## Course Outcomes and CO-PO-PSO Mapping

**Course Objectives: The course will help to**

1. Introduce CRC Mechanism.
2. Understand the concepts of Data link layer.
3. Gain the knowledge on network layer.
4. Understand the concepts of TCP and UDP Protocols.
5. Learn the concepts of sockets and DNS.

# Course Outcomes: After learning the concepts of this course, the student will be able to

1. Compute CRC Mechanisms.
2. Demonstrate and implement the Go-Back-N mechanism.
3. Demonstrate and Apply routing algorithms.
4. Illustrate and implement TCP and UDP Client and server Applications.
5. Develop DNS and Ping service.

**CO -PO MAPPING:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **PO1** | **PO2** | **PO3** | **PO4** | **PO5** | **PO6** | **PO7** | **PO8** | **PO9** | **PO10** | **PO11** | **PO12** |
| **CO1** | **1** | **2** |  | **3** |  |  |  |  |  |  |  |  |
| **CO2** |  | **1** |  | **2** |  |  |  |  |  |  |  |  |
| **CO3** |  | **2** | **1** | **2** | **3** |  |  |  |  |  |  |  |
| **CO4** |  | **1** |  |  | **2** |  |  |  |  |  |  |  |
| **CO5** | **1** | **1** |  | **2** |  |  |  |  |  |  |  |  |
| **Avg** | **1** | **1.5** | **1** | **2.25** | **2.5** |  |  |  |  |  |  |  |

**CO-PSO Matrix:**

|  |  |  |
| --- | --- | --- |
|  | **PSO1** | **PSO2** |
| **CO1** | **1** |  |
| **CO2** |  |  |
| **CO3** |  | **2** |
| **CO4** |  |  |
| **CO5** | **2** |  |
| **Average** | **1.5** | **2** |

Experiment 1: **Implement on a data set of characters the CRC . AIM:**

**Program:**

#include<stdlib.h>

#include<conio.h>

#include<stdio.h>

int main()

{

int i,j,n,g,a,arr[20],gen[20],b[20],q[20],s;

printf("Transmitter side:");

printf("\nEnter no. of data bits:");

scanf("%d",&n);

printf("Enter data:");

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

scanf("%d",&arr[i]);

printf("Enter size of generator:");

scanf("%d",&g);

do{

printf("Enter generator:");

for(j=0;j<g;j++)

scanf("%d",&gen[j]);

}

while(gen[0]!=1);

printf("\n\tThe generator matrix:");

for(j=0;j<g;j++)

printf("%d",gen[j]);

a=n+(g-1);

printf("\n\tThe appended matrix is:");

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

arr[n+i]=0;

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

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

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

q[i]= arr[i];

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

{

if(arr[i]==0)

{

for(j=i;j<g+i;++j)

arr[j] = arr[j]^0;

}

else

{

arr[i] = arr[i]^gen[0];

arr[i+1]=arr[i+1]^gen[1];

arr[i+2]=arr[i+2]^gen[2];

arr[i+3]=arr[i+3]^gen[3];

}

}

printf("\n\tThe CRC is :");

for(i=n;i< a;++i)

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

s=n+a;

for(i=n;i<s;i++)

q[i]=arr[i];

printf("\n");

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

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

}

**Output:**

Transmitter side:

Enter no. of data bits:8 Enter data:1 0 1 0 0 0 0 1 Enter size of generator:4 Enter generator:1 0 0 1 The generator matrix:1001

The appended matrix is:10100001000 The CRC is :111

10100001111

Experiment 2 :

**A) Develop a simple data link layer that performs the flow control using the sliding window protocol**

**AIM:**

**Program:**

#include<stdio.h>

int main(){

int n,f,frames[30],i;

printf("Enter window size : ");

scanf("%d",&n);

printf("Enter number of frames to transmit: ");

scanf("%d",&f);

printf("Enter %d frames: \n",f);

for(i=1;i<=f;i++){

scanf("%d",&frames[i]);

}

printf("\nWith sliding window protocol the frames will be sent in the following manner (assuming no corruption of frames)\n\n");

printf("After sending %d frames at each stage sender waits for acknowledgement sent by the receiver\n\n",n);

for(i=1;i<=f;i++){

if(i%n==0){

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

printf("\nAcknowledgement of above frames sent is received by sender\n\n");

}

else{

//printf("recieved by sender\n");

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

}

}

if(f%n!=0){

printf("\nAcknowledgement of above frames sent is received by sender\n");

}

}

**Output:**

Enter window size: 3

Enter number of frames to transmit: 5 Enter 5 frames: 12 5 89 4 6

With sliding window protocol the frames will be sent in the following manner (assuming no corruption of frames)

After sending 3 frames at each stage sender waits for acknowledgement sent by the receiver 12 5 89

Acknowledgement of above frames sent is received by sender 4 6

Acknowledgement of above frames sent is received by sender

**B. Develop a simple data link layer that performs the flow control using the Go Back N protocol in “c”**

**AIM:**

**Program:**

#include<stdio.h> int main()

{

int window=0;

printf("enter Window size : ");

scanf("%d",&window);

int sent,ack,i=0;

while(1){

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

printf("frame Transmitted %d \n",sent);

sent++;

if(sent==window){

break;

}

}

printf("enter last received acknowledgment : ");

scanf("%d",&ack);

if(ack==window){

break;

}

else{

sent = ack;

}

}

**Output:**

enter window size 8

Frame 0 has been transmitted.

Frame 1 has been transmitted.

Frame 2 has been transmitted.

Frame 3 has been transmitted.

Frame 4 has been transmitted.

Frame 5 has been transmitted.

Frame 6 has been transmitted.

Frame 7 has been transmitted.

Please enter the last Acknowledgement received. 2

Frame 2 has been transmitted.

Frame 3 has been transmitted.

Frame 4 has been transmitted.

Frame 5 has been transmitted.

Frame 6 has been transmitted

Frame 7 has been transmitted

Please enter the last Acknowledgement received. 8

Experiment 3 :

### Take an example subnet of hosts and obtain a broadcast tree for the subnet.

**AIM:**

### Program:

#include <stdio.h>

int min();

int distance[20];

int n;

main()

{

int adj[20][20],adj1[20][20],flag[30];

int i,j,root,x;

int source,count=1,y=0;

printf("enter no of nodes");

scanf("%d",&n);

printf("enter the adjacency matrix");

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

{

for(j=0;j<n;j++)

{

scanf("%d",&adj[i][j]);

}

}

printf("enter the source for broadcasting");

scanf("%d",&source);

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

{

flag[i]=0;

}

for(root=0;root<n;root++)

{

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

{

distance[i]=adj[root][i];

}

x=min();

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

{

if(distance[i]==x)

{

adj1[root][i]=x;

adj1[i][root]=x;

}

else

{

adj1[root][i]=0;

}

}

}

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

{

for(j=0;j<n;j++)

{

if(adj1[i][j]!=0)

{

adj1[j][i]=adj[i][j];

}

}

}

printf("given adjacency matrix is");

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

{

for(j=0;j<n;j++)

{

printf("%d",adj[i][j]);

}

printf("\n");

}

printf("minimal spanning tree");

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

{

for(j=0;j<n;j++)

{

printf("%d ",adj1[i][j]);

}

printf("\n");

}

root=source;

flag[root]=1;

while(count!=y)

