Faculty of Engineering and Technology

Department of Computer Science and Engineering

Lab Manual

SUBJECT CODE / TITLE: 18CSC302J/ Computer Networks

Prepared By

Dr. S. Muruganandham, Deptment of Information Technology &

Dr. C. Jothi Kumar , Deptment of Computer Science & Engineering

SRM Institute of Science and Technology

SRM Nagar, Kattankulathur-603203

18CSC302J/ Computer Networks

LIST OF EXPERIMENTS

1. Study of necessary header files with respect to socket programming.
2. Study of Basic Functions of Socket Programming
3. Simple TCP/IP Client Server Communication
4. UDP Echo Client Server Communication
5. Concurrent TCP/IP Day-Time Server
6. Half Duplex Chat Using TCP/IP
7. Full Duplex Chat Using TCP/IP
8. Implementation of File Transfer Protocol
9. Remote Command Execution Using UDP
10. ARP Implementation Using UDP

18CSC302J/ Computer Networks

Ex.No:1
Date:

STUDY OF HEADER FILES WITH RESPECT TO SOCKET PROGRAMMING

1. **stdio.h:**
   Has standard input and output library providing simple and efficient buffered stream IO interface.

2. **unistd.h:**
   It is a POSIX standard for open system interface. [Portable Operating System Interface]

3. **string.h:**
   This header file is used to perform string manipulation operations on NULL terminated strings.(Bzero -0 the m/y)

4. **stdlib.h:**
   This header file contains the utility functions such as string conversion routines, memory allocation routines, random number generator, etc.

5. **sys/types.h:**
   Defines the data type of socket address structure in unsigned long.

6. **sys/socket.h:**
   The socket functions can be defined as taking pointers to the generic socket address structure called sockaddr.

7. **netinet/in.h:**
   Defines the IPv4 socket address structure commonly called Internet socket address structure called sockaddr_in.

8. **netdb.h:**
   Defines the structure hostent for using the system call gethostbyname to get the network host entry.

9. **time.h:**
   Has structures and functions to get the system date and time and to perform time manipulation functions. We use the function ctime(), that is defined in this header file , to calculate the current date and time.

10. **sys/stat.h:**
    Contains the structure stat to test a descriptor to see if it is of a specified type. Also it is used to display file or file system status.stat() updates any time related fields.when copying from 1 file to another.

11. **sys/ioctl.h**:
    Macros and defines used in specifying an ioctl request are located in this header file. We use the function ioctl() that is defined in this header file. ioctl() function is used to perform ARP cache operations.

12. pcap.h:

Has function definitions that are required for packet capturing. Some of the functions are pcap_lookupdev(),pcap_open_live() and pcap_loop(). pcap_lookupdev() is used to initialize the network device.The device to be sniffed is opened using the pcap_open_live(). Pcap_loop() determines the number of packets to be sniffed.

13. net/if_arp.h:

Contains the definitions for Address Resolution Protocol. We use this to manipulate the ARP request structure and its data members arp_pa,arp_dev and arp_ha. The arp_ha structure's data member sa_data[ ] has the hardware address.

14. errno.h:

It sets an error number when an error and that error can be displayed using perror function. It has symbolic error names. The error number is never set to zero by any library function.

15. arpa/inet.h:

This is used to convert internet addresses between ASCII strings and network byte ordered binary values (values that are stored in socket address structures). It is used for inet_aton, inet_addr, inet_ntoa functions.

Ex No: 2
Date:

STUDY OF BASIC FUNCTIONS OF SOCKET PROGRAMMING

AIM:

To discuss some of the basic functions used for socket programming.

1.**man socket**

NAME:

Socket – create an endpoint for communication.

SYNOPSIS:

```
#include<sys/types.h>
#include<sys/socket.h>
int socket(int domain,int type,int protocol);
```

DESCRIPTION:

- Socket creates an endpoint for communication and returns a descriptor.
- The domain parameter specifies a common domain this selects the protocol family which will be used for communication.
- These families are defined in <sys/socket.h>.

FORMAT:

NAME	PURPOSE
PF_UNIX,PF_LOCAL	Local Communication.
PF_INET	IPV4 Internet Protocols.
PF_IPX	IPX-Novell Protocols.
PF_APPLETALK	Apple Talk.

- The socket has the indicated type, which specifies the communication semantics.

TYPES:

1.SOCK_STREAM:

- Provides sequenced , reliable, two-way , connection based byte streams.
- An out-of-band data transmission mechanism, may be supported.

2.SOCK_DGRAM:

- Supports datagram (connectionless, unreliable messages of a fixed maximum length).

3.SOCK_SEQPACKET:

- Provides a sequenced , reliable, two-way connection based data transmission path for datagrams of fixed maximum length.

4.SOCK_RAW:

- Provides raw network protocol access.

5.SOCK_RDM:

- Provides a reliable datagram layer that doesn't guarantee ordering.

6.SOCK_PACKET:

- Obsolete and shouldn't be used in new programs.

2.man connect:

NAME:

connect – initiate a connection on a socket.

SYNOPSIS:

```
#include<sys/types.h>
#include<sys/socket.h>
int connect(int sockfd,const (struct sockaddr*)serv_addr,socklen_t addrlen);
```

DESCRIPTION:

- The file descriptor sockfd must refer to a socket.
- If the socket is of type SOCK_DGRAM then the serv_addr address is the address to which datagrams are sent by default and the only addr from which datagrams are received.
- If the socket is of type SOCK_STREAM or SOCK_SEQPACKET , this call attempts to make a connection to another socket.

RETURN VALUE:

- If the connection or binding succeeds, zero is returned.
- On error , -1 is returned , and error number is set appropriately.

ERRORS:

EBADF	Not a valid Index.
EFAULT	The socket structure address is outside the user's address space.
ENOTSOCK	Not associated with a socket.
EISCONN	Socket is already connected.
ECONNREFUSED	No one listening on the remote address.

3.man accept

NAME:

accept/reject job is sent to a destination.

SYNOPSIS:

accept destination(s)
reject[-t] [-h server] [-r reason] destination(s)

DESCRIPTION:

- accept instructs the printing system to accept print jobs to the specified destination.
- The –r option sets the reason for rejecting print jobs.
- The –e option forces encryption when connecting to the server.

4.man send

NAME:

send, sendto, sendmsg - send a message from a socket.

