BTN415- Data Communications Programming

**Lab #6**

**Title: Data Transferring Using UDP and TCP in C++**

**Objective:**

The goal of this lab is to implement a client-server application that transfers data between a client and a server using both UDP and TCP sockets in C++. You will write two versions of the program: one using UDP (connectionless) and another using TCP (connection-oriented).

**Tasks:**

1. Write the client and server programs in C++ to transfer a file using both UDP and TCP as described in the program description section below.
2. Demonstrate the program in the lab and submit it on Blackboard.
3. Write all the steps clearly in a separate file (doc or pdf) explaining how to run your application.

**Program Description:**

In this lab, you will implement a **file transfer application** based on a **client-server model** using both **UDP and TCP** sockets. The server will act as a connectionless server for UDP and a connection-oriented server for TCP. The client will send a request to download a file from the server, and the server will transfer the file in chunks, depending on the protocol used (UDP or TCP).

**Client-Server Interaction:**

The communication between the client and server will follow a specific Protocol Data Unit (PDU) format:

* PDU Format:
* Type Data (optional)

The PDU types include:

* + D (DATA PDU) – The server sends the file data to the client.
  + L (LAST PDU) – The server signals the end of the file transfer.
  + E (ERROR PDU) – The server reports an error (e.g., file not found).
  + F (FILENAME PDU) – The client requests a file from the server.

**PDU Details:**

* **FILENAME PDU**:
  + The client sends the filename it wants to download from the server.
  + Format: F filename
* **DATA PDU**:
  + The server sends chunks of the requested file in this format.
  + Format: D <file\_chunk\_data>
* **LAST PDU**:
  + The server sends this PDU to indicate the end of file transfer.
  + Format: L <last\_file\_chunk\_data>
* **ERROR PDU**:
  + The server sends this PDU if it encounters an error (e.g., file not found).
  + Format: E <error\_message>

**Server Process:**

1. **UDP Server**:
   * The server is a connectionless, iterative server that listens for client requests.
   * Upon receiving a **FILENAME PDU** from the client, the server checks if the requested file exists.
   * If the file exists, the server breaks the file into chunks and sends each chunk in a **DATA PDU**.
   * The server sends a **LAST PDU** when the file transfer is complete.
   * If an error occurs (e.g., file not found), the server sends an **ERROR PDU**.
2. **TCP Server**:
   * The server is a connection-oriented server that listens for client connections.
   * The process is similar to the UDP server, but using TCP sockets for communication, which guarantees reliable data transfer.
   * Upon receiving a **FILENAME PDU** from the client, the server checks if the requested file exists.
   * The server sends file data in **DATA PDUs**, ending with a **LAST PDU** to indicate the end of the transfer.

**Client Process:**

1. **Client Steps**:
   * The client starts by sending a **FILENAME PDU** to the server with the requested file name.
   * The client listens for incoming **DATA PDUs** from the server, which contain file chunks.
   * Upon receiving a **LAST PDU**, the client knows the file transfer is complete.
   * If an error occurs, the client receives an **ERROR PDU** and handles it appropriately.
2. **File Saving**:
   * The client writes the received file data into a local file. The file is built from the chunks sent by the server.

**UDP and TCP Implementation in C++:**

You will need to implement two versions of this program:

1. **UDP Client and Server**:
   * The UDP server is a simple connectionless server that handles multiple requests sequentially.
   * The UDP client sends the **FILENAME PDU** and receives file data from the server in **DATA PDUs**.
2. **TCP Client and Server**:
   * The TCP server listens for incoming connections and handles file requests with a connection-oriented approach.
   * The TCP client connects to the server, sends the **FILENAME PDU**, and receives **DATA PDUs** over the TCP connection.

**Protocol Data Unit (PDU) Summary:**

| **Type** | **Description** | **Data** |
| --- | --- | --- |
| F | FILENAME PDU | F <filename> |
| D | DATA PDU | D <data\_chunk> |
| L | LAST PDU | L <last\_data\_chunk> |
| E | ERROR PDU | E <error\_message> |

**Additional Notes:**

* **File Chunks**: File data is sent in chunks of size <= 100 bytes. If the file is larger than this, it will be split into multiple chunks, with each chunk being sent as a **DATA PDU**.
* **Error Handling**: If a file is not found on the server, an **ERROR PDU** will be sent to the client indicating the failure.
* **Persistence**: The file content is reconstructed on the client side by appending received chunks.

**Deliverables:**

1. **C++ Source Code**:
   * Submit the C++ source code for both the UDP and TCP server and client programs.
2. **Protocol Documentation**:
   * Submit a detailed explanation of the protocol, including the PDU types, their structure, and the order of operations.
3. **User Manual**:
   * Provide a step-by-step guide explaining how to run the client-server programs for both UDP and TCP versions.

**Grading Criteria:**

* Correct implementation of UDP and TCP server and client programs.
* Proper handling of file transfer in both UDP and TCP versions.
* Clear and accurate protocol documentation.
* Error handling and correct file saving by the client.

**Example of Running the Application:**

1. **Start the Server**:
   * Run either the UDP or TCP server. The server listens on a specific port (e.g., 8080).
2. **Start the Client**:
   * The client sends a **FILENAME PDU** to request a file from the server.
   * The client receives the file data in **DATA PDUs**, and writes it to a local file.
3. **Complete Transfer**:
   * The server sends a **LAST PDU** to indicate the end of the file transfer, and the client finishes writing the file.