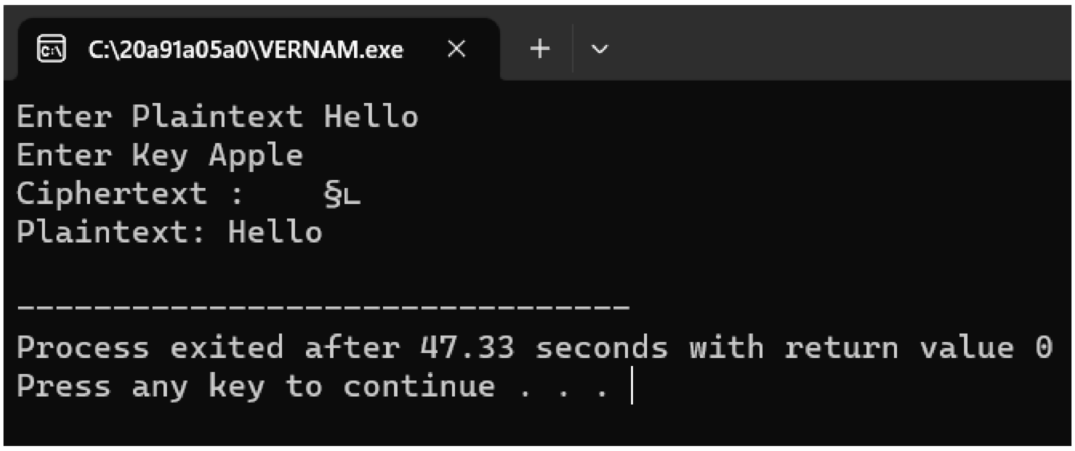
PROGRAM:

```
#include<stdio.h>

#include<string.h>

#include<stdlib.h>

void encrypt(char *plaintext, char *key,char *ciphertext)

{

int i;

for(i=0; i<strlen(plaintext);i++)

{

ciphertext[i] = plaintext[i] ^ key[i];

}

}

void decrypt(char *ciphertext , char *key , char *plaintext)

{

int i;

for(i=0; i<strlen(ciphertext);i++)

{

plaintext[i] = ciphertext[i] ^ key[i];

}

}

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

{

char plaintext[100];

char key[100];

char ciphertext[100];

printf("Enter Plaintext ");

scanf("%s",plaintext);

printf("Enter Key ");

scanf("%s",key);

encrypt(plaintext,key,ciphertext);

printf("Ciphertext : %s\n",ciphertext);
decrypt(ciphertext,key,plaintext);

printf("Plaintext: %s\n",plaintext);

return 0;

}
```

OUTPUT:



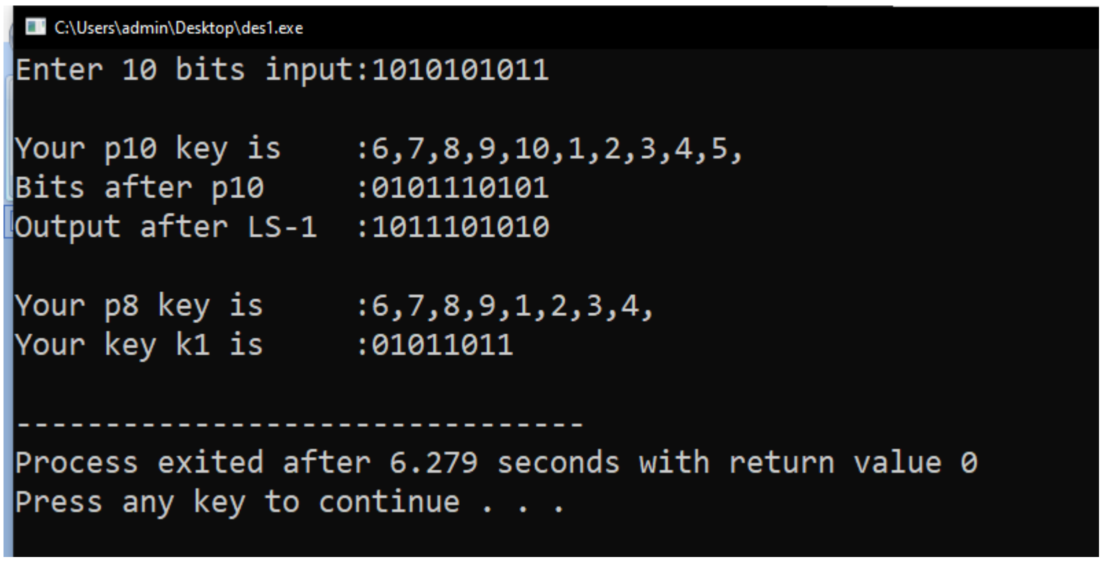
C:\20a91a05a0\VERNAM.exe

```
Enter Plaintext Hello
Enter Key Apple
Ciphertext :    §L
Plaintext: Hello

--------------------------------
Process exited after 47.33 seconds with return value 0
Press any key to continue . . .
```

PROGRAM:

```
#include<stdio.h>

int main()

{

int i, cnt=0, p8[8]={6,7,8,9,1,2,3,4};

int p10[10]={6,7,8,9,10,1,2,3,4,5};

char input[11], k1[10], k2[10], temp[11];

char LS1[5], LS2[5];

//k1, k2 are for storing interim keys

//p8 and p10 are for storing permutation key

//Read 10 bits from user...

printf("Enter 10 bits input:");

scanf("%s",input);

input[10]='\0';

//Applying p10...

for(i=0; i<10; i++)

{

cnt = p10[i];

temp[i] = input[cnt-1];

}

temp[i]='\0';

printf("\nYour p10 key is :");

for(i=0; i<10; i++)

{ printf("%d,",p10[i]);

}

printf("\nBits after p10 :");

puts(temp);

//Performing LS-1 on first half of temp

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

{

if(i==4)
temp[i]=temp[0];

else

temp[i]=temp[i+1];

}

//Performing LS-1 on second half of temp

for(i=5; i<10; i++)

{

if(i==9)

temp[i]=temp[5];

else

temp[i]=temp[i+1];

}

printf("Output after LS-1 :");

puts(temp);

printf("\nYour p8 key is :");

for(i=0; i<8; i++){

printf("%d,",p8[i]);

}

//Applying p8...

for(i=0; i<8; i++)

{

cnt = p8[i];

k1[i] = temp[cnt-1];

}

printf("\nYour key k1 is :");

puts(k1);

//This program can be extended to generate k2 as per DES algorithm.

}
```

OUTPUT:



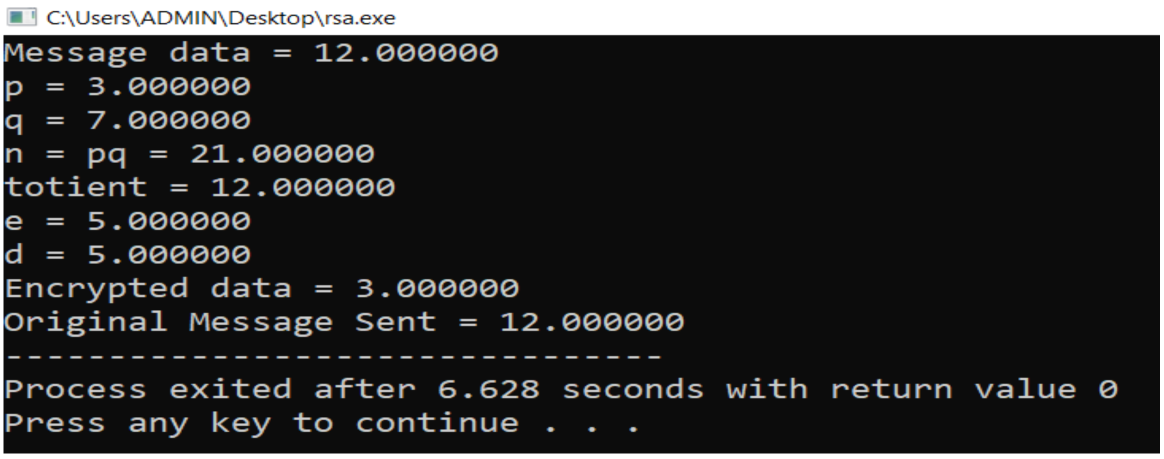
C:\Users\admin\Desktop\des1.exe

```
Enter 10 bits input:1010101011

Your p10 key is    :6,7,8,9,10,1,2,3,4,5,
Bits after p10     :0101110101
Output after LS-1  :1011101010

Your p8 key is     :6,7,8,9,1,2,3,4,
Your key k1 is     :01011011

--------------------------------
Process exited after 6.279 seconds with return value 0
Press any key to continue . . .
```

