

California State University, Fresno
Lyles College of Engineering: Department of Electrical
and Computer Engineering

Final Project Report

Experiment Title: ECE 146 Final Project – Chatting Program using Python

Course Title: ECE 146 – Computer Networks

Date Submitted: 05/06/2023

Instructor: Dr. Hovannes Kulhandjian

Prepared By:	Sections Written:
Aryan Singh	All Sections

Table of Contents

<i>Statement Of Objectives</i>	3
<i>Theoretical Background</i>	3
<i>Implementation</i>	3
Software Specifications	3
Experimental Procedure	3
Server.....	3
Client	4
<i>Results and Discussion</i>	5
Figure 1: Server Code (1).....	5
Figure 2: Server Code (2).....	6
Figure 3: Client Code (1).....	7
Figure 4: Client Code (2).....	8
Figure 5: Client Code (3).....	9
Figure 5: Chatbox for Client 1	10
Figure 6: Chatbox for Client 2	10
Figure 7: Server command prompt	11
Figure 8: Command prompt for Clients.....	12
Figure 9: Client 2 chatbox when Client 1 has left	12
<i>Conclusion</i>	13
<i>Appendix.....</i>	14
Server Code	14
Client Code	15
<i>References.....</i>	18

Statement Of Objectives

The objective of this project is to implement an instant messaging system or a chatting program using Python for a single Server and Multiple client connection application. An implementation of a Graphical User Interface is also implemented in order to make the messaging system more user friendly.

Theoretical Background

For this chatting program the coding language Python is used to construct the Server and Client applications and a Graphical User Interface.

A Client is a program which is connected to the Server through a network connection at a specific IP Address and port number. When the connection with the Server is open, then the user can interact with the Client program using the Graphical User Interface and perform various interactions with other Client programs over the network connection with the IM Server.

The Server is the program that has all the Clients connected to it using a network connection, where each client can connect and disconnect whenever they want. The Server does not use a GUI since it has no direct interaction to the users themselves.

In this chatting system the Clients have a GUI (Graphical User Interface) using which the users can communicate with their Client Computer and the Clients are connected to a Server through which they can communicate with each other. The Clients are not sending direct message to each other, rather they will be sending their messages to the Server and the Server in turn will be broadcasting their messages to the destination Clients.

Implementation

Software Specifications

- IDLE Python Integrated Development and Learning Environment
- Tkinter Graphical User Interface toolkit
- (Write down PC Specs)

Experimental Procedure

For the implementation of the Server program and the Client program using the Graphical User Interface we use the IDLE Python Programmer.

Server

We begin with the implementation of the Server by importing the socket and threading libraries and then we declare the local Host and Port connections which will be used to connect the Server to the Clients. Then we use the Internet socket connection where the

TCP protocol will be used. Then the server is bound to the local host and port constants that we declared.

After making all the connections we use 3 main functions in Server which are used to do all of the work. The 3 functions used are broadcast, receive, and handle. The broadcast function is used to send one message to all the connected clients to show their connections to the Server. The receive function accepts new connections by listening to the new incoming Client programs, so this function is used to receive any new connections coming to the Server. Finally, the handle program is used to manage individual connections to the client. The receive function is used to accept a new connection from a client and then the handle function handles this connection after receiving it.

