

Basant Solanky

12040430

Assignment- 3

Socket Programming

Part 1: Implement SSH using socket programming. [20 Points]

- **Run the server script first:** python Server.py
Enter the port number (1024-65536)
- **Run the client script:** python Client.py
Port number for this will be same as above entered.

Client.py

This file contains the client side code executed on the server port which take input from the user as username and the corresponding password. If the password matches in the server.py file then connection will be established.

```

1 import socket
2
3 client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
4 port=int(input("Enter the port number:- "))
5 client_socket.connect(('127.0.0.1', port))
6
7
8 username = input("Enter Username: ")
9 password = input("Enter password: ")
10
11
12
13 client_socket.send(username.encode('utf-8'))
14
15 auth_response = client_socket.recv(3).decode('utf-8')
16
17
18 if auth_response == 'yes':
19     client_socket.send(password.encode('utf-8'))
20     auth_response1 = client_socket.recv(3).decode('utf-8')
21
22     if auth_response1 == 'yes':
23         # Display system information
24         welcome_message = client_socket.recv(1024).decode('utf-8')
25         print(welcome_message)
26
27     while True:
28         command = input(f"{username}@remoteserver$ ")
29         client_socket.send(command.encode('utf-8'))
30
31         if command == 'exit':
32             break
33
34         # Receive and display the command output
35         output = client_socket.recv(1024).decode('utf-8')
36         print(output)
37
38     else:
39         print("Authentication failed")
40
41 else:
42     print("Authentication failed")
43
44 client_socket.close()

```

Server.py

This is server.py file which contains multi-users username and their password and the last login session which is stored in a text file named ‘last_login.txt’. Once the credentials are matched it sends a welcome message to the client, and on server its shows system informations.

```
 1 import socket
 2 import os
 3 import datetime
 4 import psutil
 5 credentials = {
 6     "mafia": "mafia",
 7     "basant": "localhost",
 8     "arch": "penguin"
 9     "Admin": "root"
10 }
11
12 last_login_file = "last_login.txt"
13
14 def authenticate_username(input_username):
15     return input_username in credentials
16
17 def authenticate_password(username, input_passwd):
18     return credentials.get(username) == input_passwd
19
20 def execute_command(command):
21     try:
22         result = os.popen(command).read()
23         return result
24     except Exception as e:
25         return f"Command Execution Error: {str(e)}"
26
27 #
28 def fetch_system_info():
29     memory_usage = psutil.virtual_memory().percent
30     processes = psutil.cpu_count(logical=False)
31     ipv4_address = socket.gethostbyname(socket.gethostname())
32     last_access_time = datetime.datetime.now().strftime("%a %d-%m-%y %H:%M:%S")
33     print(f"Client Connection successful ")
34     print("System Information")
35     print(f"Memory Usage: {memory_usage}%")
36     print(f"Processes: {processes}")
37     print(f"IPv4 Address: {ipv4_address}")
38     print(f"Last Access: {last_access_time}")
39
40     return f"\nWelcome to my server:\n"
41
42 def update_last_login():
43     with open(last_login_file, 'w') as file:
44         file.write(datetime.datetime.now().strftime("%a %d-%m-%y %H:%M:%S"))
```

```

46 def main():
47     server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
48
49     port = int(input("Enter the port number: "))
50     server_socket.bind(('127.0.0.1', port))
51     server_socket.listen(1)
52
53     print("Establishing secure connections...")
54     client_socket, client_address = server_socket.accept()
55     print(f"Connection established with {client_address}")
56
57     # Authentication
58     input_username = client_socket.recv(1024).decode('utf-8')
59     if authenticate_username(input_username):
60         client_socket.send(b'yes')
61
62         input_passwd = client_socket.recv(1024).decode('utf-8')
63         if authenticate_password(input_username, input_passwd):
64             client_socket.send(b'yes')
65
66             update_last_login()
67
68             welcome_msg = fetch_system_info()
69             client_socket.send(welcome_msg.encode('utf-8'))

