DATE :
EXPERIMENT 1

AIM:

To familiarize with the basics of network configuration files and networking commands in Linux.

THEORY:

Networking Commands

Linux networking commands are used extensively to inspect, analyze, maintain, and troubleshoot the networks connected to the system.

1.ip

It is a handy tool for configuring the network interfaces for Linux administrators. It can be used to get the details of a specific interface.

Syntax - ip a
Ip addr

2.traceroute

Linux traceroute command is a network troubleshooting utility that helps us determine the number of hops and packets traveling path required to reach a destination.It can display the routes, IP addresses, and hostnames of routers over a network.

Syntax - traceroute <destination>

3.ping

It basically checks for the network connectivity between two nodes. ping stands for Packet INternet Groper.The ping command sends the ICMP echo request to check the network connectivity.

Syntax - ping <destination>

You can limit the number of packets by including "-c" in the ping command.
Syntax - ping -c <number> <destination>

4,netstat

Linux netstat command stands for Network statistics. It displays information about different interface statistics, including open sockets, routing tables, and connection information.
Syntax - netstat

5.hostname

hostname command allows us to set and view the hostname of the system. A hostname is the name of any computer that is connected to a network that is uniquely identified over a network.
Syntax - hostname

6.ifconfig

Linux ifconfig stands for interface configurator. It is one of the most basic commands used in network inspection. ifconfig is used to initialize an interface, configure it with an IP address, and enable or disable it. It is also used to display the route and the network interface.

Syntax - ifconfig

7.arp

Linux arp command stands for Address Resolution Protocol. It is used to view and add content to the kernel's ARP table.All the systems maintain a table of IP addresses and their corresponding MAC addresses. This table is called the ARP Lookup table.

Syntax - arp

8.whois

Linux whois command is used to fetch all the information related to a website. You can get all the information about a website including the registration and the owner information.

Syntax - whois <websiteName>

9.nslookup

nslookup (stands for “Name Server Lookup”) is a useful command for getting information from the DNS server. It is a network administration tool for querying the Domain Name System (DNS) to obtain domain name or IP address mapping or any other specific DNS record. It is also used to troubleshoot DNS-related problems.

Syntax - nslookup <domainName>

10.ftp

FTP (File Transfer Protocol) is a network protocol used for transferring files from one computer system to another. Even though the safety of FTP tends to spark a lot of discussion, it is still an effective method of transferring files within a secure network.

Syntax - ftp [options] [IP address]

11.telnet

In Linux, the **telnet** command is used to create a remote connection with a system over a TCP/IP network. It allows us to administrate other systems by the terminal. We can run a program to conduct administration.

Syntax - telnet hostname/IP address

12.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.

Syntax - finger [option] [username]

Network Configuration Files

To store IP addresses and other related settings, Linux uses a separate configuration file for each network interface. All these configuration files are stored in the */etc/sysconfig/network-scripts* directory.The important linux network configuration files are:

1. /etc/hosts

This file is used to map the hostname with IP address. Once hostname and IP address are mapped, hostname can be used to access the services available on the destination IP address. A hostname can be mapped with an IP address in two ways; through the DNS server and through the */etc/hosts* file.

2. /etc/resolv.conf

The /etc/resolv.conf configuration file specifies the IP addresses of DNS servers and the search domain.

3. /etc/sysconfig/network

This file specifies routing and host information for all network interfaces.

4. /etc/nsswitch.conf

The "/etc/nsswitch.conf" file contains your settings as to how various system lookups are carried out. One of the main functions of the "nsswitch.conf is to control how your network is resolved.

