**Ex No 1 Create Socket and display Socket ID**

**Aim**

To create a Socket and display its ID

**Program**

#include<stdio.h>

#include<string.h>

#include<sys/socket.h>

#include<netinet/in.h>

#include<arpa/inet.h>

void main(){

int SocketFd;

//Create socket

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

if(SocketFd == -1)

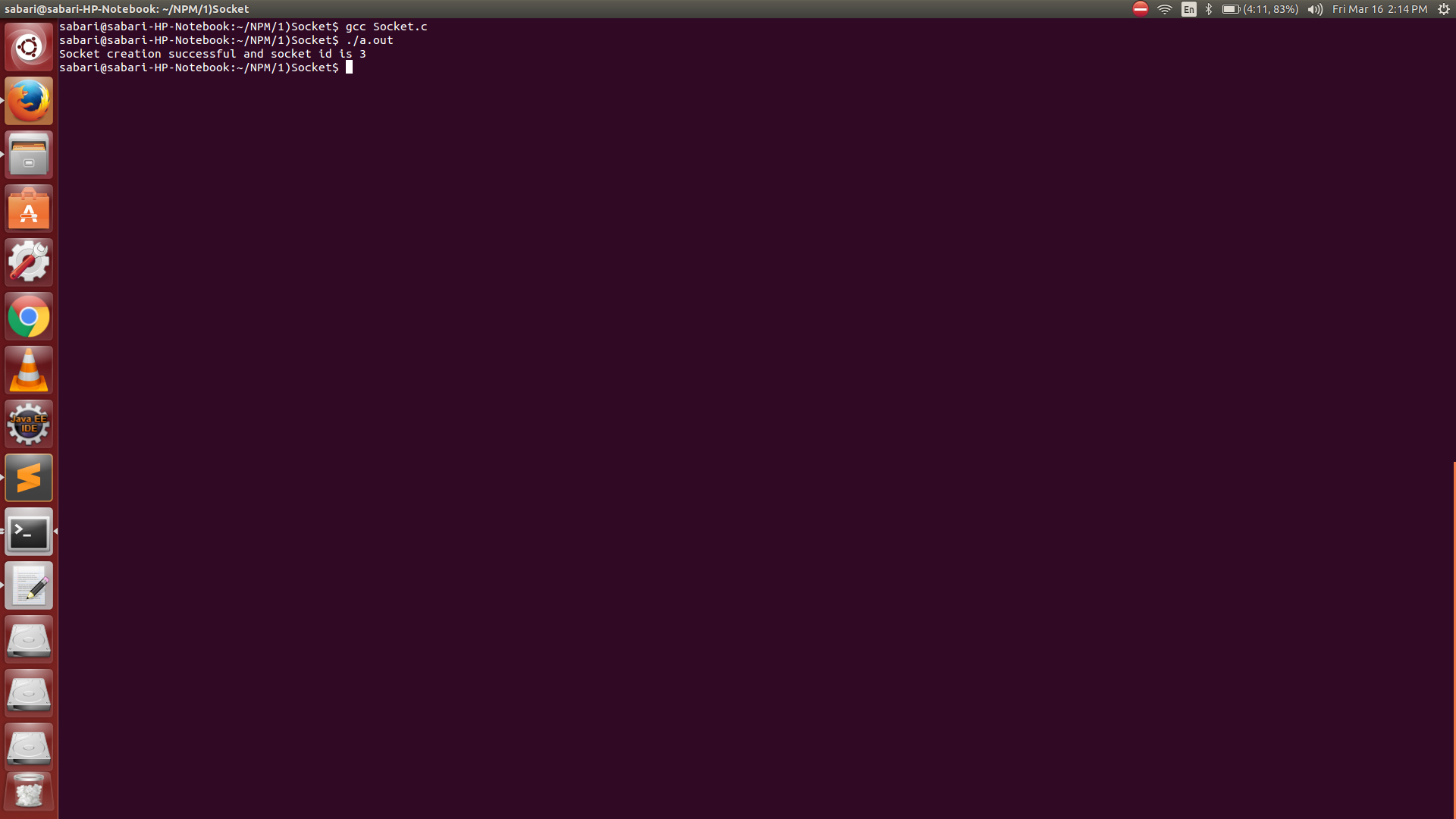
printf("Socket creation failed \n");

else

printf("Socket creation successful and socket id is %d\n",SocketFd);

}

**Output**



**Ex No 2 Implementation of Address Conversion Routines**

**Aim**

To implement Address Conversion routines

**Program**

#include<stdio.h>

#include<stdlib.h>

#include<netinet/in.h>

#include<sys/socket.h>

#include<arpa/inet.h>

int main(){

struct in\_addr ipAddress;

long int address;

char \*ptr;

char \*hostAddress = (char \*)malloc(sizeof(char));

printf("Enter the host Address \n");

fgets(hostAddress,100,stdin);

address = inet\_addr(hostAddress);

printf("%s in binary form is %ld \n",hostAddress,address);

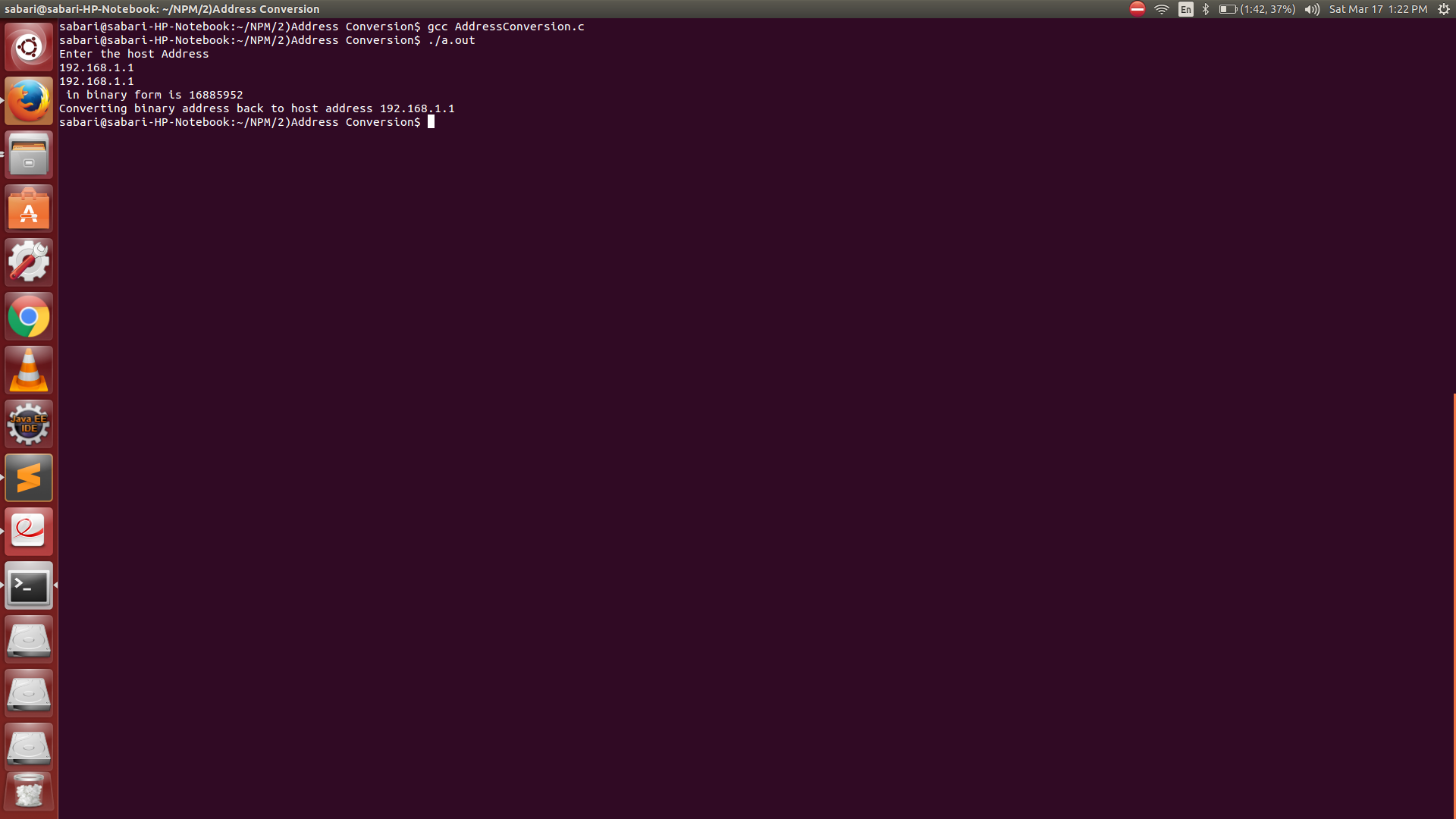
ipAddress.s\_addr = address;

