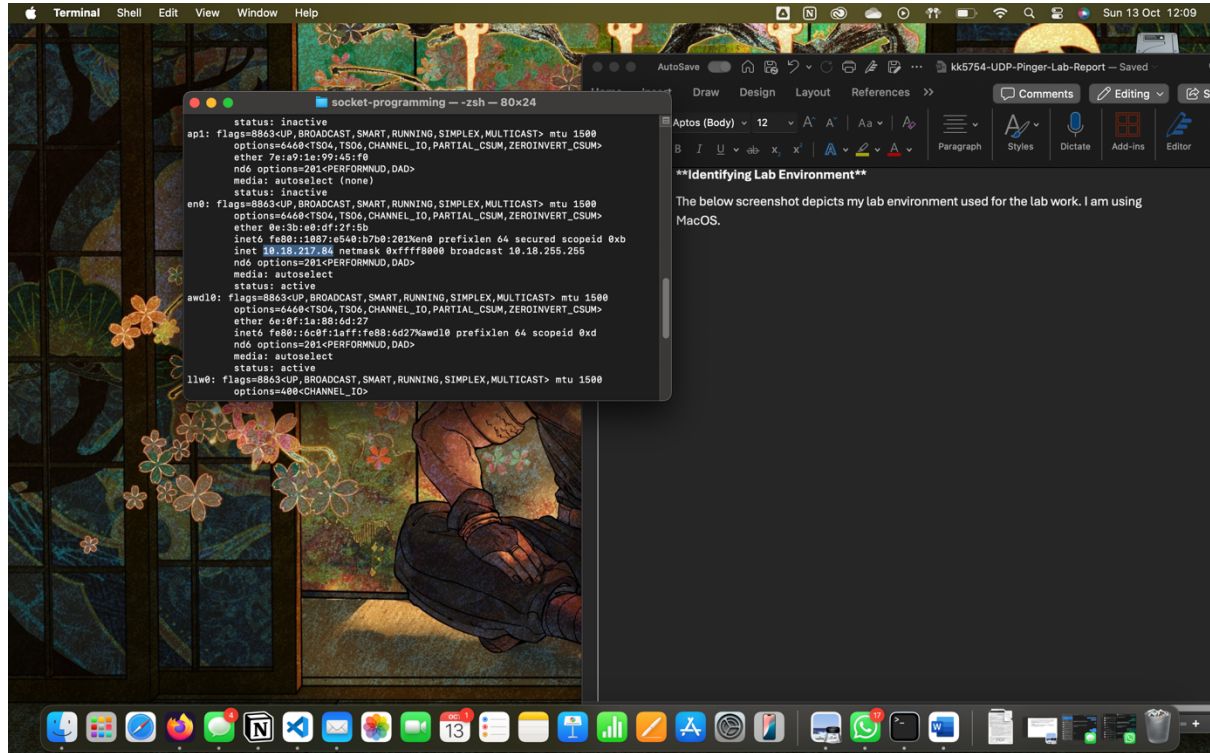


Computer Networking Socket Programming Lab 2: UDP Pinger Lab

****Identifying Lab Environment****

The below screenshot depicts my lab environment used for the lab work. I am using MacOS with Python version 3.9.11.



1. UDP Pinger Lab: Client Code Completed with commented lines for explanation of code

client.py

```
#From the skeleton code let's build the client code
from socket import *
from time import time, ctime
import sys

def ping(host, port):
    resps = [] #List for storing response
    clientSocket = socket(AF_INET, SOCK_DGRAM) #UDP Socket
    clientSocket.settimeout(1) #setting timeout value of 1s

    for seq in range(1, 11):
        startTime = time() #time capture when ping is sent
        message = f"Ping {seq} {ctime(startTime)[11:19]}" #ping message

        try:
            clientSocket.sendto(message.encode(), (host, port))

            encodedModified, serverAddress = clientSocket.recvfrom(1024) #receiving
message and measuring RTT
            endTime = time() #time capture when message is received

            server_reply = encodedModified.decode()
            rtt = (endTime - startTime) * 1000 #rtt in ms

            print(f"Reply from {serverAddress}: {server_reply}")
            print(f"RTT: {rtt:.3f} ms\n")

            resps.append((seq, server_reply, rtt)) #appending to resps list

        except timeout:
            print(f"PING {seq} Request timed out\n")
            resps.append((seq, 'Request timed out', 0))

    #Close the client socket after all pings
    clientSocket.close()

    return resps

if __name__ == '__main__':
    resps = ping('127.0.0.1', 12000)
    print(resps)
```