OUTPUT:

```
root@cc-2-4:/home/mec# ifconfig
enp1s0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 8c:ec:4b:c8:ae:5f  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

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 1  (Local Loopback)
        RX packets 100  bytes 7596 (7.4 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 100  bytes 7596 (7.4 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlp2s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.3.121  netmask 255.255.252.0  broadcast 192.168.3.255
        inet6 fe80::60fd:549e:4fcd:2187  prefixlen 64  scopeid 0x20<link>
        ether 48:5f:99:63:52:cf  txqueuelen 1000  (Ethernet)
        RX packets 31424  bytes 11369495 (10.8 MiB)
        RX errors 0  dropped 1  overruns 0  frame 0
        TX packets 5598  bytes 1258203 (1.1 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@cc-2-4:/home/mec# netstat -r
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
default         gateway         0.0.0.0         UG        0 0          0 wlp2s0
192.168.0.0     0.0.0.0         255.255.252.0   U         0 0          0 wlp2s0

root@cc-2-4:/home/mec# ping www.google.com
PING www.google.com (142.250.77.100) 56(84) bytes of data.
64 bytes from maa05s15-in-f4.1e100.net (142.250.77.100): icmp_seq=1 ttl=57 time=22.3 ms
64 bytes from maa05s15-in-f4.1e100.net (142.250.77.100): icmp_seq=2 ttl=57 time=42.8 ms
64 bytes from maa05s15-in-f4.1e100.net (142.250.77.100): icmp_seq=3 ttl=57 time=23.3 ms
64 bytes from maa05s15-in-f4.1e100.net (142.250.77.100): icmp_seq=4 ttl=57 time=23.9 ms
^Z
[1]+  Stopped                 ping www.google.com
root@cc-2-4:/home/mec# ping -c 10 www.google.com
PING www.google.com (142.250.77.100) 56(84) bytes of data.
64 bytes from maa05s15-in-f4.1e100.net (142.250.77.100): icmp_seq=1 ttl=57 time=30.4 ms
64 bytes from maa05s15-in-f4.1e100.net (142.250.77.100): icmp_seq=2 ttl=57 time=25.3 ms
64 bytes from maa05s15-in-f4.1e100.net (142.250.77.100): icmp_seq=3 ttl=57 time=31.0 ms
64 bytes from maa05s15-in-f4.1e100.net (142.250.77.100): icmp_seq=4 ttl=57 time=30.2 ms
64 bytes from maa05s15-in-f4.1e100.net (142.250.77.100): icmp_seq=5 ttl=57 time=36.7 ms
64 bytes from maa05s15-in-f4.1e100.net (142.250.77.100): icmp_seq=6 ttl=57 time=41.1 ms
64 bytes from maa05s15-in-f4.1e100.net (142.250.77.100): icmp_seq=7 ttl=57 time=29.8 ms
64 bytes from maa05s15-in-f4.1e100.net (142.250.77.100): icmp_seq=8 ttl=57 time=27.8 ms
64 bytes from maa05s15-in-f4.1e100.net (142.250.77.100): icmp_seq=9 ttl=57 time=28.0 ms
64 bytes from maa05s15-in-f4.1e100.net (142.250.77.100): icmp_seq=10 ttl=57 time=35.5 ms

--- www.google.com ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9014ms
rtt min/avg/max/mdev = 25.390/31.636/41.197/4.552 ms
```

```
mec@cc-2-2:~$ ftp
ftp> help
Commands may be abbreviated.  Commands are:

!               dir             mdelete         qc              site
$               disconnect      mdir            sendport        size
account         exit            mget            put             status
append          form            mkdir           pwd             struct
ascii           get             mls             quit            system
bell            glob            mode            quote           sunique
binary          hash            modtime         recv            tenex
bye             help            mput            reget           tick
case            idle            newer           rstatus         trace
cd              image           nmap            rhelp           type
cdup            ipany           nlist           rename          user
chmod           ipv4            ntrans          reset           umask
close           ipv6            open            restart         verbose
cr              lcd             prompt          rmdir           ?
delete          ls              passive         runique
debug           macdef          proxy           send
ftp> status
Not connected.
No proxy connection.
Connecting using address family: any.
Mode: ; Type: ; Form: ; Structure:
Verbose: on; Bell: off; Prompting: on; Globbing: on
Store unique: off; Receive unique: off
Case: off; CR stripping: on
Quote control characters: on
Ntrans: off
Nmap: off
Hash mark printing: off; Use of PORT cmds: on
Tick counter printing: off
ftp> open
(to) localhost
ftp: connect to address ::1: Connection refused
Trying 127.0.0.1...
ftp: connect: Connection refused
ftp> close
Not connected.
ftp> exit

mec@cc-2-2:~$ telnet india.colorado.edu 13
Trying 128.138.140.44...
Connected to india.colorado.edu.
Escape character is '^]'.

59712 22-05-13 09:58:04 50 0 0 832.7 UTC(NIST) *
Connection closed by foreign host.

mec@cc-2-2:~$ finger
Login     Name       Tty      Idle  Login Time   Office     Office Phone
mec       mec        tty2     1:01  May 13 14:12 (:1)
```

```
root@cc-2-4:/home/mec# traceroute google.com
traceroute to google.com (142.250.195.206), 30 hops max, 60 byte packets
 1  gateway (192.168.0.2)  4.649 ms  4.617 ms  5.819 ms
 2  14.139.184.209 (14.139.184.209)  5.800 ms  7.057 ms  7.046 ms
 3  * * *
 4  * * *
 5  * * *
 6  10.119.73.122 (10.119.73.122)  36.214 ms  22.833 ms  22.797 ms
 7  72.14.213.20 (72.14.213.20)  26.787 ms 72.14.195.128 (72.14.195.128)  26.748 ms  27.541 ms
 8  * * *
 9  142.250.228.186 (142.250.228.186)  23.047 ms 142.250.235.106 (142.250.235.106)  25.416 ms 142.251.55.90 (142.251.55.90)  24.588 ms
10  74.125.242.130 (74.125.242.130)  23.792 ms  25.107 ms 74.125.242.155 (74.125.242.155)  24.979 ms
11  108.170.253.97 (108.170.253.97)  25.724 ms 108.170.253.113 (108.170.253.113)  24.189 ms 108.170.253.97 (108.170.253.97)  22.688 ms
12  142.251.49.219 (142.251.49.219)  26.262 ms maa03s42-in-f14.1e100.net (142.250.195.206)  26.240 ms 142.251.49.219 (142.251.49.219)  26.263 ms
```

```
mec@cc-2-2:~$ whois 14.139.184.212
% [whois.apnic.net]
% Whois data copyright terms    http://www.apnic.net/db/dbcopyright.html

% Information related to '14.139.184.208 - 14.139.184.223'

% Abuse contact for '14.139.184.208 - 14.139.184.223' is 'abuseteam@nkn.in'

inetnum:        14.139.184.208 - 14.139.184.223
netname:        NKN-MEC-TRIV
descr:          Model Engineering College, Thrikkakara
country:        IN
admin-c:        NNA22-AP
tech-c:         STJ1-AP
abuse-c:        AN1623-AP
status:         ALLOCATED NON-PORTABLE
mnt-by:         MAINT-RSMANI-NKN-IN
mnt-irt:        IRT-NKN-MEC-TRIV
last-modified:  2021-02-18T13:31:37Z
source:         APNIC

irt:            IRT-NKN-MEC-TRIV
address:        Model Engineering College, Thrikkakara,
address:        Kochi Kerala
address:        Trivandrum
address:        IN
e-mail:         support.kl@nkn.in
abuse-mailbox:  abuseteam@nkn.in
admin-c:        NNA22-AP
tech-c:         STJ1-AP
auth:           # Filtered
remarks:        abuseteam@nkn.in was validated on 2022-02-20
remarks:        support.kl@nkn.in was validated on 2022-03-28
mnt-by:         MAINT-RSMANI-NKN-IN
last-modified:  2022-03-28T08:00:35Z
source:         APNIC

role:           ABUSE NKNMECTRIV
address:        Model Engineering College, Thrikkakara,
address:        Kochi Kerala
address:        Trivandrum
address:        IN
country:        ZZ
phone:          +000000000
e-mail:         support.kl@nkn.in
admin-c:        NNA22-AP
tech-c:         STJ1-AP
nic-hdl:        AN1623-AP
remarks:        Generated from irt object IRT-NKN-MEC-TRIV
remarks:        abuseteam@nkn.in was validated on 2022-02-20
remarks:        support.kl@nkn.in was validated on 2022-03-28
abuse-mailbox:  abuseteam@nkn.in
mnt-by:         APNIC-ABUSE
last-modified:  2022-03-28T08:02:30Z
source:         APNIC

role:           NKN - Network Administrator
address:        National Knowledge Network
address:        3rd Floor, Block III,
address:        Delhi IT Park, Shastri Park
address:        New Delhi - 110053
country:        IN
phone:          +91 - 1800111555
e-mail:         support@nkn.in
admin-c:        MR135-AP
tech-c:         GK397-AP
nic-hdl:        NNA22-AP
abuse-mailbox:  abuseteam@nkn.in
mnt-by:         MAINT-RSMANI-NKN-IN
last-modified:  2015-11-18T13:09:41Z
source:         APNIC

person:         Shri Titty Jacob
address:        Model Engineering College, Thrikkakara, Kochi, Kerala.PIN: 682021
country:        IN
phone:          +919446037221
e-mail:         ttjacob@mec.ac.in
nic-hdl:        STJ1-AP
mnt-by:         MAINT-RSMANI-NKN-IN
last-modified:  2021-02-18T13:30:04Z
source:         APNIC

% Information related to '14.139.184.0/24AS55824'

route:          14.139.184.0/24
origin:         AS55824
descr:          National Knowledge Network
                C/O National Informatics Centre
                Ministry Of Comm & IT  A-Block
                CGO Complex Lodhi Road
mnt-by:         MAINT-RSMANI-NKN-IN
last-modified:  2019-01-10T11:58:13Z
source:         APNIC

% This query was served by the APNIC Whois Service version 1.88.16 (WHOIS-JP1)
```

```
mec@cc-2-2:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp1s0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state DOWN group default qlen 1000
    link/ether 8c:ec:4b:c8:b3:51 brd ff:ff:ff:ff:ff:ff
3: wlp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 48:5f:99:63:64:33 brd ff:ff:ff:ff:ff:ff
    inet 192.168.2.37/22 brd 192.168.3.255 scope global dynamic wlp2s0
       valid_lft 17825sec preferred_lft 17825sec
    inet6 fe80::b98:6e96:a406:745f/64 scope link
       valid lft forever preferred_lft forever

root@CC-1-6:/home/mec/cs6a-5# nslookup
> www.google.com
Server:         192.168.0.2
Address:        192.168.0.2#53

Non-authoritative answer:
Name:   www.google.com
Address: 172.217.166.100

mec@cc-2-2:~$ hostname
cc-2-2

root@cc-2-4:/home/mec# cat /etc/hosts
127.0.0.1       localhost
127.0.1.1       cc-2-4.mec.ac.in        cc-2-4

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
root@cc-2-4:/home/mec# cat /etc/resolv.conf
# Generated by NetworkManager
search mec.ac.in
nameserver 192.168.0.2
nameserver 192.168.0.6
root@cc-2-4:/home/mec# cat /etc/nsswitch.conf
# /etc/nsswitch.conf
#
# Example configuration of GNU Name Service Switch functionality.
# If you have the `glibc-doc-reference' and `info' packages installed, try:
# `info libc "Name Service Switch"' for information about this file.

passwd:         compat
group:          compat
shadow:         compat
gshadow:        files

hosts:          files mdns4_minimal [NOTFOUND=return] dns myhostname
networks:       files

protocols:      db files
services:       db files
ethers:         db files
rpc:            db files

netgroup:       nis
```