SYNOPSIS:

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

ssize_t send(int s, const void *buf, size_t len, int flags);
ssize_t sendto(int s, const void *buf, size_t len, int flags, const struct sock_addr*to, socklen_t tolen);
ssize_t sendmsg(int s, const struct msghdr *msg, int flags);

DESCRIPTION:

- The system calls send, sendto and sendmsg are used to transmit a message to another socket.
- The send call may be used only when the socket is in a connected state.
- The only difference between send and write is the presence of flags.
- The parameter is the file descriptor of the sending socket.

5.man recv

NAME:

recv, recvfrom, recvmsg – receive a message from a socket.

SYNOPSIS:

#include<sys/types.h>
#include<sys/socket.h>
ssize_t recv(int s, void *buf, size_t len, int flags);
ssize_t recvfrom(int s, void *buf, size_t len, int flags, struct sockaddr *from, socklen_t* from len);
ssize_t recvmsg(int s, struct msghdr *msg, int flags);

DESCRIPTION:

- The recvfrom and recvmsg calls are used to receive messages from a socket, and may be used to recv data on a socket whether or not it is connection oriented.
- If from is not NULL, and the underlying protocol provides the src addr , this src addr is filled in.
- The recv call is normally used only on a connection socket and is identical to recvfrom with a NULL from parameter.

6.man read

NAME:

read, readonly, return

7.man write

NAME:

write- send a message to another user.

SYNOPSIS:

write user[ttyname]

DESCRIPTION:

- write allows you to communicate with other users, by copying lines from terminal to ……
- When you run the write and the user you are writing to get a message of the form:
  Message from yourname @yourhost on yourtty at hh:mm:…
- Any further lines you enter will be copied to the specified user's terminal.
- If the other user wants to reply they must run write as well.

8. ifconfig

NAME:

ifconfig- configure a network interface.

SYNOPSIS:

ifconfig[interface]
ifconfig interface[aftype] options | address……

DESCRIPTION:

- ifconfig is used to configure the kernel resident network interfaces.
- It is used at boot time to setup 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.

9. man bind

SYNOPSIS:

bind[-m keymap] [-lp sv psv]

10. man htons/ man htonl

NAME:

htonl, htons, ntohl, ntohs - convert values between host and network byte order.

SYNOPSIS:

```
#include<netinet/in.h>
uint32_t    htonl(uint32_t hostlong);
uint16_t    htons(uint32_t hostshort);
uint32_t    ntohl(uint32_t netlong);
uint16_t    ntohs(uint16_t netshort);
```

DESCRIPTION:

- The htonl() function converts the unsigned integer hostlong from host byte order to network byte order.
- The htons() converts the unsigned short integer hostshort from host byte order to network byte order.
- The ntohl() converts the unsigned integer netlong from network byte order to host byte order.

11. man gethostname

NAME:

gethostname, sethostname- get/set host name.

SYNOPSIS:

```
#include<unistd.h>
int gethostname(char *name,size_t len);
int sethostname(const char *name,size_t len);
```

DESCRIPTION:

- These functions are used to access or to change the host name of the current processor.
- The gethostname() returns a NULL terminated hostname(set earlier by sethostname()) in the array name that has a length of len bytes.
- In case the NULL terminated then hostname does not fit ,no error is returned, but the hostname is truncated.
- It is unspecified whether the truncated hostname will be NULL terminated.

12. man gethostbyname

NAME:

gethostbyname, gethostbyaddr, sethostent, endhostent, herror, hstr – error – get network host entry.

SYNOPSIS:

```
 #include<netdb.h>
   extern int h_errno;
   struct hostent *gethostbyname(const char *name);
#include<sys/socket.h>
   struct hostent *gethostbyaddr(const char *addr)int len, int type);
   struct hostent *gethostbyname2(const char *name,int af);
```

DESCRIPTION:

- The gethostbyname() returns a structure of type hostent for the given hostname.
- Name->hostname or IPV4/IPV6 with dot notation.
- gethostbyaddr()- struct of type hostent / host address length
- Address types- AF_INET, AF_INET6.
- sethostent() – stay open is true(1).
- TCP socket connection should be open during queries.
- Server queries for UDP datagrams.
- endhostent()- ends the use of TCP connection.
- Members of hostent structure:
    a) h_name
    b) h_aliases
    c) h_addrtype
    d) h_length
    e) h_addr-list
    f) h_addr.

RESULT:

Thus the basic functions used for Socket Programming was studied successfully.

Ex No: 3
Date:

SIMPLE TCP/IP CLIENT SERVER COMMUNICATION

GIVEN REQUIREMENTS:

There are two hosts, Client and Server. The Client accepts the message from the user and sends it to the Server. The Server receives the message and prints it.

TECHNICAL OBJECTIVE:

To implement a simple TCP Client-Server application , where the Client on establishing a connection with the Server, sends a string to the Server. The Server reads the String and prints it.

METHODOLOGY:

Server:

- Include the necessary header files.
- Create a socket using socket function with family AF_INET, type as SOCK_STREAM.
- Initialize server address to 0 using the bzero function.
- Assign the sin_family to AF_INET, sin_addr to INADDR_ANY, sin_port to a dynamically assigned port number.
- Bind the local host address to socket using the bind function.
- Listen on the socket for connection request from the client.
- Accept connection request from the client using accept function.
- Within an infinite loop, using the recv function receive message from the client and print it on the console.

Client:

- Include the necessary header files.
- Create a socket using socket function with family AF_INET, type as SOCK_STREAM.
- Initialize server address to 0 using the bzero function.
- Assign the sin_family to AF_INET.
- Get the server IP address and port number from the console.
- Using gethostbyname function assign it to a hostent structure, and assign it to sin_addr of the server address structure.
- Request a connection from the server using the connect function.
- Within an infinite loop, read message from the console and send the message to the server using the send function.