Output:

Name: Krapa Karthik **NYU ID:** N12039854 **NET ID:** kk5754

Running the UDPPingerServer.py

The image shows a VS Code editor interface with a Python project for socket programming. The Explorer sidebar on the left lists the following files: `socket-...`, `EnhancedPingCli...`, `HeartbeatClient.py`, `HeartbeatServer.py`, `EnhancedPingClient.py`, `UDPPingerClient.py`, `LICENSE`, `README.md`, `Screenshot 2024...`, `UDPPingerClient...`, and `UDPPingerServer...`. The main editor window displays the `UDPPingerClient.py` file, which contains a `ping` function. The function sends a message to a server at port 1024, receives a response, and calculates the Round Trip Time (RTT). The output shows the server address and the RTT in milliseconds. The TERMINAL panel at the bottom shows the command `python UDPPingerServer.py` being executed, resulting in a list of ping responses with timestamps.

```
def ping(host, port):
    clientSocket.sendto(message.encode(), (host, port))

    encodedModified, serverAddress = clientSocket.recvfrom(1024) #receiving message and measuring RTT
    endTime = time() #time capture when message is received

    server_reply = encodedModified.decode()
    rtt = (endTime - startTime) * 1000 #rtt in ms

    print(f"Reply from {serverAddress}: {server_reply}")
    print(f"RTT: {rtt:.3f} ms\n")

    resps.append((seq, server_reply, rtt)) #appending to resps list
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

```
(base) karthik.krapa18@10-18-217-84 socket-programming % python UDPPingerServer.py
['Ping', '3', '11:58:27']
['Ping', '6', '11:58:29']
['Ping', '9', '11:58:31']
['Ping', '10', '11:58:31']
```

Running the client.py

The screenshot displays the VS Code interface with a Python file named `UDPPingerClient.py` open. The script implements a UDP ping client that sends a message to a server at `127.0.0.1:12000` and measures the round-trip time (RTT). The terminal output shows the execution of the script, displaying the RTT for each of the 10 requests.

```

16 def ping(host, port):
17     clientSocket.sendto(message.encode(), (host, port))
18
19     encodedModified, serverAddress = clientSocket.recvfrom(1024) #receiving message and measuring RTT
20     endTime = time() #time capture when message is received
21
22     server_reply = encodedModified.decode()
23     rtt = (endTime - startTime) * 1000 #rtt in ms
24
25     print(f"Reply from {serverAddress}: {server_reply}")
26     print(f"RTT: {rtt:.3f} ms\n")
27
28     resps.append((seq, server_reply, rtt)) #appending to resps list

```

Terminal Output:

```

(base) karthik.krapal@10-18-217-84 socket-programming % python UDPPingerClient.py
PING 1 Request timed out
PING 2 Request timed out
Reply from ('127.0.0.1', 12000): Reply 3 11:58:27 1728835107.990788 3aff2ecc8b043a25c644e982f3b93bc
RTT: 0.549 ms
PING 4 Request timed out
PING 5 Request timed out
Reply from ('127.0.0.1', 12000): Reply 6 11:58:29 1728835109.994066 8cdc57f31af5f96bd7f0481c51cf3e5
RTT: 0.609 ms
PING 7 Request timed out
PING 8 Request timed out
Reply from ('127.0.0.1', 12000): Reply 9 11:58:31 1728835111.997313 2dfa018ceba3df8794e7c9f8fec14658
RTT: 0.571 ms
Reply from ('127.0.0.1', 12000): Reply 10 11:58:31 1728835111.997771 87bc3a3c966f8b62a10ac19512e7c6e1
RTT: 0.282 ms
[[('1. Request timed out', 0), ('2. Request timed out', 0), ('3. Reply 3 11:58:27 1728835107.990788 3aff2ecc8b043a25c644e982f3b93bc\n', 0.549316486725), ('4. Request timed out', 0), ('5. Request timed out', 0), ('6. Reply 6 11:58:29 1728835109.994066 8cdc57f31af5f96bd7f0481c51cf3e5\n', 0.609154696044922), ('7. Request timed out', 0), ('8. Request timed out', 0), ('9. Reply 9 11:58:31 1728835111.997313 2dfa018ceba3df8794e7c9f8fec14658\n', 0.5707740783691406), ('10. Reply 10 11:58:31 1728835111.997771 87bc3a3c966f8b62a10ac19512e7c6e1\n', 0.28228759765625)]]
(base) karthik.krapal@10-18-217-84 socket-programming %

```

- ## 2. Optional Exercise: Enhanced UDPPingerClient with RTT Statistics and Packet Loss Rate

EnhancedPingClient.py