```

```

71     while True:
72         command = client_socket.recv(1024).decode('utf-8')
73         if command == 'exit':
74             print("Client disconnected\nBye.")
75             break
76             # continue
77
78         if command.startswith('cd '):
79             os.chdir(command[3:])
80             client_socket.send(b'Directory changed.')
81         elif command.startswith('mkdir '):
82             os.mkdir(command[6:])
83             client_socket.send(b'Directory created.')
84         elif command.startswith('touch '):
85             open(command[6:], 'a').close()
86             client_socket.send(b'File created.')
87         elif command.startswith('cat '):
88             try:
89                 with open(command[4:], 'r') as file:
90                     content = file.read()
91                     client_socket.send(content.encode('utf-8'))
92             except FileNotFoundError:
93                 client_socket.send(b'File not found.')
94         elif command.startswith('echo '):
95             try:
96                 content = command[5:]
97
98                 if ' > ' in content:
99                     parts = content.split(' > ', 1)
100                    file_name = parts[-1].strip()
101                    with open(file_name, 'w') as file:
102                        file.write(parts[0])
103                        client_socket.send(b'Content updated.')
104                else:
105                    client_socket.send(content.encode('utf-8'))
106            except Exception as e:
107                client_socket.send(f'Error: {str(e)}'.encode('utf-8'))
108            elif command.startswith('ls') or command.startswith('echo'):
109                output = execute_command(command)
110                client_socket.send(output.encode('utf-8'))
111            else:
112                client_socket.send(b'Still learning new commands')
113
114            else:
115                client_socket.send(b'no')
116                print("Client disconnected")
117        else:
118            client_socket.send(b'no')
119            print("Authentication Failed:")
120
121        client_socket.close()
122        server_socket.close()
123
124 if __name__ == "__main__":
125     main()

```

This snippet shows various commands we can use in the ssh server such as:- cd, mkdir, echo, ls, touch, cat.

The screenshot displays a terminal window with two panes. The left pane shows the output of a Python script named `Server.py`. It prints system information, including memory usage (67.2%), processes (4), and last access (Wed 22-11-23 15:26:25). It also prints a message for a disconnected client and ends with a `Bye.` The right pane shows the output of a Python script named `Client.py`. It prompts for a port number (7777), username (mafia), and password (mafia). It then connects to the server and prints a welcome message. It lists files in the current directory: `Client.py`, `last_login.txt`, and `Server.py`. It creates a file named `README.md`, updates its content with "Welcome to my world", and creates a new folder named `anotherFolder`. Finally, it exits the session.

```
1 Part1
q ~ /Doc/A/Part1 > p Server.py
Enter the port number: 7777
Establishing secure connections...
Connection established with ('127.0.0.1', 38434)
Client Connection successful
System Information
Memory Usage: 67.2%
Processes: 4
IPv4 Address: 127.0.1.1
Last Access: Wed 22-11-23 15:26:25
Client disconnected
Bye.

q ~ /Doc/A/Part1 > 3m 37s 15:26:07

q ~ /Doc/A/Part1 > p Client.py
Enter the port number:- 7777
Enter Username: mafia
Enter password: mafia

Welcome to my server:)

mafia@remoteserver$ ls
Client.py
last_login.txt
Server.py

mafia@remoteserver$ touch README.md
File created.
mafia@remoteserver$ echo "Welcome to my world" > README.md
Content updated.
mafia@remoteserver$ cat README.md
"Welcome to my world"
mafia@remoteserver$ mkdir anotherFolder
Directory created.
mafia@remoteserver$ cd anotherFolder
Directory changed.
mafia@remoteserver$ exit
q ~ /Doc/A/Part1 > 3m 31s 15:29:46
```

Workflow with suitable commands

Part 2: Build a DNS resolver application using socket programming. [15 Points]

Client.py

```
1 import socket
2
3 hostname = socket.gethostname()
4 IPA = '127.0.0.1'
5 s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
6 port=int(input("Enter port number: "))
7 addr = (IPA, port)
8
9 c = "Y"
10 transaction_id = 1
11
12 while c.upper() == "Y":
13     req_domain = input("Enter domain name: ")
14     data_to_send = req_domain
15     send = s.sendto(data_to_send.encode(), addr)
16     # data_another =transaction_id
17     # send_another=s.sendto(data_another.encode(),addr)
18     data, address = s.recvfrom(1024)
19     reply_ip = data.decode().strip()
20     print(f"{transaction_id} {req_domain} {reply_ip}")
21     transaction_id += 1
22     c = input("Continue? (y/n) ")
23
24 s.close()
```