CODING:

Server: tcpserver.c

```
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>
#include<arpa/inet.h>
#include<string.h>
int main(int asrgc,char*argv[])
```

```
{
        int bd,sd,ad;
        char buff[1024];
        struct sockaddr_in cliaddr,servaddr;
        socklen_t clilen;
        clilen=sizeof(cliaddr);
        bzero(&servaddr,sizeof(servaddr));

       /*Socket address structure*/
        servaddr.sin_family=AF_INET;
        servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
        servaddr.sin_port=htons(1999);

       /*TCP socket is created, an Internet socket address structure is filled with wildcard address
       & server's well known port*/
        sd=socket(AF_INET,SOCK_STREAM,0);

      /*Bind function assigns a local protocol address to the socket*/
        bd=bind(sd,(struct sockaddr*)&servaddr,sizeof(servaddr));

        /*Listen function specifies the maximum number of connections that kernel should queue
        for this socket*/
        listen(sd,5);
        printf("Server is running….\n");

       /*The server to return the next completed connection from the front of the
       completed connection Queue calls it*/
        ad=accept(sd,(struct sockaddr*)&cliaddr,&clilen);
        while(1)
        {
            bzero(&buff,sizeof(buff));
           /*Receiving the request message from the client*/
           recv(ad,buff,sizeof(buff),0);
           printf("Message received is %s\n",buff);
         }
}
```

Client: tcpclient.c

```
#include<stdio.h>
#include<string.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<unistd.h>
#include<netinet/in.h>
#include<netdb.h>
#include<arpa/inet.h>
int main(int argc,char * argv[])
{
        int cd,sd,ad;
        char buff[1024];
        struct sockaddr_in cliaddr,servaddr;
        struct hostent *h;
```

```
 /*This function looks up a hostname and it returns a pointer to a hostent
structure that contains all the IPV4 address*/
 h=gethostbyname(argv[1]);
 bzero(&servaddr,sizeof(servaddr));

 /*Socket address structure*/
 servaddr.sin_family=AF_INET;
 memcpy((char *)&servaddr.sin_addr.s_addr,h->h_addr_list[0],h->h_length);
 servaddr.sin_port = htons(1999);

/*Creating a socket, assigning IP address and port number for that socket*/
 sd = socket(AF_INET,SOCK_STREAM,0);

 /*Connect establishes connection with the server using server IP address*/
 cd=connect(sd,(struct sockaddr*)&servaddr,sizeof(servaddr));
 while(1)
 {
      printf("Enter the message: \n");

       /*Reads the message from standard input*/
       fgets(buff,100,stdin);

     /*Send function is used on client side to send data given by user on client
     side to the server*/
      send(sd,buff,sizeof(buff)+1,0);
      printf("\n Data Sent ");
      //recv(sd,buff,strlen(buff)+1,0);
      printf("%s",buff);
      }
}
```

SAMPLE OUTPUT:

Server:

(Host Name:Root1)

```
[root@localhost 4ita33]# vi tcpserver.c
[root@localhost 4ita33]# cc tcpserver.c
[root@localhost 4ita33]# ./a.out
Server is running….
Message received is hi

Message received is hi
```

Client:

(Host Name:Root2)

```
[root@localhost 4ita33]# vi tcpclient.c
[root@localhost 4ita33]# cc tcpclient.c
[root@localhost 4ita33]# ./a.out 127.0.0.1
Enter the message:
hi
```

Data Sent hi
Enter the message:
how r u

Data Sent how r u
Enter the message:

INFERENCE:

Thus, a program to perform simple communication between client and server using TCP/IP was implemented.

Ex No: 4
Date:

UDP ECHO CLIENT SERVER COMMUNICATION

GIVEN REQUIREMENTS:

There are two hosts, Client and Server. The Client accepts the message from the user and sends it to the Server. The Server receives the message, prints it and echoes the message back to the Client.

TECHNICAL OBJECTIVE:

To implement an UDP Echo Client-Server application , where the Client on establishing a connection with the Server, sends a string to the Server. The Server reads the String, prints it and echoes it back to the Client.

METHODOLOGY:

Server:

- Include the necessary header files.
- Create a socket using socket function with family AF_INET, type as SOCK_DGRAM.
- Initialize server address to 0 using the bzero function.
- Assign the sin_family to AF_INET, sin_addr to INADDR_ANY, sin_port to SERVER_PORT, a macro defined port number.
- Bind the local host address to socket using the bind function.
- Within an infinite loop, receive message from the client using recvfrom function, print it on the console and send (echo) the message back to the client using sendto function.

Client:

- Include the necessary header files.
- Create a socket using socket function with family AF_INET, type as SOCK_DGRAM.
- Initialize server address to 0 using the bzero function.
- Assign the sin_family to AF_INET.
- Get the server IP address from the console.
- Using gethostbyname function assign it to a hostent structure, and assign it to sin_addr of the server address structure.
- Within an infinite loop, read message from the console and send the message to the server using the sendto function.
- Receive the echo message using the recvfrom function and print it on the console.

CODING:
Server: udpserver.c

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

```
#include<arpa/inet.h>
#include<sys/types.h>
int main(int argc,char *argv[])
{
        int sd;
        char buff[1024];
        struct sockaddr_in cliaddr,servaddr;
        socklen_t clilen;
        clilen=sizeof(cliaddr);

      /*UDP socket is created, an Internet socket address structure is filled with wildcard
      address & server's well known port*/
      sd=socket(AF_INET,SOCK_DGRAM,0);
      if (sd<0)
      {
              perror ("Cannot open Socket");
              exit(1);
       }
       bzero(&servaddr,sizeof(servaddr));
       /*Socket address structure*/
       servaddr.sin_family=AF_INET;
       servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
       servaddr.sin_port=htons(5669);

       /*Bind function assigns a local protocol address to the socket*/
       if(bind(sd,(struct sockaddr*)&servaddr,sizeof(servaddr))<0)
       {
            perror("error in binding the port");
            exit(1);
       }
       printf("%s","Server is Running…\n");
       while(1)
       {
              bzero(&buff,sizeof(buff));

               /*Read the message from the client*/
              if(recvfrom(sd,buff,sizeof(buff),0,(struct sockaddr*)&cliaddr,&clilen)<0)
              {
                      perror("Cannot rec data");
                      exit(1);
               }
              printf("Message is received \n",buff);

              /*Sendto function is used to echo the message from server to client side*/
              if(sendto(sd,buff,sizeof(buff),0,(struct sockadddr*)&cliaddr,clilen)<0)
              {
                      perror("Cannot send data to client");
                      exit(1);
              }
               printf("Send data to UDP Client: %s",buff);
       }
```

```
        cloSe(sd);
        return 0;
}
```