RESULT:

The experiment was executed successfully.

EXPERIMENT 2

AIM:

To familiarize and understand the use and functioning of system calls used for network programming in Linux

THEORY:

1.socket()

The socket system call creates a new socket by assigning a new descriptor. Any subsequent system calls are identified with the created socket.

Syntax **- int socket(int** *domain***, int** *type***, int** *protocol***);**

2.bind()

The bind system call associates a local network transport address with a socket. For a client process, it is not mandatory to issue a bind call. It is often necessary for a server process to issue an explicit bind request before it can accept connections or start communication with clients.

Syntax - **int bind(int** *sockfd***, const struct sockaddr** ******addr***, socklen_t** *addrlen***);**

3.connect()

The **connect**() system call connects the socket referred to by the file descriptor sockfd to the address specified by addr. The addrlen argument specifies the size of addr. The connect system call is normally called by the client process to connect to the server process.

Syntax - **int connect(int** *sockfd***, const struct sockaddr** ******addr***,**
socklen_t *addrlen***);**

4.listen()

listen() marks the socket referred to by *sockfd* as a passive socket, that is, as a socket that will be used to accept incoming connection requests using accept().There is a limit on the number of connections that can be queued up, after which any further connection requests are ignored.

Syntax - **int listen(int** *sockfd*, **int** *backlog*);

5.accept()

The accept system call is a blocking call that waits for incoming connections. Once a connection request is processed, a new socket descriptor is returned by accept. This new socket is connected to the client and the other sockets remain in LISTEN state to accept further connections.

Syntax - **int accept(int** *sockfd*, **struct sockaddr *restrict** *addr*,
socklen_t *restrict *addrlen*);

6.send()/sendto()

These system calls are used to send messages or data.send() is used in connection oriented protocols while sendto() is used in connection-less protocols.

Syntax -

send(int *sockfd*, **const void** **buf*, **size_t** *len*, **int** *flags*);

sendto(int *sockfd*, **const void** **buf*, **size_t** *len*, **int** *flags*,
const struct sockaddr **dest_addr*, **socklen_t** *addrlen*);

7.recv()/recvfrom()

These system calls are used to send messages or data.recv() is used in connection oriented protocols while recvfrom() is used in connection-less protocols.