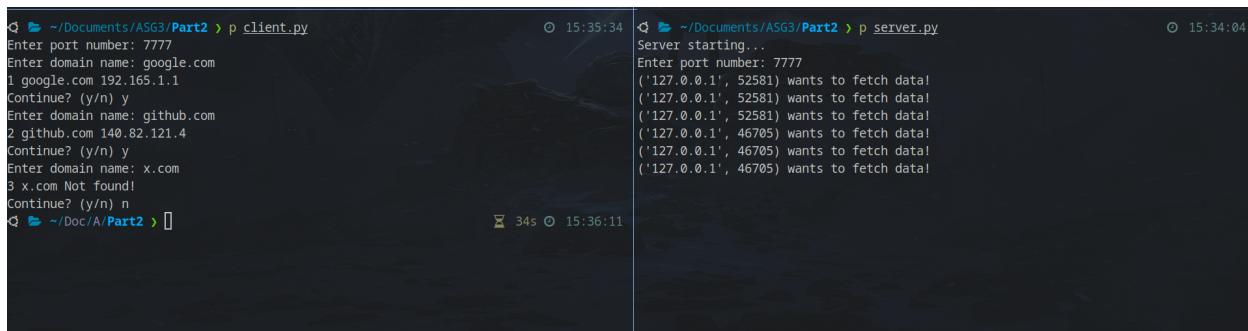
Server.py

```
1 import socket
2
3 def read_dns_table(file_path):
4     dns_table = {}
5     with open(file_path, 'r') as file:
6         for line in file:
7             parts = line.strip().split()
8             transaction_id, domain, ip = int(parts[0]), parts[1], parts[2]
9             dns_table[domain]=ip
10    return dns_table
11
12 file_path = 'query'
13 dns_table = read_dns_table(file_path)
14
15 s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
16 print("Server starting...")
17
18 port=int(input("Enter port number: "))
19 s.bind(("127.0.0.1", port))
20
21 while True:
22     data, address = s.recvfrom(1024)
23     print(f"{address} wants to fetch data!")
24     data = data.decode()
25     ip = dns_table.get(data, "Not found!").encode()
26     s.sendto(ip, address)
```

DNS queries are stored in a file and server.py fetches the IPAddress whenever client asks for.

```
1 1 google.com 192.165.1.1
2 2 example.com 203.0.113.1
3 3 facebook.com 192.168.0.1
4 4 amazon.com 172.217.10.14
5 5 twitter.com 104.244.42.1
6 6 reddit.com 151.101.1.140
7 7 linkedin.com 108.174.10.10
8 8 github.com 140.82.121.4
```

User asking for IPAddress for the domains which exists in the database.



The screenshot shows two terminal windows. The left window is titled 'client.py' and shows the user entering domain names and port numbers. The right window is titled 'server.py' and shows the server responding to these requests. The client output includes:

```
q ~ /Documents/ASG3/Part2 > p client.py
Enter port number: 7777
Enter domain name: google.com
1 google.com 192.165.1.1
Continue? (y/n) y
Enter domain name: github.com
2 github.com 140.82.121.4
Continue? (y/n) y
Enter domain name: x.com
3 x.com Not found!
Continue? (y/n) n
q ~ /Doc/A/Part2 >
```

The server output includes:

```
q ~ /Documents/ASG3/Part2 > p server.py
15:35:34
Server starting...
Enter port number: 7777
('127.0.0.1', 52581) wants to fetch data!
('127.0.0.1', 52581) wants to fetch data!
('127.0.0.1', 52581) wants to fetch data!
('127.0.0.1', 46705) wants to fetch data!
('127.0.0.1', 46705) wants to fetch data!
('127.0.0.1', 46705) wants to fetch data!
```

Workflow

Part 3: Making Echo Client/Server “Protocol Independent” [15 Points].

