSET( sd , &readfds))
    {
        if ((valread = read( sd , buffer, 1024)) == 0)
        {
            getpeername(sd , (struct sockaddr*)&address , \
                (socklen_t*)&addrlen);
            printf("Host disconnected , ip %s , port %d \n" ,
                inet_ntoa(address.sin_addr) , ntohs(address.sin_port));
            close( sd );
            client_socket[i] = 0;
        }
        else
        {
            buffer[valread] = '\0';
            send(sd , buffer , strlen(buffer) , 0 );
        }
    }
}
```

```

    }
}

```

```

    return 0;
}

```

Client Side:

```

#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>

#define MAX 80
#define PORT 8888
#define SA struct sockaddr

void func(int sockfd)
{
    char buff[MAX];
    int n;
    for (;;) {
        bzero(buff, sizeof(buff));
        printf("Enter the string : ");
        n = 0;
        while ((buff[n++] = getchar()) != '\n')
            ;
        write(sockfd, buff, sizeof(buff));
        bzero(buff, sizeof(buff));
        read(sockfd, buff, sizeof(buff));
    }
}

```

SMARTBOY

```
printf("From Server : %s", buff);
if ((strncmp(buff, "exit", 4)) == 0) {
    printf("Client Exit...\n");
    break;
}
}
}

int main()
{
    int sockfd, connfd;
    struct sockaddr_in servaddr, cli;
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        printf("socket creation failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully created..\n");
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
    servaddr.sin_port = htons(PORT);
    if (connect(sockfd, (SA*)&servaddr, sizeof(servaddr)) != 0) {
        printf("connection with the server failed...\n");
        exit(0);
    }
```

SMARTBOY

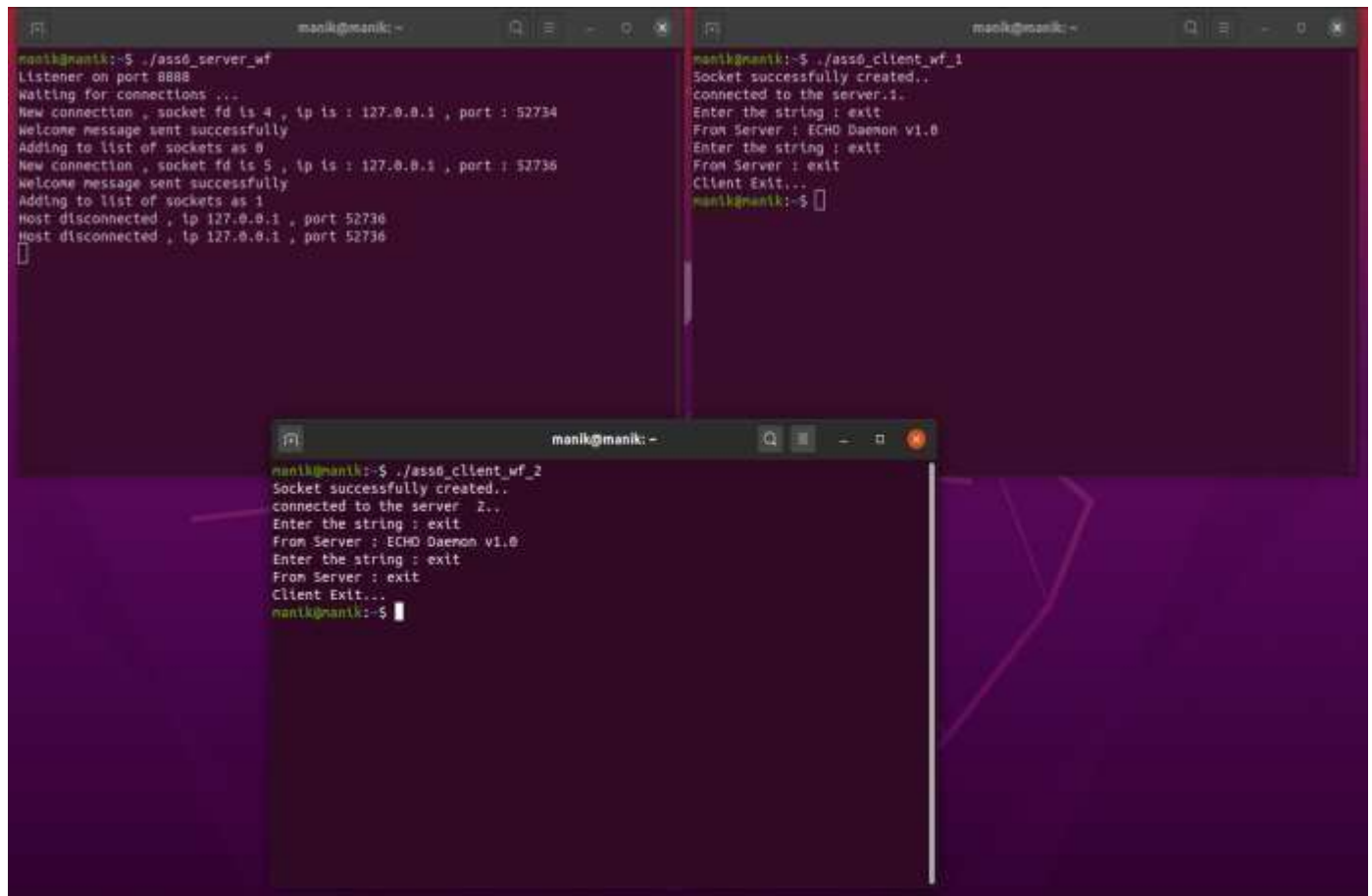
else

printf("connected to the server.1.\n");

func(sockfd);

close(sockfd);

}



The screenshot shows three terminal windows on a Linux system. The top-left window runs the server program `./ass6_server_wf`, which listens on port 8888 and handles two concurrent connections. The top-right window runs the client program `./ass6_client_wf_1`, which connects to the server and sends the string "exit". The bottom window runs the client program `./ass6_client_wf_2`, which also connects to the server and sends the string "exit".

