**PROGRAM:**

AGENT 1

#include<stdio.h>

#include<sys/types.h>

#include<netinet/in.h>

#include<string.h>

int main()

{

int i,sd,sd2,nsd,clilen,sport,len;

charsendmsg[20],recvmsg[100];

char oid[5][10]={"client1","client2","client3","cleint4","client5"};

charwsize[5][5]={"5","10","15","3","6"};

structsockaddr\_inservaddr,cliaddr;

printf("I'm the Agent - TCP Connection\n");

printf("\nEnter the Server port");

printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

scanf("%d",&sport);

sd=socket(AF\_INET,SOCK\_STREAM,0);

if(sd<0)

printf("Can't Create \n"); else

printf("Socket is Created\n");

servaddr.sin\_family=AF\_INET;

servaddr.sin\_addr.s\_addr=htonl(INADDR\_ANY);

servaddr.sin\_port=htons(sport);

sd2=bind(sd,(structsockaddr\*)&servaddr,sizeof(servaddr));

if(sd2<0)

printf(" Can't Bind\n"); else

printf("\n Binded\n");

listen(sd,5);

clilen=sizeof(cliaddr);

nsd=accept(sd,(structsockaddr\*)&cliaddr,&clilen);

if(nsd<0)

printf("Can't Accept\n"); else

printf("Accepted\n"); recv(nsd,recvmsg,100,0);

for(i=0;i<5;i++)

{

if(strcmp(recvmsg,oid[i])==0)

send(nsd,wsize[i],100,0);

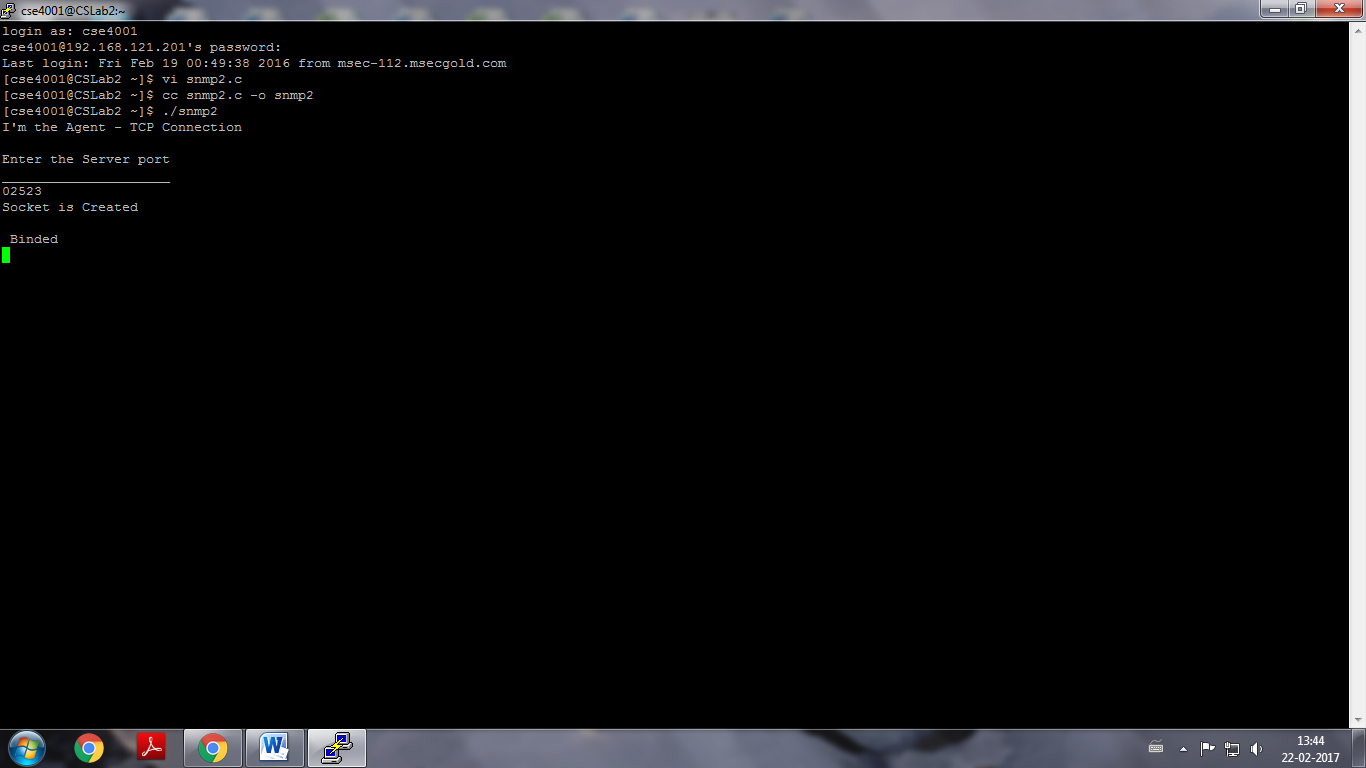
break;

}

return 0;

}

**OUTPUT:**



**PROGRAM:**

AGENT 2

#include<stdio.h>

#include<sys/types.h>

#include<netinet/in.h>

#include<string.h>

int main()

{

int i,sd,sd2,nsd,clilen,sport,len;

charendmsg[20],recvmsg[100];

charoid[5][10]={"System1","System2","System3","System4","System5"};

char mdate[5][15]={"1-10-095","10-03-8","14.03.81","11.07.07","17.12.77"};

char time[5][15]={"9am","10pm","11am","12.30pm","11.30am"};

structsockaddr\_inservaddr,cliaddr;

printf("Enter the Server port");

printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

scanf("%d",&sport);

sd=socket(AF\_INET,SOCK\_STREAM,0) ;

if(sd<0)

printf("Can't Create \n");

else

printf("Socket is Created\n");

servaddr.sin\_family=AF\_INET;

servaddr.sin\_addr.s\_addr=htonl(INADDR\_ANY);

servaddr.sin\_port=htons(sport);

sd2=bind(sd,(structsockaddr\*)&servaddr,sizeof(servaddr));

if(sd2<0)

printf(" Can't Bind\n");

else

printf("\n Binded\n");

listen(sd,5);

clilen=sizeof(cliaddr);

nsd=accept(sd,(structsockaddr\*)&cliaddr,&clilen);

if(nsd<0)

printf("Can't Accept\n");

else

printf("Accepted\n");

recv(nsd,recvmsg,100,0);

for(i=0;i<5;i++)

{

if(strcmp(recvmsg,oid[i])==0)

{

send(nsd,mdate[i],100,0);

send(nsd,time[i],100,0);

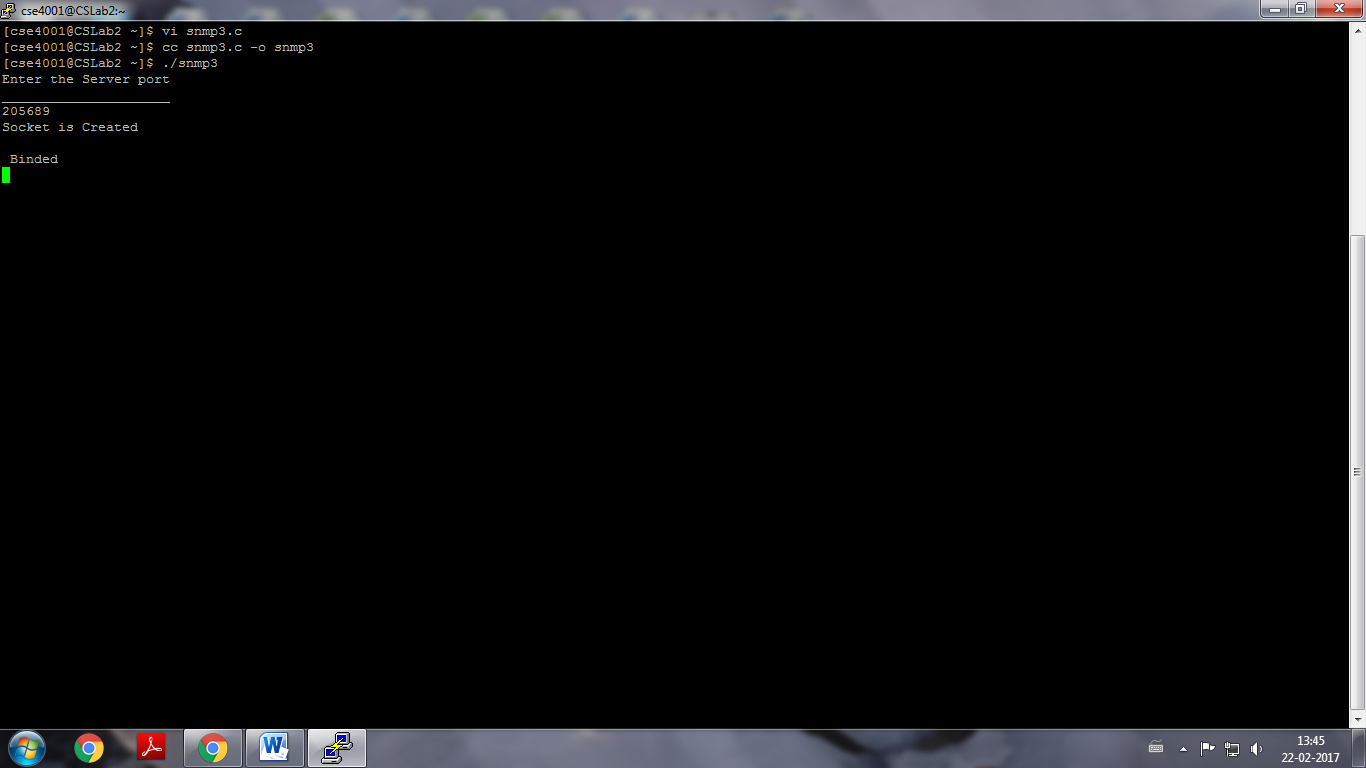
}

break;

