TCP & UDP Socket Programming Report

TCP

TCP Server

The main steps are as follows:

- Create a socket object and bind it to a specific address and port.
- Use the `listen()` method to start listening for connection requests.
- Enter an infinite loop to accept client connections.
- Create a new thread for each connection to handle client communication.
- Receive client messages within the thread, process them, and respond accordingly.
- Handle various exceptions to ensure proper connection closure.

Features:

- Implements multithreading using the `threading` module.
- Uses daemon threads to ensure all threads terminate when the main program exits.
- Uses `setsockopt()` to enable address reuse.

TCP Client

The TCP client handles connection, sending, and receiving:

- Create a socket object.
- Connect to the specified server address and port.
- Receive user input and send it to the server.
- Receive and display the server response.
- Close the connection.

Features:

- Uses Python's context manager (`with` statement) to ensure proper resource management.
- Includes comprehensive error handling to provide user-friendly feedback.

UDP

UDP Server

A UDP server is simpler than a TCP server because it does not maintain a connection state:

- Create a UDP socket and bind it to a specific address and port.
- Receive incoming UDP packets.
- Process the received data and send a response.
- Handle exceptions.

Features:

- Uses the `recvfrom()` method to receive both data and the sender's address.
- Uses the `sendto()` method to send responses to specific addresses.

UDP Client

The UDP client handles sending and receiving:

- Create a UDP socket.
- Receive user input.
- Use `sendto()` to send data packets to the server.
- Use `recvfrom()` to receive responses from the server.
- Implement timeout handling.

Features:

- Uses `settimeout()` to define a receive timeout.
- Handles possible timeout exceptions using a `try-except` structure.

Key Differences Between TCP and UDP

Connection Management:

- **TCP** requires establishing and maintaining a connection and closing it after communication.
- **UDP** is connectionless, meaning each data packet is independent.

Data Transmission Methods:

- **TCP** uses `send()` / `recv()` methods.
- **UDP** uses `sendto()` / `recvfrom()` methods, requiring the destination address to be specified.

Handling Multiple Clients:

- **TCP** requires multithreading to handle multiple clients.
- **UDP** can handle multiple client packets without additional threading.

Reliability Handling:

- **TCP** ensures reliable data transmission without additional code.
- **UDP** does not guarantee message delivery, ordering, or data integrity.