Syntax -

recv(int *sockfd*, **void** **buf*, **size_t** *len*, **int** *flags*);

recvfrom(int *sockfd*, **void *restrict** *buf*, **size_t** *len*, **int** *flags*,
struct sockaddr *restrict *src_addr*,
socklen_t *restrict *addrlen*);

8.close()

Sockets need to be closed after they are not being used anymore. Its only argument is the *socket* file descriptor and it returns 0 once it's successfully closed.
Syntax - int close(int *fd*);

RESULT:

The experiment was executed successfully.

EXPERIMENT 3

AIM:

To implement client-server communication using socket programming and TCP as transport layer protocol

THEORY:

A TCP (transmission control protocol) is a connection-oriented communication. TCP is designed to send the data packets over the network. It ensures that data is delivered to the correct destination.TCP creates a connection between the source and destination node before transmitting the data and keeps the connection alive until the communication is active.

Server-client model is communication model for sharing the resource and provides the service to different machines. Server is the main system which provides the resources and different kind of services when client requests to use it.

ALGORITHM:

Server Algorithm:-

1. Create a socket with type as SOCK_STREAM to create a tcp socket using socket() system call.
2. Bind the socket to a specific port using bind() system call.
3. Listen for new connections using the listen() system call.
4. Accept connection from client process into a temporary socket.
5. Read the message from the client using the recv() system call into a buffer.
6. Display the message and close the socket.

Client Algorithm:-

1. Create a socket with type as SOCK_STREAM to create a tcp socket using socket() system call.
2. Connect to the server using connect() system call.
3. Read the message to be sent from the user.
4. Send the message to the server using send() system call.
5. Close the socket.

PROGRAM:

Server

```
#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 connfd)
{
  char buff[MAX];
  int n;
  for (;;) {
    bzero(buff, MAX);
    read(connfd, buff, sizeof(buff));
    printf("From client: %s\t To client : ", buff);
    bzero(buff, MAX);
    n = 0;
    while ((buff[n++] = getchar()) != '\n')
      ;
    write(connfd, buff, sizeof(buff));
    // if msg contains "Exit" then server exit and chat ended.
    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 accept failed...\n");
    exit(0);
  }
  else
    printf("server accept the client...\n");
  func(connfd);
  close(sockfd);
}
```

Client

```
#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:

Server

```
gcc TCPserver.c -o server
./server
Socket successfully created..
Socket successfully binded..
Server listening..
server accept the client...
From client: hi          To client : hello
From client: exit        To client : exit
Server Exit...
```

Client

```
gcc TCPclient.c -o client
./client
Socket successfully created..
connected to the server..
Enter the string : hi
From Server : hello
Enter the string : exit
From Server : exit
Client Exit...
```

RESULT:

The experiment was executed successfully.

EXPERIMENT 4

AIM:

Implement client-server communication using socket programming and UDP as transport layer protocol

THEORY:

UDP is a connection-less protocol that, unlike TCP, does not require any handshaking prior to sending or receiving data, which simplifies its implementation.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.
Server-client model is communication model for sharing the resource and provides the service to different machines. Server is the main system which provides the resources and different kind of services when client requests to use it.

ALGORITHM:

Server Algorithm:-

1. Create a socket with type as SOCK_DGRAM to create a udp socket using socket() system call.
2. Bind the socket to a specific port using bind() system call.
3. Using the recvfrom() system call message sent from the client process into a buffer.
4. Display the message and close the socket.

Client Algorithm:-

1. Create a socket with type as SOCK_DGRAM to create a udp socket using socket() system call.
2. Read the message to be sent from the user.
3. Send the message to the server using sendto() system call.
4. Close the socket.

PROGRAM:

Server

```
#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 from server";
  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;
  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';
  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

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

OUTPUT:

Server

```
gcc UDPserver.c -o server
./server
Client : Hello from client
Hello message sent.
```

Client

```
gcc UDPclient.c -o client
./client
Hello message sent.
Server : Hello from server
```

RESULT:

The experiment was executed successfully.

EXPERIMENT 5

AIM:

To simulate sliding window flow control protocols (Stop and Wait, Go back N, Selective Repeat ARQ protocols)

THEORY:

Sliding window protocols are data link layer protocols for reliable and sequential delivery of data frames. In this protocol, multiple frames can be sent by a sender at a time before receiving an acknowledgment from the receiver. The term sliding window refers to the imaginary boxes to hold frames.

Stop-and-Wait

It is the simplest flow control method. In this, the sender will transmit one frame at a time to the receiver. The sender will stop and wait for the acknowledgement from the receiver.When the sender gets the acknowledgement (ACK), it will send the next data packet to the receiver and wait for the disclosure again, and this process will continue as long as the sender has the data to send.The sender and receiver window size is 1.

Go-Back-N

Go-Back-N ARQ protocol is also known as Go-Back-N Automatic Repeat Request.In this, if any frame is corrupted or lost, all subsequent frames have to be sent again.The size of the sender window is N in this protocol. The receiver window size is always 1.If the receiver receives a corrupted frame, it cancels it. The receiver does not accept a corrupted frame. When the timer expires, the sender sends the correct frame again.

Selective Repeat

Selective Repeat ARQ is also known as the Selective Repeat Automatic Repeat Request.In this protocol, the size of the sender window is always equal to the size of the receiver window. The size of the sliding window is always greater than 1.If the receiver receives a corrupt frame, it does not directly discard it. It sends a negative acknowledgment to the sender. The sender sends that frame again as soon as on the receiving negative acknowledgment.

ALGORITHM:

Stop and Wait:-

1. Start the program
2. Generate a random number that gives the total number of frames to be transmitted.

3. Transmit the first frame
4. Receive the acknowledgement for the first frame
5. Transmit the next frame
6. Find the remaining frames to be sent.
7. If an acknowledgement is not received for a particular frame, retransmit that frame alone again.
8. Repeat the steps 5 to 7 till the number of remaining frames to be sent becomes zero.
9. Stop the program.

