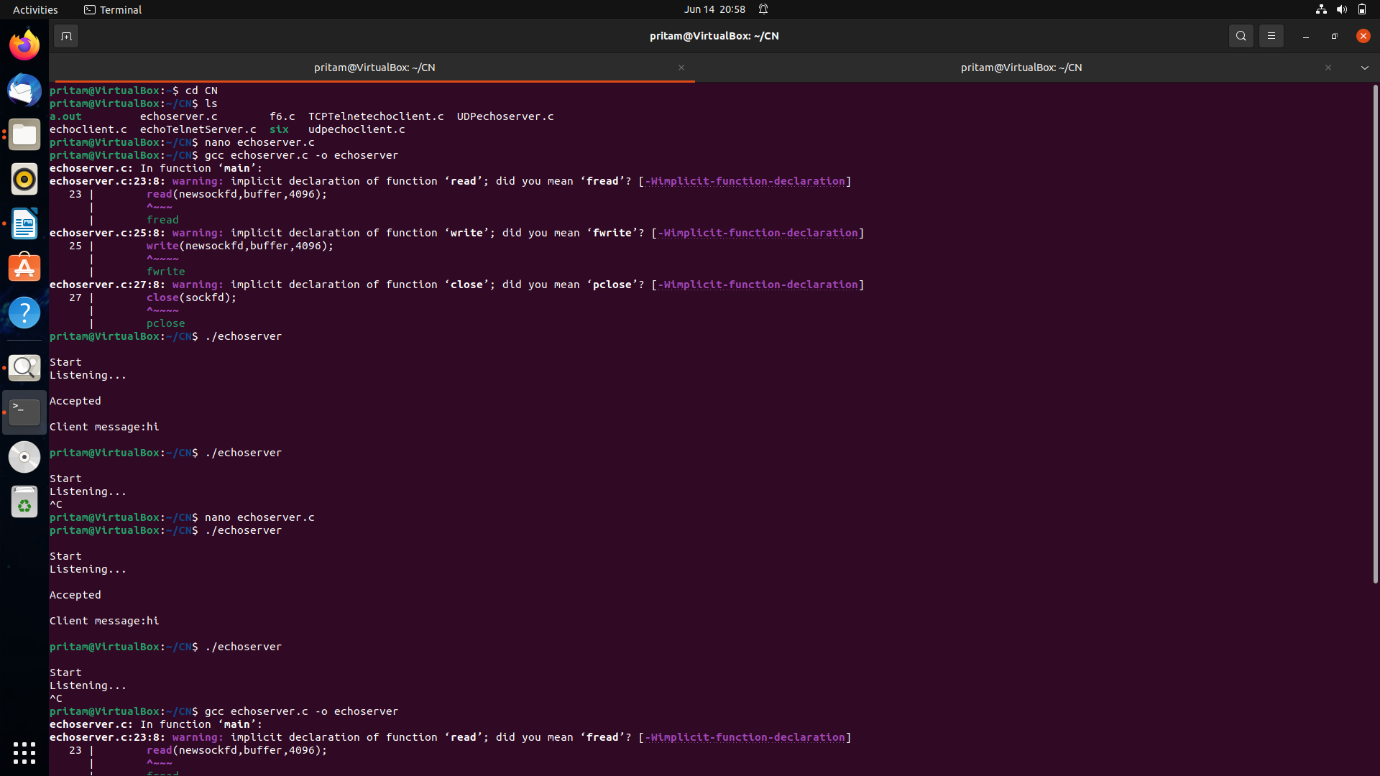
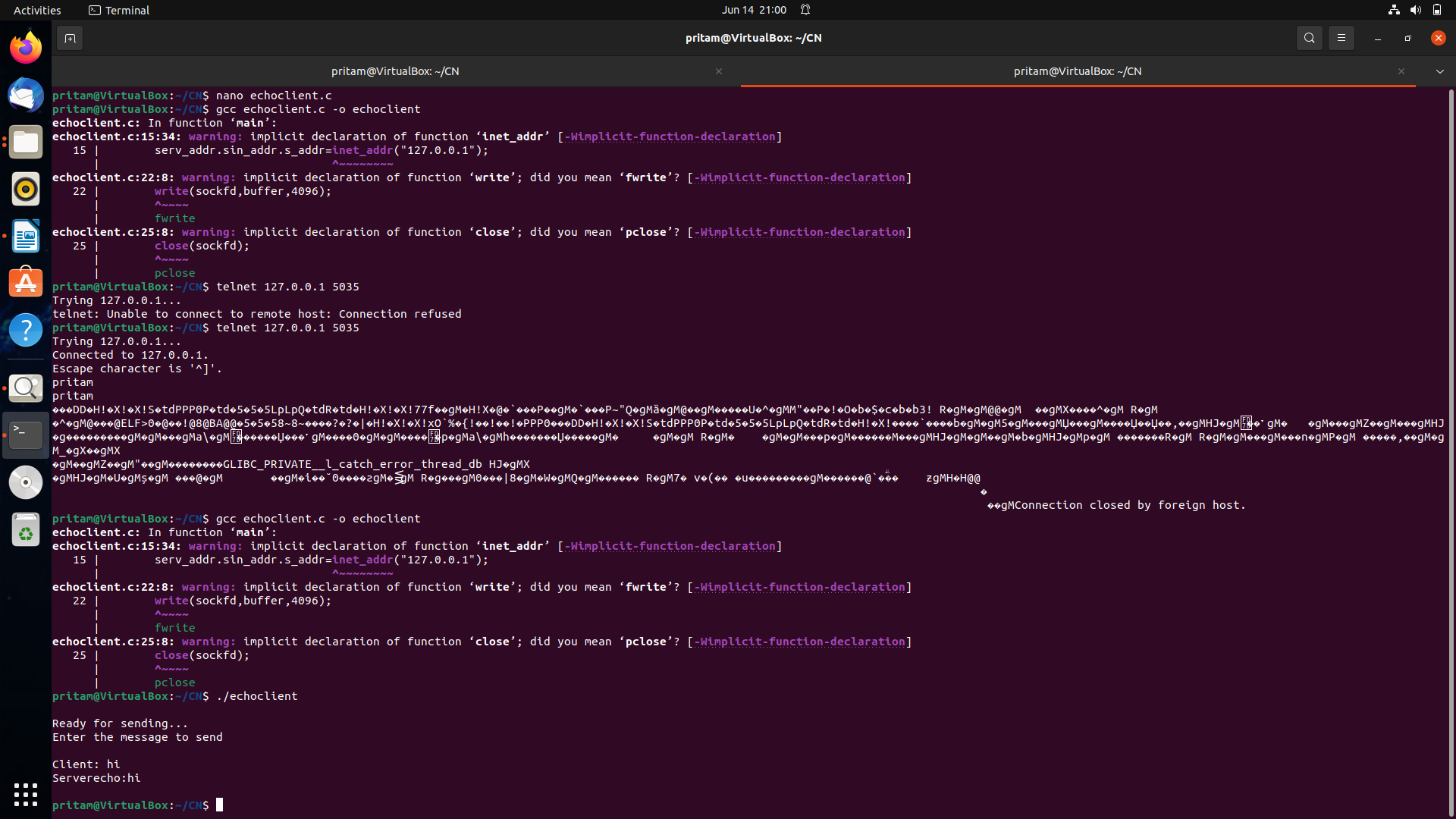
**1.**           To write a program for TCP echo client server.  
  
**SOURCE CODE:  
SERVER:**  
#include<stdio.h>  
#include<netinet/in.h>  
#include<netdb.h>  
#define SERV\_TCP\_PORT 5035  
int main(int argc,char\*\*argv)  
{  
       int sockfd,newsockfd,clength;  
       struct sockaddr\_in serv\_addr,cli\_addr;  
       char buffer[4096];  
       sockfd=socket(AF\_INET,SOCK\_STREAM,0);  
       serv\_addr.sin\_family=AF\_INET;  
       serv\_addr.sin\_addr.s\_addr=INADDR\_ANY;  
       serv\_addr.sin\_port=htons(SERV\_TCP\_PORT);  
       printf("\nStart");  
       bind(sockfd,(struct sockaddr\*)&serv\_addr,sizeof(serv\_addr));  
       printf("\nListening...");  
       printf("\n");  
       listen(sockfd,5);  
       clength=sizeof(cli\_addr);  
       newsockfd=accept(sockfd,(struct sockaddr\*)&cli\_addr,&clength);  
       printf("\nAccepted");  
       printf("\n");  
       read(newsockfd,buffer,4096);  
       printf("\nClient message:%s",buffer);  
       write(newsockfd,buffer,4096);  
       printf("\n");  
       close(sockfd);  
       return 0;  
}

**CLIENT:**  
#include<stdio.h>  
#include<sys/types.h>  
#include<sys/socket.h>  
#include<netinet/in.h>  
#include<netdb.h>  
#define SERV\_TCP\_PORT 5035  
int main(int argc,char\*argv[])  
{  
       int sockfd;  
       struct sockaddr\_in serv\_addr;  
       struct hostent \*server;  
       char buffer[4096];  
       sockfd=socket(AF\_INET,SOCK\_STREAM,0);  
       serv\_addr.sin\_family=AF\_INET;  
       serv\_addr.sin\_addr.s\_addr=inet\_addr("127.0.0.1");  
       serv\_addr.sin\_port=htons(SERV\_TCP\_PORT);  
       printf("\nReady for sending...");  
       connect(sockfd,(struct sockaddr\*)&serv\_addr,sizeof(serv\_addr));  
       printf("\nEnter the message to send\n");  
       printf("\nClient: ");  
       fgets(buffer,4096,stdin);  
       write(sockfd,buffer,4096);  
       printf("Serverecho:%s",buffer);  
       printf("\n");  
       close(sockfd);  
       return 0;  
}  
  
**OUTPUT:  
SERVER:**

  
  
  
  
**CLIENT:**

****

**2.** Write a program to implement a **UDP echo client**. Send a datagram to the serve

**SOURCE CODE:  
SERVER:**  
#include<sys/types.h>

#include<sys/socket.h>

#include<netinet/in.h>

#include<unistd.h>

#include<netdb.h>

#include<stdio.h>

#include<string.h>

#include<arpa/inet.h>

#define MAXLINE 1024

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

{

int sockfd;

int n;

socklen\_t len;

char msg[1024];

struct sockaddr\_in servaddr,cliaddr;

