

BASIC OPERATIONS ON MESSAGE QUEUES

(SHARED PROCESS ADDRESS SPACE)

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>

typedef struct msg
{
    long id;
    char txt[10];
}message;

main()
{
    int mqid,pid,rval,msgid;
    system("clear");
    printf("\nBasic operations on message queues\n");
    printf("\nPP : process id is %d\n",getpid());
    mqid=msgget((key_t)80,IPC_CREAT|0666);
    if(mqid==-1)
    {
        perror("PP : MQ-CRE-ERR");
        exit(1);
    }
    system("ipcs -q");
    printf("\nEnter the msg id : ");
    scanf("%d",&msgid);
    pid=fork();
    if(pid==-1)
    {
        perror("PP : FRK-ERR");
        msgctl(mqid,IPC_RMID,0);
        system("ipcs -q");
    }
}
```

```

        exit(1);
    }
    if(pid==0)
    {
        message m1;
        printf("\nCP : In child process\tpid=%d\n",getpid());
        printf("\nCP : Parent process id\tppid=%d\n",getppid());
        //printf("\nCP : Enter msgid\t:\t");
        //scanf("%d",&m1.id);
        m1.id=msgid;
        printf("\nEnter the message : ");
        scanf("%s",m1.txt);
        rval=msgsnd(mqid,(message*)&m1,sizeof(m1),0);
        if(rval==-1)
            perror("\nCP : Unable to send message\n");
        else
            printf("\nCP : message successfully sent\n");
    }
    else
    {
        message m2;
        rval=msgrcv(mqid,(message*)&m2,sizeof(m2),msgid,0);
        if(rval==-1)
            perror("\nPP : No message read\n");
        else
            printf("\n PP : Client message read from queue is %s\n",m2.txt);
        rval=msgctl(mqid,IPC_RMID,0);
        system("ipcs -q");
    }
}

```

OUTPUT:

```
Select Telnet 10.2.0.5

Basic operations on message queues

PP : process id is 4853

----- Message Queues -----
key      msqid      owner    perms    used-bytes  messages
0x00000050 0          cse1680  666      0           0

Enter the msg id : 1

CP : In child process  pid=4856
CP : Parent process id  ppid=4853

Enter the message : hey

CP : message successfully sent

PP : Client message read from queue is hey

----- Message Queues -----
key      msqid      owner    perms    used-bytes  messages

[cse1680@mars CNP]$
```

BASIC OPERATIONS ON MESSAGE QUEUES

(NON-SHARED PROCESS ADDRESS SPACE)

Program:

```
/*CLIENT PROGRAM*/
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>

typedef struct msg
{
    long id;
    char txt[10];
}message;

main()
{
    int mqid,rval,msgid;
    message m1;
    mqid=msgget((key_t)80,IPC_CREAT|0666);
    if(mqid==-1)
    {
        perror("MQ-CRE-ERR");
        exit(1);
    }
    system("ipcs -q");
    printf("Enter the msgid : ");
    scanf("%d",&msgid);
    m1.id=msgid;
    printf("Enter the message : ");
    scanf("%s",m1.txt);
    rval=msgsnd(mqid,(message*)&m1,sizeof(m1),0);
    if(rval==-1)
        perror("\nMessage not sent\n");
```

```

        else

            printf("\nMessage Sent successfully\n");

    }

    /*SERVER PROGRAM*/
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>

typedef struct msg
{
    long id;
    char txt[10];
}message;

main()
{
    int mqid,rval,msgid;
    message m2;
    mqid=msgget((key_t)80,IPC_CREAT|0666);
    if(mqid==-1)
    {
        perror("MQ-CRE-ERR");
        exit(1);
    }
    system("ipcs -q");

    printf("\nEnter the msgid :");
    scanf("%d",&msgid);
    rval=msgrcv(mqid,(message*)&m2,sizeof(m2),msgid,0);
    if(rval==-1)
    {
        perror("No message read");
        msgctl(mqid,IPC_RMID,0);
        system("ipcs -q");
        exit(1);
    }

```

```
    }  
    printf("\nMessage entered : %s\n",m2.txt);  
    msgctl(mqid,IPC_RMID,0);  
    system("ipcs -q");  
}
```

OUTPUT:

CLIENT

```
Telnet 10.2.0.5
[cse1680@mars CNP]$ ./cout

----- Message Queues -----
key      msgid    owner    perms    used-bytes  messages
0x00000042 229376   cse1666  666      32          2
0x00000050 786433   cse1680  666      0           0

Enter the msgid : 1
Enter the message : hey

Message Sent successfully
[cse1680@mars CNP]$
```

SERVER

```
Telnet 10.2.0.5
[cse1680@mars CNP]$ ./sout

----- Message Queues -----
key      msgid    owner    perms    used-bytes  messages
0x00000042 229376   cse1666  666      32          2
0x00000050 753665   cse1680  666      16          1

Enter the msgid :1

Message entered : hey

----- Message Queues -----
key      msgid    owner    perms    used-bytes  messages
0x00000042 229376   cse1666  666      32          2

[cse1680@mars CNP]$
```

UDP CLIENT TO ACCESS WELL KNOWN / STANDARD PORT SERVICE

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<sys/types.h>
#include<string.h>
main(int argc,char* argv[])
{
    struct sockaddr_in s,c; //sockets for server and client
    int rval,sockid,slen;
    char m1[20],m2[20];
    system("clear");
    if(argc<3)
    {
        printf("\nUSAGE : %s IP-Address Port#\n",argv[0]);
        exit(1);
    }
    sockid=socket(PF_INET,SOCK_DGRAM,17);
    if(sockid==-1)
    {
        perror("SOCK-CRE-ERR:");
        exit(1);
    }
    s.sin_family=PF_INET;
    s.sin_port=htons(atoi(argv[2]));
    s.sin_addr.s_addr=inet_addr(argv[1]);
    c.sin_port=htons(5080);

    printf("\nEnter the request message : ");
    scanf("%s",m1);
    slen=sizeof(s);
    rval=sendto(sockid,m1,sizeof(m1),0,(struct sockaddr*)&s,slen);
    if(rval==-1)
```



```

{
    perror("MSG-SEND-ERR:");
    exit(1);
}
printf("\nMessage sent successfully\n");
strncpy(m2," ",20);
rval=recvfrom(sockid,m2,sizeof(m2),0,(struct sockaddr*)&s,&slen);
if(rval==-1)
{
    perror("MSG-RCV-ERR:");
    exit(1);
}
printf("\nMessage received is : %s\n",m2);
close(sockid);
}

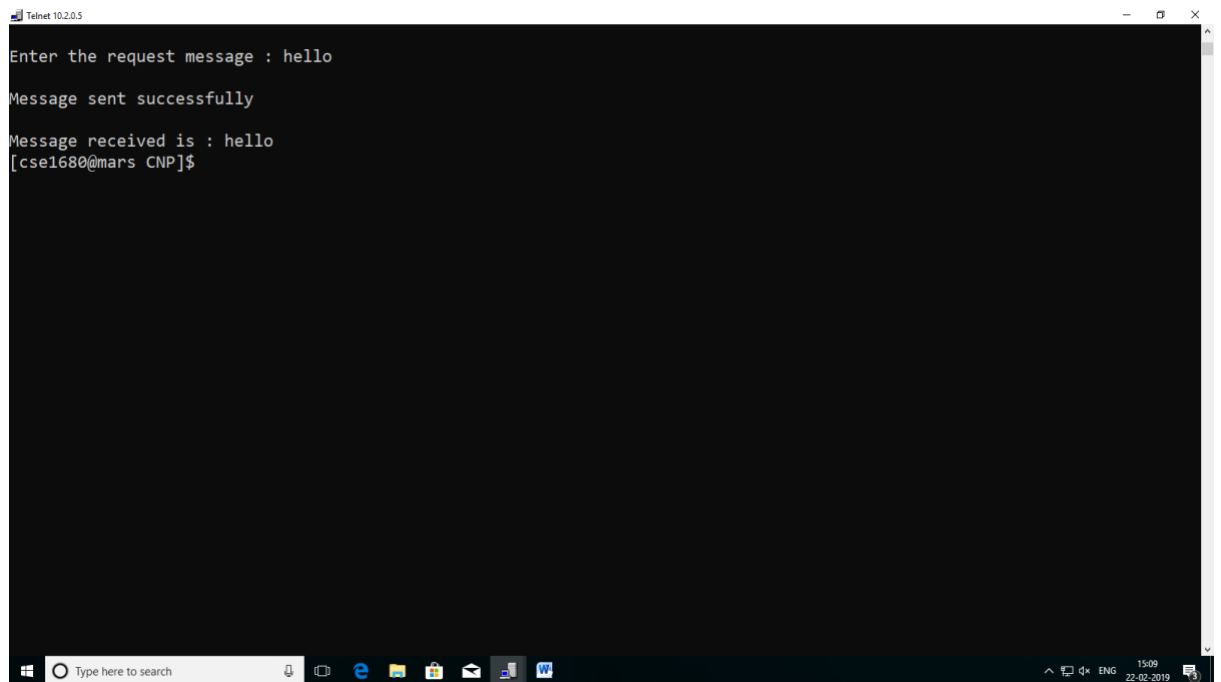
```

OUTPUT:

compilation : gcc -o cout udp_client.c

execution : ./cout 10.2.0.7 7 (7 is the port for standard echo service)

CLIENT



```
Telnet 10.2.0.5
Enter the request message : hello
Message sent successfully
Message received is : hello
[cse1680@mars CNP]$
```

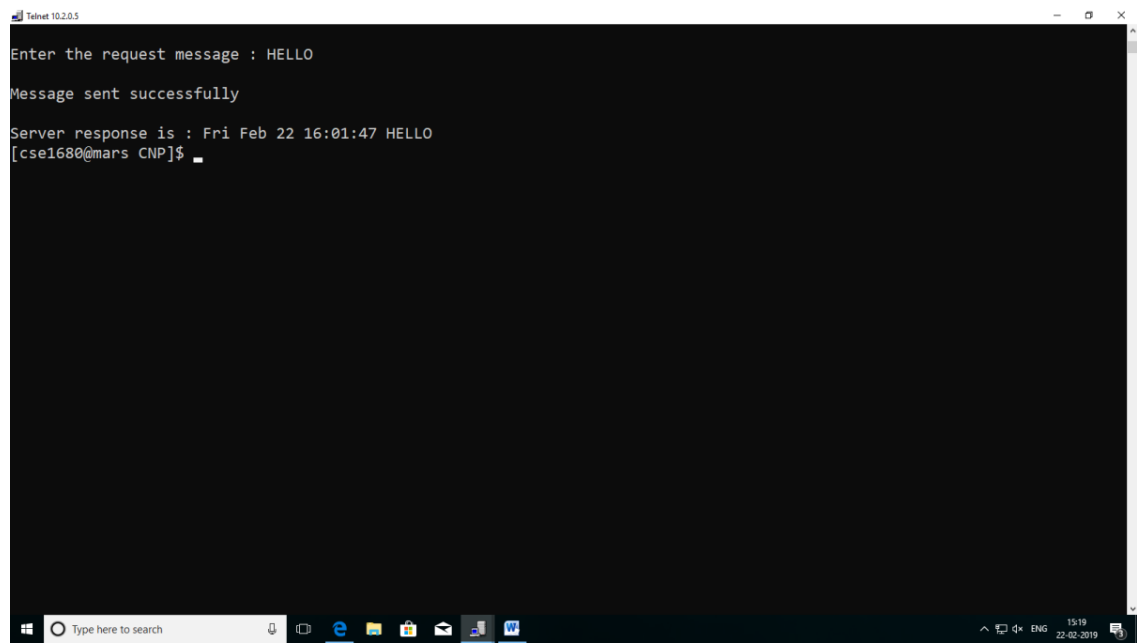
The screenshot shows a Windows desktop environment with a Telnet window titled 'Telnet 10.2.0.5'. The window has a black background with white text. The text inside the window shows the user entering 'hello' as a request message, receiving a confirmation 'Message sent successfully', and then seeing the received message 'Message received is : hello'. The prompt '[cse1680@mars CNP]\$' is visible at the bottom of the window. The Windows taskbar is visible at the bottom of the screen, showing the search bar and several application icons.

OUTPUT:

compilation : gcc -o cout udp_client.c

execution : ./cout 10.2.0.5 13 (13 is the port for standard daytime service)

CLIENT



```
Telnet 10.2.0.5
Enter the request message : HELLO
Message sent successfully
Server response is : Fri Feb 22 16:01:47 HELLO
[cse1680@mars CNP]$
```

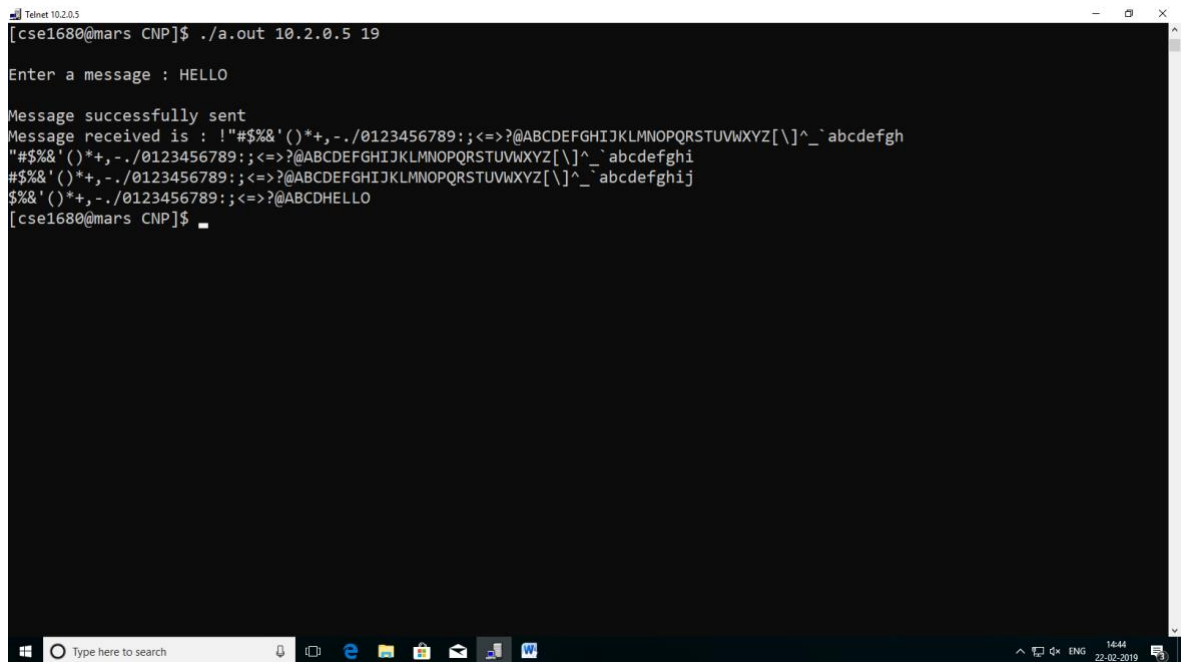
The screenshot shows a Windows desktop environment with a Telnet client window titled "Telnet 10.2.0.5". The window has a black background with white text. The user has entered "HELLO" as the request message, and the client reports "Message sent successfully". The server response is "Fri Feb 22 16:01:47 HELLO". The prompt "[cse1680@mars CNP]\$" is visible at the bottom of the window. The Windows taskbar is visible at the bottom of the screen, showing the search bar and several application icons.

OUTPUT:

compilation : gcc -o cout udp_client.c

execution : ./cout 10.2.0.5 19 (19 is the port for standard char-gen service)

CLIENT



```
Telnet 10.2.0.5
[cse1680@mars CNP]$ ./a.out 10.2.0.5 19

Enter a message : HELLO

Message successfully sent
Message received is : !"#%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMN
PQRSTUVWXYZ[\]^_`abcdefg
h"#%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMN
PQRSTUVWXYZ[\]^_`abcde
fghi"#%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMN
PQRSTUVWXYZ[\]^_`abcde
fghij"$%&'()*+,-./0123456789:;<=>?@ABCDHELLO
[cse1680@mars CNP]$
```

TCP CLIENT TO ACCESS WELL KNOW SERVICE/STANDARD PORT SERVICE

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>

main(int argc,char* argv[])
{
    int sockid,rval;
    char m1[20],m2[20];
    sockid=socket(AF_INET,SOCK_STREAM,0);
    if(sockid==-1)
    {
        perror("SOCK-CRE-ERR");
        exit(1);
    }
    struct sockaddr_in s;
    system("clear");
    if(argc<3)
    {
        printf("\nUSAGE : %s IP_ADDR PORT#\n",argv[0]);
        exit(0);
    }
    s.sin_family=AF_INET;
    s.sin_port=htons(atoi(argv[2]));
    s.sin_addr.s_addr=inet_addr(argv[1]);

    rval=connect(sockid,(struct sockaddr*)&s, sizeof(s));
    if(rval==-1)
    {
        perror("CONN-ERR:");
        close(sockid);
        exit(1);
    }
}
```

```

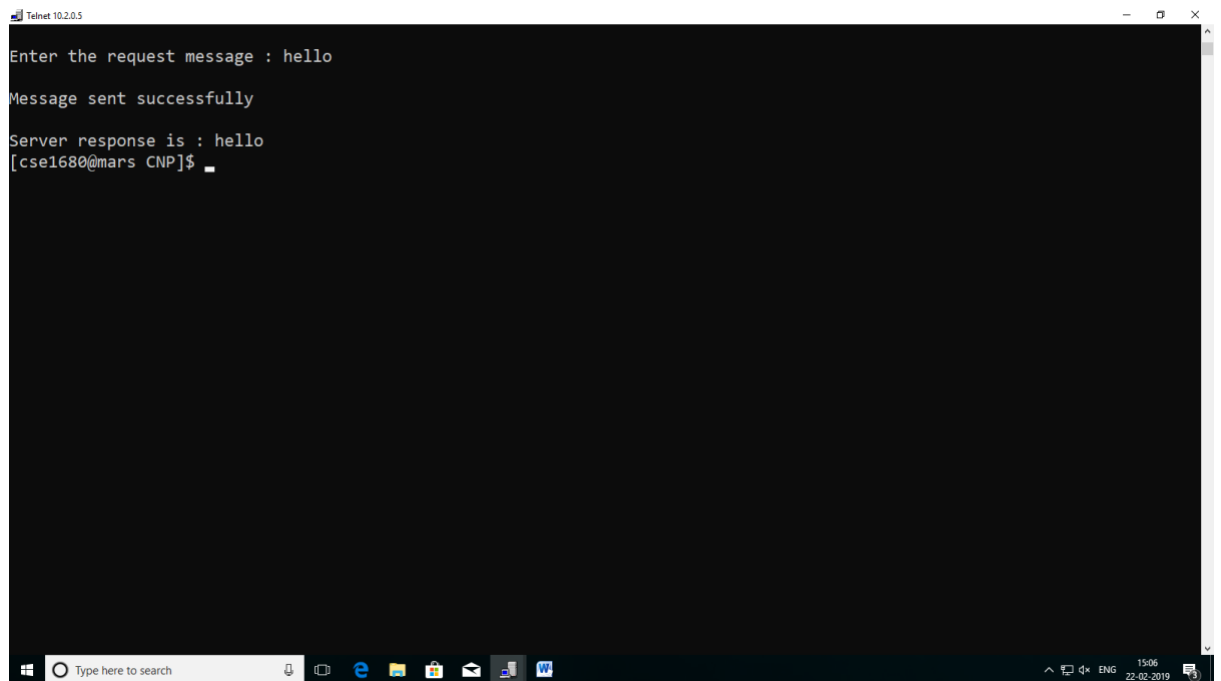
    }
    printf("\nEnter the request message : ");
    scanf("%s",m1);
    rval=send(sockid,m1,sizeof(m1),0);
    if(rval==-1)
    {
        perror("MSG-SND-ERR:");
        close(sockid);
        exit(1);
    }
    printf("\nMessage sent successfully\n");
    rval=recv(sockid,m2,sizeof(m2),0);
    if(rval==-1)
    {
        perror("MSG-RCV-ERR:");
        close(sockid);
        exit(1);
    }
    printf("\nServer response is : %s\n",m2);
    close(sockid);
}

```

OUTPUT:

compilation : gcc -o cout tcp_client.c

execution : ./cout 10.2.0.7 7 (7 is the port for standard echo service)



```
Telnet 10.2.0.5
Enter the request message : hello
Message sent successfully
Server response is : hello
[cse1680@mars CNP]$
```

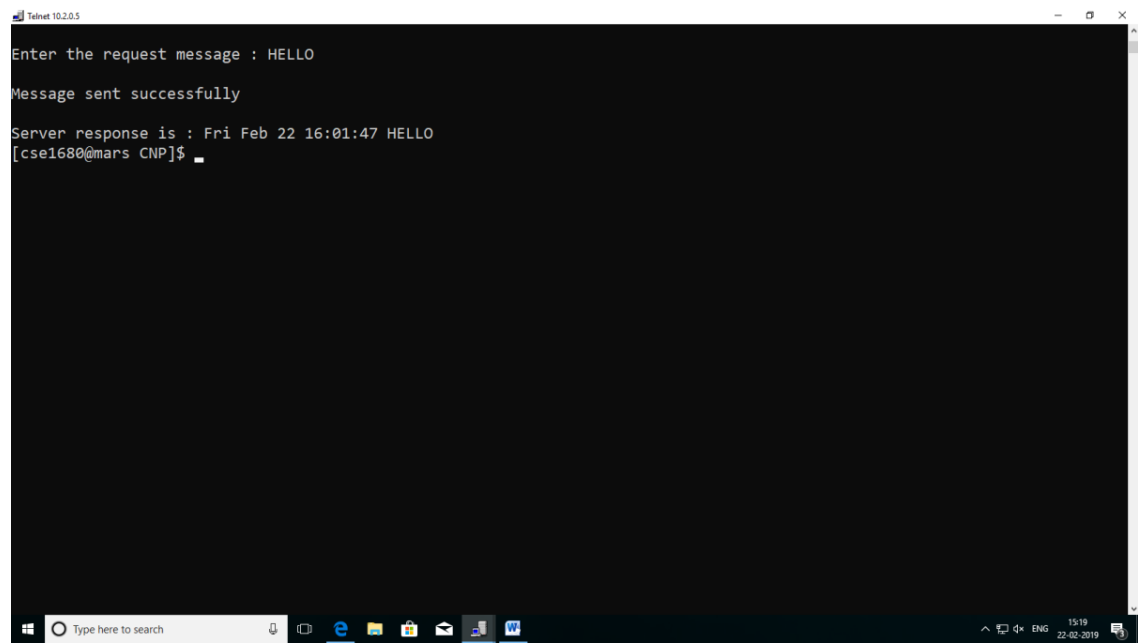
The screenshot shows a Windows desktop environment with a Telnet window open. The window title is 'Telnet 10.2.0.5'. The text inside the window shows a successful Telnet session where the user entered 'hello', the message was sent successfully, and the server responded with 'hello'. The prompt at the bottom of the window is '[cse1680@mars CNP]\$'. The Windows taskbar is visible at the bottom, showing the search bar and several application icons.

OUTPUT:

compilation : gcc -o cout tcp_client.c

execution : ./cout 10.2.0.5 13 (13 is the port for standard daytime service)

CLIENT



```
Telnet 10.2.0.5
Enter the request message : HELLO
Message sent successfully
Server response is : Fri Feb 22 16:01:47 HELLO
[cse1680@mars CNP]$
```

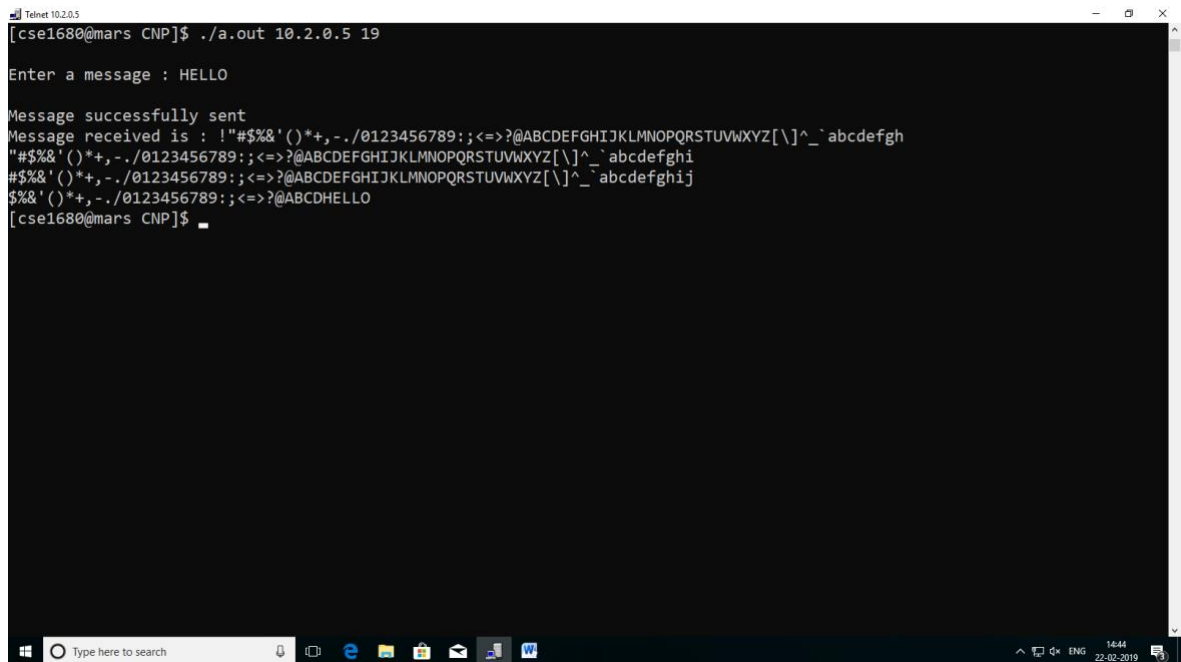
The screenshot shows a Windows desktop environment with a Telnet client window titled 'Telnet 10.2.0.5'. The window has a black background with white text. The user has entered 'HELLO' as the request message, and the client reports 'Message sent successfully'. The server response is 'Fri Feb 22 16:01:47 HELLO'. The prompt '[cse1680@mars CNP]\$' is visible at the bottom of the window. The Windows taskbar is visible at the bottom of the screen, showing the search bar and several application icons.

OUTPUT:

compilation : gcc -o cout tcp_client.c

execution : ./cout 10.2.0.5 19 (19 is the port for standard char-gen service)

CLIENT



```
Telnet 10.2.0.5
[cse1680@mars CNP]$ ./a.out 10.2.0.5 19

Enter a message : HELLO

Message successfully sent
Message received is : !"#%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNopQRSTUVWXYZ[\]^_`abcdefg
h"#%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNopQRSTUVWXYZ[\]^_`abcdefghi
"#%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNopQRSTUVWXYZ[\]^_`abcdefghij
"%&'()*+,-./0123456789:;<=>?@ABCDHELLO
[cse1680@mars CNP]$
```

USER-DEFINED UDP ECHO SERVICE

Program:

/*CLIENT PROGRAM*/

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<sys/types.h>
#include<string.h>
main(int argc,char* argv[])
{
    struct sockaddr_in s,c; //sockets for server and client
    int rval,sockid,slen;
    char m1[20],m2[20];
    system("clear");
    if(argc<3)
    {
        printf("\nUSAGE : %s IP-Address Port#\n",argv[0]);
        exit(1);
    }
    sockid=socket(PF_INET,SOCK_DGRAM,17);
    if(sockid==-1)
    {
        perror("SOCK-CRE-ERR:");
        exit(1);
    }
    s.sin_family=PF_INET;
    s.sin_port=htons(atoi(argv[2]));
    s.sin_addr.s_addr=inet_addr(argv[1]);
    c.sin_port=htons(5080);

    printf("\nEnter the request message : ");
    scanf("%s",m1);
    slen=sizeof(s);
    rval=sendto(sockid,m1,sizeof(m1),0,(struct sockaddr*)&s,slen);
    if(rval==-1)
```

```

        {
            perror("MSG-SEND-ERR:");
            exit(1);
        }
        printf("\nMessage sent successfully\n");
        strncpy(m2," ",20);
        rval=recvfrom(sockid,m2,sizeof(m2),0,(struct sockaddr*)&s,&slen);
        if(rval==-1)
        {
            perror("MSG-RCV-ERR:");
            exit(1);
        }
        printf("\nMessage received is : %s\n",m2);
        close(sockid);
    }
}

/*SERVER PROGRAM*/
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>

main(int argc,char* argv[])
{
    int sockid,rval,clen;
    char buffer[20];
    struct sockaddr_in s,c;
    if(argc<3)
    {
        printf("\nUSAGE : %s IP_ADDRESS PORT#\n",argv[0]);
        exit(0);
    }
    sockid=socket(PF_INET,SOCK_DGRAM,17);
    if(sockid==-1)
    {
        perror("SOCK-CRE-ERR:");
        exit(1);
    }