```
manik@manik:~$ ./ass6_server_wf
Listener on port 8888
Waiting for connections ...
New connection , socket fd is 4 , ip is : 127.0.0.1 , port : 52734
Welcome message sent successfully
Adding to list of sockets as 0
New connection , socket fd is 5 , ip is : 127.0.0.1 , port : 52736
Welcome message sent successfully
Adding to list of sockets as 1
Host disconnected , ip 127.0.0.1 , port 52736
Host disconnected , ip 127.0.0.1 , port 52736
[]

manik@manik:~$ ./ass6_client_wf_1
Socket successfully created..
connected to the server.1.
Enter the string : exit
From Server : ECHO Daemon v1.0
Enter the string : exit
From Server : exit
Client Exit...
manik@manik:~$

manik@manik:~$ ./ass6_client_wf_2
Socket successfully created..
connected to the server 2..
Enter the string : exit
From Server : ECHO Daemon v1.0
Enter the string : exit
From Server : exit
Client Exit...
manik@manik:~$
```

2. Design a concurrent server for handling multiple clients using fork(). [without using select()]

Server Side:

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

```
#include <stdlib.h>
```

```
#include <string.h>
```

```
#include <unistd.h>
```

```
#include <sys/socket.h>
```

SMARTBOY

```
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#define PORT 4444
int main(){
    int sockfd, ret;
    struct sockaddr_in serverAddr;
    int newSocket;
    struct sockaddr_in newAddr;
    socklen_t addr_size;
    char buffer[1024];
    pid_t childpid;
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if(sockfd < 0){
        printf("[-]Error in connection.\n");
        exit(1);
    }
    printf("[+]Server Socket is created.\n");
    memset(&serverAddr, '\0', sizeof(serverAddr));
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(PORT);
    serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");

    ret = bind(sockfd, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
    if(ret < 0){
        printf("[-]Error in binding.\n");
        exit(1);
    }
}
```


SMARTBOY