}

}

**OUTPUT:**



**PROGRAM:**

MANAGER

#include<stdio.h>

#include<sys/types.h>

#include<netinet/in.h>

#include<strings.h>

int main()

{

intcsd,cport,len,i;

char sendmsg[20],rcvmsg[100],rmsg[100],oid[100];

structsockaddr\_inservaddr;

printf("Enter the port \n");

scanf("%d",&cport);

csd=socket(AF\_INET,SOCK\_STREAM,0) ;

if(csd<0)

printf("Can't Create \n");

else

printf("Scocket is Created\n");

servaddr.sin\_family=AF\_INET;

servaddr.sin\_addr.s\_addr=htonl(INADDR\_ANY);

servaddr.sin\_port=htons(cport);

if(connect(csd,(structsockaddr\*)&servaddr,sizeof(servaddr))<0)

printf("Can't Connect\n");

else

printf("Connected\n");

printf("\n 1.TCP Connection\n");

printf("\n 2. System \n");

printf("Enter the number for the type of informtion needed....\n");

scanf("%d",&i);

if(i==1)

{

printf("Enter the Object ID for Client\n");

scanf("%s",oid);

send(csd,oid,100,0);

recv(csd,rmsg,100,0);

printf("\n The window size of %s is %s \n",oid,rmsg);

}

else

{

printf("\nEnter the Object ID for the System\n");

scanf("%s",oid);

send(csd,oid,100,0);

recv(csd,rmsg,100,0) ;

printf("\nThe Manufacturing date for %s is %s",oid,rmsg);

bzero(rmsg,100);

recv(csd,rmsg,100,0);

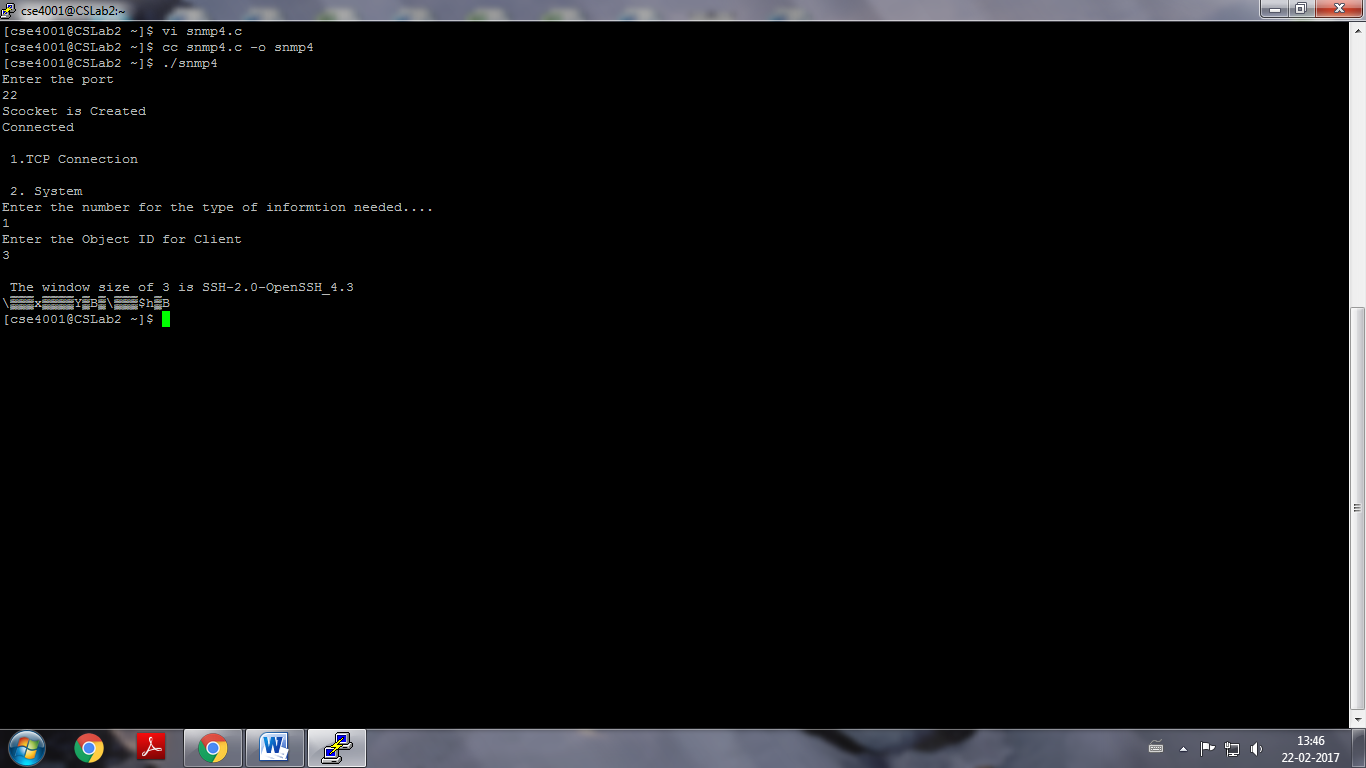
printf("\n The time of last utilization for %s is %s \n",oid,rmsg);

}

return 0;

}

**OUTPUT:**



**PROGRAM:**

**LINK STATE**

set ns [new Simulator]

setnf [open out.nam w]

$ns namtrace-all $nf

settr [open out.tr w]

$ns trace-all $tr

proc finish {} {

globalnf ns tr

$ns flush-trace

close $tr

execnamout.nam&

exit 0

}

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]

$ns duplex-link $n0 $n1 10Mb 10ms DropTail

$ns duplex-link $n1 $n3 10Mb 10ms DropTail

$ns duplex-link $n2 $n1 10Mb 10ms DropTail

$ns duplex-link-op $n0 $n1 orient right-down

$ns duplex-link-op $n1 $n3 orient right

$ns duplex-link-op $n2 $n1 orient right-up

settcp [new Agent/TCP]

$ns attach-agent $n0 $tcp

set ftp [new Application/FTP]

$ftp attach-agent $tcp

set sink [new Agent/TCPSink]

$ns attach-agent $n3 $sink

setudp [new Agent/UDP]

$ns attach-agent $n2 $udp

setcbr [new Application/Traffic/CBR]

$cbr attach-agent $udp

set null [new Agent/Null]

$ns attach-agent $n3 $null

$ns connect $tcp $sink

$ns connect $udp $null

$ns rtmodel-at 1.0 down $n1 $n3

$ns rtmodel-at 2.0 up $n1 $n3

$ns rtproto LS

$ns at 0.0 "$ftp start"

$ns at 0.0 "$cbr start"

$ns at 5.0 "finish"

$ns run

**FLOODING**

set MESSAGE\_PORT 42

# parameters for topology generator

setgroup\_size 7

setnum\_groups 5

setnum\_nodes [expr $group\_size \* $num\_groups]

set ns [new Simulator]

set f [open flooding.tr w]

$ns trace-all $f

setnf [open flooding.nam w]

$ns namtrace-all $nf

# subclass Agent/MessagePassing to make it do flooding

Class Agent/MessagePassing/Flooding -superclass Agent/MessagePassing

Agent/MessagePassing/Flooding instprocsend\_message {size msgidmsg} {

$self instvarmessages\_seen node\_

global ns MESSAGE\_PORT

$ns trace-annotate "Node [$node\_ node-addr] is sending {$msgid:$msg}"

lappendmessages\_seen $msgid

$self send\_to\_neighbors -1 $MESSAGE\_PORT $size "$msgid:$msg"

}

Agent/MessagePassing/Flooding instprocsend\_to\_neighbors {skip port size data} {

$self instvar node\_

foreach x [$node\_ neighbors] {

setaddr [$x set address\_]

if {$addr != $skip} {

$self sendto $size $data $addr $port

}

}

}

Agent/MessagePassing/Flooding instprocrecv {source sport size data} {

$self instvarmessages\_seen node\_

global ns

# extract message ID from message

setmessage\_id [lindex [split $data ":"] 0]

if {[lsearch $messages\_seen $message\_id] == -1} {

lappendmessages\_seen $message\_id

$ns trace-annotate "Node [$node\_ node-addr] received {$data}"

$self send\_to\_neighbors $source $sport $size $data

} else {

$ns trace-annotate "Node [$node\_ node-addr] received redundant copy of message #$message\_id"

}

}

## Topology Generator

# create a bunch of nodes

for {set i 0} {$i< $num\_nodes} {incri} {

set n($i) [$ns node]

}

# create links between the nodes

for {set g 0} {$g < $num\_groups} {incr g} {

for {set i 0} {$i< $group\_size} {incri} {

$ns duplex-link $n([expr $g\*$group\_size+$i]) $n([expr $g\*$group\_size+($i+1)%$group\_size]) 2Mb 15ms DropTail

}

$ns duplex-link $n([expr $g\*$group\_size]) $n([expr (($g+1)%$num\_groups)\*$group\_size+2]) 2Mb 15ms DropTail

if {$g%2} {

$ns duplex-link $n([expr $g\*$group\_size+3]) $n([expr (($g+3)%$num\_groups)\*$group\_size+1]) 2Mb 15ms DropTail

}

}

# attach a new Agent/MessagePassing/Flooding to each node on port $MESSAGE\_PORT

for {set i 0} {$i< $num\_nodes} {incri} {

set a($i) [new Agent/MessagePassing/Flooding]

$n($i) attach $a($i) $MESSAGE\_PORT

$a($i) set messages\_seen {}

}

# now set up some events

$ns at 0.2 "$a(5) send\_message 900 1 {first message}"

$ns at 0.5 "$a(17) send\_message 700 2 {another one}"

$ns at 1.0 "$a(24) send\_message 500 abc {yet another one}"

$ns at 2.0 "finish"

proc finish {} {

global ns f nf

$ns flush-trace

close $f

close $nf

puts "running nam..."

execnamflooding.nam&

exit 0}

$ns run

**DISTANCE VECTOR ROUTING**

set ns [new Simulator]

setnf [open out.nam w]