```

```

    }
    s.sin_family=PF_INET;
    s.sin_port=htons(atoi(argv[2]));
    s.sin_addr.s_addr=inet_addr(argv[1]);
    rval=bind(sockid,(struct sockaddr*)&s,sizeof(s));
    if(rval==-1)
    {
        perror("BIND-ERR");
        close(sockid);
        exit(1);
    }
    clen=sizeof(c);
    rval=recvfrom(sockid,buffer,sizeof(buffer),0,(struct sockaddr*)&c,&clen);
    if(rval==-1)
    {
        perror("MSG-RCV-ERR:");
    }
    else
    {
        printf("\nRequest received\nRequest message is : %s\n",buffer);
    }
    rval=sendto(sockid,buffer,sizeof(buffer),0,(struct sockaddr*)&c,sizeof(c));
    if(rval==-1)
    {
        perror("MSG-SND-ERR:");
    }
    else
    {
        printf("\nResponse sent successfully\n");
    }
    close(sockid);
}

```

OUTPUT:

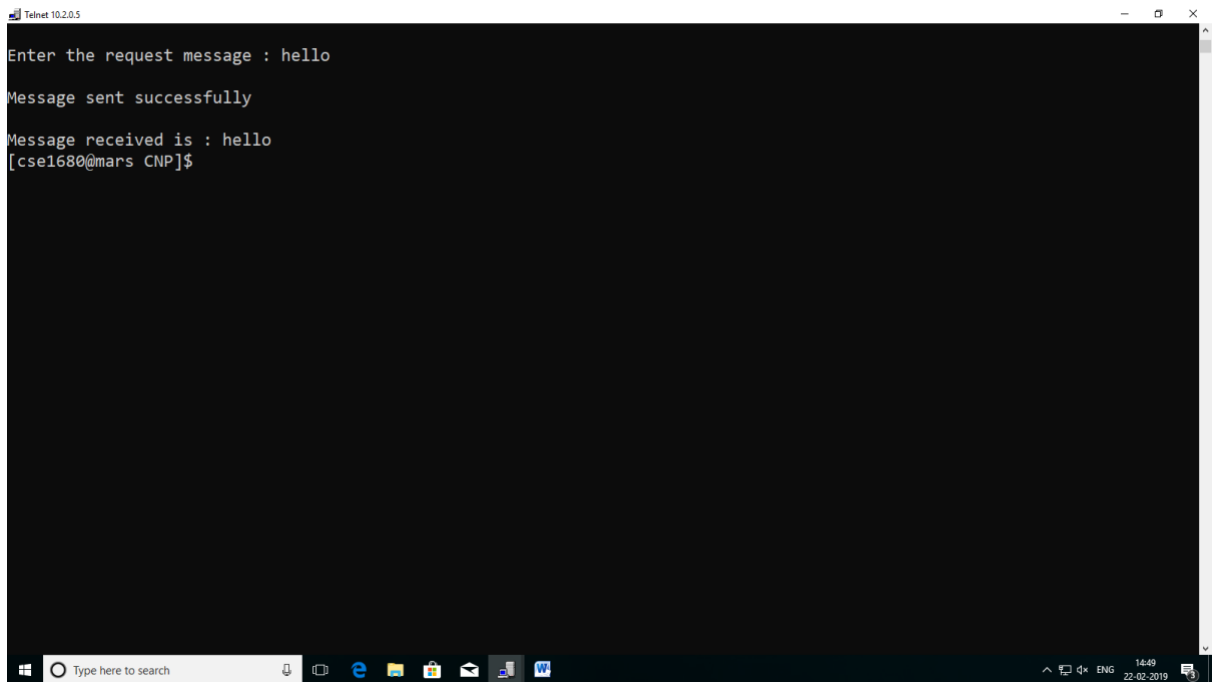
compilation : gcc -o cout udp_client.c

gcc -o sout udp_echosrv.c

execution : ./sout 10.2.0.5 5080 (server terminal)

./cout 10.2.0.5 5080 (client terminal)

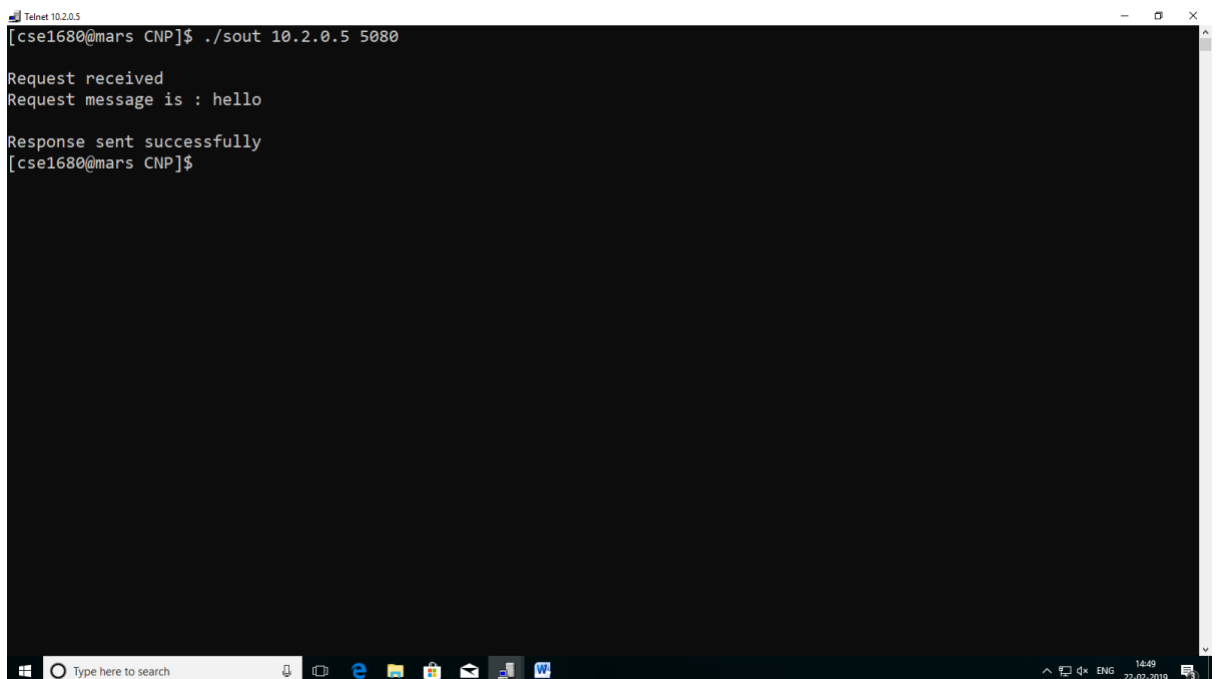
CLIENT



```
Telnet 10.2.0.5
Enter the request message : hello
Message sent successfully
Message received is : hello
[cse1680@mars CNP]$
```

The screenshot shows a Windows desktop environment with a Telnet client window titled 'Telnet 10.2.0.5'. The window contains the following text: 'Enter the request message : hello', 'Message sent successfully', 'Message received is : hello', and the prompt '[cse1680@mars CNP]\$'. The Windows taskbar is visible at the bottom, showing the search bar and several application icons.

SERVER



```
Telnet 10.2.0.5
[cse1680@mars CNP]$ ./sout 10.2.0.5 5080
Request received
Request message is : hello
Response sent successfully
[cse1680@mars CNP]$
```

The screenshot shows a Windows desktop environment with a Telnet server window titled 'Telnet 10.2.0.5'. The window contains the following text: '[cse1680@mars CNP]\$./sout 10.2.0.5 5080', 'Request received', 'Request message is : hello', 'Response sent successfully', and the prompt '[cse1680@mars CNP]\$'. The Windows taskbar is visible at the bottom, showing the search bar and several application icons.

USER-DEFINED UDP DAYTIME SERVICE

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>

main(int argc,char* argv[])
{
    int sockid,rval,clen;
    char buffer[20],smsg[30];
    time_t t;
    struct sockaddr_in s,c;
    if(argc<3)
    {
        printf("\nUSAGE : %s IP_ADDRESS PORT#\n",argv[0]);
        exit(0);
    }
    sockid=socket(PF_INET,SOCK_DGRAM,17);
    if(sockid==-1)
    {
        perror("SOCK-CRE-ERR:");
        exit(1);
    }
    s.sin_family=PF_INET;
    s.sin_port=htons(atoi(argv[2]));
    s.sin_addr.s_addr=inet_addr(argv[1]);
    rval=bind(sockid,(struct sockaddr*)&s,sizeof(s));
    if(rval==-1)
    {
        perror("BIND-ERR");
        close(sockid);
        exit(1);
    }
    clen=sizeof(c);
```

```

rval=recvfrom(sockid,buffer,sizeof(buffer),0,(struct sockaddr*)&c,&crlen);
if(rval==-1)
{
    perror("MSG-RCV-ERR:");
}
else
{
    printf("\nRequest received\nRequest message is : %s\n",buffer);
}
t=time(0);
strcpy(smsg,ctime(&t));
rval=sendto(sockid,smsg,sizeof(smsg),0,(struct sockaddr*)&c,sizeof(c));
if(rval==-1)
{
    perror("MSG-SND-ERR:");
}
else
{
    printf("\nResponse sent successfully\n");
}
close(sockid);
}

```

OUTPUT:

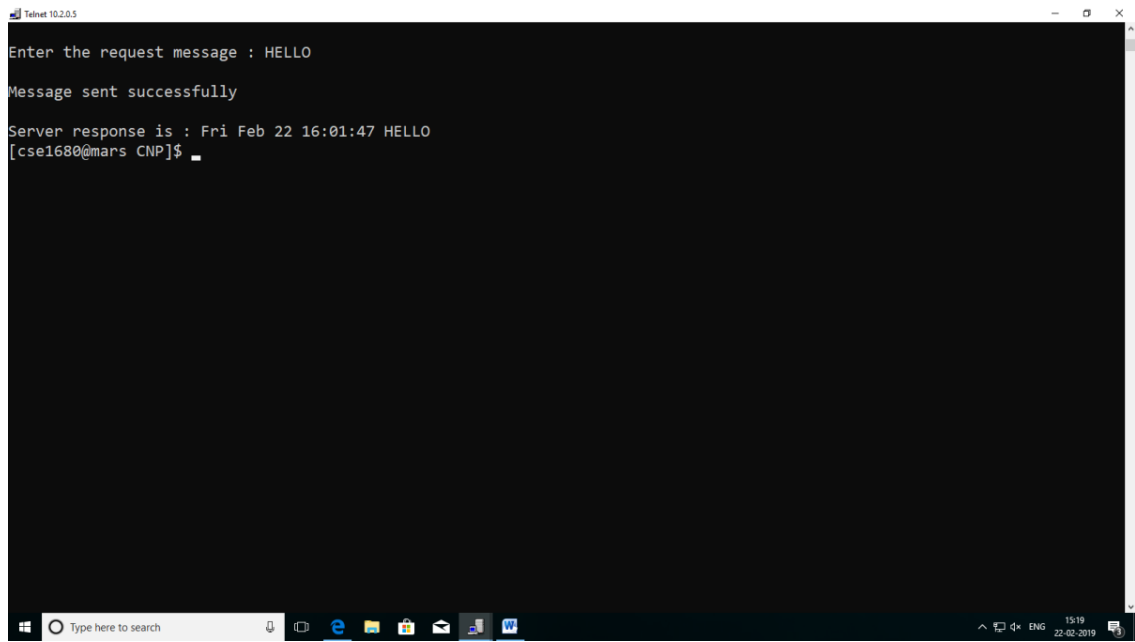
compilation : gcc -o cout udp_client.c

gcc -o sout udp_dts.c

execution : ./sout 10.2.0.5 5080 (server terminal)

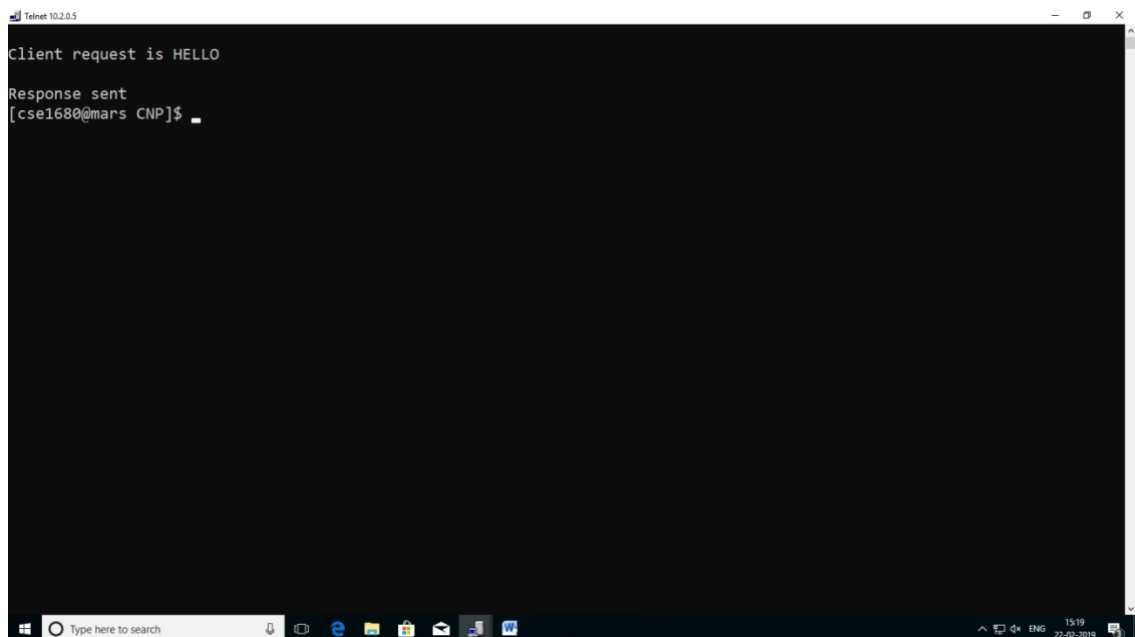
./cout 10.2.0.5 5080 (client terminal)

CLIENT



```
Telnet 10.2.0.5
Enter the request message : HELLO
Message sent successfully
Server response is : Fri Feb 22 16:01:47 HELLO
[cse1680@mars CNP]$
```

SERVER



```
Telnet 10.2.0.5
Client request is HELLO
Response sent
[cse1680@mars CNP]$
```


USER-DEFINED TCP ECHO SERVICE

Program:

```
/*CLIENT PROGRAM*/
```

```
#include<stdio.h>
```

```
#include<stdlib.h>
```

```
#include<sys/types.h>
```

```
#include<sys/socket.h>
```

```
#include<netinet/in.h>
```

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

```
{
```

```
    int sockid,rval;
```

```
    char m1[20],m2[20];
```

```
    sockid=socket(AF_INET,SOCK_STREAM,0);
```

```
    if(sockid==-1)
```

```
    {
```

```
        perror("SOCK-CRE-ERR");
```

```
        exit(1);
```

```
    }
```

```
    struct sockaddr_in s;
```

```
    system("clear");
```

```
    if(argc<3)
```

```
    {
```

```
        printf("\nUSAGE : %s IP_ADDR PORT#\n",argv[0]);
```

```
        exit(0);
```

```
    }
```

```
    s.sin_family=AF_INET;
```

```
    s.sin_port=htons(atoi(argv[2]));
```

```
    s.sin_addr.s_addr=inet_addr(argv[1]);
```

```
    rval=connect(sockid,(struct sockaddr*)&s, sizeof(s));
```

```
    if(rval==-1)
```

```
    {
```

```
        perror("CONN-ERR:");
```

```
        close(sockid);
```

```
        exit(1);
```

```

    }
    printf("\nEnter the request message : ");
    scanf("%s",m1);
    rval=send(sockid,m1,sizeof(m1),0);
    if(rval==-1)
    {
        perror("MSG-SND-ERR:");
        close(sockid);
        exit(1);
    }
    printf("\nMessage sent successfully\n");
    rval=recv(sockid,m2,sizeof(m2),0);
    if(rval==-1)
    {
        perror("MSG-RCV-ERR:");
        close(sockid);
        exit(1);
    }
    printf("\nServer response is : %s\n",m2);
    close(sockid);
}

/*SERVER PROGRAM*/
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>

main(int argc,char*argv[])
{
    int sid,sid1,rval;// sid is half association. sid1 is full association
    struct sockaddr_in s,c;
    char buffer[20];
    int clen; //accept() uses value-result parameter
    system("clear");
    if(argc<3)
    {

```

```

        printf("\nUSAGE : %s IP_ADDRESS PORT#\n",argv[0]);
        exit(0);
    }
    sid=socket(AF_INET,SOCK_STREAM,6);//3rd parameter can also be 0
    if(sid==-1)
    {
        perror("SOCK-CRE-ERR:");
        exit(1);
    }
    /*DEFINING NAME OF THE SERVICE*/
    s.sin_family=AF_INET;
    s.sin_port=htons(atoi(argv[2]));
    s.sin_addr.s_addr=inet_addr(argv[1]);

    /*BIND SOCKET- indicates the process that is listening*/
    rval=bind(sid,(struct sockaddr*)&s,sizeof(s));
    if(rval==-1)
    {
        perror("BIND-ERR:");
        close(sid);
        exit(1);
    }
    rval=listen(sid,5);//range : 1-5
    if(rval==-1)
    {
        perror("LISTEN-ERR:");
        close(sid);
        exit(1);
    }
    clen=sizeof(c);
    sid1=accept(sid,(struct sockaddr*)&c,&clen);
    //sid1 is a full association tuple and has information of client,server and communication
    protocol i.e serving socket
    rval=recv(sid1,buffer,sizeof(buffer),0);
    if(rval==-1)
    {
        perror("MSG-RCV-ERR:");

```

```
    }  
    else  
    {  
        printf("\nClient request is %s\n",buffer);  
    }  
    rval=send(sid1,buffer,sizeof(buffer),0);  
    if(rval==-1)  
    {  
        perror("MSG-SND-ERR:");  
    }  
    else  
    {  
        printf("\nResponse sent\n");  
    }  
    close(sid);  
    close(sid1);  
}
```

OUTPUT:

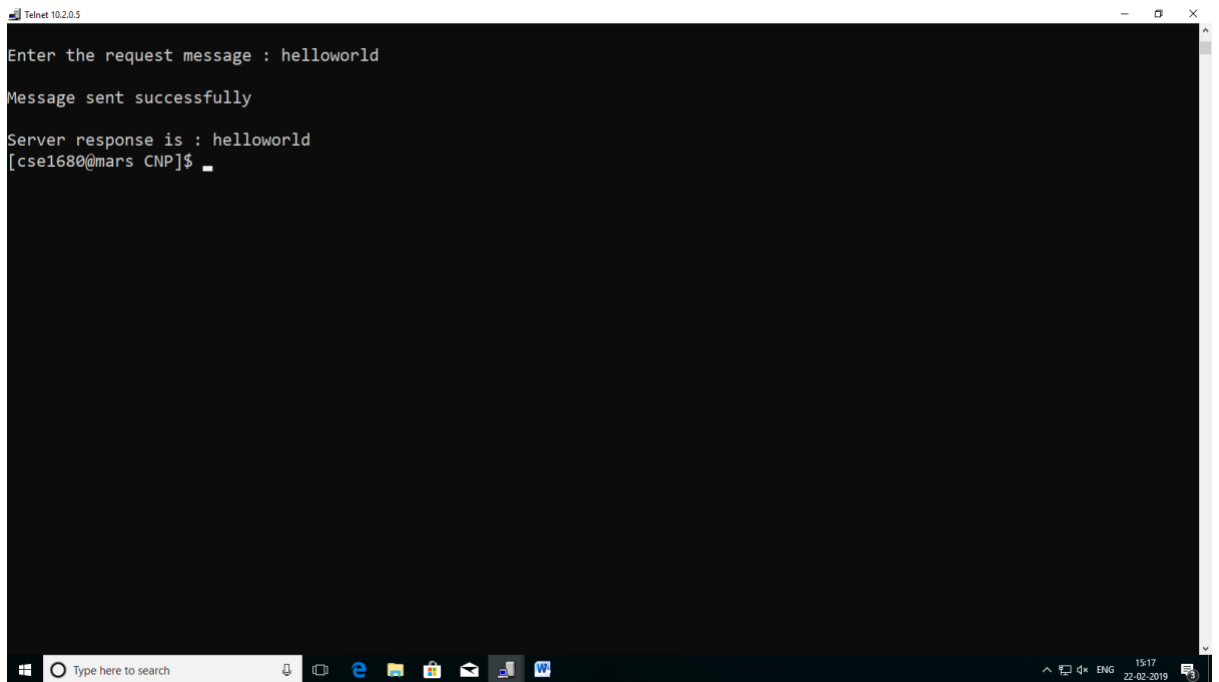
compilation : gcc -o cout tcp_client.c

gcc -o sout tcpserver_echo.c

execution : ./sout 10.2.0.5 5080 (server terminal)

./cout 10.2.0.5 5080 (client terminal)

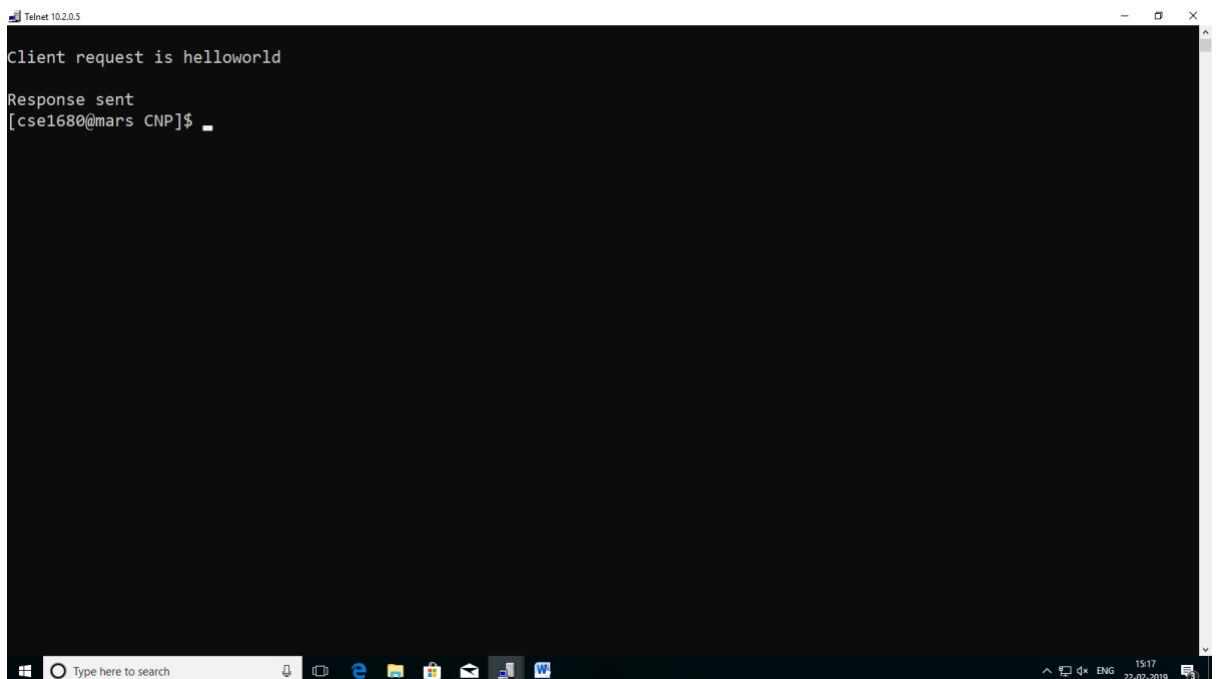
CLIENT



```
Telnet 10.2.0.5
Enter the request message : helloworld
Message sent successfully
Server response is : helloworld
[cse1680@mars CNP]$
```

The screenshot shows a Windows desktop environment with a Telnet client window titled 'Telnet 10.2.0.5'. The window has a black background with white text. The user enters 'helloworld' as the request message. The terminal displays 'Message sent successfully' and 'Server response is : helloworld'. The prompt '[cse1680@mars CNP]\$' is visible at the bottom. The Windows taskbar is visible at the bottom of the screen, showing the search bar and several application icons.

SERVER



```
Telnet 10.2.0.5
Client request is helloworld
Response sent
[cse1680@mars CNP]$
```

The screenshot shows a Windows desktop environment with a Telnet server window titled 'Telnet 10.2.0.5'. The window has a black background with white text. The terminal displays 'Client request is helloworld' and 'Response sent'. The prompt '[cse1680@mars CNP]\$' is visible at the bottom. The Windows taskbar is visible at the bottom of the screen, showing the search bar and several application icons.

USER-DEFINED TCP DAYTIME SERVICE

Program:

```
/*CLIENT PROGRAM*/
```

```
#include<stdio.h>
```

```
#include<stdlib.h>
```

```
#include<sys/types.h>
```

```
#include<sys/socket.h>
```

```
#include<netinet/in.h>
```

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

```
{
```

```
    int sockid,rval;
```

```
    char m1[20],m2[20];
```

```
    sockid=socket(AF_INET,SOCK_STREAM,0);
```

```
    if(sockid==-1)
```

```
    {
```

```
        perror("SOCK-CRE-ERR");
```

```
        exit(1);
```

```
    }
```

```
    struct sockaddr_in s;
```

```
    system("clear");
```

```
    if(argc<3)
```

```
    {
```

```
        printf("\nUSAGE : %s IP_ADDR PORT#\n",argv[0]);
```

```
        exit(0);
```

```
    }
```

```
    s.sin_family=AF_INET;
```

```
    s.sin_port=htons(atoi(argv[2]));
```

```
    s.sin_addr.s_addr=inet_addr(argv[1]);
```

```
    rval=connect(sockid,(struct sockaddr*)&s, sizeof(s));
```

```
    if(rval==-1)
```

```
    {
```

```
        perror("CONN-ERR:");
```

```
        close(sockid);
```

```
        exit(1);
```

```

    }
    printf("\nEnter the request message : ");
    scanf("%s",m1);
    rval=send(sockid,m1,sizeof(m1),0);
    if(rval==-1)
    {
        perror("MSG-SND-ERR:");
        close(sockid);
        exit(1);
    }
    printf("\nMessage sent successfully\n");
    rval=recv(sockid,m2,sizeof(m2),0);
    if(rval==-1)
    {
        perror("MSG-RCV-ERR:");
        close(sockid);
        exit(1);
    }
    printf("\nServer response is : %s\n",m2);
    close(sockid);
}

/*SERVER PROGRAM*/

#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<time.h>
#include<string.h>

main(int argc,char*argv[])
{
    int sid,sid1,rval;// sid is half association. sid1 is full association
    time_t t=time(0);
    struct sockaddr_in s,c;
    char buffer[20],smsg[30];

```

```

strcpy(smsg,ctime(&t));
int clen; //accept() uses value-result parameter
system("clear");
if(argc<3)
{
    printf("\nUSAGE : %s IP_ADDRESS PORT#\n",argv[0]);
    exit(0);
}
sid=socket(AF_INET,SOCK_STREAM,6);//3rd parameter can also be 0
if(sid==-1)
{
    perror("SOCK-CRE-ERR:");
    exit(1);
}
/*DEFINING NAME OF THE SERVICE*/
s.sin_family=AF_INET;
s.sin_port=htons(atoi(argv[2]));
s.sin_addr.s_addr=inet_addr(argv[1]);

/*BIND SOCKET- indicates the process that is listening*/
rval=bind(sid,(struct sockaddr*)&s,sizeof(s));
if(rval==-1)
{
    perror("BIND-ERR:");
    close(sid);
    exit(1);
}
rval=listen(sid,5);//range : 1-5
if(rval==-1)
{
    perror("LISTEN-ERR:");
    close(sid);
    exit(1);
}
clen=sizeof(c);
sid1=accept(sid,(struct sockaddr*)&c,&clen);

```


//sid1 is a full association tuple and has information of client,server and communication
protocol i.e serving socket