ptr = inet\_ntoa(ipAddress);

printf("Converting binary address back to host address %s\n",ptr);

return 0; }

**Output**



**Ex No 3 Develop a Client Server Application for Chat using TCP**

**Aim**

To develop a client Server Application for chat using TCP

**Program**

**Server**

#include<stdio.h>

#include<string.h>

#include<sys/socket.h>

#include<netinet/in.h>

#include<arpa/inet.h>

void communicateClient(int connectionSocket){

char receive\_buffer[1024];

char send\_buffer[1024];

while(1){

bzero(receive\_buffer,1024);

read(connectionSocket,receive\_buffer,sizeof(receive\_buffer));

printf("Message from client \n");

printf("%s \n",receive\_buffer);

bzero(send\_buffer,1024);

printf("Enter the message for client \n");

fgets(send\_buffer,1024,stdin);

write(connectionSocket,send\_buffer,sizeof(send\_buffer));

if(strncmp(send\_buffer,"exit",4)==0)

break;

}

}

void main(){

int serverSocket,connectionSocket,clientAddressLength;

int bind\_result,listen\_result;

struct sockaddr\_in serverAddress,clientAddress;

//Create socket

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

if(serverSocket == -1)

printf("Socket creation failed \n");

else

printf("Socket creation successful \n");

//Configure server address

serverAddress.sin\_family = AF\_INET;

serverAddress.sin\_port = htons(8000);

serverAddress.sin\_addr.s\_addr = INADDR\_ANY;

//Binds the created socket with the given address

bind\_result = bind(serverSocket,(struct sockaddr \*)&serverAddress,sizeof(serverAddress));

if(bind\_result == -1)

printf("Bind process failed \n");

else

printf("Bind successful \n");

//Listens for client connection on the specified socket

listen\_result = listen(serverSocket,10); //10 is the backlog value that specifies maximum number of clients can wait in the connection queue

if(listen\_result == -1 )

printf("Server is not listening \n");

else

printf("Server is listening \n");

clientAddressLength = sizeof(clientAddress);

connectionSocket = accept(serverSocket,(struct sockaddr \*)&clientAddress,&clientAddressLength);

if(connectionSocket == -1 )

printf("New connection rejected \n");

else{

printf("New connection accepted \n");

communicateClient(connectionSocket);

}

}

**Client**

#include<stdio.h>

#include<string.h>

#include<sys/socket.h>

#include<netinet/in.h>

#include<arpa/inet.h>

void communicateServer(int clientSocket){

char send\_buffer[1024];

char receive\_buffer[1024];

while(1){

bzero(send\_buffer,1024);

printf("Enter the message for server \n");

fgets(send\_buffer,1024,stdin);

write(clientSocket,send\_buffer,sizeof(send\_buffer));

bzero(receive\_buffer,1024);

read(clientSocket,receive\_buffer,sizeof(receive\_buffer));

printf("Message from server \n");

printf("%s \n",receive\_buffer);

if(strncmp(receive\_buffer,"exit",4)==0)

break;

}

}

void main(){

int clientSocket,connectionResult;

struct sockaddr\_in serverAddress;

//Create a socket

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

if(clientSocket == -1 )

printf("Socket creation failed \n");

else

printf("Socket creation successful \n");

serverAddress.sin\_family = AF\_INET;

serverAddress.sin\_port = htons(8000);

serverAddress.sin\_addr.s\_addr = INADDR\_ANY;

//Connect with server

connectionResult = connect(clientSocket,(struct sockaddr \*)&serverAddress,sizeof(serverAddress));

if(connectionResult == -1 )

printf("Connection failed \n");

else{

printf("Connection scuccessful \n");

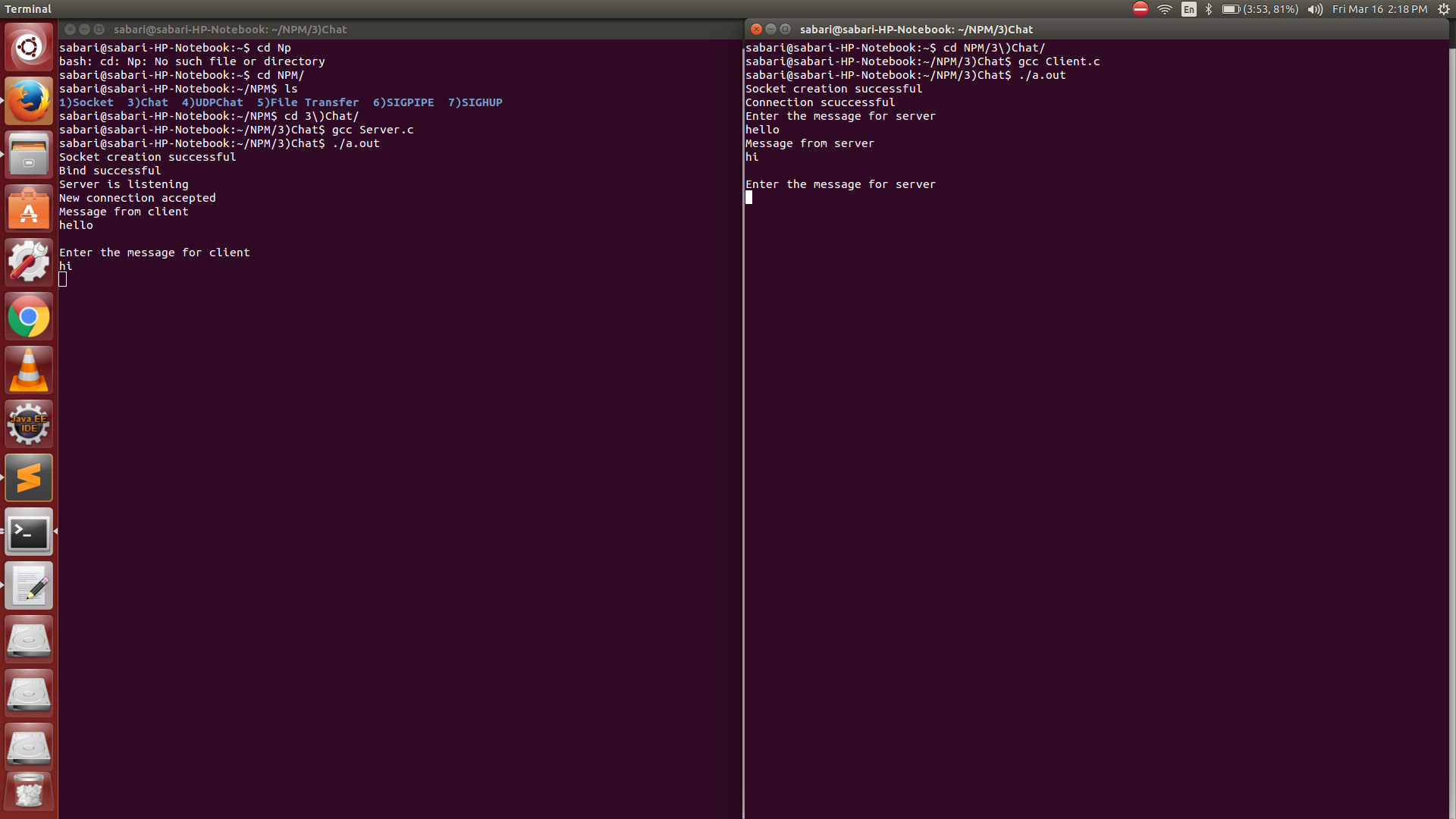
communicateServer(clientSocket);

close(clientSocket);