Client: udpclient.c

```
#include<sys/types.h>
#include<sys/socket.h>
#include<stdio.h>
#include<unistd.h>
#include<string.h>
#include<netinet/in.h>
#include<netdb.h>
int main(int argc,char*argv[])
{
        int sd;
        char buff[1024];
        struct sockaddr_in servaddr;
        socklen_t len;
        len=sizeof(servaddr);

      /*UDP socket is created, an Internet socket address structure is filled with
      wildcard address & server's well  known port*/
        sd = socket(AF_INET,SOCK_DGRAM,0);
        if(sd<0)
        {
                perror("Cannot open socket");
                exit(1);
        }
        bzero(&servaddr,len);

        /*Socket address structure*/
        servaddr.sin_family=AF_INET;
        servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
        servaddr.sin_port=htons(5669);
        while(1)
        {
                printf("Enter Input data : \n");
                bzero(buff,sizeof(buff));

                 /*Reads the message from standard input*/
                fgets(buff,sizeof (buff),stdin);

                 /*sendto is used to transmit the request message to the server*/
                if(sendto (sd,buff,sizeof (buff),0,(struct sockaddr*)&servaddr,len)<0)
                {
                        perror("Cannot send data");
                        exit(1);
                 }
                printf("Data sent to UDP Server:%s",buff);
```

```
                bzero(buff,sizeof(buff));
                /*Receiving the echoed message from server*/
                if(recvfrom (sd,buff,sizeof(buff),0,(struct sockaddr*)&servaddr,&len)<0)
                 {
                        perror("Cannot receive data");
                       exit(1);
                 }
                printf("Received Data from server: %s",buff);
          }
        close(sd);
        return 0;
}
```

SAMPLE OUTPUT:

Server:

(Host Name:Root1)

```
[root@localhost 4ita33]# vi udpserver.c
[root@localhost 4ita33]# cc udpserver.c
[root@localhost 4ita33]# ./a.out
Server is Running…

Message is received
Send data to UDP Client: hi

Message is received
Send data to UDP Client: how are u
```

Client:

(Host Name:Root2)

```
[root@localhost 4ita33]# vi udpclient.c
[root@localhost 4ita33]# cc udpclient.c
[root@localhost 4ita33]# ./a.out 127.0.0.1
Enter input data :
hi
Data sent to UDP Server:hi
Received Data from server: hi

Enter input data :
how are u
Data sent to UDP Server:how are u
Received Data from server: how are u

Enter input data :
```

INFERENCE:

Thus, the UDP ECHO client server communication is established by sending the message from the client to the server and server prints it and echoes the message back to the client.

Ex No:5
Date:

CONCURRENT TCP/IP DAY-TIME SERVER

GIVEN REQUIREMENTS:

There are two hosts, Client and Server. The Client requests the concurrent server for the date and time. The Server sends the date and time, which the Client accepts and prints.

TECHNICAL OBJECTIVE:

To implement a TCP/IP day time server (concurrent server) that handles multiple client requests. Once the client establishes connection with the server, the server sends its day-time details to the client which the client prints in its console.

METHODOLOGY:

Server:

- Include the necessary header files.
- Create a socket using socket function with family AF_INET, type as SOCK_STREAM.
- Initialize server address to 0 using the bzero function.
- Assign the sin_family to AF_INET, sin_addr to INADDR_ANY, sin_port to statically assigned port number.
- Bind the local host address to socket using the bind function.
- Within a for loop, accept connection request from the client using accept function.
- Use the fork system call to spawn the processes.
- Calculate the current date and time using the ctime() function. Change the format so that it is appropriate for human readable form and send the date and time to the client using the write function.

Client:

- Include the necessary header files.
- Create a socket using socket function with family AF_INET, type as SOCK_STREAM.
- Initialize server address to 0 using the bzero function.
- Assign the sin_family to AF_INET.
- Get the server IP address from the console.
- Using gethostbyname function assign it to a hostent structure, and assign it to sin_addr of the server address structure.
- Request a connection from the server using the connect function.
- Within an infinite loop, receive the date and time from the server using the read function and print the date and time on the console.

CODING:
Server: dtserver.c

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

```
#include<string.h>
#include<netinet/in.h>
#include<netdb.h>
0int main(int argc,char *argv[])
{
        int sd,ad;
        char buff[1024];
        struct sockaddr_in servaddr,cliaddr;
        //socklen_t clilen=sizeof(cliaddr);
        time_t t1;
        bzero(&servaddr,sizeof(servaddr));
       /*Socket address structure*/
        servaddr.sin_family=AF_INET;
        servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
        servaddr.sin_port=htons(1507);
         /*TCP socket is created, an Internet socket address structure is filled with
        wildcard address & server's well known port*/
        sd=socket(AF_INET,SOCK_STREAM,0);

         /*Bind function assigns a local protocol address to the socket*/
        bind(sd,(struct sockaddr*)&servaddr,sizeof(servaddr));

        /*Listen function specifies the maximum number of connections that kernel  should queue
for this socket*/
        listen(sd,5);
        printf("Server is running…\n");
        /*The server to return the next completed connection from the front of the
        completed connection Queue calls it*/
          ad=accept(sd,(struct sockaddr *)NULL,NULL);
          while(1)
          {
                   bzero(&buff,sizeof(buff));

                  /*Library function time returns the Coordinated Universal Time*/
        t1=time(NULL);

                  /*Prints the converted string format*/
        snprintf(buff,sizeof(buff),"%24s\r\n",ctime(&t1));
                  send(ad,buff,sizeof(buff),0);
            }
```

Client: dtclient.c

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netdb.h>
#include<netinet/in.h>
#include<unistd.h>
#include<time.h>
int main(int argc,char *argv[])
```

```
{
        int sd,ad;
        char buff[1024];
        struct sockaddr_in cliaddr,servaddr;
        struct hostent *h;
        h=gethostbyname(argv[1]);
        bzero(&servaddr,sizeof(servaddr));

       /*Socket address structure*/
        servaddr.sin_family=AF_INET;
        memcpy((char*)&servaddr.sin_addr.s_addr,h->h_addr_list[0],h->h_length);
        servaddr.sin_port=htons(1507);

        /*TCP socket is created, an Internet socket address structure is filled with
       wildcard address & server's well  known port*/
        sd=socket(AF_INET,SOCK_STREAM,0);