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

bzero(&servaddr,sizeof(servaddr));

servaddr.sin\_family=AF\_INET;

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

servaddr.sin\_port=htons(5035);

printf("\n\n Binded");

bind(sockfd,(struct sockaddr\*)&servaddr,sizeof(servaddr));

printf("\n\n Listening...");

for(;;)

{

printf("\n ");

len=sizeof(cliaddr);

n=recvfrom(sockfd,msg,MAXLINE,0,(struct sockaddr\*)&cliaddr,&len);

printf("\n Client's Message : %s\n",msg);

if(n<6)

perror("send error");

sendto(sockfd,msg,n,0,(struct sockaddr\*)&cliaddr,len);

}

return 0;

}

**CLIENT:**

#include<sys/types.h>

#include<sys/socket.h>

#include<netinet/in.h>

#include<string.h>

#include<arpa/inet.h>

#include<string.h>

#include<arpa/inet.h>

#include<stdio.h>

#define MAXLINE 1024

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

{

int sockfd;

int n;

socklen\_t len;

char sendline[1024],recvline[1024];

struct sockaddr\_in servaddr;

strcpy(sendline,"");

printf("\n Enter the message : ");

scanf("%s",sendline);

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

bzero(&servaddr,sizeof(servaddr));

servaddr.sin\_family=AF\_INET;

servaddr.sin\_addr.s\_addr=inet\_addr("127.0.0.1");

servaddr.sin\_port=htons(5035);

connect(sockfd,(struct sockaddr\*)&servaddr,sizeof(servaddr));

len=sizeof(servaddr);

sendto(sockfd,sendline,MAXLINE,0,(struct sockaddr\*)&servaddr,len);

n=recvfrom(sockfd,recvline,MAXLINE,0,NULL,NULL);

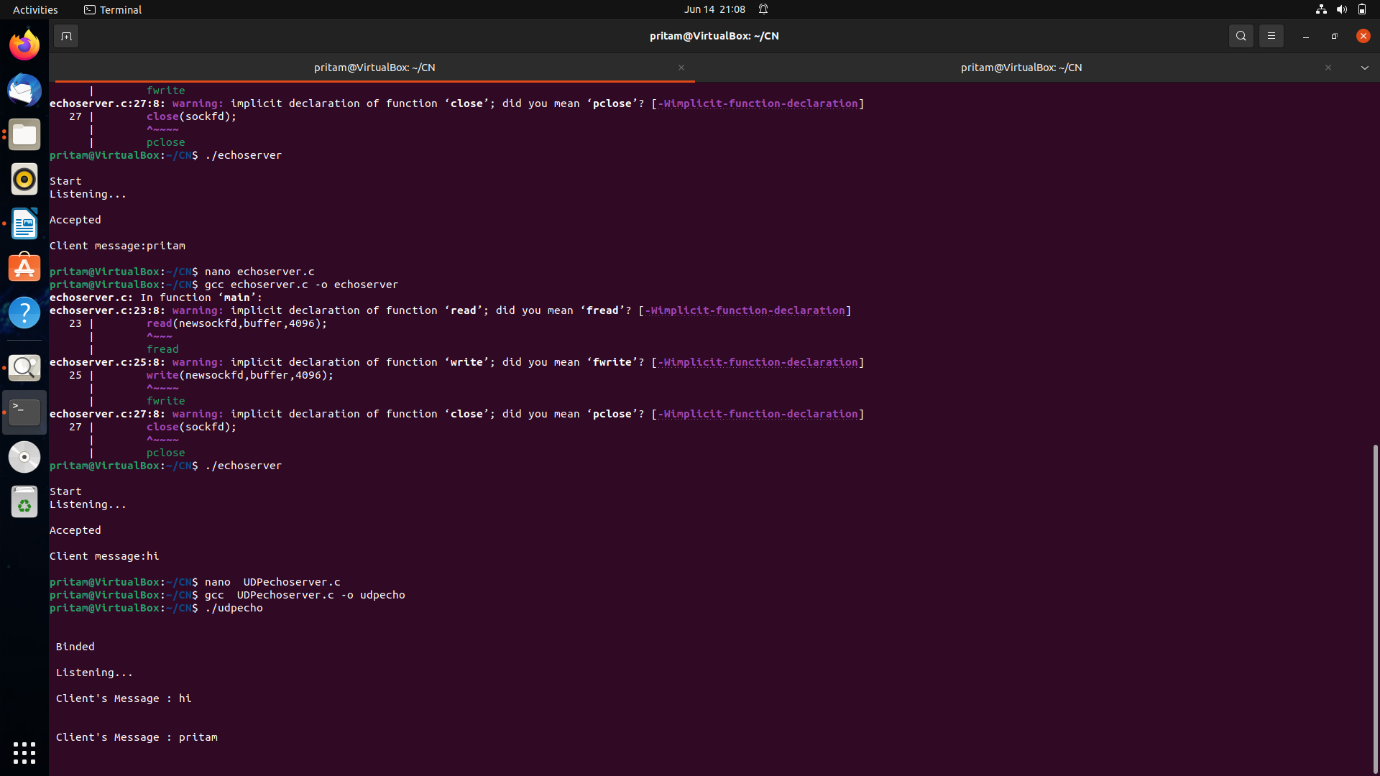
recvline[n]=0;

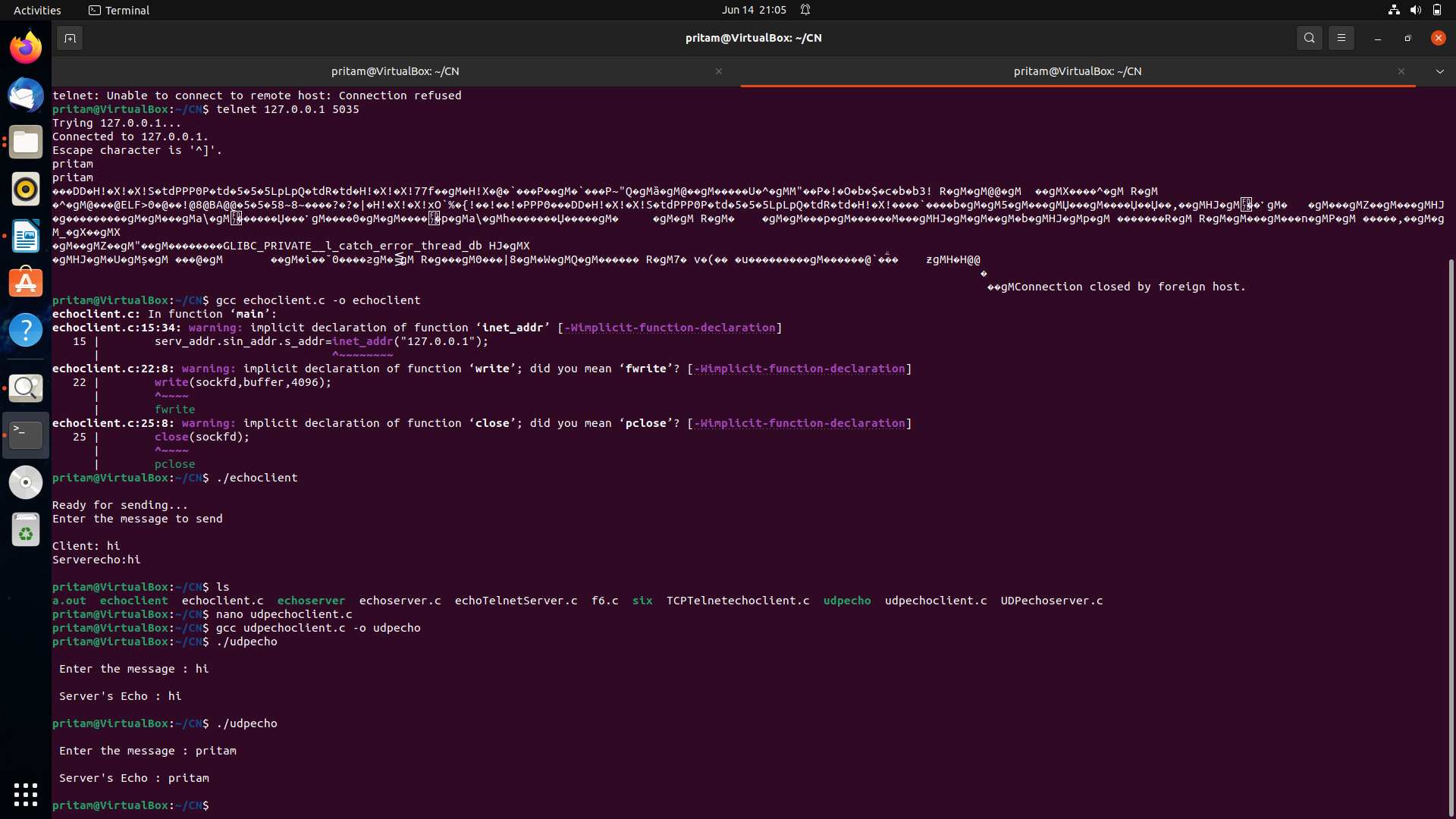
printf("\n Server's Echo : %s\n\n",recvline);

return 0;