}

}

**Output**



**Ex No 4 Implementation of UDP Client Server Communication using Bind, SendTo, and RecvFrom System Calls**

**Aim**

To develop a client Server Application for chat using UDP

**Program**

#include<stdio.h>

#include<arpa/inet.h>

#include<netinet/in.h>

#include<sys/socket.h>

#include<string.h>

void communicateClient(int connectionSocket,struct sockaddr\_in clientAddress){

char receive\_buffer[1024];

char send\_buffer[1024];

int clientAddressLength = sizeof(clientAddress);

while(1){

bzero(receive\_buffer,1024);

recvfrom(connectionSocket,receive\_buffer,sizeof(receive\_buffer),0,(struct sockaddr \*)&clientAddress,&clientAddressLength);

printf("Message from client \n");

printf("%s \n",receive\_buffer);

bzero(send\_buffer,1024);

printf("Enter the message for client \n");

fgets(send\_buffer,1024,stdin);

sendto(connectionSocket,send\_buffer,sizeof(send\_buffer),0,(struct sockaddr \*)&clientAddress,sizeof(clientAddress));

if(strncmp(send\_buffer,"exit",4)==0)

break;

}

}

void main(){

int serverSocket;

int bindResult;

char buffer[100];

serverSocket = socket(AF\_INET,SOCK\_DGRAM,0);

struct sockaddr\_in serverAddress,clientAddress;

if(serverSocket == -1)

printf("Socket is not created \n");

else{

printf("Socket created successfully \n");

serverAddress.sin\_family = AF\_INET;

serverAddress.sin\_port = htons(9000);

serverAddress.sin\_addr.s\_addr = INADDR\_ANY;

bindResult = bind(serverSocket,(struct sockaddr \*)&serverAddress,sizeof(serverAddress));

if(bindResult == -1)

printf("Bind not successful \n");

else{

printf("Bind successful \n");

communicateClient(serverSocket,clientAddress);

/\*recvfrom(serverSocket,buffer,sizeof(buffer),0,(struct sockaddr \*)&clientAddress,&clientAddressLength);

printf("%s \n ",buffer);

sendto(serverSocket,buffer,sizeof(buffer),0,(struct sockaddr \*)&clientAddress,sizeof(clientAddress));\*/

}

}

}

**Client**

#include<stdio.h>

#include<netinet/in.h>

#include<arpa/inet.h>

#include<string.h>

#include<sys/socket.h>

void communicateServer(int clientSocket,struct sockaddr\_in serverAddress){

char send\_buffer[1024];

char receive\_buffer[1024];

while(1){

bzero(send\_buffer,1024);

printf("Enter the message for server \n");

fgets(send\_buffer,1024,stdin);

sendto(clientSocket,send\_buffer,sizeof(send\_buffer),0,(struct sockaddr \*)&serverAddress,sizeof(serverAddress));

bzero(receive\_buffer,1024);

recvfrom(clientSocket,receive\_buffer,sizeof(receive\_buffer),0,NULL,NULL);

printf("Message from server \n");

printf("%s \n",receive\_buffer);

if(strncmp(receive\_buffer,"exit",4)==0)

break;

}

}

void main(){

int clientSocket;

char buffer[100];

struct sockaddr\_in serverAddress;

clientSocket = socket(AF\_INET,SOCK\_DGRAM,0);

if(clientSocket == -1)

printf("Socket creation failed \n");

else{

printf("Socket creation successful \n");

serverAddress.sin\_family = AF\_INET;

serverAddress.sin\_port = htons(9000);

serverAddress.sin\_addr.s\_addr = INADDR\_ANY;

communicateServer(clientSocket,serverAddress);

/\*int serverAddressLength = sizeof(serverAddress);

sendto(clientSocket,"Hello Server",13,0,(struct sockaddr \*)&serverAddress,serverAddressLength);

printf("test \n");

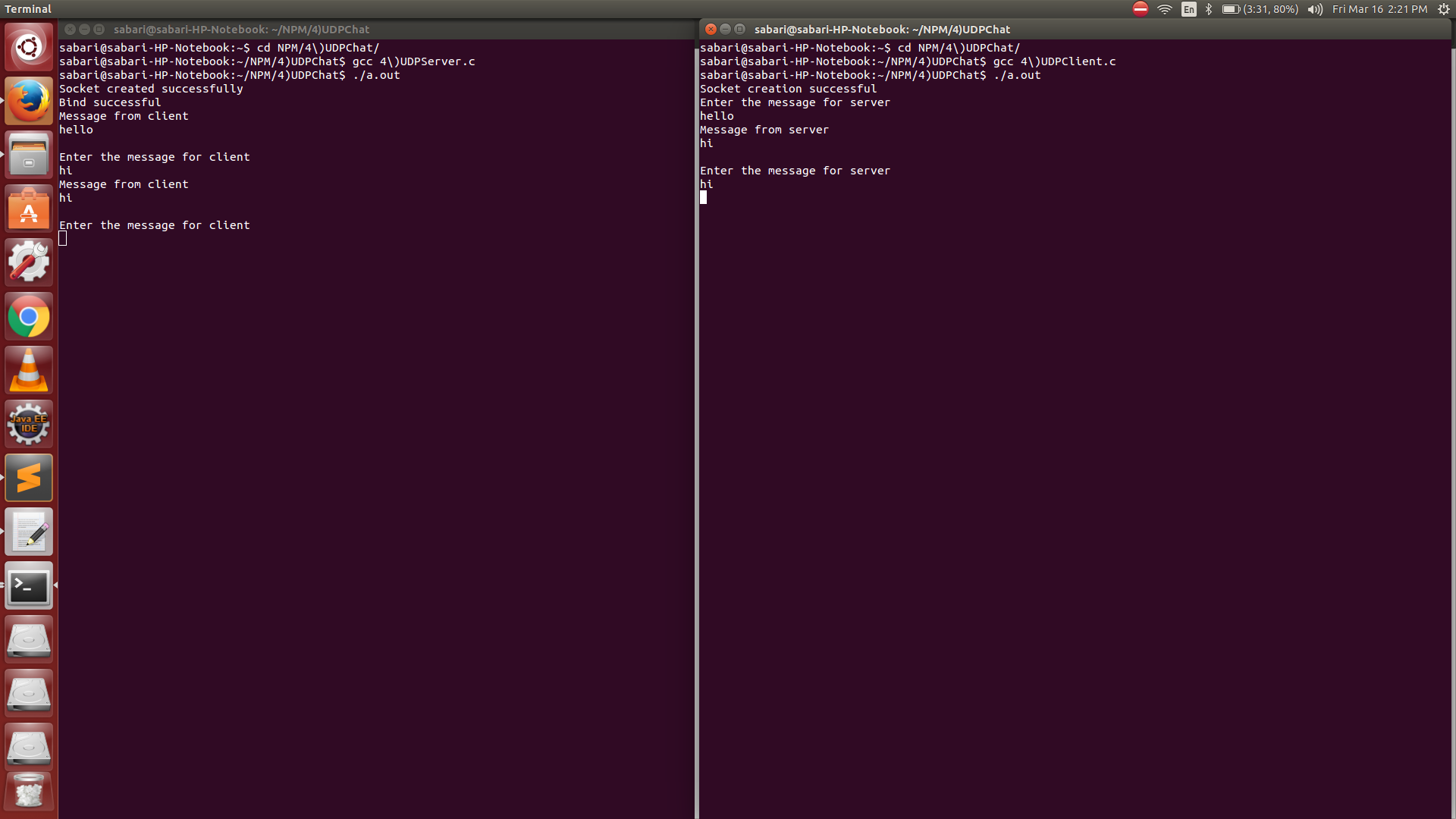
int res = recvfrom(clientSocket,buffer,sizeof(buffer),0,NULL,NULL);

printf("%s \n",buffer);\*/

}

}

**Output**



**Ex No 5 Design TCP Client and Server application to transfer file**

**Aim**

To design a TCP Client and Server Application to transfer files

**Program**

**Server**

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<sys/socket.h>

#include<netinet/in.h>

#include<arpa/inet.h>

#include<fcntl.h>

#include<errno.h>

void communicateClient(int connectionSocket){

char \*buff = (char \*)malloc(100);

int file;

char \*filename = (char \*)malloc(sizeof(char));

read(connectionSocket,buff,sizeof(buff));

int len = strlen(buff);

int i=0;

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

filename[i]=buff[i];

filename[i]='\0';

printf("Filename %s \n",filename);

file = open(filename,O\_RDONLY);

printf("%d \n",file);

if(file <= 0){

printf("File couldn't be opened with error %s \n",strerror(errno));

strcpy(buff,"File couldn't be opened");

}

else

read(file,buff,sizeof(buff));

write(connectionSocket,buff,sizeof(buff));

