Server Program:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* SERVER CODE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

int main(){  
int welcomeSocket, newSocket;  
char buffer[1024];  
struct sockaddr\_in serverAddr;  
struct sockaddr\_storage serverStorage;  
socklen\_t addr\_size;

/\*—- Create the socket. The three arguments are: —-\*/  
/\* 1) Internet domain 2) Stream socket 3) Default protocol (TCP in this case) \*/  
welcomeSocket = socket(PF\_INET, SOCK\_STREAM, 0);

/\*—- Configure settings of the server address struct —-\*/  
/\* Address family = Internet \*/  
serverAddr.sin\_family = AF\_INET;  
/\* Set port number, using htons function to use proper byte order \*/  
serverAddr.sin\_port = htons(7891);  
/\* Set IP address to localhost \*/  
serverAddr.sin\_addr.s\_addr = inet\_addr(“127.0.0.1”);  
/\* Set all bits of the padding field to 0 \*/  
memset(serverAddr.sin\_zero, ‘\0’, sizeof serverAddr.sin\_zero);

/\*—- Bind the address struct to the socket —-\*/  
bind(welcomeSocket, (struct sockaddr \*) &serverAddr, sizeof(serverAddr));

/\*—- Listen on the socket, with 5 max connection requests queued —-\*/  
if(listen(welcomeSocket,5)==0)  
printf(“Listening\n”);  
else  
printf(“Error\n”);

/\*—- Accept call creates a new socket for the incoming connection —-\*/  
addr\_size = sizeof serverStorage;  
newSocket = accept(welcomeSocket, (struct sockaddr \*) &serverStorage, &addr\_size);

/\*—- Send message to the socket of the incoming connection —-\*/  
strcpy(buffer,”Hello World\n”);  
send(newSocket,buffer,13,0);

return 0;  
}

Client Program:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* CLIENT CODE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

int main(){  
int clientSocket;  
char buffer[1024];  
struct sockaddr\_in serverAddr;  
socklen\_t addr\_size;

/\*—- Create the socket. The three arguments are: —-\*/  
/\* 1) Internet domain 2) Stream socket 3) Default protocol (TCP in this case) \*/  
clientSocket = socket(PF\_INET, SOCK\_STREAM, 0);

/\*—- Configure settings of the server address struct —-\*/  
/\* Address family = Internet \*/  
serverAddr.sin\_family = AF\_INET;  
/\* Set port number, using htons function to use proper byte order \*/  
serverAddr.sin\_port = htons(7891);  
/\* Set IP address to localhost \*/  
serverAddr.sin\_addr.s\_addr = inet\_addr(“127.0.0.1”);  
/\* Set all bits of the padding field to 0 \*/  
memset(serverAddr.sin\_zero, ‘\0’, sizeof serverAddr.sin\_zero);

/\*—- Connect the socket to the server using the address struct —-\*/  
addr\_size = sizeof serverAddr;  
connect(clientSocket, (struct sockaddr \*) &serverAddr, addr\_size);

/\*—- Read the message from the server into the buffer —-\*/  
recv(clientSocket, buffer, 1024, 0);

/\*—- Print the received message —-\*/  
printf(“Data received: %s”,buffer);

return 0;  
}

//// For file transfer

The project is divided into two files:

1. client.c
2. server.c

The client.c file contains the code for client-side, which read the text file and sends it to the server and the server.c file receives the data from the client and saves it in a text file.

**Client**

The client performs the following functions.

1. Start the program
2. Declare the variables and structures required.
3. A socket is created and the connect function is executed.
4. The file is opened.
5. The data from the file is read and sent to the server.
6. The socket is closed.
7. The program is stopped.

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <arpa/inet.h>

#define SIZE 1024

void send\_file(FILE \*fp, int sockfd){

int n;

char data[SIZE] = {0};

while(fgets(data, SIZE, fp) != NULL) {

if (send(sockfd, data, sizeof(data), 0) == -1) {

perror("[-]Error in sending file.");

exit(1);

}

bzero(data, SIZE);

}

}

int main(){

char \*ip = "127.0.0.1";

int port = 8080;

int e;

int sockfd;

struct sockaddr\_in server\_addr;

FILE \*fp;

char \*filename = "send.txt";

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

if(sockfd < 0) {

perror("[-]Error in socket");

exit(1);

}

printf("[+]Server socket created successfully.\n");

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

server\_addr.sin\_port = port;

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

e = connect(sockfd, (struct sockaddr\*)&server\_addr, sizeof(server\_addr));

if(e == -1) {

perror("[-]Error in socket");

exit(1);

}

printf("[+]Connected to Server.\n");

fp = fopen(filename, "r");

if (fp == NULL) {

perror("[-]Error in reading file.");

exit(1);

}

send\_file(fp, sockfd);

printf("[+]File data sent successfully.\n");

printf("[+]Closing the connection.\n");

close(sockfd);

return 0;

}

**Server**

The client performs the following functions.