{

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

{

if(adj1[root][i]!=0 && flag[root]==1 && flag[i]!=1)

{

printf("%d sends message to %d \n",root,i);

flag[i]=1;

}

}

if(root<n-1)

{

root++;

}

else

{

root=0;

}

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

{

if(flag[i]==0)

{

break;

}

}

if(i==n)

{

count=y;

}

}

}

int min()

{

int i,j=0;

int mini;

int distance1[10];

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

{

if(distance[i]!=0)

{

distance1[j]=distance[i];

j++;

}

}

mini=distance1[0];

for(i=1;i<j;i++) {

if(distance1[i]<mini)

{

mini=distance1[i];

}

}

return(mini);

}

### Output:

1. Enter no of nodes2

Enter the adjacency matrix

0 2

2 0

enter the source for broadcasting1 given adjacency matrix is

0 2

2 0

minimal spanning tree is

0 2

1. 0

1 sends message to 0

1. Enter no of nodes 3

Enter the adjacency matrix

0 1 2

1 0 5

2 5 0

enter the source for broadcasting2

given adjacency matrix is

0 1 2

1 0 5

2 5 0

1

2 5

minimal spanning tree is

0 1 2

1 0 0

2 0 0

1

2

1. sends message to 0

0 sends message to 1

Experiment 4 :

### Implement distance vector routing algorithm for obtaining routing tables at each node.

### AIM:

**Program:**

#include <stdio.h>

struct node {

int dist[20];

int from[20];

} route[10];

int main() {

int dm[20][20], no;

printf("Enter no of nodes: ");

scanf("%d", &no);

printf("Enter the distance matrix:\n");

for (int i = 0; i < no; i++) {

for (int j = 0; j < no; j++) {

scanf("%d", &dm[i][j]);

/\* Set distance from i to i as 0 \*/

dm[i][i] = 0;

route[i].dist[j] = dm[i][j];

route[i].from[j] = j;

}

}

int flag;

do {

flag = 0;

for (int i = 0; i < no; i++) {

for (int j = 0; j < no; j++) {

for (int k = 0; k < no; k++) {

if ((route[i].dist[j]) > (route[i].dist[k] + route[k].dist[j])) {

route[i].dist[j] = route[i].dist[k] + route[k].dist[j];

route[i].from[j] = k;

flag = 1;

}

}

}

}

} while (flag);

for (int i = 0; i < no; i++) {

printf("\nRouter info for router: %d\n", i + 1);

printf("Dest\tNext Hop\tDist\n");

for (int j = 0; j < no; j++)

printf("%d\t%d\t\t%d\n", j+1, route[i].from[j]+1, route[i].dist[j]);

}

return 0;

}

**Output:**

Enter no of nodes: 3

Enter the distance matrix:

0 2 99

1 0 99

3 2 0

Router info for router: 1

Dest Next Hop Dist

1 1 0

2 2 2

3 3 99

Router info for router: 2

Dest Next Hop Dist

1 1 1

2 2 0

3 3 99

Router info for router: 3

Dest Next Hop Dist

1 1 3

2 2 2

3 3 0

\*/

### Experiment 5:

**Design the following**

a. TCP iterative Client and server application to reverse the given input sentence.

**Server Program**

// Server C code to reverse a

// string by sent from client

#include <netinet/in.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <unistd.h>

#define PORT 8090

// Driver code

int main()

{

    int server\_fd, new\_socket, valread;

    struct sockaddr\_in address;

    char str[100];

    int addrlen = sizeof(address);

    char buffer[1024] = { 0 };

    char\* hello = "Hello from server";

    // Creating socket file descriptor

    if ((server\_fd = socket(AF\_INET,

                          SOCK\_STREAM, 0)) == 0) {

        perror("socket failed");

        exit(EXIT\_FAILURE);

    }

    address.sin\_family = AF\_INET;

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

    address.sin\_port = htons(PORT);

    // Forcefully attaching socket to

    // the port 8090

    if (bind(server\_fd, (struct sockaddr\*)&address,

                          sizeof(address)) < 0) {

        perror("bind failed");

        exit(EXIT\_FAILURE);

    }

    // puts the server socket in passive mode

    if (listen(server\_fd, 3) < 0) {

        perror("listen");

        exit(EXIT\_FAILURE);

    }

    if ((new\_socket = accept(server\_fd,

                  (struct sockaddr\*)&address,

                  (socklen\_t\*)&addrlen)) < 0) {

        perror("accept");

        exit(EXIT\_FAILURE);

    }

    // read string send by client

    valread = read(new\_socket, str,

                   sizeof(str));

    int i, j, temp;

    int l = strlen(str);

    printf("\nString sent by client:%s\n", str);

    // loop to reverse the string

    for (i = 0, j = l - 1; i < j; i++, j--) {

        temp = str[i];

        str[i] = str[j];

        str[j] = temp;

    }

    // send reversed string to client

    // by send system call

    send(new\_socket, str, sizeof(str), 0);

    printf("\nModified string sent to client\n");

    return 0;

}

**Client Program:**

// C client code to send string to reverse

#include <arpa/inet.h>

#include <netinet/in.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <unistd.h>

#define PORT 8090

// Driver code

int main()

{

    struct sockaddr\_in address;

    int sock = 0, valread;

    struct sockaddr\_in serv\_addr;

    char str[100];

    printf("\nInput the string:");

    scanf("%[^\n]s", str);

    char buffer[1024] = { 0 };

    // Creating socket file descriptor

    if ((sock = socket(AF\_INET,

                       SOCK\_STREAM, 0))

        < 0) {

        printf("\n Socket creation error \n");

        return -1;

    }

    memset(&serv\_addr, '0', sizeof(serv\_addr));

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

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

    // Convert IPv4 and IPv6 addresses from

    // text to binary form 127.0.0.1 is local

    // host IP address, this address should be

    // your system local host IP address

    if (inet\_pton(AF\_INET, "127.0.0.1",

                  &serv\_addr.sin\_addr)

        <= 0) {

        printf("\nAddress not supported \n");

        return -1;

    }

    // connect the socket

    if (connect(sock, (struct sockaddr\*)&serv\_addr,

                sizeof(serv\_addr))

        < 0) {

        printf("\nConnection Failed \n");

        return -1;

    }

    int l = strlen(str);

    // send string to server side

    send(sock, str, sizeof(str), 0);

    // read string sent by server

    valread = read(sock, str, l);

    printf("%s\n", str);

    return 0;

}

**Output:**

enter the data to be send: hellow

line send

reverse of the given sentence is : wolleh

**5 B) TCP client and server application to transfer file.**

**Server program** #include <sys/socket.h> #include <netinet/in.h> #include <arpa/inet.h> #include <stdio.h> #include <stdlib.h> #include <unistd.h> #include <errno.h> #include <string.h> #include <sys/types.h> int main(void)

{

int listenfd = 0; int connfd = 0;

struct sockaddr\_in serv\_addr; char sendBuff[1025];

int numrv;

listenfd = socket(AF\_INET, SOCK\_STREAM, 0); printf("Socket retrieve success\n"); memset(&serv\_addr, '0', sizeof(serv\_addr));

memset(sendBuff, '0', sizeof(sendBuff));

serv\_addr.sin\_family = AF\_INET; serv\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY); serv\_addr.sin\_port = htons(5000);