```
    rval=recv(sid1,buffer,sizeof(buffer),0);
    if(rval==-1)
    {
        perror("MSG-RCV-ERR:");
    }
    else
    {
        printf("\nClient request is %s\n",buffer);
    }
    strcpy(smsg,ctime(&t));//const time_t* if error
    rval=send(sid1,smsg,sizeof(smsg),0);
    if(rval==-1)
    {
        perror("MSG-SND-ERR:");
    }
    else
    {
        printf("\nResponse sent\n");
    }
    close(sid);
    close(sid1);
}
```

OUTPUT:

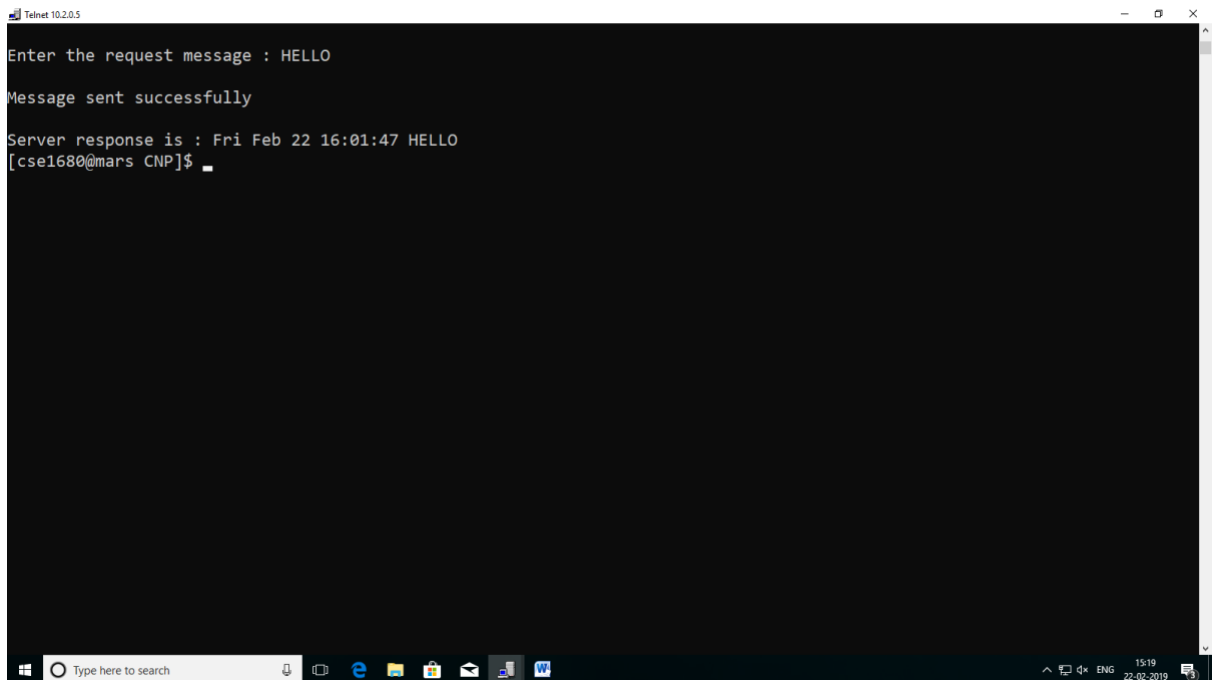
compilation : gcc -o cout tcp_client.c

gcc -o sout tcpserver_dts.c

execution : ./sout 10.2.0.5 5080 (server terminal)

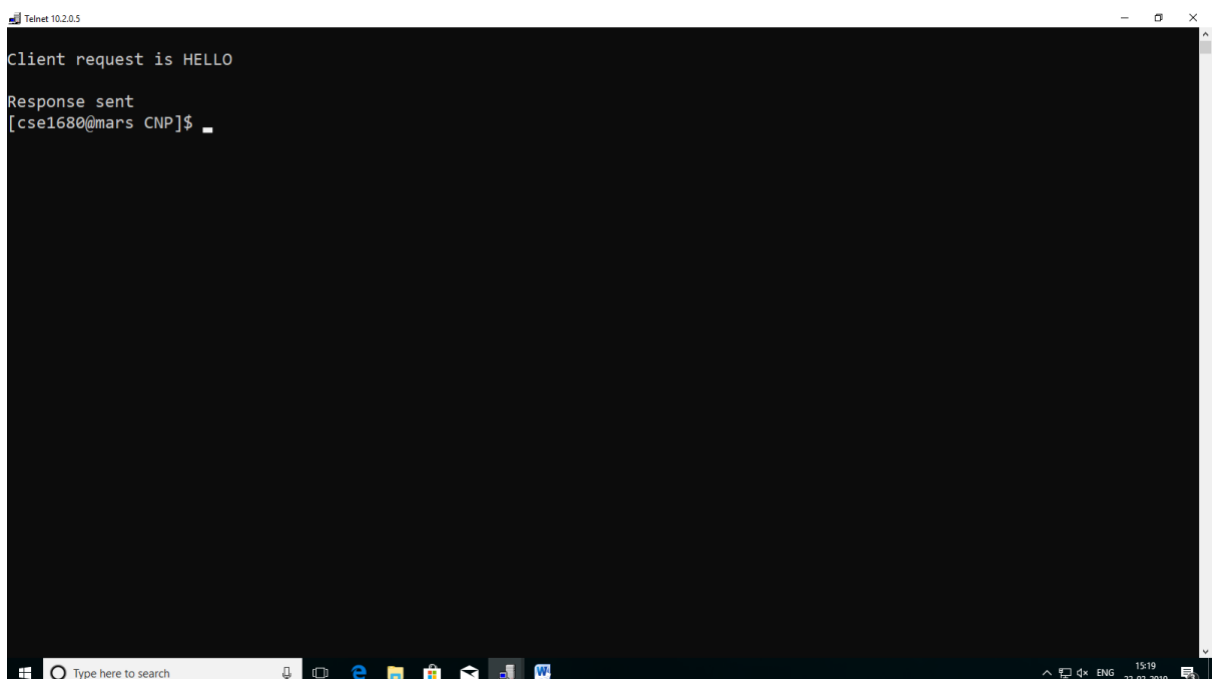
./cout 10.2.0.5 5080 (client terminal)

CLIENT



```
Telnet 10.2.0.5
Enter the request message : HELLO
Message sent successfully
Server response is : Fri Feb 22 16:01:47 HELLO
[cse1680@mars CNP]$
```

SERVER



```
Telnet 10.2.0.5
Client request is HELLO
Response sent
[cse1680@mars CNP]$
```

TCP ECHO SERVICE (ITERATIVE)

Program:

/*CLIENT PROGRAM*/

```
#include<stdio.h>
```

```
#include<stdlib.h>
```

```
#include<sys/types.h>
```

```
#include<sys/socket.h>
```

```
#include<netinet/in.h>
```

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

```
{
```

```
    int sockid,rval;
```

```
    char m1[20],m2[20];
```

```
    sockid=socket(AF_INET,SOCK_STREAM,0);
```

```
    if(sockid==-1)
```

```
    {
```

```
        perror("SOCK-CRE-ERR");
```

```
        exit(1);
```

```
    }
```

```
    struct sockaddr_in s;
```

```
    system("clear");
```

```
    if(argc<3)
```

```
    {
```

```
        printf("\nUSAGE : %s IP_ADDR PORT#\n",argv[0]);
```

```
        exit(0);
```

```
    }
```

```
    s.sin_family=AF_INET;
```

```
    s.sin_port=htons(atoi(argv[2]));
```

```
    s.sin_addr.s_addr=inet_addr(argv[1]);
```

```
    rval=connect(sockid,(struct sockaddr*)&s, sizeof(s));
```

```
    if(rval==-1)
```

```
    {
```

```
        perror("CONN-ERR:");
```

```
        close(sockid);
```

```
        exit(1);
```

```

    }
    printf("\nEnter the request message : ");
    scanf("%s",m1);
    rval=send(sockid,m1,sizeof(m1),0);
    if(rval==-1)
    {
        perror("MSG-SND-ERR:");
        close(sockid);
        exit(1);
    }
    printf("\nMessage sent successfully\n");
    rval=recv(sockid,m2,sizeof(m2),0);
    if(rval==-1)
    {
        perror("MSG-RCV-ERR:");
        close(sockid);
        exit(1);
    }
    printf("\nServer response is : %s\n",m2);
    close(sockid);
}

/*SERVER PROGRAM*/
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>

main(int argc,char*argv[])
{
    int sid,sid1,rval,itr,i;// sid is half association. sid1 is full association
    struct sockaddr_in s,c;
    char buffer[20];
    int clen; //accept() uses value-result parameter
    system("clear");
    if(argc<3)
    {

```

```

        printf("\nUSAGE : %s IP_ADDRESS PORT#\n",argv[0]);
        exit(0);
    }
    printf("\nEnter the number of clients to serve/ server iterations : ");
    scanf("%d",&itr);
    sid=socket(AF_INET,SOCK_STREAM,6);//3rd parameter can also be 0
    if(sid==-1)
    {
        perror("SOCK-CRE-ERR:");
        exit(1);
    }
    /*DEFINING NAME OF THE SERVICE*/
    s.sin_family=AF_INET;
    s.sin_port=htons(atoi(argv[2]));
    s.sin_addr.s_addr=inet_addr(argv[1]);

    /*BIND SOCKET- indicates the process that is listening*/
    rval=bind(sid,(struct sockaddr*)&s,sizeof(s));
    if(rval==-1)
    {
        perror("BIND-ERR:");
        close(sid);
        exit(1);
    }
    rval=listen(sid,5);//range : 1-5
    if(rval==-1)
    {
        perror("LISTEN-ERR:");
        close(sid);
        exit(1);
    }
    for(i=1;i<=itr;i++)
    {
        clen=sizeof(c);
        sid1=accept(sid,(struct sockaddr*)&c,&clen);
        //sid1 is a full association tuple and has information of client,server and
communication protocol i.e serving socket

```

```

    rval=recv(sid1,buffer,sizeof(buffer),0);
    if(rval==-1)
    {
        perror("MSG-RCV-ERR:");
    }
    else
    {
        printf("\nClient request is %s\n",buffer);
    }
    rval=send(sid1,buffer,sizeof(buffer),0);
    if(rval==-1)
    {
        perror("MSG-SND-ERR:");
    }
    else
    {
        printf("\nResponse sent\n");
    }
    close(sid1);//closing the serving socket
}
close(sid);//closing the listening socket
}

```

OUTPUT:

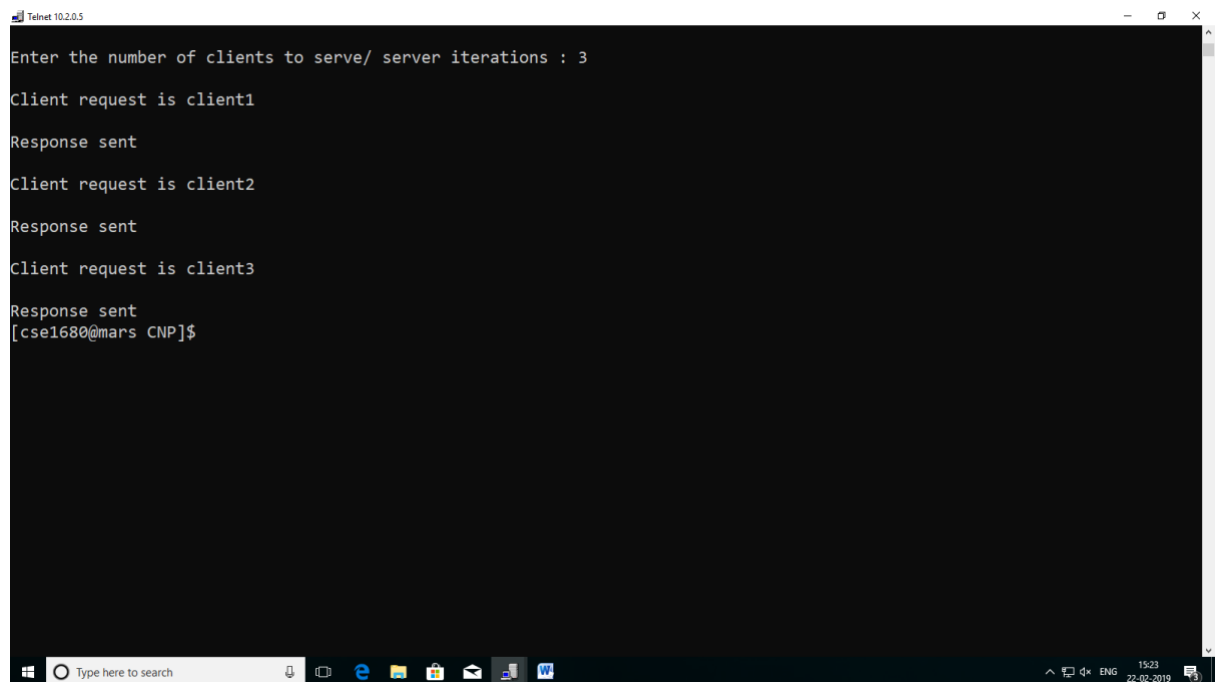
compilation : gcc -o cout tcp_client.c

gcc -o sout tcp_itrsrv_echo.c

execution : ./sout 10.2.0.5 5080 (server terminal)

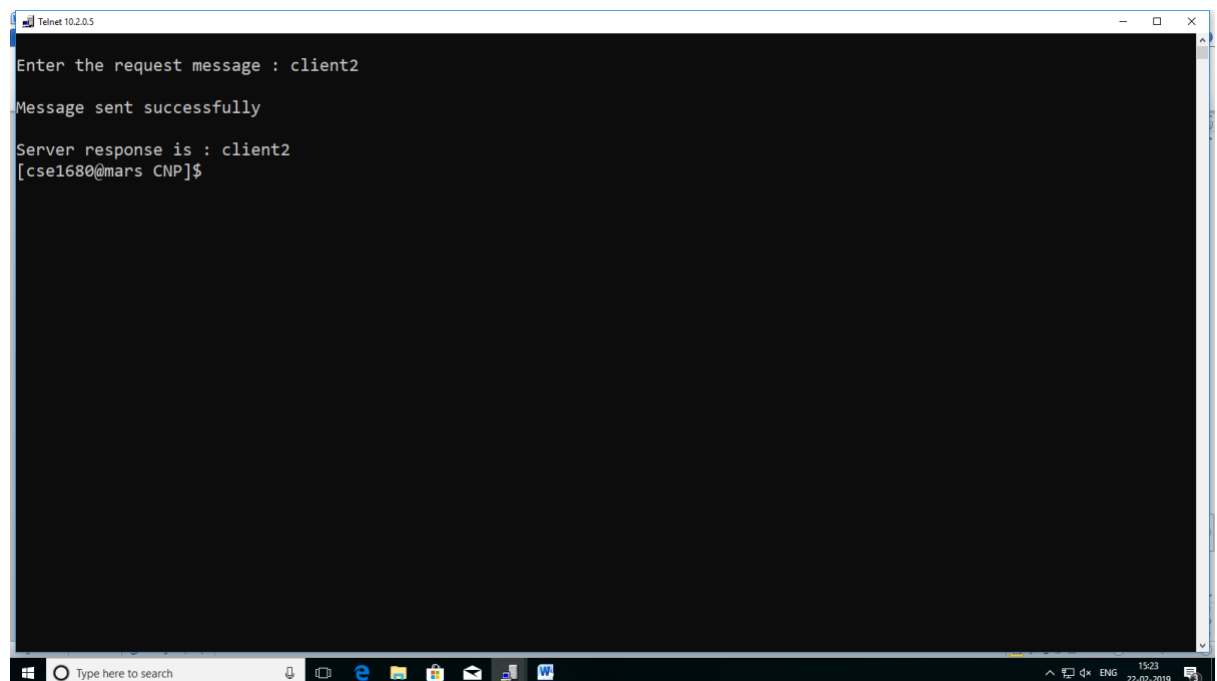
./cout 10.2.0.5 5080 (client terminal) ...(for 3 times)

SERVER



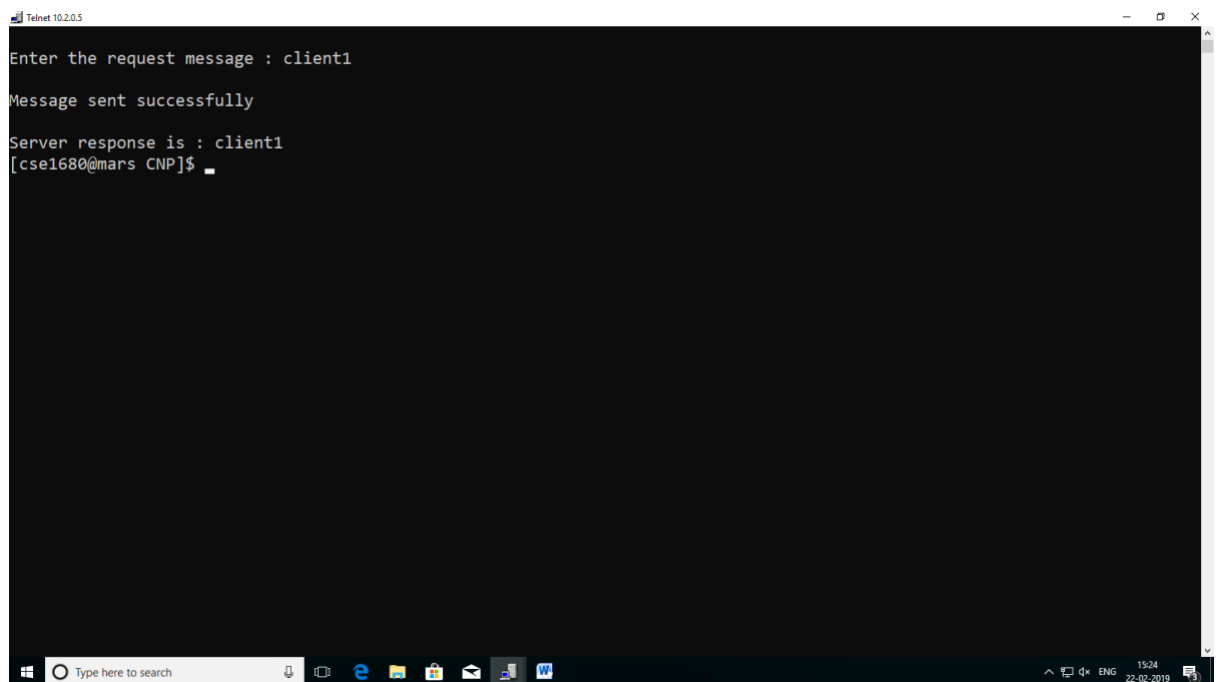
```
Telnet 10.2.0.5
Enter the number of clients to serve/ server iterations : 3
Client request is client1
Response sent
Client request is client2
Response sent
Client request is client3
Response sent
[cse1680@mars CNP]$
```

CLIENT 2



```
Telnet 10.2.0.5
Enter the request message : client2
Message sent successfully
Server response is : client2
[cse1680@mars CNP]$
```

CLIENT 1



```
Telnet 10.2.0.5

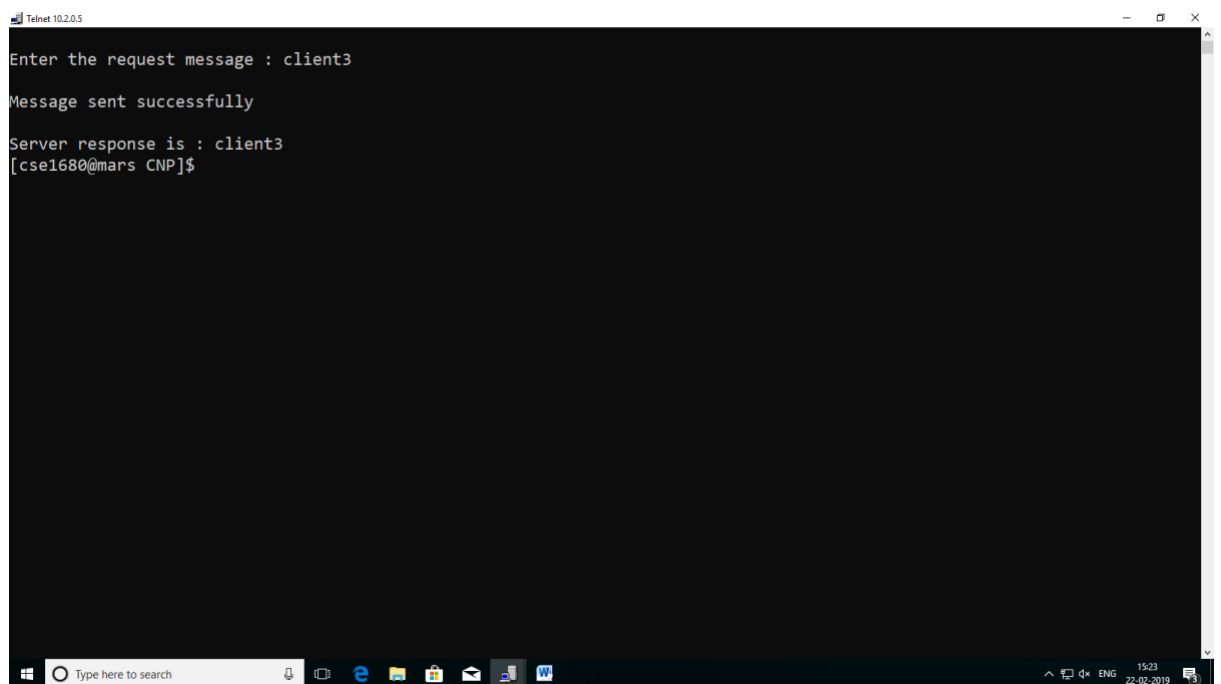
Enter the request message : client1

Message sent successfully

Server response is : client1
[cse1680@mars CNP]$
```

The screenshot shows a Windows desktop with a Telnet window open. The window title is 'Telnet 10.2.0.5'. The text inside the window shows a sequence of commands and responses: 'Enter the request message : client1', 'Message sent successfully', 'Server response is : client1', and the prompt '[cse1680@mars CNP]\$' with a cursor. The Windows taskbar is visible at the bottom, showing the search bar and several application icons. The system tray on the right shows the time as 15:24 and the date as 22-02-2019.

CLIENT 3



```
Telnet 10.2.0.5

Enter the request message : client3

Message sent successfully

Server response is : client3
[cse1680@mars CNP]$
```

This screenshot is similar to the one above, showing a Telnet window with the same title 'Telnet 10.2.0.5'. The text inside shows: 'Enter the request message : client3', 'Message sent successfully', 'Server response is : client3', and the prompt '[cse1680@mars CNP]\$' with a cursor. The Windows taskbar and system tray are also visible, with the system tray showing the time as 15:23 and the date as 22-02-2019.

TCP ECHO SERVICE (CONCURRENT)

Program:

```
/*CLIENT PROGRAM*/
```

```
#include<stdio.h>
```

```
#include<stdlib.h>
```

```
#include<sys/types.h>
```

```
#include<sys/socket.h>
```

```
#include<netinet/in.h>
```

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

```
{
```

```
    int sockid,rval;
```

```
    char m1[20],m2[20];
```

```
    sockid=socket(AF_INET,SOCK_STREAM,0);
```

```
    if(sockid==-1)
```

```
    {
```

```
        perror("SOCK-CRE-ERR");
```

```
        exit(1);
```

```
    }
```

```
    struct sockaddr_in s;
```

```
    system("clear");
```

```
    if(argc<3)
```

```
    {
```

```
        printf("\nUSAGE : %s IP_ADDR PORT#\n",argv[0]);
```

```
        exit(0);
```

```
    }
```

```
    s.sin_family=AF_INET;
```

```
    s.sin_port=htons(atoi(argv[2]));
```

```
    s.sin_addr.s_addr=inet_addr(argv[1]);
```

```
    rval=connect(sockid,(struct sockaddr*)&s, sizeof(s));
```

```
    if(rval==-1)
```

```
    {
```

```
        perror("CONN-ERR:");
```

```
        close(sockid);
```

```
        exit(1);
```

```

    }
    printf("\nEnter the request message : ");
    scanf("%s",m1);
    rval=send(sockid,m1,sizeof(m1),0);
    if(rval==-1)
    {
        perror("MSG-SND-ERR:");
        close(sockid);
        exit(1);
    }
    printf("\nMessage sent successfully\n");
    rval=recv(sockid,m2,sizeof(m2),0);
    if(rval==-1)
    {
        perror("MSG-RCV-ERR:");
        close(sockid);
        exit(1);
    }
    printf("\nServer response is : %s\n",m2);
    close(sockid);
}

/*SERVER PROGRAM*/
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>

main(int argc,char*argv[])
{
    int sid,sid1,rval,itr,i,pid;// sid is half association. sid1 is full association
    struct sockaddr_in s,c;
    char buffer[20];
    int clen; //accept() uses value-result parameter
    system("clear");
    if(argc<3)

```

```

{
    printf("\nUSAGE : %s IP_ADDRESS PORT#\n",argv[0]);
    exit(0);
}
printf("\nEnter the number of clients to serve/ server iterations : ");
scanf("%d",&itr);
sid=socket(AF_INET,SOCK_STREAM,6);//3rd parameter can also be 0
if(sid==-1)
{
    perror("SOCK-CRE-ERR:");
    exit(1);
}
/*DEFINING NAME OF THE SERVICE*/
s.sin_family=AF_INET;
s.sin_port=htons(atoi(argv[2]));
s.sin_addr.s_addr=inet_addr(argv[1]);

/*BIND SOCKET- indicates the process that is listening*/
rval=bind(sid,(struct sockaddr*)&s,sizeof(s));
if(rval==-1)
{
    perror("BIND-ERR:");
    close(sid);
    exit(1);
}
rval=listen(sid,5);//range : 1-5
if(rval==-1)
{
    perror("LISTEN-ERR:");
    close(sid);
    exit(1);
}
for(i=1;i<=itr;i++)
{
    clen=sizeof(c);
    sid1=accept(sid,(struct sockaddr*)&c,&clen);
    if(sid1==-1)

```

```

{
    perror("ACCEPT-ERR:");
    close(sid);
    exit(1);
}
pid=fork();
if(pid==-1)
{
    perror("FRK-ERR:");
    close(sid1);
    close(sid);
    exit(1);
}

```

//sid1 is a full association tuple and has information of client,server and communication protocol i.e serving socket

```

if(pid==0) //CHILD
{
    rval=recv(sid1,buffer,sizeof(buffer),0);
    if(rval==-1)
    {
        perror("MSG-RCV-ERR:");
    }
    else
    {
        printf("\nClient request is %s\n",buffer);
    }
    rval=send(sid1,buffer,sizeof(buffer),0);
    if(rval==-1)
    {
        perror("MSG-SND-ERR:");
    }
    else
    {
        printf("\nResponse sent\n");
    }
}

```