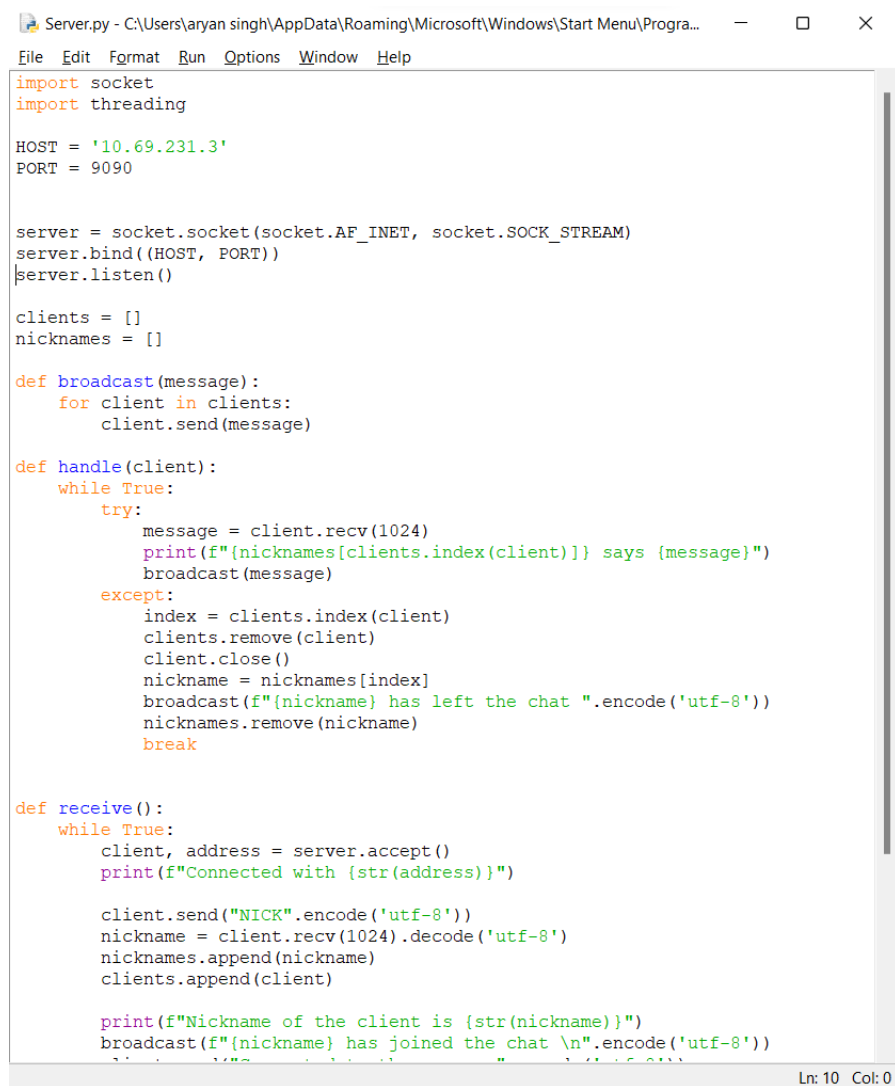
Client

For programming the Client, we import the socket and threading libraries like we did for the Server, but we also import the tkinter libraries to create a GUI as well. We then declare the same port and local host constant which will be used to connect the client to the server. A class of client is then created in which we define a constructor where the host and port are passed, and the socket is also defined. The client is then connected to the server using the sock connect instead of binding which was used for the Server. The message that will be sent to the server is then asked from the user using the tkinter functions. Once we have received the message from the user then the GUI implementation is done. For that purpose, the GUI is created as its own separate function and another function to receive and handle the server connection link. The GUI is created in the gui_loop function, using the tkinter library, where the background color is set to gray, and the chat label is also created with the gray background color and the font set to Arial and font size set as 12. For appearance some padding is added after this to make the textbox look more presentable. Similarly, the text area is also made using the scrolledtext library from tkinter so that we are able to scroll through the text messages. After making the whole chat window we create the button using which the clients can send the messages to the server.

This will complete the Server and Client program implementations and then when we run these codes using command prompt or the IDLE programmer, then we will be able to see the results.