```
}

printf("[+]Bind to port %d\n", 4444);

if(listen(sockfd, 10) == 0){
    printf("[+]Listening... \n");
}else{
    printf("[-]Error in binding.\n");
}

while(1){
    newSocket = accept(sockfd, (struct sockaddr*)&newAddr, &addr_size);
    if(newSocket < 0){
        exit(1);
    }

    printf("Connection          accepted          from          %s:%d\n",
inet_ntoa(newAddr.sin_addr), ntohs(newAddr.sin_port));

    if((childpid = fork()) == 0){
        close(sockfd);
        while(1){
            recv(newSocket, buffer, 1024, 0);
            if(strcmp(buffer, ":exit") == 0){
                printf("Disconnected          from          %s:%d\n",
inet_ntoa(newAddr.sin_addr), ntohs(newAddr.sin_port));
                break;
            }else{
                printf("Client: %s\n", buffer);
                send(newSocket, buffer, strlen(buffer), 0);
                bzero(buffer, sizeof(buffer));
            }
        }
    }
}
```

SMARTBOY

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

Client Side:

```
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <unistd.h>  
#include <sys/socket.h>  
#include <sys/types.h>  
#include <netinet/in.h>  
#include <arpa/inet.h>  
#define PORT 4444  
int main(){  
    int clientSocket, ret;  
    struct sockaddr_in serverAddr;  
    char buffer[1024];  
    clientSocket = socket(AF_INET, SOCK_STREAM, 0);  
    if(clientSocket < 0){  
        printf("[-]Error in connection.\n");  
        exit(1);  
    }  
    printf("[+]Client Socket is created.\n");  
    memset(&serverAddr, '\0', sizeof(serverAddr));  
    serverAddr.sin_family = AF_INET;  
    serverAddr.sin_port = htons(PORT);  
    serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
```

SMARTBOY

```
ret = connect(clientSocket, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
if(ret < 0){
    printf("[-]Error in connection.\n");
    exit(1);
}
printf("[+]Connected to Server.\n");
while(1){
    printf("Client: \t");
    scanf("%s", &buffer[0]);
    send(clientSocket, buffer, strlen(buffer), 0);
    if(strcmp(buffer, ":exit") == 0){
        close(clientSocket);
        printf("[-]Disconnected from server.\n");
        exit(1);
    }
    if(recv(clientSocket, buffer, 1024, 0) < 0){
        printf("[-]Error in receiving data.\n");
    }else{
        printf("Server: \t%s\n", buffer);
    }
}
return 0;
}
```

SMARTBOY

```
manik@manik: ~  
manik@manik:~$ gedit ass6_server_f.c  
manik@manik:~$ gcc ass6_server_f.c -o ass6_server_f  
manik@manik:~$ ./ass6_server_f  
[+]Server Socket is created.  
[+]Bind to port 4444  
[+]Listening....  
Connection accepted from 127.0.0.1:36180  
Client: hello  
Client: 1  
Connection accepted from 127.0.0.1:36182  
Client: hello  
Client: 2  
Connection accepted from 127.0.0.1:36184  
Client: hello  
Client: 3  
Disconnected from 127.0.0.1:36180  
Disconnected from 127.0.0.1:36182  
Disconnected from 127.0.0.1:36184  
[]  
  
manik@manik:~$ gedit ass6_client_f_1.c  
manik@manik:~$ gcc ass6_client_f_1.c -o ass6_client_1  
manik@manik:~$ ./ass6_client_f_1  
[+]Client Socket is created.  
[+]Connected to Server.  
Client:      hello 1  
Server:      hello  
Client:      Server:      1  
Client:      :exit  
[-]Disconnected from server.  
manik@manik:~$ []  
  