```
        close(sid1);//closing the serving socket
        exit(0);
    }
    else //PARENT
        close(sid1);//parent also has a copy of the serving socket. So close it
here.
    }
    close(sid);//closing the listening socket
    exit(0);
}
```

OUTPUT:

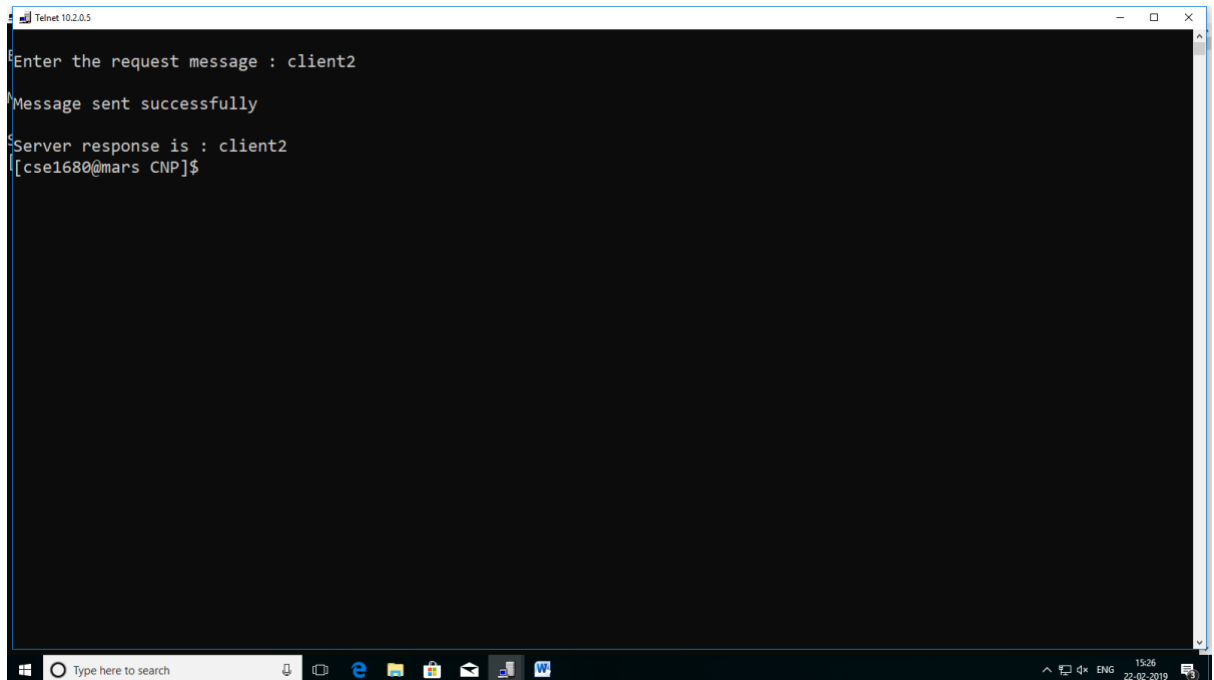
compilation : gcc -o cout tcp_client.c

gcc -o sout tcp_con_ES.c

execution : ./sout 10.2.0.5 5080 (server terminal)

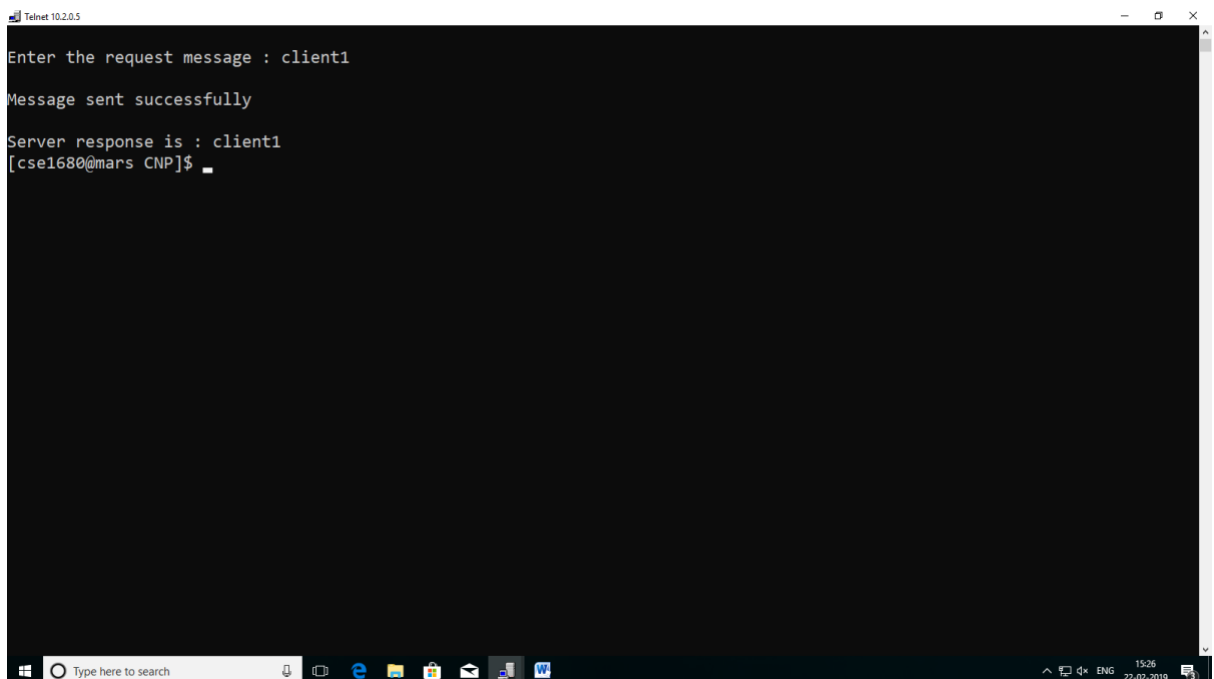
./cout 10.2.0.5 5080 (client terminal) ...(for 3 times)

CLIENT 2



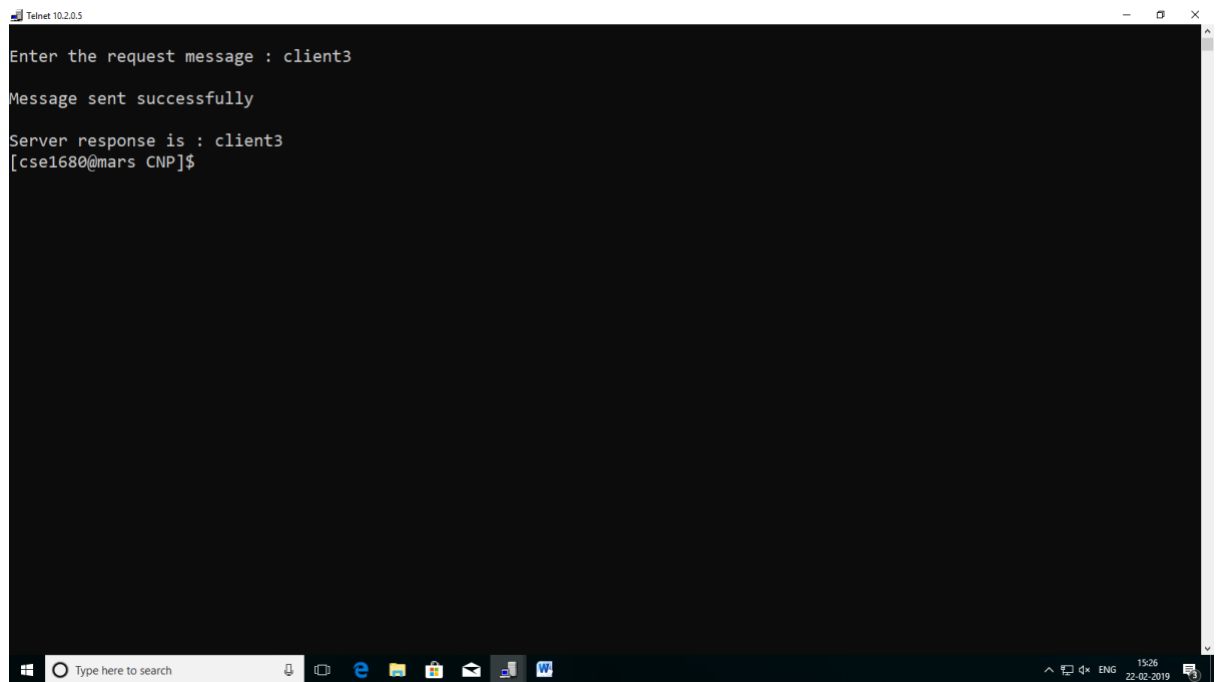
```
Telnet 10.2.0.5
Enter the request message : client2
Message sent successfully
Server response is : client2
[cse1680@mars CNP]$
```

CLIENT 1



```
Telnet 10.2.0.5
Enter the request message : client1
Message sent successfully
Server response is : client1
[cse1680@mars CNP]$
```

CLIENT 3



A screenshot of a Telnet 10.2.0.5 window. The window has a title bar with the text 'Telnet 10.2.0.5' and standard window controls. The main area is black with white text. The text shows the user entering 'client3' as a request message, receiving a 'Message sent successfully' confirmation, and then seeing the 'Server response is : client3'. The prompt '[cse1680@mars CNP]\$' is visible at the bottom of the text area. The Windows taskbar is visible at the bottom of the screen, showing the search bar and several application icons.

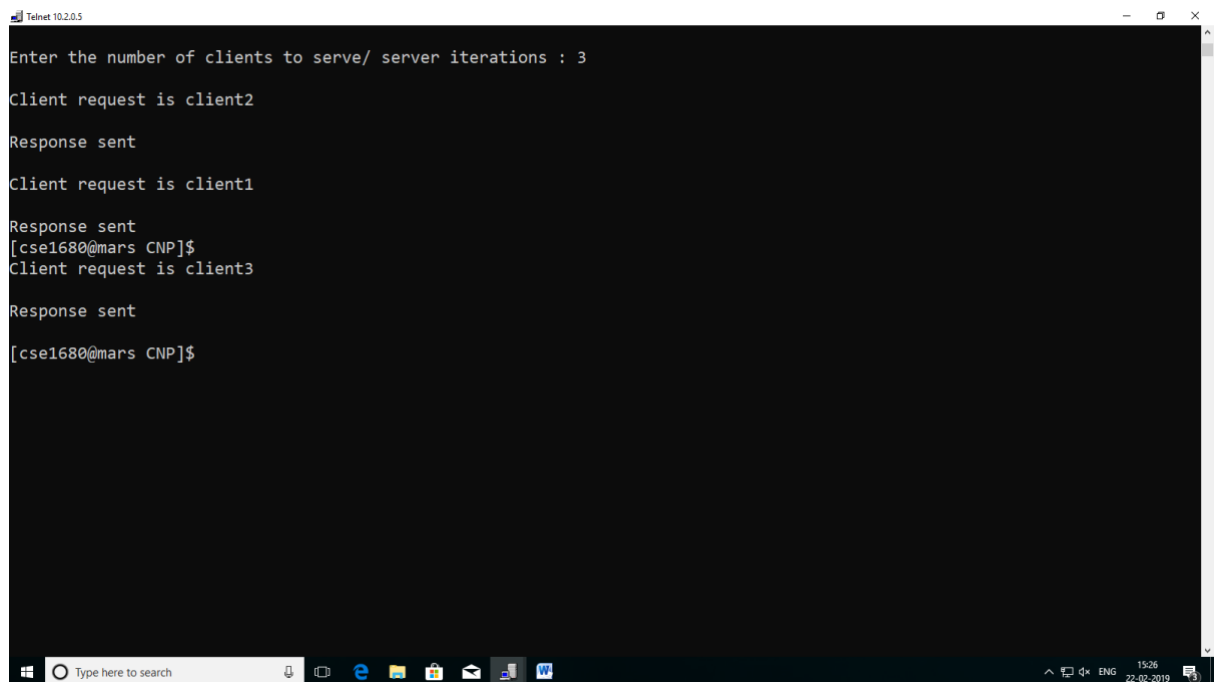
```
Telnet 10.2.0.5

Enter the request message : client3

Message sent successfully

Server response is : client3
[cse1680@mars CNP]$
```

SERVER



A screenshot of a Telnet 10.2.0.5 window. The window has a title bar with the text 'Telnet 10.2.0.5' and standard window controls. The main area is black with white text. The text shows the user entering '3' as the number of clients to serve. The server then processes three requests: 'client2', 'client1', and 'client3', each followed by a 'Response sent' message. The prompt '[cse1680@mars CNP]\$' is visible at the bottom of the text area. The Windows taskbar is visible at the bottom of the screen, showing the search bar and several application icons.

```
Telnet 10.2.0.5

Enter the number of clients to serve/ server iterations : 3

Client request is client2

Response sent

Client request is client1

Response sent
[cse1680@mars CNP]$
Client request is client3

Response sent

[cse1680@mars CNP]$
```

TCP DAYTIME SERVICE (ITERATIVE)

Program:

```
/*CLIENT PROGRAM*/
```

```
#include<stdio.h>
```

```
#include<stdlib.h>
```

```
#include<sys/types.h>
```

```
#include<sys/socket.h>
```

```
#include<netinet/in.h>
```

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

```
{
```

```
    int sockid,rval;
```

```
    char m1[20],m2[20];
```

```
    sockid=socket(AF_INET,SOCK_STREAM,0);
```

```
    if(sockid==-1)
```

```
    {
```

```
        perror("SOCK-CRE-ERR");
```

```
        exit(1);
```

```
    }
```

```
    struct sockaddr_in s;
```

```
    system("clear");
```

```
    if(argc<3)
```

```
    {
```

```
        printf("\nUSAGE : %s IP_ADDR PORT#\n",argv[0]);
```

```
        exit(0);
```

```
    }
```

```
    s.sin_family=AF_INET;
```

```
    s.sin_port=htons(atoi(argv[2]));
```

```
    s.sin_addr.s_addr=inet_addr(argv[1]);
```

```
    rval=connect(sockid,(struct sockaddr*)&s, sizeof(s));
```

```
    if(rval==-1)
```

```
    {
```

```
        perror("CONN-ERR:");
```

```
        close(sockid);
```

```
        exit(1);
```



```

    }
    printf("\nEnter the request message : ");
    scanf("%s",m1);
    rval=send(sockid,m1,sizeof(m1),0);
    if(rval==-1)
    {
        perror("MSG-SND-ERR:");
        close(sockid);
        exit(1);
    }
    printf("\nMessage sent successfully\n");
    rval=recv(sockid,m2,sizeof(m2),0);
    if(rval==-1)
    {
        perror("MSG-RCV-ERR:");
        close(sockid);
        exit(1);
    }
    printf("\nServer response is : %s\n",m2);
    close(sockid);
}

/*SERVER PROGRAM*/
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>

main(int argc,char*argv[])
{
    int sid,sid1,rval,itr,i;// sid is half association. sid1 is full association
    struct sockaddr_in s,c;
    char buffer[20],smsg[30];
    time_t t;
    int clen; //accept() uses value-result parameter
    system("clear");
    if(argc<3)

```

```

{
    printf("\nUSAGE : %s IP_ADDRESS PORT#\n",argv[0]);
    exit(0);
}
printf("\nEnter the number of clients to serve/ server iterations : ");
scanf("%d",&itr);
sid=socket(AF_INET,SOCK_STREAM,6);//3rd parameter can also be 0
if(sid==-1)
{
    perror("SOCK-CRE-ERR:");
    exit(1);
}
/*DEFINING NAME OF THE SERVICE*/
s.sin_family=AF_INET;
s.sin_port=htons(atoi(argv[2]));
s.sin_addr.s_addr=inet_addr(argv[1]);

/*BIND SOCKET- indicates the process that is listening*/
rval=bind(sid,(struct sockaddr*)&s,sizeof(s));
if(rval==-1)
{
    perror("BIND-ERR:");
    close(sid);
    exit(1);
}
rval=listen(sid,5);//range : 1-5
if(rval==-1)
{
    perror("LISTEN-ERR:");
    close(sid);
    exit(1);
}
for(i=1;i<=itr;i++)
{
    clen=sizeof(c);
    sid1=accept(sid,(struct sockaddr*)&c,&clen);

```

//sid1 is a full association tuple and has information of client,server and communication protocol i.e serving socket

```
    rval=recv(sid1,buffer,sizeof(buffer),0);
    if(rval==-1)
    {
        perror("MSG-RCV-ERR:");
    }
    else
    {
        printf("\nClient request is %s\n",buffer);
    }
    t=time(0);
    strcpy(smsg,ctime(&t));
    rval=send(sid1,smsg,sizeof(smsg),0);
    if(rval==-1)
    {
        perror("MSG-SND-ERR:");
    }
    else
    {
        printf("\nResponse sent\n");
    }
    close(sid1);//closing the serving socket
}
close(sid);//closing the listening socket
}
```

OUTPUT:

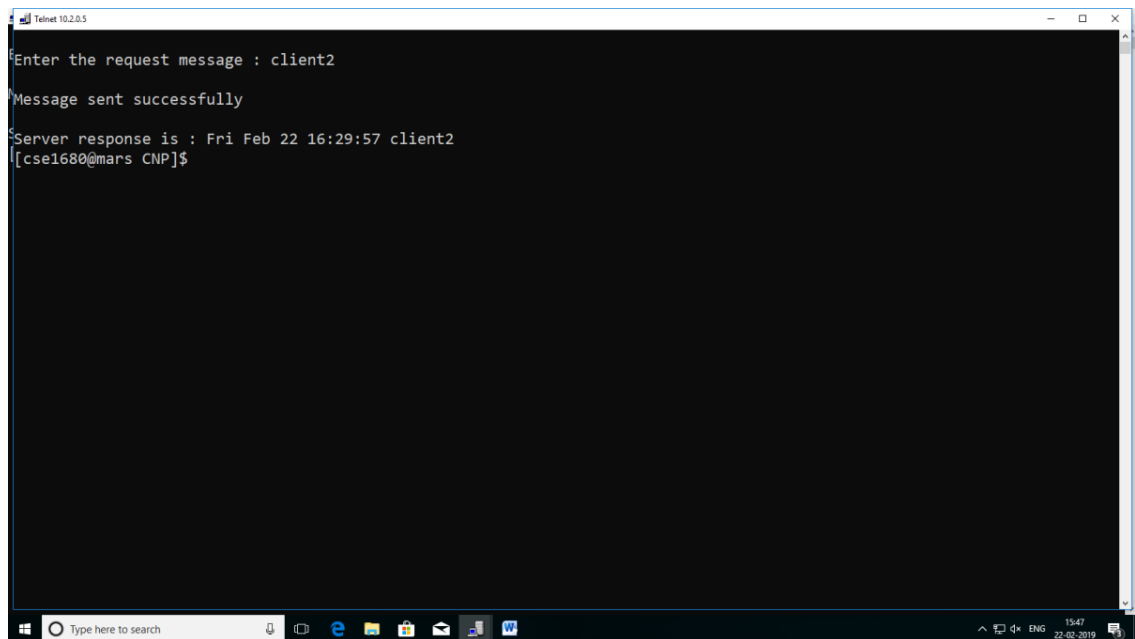
compilation : gcc -o cout tcp_client.c

gcc -o sout tcp_itrsrv_dts.c

execution : ./sout 10.2.0.5 5080 (server terminal)

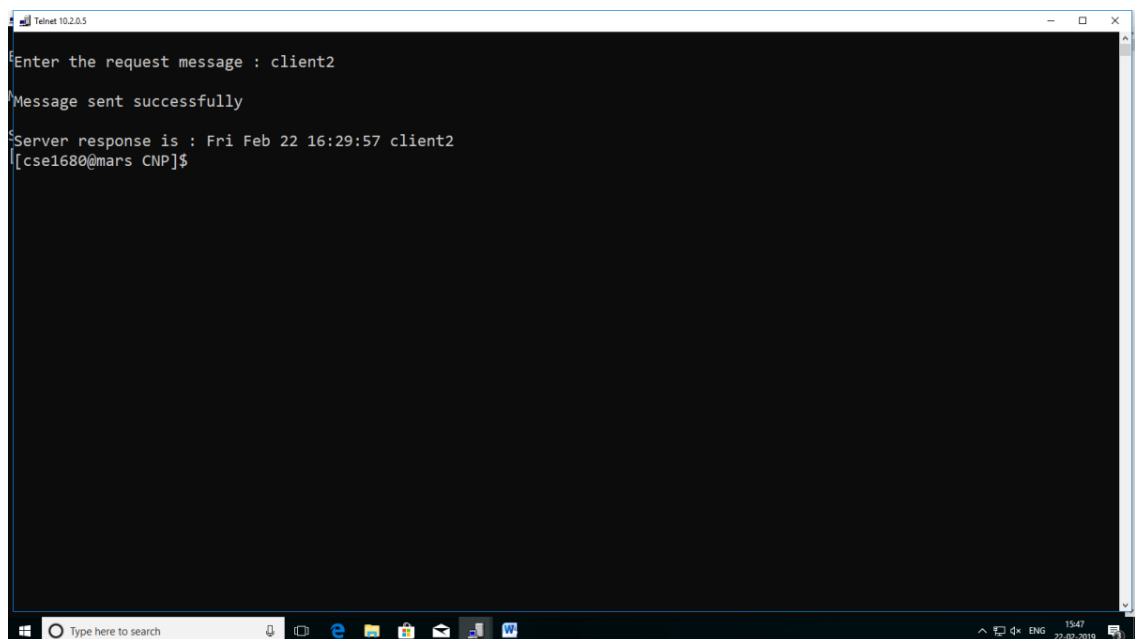
./cout 10.2.0.5 5080 (client terminal) ...(for 3 times)

CLIENT 1



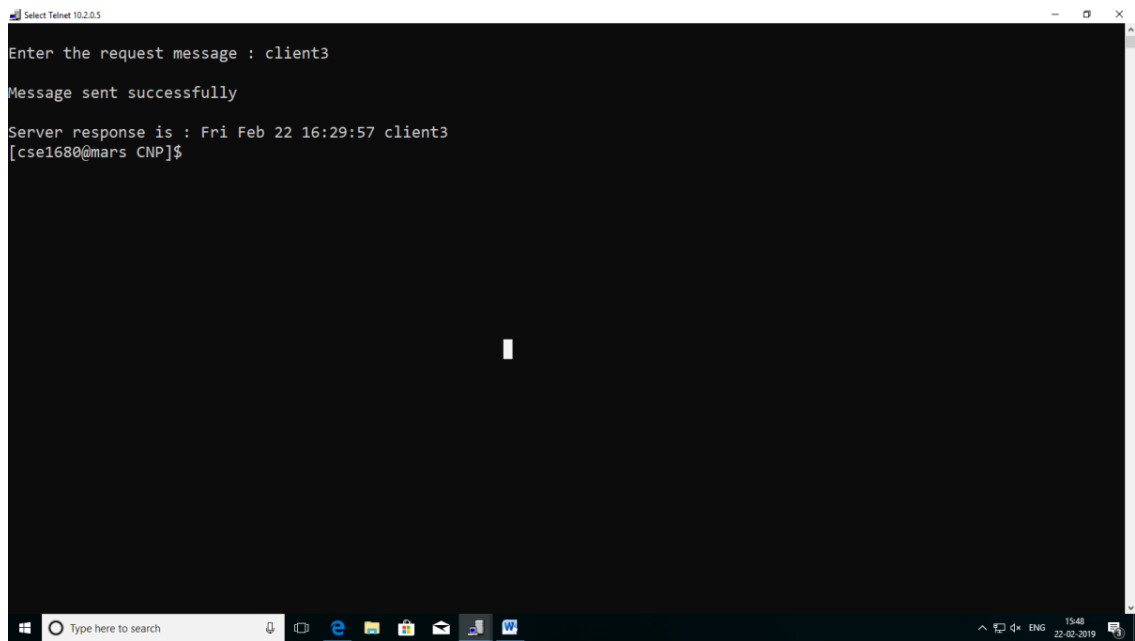
```
Telnet 10.2.0.5
Enter the request message : client2
Message sent successfully
Server response is : Fri Feb 22 16:29:57 client2
[cse1680@mars CNP]$
```

CLIENT 2



```
Telnet 10.2.0.5
Enter the request message : client2
Message sent successfully
Server response is : Fri Feb 22 16:29:57 client2
[cse1680@mars CNP]$
```

CLIENT 3



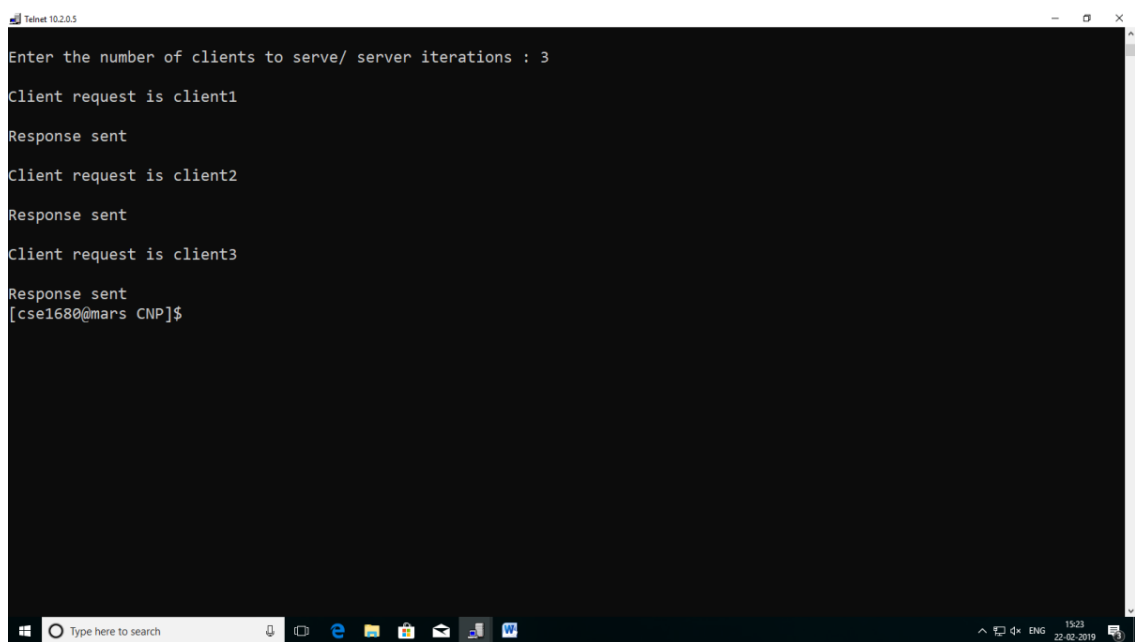
```
Select Telnet 10.2.0.5

Enter the request message : client3
Message sent successfully

Server response is : Fri Feb 22 16:29:57 client3
[cse1680@mars CNP]$
```

The screenshot shows a Windows desktop with a Telnet window titled "Select Telnet 10.2.0.5". The window has a black background with white text. The user has entered "client3" as the request message, and the system has responded with "Message sent successfully". Below that, the server response is displayed as "Fri Feb 22 16:29:57 client3". The prompt "[cse1680@mars CNP]\$" is visible at the bottom of the window. The Windows taskbar is visible at the bottom of the screen.

SERVER



```
Telnet 10.2.0.5

Enter the number of clients to serve/ server iterations : 3
Client request is client1
Response sent
Client request is client2
Response sent
Client request is client3
Response sent
[cse1680@mars CNP]$
```

The screenshot shows a Windows desktop with a Telnet window titled "Telnet 10.2.0.5". The window has a black background with white text. The user has entered "3" as the number of clients to serve. The server then processes three client requests: "client1", "client2", and "client3", each followed by a "Response sent" message. The prompt "[cse1680@mars CNP]\$" is visible at the bottom of the window. The Windows taskbar is visible at the bottom of the screen.

TCP DAYTIME SERVICE (CONCURRENT)

Program:

```
/*CLIENT PROGRAM*/
```

```
#include<stdio.h>
```

```
#include<stdlib.h>
```

```
#include<sys/types.h>
```

```
#include<sys/socket.h>
```

```
#include<netinet/in.h>
```

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

```
{
```

```
    int sockid,rval;
```

```
    char m1[20],m2[20];
```

```
    sockid=socket(AF_INET,SOCK_STREAM,0);
```

```
    if(sockid==-1)
```

```
    {
```

```
        perror("SOCK-CRE-ERR");
```

```
        exit(1);
```

```
    }
```

```
    struct sockaddr_in s;
```

```
    system("clear");
```

```
    if(argc<3)
```

```
    {
```

```
        printf("\nUSAGE : %s IP_ADDR PORT#\n",argv[0]);
```

```
        exit(0);
```

```
    }
```

```
    s.sin_family=AF_INET;
```

```
    s.sin_port=htons(atoi(argv[2]));
```

```
    s.sin_addr.s_addr=inet_addr(argv[1]);
```

```
    rval=connect(sockid,(struct sockaddr*)&s, sizeof(s));
```

```
    if(rval==-1)
```

```
    {
```

```
        perror("CONN-ERR:");
```

```
        close(sockid);
```

```
        exit(1);
```

```

    }
    printf("\nEnter the request message : ");
    scanf("%s",m1);
    rval=send(sockid,m1,sizeof(m1),0);
    if(rval==-1)
    {
        perror("MSG-SND-ERR:");
        close(sockid);
        exit(1);
    }
    printf("\nMessage sent successfully\n");
    rval=recv(sockid,m2,sizeof(m2),0);
    if(rval==-1)
    {
        perror("MSG-RCV-ERR:");
        close(sockid);
        exit(1);
    }
    printf("\nServer response is : %s\n",m2);
    close(sockid);
}

/*SERVER PROGRAM*/

#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<string.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<time.h>
main(int argc,char*argv[])
{
    int sid,sid1,rval,itr,i,pid;// sid is half association. sid1 is full association
    struct sockaddr_in s,c;
    char buffer[20],smsg[30];
    time_t t=time(0);

```

```

int clen; //accept() uses value-result parameter
system("clear");
if(argc<3)
{
    printf("\nUSAGE : %s IP_ADDRESS PORT#\n",argv[0]);
    exit(0);
}
printf("\nEnter the number of clients to serve/ server iterations : ");
scanf("%d",&itr);
sid=socket(AF_INET,SOCK_STREAM,6);//3rd parameter can also be 0
if(sid==-1)
{
    perror("SOCK-CRE-ERR:");
    exit(1);
}
/*DEFINING NAME OF THE SERVICE*/
s.sin_family=AF_INET;
s.sin_port=htons(atoi(argv[2]));
s.sin_addr.s_addr=inet_addr(argv[1]);

/*BIND SOCKET- indicates the process that is listening*/
rval=bind(sid,(struct sockaddr*)&s,sizeof(s));
if(rval==-1)
{
    perror("BIND-ERR:");
    close(sid);
    exit(1);
}
rval=listen(sid,5);//range : 1-5
if(rval==-1)
{
    perror("LISTEN-ERR:");
    close(sid);
    exit(1);
}
for(i=1;i<=itr;i++)
{

```