PROGRAM:

```
#include<stdio.h>

#include<math.h>

//to find gcd

int gcd(int a, int h)

{

int temp;

while(1)

{

temp = a%h;

if(temp==0)

return h;

a = h;

h = temp;

}

}

int main()

{

//2 random prime numbers

double p = 3;

double q = 7;

double n=p*q;

double count;

double totient = (p-1)*(q-1);

//public key

//e stands for encrypt

double e=2;

//for checking co-prime which satisfies e>1

while(e<totient){

count = gcd(e,totient);

if(count==1)
break;

else

e++;

}

//private key

//d stands for decrypt

double d;

//k can be any arbitrary value

double k = 2;

//choosing d such that it satisfies d*e = 1 + k * totient

d = (1 + (k*totient))/e;

double msg = 12;

double c = pow(msg,e);

double m = pow(c,d);

c=fmod(c,n);

m=fmod(m,n);

printf("Message data = %lf",msg);

printf("\np = %lf",p);

printf("\nq = %lf",q);

printf("\nn = pq = %lf",n);

printf("\ntotient = %lf",totient);

printf("\ne = %lf",e);

printf("\nd = %lf",d);

printf("\nEncrypted data = %lf",c);

printf("\nOriginal Message Sent = %lf",m);

return 0;

}
```

OUTPUT:

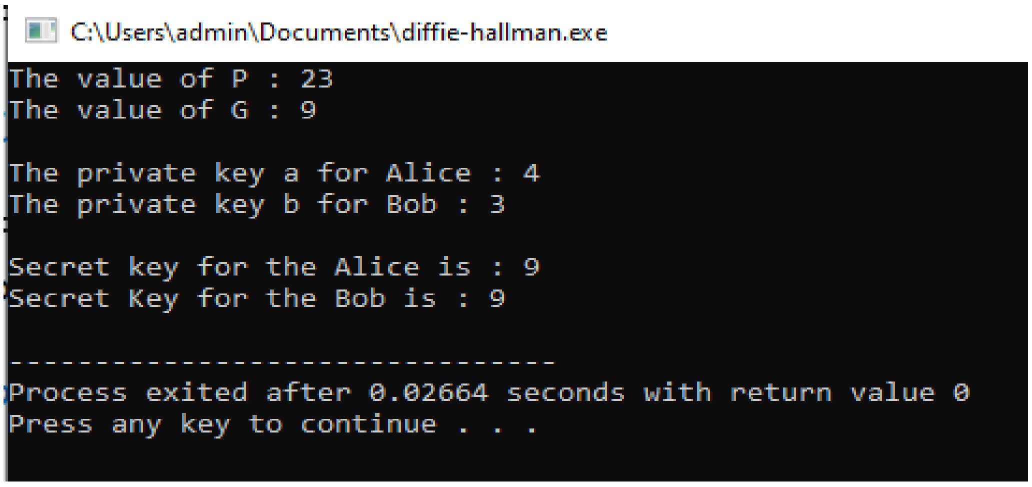

C:\Users\ADMIN\Desktop\rsa.exe

```
Message data = 12.000000
p = 3.000000
q = 7.000000
n = pq = 21.000000
totient = 12.000000
e = 5.000000
d = 5.000000
Encrypted data = 3.000000
Original Message Sent = 12.000000
--------------------------------
Process exited after 6.628 seconds with return value 0
Press any key to continue . . .
```

PROGRAM:

```
#include<stdio.h>
#include<math.h>
// Power function to return value of a ^ b mod P
long long int power(long long int a, long long int b, long long int P)
{
if (b == 1)
return a;
else
return (((long long int)pow(a, b)) % P);
}
//Driver program
int main()
{
long long int P, G, x, a, y, b, ka, kb;
// Both the persons will be agreed upon the
// public keys G and P
P = 23;
// A prime number P is taken
printf("The value of P : %lld\n", P);
G = 9;
// A primitive root for P, G is taken
printf("The value of G : %lld\n\n", G);
// Alice will choose the private key a
a = 4;
// a is the chosen private key
printf("The private key a for Alice : %lld\n", a);
x = power(G, a, P);
// gets the generated key
// Bob will choose the private key b
b = 3;
// b is the chosen private key

printf("The private key b for Bob : %lld\n\n", b);

y = power(G, b, P);

// gets the generated key

// Generating the secret key after the exchange

// of keys

ka = power(y, a, P); // Secret key for Alice

kb = power(x, b, P); // Secret key for Bob

printf("Secret key for the Alice is : %lld\n", ka);

printf("Secret Key for the Bob is : %lld\n", kb);

return 0;

}
```

OUTPUT:



C:\Users\admin\Documents\diffie-hallman.exe

```
The value of P : 23
The value of G : 9

The private key a for Alice : 4
The private key b for Bob : 3

Secret key for the Alice is : 9
Secret Key for the Bob is : 9

--------------------------------
Process exited after 0.02664 seconds with return value 0
Press any key to continue . . .
```

PROGRAM:

```
#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <math.h>

int e1, e2;

int p, d;

int C1, C2;

FILE *out1, *out2;

int gcd(int a, int b)

{

int q, r1, r2, r;

if (a > b)

{

r1 = a;

r2 = b;

}

else {

r1 = b;

r2 = a;

}

while (r2 > 0)

{

q = r1 / r2;

r = r1 - q * r2;

r1 = r2;

r2 = r;

}

return r1;

}

int FastExponention(int bit, int n, int* y, int* a)
```

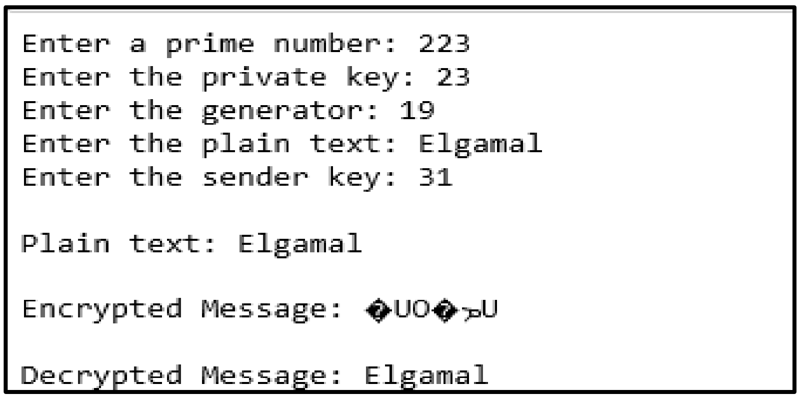
```
{
if (bit == 1) {
*y = (*y * (*a)) % n;
}
*a = (*a) * (*a) % n;
}
int FindT(int a, int m, int n)
{
int r;
int y = 1;
while (m > 0)
{
r = m % 2;
FastExponention(r, n, &y, &a);
m = m / 2;
}
return y;
}
int PrimarityTest(int a, int i)
{
int n = i - 1;
int k = 0;
int m, T;
while (n % 2 == 0)
{
k++;
n = n / 2;
}
m = n;
T = FindT(a, m, i);
if (T == 1 || T == i - 1) {
```

```
return 1;
}
int j;
for (j = 0; j < k; j++)
{
T = FindT(T, 2, i);
if (T == 1) {
return 0;
}
if (T == i - 1) {
return 1;
}
}
return 0;
}
int PrimitiveRoot(int p)
{
int flag;
int a;
for (a = 2; a < p; a++)
{
flag = 1;
int i;
for (i = 1; i < p; i++)
{
if (FindT(a, i, p) == 1 && i < p - 1) {
flag = 0;
}
else if (flag && FindT(a, i, p) == 1 && i == p - 1) {
return a;
}
}
}
}
int KeyGeneration()
{
do {
do
p = rand() + 256;
while (p % 2 == 0);
} while (!PrimarityTest(2, p));
p = 107;
e1 = 2;
do {
d = rand() % (p - 2) + 1; // 1 <= d <= p-2
} while (gcd(d, p) != 1);
d = 67;
e2 = FindT(e1, d, p);
}
int Encryption(int Plaintext)
{
out1 = fopen("cipher1.txt", "a+");
out2 = fopen("cipher2.txt", "a+");
int r;
do {
r = rand() % (p - 1) + 1; // 1 < r < p
}
while (gcd(r, p) != 1);
C1 = FindT(e1, r, p);
C2 = FindT(e2, r, p) * Plaintext % p;
fprintf(out1, "%d ", C1);
fprintf(out2, "%d ", C2);
```

```
fclose(out1);

fclose(out2);

}

int Decryption(int C1, int C2)

{

FILE* out = fopen("result.txt", "a+");

int decipher = C2 * FindT(C1, p - 1 - d, p) % p;

fprintf(out, "%c", decipher);

fclose(out);

}

int main()

{

FILE *out, *inp;

// destroy contents of these files (from previous runs, if any)

out = fopen("result.txt", "w+");

fclose(out);

out = fopen("cipher1.txt", "w+");

fclose(out);

out = fopen("cipher2.txt", "w+");

fclose(out);

KeyGeneration();

inp = fopen("plain.txt", "r+");

if (inp == NULL)

{

printf("Error opening Source File.\n");

exit(1);

}

while (1)

{

char ch = getc(inp);

if (ch == EOF) {
```

```
break; // M < p

}

Encryption(toascii(ch));

}

fclose(inp);

FILE *inp1, *inp2;

inp1 = fopen("cipher1.txt", "r");

inp2 = fopen("cipher2.txt", "r");

int C1, C2;

while (1)

{

int ret = fscanf(inp1, "%d", &C1);

fscanf(inp2, "%d", &C2);

if (ret == -1) {

break;

}

Decryption(C1, C2);

}

fclose(inp1);

fclose(inp2);

return 0;

}
```