$ns namtrace-all $nf

settr [open out.tr w]

$ns trace-all $tr

proc finish {} {

globalnf ns tr

$ns flush-trace

close $tr

execnamout.nam&

exit 0

}

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]

$ns duplex-link $n0 $n1 10Mb 10ms DropTail

$ns duplex-link $n1 $n3 10Mb 10ms DropTail

$ns duplex-link $n2 $n1 10Mb 10ms DropTail

$ns duplex-link-op $n0 $n1 orient right-down

$ns duplex-link-op $n1 $n3 orient right

$ns duplex-link-op $n2 $n1 orient right-up

settcp [new Agent/TCP]

$ns attach-agent $n0 $tcp

set ftp [new Application/FTP]

$ftp attach-agent $tcp

set sink [new Agent/TCPSink]

$ns attach-agent $n3 $sink

setudp [new Agent/UDP]

$ns attach-agent $n2 $udp

setcbr [new Application/Traffic/CBR]

$cbr attach-agent $udp

set null [new Agent/Null]

$ns attach-agent $n3 $null

$ns connect $tcp $sink

$ns connect $udp $null

$ns rtmodel-at 1.0 down $n1 $n3

$ns rtmodel-at 2.0 up $n1 $n3

$ns rtproto DV

$ns at 0.0 "$ftp start"

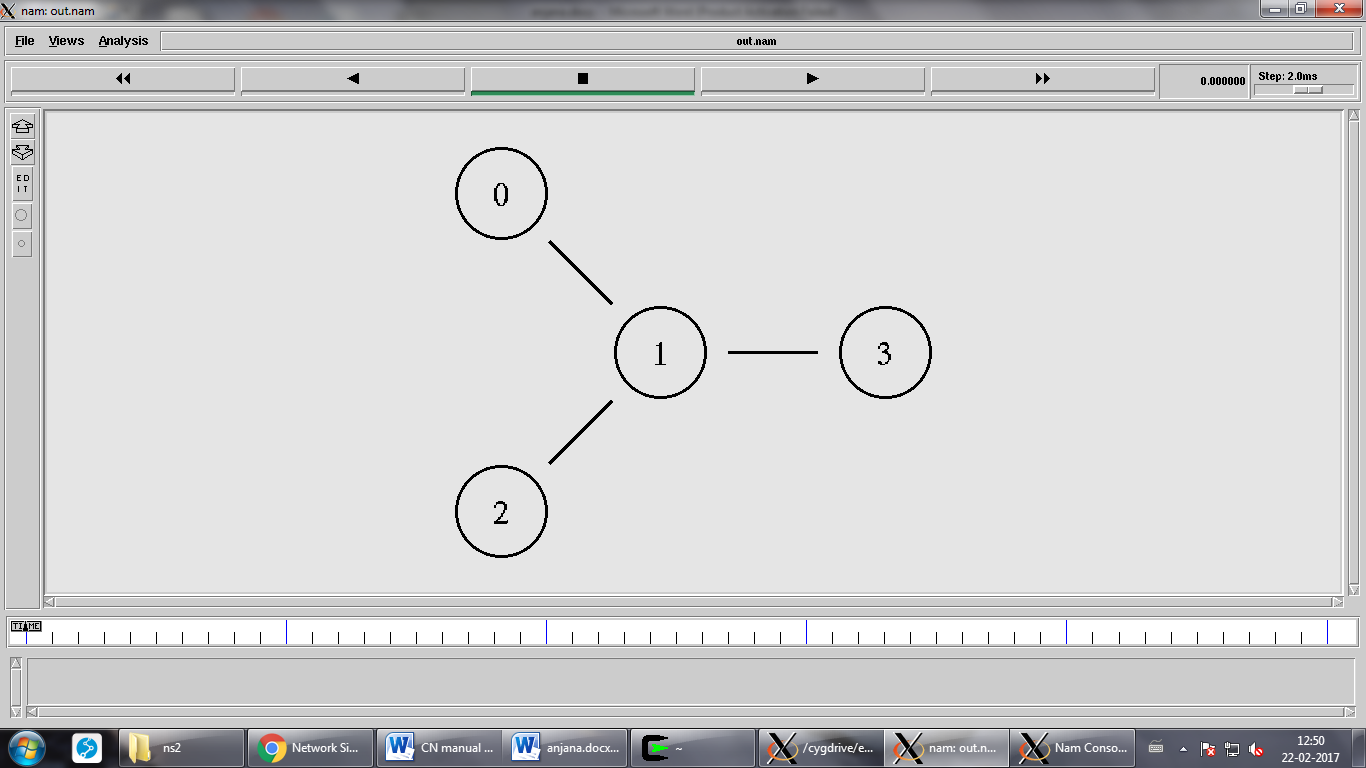
$ns at 0.0 "$cbr start"

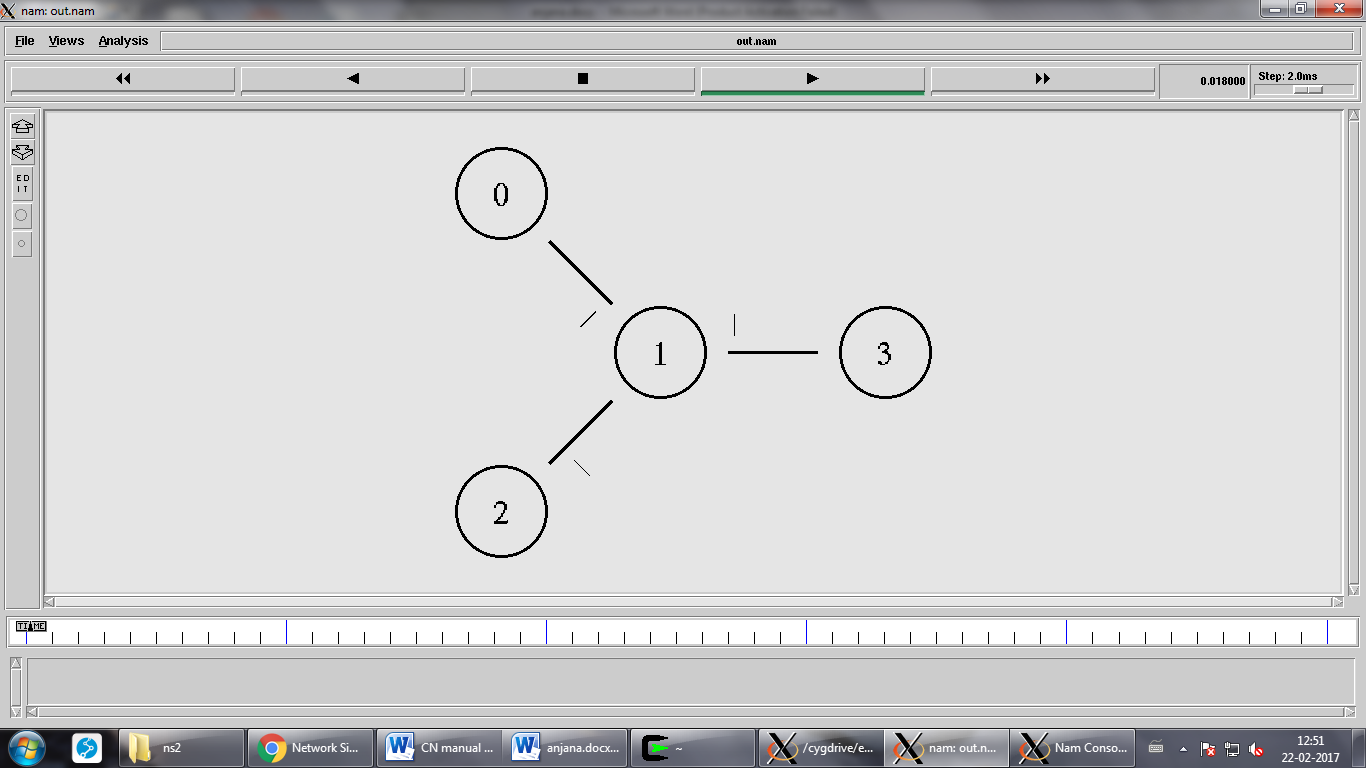
$ns at 5.0 "finish"