[Type text] [Type text] [Type text]

bind(listenfd, (structsockaddr\*)&serv\_addr,sizeof(serv\_addr)); if(listen(listenfd, 10) == -1)

{

printf("Failed to listen\n"); return -1;

}

while(1)

{

connfd = accept(listenfd, (struct sockaddr\*)NULL ,NULL);

/\* Open the file that we wish to transfer \*/ FILE \*fp = fopen("fifoserver.c","rb"); if(fp==NULL)

{

printf("File opern error"); return 1;

}

/\* Read data from file and send it \*/ while(1){

/\* First read file in chunks of 256 bytes \*/ [Type text] [Type text] [Type text] unsigned char buff[256]={0};

int nread = fread(buff,1,256,fp); printf("Bytes read %d \n", nread);

/\* If read was success, send data. \*/ if(nread > 0)

{

printf("Sending \n"); write(connfd, buff, nread);

}

**Client Program:** #include <sys/socket.h> #include <sys/types.h> #include <netinet/in.h> #include <netdb.h> #include <stdio.h> #include <string.h> #include <stdlib.h> #include <unistd.h> #include <errno.h> #include <arpa/inet.h> int main(void)

{

int sockfd = 0;

int bytesReceived = 0; char recvBuff[256];

memset(recvBuff, '0', sizeof(recvBuff)); struct sockaddr\_in serv\_addr;

/\* Create a socket first \*/

if((sockfd = socket(AF\_INET, SOCK\_STREAM, 0))< 0)

{

printf("\n Error : Could not create socket \n"); return 1;

}

/\* Initialize sockaddr\_in data structure \*/ serv\_addr.sin\_family = AF\_INET; serv\_addr.sin\_port = htons(5000); // port serv\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

/\* Attempt a connection \*/

if(connect(sockfd, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr))<0)

{

printf("\n Error : Connect Failed \n"); [Type text] [Type text] [Type text] return 1;

}

/\* Create file where data will be stored \*/ FILE \*fp;

fp = fopen("fifoserver.c","ab"); if(NULL == fp)

{

printf("Error opening file"); return 1;

}

/\* Receive data in chunks of 256 bytes \*/ while((bytesReceived = read(sockfd, recvBuff, 256)) > 0)

{

printf("Bytes received %d\n",bytesReceived);

// recvBuff[n] = 0;

fwrite(recvBuff, 1,bytesReceived,fp);

// printf("%s \n", recvBuff);

}

if(bytesReceived < 0)

{

printf("\n Read Error \n");

}

return 0;

}

**Out put** :

Server side

Socket retrieve success Bytes read 256 Sending

Bytes read 256 Sending

Bytes read 256 Sending

Bytes read 28 Sending

End of file Client side

Bytes received 256

Bytes received 256

Bytes received 256

Bytes received 28

**5 C) TCP concurrent server to convert a given text into upper case using multiplexing system call “select”.**

**Server program:**

#include <stdio.h> #include <stdlib.h> #include <unistd.h> #include <errno.h> #include <string.h> #include <sys/types.h> #include <sys/stat.h> #include <sys/select.h> #include <sys/time.h> #include <sys/socket.h> #include <netinet/in.h> #include <arpa/inet.h> #include <fcntl.h> #define MAXLINE 100

#define SERV\_PORT 13153 int main(int argc, char \*\*argv)

{

int k, i, maxi, maxfd, listenfd, connfd, sockfd; int nready, client[FD\_SETSIZE];

ssize\_t n;

fd\_set rset, allset;

char line[MAXLINE],buf[100]; socklen\_t clilen;

struct sockaddr\_in cliaddr, servaddr;

listenfd = socket(AF\_INET, SOCK\_STREAM, 0); if (listenfd < 0 )

{

perror("socket" ); exit(1);

}

bzero(&servaddr, sizeof(servaddr)); servaddr.sin\_family = AF\_INET; servaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY); servaddr.sin\_port = htons(SERV\_PORT);

bind(listenfd, (struct sockaddr \*) &servaddr, sizeof(servaddr)); listen(listenfd,5);

maxfd = listenfd; /\* initialize \*/

maxi = -1; /\* index into client[] array \*/ for (i = 0; i < FD\_SETSIZE; i++)

client[i] = -1; /\* -1 indicates available entry \*/ FD\_ZERO(&allset);

FD\_SET(listenfd, &allset);

/\* end fig01 \*/

/\* include fig02 \*/ for ( ; ; ) {

printf("Server:I am waiting Start of Main Loop\n");

rset = allset; /\* structure assignment \*/

nready = select(maxfd+1, &rset, NULL, NULL, NULL);

if (FD\_ISSET(listenfd, &rset)) { /\* new client connection \*/ clilen = sizeof(cliaddr);

connfd = accept(listenfd, (struct sockaddr \*) &cliaddr, &clilen);

#ifdef NOTDEF

printf("new client: %s, port %d\n", inet\_ntop(AF\_INET, &cliaddr.sin\_addr, buf, NULL), ntohs(cliaddr.sin\_port));

#endif

for (i = 0; i < FD\_SETSIZE; i++)

if (client[i] < 0) {

client[i] = connfd; /\* save descriptor \*/ break;

}

if (i == FD\_SETSIZE)

{

printf("too many clients"); exit(0);

}

FD\_SET(connfd, &allset); /\* add new descriptor to set \*/ if (connfd > maxfd)

maxfd = connfd; /\* for select \*/ if (i > maxi)

maxi = i; /\* max index in client[] array \*/ if (--nready <= 0)

continue; /\* no more readable descriptors

\*/

}

for (i = 0; i <= maxi; i++) { /\* check all clients for data \*/ if ( (sockfd = client[i]) < 0)

continue;

if (FD\_ISSET(sockfd, &rset)) {

if ( (n = read(sockfd, line, MAXLINE)) == 0) {

/\*4connection closed by client \*/ close(sockfd);

FD\_CLR(sockfd, &allset); client[i] = -1;

} else

{

printf("\n output at server\n"); for(k=0;line[k]!='\0';k++)

printf("%c",toupper(line[k])); write(sockfd, line, n);

}

if (--nready <= 0)

break; /\* no more readable descriptors

\*/

}

}

}

}

/

**Client program :**

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

#define MAXBUFFER 1024 void sendstring(int , char \*); int main( int C, char \*V[] )

{

int sd,fd; char c;

struct sockaddr\_in serveraddress; char text[100];

int i=0;

sd = socket( AF\_INET, SOCK\_STREAM, 0 ); if( sd < 0 ) {

perror( "socket" ); exit( 1 );

}

if (V[1] == NULL ) {

printf ("PL specfiy the server's IP Address \n"); exit(0);

}

if (V[2] == NULL ) {

printf ("PL specify the server's Port No \n"); exit(0);

}

// if (V[3] == NULL ) {

// printf ("PL specfiy the string to be send to the server \n");

// exit(0);

// }

memset( &serveraddress, 0, sizeof(serveraddress) ); serveraddress.sin\_family = AF\_INET; serveraddress.sin\_port = htons(atoi(V[2]));//PORT NO

serveraddress.sin\_addr.s\_addr = inet\_addr(V[1]);//ADDRESS if (connect(sd,(struct sockaddr\*)&serveraddress, sizeof(serveraddress))<0)

{

printf("Cannot Connect to server"); exit(1);

}

printf("enter sentence to end enter #"); while(1)