Go Back N:-

Algorithm 8 GoBack-N Protocol - Sender

```
1: S_w ← 2^m − 1
2: S_f = S_n = 0
3: while True do
4:     WaitForEvent()
5:     if Event(RequestToSend) then
6:         if S_n − S_f ≥ S_w then
7:             Sleep()
8:         end if
9:         GetData()
10:        MakeFrame(S_n)
11:        StoreFrame(S_n)
12:        SendFrame(S_n)
13:        S_n ← (S_n + 1)%S_w
14:        if Timer is not running then
15:            StartTimer()
16:        end if
17:    end if
18:    if Event(ArrivalNotification) then
19:        Receive(ACK)
20:        if Corrupted(ACK) then
21:            Sleep()
22:        end if
23:        if ackNo > S_f and ackNo <= S_n then
24:            while S_f ≤ ackNo do
25:                PurgeFrame(S_n)
26:                S_f ← (S_f + 1)%S_w
27:            end while
28:        end if
29:        StopTimer()
30:    end if
31:    if Event(Timeout) then
32:        StartTimer()
33:        temp ← S_f
34:        while temp < S_n do
35:            SendFrame(S_n)
36:            S_f ← (S_f + 1)%S_w
37:        end while
38:    end if
39: end while
```

Algorithm 9 GoBack-N Receiver

```
1: R_n ← 0
2: while True do
3:     WaitForEvent()
4:     if Event(ArrivalNotification) then
5:         Receive(frame)
6:         if Corrupted(frame) then
7:             Sleep()
8:         end if
9:         if seqNo == R_n then
10:            DeliverData()
11:            R_n ← (R_n + 1)%2^m
12:        end if
13:        SendACK(R_n)
14:    end if
15: end while
```

Selective Repeat:-

Algorithm 10 Selective Repeat ARQ - Sender

```
1: S_w ← 2^(m−1)
2: S_f = S_n = 0
3: while True do
4:     WaitForEvent()
5:     if Event(RequestToSend) then
6:         if S_n − S_f ≥ S_w then
7:             Sleep()
8:         end if
9:         GetData()
10:        MakeFrame(S_n)
11:        StoreFrame(S_n)
12:        SendFrame(S_n)
13:        S_n ← (S_n + 1)%S_w
14:        StartTimer(S_n)
15:    end if
16:    if Event(ArrivalNotification) then
17:        Receive(frame)
18:        if Corrupted(frame) then
19:            Sleep()
20:        end if
21:        if FrameType == NAK then
22:            if nakNo in (S_f, S_n] then
23:                Resend(nakNo)
24:                StartTimer(nakNo)
25:            end if
26:        else if FrameType == ACK then
27:            if ackNo in (S_f, S_n] then
28:                while S_f < ackNo do
29:                    Purge(S_f)
30:                    StopTimer(S_f)
31:                    S_f ← (S_f + 1)%2^m
32:                end while
33:            end if
34:        end if
35:    end if
36:    if Event(Timeout T_i) then
37:        StartTimer(T_i)
38:        SendFrame(T_i)
39:    end if
40: end while
```

Algorithm 11 Selective Repeat - Receiver

```
1: R_n ← 0
2: nakSent ← False
3: ackNeeded ← False
4: for all slot in slots do
5:     Marked(slot) ← False
6: end for
7: while True do
8:     WaitForEvent()
9:     if Event(ArrivalNotification) then
10:        Receive(frame)
11:        if Corrupted(frame) and not nakSent then
12:            SendNAK(R_n)
13:            nakSent ← True
14:            Sleep()
15:        end if
16:        if seqNo != R_n and not nakSent then
17:            SendNAK(R_n)
18:            nakSent ← True
19:            if seqno in window and not Marked(seqno) then
20:                StoreFrame(seqno)
21:                Marked(seqno) ← True
22:                while Marked(R_n) do
23:                    DeliverData(R_n)
24:                    Purge(R_n)
25:                    R_n ← (R_n + 1) % 2^m
26:                    ackNeeded ← True
27:                end while
28:                if ackNeeded then
29:                    SendACK(R_n)
30:                    ackNeeded ← False
31:                    nakSent ← False
32:                end if
33:            end if
34:        end if
35:    end if
36: end while
```

PROGRAM:

Stop and Wait:-

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
int i,j=0,noframes,x,x1=10,x2;
printf("enter the number of frames\n");
scanf("%d",&noframes);
printf("\n number of frames is %d",noframes);
while(noframes>0)
{
printf("\nsending frame %d",i);
srand(x1++); //The srand() function sets the starting point for producing a series of
pseudo-random integers
x = rand()%10;
if(x%2 == 0)
{
for (x2=1; x2<2; x2++)
{
 printf("\nwaiting for %d seconds\n", x2);
 sleep(x2);
 printf("Missing Acknowledgement %d",i);
}
printf("\nsending frame %d",i);
srand(x1++);
x = rand()%10;
}
printf("\nack received for frame %d",j);
noframes-=1;
i++;
j++;
}
printf("\n end of stop and wait protocol");
}
```

OUTPUT:

```
mec@cc-2-11:~/saw_arq/stopandwait$ gcc stopandwait.c
mec@cc-2-11:~/saw_arq/stopandwait$ ./a.out
enter the number of frames
6

 number of frames is 6
sending frame 1
ack received for frame 1
sending frame 2
ack received for frame 2
sending frame 3

waiting for 1 seconds
Missing Acknowledgement 3
sending frame 3
ack received for frame 3
sending frame 4
ack received for frame 4
sending frame 5
ack received for frame 5
sending frame 6

waiting for 1 seconds
Missing Acknowledgement 6
sending frame 6
ack received for frame 6
 end of stop and wait protocolmec@cc-2-11:~/saw_arq/stopandwait$
```

Go Back N:-

Client

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<sys/time.h>
#include<sys/wait.h>
#include<string.h>
#include<unistd.h>
#include<arpa/inet.h>
int main() {
 int c_sock;
 c_sock = socket(AF_INET, SOCK_STREAM, 0);
 struct sockaddr_in client;
 memset(&client, 0, sizeof(client));
 client.sin_family = AF_INET;
```

```
 client.sin_port = htons(9009);
 client.sin_addr.s_addr = inet_addr("127.0.0.1");
  if(connect(c_sock, (struct sockaddr*)&client, sizeof(client)) == -1)
{
 printf("Connection failed");
 return 0;
 }
 printf("\n\tClient -with individual acknowledgement scheme\n\n");
 char msg1[50]="acknowledgement of :";
 char msg2[50];
 char buff[100];
 int flag=1,flg=1;
 for(int i=0;i<=9;i++) {
 flg=1;
 bzero(buff,sizeof(buff));
 bzero(msg2,sizeof(msg2));
 if(i==8&&flag==1){
 printf("here\n"); //simulating loss
 flag=0;
 read(c_sock,buff,sizeof(buff));
 }
 int n = read(c_sock, buff, sizeof(buff));
 if(buff[strlen(buff)-1]!=i+'0'){ //out of order
 printf("Discarded as out of order \n");
 i--;
 }
 else{
 printf("Message received from server : %s \t %d\n",buff,i);
 printf("Acknowledgement sent for message \n");
 strcpy(msg2,msg1);
 msg2[strlen(msg2)]=i+'0';
 write(c_sock,msg2, sizeof(msg2));
 }
 }
  close(c_sock);
 return 0;
}
```

Server

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<sys/time.h>
#include<netinet/in.h>
#include<string.h>
#include<unistd.h>
#include<arpa/inet.h>
#include<fcntl.h>
int main() {
 int s_sock, c_sock;
 s_sock = socket(AF_INET, SOCK_STREAM, 0);
 struct sockaddr_in server, other;
 memset(&server, 0, sizeof(server));
 memset(&other, 0, sizeof(other));
 server.sin_family = AF_INET;
 server.sin_port = htons(9009);
 server.sin_addr.s_addr = INADDR_ANY;
 socklen_t add;

 if(bind(s_sock, (struct sockaddr*)&server, sizeof(server)) == -1) {
 printf("Binding failed\n");
 return 0;
 }
 printf("\tServer Up\n Go back n (n=3) used to send 10 messages \n\n");
 listen(s_sock, 10);
 add = sizeof(other);
 c_sock = accept(s_sock, (struct sockaddr*)&other, &add);
 time_t t1,t2;
 char msg[50]="server message :";
 char buff[50];
 int flag=0;
fd_set set1,set2,set3;
 struct timeval timeout1,timeout2,timeout3;
 int rv1,rv2,rv3;
 int i=-1;
```

```
qq:
i=i+1;
bzero(buff,sizeof(buff));
char buff2[60];
bzero(buff2,sizeof(buff2));
strcpy(buff2,"server message :");
buff2[strlen(buff2)]=i+'0';
buff2[strlen(buff2)]='\0';
printf("Message sent to client :%s \n",buff2);
write(c_sock, buff2, sizeof(buff2));
usleep(1000);
i=i+1;
bzero(buff2,sizeof(buff2));
strcpy(buff2,msg);
buff2[strlen(msg)]=i+'0';
printf("Message sent to client :%s \n",buff2);
write(c_sock, buff2, sizeof(buff2));
i=i+1;
usleep(1000);
qqq:
bzero(buff2,sizeof(buff2));
strcpy(buff2,msg);
buff2[strlen(msg)]=i+'0';
printf("Message sent to client :%s \n",buff2);
write(c_sock, buff2, sizeof(buff2));
FD_ZERO(&set1);
FD_SET(c_sock, &set1);
timeout1.tv_sec = 2;
timeout1.tv_usec = 0;
rv1 = select(c_sock + 1, &set1, NULL, NULL, &timeout1);
if(rv1 == -1)
perror("select error ");
else if(rv1 == 0){
printf("Going back from %d:timeout \n",i);
i=i-3;
goto qq;}
else{
read(c_sock, buff, sizeof(buff));
```

```
printf("Message from Client: %s\n", buff);
i++;
if(i<=9)
goto qqq;
}
qq2:
FD_ZERO(&set2);
FD_SET(c_sock, &set2);
timeout2.tv_sec = 3;
timeout2.tv_usec = 0;
rv2 = select(c_sock + 1, &set2, NULL, NULL, &timeout2);
if(rv2 == -1)
perror("select error "); // an error accured
else if(rv2 == 0){
printf("Going back from %d:timeout on last 2\n",i-1);
i=i-2;
bzero(buff2,sizeof(buff2));
strcpy(buff2,msg);
buff2[strlen(buff2)]=i+'0';
write(c_sock, buff2, sizeof(buff2));
usleep(1000);
bzero(buff2,sizeof(buff2));
i++;
strcpy(buff2,msg);
buff2[strlen(buff2)]=i+'0';
write(c_sock, buff2, sizeof(buff2));
goto qq2;} // a timeout occured
else{
read(c_sock, buff, sizeof(buff));
printf("Message from Client: %s\n", buff);
bzero(buff,sizeof(buff));
read(c_sock, buff, sizeof(buff));
printf("Message from Client: %s\n", buff);
}
close(c_sock);
close(s_sock);
return 0;
}
```

OUTPUT:

```
mec@cc-2-11:~/saw_arq/gobackn$ ./server
        Server Up
 Go back n (n=3) used to send 10 messages