manik@manik:~$ gcc ass6_client_f_1.c -o ass6_client_f_1  
manik@manik:~$ ./ass6_client_f_1  
[+]Client Socket is created.  
[+]Connected to Server.  
Client:      hello 2  
Server:      hello  
Client:      Server:      2  
Client:      :exit  
[-]Disconnected from server.  
manik@manik:~$ []  
  
manik@manik:~$ gcc ass6_client_f_1.c -o ass6_client_f_1  
manik@manik:~$ ./ass6_client_f_1  
[+]Client Socket is created.  
[+]Connected to Server.  
Client:      hello 3  
Server:      hello  
Client:      Server:      3  
Client:      :exit  
[-]Disconnected from server.  
manik@manik:~$
```

Assignment: 7**1. Design sending and receiving processes to implement multicasting using socket programming.****Send Multicast:**

```
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
struct in_addr    localInterface;
struct sockaddr_in groupSock;
int              sd;
int              datalen;
//char           databuf[1024];

int main (int argc, char *argv[])
{
    sd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sd < 0) {
        perror("opening datagram socket");
        exit(1);
    }
    char databuf[] = "This is a Multicast Message";
    memset((char *) &groupSock, 0, sizeof(groupSock));
    groupSock.sin_family = AF_INET;
    groupSock.sin_addr.s_addr = inet_addr("225.1.1.1");
    groupSock.sin_port = htons(5555);
    {
        char loopch=0;

        if (setsockopt(sd, IPPROTO_IP, IP_MULTICAST_LOOP,
                       (char *)&loopch, sizeof(loopch)) < 0) {
            perror("setting IP_MULTICAST_LOOP:");
            close(sd);
            exit(1);
        }
    }
}
```

SMARTBOY

```
} localInterface.s_addr = inet_addr("192.168.43.172");
if (setsockopt(sd, IPPROTO_IP, IP_MULTICAST_IF,
              (char *)&localInterface,
              sizeof(localInterface)) < 0) {
    perror("setting local interface");
    exit(1);
}
datalen = 27;
if (sendto(sd, databuf, datalen, 0,
           (struct sockaddr*)&groupSock,
           sizeof(groupSock)) < 0)
{
    perror("sending datagram message");
}
}
```

Receive Multicast:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
struct sockaddr_in  localSock;
struct ip_mreq      group;
int                 sd;
int                 datalen;
char                databuf[1024];

int main (int argc, char *argv[])
{
    sd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sd < 0) {
        perror("opening datagram socket");
        exit(1);
    }
    {
        int reuse=1;

        if (setsockopt(sd, SOL_SOCKET, SO_REUSEADDR,
                      (char *)&reuse, sizeof(reuse)) < 0) {
```

SMARTBOY

```
perror("setting SO_REUSEADDR");
close(sd);
exit(1);
}
}
memset((char *) &localSock, 0, sizeof(localSock));
localSock.sin_family = AF_INET;
localSock.sin_port = htons(5555);
localSock.sin_addr.s_addr = INADDR_ANY;

if (bind(sd, (struct sockaddr*)&localSock, sizeof(localSock))) {
    perror("binding datagram socket");
    close(sd);
    exit(1);
}
group.imr_multiaddr.s_addr = inet_addr("225.1.1.1");
group.imr_interface.s_addr = inet_addr("192.168.43.172");
if (setsockopt(sd, IPPROTO_IP, IP_ADD_MEMBERSHIP,
               (char *)&group, sizeof(group)) < 0) {
    perror("adding multicast group");
    close(sd);
    exit(1);
}
datalen = sizeof(databuf);
if (read(sd, databuf, datalen) < 0) {
    perror("reading datagram message");
    close(sd);
    exit(1);
}
else
    printf("%s",databuf);
}
```