```

clen=sizeof(c);
sid1=accept(sid,(struct sockaddr*)&c,&clen);
if(sid1==-1)
{
    perror("ACCEPT-ERR:");
    close(sid);
    exit(1);
}
pid=fork();
if(pid==-1)
{
    perror("FRK-ERR:");
    close(sid1);
    close(sid);
    exit(1);
}

```

//sid1 is a full association tuple and has information of client,server and communication protocol i.e serving socket

```

if(pid==0) //CHILD
{
    rval=recv(sid1,buffer,sizeof(buffer),0);
    if(rval==-1)
    {
        perror("MSG-RCV-ERR:");
    }
    else
    {
        printf("\nClient request is %s\n",buffer);
    }
    strcpy(smsg,ctime(&t));
    rval=send(sid1,smsg,sizeof(smsg),0);
    if(rval==-1)
    {
        perror("MSG-SND-ERR:");
    }
}

```

```

        else
        {
            printf("\nResponse sent\n");
        }
        close(sid1);//closing the serving socket
        exit(0);
    }
    else //PARENT
        close(sid1);//parent also has a copy of the serving socket. So close it
here.
    }
    close(sid);//closing the listening socket
    exit(0);
}

```

OUTPUT:

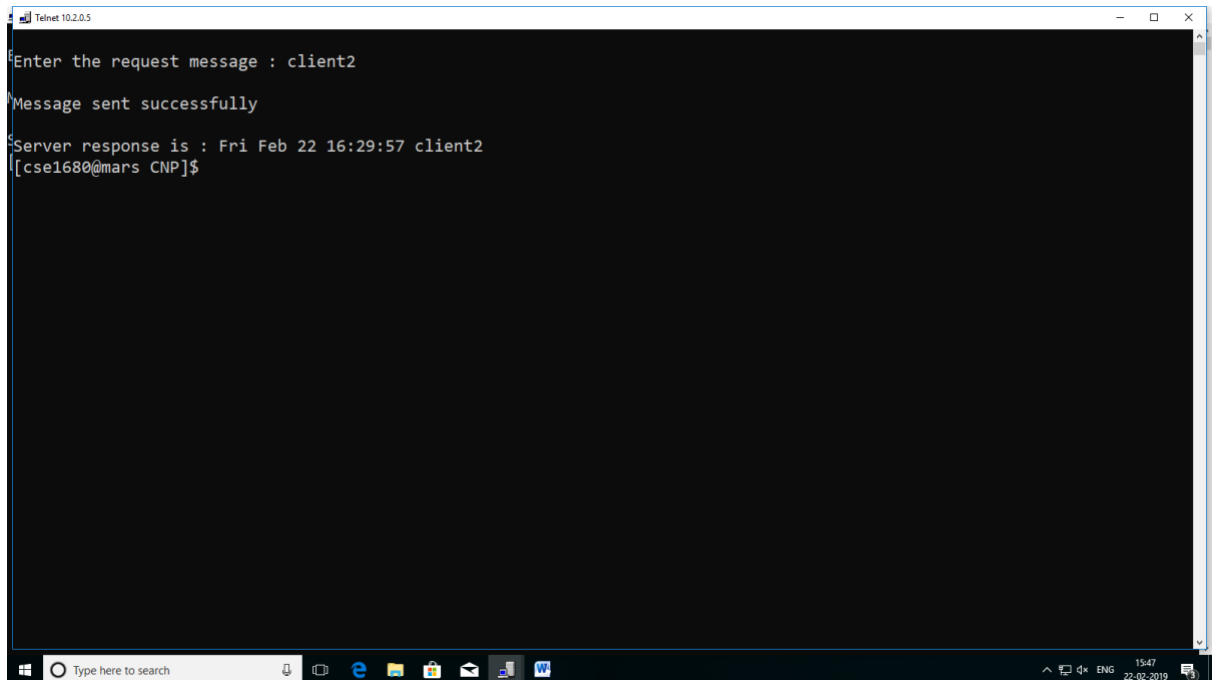
compilation : gcc -o cout tcp_client.c

gcc -o sout tcp_con_ES_dts.c

execution : ./sout 10.2.0.5 5080 (server terminal)

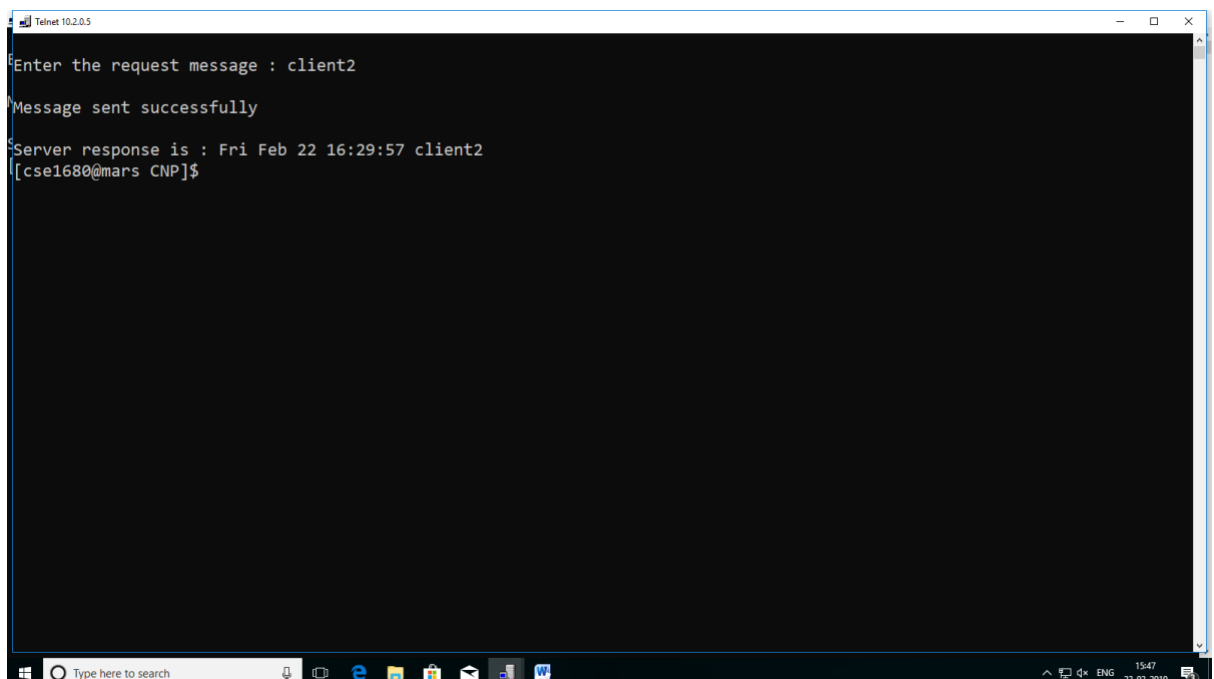
./cout 10.2.0.5 5080 (client terminal) ...(for 3 times)

CLIENT 2



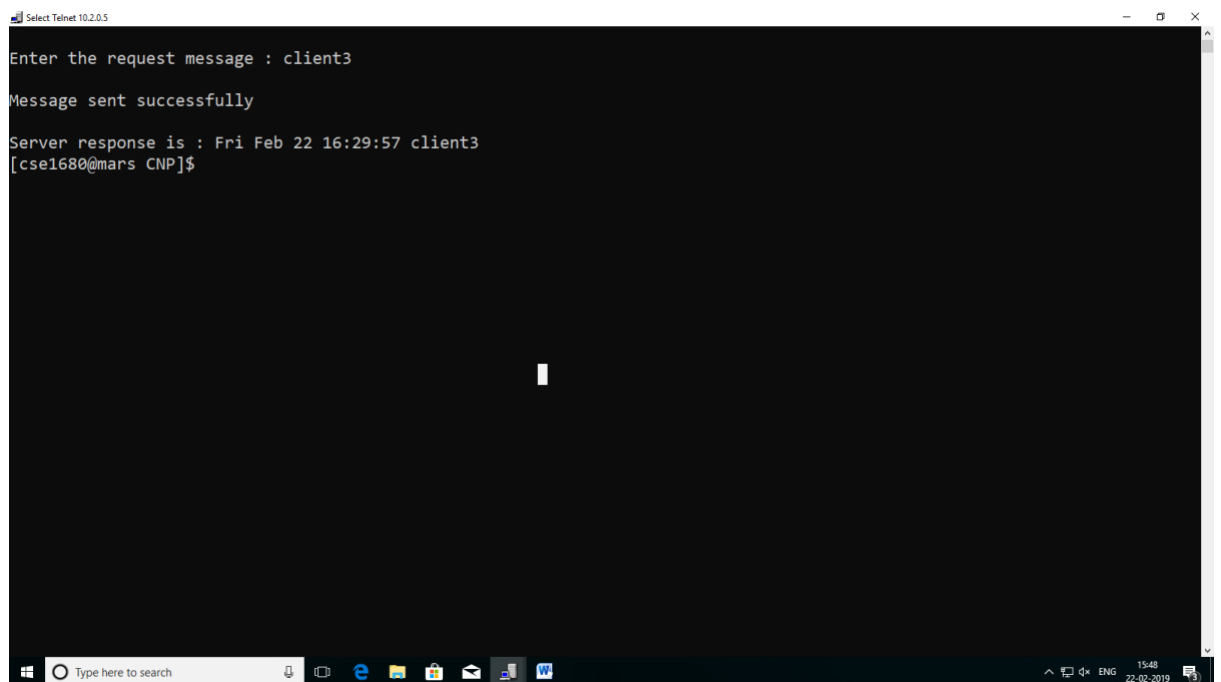
```
Telnet 10.2.0.5
Enter the request message : client2
Message sent successfully
Server response is : Fri Feb 22 16:29:57 client2
[cse1680@mars CNP]$
```

CLIENT 1



```
Telnet 10.2.0.5
Enter the request message : client2
Message sent successfully
Server response is : Fri Feb 22 16:29:57 client2
[cse1680@mars CNP]$
```

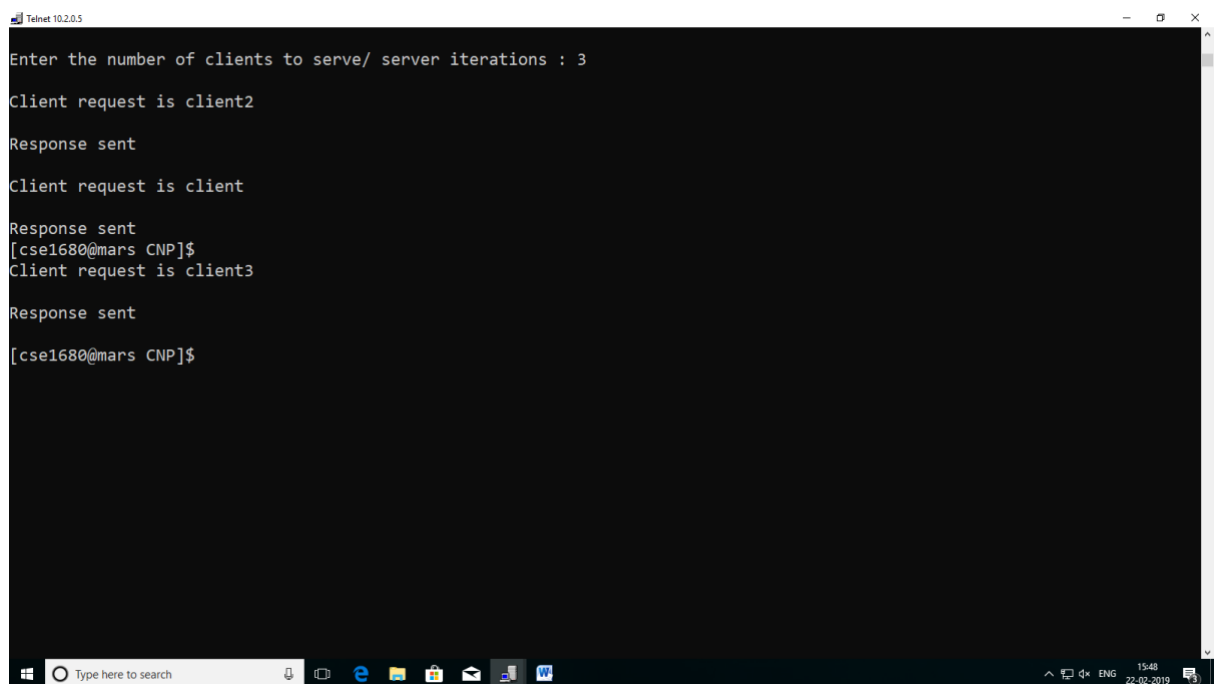
CLIENT 3



The screenshot shows a Telnet 10.2.0.5 window with a black background and white text. The text displays the process of sending a request to 'client3' and receiving a response. The prompt '[cse1680@mars CNP]\$' is visible at the bottom of the command area. The Windows taskbar at the bottom shows the search bar and several application icons.

```
Select Telnet 10.2.0.5
Enter the request message : client3
Message sent successfully
Server response is : Fri Feb 22 16:29:57 client3
[cse1680@mars CNP]$
```

SERVER



The screenshot shows a Telnet 10.2.0.5 window with a black background and white text. It displays a sequence of client requests and server responses. The requests are for 'client2' and 'client3'. The prompt '[cse1680@mars CNP]\$' is visible at the bottom of the command area. The Windows taskbar at the bottom shows the search bar and several application icons.

```
Telnet 10.2.0.5
Enter the number of clients to serve/ server iterations : 3
Client request is client2
Response sent
Client request is client
Response sent
[cse1680@mars CNP]$
Client request is client3
Response sent
[cse1680@mars CNP]$
```

BASIC OPERATIONS ON SEMAPHORES

Program:

```
#include<stdio.h>

#include<stdlib.h>

#include<sys/types.h>

#include<semaphore.h>

#include<sys/shm.h>

#include<sys/ipc.h>


main(int argc,char* argv[])

{

    int shmid,rval,sval,csval;

    sem_t *sem_phore;

    system("clear");

    if(argc<3)

    {

        printf("\nUSAGE : %s labelForSHM ByteSize\n",argv[0]);

        exit(0);

    }

    shmid=shmget((key_t)atoi(argv[1]),atoi(argv[2]),IPC_CREAT|0666);

    if(shmid==-1)

    {

        perror("SHM-CRE-ERR:");

        exit(1);

    }

    sem_phore=(sem_t*)shmat(shmid,0,0);

    if(!sem_phore)

    {
```

```

        perror("SHM-ATT-ERR:");

        shmctl(shmid,IPC_RMID,0);

        exit(1);
    }

    printf("\nEnter the initial value for the semaphore: ");
    scanf("%d",&sval);

    /*INITIALIZE VALUE OF SEMAPHORE*/

    rval=sem_init(sem_phore,1,sval);
    if(rval==-1)
    {
        perror("Unable to initialize semaphore:");
        shmdt(sem_phore);
        shmctl(shmid,IPC_RMID,0);
        exit(1);
    }

    rval=sem_getvalue(sem_phore,&csval);//sem_phore gets the value in csval
    if(rval==-1)
    {
        perror("Unable to get value of the semaphore: ");
        shmdt(sem_phore);
        shmctl(shmid,IPC_RMID,0);
        exit(1);
    }

    printf("\nInitialized value of semaphore is %d\n",csval);

    //EXECUTE WAIT OPERATION

    rval=sem_wait(sem_phore);
    if(rval==-1)
    {

```

```

        perror("WAIT-FAILURE:");

        shmdt(sem_phore);

        shmctl(shmid,IPC_RMID,0);

        exit(1);
    }

    rval=sem_getvalue(sem_phore,&csval);
    if(rval==-1)
    {
        perror("Unable to get semaphore value:");

        shmdt(sem_phore);

        shmctl(shmid,IPC_RMID,0);

        exit(1);
    }

    printf("\nSemaphore value after wait is %d\n",csval);

    //EXECUTE SIGNAL OPERATION

    rval=sem_post(sem_phore);
    if(rval==-1)
    {
        perror("Unable to get semaphore value:");

        shmdt(sem_phore);

        shmctl(shmid,IPC_RMID,0);

        exit(1);
    }

    rval=sem_getvalue(sem_phore,&csval);
    if(rval==-1)
    {
        perror("Unable to get semaphore value: ");

        shmdt(sem_phore);

```

```

        shmctl(shmid,IPC_RMID,0);

        exit(1);
    }

    printf("\nSemaphore value after signal operation is %d\n",csval);
    rval=sem_destroy(sem_phore);
    if(rval== -1)
    {
        perror("SEM-DESTROY-ERR:");
        shmdt(sem_phore);
        shmctl(shmid,IPC_RMID,0);
        exit(1);
    }
    rval=shmdt(sem_phore);
    if(rval== -1)
    {
        perror("SHM-DETACH-ERR:");
        shmdt(sem_phore);
        shmctl(shmid,IPC_RMID,0);
        exit(1);
    }
    rval=shmctl(shmid,IPC_RMID,0);
    if(rval== -1)
    {
        perror("SHM-REM-ERR:");
        shmdt(sem_phore);
        shmctl(shmid,IPC_RMID,0);
        exit(1);
    }
}

```

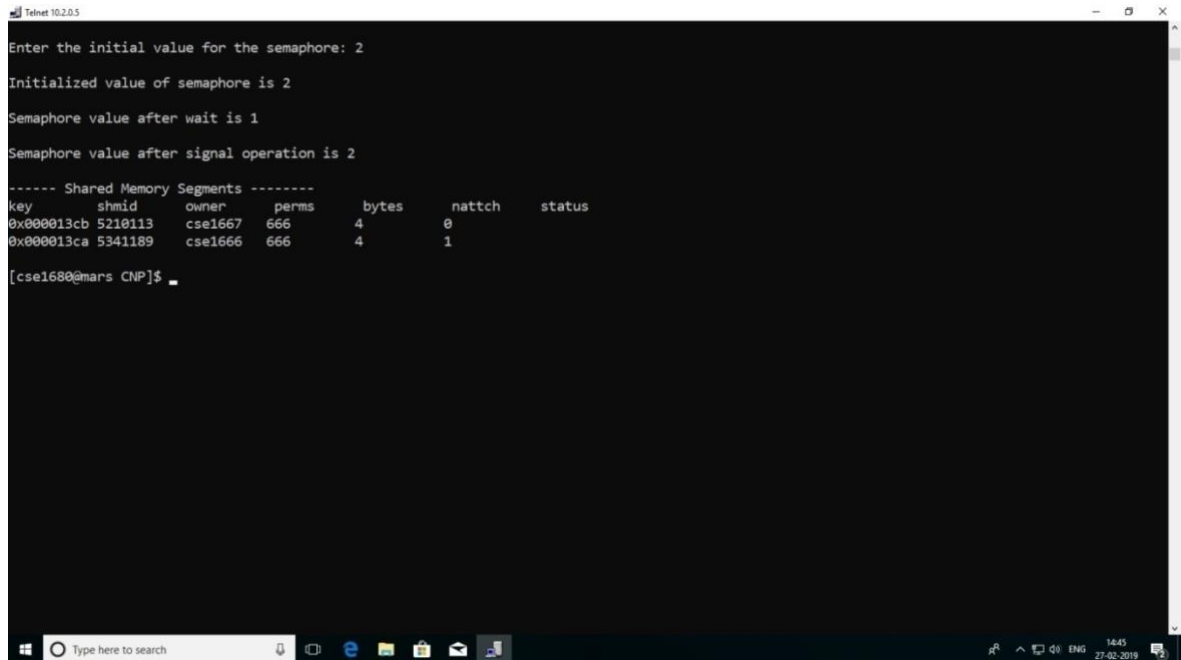


```
    system("ipcs -m");  
}
```

OUTPUT:

compilation : gcc -lpthread operations_Sem.c

execution : ./a.out



```
Telnet 10.20.5

Enter the initial value for the semaphore: 2
Initialized value of semaphore is 2
Semaphore value after wait is 1
Semaphore value after signal operation is 2

----- Shared Memory Segments -----
key      shmid  owner    perms   bytes   nattch   status
0x000013cb 5210113  cse1667  666     4       0
0x000013ca 5341189  cse1666  666     4       1

[cse1680@mars CNP]$
```

The screenshot shows a Telnet session with a black background and white text. The user enters '2' for the semaphore's initial value. The program outputs the initialized value (2), the value after a wait operation (1), and the value after a signal operation (2). It then displays a table of shared memory segments. The table has columns for key, shmid, owner, perms, bytes, nattch, and status. Two segments are listed: one with key 0x000013cb and another with key 0x000013ca. The terminal window has a title bar 'Telnet 10.20.5' and a Windows taskbar at the bottom.

IMPLEMENTING IPC USING MESSAGE QUEUES AND SEMAPHORES

Program:

```
#include<stdio.h>
#include<sys/types.h>
#include<semaphore.h>
#include<unistd.h>
#include<stdlib.h>
#include<sys/shm.h>
#include<sys/ipc.h>
#include<sys/msg.h>

typedef struct msg
{
    long id;
    char txt[10];
}message;

main(int argc, char* argv[])
{
    message m1,m2;
    int label, mem_size;
    int shmid,sval,inval,rval,mqid,pid;
    int msgid; //ID for data
    sem_t *s;
    system("clear");

    if(argc<3)
    {
        printf("\nUSAGE : %s LabelForResources MemorySize(in
bytes)\n",argv[0]);
        exit(1);
    }
    label=atoi(argv[1]);
    mqid=msgget((key_t)label,IPC_CREAT|0666);
    if(mqid==-1)
```

```

{
    perror("MSG-Q-CRE-ERR:");
    exit(1);
}
system("ipcs -q");
mem_size=atoi(argv[2]); //4 bytes (sizeof(int))
//mem_size=4*atoi(argv[2])
shmid=shmget((key_t)label,mem_size,IPC_CREAT|0666);
if(shmid==-1)
{
    perror("SHM-MEM-CRE-ERR:");
    msgctl(mqid,IPC_RMID,0);
    exit(1);
}
system("ipcs -m");
s=(sem_t*)shmat(shmid,0,0);
if(s==NULL)
{
    perror("SHM-AT-ERR:");
    shmctl(shmid,IPC_RMID,0);
    shmdt(s);
    msgctl(mqid,IPC_RMID,0);
    exit(1);
}
printf("\nEnter the initialization value for the semaphore :");
scanf("%d",&inval);
rval=sem_init(s,1,inval);
if(rval==-1)
{
    perror("Unable to initialize semaphore value:");
    msgctl(mqid,IPC_RMID,0);
    sem_destroy(s);
    shmdt(s);
    shmctl(shmid,IPC_RMID,0);
    exit(1);
}
rval=sem_getvalue(s,&sval);

```

```

if(rval==-1)
{
    perror("Unable to get semaphore value:");
    msgctl(mqid,IPC_RMID,0);
    sem_destroy(s);
    shmdt(s);
    shmctl(shmid,IPC_RMID,0);
    exit(1);
}
printf("\nSemaphore value after initialization is %d\n",sval);
printf("\nEnter the message ID to be used between parent and child: ");
scanf("%d",&msgid);

pid=fork();
if(pid==-1)
{
    perror("FRK-ERR:");
    msgctl(mqid,IPC_RMID,0);
    sem_destroy(s);
    shmdt(s);
    shmctl(shmid,IPC_RMID,0);
    exit(1);
}
if(pid==0)
{
    printf("\nCHILD : pid = %d\n",getpid());
    rval=sem_wait(s);
    if(rval==-1)
    {
        perror("SEM-WAIT-ERR:");
        msgctl(mqid,IPC_RMID,0);
        sem_destroy(s);
        shmdt(s);
        shmctl(shmid,IPC_RMID,0);
        exit(1);
    }
    printf("\nCHILD : Enter the message to be sent to parent : ");

```

```

scanf("%s",m1.txt);
m1.id=msgid;
rval=msgsnd(mqid,(message*)&m1,sizeof(m1),0);
if(rval==-1)
{
    perror("MSG-SND-ERR:");
    msgctl(mqid,IPC_RMID,0);
    sem_destroy(s);
    shmdt(s);
    shmctl(shmid,IPC_RMID,0);
    exit(1);
}
rval=sem_post(s);
if(rval==-1)
{
    perror("SEM-POST-ERR:");
    msgctl(mqid,IPC_RMID,0);
    sem_destroy(s);
    shmdt(s);
    shmctl(shmid,IPC_RMID,0);
    exit(1);
}
exit(1);
}
else
{
    rval=sem_wait(s);
    if(rval==-1)
    {
        perror("SEM-WAIT-ERR:");
        msgctl(mqid,IPC_RMID,0);
        sem_destroy(s);
        shmdt(s);
        shmctl(shmid,IPC_RMID,0);
        exit(1);
    }
}

```

```

printf("\nPARENT : Waiting to receive message from child\n");
rval=msgrcv(mqid,(message*)&m2,sizeof(m2),msgid,0);
if(rval== -1)
{
    perror("MSG-RCV-ERR:");
    msgctl(mqid,IPC_RMID,0);
    sem_destroy(s);
    shmdt(s);
    shmctl(shmid,IPC_RMID,0);
    exit(1);
}
printf("\nPARENT : Message received is : %s\n",m2.txt);
rval=sem_post(s);
if(rval== -1)
{
    perror("SEM-POST-ERR:");
    msgctl(mqid,IPC_RMID,0);
    sem_destroy(s);
    shmdt(s);
    shmctl(shmid,IPC_RMID,0);
    exit(1);
}
msgctl(mqid,IPC_RMID,0);
sem_destroy(s);
shmdt(s);
shmctl(shmid,IPC_RMID,0);
}
}

```

OUTPUT:

compilation : `gcc -lpthread sem_msgQ.c`

execution : `./a.out`

```
Telnet 10.20.5

----- Message Queues -----
key      msqid      owner      perms      used-bytes  messages
0x00000072 0          cse16114   666        0           0
0x00000050 425985     cse1680    666        0           0
0x00000062 262146     cse1666    666        0           0
0x0000003d 360451     cse1661    666        0           0

----- Shared Memory Segments -----
key      shmid      owner      perms      bytes      nattch     status
0x00000050 4980738     cse1680    666        4           0
0x00000000 4784132     cse1666    666        4           6      dest
0x00000062 4816901     cse1666    666        4           0
0x0000003d 4915206     cse1661    666        4           1

Enter the initialization value for the semaphore :1
Semaphore value after initialization is 1

Enter the message ID to be used between parent and child: 4
CHILD : pid = 3684

CHILD : Enter the message to be sent to parent : HelloParent
PARENT : Waiting to receive message from child
PARENT : Message received is : HelloParent
[cse1680@mars CNP]$
```


IMPLEMENTING TALKER-LISTENER

Program:

```
/*TALKER PROGRAM*/

#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<string.h>

main(int argc,char* argv[])
{
    int rval,sockid,itr,i;
    char msg[100];
    struct sockaddr_in lis;
    system("clear");
    if(argc<3)
    {
        printf("\nUSAGE : %s IP_ADDR PORT#\n",argv[0]);
        exit(1);
    }
    /*DEFINITION OF SERVICE*/
    lis.sin_family=AF_INET;
    lis.sin_port=htons(atoi(argv[2]));
    lis.sin_addr.s_addr=inet_addr(argv[1]);

    sockid=socket(AF_INET,SOCK_DGRAM,0);
    if(sockid==-1)
    {
        perror("SOCK-CRE-ERR:");
        exit(1);
    }

    printf("\nEnter the number of messages to be sent: ");
    scanf("%d",&itr);
    for(i=1;i<=itr;i++)
```

```

    {
        strncpy(msg, " ",100);
        printf("Enter the message %d : ",i);
        scanf("%s",msg);
        rval=sendto(sockid,msg,sizeof(msg),0,(struct sockaddr*)&lis,sizeof(lis));
        if(rval<=0)
        {
            perror("MSG-SND-ERR:");
            close(sockid);
            exit(1);
        }
        printf("\nMessage sent successfully\n");
        if(strcmp(msg,"EXIT")==0)
        {
            close(sockid);
            exit(1);
        }
    }
    close(sockid);
}

/*LISTENER PROGRAM*/

#include<stdio.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<string.h>

main(int argc,char* argv[])
{
    int sockid,rval,count=0,tlen;
    char msg[100];
    struct sockaddr_in lis,talk;
    system("clear");
    if(argc<3)
    {

```