```
from socket import *
from time import time, ctime
import sys

def ping(host, port):
    rtt = [] #to store the rtt
    packet_loss_count = 0 #packet lost counter

    clientSocket = socket(AF_INET, SOCK_DGRAM)
    clientSocket.settimeout(1)

    for seq in range(1, 11):
        startTime = time()
        message = f"Ping {seq} {ctime(startTime)[11:19]}"

        try:
            clientSocket.sendto(message.encode(), (host, port))

            encodedModified, serverAddress = clientSocket.recvfrom(1024)
            endTime = time()

            server_reply = encodedModified.decode()
            rtt = (endTime - startTime) * 1000 #rtt in ms
            rtt.append(rtt) #storing the rtt in list for getting stat report on
this

            print(f"Reply from {serverAddress}: {server_reply}")
            print(f"RTT: {rtt:.3f} ms\n")

        except timeout:
            print(f"PING {seq} Request timed out\n")
            packet_loss_count += 1

    #closing client
    clientSocket.close()

    #rtt stats
    if rtt:
        print("UDP Pinger Report:")
        print(f"Maximum RTT: {max(rtt):.3f} ms") #max value from the list
        print(f"Minimum RTT: {min(rtt):.3f} ms") #min value from the list
        print(f"Average RTT: {sum(rtt)/len(rtt):.3f} ms") #avg calculation from
the list
    else:
        print("No successful pings, all requests timed out.")

