M.V.S.R. ENGINEERING COLLEGE

(Affiliated to Osmania University & Recognized by AICTE)

Nadergul, RangaReddy Dist.

CERTIFICATE

Department of COMPUTER **SCIENCE & ENGINEERING**

Certified that this is a bonafide work of lab experiments carried out by Mr/Ms.________________________ bearing Roll.No.____________________ under the course of *Computer Networks* Laboratory prescribed by Osmania University for B.E. *III/IV Sem -2* of Computer Science & Engineering during the academic year 2016-2017.

Internal Examiner *External Examiner*

M.V.S.R. ENGINEERING COLLEGE

(Affiliated to Osmania University, Hyderabad)

Nadergul(P.O.), Hyderabad-501510

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Subject :	**Computer Networks Lab**	Acad. Year :	**2016-17**
Class :	**BE III /IV Sem - II**	Section :	**I/II/III**

INDEX

S.No.	Name of the Program	Date		Pages	
		Experiment	Submission	From	To
1	Write a program to illustrate connection oriented iterative Server				
2	Write a program to illustrate connection less iterative Server				
3	Write a program to illustrate connection oriented concurrent Server				
4	Write a program to illustrate connection less concurrent Server				
5	WAP to implement getsockname() and getpeername() of server and client				
6	WAP to implement time server (user defined)				
7	Write a program to implement time using predefined port (13)				
8	WAP to illustrate advanced socket options				
9	WAP to illustrate advanced system calls readv() & writev()				
10	WAP to implement asynchronous I/O				
11	Build a concurrent Multithreaded File Transfer Server. Use separate Threads to allow the server to handle multiple clients concurrently.				
12	Write a program to implement REMOTE PROCEDURE CALL				

INTRODUCTION

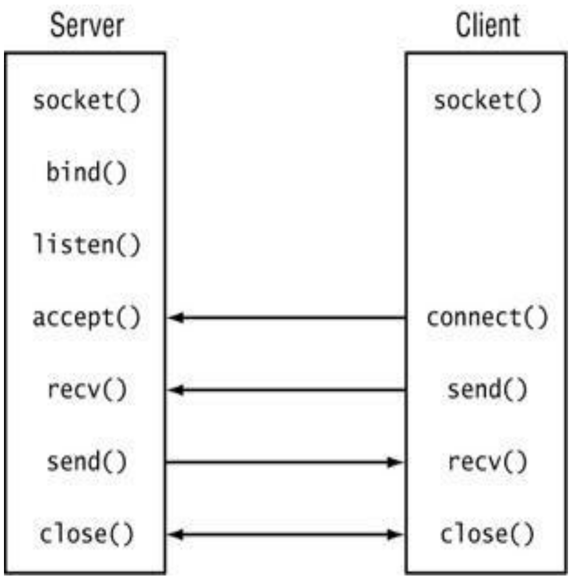
Socket: An interface between an application process and transport layer.

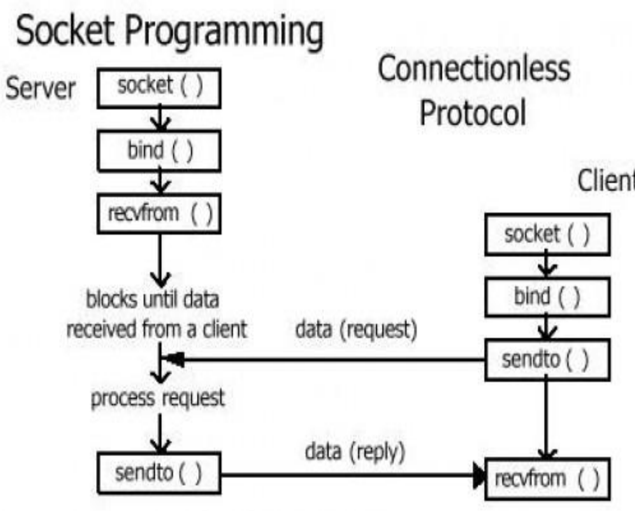
Types of Internet Sockets

● Stream Sockets (SOCK_STREAM) – Connection oriented – Rely on TCP to provide reliable two-way connected communication

● Datagram Sockets (SOCK_DGRAM) – Rely on UDP – Connection is unreliable.

Socket Life Cycle:





a) Connection-Oriented b) Connection-Less

Methods:

socket() -- Get the file descriptor

● int socket(int domain, int type, int protocol);

– domain should be set to PF_INET – type can be SOCK_STREAM or SOCK_DGRAM

– set protocol to 0 to have socket choose the correct protocol based on type

– socket() returns a socket descriptor for use in later system calls or -1 on error

struct sockaddr: Holds socket address information for many types of sockets

● **struct sockaddr_in:** A parallel structure that makes it easy to reference elements of the socket address

● sin_port and sin_addr must be in Network Byte Order

To convert binary IP to string: inet_noa()

bind() - Used to associate a socket with a port on the local machine.

int bind(int sockfd, struct sockaddr *my_addr, int addrlen).

connect() - Connects to a remote host

int connect(int sockfd, struct sockaddr *serv_addr, int addrlen)

listen() - Waits for incoming connections

int listen(int sockfd, int backlog);

accept() - gets the pending connection on the port you are listening on

int accept(int sockfd, void *addr, int *addrlen);

send() and recv() - The two functions are for communicating over stream sockets or connected datagram sockets.

int send(int sockfd, const void *msg, int len, int flags);

int recv(int sockfd, void *buf, int len, int flags);

sendto() and recvfrom() - DGRAM style

int sendto(int sockfd, const void *msg, int len, int flags, const struct sockaddr *to, int tolen);

int recvfrom(int sockfd, void *buf, int len, int flags, struct sockaddr *from, int *fromlen);

1 Write a program to illustrate connection oriented iterative Server

<u>Server Program:</u>

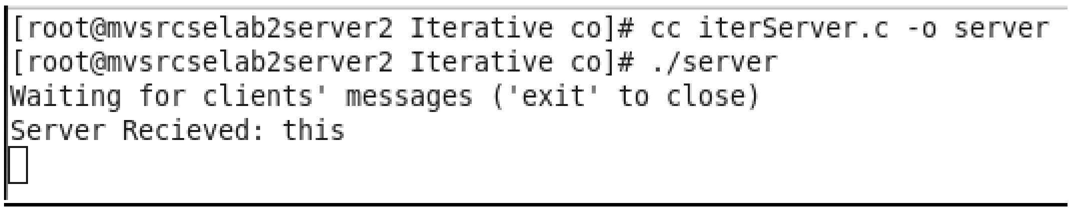
```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>

int main()
{
    int sockfd,newsockfd,clilen;
    struct sockaddr_in serv_addr,cli_addr;
    char a[50];
    sockfd=socket(AF_INET,SOCK_STREAM,0);        //create a socket for communication
                                                  //AF_INET for IPv4 addresses
            //SOCK_STREAM provides reliable, two-way, connection-based byte streams
            //0 for default protocol for the socket
    if(sockfd<0)
    {
        printf("socket failed\n");
        exit(0);
    }
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
                        //INADDR_ANY - Accept    connections from any address (client)
                        //change address to the client IPv4 Address to accept only on client
    if(serv_addr.sin_addr.s_addr < 0)
    {
                printf("Invalid IP address: Unable to decode\n");
                exit(0);
            }
    serv_addr.sin_port = htons(4568);
    if(bind(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr))<0)
    {
                    printf("Bind failed\n");
                    exit(1);
    }
    if(listen(sockfd,5)<0)
    {
                    printf("Listen failed\n");
                    exit(0);
    }
    clilen=sizeof(cli_addr);
    printf("Waiting for clients' messages (\'exit\' to close)\n");
    while(1)
    {
            newsockfd=accept(sockfd,(struct sockaddr *)&cli_addr, (socklen_t *) &clilen);
                    memset(a, 0, sizeof(a));
                    read(newsockfd,a,50);
                    printf("Server Recieved: %s\n",a);
                    write(newsockfd,a,50);
                    close(newsockfd);
                    if(!strcmp(a,"exit"))
                    {
                            printf("Exiting server\n");
                            break;
                    }
            }
    return 0; }
```

Client Program:

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

int main()
{
        int sockfd;
        struct sockaddr_in serv_addr;
        char a[50],a1[50];
        sockfd=socket(AF_INET,SOCK_STREAM,0);
        if(sockfd<0)
        {
                printf("socket failed\n");
                exit(0);
        }
        serv_addr.sin_family=AF_INET;
        serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
                        //Change address to server's IPv4 address, don’t change if on same machine
        serv_addr.sin_port=htons(4568);
        if(connect(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr))<0)
        {
                printf("Connection failed\n");
                exit(0);
        }
        memset(a, 0, sizeof(a));
        printf("Enter the msg :\n");
        scanf("%s",a);
        write(sockfd,a,50);
        read(sockfd,a1,50);
        printf("Client Received the msg: %s\n",a1);
        close(sockfd);
        if(!strcmp(a1,"exit"))
                printf("Closing client program\n");
        return 0;
}
```

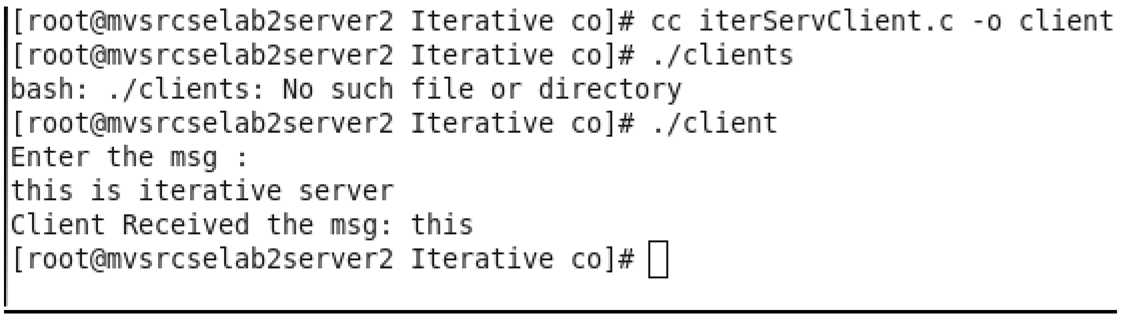
Output:

Server:



```
[root@mvsrcselab2server2 Iterative co]# cc iterServer.c -o server
[root@mvsrcselab2server2 Iterative co]# ./server
Waiting for clients' messages ('exit' to close)
Server Recieved: this
```

Client:



```
[root@mvsrcselab2server2 Iterative co]# cc iterServClient.c -o client
[root@mvsrcselab2server2 Iterative co]# ./clients
bash: ./clients: No such file or directory
[root@mvsrcselab2server2 Iterative co]# ./client
Enter the msg :
this is iterative server
Client Received the msg: this
[root@mvsrcselab2server2 Iterative co]#
```

2 Write a program to illustrate connection less iterative Server

Server Program:

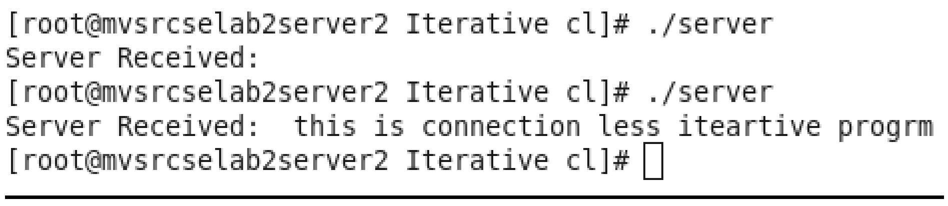
```
#include<stdlib.h>
#include<stdio.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
int main()
{
        int sockfd,newsockfd,clilen;
        struct sockaddr_in serv_addr,cli_addr;
        char msg[50];
        sockfd=socket(AF_INET,SOCK_DGRAM,0);
        if(sockfd<0)
        {
        printf("\n Socket Failed");
        exit(0);
        }
        serv_addr.sin_family=AF_INET;
        serv_addr.sin_addr.s_addr=htonl(INADDR_ANY);
        serv_addr.sin_port=htons(3456);
        if(bind(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr))<0)
        {
           printf("\n Bind Failed");
           exit(0);
         }
          clilen=sizeof(cli_addr);
          recvfrom(sockfd,msg,80,0,(struct sockaddr *)&cli_addr,&clilen);
          printf("Server Received: %s",msg);
          sendto(sockfd,msg,80,0,(struct sockaddr *) &cli_addr,clilen);

        write(sockfd);
        close(sockfd);
         }
```

Client:

```
#include<stdlib.h>
#include<stdio.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<string.h>
#include<sys/socket.h>
int main()
{
        int sockfd,n,clilen,servlen;
        struct sockaddr_in cli_addr,serv_addr;
        char msg[50],msg1[50];
        sockfd=socket(AF_INET,SOCK_DGRAM,0);
        if(sockfd<0)
        {
         printf("\n Sokcet Failed");
         exit(0);
        }
        serv_addr.sin_family=AF_INET;
        serv_addr.sin_addr.s_addr=inet_addr("127.0.0.1");
        serv_addr.sin_port=htons(3456);
       cli_addr.sin_family=AF_INET;
      cli_addr.sin_addr.s_addr=htonl(INADDR_ANY);
      cli_addr.sin_port=htons(0);
      if(bind(sockfd,(struct sockaddr*)&cli_addr,sizeof(cli_addr))<0)
       {
        printf("Client cantt bind");
        exit(1);
       }
       printf("Enter Strin");
       fgets(msg,50,stdin);
        if(sendto(sockfd,msg,50,0,(struct sockaddr *)&serv_addr,sizeof(serv_addr))<0)
        {
          printf("Client send to error");
          exit(0);
        }
        servlen=sizeof(serv_addr);
        n=recvfrom(sockfd,msg1,50,0,(struct sockaddr *)&serv_addr,&servlen);
        if(n<0)
        {
                printf("Recv error");
                exit(1);
        }
        else
        {
                printf("\n  Client received msg : %s",msg1);
        }
        close(sockfd);
}
```

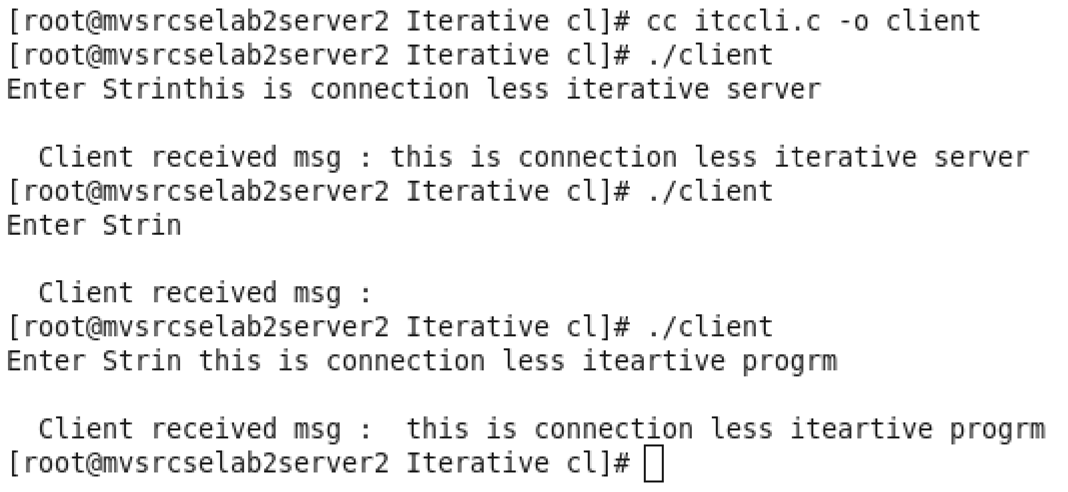
Output:

Server:



```
[root@mvsrcselab2server2 Iterative cl]# ./server
Server Received:
[root@mvsrcselab2server2 Iterative cl]# ./server
Server Received:  this is connection less iteartive progrm
[root@mvsrcselab2server2 Iterative cl]# 
```

Client:



```
[root@mvsrcselab2server2 Iterative cl]# cc itccli.c -o client
[root@mvsrcselab2server2 Iterative cl]# ./client
Enter Strinthis is connection less iterative server

  Client received msg : this is connection less iterative server
[root@mvsrcselab2server2 Iterative cl]# ./client
Enter Strin

  Client received msg :
[root@mvsrcselab2server2 Iterative cl]# ./client
Enter Strin this is connection less iteartive progrm

  Client received msg :  this is connection less iteartive progrm
[root@mvsrcselab2server2 Iterative cl]# 
```

3 Write a program to illustrate connection oriented concurrent Server

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

int main()
{
    int sockfd,newsockfd,clilen, pid;
    struct sockaddr_in serv_addr,cli_addr;
    char a[50];
    sockfd=socket(AF_INET,SOCK_STREAM,0);          //create a socket for communication
                                                   //AF_INET for IPv4 addresses
            //SOCK_STREAM provides reliable, two-way, connection-based byte streams
            //0 for default protocol for the socket
    if(sockfd<0)
    {
        printf("socket failed\n");
        exit(0);
    }
    serv_addr.sin_family = AF_INET;
              //Set address to accept connection from any client with any IP address
    serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
             //INADDR_ANY - Accept connections from any address (client)
                  //change address to the client IPv4 Address to accept only on client
    if(serv_addr.sin_addr.s_addr < 0)
    {
                  printf("Invalid IP address: Unable to decode\n");
                  exit(0);
```

```
                }
        serv_addr.sin_port = htons(3100);
        if(bind(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr))<0)
        {
                        printf("Bind failed\n");
                        exit(1);
        }
        if(listen(sockfd,5)<0)
        {
                        printf("Listen failed\n");
                        exit(0);
        }
        clilen=sizeof(cli_addr);
        printf("Waiting for clients\n");
           while(1)
             {
                    //Accept connection from clients
                    newsockfd=accept(sockfd,(struct sockaddr *)&cli_addr, (socklen_t *) &clilen);
                        pid = fork();    //create a new process to serve each request
                        if(pid==0)
                        {
                              //Child process serving requests will execute this block
                        while(1)
                              {
                                    memset(a, 0, sizeof(a));
                                    read(newsockfd,a,50);//Read message from client
                        //Also print the process id of the instance to check if concurrency works
                        printf("Instance : %d \n\tServer Recieved: %s\n",(int)getpid(),a);
                        write(newsockfd,a,50);          //Return the same message to the client
                                    if(!strcmp(a, "exit"))
                                    {
                                    printf("Closing connection : Instance %d\n", (int)getpid());
                                    break;
```

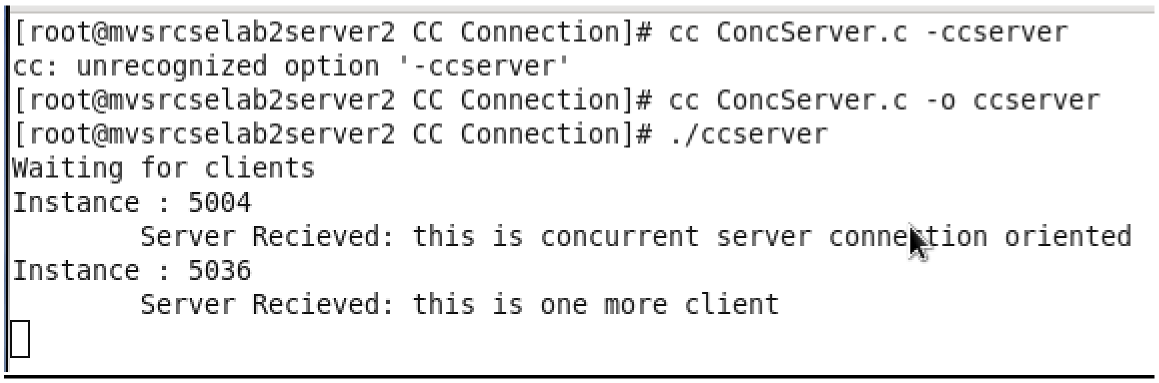
```
                                        }
                                }
                                close(newsockfd);      //Close the connection
                                break;      //Break the loop to end the process (serving process)
                        }
                }
    return 0;
}
```