}

**OUTPUT:  
SERVER:**

****

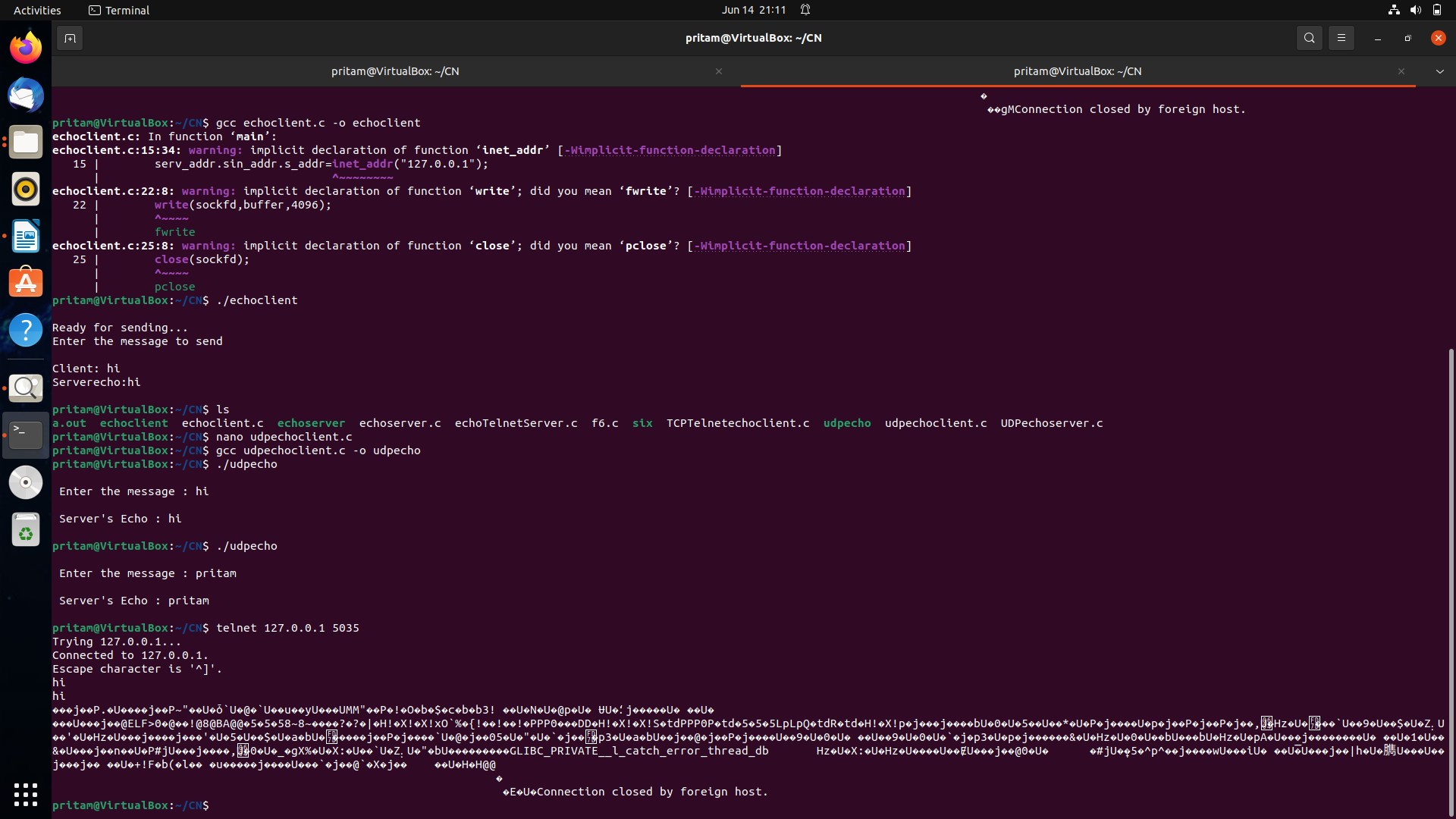
**CLIENT:** 

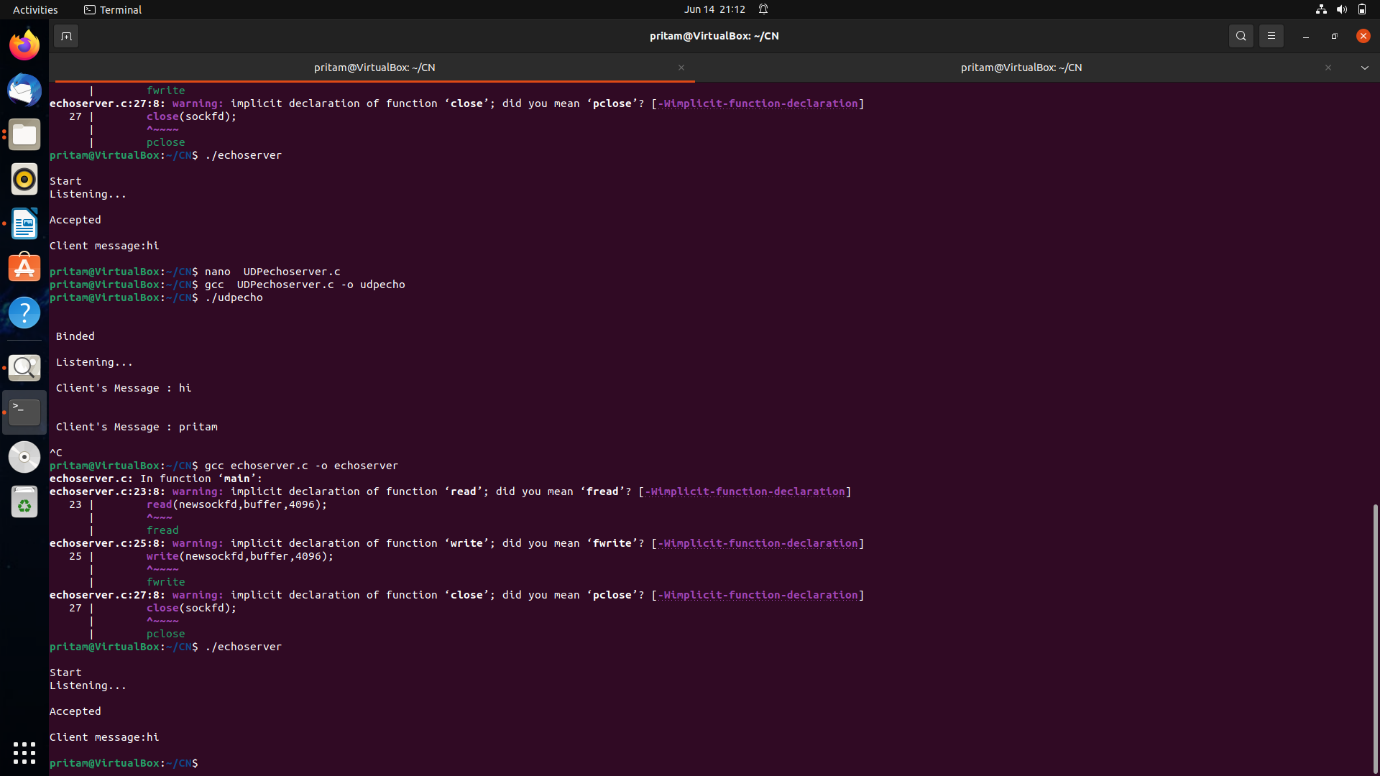
3. Write a program to implement a **TCP echo server**. Connect using telnet.

**SOURCE CODE:  
SERVER:**  
#include<stdio.h>  
#include<netinet/in.h>  
#include<netdb.h>  
#define SERV\_TCP\_PORT 5035  
int main(int argc,char\*\*argv)  
{  
       int sockfd,newsockfd,clength;  
       struct sockaddr\_in serv\_addr,cli\_addr;  
       char buffer[4096];  
       sockfd=socket(AF\_INET,SOCK\_STREAM,0);  
       serv\_addr.sin\_family=AF\_INET;  
       serv\_addr.sin\_addr.s\_addr=INADDR\_ANY;  
       serv\_addr.sin\_port=htons(SERV\_TCP\_PORT);  
       printf("\nStart");  
       bind(sockfd,(struct sockaddr\*)&serv\_addr,sizeof(serv\_addr));  
       printf("\nListening...");  
       printf("\n");  
       listen(sockfd,5);  
       clength=sizeof(cli\_addr);  
       newsockfd=accept(sockfd,(struct sockaddr\*)&cli\_addr,&clength);  
       printf("\nAccepted");  
       printf("\n");  
       read(newsockfd,buffer,4096);  
       printf("\nClient message:%s",buffer);  
       write(newsockfd,buffer,4096);  
       printf("\n");       return 0;  
}

OUTPUT:

T1: telnet host ip post number

T2:response server



4. Write a program to implement a **UDP echo server**. Get the service using telnet.

**SOURCE CODE:  
SERVER:**  
#include<sys/types.h>

#include<sys/socket.h>

#include<netinet/in.h>

#include<unistd.h>

#include<netdb.h>

#include<stdio.h>

#include<string.h>

#include<arpa/inet.h>

#define MAXLINE 1024

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

{

int sockfd;

int n;

socklen\_t len;

char msg[1024];

struct sockaddr\_in servaddr,cliaddr;

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

bzero(&servaddr,sizeof(servaddr));

servaddr.sin\_family=AF\_INET;

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

servaddr.sin\_port=htons(5035);

printf("\n\n Binded");

bind(sockfd,(struct sockaddr\*)&servaddr,sizeof(servaddr));

printf("\n\n Listening...");

for(;;)

{

printf("\n ");

len=sizeof(cliaddr);

n=recvfrom(sockfd,msg,MAXLINE,0,(struct sockaddr\*)&cliaddr,&len);

printf("\n Client's Message : %s\n",msg);

if(n<6)

perror("send error");

sendto(sockfd,msg,n,0,(struct sockaddr\*)&cliaddr,len);