Message sent to client :server message :0
Message sent to client :server message :1
Message sent to client :server message :2
Message from Client: acknowledgement of :0
Message sent to client :server message :3
Message from Client: acknowledgement of :1
Message sent to client :server message :4
Message from Client: acknowledgement of :2
Message sent to client :server message :5
Message from Client: acknowledgement of :3
Message sent to client :server message :6
Going back from 6:timeout
Message sent to client :server message :4
Message sent to client :server message :5
Message sent to client :server message :6
Message from Client: acknowledgement of :4
Message sent to client :server message :7
Message from Client: acknowledgement of :5
Message sent to client :server message :8
Message from Client: acknowledgement of :6
Message sent to client :server message :9
Message from Client: acknowledgement of :7
Going back from 9:timeout on last 2
Message from Client: acknowledgement of :8
Message from Client: acknowledgement of :9
```

```
mec@cc-2-11:~/saw_arq/gobackn$ ./client

        Client -with individual acknowledgement scheme

Message received from server : server message :0
Acknowledgement sent for message
Message received from server : server message :1
Acknowledgement sent for message
Message received from server : server message :2
Acknowledgement sent for message
Message received from server : server message :3
Acknowledgement sent for message
Discarded as out of order
Discarded as out of order
Message received from server : server message :4
Acknowledgement sent for message
Message received from server : server message :5
Acknowledgement sent for message
Message received from server : server message :6
Acknowledgement sent for message
Message received from server : server message :7
Acknowledgement sent for message
here
Message received from server : server message :8
Acknowledgement sent for message
Message received from server : server message :9
Acknowledgement sent for message
```

Selective Repeat:-

Client

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

int isfaulty(){ //simulating corruption of message

int d=rand()%4;
return (d>2);

}
int main() {
 srand(time(0));
 int c_sock;
 c_sock = socket(AF_INET, SOCK_STREAM, 0);
 struct sockaddr_in client;
 memset(&client, 0, sizeof(client));
 client.sin_family = AF_INET;
 client.sin_port = htons(9009);
 client.sin_addr.s_addr = inet_addr("127.0.0.1");


 if(connect(c_sock, (struct sockaddr*)&client, sizeof(client)) == -1) {
 printf("Connection failed");
 return 0;
 }
 printf("\n\tClient -with individual acknowledgement scheme\n\n");
 char msg1[50]="acknowledgement of";
 char msg3[50]="negative ack ";
```

```
char msg2[50];
char buff[100];
int count=-1,flag=1;
while(count<8){
bzero(buff,sizeof(buff));
bzero(msg2,sizeof(msg2));
if(count==7&&flag==1){
printf("here\n"); //simulate loss
flag=0;
read(c_sock,buff,sizeof(buff));
continue;
}
int n = read(c_sock, buff, sizeof(buff));
char i=buff[strlen(buff)-1];
printf("Message received from server : %s \n",buff);
int isfault=isfaulty();
printf("corruption status : %d \n",isfault);
printf("Response/acknowledgement sent for message \n");
if(isfault)
strcpy(msg2,msg3);
else{
strcpy(msg2,msg1);
count++;}
msg2[strlen(msg2)]=i;
write(c_sock,msg2, sizeof(msg2));
}
```

Server

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

```
void rsendd(int ch,int c_sock){
char buff2[60];
 bzero(buff2,sizeof(buff2));
 strcpy(buff2,"reserver message :");
 buff2[strlen(buff2)]=(ch)+'0';
 buff2[strlen(buff2)]='\0';
 printf("Resending Message to client :%s \n",buff2);
 write(c_sock, buff2, sizeof(buff2));
 usleep(1000);
}
int main() {
 int s_sock, c_sock;
 s_sock = socket(AF_INET, SOCK_STREAM, 0);
 struct sockaddr_in server, other;
 memset(&server, 0, sizeof(server));
 memset(&other, 0, sizeof(other));
 server.sin_family = AF_INET;
 server.sin_port = htons(9009);
 server.sin_addr.s_addr = INADDR_ANY;
 socklen_t add;


 if(bind(s_sock, (struct sockaddr*)&server, sizeof(server)) == -1) {
 printf("Binding failed\n");
 return 0;
 }
 printf("\tServer Up\n Selective repeat scheme\n\n");
 listen(s_sock, 10);
 add = sizeof(other);
 c_sock = accept(s_sock, (struct sockaddr*)&other, &add);
 time_t t1,t2;
 char msg[50]="server message :";
 char buff[50];
 int flag=0;

fd_set set1,set2,set3;
 struct timeval timeout1,timeout2,timeout3;
```

```
 int rv1,rv2,rv3;

int tot=0;
int ok[20];
memset(ok,0,sizeof(ok));

while(tot<9){
int toti=tot;
for(int j=(0+toti);j<(3+toti);j++){
bzero(buff,sizeof(buff));
 char buff2[60];
 bzero(buff2,sizeof(buff2));
 strcpy(buff2,"server message :");
 buff2[strlen(buff2)]=(j)+'0';
 buff2[strlen(buff2)]='\0';
 printf("Message sent to client :%s \t%d\t%d\n",buff2,tot,j);
 write(c_sock, buff2, sizeof(buff2));
 usleep(1000);
}
for(int k=0+toti;k<(toti+3);k++){
 qq:
 FD_ZERO(&set1);
 FD_SET(c_sock, &set1);
 timeout1.tv_sec = 2;
 timeout1.tv_usec = 0;

 rv1 = select(c_sock + 1, &set1, NULL, NULL, &timeout1);
 if(rv1 == -1)
 perror("select error ");
 else if(rv1 == 0){
 printf("Timeout for message :%d \n",k);
 rsendd(k,c_sock);
 goto qq;} // a timeout occured
 else{
 read(c_sock, buff, sizeof(buff));
 printf("Message from Client: %s\n", buff);
 if(buff[0]=='n'){
 printf(" corrupt message acknowledgement (msg %d) \n",buff[strlen(buff)-1]-'0');
```

```
 rsendd((buff[strlen(buff)-1]-'0'),c_sock);
 goto qq;}
 else
 tot++;
 }
}
}
 close(c_sock);
 close(s_sock);
 return 0;
}
```