OUTPUT:



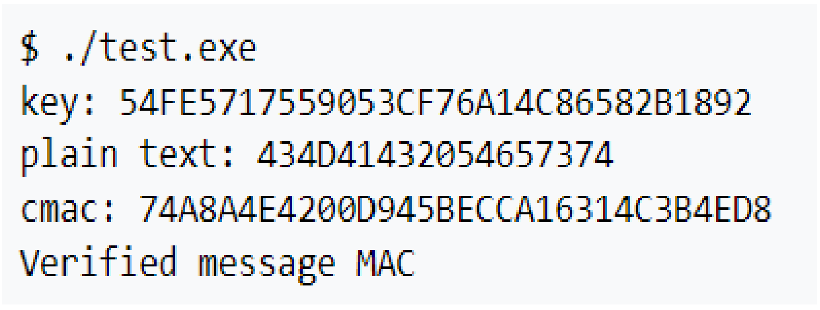
```
Enter a prime number: 223
Enter the private key: 23
Enter the generator: 19
Enter the plain text: Elgamal
Enter the sender key: 31

Plain text: Elgamal

Encrypted Message: �UO�ڼU

Decrypted Message: Elgamal
```

PROGRAM:

```
#include "cryptlib.h"
#include "secblock.h"
#include "osrng.h"
#include "files.h"
#include "cmac.h"
#include "aes.h"
#include "hex.h"
using namespace CryptoPP;
#include <iostream>
#include <string>
using namespace std;
int main(int argc, char* argv[])
{
AutoSeededRandomPool prng;
SecByteBlock key(AES::DEFAULT_KEYLENGTH);
prng.GenerateBlock(key, key.size());
string mac, plain = "CMAC Test";
HexEncoder encoder(new FileSink(cout));
// Pretty print key
cout << "key: ";
encoder.Put(key, key.size());
encoder.MessageEnd();
cout << endl;
cout << "plain text: ";
encoder.Put((const byte*)plain.data(), plain.size());
encoder.MessageEnd();
cout << endl;
try
{
CMAC<AES> cmac(key.data(), key.size());
cmac.Update((const byte*)plain.data(), plain.size());
mac.resize(cmac.DigestSize());
cmac.Final((byte*)&mac[0]);
}
catch(const CryptoPP::Exception& e)
{
cerr << e.what() << endl;
exit(1);}
// Pretty print
cout << "cmac: ";
encoder.Put((const byte*)mac.data(), mac.size());
encoder.MessageEnd();
cout << endl;
// Verify
try
{
CMAC<AES> cmac(key.data(), key.size());
cmac.Update((const byte*)plain.data(), plain.size());
// Call Verify() instead of Final()
bool verified = cmac.Verify((byte*)&mac[0]);
if (!verified)
throw Exception(Exception::DATA_INTEGRITY_CHECK_FAILED, "CMAC: message MAC
not valid");
cout << "Verified message MAC" << endl;
}
catch(const CryptoPP::Exception& e)
{
cerr << e.what() << endl;
exit(1);
}
return 0;
```

}

OUTPUT:



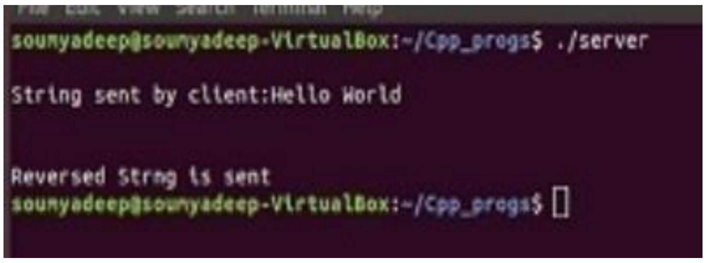
```
$ ./test.exe
key: 54FE5717559053CF76A14C86582B1892
plain text: 434D41432054657374
cmac: 74A8A4E4200D945BECCA16314C3B4ED8
Verified message MAC
```

PROGRAM:

TCP Server:

```
#include<string.h>
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/socket.h>
#include<sys/types.h>
#define MAXLINE 20
#define SERV_PORT 5777
main(int argc,char *argv) {
int i,j;
ssize_t n;
char line[MAXLINE];
char revline[MAXLINE];
int listenfd,connfd,clilen;
struct sockaddr_in servaddr,cliaddr;
listenfd=socket(AF_INET,SOCK_STREAM,0);
bzero(&servaddr,sizeof(servaddr));
servaddr.sin_family=AF_INET; servaddr.sin_port=htons(SERV_PORT);
bind(listenfd,(struct sockaddr*)&servaddr,sizeof(servaddr));
listen(listenfd,1);
for( ; ; ) {
clilen=sizeof(cliaddr);
connfd=accept(listenfd,(struct sockaddr*)&cliaddr,&clilen);
printf("connect to client");
while(1) {
if((n=read(connfd,line,MAXLINE))==0)
break;
line[n-1]='\0';
j=0;
for(i=n-2;i>=0;i--)

revline[j++]=line[i];

revline[j]='\0';

write(connfd,revline,n);

}

}

}
```

OUTPUT:

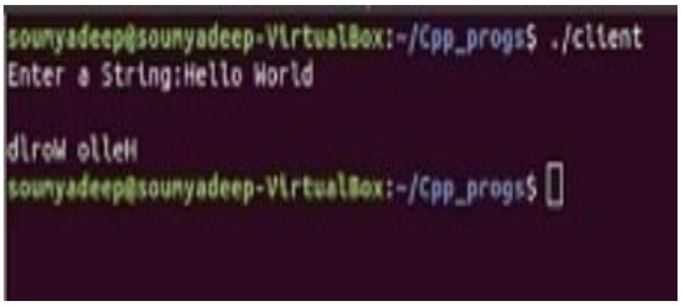


```
sounyadeep@sounyadeep-VirtualBox:~/Cpp_progs$ ./server

String sent by client:Hello World


Reversed Strng is sent
sounyadeep@sounyadeep-VirtualBox:~/Cpp_progs$
```

<u>TCP Client:</u>

```
#include<string.h>

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/socket.h>

#include<netinet/in.h>

#include<sys/types.h>

#define MAXLINE 20

#define SERV_PORT 5777

main(int argc,char *argv)

{

char sendline[MAXLINE],revline[MAXLINE];

int sockfd;

struct sockaddr_in servaddr;

sockfd=socket(AF_INET,SOCK_STREAM,0);

bzero(&servaddr,sizeof(servaddr));

servaddr.sin_family=AF_INET;
servaddr.sin_port=ntohs(SERV_PORT);
connect(sockfd,(struct sockaddr*)&servaddr,sizeof(servaddr));
printf("\n enter the data to be send");
while(fgets(sendline,MAXLINE,stdin)!=NULL)
{
}
exit(0);
}
```

OUTPUT:



```
soumyadeep@soumyadeep-VirtualBox:~/Cpp_progs$ ./client
Enter a String:Hello World

dlroW olleH
soumyadeep@soumyadeep-VirtualBox:~/Cpp_progs$ 
```

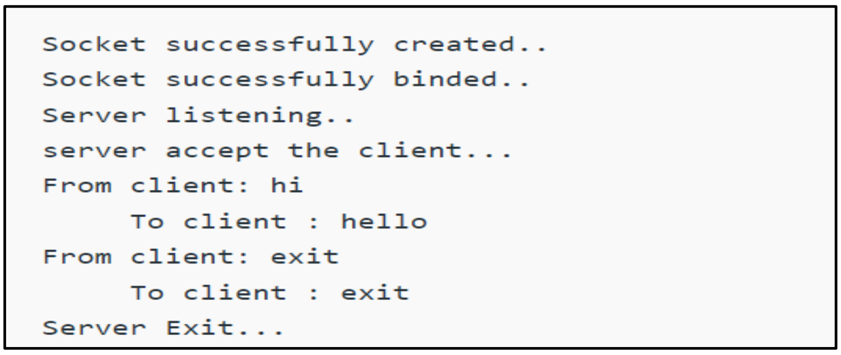
PROGRAM:

TCP 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>

#include <unistd.h> // read(), write(), close()

#define MAX 80

#define PORT 8080

#define SA struct sockaddr

void func(int connfd) // Function designed for chat between client and server.