       /*Connect establishes connection with the server using server IP address*/
        connect(sd,(struct sockaddr*)&servaddr,sizeof(servaddr));
        recv(sd,buff,sizeof(buff),0);
        printf("Day time of server is: %s\n",buff);
}
```

SAMPLE OUTPUT:

Server:

(Host Name:Root1)

```
[root@localhost 4ita33]# vi dtserver.c
[root@localhost 4ita33]# cc dtserver.c
[root@localhost 4ita33]# ./a.out
```

Server is running…

Client:

(Host Name:Root2)

```
[root@localhost 4ita33]# vi dtclient.c
[root@localhost 4ita33]# cc dtclient.c
[root@localhost 4ita33]# ./a.out 127.0.0.1
```

Day time of server is: Sat Oct 27 18:02:21 2007

INFERENCE:

Thus the concurrent daytime client- server communication is established by sending the request message from the client to the concurrent server and the server sends its time to all the clients and displays it.

Ex No:6
Date:

HALF DUPLEX CHAT USING TCP/IP

GIVEN REQUIREMENTS:

There are two hosts, Client and Server. Both the Client and the Server exchange message i.e. they send messages or receive message from the other. There is only a single way communication between them.

TECHNICAL OBJECTIVE:

To implement a half duplex application, where the Client establishes a connection with the Server. The Client can send and the server well receive messages at the same time.

METHODOLOGY:

Server:

- Include the necessary header files.
- Create a socket using socket function with family AF_INET, type as SOCK_STREAM.
- Initialize server address to 0 using the bzero function.
- Assign the sin_family to AF_INET, sin_addr to INADDR_ANY, sin_port to dynamically assigned port number.
- Bind the local host address to socket using the bind function.
- Listen on the socket for connection request from the client.
- Accept connection request from the Client using accept function.
- Fork the process to receive message from the client and print it on the console.
- Read message from the console and send it to the client.

Client:

- Include the necessary header files.
- Create a socket using socket function with family AF_INET, type as SOCK_STREAM.
- Initialize server address to 0 using the bzero function.
- Assign the sin_family to AF_INET.
- Get the server IP address and the Port number from the console.
- Using gethostbyname function assign it to a hostent structure, and assign it to sin_addr of the server address structure.
- Request a connection from the server using the connect function.
- Fork the process to receive message from the server and print it on the console.
- Read message from the console and send it to the server.

CODING:

Server: hserver.c

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

```
int main(int argc,char *argv[])
{
int n,sd,ad;
struct sockaddr_in servaddr,cliaddr;
socklen_t clilen,servlen;
char buff[10000],buff1[10000];
bzero(&servaddr,sizeof(servaddr));

        /*Socket address structure*/
          servaddr.sin_family=AF_INET;
          servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
          servaddr.sin_port=htons(5000);

        /*TCP socket is created, an Internet socket address structure is filled with
          wildcard address & server's well known port*/
          sd=socket(AF_INET,SOCK_STREAM,0);

          /*Bind function assigns a local protocol address to the socket*/
          bind(sd,(struct sockaddr*)&servaddr,sizeof(servaddr));

          /*Listen function specifies the maximum number of connections that kernel
          should queue for this socket*/
          listen(sd,5);
          printf("%s\n","server is running…");

          /*The server to return the next completed connection from the front of the
         completed connection Queue calls it*/
          ad=accept(sd,(struct sockaddr*)&cliaddr,&clilen);
          while(1)
          {
                    bzero(&buff,sizeof(buff));

                    /*Receiving the request from client*/
                    recv(ad,buff,sizeof(buff),0);

                    printf("Receive from the client:%s\n",buff);
                    n=1;
                    while(n==1)
                    {
                              bzero(&buff1,sizeof(buff1));
                              printf("%s\n","Enter the input data:");

                              /*Read the message from client*/
                              fgets(buff1,10000,stdin);

                              /*Sends the message to client*/
                              send(ad,buff1,strlen(buff1)+1,0);
                              printf("%s\n","Data sent");
                              n=n+1;
                    }
          }
```

```
	return 0;
}
```

Client: hclient.c

```
#include<sys/types.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<netinet/in.h>
#include<unistd.h>
#include<stdio.h>
#include<netdb.h>
int main(int argc,char *argv[])
{
	int n,sd,cd;
	struct sockaddr_in servaddr,cliaddr;
	socklen_t servlen,clilen;
	char buff[10000],buff1[10000];
	bzero(&servaddr,sizeof(servaddr));

	/*Socket address structure*/
	servaddr.sin_family=AF_INET;
	servaddr.sin_addr.s_addr=inet_addr(argv[1]);
	servaddr.sin_port=htons(5000);

	/*Creating a socket, assigning IP address and port number for that socket*/
	sd=socket(AF_INET,SOCK_STREAM,0);

	/*Connect establishes connection with the server using server IP address*/
	cd=connect(sd,(struct sockaddr*)&servaddr,sizeof(servaddr));
	while(1)
	{
		bzero(&buff,sizeof(buff));
		printf("%s\n","Enter the input data:");

		/*This function is used to read from server*/
		fgets(buff,10000,stdin);

		/*Send the message to server*/
		send(sd,buff,strlen(buff)+1,0);
		printf("%s\n","Data sent");
		n=1;
		while(n==1)
		{
			bzero(&buff1,sizeof(buff1));

			/*Receive the message from server*/
			recv(sd,buff1,sizeof(buff1),0);
			printf("Received from the server:%s\n",buff1);
			n=n+1;
		}
	}
```

```
        return 0;
}
```

SAMPLE OUTPUT:

Server:

(Host Name:Root1)

```
[root@localhost 4ita33]# vi hserver.c
[root@localhost 4ita33]# cc hserver.c
[root@localhost 4ita33]# ./a.out

Server is running…
Receive from the client:hi

Enter the input data:
how are u da ..
Data sent

Receive from the client:me fine da ...

Enter the input data:
```

Client:

(Host Name:Root2)

```
[root@localhost 4ita33]# vi hclient.c
[root@localhost 4ita33]# cc hclient.c
[root@localhost 4ita33]# ./a.out 127.0.0.1
Enter the input data:
hi
Data sent:
Received from the server:how are u da ..