Client Program:

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

int main()
{
        int sockfd;
        struct sockaddr_in serv_addr;
        char a[50],a1[50], *pos;
        sockfd=socket(AF_INET,SOCK_STREAM,0);
        if(sockfd<0)
        {
                printf("socket failed\n");
                exit(0);
        }
        serv_addr.sin_family=AF_INET;
        serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
                        //Change address to server's IPv4 address, dont change if on same machine
        serv_addr.sin_port=htons(3100);
        if(connect(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr))<0)
        {
                printf("Connection failed\n");
                exit(0);
        }
        memset(a, 0, sizeof(a));
        while(1)
        {
                printf("Enter the msg :\n");
                fgets(a,sizeof(a), stdin);      // read entire line into a[]
                        //This blocks removes trailing newline character (if present) left form fgets
                if( (pos = strchr(a, '\n'))!= NULL)
                        *pos = '\0';
                write(sockfd,a,50);
                read(sockfd,a1,50);
                printf("Client Received the msg: %s\n",a1);
                if(!strcmp(a, "exit"))
                {
                        printf("Closing connection\n");
                        break;
                }
        }
        close(sockfd);
        if(!strcmp(a1,"exit"))
                printf("Closing client program\n");
        return 0;
}
```

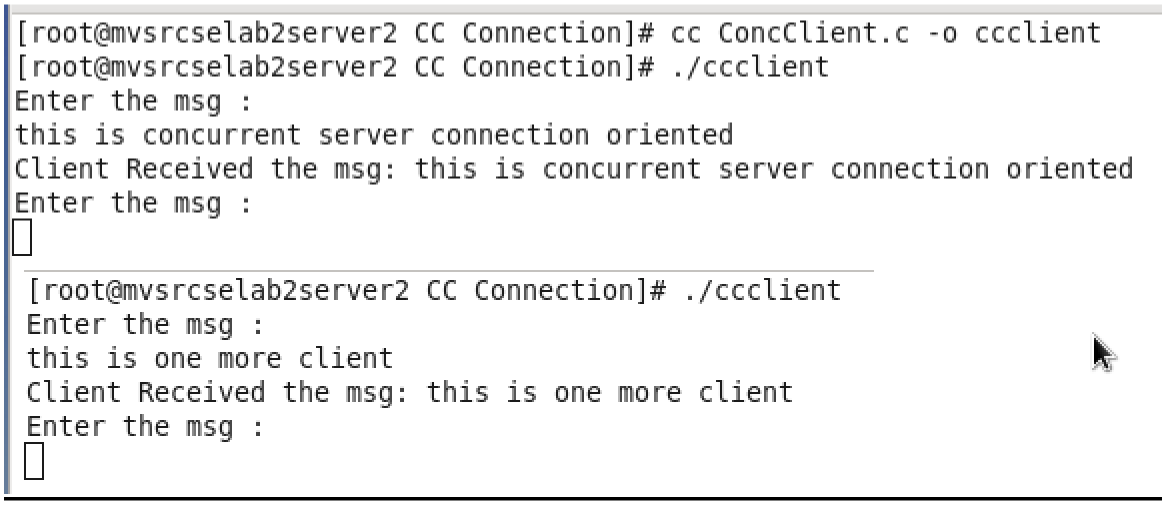
Output:

Server:



```
[root@mvsrcselab2server2 CC Connection]# cc ConcServer.c -ccserver
cc: unrecognized option '-ccserver'
[root@mvsrcselab2server2 CC Connection]# cc ConcServer.c -o ccserver
[root@mvsrcselab2server2 CC Connection]# ./ccserver
Waiting for clients
Instance : 5004
        Server Recieved: this is concurrent server connection oriented
Instance : 5036
        Server Recieved: this is one more client
```

Client:



```
[root@mvsrcselab2server2 CC Connection]# cc ConcClient.c -o ccclient
[root@mvsrcselab2server2 CC Connection]# ./ccclient
Enter the msg :
this is concurrent server connection oriented
Client Received the msg: this is concurrent server connection oriented
Enter the msg :
```

```
[root@mvsrcselab2server2 CC Connection]# ./ccclient
Enter the msg :
this is one more client
Client Received the msg: this is one more client
Enter the msg :
```

4 Write a program to illustrate connection less concurrent Server

Server Program:

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

int main()
{
        int sockfd, n,clilen, pid;
        struct sockaddr_in serv_addr,cli_addr;
        char a[50];
        sockfd=socket(AF_INET,SOCK_DGRAM,0);       //create a socket for communication
                                                   //AF_INET for IPv4 addresses
                        //Communication with Datagrams (UDP - Connectionless, non-reliable)
                        //0 for default protocol for the socket
        if(sockfd<0)
        {
                printf("socket failed\n");
                exit(0);
        }
        serv_addr.sin_family = AF_INET;
                    //Set address to accept connection from any client with any IP address
        serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
                    //INADDR_ANY - Accept connections from any address (client)
                  //change address to the client IPv4 Address to accept only on client
        if(serv_addr.sin_addr.s_addr < 0)
        {
                        printf("Invalid IP address: Unable to decode\n");
                        exit(0);
                }
        serv_addr.sin_port = htons(3100);
        if(bind(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr))<0)
        {
                        printf("Bind failed\n");
                        exit(1);
        }
        clilen=sizeof(cli_addr);
        printf("Waiting for clients\n");
        while(1)
        {
                        memset(a, 0, sizeof(a));
//Read messages from clients (without connection) into a[]; type "man 2 recvfrom" in terminal
for details
n = recvfrom(sockfd, a, 50, 0, (struct sockaddr *)&cli_addr, (socklen_t *) &clilen);
if(n>0)
{
pid = fork();   //create a new process to serve each request
if(pid==0)
{
        //Child process serving requests will execute this block
        //read(newsockfd,a,50);       //Read message from client
        //Also print the process id of the instance to check if concurrency works
printf("Instance : %d \n\tServer Recieved: %s\n",(int)getpid(),a);
if( sendto(sockfd, a, 50, 0, (struct sockaddr *)&cli_addr, (socklen_t) clilen) < 0)
        //Return the same message to the client
        {
        printf("UDP sending failed\nExiting... \n");
        close(sockfd);
        exit(1);
                }
        close(sockfd); //Close the connection
```

```
            break;  //Break the loop to end the process (serving process)
                        }
                    }
                    else
                    {
                        printf("UDP receiving failed\nExiting... \n");
                        close(sockfd);
                        exit(1);
                    }
                }
        return 0;
}
```

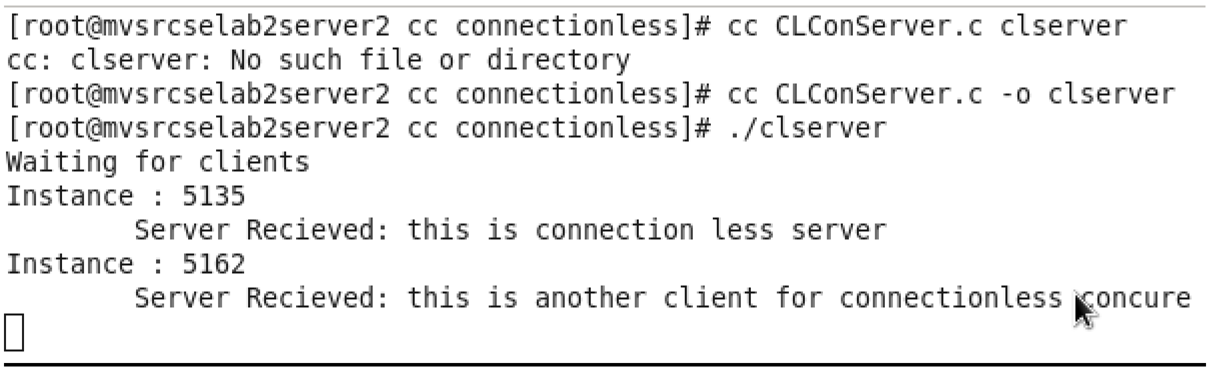
Client Program:

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

int main()
{
    int sockfd, servlen;
    struct sockaddr_in serv_addr;
    char a[50],a1[50], *pos;
    servlen = sizeof(serv_addr);
    sockfd=socket(AF_INET,SOCK_DGRAM,0);
    if(sockfd<0)
    {
        printf("socket failed\n");
```