1. Start the program.
2. Declare the variables and structures required.
3. The socket is created using the socket function.
4. The socket is binded to the specific port.
5. Start listening for the connections.
6. Accept the connection from the client.
7. Create a new file.
8. Receives the data from the client.
9. Write the data into the file.
10. The program is stopped.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <arpa/inet.h>

#define SIZE 1024

void write\_file(int sockfd){

int n;

FILE \*fp;

char \*filename = "recv.txt";

char buffer[SIZE];

fp = fopen(filename, "w");

while (1) {

n = recv(sockfd, buffer, SIZE, 0);

if (n <= 0){

break;

return;

}

fprintf(fp, "%s", buffer);

bzero(buffer, SIZE);

}

return;

}

int main(){

char \*ip = "127.0.0.1";

int port = 8080;

int e;

int sockfd, new\_sock;

struct sockaddr\_in server\_addr, new\_addr;

socklen\_t addr\_size;

char buffer[SIZE];

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

if(sockfd < 0) {

perror("[-]Error in socket");

exit(1);

}

printf("[+]Server socket created successfully.\n");

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

server\_addr.sin\_port = port;

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

e = bind(sockfd, (struct sockaddr\*)&server\_addr, sizeof(server\_addr));

if(e < 0) {

perror("[-]Error in bind");

exit(1);

}

printf("[+]Binding successfull.\n");

if(listen(sockfd, 10) == 0){

printf("[+]Listening....\n");

}else{

perror("[-]Error in listening");

exit(1);

}

addr\_size = sizeof(new\_addr);

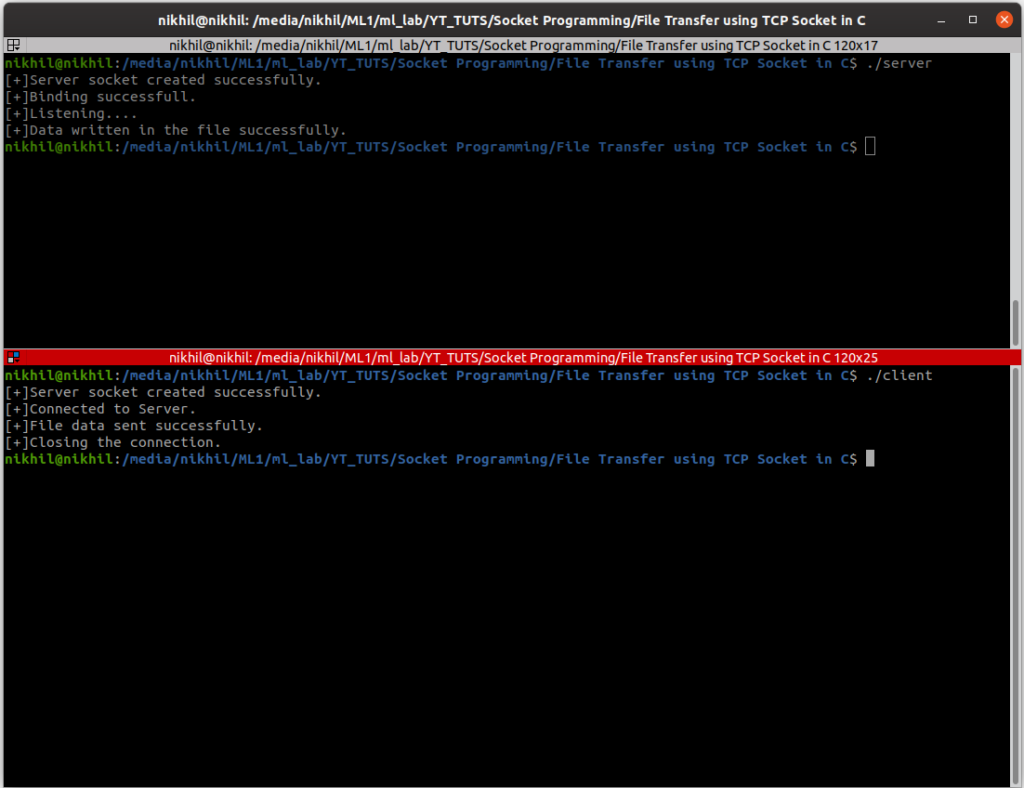
new\_sock = accept(sockfd, (struct sockaddr\*)&new\_addr, &addr\_size);