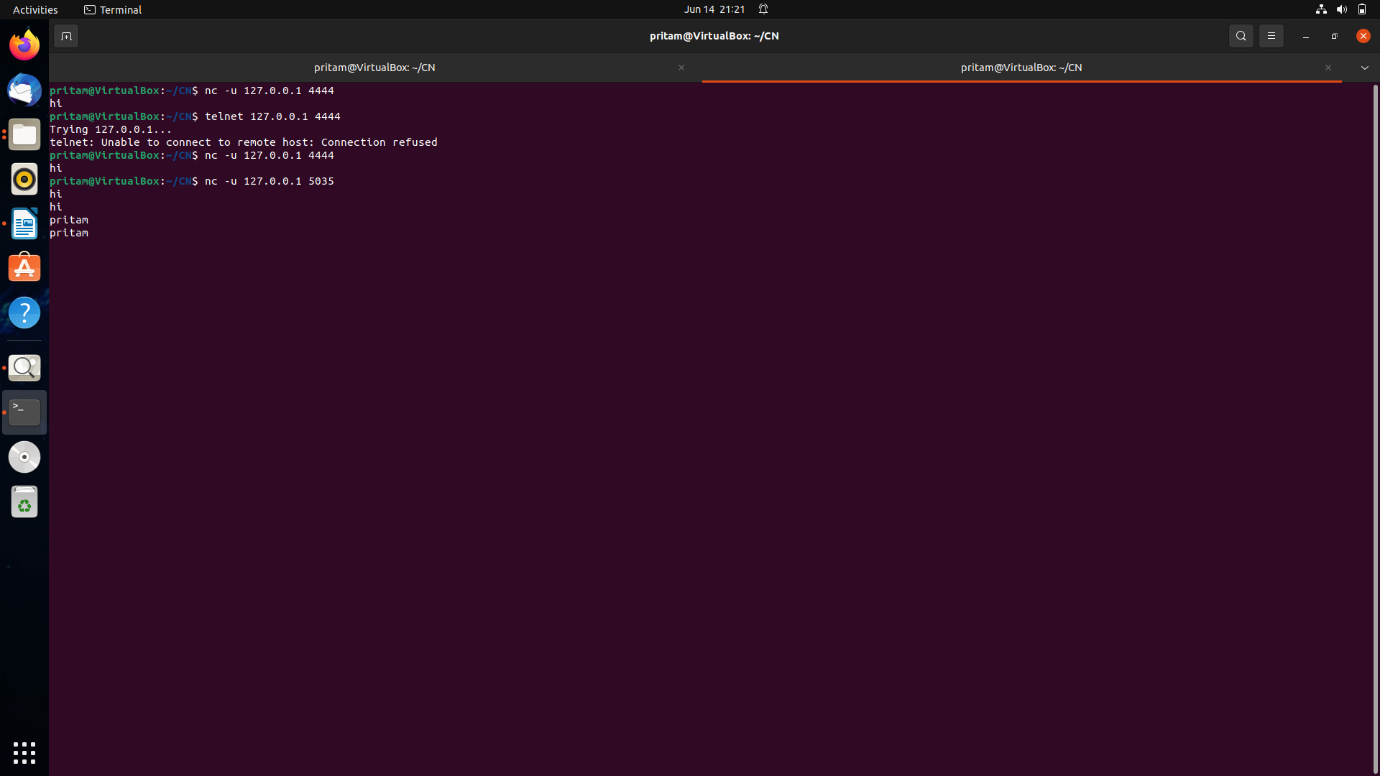
}

return 0;

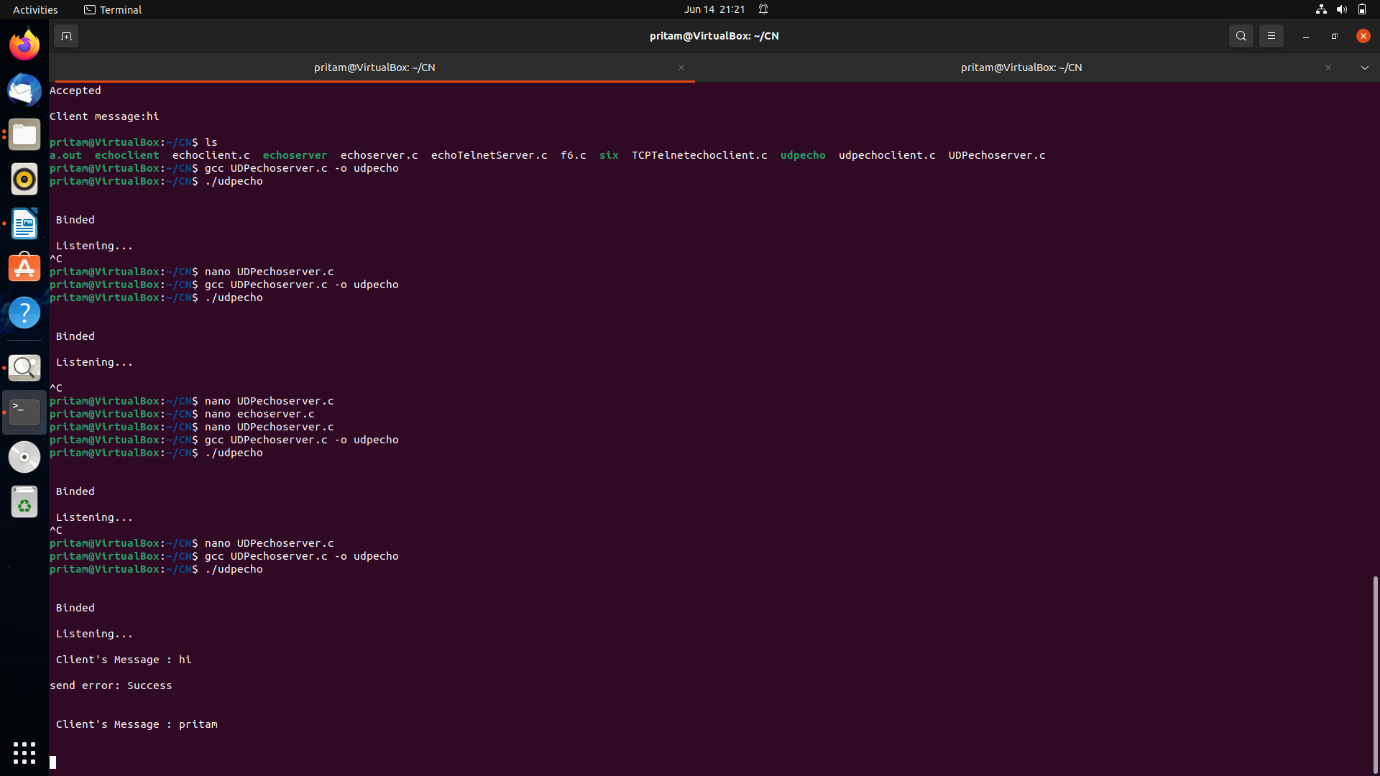
}

OUTPUT:

T1: telnet host ip post number



T2:response server



7. Write a program to implement a **TCP daytime client.** Connect to the server.

**SOURCE CODE:  
SERVER**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <time.h>

#include <wait.h>

#define PORT 50001

#define BUFFERSIZE 50 // must be greater than 26

void child\_handler( int signumber );

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

{

int ser\_sd, temp\_sd, n ;

struct sockaddr\_in server\_addr;

pid\_t pid;

time\_t now;

char buffer[BUFFERSIZE];

ssize\_t i;

struct sigaction act;

// SIGCHLD handler is used to reap zombie

act.sa\_handler = child\_handler;

act.sa\_flags = SA\_RESTART;

sigaction( SIGCHLD, &act, NULL );

if( argc != 1 )

{

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

exit(1);

}

// IPv4 TCP socket is created.

ser\_sd = socket( PF\_INET, SOCK\_STREAM, IPPROTO\_TCP );

if( ser\_sd == -1 ) { perror("socket-call"); exit(1); }

// members of server address are initialised

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

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

server\_addr.sin\_port = htons( PORT );

// socket is bound to [server\_addr]

n = bind( ser\_sd, (struct sockaddr \*) &server\_addr,

sizeof(server\_addr) );

if( n == -1 ) { perror("bind-call"); exit(1); }

n = listen( ser\_sd, 5 );

if( n == -1 ) { perror("listen-call"); exit(1); }

while (1)

{

printf("%s: daytime server listening on TCP port %u\n",

argv[0], PORT );

temp\_sd = accept( ser\_sd, NULL, 0 );

if( temp\_sd == -1 ) perror("accept-call");

pid = fork();

if( pid == -1 ) {perror("fork-call"); exit(1); }

if( pid == 0 ) // child's block

{

memset( buffer, '\0', BUFFERSIZE );

time( &now );

strcpy( buffer, ctime( &now ) );

// now [buffer] contains time in human readable form

i = write( temp\_sd, buffer, strlen(buffer) );

if( i == -1 ) { perror("socket-write"); exit(1); }

shutdown( temp\_sd, SHUT\_RDWR );

close( temp\_sd );

return 0;

} // child's block

// parent's block

// useless descriptor is closed

close( temp\_sd );

} // endless-loop block

} // end of main

// signal handler

void child\_handler( int signumber )

{

pid\_t p;

printf("signal-handler: received %u \n", signumber );

p = waitpid( -1, NULL, WNOHANG );

if( p == -1 ) perror("waitpid");

return;

}

**CLIENT:**

#include <sys/types.h>

#include <sys/socket.h>

#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include <stdlib.h>

#include <arpa/inet.h>

#define BUFFERSIZE 40 // must be greater than 26

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

{

int ser\_sd,n;

ssize\_t i,j;

char buffer[BUFFERSIZE];

struct sockaddr\_in server\_addr;

if( argc != 3 )

{

printf( "Usage: %s server-address server-port\n", argv[0] );

exit(1);

}

ser\_sd = socket( PF\_INET, SOCK\_STREAM, IPPROTO\_TCP );

if( ser\_sd == -1 ) { perror("socket-call"); exit( 1 ); }

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

n = inet\_aton( argv[1], &(server\_addr.sin\_addr) );

if( n == 0 ) { printf("Invalid address\n"); exit(1); }

server\_addr.sin\_port = htons( atoi(argv[2]) );

n = connect( ser\_sd, (struct sockaddr \*) &server\_addr,

sizeof(server\_addr) );

if( n == -1 ) { perror("connect-call"); exit(1); }

// nothing needs to be written on server's socket.