}

void main(){

int serverSocket,connectionSocket,clientAddressLength;

int bind\_result,listen\_result;

struct sockaddr\_in serverAddress,clientAddress;

//Create socket

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

if(serverSocket == -1)

printf("Socket creation failed \n");

else{

printf("Socket creation successful \n");

//Configure server address

serverAddress.sin\_family = AF\_INET;

serverAddress.sin\_port = htons(8000);

serverAddress.sin\_addr.s\_addr = INADDR\_ANY;

//Binds the created socket with the given address

bind\_result = bind(serverSocket,(struct sockaddr \*)&serverAddress,sizeof(serverAddress));

if(bind\_result == -1)

printf("Bind process failed \n");

else{

printf("Bind successful \n");

//Listens for client connection on the specified socket

listen\_result = listen(serverSocket,10); //10 is the backlog value that specifies maximum number of clients can wait in the connection queue

if(listen\_result == -1 )

printf("Server is not listening \n");

else{

printf("Server is listening \n");

clientAddressLength = sizeof(clientAddress);

connectionSocket = accept(serverSocket,(struct sockaddr \*)&clientAddress,&clientAddressLength);

if(connectionSocket == -1 )

printf("New connection rejected \n");

else{

printf("New connection accepted \n");

communicateClient(connectionSocket);

}

}

}

}

}

**Client**

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<sys/socket.h>

#include<netinet/in.h>

#include<arpa/inet.h>

void communicateServer(int clientSocket){

char \*buff = (char \*)malloc(sizeof(char));

printf("Enter the filename \n");

fgets(buff,255,stdin);

write(clientSocket,buff,sizeof(buff));

read(clientSocket,buff,sizeof(buff));

printf("Reply from server \n %s",buff);

}

void main(){

int clientSocket,connectionResult;

struct sockaddr\_in serverAddress;

//Create a socket

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

if(clientSocket == -1 )

printf("Socket creation failed \n");

else{

printf("Socket creation successful \n");

serverAddress.sin\_family = AF\_INET;

serverAddress.sin\_port = htons(8000);

serverAddress.sin\_addr.s\_addr = INADDR\_ANY;

//Connect with server

connectionResult = connect(clientSocket,(struct sockaddr \*)&serverAddress,sizeof(serverAddress));

if(connectionResult == -1 )

printf("Connection failed \n");

else{

printf("Connection scuccessful \n");

communicateServer(clientSocket);

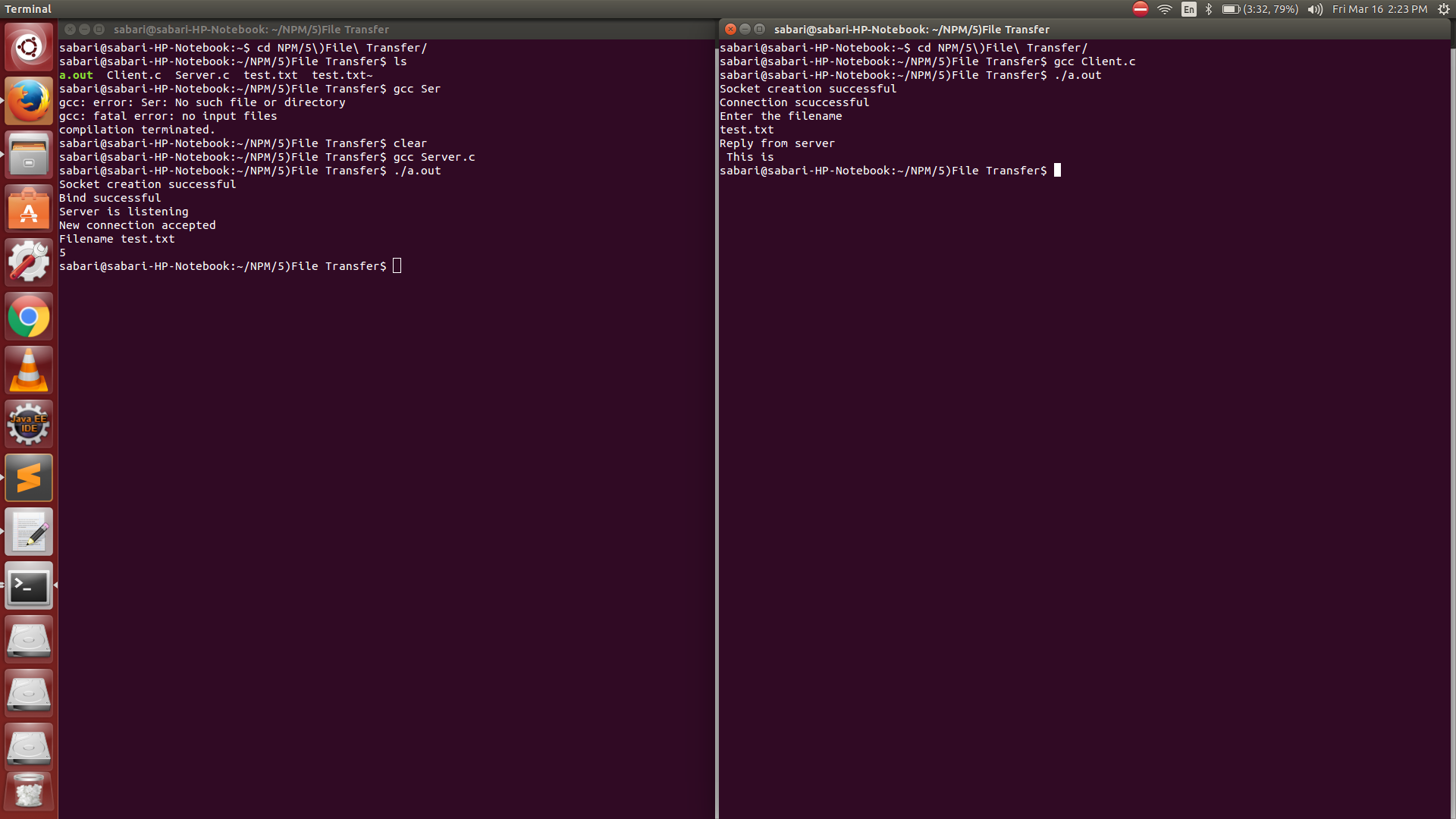
close(clientSocket);

}

}

**}**

**Output**



**Ex No 6 Demonstration to generate SIGPIPE Error with Socket**

**Aim**

To generate a SIGPIPE error with socket

**Program**

**Server**

#include<stdio.h>

#include<string.h>

#include<sys/socket.h>

#include<netinet/in.h>

#include<arpa/inet.h>

void communicateClient(int connectionSocket){

char receive\_buffer[1024];

char send\_buffer[1024];

while(1){

bzero(receive\_buffer,1024);

read(connectionSocket,receive\_buffer,sizeof(receive\_buffer));

printf("Message from client \n");

printf("%s \n",receive\_buffer);

bzero(send\_buffer,1024);

printf("Enter the message for client \n");

fgets(send\_buffer,1024,stdin);

write(connectionSocket,send\_buffer,sizeof(send\_buffer));

if(strncmp(send\_buffer,"exit",4)==0)

break;

}

}

void main(){

int serverSocket,connectionSocket,clientAddressLength;

int bind\_result,listen\_result;

struct sockaddr\_in serverAddress,clientAddress;

//Create socket

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

if(serverSocket == -1)

printf("Socket creation failed \n");

else

printf("Socket creation successful \n");

//Configure server address

serverAddress.sin\_family = AF\_INET;

serverAddress.sin\_port = htons(8000);

serverAddress.sin\_addr.s\_addr = INADDR\_ANY;

//Binds the created socket with the given address

bind\_result = bind(serverSocket,(struct sockaddr \*)&serverAddress,sizeof(serverAddress));

if(bind\_result == -1)

printf("Bind process failed \n");

else

printf("Bind successful \n");

//Listens for client connection on the specified socket

listen\_result = listen(serverSocket,10); //10 is the backlog value that specifies maximum number of clients can wait in the connection queue

if(listen\_result == -1 )

printf("Server is not listening \n");

else

printf("Server is listening \n");

clientAddressLength = sizeof(clientAddress);

connectionSocket = accept(serverSocket,(struct sockaddr \*)&clientAddress,&clientAddressLength);

if(connectionSocket == -1 )

printf("New connection rejected \n");

else{

printf("New connection accepted \n");

communicateClient(connectionSocket);