```

        printf("\nUSAGE : %s IP_ADDR PORT#\n",argv[0]);
        exit(1);
    }
    /*DEFINITION OF SERVICE*/
    lis.sin_family=AF_INET;
    lis.sin_port=htons(atoi(argv[2]));
    lis.sin_addr.s_addr=inet_addr(argv[1]);

    sockid=socket(AF_INET,SOCK_DGRAM,0);
    if(sockid==-1)
    {
        perror("SOCK-CRE-ERR:");
        exit(1);
    }

    /*BIND - SERVER IS LISTENING*/
    rval=bind(sockid,(struct sockaddr*)&lis,sizeof(lis));
    if(rval==-1)
    {
        perror("BIND-ERR:");
        close(sockid);
        exit(1);
    }
    printf("\nThe Listener is ready to accept messages\n");
    tlen=sizeof(talk);
    while(1)
    {
        strncpy(msg, " ",100);
        rval=recvfrom(sockid,msg,sizeof(msg),0,(struct sockaddr*)&talk,&tlen);
        if(rval==-1)
        {
            perror("MSG-RCV-ERR:");
            close(sockid);
            exit(1);
        }
        printf("\nMessage %d read is %s\n",++count,msg);
        if(!strcmp(msg,"EXIT"))

```

```
                                break;
                            }
                        close(sockid);
                    }
                }
```

OUTPUT:

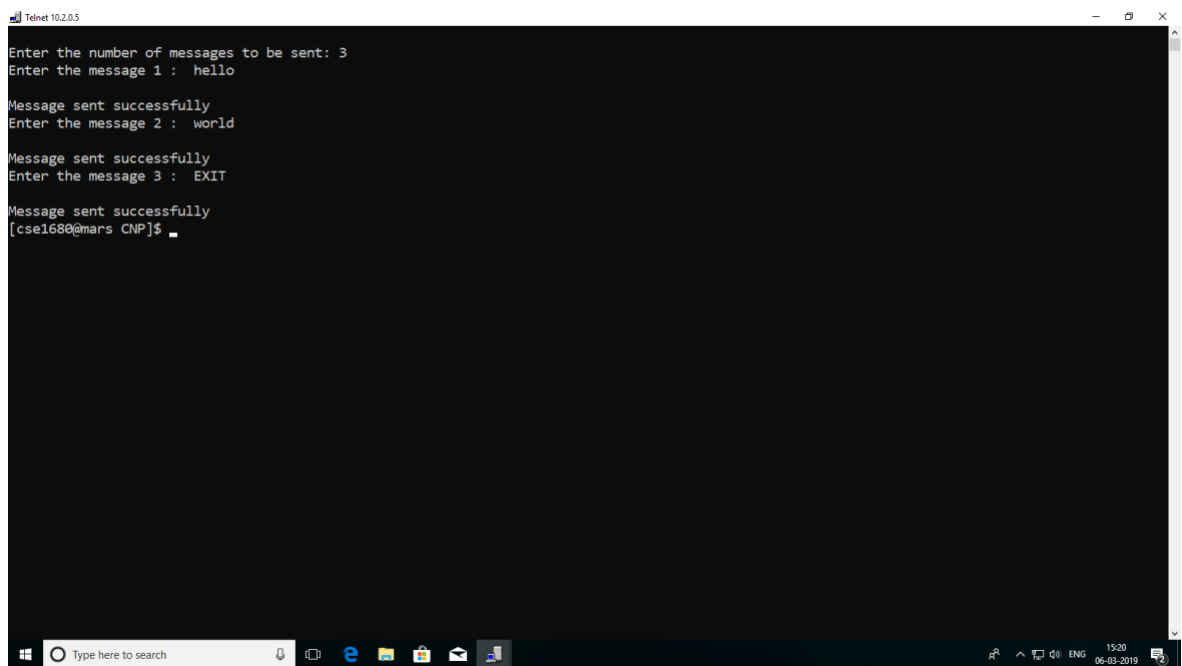
compilation : `gcc -o t udpTalker1.c`

`gcc -o l udpListener.c`

execution : `./t 10.2.0.5 5080` (Execute first and in terminal 1)

`./t 10.2.0.5 5080` (Execute second and in terminal 2)

TALKER



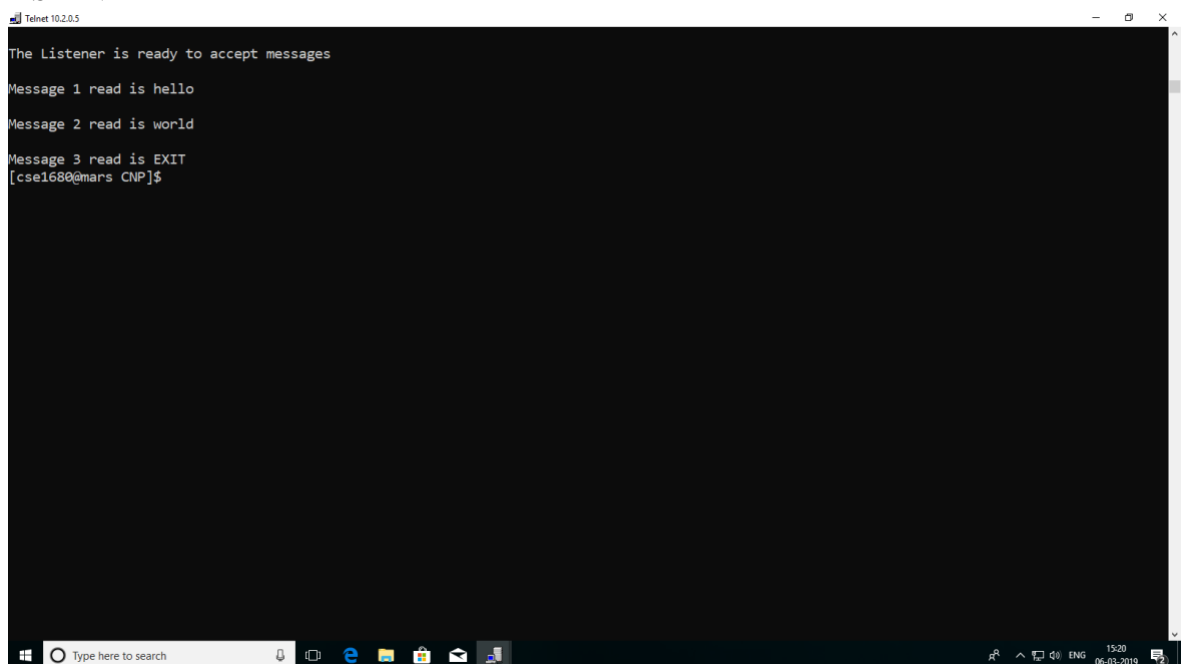
```
Telnet 10.2.0.5
Enter the number of messages to be sent: 3
Enter the message 1 : hello

Message sent successfully
Enter the message 2 : world

Message sent successfully
Enter the message 3 : EXIT

Message sent successfully
[cse1680@mars CNP]$
```

LISTENER



```
Telnet 10.2.0.5
The Listener is ready to accept messages

Message 1 read is hello

Message 2 read is world

Message 3 read is EXIT
[cse1680@mars CNP]$
```

IMPLEMENT MINI DNS

Program:

```
/*CREATING DNSfile.txt*/

lab7 10.2.0.7

lab9 10.2.0.9

lab6 10.2.0.6

/*DNS CLIENT PROGRAM*/

#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<string.h>

main(int argc,char* argv[])
{
    struct sockaddr_in dnss;
    int sockid,rval;
    char sym[20],IP[20];
    int slen;
    system("clear");
    if(argc<3)
    {
        printf("\nUSAGE : %s IP_ADDR PORT#\n",argv[0]);
        exit(1);
    }
    sockid=socket(AF_INET,SOCK_DGRAM,0);
    if(sockid==-1)
    {
        perror("SOCK-CRE-ERR:");
        exit(1);
    }
    dnss.sin_family=AF_INET;
    dnss.sin_port=htons(atoi(argv[2]));
```

```

    dnss.sin_addr.s_addr=inet_addr(argv[1]);

    printf("\nEnter the symbolic name of resource : ");
    scanf("%s",sym);

    rval=sendto(sockid,sym,sizeof(sym),0,(struct sockaddr*)&dnss,sizeof(dnss));
    if(rval==-1)
    {
        perror("MSG-SND-ERR:");
        close(sockid);
        exit(1);
    }
    printf("\nWaiting to receive from DNS Server\n");
    slen=sizeof(dnss);
    strncpy(IP," ",20);
    rval=recvfrom(sockid,IP,sizeof(IP),0,(struct sockaddr*)&dnss,&slen);
    if(rval==-1)
    {
        perror("MSG-RCV-ERR:");
        close(sockid);
        exit(1);
    }
    printf("\nEquivalent IP address of %s is %s\n",sym,IP);
    close(sockid);
}

/*DNS SERVER PROGRAM*/

#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<string.h>

main(int argc,char* argv[])
{
    struct sockaddr_in dnss,dnsc;

```

```

int rval,sockid,flag=0,clen;
char sym[20],IP[20],dnsFile[20],dnsName[20];
FILE *fptr;
system("clear");
if(argc<3)
{
    printf("\nUSAGE : %s IP_ADDR PORT#\n",argv[0]);
    exit(1);
}
dnss.sin_family=AF_INET;
dnss.sin_port=htons(atoi(argv[2]));
dnss.sin_addr.s_addr=inet_addr(argv[1]);

sockid=socket(AF_INET,SOCK_DGRAM,0);
if(sockid==-1)
{
    perror("SOCK-CRE-ERR:");
    exit(1);
}

rval=bind(sockid,(struct sockaddr*)&dnss,sizeof(dnss));
if(rval==-1)
{
    perror("BIND-ERR:");
    close(sockid);
    exit(1);
}

printf("\nDNS Server waiting for request\n");
printf("\nEnter the DNS file name : ");
scanf("%s",dnsFile);
/*OPEN THE FILE*/
fptr=fopen(dnsFile,"r");
if(fptr==NULL)
{
    perror("FILE-OPEN-ERR:");
    close(sockid);

```



```

        exit(1);
    }
    clen=sizeof(dnsc);
    rval=recvfrom(sockid,sym,sizeof(sym),0,(struct sockaddr*)&dnsc,&clen);
    if(rval==-1)
    {
        perror("MSG-RCV-ERR:");
        close(sockid);
        fclose(fptr);
        exit(1);
    }
    printf("\nIP requested for %s\n",sym);
    while((fscanf(fptr,"%s%s",dnsName,IP) != EOF) )
    {
        if(strcmp(dnsName,sym)==0)
        {
            rval=sendto(sockid,IP,sizeof(IP),0,(struct
sockaddr*)&dnsc,clen);
            if(rval==-1)
            {
                perror("MSG-SND-ERR:");
                fclose(fptr);
                close(sockid);
                exit(1);
            }
            flag=1;
        }
        printf("\n flag value in loop is %d\n",flag);
        if(flag==1) //INDICATES THAT MATCH IS FOUND
            break;
    }
    if(flag==0)
    {
        printf("\n invalid domain name case\n");
        rval=sendto(sockid,"NOT FOUND",sizeof("NOT FOUND"),0,(struct
sockaddr*)&dnsc,clen);
        if(rval==-1)

```

```
        {  
            perror("MSG-SND-ERR:");  
            fclose(fptr);  
            close(sockid);  
            exit(1);  
        }  
    }  
    fclose(fptr);  
    close(sockid);  
}
```

OUTPUT:

```
compilation : gcc -o cout dnsClient.c
```

```
gcc -o sout dnsServer.c
```

execution : `./sout 10.2.0.5 5080` (Execute server first and in terminal 1)

./cout 10.2.0.5 5080 (Execute client second and in terminal 2)

DNSfile.txt

Telnet 10.2.0.5

```
lab7 10.2.0.7
lab9 10.2.0.9
lab6 10.2.0.6
```

"DNSfile.txt" 3L, 42C

2,1 All

Type here to search

15:08 13-03-2019

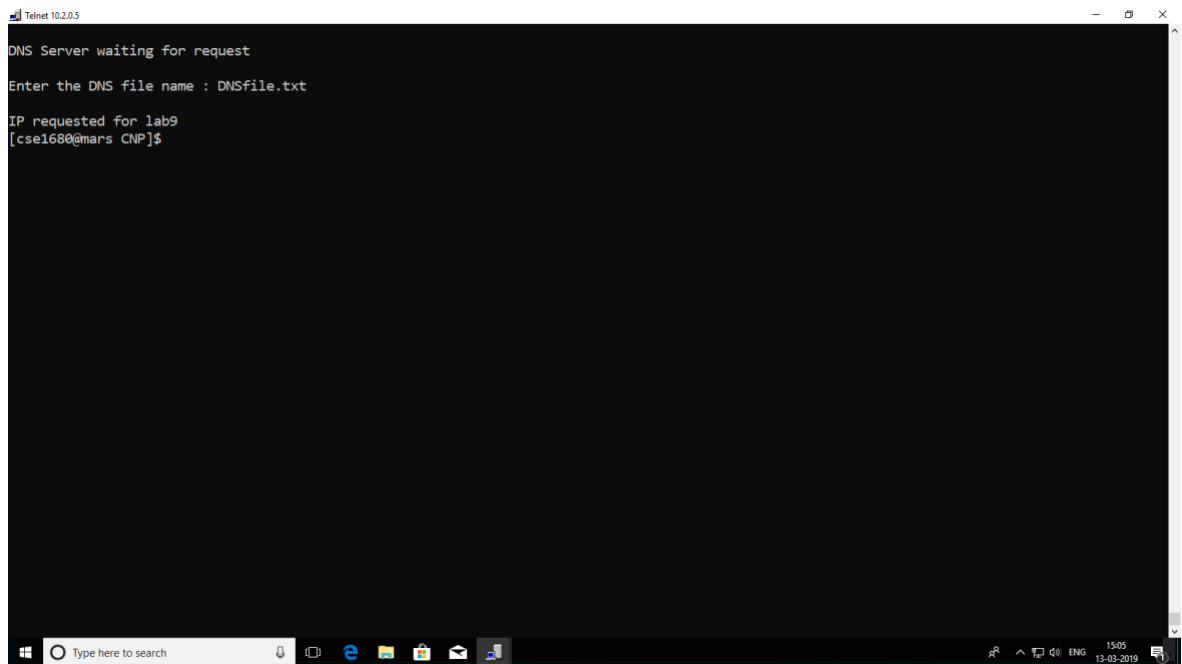
DNS CLIENT

The screenshot shows a Windows desktop with a Telnet window open. The window title is "Telnet 10.2.0.5". The text displayed in the window is as follows:

```
Enter the symbolic name of resource : lab9  
Waiting to receive from DNS Server  
Equivalent IP address of lab9 is 10.2.0.9  
[cse1680@mars CNP]$
```

The bottom of the screen shows the Windows taskbar with the search bar containing "Type here to search" and several application icons. The system tray on the right indicates the time is 15:06 and the date is 13-03-2019.

DNS SERVER

A screenshot of a Telnet window titled 'Telnet 10.2.0.5'. The window has a black background with white text. The text inside the window reads: 'DNS Server waiting for request', 'Enter the DNS file name : DNSfile.txt', 'IP requested for lab9', and '[cse1680@mars CNP]\$'. The window is open on a Windows desktop, with the taskbar visible at the bottom. The taskbar includes the Start button, a search bar with the text 'Type here to search', and several application icons. The system tray on the right shows the date and time as '15:05 13-03-2019' and the language as 'ENG'.

```
Telnet 10.2.0.5

DNS Server waiting for request

Enter the DNS file name : DNSfile.txt

IP requested for lab9
[cse1680@mars CNP]$
```

DEMONSTRATE NON-BLOCKING I/O USING select SYSTEM CALL

Program:

```
/*selectNB.c*/

#include<stdio.h>
#include<stdlib.h>
#include<sys/time.h>
#include<sys/types.h>
#include<unistd.h>

int main(void)
{
    fd_set rfd;
    struct timeval tv;
    int ret;
    char a[100];
    FD_ZERO(&rfd);
    FD_SET(0,&rfd);

    tv.tv_sec=5;
    tv.tv_usec=0;

    ret=select(1,&rfd,NULL,NULL,&tv);

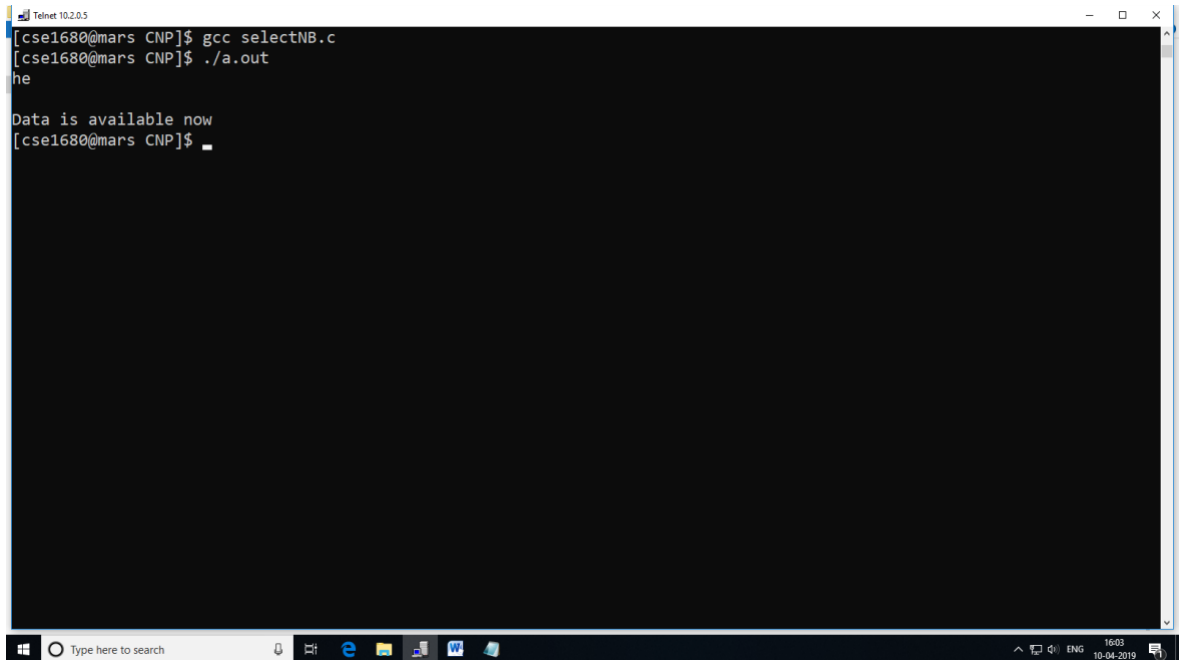
    if(ret==-1)
        perror("select()");
    else if(ret)
    {
        printf("\nData is available now\n");
        scanf("%s",&a);
    }
    else
        printf("No data within five Seconds.\n");
    exit(EXIT_SUCCESS);
}
```

OUTPUT:

Compilation : *gcc selectNB.c*

Execution : *./a.out*

WHEN DATA IS AVAILABLE

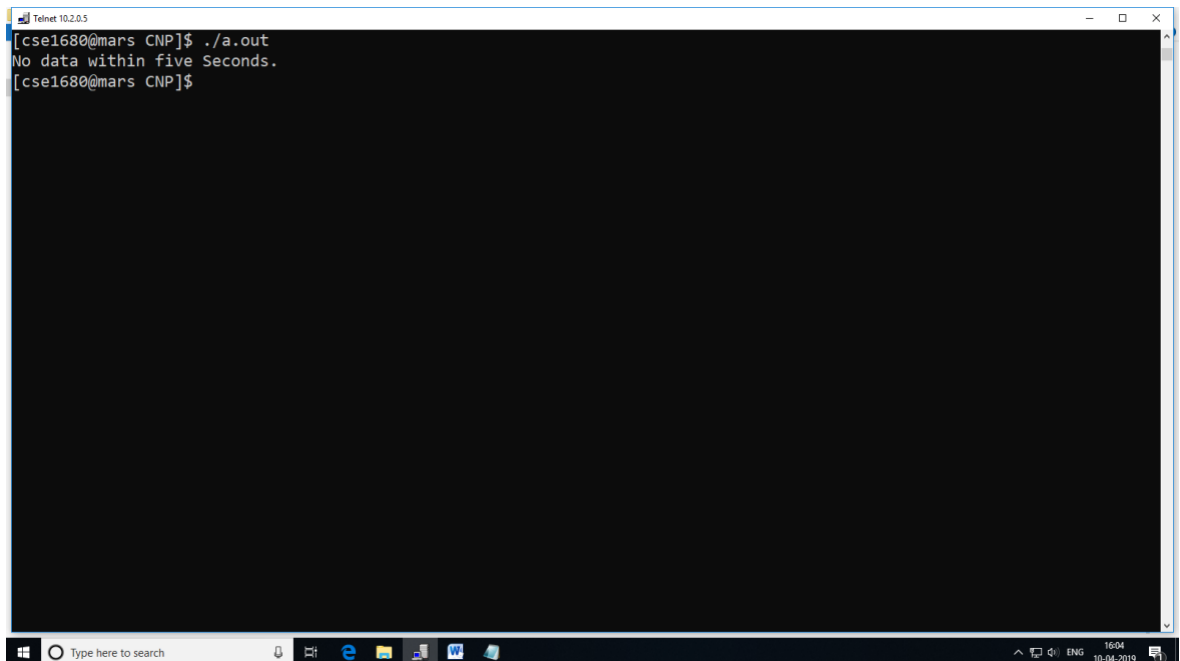


A screenshot of a Telnet 10.2.0.5 window. The terminal shows the following commands and output:

```
[cse1680@mars CNP]$ gcc selectNB.c
[cse1680@mars CNP]$ ./a.out
he
Data is available now
[cse1680@mars CNP]$
```

The window has a standard Windows taskbar at the bottom with the search bar and system tray showing the date 10-04-2019 and time 16:03.

WHEN NO DATA IS AVAILABLE



A screenshot of a Telnet 10.2.0.5 window. The terminal shows the following commands and output:

```
[cse1680@mars CNP]$ ./a.out
No data within five Seconds.
[cse1680@mars CNP]$
```

The window has a standard Windows taskbar at the bottom with the search bar and system tray showing the date 10-04-2019 and time 16:04.

DEMONSTRATE SCATTER AND GATHER I/O USING readv() AND writev()

Program:

```
/*readFrom.txt*/
```

```
HelloWorld
```

```
/*writeTo.txt*/
```

```
GoodAfternoon
```

```
à,, P ^uä ðuä¿p- ä, .N= t
```

```
/*readWrite.c*/
```

```
#include<stdio.h>
```

```
#include<stdlib.h>
```

```
#include<sys/uio.h>
```

```
#include<string.h>
```

```
#include<sys/types.h>
```

```
#include<fcntl.h>
```

```
#include<sys/stat.h>
```

```
main()
```

```
{
```

```
    struct iovec iov[3];
```

```
    char s[40],s1[30],s2[25],fname[10],cmdstr[40];
```

```
    int i,rval,fd,choice;
```

```
    system("clear");
```

```
    iov[0].iov_base=s;
```

```
    iov[0].iov_len=sizeof(s);
```

```
    iov[1].iov_base=s1;
```

```
    iov[1].iov_len=sizeof(s1);
```

```
    iov[2].iov_base=s2;
```

```
    iov[2].iov_len=sizeof(s2);
```

```

printf("\nEnter your choice to read data from standard input or file:\n\t1.Standard
Input\n\t2.File\n");
scanf("%d",&choice);
printf("\nChoice : %d\n",choice);
if(choice==1)
{
    fd=0;
    printf("\nReading from keyboard and writing to multiple buffers\n");
}
else
{
    if(choice==2)
    {
        printf("\nEnter the filename: ");
        scanf("%s",fname);

        fd=open(fname,O_RDONLY);
        if(fd==-1)
        {
            perror("FILE-OPEN-ERR:");
            exit(1);
        }
    }
    else
    {
        printf("\nInvalid Choice\n");
        exit(1);
    }
}
rval=readv(fd,(struct iovec*)iov,3);
printf("\nAfter readv\n");
s[39]='\0';
s1[29]='\0';
s2[24]='\0';

printf("\nrval = %d\n",rval);

```



```

if(rval!=-1)
{
    printf("\nContents of the three buffers are:\n");
    printf("Data in s is %s\t%d\n",s,strlen(s));
    printf("Data in s1 is %s\t%d\n",s1,strlen(s1));
    printf("Data in s2 is %s\t%d\n",s2,strlen(s2));
}
else
{
    perror("RDV-ERR:");
}
if(choice==2)
    close(fd);
printf("\nEnter the filename to write to: ");
scanf("%s",fname);
fd=open(fname,O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
if(fd!=-1)
    rval=writev(fd,(struct iovec*)iov,3);
if(rval!=-1)
    printf("\n%d Bytes written to %s file\n",rval,fname);
close(fd);
strncpy(cmdstr," ",40);
strcpy(cmdstr,"chmod 700 ");
strcat(cmdstr,fname);
strcat(cmdstr,"; ");
strcat(cmdstr,"cat ");
strcat(cmdstr,fname);
printf("\ncmdstr = %s\n",cmdstr);
system(cmdstr);
}

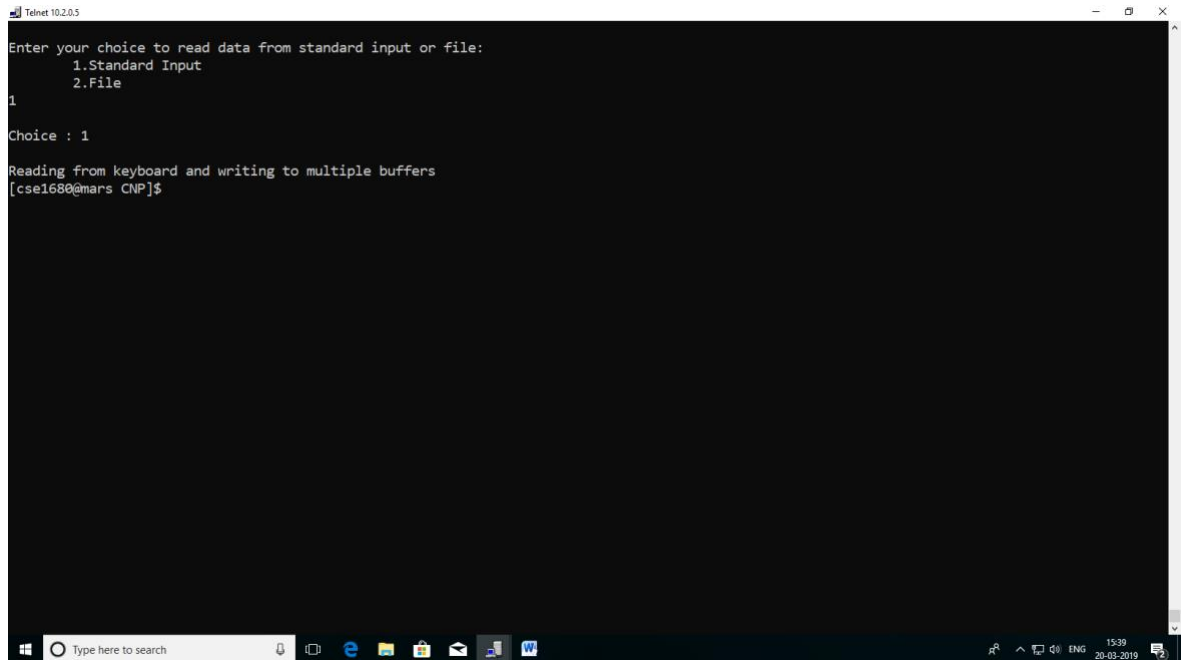
```

OUTPUT:

Compilation : `gcc readWrite.c`

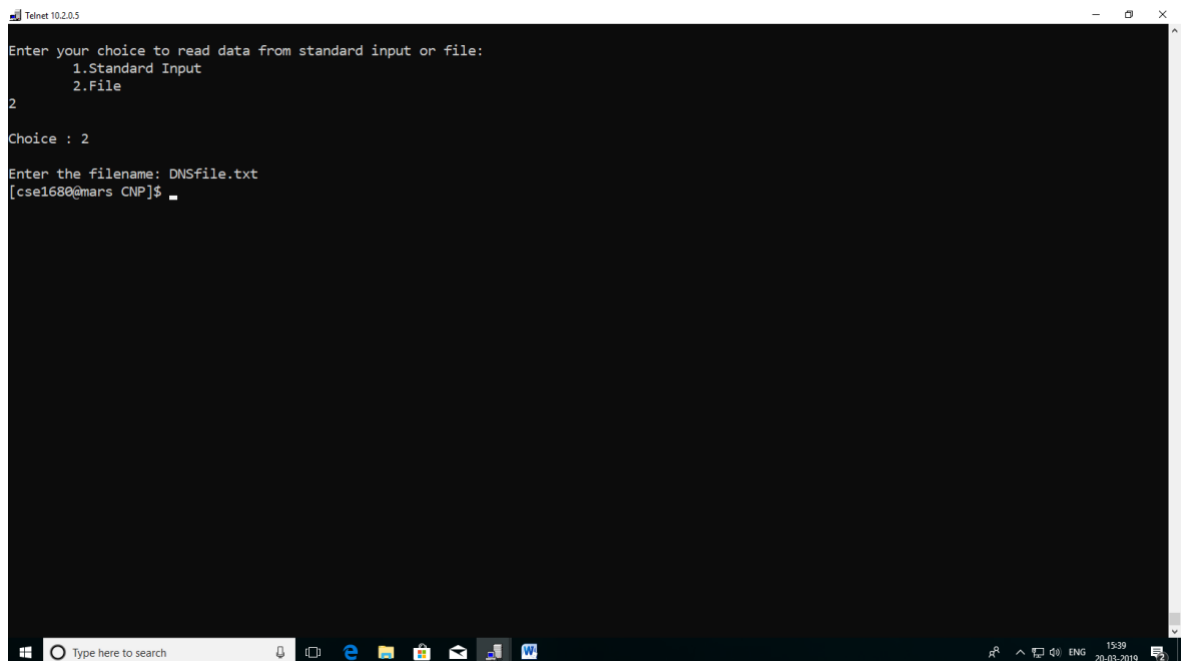
Execution : `./a.out`

STANDARD INPUT

A screenshot of a Telnet window titled 'Telnet 10.2.0.5'. The window shows a program prompt 'Enter your choice to read data from standard input or file:' with two options: '1.Standard Input' and '2.File'. The user has entered '1'. The program then displays 'Choice : 1' and 'Reading from keyboard and writing to multiple buffers'. The prompt '[cse1680@mars CNP]\$' is visible at the bottom. The Windows taskbar is visible at the bottom of the window.