memset( buffer, '\0', BUFFERSIZE );

i = read( ser\_sd, buffer, BUFFERSIZE );

if( i == -1 ) { perror("read"); exit(1); }

printf("bytes read from server's socket=%u\n", i );

write( STDOUT\_FILENO,"Reply from server->", 19 );

j = write( STDOUT\_FILENO, buffer, i );

if( j == -1 ) { perror("write"); exit(1); }

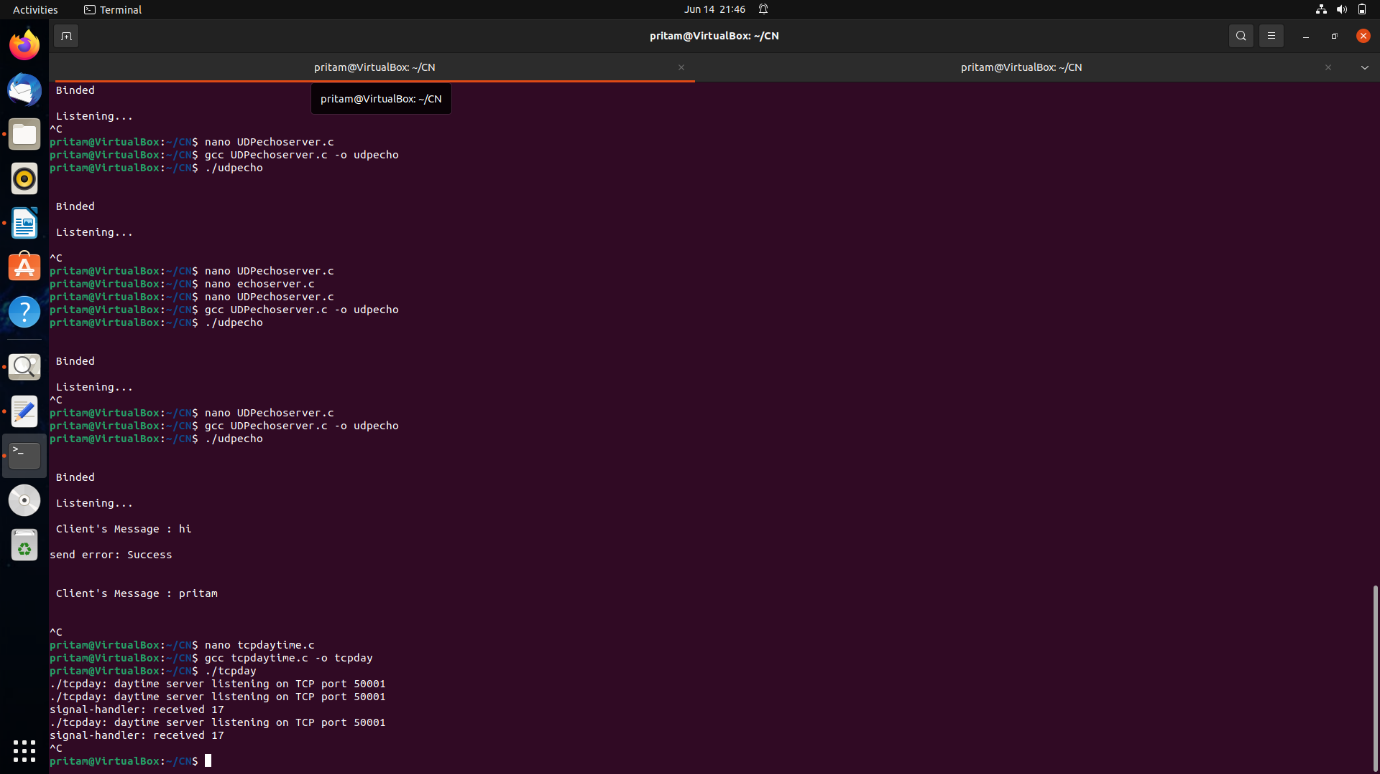
shutdown(ser\_sd, SHUT\_RDWR);

close(ser\_sd);

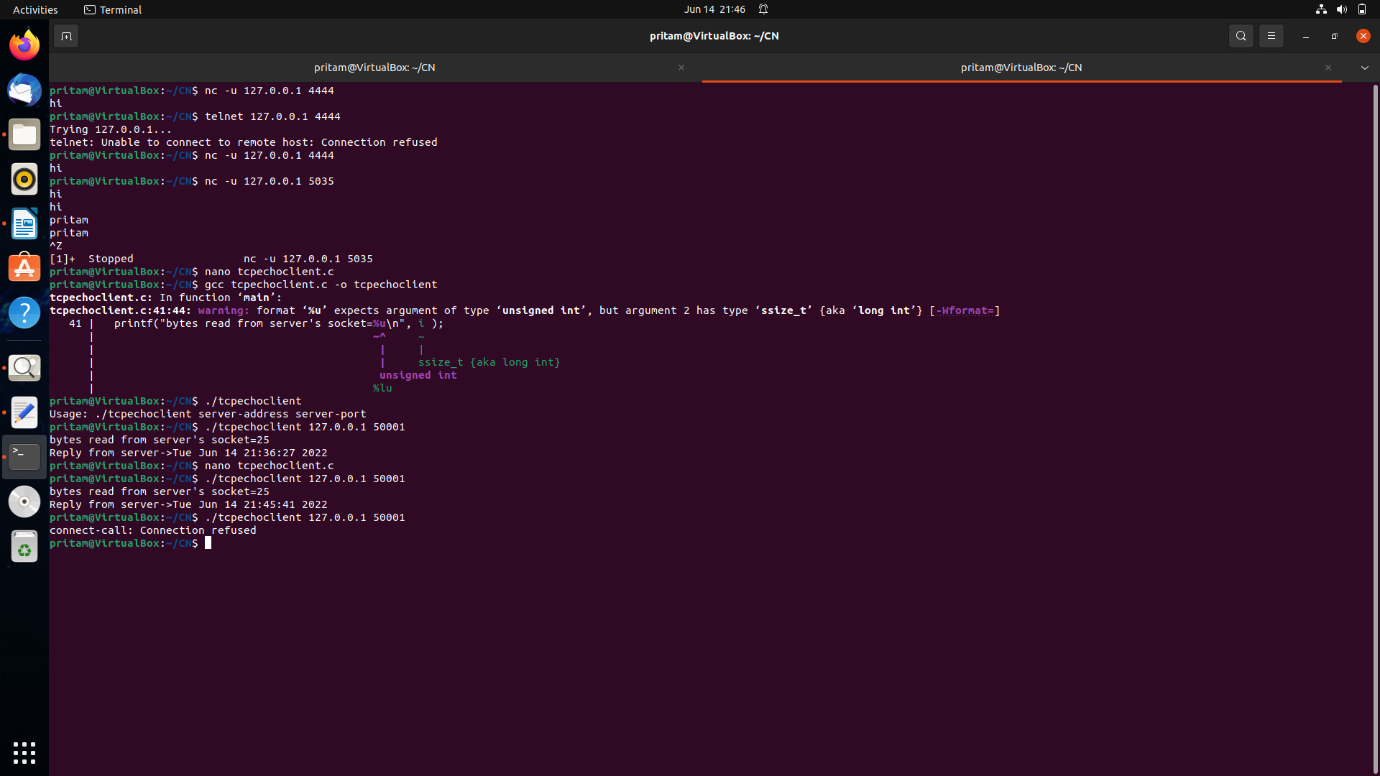
return 0;

}

**OUTPUT:  
SERVER:**

****

**CLIENT:**

****

7. Write a program to implement a **UDP daytime client.** Send a datagram to the server.

**SOURCE CODE:  
SERVER**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <time.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#define PORT 50002

#define BUFFERSIZE 26

// should be 26 or more

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

{

int ser\_sd, n, i,j ;

struct sockaddr\_in server\_addr, client\_addr;

socklen\_t clientlength;

time\_t now;

char buffer[BUFFERSIZE];

if( argc != 1 )

{ printf("usage: %s\n", argv[0]); exit(1); }

// UDP socket is created

ser\_sd = socket( PF\_INET, SOCK\_DGRAM, IPPROTO\_UDP );

if( ser\_sd == -1 ) { perror("socket-call"); exit(1); }

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

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

server\_addr.sin\_port = htons(PORT);

// bind socket with address of server ( local endpoint )

n = bind( ser\_sd, (struct sockaddr \*) & server\_addr,

sizeof(server\_addr) );

clientlength = sizeof(client\_addr);

while( 1 ) // endless loop

{

printf( "%s: daytime sever on UDP port %u \n",

argv[0], PORT );

i = recvfrom( ser\_sd, buffer, BUFFERSIZE,0,

(struct sockaddr \*) &client\_addr,

&clientlength );

if( i == -1 ) perror("recvfrom-call");

/\*

contents of buffer are useless to daytime server.

[ recvfrom ] call stores client's address in

structure [ client\_addr ]. This is used by the

server to send a datagram to a client.

\*/

memset( buffer, '\0', BUFFERSIZE );

time( &now );

strcpy( buffer, ctime( &now ) );

/\*

// printing client's address and port are

// not essential for server operation.

printf("client's IP address = %s \n",

inet\_ntoa(client\_addr.sin\_addr) );

printf("client's port = %u \n",

ntohs(client\_addr.sin\_port) );

\*/

i = strlen(buffer);

j = sendto( ser\_sd, buffer, i, 0,

(struct sockaddr \*) &client\_addr, clientlength);

if( j == -1 ) perror("sendto-call");

} // endless-loop

}

**Client:**

#include <sys/socket.h>

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <arpa/inet.h>

#define SIZE 26 // 26 or more

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

{

int sd, n;

ssize\_t i;

char buffer[SIZE];

struct sockaddr\_in server\_addr;

struct sockaddr\_in local\_addr;

if( argc != 3 )

{

printf("usage -> prog-name server-address server-port \n");

printf("example -> %s 172.16.4.140 7 \n", argv[0] );

exit(1);

}

// an IPv4 datagram socket ( UDP socket ) is created

sd = socket( AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP );

// zero may be used for IPPROTO\_UDP

if( sd == -1 ) { perror("socket-call"); exit(1); }

// initialise members of [local\_addr]

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

local\_addr.sin\_addr.s\_addr = htonl( INADDR\_ANY ); // all interfaces

local\_addr.sin\_port = htons(0); // any free port

// Bind the local address with the socket. This is needed as we are

// interested in a reply from the server. The server needs to know

// our address to send a reply.

n = bind( sd, (struct sockaddr \*) &local\_addr,sizeof(local\_addr) );

if( n == -1 ) { perror("bind-call"); exit(1); }

// Initialise the members of server address

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

n = inet\_aton( argv[1], &(server\_addr.sin\_addr) );

if( n == 0 ) { perror("invalid-address"); exit(1); }

server\_addr.sin\_port = htons( atoi(argv[2]) );