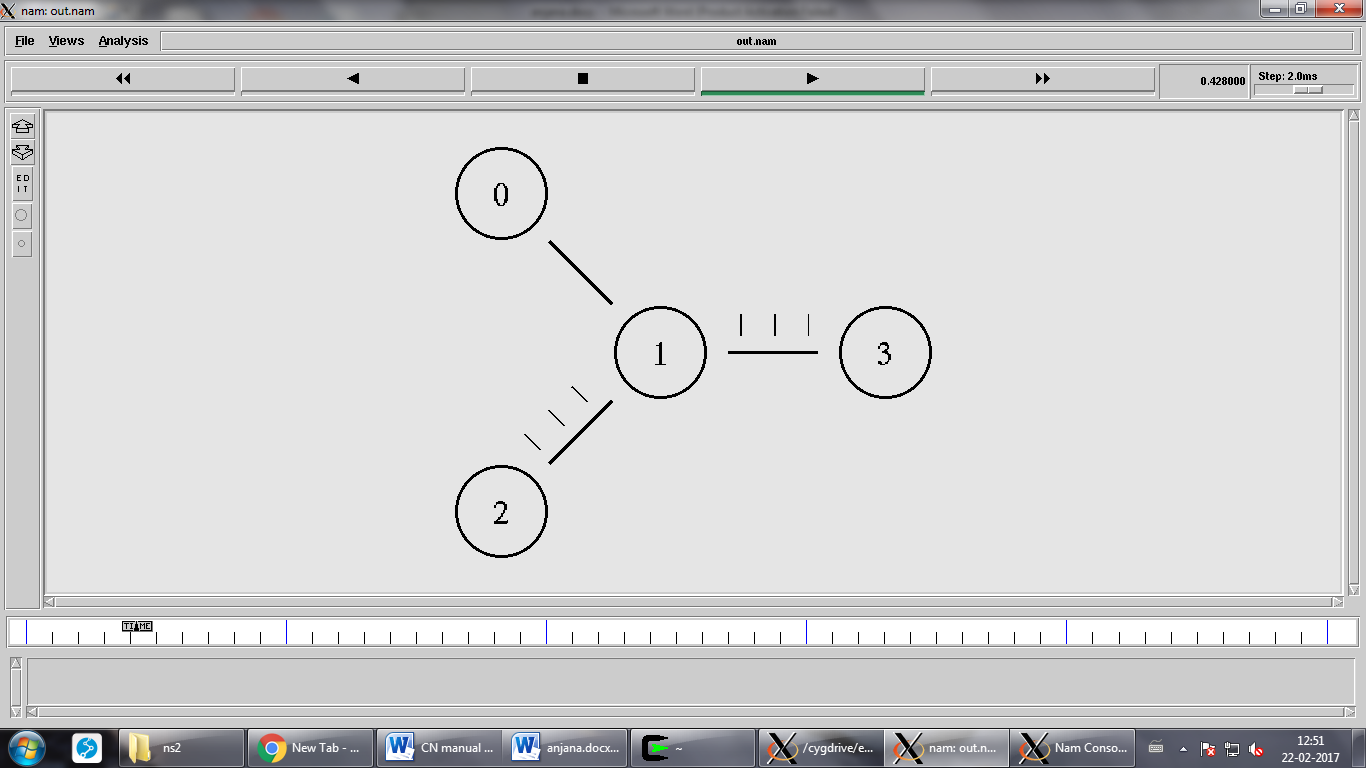
$ns run

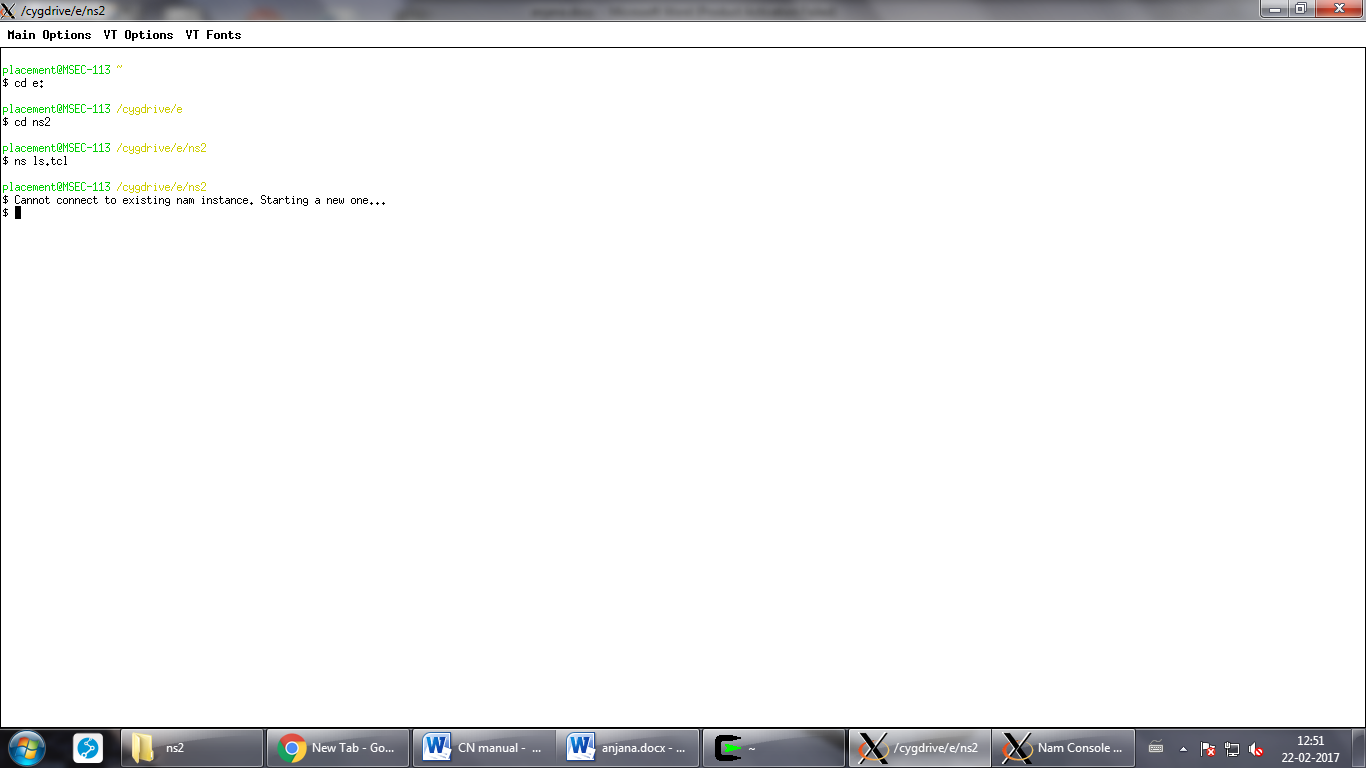
**OUTPUT:**

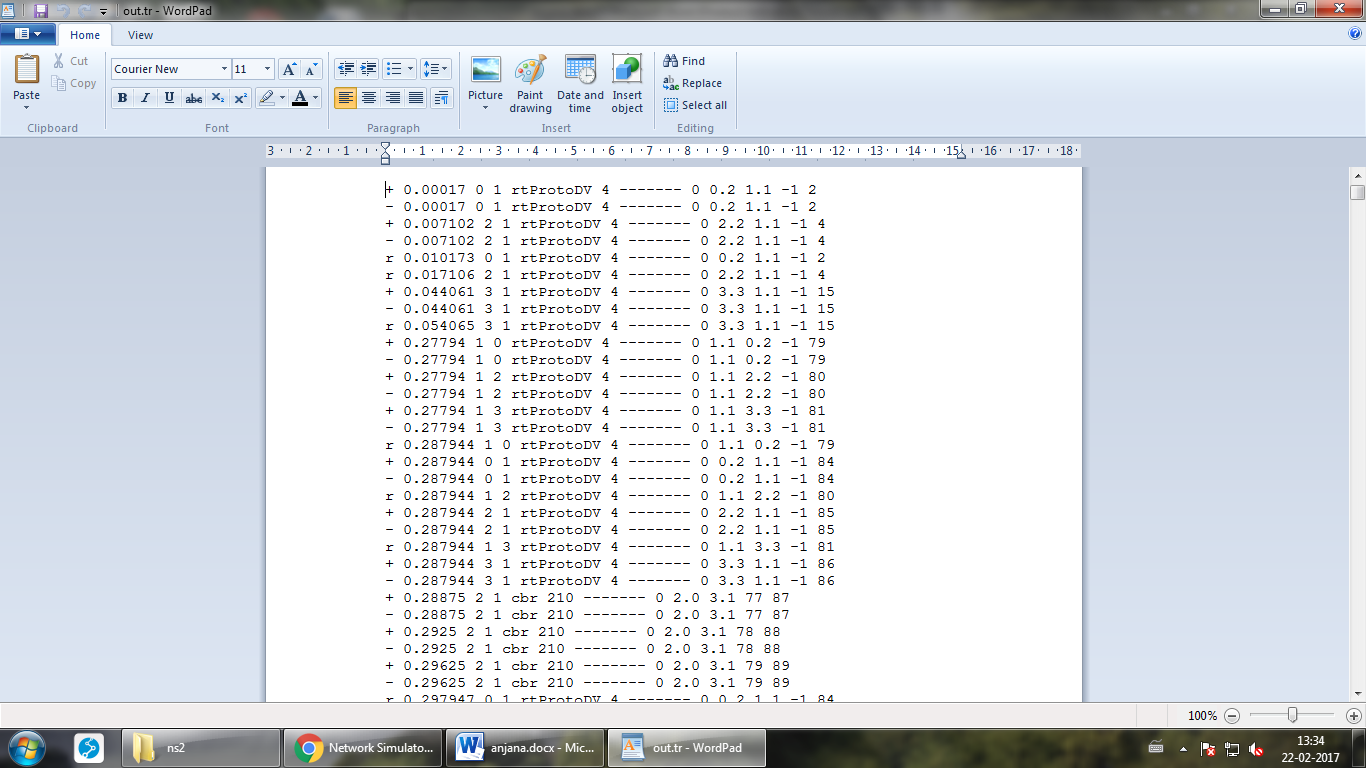
**LINK STATE**



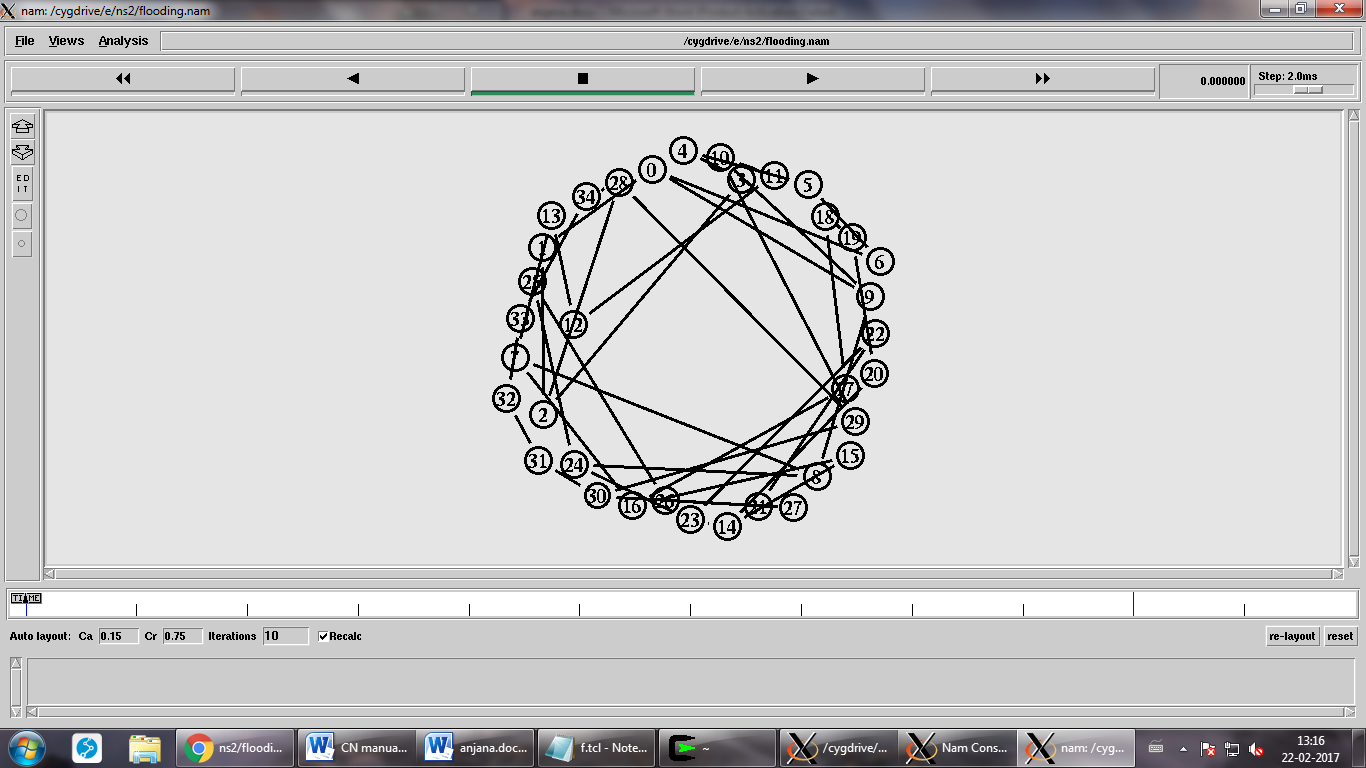


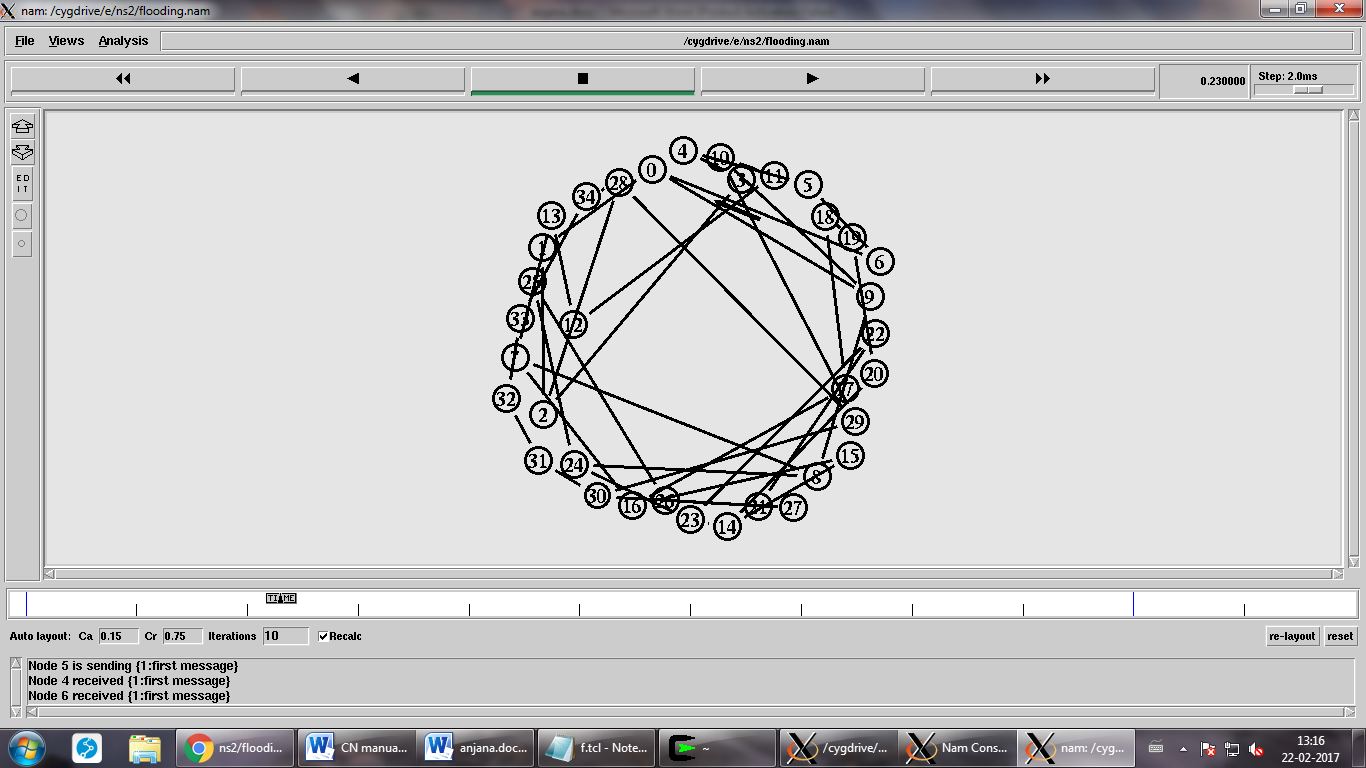


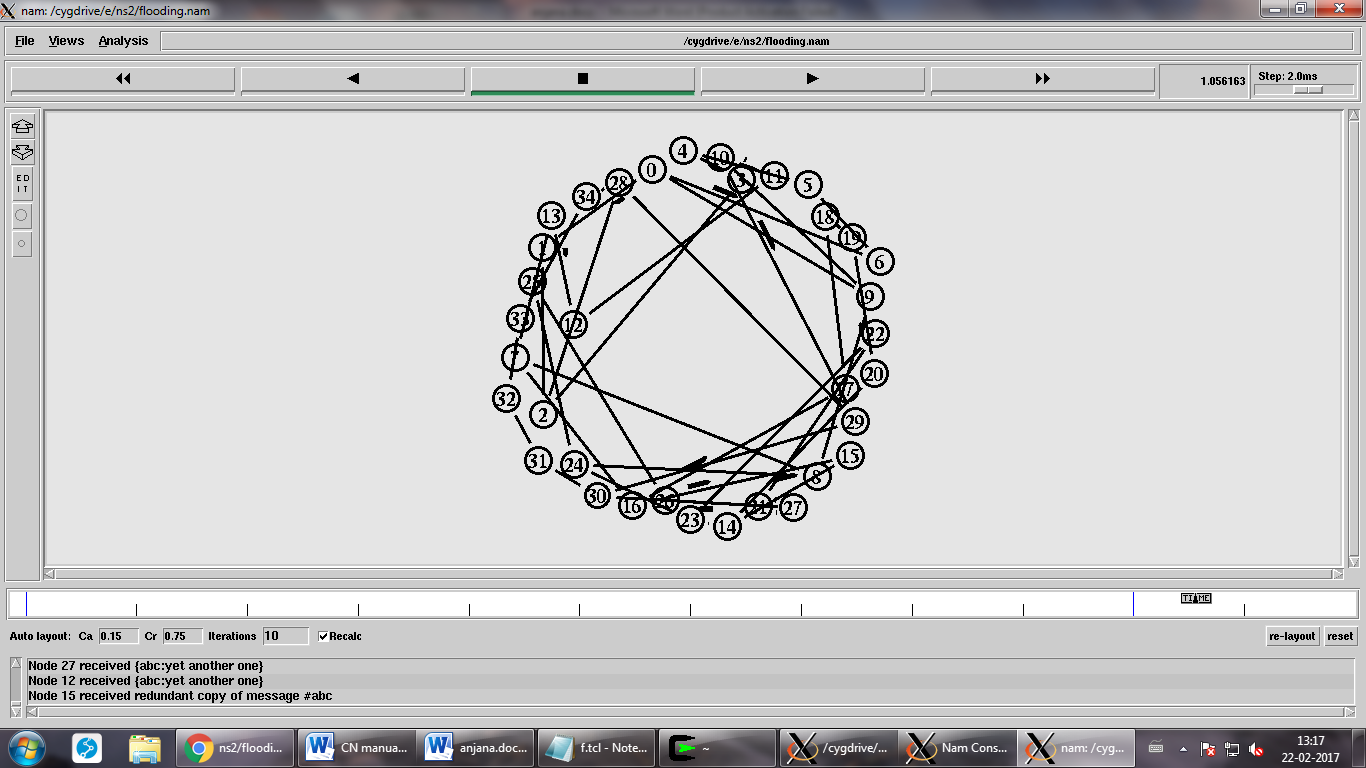


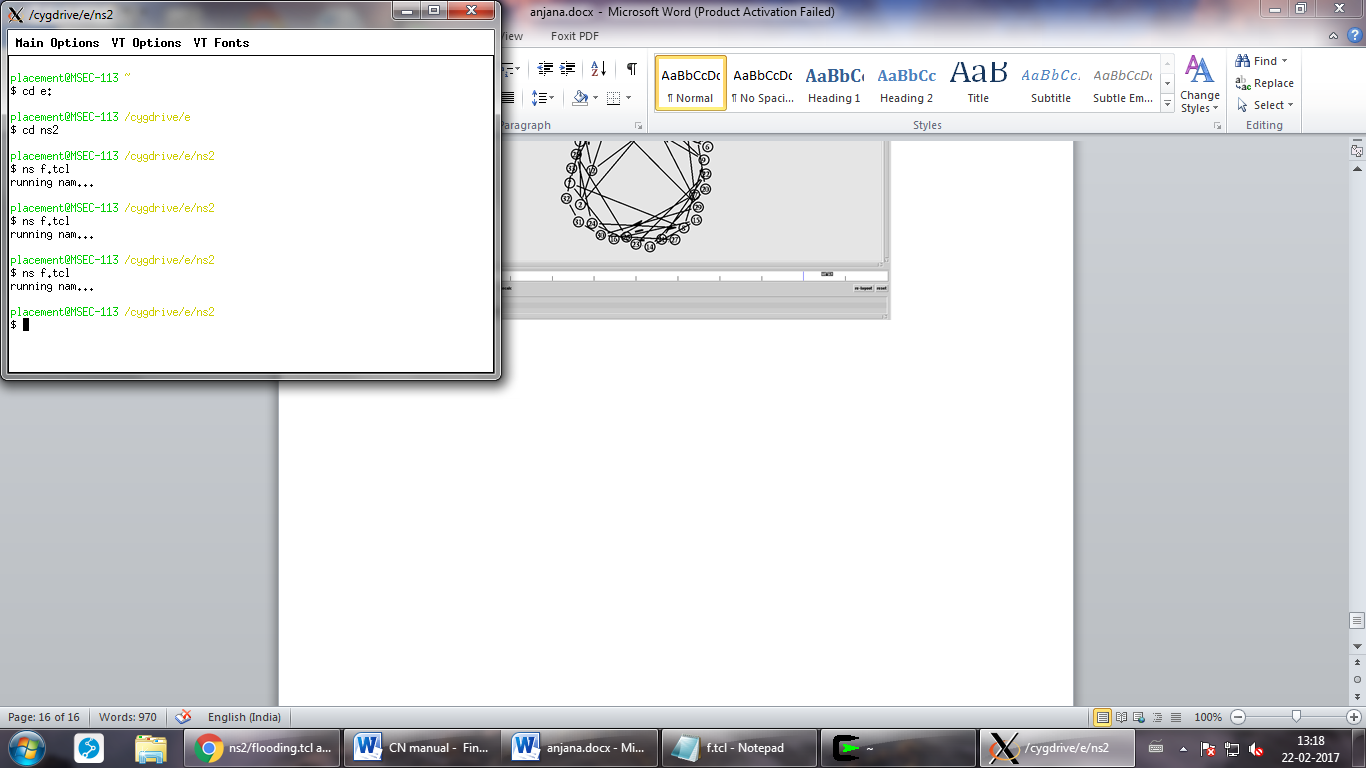


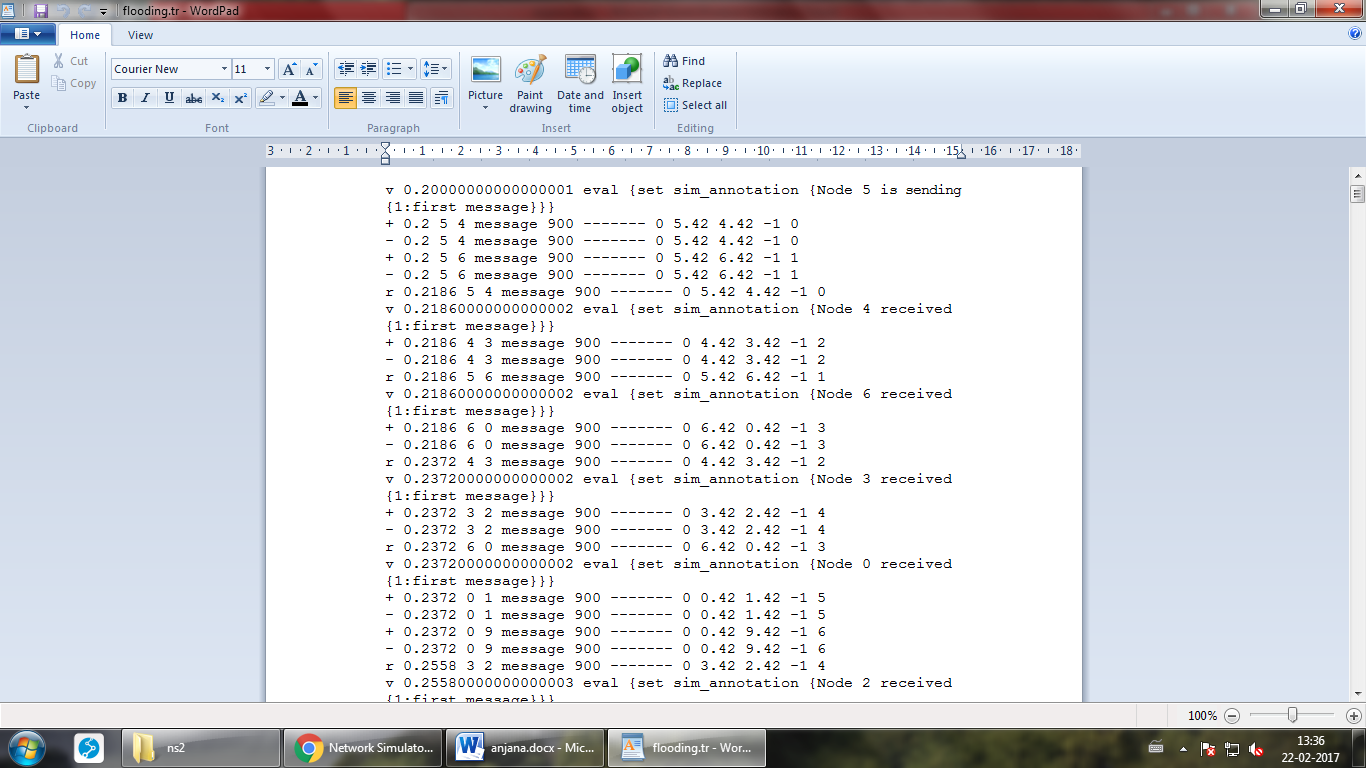
**FLOODING**



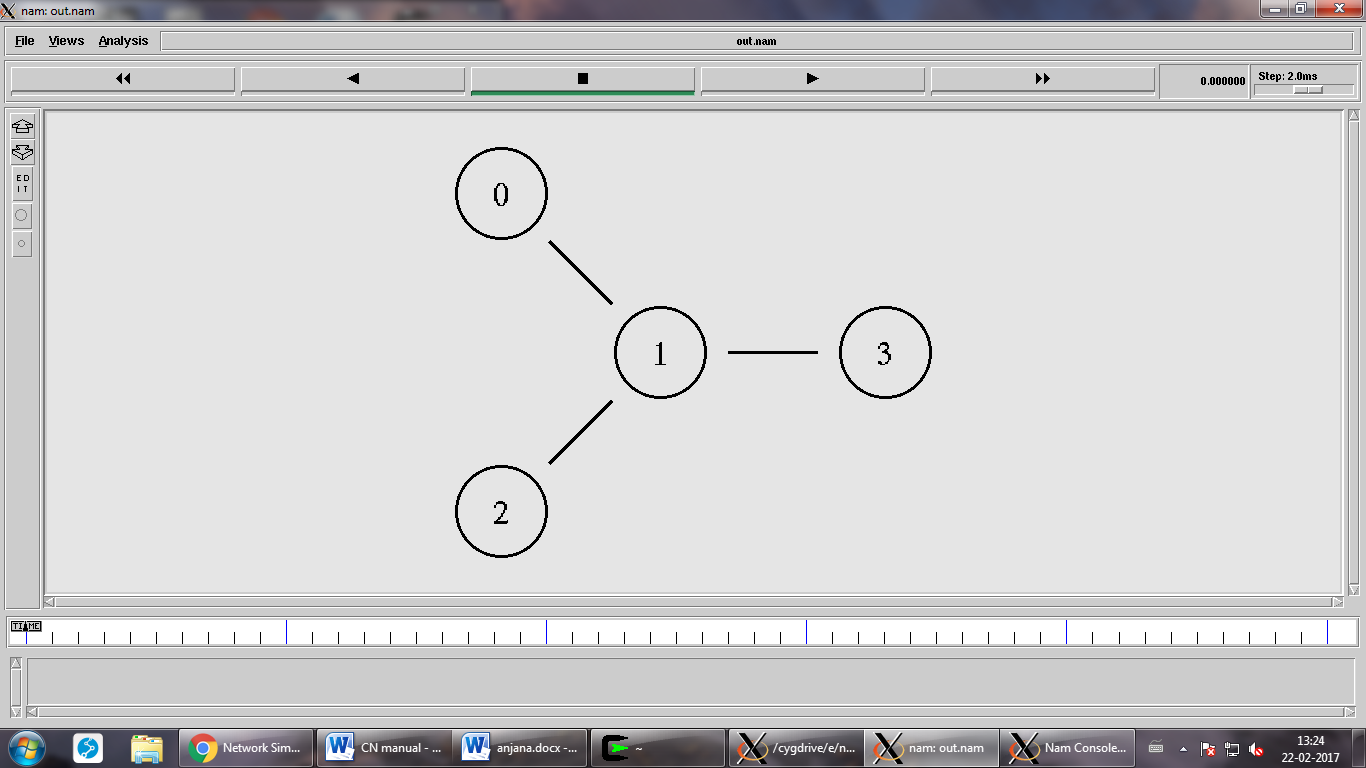


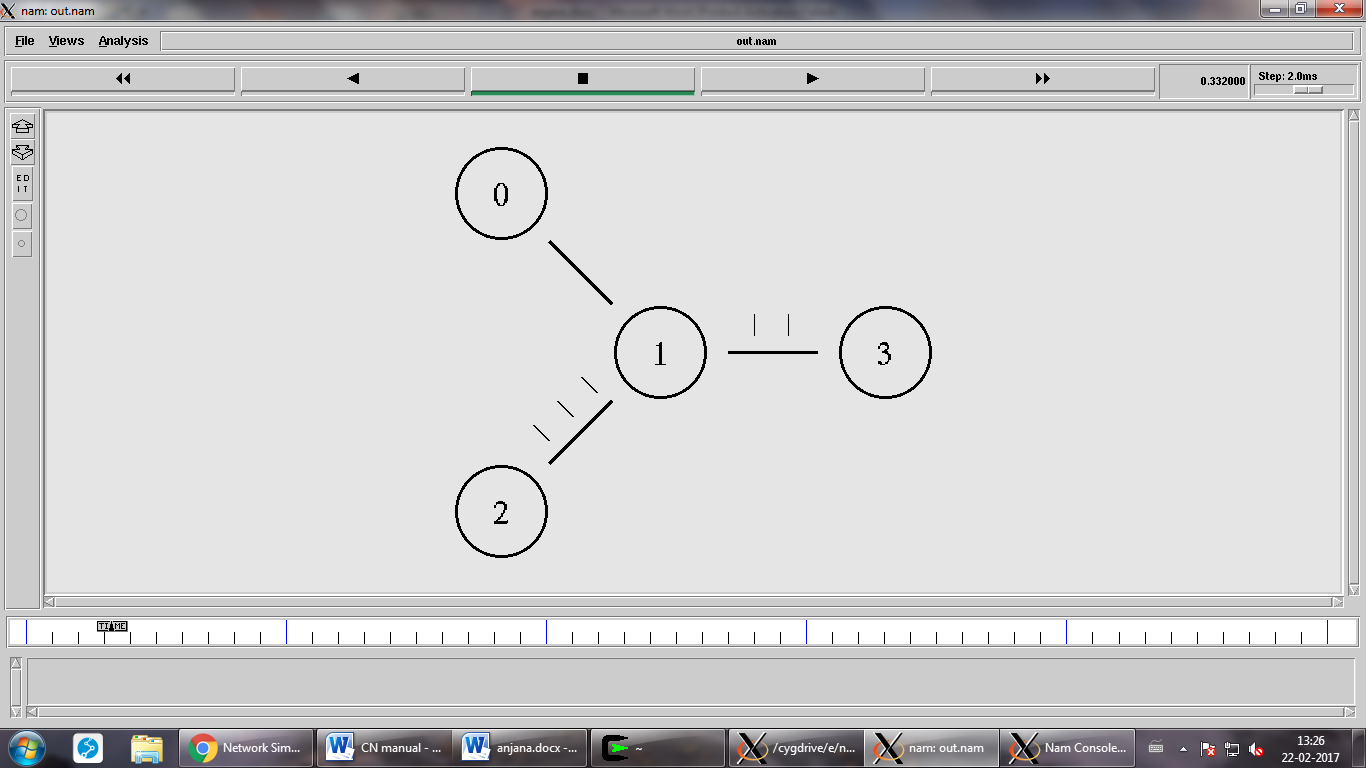


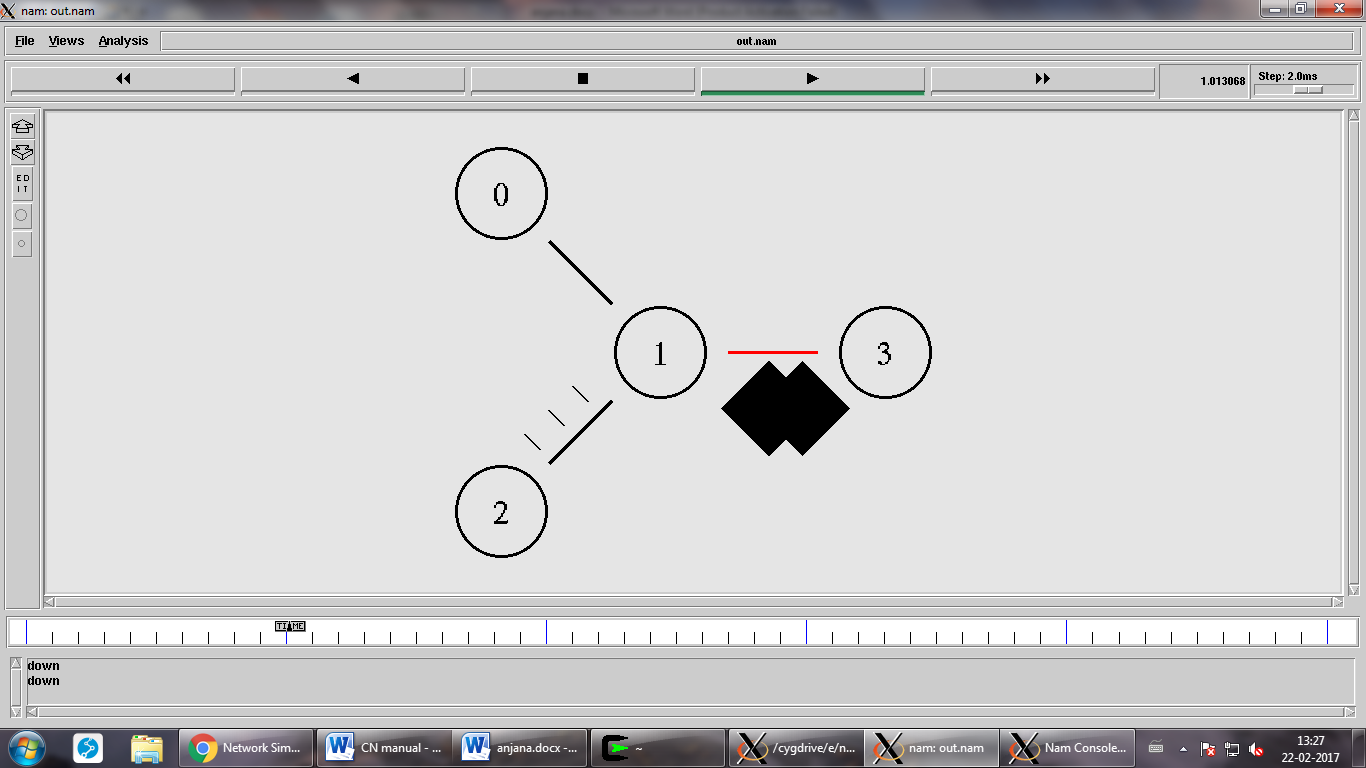


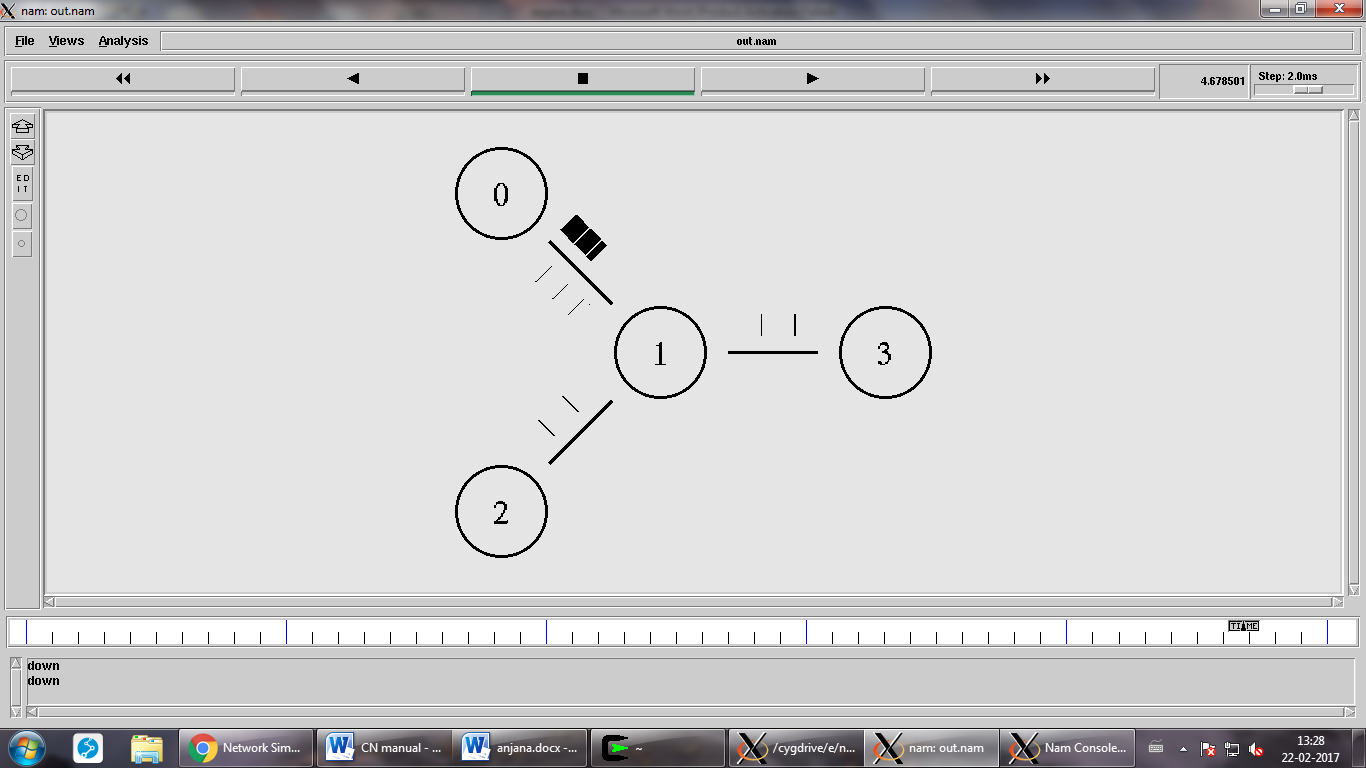


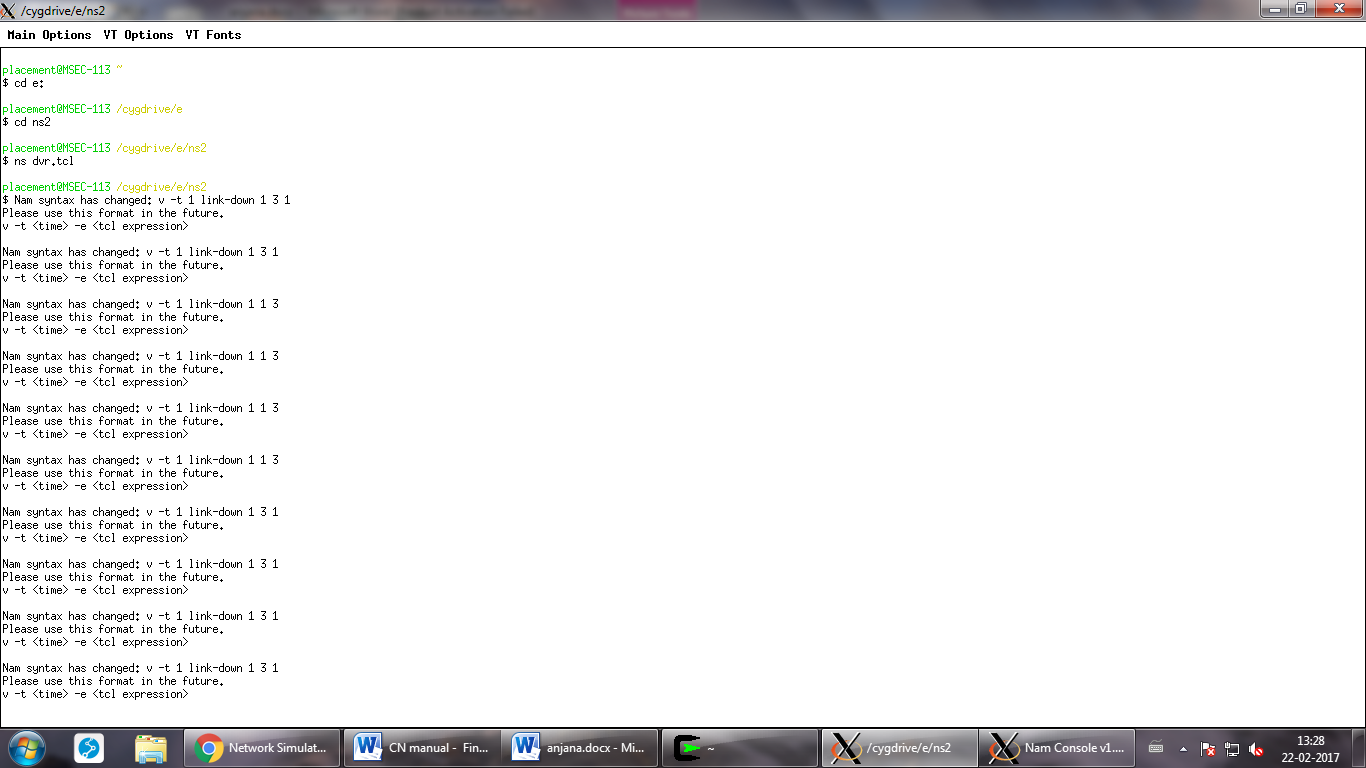
**DISTANCE VECTOR ROUTING**











EXPT:

DATE:

**MINI PROJECT : IMPLEMENTATION OF HTTP SERVER IN C**

**AIM:**

To implement http server in turbo c.

**OBJECTIVE:**

To understand the concept of implementing http server in turbo c.

**SOFTWARE REQUIRED:**

OS-Windows Xp

Putty

**DESCRIPTION:**

This code simply creates a web server with root in the current working directory and default port as 10000.Can handle a maximum of 1000 clients. fork() is used to handle each client. I know it's not a very efficient way of doing it but this code is just to demonstrate a very simple use-case and it's okay to be sloppy and miss details.