- **Run the server script first:** python Server.py <localhost/ip6-localhost> <port>
 - **Run the client script:** python Client.py <localhost/ip6-localhost> <port> <packet_count> <interval> <packet_size>

Server.py

```
1 import socket
2 import sys
3
4 if (len(sys.argv) != 3):
5     print("To use this file:\n python3 part3_server.py <localhost/ip6-localhost> <port>")
6     exit(0)
7
8 # Host received from the user
9 host = sys.argv[1]
10 # Port number received from the user
11 port = sys.argv[2]
12 bufsize = 1024
13
14 # Get the information about the host using getaddrinfo
15 addr_info = socket.getaddrinfo(host,port)
16 # Created a UDP socket using the family info received from getaddrinfo
17 UDPServerSocket = socket.socket(family=addr_info[0][0], type=socket.SOCK_DGRAM)
18 # Binding the server to the address received from the getaddrinfo
19 UDPServerSocket.bind(addr_info[0][4])
20
21 if (host == "ip6-localhost"):
22     print("UDP ipv6 server is running!")
23 else:
24     print("UDP ipv4 server is running")
25
26 while (True):
27     # Message and Address(client's) received from the client
28     message,address = UDPServerSocket.recvfrom(bufsize)
29     # Send the message back to the client
30     UDPServerSocket.sendto(message,address)
31
```

Client.py

```
 1 import socket
 2 import sys
 3 import datetime
 4 import time
 5
 6 if (len(sys.argv) != 6):
 7     print("To use this file, run the file in the following manner:")
 8     print("python3 part3_client.py <localhost/ip6-localhost> <port> <packet_count> <interval> <packet_size>")
 9     exit()
10 # Hostname received from the user (localhost or ip6-localhost)
11 host = sys.argv[1]
12 # Packet count from the user
13 packet_count = int(sys.argv[3])
14 # Interval between the packets transmission
15 interval = int(sys.argv[4])
16 # Packet Size from the user
17 packet_size = int(sys.argv[5])
18
19 # port number of the client (hardcoded)
20 port = 8000
21 timeout = 1
22
23 # If packet size is greater than 1024 then
24 if (packet_size > 1024):
25     print("Packet Size should be lower than 1024")
26     exit()
27
28 # UDP client Socket Initialised
29 UDPClientSocket = socket.socket(family=socket.getaddrinfo(host, port)[0][0], type=socket.SOCK_DGRAM)
30
31 loss_count = 0
32 received_count = 0
33 total_packets = packet_count
34 total_time = 0
35 []
K NORMAL  ⌂Client.py ⌂ 0 △ 0 Ln 35, Col 0  UTF-8 {} python □ Part3
```

```
37 while(packet_count):
38
39     # Message To send
40     message = "z"*packet_size
41     packet_received = False # Flag to check receiving of the file
42     UDPClientSocket.sendto(bytes(message,'ascii'),(host,int(sys.argv[2])))
43     start_time = datetime.datetime.now() # Start time flag
44     while ((datetime.datetime.now() - start_time).total_seconds() < timeout):
45         if (UDPClientSocket.recvfrom(packet_size)):
46             end_time = datetime.datetime.now() # End time flag
47             packet_received = True
48             print(f"RTT => {(end_time - start_time).total_seconds()}")
49             print("=====")
50             total_time += (end_time - start_time).total_seconds()
51             received_count += 1
52             break
53     if (packet_received == False):
54         print("Packet Loss")
55         print("=====")
56         loss_count += 1
57     packet_count = packet_count - 1
58     time.sleep(interval)
59 print("===== Overall Details =====")
60 print(f"Packet Received: {received_count}")
61 print(f"Packet Sent: {total_packets}")
62 print(f"Loss: {((loss_count/total_packets)*100)%}")
63 print(f"Average RTT: {total_time/total_packets}")

K NORMAL  ⌂Client.py ⌂ 0 △ 0 Ln 37, Col 6  UTF-8 {} python □ Part3
```

