**Problem Statement:**

Write a program to implement simple communication between Client-Server using sockets utility(TCP and UDP) and demonstrate the packets captured traces using Wireshark Packet Analyzer Tool.

**Code:**

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

#include <unistd.h> // read(), write(), close() #define MAX 80

#define PORT 8080 #define SA struct sockaddr

// 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.c

#include <arpa/inet.h> // inet\_addr() #include <netdb.h>

#include <stdio.h> #include <stdlib.h> #include <string.h>

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

#include <unistd.h> // read(), write(), close() #define MAX 80

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

{

char buff[MAX]; int n;

for (;;) {

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

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

;

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

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

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

break;

}

}

}

int main()

{

int sockfd, connfd;

struct sockaddr\_in servaddr, cli;

// socket create and verification

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

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

}

else

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

bzero(&servaddr, sizeof(servaddr));

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

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

// connect the client socket to server socket

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

!= 0) {

}

else

printf("connection with the server failed...\n"); exit(0);

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

// function for chat func(sockfd);

// close the socket close(sockfd);

}

# Communication between Client-Server using UDP

**server\_udp.cpp**

// Server side implementation of UDP client-server model #include <bits/stdc++.h>

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

#define PORT 8080

#define MAXLINE 1024

// Driver code int main() {

int sockfd;

char buffer[MAXLINE];

const char \*hello = "Hello from server"; struct sockaddr\_in servaddr, cliaddr;

// Creating socket file descriptor

if ( (sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0 ) { perror("socket creation failed"); exit(EXIT\_FAILURE);

}

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

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

// Filling server information servaddr.sin\_family = AF\_INET; // IPv4 servaddr.sin\_addr.s\_addr = INADDR\_ANY; servaddr.sin\_port = htons(PORT);

// Bind the socket with the server address

if ( bind(sockfd, (const struct sockaddr \*)&servaddr, sizeof(servaddr)) < 0 )

{

perror("bind failed"); exit(EXIT\_FAILURE);

}

socklen\_t len; int n;

len = sizeof(cliaddr); //len is value/result

n = recvfrom(sockfd, (char \*)buffer, MAXLINE,

MSG\_WAITALL, ( struct sockaddr \*) &cliaddr, &len);

buffer[n] = '\0';

printf("Client : %s\n", buffer);

sendto(sockfd, (const char \*)hello, strlen(hello), MSG\_CONFIRM, (const struct sockaddr \*) &cliaddr,

len);

std::cout<<"Hello message sent."<<std::endl;

return 0;

}

# client\_udp.cpp

// Client side implementation of UDP client-server model #include <bits/stdc++.h>

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

#include <arpa/inet.h> #include <netinet/in.h>

#define PORT 8080

#define MAXLINE 1024

// Driver code int main() {

int sockfd;

char buffer[MAXLINE];

const char \*hello = "Hello from client"; struct sockaddr\_in servaddr;

// Creating socket file descriptor

if ( (sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0 ) { perror("socket creation failed"); exit(EXIT\_FAILURE);

}

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

// Filling server information servaddr.sin\_family = AF\_INET; servaddr.sin\_port = htons(PORT); servaddr.sin\_addr.s\_addr = INADDR\_ANY;

int n; socklen\_t len;

sendto(sockfd, (const char \*)hello, strlen(hello), MSG\_CONFIRM, (const struct sockaddr \*) &servaddr,

sizeof(servaddr)); std::cout<<"Hello message sent."<<std::endl;

n = recvfrom(sockfd, (char \*)buffer, MAXLINE,

MSG\_WAITALL, (struct sockaddr \*) &servaddr, &len);

buffer[n] = '\0';

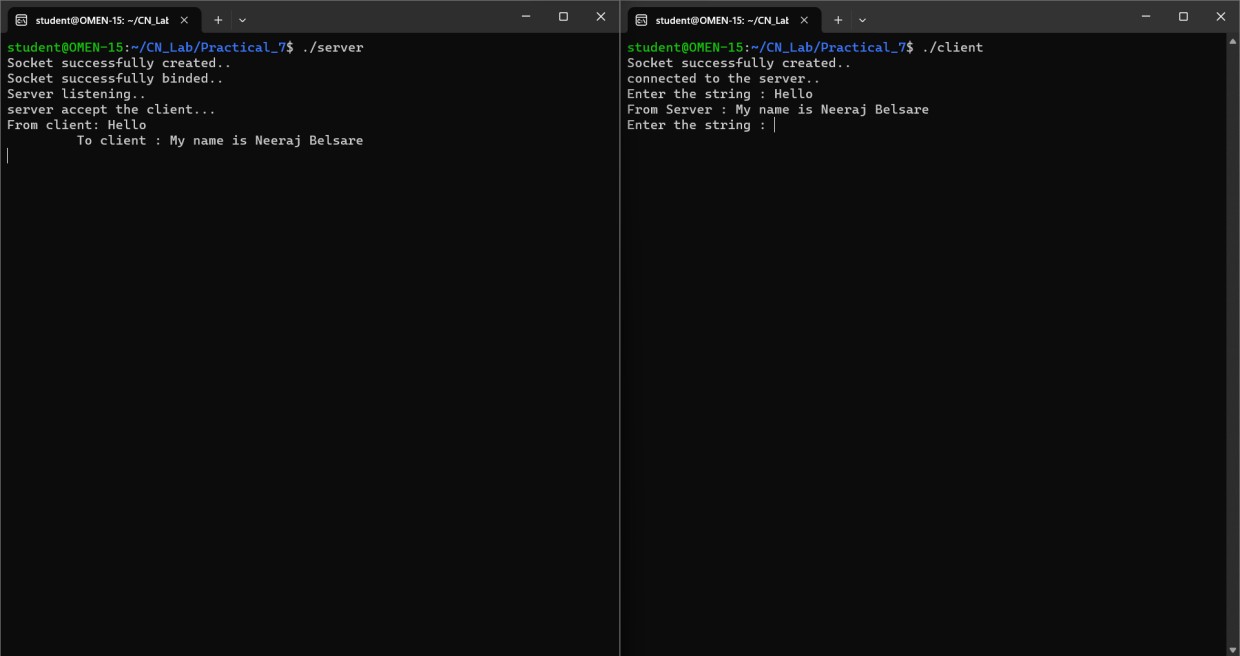
std::cout<<"Server :"<<buffer<<std::endl;

close(sockfd); return 0;

}

**OUTPUT:**

1. **Communication between Client-Server using TCP**



# Communication between Client-Server using UDP