The primary function of a web server is to store, process and deliver [web pages](https://en.wikipedia.org/wiki/Web_page) to [clients](https://en.wikipedia.org/wiki/Client_(computing)). The communication between client and server takes place using the [Hypertext Transfer Protocol (HTTP)](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol). Pages delivered are most frequently [HTML documents](https://en.wikipedia.org/wiki/HTML), which may include [images](https://en.wikipedia.org/wiki/Image), [style sheets](https://en.wikipedia.org/wiki/Style_sheet_(web_development)) and [scripts](https://en.wikipedia.org/wiki/JavaScript) in addition to text content.

Multiple web servers may be used for a high traffic website, here [Dell](https://en.wikipedia.org/wiki/Dell) servers are installed together being used for the [Wikimedia Foundation](https://en.wikipedia.org/wiki/Wikimedia_Foundation)

A [user agent](https://en.wikipedia.org/wiki/User_agent), commonly a [web browser](https://en.wikipedia.org/wiki/Web_browser) or [web crawler](https://en.wikipedia.org/wiki/Web_crawler), initiates communication by making a [request](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Request_message) for a specific resource using HTTP and the server responds with the content of that resource or an [error message](https://en.wikipedia.org/wiki/List_of_HTTP_status_codes#4xx_Client_Error) if unable to do so. The resource is typically a real file on the server's [secondary storage](https://en.wikipedia.org/wiki/Secondary_memory), but this is not necessarily the case and depends on how the web server is [implemented](https://en.wikipedia.org/wiki/Implementation).