// send zero byte (3rd argument) to server. The daytime

// server does not use the data of the datagram.

n = sendto( sd, buffer, 0, 0,

( struct sockaddr \*) &server\_addr,

sizeof( server\_addr) );

if( n == -1 ) { perror("sendto-call"); exit(1); }

memset( buffer, '\0', SIZE );

i = read( sd, buffer, SIZE);

if( i == -1 ) { perror("read-call"); exit(1); }

printf("u-e-c: Received %u bytes from server \n", i );

write( STDOUT\_FILENO, "u-e-c: From server->", 20 );

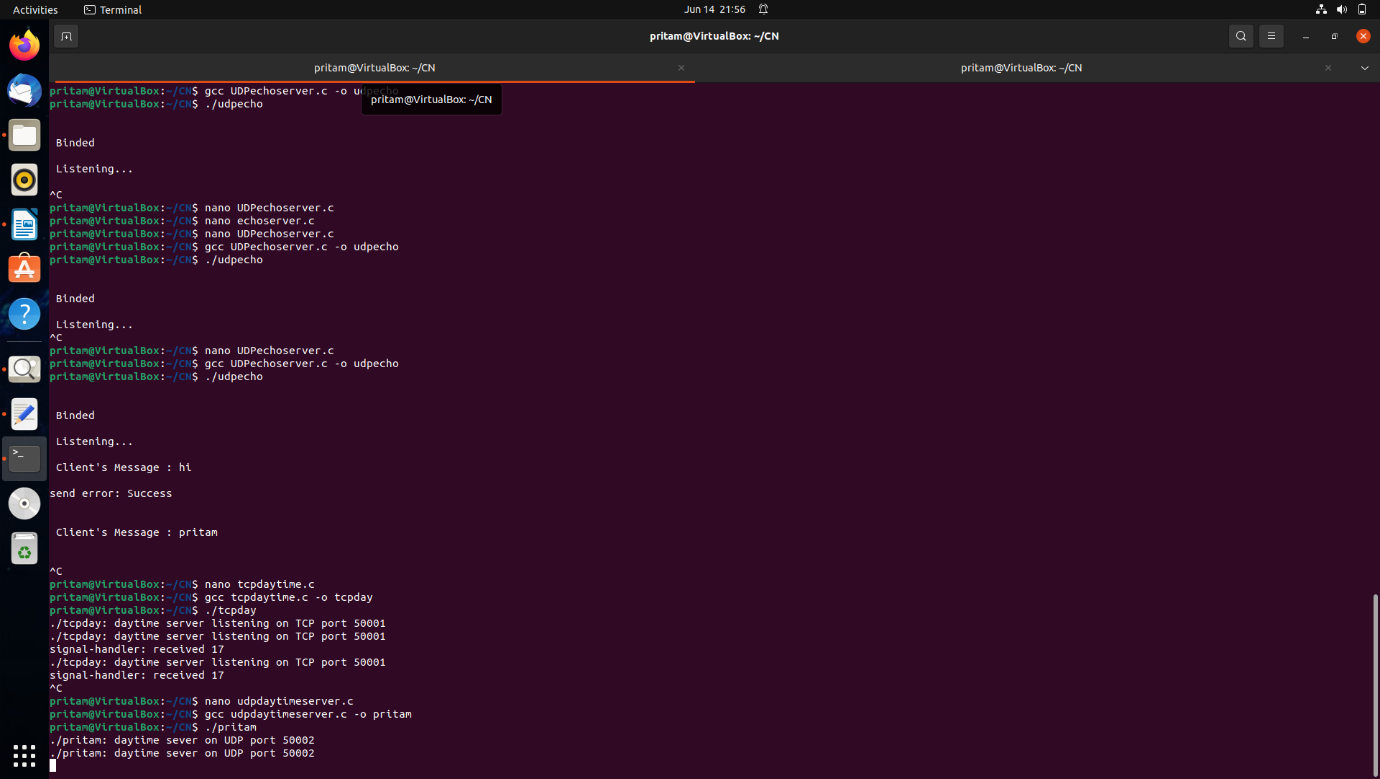
write( STDOUT\_FILENO, buffer, i );

shutdown( sd, SHUT\_RDWR );

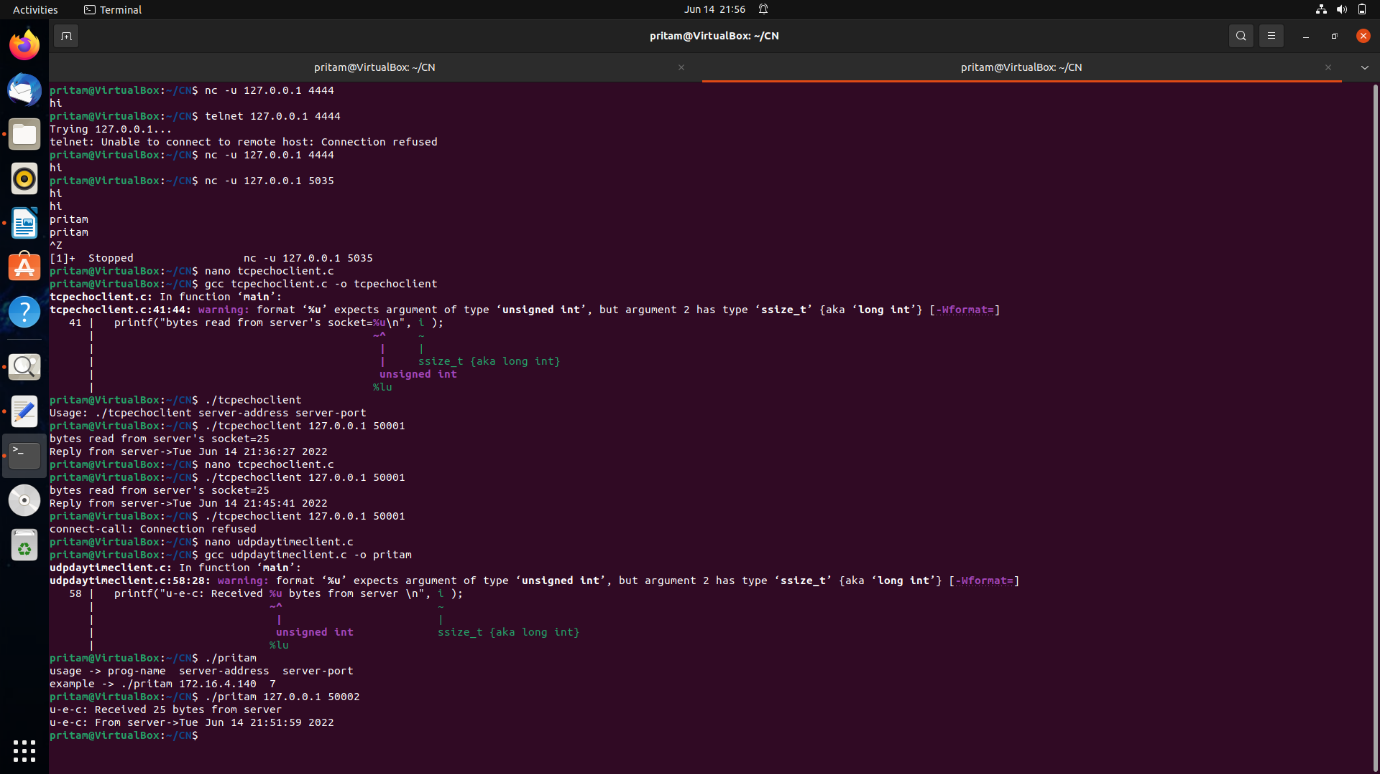
return 0;

}

**OUTPUT:  
SERVER:**

****

**CLIENT:**

****

9. Write a program to implement a **TCP daytime concurrent server.** Connect using telnet.

**SOURCE CODE:  
SERVER**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <time.h>

#include <wait.h>

#define PORT 50001

#define BUFFERSIZE 50 // must be greater than 26

void child\_handler( int signumber );

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

{

int ser\_sd, temp\_sd, n ;

struct sockaddr\_in server\_addr;

pid\_t pid;

time\_t now;

char buffer[BUFFERSIZE];

ssize\_t i;

struct sigaction act;

// SIGCHLD handler is used to reap zombie

act.sa\_handler = child\_handler;

act.sa\_flags = SA\_RESTART;

sigaction( SIGCHLD, &act, NULL );

if( argc != 1 )

{

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

exit(1);

}

// IPv4 TCP socket is created.

ser\_sd = socket( PF\_INET, SOCK\_STREAM, IPPROTO\_TCP );

if( ser\_sd == -1 ) { perror("socket-call"); exit(1); }

// members of server address are initialised

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

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

server\_addr.sin\_port = htons( PORT );

// socket is bound to [server\_addr]

n = bind( ser\_sd, (struct sockaddr \*) &server\_addr,

sizeof(server\_addr) );

if( n == -1 ) { perror("bind-call"); exit(1); }

n = listen( ser\_sd, 5 );

if( n == -1 ) { perror("listen-call"); exit(1); }

while (1)

{

printf("%s: daytime server listening on TCP port %u\n",

argv[0], PORT );

temp\_sd = accept( ser\_sd, NULL, 0 );

if( temp\_sd == -1 ) perror("accept-call");

pid = fork();

if( pid == -1 ) {perror("fork-call"); exit(1); }

if( pid == 0 ) // child's block

{

memset( buffer, '\0', BUFFERSIZE );

time( &now );

strcpy( buffer, ctime( &now ) );

// now [buffer] contains time in human readable form

i = write( temp\_sd, buffer, strlen(buffer) );

if( i == -1 ) { perror("socket-write"); exit(1); }

shutdown( temp\_sd, SHUT\_RDWR );

close( temp\_sd );

return 0;

} // child's block

// parent's block

// useless descriptor is closed

close( temp\_sd );

} // endless-loop block

} // end of main

// signal handler

void child\_handler( int signumber )

{

pid\_t p;

printf("signal-handler: received %u \n", signumber );

p = waitpid( -1, NULL, WNOHANG );