write\_file(new\_sock);

printf("[+]Data written in the file successfully.\n");

return 0;

}



/////// for calculator

**Client Side Implementation**

|  |
| --- |
| // Java program to illustrate Client Side Programming  // for Simple Calculator using TCP  import java.io.DataInputStream;  import java.io.DataOutputStream;  import java.io.IOException;  import java.net.InetAddress;  import java.net.Socket;  import java.net.UnknownHostException;  import java.util.Scanner;    public class Calc\_Client  {      public static void main(String[] args) throws IOException      {          InetAddress ip = InetAddress.getLocalHost();          int port = 4444;          Scanner sc = new Scanner(System.in);            // Step 1: Open the socket connection.          Socket s = new Socket(ip, port);            // Step 2: Communication-get the input and output stream          DataInputStream dis = new DataInputStream(s.getInputStream());          DataOutputStream dos = new DataOutputStream(s.getOutputStream());            while (true)          {              // Enter the equation in the form-              // "operand1 operation operand2"              System.out.print("Enter the equation in the form: ");              System.out.println("'operand operator operand'");                String inp = sc.nextLine();                if (inp.equals("bye"))                  break;                // send the equation to server              dos.writeUTF(inp);                // wait till request is processed and sent back to client              String ans = dis.readUTF();              System.out.println("Answer=" + ans);          }      }  } |

**Output**

Enter the equation in the form: 'operand operator operand'

5 \* 6

Answer=30

Enter the equation in the form: 'operand operator operand'

5 + 6

Answer=11

Enter the equation in the form: 'operand operator operand'

9 / 3

Answer=3

**Server-Side Programming**

Steps involved on the server side are as follows-

1. Establish a socket connection.
2. **Process the equations coming from client:** In server side also we open both the inputStream and outputStream. After receiving the equation, we process it and returns the result back to client by writing on the outputStream of the socket.
3. Close the connection.

**Server Side Implementation**

|  |
| --- |
| // Java program to illustrate Server Side Programming  // for Simple Calculator using TCP  import java.io.DataInputStream;  import java.io.DataOutputStream;  import java.io.IOException;  import java.net.ServerSocket;  import java.net.Socket;  import java.util.StringTokenizer;    public class Calc\_Server  {      public static void main(String args[]) throws IOException      {            // Step 1: Establish the socket connection.          ServerSocket ss = new ServerSocket(4444);          Socket s = ss.accept();            // Step 2: Processing the request.          DataInputStream dis = new DataInputStream(s.getInputStream());          DataOutputStream dos = new DataOutputStream(s.getOutputStream());            while (true)          {              // wait for input              String input = dis.readUTF();                if(input.equals("bye"))                  break;                System.out.println("Equation received:-" + input);              int result;                // Use StringTokenizer to break the equation into operand and              // operation              StringTokenizer st = new StringTokenizer(input);                int oprnd1 = Integer.parseInt(st.nextToken());              String operation = st.nextToken();              int oprnd2 = Integer.parseInt(st.nextToken());                // perform the required operation.              if (operation.equals("+"))              {                  result = oprnd1 + oprnd2;              }                else if (operation.equals("-"))              {                  result = oprnd1 - oprnd2;              }              else if (operation.equals("\*"))              {                  result = oprnd1 \* oprnd2;              }              else              {                  result = oprnd1 / oprnd2;              }              System.out.println("Sending the result...");                // send the result back to the client.              dos.writeUTF(Integer.toString(result));          }      }  } |

**Output:**

Equation received:-5 \* 6

Sending the result...

Equation received:-5 + 6

Sending the result...

Equation received:-9 / 3

Sending the result...