2. Design sending and receiving processes to implement broadcasting using connectionless socket programming.

Send Broadcast:

```
#include<stdio.h>
#include<netinet/in.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netdb.h>
#include<string.h>
#include<stdlib.h>
#define MAX 80
#define PORT 43454
#define SA struct sockaddr

int main()
{
    int sockfd;
    struct sockaddr_in servaddr, cli;
    char buff[] = "This is a broadcast Message";
    sockfd=socket(AF_INET,SOCK_DGRAM,0);
    int broadcastPermission = 1;
    setsockopt(sockfd, SOL_SOCKET, SO_BROADCAST, (void *) &broadcastPermission,
    sizeof(broadcastPermission));
    if(sockfd==-1)
    {
        printf("socket creation failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully created..\n");
    bzero(&servaddr,sizeof(servaddr));
    servaddr.sin_family=AF_INET;
    servaddr.sin_addr.s_addr=inet_addr("192.168.43.255");
    servaddr.sin_port=htons(PORT);
    printf("\n Sending Broadcast");
    sendto(sockfd,buff,sizeof(buff),0,(SA *)&servaddr,sizeof(servaddr));
    printf("\n..... sent");
    close(sockfd);
}
```


Receive Broadcast:

```

#include<sys/socket.h>
#include<netdb.h>
#include<string.h>
#include<stdlib.h>
#include<stdio.h>
#define MAX 80
#define PORT 43454
#define SA struct sockaddr

int main()
{
    char buff[MAX];
    int sockfd,len,n;
    struct sockaddr_in servaddr;
    sockfd=socket(AF_INET,SOCK_DGRAM,0);
    if(sockfd== -1)
    {
        printf("socket creation failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully created..\n");
    bzero(&servaddr,sizeof(len));
    servaddr.sin_family=AF_INET;
    servaddr.sin_addr.s_addr=INADDR_ANY;//inet_addr("192.168.43.");
    servaddr.sin_port=htons(PORT);

    if((bind(sockfd,(SA *)&servaddr,sizeof(servaddr)))!=0)
    {
        printf("socket bind failed...\n");
        exit(0);
    }
    len=sizeof(servaddr);

    recvfrom(sockfd,buff,sizeof(buff),0,(SA *)&servaddr,&len);
    printf("Message Recieved : %s\n",buff);

    close(sockfd);
}

```

Assignment: 8

TCP Dump

Introduction to tcp dump:

Tcpdump is a data-network packet analyzer computer program that runs under a command line interface. It allows to capture and analyze network traffic going through your system.

Tcpdump requires a library known as **libpcap**, for network packet capture. It will be automatically added as dependency.

Installation on Linux:

Tcpdump is included with several Linux distributions, so chances are, you already have it installed. Check whether tcpdump is installed on your system with the following command:

```
manik@manik:~$ which tcpdump
/usr/sbin/tcpdump
manik@manik:~$
```

If tcpdump is not installed then use the following command:

```
$ sudo dnf install -y tcpdump
```

Capturing Packets with tcpdump:

1. To see which interfaces are available for capture, use the any one of the following command:

```
manik@manik:~$ sudo tcpdump -D
[sudo] password for manik:
1.enp0s3 [Up, Running]
2.lo [Up, Running, Loopback]
3.any (Pseudo-device that captures on all interfaces) [Up, Running]
4.bluetooth-monitor (Bluetooth Linux Monitor) [none]
5.nflog (Linux netfilter log (NFLOG) interface) [none]
6.nfqueue (Linux netfilter queue (NFQUEUE) interface) [none]
manik@manik:~$ sudo tcpdump --list-interfaces
1.enp0s3 [Up, Running]
2.lo [Up, Running, Loopback]
3.any (Pseudo-device that captures on all interfaces) [Up, Running]
4.bluetooth-monitor (Bluetooth Linux Monitor) [none]
5.nflog (Linux netfilter log (NFLOG) interface) [none]
6.nfqueue (Linux netfilter queue (NFQUEUE) interface) [none]
manik@manik:~$
```