Enter the input data:
me fine da ...
Data sent
```

INFERENCE:

Thus the chat application full duplex communication is established by sending the request from the client to the server, server gets the message and gives response to the client and prints it.

Ex No:7
Date:

FULL DUPLEX CHAT USING TCP/IP

GIVEN REQUIREMENTS:

There are two hosts, Client and Server. Both the Client and the Server exchange message i.e. they send messages to and receive message from the other. There is a two way communication between them.

TECHNICAL OBJECTIVE:

To implement a full duplex application, where the Client establishes a connection with the Server. The Client and Server can send as well as receive messages at the same time. Both the Client and Server exchange messages.

METHODOLOGY:

Server:

- Include the necessary header files.
- Create a socket using socket function with family AF_INET, type as SOCK_STREAM.
- Initialize server address to 0 using the bzero function.
- Assign the sin_family to AF_INET, sin_addr to INADDR_ANY, sin_port to dynamically assigned port number.
- Bind the local host address to socket using the bind function.
- Listen on the socket for connection request from the client.
- Accept connection request from the Client using accept function.
- Fork the process to receive message from the client and print it on the console.
- Read message from the console and send it to the client.

Client:

- Include the necessary header files.
- Create a socket using socket function with family AF_INET, type as SOCK_STREAM.
- Initialize server address to 0 using the bzero function.
- Assign the sin_family to AF_INET.
- Get the server IP address and the Port number from the console.
- Using gethostbyname function assign it to a hostent structure, and assign it to sin_addr of the server address structure.
- Request a connection from the server using the connect function.
- Fork the process to receive message from the server and print it on the console.
- Read message from the console and send it to the server.

CODING:

Server: fserver.c

```
#include<sys/types.h>
#include<sys/socket.h>
#include<stdio.h>
#include<unistd.h>
#include<netdb.h>
#include<arpa/inet.h>
```

```
#include<netinet/in.h>
int main(int argc,char *argv[])
{
        int ad,sd;
        struct sockaddr_in servaddr,cliaddr;
        socklen_t servlen,clilen;
        char buff[1000],buff1[1000];
        pid_t cpid;
        bzero(&servaddr,sizeof(servaddr));

       /*Socket address structure*/
        servaddr.sin_family=AF_INET;
        servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
        servaddr.sin_port=htons(5500);

        /*TCP socket is created, an Internet socket address structure is filled with
        wildcard address & server's well known port*/
        sd=socket(AF_INET,SOCK_STREAM,0);

        /*Bind function assigns a local protocol address to the socket*/
        bind(sd,(struct sockaddr*)&servaddr,sizeof(servaddr));

        /*Listen function specifies the maximum number of connections that kernel  should queue
for this socket*/
        listen(sd,5);
        printf("%s\n","Server is running.....");

       /*The server to return the next completed connection from the front of the
        completed connection Queue calls it*/
        ad=accept(sd,(struct sockaddr*)&cliaddr,&clilen);

        /*Fork system call is used to create a new process*/
        cpid=fork();



        if(cpid==0)
        {
                while(1)
                {
                        bzero(&buff,sizeof(buff));

                        /*Receiving the request from client*/
                        recv(ad,buff,sizeof(buff),0);
                        printf("Received  message from the client:%s\n",buff);
                }
           }
        else
        {
                 while(1)
                 {
```

```
                        bzero(&buff1,sizeof(buff1));
                        printf("%s\n","Enter the input data:");

                /*Read the message from client*/
                        fgets(buff1,10000,stdin);

                        /*Sends the message to client*/
                        send(ad,buff1,strlen(buff1)+1,0);
                        printf("%s\n","Data sent…");
                 }
        }
        return 0;
}
```

Client: fclient.c

```
#include<sys/socket.h>
#include<sys/types.h>
#include<stdio.h>
#include<arpa/inet.h>
#include<unistd.h>
#include<netdb.h>
#include<netinet/in.h>
int main(int argc,char *argv[])
{
        int sd,cd;
        struct sockaddr_in servaddr,cliaddr;
        socklen_t servlen,clilen;
        char buff[1000],buff1[1000];
        pid_t cpid;
        bzero(&servaddr,sizeof(servaddr));
        servaddr.sin_family=AF_INET;
        servaddr.sin_addr.s_addr=inet_addr(argv[1]);
        servaddr.sin_port=htons(5500);

        /*Creating a socket, assigning IP address and port number for that socket*/
        sd=socket(AF_INET,SOCK_STREAM,0);

        /*Connect establishes connection with the server using server IP address*/
        cd=connect(sd,(struct sockaddr*)&servaddr,sizeof(servaddr));

        /*Fork is used to create a new process*/
        cpid=fork();
        if(cpid==0)
        {
                while(1)
                {
                        bzero(&buff,sizeof(buff));
```

```
                printf("%s\n","Enter the input data:");

                /*This function is used to read from server*/
                fgets(buff,10000,stdin);

                 /*Send the message to server*/
                send(sd,buff,strlen(buff)+1,0);
                printf("%s\n","Data sent…");
        }
    }

    else
    {
        while(1)
        {
                bzero(&buff1,sizeof(buff1));

                /*Receive the message from server*/
                recv(sd,buff1,sizeof(buff1),0);
                printf("Received message from the server:%s\n",buff1);
        }
    }
    return 0;
}
```

SAMPLE OUTPUT:

Server:

(Host Name:Root1)

```
[root@localhost 4ita33]# vi fserver.c
[root@localhost 4ita33]# cc fserver.c
[root@localhost 4ita33]# ./a.out

Server is running.....
Enter the input data:
Received  message from the client:hi
how are u
Data sent…

Enter the input data:
Received message from the client:i am fine
```

Client:

(Host Name:Root2)