{

char buff[MAX];

int n;

// infinite loop for chat

for (;;) {

bzero(buff, MAX);

read(connfd, buff, sizeof(buff)); // read the message from client and copy it in buffer

printf("From client: %s\t To client : ", buff);

bzero(buff, MAX);

n = 0;

while ((buff[n++] = getchar()) != '\n') // copy server message in the buffer

write(connfd, buff, sizeof(buff)); // and send that buffer to client

// if msg contains "Exit" then server exit and chat ended.

if (strncmp("exit", buff, 4) == 0) {

printf("Server Exit...\n");

break;

}
}
}
// Driver function
int main()
{
int sockfd, connfd, len;
struct sockaddr_in servaddr, cli;
// socket create and verification
sockfd = socket(AF_INET, SOCK_STREAM, 0);
if (sockfd == -1) {
printf("socket creation failed...\n");
exit(0);
}
printf("Socket successfully created..\n");
bzero(&servaddr, sizeof(servaddr));
// assign IP, PORT
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(PORT);
// Binding newly created socket to given IP and verification
if ((bind(sockfd, (SA*)&servaddr, sizeof(servaddr))) != 0) {
printf("socket bind failed...\n");
exit(0);
}
else
printf("Socket successfully binded..\n");
// Now server is ready to listen and verification
if ((listen(sockfd, 5)) != 0) {
printf("Listen failed...\n");
exit(0);
}
```

```
else

printf("Server listening..\n");

len = sizeof(cli);

// Accept the data packet from client and verification

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); // Function for chatting between client and server

close(sockfd);

}
```

OUTPUT:



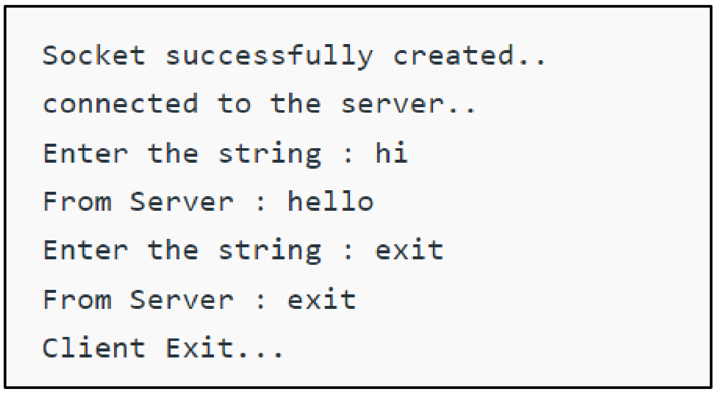
```
Socket successfully created..
Socket successfully binded..
Server listening..
server accept the client...
From client: hi
     To client : hello
From client: exit
     To client : exit
Server Exit...
```

TCP Client:

```
#include <arpa/inet.h> // inet_addr()

#include <netdb.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <strings.h> // bzero()

#include <sys/socket.h>

#include <unistd.h> // read(), write(), close()

#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;

// socket create and verification

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));

// assign IP, PORT

servaddr.sin_family = AF_INET;

servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");

servaddr.sin_port = htons(PORT);

// connect the client socket to server socket

if (connect(sockfd, (SA*)&servaddr, sizeof(servaddr))

!= 0) {

printf("connection with the server failed...\n");

exit(0);

}

else

printf("connected to the server..\n");

// function for chat

func(sockfd);

// close the socket

close(sockfd);

}
```

OUTPUT:



```
Socket successfully created..
connected to the server..
Enter the string : hi
From Server : hello
Enter the string : exit
From Server : exit
Client Exit...
```

PROGRAM:

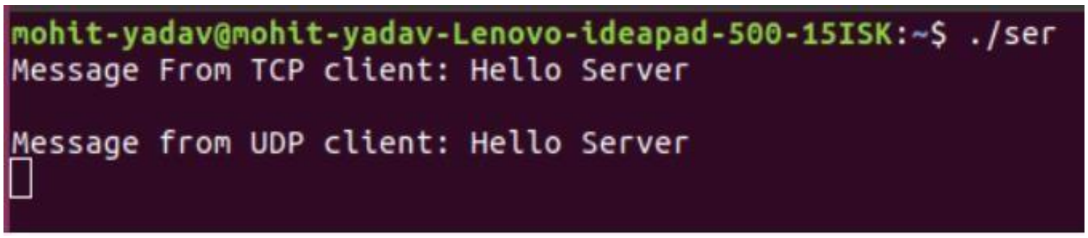
TCP Server:

```
#include <arpa/inet.h>

#include <errno.h>

#include <netinet/in.h>

#include <signal.h>

#include <stdio.h>

#include <stdlib.h>

#include <strings.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <unistd.h>

#define PORT 5000

#define MAXLINE 1024

int max(int x, int y)

{

if (x > y)

return x;

else

}

return y;

int main()

{

int listenfd, connfd, udpfd, nready, maxfdp1;

char buffer[MAXLINE];

pid_t childpid;

fd_set rset;

ssize_t n;

socklen_t len;

const int on = 1;

struct sockaddr_in cliaddr, servaddr;
```

```
char* message = "Hello Client";
void sig_chld(int);
/* create listening TCP socket */
listenfd = socket(AF_INET, SOCK_STREAM, 0);
bzero(&servaddr, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(PORT);
// binding server addr structure to listenfd
bind(listenfd, (struct sockaddr*)&servaddr, sizeof(servaddr));
listen(listenfd, 10);
/* create UDP socket */
udpfd = socket(AF_INET, SOCK_DGRAM, 0);
// binding server addr structure to udp sockfd
bind(udpfd, (struct sockaddr*)&servaddr, sizeof(servaddr));
// clear the descriptor set
FD_ZERO(&rset);
// get maxfd
maxfdp1 = max(listenfd, udpfd) + 1;
for (;;) {
// set listenfd and udpfd in readset
FD_SET(listenfd, &rset);
FD_SET(udpfd, &rset);
// select the ready descriptor
nready = select(maxfdp1, &rset, NULL, NULL, NULL);
// if tcp socket is readable then handle
// it by accepting the connection
if (FD_ISSET(listenfd, &rset)) {
len = sizeof(cliaddr);
connfd = accept(listenfd, (struct sockaddr*)&cliaddr, &len);
if ((childpid = fork()) == 0) {
```

```
close(listenfd);
bzero(buffer, sizeof(buffer));
printf("Message From TCP client: ");
read(connfd, buffer, sizeof(buffer));
puts(buffer);
write(connfd, (const char*)message, sizeof(buffer));
close(connfd);
exit(0);
}
close(connfd);
}
// if udp socket is readable receive the message.
if (FD_ISSET(udpfd, &rset)) {
len = sizeof(cliaddr);
bzero(buffer, sizeof(buffer));
printf("\nMessage from UDP client: ");
n = recvfrom(udpfd, buffer, sizeof(buffer), 0, (struct sockaddr*)&cliaddr, &len);
puts(buffer);
sendto(udpfd, (const char*)message, sizeof(buffer), 0,
(struct sockaddr*)&cliaddr, sizeof(cliaddr));
}
}
}
```

OUTPUT:

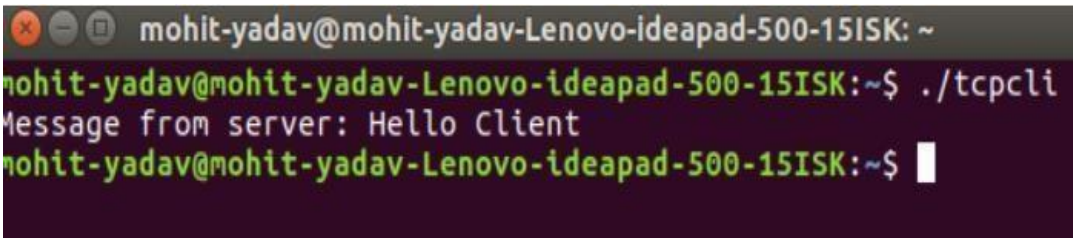


```
mohit-yadav@mohit-yadav-Lenovo-ideapad-500-15ISK:~$ ./ser
Message From TCP client: Hello Server

Message from UDP client: Hello Server
```

TCP Client:

```
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#define PORT 5000
#define MAXLINE 1024
int main()
{
int sockfd;
char buffer[MAXLINE];
char* message = "Hello Server";
struct sockaddr_in servaddr;
int n, len;
// Creating socket file descriptor
if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
printf("socket creation failed");
exit(0);
}
memset(&servaddr, 0, sizeof(servaddr));
// Filling server information
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(PORT);
servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
if (connect(sockfd, (struct sockaddr*)&servaddr, sizeof(servaddr)) < 0) {
printf("\n Error : Connect Failed \n");
}
memset(buffer, 0, sizeof(buffer));
strcpy(buffer, "Hello Server");
write(sockfd, buffer, sizeof(buffer));

printf("Message from server: ");

read(sockfd, buffer, sizeof(buffer));

puts(buffer);

close(sockfd);