}

}

**Client**

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<sys/socket.h>

#include<netinet/in.h>

#include<arpa/inet.h>

#include<signal.h>

void sig\_pipe(){

printf("Server terminated prematurely \n");

exit(0);

}

void communicateServer(int clientSocket){

char send\_buffer[1024];

char receive\_buffer[1024];

while(1){

bzero(send\_buffer,1024);

printf("Enter the message for server \n");

fgets(send\_buffer,1024,stdin);

write(clientSocket,send\_buffer,sizeof(send\_buffer));

bzero(receive\_buffer,1024);

read(clientSocket,receive\_buffer,sizeof(receive\_buffer));

printf("Message from server \n");

printf("%s \n",receive\_buffer);

/\*if(strncmp(receive\_buffer,"exit",4)==0)

break;\*/

}

}

void main(){

int clientSocket,connectionResult;

struct sockaddr\_in serverAddress;

//Create a socket

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

if(clientSocket == -1 )

printf("Socket creation failed \n");

else

printf("Socket creation successful \n");

serverAddress.sin\_family = AF\_INET;

serverAddress.sin\_port = htons(8000);

serverAddress.sin\_addr.s\_addr = INADDR\_ANY;

signal(SIGPIPE,sig\_pipe);

//Connect with server

connectionResult = connect(clientSocket,(struct sockaddr \*)&serverAddress,sizeof(serverAddress));

if(connectionResult == -1 )

printf("Connection failed \n");

else{

printf("Connection scuccessful \n");

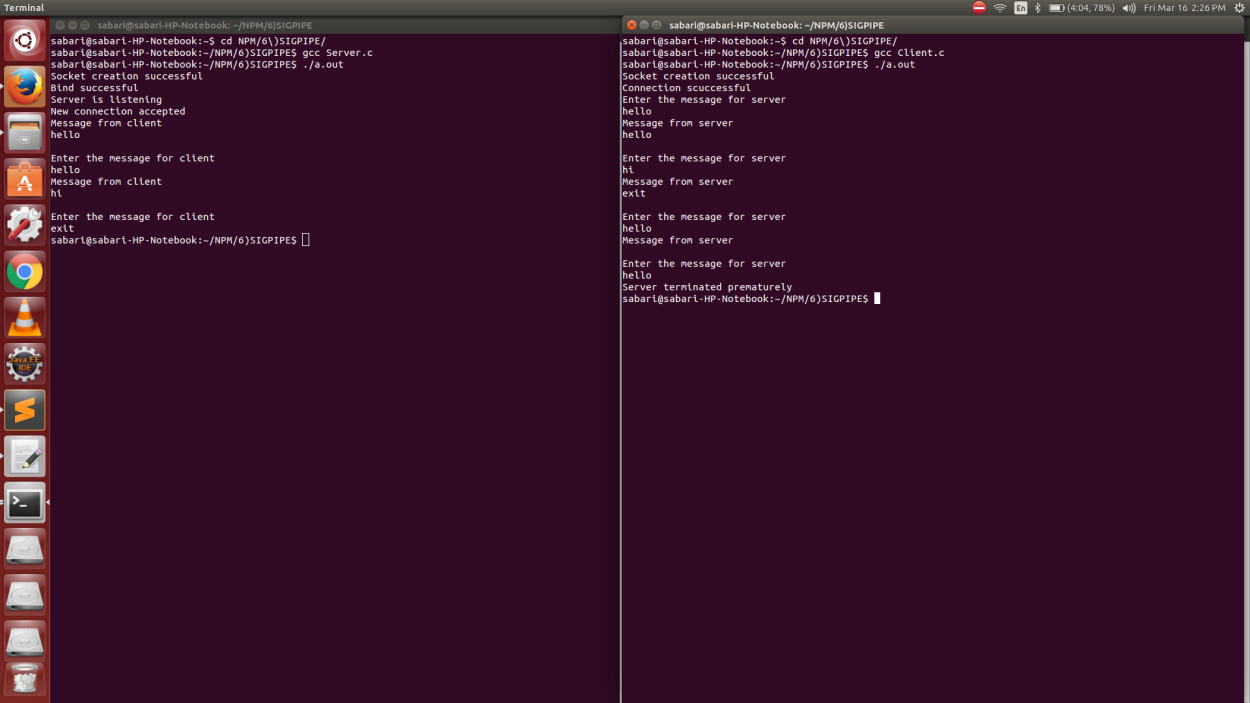
communicateServer(clientSocket);

//close(clientSocket);

}

}

Output



**Ex No 7 Demonstration to restart server by capturing SIGHUP signal**

**Aim**

To restart a server by capturing SIGHUP signal

**Program**

**Server**

#include<netinet/in.h>

#include<sys/types.h>

#include<sys/socket.h>

#include<unistd.h>

#include<arpa/inet.h>

#include<stdio.h>

#include<string.h>

#include<signal.h>

#include<stdlib.h>

int sockfd;

void myhand()

{

printf("SIGHUPcaught\n");

printf("restarting server\n");

close(sockfd);

execl("/udpserver","udpserver",NULL);

printf("Server is not restarted\n");

}

int main()

{

char msg[100]="",rply[100]="";

int len;

struct sockaddr\_in server,client;

signal(SIGHUP,myhand);

sockfd=socket(AF\_INET,SOCK\_DGRAM,0);

if(sockfd<0)

{

printf("socket error correction\n");

}

printf("socket created successsssssssssfully\n");

bzero(&server,sizeof(server));

server.sin\_port=htons(3230);

server.sin\_family=AF\_INET;

inet\_aton("127.0.0.1",&server.sin\_addr);

if(bind(sockfd,(struct sockaddr\*)&server,sizeof(server))==0)

while(1)

{

recvfrom(sockfd,(char \*)msg,100,0,(struct sockaddr \*)&client,&len);

puts(msg);

sendto(sockfd,(char \*)rply,strlen(msg),0,(struct sockaddr \*)&client,len);

}

close(sockfd);

}

**Client**

#include<netinet/in.h>

#include<sys/types.h>

#include<sys/socket.h>

#include<unistd.h>

#include<arpa/inet.h>

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

int main()

{

char msg[100]="",msg1[100]="";

int sockfd,confd;

struct sockaddr\_in server,client;

sockfd=socket(AF\_INET,SOCK\_DGRAM,0);

if(sockfd<0)

{

printf("socket error correction\n");

}

printf("socket created successfully\n");

bzero(&server,sizeof(server));

server.sin\_port=htons(3230);

server.sin\_family=AF\_INET;

inet\_aton("127.0.0.1",&server.sin\_addr);

int len=sizeof(server);

while(1)

{

printf("enter the msg\n");

gets(msg);

sendto(sockfd,(char \*)msg,strlen(msg),0,(struct sockaddr \*)&server,len);

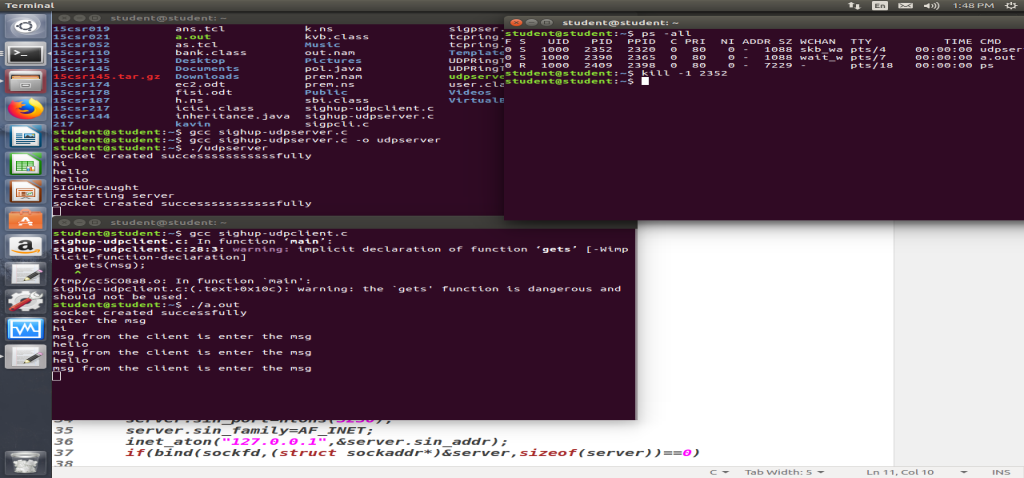
recvfrom(sockfd,(char \*)msg,100,0,(struct sockaddr \*)&server,&len);

printf("msg from the client is %s",msg);