While the primary function is to serve content, a full implementation of HTTP also includes ways of receiving content from clients. This feature is used for submitting [web forms](https://en.wikipedia.org/wiki/Form_(web)), including [uploading](https://en.wikipedia.org/wiki/Upload) of files.

**ALGORITHM:**

Step 1:Start

Step 2:Port number and address are initiated.

Step 3:Port number is linked via server.

Step 4:The server displays the port number on the screen.

Step 5:Stop

**PROGRAM:**

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<sys/socket.h>

#include<arpa/inet.h>

#include<netdb.h>

#include<signal.h>

#include<fcntl.h>

#define CONNMAX 1000

#define BYTES 1024

char \*ROOT;

intlistenfd, clients[CONNMAX];

void error(char \*);

voidstartServer(char \*);

void respond(int);

int main(intargc, char\* argv[])

{

structsockaddr\_inclientaddr;

socklen\_taddrlen;

char c;

//Default Values PATH = ~/ and PORT=10000

char PORT[6];

ROOT = getenv("PWD");

strcpy(PORT,"10000");

int slot=0;

//Parsing the command line arguments

while ((c = getopt (argc, argv, "p:r:")) != -1)

switch (c)

{

case 'r':

ROOT = malloc(strlen(optarg));

strcpy(ROOT,optarg);

break;

case 'p':

strcpy(PORT,optarg);

break;

case '?':

fprintf(stderr,"Wrong arguments given!!!\n");

exit(1);

default:

exit(1);

}