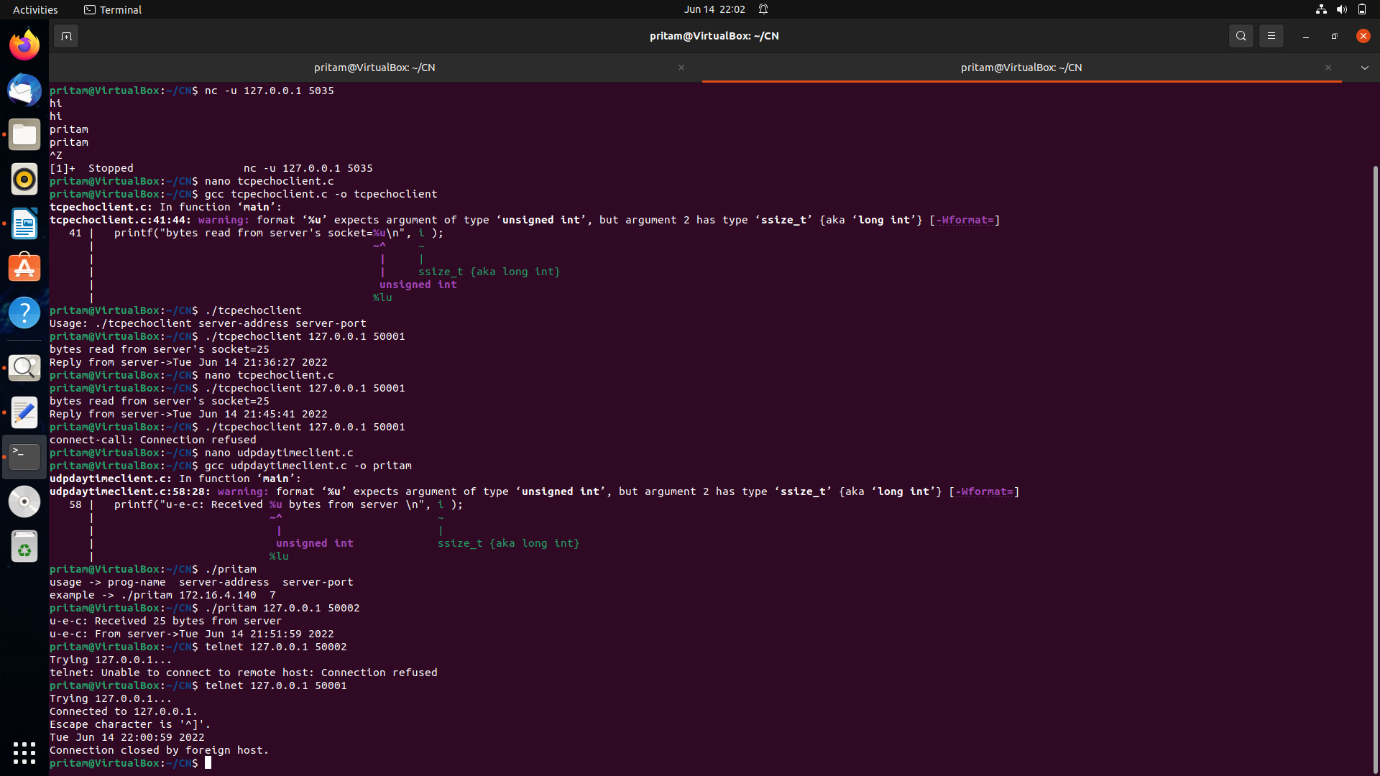
if( p == -1 ) perror("waitpid");

return;

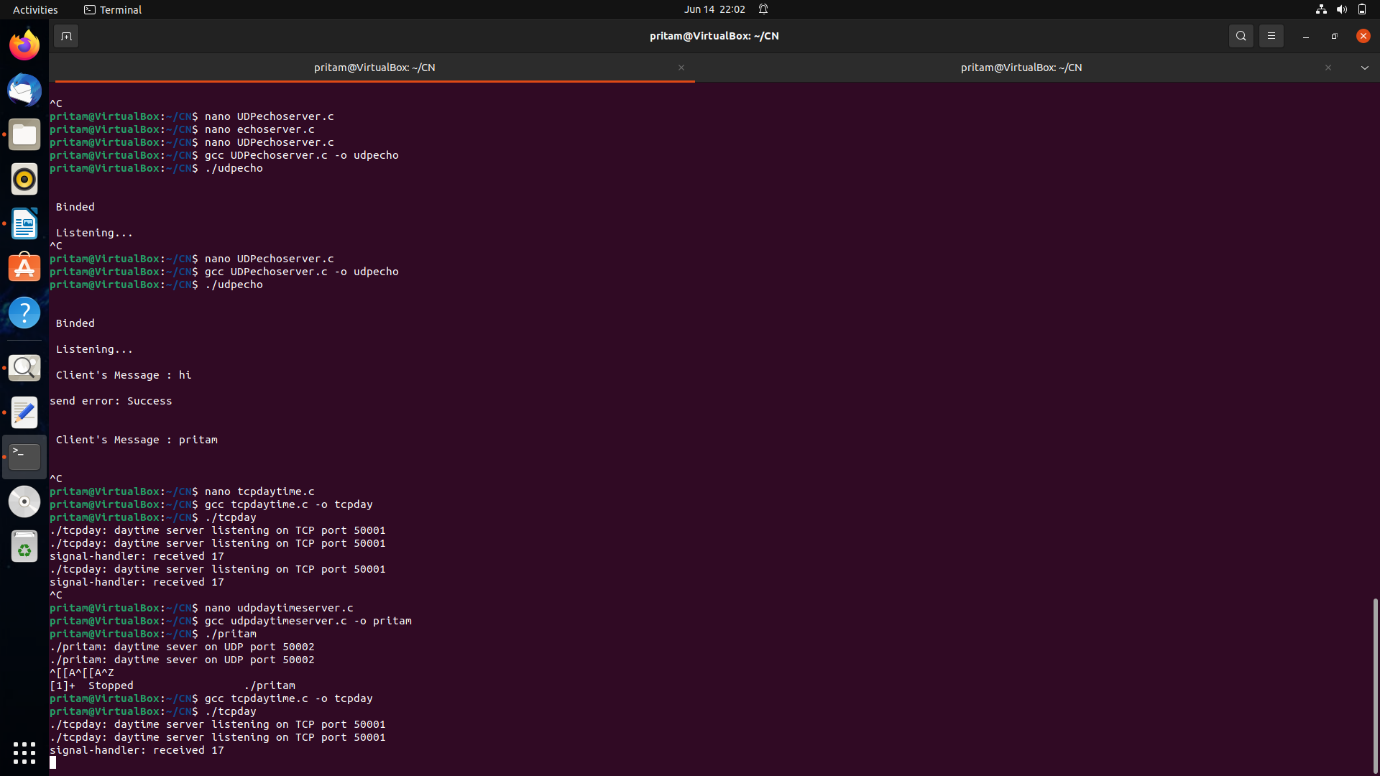
}

OUTPUT:

T1: telnet host ip post number



T2:response server



10. Write a program to implement a **UDP daytime concurrent server.** Send a datagram using telnet.

**SOURCE CODE:  
SERVER**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <time.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#define PORT 50002

#define BUFFERSIZE 26

// should be 26 or more

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

{

int ser\_sd, n, i,j ;

struct sockaddr\_in server\_addr, client\_addr;

socklen\_t clientlength;

time\_t now;

char buffer[BUFFERSIZE];

if( argc != 1 )

{ printf("usage: %s\n", argv[0]); exit(1); }

// UDP socket is created

ser\_sd = socket( PF\_INET, SOCK\_DGRAM, IPPROTO\_UDP );

if( ser\_sd == -1 ) { perror("socket-call"); exit(1); }

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

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

server\_addr.sin\_port = htons(PORT);

// bind socket with address of server ( local endpoint )

n = bind( ser\_sd, (struct sockaddr \*) & server\_addr,

sizeof(server\_addr) );

clientlength = sizeof(client\_addr);

while( 1 ) // endless loop

{

printf( "%s: daytime sever on UDP port %u \n",

argv[0], PORT );

i = recvfrom( ser\_sd, buffer, BUFFERSIZE,0,

(struct sockaddr \*) &client\_addr,

&clientlength );

if( i == -1 ) perror("recvfrom-call");

/\*

contents of buffer are useless to daytime server.

[ recvfrom ] call stores client's address in

structure [ client\_addr ]. This is used by the

server to send a datagram to a client.

\*/

memset( buffer, '\0', BUFFERSIZE );

time( &now );

strcpy( buffer, ctime( &now ) );

/\*

// printing client's address and port are

// not essential for server operation.

printf("client's IP address = %s \n",

inet\_ntoa(client\_addr.sin\_addr) );

printf("client's port = %u \n",

ntohs(client\_addr.sin\_port) );

\*/

i = strlen(buffer);