OUTPUT:

```
mec@cc-2-11:~/saw_arq/selectiverepeat$ ./c

        Client -with individual acknowledgement scheme

Message received from server : server message :0
corruption status : 0
Response/acknowledgement sent for message
Message received from server : server message :1
corruption status : 0
Response/acknowledgement sent for message
Message received from server : server message :2
corruption status : 0
Response/acknowledgement sent for message
Message received from server : server message :3
corruption status : 0
Response/acknowledgement sent for message
Message received from server : server message :4
corruption status : 1
Response/acknowledgement sent for message
Message received from server : server message :5
corruption status : 0
Response/acknowledgement sent for message
Message received from server : reserver message :4
corruption status : 1
Response/acknowledgement sent for message
Message received from server : reserver message :4
corruption status : 0
Response/acknowledgement sent for message
Message received from server : server message :6
corruption status : 0
Response/acknowledgement sent for message
Message received from server : server message :7
corruption status : 1
Response/acknowledgement sent for message
Message received from server : server message :8
corruption status : 0
Response/acknowledgement sent for message
here
Message received from server : reserver message :8
corruption status : 0
Response/acknowledgement sent for message
```

```
mec@cc-2-11:~/saw_arq/selectiverepeat$ ./s
        Server Up
 Selective repeat scheme

Message sent to client :server message :0       0       0
Message sent to client :server message :1       0       1
Message sent to client :server message :2       0       2
Message from Client: acknowledgement of0
Message from Client: acknowledgement of1
Message from Client: acknowledgement of2
Message sent to client :server message :3       3       3
Message sent to client :server message :4       3       4
Message sent to client :server message :5       3       5
Message from Client: acknowledgement of3
Message from Client: negative ack 4
 corrupt message acknowledgement (msg 4)
Resending Message to client :reserver message :4
Message from Client: acknowledgement of5
Message from Client: negative ack 4
 corrupt message acknowledgement (msg 4)
Resending Message to client :reserver message :4
Message from Client: acknowledgement of4
Message sent to client :server message :6       6       6
Message sent to client :server message :7       6       7
Message sent to client :server message :8       6       8
Message from Client: acknowledgement of6
Message from Client: negative ack 7
 corrupt message acknowledgement (msg 7)
Resending Message to client :reserver message :7
Message from Client: acknowledgement of8
Timeout for message :8
Resending Message to client :reserver message :8
Message from Client: acknowledgement of8
```

RESULT:

The experiment was executed successfully.

EXPERIMENT 6

AIM:

To implement and simulate algorithm for Distance Vector Routing protocol.

THEORY:

Distance Vector Routing protocol is a dynamic routing protocol. With this protocol, every router in the network creates a routing table which helps them in determining the shortest path through the network. All the routers in the network are aware of every other router in the network and they keep on updating their routing table periodically. This protocol uses the principle of Bellman-Ford's algorithm.

Distance-vector routing protocols measure the distance by the number of routers a packet has to pass; one router counts as one hop. To determine the best route across a network, routers using a distance-vector protocol exchange information with one another, usually routing tables plus hop counts for destination networks and possibly other traffic information.

ALGORITHM:

1. Each router prepares its routing table. By their local knowledge each router knows about all the routers present in the network and distance to its neighboring router.
2. Each router exchanges its distance vector with its neighboring routers.
3. Each router prepares a new routing table using the distance vectors it has obtained from its neighbors.
4. This step is repeated for n-2 times if there are n routers in the network.
5. After this, the routing table converges and becomes stable.

PROGRAM:

```
/*
Distance Vector Routing in this program is implemented using Bellman Ford Algorithm:-
*/
#include<stdio.h>
struct node
{
   unsigned dist[20];
   unsigned from[20];
}rt[10];
int main()
```

```
{
   int costmat[20][20];
   int nodes,i,j,k,count=0;
   printf("\nEnter the number of nodes : ");
   scanf("%d",&nodes);//Enter the nodes
   printf("\nEnter the cost matrix :\n");
   for(i=0;i<nodes;i++)
   {
      for(j=0;j<nodes;j++)
      {
         scanf("%d",&costmat[i][j]);
         costmat[i][i]=0;
         rt[i].dist[j]=costmat[i][j];//initialise the distance equal to cost matrix
         rt[i].from[j]=j;
      }
   }
      do
      {
         count=0;
         for(i=0;i<nodes;i++)//We choose arbitary vertex k and we calculate the direct
distance from the node i to k using the cost matrix
         //and add the distance from k to node j
         for(j=0;j<nodes;j++)
         for(k=0;k<nodes;k++)
            if(rt[i].dist[j]>costmat[i][k]+rt[k].dist[j])
            {//We calculate the minimum distance
               rt[i].dist[j]=rt[i].dist[k]+rt[k].dist[j];
               rt[i].from[j]=k;
               count++;
            }
      }while(count!=0);
      for(i=0;i<nodes;i++)
      {
         printf("\n\n For router %d\n",i+1);
         for(j=0;j<nodes;j++)
         {
            printf("\t\nnode %d via %d Distance %d ",j+1,rt[i].from[j]+1,rt[i].dist[j]);
         }
      }
   printf("\n\n");
}
return 0;
```

OUTPUT:

```
mec@cc-3-10:~/CS6A/$ nano dvr.c
mec@cc-3-10:~/CS6A/$ gcc dvr.c
mec@cc-3-10:~/CS6A/$ ./a.out

 Enter the number of nodes : 3

 Enter the cost matrix :
 0 5 1 5 0 2 1 2 0

 Routing table of Node 1
 Node    Cost    Next hop
   1      0          1
   2      3          3
   3      1          3

 Routing table of Node 2
 Node    Cost    Next hop
   1      3          3
   2      0          2
   3      2          3

 Routing table of Node 3
 Node    Cost    Next hop
   1      1          1
   2      2          2
   3      0          3
```

RESULT:

The experiment was executed successfully.