Computer Networks Lab 8

TITLE: ***Implementation of Socket Programming.***

Objective:

Implementation of Socket Programming program using
a. TCP Client Server programming.
b. UDP Client Server Programming.

Problem Statement:

TCP is an abbreviation for Transmission Control Protocol. It is a transport layer protocol that facilitates the transmission of packets from source to destination. It is a connection-oriented protocol. It means that after establishing a stable connection, one can easily transmit data in two directions.

UDP stands for User Datagram Protocol. UDP is the simplest transport layer communication protocol. It contains a minimum amount of communication mechanisms. It is considered an unreliable protocol, and it is based on best-effort delivery services. UDP provides no acknowledgment mechanism, which means that the receiver does not send the acknowledgment for the received packet, and the sender also does not wait for the acknowledgment for the packet that it has sent.

Algorithm:

TCP Server:

1. using create(), Create a TCP socket.
2. using bind(), Bind the socket to the server address.
3. using listen(), put the server socket in a passive mode, where it waits for the client to approach the server to make a connection

4. using accept(), At this point, the connection is established between the client and server, and they are ready to transfer data.
5. Go back to Step 3.

TCP Client:

1. Create a TCP Socket.
2. Connect the newly created client socket to the server.

UDP Server:

1. Create a UDP socket.
2. Bind the socket to the server address.
3. Wait until the datagram packet arrives from the client.
4. Process the datagram packet and send a reply to the client.
5. Go back to Step 3.

UDP Client:

1. Create a UDP socket.
2. Send a message to the server.
3. Wait until a response from the server is received.
4. Process the reply and go back to step 2, if necessary.
5. Close the socket descriptor and exit.

CODE:

TCP Client Side:

```
import socket
host = 'localhost'
port = 9900
# Create a TCP/IP socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Connect the socket to the server
server_address = (host, port)
print("Connecting to %s port %s" % server_address)
s.connect(server_address)
# Send data
try:
 message = input("Write a message:")
 print("Sending %s" % message)
 s.sendall(message.encode('utf-8'))
```

```
# Look for the response
 amount_received = 0
 amount_expected = len(message)
while amount_received < amount_expected:
 data = s.recv(16)
 amount_received += len(data)
 print("-------------at server-------------")
 print("Received: %s" % data)
print("---------------------------------")
except socket.error as e:
print("Socket error: %s" % str(e))
except Exception as e:
print("Other exception: %s" % str(e))
finally:
print("Closing connection to the server")
s.close()
```

```
Connecting to localhost port 9900
Write a message:Welcome
Sending Welcome
-------------at server-------------
Received: b'Welcome'
---------------------------------
Closing connection to the server
```

Explanation:

A TCP socket is created and the socket is connected to the server with port 9900. After the connection is established, the client sends the message “Welcome” to the server and waits for the response. If the message is received it is acknowledged by the client. The connection is closed.

TCP Server Side:

```
import socket
host = 'localhost'
data_payload = 2048
port = 9900
# Create a TCP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Enable reuse address/port
```

```
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
# Bind the socket to the port
server_address = (host, port)
print("Starting up echo server on %s port %s" % server_address)
sock.bind(server_address)
# Listen to clients, backlog argument
sock.listen(5)
while True:
print("Waiting to receive message from client")
client, address = sock.accept()
data = client.recv(data_payload)
if data:
 print("Data: %s" % data)
 client.send(data)
 print("sent %s bytes back to %s" % (data, address))
# end connection
client.close()
```

```
Starting up echo server on localhost port 9900
Waiting to receive message from client
Data: b'Welcome'
sent b'Welcome' bytes back to ('127.0.0.1', 57894)
Waiting to receive message from client
```

Explanation:

The socket server running on port 9900 will wait for the client's request and the server will successfully received the client's data. The acknowledgment is sent back to the client by the server.

UDP Client Side:

```
import socket
host = 'localhost'
data_payload = 2048
port = 9900
# Create a UDP socket
sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
server_address = (host, port)
print ("Connecting to %s port %s" % server_address)
try:
# Send data
```

```
message = input("Enter your message:")
print ("Sending %s" % message)
sent = sock.sendto(message.encode('utf-8'), server_address)
# Receive response
data, server = sock.recvfrom(data_payload)
print("----------At server----------")
print ("received %s" % data)
print("---------------------------------")
finally:
print ("Closing connection to the server")
sock.close()
```

```
Connecting to localhost port 9900
Enter your message:Welcome
Sending Welcome
----------At server----------
received b'Welcome'
---------------------------------
Closing connection to the server
```

UDP Server Side:

```
import socket
host = 'localhost'
data_payload = 2048
port = 9900
# Create a UDP socket
sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
# Bind the socket to the port
server_address = (host, port)
print ("Starting up echo server on %s port %s" % server_address)
sock.bind(server_address)
while True:
print ("Waiting to receive message from client")
data, address = sock.recvfrom(data_payload)
print ("received %s bytesfrom %s" % (len(data), address))
print ("Data: %s" %data)
if data:
 sent = sock.sendto(data, address)
 print("sent %s bytes back to %s" % (sent,address))
```

```
Starting up echo server on localhost port 9900
Waiting to receive message from client
received 7 bytesfrom ('127.0.0.1', 51327)
Data: b'Welcome'
sent 7 bytes back to ('127.0.0.1', 51327)
Waiting to receive message from client
```

Problems Faced:

Initially, I found it difficult to establish the connection between the client and server in both protocols.

Conclusion:

From this experiment, we can infer that UDP is used for sharing short messages while TCP is used for long messages. Despite its unreliability, UDP is message-oriented. It establishes boundaries for communication. The advantage of the TCP protocol is that it is not message-oriented and does not impose limits on exchanging messages.