**Server-Client Chat**

**Using**

**Transmission Control Protocol (TCP)**



A

Linux Programming Project Report

**Bachelor of Technology**

**In**

**Computer Science & Engineering**

**By**

Loke.Abhijith 2103A53007

**Under the Guidance of**

**Mr. T. SAMPATH KUMAR**

Professor, School of Computer Science and Engineering

**Submitted to**





**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**CERTIFICATE**

This is to certify that the Linux Programming – Course Project Report entitled **“Server-Client Chat using Transmission Control Protocol (TCP)”** is a record of bonafide work carried out by the student **P. NITHIN REDDY** bearing Roll No. 2103A53009 during the academic year 2022-23 in partial fulfilment of the award of the degree of **Bachelor of technology in Computer Science & Engineering** by the SR University.

**Lab In-charge Head of the Department**

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
| **TOPICS** | **PAGE NO** |
| Abstract | 4 |
| Objective of project | 5 |
| Definition of elements used in project | 6-8 |
| Design | 9 |
| Source Code | 10-12 |
| Output | 12 |

**ABSTRACT**

Transmission Control Protocol is referred to as TCP. The packet transmission from source to destination is made easier by this transport layer protocol. Since it is a connection-oriented protocol, communication between computer devices in a network begins with the link being established. Together, they are referred to as a TCP/IP since they are utilised with an IP protocol. The TCP's primary role is to retrieve data from the application layer. The data is then divided into numerous packets, each of which is given a number, before being transmitted to the destination. The packets are reassembled by the TCP and sent to the application layer from the opposite side. Since TCP is a connection-oriented protocol, the connection will continue to exist as long as the sender and receiver are still in contact.

**OBJECTIVE**

* The server returns an ASCII character string of the current date and time in an unspecified format.
* On UNIX-like operating systems a daytime server is usually built into the inetd (or xinetd) daemon. The service is usually not enabled by default.
* The client requests server for the day and time and server gives the information about day and time.

**DEFINITIONS OF THE ELEMENTS USED IN THE PROJECT**

* Here I used basics of TCP protocol.
* Among the two users, I assumed one is server and the other is client.
* For Server the following functions are included. They are:
  + Socketfd (Socket Creation)
  + Bind
  + Listen
  + Accept
  + Read
  + Write
  + Close
* For Client the following functions are included. They are:
  + Socketfd (Socket Creation)
  + Connect
  + Write
  + Read
  + Close

**STAGES OF SERVER**

1. **Socket Creation:**

*int sockfd = socket (domain, type, protocol);*

* 1. **Sockfd:** Socket descriptor, an integer. If its value is non negative integer, then that means socket created. Else If the value is -1, then there is an error in creating socket.
  2. **Domain:** Communication domain is indicated by an integer. For interprocess communication on the same host, we use the AF\_ LOCAL provided in the POSIX standard. We utilise AF INET for processes connected by IPV4 and AF I NET 6 for processes linked by IPV6 to communicate with one another.
  3. **Type:** This specifies communication type. For TCP (reliable, connection oriented) communication we specify it as “SOCK\_STREAM” and for UDP (unreliable, connectionless) communication we specify it as “SOCK\_DGRAM”.
  4. **Protocol:** Internet Protocol (IP) has a protocol value of 0. This is the same number that is displayed in the protocol field of a packet's IP header.

1. **Bind:**

*int bind (int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);*

* The bind function ties the newly created socket to the address and port number supplied in addr. In the sample code, the server is bound to localhost, therefore the IP address is specified using INADDR\_ANY.

1. **Listen:**

*int listen (int sockfd, int backlog);*

* It puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection. The backlog, defines the maximum length to which the queue of pending connections for sockfd may grow. If a connection request arrives when the queue is full, the client may receive an error with an indication of ECONNREFUSED.

1. **Accept:**

*int new\_socket = accept (int sockfd, struct sockaddr \*addr, socklen\_t \*addrlen);*

* It extracts the first connection request on the queue of pending connections for the listening socket, sockfd, creates a new connected socket, and returns a new file descriptor referring to that socket. At this point, the connection is established between client and server, and they are ready to transfer data.

