INDEX SHEET

PARTICULARS OF EXPERIMENTS PERFORMED

S.No	Name of the Experiment	Page No.	Date of Expt	Date of Sub-mission	Remarks/ Sig-nature
1	Understanding and using of commands like ifconfig, netstat, ping, arp, telnet, ftp, finger, traceroute, whoisetc.				
2	Implementation of Connec-tion oriented concurrent ser-vice (TCP).				
3	Implementation of Connec-tionless Iterative time service (UDP).				
4	Implementation of Select sys-tem call.				
5	Implementation of gesockopt (), setsockopt () system calls.				
6	Implementation of getpeer-name () system call.				
7	Implementation of remote command execution using socket system calls.				
8	Implementation of Distance Vector Routing Algorithm.				
9	Implementation of FTP.				

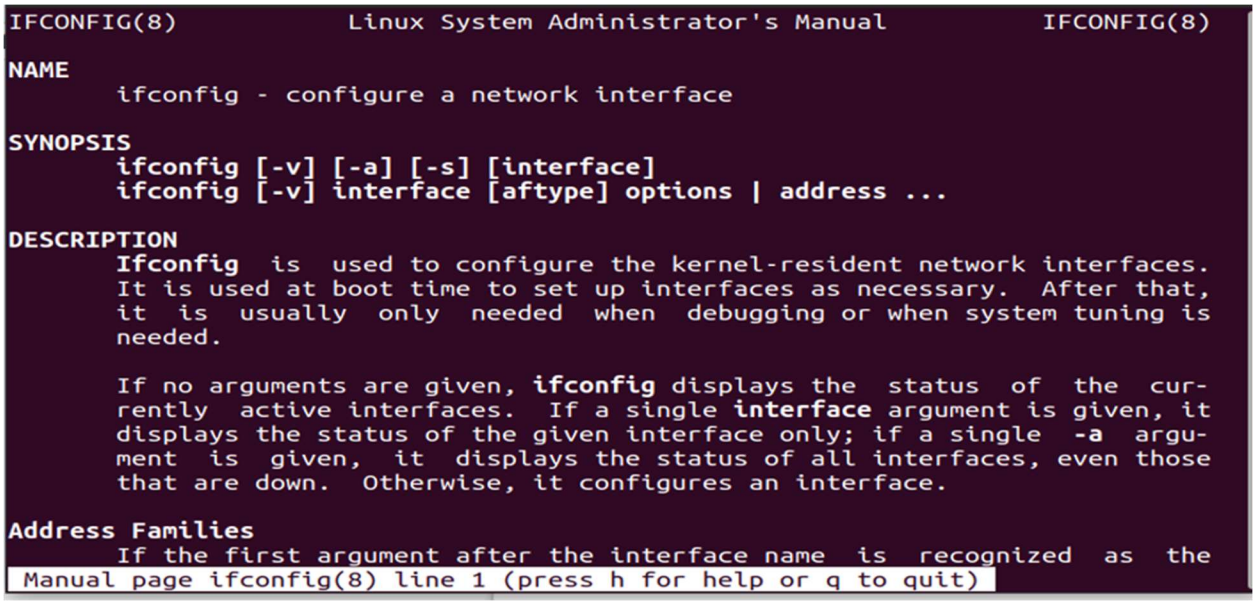
EXPERIMENT - 1

Aim:

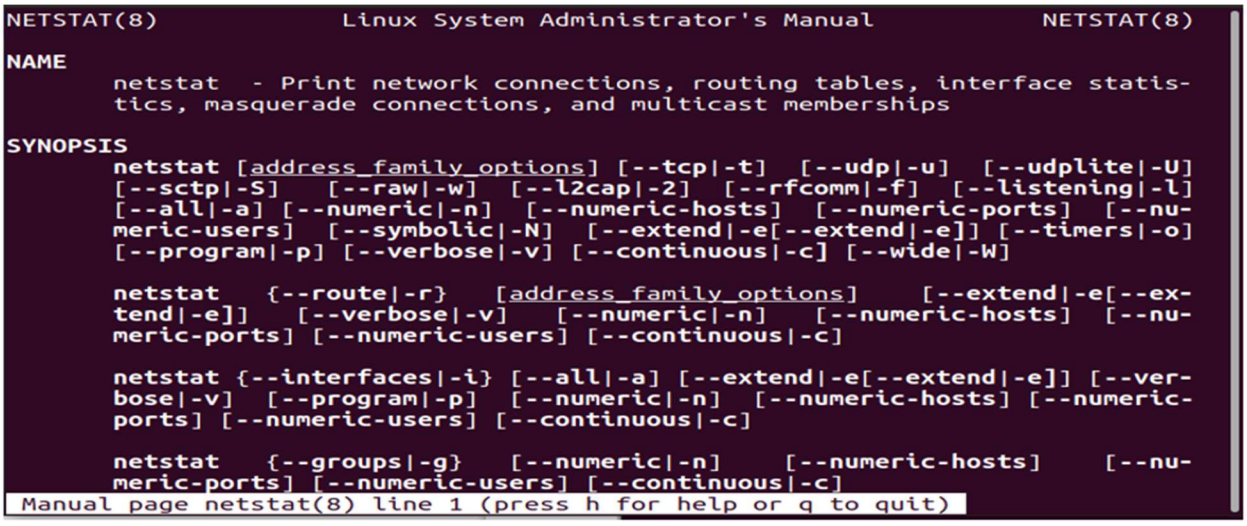
Understanding and using of commands like ifconfig, netstat, ping, arp, telnet, ftp, finger, traceroute, whois

Here, The man pages were taken screenshots and placed here

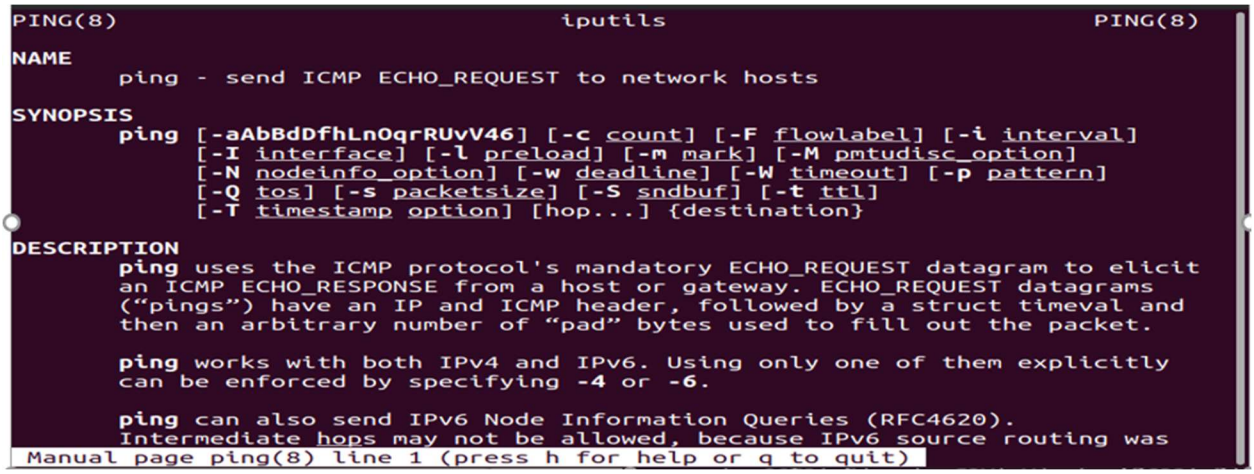
Ifconfig:



Netstat:

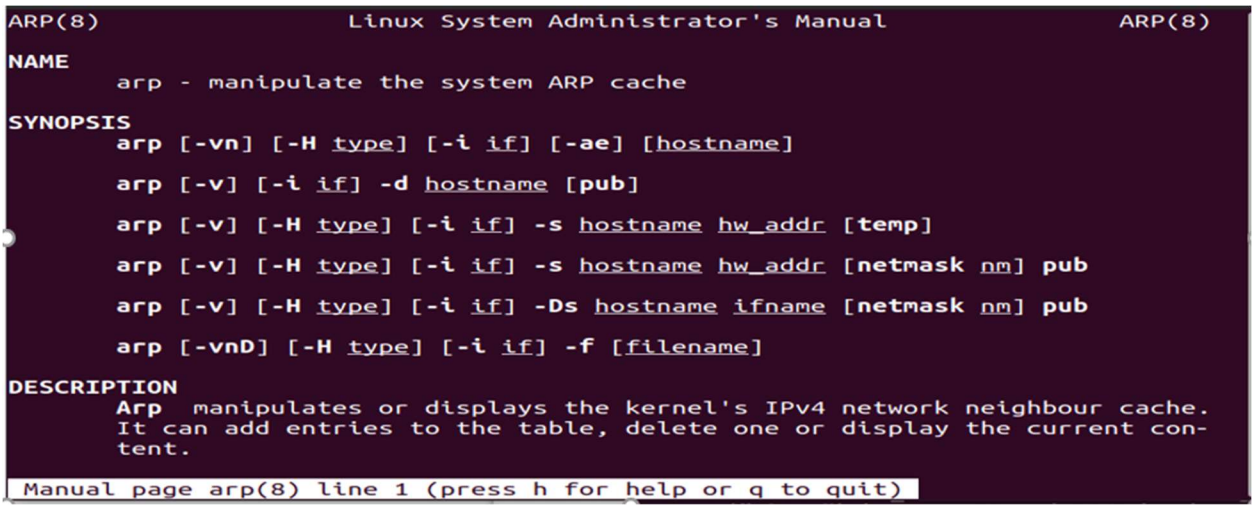


ping:



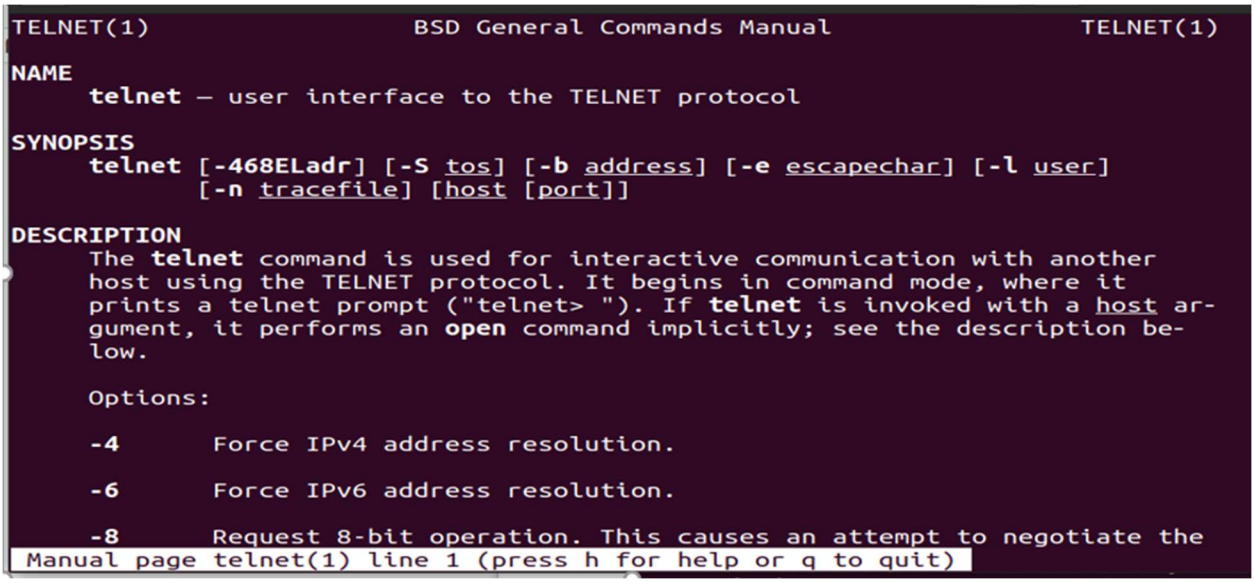
```
PING(8)                                 iputils                                 PING(8)

NAME
       ping - send ICMP ECHO_REQUEST to network hosts

SYNOPSIS
       ping [-aAbBdDfhLnOqrRUvV46] [-c count] [-F flowlabel] [-i interval]
            [-I interface] [-l preload] [-m mark] [-M pmtudisc_option]
            [-N nodeinfo_option] [-w deadline] [-W timeout] [-p pattern]
            [-Q tos] [-s packetsize] [-S sndbuf] [-t ttl]
            [-T timestamp option] [hop...] {destination}

DESCRIPTION
       ping uses the ICMP protocol's mandatory ECHO_REQUEST datagram to elicit
       an ICMP ECHO_RESPONSE from a host or gateway. ECHO_REQUEST datagrams
       ("pings") have an IP and ICMP header, followed by a struct timeval and
       then an arbitrary number of "pad" bytes used to fill out the packet.

       ping works with both IPv4 and IPv6. Using only one of them explicitly
       can be enforced by specifying -4 or -6.

       ping can also send IPv6 Node Information Queries (RFC4620).
       Intermediate hops may not be allowed, because IPv6 source routing was
 Manual page ping(8) line 1 (press h for help or q to quit)
```

arp:



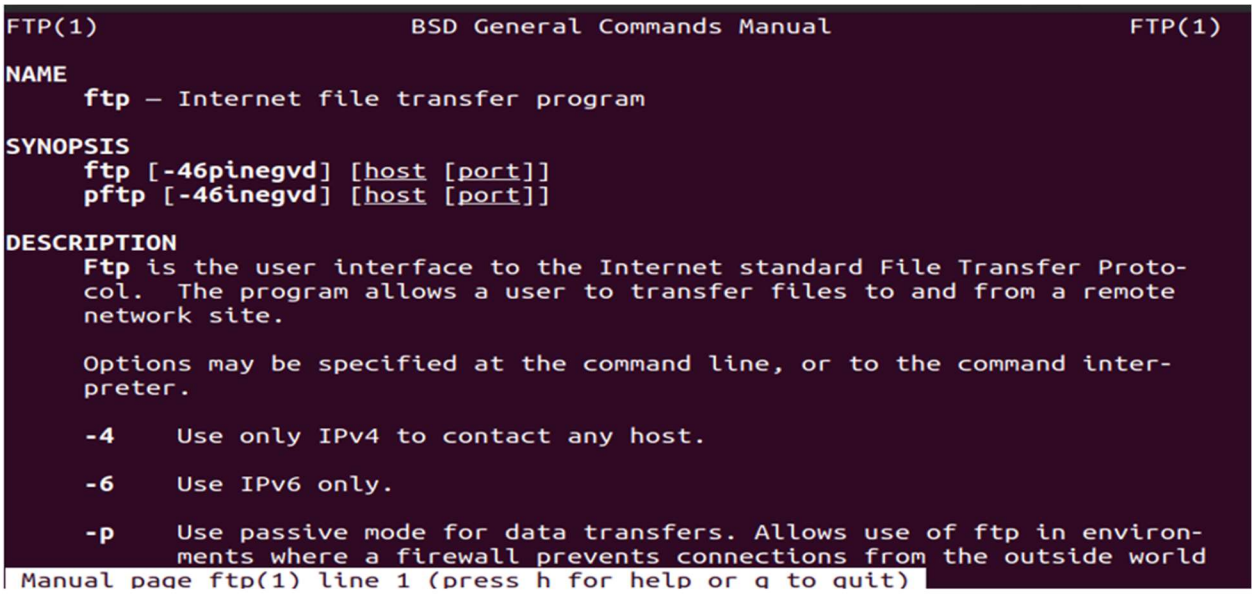
```
ARP(8)                  Linux System Administrator's Manual                 ARP(8)

NAME
       arp - manipulate the system ARP cache

SYNOPSIS
       arp [-vn] [-H type] [-i if] [-ae] [hostname]

       arp [-v] [-i if] -d hostname [pub]

       arp [-v] [-H type] [-i if] -s hostname hw_addr [temp]

       arp [-v] [-H type] [-i if] -s hostname hw_addr [netmask nm] pub

       arp [-v] [-H type] [-i if] -Ds hostname ifname [netmask nm] pub

       arp [-vnD] [-H type] [-i if] -f [filename]

DESCRIPTION
       Arp  manipulates or displays the kernel's IPv4 network neighbour cache.
       It can add entries to the table, delete one or display the current con-
       tent.

 Manual page arp(8) line 1 (press h for help or q to quit)
```

telnet:



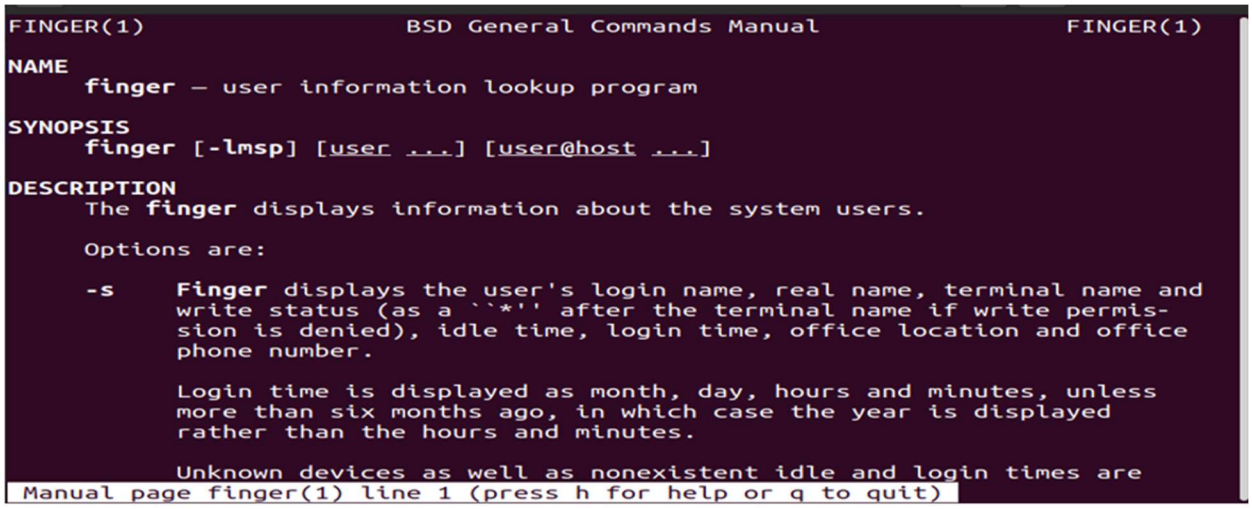
```
TELNET(1)                  BSD General Commands Manual                  TELNET(1)

NAME
     telnet – user interface to the TELNET protocol

SYNOPSIS
     telnet [-468ELadr] [-S tos] [-b address] [-e escapechar] [-l user]
            [-n tracefile] [host [port]]

DESCRIPTION
     The telnet command is used for interactive communication with another
     host using the TELNET protocol. It begins in command mode, where it
     prints a telnet prompt ("telnet> "). If telnet is invoked with a host ar-
     gument, it performs an open command implicitly; see the description be-
     low.

     Options:

     -4      Force IPv4 address resolution.

     -6      Force IPv6 address resolution.

     -8      Request 8-bit operation. This causes an attempt to negotiate the
 Manual page telnet(1) line 1 (press h for help or q to quit)
```

ftp:



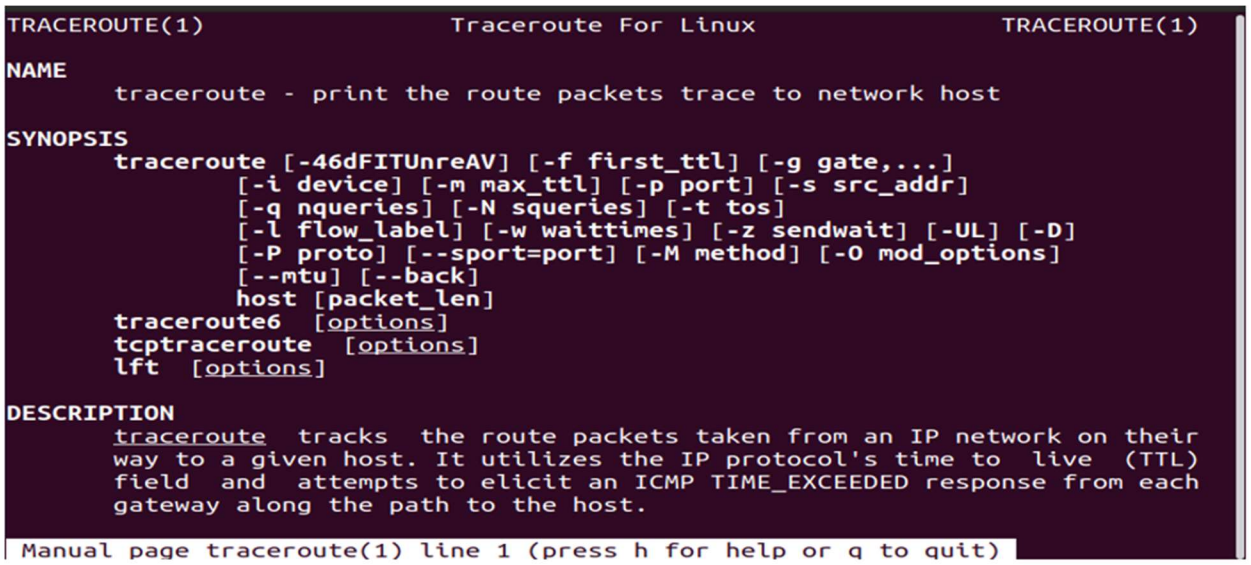
```
FTP(1)                     BSD General Commands Manual                     FTP(1)

NAME
     ftp – Internet file transfer program

SYNOPSIS
     ftp [-46pinegvd] [host [port]]
     pftp [-46inegvd] [host [port]]

DESCRIPTION
     Ftp is the user interface to the Internet standard File Transfer Proto-
     col.  The program allows a user to transfer files to and from a remote
     network site.

     Options may be specified at the command line, or to the command inter-
     preter.

     -4    Use only IPv4 to contact any host.

     -6    Use IPv6 only.

     -p    Use passive mode for data transfers. Allows use of ftp in environ-
           ments where a firewall prevents connections from the outside world
 Manual page ftp(1) line 1 (press h for help or q to quit)
```

finger:



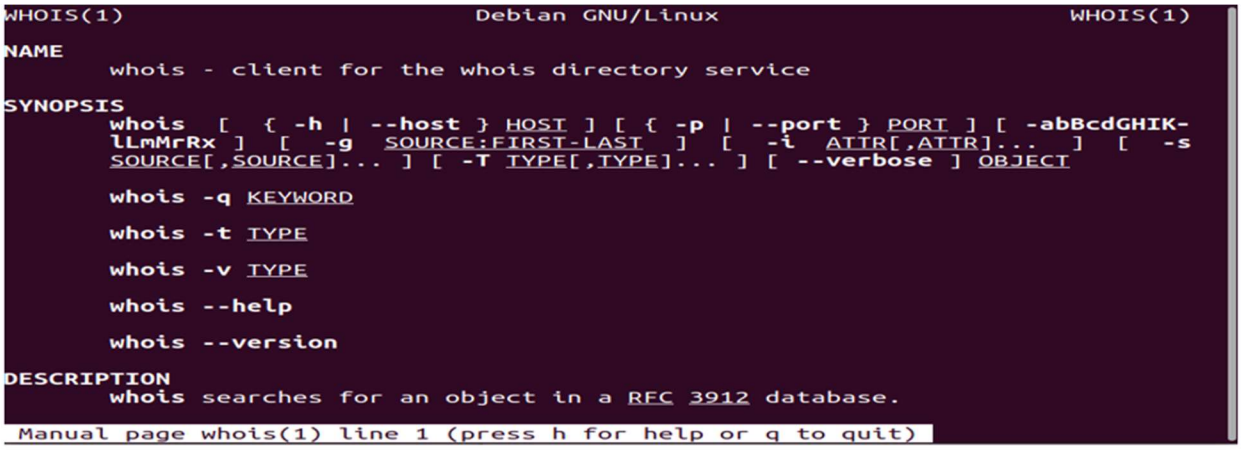
```
FINGER(1)                 BSD General Commands Manual                FINGER(1)

NAME
     finger – user information lookup program

SYNOPSIS
     finger [-lmsp] [user ...] [user@host ...]

DESCRIPTION
     The finger displays information about the system users.

     Options are:

     -s    Finger displays the user's login name, real name, terminal name and
           write status (as a ``*'' after the terminal name if write permis-
           sion is denied), idle time, login time, office location and office
           phone number.

           Login time is displayed as month, day, hours and minutes, unless
           more than six months ago, in which case the year is displayed
           rather than the hours and minutes.

           Unknown devices as well as nonexistent idle and login times are
 Manual page finger(1) line 1 (press h for help or q to quit)
```

traceroute:



```
TRACEROUTE(1)                 Traceroute For Linux                 TRACEROUTE(1)

NAME
       traceroute - print the route packets trace to network host

SYNOPSIS
       traceroute [-46dFITUnreAV] [-f first_ttl] [-g gate,...]
                  [-i device] [-m max_ttl] [-p port] [-s src_addr]
                  [-q nqueries] [-N squeries] [-t tos]
                  [-l flow_label] [-w waittimes] [-z sendwait] [-UL] [-D]
                  [-P proto] [--sport=port] [-M method] [-O mod_options]
                  [--mtu] [--back]
                  host [packet_len]
       traceroute6  [options]
       tcptraceroute  [options]
       lft  [options]

DESCRIPTION
       traceroute  tracks  the route packets taken from an IP network on their
       way to a given host. It utilizes the IP protocol's time to  live  (TTL)
       field  and  attempts to elicit an ICMP TIME_EXCEEDED response from each
       gateway along the path to the host.

 Manual page traceroute(1) line 1 (press h for help or q to quit)
```

whois:


WHOIS(1) Debian GNU/Linux WHOIS(1)
NAME
whois - client for the whois directory service
SYNOPSIS
whois [{ -h | --host } HOST] [{ -p | --port } PORT] [-abBcdGHIK-
lLmMrRx] [-g SOURCE:FIRST-LAST] [-i ATTR[,ATTR]...] [-s
SOURCE[,SOURCE]...] [-T TYPE[,TYPE]...] [--verbose] OBJECT
whois -q KEYWORD
whois -t TYPE
whois -v TYPE
whois --help
whois --version
DESCRIPTION
whois searches for an object in a RFC 3912 database.
Manual page whois(1) line 1 (press h for help or q to quit)

EXPERIMENT – 2

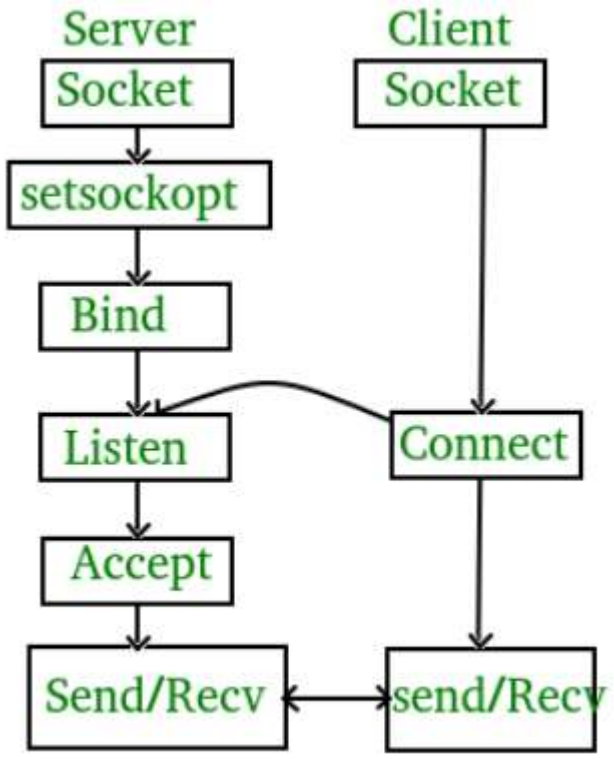
AIM:

Implementation of Connection oriented concurrent service (TCP).

DESCRIPTION:

If we are creating a connection between client and server using TCP then it has few functionality like, TCP is suited for applications that require high reliability, and transmission time is relatively less critical. It is used by other protocols like HTTP, HTTPs, FTP, SMTP, Telnet. TCP rearranges data packets in the order specified. There is absolute guarantee that the data transferred remains intact and arrives in the same order in which it was sent. TCP does Flow Control and requires three packets to set up a socket connection, before any user data can be sent. TCP handles reliability and congestion control. It also does error checking and error recovery. Erroneous packets are retransmitted from the source to the destination.