{

c=getchar(); if(c=='#') break; text[i++]=c;

}

text[i]='\0';

sendstring(sd,text); close(sd);

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* FUNCTION NAME:sendstring
* DESCRIPTION: sends a string over the socket .
* NOTES : No Error Checking is done .
* RETURNS :void

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void sendstring(

int sd, /\*Socket Descriptor\*/

char \*fname) /\*Array Containing the string \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

{ int n , byteswritten=0 , written ; char buffer[MAXBUFFER]; strcpy(buffer , fname); n=strlen(buffer);

while (byteswritten<n)

{

written=write(sd , buffer+byteswritten,(n-byteswritten)); byteswritten+=written;

29

}

printf("String : %s sent to server \n",buffer);

}

**Execution Steps:**

b) Concurrent Server Application Using Select. Compiling and running server.

root@localhost week7and8]# cc tcpservselect01.c [root@localhost week7and8]# mv a.out tcpservselect1 [root@localhost week7and8]# ./tcpservselect1

Server:I am waiting Start of Main Loop

Server:I am waiting Start of Main Loop

output at server

A B C DServer:I am waiting Start of Main Loop

output at server

A B C DServer:I am waiting Start of Main Loop

Server:I am waiting Start of Main Loop

Compiling and running Client.

root@localhost week7and8]# ./tcpclient 127.0.0.1 13153 enter sentence to end enter #abcd#

String : abcd sent to server

1. **D) TCP concurrent server to echo given set of sentences using poll functions. Server program**

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

#include <limits.h> /\* for OPEN\_MAX \*/ #include <poll.h>

#include <errno.h> #define MAXLINE 100

#define SERV\_PORT 13154

#define POLLRDNORM 5

#define INFTIM 5

#define OPEN\_MAX 5

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

{

int k, i, maxi, listenfd, connfd, sockfd; int nready;

ssize\_t n;

char line[MAXLINE]; socklen\_t clilen;

struct pollfd client[OPEN\_MAX]; struct sockaddr\_in cliaddr, servaddr;

listenfd = socket(AF\_INET, SOCK\_STREAM, 0); bzero(&servaddr, sizeof(servaddr)); servaddr.sin\_family =

AF\_INET; servaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY); servaddr.sin\_port = htons(SERV\_PORT);

bind(listenfd, (struct sockaddr \*) &servaddr, sizeof(servaddr)); listen(listenfd, 5);

client[0].fd = listenfd;

client[0].events = POLLRDNORM; for (i = 1; i < OPEN\_MAX; i++)

client[i].fd = -1; /\* -1 indicates available entry \*/ maxi = 0; /\* max index into client[] array \*/

/\* end fig01 \*/

/\* include fig02 \*/ for ( ; ; ) {

nready = poll(client, maxi+1, INFTIM);

if (client[0].revents & POLLRDNORM) { /\* new client connection \*/ clilen = sizeof(cliaddr);

connfd = accept(listenfd, (struct sockaddr \*) &cliaddr, &clilen); #ifdef NOTDEF

printf("new client: %s\n", sock\_ntop((struct sockaddr \*) &cliaddr, clilen));

#endif

for (i = 1; i < OPEN\_MAX; i++) if (client[i].fd < 0) {

client[i].fd = connfd; /\* save descriptor \*/ break;

}

if (i == OPEN\_MAX)

{

printf("too many clients"); exit(0);

}

client[i].events = POLLRDNORM; if (i > maxi)

maxi = i; /\* max index in client[] array \*/ if (--nready <= 0)

continue; /\* no more readable descriptors

\*/

}

for (i = 1; i <= maxi; i++) { /\* check all clients for data \*/ if ( (sockfd = client[i].fd) < 0)

continue;

if (client[i].revents & (POLLRDNORM | POLLERR)) { if ( (n = read(sockfd, line, MAXLINE)) < 0) {if (errno == ECONNRESET) {

/\*4connection reset by client \*/ #ifdef NOTDEF

printf("client[%d] aborted connection\n", i); #endif

close(sockfd); client[i].fd = -1;

} else

printf("readline error");

} else if (n == 0) {

/\*4connection closed by client \*/ #ifdef NOTDEF

printf("client[%d] closed connection\n", i); #endif

close(sockfd); client[i].fd = -1;

} else{ printf("\n data from client is \n"); k=strlen(line);

printf(" length=%d data = %s\n", k,line);

//write(sockfd, line, n);

strcpy(line," ");

}

if (--nready <= 0)

break; /\* no more readable descriptors

\*/

}

}

}

}

**client program:**

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

#include <sys/stat.h> #include <sys/socket.h> #include <netinet/in.h> #include <arpa/inet.h> #include <fcntl.h>

#define MAXBUFFER 1024

void sendstring(int , char \*); int main( int C, char \*V[] )

{

int sd,fd; char c;

struct sockaddr\_in serveraddress; char text[100];

int i=0;

sd = socket( AF\_INET, SOCK\_STREAM, 0 ); if( sd < 0 ) {

perror( "socket" ); exit( 1 );

}

if (V[1] == NULL ) {

printf ("PL specfiy the server's IP Address \n"); exit(0);

}

if (V[2] == NULL ) {

printf ("PL specify the server's Port No \n"); exit(0);

}

// if (V[3] == NULL ) {

// printf ("PL specfiy the string to be send to the server \n");

// exit(0);

// }

memset( &serveraddress, 0, sizeof(serveraddress) ); serveraddress.sin\_family = AF\_INET; serveraddress.sin\_port = htons(atoi(V[2]));//PORT NO

serveraddress.sin\_addr.s\_addr = inet\_addr(V[1]);//ADDRESS if (connect(sd,(struct sockaddr\*)&serveraddress, sizeof(serveraddress))<0)

printf("Cannot Connect to server"); exit(1);

}

printf("enter sentence to end enter #"); while(1)

{

c=getchar(); if(c=='#') break; text[i++]=c;

}

text[i]='\0'; sendstring(sd,text); close(sd);

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* FUNCTION NAME:sendstring
* DESCRIPTION: sends a string over the socket .
* NOTES : No Error Checking is done .
* RETURNS :void

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void sendstring(

int sd, /\*Socket Descriptor\*/

char \*fname) /\*Array Containing the string \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

{ int n , byteswritten=0 , written ; char buffer[MAXBUFFER]; strcpy(buffer , fname); n=strlen(buffer);

while (byteswritten<n)

{

written=write(sd , buffer+byteswritten,(n-byteswritten)); byteswritten+=written;

}

printf("String : %s sent to server \n",buffer);

}

**Concurrent Server Application Using Poll.**

Compiling and running server. [root@localhost week8]# cc tcpservpoll01.c [root@localhost week8]# mv a.out pollserv [root@localhost week8]# ./pollserv

data from client is

data = aaaaaaaaaaaaaaaaaaaaaaaa Compiling and running Client. [root@localhost week8]#cc democlient.c [root@localhost week8]#mv a.out client

[root@localhost week8]# ./client 127.0.0.1 13153

enter sentence to end enter #aaaaaaaaaaaaaaaaaaaaaaaa# String : aaaaaaaaaaaaaaaaaaaaaaaa sent to server

### Experiment 6 :

**Design the following**

1. **A) UDP Client and server application to reverse the given input sentence. 2018-2019 173. Server program :**