1. **Read and Write Operations:**

* Their will be alternate read and write operations done by the server.
* For a corresponding write operation from client, there will be a corresponding read operation from the server.
* For a corresponding read operation from client, there will be a corresponding write operation from the server.

1. **Close:**

* This will close or disconnect the server and the client.

**STAGES OF CLIENT**

1. **Socket Connection:**

*int sockfd = socket (domain, type, protocol);*

* 1. **Sockfd:** Socket descriptor, an integer. If its value is non negative integer, then that means socket created. Else If the value is -1, then there is an error in creating socket.
  2. **Domain:** Communication domain is indicated by an integer. For interprocess communication on the same host, we use the AF\_ LOCAL provided in the POSIX standard. We utilise AF INET for processes connected by IPV4 and AF I NET 6 for processes linked by IPV6 to communicate with one another.
  3. **Type:** This specifies communication type. For TCP (reliable, connection oriented) communication we specify it as “SOCK\_STREAM” and for UDP (unreliable, connectionless) communication we specify it as “SOCK\_DGRAM”.
* **Protocol:** Internet Protocol (IP) has a protocol value of 0. This is the same number that is displayed in the protocol field of a packet's IP header.

1. **Connect:**

*Int connect (int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);*

* The connect() system call connects the socket referred to by the file descriptor sockfd to the address specified by addr. Server’s address and port is specified in addr.

1. **Read and Write Operations:**

* There will be alternate read and write operations done by the client.
* For a corresponding read operation from server, there will be a corresponding write operation from the client.
* For a corresponding write operation from server, there will be a corresponding read operation from the client.

1. **Close:**

* This will close or disconnect the server and the client.

**DESIGN**

**Server Process Client Process**

Close()

Close()

Write()

Read()

Accept()

Listen()

Bind()

Socket()

Socket()

Connect()

Write()

Read()

**Source Code**

**Server.c**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| #include<stdio.h>  #include<stdlib.h>  #include<sys/types.h>  #include<sys/socket.h>  #include<netinet/in.h>  #include<unistd.h>  #include<time.h>  //port value and the max size for buffer  #define PORT 9002 //the port users will be connecting to  #define BACKLOG 10 //how many pending connections queue will hold  int main(){  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Time Setup\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  time\_t currentTime ;  time(&currentTime);  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  int countClient = 0;  int socket\_descriptor = socket(AF\_INET, SOCK\_STREAM, 0);  struct sockaddr\_in serverAddress;  serverAddress.sin\_family = AF\_INET;  serverAddress.sin\_addr.s\_addr=INADDR\_ANY;   |  | | --- | |  | | serverAddress.sin\_port=htons(PORT);  //binding address  bind(socket\_descriptor,(struct sockaddr\*)&serverAddress,sizeof(serverAddress)); | |  | | //listening to the queue  listen(socket\_descriptor,BACKLOG);  printf("\nServer Started ...");  while(1){  countClient++;  printf("\n");  int client\_socket = accept(socket\_descriptor, NULL, NULL);  // char \*string = asctime(timeinfo);  printf("\nClient %d has requested for time at %s", countClient, ctime(&currentTime));  //sending time to the client side  // ctime(&time\_from\_pc)  send(client\_socket,ctime(&currentTime), 30, 0);  }  return 0;  }  **Client.c**  #include <stdio.h>  #include <stdlib.h>  #include <sys/types.h>  #include <sys/socket.h>  #include <netinet/in.h>  #include "string.h"  #define PORT 9002 //the port users will be connecting to  #define MAXLINE 30 //for buffer size  int main(){  int socket\_descriptor = socket(AF\_INET, SOCK\_STREAM, 0);  char serverResponse[MAXLINE];  struct sockaddr\_in serverAddress;  serverAddress.sin\_family = AF\_INET;  serverAddress.sin\_addr.s\_addr = INADDR\_ANY;  serverAddress.sin\_port = htons(PORT);  connect(socket\_descriptor, (struct sockaddr \*)&serverAddress, sizeof(serverAddress));  recv(socket\_descriptor,serverResponse,MAXLINE-1,0);  printf("\nTIME FROM SERVER %s\n",serverResponse);  return 0;  }  **OUTPUT**  **Server Output:**    **Client Output:** | |
|  |
|  |  |