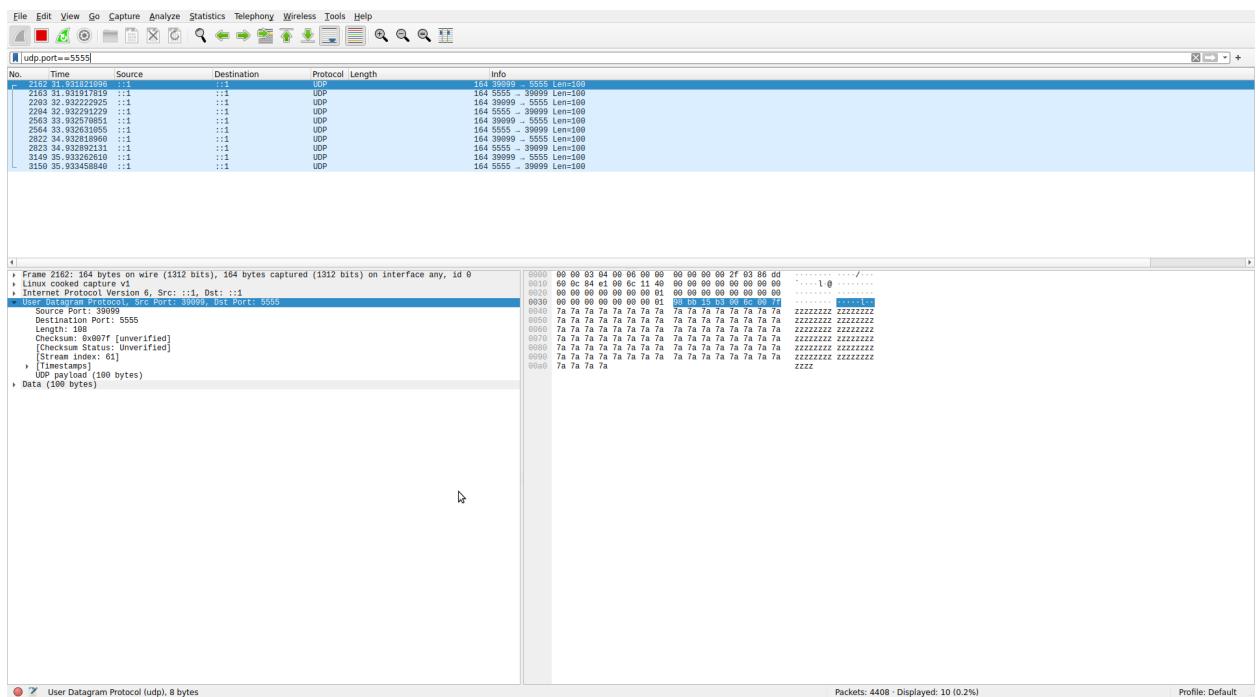
Ipv6-localhost port:- 5555

```

1 part3
Q ~ /Documents/ASG3/part3 🐍 main !302 ?4 > python Server.py ip6-localhost 5555
UDP ipv6 server is running!
O 19:45:27

Q ~ /Documents/ASG3/part3 🐍 main !302 ?4 > python Client.py ip6-localhost 5555 5 1 100
RTT => 5.6e-05
=====
RTT => 3.9e-05
=====
RTT => 3.4e-05
=====
RTT => 3.7e-05
=====
RTT => 3.7e-05
=====
===== Overall Details =====
Packet Received: 5
Packet Sent: 5
Loss: 0.0%
Average RTT: 4.06e-05
Q ~ /Documents/ASG3/part3 🐍 main !302 ?4 >
O 19:44:56 [0/3]
Q ~ /Documents/ASG3/part3 🐍 main !302 ?4 > 5s O 19:45:58