```
			exit(0);
		}
		serv_addr.sin_family=AF_INET;
		serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
				//Change address to server's IPv4 address, dont change if on same machine
		serv_addr.sin_port=htons(3100);
		memset(a, 0, sizeof(a));
		printf("Enter the msg :\n");
		fgets(a,sizeof(a), stdin);		// read entire line into a[]
			//This blocks removes trailing newline character (if present) left form fgets
		if( (pos = strchr(a, '\n'))!= NULL)
			*pos = '\0';
		if(sendto(sockfd, a, 50, 0, (struct sockaddr *)&serv_addr, (socklen_t) servlen) < 0)
		{
			printf("UDP client : Message sending failed\nExiting...");
			close(sockfd);
			exit(1);
		}
		if(recvfrom(sockfd, a1, 50, 0, (struct sockaddr *)&serv_addr,(socklen_t *) &servlen) < 0)
		{
			printf("UDP client : Message receiveing failed\nExiting...");
			close(sockfd);
			exit(1);
		}
		printf("Client Received the msg: %s\n",a1);
			close(sockfd);
		return 0;
}
```

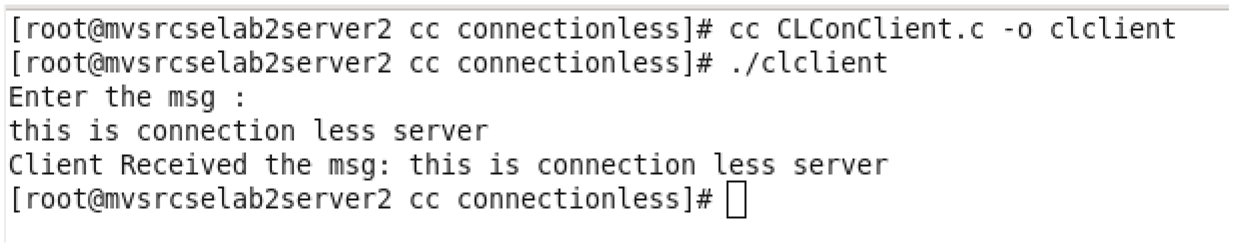
Output:

Server:

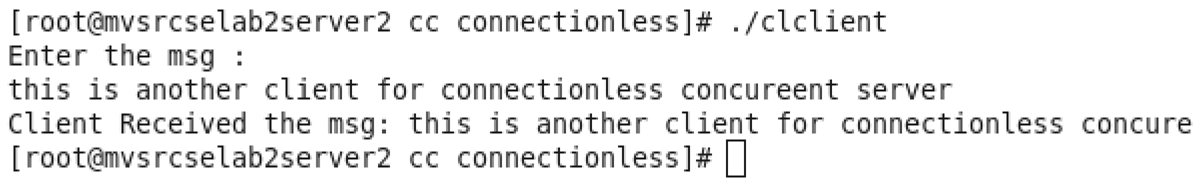


```
[root@mvsrcselab2server2 cc connectionless]# cc CLConServer.c clserver
cc: clserver: No such file or directory
[root@mvsrcselab2server2 cc connectionless]# cc CLConServer.c -o clserver
[root@mvsrcselab2server2 cc connectionless]# ./clserver
Waiting for clients
Instance : 5135
        Server Recieved: this is connection less server
Instance : 5162
        Server Recieved: this is another client for connectionless concure
```

Client:



```
[root@mvsrcselab2server2 cc connectionless]# cc CLConClient.c -o clclient
[root@mvsrcselab2server2 cc connectionless]# ./clclient
Enter the msg :
this is connection less server
Client Received the msg: this is connection less server
[root@mvsrcselab2server2 cc connectionless]#
```



```
[root@mvsrcselab2server2 cc connectionless]# ./clclient
Enter the msg :
this is another client for connectionless concureent server
Client Received the msg: this is another client for connectionless concure
[root@mvsrcselab2server2 cc connectionless]#
```

5. WAP to implement getsockname () and getpeername () of server and client

Server Program:

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <stdlib.h>
int main(int argc, char* argv[])
{
        int sockfd,newsockfd,clilen,servlen;
        struct sockaddr_in serv_addr,cli_addr, temp;
        char a[50];
        sockfd=socket(AF_INET,SOCK_STREAM,0);
        serv_addr.sin_family=AF_INET;
        if(argc == 1)
        {
            serv_addr.sin_addr.s_addr=htonl(INADDR_ANY);
                //INADDR_ANY to accept connections from any host
                //assign inet_addr("11.3.4.1
") to accept connection only from specific host
        }
        else
        {
                serv_addr.sin_addr.s_addr= inet_addr(argv[1]);
                if(serv_addr.sin_addr.s_addr == -1)
                {
                printf("\nInvalid IP address for client\n");
                 printf("Usage :\t%s [IPADDR]\n\nIPADDR\t:\tIP Address of client in numbers-
and-dots (octet) notation\n", argv[0]);
                printf("\nIf IPADDR is not specified accepts connections from any
hosts\n\nExiting program...\n");
```

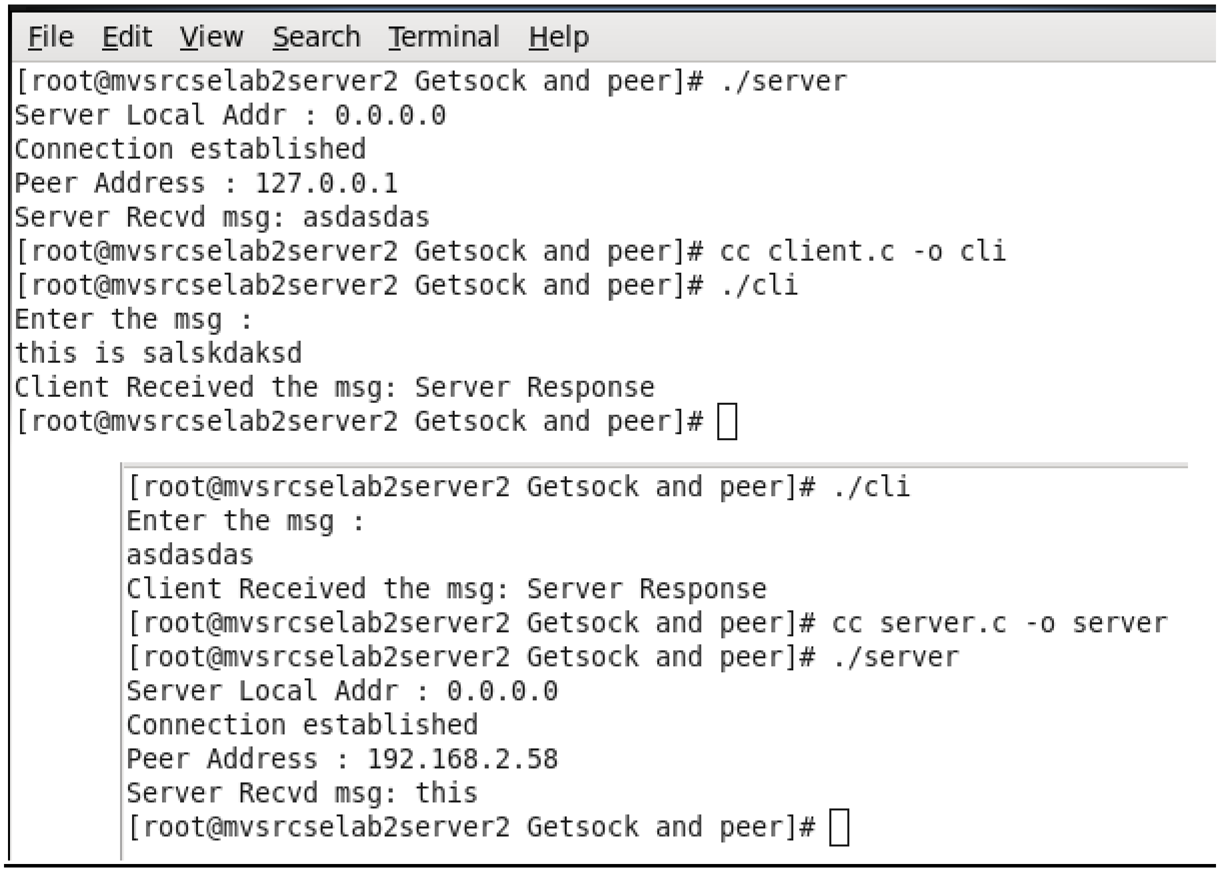
```
                        close(sockfd);
                }
                exit(1);
        }
        serv_addr.sin_port=htons(3100);
        bind(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr));
        servlen=sizeof(serv_addr);
                //get scoket name into 'temp'
        getsockname(sockfd,(struct sockaddr*) &temp, (socklen_t *)&servlen);
                //Print bound IP address from 'temp'
        printf("Server Local Addr : %s\n", inet_ntoa(temp.sin_addr));
                //Listen on socket for connections
        listen(sockfd,5);
        clilen=sizeof(temp);
                //Accept connection
        newsockfd=accept(sockfd,(struct sockaddr*)&cli_addr, (socklen_t *)&clilen);
        if(newsockfd<0)
                printf("Connection not established\n");
        else
        {
                printf("Connection established\n");
                read(newsockfd,a,30);
                                //Set the peer's IP address into 'temp'
                getpeername(newsockfd,(struct sockaddr*)&temp, (socklen_t *)&clilen);
                                //Print IP address from 'temp'
                printf("Peer Address : %s\n",inet_ntoa(temp.sin_addr));
                printf("Server Recvd msg: %s\n",a);
                write(newsockfd,"Server Response",50);
                close(newsockfd);
        }
        return 0;
}
```

Client Program:

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

int main(int argc, char * argv[])
{
	int sockfd;
	struct sockaddr_in serv_addr;
	char a[50],a1[50];
	sockfd=socket(AF_INET,SOCK_STREAM,0);
	if(sockfd<0)
	{
		printf("socket failed\n");
		exit(0);
	}
	serv_addr.sin_family=AF_INET;
	if(argc == 1)
		serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
	else
	{
		//Change address to server's IPv4 address from input argument
		//Type in 'ifconfig' to check host's IP address
		serv_addr.sin_addr.s_addr = inet_addr(argv[1]);
		if(serv_addr.sin_addr.s_addr == -1)
		{
			printf("\nInvalid IP address for server\n");
```

