IT 307- Exploring the Networks
Lab 10- TCP and UDP Client-Server Communication

Basics of TCP Socket Communication

TCP (Transmission Control Protocol) is a connection-oriented protocol, which means that a connection is established and maintained until the application programs at each end have finished exchanging messages. It guarantees that all data packets will be delivered in the order in which they were sent and without duplication.

Steps:

1. **Server Listening:** The server process starts by setting up a socket and binding it to an IP address and port. It then listens for connections.

2. **Connection Establishment:** The client initiates the connection using the server's IP address and port. The server accepts the connection to start communication.

3. **Data Transfer:** Data is sent over the socket in a stream-like fashion, and TCP ensures the integrity of data.

4. **Termination:** Either the client or server can initiate termination of the connection, and a proper handshake is performed to close the connection gracefully.

Basics of UDP Socket Communication

UDP (User Datagram Protocol) is a connectionless protocol. It does not establish a connection before sending data, does not guarantee order or delivery, and does not check for errors as rigorously as TCP.

Steps:

1. **Socket Creation:** Both the client and server create socket objects.

2. **Data Transfer:** The client sends datagrams (messages) to the server, which may or may not arrive, and may arrive out of order. There is no acknowledgment of receipt.

3. **No Connection State:** UDP is stateless, meaning that each datagram is an independent packet and does not relate to any other datagram.

TCP Chat Application Code (Python Implementation)

Now, let's integrate the basics into the step-by-step code explanation for a simple TCP-based chat application.

Server Side:

1. **Setting Up the Server Socket:**

   Import the socket library.
   Create a TCP/IP socket with socket.socket(socket.AF_INET, socket.SOCK_STREAM).
   Bind the socket to an IP address and port.
   Listen for incoming connections with socket.listen().

2. **Handling a Client Connection:**

   Accept a connection with socket.accept(), which blocks and waits for an incoming connection.
   Receive data with socket.recv().
   Send data back with socket.send() or socket.sendall().

3. **Closing the Connection:**

   Close the client socket when done with socket.close().

Client Side:

1. **Connecting to the Server:**

   Create a TCP/IP socket.
   Connect to the server using its IP address and port with socket.connect().

2. **Communicating with the Server:**

   Send data to the server with socket.send() or socket.sendall().
   Receive server responses with socket.recv().

3. **Shutting Down:**

   Close the socket to end communication with socket.close().

TCP Server Side implementation

```
import socket

def start_client(server_host='localhost', server_port=12345):
    # Create a TCP/IP socket
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # Connect the socket to the server's address and port
    client_socket.connect((server_host, server_port))

    try:
        while True:
```

```
            # Send data
            message = input("Enter your message ('exit' to close): ")
            client_socket.sendall(message.encode('utf-8'))

            if message == 'exit':
                print("Closing connection.")
                break

            # Receive data back from the server
            reply = client_socket.recv(1024).decode('utf-8')
            print(f"Server: {reply}")

    finally:
        # Clean up the connection
        client_socket.close()
        print("Connection closed.")

if __name__ == '__main__':
    start_client()
```

. TCP Client Side implementation

```
import socket

def start_client(server_host='localhost', server_port=12345):
    # Create a TCP/IP socket
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # Connect the socket to the server's address and port
    client_socket.connect((server_host, server_port))

    try:
        while True:
            # Send data
            message = input("Enter your message ('exit' to close): ")
            client_socket.sendall(message.encode('utf-8'))

            if message == 'exit':
                print("Closing connection.")
                break

            # Receive data back from the server
            reply = client_socket.recv(1024).decode('utf-8')
            print(f"Server: {reply}")

    finally:
        # Clean up the connection
        client_socket.close()
        print("Connection closed.")
```

```
if __name__ == '__main__':
    start_client()
```

UDP Chat Application Code (Python Implementation)

Server Side:

1. **Setting up the Server Socket:**

   ```
   import socket
   server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
   ```

2. **Bind Socket:**

   Bind the server socket to a public host and port.

   ```
   server_socket.bind(('localhost', 12345))
   ```

3. **Receive data:**

   Use recvfrom() to receive UDP datagrams (blocks until data is received). This method returns the data as well as the address of the client sending the data

   ```
   data, client_address = server_socket.recvfrom(1024)
   ```

4. **Process and Respond:**

   Process the data and respond using sendto(), sending the data back to the client_address.

   ```
   server_socket.sendto(data, client_address)
   ```

5. **Repeat or close:**

   Typically, the server will continue to listen for data in a loop. If it needs to shut down, close the socket with close().

   ```
   server_socket.close()
   ```

Client Side:

1. **Create UDP Socket:**

   Similarly, create a socket object for the client.

   ```
   client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
   ```

2. **Send Data:**

   Use sendto() to send data to the server's IP address and port.

   ```
   client_socket.sendto(data, ('localhost', 12345))
   ```

3. **Receive Response:**

   Use recvfrom() to receive the response from the server.

   ```
   data, server = client_socket.recvfrom(1024)
   ```

4. **Close Socket:**
   client_socket.close()

UDP Server Side implementation:

```
while True:
    data, client_address = server_socket.recvfrom(1024)
    print(f"Received message from client: {data.decode()}")
    message = input("Enter reply: ")
    server_socket.sendto(message.encode(), client_address)
```

UDP Client Side implementation:

```
while True:
    message = input("Enter message: ")
    client_socket.sendto(message.encode(), ('localhost', 12345))
    if message.lower() == "exit":
        break
    data, server = client_socket.recvfrom(1024)
    print(f"Received reply from server: {data.decode()}")
```