}

close(sockfd);

}



**ExNo 8 Implementation of Ping and Traceroute**

**Aim**

To implement ping and traceroute

**Program**

#include <stdio.h>

#include <signal.h>

#include <arpa/inet.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <unistd.h>

#include <netinet/in.h>

#include <netinet/ip.h>

#include <netinet/ip\_icmp.h>

#include <netdb.h>

#include <setjmp.h>

#include <errno.h>

#include<stdlib.h>

#include<string.h>

#define PACKET\_SIZE 4096

#define MAX\_WAIT\_TIME 5

#define MAX\_NO\_PACKETS 3

char sendpacket[PACKET\_SIZE];

char recvpacket[PACKET\_SIZE];

int sockfd, datalen = 56;

int nsend = 0, nreceived = 0;

struct sockaddr\_in dest\_addr;

pid\_t pid;

struct sockaddr\_in from;

struct timeval tvrecv;

void statistics(int signo);

unsigned short cal\_chksum(unsigned short \*addr, int len);

int pack(int pack\_no);

void send\_packet(void);

void recv\_packet(void);

int unpack(char \*buf, int len);

void tv\_sub(struct timeval \*out, struct timeval \*in);

void statistics(int signo)

{

printf("\n--------------------PING statistics-------------------\n");

printf("%d packets transmitted, %d received , %%%d lost\n", nsend, nreceived, (nsend - nreceived) / nsend \*

100);

close(sockfd);

exit(1); }

unsigned short cal\_chksum(unsigned short \*addr, int len)

{ int nleft = len;

int sum = 0;

unsigned short \*w = addr;

unsigned short answer = 0;

while (nleft > 1)

{ sum += \*w++;

nleft -= 2 }

if (nleft == 1)

{\*(unsigned char\*)(&answer) = \*(unsigned char\*)w;

sum += answer; }

sum = (sum >> 16) + (sum &0xffff);

sum += (sum >> 16);

answer = ~sum;

return answer; }

int pack(int pack\_no)

{

int i, packsize;

struct icmp \*icmp;

struct timeval \*tval;

icmp = (struct icmp\*)sendpacket;

icmp->icmp\_type = ICMP\_ECHO;

icmp->icmp\_code = 0;

icmp->icmp\_cksum = 0;

icmp->icmp\_seq = pack\_no;

icmp->icmp\_id = pid;

packsize = 8+datalen;

tval = (struct timeval\*)icmp->icmp\_data;

gettimeofday(tval, NULL);

icmp->icmp\_cksum = cal\_chksum((unsigned short\*)icmp, packsize);

return packsize;

}

void send\_packet()

{

int packetsize;

while (nsend < MAX\_NO\_PACKETS)

{

nsend++;

packetsize = pack(nsend);

if (sendto(sockfd, sendpacket, packetsize, 0, (struct sockaddr\*)

&dest\_addr, sizeof(dest\_addr)) < 0)

{

perror("sendto error");

continue;

} sleep(1);

}

}

void recv\_packet()

{

int n, fromlen;

extern int errno;

signal(SIGALRM, statistics);

fromlen = sizeof(from);

while (nreceived <nsend)

{

alarm(MAX\_WAIT\_TIME);

if ((n = recvfrom(sockfd, recvpacket, sizeof(recvpacket), 0, (struct sockaddr\*) &from, &fromlen)) < 0)

{

if (errno == EINTR)

continue;

perror("recvfrom error");

continue;

} gettimeofday(&tvrecv, NULL);

if (unpack(recvpacket, n) == - 1)

continue;

nreceived++;

}

}

int unpack(char \*buf, int len)

{

int i, iphdrlen;

struct ip \*ip;

struct icmp \*icmp;

struct timeval \*tvsend;

double rtt;

ip = (struct ip\*)buf;

iphdrlen = ip->ip\_hl << 2;

icmp = (struct icmp\*)(buf + iphdrlen);

len -= iphdrlen;

if (len < 8)

{

printf("ICMP packets\'s length is less than 8\n");

return - 1;

}

if ((icmp->icmp\_type == ICMP\_ECHOREPLY) && (icmp->icmp\_id == pid))

{

tvsend = (struct timeval\*)icmp->icmp\_data;

tv\_sub(&tvrecv, tvsend);

rtt = tvrecv.tv\_sec \* 1000+tvrecv.tv\_usec / 1000;

printf("%d byte from %s: icmp\_seq=%u ttl=%d rtt=%.3f ms\n", len,inet\_ntoa(from.sin\_addr), icmp->icmp\_seq,

ip->ip\_ttl, rtt);

}

else

return - 1;

}

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

{

struct hostent \*host;

struct protoent \*protocol;

unsigned long inaddr = 0l;

int waittime = MAX\_WAIT\_TIME;

int size = 50 \* 1024;

if (argc < 2)

{

printf("usage:%s hostname/IP address\n", argv[0]);

exit(1);

} if ((protocol = getprotobyname("icmp")) == NULL)

{

perror("getprotobyname");

exit(1);

}

if ((sockfd = socket(AF\_INET, SOCK\_RAW, protocol->p\_proto)) < 0)

{

perror("socket error");

exit(1);

}

setuid(getuid());

setsockopt(sockfd, SOL\_SOCKET, SO\_RCVBUF, &size, sizeof(size));

bzero(&dest\_addr, sizeof(dest\_addr));

dest\_addr.sin\_family = AF\_INET;

if (inaddr = inet\_addr(argv[1]) == INADDR\_NONE)

{

if ((host = gethostbyname(argv[1])) == NULL)

{ perror("gethostbyname error");

exit(1);

}

memcpy((char\*) &dest\_addr.sin\_addr, host->h\_addr, host->h\_length);

}

else

dest\_addr.sin\_addr.s\_addr = inet\_addr(argv[1]);

pid = getpid();

printf("PING %s(%s): %d bytes data in ICMP packets.\n", argv[1], inet\_ntoa

(dest\_addr.sin\_addr), datalen);

send\_packet();

recv\_packet();

statistics(SIGALRM);

return 0;

}

void tv\_sub(struct timeval \*out, struct timeval \*in)

{

if ((out->tv\_usec -= in->tv\_usec) < 0)

{ --out->tv\_sec;

out->tv\_usec += 1000000;

} out->tv\_sec -= in->tv\_sec;

}

OUTPUT:

gcc ping.c

./a.out google.com

Ping google.com(216.58.197.98) 56 bytes in ICMP packets

64 Bytes from 216.58.197.98 ICMP-seq 1 ttl=55 rtt=3000 ms

64 Bytes from 216.58.197.98 ICMP-seq 2 ttl=55 rtt=2000 ms

64 Bytes from 216.58.197.98 ICMP-seq 3 ttl=55 rtt=1000 ms

--------------------PING statistics-------------------

3 packets transmitted, 3 received, %0 loss

**ExNo 9 Performance Analysis of TCP/UDP using Simulation Tool**

**Aim**

To analyze performance of TCP/UDP using simulation tool

**ExNo 10 Simulation of LAN(CSMA/CD) using simulation tool**

**Aim**

To simulate LAN(CSMA/CD) using Simulation Tool

**Program**

set ns [new Simulator]

#define color for data flows

$ns color 1 Blue

$ns color 2 Red

#open tracefiles

set tracefile1 [open out.tr w]

set winfile [open winfile w]

$ns trace-all $tracefile1

#open nam file

set namfile [open out.nam w]

$ns namtrace-all $namfile

#define the finish procedure

proc finish {} {

global ns tracefile1 namfile

$ns flush-trace

close $tracefile1

close $namfile

exec nam out.nam &

exit 0

}

#create six nodes

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]

set n4 [$ns node]

set n5 [$ns node]

set n6 [$ns node]

set n7 [$ns node]

set n8 [$ns node]

$n1 color Red

$n1 shape square

#create links between the nodes

$ns duplex-link $n0 $n2 2Mb 10ms DropTail

$ns duplex-link $n1 $n2 2Mb 10ms DropTail

$ns simplex-link $n2 $n3 0.3Mb 100ms DropTail

$ns simplex-link $n3 $n2 0.3Mb 100ms DropTail

set lan [$ns newLan "$n3 $n4 $n5 $n6 $n7 $n8" 0.5Mb 40ms LL Queue/DropTail MAC/Csma/Cd Channel]