SMARTBOY

2. any allows capturing in any active interface. Tcpcap continues to capture packets until it receives an interrupt signal. You can interrupt capturing by pressing Ctrl+C.

```
manik@manik:~$ sudo tcpdump --interface any
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked v1), capture size 262144 bytes
12:01:14.791673 IP 172.16.73.30.bootpc > 255.255.255.255.bootps: BOOTP/DHCP, Request from 00:21:6a:5b:56:e6 (oui Unknown), length 300
12:01:14.792532 IP localhost.36828 > localhost.domain: 9617+ [1au] PTR? 255.255.255.255.in-addr.arpa. (57)
12:01:14.792964 IP localhost.domain > localhost.36828: 9617 ServFail 0/0/1 (57)
12:01:14.793186 IP localhost.55188 > localhost.domain: 9617+ [1au] PTR? 255.255.255.255.in-addr.arpa. (57)
12:01:14.794778 IP localhost.60593 > localhost.domain: 3623+ [1au] PTR? 53.0.0.127.in-addr.arpa. (52)
12:01:14.796294 ARP, Request who-has 192.168.29.120 tell 172.16.70.90, length 46
12:01:14.796325 ARP, Request who-has 172.16.93.166 tell 172.16.64.1, length 46
12:01:14.796329 ARP, Request who-has 172.16.80.177 tell 172.16.64.1, length 46
12:01:14.796332 ARP, Request who-has 172.16.75.77 tell 172.16.64.1, length 46
12:01:14.796630 IP localhost.43577 > localhost.domain: 7590+ [1au] PTR? 120.29.168.192.in-addr.arpa. (56)
12:01:14.797002 IP localhost.domain > localhost.43577: 7590 ServFail 0/0/1 (56)
12:01:14.894500 IP 172.16.76.170.netbios-ns > 172.16.95.255.netbios-ns: UDP, length 50
12:01:14.894739 IP localhost.58947 > localhost.domain: 21076+ [1au] PTR? 255.95.16.172.in-addr.arpa. (55)
12:01:14.895003 IP localhost.domain > localhost.58947: 21076 ServFail 0/0/1 (55)
12:01:14.895227 IP localhost.45154 > localhost.domain: 21076+ [1au] PTR? 255.95.16.172.in-addr.arpa. (55)
12:01:14.898636 IP 172.16.75.249.netbios-ns > 172.16.95.255.netbios-ns: UDP, length 50
12:01:14.898661 ARP, Request who-has 172.16.69.180 tell 172.16.81.153, length 46
12:01:14.898943 IP localhost.37009 > localhost.domain: 14485+ [1au] PTR? 249.75.16.172.in-addr.arpa. (55)
12:01:14.899217 IP localhost.domain > localhost.37009: 14485 ServFail 0/0/1 (55)
12:01:14.899376 IP localhost.54251 > localhost.domain: 14485+ [1au] PTR? 249.75.16.172.in-addr.arpa. (55)
```

3. To limit the number of packets captured and stop tcpdump, use the -c (for count) option:

```
manik@manik:~$ sudo tcpdump -i any -c 7
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked v1), capture size 262144 bytes
12:07:58.552601 ARP, Request who-has 192.168.43.218 tell 172.16.70.90, length 46
12:07:58.557533 ARP, Request who-has 172.16.64.152 tell 172.16.64.1, length 46
12:07:58.557550 ARP, Request who-has 172.16.72.186 tell 172.16.64.1, length 46
12:07:58.557554 ARP, Request who-has 192.168.43.218 tell 172.16.75.193, length 46
12:07:58.557558 ARP, Request who-has 172.16.93.12 tell 172.16.64.1, length 46
12:07:58.557562 ARP, Request who-has 172.16.73.49 tell 172.16.64.1, length 46
12:07:58.557567 ARP, Request who-has 172.16.70.21 tell 172.16.64.1, length 46
7 packets captured
95 packets received by filter
81 packets dropped by kernel
manik@manik:~$
```