```
Telnet 10.2.0.5
Enter your choice to read data from standard input or file:
  1.Standard Input
  2.File
1
Choice : 1
Reading from keyboard and writing to multiple buffers
[cse1680@mars CNP]$
```

FILE

A screenshot of a Telnet window titled 'Telnet 10.2.0.5'. The window shows a program prompt 'Enter your choice to read data from standard input or file:' with two options: '1.Standard Input' and '2.File'. The user has entered '2'. The program then displays 'Choice : 2' and 'Enter the filename: DNSfile.txt'. The prompt '[cse1680@mars CNP]\$' is visible at the bottom. The Windows taskbar is visible at the bottom of the window.

```
Telnet 10.2.0.5
Enter your choice to read data from standard input or file:
  1.Standard Input
  2.File
2
Choice : 2
Enter the filename: DNSfile.txt
[cse1680@mars CNP]$
```

DEMONSTRATE SOCKET OPTIONS

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<netinet/in.h>
#include<string.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/tcp.h>
main()
{
    int optlen,gs,sockopt,sockid;
    sockid=socket(AF_INET,SOCK_STREAM,0);
    if(sockid==-1)
    {
        perror("SOCK-CRE-ERR:");
        exit(1);
    }
    /*LEVEL-SOCKET OPTION-SOCKET TYPE*/
    optlen=sizeof(sockopt);
    gs=getsockopt(sockid,SOL_SOCKET,SO_TYPE,&sockopt,&optlen);
    if(gs==-1)
    {
        perror("GETSOCK-ERR:");
        close(sockid);
        exit(1);
    }
    switch(sockopt)
    {
        case SOCK_DGRAM                :    printf("\nDatagram
Socket\n");
        break;
        case SOCK_STREAM                :    printf("\nStream Socket\n");
        break;
        case SOCK_RAW                    :    printf("\nRaw Socket\n");
        break;
```

```

                                default                                :                                printf("\nUnknown Socket
type\n");
    }
    /*LEVEL-SOCKET OPTION-SEND QUEUE BUFFER SIZE*/
    optlen=sizeof(sockopt);
    gs=getsockopt(sockid,SOL_SOCKET,SO_SNDBUF,&sockopt,&optlen);
    if(gs!=-1)
        printf("\nSend buffer size is %d\n",sockopt);
    else
        perror("SND-BUF-ERR:");
    //Usage of setsockopt function
    /*LEVEL-SOCKET OPTION-SEND QUEUE BUFFER SIZE setting size of 2048
Note Default size is 4096*/
    sockopt=2048;
    optlen=sizeof(sockopt);
    gs=getsockopt(sockid,SOL_SOCKET,SO_SNDBUF,&sockopt,&optlen);
    if(gs!=-1)
        printf("\nSend Buffer size is %d\n",sockopt);
    else
        perror("SND-BUF-ERR:");

    /*LEVEL-SOCKET OPTION-RECEIVE QUEUE BUFFER SIZE*/
    optlen=sizeof(sockopt);
    gs=getsockopt(sockid,SOL_SOCKET,SO_RCVBUF,&sockopt,&optlen);
    if(gs!=-1)
        printf("\nReceive Buffer size is %d\n",sockopt);
    else
        perror("RCV-BUF-ERR:");

    /*LEVEL-TCP OPTION-TCP MAX SEGMENT SIZE*/
    optlen=sizeof(sockopt);
    gs=getsockopt(sockid,IPPROTO_TCP,TCP_MAXSEG,&sockopt,&optlen);
    if(gs!=-1)
        printf("\nTCP MAX Segment size is %d\n",sockopt);
    else
        perror("TCP-SEG-ERR:");

```

```

/*LEVEL-TCP OPTION-TCP NO-DELAY*/
optlen=sizeof(sockopt);
sockopt=1;
gs=setsockopt(sockid,IPPROTO_TCP,TCP_NODELAY,&sockopt,optlen);
if(gs!=-1)
    printf("\nNODELAY FLAG is set\n");
else
    perror("TCP-NODELAY-ERR:");

//Usage of setsockopt function
/*LEVEL-TCP OPTION-TCP NODELAY*/
optlen=sizeof(sockopt);
gs=getsockopt(sockid,IPPROTO_TCP,TCP_NODELAY,&sockopt,&optlen);
if(gs!=-1)
    if(sockopt==1)
        printf("\nNODELAY flag is set\n");
    else
        printf("\nNODELAY flag not set\n");
else
    perror("TCP-NODELAY-ERR:");

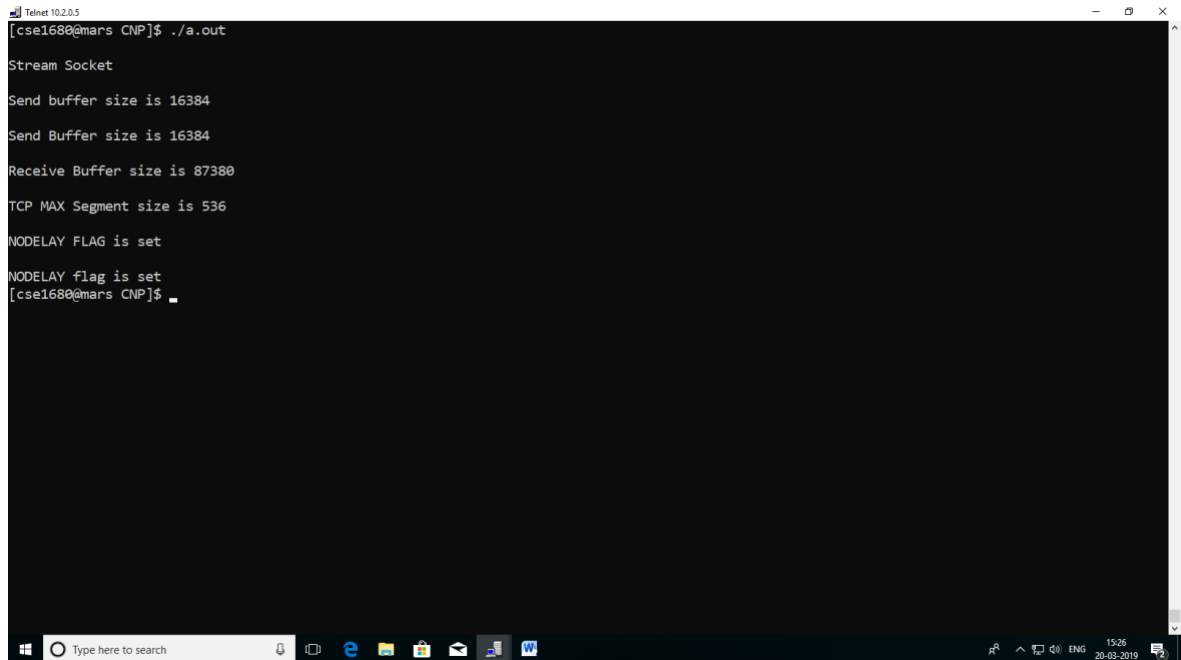
close(sockid);
}

```

OUTPUT:

Compilation : *gcc sockOpt.c*

Execution : *./a.out*



```
Telnet 10.20.5
[cse1680@mars CNP]$ ./a.out
Stream Socket
Send buffer size is 16384
Send Buffer size is 16384
Receive Buffer size is 87380
TCP MAX Segment size is 536
NODELAY FLAG is set
NODELAY flag is set
[cse1680@mars CNP]$
```

DEMONSTRATE USAGE OF select() SYSTEM CALL WITH SOCKET DESCRIPTOR

Program:

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

main(int argc,char* argv[])
{
    fd_set rdfs;
    int rval,sockfd;
    char buf1[3],buf2[30];
    struct sockaddr_in s;
    struct timeval tv;

    system("clear");
    if(argc<2)
    {
        printf("\nUSAGE : %s IP_ADDR PORT#\n",argv[0]);
        exit(1);
    }

    s.sin_family=AF_INET;
    s.sin_port=htons(atoi(argv[2]));
    s.sin_addr.s_addr=inet_addr(argv[1]);

    sockfd=socket(AF_INET,SOCK_STREAM,6);
    if(sockfd==-1)
    {
        perror("SOCK-CRE-ERR:");
        exit(1);
    }
```

```

rval=connect(sockfd,(struct sockaddr *)&s,sizeof(s));
if(rval==-1)
{
    perror("CONNECT_ERR");
    close(sockfd);
    exit(1);
}

printf("\n Enter the data to send:\t");
scanf("%s",buf1);

send(sockfd,(char *) buf1,sizeof(buf1),0);
do
{
    FD_ZERO(&rdfs);
    FD_SET(sockfd,&rdfs);
    rval=select(sockfd+1,&rdfs,NULL,NULL,NULL);
    printf("\nrval of select = %d\n",rval);
}while(rval==-1);

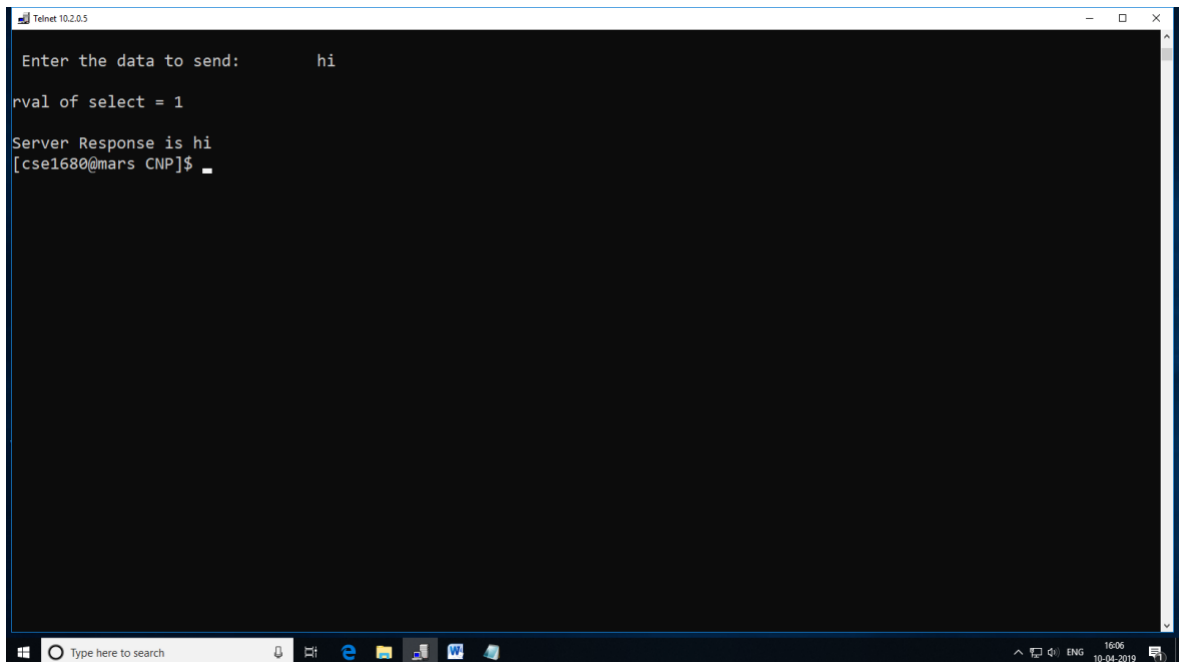
if(rval>0)
{
    if(FD_ISSET(sockfd,&rdfs))
    {
        rval=recv(sockfd,(char *)buf2,sizeof(buf2),0);
        if(rval==0)
            printf("\nNo response from server\n");
        else
            printf("\nServer Response is %s\n",buf2);
    }
}
else
    perror("SELECT_ERR");
close(sockfd);
}

```


OUTPUT:

Compilation : gcc sockdtr.c

Execution : ./a.out 10.2.0.5 7 (7 is the port for Standard Echo Service)



```
Telnet 10.2.0.5

Enter the data to send:      hi
rval of select = 1
Server Response is hi
[cse1680@mars CNP]$
```

The screenshot shows a Telnet window titled 'Telnet 10.2.0.5'. The user enters 'hi' in response to the prompt 'Enter the data to send:'. The program then outputs 'rval of select = 1' and 'Server Response is hi'. The prompt '[cse1680@mars CNP]\$' is visible at the bottom of the window. The Windows taskbar is visible at the bottom of the screen.

UTILITIES : ARP (ADDRESS RESOLUTION PROTOCOL)

Description:

ARP manipulates or displays the kernel's IPv4 network neighbor cache. It can add entries to the table, delete one or display the current content.

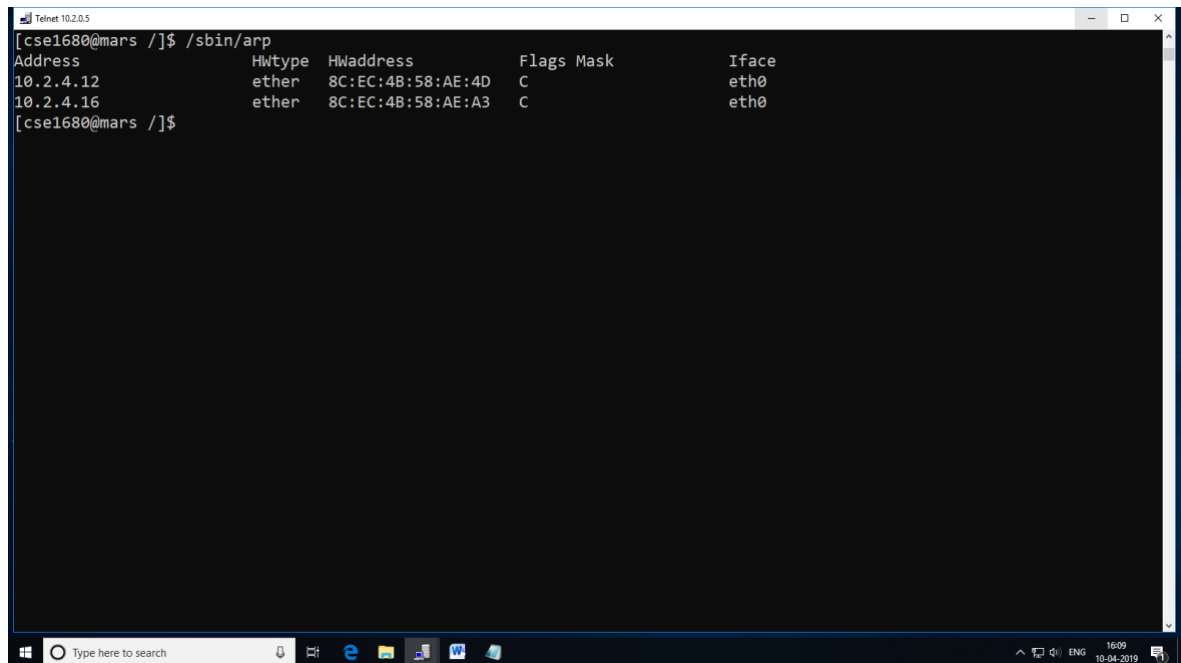
ARP stands for *Address Resolution Protocol*, which is used to find the media access control address of a network neighbor for a given IPv4 Address.

Although every machine on the Internet has one or more IP addresses, these are not sufficient for sending packets. Data Link Layer NICs (**N**etwork **I**nterface **C**ards), such as Ethernet cards, do not understand the Internet addresses. Thus, there is a need to map the logical (IP) addresses to the hardware (MAC) addresses.

The solution is to let the host, that wishes to communicate with the other hosts, output a broadcast packet on to the Ethernet (or the network) asking who owns the IP address. The host whose IP address matches will respond with its MAC address. Now both logical and hardware addresses are known, so the routing can be done uninterruptedly.

OUTPUT:

Execution : *cd / (exit all directories and come to the login)*
/sbin/arp



The screenshot shows a Telnet session window titled 'Telnet 10.2.0.5'. The user is logged in as 'cse1680@mars' and is in the root directory '/'. They have executed the command '/sbin/arp'. The output is a table with the following columns: Address, Hwtype, Hwaddress, Flags, Mask, and Iface. There are two entries in the table, both for the interface 'eth0'.

Address	Hwtype	Hwaddress	Flags	Mask	Iface
10.2.4.12	ether	8C:EC:4B:58:AE:4D	C		eth0
10.2.4.16	ether	8C:EC:4B:58:AE:A3	C		eth0

The prompt returns to '[cse1680@mars /]\$'.

UTILITIES : IFCONFIG

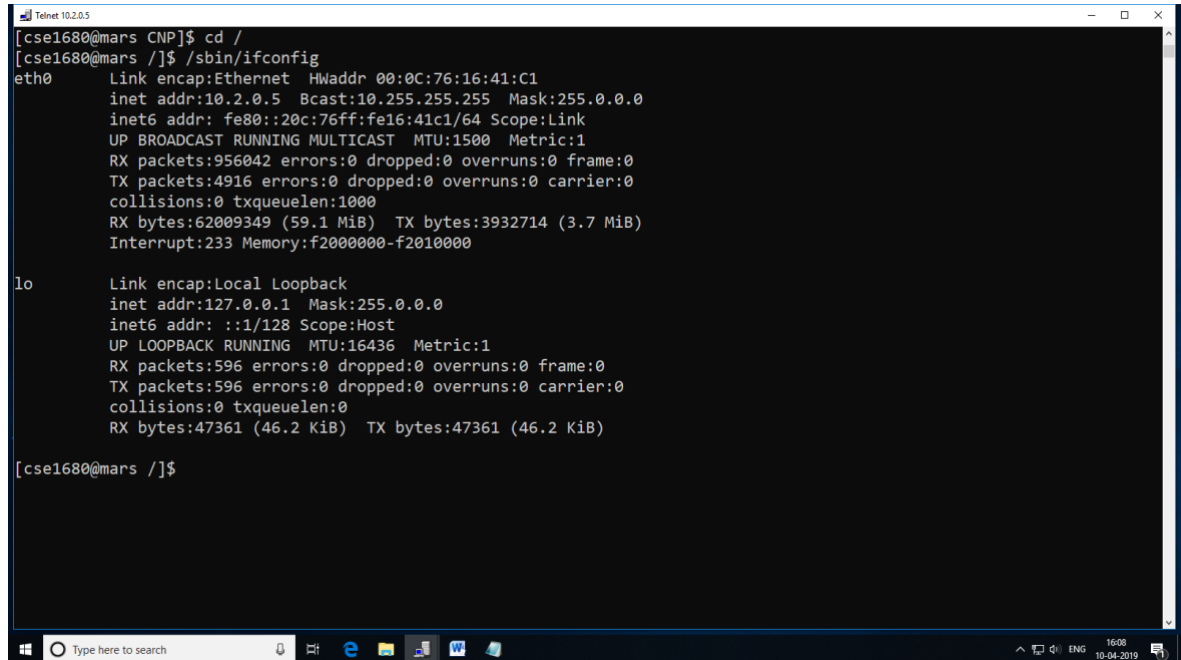
Description:

Ifconfig is used to configure the kernel-resident network interfaces. It is used at boot time to set up interfaces as necessary. After that, it is usually only needed when debugging or when system tuning is needed.

If no arguments are given, ifconfig displays the status of the currently active interfaces. If a single interface argument is given, it displays the status of the given interface only; if a single *-a* argument is given, it displays the status of all interfaces, even those that are down. Otherwise, it configures an interface.

OUTPUT:

Execution : *cd / (exit all directories and come to the login)*
/sbin/ifconfig



```
Telnet 10.2.0.5
[cse1680@mars CNP]$ cd /
[cse1680@mars /]$ /sbin/ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0C:76:16:41:C1
          inet addr:10.2.0.5  Bcast:10.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::20c:76ff:fe16:41c1/64  Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:956042 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4916 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:62009349 (59.1 MiB)  TX bytes:3932714 (3.7 MiB)
          Interrupt:233 Memory:f2000000-f2010000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128  Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:596 errors:0 dropped:0 overruns:0 frame:0
          TX packets:596 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:47361 (46.2 KiB)  TX bytes:47361 (46.2 KiB)

[cse1680@mars /]$
```

UTILITIES : PING

Description:

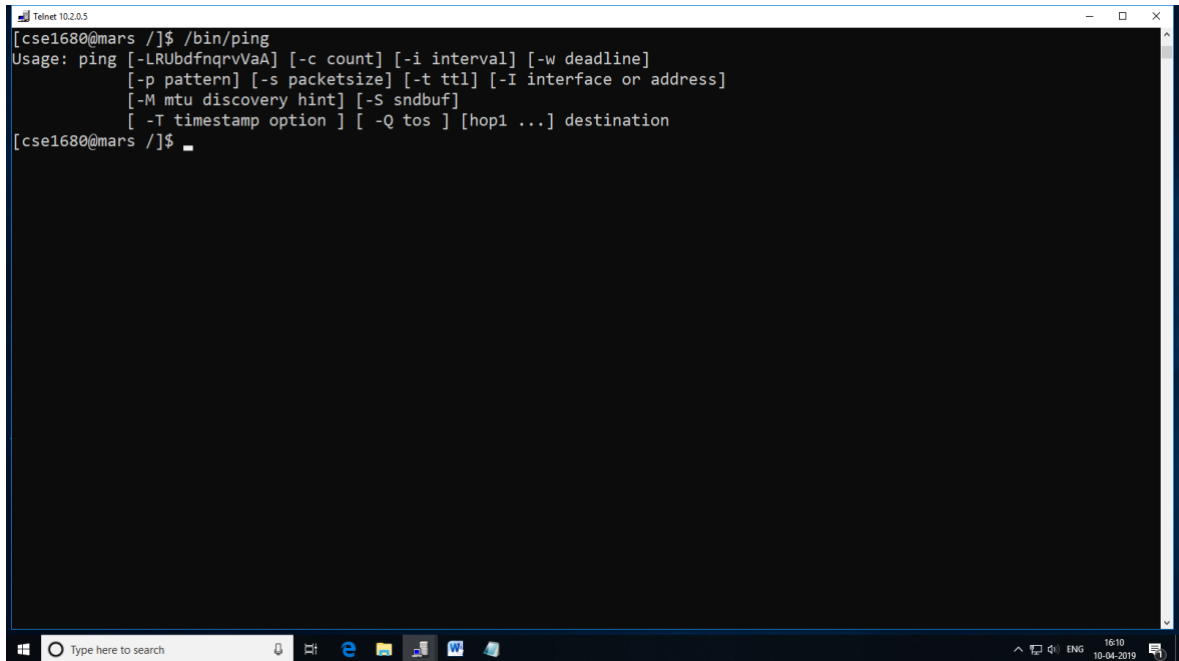
ping uses the ICMP (Internet Control Message Protocol) protocol's mandatory *ECHO_REQUEST* datagram to elicit an ICMP *ECHO_RESPONSE* from a host or gateway. *ECHO_REQUEST* datagrams ("pings") have an IP and ICMP header, followed by a *struct timeval* and then an arbitrary number of "pad" bytes used to fill out the packet.

ping is used to test and check if the network/transmission line/ the service is working fine or not. When the message is sent from the host it must be echoed by the peer entity. If the response message is the exact same as the request message the service is up and on the Internet. Otherwise, some problem lies in the service or transmission. It is thus used for verifying if the service is alive and working fine.

ping6 is IPv6 version of *ping*, and can also send Node Information Queries (RFC4620). Intermediate hops may not be allowed, because IPv6 source routing was deprecated (RFC5095).

OUTPUT:

Execution : cd / (exit all directories and come to the login)
/bin/ping



The screenshot shows a Telnet window titled 'Telnet: 10.2.0.5'. The prompt is '[cse1680@mars ~]\$'. The user has entered '/bin/ping'. The output shows the usage of the ping command: 'Usage: ping [-LRUbdnqrVvA] [-c count] [-i interval] [-w deadline] [-p pattern] [-s packetsize] [-t ttl] [-I interface or address] [-M mtu discovery hint] [-S sndbuf] [-T timestamp option] [-Q tos] [hop1 ...] destination'. The prompt is now '[cse1680@mars ~]\$' with a cursor.