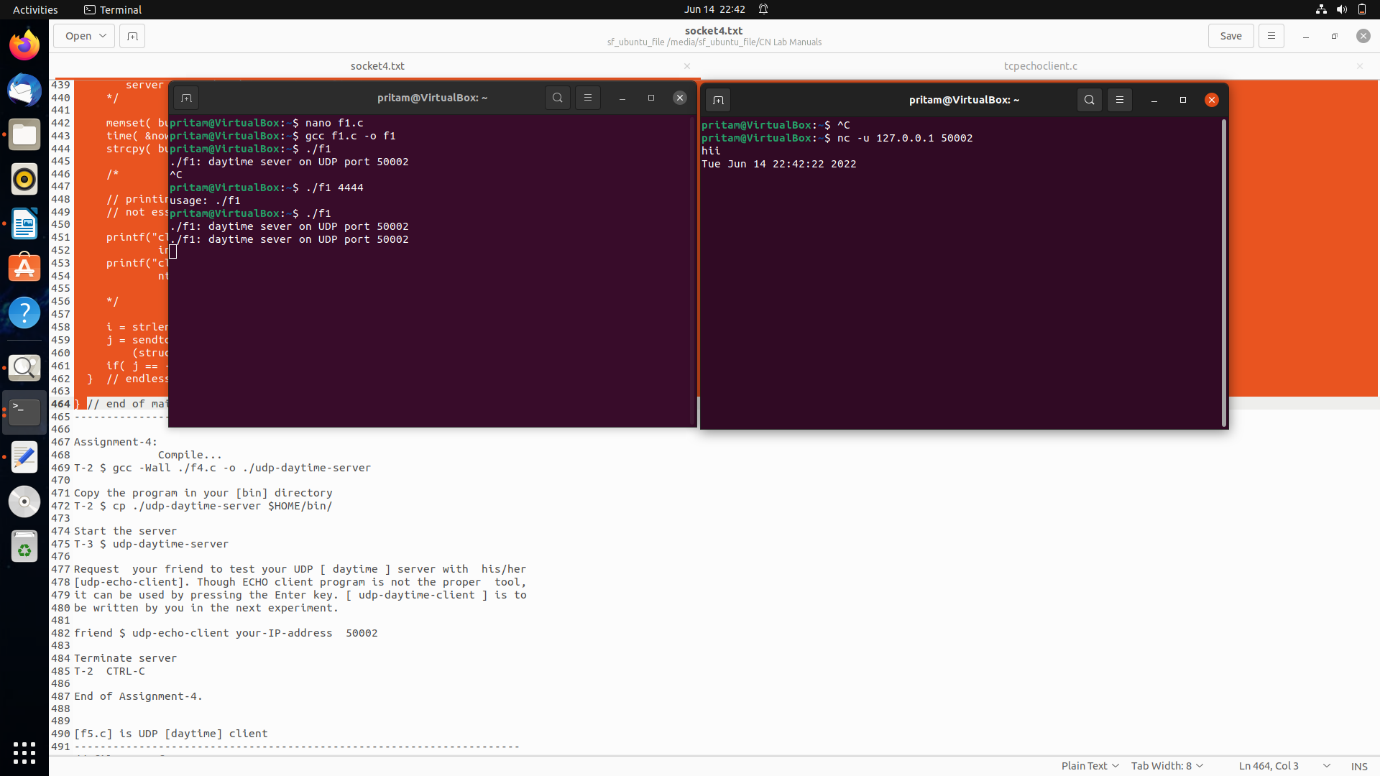
j = sendto( ser\_sd, buffer, i, 0,

(struct sockaddr \*) &client\_addr, clientlength);

if( j == -1 ) perror("sendto-call");

} // endless-loop

}



5. Write a program to implement a **TCP echo concurrent server.** Connect using telnet.

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <signal.h>

#include <wait.h>

#include <errno.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#define SERVER\_PORT 65535

#define BUFFERSIZE 1024

void pipe\_handler( int signumber );

void child\_handler( int signumber );

extern int errno;

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

{

int ser\_sd;

int tempsockfd, n ;

ssize\_t i, j ;

char buffer[BUFFERSIZE];

struct sockaddr\_in server\_addr;

struct sigaction act;

act.sa\_handler = child\_handler;

act.sa\_flags = SA\_RESTART;

sigaction( SIGCHLD, &act, NULL );

ser\_sd = socket( PF\_INET, SOCK\_STREAM, IPPROTO\_TCP );

if( ser\_sd == -1 ) { perror("socket-call"); exit(1); }

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

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

server\_addr.sin\_port = htons(SERVER\_PORT);

n = bind( ser\_sd, (struct sockaddr \*) & server\_addr,

sizeof(server\_addr) );

if( n == -1) { perror("bind-call"); exit(1); }

n = listen( ser\_sd, 1 );

if( n == -1 ) { perror("listen-call"); exit(1); }

while ( 1 ) // endless loop

{

pid\_t pid;

printf( "\n %s : waiting for client's request on port %u \n",

argv[0], SERVER\_PORT );

tempsockfd = accept( ser\_sd, NULL, 0 );

if( tempsockfd == -1 ) perror("accept-call");

// server( main ) forks a child to handle a client's request

pid = fork();

if ( pid == -1 ) { perror("fork-call"); exit(1); }

if ( pid == 0 ) // child process handles a client's request

{

// clear buffer before reading into it

memset( buffer, '\0', BUFFERSIZE );

i = read( tempsockfd, buffer, BUFFERSIZE );

if( i == -1 ) { perror("tempsockfd-read"); exit(1); }

j = write( STDOUT\_FILENO,"Received from client->", 22 );

if( j == -1 ) { perror("screen-write"); exit(1); }

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

j = write( tempsockfd, buffer, i );

if( j == -1 ) { perror("socket-write"); exit(1); }

n = shutdown(tempsockfd,SHUT\_RDWR);

if ( n == -1 ) { perror("shutdown"); exit(1); }

n = close(tempsockfd);

if( n == -1 ) { perror("close-tempsockfd"); exit(1); }

exit(0);

} // end of child's block

// rest is parent's block. Main closes useless socket

close(tempsockfd);

} // end of endless-loop block

} // end of main

// signal handler

void child\_handler( int signumber )

{

pid\_t p;

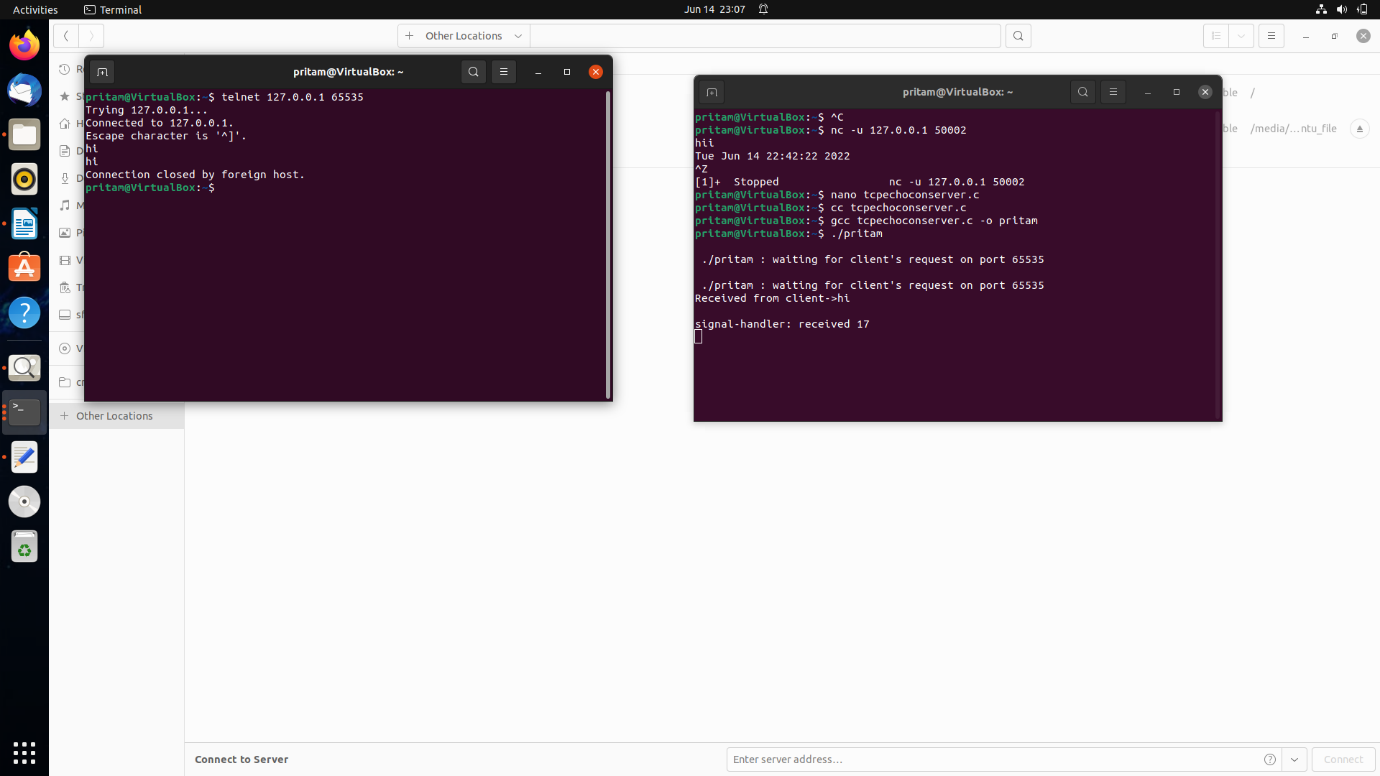
printf("signal-handler: received %u \n", signumber );

p = waitpid( -1, NULL, WNOHANG );

if( p == -1 ) perror("waitpid");

return;

}



**6.** Write a program to implement a **UDP echo concurrent server.** Get the service using telnet.

#include<stdio.h>

#include <sys/types.h> /\* See NOTES \*/

#include <sys/socket.h>

#include <unistd.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <stdlib.h>

#define IP "127.0.0.1"

#define PORT 9999

int main(){

printf("[+]Program is running FD : %d\n",getpid());

// Create socket

int server\_socket = socket(AF\_INET,SOCK\_DGRAM, IPPROTO\_UDP);

if(server\_socket < 0){perror("Socket error"); exit(1);}

//assign address and port

struct sockaddr\_in server\_address;

server\_address.sin\_family = AF\_INET;

server\_address.sin\_port = htons(PORT);

int n = inet\_aton(IP , &(server\_address.sin\_addr));

if(n < 0){perror("Invalid address"); exit(1);}

// bind the address

int bfd = bind(server\_socket,(struct sockaddr \*) &server\_address, sizeof(server\_address));

if(bfd < 0){perror("Bind error"); exit(1);}

printf("[+]IP : %s : %d \n[+]Waiting for client request ...\n",IP,PORT);

// Read write

while(1){

char buff[1024];

struct sockaddr\_in client\_address;

int len = sizeof(client\_address);

int refd = recvfrom(server\_socket,buff,sizeof(buff),0,(struct sockaddr \*)&client\_address,&len);

if(refd<0){ perror("Receive error"); exit(1);}

printf("Received from %s : %d ->\n",inet\_ntoa(client\_address.sin\_addr),ntohs(client\_address.sin\_port));

write(STDOUT\_FILENO,buff,refd);

int sfd = sendto(server\_socket,buff,refd,0,(struct sockaddr \*)&client\_address, sizeof(client\_address));

if(sfd<0){ perror("Send error"); exit(1);}

}

return 0;

}