The entire process can be broken down into following steps:



PROGRAM :-

tcpconser.c

```
#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')
            ;

        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");
```

```
        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 acccept failed...\n");
        exit(0);
    }
    else
        printf("server acccept the client...\n");

    func(connfd);

    close(sockfd);
}
```

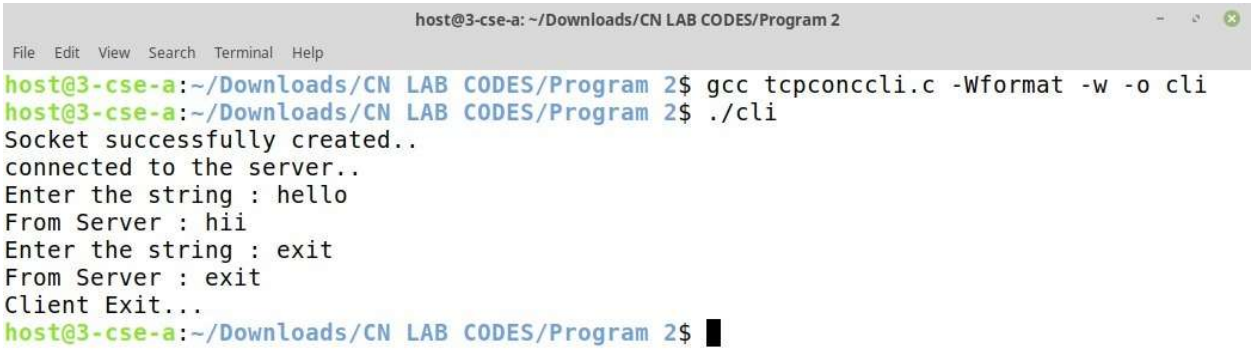
tcpconcli.c

```
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#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 ((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);
    }
    else
        printf("connected to the server..\n");

    func(sockfd);
```

```
    close(sockfd);
}
```

OUTPUT:



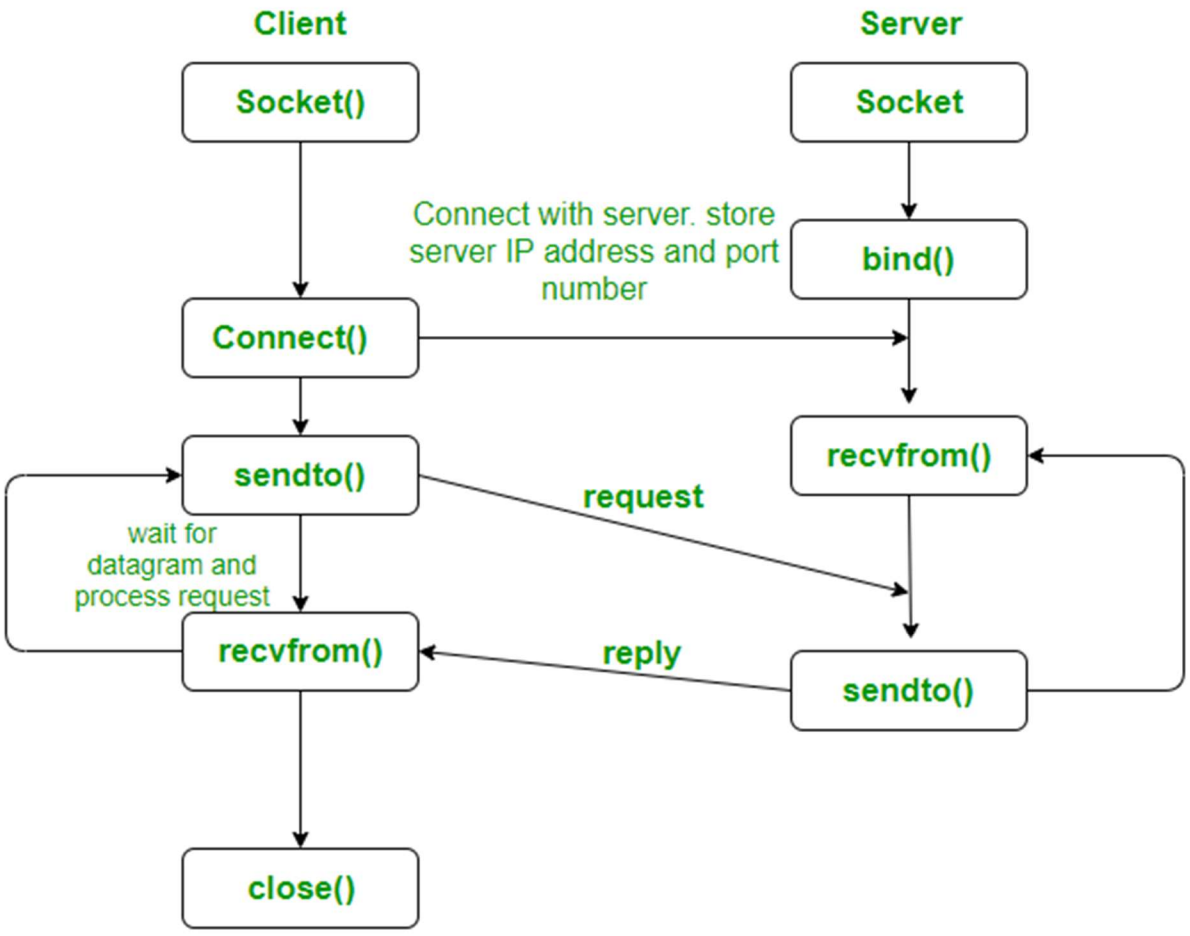
EXPERIMENT – 3

AIM:

Implementation of Connectionless Iterative time service (UDP).

DESCRIPTION:

In UDP, the client does not form a connection with the server like in TCP and instead just sends a datagram. Similarly, the server need not accept a connection and just waits for datagrams to arrive. Datagrams upon arrival contain the address of sender which the server uses to send data to the correct client.



The entire process can be broken down into following steps :

UDP Server :

1. Create UDP socket.
2. Bind the socket to server address.
3. Wait until datagram packet arrives from client.
4. Process the datagram packet and send a reply to client.
5. Go back to Step 3.

UDP Client :

1. Create UDP socket.
2. Send message to server.
3. Wait until response from server is recieved.
4. Process reply and go back to step 2, if necessary.
5. Close socket descriptor and exit.

PROGRAM:

udpiterser.c

```
// Server side implementation of UDP client-server model
#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
// Driver code
int main() {
    int sockfd;
    char buffer[MAXLINE];
    char *hello = "Hello from server";
```

```
struct sockaddr_in servaddr, cliaddr;

// Creating socket file descriptor
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));

// Filling server information
servaddr.sin_family    = AF_INET; // IPv4
servaddr.sin_addr.s_addr = INADDR_ANY;
servaddr.sin_port = htons(PORT);

// Bind the socket with the server address
if ( bind(sockfd, (const struct sockaddr *)&servaddr,
        sizeof(servaddr)) < 0 )
{
    perror("bind failed");
    exit(EXIT_FAILURE);
}

int len, n;

len = sizeof(cliaddr);  //len is value/resuslt
```

```
    n = recvfrom(sockfd, (char *)buffer, MAXLINE,
                 MSG_WAITALL, ( struct sockaddr *) &cliaddr,
                 &len);
    buffer[n] = '\0';
    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;
}
```

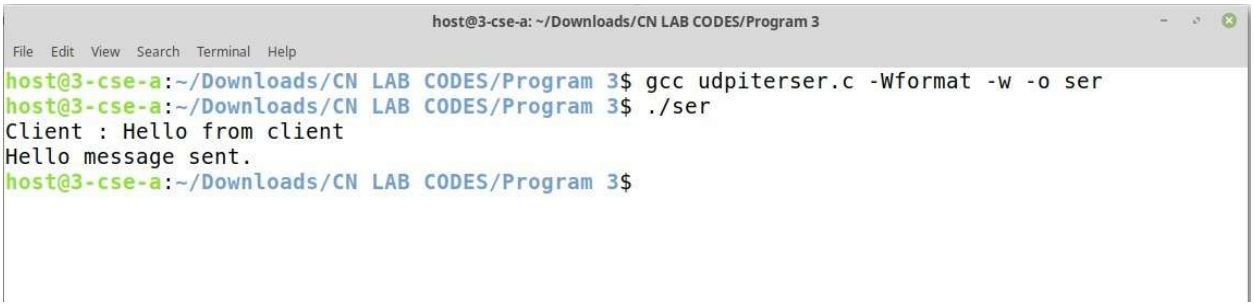
udpitercli.c

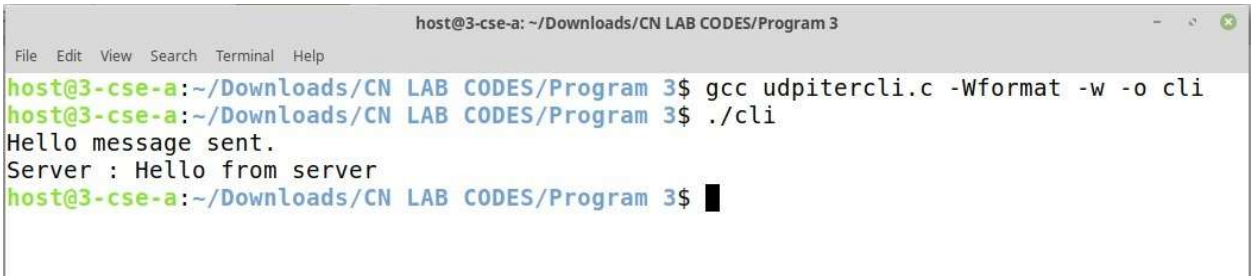
```
// Client side implementation of UDP client-server model
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#define PORT 8080
#define MAXLINE 1024
// Driver code
int main() {
    int sockfd;
    char buffer[MAXLINE];
```

```
char *hello = "Hello from client";
struct sockaddr_in     servaddr;

// Creating socket file descriptor
if ( (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ) {
    perror("socket creation failed");
    exit(EXIT_FAILURE);
}

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

// Filling server information
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);
buffer[n] = '\0';
printf("Server : %s\n", buffer);
```

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

OUTPUT:





EXPERIMENT – 4

AIM:

Implementation of Select system call.

DESCRIPTION:

Select function is used to select between TCP and UDP socket. This function gives instructions to the kernel to wait for any of the multiple events to occur and awakens the process only after one or more events occur or a specified time passes.