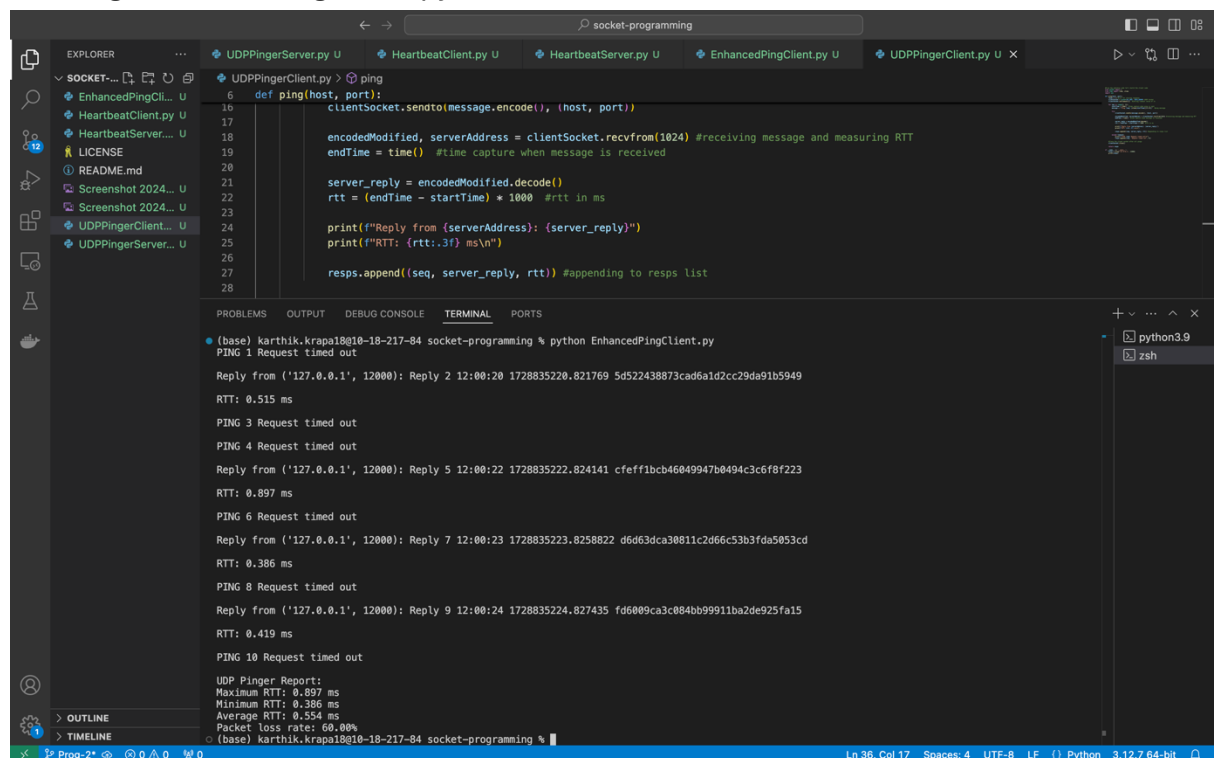
    #Calculating Packet Loss Rate
```

```
total_packets = 10 #since 10 Pings were made from the client skeleton code earlier
packet_loss_rate = (packet_loss_count / total_packets) * 100
print(f"Packet loss rate: {packet_loss_rate:.2f}%")

if __name__ == '__main__':
    ping('127.0.0.1', 12000)
```

Output:

Running EnhancedPingClient.py



```
socket-programming
UDPPingerServer.py U HeartbeatClient.py U HeartbeatServer.py U EnhancedPingClient.py U UDPPingerClient.py U X
6 def ping(host, port):
16     clientSocket.sendto(message.encode(), (host, port))
17
18     encodedModified, serverAddress = clientSocket.recvfrom(1024) #receiving message and measuring RTT
19     endTime = time() #time capture when message is received
20
21     server_reply = encodedModified.decode()
22     rtt = (endTime - startTime) * 1000 #rtt in ms
23
24     print(f"Reply from {serverAddress}: {server_reply}")
25     print(f"RTT: {rtt:.3f} ms\n")
26
27     resps.append((seq, server_reply, rtt)) #appending to resps list
28
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
(base) karthik.krapa18@10-18-217-84 socket-programming % python EnhancedPingClient.py
PING 1 Request timed out

Reply from ('127.0.0.1', 12000): Reply 2 12:00:20 1728835220.821769 5d522438873cad6a1d2cc29da91b5949
RTT: 0.515 ms

PING 3 Request timed out
PING 4 Request timed out

Reply from ('127.0.0.1', 12000): Reply 5 12:00:22 1728835222.824141 cfeff1bcb46049947b0494c3c6f8f223
RTT: 0.897 ms

PING 6 Request timed out

Reply from ('127.0.0.1', 12000): Reply 7 12:00:23 1728835223.8258822 d6d63dca30811c2d66c53b3fda5053cd
RTT: 0.386 ms

PING 8 Request timed out

Reply from ('127.0.0.1', 12000): Reply 9 12:00:24 1728835224.827435 fd6009ca3c084bb99911ba2de925fa15
RTT: 0.419 ms

PING 10 Request timed out

UDP Pinger Report:
Maximum RTT: 0.897 ms
Minimum RTT: 0.386 ms
Average RTT: 0.554 ms
Packet loss rate: 0.00%
(base) karthik.krapa18@10-18-217-84 socket-programming %
```

3. Optional Exercise: UDP Heartbeat

HeartbeatServer.py

```
from socket import *
import time
import sys

def serve(port, heartbeat_timeout=5):
    serverSocket = socket(AF_INET, SOCK_DGRAM)
    serverSocket.bind(('', port))
    last_seq = -1 #to keep track of sequence number for detecting lost packets
    last_heartbeat_time = time.time() #the last heartbeat was received time

    print("Server is ready to receive heartbeat signals.")
```

```
while True:
    try:
        message, address = serverSocket.recvfrom(1024)
        current_time = time.time()
        heartbeat = message.decode().split() #heartbeat message decode
        seq = int(heartbeat[1])
        c_time = float(heartbeat[2])

        print(f"Received Heartbeat {seq} from {address} at {current_time}, sent
at {c_time}")

        if seq != last_seq + 1 and last_seq != -1: #checking packet loss
            print(f"Warning: Packet loss detected! Expected seq {last_seq + 1},
but got {seq}.")

        if current_time - last_heartbeat_time > heartbeat_timeout: #checking
heartbeat timeout
            print("Warning: Heartbeat timeout. Client may have stopped.")