Results and Discussion

1. For the server code, the host and port connections are made in Figure 1, along with socket implementation and the 3 functions that were mentioned in the procedure. The 3 functions broadcast, handle and receive can be observed in figure 2. The broadcast function sends the message to all the clients, the handle function displays in the server window the message that the client is typing to the other client. In case the client has left the chat, then it is the job of this function to display in the chat box that the client has left. Finally, the receive function shows the connection that has been made with a client, and then decodes the message sent by the client in order to send the message to the other client.



```

Server.py - C:\Users\aryan singh\AppData\Roaming\Microsoft\Windows\Start Menu\Progra...
File Edit Format Run Options Window Help

import socket
import threading

HOST = '10.69.231.3'
PORT = 9090

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind((HOST, PORT))
server.listen()

clients = []
nicknames = []

def broadcast(message):
    for client in clients:
        client.send(message)

def handle(client):
    while True:
        try:
            message = client.recv(1024)
            print(f"{nicknames[clients.index(client)]} says {message}")
            broadcast(message)
        except:
            index = clients.index(client)
            clients.remove(client)
            client.close()
            nickname = nicknames[index]
            broadcast(f"{nickname} has left the chat ".encode('utf-8'))
            nicknames.remove(nickname)
            break

def receive():
    while True:
        client, address = server.accept()
        print(f"Connected with {str(address)}")

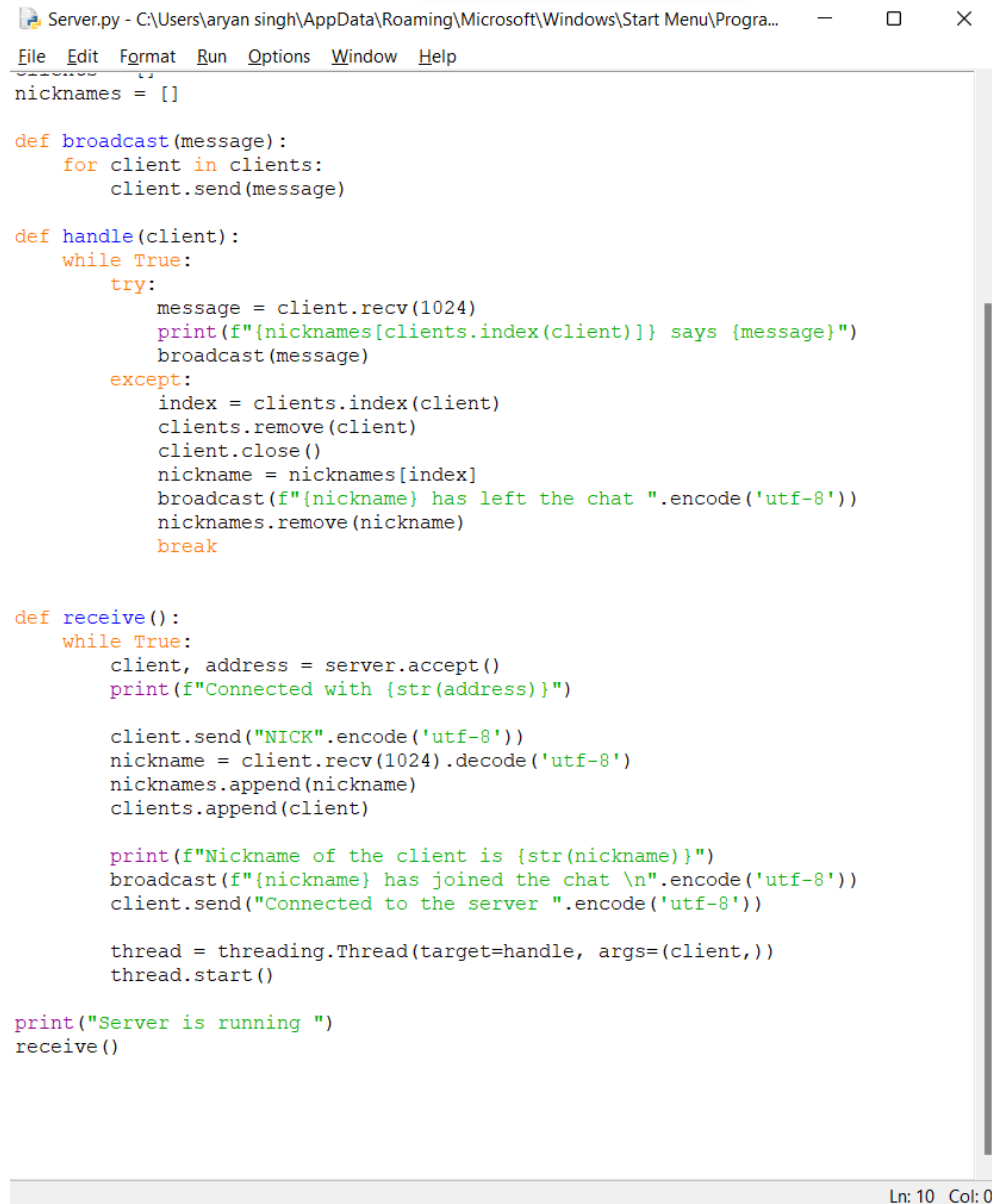
        client.send("NICK".encode('utf-8'))
        nickname = client.recv(1024).decode('utf-8')
        nicknames.append(nickname)
        clients.append(client)

        print(f"Nickname of the client is {str(nickname)}")
        broadcast(f"{nickname} has joined the chat \n".encode('utf-8'))