#Give node position

$ns duplex-link-op $n0 $n2 orient right-down

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

$ns simplex-link-op $n2 $n3 orient right

$ns simplex-link-op $n3 $n2 orient left

#set queue size of link(n2-n3) to 20

$ns queue-limit $n2 $n3 20

#setup TCP connection

set tcp [new Agent/TCP/Newreno]

$ns attach-agent $n0 $tcp

set sink [new Agent/TCPSink/DelAck]

$ns attach-agent $n4 $sink

$ns connect $tcp $sink

$tcp set fid\_ 1

$tcp set packet\_size\_ 552

#set ftp over tcp connection

set ftp [new Application/FTP]

$ftp attach-agent $tcp

#setup a UDP connection

set udp [new Agent/UDP]

$ns attach-agent $n1 $udp

set null [new Agent/Null]

$ns attach-agent $n5 $null

$ns connect $udp $null

$udp set fid\_ 2

#setup a CBR over UDP connection

set cbr [new Application/Traffic/CBR]

$cbr attach-agent $udp

$cbr set type\_ CBR

$cbr set packet\_size\_ 1000

$cbr set rate\_ 0.01Mb

$cbr set random\_ false

#scheduling the events

$ns at 0.1 "$cbr start"

$ns at 1.0 "$ftp start"

$ns at 124.0 "$ftp stop"

$ns at 125.5 "$cbr stop"

proc plotWindow {tcpSource file} {

global ns

set time 0.1

set now [$ns now]

set cwnd [$tcpSource set cwnd\_]

puts $file "$now $cwnd"

$ns at [expr $now+$time] "plotWindow $tcpSource $file"

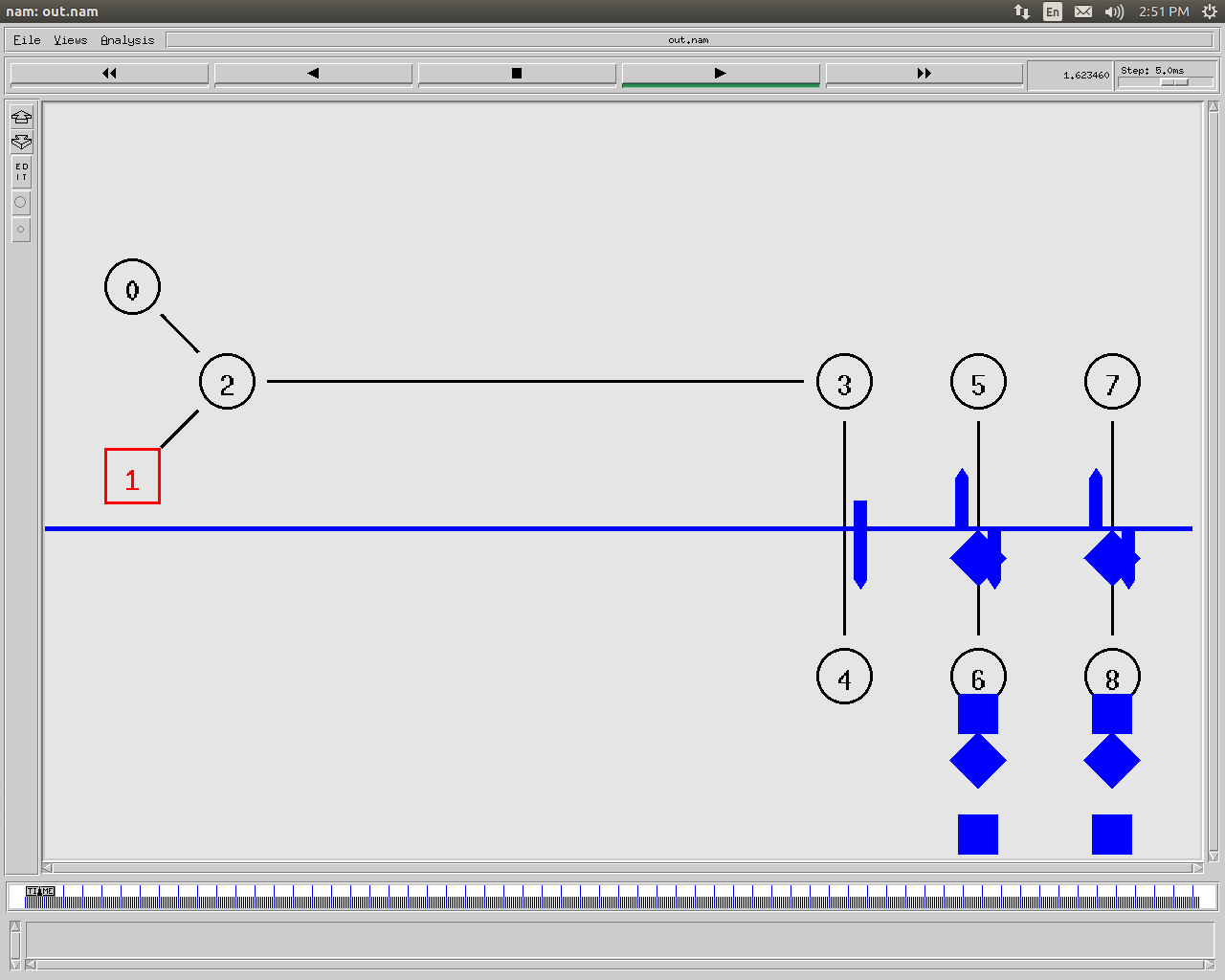
}

$ns at 0.1 "plotWindow $tcp $winfile"

$ns at 125.0 "finish"

$ns run

**Output**



**ExNo 11 Simulation of Distance Vector/Link State Routing Protocol**

**Aim**

To Simulate Distance Vector/Link State Routing Protocol using simulation tool

**Program**

set ns [new Simulator]

$ns rtproto DV

set nf [open out.nam w]

$ns namtrace-all $nf

proc finish {} {

global ns nf

$ns flush-trace

close $nf

exec nam out.nam &

exit 0

}

for {set i 0} {$i < 7} {incr i} {

set n($i) [$ns node]

}

for {set i 0} {$i < 7} {incr i} {

$ns duplex-link $n($i) $n([expr ($i+1)%7]) 1Mb 10ms DropTail

}

set udp0 [new Agent/UDP]

$ns attach-agent $n(0) $udp0

set cbr0 [new Application/Traffic/CBR]

$cbr0 set packetSize\_ 500

$cbr0 set interval\_ 0.025

$cbr0 attach-agent $udp0

set null0 [new Agent/Null]

$ns attach-agent $n(3) $null0

$ns connect $udp0 $null0

$ns at 0.5 "$cbr0 start"

$ns rtmodel-at 1.0 down $n(1) $n(2)