```
                        printf("Usage :\t%s IPADDR\n\nIPADDR\t:\tIP Address of server in
numbers-and-dots (octet) notation\n", argv[0]);
                        printf("\nIf IPADDR is not specified looks for server in
localhost\n\nExiting program...\n");
                        close(sockfd);
                }
                exit(1);
        }
        serv_addr.sin_port=htons(3100);
        if(connect(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr))<0)
        {
                printf("Connection failed\n");
                exit(0);
        }
        memset(a, 0, sizeof(a));
        printf("Enter the msg :\n");
        scanf("%s",a);
        write(sockfd,a,50);
        read(sockfd,a1,50);
        printf("Client Received the msg: %s\n",a1);
        close(sockfd);
        if(!strcmp(a1,"exit"))
                printf("Closing client program\n");
        return 0;
}
```

Output:



```
[root@mvsrcselab2server2 Getsock and peer]# ./server
Server Local Addr : 0.0.0.0
Connection established
Peer Address : 127.0.0.1
Server Recvd msg: asdasdas
[root@mvsrcselab2server2 Getsock and peer]# cc client.c -o cli
[root@mvsrcselab2server2 Getsock and peer]# ./cli
Enter the msg :
this is salskdaksd
Client Received the msg: Server Response
[root@mvsrcselab2server2 Getsock and peer]# 
```

```
[root@mvsrcselab2server2 Getsock and peer]# ./cli
Enter the msg :
asdasdas
Client Received the msg: Server Response
[root@mvsrcselab2server2 Getsock and peer]# cc server.c -o server
[root@mvsrcselab2server2 Getsock and peer]# ./server
Server Local Addr : 0.0.0.0
Connection established
Peer Address : 192.168.2.58
Server Recvd msg: this
[root@mvsrcselab2server2 Getsock and peer]# 
```

6. WAP to implement time server (user defined)

Server Program:

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#include <time.h>
int main()
{
   int sockfd,newsockfd,clilen;
     struct sockaddr_in serv_addr,cli_addr;
     char a[50];
     time_t now;
     struct tm present;
     sockfd=socket(AF_INET,SOCK_STREAM,0);     //create a socket for communication
                                               //AF_INET for IPv4 addresses
              //SOCK_STREAM provides reliable, two-way, connection-based byte streams
              //0 for default protocol for the socket
     if(sockfd<0)
     {
          printf("socket failed\n");
          exit(1);
     }
     serv_addr.sin_family = AF_INET;
     serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
                    //INADDR_ANY - Accept connections from any address (client)
               //change address to the client IPv4 Address to accept only on client
     if(serv_addr.sin_addr.s_addr < 0)
     {
                    printf("Invalid IP address: Unable to decode\n");
```

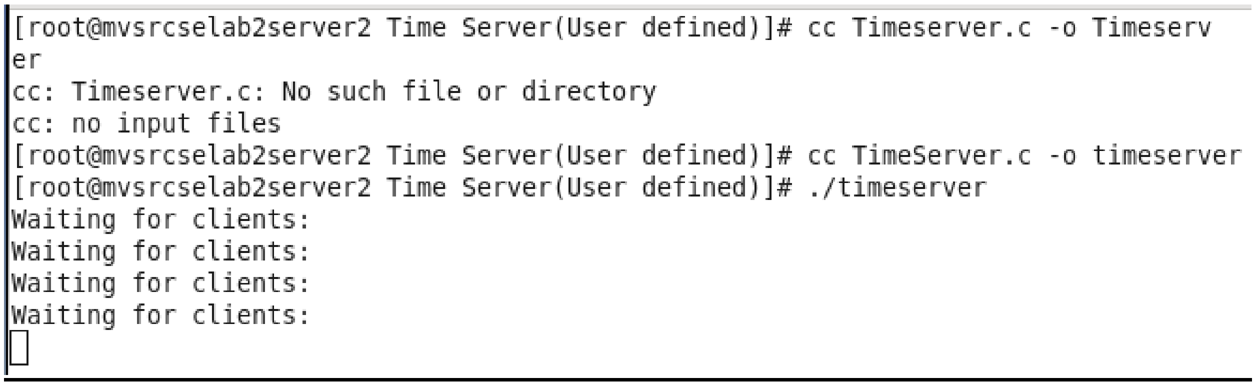
```
                        exit(1);
                }
        serv_addr.sin_port = htons(3100);
        if(bind(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr))<0)
        {
                        printf("Bind failed\n");
                        exit(1);
        }
        if(listen(sockfd,5)<0)
        {
                        printf("Listen failed\n");
                        exit(1);
        }
        clilen=sizeof(cli_addr);
        while(1)
        {
                printf("Waiting for clients: \n");
                newsockfd=accept(sockfd,(struct sockaddr *)&cli_addr, (socklen_t *) &clilen);
                memset(a, 0, sizeof(a));
                read(newsockfd,a,50);
                time(&now);   //get the present time in seconds - see 'man 2 time' on terminal
                present = *localtime(&now);
            //localtime breaks time_t variable 'now' into 'stuct tm' and returns the pointer to the newly
//created structure. The structure is copied into 'present'\
            sprintf(a,"Time: %d-%d-%d %d:%d:%d\n", present.tm_year + 1900, present.tm_mon +
            1, present.tm_mday, present.tm_hour, present.tm_min, present.tm_sec);
                //The value from the structure 'present' is encoded as date and time
                //The formatted date and time (as string) is copied into 'char *a' using sprintf
            write(newsockfd,a,50);          //Write (transmit) a into socket
            close(newsockfd);
                }
            return 0;
}
```

Client Program:

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
int main()
{
        int sockfd;
        struct sockaddr_in serv_addr;
        char a[50],a1[50];
        sockfd=socket(AF_INET,SOCK_STREAM,0);
        if(sockfd<0)
        {
                printf("socket failed\n");
                exit(0);
        }
        serv_addr.sin_family=AF_INET;
        serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
                        //Change address to server's IPv4 address, dont change if on same machine
        serv_addr.sin_port=htons(3100);
                if(connect(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr))<0)
        {
                printf("Connection failed\n");
                exit(0);
        }
        memset(a, 0, sizeof(a));
        //printf("Press enter to get time\n");
        //scanf("%s",a);
        write(sockfd,a,50);
        read(sockfd,a1,50);
        printf("Client Received %s\n",a1);
        close(sockfd);
        if(!strcmp(a1,"exit"))
        printf("Closing client program\n");
        return 0;
}
```

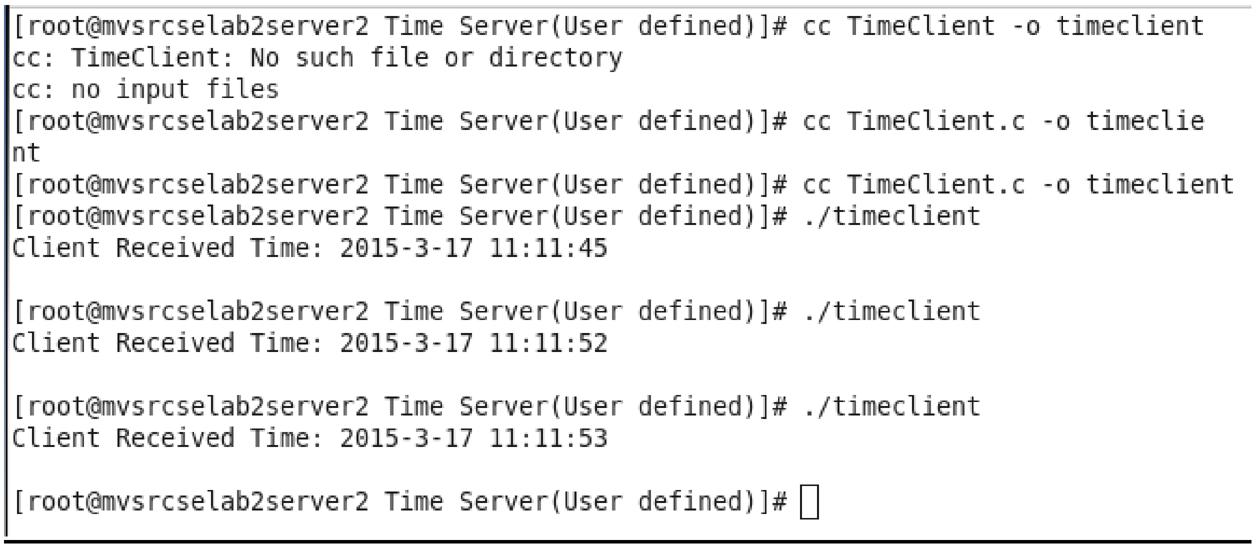
Output:

Server:



```
[root@mvsrcselab2server2 Time Server(User defined)]# cc Timeserver.c -o Timeserv
er
cc: Timeserver.c: No such file or directory
cc: no input files
[root@mvsrcselab2server2 Time Server(User defined)]# cc TimeServer.c -o timeserver
[root@mvsrcselab2server2 Time Server(User defined)]# ./timeserver
Waiting for clients:
Waiting for clients:
Waiting for clients:
Waiting for clients:
```

Client:

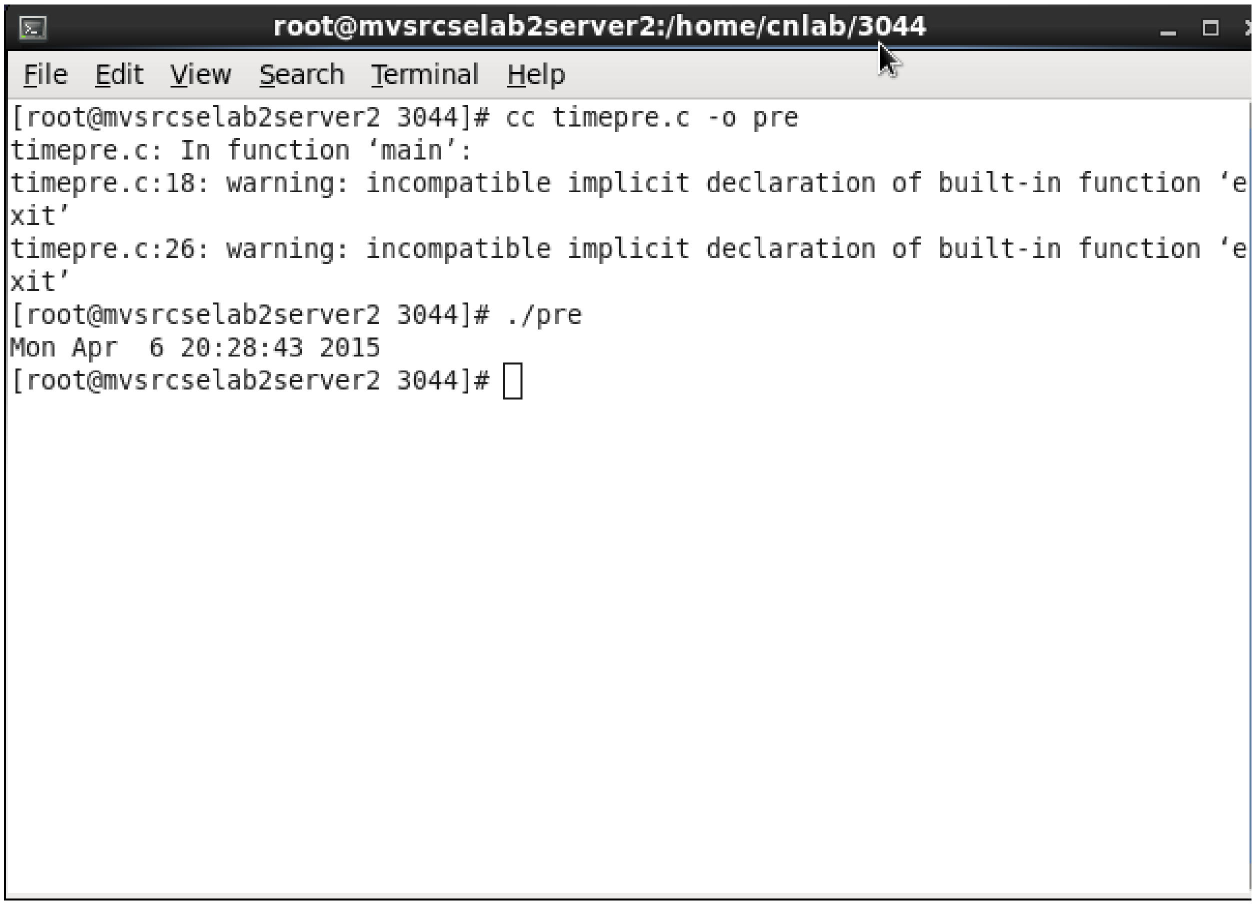


```
[root@mvsrcselab2server2 Time Server(User defined)]# cc TimeClient -o timeclient
cc: TimeClient: No such file or directory
cc: no input files
[root@mvsrcselab2server2 Time Server(User defined)]# cc TimeClient.c -o timeclie
nt
[root@mvsrcselab2server2 Time Server(User defined)]# cc TimeClient.c -o timeclient
[root@mvsrcselab2server2 Time Server(User defined)]# ./timeclient
Client Received Time: 2015-3-17 11:11:45

[root@mvsrcselab2server2 Time Server(User defined)]# ./timeclient
Client Received Time: 2015-3-17 11:11:52

[root@mvsrcselab2server2 Time Server(User defined)]# ./timeclient
Client Received Time: 2015-3-17 11:11:53

[root@mvsrcselab2server2 Time Server(User defined)]#
```

7 Write a program to implement time using predefined port (13)

```
#include<stdio.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<sys/types.h>
#include<time.h>
#include<string.h>
int main()
{
int sockfd;
struct sockaddr_in serv_addr;
time_t  now;
char timestr[100];
char a[50],a1[50];
sockfd=socket(AF_INET,SOCK_STREAM,0);
if(sockfd<0)
{
printf("\n Socket Failed");
exit(0);
}
serv_addr.sin_family=AF_INET;
serv_addr.sin_addr.s_addr=inet_addr("127.0.0.1");
serv_addr.sin_port=htons(13);
if(connect(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr))<0)
{
printf("\n Connection Failed");
exit(0);
}
time(&now);
sprintf(timestr,"%s",ctime(&now));
printf("%s",timestr);
close(sockfd);
}
```

Output:

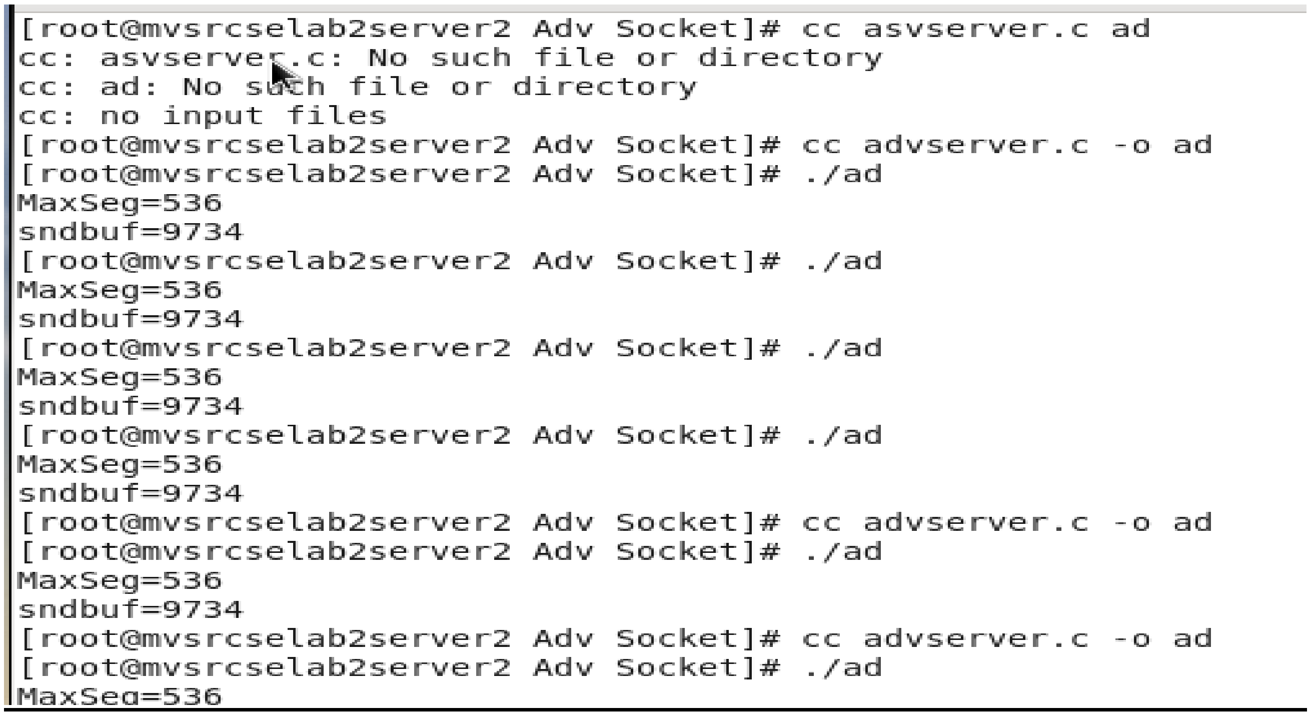

root@mvsrcselab2server2:/home/cnlab/3044
File Edit View Search Terminal Help

```
[root@mvsrcselab2server2 3044]# cc timepre.c -o pre
timepre.c: In function ‘main’:
timepre.c:18: warning: incompatible implicit declaration of built-in function ‘e
xit’
timepre.c:26: warning: incompatible implicit declaration of built-in function ‘e
xit’
[root@mvsrcselab2server2 3044]# ./pre
Mon Apr  6 20:28:43 2015
[root@mvsrcselab2server2 3044]# 
```

8. WAP to illustrate getsockopt() and setsockopt()

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netinet/tcp.h>
int main()
{
	int sockfd,MAXSEG,sndbuf,optlen;
	sockfd=socket(AF_INET,SOCK_STREAM,0);
	optlen=sizeof(MAXSEG);
	getsockopt(sockfd, IPPROTO_TCP, TCP_MAXSEG, &MAXSEG,(socklen_t *) &optlen);
	printf("MaxSeg=%d\n",MAXSEG);
	sndbuf=4869;
	setsockopt(sockfd, SOL_SOCKET, SO_SNDBUF, (char *)&sndbuf, sizeof(sndbuf));
	optlen=sizeof(sndbuf);
	getsockopt(sockfd, SOL_SOCKET, SO_SNDBUF, (char *)&sndbuf,(socklen_t *)
&optlen);
	printf("sndbuf=%d\n",sndbuf);
	return 0;
}
```

Output:



```
[root@mvsrcselab2server2 Adv Socket]# cc asvserver.c ad
cc: asvserver.c: No such file or directory
cc: ad: No such file or directory
cc: no input files
[root@mvsrcselab2server2 Adv Socket]# cc advserver.c -o ad
[root@mvsrcselab2server2 Adv Socket]# ./ad
MaxSeg=536
sndbuf=9734
[root@mvsrcselab2server2 Adv Socket]# ./ad
MaxSeg=536
sndbuf=9734
[root@mvsrcselab2server2 Adv Socket]# ./ad
MaxSeg=536
sndbuf=9734
[root@mvsrcselab2server2 Adv Socket]# ./ad
MaxSeg=536
sndbuf=9734
[root@mvsrcselab2server2 Adv Socket]# cc advserver.c -o ad
[root@mvsrcselab2server2 Adv Socket]# ./ad
MaxSeg=536
sndbuf=9734
[root@mvsrcselab2server2 Adv Socket]# cc advserver.c -o ad
[root@mvsrcselab2server2 Adv Socket]# ./ad
MaxSeg=536
```