```

Ln: 10 Col: 0

Figure 1: Server Code (1)



```

Server.py - C:\Users\aryan singh\AppData\Roaming\Microsoft\Windows\Start Menu\Progra...
File Edit Format Run Options Window Help
nicknames = []

def broadcast(message):
    for client in clients:
        client.send(message)

def handle(client):
    while True:
        try:
            message = client.recv(1024)
            print(f"{nicknames[clients.index(client)]} says {message}")
            broadcast(message)
        except:
            index = clients.index(client)
            clients.remove(client)
            client.close()
            nickname = nicknames[index]
            broadcast(f"{nickname} has left the chat ".encode('utf-8'))
            nicknames.remove(nickname)
            break

def receive():
    while True:
        client, address = server.accept()
        print(f"Connected with {str(address)}")

        client.send("NICK".encode('utf-8'))
        nickname = client.recv(1024).decode('utf-8')
        nicknames.append(nickname)
        clients.append(client)

        print(f"Nickname of the client is {str(nickname)}")
        broadcast(f"{nickname} has joined the chat \n".encode('utf-8'))
        client.send("Connected to the server ".encode('utf-8'))

        thread = threading.Thread(target=handle, args=(client,))
        thread.start()

print("Server is running ")
receive()

```

Ln: 10 Col: 0

Figure 2: Server Code (2)

2. For the client we use the socket and threading libraries, however since we need to apply a Graphical User Interface, we need to include the tkinter libraries too, which can be seen in Figure 3. The client class is also defined in which there are multiple functions to receive the messages and write them on the chat box. The Graphical user Interface is also implemented in this program where the background color, font type and size and padding configurations are set. The Interface includes the configuration of the aforementioned configurations for the chat label, text area, message label and the message input area as well. Apart from these we also need a send button which will send the message to the client and the server (Figure 4).



```

Client.py - C:\Users\aryan singh\AppData\Roaming\Microsoft\Windows\Start Menu\Progra...
File Edit Format Run Options Window Help

import socket
import threading
import tkinter
import tkinter.scrolledtext
from tkinter import simpledialog

HOST = '10.69.231.3'
PORT = 9090

class Client:

    def write(self):
        message = f"{self.nickname}: {self.input_area.get('1.0', 'end')}}"
        self.sock.send(message.encode('utf-8'))
        self.input_area.delete('1.0', 'end')

    def stop(self):
        self.running = False
        self.win.destroy()
        self.sock.close()
        exit(0)

    def receive(self):
        while self.running:
            try:
                message = self.sock.recv(1024).decode('utf-8')
                if message == 'NICK':
                    self.sock.send(self.nickname.encode('utf-8'))
                else:
                    if self.gui_done:
                        self.text_area.config(state='normal')
                        self.text_area.insert('end', message)
                        self.text_area.yview('end')
                        self.text_area.config(state='disabled')
            except ConnectionAbortedError:
                break
            except:
                print("Error")
                self.sock.close()
                break

    def __init__(self, host, port):