printf("Server started at port no. %s%s%s with root directory as %s%s%s\n","\033[92m",PORT,"\033[0m","\033[92m",ROOT,"\033[0m");

// Setting all elements to -1: signifies there is no client connected

inti;

for (i=0; i<CONNMAX; i++)

clients[i]=-1;

startServer(PORT);

// ACCEPT connections

while (1)

{

addrlen = sizeof(clientaddr);

clients[slot] = accept (listenfd, (structsockaddr \*) &clientaddr, &addrlen);

if (clients[slot]<0)

error ("accept() error");

else

{

if ( fork()==0 )

{

respond(slot);

exit(0);

}

}

while (clients[slot]!=-1) slot = (slot+1)%CONNMAX;

}

return 0;

}

//start server

voidstartServer(char \*port)

{

structaddrinfo hints, \*res, \*p;

// getaddrinfo for host

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

hints.ai\_family = AF\_INET;

hints.ai\_socktype = SOCK\_STREAM;

hints.ai\_flags = AI\_PASSIVE;

if (getaddrinfo( NULL, port, &hints, &res) != 0)

{

perror ("getaddrinfo() error");

exit(1);

}

// socket and bind

for (p = res; p!=NULL; p=p->ai\_next)

{

listenfd = socket (p->ai\_family, p->ai\_socktype, 0);

if (listenfd == -1) continue;

if (bind(listenfd, p->ai\_addr, p->ai\_addrlen) == 0) break;

}

if (p==NULL)

{

perror ("socket() or bind()");

exit(1);

}

freeaddrinfo(res);

// listen for incoming connections

if ( listen (listenfd, 1000000) != 0 )

{

perror("listen() error");

exit(1);

}

}

//client connection

void respond(int n)

{