The entire process can be broken down into following steps :

Server:

1. Create TCP i.e Listening socket
2. Create a UDP socket
3. Bind both socket to server address.
4. Initialize a descriptor set for select and calculate maximum of 2 descriptor for which we will wait.
5. Call select and get the ready descriptor(TCP or UDP)
6. Handle new connection if ready descriptor is of TCP OR receive data gram if ready descriptor is of UDP

UDP Client:

1. Create UDP socket.
2. Send message to server.
3. Wait until response from server is recieved.
4. Close socket descriptor and exit.

TCP Client:

1. Create a TCP scoket.
2. Call connect to establish connection with server
3. When the connection is accepted write message to server
4. Read response of Server
5. Close socket descriptor and exit

PROGRAM:

selserver.c

```
// Server program
#include <arpa/inet.h>
#include <errno.h>
#include <netinet/in.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
#define PORT 5000
#define MAXLINE 1024
int max(int x, int y)
{
    if (x > y)
        return x;
    else
        return y;
}
int main()
{
    int listenfd, connfd, udpfd, nready, maxfdp1;
    char buffer[MAXLINE];
    pid_t childpid;
    fd_set rset;
```

```
ssize_t n;
socklen_t len;
const int on = 1;
struct sockaddr_in cliaddr, servaddr;
char* message = "Hello Client";
void sig_chld(int);

/* create listening TCP socket */
listenfd = socket(AF_INET, SOCK_STREAM, 0);
bzero(&servaddr, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(PORT);

// binding server addr structure to listenfd
bind(listenfd, (struct sockaddr*)&servaddr, sizeof(servaddr));
listen(listenfd, 10);

/* create UDP socket */
udpfd = socket(AF_INET, SOCK_DGRAM, 0);
// binding server addr structure to udp sockfd
bind(udpfd, (struct sockaddr*)&servaddr, sizeof(servaddr));

// clear the descriptor set
FD_ZERO(&rset);

// get maxfd
maxfdp1 = max(listenfd, udpfd) + 1;
```

```
    for (;;) {

        // set listenfd and udpfd in readset
        FD_SET(listenfd, &rset);
        FD_SET(udpfd, &rset);

        // select the ready descriptor
        nready = select(maxfdp1, &rset, NULL, NULL, NULL);

        // if tcp socket is readable then handle
        // it by accepting the connection
        if (FD_ISSET(listenfd, &rset)) {
            len = sizeof(cliaddr);
            connfd = accept(listenfd, (struct sockaddr*)&cliaddr,
&len);
            if ((childpid = fork()) == 0) {
                close(listenfd);
                bzero(buffer, sizeof(buffer));
                printf("Message From TCP client: ");
                read(connfd, buffer, sizeof(buffer));
                puts(buffer);
                write(connfd, (const char*)message, sizeof(buffer));
                close(connfd);
                exit(0);
            }
            close(connfd);
        }
        // if udp socket is readable receive the message.
```

```
        if (FD_ISSET(udpfd, &rset)) {
            len = sizeof(cliaddr);
            bzero(buffer, sizeof(buffer));
            printf("\nMessage from UDP client: ");
            n = recvfrom(udpfd, buffer, sizeof(buffer), 0,
                         (struct sockaddr*)&cliaddr, &len);
            puts(buffer);
            sendto(udpfd, (const char*)message, sizeof(buffer), 0,
                   (struct sockaddr*)&cliaddr, sizeof(cliaddr));
        }
    }
}
```

seltcpcli.c

```
// TCP Client program
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#define PORT 5000
#define MAXLINE 1024
int main()
{
    int sockfd;
    char buffer[MAXLINE];
    char* message = "Hello Server";
    struct sockaddr_in servaddr;
```

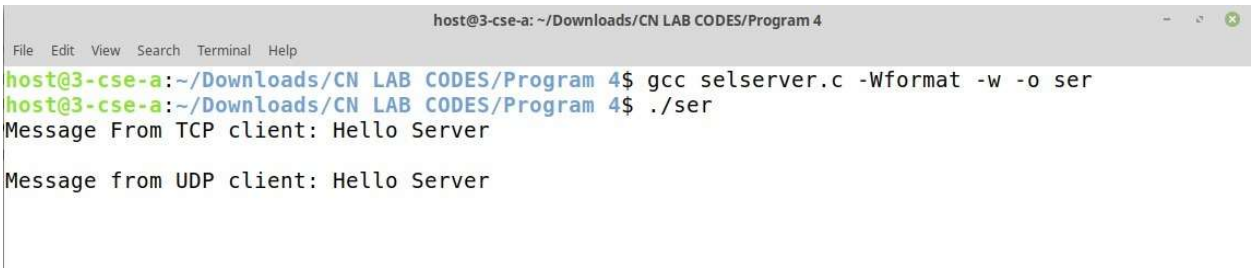
```
    int n, len;
    // Creating socket file descriptor
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("socket creation failed");
        exit(0);
    }

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

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

    if (connect(sockfd, (struct sockaddr*)&servaddr,
                            sizeof(servaddr)) < 0) {
        printf("\n Error : Connect Failed \n");
    }

    memset(buffer, 0, sizeof(buffer));
    strcpy(buffer, "Hello Server");
    write(sockfd, buffer, sizeof(buffer));
    printf("Message from server: ");
    read(sockfd, buffer, sizeof(buffer));
    puts(buffer);
    close(sockfd);
}
```

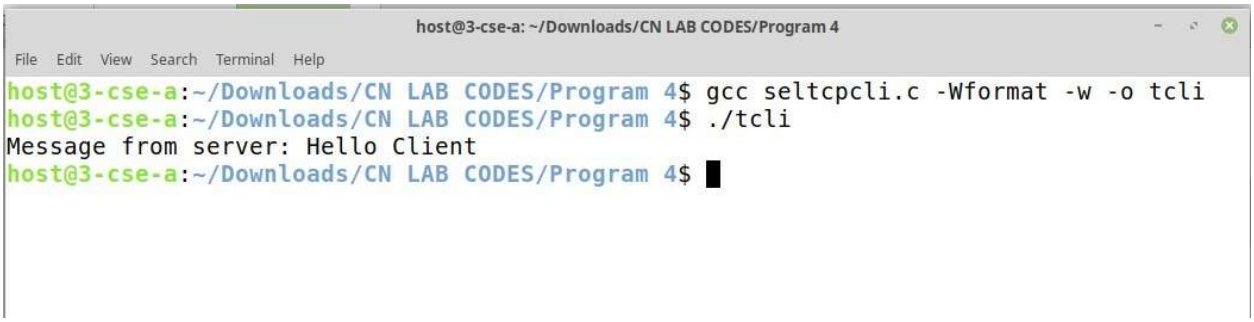
seludpcli.c

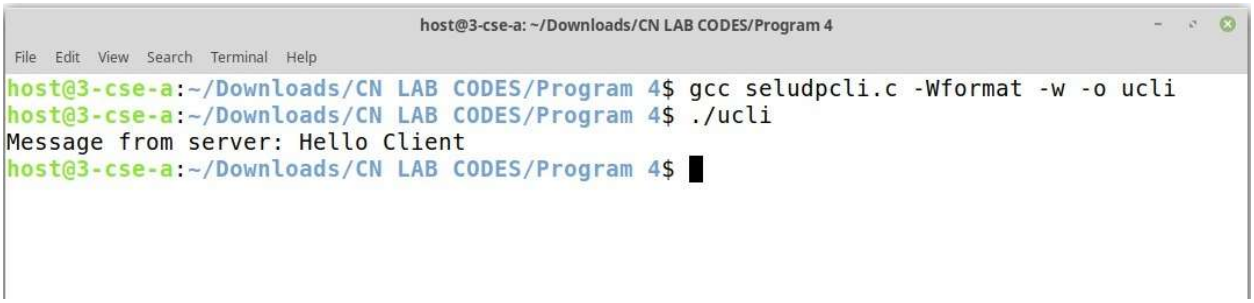
```
// UDP client program
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <sys/socket.h>
#include <sys/types.h>
#define PORT 5000
#define MAXLINE 1024
int main()
{
    int sockfd;
    char buffer[MAXLINE];
    char* message = "Hello Server";
    struct sockaddr_in servaddr;

    int n, len;
    // Creating socket file descriptor
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        printf("socket creation failed");
        exit(0);
    }

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

    // Filling server information
```

```
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(PORT);
    servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
    // send hello message to server
    sendto(sockfd, (const char*)message, strlen(message),
           0, (const struct sockaddr*)&servaddr,
           sizeof(servaddr));

    // receive server's response
    printf("Message from server: ");
    n = recvfrom(sockfd, (char*)buffer, MAXLINE,
                 0, (struct sockaddr*)&servaddr,
                 &len);
    puts(buffer);
    close(sockfd);
    return 0;
}
```

OUTPUT:




host@3-cse-a: ~/Downloads/CN LAB CODES/Program 4
File Edit View Search Terminal Help
host@3-cse-a:~/Downloads/CN LAB CODES/Program 4$ gcc seltcpcli.c -Wformat -w -o tcli
host@3-cse-a:~/Downloads/CN LAB CODES/Program 4$ ./tcli
Message from server: Hello Client
host@3-cse-a:~/Downloads/CN LAB CODES/Program 4$


host@3-cse-a: ~/Downloads/CN LAB CODES/Program 4
File Edit View Search Terminal Help
host@3-cse-a:~/Downloads/CN LAB CODES/Program 4$ gcc seludpcli.c -Wformat -w -o ucli
host@3-cse-a:~/Downloads/CN LAB CODES/Program 4$ ./ucli
Message from server: Hello Client
host@3-cse-a:~/Downloads/CN LAB CODES/Program 4$

EXPERIMENT – 5

AIM:

Implementation of gesockopt (), setsockopt () system calls.

DESCRIPTION:

The getsockopt() and setsockopt() system calls manipulate the options associated with a socket. Options may exist at multiple protocol levels; they are always present at the uppermost "socket" level.