        #Updating last sequence number and heartbeat time
        last_seq = seq
        last_heartbeat_time = current_time

    except KeyboardInterrupt:
        serverSocket.close()
        sys.exit()

if __name__ == '__main__':
    serve(12000)
```

HeartbeatClient.py

```
from socket import *
import time
import sys

def heartbeat(host, port, interval=2):
    seq = 0 #Sequence number for heartbeat messages

    clientSocket = socket(AF_INET, SOCK_DGRAM)

    while True:
        try:
            current_time = time.time() #Current time
            message = f"Heartbeat {seq} {current_time}"

            #Sending heartbeat to server
            clientSocket.sendto(message.encode(), (host, port))
```

```
print(f"Sent Heartbeat {seq} at {current_time}")
seq += 1

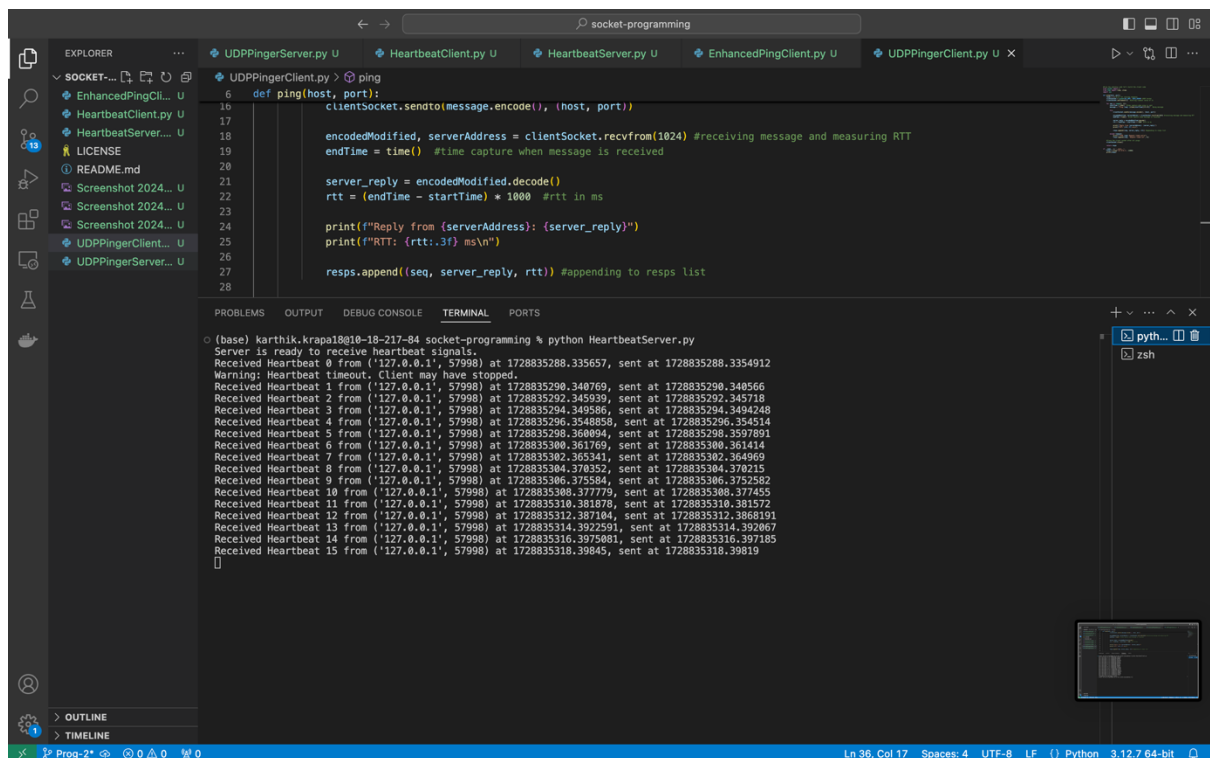
#Wait for the specified interval before sending the next heartbeat
time.sleep(interval)

except KeyboardInterrupt:
    print("Stopping the heartbeat client.")
    clientSocket.close()
    sys.exit()

if __name__ == '__main__':
    heartbeat('127.0.0.1', 12000)
```

Output:

Running HeartbeatServer.py



The screenshot shows a VS Code editor with the file `HeartbeatServer.py` open. The code defines a `ping` function that sends a heartbeat message to a client and receives a response. The terminal output shows the server running and receiving heartbeats from the client. The output is as follows:

```
[base] karthik.krapa18010-18-217-84 socket-programming % python HeartbeatServer.py
Server is ready to receive heartbeat signals.
Received Heartbeat 0 from ('127.0.0.1', 57998) at 1728835288.335657, sent at 1728835288.3354912
Warning: Heartbeat timeout. Client may have stopped.
Received Heartbeat 1 from ('127.0.0.1', 57998) at 1728835290.340769, sent at 1728835290.340566
Received Heartbeat 2 from ('127.0.0.1', 57998) at 1728835292.345939, sent at 1728835292.345718
Received Heartbeat 3 from ('127.0.0.1', 57998) at 1728835294.349586, sent at 1728835294.349428
Received Heartbeat 4 from ('127.0.0.1', 57998) at 1728835296.3548858, sent at 1728835296.354514
Received Heartbeat 5 from ('127.0.0.1', 57998) at 1728835298.360894, sent at 1728835298.3597891
Received Heartbeat 6 from ('127.0.0.1', 57998) at 1728835300.361769, sent at 1728835300.361414
Received Heartbeat 7 from ('127.0.0.1', 57998) at 1728835302.365341, sent at 1728835302.364969
Received Heartbeat 8 from ('127.0.0.1', 57998) at 1728835304.370352, sent at 1728835304.370215
Received Heartbeat 9 from ('127.0.0.1', 57998) at 1728835306.375584, sent at 1728835306.3752582
Received Heartbeat 10 from ('127.0.0.1', 57998) at 1728835308.377779, sent at 1728835308.377455
Received Heartbeat 11 from ('127.0.0.1', 57998) at 1728835310.381878, sent at 1728835310.381572
Received Heartbeat 12 from ('127.0.0.1', 57998) at 1728835312.387104, sent at 1728835312.3868191
Received Heartbeat 13 from ('127.0.0.1', 57998) at 1728835314.3922591, sent at 1728835314.392067
Received Heartbeat 14 from ('127.0.0.1', 57998) at 1728835316.3975081, sent at 1728835316.397185
Received Heartbeat 15 from ('127.0.0.1', 57998) at 1728835318.39845, sent at 1728835318.39819
[]
```

Running HeartbeatClient.py

Name: Krapa Karthik NYU ID: N12039854 NET ID: kk5754

```
6 def ping(host, port):
16     clientSocket.sendto(message.encode(), (host, port))
17
18     encodedModified, serverAddress = clientSocket.recvfrom(1024) #receiving message and measuring RTT
19     endTime = time() #time capture when message is received
20
21     server_reply = encodedModified.decode()
22     rtt = (endTime - startTime) * 1000 #rtt in ms
23
24     print(f"Reply from {serverAddress}: {server_reply}")
25     print(f"RTT: {rtt:.3f} ms\n")
26
27     resps.append((seq, server_reply, rtt)) #appending to resps list
28
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
• (base) karthik.krapa18@10-18-217-84 socket-programming % python HeartbeatClient.py
Sent Heartbeat 0 at 1728835288.3354912
Sent Heartbeat 1 at 1728835290.340566
Sent Heartbeat 2 at 1728835292.345718
Sent Heartbeat 3 at 1728835294.3494248
Sent Heartbeat 4 at 1728835296.354514
Sent Heartbeat 5 at 1728835298.3597891
Sent Heartbeat 6 at 1728835300.361414
Sent Heartbeat 7 at 1728835302.364969
Sent Heartbeat 8 at 1728835304.370215
Sent Heartbeat 9 at 1728835306.3752582
Sent Heartbeat 10 at 1728835308.377455
Sent Heartbeat 11 at 1728835310.381572
Sent Heartbeat 12 at 1728835312.3868191
Sent Heartbeat 13 at 1728835314.392067
Sent Heartbeat 14 at 1728835316.397185
Sent Heartbeat 15 at 1728835318.39819
^CStopping the heartbeat client.
(base) karthik.krapa18@10-18-217-84 socket-programming %
```

python3.9
zsh

Ln 36, Col 17 Spaces: 4 UTF-8 LF Python 3.12.7 64-bit