charmesg[99999], \*reqline[3], data\_to\_send[BYTES], path[99999];

int rcvd, fd, bytes\_read;

memset( (void\*)mesg, (int)'\0', 99999 );

rcvd=recv(clients[n], mesg, 99999, 0);

if (rcvd<0) // receive error

fprintf(stderr,("recv() error\n"));

else if (rcvd==0) // receive socket closed

fprintf(stderr,"Client disconnected upexpectedly.\n");

else // message received

{

printf("%s", mesg);

reqline[0] = strtok (mesg, " \t\n");

if ( strncmp(reqline[0], "GET\0", 4)==0 )

{

reqline[1] = strtok (NULL, " \t");

reqline[2] = strtok (NULL, " \t\n");

if ( strncmp( reqline[2], "HTTP/1.0", 8)!=0 &&strncmp( reqline[2], "HTTP/1.1", 8)!=0 )

{

write(clients[n], "HTTP/1.0 400 Bad Request\n", 25);

}

else

{

if ( strncmp(reqline[1], "/\0", 2)==0 )

reqline[1] = "/index.html"; //Because if no file is specified, index.html will be opened by default (like it happens in APACHE...

strcpy(path, ROOT);

strcpy(&path[strlen(ROOT)], reqline[1]);

printf("file: %s\n", path);

if ( (fd=open(path, O\_RDONLY))!=-1 ) //FILE FOUND

{

send(clients[n], "HTTP/1.0 200 OK\n\n", 17, 0);

while ( (bytes\_read=read(fd, data\_to\_send, BYTES))>0 )

write (clients[n], data\_to\_send, bytes\_read);

}

else write(clients[n], "HTTP/1.0 404 Not Found\n", 23); //FILE NOT FOUND

}

}

}

//Closing SOCKET

shutdown (clients[n], SHUT\_RDWR); //All further send and recieve operations are DISABLED...

close(clients[n]);

clients[n]=-1;

}

**OUTPUT:**