}
```

OUTPUT:



mohit-yadav@mohit-yadav-Lenovo-ideapad-500-15ISK: ~

```
nohit-yadav@mohit-yadav-Lenovo-ideapad-500-15ISK:~$ ./tcpcli
Message from server: Hello Client
nohit-yadav@mohit-yadav-Lenovo-ideapad-500-15ISK:~$
```

PROGRAM:

```
#include <stdio.h>

#include <stdlib.h>

#include <sys/ioctl.h>

#include <sys/poll.h>

#include <sys/socket.h>

#include <sys/time.h>

#include <netinet/in.h>

#include <errno.h>

#define SERVER_PORT 12345

#define TRUE 1

#define FALSE 0

main (int argc, char *argv[])

{

int len, rc, on = 1;

int listen_sd = -1, new_sd = -1;

int desc_ready, end_server = FALSE, compress_array = FALSE;

int close_conn;

char buffer[80];

struct sockaddr_in6 addr;

int timeout;

struct pollfd fds[200];

int nfds = 1, current_size = 0, i, j;

listen_sd = socket(AF_INET6, SOCK_STREAM, 0);

if (listen_sd < 0)

{

perror("socket() failed");

exit(-1);

}

rc = setsockopt(listen_sd, SOL_SOCKET, SO_REUSEADDR,

(char *)&on, sizeof(on));
```

```
if (rc < 0)

{

perror("setsockopt() failed");

close(listen_sd);

exit(-1);

}

rc = ioctl(listen_sd, FIONBIO, (char *)&on);

if (rc < 0)

{

perror("ioctl() failed");

close(listen_sd);

exit(-1);

}

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

addr.sin6_family = AF_INET6;

memcpy(&addr.sin6_addr, &in6addr_any, sizeof(in6addr_any));

addr.sin6_port = htons(SERVER_PORT);

rc = bind(listen_sd,

(struct sockaddr *)&addr, sizeof(addr));

if (rc < 0)

{

perror("bind() failed");

close(listen_sd);

exit(-1);

}

rc = listen(listen_sd, 32);

if (rc < 0)

{

perror("listen() failed");

close(listen_sd);

exit(-1);
```

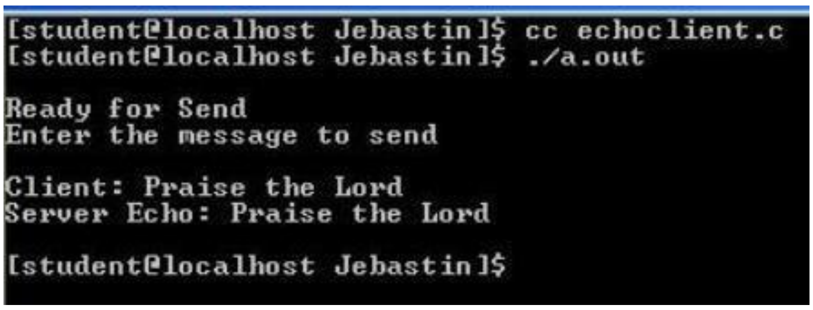
```
}
memset(fds, 0 , sizeof(fds));
fds[0].fd = listen_sd;
fds[0].events = POLLIN;
timeout = (3 * 60 * 1000);
do
{
printf("Waiting on poll()...\n");
rc = poll(fds, nfds, timeout);
if (rc < 0)
{
perror(" poll() failed");
break;
}
if (rc == 0)
{
printf(" poll() timed out. End program.\n");
break;
}
current_size = nfds;
for (i = 0; i < current_size; i++)
{
if(fds[i].revents == 0)
continue;
if(fds[i].revents != POLLIN)
{
printf(" Error! revents = %d\n", fds[i].revents);
end_server = TRUE;
break;
}
if (fds[i].fd == listen_sd)
```

```
{
printf(" Listening socket is readable\n");
do
{
new_sd = accept(listen_sd, NULL, NULL);
if (new_sd < 0)
{
if (errno != EWOULDBLOCK)
{
perror(" accept() failed");
end_server = TRUE;
}
break;
}
printf(" New incoming connection - %d\n", new_sd);
fds[nfds].fd = new_sd;
fds[nfds].events = POLLIN;
nfds++;
} while (new_sd != -1);
}
else
{
printf(" Descriptor %d is readable\n", fds[i].fd);
close_conn = FALSE;
do
{
rc = recv(fds[i].fd, buffer, sizeof(buffer), 0);
if (rc < 0)
{
if (errno != EWOULDBLOCK)
{
```

```
perror(" recv() failed");

close_conn = TRUE;

}

break;

}

if (rc == 0)

{

printf(" Connection closed\n");

close_conn = TRUE;

break;

}

len = rc;

printf(" %d bytes received\n", len);

rc = send(fds[i].fd, buffer, len, 0);

if (rc < 0)

{

perror(" send() failed");

close_conn = TRUE;

break;

}

} while(TRUE);

if (close_conn)

{

close(fds[i].fd);

fds[i].fd = -1;

compress_array = TRUE;

}

} /* End of existing connection is readable */

} /* End of loop through pollable descriptors */

if (compress_array)

{
```

```
compress_array = FALSE;
for (i = 0; i < nfds; i++)
{
if (fds[i].fd == -1)
{
for(j = i; j < nfds; j++)
{
fds[j].fd = fds[j+1].fd;
}
i--;
nfds--;
}
}
}
} while (end_server == FALSE); /* End of serving running. */
for (i = 0; i < nfds; i++)
{
if(fds[i].fd >= 0)
close(fds[i].fd);
}
}
```

OUTPUT:



```
[student@localhost Jebastin]$ cc echoclient.c
[student@localhost Jebastin]$ ./a.out

Ready for Send
Enter the message to send

Client: Praise the Lord
Server Echo: Praise the Lord

[student@localhost Jebastin]$
```

PROGRAM:

UDP Client:

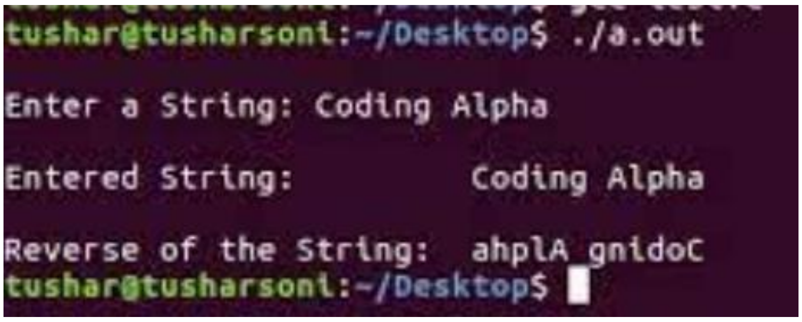
```
#include <sys/socket.h>
#include <netdb.h>
#include <string.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#define S_PORT 43454
#define C_PORT 43455
#define ERROR -1
#define IP_STR "127.0.0.1"
int main(int argc, char const *argv[]) {
int sfd, len;
char str_buf[2048];
struct sockaddr_in servaddr, clientaddr;
socklen_t addrlen;
sfd = socket(AF_INET, SOCK_DGRAM,IPPROTO_UDP);
if (sfd == ERROR) {
perror("Could not open a socket");
return 1;
}
memset((char *) &servaddr, 0, sizeof(servaddr));
servaddr.sin_family=AF_INET;
servaddr.sin_addr.s_addr=inet_addr(IP_STR);
servaddr.sin_port=htons(S_PORT);
memset((char *) &clientaddr, 0, sizeof(clientaddr));
clientaddr.sin_family=AF_INET;
clientaddr.sin_addr.s_addr=inet_addr(IP_STR);
clientaddr.sin_port=htons(C_PORT);
if((bind(sfd,(struct sockaddr *)&clientaddr,sizeof(clientaddr)))!=0) {
perror("Could not bind socket");
return 2;
}
printf("Client is running on %s:%d\n", IP_STR, C_PORT);
printf("Enter a string: ");
scanf("%[^\n]%*c",str_buf);
len = strlen(str_buf);
sendto(sfd, &len, sizeof(len), 0, (struct sockaddr *)&servaddr, sizeof(servaddr));
sendto(sfd, str_buf, len, 0, (struct sockaddr *)&servaddr, sizeof(servaddr));
addrlen = sizeof(clientaddr);
recvfrom(sfd, &len, sizeof(len), 0, (struct sockaddr *)&clientaddr, &addrlen);
recvfrom(sfd, str_buf, len, 0, (struct sockaddr *)&clientaddr, &addrlen);
printf("Server Replied: %s\n", str_buf);
return 0;
}
```

UDP Server:

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#define S_PORT 43454
#define C_PORT 43455
#define ERROR -1
#define IP_STR "127.0.0.1"
```

```
void strrev(char *str, int len) {
int i, j;
char temp;
for (i = 0, j = len -1; i < j; ++i, --j) {
temp = str[i];
str[i] = str[j];
str[j] = temp;
}
}
int main(int argc, char const *argv[]) {
int sfd, len;
char *str_buf;
struct sockaddr_in servaddr, clientaddr;
sfd = socket(AF_INET, SOCK_DGRAM,IPPROTO_UDP);
if (sfd == ERROR) {
perror("Could not open a socket");
return 1;
}
memset((char *) &servaddr, 0, sizeof(servaddr));
servaddr.sin_family=AF_INET;
servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
servaddr.sin_port=htons(S_PORT);
memset((char *) &clientaddr, 0, sizeof(clientaddr));
clientaddr.sin_family=AF_INET;
clientaddr.sin_addr.s_addr=inet_addr(IP_STR);
clientaddr.sin_port=htons(C_PORT);
if((bind(sfd,(struct sockaddr *)&servaddr,sizeof(servaddr)))!=0) {
perror("Could not bind socket");
return 2;
}
printf("Server is running on %s:%d\n", IP_STR, S_PORT);
while(1) {
recvfrom(sfd, &len, sizeof(len), 0, (struct sockaddr *)&clientaddr, (socklen_t *)&clientaddr);
str_buf = (char *) malloc(len*sizeof(char));
recvfrom(sfd, str_buf, len, 0, (struct sockaddr *)&clientaddr, (socklen_t *)&clientaddr);
printf("Client at %s:%d said: %s\t", inet_ntoa(clientaddr.sin_addr), ntohs(clientaddr.sin_port),
str_buf);
strrev(str_buf,len);
sendto(sfd, &len, sizeof(len), 0, (struct sockaddr *)&clientaddr, sizeof(clientaddr));
sendto(sfd, str_buf, len, 0, (struct sockaddr *)&clientaddr, sizeof(clientaddr));
printf("The reverse is: %s\n", str_buf);
free(str_buf);
}
return 0;
}
```