#include <stdio.h> #include <stdlib.h> #include <errno.h> #include <string.h> #include <sys/types.h> #include <sys/socket.h> #include <netinet/in.h> #include <arpa/inet.h> #include <sys/wait.h> #include <fcntl.h> #include <signal.h> #define BUFSIZE 512

#define MYPORT 11710

#define MAXNAME 100 int main(int C, char \*\*V )

{

int sd,n,ret;

struct sockaddr\_in serveraddress,cliaddr; socklen\_t length;

char clientname[MAXNAME],datareceived[BUFSIZE]; sd = socket( AF\_INET, SOCK\_DGRAM, 0 );

if( sd < 0 ) { perror( "socket" ); exit( 1 );

}

memset( &serveraddress, 0, sizeof(serveraddress) ); memset( &cliaddr, 0, sizeof(cliaddr) ); serveraddress.sin\_family = AF\_INET; serveraddress.sin\_port = htons(MYPORT);//PORT NO

serveraddress.sin\_addr.s\_addr = htonl(INADDR\_ANY);//IP ADDRESS ret=bind(sd,(struct sockaddr\*)&serveraddress,sizeof(serveraddress)); if(ret<0)

{

perror("BIND FAILS");

exit(1);

}

for(;;)

{

printf("I am waiting\n");

/\*Received a datagram\*/ length=sizeof(cliaddr); n=recvfrom(sd,datareceived,BUFSIZE,0, (struct sockaddr\*)&cliaddr , &length); printf("Data Received from %s\n", inet\_ntop(AF\_INET,&cliaddr.sin\_addr, clientname,sizeof(clientname)));

/\*Sending the Received datagram back\*/ datareceived[n]='\0';

printf("I have received %s\n",datareceived); sendto(sd,datareceived,n,0,(struct sockaddr \*)&cliaddr,length);

}

}

client program :

#include <stdio.h> #include <stdlib.h> #include <errno.h> #include <string.h> #include <sys/types.h> #include <sys/socket.h> #include <netinet/in.h> #include <arpa/inet.h> #include <fcntl.h> #include <signal.h> #include <unistd.h> #define BUFSIZE 512 static void sig\_usr(int);

void str\_cli(FILE \*fp , int sockfd , struct sockaddr \*server , socklen\_t len); int main( int C, char \*argv[] )

{

int sd;

struct sockaddr\_in serveraddress;

/\*Installing signal Handlers\*/ signal(SIGPIPE,sig\_usr); signal(SIGINT,sig\_usr);

if (NULL==argv[1])

{

printf("Please enter the IP Address of the server\n"); exit(0);

}

if (NULL==argv[2])

{

printf("Please enter the Port Number of the server\n"); exit(0);

}

sd = socket( AF\_INET, SOCK\_DGRAM, 0 ); if( sd < 0 )

{

perror( "socket" ); exit( 1 );

}

memset( &serveraddress, 0, sizeof(serveraddress) ); serveraddress.sin\_family = AF\_INET; serveraddress.sin\_port = htons(atoi(argv[2]));//PORT NO

serveraddress.sin\_addr.s\_addr = inet\_addr(argv[1]);//ADDRESS printf("Client Starting service\n");

printf("Enter Data For the server\n"); str\_cli(stdin,sd ,(struct sockaddr \*)&serveraddress, sizeof(serveraddress));

}

void str\_cli(FILE \*fp, /\*Here to be used as stdin as argument\*/ int sockfd ,

struct sockaddr \*to ,socklen\_t length) /\*Connection Socket \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

{

int maxdes,n;

fd\_set rset;

char sendbuf[BUFSIZE] , recvbuf[BUFSIZE] ,servername[100]; struct sockaddr\_in serveraddr;

socklen\_t slen; FD\_ZERO(&rset);

maxdes=(sockfd>fileno(fp)?sockfd+1:fileno(fp)+1); for(;;){

FD\_SET(fileno(fp) , &rset); FD\_SET(sockfd , &rset); select(maxdes,&rset,NULL,NULL,NULL); if(FD\_ISSET(sockfd , & rset))

{

slen=sizeof(serveraddr); n=recvfrom(sockfd,recvbuf,BUFSIZE,0, (struct sockaddr\*)&serveraddr,&slen); printf("Data Received from server %s:\n", inet\_ntop(AF\_INET,&serveraddr.sin\_addr, servername,sizeof(servername))); write(1,recvbuf,n);

printf("Enter Data For the server\n");

}

if(FD\_ISSET(fileno(fp) , & rset))

{

/\*Reading data from the keyboard\*/ fgets(sendbuf,BUFSIZE,fp);

n = strlen (sendbuf);

/\*Sending the read data over socket\*/ sendto(sockfd,sendbuf,n,0,to,length); printf("Data Sent To Server\n");

}

}

}

UDP Client Server Application. Compiling and running server. [user@localhost week9]$ cc udp\_server.c

[user@localhost week9]$ mv a.out udp\_server

[user@localhost week9]$ ./ udp\_server I am waiting

Data Received from 127.0.0.1 I have received abcd efgh

rev is hgfe dcba

I am waiting

Compiling and running client. user@localhost week9]$ cc udp\_client.c [user@localhost week9]$ mv a.out udp\_client

[user@localhost week9]$ ./ udp\_client 127.0.0.1 11710 Client Starting service

Enter Data For the server abcd efgh

Data Sent To Server

Data Received from server 127.0.0.1: abcd efgh

Enter Data For the server

**6 B) UDP Client and server to transfer file**

// server code for UDP socket programming #include <arpa/inet.h>

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

#define IP\_PROTOCOL 0

#define PORT\_NO 15050

#define NET\_BUF\_SIZE 32 #define cipherKey 'S' #define sendrecvflag 0

#define nofile "File Not Found!"

// function to clear buffer void clearBuf(char\* b)

{

int i;

for (i = 0; i < NET\_BUF\_SIZE; i++) b[i] = '\0';

}

// function to encrypt char Cipher(char ch)

{

return ch ^ cipherKey;

}

// function sending file

int sendFile(FILE\* fp, char\* buf, int s)

{

int i, len;

if (fp == NULL) {

strcpy(buf, nofile); len = strlen(nofile); buf[len] = EOF;

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

buf[i] = Cipher(buf[i]); return 1;

}

char ch, ch2;

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

ch = fgetc(fp); ch2 = Cipher(ch); buf[i] = ch2;

if (ch == EOF)

return 1;

}

return 0;

}

// driver code int main()

{

int sockfd, nBytes;

struct sockaddr\_in addr\_con; int addrlen = sizeof(addr\_con);

addr\_con.sin\_family = AF\_INET; addr\_con.sin\_port = htons(PORT\_NO); addr\_con.sin\_addr.s\_addr = INADDR\_ANY; char net\_buf[NET\_BUF\_SIZE];

FILE\* fp;

// socket()

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

if (sockfd < 0)

printf("\nfile descriptor not received!!\n");

else

printf("\nfile descriptor %d received\n", sockfd);

// bind()

if (bind(sockfd, (struct sockaddr\*)&addr\_con, sizeof(addr\_con)) == 0) printf("\nSuccessfully binded!\n");

else

printf("\nBinding Failed!\n");

while (1) {

printf("\nWaiting for file name...\n");