```



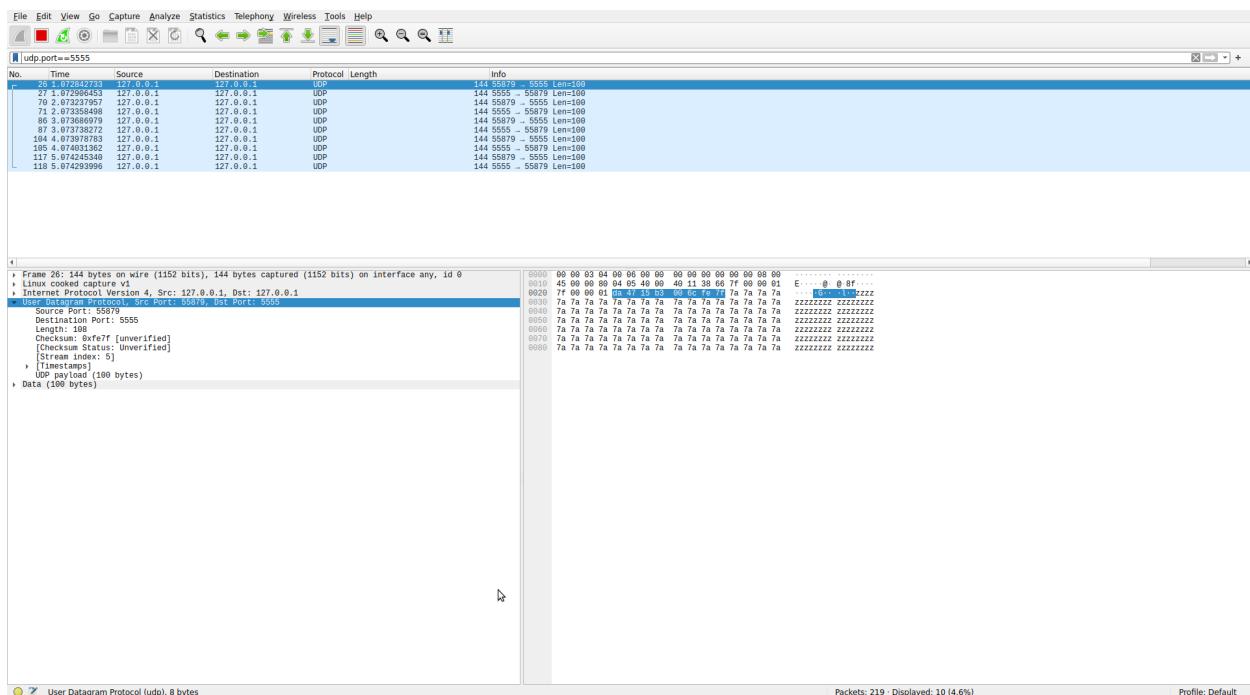
Wireshark Output

localhost

port:- 5555

```
1 part3
Q ~ /Documents/ASG3/part3 🐍 main !302 ?4 > python Server.py localhost 5555
UPD ipv4 server is running

Q ~ /Documents/ASG3/part3 🐍 main !302 ?4 > python Client.py localhost 5555 5 1 100
RTT => 0.000333
=====
RTT => 7.3e-05
=====
RTT => 0.000103
=====
RTT => 0.000119
=====
RTT => 7.5e-05
=====
===== Overall Details =====
Packet Received: 5
Packet Sent: 5
Loss: 0.0%
Average RTT: 0.0001406
Q ~ /Documents/ASG3/part3 🐍 main !302 ?4 >
```



Wireshark Output

The server receives a certain amount of packets from the client and returns each packet to the client via echo. The client measures every packet's round-trip time.

Part 4: Create your own client server application [15 Points]

Add any one feature to Client/Server and demonstrate them. In the report, you must describe the new features with their benefit. Significance of the feature will impact the marks given.

- **Run the server script first:** python Server.py <port>
- **Run the client script:** python Client.py <port>

Client.py

```
1 import socket
2 #import select
3 import sys
4
5 server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
6 if len(sys.argv) != 2:
7     print ("Correct usage: script, port number")
8     exit()
9 IPA = socket.gethostname()
10
11 Port = int(sys.argv[1])
12 server.connect((IPA, Port))
13
14 while True:
15
16     # maintains a list of possible input streams
17     sockets_list = [sys.stdin, server]
18     read_sockets,write_socket, error_socket = select.select(sockets_list,[],[])
19
20     for socks in read_sockets:
21         if socks == server:
22             message = socks.recv(2048)
23             print(f"Received Message: {message}")
24         else:
25             message = sys.stdin.readline()
26             server.send(bytes(message,'ascii'))
27             sys.stdout.write("<User>")
28             sys.stdout.write(message)
29             sys.stdout.flush()
30
31
```

After running both scripts, the client is connected to the server, and we can connect multiple clients to the server. Every client can see the messages sent from another client, and the server keeps track of the messages with the message's timestamp.