OUTPUT:



```
tushar@tusharsoni:~/Desktop$ ./a.out

Enter a String: Coding Alpha

Entered String:         Coding Alpha

Reverse of the String:  ahplA gnidoC
tushar@tusharsoni:~/Desktop$ 
```

PROGRAM:

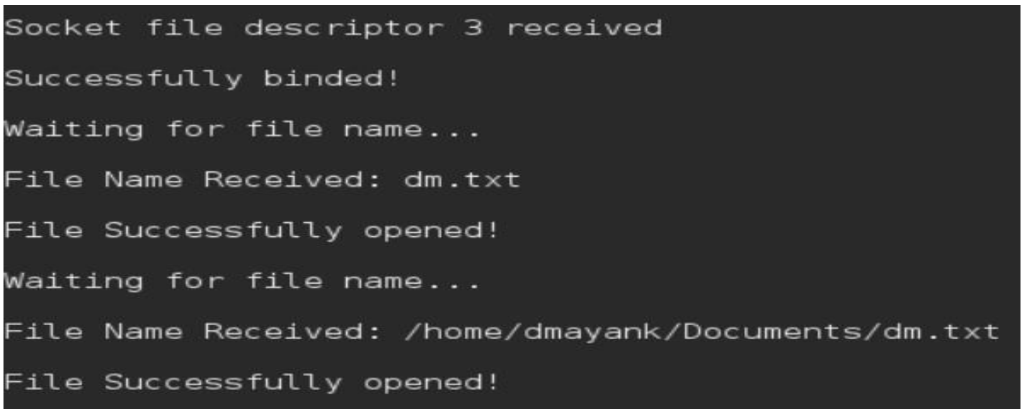
Server Implementation:

```
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
#define IP_PROTOCOL 0
#define PORT_NO 15050
#define NET_BUF_SIZE 32
#define cipherKey 'S'
#define sendrecvflag 0
#define nofile "File Not Found!"
// function to clear buffer
void clearBuf(char* b)
{
int i;
for (i = 0; i < NET_BUF_SIZE; i++)
b[i] = '\0';
}
// function to encrypt
char Cipher(char ch)
{
return ch ^ cipherKey;
}
// function sending file
int sendFile(FILE* fp, char* buf, int s)
{
```

```
int i, len;
if (fp == NULL) {
strcpy(buf, nofile);
len = strlen(nofile);
buf[len] = EOF;
for (i = 0; i <= len; i++)
buf[i] = Cipher(buf[i]);
return 1;
}
char ch, ch2;
for (i = 0; i < s; i++) {
ch = fgetc(fp);
ch2 = Cipher(ch);
buf[i] = ch2;
if (ch == EOF)
return 1;
}
return 0;
}
// driver code
int main()
{
int sockfd, nBytes;
struct sockaddr_in addr_con;
int addrlen = sizeof(addr_con);
addr_con.sin_family = AF_INET;
addr_con.sin_port = htons(PORT_NO);
addr_con.sin_addr.s_addr = INADDR_ANY;
char net_buf[NET_BUF_SIZE];
FILE* fp;
// socket()
```

```
sockfd = socket(AF_INET, SOCK_DGRAM, IP_PROTOCOL);
if (sockfd < 0)
printf("\nfile descriptor not received!!\n");
else
printf("\nfile descriptor %d received\n", sockfd);
// bind()
if (bind(sockfd, (struct sockaddr*)&addr_con, sizeof(addr_con)) == 0)
printf("\nSuccessfully binded!\n");
else
printf("\nBinding Failed!\n");
while (1) {
printf("\nWaiting for file name...\n");
// receive file name
clearBuf(net_buf);
nBytes = recvfrom(sockfd, net_buf,
NET_BUF_SIZE, sendrecvflag,
(struct sockaddr*)&addr_con, &addrlen);
fp = fopen(net_buf, "r");
printf("\nFile Name Received: %s\n", net_buf);
if (fp == NULL)
printf("\nFile open failed!\n");
else
printf("\nFile Successfully opened!\n");
while (1) {
// process
if (sendFile(fp, net_buf, NET_BUF_SIZE)) {
sendto(sockfd, net_buf, NET_BUF_SIZE,
sendrecvflag,
(struct sockaddr*)&addr_con, addrlen);
break;
}
// send
sendto(sockfd, net_buf, NET_BUF_SIZE,
sendrecvflag,
(struct sockaddr*)&addr_con, addrlen);
clearBuf(net_buf);
}
if (fp != NULL)
fclose(fp);
}
return 0;
}
```

OUTPUT:



```
Socket file descriptor 3 received
Successfully binded!
Waiting for file name...
File Name Received: dm.txt
File Successfully opened!
Waiting for file name...
File Name Received: /home/dmayank/Documents/dm.txt
File Successfully opened!
```

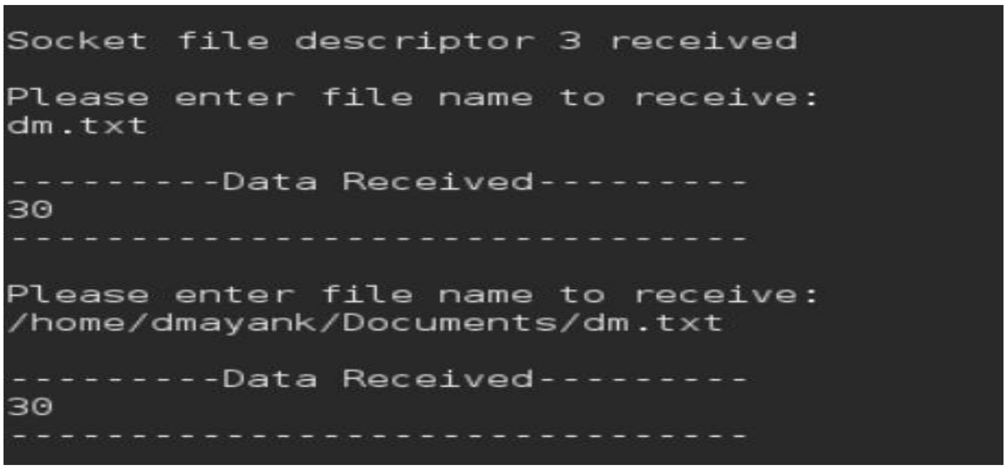
Client Implementation:

```
// client code for UDP socket programming
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
#define IP_PROTOCOL 0
```

```
#define IP_ADDRESS "127.0.0.1" // localhost

#define PORT_NO 15050

#define NET_BUF_SIZE 32

#define cipherKey 'S'

#define sendrecvflag 0

// function to clear buffer

void clearBuf(char* b)

{

int i;

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

b[i] = '\0';

}

char Cipher(char ch)

{

return ch ^ cipherKey;

}

int recvFile(char* buf, int s)

{

int i;

char ch;

for (i = 0; i < s; i++) {

ch = buf[i];

ch = Cipher(ch);

if (ch == EOF)

return 1;

else

}

printf("%c", ch);

return 0;

}

int main()
```

```
{
int sockfd, nBytes;
struct sockaddr_in addr_con;
int addrlen = sizeof(addr_con);
addr_con.sin_family = AF_INET;
addr_con.sin_port = htons(PORT_NO);
addr_con.sin_addr.s_addr = inet_addr(IP_ADDRESS);
char net_buf[NET_BUF_SIZE];
FILE* fp;
sockfd = socket(AF_INET, SOCK_DGRAM,IP_PROTOCOL);
if (sockfd < 0)
printf("\nfile descriptor not received!!\n");
else
printf("\nfile descriptor %d received\n", sockfd);
while (1) {
printf("\nPlease enter file name to receive:\n");
scanf("%s", net_buf);
sendto(sockfd, net_buf, NET_BUF_SIZE, sendrecvflag, (struct sockaddr*)&addr_con, addrlen);
printf("\n---------Data Received -------- \n");
while (1) {
clearBuf(net_buf);
nBytes = recvfrom(sockfd, net_buf, NET_BUF_SIZE,sendrecvflag, (struct sockaddr*)
&addr_con,&addrlen);
if (recvFile(net_buf, NET_BUF_SIZE)) {
break;
}
}
printf("\n \n");
}
return 0;
}
```