// receive file name clearBuf(net\_buf);

nBytes = recvfrom(sockfd, net\_buf,

NET\_BUF\_SIZE, sendrecvflag,

(struct sockaddr\*)&addr\_con, &addrlen);

fp = fopen(net\_buf, "r");

printf("\nFile Name Received: %s\n", net\_buf);

if (fp == NULL)

printf("\nFile open failed!\n");

else

printf("\nFile Successfully opened!\n");

while (1) {

// process

if (sendFile(fp, net\_buf, NET\_BUF\_SIZE)) { sendto(sockfd, net\_buf, NET\_BUF\_SIZE,

sendrecvflag,

(struct sockaddr\*)&addr\_con, addrlen);

break;

}

// send

sendto(sockfd, net\_buf, NET\_BUF\_SIZE, sendrecvflag,

(struct sockaddr\*)&addr\_con, addrlen); clearBuf(net\_buf);

}

if (fp != NULL)

fclose(fp);

}

return 0;

}

// client code for UDP socket programming #include <arpa/inet.h>

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

#define IP\_PROTOCOL 0

#define IP\_ADDRESS "127.0.0.1" // localhost #define PORT\_NO 15050

#define NET\_BUF\_SIZE 32 #define cipherKey 'S' #define sendrecvflag 0

// function to clear buffer void clearBuf(char\* b)

{

int i;

for (i = 0; i < NET\_BUF\_SIZE; i++) b[i] = '\0';

}

// function for decryption char Cipher(char ch)

{

return ch ^ cipherKey;

}

// function to receive file

int recvFile(char\* buf, int s)

{

int i; char ch;

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

ch = buf[i];

ch = Cipher(ch); if (ch == EOF)

return 1;

else

}

printf("%c", ch);

return 0;

}

// driver code int main()

{

int sockfd, nBytes;

struct sockaddr\_in addr\_con; int addrlen = sizeof(addr\_con);

addr\_con.sin\_family = AF\_INET; addr\_con.sin\_port = htons(PORT\_NO);

addr\_con.sin\_addr.s\_addr = inet\_addr(IP\_ADDRESS); char net\_buf[NET\_BUF\_SIZE];

FILE\* fp;

// socket()

sockfd = socket(AF\_INET, SOCK\_DGRAM,

IP\_PROTOCOL);

if (sockfd < 0)

printf("\nfile descriptor not received!!\n");

else

printf("\nfile descriptor %d received\n", sockfd);

while (1) {

printf("\nPlease enter file name to receive:\n"); scanf("%s", net\_buf);

sendto(sockfd, net\_buf, NET\_BUF\_SIZE, sendrecvflag, (struct sockaddr\*)&addr\_con, addrlen);

printf("\n---------Data Received \n");

while (1) {

// receive clearBuf(net\_buf);

nBytes = recvfrom(sockfd, net\_buf, NET\_BUF\_SIZE,

sendrecvflag, (struct sockaddr\*)&addr\_con, &addrlen);

// process

if (recvFile(net\_buf, NET\_BUF\_SIZE)) { break;

}

}

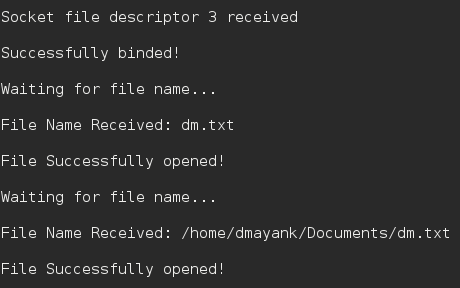
printf("\n \n");

}

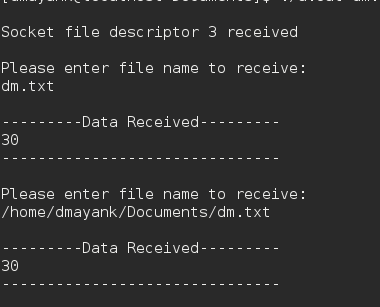
return 0;

}

Server:



Client :



**Experiment 7: Programs to demonstrate the usage of Advanced socket system calls like getsockopt( ), setsockopt(), getpeername ( ),getsockname( ),readv( ) and writev( ).**

Elementary Socket System Calls:

### Socket:

To do network I/O, the first thing a process must do is to call the socket system call, specifying the

type of communication protocol desired. #include <sys/types.h>

#include <sys/socket.h>

int socket(int family, int type, int protocol);

The socket type is one of the following:

SOCK\_STREAM stream socket SOCK\_DGRAM datagram socket SOCK\_RAW raw socket SOCK\_SEQPACKET sequenced packet socket

SOCK\_RDM reliably delivered message socket (not implemented yet)

### readv and writev system calls:

These two functions are similar to read and write , but readv and writev let us read into or write from one or more buffers with a single function call. These operations are called scatter read (since the input data is scattered into multiple application buffers) and gather write (since multiple buffers are gathered for a single output operation).

#include <sys/uio.h>

int readv(int fd , struct iovec iov[] , int iovcount) ; int writev(int fd , struct iovec iov[] , int iovcount) ;

These two system calls use the following structure that is defined in

<syst/uio.h>:

struct iovec{

caddr\_t iov\_base; /\*strating address of buffer\*/ int iov\_len; /\*size of buffer in size\*/

The writev system call write the buffers specified by iov[0], iov[1], throughiov[iovcount-1].



The readv sytem call does the input equivalent. It always fills one buffer (as specified but the iov\_len value) before proceeding to the next buffer in the iov array.

Both system calls return the total number of bytes read and written.

1. **getpeername** - get the name of the peer socket #include <sys/socket.h>

int getpeername(int socket, struct sockaddr \*address,socklen\_t \*address\_len);

The getpeername() function retrieves the peer address of the specified socket, stores this address

in the sockaddr structure pointed to by the address argument, and stores the length of this address in the object pointed to by the address\_len argument.

If the actual length of the address is greater than the length of the supplied sockaddr structure, the stored address will be truncated.

If the protocol permits connections by unbound clients, and the peer is not bound, then the value stored in the object pointed to by address is unspecified.

1. **getsockname** - get the socket name #include <sys/socket.h>

int getsockname(int socket, struct sockaddr \*address,socklen\_t \*address\_len);

The getsockname() function retrieves the locally-bound name of the specified socket,stores this address in the sockaddr structure pointed to by the address argument, and stores the length of this address in the object pointed to by the address\_len argument.

If the actual length of the address is greater than the length of the supplied sockaddr structure, the stored address will be truncated.

If the socket has not been bound to a local name, the value stored in the object pointed to by address is unspecified.

1. **getsockopt and setsockopt** allow socket options values to be queried and set, respectively.

int getsockopt (sockid, level, optName, optVal, optLen); sockid: integer, socket descriptor



level: integer, the layers of the protocol stack (socket, TCP, IP) optName: integer, option

optVal: pointer to a buffer; upon return it contains the value of the specified option optLen: integer, in-out parameter it returns -1 if an error occured

int setsockopt (sockid, level, optName, optVal, optLen); optLen is now only an input parameter

**Experiment 8: Implementation of concurrent chat server that allows current logged in users to communicate one with other**

Program Objective: Determine the number of Users currently logged in and establish chat session with them.

Program Description:

The command that counts the number of users logged in is who |wc –l. Using this command, determine the number of users currently available for chat.

Steps Server:

Include appropriate header files. Create a TCP Socket.