9. WAP to illustrate advanced system calls readv() & writev()

Server Program:

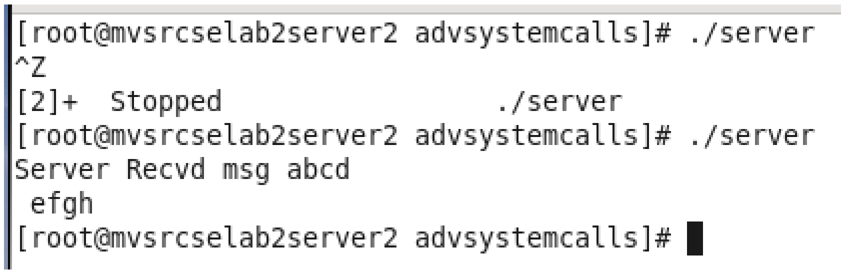
```
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/uio.h>
#include <sys/types.h>
#include <arpa/inet.h>
#include <unistd.h>

int main()
{
        int sockfd,newsockfd,clilen;
        struct sockaddr_in serv_addr,cli_addr;
        struct iovec iov[2];
        char b[50],b1[50];
        sockfd=socket(AF_INET,SOCK_STREAM,0);
        serv_addr.sin_family=AF_INET;
        serv_addr.sin_addr.s_addr=inet_addr("127.0.0.1");
        serv_addr.sin_port=htons(3100);
        bind(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr));
        listen(sockfd,1);
        clilen=sizeof(cli_addr);
        newsockfd=accept(sockfd,(struct sockaddr *)&cli_addr, (socklen_t *) &clilen);
        if(newsockfd<0)
        {
                printf("\n Connection Failed");
        }
        iov[0].iov_base=b;
        iov[0].iov_len=50;
        iov[1].iov_base=b1;
        iov[1].iov_len=50;
        readv(newsockfd,&iov[0],2);
        printf("Server Recvd msg %s \n %s\n",b,b1);
        writev(newsockfd,&iov[0],2);
        close(newsockfd);
        return 0;
}
```

Client:

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

int main()
{
int sockfd;
char a[50],a1[50],b[50],b1[50];
struct sockaddr_in serv_addr;
struct iovec iov[2];
sockfd=socket(AF_INET,SOCK_STREAM,0);
serv_addr.sin_family=AF_INET;
serv_addr.sin_addr.s_addr=inet_addr("127.0.0.1");
 serv_addr.sin_port=htons(3100);
 connect(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr));
 printf("Enter 1st msg :\n");
 scanf("%s",a);
 printf("Enter 2nd msg :\n");
 scanf("%s",a1);
 iov[0].iov_base=a;
 iov[0].iov_len=50;
 iov[1].iov_base=a1;
 iov[1].iov_len=50;
 writev(sockfd,&iov[0],2);
 iov[0].iov_base=b;
 iov[0].iov_len=50;
 iov[1].iov_base=b1;
 iov[1].iov_len=50;
 readv(sockfd,&iov[0],2);
 printf("\n Client Recvd msg  %s %s",b,b1);
 close(sockfd);
 return 0;
}
```

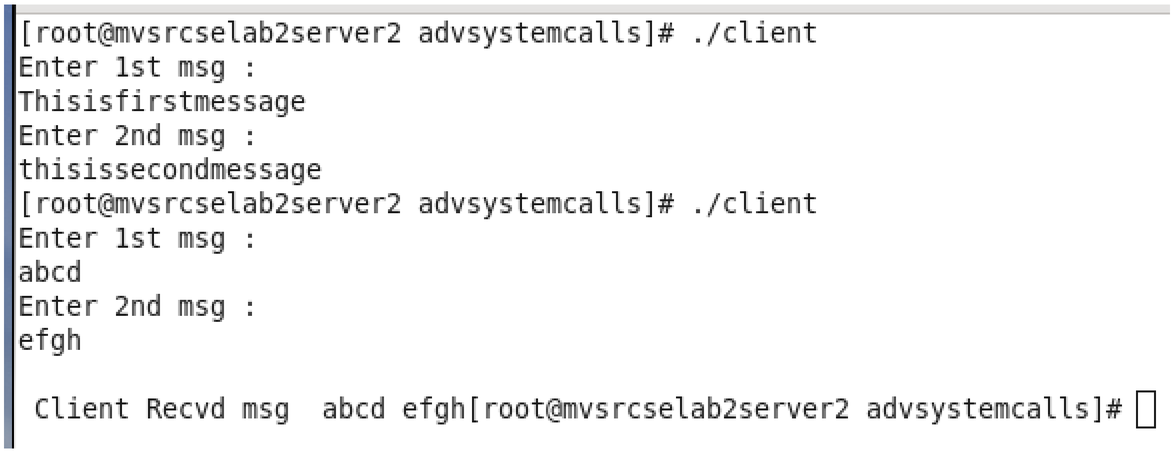
<u>Output:</u>

Server:



```
[root@mvsrcselab2server2 advsystemcalls]# ./server
^Z
[2]+  Stopped                 ./server
[root@mvsrcselab2server2 advsystemcalls]# ./server
Server Recvd msg abcd
 efgh
[root@mvsrcselab2server2 advsystemcalls]# 
```

Client:



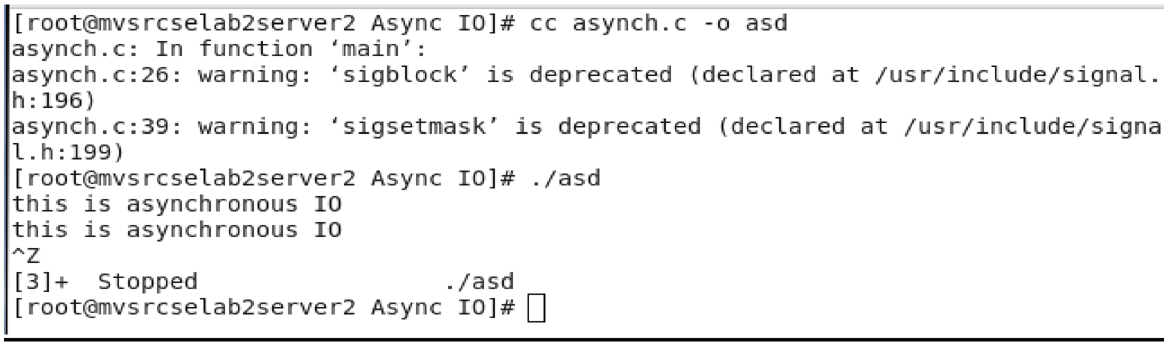
```
[root@mvsrcselab2server2 advsystemcalls]# ./client
Enter 1st msg :
Thisisfirstmessage
Enter 2nd msg :
thisissecondmessage
[root@mvsrcselab2server2 advsystemcalls]# ./client
Enter 1st msg :
abcd
Enter 2nd msg :
efgh

 Client Recvd msg  abcd efgh[root@mvsrcselab2server2 advsystemcalls]# 
```

10. WAP to implement asynchronous I/O

```
#include <fcntl.h>
#include <stdio.h>
#include <sys/types.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <stdlib.h>
#include <signal.h>
#define BUFFSIZE 1024
int sigflag;
int main()
{
	int n;
	char buff[BUFFSIZE];
	void sigio_func();
	signal(SIGIO,(void *)sigio_func);
	if(fcntl(0,F_SETOWN,getpid())<0)
		printf(" F_SETOWN Error ");
	if(fcntl(0,F_SETFL,FASYNC)<0)
		printf(" F_SETFL Error");
	for( ; ; )
	{
		sigblock(sigmask(SIGIO));
		while(sigflag==0)
			sigpause(0);
		if((n=read(0,buff,BUFFSIZE))>0)
		{
			if(write(1,buff,n)!=n)
				printf("Write Error");
		}
		else if(n<0)
			printf("Read Error");
		else if(n==0)
```

```
                exit(0);
            sigflag=0;
            sigsetmask(0);
        }
        return 0;
}
void sigio_func()
{
        sigflag=1;
        return;
}
```

Output:



```
[root@mvsrcselab2server2 Async IO]# cc asynch.c -o asd
asynch.c: In function ‘main’:
asynch.c:26: warning: ‘sigblock’ is deprecated (declared at /usr/include/signal.
h:196)
asynch.c:39: warning: ‘sigsetmask’ is deprecated (declared at /usr/include/signa
l.h:199)
[root@mvsrcselab2server2 Async IO]# ./asd
this is asynchronous IO
this is asynchronous IO
^Z
[3]+  Stopped                 ./asd
[root@mvsrcselab2server2 Async IO]# 
```

11. Build a concurrent Multithreaded File Transfer Server. Use separate Threads to allow the server to handle multiple clients concurrently.

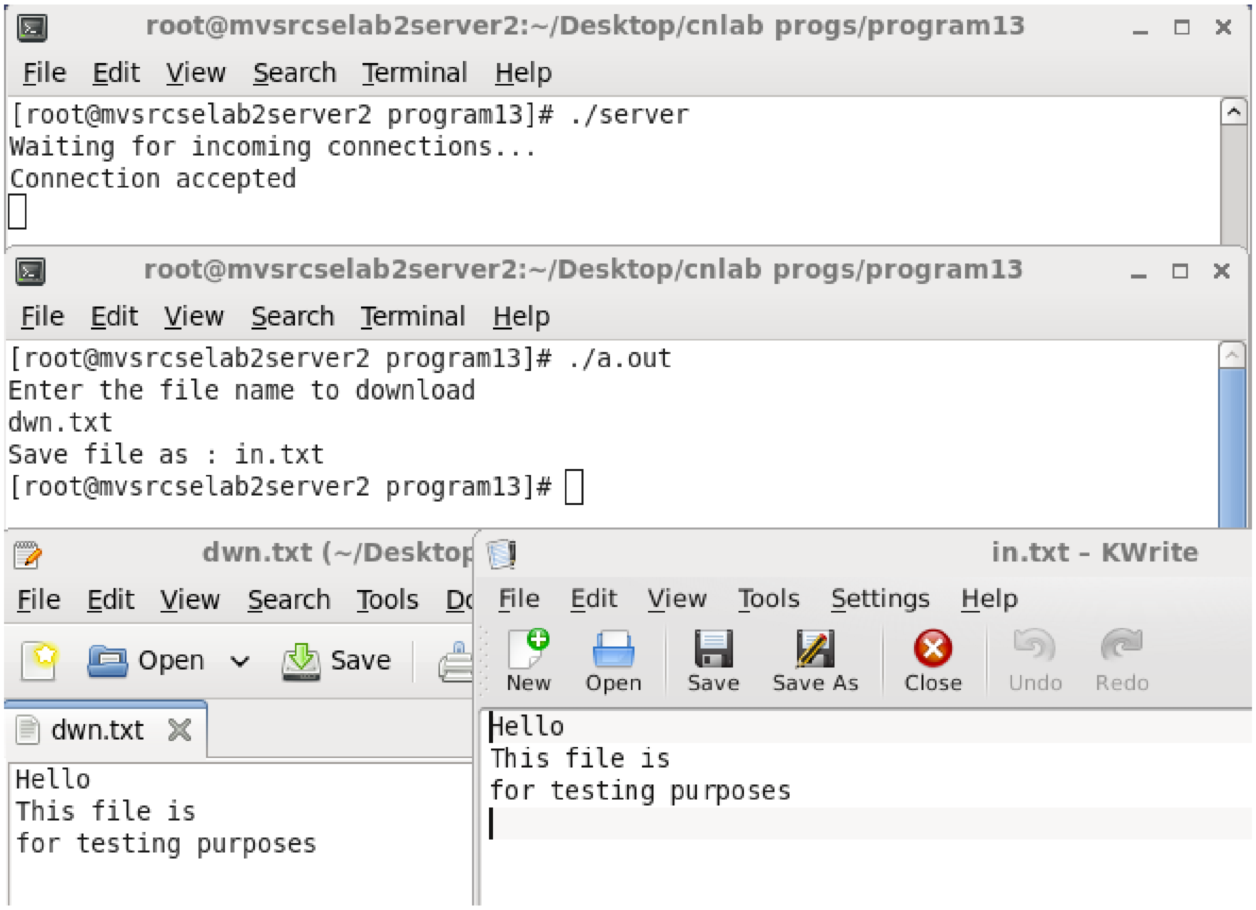
```
#include<stdio.h>
#include<string.h>    //strlen
#include<stdlib.h>    //strlen
#include<sys/socket.h>
#include<arpa/inet.h> //inet_addr
#include<unistd.h>    //write
#include<pthread.h> //for threading , link with lpthread
                              //the thread function
void *connection_handler(void *);
int clients;
 int main()
{
    int socket_desc , client_sock , c , *new_sock;
    struct sockaddr_in server , client;
                 //Create socket
    socket_desc = socket(AF_INET , SOCK_STREAM , 0);
    if (socket_desc == -1)
    {
        printf("Could not create socket");
    }
          //puts("Socket created");
         //Prepare the sockaddr_in structure
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons( 3100 );
            //Bind
    if( bind(socket_desc,(struct sockaddr *)&server , sizeof(server)) < 0)
    {
            //print the error message
        perror("bind failed. Error");
        return 1;
    }
          //puts("bind done");
         //Listen
    listen(socket_desc , 3);
         //Accept incoming connection
    puts("Waiting for incoming connections...");
    c = sizeof(struct sockaddr_in);
   while( (client_sock = accept(socket_desc, (struct sockaddr *)&client, (socklen_t*)&c)) )
    {
        puts("Connection accepted");
         pthread_t sniffer_thread;
        new_sock = malloc(1);
        *new_sock = client_sock;
                //Create a new thread to handle the request
        if( pthread_create( &sniffer_thread , NULL ,  connection_handler , (void*) new_sock) < 0)
        {
            perror("could not create thread");
            return 1;
        }
            //puts("Handler assigned");
    }
         if (client_sock < 0)
    {
        perror("accept failed");
        return 1;
    }
         return 0;
}
void *connection_handler(void *socket_desc)
{
    //Get the socket descriptor
    int sock = *(int*)socket_desc;
    int read_size, fchar;
    char file[20], ffile[20], content[1000];
    FILE *fp;
            memset(file, 0, sizeof(file));
                       //Read file name from client to transmit
    if( (read_size = recv(sock , file , sizeof(file), 0)) < 0 )
    {
                       perror("Problem reading filename\n");
                       pthread_exit(NULL);
        }
                //prepend './' to filename
        sprintf(ffile, "./%s", file);
        if((fp = fopen(ffile, "r")) == NULL)
        {
                char error[50];
                sprintf(error,"ERROR: Server cannot locate file \"%s\"",ffile);
                perror(error);
                write(sock, error, sizeof(error));
                pthread_exit(NULL);
        }
        printf("Transmitting file : %s...",ffile);
        memset(content, 0, 1000);
        while(1)
        {
                fchar = fread(content, 1, sizeof(content), fp);
                if(fchar > 0)
                {
                        //puts(content);
                        write(sock, content, fchar);
                }
                if(fchar < 1000)
                {
                        if(feof(fp))
                                break;
                }
        }
        fclose(fp);
        //Free the socket pointer
    free(socket_desc);
    pthread_exit(NULL);
}
```

Client:

```
//File downloading client
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>

int main()
{
        int sockfd, fchar;
        struct sockaddr_in serv_addr;
        char a[50],a1[100], *pos, saveas[20], content[1000];
        FILE *fp;
        sockfd=socket(AF_INET,SOCK_STREAM,0);
        if(sockfd<0)
        {
                printf("socket failed\n");
                exit(0);
        }
        serv_addr.sin_family=AF_INET;
        serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
                //Change address to server's IPv4 address, dont change if on same machine
        serv_addr.sin_port=htons(3100);
        if(connect(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr))<0)
        {
                printf("Connection failed\n");
                exit(0);
        }
        memset(a, 0, sizeof(a));
        memset(a1, 0, sizeof(a));
        printf("Enter the file name to download\n");
        fgets(a,sizeof(a), stdin);          // read entire line into a[]
                //This blocks remove trailing newline character (if present) left from fgets
        if( (pos = strchr(a, '\n'))!= NULL)
                *pos = '\0';
                printf("Save file as : ");
        fgets(saveas,sizeof(saveas), stdin);    // read entire line into a[]
                //This blocks remove trailing newline character (if present) left from fgets
        if( (pos = strchr(saveas, '\n'))!= NULL)
                *pos = '\0';
                if((fp = fopen(saveas, "w"))== NULL)
        {
                perror("Cannot create file\n");
                return 1;
        }
        write(sockfd,a,50);
        memset(content, 0, sizeof(content));
        while((fchar = read(sockfd, content, sizeof(content))) > 0)
        {
                if(fchar < 1000)
                {
                        //puts(content);
                        fwrite(content, 1, fchar, fp);
                        break;
                }
                      //puts(content);
                fwrite(content, 1, fchar, fp);
                memset(content, 0, sizeof(content));
                printf("%s",content);
        }
            //fputc(EOF, fp);
        fclose(fp);
```

```
        close(sockfd);
        return 0;
}
```

Output:


root@mvsrcselab2server2:~/Desktop/cnlab progs/program13
File Edit View Search Terminal Help

```
[root@mvsrcselab2server2 program13]# ./server
Waiting for incoming connections...
Connection accepted
```

root@mvsrcselab2server2:~/Desktop/cnlab progs/program13
File Edit View Search Terminal Help

```
[root@mvsrcselab2server2 program13]# ./a.out
Enter the file name to download
dwn.txt
Save file as : in.txt
[root@mvsrcselab2server2 program13]#
```

dwn.txt (~/Desktop
File Edit View Search Tools Do
Open Save
dwn.txt

```
Hello
This file is
for testing purposes
```

in.txt - KWrite
File Edit View Tools Settings Help
New Open Save Save As Close Undo Redo

```
Hello
This file is
for testing purposes
```

12. Write a program to implement REMOTE PROCEDURE CALL

```
/* SI.X */
struct record
{  int p;
 int n;
 int r;  };
 program SI_PROG
 {
   version SI_VERS
    {
       long si(record)=1;
    }=1;
}=0x21234567;
```

Compilation: rpcgen si.x

Server Program:

```
#include"si.h"
#include<stdio.h>
#include<rpc/rpc.h>
#include<sys/types.h>
long *si_1(arecord,c1)
        struct record *arecord;
        CLIENT *c1;
{
        static long result;
        result=(arecord->p*arecord->n*arecord->r)/100;
        return(&result);  }
```

Output:

```
cc rpserv.c -o rpserv si_svc.c si_xdr.c
./rpserv
```

Client Program

```
#include"si.h"
#include<stdio.h>
#include<rpc/rpc.h>
#include<sys/types.h>
main(int argc,char *argv[])
{
        CLIENT *c1;
        char *server;
        long *lresu;
        struct record arecord;
        arecord.p=1000;
        arecord.n=2;
        arecord.r=4;
        server=argv[1];
        c1=clnt_create(server,SI_PROG,SI_VERS,"UDP");
        lresu=si_1(&arecord,c1);
        printf("si=%ld",*(lresu));
        clnt_destroy(c1);
}
```

Output:

```
cc rpcl.c -o rpcl si_clnt.c si_xdr.c
    ./rpcl  127.0.0.1
    SI=80
```