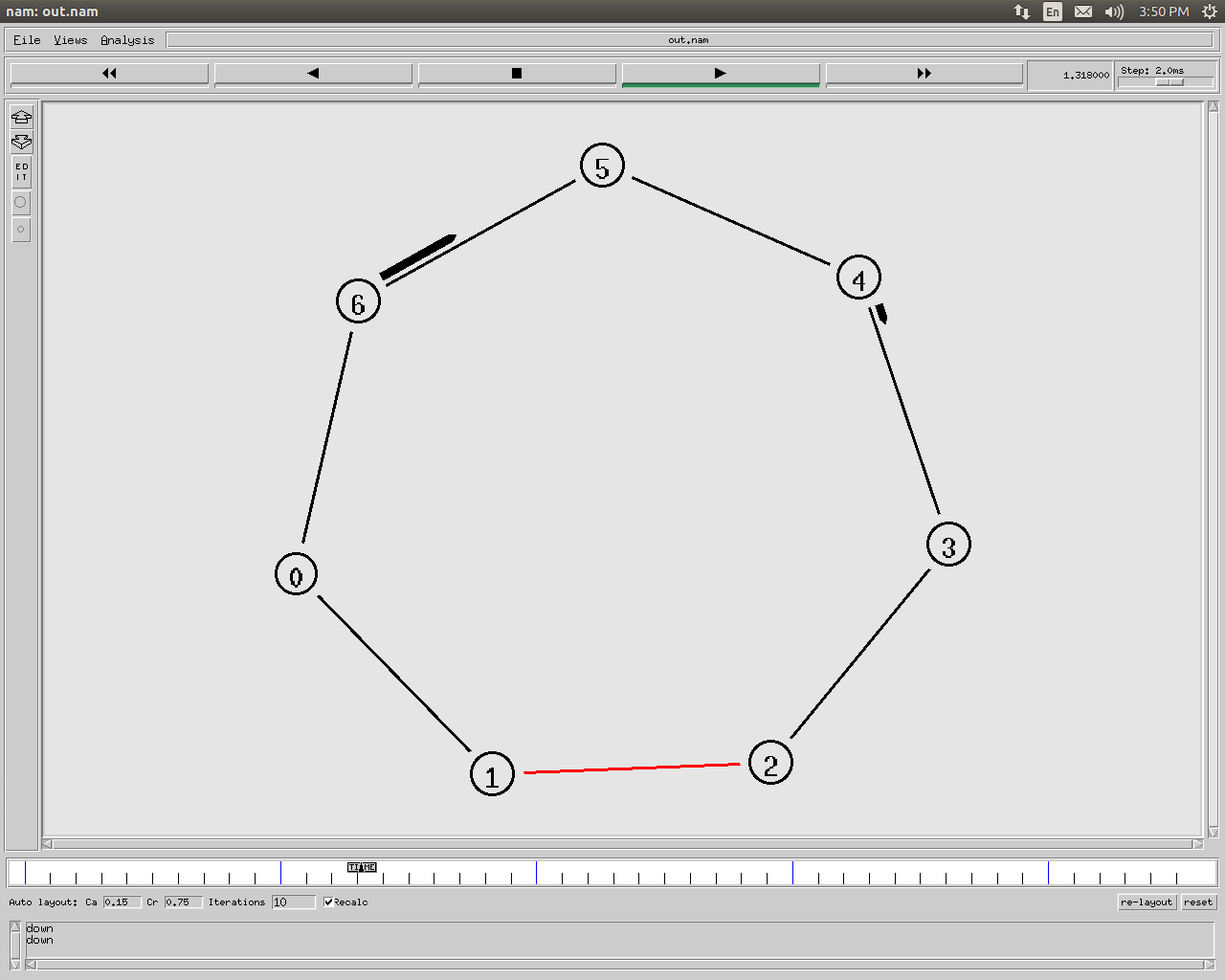
$ns rtmodel-at 3.0 up $n(1) $n(2)

$ns at 4.5 "$cbr0 stop"

$ns at 5.0 "finish"

$ns run

**Output**

****

**ExNo 12 Simulation and Performance analysis of 802.11 with AODV,DSR, routing protocols using simulation tool**

**Aim**

To analyse performance of 802.11 with AODV,DSR,routing protocols using simulation tool.

**Program**

set x 500

set y 500

set n 2

set ns\_ [new Simulator]

set topo [new Topography]

$topo load flatgrid $x $y

set tracefd [open wireless.tr w]

$ns\_ trace-all $tracefd

$ns\_ use-newtrace

set namtrace-all-wireless $namtracefd $x $y

set god\_ [create-god $n]

$ns\_ node-config -adhocRouting AODV \

--llType LL \

-macType Mac/802\_11 \

-ifqType Queue/DropTail/PriQueue \

-ifqLen 50 \

-antType Antenna/OmniAntenna \

-propType Propagation/TwoRayGround \

-phyType Phy/WirelessPhy \

-channelType Channel/WirelessChannel \

-topoInstance $topo \

-agentTrace ON \

-routerTrace ON \

-macTrace OFF \

for {set i 0} { $i < $n } {incr i} {

set node\_($i)[$ns\_ node]

}

$node\_(0) set X\_ 50.0

$node\_(0) set Y\_ 50.0

$node\_(0) set Z\_ 0.0

$node\_(1) set X\_ 200.0

$node\_(1) set Y\_ 150.0

$node\_(1) set Z\_ 0.0

for {set i 0} { $i < $n } {incr i} {

$ns\_ initial\_node\_pos $node\_($i) 30

}

set tcp0 [new Agent/UDP]

$ns\_ attach-agent $node\_(0) $tcp0

set sink0 [new Agent/Null]

$ns\_ attach-agent $node\_(1) $sink0

set cbr0 [new Application/Traffic/CBR]

$cbr0 set packetSize\_ 500

$cbr0 set interval\_ 0.025

$cbr0 attach-agent $tcp0

$ns\_ at 1 "$cbr0 start"

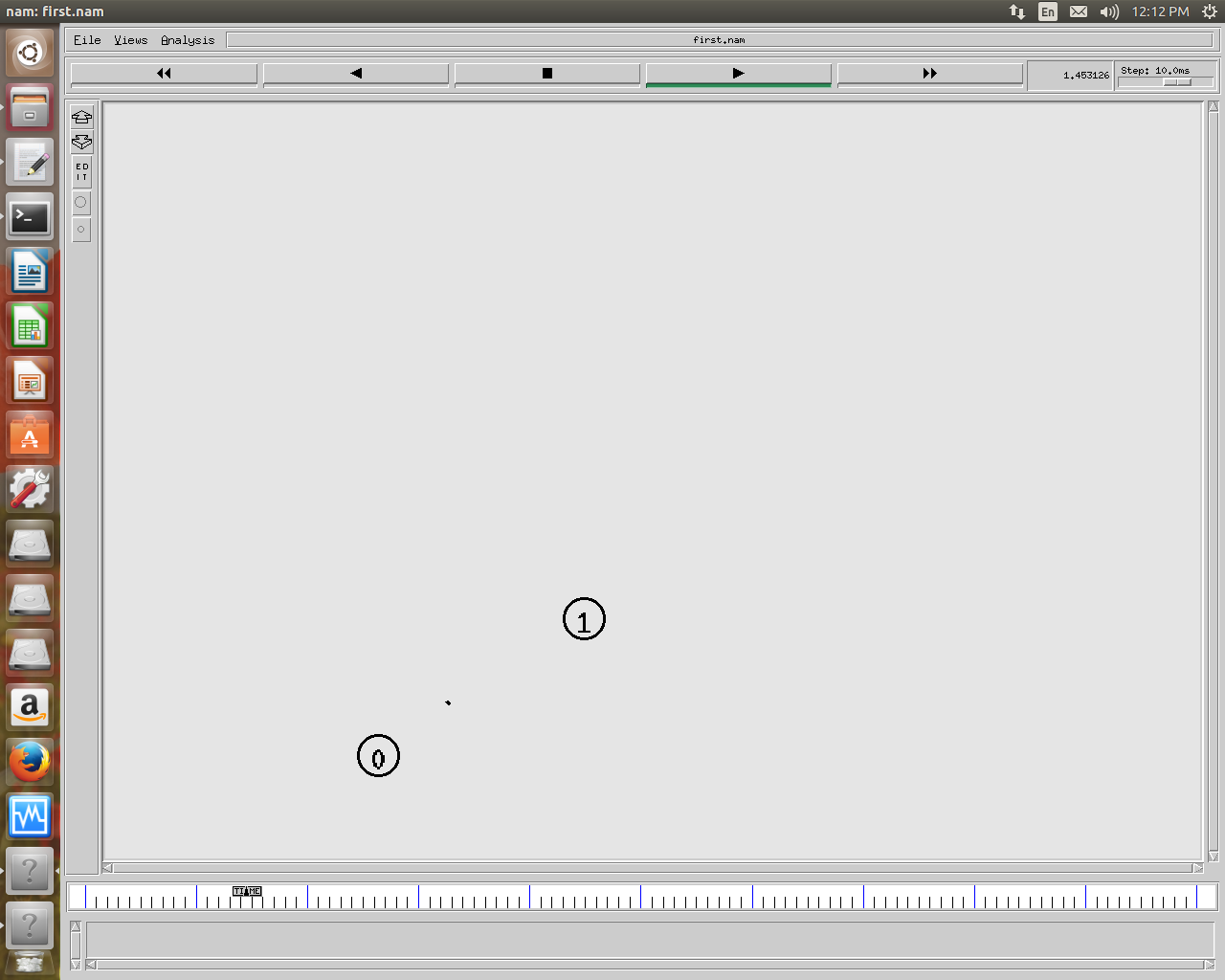
$ns\_ at 5 "$node\_(0) setdest 100 200 10"

$ns\_ at 10 "$cbr0 stop"

$ns\_ at 100.0 "$ns halt"

$ns\_ run

**Output**

****