```
Telnet: 10.2.0.5
[cse1680@mars ~]$ /bin/ping
Usage: ping [-LRUbdnqrVvA] [-c count] [-i interval] [-w deadline]
          [-p pattern] [-s packetsize] [-t ttl] [-I interface or address]
          [-M mtu discovery hint] [-S sndbuf]
          [-T timestamp option] [-Q tos] [hop1 ...] destination
[cse1680@mars ~]$
```

UTILITIES : TRACEROUTE

Description:

Traceroute attempts to trace the route an IP packet follows to some internet host. It finds out intermediate hops by launching probe packets with a small **Time-to-Live (*TtL*)** value, and then listens for an ICMP reply of *TIME EXCEEDED* from an intermediate router. Traceroute starts probing with a *TtL* of one, and increments by one until an ICMP port unreachable reply is received. This means the probe either got through to host, or hit the maximum *TtL*.

The host explicitly sets Time to Live field value of the IP packet in incrementally small values making sure that the packet's *TtL* exhausts at each router on the route from the host to the destination. When the *TtL* field of the IP packet hits 0 (zero) at any intermediate router then the router dispatches a *TIME EXCEEDED* ICMP message encapsulated in an IP packet. The host, from this diagnostic message, extracts the IP address of the router and thus, comes to know of its existence in the path at the '*hop count*' distance that it had set in the *TtL* field of the original packet. This way, ICMP can be smartly used to trace the routers between the sender host and the destination host.

host is the only mandatory argument, and specifies the target system, either as an IP address, or as a host name. Parameter size determines the size of the probe packets in bytes.

OUTPUT:

Execution : *cd / (exit all directories and come to the login)*

/bin/traceroute

```
Telnet 10.2.0.5
cse1674          *pts/1          Apr 10 16:54          (10.2.4.16)
cse1680          *pts/2          Apr 10 16:33          (10.2.4.16)
[cse1680@mars ~]$ /bin/traceroute
Usage:
  /bin/traceroute [ -4dFITUnrAV ] [ -f first_ttl ] [ -g gate,... ] [ -i device ] [ -m max_ttl ] [ -N squeries ] [ -p port ] [ -t tos ] [ -l flow_label ] [ -w waittime ] [ -q nqueries ] [ -s src_addr ] [ -z sendwait ] host [ packetlen ]
Options:
  -4                      Use IPv4
  -6                      Use IPv6
  -d --debug              Enable socket level debugging
  -F --dont-fragment      Set DF (don't fragment bit) on
  -f first_ttl --first=first_ttl
                          Start from the first_ttl hop (instead from 1)
  -g gate,... --gateway=gate,...
                          Route packets throw the specified gateway
                          (maximum 8 for IPv4 and 127 for IPv6)
  -I --icmp               Use ICMP ECHO for tracerouting
  -T --tcp                Use TCP SYN for tracerouting
  -U --udp                Use UDP datagram (default) for tracerouting
  -i device --interface=device
                          Specify a network interface to operate with
  -m max_ttl --max-hops=max_ttl
                          Set the max number of hops (max TTL to be
                          reached). Default is 30
  -N squeries --sim-queries=squeries
                          Set the number of probes to be tried
                          simultaneously (default is 16)
  -n                      Do not resolve IP addresses to their domain names
  -p port --port=port     Use destination port port. It is an initial value
```

UTILITIES : FINGER

Description:

Finger may be used to look up users on a remote machine. *finger* is a program you can use to find information about computer users. It usually lists the login name, the full name, and possibly other details about the user you are fingering. These details may include the office location and phone number (if known), login time, idle time, time mail was last read, and the user's plan and project files. The information listed varies, and you may not be able to get any information from some sites.

In some cases, you may be able to use the *finger* command to verify an address or find more information for someone at another institution about whom you already have some information. The *finger* command is available on most Unix systems. It differs from the '*who is*' command, which you can use simply to find the email address of someone at another institution.

OUTPUT:

Execution : *cd / (exit all directories and come to the login)*
finger

```
Telnet 10.2.0.5
[cse1680@mars /]$ finger
Login      Name      Tty      Idle      Login Time   Office      Office Phone  Host
cse1669    Name      *pts/0    Apr 10 16:53
cse1674    *pts/1    Apr 10 16:54
cse1680    *pts/2    Apr 10 16:33
[cse1680@mars /]$
```

RPC (REMOTE PROCEDURE CALL) : ADDITION OF TWO NUMBERS

Program:

```

/*addition.x*/

struct nums
{
    int a;
    int b;
};

program sum_prog{
version sum_ver{
    int add(nums)=1;
    }=1;
    }=0x20001080;

/*addition_client.c*/

/*
 * This is sample code generated by rpcgen.
 * These are only templates and you can use them
 * as a guideline for developing your own functions.
 */

#include "addition.h"

void
sum_prog_1(char *host)
{
    CLIENT *clnt;
    int *result_1;
    nums add_1_arg; //structure variable

#ifdef  DEBUG
    clnt = clnt_create (host, sum_prog, sum_ver, "udp");
    if (clnt == NULL) {
        clnt_pcreateerror (host);
        exit (1);
    }

```

```

    }
    /*ASK USER TO ENTER TWO NUMBERS*/
    printf("\nEnter 2 numbers to add : ");
    scanf("%d%d",&add_1_arg.a,&add_1_arg.b);
#endif    /* DEBUG */
    result_1=(int*)malloc(sizeof(int));
    result_1 = add_1(&add_1_arg, clnt);
    if (result_1 == (int *) NULL) {
        clnt_perror (clnt, "call failed");

    }

    printf("\nSum of %d\t%d\t is %d\n",add_1_arg.a,add_1_arg.b,*result_1);
#ifdef    DEBUG
    clnt_destroy (clnt);
#endif    /* DEBUG */
}

int
main (int argc, char *argv[])
{
    char *host;

    if (argc < 2) {
        printf ("usage: %s server_host\n", argv[0]);
        exit (1);
    }
    host = argv[1];
    sum_prog_1 (host);
    exit (0);
}
/*addition_clnt.c*/

/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

```

```

#include <memory.h> /* for memset */
#include "addition.h"

/* Default timeout can be changed using clnt_control() */
static struct timeval TIMEOUT = { 25, 0 };

int *
add_1(nums *argp, CLIENT *clnt)
{
    static int clnt_res;

    memset((char *)&clnt_res, 0, sizeof(clnt_res));
    if (clnt_call (clnt, add,
                   (xdrproc_t) xdr_nums, (caddr_t) argp,
                   (xdrproc_t) xdr_int, (caddr_t) &clnt_res,
                   TIMEOUT) != RPC_SUCCESS) {
        return (NULL);
    }
    return (&clnt_res);
}

/*addition_server.c*/

/*
 * This is sample code generated by rpcgen.
 * These are only templates and you can use them
 * as a guideline for developing your own functions.
 */

#include "addition.h"

int *
add_1_svc(nums *argp, struct svc_req *rqstp)
{
    static int result;

    result=argp->a+argp->b;
    return &result;
}

```

```

}
/*addition_svc.c*/

/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#include "addition.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>

#ifdef SIG_PF
#define SIG_PF void(*)(int)
#endif

static void
sum_prog_1(struct svc_req *rqstp, register SVCXPRT *transp)
{
    union {
        nums add_1_arg;
    } 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 add:

```

```

        _xdr_argument = (xdrproc_t) xdr_nums;
        _xdr_result = (xdrproc_t) xdr_int;
        local = (char (*)(char *, struct svc_req *)) add_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;
}

int
main (int argc, char **argv)
{
        register SVCXPRT *transp;

        pmap_unset (sum_prog, sum_ver);

        transp = svcudp_create(RPC_ANYSOCK);
        if (transp == NULL) {
                fprintf (stderr, "%s", "cannot create udp service.");
                exit(1);
        }

```



```

    if (!svc_register(transp, sum_prog, sum_ver, sum_prog_1, IPPROTO_UDP)) {
        fprintf (stderr, "%s", "unable to register (sum_prog, sum_ver, 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, sum_prog, sum_ver, sum_prog_1, IPPROTO_TCP)) {
        fprintf (stderr, "%s", "unable to register (sum_prog, sum_ver, tcp).");
        exit(1);
    }

    svc_run ();
    fprintf (stderr, "%s", "svc_run returned");
    exit (1);
    /* NOTREACHED */
}

```

/*addition_xdr.c*/

/*

* Please do not edit this file.

* It was generated using rpcgen.

*/

#include "addition.h"

bool_t

xdr_nums (XDR *xdrs, nums *objp)

{

 register int32_t *buf;

```

        if (!xdr_int (xdrs, &objp->a))
            return FALSE;
        if (!xdr_int (xdrs, &objp->b))
            return FALSE;
        return TRUE;
    }

/*addition.h*/

/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#ifndef _ADDITION_H_RPCGEN
#define _ADDITION_H_RPCGEN

#include <rpc/rpc.h>

#ifdef __cplusplus
extern "C" {
#endif

struct nums {
    int a;
    int b;
};

typedef struct nums nums;

#define sum_prog 0x20001080
#define sum_ver 1

#if defined(__STDC__) || defined(__cplusplus)
#define add 1
extern int * add_1(nums *, CLIENT *);
extern int * add_1_svc(nums *, struct svc_req *);

```

```
extern int sum_prog_1_freeresult (SVCXPRT *, xdrproc_t, caddr_t);
```

```
#else /* K&R C */
```

```
#define add 1
```

```
extern int * add_1();
```

```
extern int * add_1_svc();
```

```
extern int sum_prog_1_freeresult ();
```

```
#endif /* K&R C */
```

```
/* the xdr functions */
```

```
#if defined(__STDC__) || defined(__cplusplus)
```

```
extern bool_t xdr_nums (XDR *, nums*);
```

```
#else /* K&R C */
```

```
extern bool_t xdr_nums ();
```

```
#endif /* K&R C */
```

```
#ifdef __cplusplus
```

```
}
```

```
#endif
```

```
#endif /* !_ADDITION_H_RPCGEN */
```

```
/*Makefile.addition*/
```

```
# This is a template Makefile generated by rpcgen
```

```
# Parameters
```

```
CLIENT = addition_client
```

```
SERVER = addition_server
```

```
SOURCES_CLNT.c =
```

SOURCES_CLNT.h =

SOURCES_SVC.c =

SOURCES_SVC.h =

SOURCES.x = addition.x

TARGETS_SVC.c = addition_svc.c addition_server.c addition_xdr.c

TARGETS_CLNT.c = addition_clnt.c addition_client.c addition_xdr.c

TARGETS = addition.h addition_xdr.c addition_clnt.c addition_svc.c addition_client.c
addition_server.c

OBJECTS_CLNT = \$(SOURCES_CLNT.c:%.c=%.o) \$(TARGETS_CLNT.c:%.c=%.o)

OBJECTS_SVC = \$(SOURCES_SVC.c:%.c=%.o) \$(TARGETS_SVC.c:%.c=%.o)

Compiler flags

CFLAGS += -g

LDLIBS += -lnsl

RPCGENFLAGS =

Targets

all : \$(CLIENT) \$(SERVER)

\$(TARGETS) : \$(SOURCES.x)

rpcgen \$(RPCGENFLAGS) \$(SOURCES.x)

\$(OBJECTS_CLNT) : \$(SOURCES_CLNT.c) \$(SOURCES_CLNT.h) \$(TARGETS_CLNT.c)

\$(OBJECTS_SVC) : \$(SOURCES_SVC.c) \$(SOURCES_SVC.h) \$(TARGETS_SVC.c)

`$(CLIENT) : $(OBJECTS_CLNT)`

`$(LINK.c) -o $(CLIENT) $(OBJECTS_CLNT) $(LDLIBS)`

`$(SERVER) : $(OBJECTS_SVC)`

`$(LINK.c) -o $(SERVER) $(OBJECTS_SVC) $(LDLIBS)`

`clean:`

`$(RM) core $(TARGETS) $(OBJECTS_CLNT) $(OBJECTS_SVC) $(CLIENT)`
`$(SERVER)`

OUTPUT:

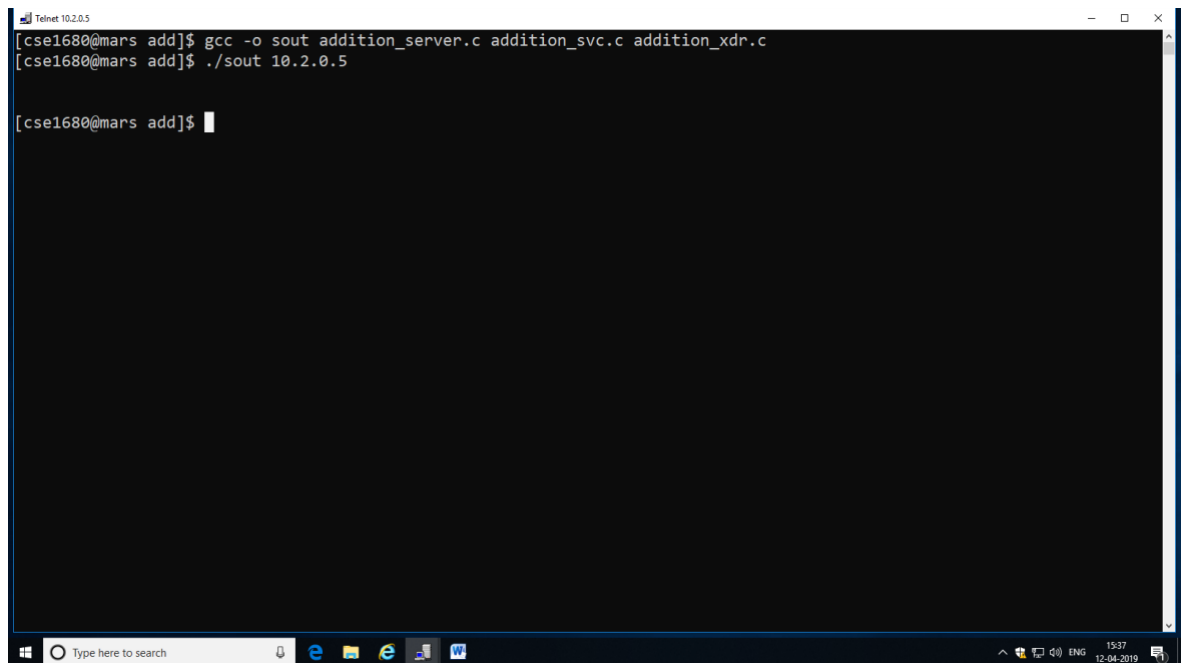
Compilation : rpcgen -a addition.x (generates the files)

gcc -o cout addition_client.c addition_clnt.c addition_xdr.c

```
gcc -o sout addition_server.c addition_svc.c addition_xdr.c
```

Execution : ./sout 10.2.0.5 (Execute in a separate terminal window first)
./cout 10.2.0.5 (Execute in a separate terminal window second)

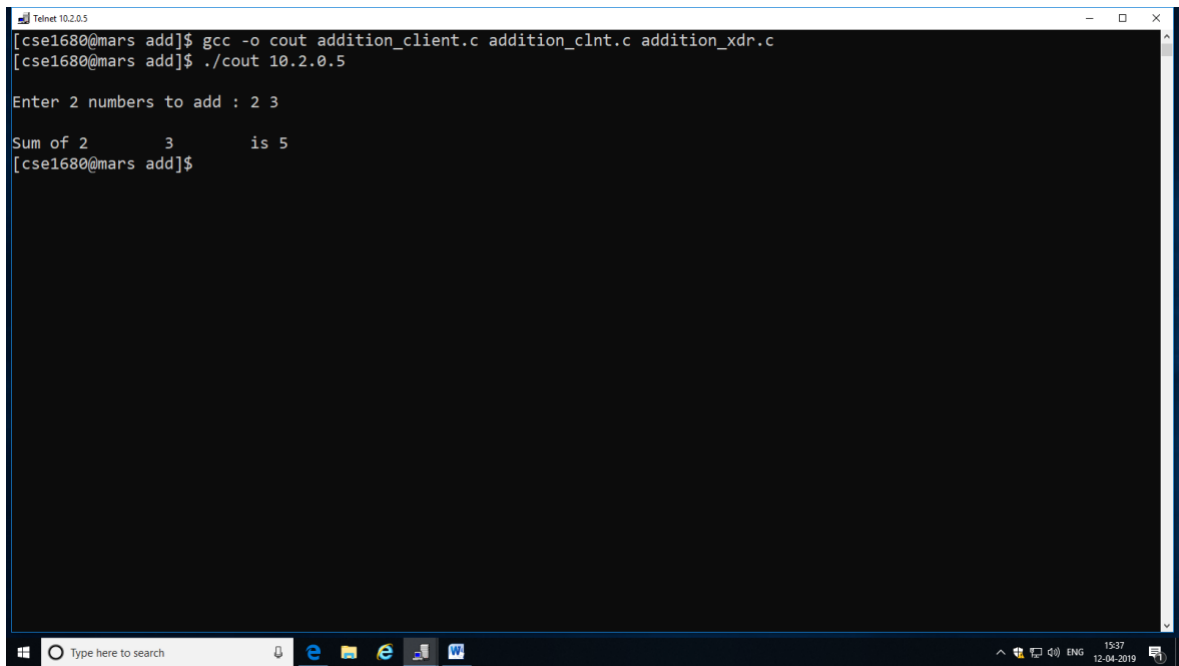
SERVER



The screenshot shows a Telnet session window titled 'Telnet 10.2.0.5'. The prompt is '[cse1680@mars add]\$'. The user has entered the command 'gcc -o sout addition_server.c addition_svc.c addition_xdr.c' and the output is '[cse1680@mars add]\$./sout 10.2.0.5'. The prompt is now '[cse1680@mars add]\$' with a cursor. The window has a standard Windows taskbar at the bottom with the search bar and system tray.

```
Telnet 10.2.0.5
[cse1680@mars add]$ gcc -o sout addition_server.c addition_svc.c addition_xdr.c
[cse1680@mars add]$ ./sout 10.2.0.5
[cse1680@mars add]$
```

CLIENT



The screenshot shows a Telnet window titled 'Telnet 10.2.0.5'. The terminal text is as follows:

```
[cse1680@mars add]$ gcc -o cout addition_client.c addition_clnt.c addition_xdr.c
[cse1680@mars add]$ ./cout 10.2.0.5

Enter 2 numbers to add : 2 3

Sum of 2      3      is 5
[cse1680@mars add]$
```

The window has a standard Windows taskbar at the bottom with a search bar and several application icons. The system tray on the right shows the date and time as 12-04-2019 15:37.

RPC(REMOTE PROCEDURE CALL) : SENDING/RESPONDING TO MESSAGE

Program:

```
/*message.x*/

program msg_prog{
version msg_ver{
    string message()=1;
    }=1;
    }=0x200000080;

/*message_client.c*/

/*
 * This is sample code generated by rpcgen.
 * These are only templates and you can use them
 * as a guideline for developing your own functions.
 */

#include "message.h"

void
msg_prog_1(char *host)
{
    CLIENT *clnt;
    char * *result_1;
    char *message_1_arg;

#ifdef  DEBUG
    clnt = clnt_create (host, msg_prog, msg_ver, "udp");
    if (clnt == NULL) {
        clnt_pcreateerror (host);
        exit (1);
    }
#endif  /* DEBUG */
    *result_1=(char*)malloc(sizeof(char)*30);
    result_1 = message_1((void*)&message_1_arg, clnt);
}
```



```

        if (result_1 == (char **) NULL) {
            clnt_perror (clnt, "call failed");
        }
        else
            printf("\nRPC Server Response is %s\n",*result_1);
#endif   DEBUG
        clnt_destroy (clnt);
#endif   /* DEBUG */
    }

```

```

int
main (int argc, char *argv[])
{
    char *host;

    if (argc < 2) {
        printf ("usage: %s server_host\n", argv[0]);
        exit (1);
    }
    host = argv[1];
    msg_prog_1 (host);
    exit (0);
}

```

```

/*message_clnt.c*/

```

```

/*

```

```

 * Please do not edit this file.
 * It was generated using rpcgen.
 */

```

```

#include <memory.h> /* for memset */
#include "message.h"

```

```

/* Default timeout can be changed using clnt_control() */
static struct timeval TIMEOUT = { 25, 0 };

```

```

char **
message_1(void *argp, CLIENT *clnt)
{
    static char *clnt_res;

    memset((char *)&clnt_res, 0, sizeof(clnt_res));
    if (clnt_call (clnt, message,
                    (xdrproc_t) xdr_void, (caddr_t) argp,
                    (xdrproc_t) xdr_wrapstring, (caddr_t) &clnt_res,
                    TIMEOUT) != RPC_SUCCESS) {
        return (NULL);
    }
    return (&clnt_res);
}
/*message_server.c*/

/*
 * This is sample code generated by rpcgen.
 * These are only templates and you can use them
 * as a guideline for developing your own functions.
 */

#include "message.h"
#include<string.h>

char **
message_1_svc(void *argp, struct svc_req *rqstp)
{
    static char * result;
    result=(char*)malloc(sizeof(char)*30);
    strcpy(result,"Hello-Response from RPC Server");
    return &result;
}
/*message_svc.c*/

/*
 * Please do not edit this file.

```

```
* It was generated using rpcgen.
```

```
*/
```

```
#include "message.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
```

```
msg_prog_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 message:
```

```
        _xdr_argument = (xdrproc_t) xdr_void;
```

```
        _xdr_result = (xdrproc_t) xdr_wrapstring;
```

```
        local = (char *(*)(char *, struct svc_req *)) message_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;
}

int
main (int argc, char **argv)
{
    register SVCXPRT *transp;

    pmap_unset (msg_prog, msg_ver);

    transp = svcudp_create(RPC_ANYSOCK);
    if (transp == NULL) {
        fprintf (stderr, "%s", "cannot create udp service.");
        exit(1);
    }
    if (!svc_register(transp, msg_prog, msg_ver, msg_prog_1, IPPROTO_UDP)) {
        fprintf (stderr, "%s", "unable to register (msg_prog, msg_ver, 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, msg_prog, msg_ver, msg_prog_1, IPPROTO_TCP)) {
        fprintf (stderr, "%s", "unable to register (msg_prog, msg_ver, tcp).");
        exit(1);
    }

    svc_run ();
    fprintf (stderr, "%s", "svc_run returned");
    exit (1);
    /* NOTREACHED */
}
/*message.h*/
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#ifndef _MESSAGE_H_RPCGEN
#define _MESSAGE_H_RPCGEN

#include <rpc/rpc.h>

#ifdef __cplusplus
extern "C" {
#endif

#define msg_prog 0x20000080
#define msg_ver 1

#ifdef __STDC__ || defined(__cplusplus)
#define message 1

```

```
extern char ** message_1(void *, CLIENT *);
extern char ** message_1_svc(void *, struct svc_req *);
extern int msg_prog_1_freeresult (SVCXPRT *, xdrproc_t, caddr_t);
```

```
#else /* K&R C */
#define message 1
extern char ** message_1();
extern char ** message_1_svc();
extern int msg_prog_1_freeresult ();
#endif /* K&R C */
```

```
#ifdef __cplusplus
}
#endif
```

```
#endif /* !_MESSAGE_H_RPCGEN */
```

```
/*Makefile.message*/
```

```
# This is a template Makefile generated by rpcgen
```

```
# Parameters
```

```
CLIENT = message_client
```

```
SERVER = message_server
```

```
SOURCES_CLNT.c =
```

```
SOURCES_CLNT.h =
```

```
SOURCES_SVC.c =
```

```
SOURCES_SVC.h =
```

```
SOURCES.x = message.x
```

TARGETS_SVC.c = message_svc.c message_server.c

TARGETS_CLNT.c = message_clnt.c message_client.c

TARGETS = message.h message_clnt.c message_svc.c message_client.c message_server.c

OBJECTS_CLNT = \$(SOURCES_CLNT.c:%.c=%.o) \$(TARGETS_CLNT.c:%.c=%.o)

OBJECTS_SVC = \$(SOURCES_SVC.c:%.c=%.o) \$(TARGETS_SVC.c:%.c=%.o)

Compiler flags

CFLAGS += -g

LDLIBS += -lnsl

RPCGENFLAGS =

Targets

all : \$(CLIENT) \$(SERVER)

\$(TARGETS) : \$(SOURCES.x)

rpcgen \$(RPCGENFLAGS) \$(SOURCES.x)

\$(OBJECTS_CLNT) : \$(SOURCES_CLNT.c) \$(SOURCES_CLNT.h) \$(TARGETS_CLNT.c)

\$(OBJECTS_SVC) : \$(SOURCES_SVC.c) \$(SOURCES_SVC.h) \$(TARGETS_SVC.c)

\$(CLIENT) : \$(OBJECTS_CLNT)

\$(LINK.c) -o \$(CLIENT) \$(OBJECTS_CLNT) \$(LDLIBS)

\$(SERVER) : \$(OBJECTS_SVC)

`$(LINK.c) -o $(SERVER) $(OBJECTS_SVC) $(LDLIBS)`

clean:

`$(RM) core $(TARGETS) $(OBJECTS_CLNT) $(OBJECTS_SVC) $(CLIENT)`
`$(SERVER)`

OUTPUT:

Compilation : *rpcgen -a message.x (generates the files)*
gcc -o cout message_client.c message_clnt.c

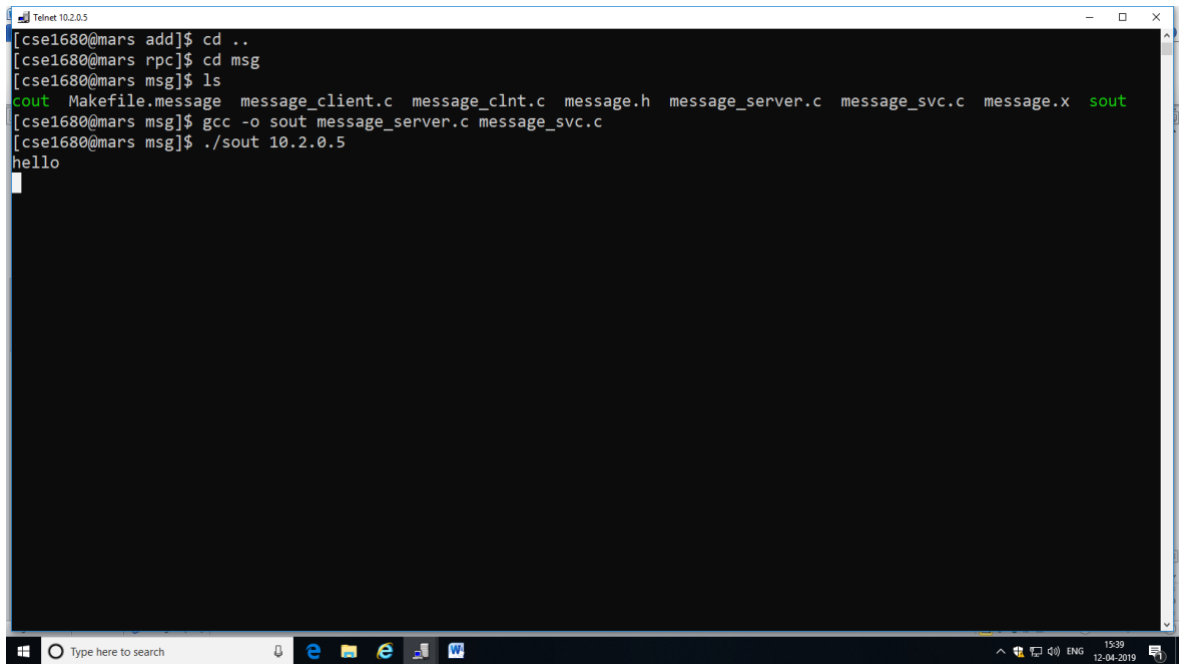
`gcc -o sout message_server.c message_svc.c`

Execution : `./sout 10.2.0.5` (Execute in a separate terminal window and first)

`./cout 10.2.0.5` (Execute in a separate terminal window and

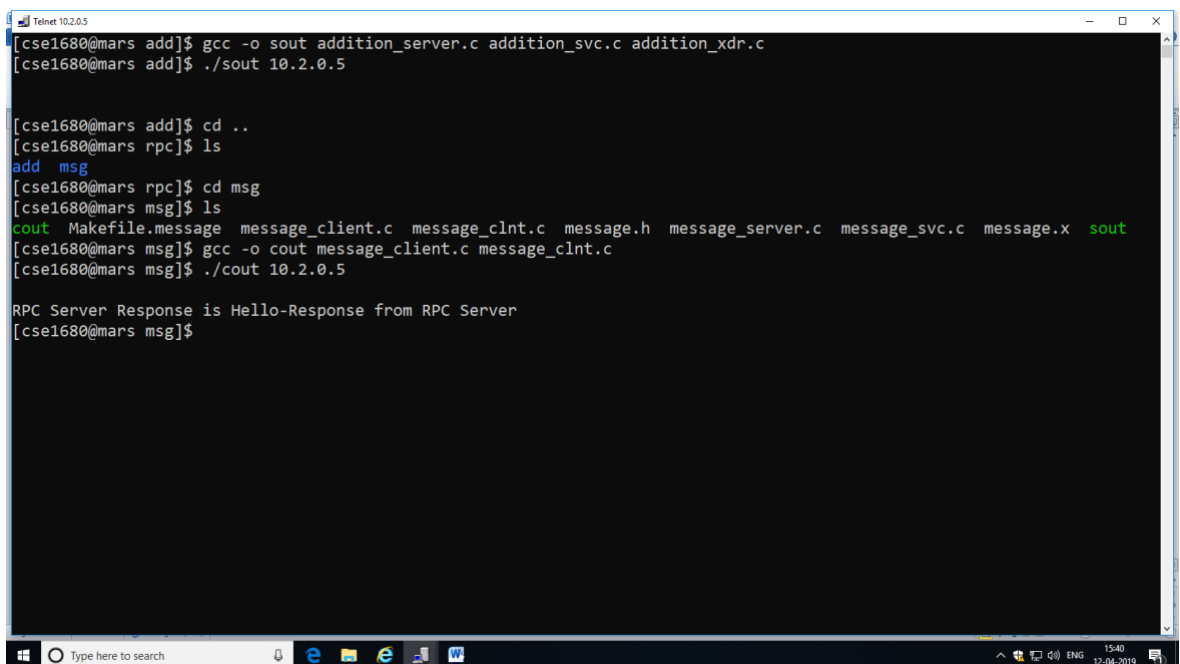
second)

SERVER



```
Telnet 10.2.0.5
[cse1680@mars add]$ cd ..
[cse1680@mars rpc]$ cd msg
[cse1680@mars msg]$ ls
cout Makefile.message message_client.c message_clnt.c message.h message_server.c message_svc.c message.x sout
[cse1680@mars msg]$ gcc -o sout message_server.c message_svc.c
[cse1680@mars msg]$ ./sout 10.2.0.5
hello
```

CLIENT



```
Telnet 10.2.0.5
[cse1680@mars add]$ gcc -o sout addition_server.c addition_svc.c addition_xdr.c
[cse1680@mars add]$ ./sout 10.2.0.5

[cse1680@mars add]$ cd ..
[cse1680@mars rpc]$ ls
add msg
[cse1680@mars rpc]$ cd msg
[cse1680@mars msg]$ ls
cout Makefile.message message_client.c message_clnt.c message.h message_server.c message_svc.c message.x sout
[cse1680@mars msg]$ gcc -o cout message_client.c message_clnt.c
[cse1680@mars msg]$ ./cout 10.2.0.5

RPC Server Response is Hello-Response from RPC Server
[cse1680@mars msg]$
```