OUTPUT:



```
Socket file descriptor 3 received

Please enter file name to receive:
dm.txt

---------Data Received---------
30
-------------------------------

Please enter file name to receive:
/home/dmayank/Documents/dm.txt

---------Data Received---------
30
-------------------------------
```

PROGRAM:

```
#include<stdio.h>

#include<unistd.h>

int main() {

int pipefds[2];

int returnstatus;

int pid;

char writemessages[2][20]={"Hi", "Hello"};

char readmessage[20];

returnstatus = pipe(pipefds);

if (returnstatus == -1) {

printf("Unable to create pipe\n");

return 1;

}

pid = fork();

if (pid == 0) {

read(pipefds[0], readmessage, sizeof(readmessage));

printf("Child Process - Reading from pipe – Message 1 is %s\n", readmessage);

read(pipefds[0], readmessage, sizeof(readmessage));

printf("Child Process - Reading from pipe – Message 2 is %s\n", readmessage);

} else { //Parent process

printf("Parent Process - Writing to pipe - Message 1 is %s\n", writemessages[0]);

write(pipefds[1], writemessages[0], sizeof(writemessages[0]));

printf("Parent Process - Writing to pipe - Message 2 is %s\n", writemessages[1]);

write(pipefds[1], writemessages[1], sizeof(writemessages[1]));

}

return 0;

}
```

OUTPUT:

```
Parent Process - Writing to pipe - Message 1 is Hi
Parent Process - Writing to pipe - Message 2 is Hello
Child Process - Reading from pipe – Message 1 is Hi
Child Process - Reading from pipe – Message 2 is Hello
```

II)FIFO

PROGRAM:

```
#include <stdio.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <fcntl.h>

#include <unistd.h>

#include <string.h>

#define FIFO_FILE "MYFIFO"

int main() {

int fd;

int end_process;

int stringlen;

char readbuf[80];

char end_str[5];

printf("FIFO_CLIENT: Send messages, infinitely, to end enter \"end\"\n");

fd = open(FIFO_FILE, O_CREAT|O_WRONLY);

strcpy(end_str, "end");

while (1) {

printf("Enter string: ");

fgets(readbuf, sizeof(readbuf), stdin);

stringlen = strlen(readbuf);

readbuf[stringlen - 1] = '\0';

end_process = strcmp(readbuf, end_str);

//printf("end_process is %d\n", end_process);

if (end_process != 0) {

write(fd, readbuf, strlen(readbuf));

printf("Sent string: \"%s\" and string length is %d\n", readbuf, (int)strlen(readbuf));

} else {

write(fd, readbuf, strlen(readbuf));

printf("Sent string: \"%s\" and string length is %d\n", readbuf, (int)strlen(readbuf));
```

close(fd);

break;

}

}

return 0;

}

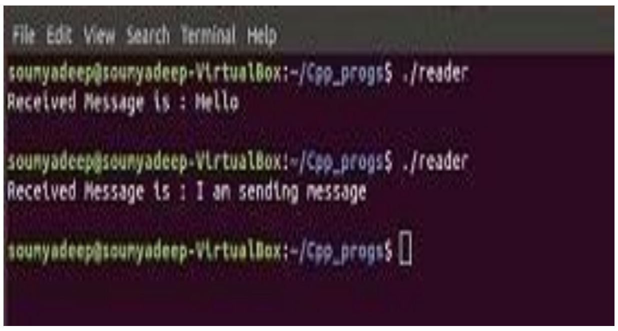
OUTPUT:

```
FIFO_CLIENT: Send messages, infinitely, to end enter "end"
Enter string: this is string 1
Sent string: "this is string 1" and string length is 16
Enter string: fifo test
Sent string: "fifo test" and string length is 9
Enter string: fifo client and server
Sent string: "fifo client and server" and string length is 22
Enter string: end
Sent string: "end" and string length is 3
```

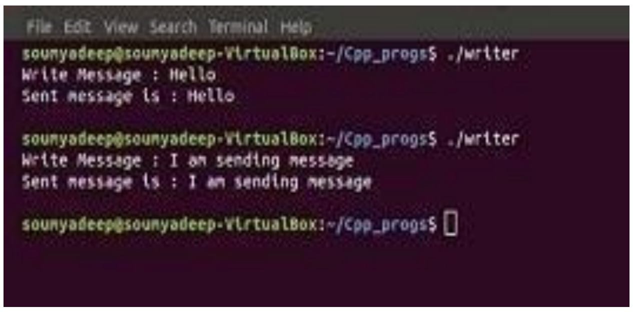
PROGRAM:

```
#include <stdio.h>

#include <sys/ipc.h>

#include <sys/msg.h>

// structure for message queue

struct msg_buffer {

long msg_type;

char msg[100];

} message;

main() {

key_t my_key;

int msg_id;

my_key = ftok("progfile", 65); //create unique key

msg_id = msgget(my_key, 0666 | IPC_CREAT); //create message queue and return id

message.msg_type = 1;

printf("Write Message : ");

fgets(message.msg, 100, stdin);

msgsnd(msg_id, &message, sizeof(message), 0); //send message

printf("Sent message is : %s \n", message.msg);

}
```

OUTPUT:



```
sounyadeep@sounyadeep-VirtualBox:~/Cpp_progs$ ./reader
Received Message is : Hello

sounyadeep@sounyadeep-VirtualBox:~/Cpp_progs$ ./reader
Received Message is : I am sending message

sounyadeep@sounyadeep-VirtualBox:~/Cpp_progs$
```



```
sounyadeep@sounyadeep-VirtualBox:~/Cpp_progs$ ./writer
Write Message : Hello
Sent message is : Hello

sounyadeep@sounyadeep-VirtualBox:~/Cpp_progs$ ./writer
Write Message : I am sending message
Sent message is : I am sending message

sounyadeep@sounyadeep-VirtualBox:~/Cpp_progs$
```

PROGRAM:

```
#include "rpctime.h"
#include <stdio.h>
#include <stdlib.h>
#include <rpc/pmap_clnt.h>
#include <string.h>
#include <memory.h>
#include <sys/socket.h>
#include <netinet/in.h>
#ifndef SIG_PF
#define SIG_PF void(*)(int)
#endif
static void
rpctime_1(struct svc_req *rqstp, register SVCXPRT *transp)
{
union {int fill;
} argument;
char *result;
xdrproc_t _xdr_argument, _xdr_result;
char *(*local)(char *, struct svc_req *);
switch (rqstp->rq_proc) {
case NULLPROC:(void) svc_sendreply (transp, (xdrproc_t) xdr_void, (char *)NULL);
return;
case GETTIME:
_xdr_argument = (xdrproc_t) xdr_void;
_xdr_result = (xdrproc_t) xdr_long;
local = (char *(*)(char *, struct svc_req *)) gettime_1_svc;
break;
default:
svcerr_noproc (transp);
return;
```

```
}
memset ((char *)&argument, 0, sizeof (argument));
if (!svc_getargs (transp, (xdrproc_t) _xdr_argument, (caddr_t) &argument)) {
svcerr_decode (transp);
return;
}
result = (*local)((char *)&argument, rqstp);
if (result != NULL && !svc_sendreply(transp, (xdrproc_t) _xdr_result, result)) {
svcerr_systemerr (transp);
}
if (!svc_freeargs (transp, (xdrproc_t) _xdr_argument, (caddr_t) &argument)) {
fprintf (stderr, "%s", "unable to free arguments");
exit (1);
}
return;
}
intmain (int argc, char **argv){
register SVCXPRT *transp;
pmap_unset (RPCTIME, RPCTIMEVERSION);
transp = svcudp_create(RPC_ANYSOCK);
if (transp == NULL) {
fprintf (stderr, "%s", "cannot create udp service.");
exit(1);
}
if (!svc_register(transp, RPCTIME, RPCTIMEVERSION, rpctime_1, IPPROTO_UDP)) {
fprintf (stderr, "%s", "unable to register (RPCTIME, RPCTIMEVERSION,udp).");
exit(1);
}
transp = svctcp_create(RPC_ANYSOCK, 0, 0);if (transp == NULL) {
fprintf (stderr, "%s", "cannot create tcp service.");
exit(1);
```

```
}
if (!svc_register(transp, RPCTIME, RPCTIMEVERSION, rpctime_1, IPPROTO_TCP)) {
fprintf (stderr, "%s", "unable to register (RPCTIME, RPCTIMEVERSION, tcp).");
exit(1);
}
svc_run ();
fprintf (stderr, "%s", "svc_run returned");
exit (1);
}
```