When manipulating socket options the level at which the option resides and the name of the option must be specified. To manipulate options at the socket level, level is specified as SOL_SOCKET.To manipulate options at any other level the protocol number of the appropriate protocol controlling the option is supplied. For example, to indicate that an option is to be interpreted by the TCP protocol, level should be set to the protocol number of TCP.

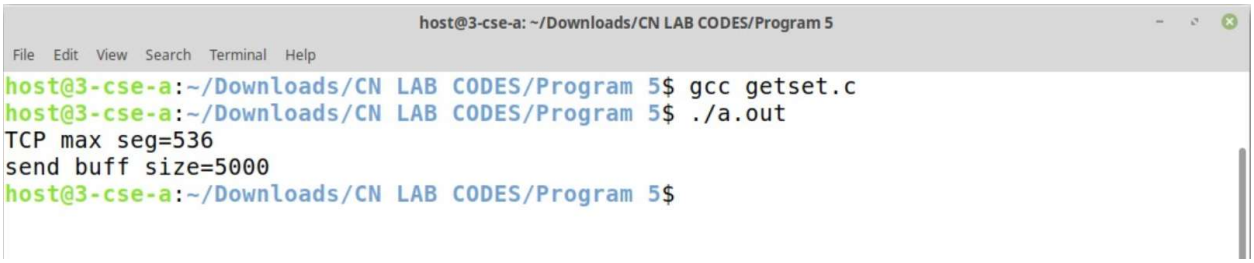
The optval and optlen arguments are used to access option values for setsockopt(). For getsockopt() they identify a buffer in which the value for the requested option(s) are to be returned. For getsockopt(), optlen is a value-result argument, initially containing the size of the buffer pointed to by optval, and modified on return to indicate the actual size of the value returned. If no option value is to be supplied or returned, optval may be NULL.

PROGRAM:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<string.h>
#include<netinet/in.h>
#include<netinet/tcp.h>
void main()
{
        int sockfd,maxseg,sendbuff,optlen;
        sockfd=socket(AF_INET,SOCK_STREAM,0);
        optlen=sizeof(maxseg);
```

```
     if(getsockopt(sockfd,IPPROTO_TCP,TCP_MAXSEG,(char*)&maxseg,&optle
n)<0)
                printf("Max seg error");
       else
                printf("TCP max seg=%d\n",maxseg);
       sendbuff=2500;
     if(setsockopt(sockfd,SOL_SOCKET,SO_SNDBUF,(char*)&sendbuff,sizeof
(sendbuff))<0)
                printf("set error");
        optlen=sizeof(sendbuff);
        getsockopt(sockfd,SOL_SOCKET,SO_SNDBUF,(char
*)&sendbuff,&optlen);
        printf("send buff size=%d\n",sendbuff);
}
```

OUTPUT:



EXPERIMENT – 6

AIM:

Implementation of getpeername () system call.

DESCRIPTION:

getpeername - get name of connected peer socket.

SYNTAX:

The **getpeername**() returns the name of the peer connected to socket *s*.
The *namelen* parameter should be initialized to indicate the amount of space pointed to by *name*. On return it contains the actual size of the name returned (in bytes). The name is truncated if the buffer provided is too small.

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

PROGRAM:

getselserver.c

```
// Server program
#include <arpa/inet.h>
#include <errno.h>
#include <netinet/in.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
#define PORT 5000
#define MAXLINE 1024
int main(int argc,char *argv[])
{
```

```
int s,s2,t,len;
struct sockaddr_in local,rem;
char str[100];
s=socket(AF_INET,SOCK_STREAM,0);
if(s==-1)
{
        perror("socket");
        exit(1);
}
bzero((char *)&local,sizeof(local));
local.sin_family=AF_INET;
local.sin_port=htons(atoi(argv[1]));
local.sin_addr.s_addr=htonl(INADDR_ANY);
if(bind(s,(struct sockaddr *)&local,sizeof(local))==-1)
{
        perror("bind");
        exit(1);
}
if(listen(s,5)==-1)
{
        perror("listen");
        exit(1);
}
for(;;)
{
        int done,n;
        printf("waiting for a connection……\n");
        t=sizeof(rem);
```

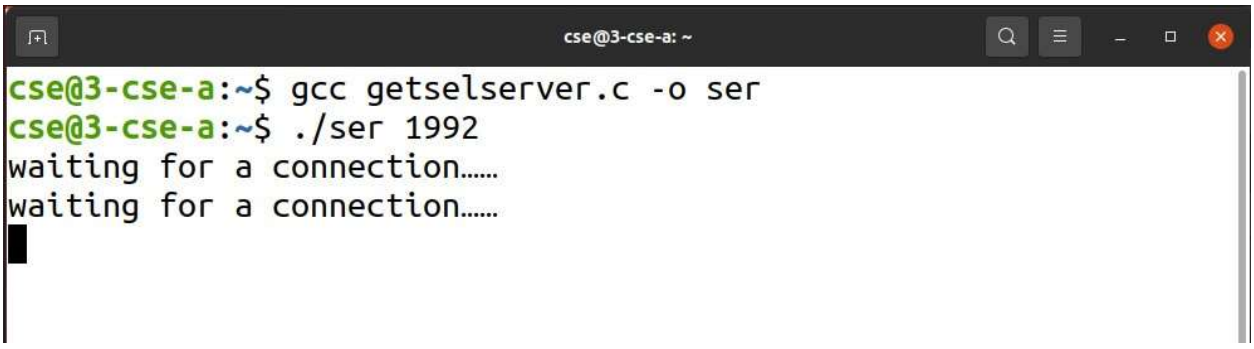
```
                s2=accept(s,(struct sockaddr *)&rem,&t);
                if(s2==-1)
                {
                        perror("accept");
                        exit(1);
                }
        }
        close(s2);
        return 0;
}
```

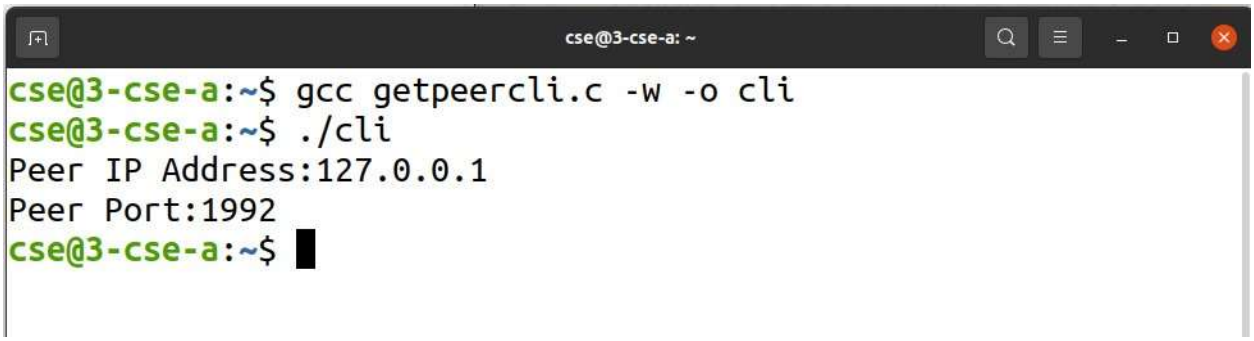
getpeercli.c

```
#include <arpa/inet.h>
#include <errno.h>
#include <netinet/in.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
#define ERROR -1
main()
{
        int s,k;
        struct sockaddr_in server,addr;
        socklen_t len;
        s=socket(AF_INET,SOCK_STREAM,0);
```

```
        server.sin_family=AF_INET;
        inet_aton("127.0.0.1",&server.sin_addr);
        server.sin_port=htons(1992);
        k=connect(s,(struct sockaddr*)&server,sizeof(server));
        if(k<0)
        {
                perror("connected");
                exit(0);
        }
        len=sizeof(addr);
        getpeername(s,(struct sockaddr*)&addr,&len);
        printf("Peer IP Address:%s\n",inet_ntoa(addr.sin_addr));
        printf("Peer Port:%d\n",ntohs(addr.sin_port));
}
```

OUTPUT:





EXPERIMENT – 7

AIM:

Implementation of remote command execution using socket system calls.

DESCRIPTION:

1.The program is used for the communication of client with the server. In this program the server listens to the client defined in the program as"listen(sfd,5);".
2.The example illustrates as the server listens to the maximum of 5 clients at a time.
3.The client program connects with the server through the"connect"system call and the server binds with the client with"bind"system call. this implements the socket method.
4.This also consists of the AF_INET family and SOCK_STREAM implementation with protocol suit determined by the operating system.
5.The server and client communications in the string data exchanging manner. In this the server listens to the clients and display the shell commands.

PROGRAM:

Rceserver.c

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<errno.h>
int main()
{
      int sd,acpt,len,bytes,port;
      char send[50],receiv[50];
      struct sockaddr_in serv,cli;
```

```
if((sd=socket(AF_INET,SOCK_STREAM,0))<0){
      printf("Error in socket\n");
      exit(0);
}
bzero(&serv,sizeof(serv));
printf("Enter the port number  :  ");
scanf("%d",&port);
serv.sin_family=AF_INET;
serv.sin_port=htons(port);
serv.sin_addr.s_addr=htonl(INADDR_ANY);
if(bind(sd,(struct sockaddr *)&serv,sizeof(serv))<0){
      printf("Error in bind\n");
      exit(0);
}
if(listen(sd,3)<0){
      printf("Error in listen\n");
      exit(0);
}
if((acpt=accept(sd,(struct sockaddr*)NULL,NULL))<0){
      printf("\n\t Error in accept");
      exit(0);
}
while(1){
      bytes=recv(acpt,receiv,50,0);
      receiv[bytes]='\0';
      if(strcmp(receiv ,"end")==0){
            close(acpt);
            close(sd);
            exit(0);
```