```
[root@localhost 4ita33]# vi fclient.c
[root@localhost 4ita33]# cc fclient.c
[root@localhost 4ita33]# ./a.out 127.0.0.1
```

Enter the input data:
hi
Data sent…

Enter the input data:
Received message from the server:how are u
i am fine
Data sent…

Enter the input data:

INFERENCE:

Thus the chat application full duplex communication is established by sending the request from the client to the server, server gets the message and gives response to the client and prints it.

Ex No: 8
Date:

IMPLEMENTATION OF FILE TRANSFER PROTOCOL

GIVEN REQUIREMENTS:

There are two hosts, Client and Server. The Client sends the name of the file it needs from the Server and the Server sends the contents of the file to the Client, where it is stored in a file.

TECHNICAL OBJECTIVE:

To implement FTP application, where the Client on establishing a connection with the Server sends the name of the file it wishes to access remotely. The Server then sends the contents of the file to the Client, where it is stored.

METHODOLOGY:

Server:

- Include the necessary header files.
- Create a socket using socket function with family AF_INET, type as SOCK_STREAM.
- Initialize server address to 0 using the bzero function.
- Assign the sin_family to AF_INET, sin_addr to INADDR_ANY, sin_port to dynamically assigned port number.
- Bind the local host address to socket using the bind function.
- Listen on the socket for connection request from the client.
- Accept connection request from the Client using accept function.
- Within an infinite loop, receive the file name from the Client.
- Open the file, read the file contents to a buffer and send the buffer to the Client.

Client:

- Include the necessary header files.
- Create a socket using socket function with family AF_INET, type as SOCK_STREAM.
- Initialize server address to 0 using the bzero function.
- Assign the sin_family to AF_INET.
- Get the server IP address and the Port number from the console.
- Using gethostbyname function assign it to a hostent structure, and assign it to sin_addr of the server address structure.
- Within an infinite loop, send the name of the file to be viewed to the Server.
- Receive the file contents, store it in a file and print it on the console.

CODING:

Server: ftps.c

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

```
#include<string.h>
int main(int argc,char *argv[])
{
int sd,ad,size;
struct sockaddr_in servaddr,cliaddr;
socklen_t clilen;
 clilen=sizeof(cliaddr);
struct stat x;
char buff[100],file[10000];
FILE *fp;

bzero(&servaddr,sizeof(servaddr));
servaddr.sin_family=AF_INET;
servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
servaddr.sin_port=htons(1500);

sd=socket(AF_INET,SOCK_STREAM,0);
bind(sd,(struct sockaddr*)&servaddr,sizeof(servaddr));
listen(sd,5);
printf("%s\n","Server Is Running....");
ad=accept(sd,(struct sockaddr*)&cliaddr,&clilen);
while(1)
{
bzero(buff,sizeof(buff));
bzero(file,sizeof(file));
recv(ad,buff,sizeof(buff),0);
fp=fopen(buff,"r");
stat(buff,&x);
size=x.st_size;
fread(file,sizeof(file),1,fp);
send(ad,file,sizeof(file),0);
}

}
```

Client: ftpc.c

```
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<netdb.h>
#include<stdio.h>
#include<unistd.h>
int main(int argc,char *argv[])
{
int sd,cd;
struct sockaddr_in servaddr,cliaddr;
socklen_t clilen;
char buff[100],file[10000];
struct hostent *h;
```

```
h=gethostbyname(argv[1]);
bzero(&servaddr,sizeof(servaddr));
servaddr.sin_family=h->h_addrtype;
memcpy((char *)&servaddr.sin_addr.s_addr,h->h_addr_list[0],h->h_length);
servaddr.sin_port=htons(1500);

sd=socket(AF_INET,SOCK_STREAM,0);
cd=connect(sd,(struct sockaddr*)&servaddr,sizeof(servaddr));

while(1)
{
printf("%s\n","Enter the File Name :");
scanf("%s",buff);
send(sd,buff,strlen(buff)+1,0);
printf("%s\n","File Output :");
recv(sd,file,sizeof(file),0);
printf("%s",file);
}
return 0;
}
```

SAMPLE OUTPUT:

Server:

(Host Name:Root1)

```
[root@localhost 4ita33]# vi ftps.c
[root@localhost 4ita33]# cc ftps.c
[root@localhost 4ita33]# ./a.out

Server is Running…
FILE REACHED
File output : this is my network lab
```

Client:

(Host Name:Root2)

```
[root@localhost 4ita33]# vi ftpc.c
[root@localhost 4ita33]# cc ftpc.c
[root@localhost 4ita33]# ./a.out
Enter the filename:
ita.txt
Sending the file content
Data sent.....
```

INFERENCE:

Thus the FTP client-server communication is established and data is transferred between the client and server machines.

Ex No: 9
Date:

REMOTE COMMAND EXECUTION USING UDP

GIVEN REQUIREMENTS:

There are two hosts, Client and Server. The Client sends a command to the Server, which executes the command and sends the result back to the Client.

TECHNICAL OBJECTIVE:

Remote Command execution is implemented through this program using which Client is able to execute commands at the Server. Here, the Client sends the command to the Server for remote execution. The Server executes the command and the send result of the execution back to the Client.

METHODOLOGY:

Server:

- Include the necessary header files.
- Create a socket using socket function with family AF_INET, type as SOCK_DGRAM.
- Initialize server address to 0 using the bzero function.
- Assign the sin_family to AF_INET, sin_addr to INADDR_ANY, sin_port to dynamically assigned port number.
- Bind the local host using the bind() system call.
- Within an infinite loop, receive the command to be executed from the client.
- Append text "> temp.txt" to the command.
- Execute the command using the "system()" system call.
- Send the result of execution to the Client using a file buffer.

Client:

- Include the necessary header files.
- Create a socket using socket function with family AF_INET, type as SOCK_DGRAM.
- Initialize server address to 0 using the bzero function.
- Assign the sin_family to AF_INET.
- Get the server IP address and the Port number from the console.
- Using gethostbyname() function assign it to a hostent structure, and assign it to sin_addr of the server address structure.
- Obtain the command to be executed in the server from the user.
- Send the command to the server.
- Receive the output from the server and print it on the console.

CODING:

Server: udpremoteserver.c

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

```
#include<string.h>
#include<sys/stat.h>
#include<arpa/inet.h>
#include<unistd.h>
int main(int argc,char* argv[])
{
	int sd,size;
	char buff[1024],file[10000];
	struct sockaddr_in cliaddr,servaddr;
	    FILE *fp;
	    struct stat x;
	socklen_t clilen;
	clilen=sizeof(cliaddr);
	bzero(&servaddr,sizeof(servaddr));
	servaddr.sin_family=AF_INET;
	servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
	servaddr.sin_port=htons(9976);
	sd=socket(AF_INET,SOCK_DGRAM,0);
	if(sd<0)
	{
		printf("Socket CReation Error");
	}
	bind(sd,(struct sockaddr *)&servaddr,sizeof(servaddr));
	while(1)
	    {
	      bzero(buff,sizeof(buff));
	      recvfrom(sd,buff,sizeof(buff),0,(struct sockaddr *)&cliaddr,&clilen);
			strcat(buff,">file1");
			system(buff);
	      fp=fopen("file1","r");
			stat("file1",&x);
			size=x.st_size;
			fread(file,size,1,fp);
	      sendto(sd,file,sizeof(file),0,(struct sockaddr *)&cliaddr,sizeof(cliaddr));
	      printf("Data Sent to UDPCLIENT %s",buff);
	}
	close(sd);
return 0; }
```

Client: udpremoteclient.c

