Programming Lab: UDP+TCP

This lab aims to practice **implementing client-server communication using TCP (Transmission Control Protocol) and UDP (User Datagram Protocol).** By analyzing the provided code and running the applications, the goals are:

1. Understand the main differences between TCP (connection-oriented) and UDP (connectionless).
2. Observe the behavior of the protocol in real scenarios, such as handshakes (TCP), data transmission and error handling.
3. Compare the reliability of TCP with the efficiency of UDP using the figures generated.

tcp_client.py M

Lab 2 > tcp_client.py > ...

You, 1 second ago | 1 author (You)

```python
from socket import * # Imports all symbols from the socket module

serverName = 'localhost' # Defines the server address (same machine in this case)
serverPort = 12000 # Defines the port number to connect to

clientSocket = socket(AF_INET, SOCK_STREAM) # Creates TCP socket (SOCK_STREAM means TCP)
clientSocket.connect((serverName,serverPort)) # Establishes connection with server

sentence = input('Input lowercase sentence:') # Gets user input from keyboard
clientSocket.send(sentence.encode()) # Converts string to bytes and sends to server

modifiedSentence = clientSocket.recv(1024).decode() # Receives response and converts bytes to string
print('From Server:', modifiedSentence) # Prints the modified message from server

clientSocket.close() # Closes the connection
```

Figure 1 - TCP Client

Functionality: The TCP client initiates a connection to the server (connect()), sends a message (e.g. “hello”) and waits for a modified response (e.g. “HELLO”).

Note: The connection is established before the data is exchanged (handshake), ensuring reliable delivery.

tcp_server.py

Lab 2 > tcp_server.py > ...

You, 17 minutes ago | 1 author (You)

```python
from socket import * # Imports all symbols from the socket module

serverPort = 12000 # Defines the port number the server will listen on
serverSocket = socket(AF_INET,SOCK_STREAM) # Creates a TCP socket (SOCK_STREAM indicates TCP)
serverSocket.bind(('',serverPort)) # Binds the socket to local address and specified port
serverSocket.listen(1) # Sets up the socket to accept connections, with max queue of 1
print('The server is ready to receive') # Indicates server is ready

while 1: # Infinite loop to handle multiple clients sequentially
    connectionSocket, addr = serverSocket.accept() # Waits and accepts new connection, returns new socket and client address
    sentence = connectionSocket.recv(1024).decode() # Receives data from client and converts bytes to string
    capitalizedSentence = sentence.upper() # Converts received string to uppercase
    connectionSocket.send(capitalizedSentence.encode()) # Sends response converted to bytes      You, 17 minutes ago • Lab 2 - 17,
    connectionSocket.close() # Closes the connection with current client
```

Figure 2 - TCP Server

Functionality: The TCP server listens for connections (listen()), accepts one (accept()), processes the message received (e.g. converts to upper case) and sends the reply.

Note: After sending, the connection is closed (close()), following a “one-to-one” model.

udp_client.py

Lab 2 > udp_client.py > ...

```python
from socket import * # Imports all socket-related symbols

serverName = 'localhost'  # Sets server address to local machine
serverPort = 12000 # Defines the destination port number

clientSocket = socket(AF_INET, SOCK_DGRAM) # Creates UDP socket (SOCK_DGRAM indicates UDP)
message = input('Input lowercase sentence:') # Gets user input from keyboard
clientSocket.sendto(message.encode(), (serverName, serverPort)) # Sends encoded message to server with destination address
modifiedMessage, serverAddress = clientSocket.recvfrom(2048) # Receives response and server address, buffer size 2048 bytes

print(modifiedMessage.decode()) # Prints decoded message from server
clientSocket.close() # Closes the UDP socket
```

Figure 3 - UDP Client

Functionality: The UDP client sends datagrams directly to the server (sendto()) without establishing a connection. There is no guarantee of delivery or order.

Note: Messages can get lost or arrive out of order, but the protocol is faster due to lower overhead.

udp_server.py

Lab 2 > udp_server.py > ...

You, 17 minutes ago | 1 author (You)

```python
from socket import * # Imports all socket-related symbols      You, 17 minutes ago • Lab 2 - 17/02/2025

serverPort = 12000 # Defines the port number the server will listen on

serverSocket = socket(AF_INET, SOCK_DGRAM) # Creates UDP socket (SOCK_DGRAM indicates UDP)

serverSocket.bind(('', serverPort)) # Binds the socket to all available interfaces on specified port

print("The server is ready to receive") # Indicates server is ready

while 1: # Infinite loop to handle multiple client requests
    message, clientAddress = serverSocket.recvfrom(2048) # Receives message and client address, buffer size 2048 bytes
    modifiedMessage = message.decode().upper() # Decodes received bytes and converts to uppercase
    serverSocket.sendto(modifiedMessage.encode(), clientAddress) # Sends encoded response back to client
```

Figure 4 - UDP Server

Functionality: The UDP server waits for datagrams (recvfrom()), processes the message and sends the reply to the client address.

Note: No connection state, allowing “one-to-many” or “many-to-many” communication.