Client Side:

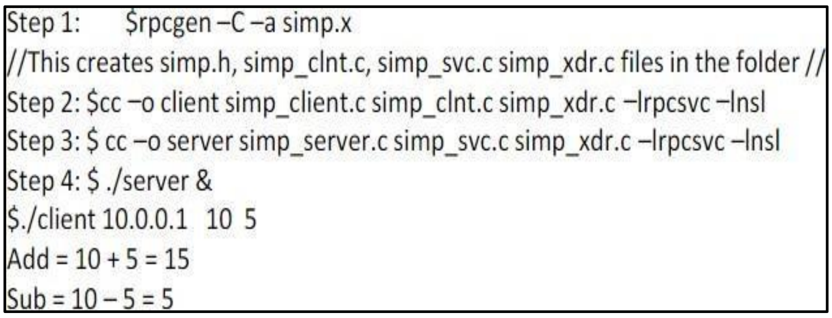
```
#include "rpctime.h"
voidrpctime_1(char *host){
CLIENT *clnt;47long *result_1;
char *gettime_1_arg;
#ifndef DEBUGclnt = clnt_create (host, RPCTIME, RPCTIMEVERSION, "udp");
if (clnt == NULL) {
clnt_pcreateerror (host);
exit (1);
}
#endif /* DEBUG */
result_1 = gettime_1((void*)&gettime_1_arg, clnt);
if (result_1 == (long *) NULL) {
clnt_perror (clnt, "call failed");
}
Else
printf("%d |%s", *result_1, ctime(result_1));
#ifndef DEBUGclnt_destroy (clnt);
#endif /* DEBUG */}
intmain (int argc, char *argv[]){
char *host;
if (argc < 2) {
```

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

exit (1);

}

host = argv[1];

rpctime_1 (host);

exit (0);

}
```

rpctime_cntl.c

```
#include <memory.h> /* for memset */

#include "rpctime.h"/* Default timeout can be changed using clnt_control() */

static struct timeval TIMEOUT = { 25, 0 };

long *gettime_1(void *argp, CLIENT *clnt){

static long clnt_res;memset((char *)&clnt_res, 0, sizeof(clnt_res));

if (clnt_call (clnt, GETTIME,(xdrproc_t) xdr_void, (caddr_t) argp,(xdrproc_t) xdr_long, (caddr_t)

&clnt_res,TIMEOUT) != RPC_SUCCESS) {

return (NULL);

}

return (&clnt_res);

}
```

OUTPUT:



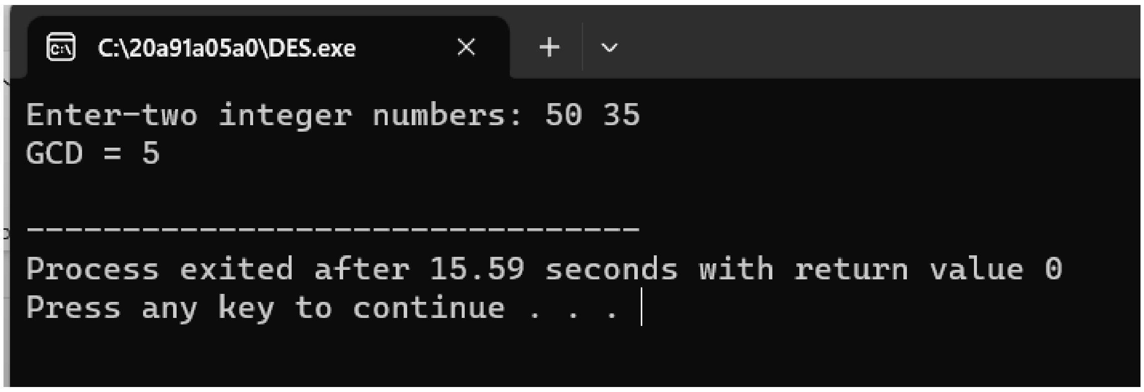
```
Step 1:      $rpcgen –C –a simp.x
//This creates simp.h, simp_clnt.c, simp_svc.c simp_xdr.c files in the folder //
Step 2: $cc –o client simp_client.c simp_clnt.c simp_xdr.c –lrpcsvc –lnsl
Step 3: $ cc –o server simp_server.c simp_svc.c simp_xdr.c –lrpcsvc –lnsl
Step 4: $ ./server &
$./client 10.0.0.1   10  5
Add = 10 + 5 = 15
Sub = 10 – 5 = 5
```

Augumented Experiments:

PROGRAM:

```
#include <stdio.h>
 void main() {
int m, n; /* given numbers */
 clrscr();
printf("Enter-two integer numbers: ");
scanf ("%d %d", &m, &n);
while (n > 0) {
int r = m % n;
m = n;
n = r;
}
printf ("GCD = %d \n",m); getch();
}
```

OUTPUT:

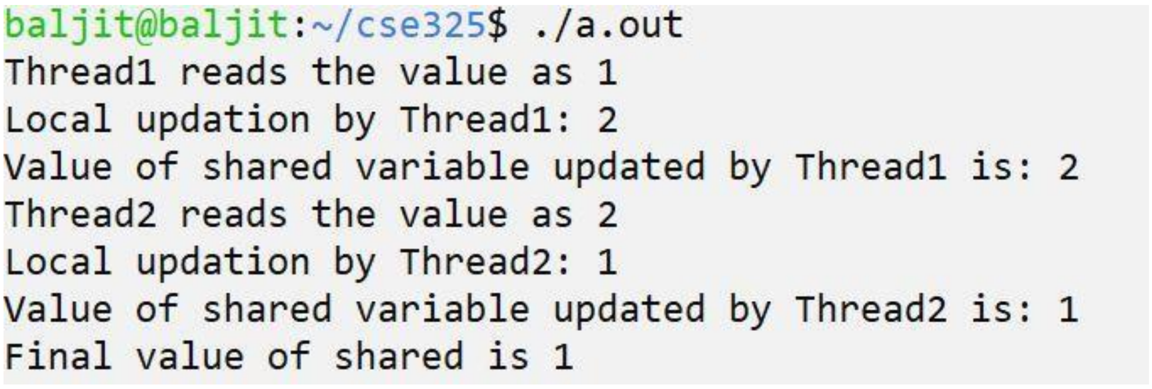


```
Enter-two integer numbers: 50 35
GCD = 5

--------------------------------
Process exited after 15.59 seconds with return value 0
Press any key to continue . . .
```

PROGRAM

```
#include<pthread.h>

#include<stdio.h>

#include<semaphore.h>

#include<unistd.h>

void *fun1();

void *fun2();

int shared=1; //shared variable

sem_t s; //semaphore variable

int main()

{

sem_init(&s,0,1); //initialize semaphore variable - 1st argument is address of variable, 2nd is number
of

processes sharing semaphore, 3rd argument is the initial value of semaphore variable

pthread_t thread1, thread2;

pthread_create(&thread1, NULL, fun1, NULL);

pthread_create(&thread2, NULL, fun2, NULL);

pthread_join(thread1, NULL);

pthread_join(thread2,NULL);

printf("Final value of shared is %d\n",shared); //prints the last updated value of shared variable

}

void *fun1()

{

 int x;

 sem_wait(&s); //executes wait operation on s

 x=shared;//thread1 reads value of shared variable

 printf("Thread1 reads the value as %d\n",x);

 x++; //thread1 increments its value

140

 printf("Local updation by Thread1: %d\n",x);

 sleep(1); //thread1 is preempted by thread 2
 shared=x; //thread one updates the value of shared variable
 printf("Value of shared variable updated by Thread1 is: %d\n",shared);
 sem_post(&s);
}
void *fun2()
{
 int y;
 sem_wait(&s);
 y=shared;//thread2 reads value of shared variable
 printf("Thread2 reads the value as %d\n",y);
 y--; //thread2 increments its value
 printf("Local updation by Thread2: %d\n",y);
 sleep(1); //thread2 is preempted by thread 1
 shared=y; //thread2 updates the value of shared variable
 printf("Value of shared variable updated by Thread2 is: %d\n",shared);
 sem_post(&s);
}
```

OUTPUT:



```
baljit@baljit:~/cse325$ ./a.out
Thread1 reads the value as 1
Local updation by Thread1: 2
Value of shared variable updated by Thread1 is: 2
Thread2 reads the value as 2
Local updation by Thread2: 1
Value of shared variable updated by Thread2 is: 1
Final value of shared is 1
```