# EXPERIMENT: 31

**IMPLEMENTATION OF SERVER – CLIENT USING TCP SOCKET PROGRAMMING**

**Aim:**

To implement a server-client communication model using TCP socket programming in C.

**Software/Apparatus Required:**

* C Compiler (GCC or any compatible compiler)
* Linux-based OS (or any OS supporting POSIX sockets)
* Text editor (e.g., Vim, Nano, or any IDE)

**Procedure:**

**Step 1: Write the Server-Side Code**

1. Open a text editor and write the server-side C program as provided.
2. Save the file as server.c.

**Step 2: Write the Client-Side Code**

1. Open a text editor and write the client-side C program as provided.
2. Save the file as client.c.

**Step 3: Compile the Programs**

1. Open the terminal and navigate to the directory containing the server.c and client.c files.
2. Compile the server program using the following command: gcc server.c -o server
3. Compile the client program using the following command:

gcc client.c -o client

**Step 4: Run the Server**

1. Execute the server program using the following command:

./server

1. The server will start listening on port 8080.

**Step 5: Run the Client**

1. Open another terminal window and navigate to the same directory.
2. Execute the client program using the following command:

./client

1. The client will connect to the server running on 127.0.0.1 (localhost) and port 8080.

**Step 6: Test the Communication**

1. On the client side, type a message and press Enter. The message will be sent to the server.
2. The server will receive the message, display it, and prompt for a response.
3. The server's response will be sent back to the client and displayed on the client's terminal.
4. To end the communication, type "exit" on either the client or server side.

**Code:**

**//SERVER SIDE**

#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 SIDE**

// Online C compiler to run C program online

#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) { 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);

}

**Output:**

1. Server-side output:

Copy

Socket successfully created..

Socket successfully binded..

Server listening..

server accept the client...

From client: <Client Message> To client: <Server Response>

1. Client-side output:

Copy

Socket successfully created.. connected to the server..

Enter the string: <Client Message>

From Server: <Server Response>

**Result:**

Thus, the server-client communication using TCP socket programming was implemented successfully.