Fill in the socket address structure (with server information) Bind the address and port using bind() system call.

Server executes listen() system call to indicate its willingness to receive connections. Accept the next completed connection from the client process by using an accept()

system call.

Receive a message from the Client using recv() system call.



Send the reply of the message made by the client using send() system call.

Client

Create a TCP Socket.



Fill in the socket address structure (with server information) Establish connection to the Server using connect() system call. Send a chat message to the Server using send() system call.

Receive the reply message made to the server using recv() system call. Write the result thus obtained on the standard output.

Server program :

#include <stdio.h> #include <netdb.h> #include <netinet/in.h> #include <stdlib.h> #include <string.h> #include <sys/socket.h> #include <sys/types.h> #define MAX 80

#define PORT 8080 #define SA struct sockaddr

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

{

char buff[MAX];

int n;

// infinite loop for chat for (;;) {

bzero(buff, MAX);

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

// print buffer which contains the client contents printf("From client: %s\t To client : ", buff); bzero(buff, MAX);

n = 0;

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

;

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

// if msg contains "Exit" then server exit and chat ended. if (strncmp("exit", buff, 4) == 0) {

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

}

}

}

// Driver function int main()

{

int sockfd, connfd, len;

struct sockaddr\_in servaddr, cli;

// socket create and verification

sockfd = socket(AF\_INET, SOCK\_STREAM, 0); if (sockfd == -1) {

printf("socket creation failed...\n"); exit(0);

}

else

printf("Socket successfully created..\n");

bzero(&servaddr, sizeof(servaddr));

// assign IP, PORT servaddr.sin\_family = AF\_INET;

servaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY); servaddr.sin\_port = htons(PORT);

// Binding newly created socket to given IP and verification if ((bind(sockfd, (SA\*)&servaddr, sizeof(servaddr))) != 0) {

printf("socket bind failed...\n"); exit(0);

}

else

printf("Socket successfully binded..\n");

// Now server is ready to listen and verification if ((listen(sockfd, 5)) != 0) {

printf("Listen failed...\n"); exit(0);

}

else

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

len = sizeof(cli);

// Accept the data packet from client and verification connfd = accept(sockfd, (SA\*)&cli, &len);

if (connfd < 0) {

printf("server accept failed...\n"); exit(0);

}

else

printf("server accept the client...\n");

// Function for chatting between client and server func(connfd);

// After chatting close the socket close(sockfd);

}

Client program : #include <netdb.h> #include <stdio.h> #include <stdlib.h> #include <string.h> #include <sys/socket.h> #define MAX 80

#define PORT 8080 #define SA struct sockaddr void func(int sockfd)

{

char buff[MAX]; int n;

for (;;) {

bzero(buff, sizeof(buff)); printf("Enter the string : "); n = 0;

while ((buff[n++] = getchar()) != '\n')

;

write(sockfd, buff, sizeof(buff)); bzero(buff, sizeof(buff)); read(sockfd, buff, sizeof(buff)); printf("From Server : %s", buff);

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

printf("Client Exit...\n"); break;

}

}

}

int main()

{

int sockfd, connfd;

struct sockaddr\_in servaddr, cli;

// socket create and verification

sockfd = socket(AF\_INET, SOCK\_STREAM, 0); if (sockfd == -1) {

printf("socket creation failed...\n"); exit(0);

}

else

printf("Socket successfully created..\n");

bzero(&servaddr, sizeof(servaddr));

// assign IP, PORT servaddr.sin\_family = AF\_INET;

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

// connect the client socket to server socket

if (connect(sockfd, (SA\*)&servaddr, sizeof(servaddr)) != 0) { printf("connection with the server failed...\n"); exit(0);

}

else

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

// function for chat func(sockfd);

// close the socket close(sockfd);

}

### Compilation –

Server side:

gcc server.c -o server

./server Client side:

gcc client.c -o client

./client

### Output –

Server side:

Socket successfully created.. Socket successfully binded.. Server listening..

server accept the client... From client: hi

To client : hello From client: exit

To client : exit Server Exit...

Client side:

Socket successfully created.. connected to the server..

Enter the string : hi From Server : hello Enter the string : exit From Server : exit Client Exit...

**Experiment 9 : Implementation of DNS.**

#include<stdio.h> #include<stdlib.h> #include<errno.h> #include<netdb.h> #include<sys/types.h> #include<sys/socket.h>

#include<netinet/in.h>int main(int argc,char \*argv[1])

{

struct hostent \*hen;if(argc!=2)

{

fprintf(stderr,"Enter the hostname \n");exit(1);

}

hen=gethostbyname(argv[1]); if(hen==NULL){fprintf(stderr,"Host not found \n");

}

printf("Hostname is %s \n",hen->h\_name);

printf("IP address is %s \n",inet\_ntoa(\*((struct in\_addr \*)hen->h\_addr)));

}

Input: ./a.out [www.google.com](http://www.google.com/) Output:

Host name is [www.google.com](http://www.google.com/) IP Address is 173.194.73.99

**Experiment 10 : Implementation of Ping service.**

// C program to Implement Ping

// C program to Implement Ping

// compile as -o ping

// run as sudo ./ping <hostname>

#include <stdio.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <netdb.h>

#include <unistd.h>

#include <string.h>

#include <stdlib.h>

#include <netinet/ip\_icmp.h>

#include <time.h>

#include <fcntl.h>

#include <signal.h>

#include <time.h>

// Define the Packet Constants

// ping packet size

#define PING\_PKT\_S 64

// Automatic port number

#define PORT\_NO 0

// Automatic port number

#define PING\_SLEEP\_RATE 1000000 x

// Gives the timeout delay for receiving packets

// in seconds

#define RECV\_TIMEOUT 1

// Define the Ping Loop

int pingloop=1;

// ping packet structure

struct ping\_pkt

{

struct icmphdr hdr;

char msg[PING\_PKT\_S-sizeof(struct icmphdr)];

};

// Calculating the Check Sum

unsigned short checksum(void \*b, int len)

{ unsigned short \*buf = b;

unsigned int sum=0;

unsigned short result;

for ( sum = 0; len > 1; len -= 2 )

sum += \*buf++;

if ( len == 1 )

sum += \*(unsigned char\*)buf;

sum = (sum >> 16) + (sum & 0xFFFF);

sum += (sum >> 16);

result = ~sum;

return result;

}

// Interrupt handler

void intHandler(int dummy)

{

pingloop=0;

}

// Performs a DNS lookup

char \*dns\_lookup(char \*addr\_host, struct sockaddr\_in \*addr\_con)

{

printf("\nResolving DNS..\n");

struct hostent \*host\_entity;

char \*ip=(char\*)malloc(NI\_MAXHOST\*sizeof(char));

int i;

if ((host\_entity = gethostbyname(addr\_host)) == NULL)

{

// No ip found for hostname

return NULL;

}

//filling up address structure

strcpy(ip, inet\_ntoa(\*(struct in\_addr \*)

host\_entity->h\_addr));

(\*addr\_con).sin\_family = host\_entity->h\_addrtype;

(\*addr\_con).sin\_port = htons (PORT\_NO);

(\*addr\_con).sin\_addr.s\_addr = \*(long\*)host\_entity->h\_addr;

return ip;

}

// Resolves the reverse lookup of the hostname

char\* reverse\_dns\_lookup(char \*ip\_addr)

