Text

Description automatically generated

|  |  |
| --- | --- |
| **Student Name** | **Arun Adhikari** |
| **SRN No** | 202100406 |
| **Roll No** | 38 |
| **Program** | Computer Engg |
| **Year** | Third Year |
| **Division** | G |
| **Subject** | Computer Network Laboratory (BTECCE21506) |
| **Assignment No** | One |

**Title :-** Socket Programming for TCP Client and TCP Server.

**Problem Statement :** socket programming with a TCP client and server in Python, the task is to create two separate programs: one for the TCP server and another for the TCP client. These programs will demonstrate the fundamentals of establishing a connection over the network, sending data, and receiving responses using the Transmission Control Protocol (TCP).

**Introduction to TCP :-**

* Transmission Control Protocol (TCP) is a connection-oriented protocol that ensures reliable data transmission between client and server.
* Unlike UDP, TCP guarantees packet delivery, maintains the order of packets, and performs error checking.
* TCP is suitable for applications where reliability is critical, such as web browsing, file transfers, and email.

**2. Socket Programming Basics:**

* A Socket Is An Endpoint For Sending Or Receiving Data Across A Computer Network.
* In UDP Socket Programming, The Client And Server Communicate By Sending And Receiving Datagrams Through Their Respective Sockets.
* Key Functions For UDP Socket Programming In C/C++:
  + Socket(): Creates A New Socket.
  + Bind(): Associates A Socket With A Local Address (Used In The Server).
  + Sendto(): Sends A Datagram To A Specified Address.
  + Recvfrom(): Receives A Datagram From A Socket.
  + Close(): Closes The Socket.

**3** **TCP Server**

* The server's role is to listen for incoming client requests, process them, and send back responses.
* Steps for implementing a TCP server:
  1. Create a socket using socket() function with parameters specifying the use of TCP.
  2. Bind the socket to a specific port number on the server machine using bind().
  3. Listen for incoming connections using listen().
  4. Accept a client connection using accept().
  5. Receive the client's request using recv().
  6. Process the request (perform the specified arithmetic operation).
  7. Send the result back to the client using send().
  8. Close the client connection.

**4. TCP Client**

* The client sends a request to the server and waits for a response.
* Steps for implementing a TCP client:
  1. Create a socket using socket().
  2. Prepare the server address (IP address and port number) to which requests will be sent.
  3. Connect to the server using connect().
  4. Send a request to the server using send(), including the arithmetic operation (e.g., "5 + 3").
  5. Receive the response from the server using recv(), which will contain the result of the calculation.
  6. Display the result to the user.
  7. Close the socket when the operation is complete.

**Benefits of Using TCP for a Calculator Application (Short Version)**

1. Reliable Transmission: Guarantees accurate and ordered delivery of messages.
2. Connection-Oriented: Establishes a dedicated connection, reducing data loss.
3. Error Checking: Automatically retransmits lost or corrupted packets.
4. Flow Control: Manages data transmission rates to prevent overwhelming the server.
5. Congestion Control: Adjusts data flow to maintain performance during high traffic.

**7. Challenges with TCP**

1. Higher Latency:
   * TCP's connection-oriented nature introduces latency due to the initial handshake process (SYN, SYN-ACK, ACK) required to establish a connection before any data can be sent. This can slow down response times for applications requiring quick interactions.
2. Overhead:
   * TCP has significant overhead due to its reliability features, such as error checking, acknowledgments, and flow control. This can lead to increased resource usage (CPU and memory) and may not be suitable for lightweight applications where performance is critical.

**Conclusion:-** The implementation of a TCP client-server calculator application highlights the essential principles of socket programming and reliable network communication using the Transmission Control Protocol (TCP). This project effectively demonstrates how to establish a connection between a client and a server, enabling seamless interaction for performing basic arithmetic operations.

In this application, the client provides a user-friendly interface, allowing users to select operations such as addition, subtraction, multiplication, and division. Upon receiving user inputs, the client sends a formatted request to the server. The server, upon receiving this request, processes the operation and returns the computed result to the client. The TCP protocol ensures reliable delivery of messages, maintaining the order of packets and providing built-in error-checking mechanisms. This reliability is crucial for applications where data integrity is paramount, such as financial transactions and remote data processing.

**Output:-**