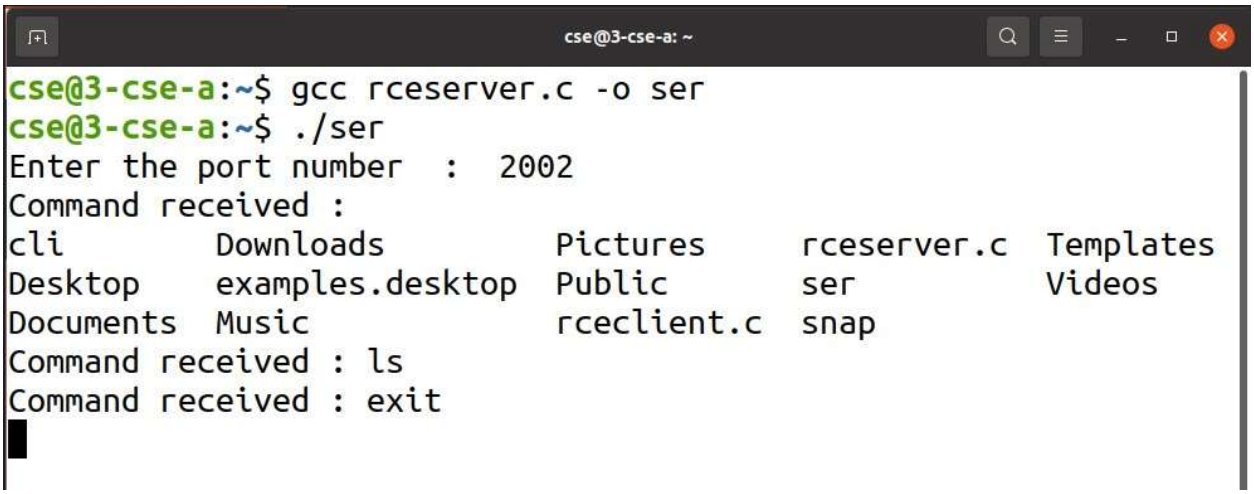
```
	}
	else{
		printf("Command received : %s",receiv);
		system(receiv);
		printf("\n");
	}
	}
}
```

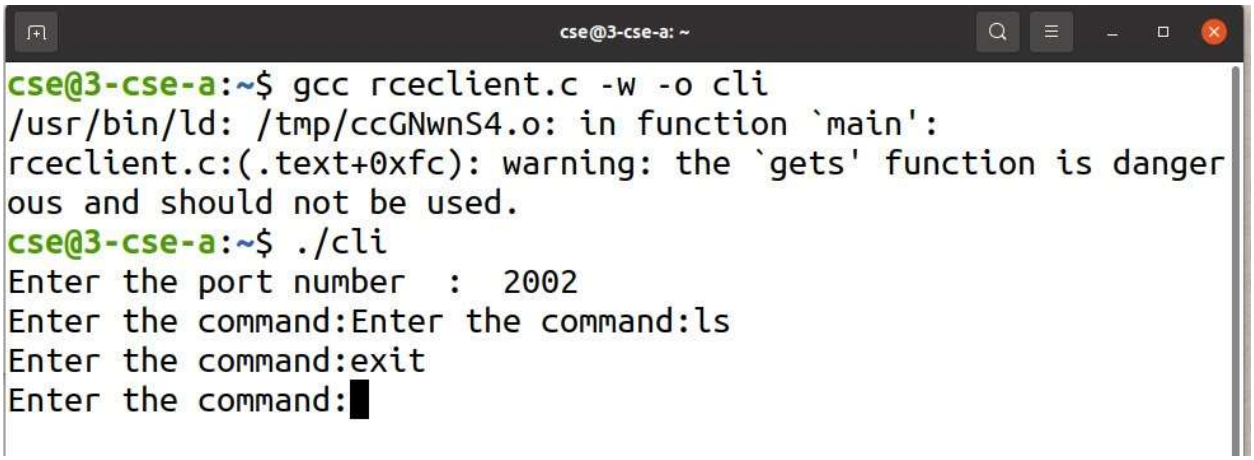
Rceclient.c

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<errno.h>
int main()
{
	int sd,acpt,len,bytes,port;
	char send1[50],receiv[50];
	struct sockaddr_in serv,cli;
	if((sd=socket(AF_INET,SOCK_STREAM,0))<0){
		printf("Error in socket\n");
		exit(0);
	}
	bzero(&serv,sizeof(serv));
	printf("Enter the port number  :  ");
```

```
        scanf("%d",&port);
        serv.sin_family=AF_INET;
        serv.sin_port=htons(port);
        serv.sin_addr.s_addr=htonl(INADDR_ANY);
        if(connect(sd,(struct sockaddr *)&serv,sizeof(serv))<0){
              printf("Error in connection\n");
              exit(0);
        }

        while(1)
        {
              printf("Enter the command:");
              gets(send1);
              if(strcmp(send1,"end")!=0){
                    send(sd,send1,50,0);
              }
              else{
                    send(sd,send1,50,0);
                    close(sd);
                    break;
              }
        }
  }
```

OUTPUT:


cse@3-cse-a: ~
cse@3-cse-a:~$ gcc rceserver.c -o ser
cse@3-cse-a:~$ ./ser
Enter the port number : 2002
Command received :
cli Downloads Pictures rceserver.c Templates
Desktop examples.desktop Public ser Videos
Documents Music rceclient.c snap
Command received : ls
Command received : exit


cse@3-cse-a: ~
cse@3-cse-a:~$ gcc rceclient.c -w -o cli
/usr/bin/ld: /tmp/ccGNwnS4.o: in function `main':
rceclient.c:(.text+0xfc): warning: the `gets' function is danger
ous and should not be used.
cse@3-cse-a:~$ ./cli
Enter the port number : 2002
Enter the command:Enter the command:ls
Enter the command:exit
Enter the command:

EXPERIMENT – 8

AIM

Implementation of Distance Vector Routing Algorithm.

DESCRIPTION

A distance-vector routing (DVR) protocol requires that a router inform its neighbors of topology changes periodically. Historically known as the old ARPANET routing algorithm (or known as Bellman-Ford algorithm).

1. A router transmits its distance vector to each of its neighbors in a routing packet.
2. Each router receives and saves the most recently received distance vector from each of its neighbors.
3. A router recalculates its distance vector when:
   a. It receives a distance vector from a neighbor containing different information than before.
   b. It discovers that a link to a neighbor has gone down.

From time-to-time, each node sends its own distance vector estimate to neighbors.
When a node x receives new DV estimate from any neighbor v, it saves v's distance vector and it updates its own DV using B-F equation:
$D_x(y) = \min \{ C(x,v) + D_v(y), D_x(y) \}$ for each node $y \in N$

PROGRAM:

```
import java.io.*;

public class dvr

{

   static int graph[][];

   static int via[][];

   static int rt[][];

   static int v;

   static int e;

   public static void main(String args[]) throws IOException

   {

      BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
```

```
 System.out.println("Please enter the number of Vertices: ");
 v = Integer.parseInt(br.readLine());

System.out.println("Please enter the number of Edges: ");
e = Integer.parseInt(br.readLine());
graph = new int[v][v];
via = new int[v][v];
rt = new int[v][v];
for(int i = 0; i < v; i++)
  for(int j = 0; j < v; j++)
  {
      if(i == j)
         graph[i][j] = 0;
      else
         graph[i][j] = 9999;
  }
for(int i = 0; i < e; i++)
{
    System.out.println("Please enter data for Edge " + (i + 1) + ":");
    System.out.print("Source: ");
    int s = Integer.parseInt(br.readLine());
    s--;
    System.out.print("Destination: ");
    int d = Integer.parseInt(br.readLine());
    d--;
    System.out.print("Cost: ");
    int c = Integer.parseInt(br.readLine());
    graph[s][d] = c;
    graph[d][s] = c;
```

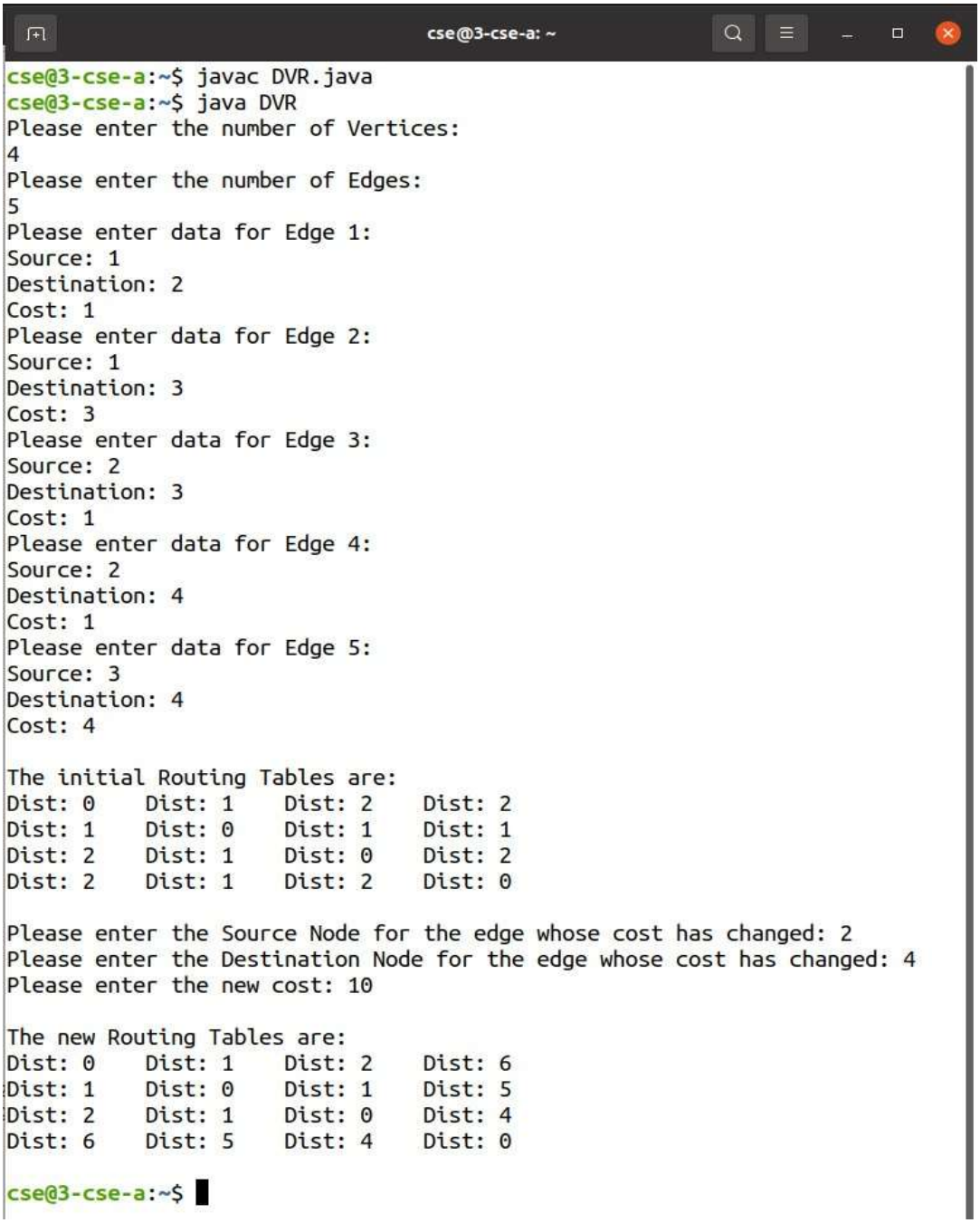
```
    }
    dvr_calc_disp("The initial Routing Tables are: ");
    System.out.print("Please enter the Source Node for the edge whose cost
has changed: ");
    int s = Integer.parseInt(br.readLine());
    s--;
    System.out.print("Please enter the Destination Node for the edge whose
cost has changed: ");
    int d = Integer.parseInt(br.readLine());
    d--;
    System.out.print("Please enter the new cost: ");
    int c = Integer.parseInt(br.readLine());
    graph[s][d] = c;
    graph[d][s] = c;
    dvr_calc_disp("The new Routing Tables are: ");
}
static void dvr_calc_disp(String message)
{
    System.out.println();
    init_tables();
    update_tables();
    System.out.println(message);
    print_tables();
    System.out.println();
}
static void update_table(int source)
{
    for(int i = 0; i < v; i++)
    {
       if(graph[source][i] != 9999)
```