4. When troubleshooting network issues, it is often easier to use the IP addresses and port numbers; disable name resolution by using the option -n
5. In the above case we can disable port resolution with -nn:

SMARTBOY

```
manik@manik:~$ sudo tcpdump -i any -c7 -nn
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked v1), capture size 262144 bytes
12:10:24.853177 IP 172.16.76.103.54915 > 172.16.95.255.54915: UDP, length 263
12:10:24.854976 ARP, Request who-has 172.16.75.82 tell 172.16.83.74, length 46
12:10:24.854993 ARP, Request who-has 172.16.80.186 tell 172.16.80.253, length 46
12:10:24.854997 IP 0.0.0.0.68 > 255.255.255.255.67: BOOTP/DHCP, Request from 78:0c:b8:09:56:14, length 300
12:10:24.855006 IP 0.0.0.0.68 > 255.255.255.255.67: BOOTP/DHCP, Request from 78:0c:b8:09:56:14, length 328
12:10:24.855009 IP 0.0.0.0.68 > 255.255.255.255.67: BOOTP/DHCP, Request from 78:0c:b8:09:56:14, length 300
12:10:24.855013 IP 0.0.0.0.68 > 255.255.255.255.67: BOOTP/DHCP, Request from 78:0c:b8:09:56:14, length 328
7 packets captured
7 packets received by filter
0 packets dropped by kernel
manik@manik:~$ sudo tcpdump -i any -c7 -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked v1), capture size 262144 bytes
12:10:55.075994 ARP, Request who-has 172.16.79.46 (ff:ff:ff:ff:ff:ff) tell 172.16.79.46, length 46
12:10:55.078978 ARP, Request who-has 172.16.82.201 tell 172.16.83.57, length 46
12:10:55.079049 ARP, Request who-has 172.16.79.12 tell 172.16.70.90, length 46
12:10:55.079054 ARP, Request who-has 172.16.64.1 tell 172.16.79.46, length 46
12:10:55.079058 ARP, Request who-has 172.16.84.19 tell 172.16.64.1, length 46
12:10:55.178919 ARP, Request who-has 172.16.94.104 tell 172.16.64.1, length 46
12:10:55.180976 IP 0.0.0.0.68 > 255.255.255.255.67: BOOTP/DHCP, Request from a6:ae:b8:da:65:41, length 304
7 packets captured
12 packets received by filter
0 packets dropped by kernel
manik@manik:~$
```

As shown above, (-nn) the capture output now displays the IP addresses and port numbers. This also prevents tcpdump from issuing DNS lookups, which helps to lower network traffic while troubleshooting network issues.

6. One of tcpdump's most powerful features is its ability to filter the captured packets using a variety of parameters, such as source and destination IP addresses, ports, protocols, etc.

Some of them are not related to the issue which we are troubleshooting. So to filter the packets based on protocol specifying the protocol in the command line, as shown in following screenshot:

```
manik@manik:~$ sudo tcpdump -i any -c5 icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked v1), capture size 262144 bytes
```

7. Print the tcpdump and libpcap version strings and exit.

```
manik@manik:~$ sudo tcpdump --version
tcpdump version 4.9.3
libpcap version 1.9.1 (with TPACKET_V3)
OpenSSL 1.1.1f 31 Mar 2020
manik@manik:~$
```

SMARTBOY

- 8. Host:** Limit capture to only packets related to specific host by using the filter known as host.

```
$sudo tcpdump -i any -c 7 -nn host 54.204.39.18
```

- 9. Port:** To filter packets based on the desired service or port, use the port filter.

```
$sudo tcpdump -i any -c 7 -nn port 50
```

- 10. Source IP/hostname:** You can also filter packets based on the source or destination IP Address or hostname.

```
$sudo tcpdump -i any -c 7 -nn src 192.168.122.98
```