{

struct sockaddr\_in temp\_addr;

socklen\_t len;

char buf[NI\_MAXHOST], \*ret\_buf;

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

temp\_addr.sin\_addr.s\_addr = inet\_addr(ip\_addr);

len = sizeof(struct sockaddr\_in);

if (getnameinfo((struct sockaddr \*) &temp\_addr, len, buf,

sizeof(buf), NULL, 0, NI\_NAMEREQD))

{

printf("Could not resolve reverse lookup of hostname\n");

return NULL;

}

ret\_buf = (char\*)malloc((strlen(buf) +1)\*sizeof(char) );

strcpy(ret\_buf, buf);

return ret\_buf;

}

// make a ping request

void send\_ping(int ping\_sockfd, struct sockaddr\_in \*ping\_addr,

char \*ping\_dom, char \*ping\_ip, char \*rev\_host)

{

int ttl\_val=64, msg\_count=0, i, addr\_len, flag=1,

msg\_received\_count=0;

struct ping\_pkt pckt;

struct sockaddr\_in r\_addr;

struct timespec time\_start, time\_end, tfs, tfe;

long double rtt\_msec=0, total\_msec=0;

struct timeval tv\_out;

tv\_out.tv\_sec = RECV\_TIMEOUT;

tv\_out.tv\_usec = 0;

clock\_gettime(CLOCK\_MONOTONIC, &tfs);

// set socket options at ip to TTL and value to 64,

// change to what you want by setting ttl\_val

if (setsockopt(ping\_sockfd, SOL\_IP, IP\_TTL,

&ttl\_val, sizeof(ttl\_val)) != 0)

{

printf("\nSetting socket options

to TTL failed!\n");

return;

}

else

{

printf("\nSocket set to TTL..\n");

}

// setting timeout of recv setting

setsockopt(ping\_sockfd, SOL\_SOCKET, SO\_RCVTIMEO,

(const char\*)&tv\_out, sizeof tv\_out);

// send icmp packet in an infinite loop

while(pingloop)

{

// flag is whether packet was sent or not

flag=1;

//filling packet

bzero(&pckt, sizeof(pckt));

pckt.hdr.type = ICMP\_ECHO;

pckt.hdr.un.echo.id = getpid();

for ( i = 0; i < sizeof(pckt.msg)-1; i++ )

pckt.msg[i] = i+'0';

pckt.msg[i] = 0;

pckt.hdr.un.echo.sequence = msg\_count++;

pckt.hdr.checksum = checksum(&pckt, sizeof(pckt));

usleep(PING\_SLEEP\_RATE);

//send packet

clock\_gettime(CLOCK\_MONOTONIC, &time\_start);

if ( sendto(ping\_sockfd, &pckt, sizeof(pckt), 0,

(struct sockaddr\*) ping\_addr,

sizeof(\*ping\_addr)) <= 0)

{

printf("\nPacket Sending Failed!\n");

flag=0;

}

//receive packet

addr\_len=sizeof(r\_addr);

if ( recvfrom(ping\_sockfd, &pckt, sizeof(pckt), 0,

(struct sockaddr\*)&r\_addr, &addr\_len) <= 0

&& msg\_count>1)

{

printf("\nPacket receive failed!\n");

}

else

{

clock\_gettime(CLOCK\_MONOTONIC, &time\_end);

double timeElapsed = ((double)(time\_end.tv\_nsec -

time\_start.tv\_nsec))/1000000.0

rtt\_msec = (time\_end.tv\_sec-

time\_start.tv\_sec) \* 1000.0

+ timeElapsed;

// if packet was not sent, don't receive

if(flag)

{

if(!(pckt.hdr.type ==69 && pckt.hdr.code==0))

{

printf("Error..Packet received with ICMP

type %d code %d\n",

pckt.hdr.type, pckt.hdr.code);

}

else

{

printf("%d bytes from %s (h: %s)

(%s) msg\_seq=%d ttl=%d

rtt = %Lf ms.\n",

PING\_PKT\_S, ping\_dom, rev\_host,

ping\_ip, msg\_count,

ttl\_val, rtt\_msec);

msg\_received\_count++;

}

}

}

}

clock\_gettime(CLOCK\_MONOTONIC, &tfe);

double timeElapsed = ((double)(tfe.tv\_nsec -

tfs.tv\_nsec))/1000000.0;

total\_msec = (tfe.tv\_sec-tfs.tv\_sec)\*1000.0+

timeElapsed

printf("\n===%s ping statistics===\n", ping\_ip);

printf("\n%d packets sent, %d packets received, %f percent

packet loss. Total time: %Lf ms.\n\n",

msg\_count, msg\_received\_count,

((msg\_count - msg\_received\_count)/msg\_count) \* 100.0,

total\_msec);

}

// Driver Code

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

{

int sockfd;

char \*ip\_addr, \*reverse\_hostname;

struct sockaddr\_in addr\_con;

int addrlen = sizeof(addr\_con);

char net\_buf[NI\_MAXHOST];

if(argc!=2)

{

printf("\nFormat %s <address>\n", argv[0]);

return 0;

}

ip\_addr = dns\_lookup(argv[1], &addr\_con);

if(ip\_addr==NULL)

{

printf("\nDNS lookup failed! Could

not resolve hostname!\n");

return 0;

}

reverse\_hostname = reverse\_dns\_lookup(ip\_addr);

printf("\nTrying to connect to '%s' IP: %s\n",

argv[1], ip\_addr);

printf("\nReverse Lookup domain: %s",

reverse\_hostname);

//socket()

sockfd = socket(AF\_INET, SOCK\_RAW, IPPROTO\_ICMP);

if(sockfd<0)

{

printf("\nSocket file descriptor not received!!\n");

return 0;

}

else

printf("\nSocket file descriptor %d received\n", sockfd);

signal(SIGINT, intHandler);//catching interrupt

//send pings continuously

send\_ping(sockfd, &addr\_con, reverse\_hostname,

ip\_addr, argv[1]);

return 0;

}

**Output:**

Resolving DNS..

Trying to connect to 'google.com' IP: 172.217.27.206

Reverse Lookup domain: bom07s15-in-f14.1e100.net

Socket file descriptor 3 received

Socket set to TTL..

64 bytes from bom07s15-in-f14.1e100.net (h: google.com) (172.217.27.206)

msg\_seq=1 ttl=64 rtt = 57.320584 ms.

64 bytes from bom07s15-in-f14.1e100.net (h: google.com) (172.217.27.206)

msg\_seq=2 ttl=64 rtt = 58.666775 ms.

64 bytes from bom07s15-in-f14.1e100.net (h: google.com) (172.217.27.206)

msg\_seq=3 ttl=64 rtt = 58.081148 ms.

64 bytes from bom07s15-in-f14.1e100.net (h: google.com) (172.217.27.206)

msg\_seq=4 ttl=64 rtt = 58.700630 ms.

64 bytes from bom07s15-in-f14.1e100.net (h: google.com) (172.217.27.206)

msg\_seq=5 ttl=64 rtt = 58.281802 ms.

64 bytes from bom07s15-in-f14.1e100.net (h: google.com) (172.217.27.206)

msg\_seq=6 ttl=64 rtt = 58.360916 ms.

===172.217.27.206 ping statistics===

6 packets sent, 6 packets received, 0.000000 percent packet loss.

Total time: 6295.187804 ms.