Server.py

```
1 import socket
2 import sys
3 from _thread import *
4 import time
5
6 server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
7 server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
8
9 if len(sys.argv) != 2:
10     print("Correct usage: script, port number")
11     exit()
12 IPA = socket.gethostname()
13
14 Port = int(sys.argv[1])
15 server.bind((IPA, Port)) # We are setting up the ip_address and port number for the server
16 server.listen(100) # We can set[]the number of users to 100
17 all_clients = []
18
19
20 # New threads will be created for different users for this function
21 def clientthread(conn, addr):
22     conn.send(b"Hola amigos!")
23     while True:
24         try:
25             message = conn.recv(2048)
26             if message:
27                 # Get the current time
28                 current_time = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
29
30                 # Print the message along with the timestamp
31                 print(f"({addr[0]}) [{current_time}]: {message.decode()}")
32
33                 # Calls broadcast function to send message to all
34                 message_to_send = f"({addr[0]}) [{current_time}]: {message.decode()}"
35                 broadcast(message_to_send, conn)
36             else:
37                 remove(conn)
38         except:
39             continue
40
```

```

42 # broadcast the message to all other users
43 def broadcast(message, connection):
44     for clients in all_clients:
45         if clients != connection:
46             try:
47                 clients.send(bytes(message, 'ascii'))
48             except:
49                 clients.close()
50                 # if the link is broken, we remove the client
51                 remove(clients)
52
53
54 def remove(connection):
55     if connection in all_clients:
56         all_clients.remove(connection)
57
58 while True:
59     conn, addr = server.accept()
60     all_clients.append(conn)
61     # prints the address of the user that just connected
62     print(addr[0] + " connected")
63     # creates an individual thread for every user
64     # that connects
65     start_new_thread(clientthread, (conn, addr))

```

Viewing messages from every user connected to the server in one location with this server/client application is possible.

Server view with timestamp

```

Part4
~/Doc/A/Part4 > python Server.py 3000
127.0.0.1 connected
(127.0.0.1) [2023-11-20 19:52:10]: Hello
(127.0.0.1) [2023-11-20 19:52:15]: How are you?

127.0.0.1 connected
(127.0.0.1) [2023-11-20 19:52:35]: I am fine.

(127.0.0.1) [2023-11-20 19:52:51]: Tell me about yourself.

(127.0.0.1) [2023-11-20 19:53:08]: I am also doing good.

(127.0.0.1) [2023-11-20 19:53:26]: Ohh nice, Bye!

(127.0.0.1) [2023-11-20 19:53:30]: Bye!

```

```

python3 4
~/Doc/A/Part4 > python Client.py 3000
Received Message: b'Hello amigos!'
Hello
<User>Hello
How are you?
<User>How are you?
Received Message: b'(127.0.0.1) [2023-11-20 19:52:35]: I am fine.\n'
Received Message: b'(127.0.0.1) [2023-11-20 19:52:51]: Tell me about your self.\n'
I am also doing good:
<User>I am also doing good:
Received Message: b'(127.0.0.1) [2023-11-20 19:53:26]: Ohh nice, Bye!\n'
Bye!
<User>Bye!

python3 4
~/Doc/A/Part4 > python Client.py 3000
Received Message: b'Hello amigos!'
I am fine.
<User>I am fine.
Tell me about yourself.
<User>Tell me about yourself.
Received Message: b'(127.0.0.1) [2023-11-20 19:53:08]: I am also doing good:\n'
Ohh nice, Bye!
<User>Ohh nice, Bye!
Received Message: b'(127.0.0.1) [2023-11-20 19:53:30]: Bye!\n'

```

Ref:- <https://www.geeksforgeeks.org/simple-chat-room-using-python/>

https://youtu.be/Um_oIBPs48w?si=x4n0t2MxcD_BlvbG