```
#include<sys/types.h>
#include<sys/socket.h>
#include<stdio.h>
#include<unistd.h>
#include<netdb.h>
#include<netinet/in.h>
#include<string.h>
#include<arpa/inet.h>
#include<sys/stat.h>
int main(int argc,char* argv[])
{
```

```
    int sd;
    char buff[1024],file[10000];
    struct sockaddr_in cliaddr,servaddr;
    struct hostent *h;
        socklen_t servlen;
        servlen=sizeof(servaddr);
    h=gethostbyname(argv[1]);
    bzero(&servaddr,sizeof(servaddr));
    servaddr.sin_family=h->h_addrtype;
    memcpy((char *)&servaddr.sin_addr,h->h_addr_list[0],h->h_length);
    servaddr.sin_port=htons(9976);
    sd=socket(AF_INET,SOCK_DGRAM,0);
    if(sd<0)
    {
          printf("Socket CReation Error");
        }
    bind(sd,(struct sockaddr *)&servaddr,sizeof(servaddr));
    while(1)
    {
            printf("\nEnter the command to be executed");
            fgets(buff,1024,stdin);
sendto(sd,buff,strlen(buff)+1,0,(struct sockaddr *)&servaddr,sizeof(servaddr));
            printf("\nData Sent");
recvfrom(sd,file,strlen(file)+1,0,(struct sockaddr *)&servaddr,&servlen);
            printf("Recieved From UDPSERVER %s",file);
    }
    return 0;
}
```

SAMPLE OUTPUT:

Server:

(Host Name:Root1)

```
[root@localhost 4ita33]# vi udpremoteserver.c
[root@localhost 4ita33]# cc udpremoteserver.c
[root@localhost 4ita33]# ./a.out
Server is running.....
VIM(1)                                          VIM(1)
NAME
  vim - Vi IMproved, a programmers text editor
SYNOPSIS
vim [options] [file ..]
vim [options] -
vim [options] -t tag
vim [options] -q [errorfile}
ex
review rview renbjc
```

Client:

(Host Name:Root2)

```
[root@localhost 4ita33]# vi udpremoteclient.c
[root@localhost 4ita33]# cc udpremoteclient.c
```

```
[root@localhost 4ita33]# ./a.out 127.0.0.1
Enter command:
man vi
Command sent to server

Enter command:
```

INFERENCE:

Thus the Remote Command Execution between the client and server is implemented.

Ex No: 10
Date:

ARP IMPLEMENTATION USING UDP

GIVEN REQUIREMENTS:

There is a single host. The IP address of any Client in the network is given as input and the corresponding hardware address is got as the output.

TECHNICAL OBJECTIVE:

Address Resolution Protocol (ARP) is implemented through this program. The IP address of any Client is given as the input. The ARP cache is looked up for the corresponding hardware address. This is returned as the output. Before compiling that Client is pinged.

METHODOLOGY:

- Include the necessary header files.
- Create a socket using socket function with family AF_INET, type as SOCK_DGRAM.
- Declare structures arpreq (as NULL structure, if required) and sockaddr_in.
- Initialize server address to 0 using the bzero function.
- Assign the sin_family to AF_INET and sin_addr using inet_aton().
- Using the object of arpreq structure assign the name of the Network Device to the data member arp_dev like, arp_dev=”eth0”.
- Ping the required Client.
- Using the ioctl() we get the ARP cache entry for the given IP address.
- The output of the ioctl() function is stored in the sa_data[0] datamember of the arp_ha structure which is in turn a data member of structure arpreq.
- Print the hardware address of the given IP address on the output console.

CODING:
ARP: arp.c

```
#include<sys/types.h>
#include<sys/socket.h>
#include<net/if_arp.h>
#include<sys/ioctl.h>
#include<stdio.h>
#include<unistd.h>
#include<netinet/in.h>
#include<arpa/inet.h>
int main(int argc,char *argv[])
{
struct sockaddr_in sin={0};
struct arpreq myarp={{0}};
unsigned char *ptr;
int sd;
sin.sin_family=AF_INET;
```

```
if(inet_aton(argv[1],&sin.sin_addr)==0)
{
printf("Ip address Entered '%s' is not valid \n",argv[1]);
exit(0);
}
memcpy(&myarp.arp_pa,&sin,sizeof(myarp.arp_pa));
strcpy(myarp.arp_dev,"eth0");
sd=socket(AF_INET,SOCK_DGRAM,0);
if(ioctl(sd,SIOCGARP,&myarp)==1)
{
printf("No Entry in ARP Cache for '%s'",argv[1]);
exit(0);
}
ptr=&myarp.arp_ha.sa_data[0];
printf("MAC Address For '%s' : ",argv[1]);
printf("%X:%X:%X:%X:%X:%X\n",*ptr,*(ptr+1),*(ptr+2),*(ptr+3),*(ptr+4),*(ptr+5),*(ptr+5));
return 0;
}
```

SAMPLE OUTPUT:

Host: arp.c

(Host Name:Root1)

```
[root@localhost 4ita33]# vi arp.c
[root@localhost 4ita33]# ping  172.16.29.51

PING 172.16.29.51 (172.16.29.51) 56(84) bytes of data.
64 bytes from172.16.29.51: icmp_seq=1 ttl=64 time=1.19 ms
64 bytes from 172.16.29.51: icmp_seq=2 ttl=64 time=0.817 ms

-- 172.16.29.51ping statistics ---

2 packets transmitted, 2 received, 0% packet loss, time 999ms

rtt min/avg/max/mdev = 0.817/1.005/1.193/0.188 ms

[root@localhost 4ita33]# cc arp.c
[root@localhost 4ita33]# ./a.out 172.16.29.51

Hardware Address is: 172.16.29.51:
The MAC address is:0:8:5C:5D:47:50:
```

INFERENCE:

Thus the ARP implementation is developed to gets the MAC address of the remote machine’s IP address from ARP cache and prints it.