Text

Description automatically generated

|  |  |
| --- | --- |
| **Student Name** | **Arun Adhiakri** |
| **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 UDP Client And UDP Server**

**Problem Statement :** Implement A UDP-Based Socket Programming Assignment That Involves Creating A Client-Server Application Where The UDP Client Sends A Message To The UDP Server, And The Server Responds Back With An Acknowledgment.

**Introduction To UDP:**

* UDP Is A Connectionless And Lightweight Protocol Used For Sending Datagrams (Packets) Without Establishing A Connection.
* Unlike TCP, It Does Not Guarantee Packet Delivery, Ordering, Or Error Checking, Making It Faster But Less Reliable.
* UDP Is Suitable For Applications Where Speed Is Critical, And Occasional Packet Loss Is Acceptable, Such As Online Gaming Or Video Streaming.

**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. UDP Server:**

* The Server's Role Is To Wait For Incoming Client Requests, Process Them, And Send Back Responses.
* Steps For Implementing A UDP Server:
  1. Create A Socket Using Socket() Function With Parameters Specifying The Use Of The UDP Protocol.
  2. Bind The Socket To A Specific Port Number On The Server Machine Using Bind().
  3. Listen For Incoming Requests Using Recvfrom(), Which Waits For A Datagram From The Client.
  4. Process The Client's Request, Which In The Case Of A Calculator, Involves Performing The Specified Arithmetic Operation.
  5. Send The Result Back To The Client Using Sendto().
  6. Repeat Steps 3 To 5 For New Incoming Requests Or Close The Socket When Done.

**4. UDP Client:**

* The Client Sends A Request To The Server And Waits For A Response.
* Steps For Implementing A UDP Client:
  1. Create A Socket Using Socket().
  2. Prepare The Server Address (IP Address And Port Number) To Which Requests Will Be Sent.
  3. Send A Request To The Server Using Sendto(), Including The Arithmetic Operation (E.G., "5 + 3").
  4. Receive The Response From The Server Using Recvfrom(), Which Will Contain The Result Of The Calculation.
  5. Display The Result To The User.
  6. Close The Socket When The Operation Is Complete.

**5. Communication Flow For Calculator Application:**

* Client Request Format: The Client Sends A String Representing An Arithmetic Operation (E.G., "5 + 3" Or "10 \* 2").
* Server Processing: The Server Parses The Received String To Identify The Operation And Operands, Performs The Calculation, And Formats The Result As A String.
* Server Response Format: The Result Of The Calculation (E.G., "8" For "5 + 3") Is Sent Back To The Client.
* Handling Errors: The Server Should Handle Invalid Inputs Gracefully And Return Appropriate Error Messages.

**Benefits Of Using UDP For A Calculator Application:**

* Lower Latency: Since There Is No Connection Establishment, Communication Is Faster.
* Simple Implementation: UDP Requires Fewer Resources, Making It Suitable For Lightweight Applications.
* Broadcast Support: UDP Allows Sending Data To Multiple Clients Simultaneously If Needed.

**7. Challenges With UDP:**

* Lack Of Reliability: UDP Does Not Guarantee Message Delivery Or Order, Which May Require Implementing Additional Checks If Reliability Is Crucial.
* No Built-In Acknowledgment: Unlike TCP, There Is No Built-In Mechanism To Confirm Receipt Of Packets.

**8. Client Implementation Logic:**

1. Create A UDP Socket To Communicate With The Server.
2. Define The Server Address (Localhost, Port 6789).
3. Display A Menu Of Arithmetic Operations (Addition, Subtraction, Multiplication, Division, Exit).
4. Loop Until Exit:
   * Prompt The User To Select An Operation (1-5).
   * If '5', Send "BYE" To The Server And Exit.
   * Validate The Operation Input; If Invalid, Prompt Again.
   * Get Two Numbers From The User.
   * Format The Message As "{Operation} {Num1} {Num2}" And Send It To The Server.
   * Wait For The Server's Result And Display It.
5. Close The Socket When Done.

**Server Implementation Logic:**

1. Create And Bind A UDP Socket To Localhost On Port 6789.
2. Loop To Receive Requests:
   * Use Recvfrom() To Get Data From The Client.
   * Split The Received Message Into Operation And Numbers.
   * Validate The Format; If Invalid, Send An Error Response.
   * Perform The Requested Arithmetic Operation Using The Perform\_Operation() Function:
     + Operation '1': Addition
     + Operation '2': Subtraction
     + Operation '3': Multiplication
     + Operation '4': Division (Check For Zero)
   * Send The Result Back To The Client.
3. Continue Processing Requests Until A "BYE" Message Is Received (Optional For This Implementation).

This Structure Allows The Client To Request Calculations And The Server To Respond With Results Using UDP Communication.

**Conclusion:-** This UDP Client-Server Calculator Allows Users To Perform Basic Arithmetic Operations Over A Network Using UDP Communication. The Client Interacts With The User, Collects Input, And Communicates With The Server To Perform Calculations, While The Server Processes Requests And Returns Results. The Design Demonstrates The Use Of Sockets For Inter-Process Communication, Highlighting The Simplicity And Efficiency Of The UDP Protocol For Real-Time Applications.

**Output:-**