```
        {
     int dist = graph[source][i];
     for(int j = 0; j < v; j++)
     {
      int inter_dist = rt[i][j];
      if(via[i][j] == source)
       inter_dist = 9999;
      if(dist + inter_dist < rt[source][j])
      {
       rt[source][j] = dist + inter_dist;
       via[source][j] = i;
      }
     }
    }
  }
}

static void update_tables()
{
  int k = 0;
  for(int i = 0; i < 4*v; i++)
  {
   update_table(k);
   k++;
   if(k == v)
    k = 0;
  }
}
```

```
static void init_tables()
{
  for(int i = 0; i < v; i++)
  {
   for(int j = 0; j < v; j++)
   {
    if(i == j)
    {
     rt[i][j] = 0;
     via[i][j] = i;
    }
    else
    {
     rt[i][j] = 9999;
     via[i][j] = 100;
    }
   }
  }
}

static void print_tables()
{
  for(int i = 0; i < v; i++)
  {
   for(int j = 0; j < v; j++)
   {
    System.out.print("Dist: " + rt[i][j] + "    ");
   }
   System.out.println();
```

```
    }
}


}
```

OUTPUT:


cse@3-cse-a: ~
cse@3-cse-a:~$ javac DVR.java
cse@3-cse-a:~$ java DVR
Please enter the number of Vertices:
4
Please enter the number of Edges:
5
Please enter data for Edge 1:
Source: 1
Destination: 2
Cost: 1
Please enter data for Edge 2:
Source: 1
Destination: 3
Cost: 3
Please enter data for Edge 3:
Source: 2
Destination: 3
Cost: 1
Please enter data for Edge 4:
Source: 2
Destination: 4
Cost: 1
Please enter data for Edge 5:
Source: 3
Destination: 4
Cost: 4
The initial Routing Tables are:
Dist: 0 Dist: 1 Dist: 2 Dist: 2
Dist: 1 Dist: 0 Dist: 1 Dist: 1
Dist: 2 Dist: 1 Dist: 0 Dist: 2
Dist: 2 Dist: 1 Dist: 2 Dist: 0
Please enter the Source Node for the edge whose cost has changed: 2
Please enter the Destination Node for the edge whose cost has changed: 4
Please enter the new cost: 10
The new Routing Tables are:
Dist: 0 Dist: 1 Dist: 2 Dist: 6
Dist: 1 Dist: 0 Dist: 1 Dist: 5
Dist: 2 Dist: 1 Dist: 0 Dist: 4
Dist: 6 Dist: 5 Dist: 4 Dist: 0
cse@3-cse-a:~$

EXPERIMENT – 10

AIM:

Implementation of FTP

DESCRIPTION:

The File Transfer Protocol (FTP) is a standard network protocol used to transfer computer files from one host to another host over a TCP-based network, such as the Internet.

FTP is built on a client-server architecture and uses separate control and data connections between the client and the server. FTP users may authenticate themselves using a clear-text sign-in protocol, normally in the form of a username and password, but can connect anonymously if the server is configured to allow it. For secure transmission that protects the username and password, and encrypts the content, FTP is often secured with SSL/TLS (FTPS)

PROGRAM:

Ftpserver.java

```
import java.io.BufferedInputStream;
import java.io.BufferedOUTPUT:Stream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOUTPUT:Stream;
import java.io.InputStream;
import java.io.OUTPUT:Stream;
import java.net.ServerSocket;
import java.net.Socket;


public class ftpserver {
    public static void main(String[] args) throws Exception {
        // create socket
        ServerSocket servsock = new ServerSocket(13267);
        while (true) {
            System.out.println("Waiting...");
```

```
        Socket sock = servsock.accept();
        System.out.println("Accepted connection : " + sock);
        OUTPUT:Stream os = sock.getOUTPUT:Stream();
        //new FileServer().send(os);
        InputStream is = sock.getInputStream();
        new ftpserver().receiveFile(is);
        sock.close();
    }
}

public void send(OUTPUT:Stream os) throws Exception {
    // sendfile
    File myFile = new File("/home/niles/opt/eclipse/about.html");
    byte[] mybytearray = new byte[(int) myFile.length() + 1];
    FileInputStream fis = new FileInputStream(myFile);
    BufferedInputStream bis = new BufferedInputStream(fis);
    bis.read(mybytearray, 0, mybytearray.length);
    System.out.println("Sending...");
    os.write(mybytearray, 0, mybytearray.length);
    os.flush();
}

public void receiveFile(InputStream is) throws Exception {
    int filesize = 6022386;
    int bytesRead;
    int current = 0;
    byte[] mybytearray = new byte[filesize];

    FileOUTPUT:Stream fos = new FileOUTPUT:Stream("def");
```

```
        BufferedOUTPUT:Stream bos = new BufferedOUTPUT:Stream(fos);
        bytesRead = is.read(mybytearray, 0, mybytearray.length);
        current = bytesRead;

        do {
            bytesRead = is.read(mybytearray, current,
                    (mybytearray.length - current));
            if (bytesRead >= 0)
                current += bytesRead;
        } while (bytesRead > -1);

        bos.write(mybytearray, 0, current);
        bos.flush();
        bos.close();
    }
}
```

Ftpclient.java

```
import java.io.BufferedInputStream;
import java.io.BufferedOUTPUT:Stream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOUTPUT:Stream;
import java.io.InputStream;
import java.io.OUTPUT:Stream;
import java.net.Socket;

public class ftpclient {
    public static void main(String[] args) throws Exception {
```

```
        long start = System.currentTimeMillis();

        // localhost for testing
        Socket sock = new Socket("127.0.0.1", 13267);
        System.out.println("Connecting...");
        InputStream is = sock.getInputStream();
        // receive file
        new ftpclient().receiveFile(is);
        OUTPUT:Stream os = sock.getOUTPUT:Stream();
        //new FileClient().send(os);
        long end = System.currentTimeMillis();
        System.out.println(end - start);

        sock.close();
    }


    public void send(OUTPUT:Stream os) throws Exception {
        // sendfile
        File myFile = new File("/home/niles/opt/eclipse/about.html");
        byte[] mybytearray = new byte[(int) myFile.length() + 1];
        FileInputStream fis = new FileInputStream(myFile);
        BufferedInputStream bis = new BufferedInputStream(fis);
        bis.read(mybytearray, 0, mybytearray.length);
        System.out.println("Sending...");
        os.write(mybytearray, 0, mybytearray.length);
        os.flush();
    }
```

```
    public void receiveFile(InputStream is) throws Exception {
        int filesize = 6022386;
        int bytesRead;
        int current = 0;
        byte[] mybytearray = new byte[filesize];

        FileOUTPUT:Stream fos = new FileOUTPUT:Stream("def");
        BufferedOUTPUT:Stream bos = new BufferedOUTPUT:Stream(fos);
        bytesRead = is.read(mybytearray, 0, mybytearray.length);
        current = bytesRead;

        do {
            bytesRead = is.read(mybytearray, current,
                    (mybytearray.length - current));
            if (bytesRead >= 0)
                current += bytesRead;
        } while (bytesRead > -1);

        bos.write(mybytearray, 0, current);
        bos.flush();
        bos.close();
    }
}
```

OUTPUT:


cse@3-cse-a: ~
cse@3-cse-a:~$ javac FileServer.java
cse@3-cse-a:~$ java FileServer
Waiting...
Accepted connection : Socket[addr=/127.0.0.1,port=47838,localport
=13267]


cse@3-cse-a: ~
cse@3-cse-a:~$ javac FileClient.java
cse@3-cse-a:~$ java FileClient
Connecting...