```

Ln: 23 Col: 0

Figure 3: Client Code (1)

```

Client.py - C:\Users\aryan singh\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Python 3.11\Python 3.11.3
File Edit Format Run Options Window Help
    self.sock.close()
    break

def __init__(self, host, port):

    self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.sock.connect((host, port))

    msg = tkinter.Tk()
    msg.withdraw()

    self.nickname = simpledialog.askstring("Nickname", "Please choose a Nickname", parent=msg)

    self.gui_done = False
    self.running = True

    gui_thread = threading.Thread(target=self.gui_loop)
    receive_thread = threading.Thread(target=self.receive)

    gui_thread.start()
    receive_thread.start()

def gui_loop(self):
    self.win = tkinter.Tk()
    self.win.configure(bg="black")

    self.chat_label = tkinter.Label(self.win, text="chat: ", bg="black")
    self.chat_label.config(font=("Arial", 12))
    self.chat_label.pack(padx=20, pady=5)

    self.text_area = tkinter.scrolledtext.ScrolledText(self.win)
    self.text_area.pack(padx=20, pady=5)
    self.text_area.config(state='disabled')

    self.msg_label = tkinter.Label(self.win, text="Message: ", bg="black")
    self.msg_label.config(font=("Arial", 12))
    self.msg_label.pack(padx=20, pady=5)

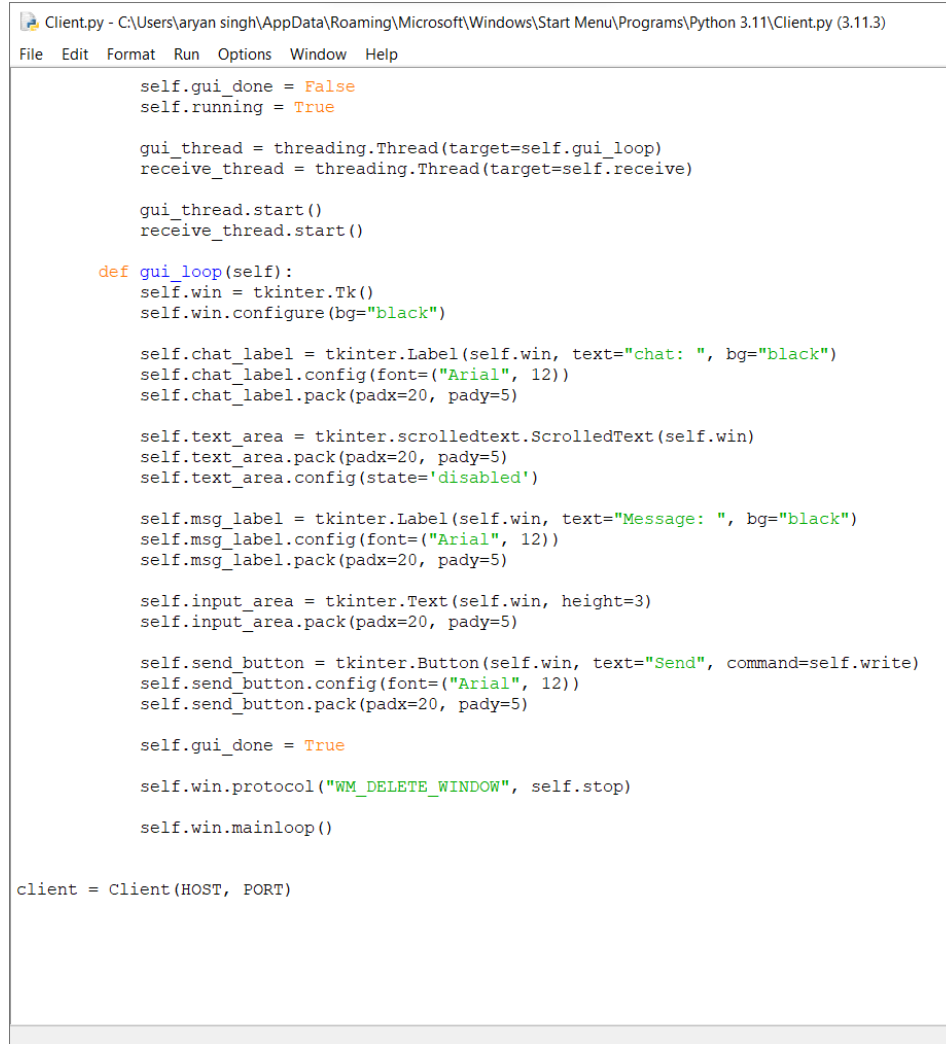
    self.input_area = tkinter.Text(self.win, height=3)
    self.input_area.pack(padx=20, pady=5)

    self.send_button = tkinter.Button(self.win, text="Send", command=self.write)
    self.send_button.config(font=("Arial", 12))
    self.send_button.pack(padx=20, pady=5)

```

Ln: 101

Figure 4: Client Code (2)



```

Client.py - C:\Users\aryan singh\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Python 3.11\Client.py (3.11.3)
File Edit Format Run Options Window Help

    self.gui_done = False
    self.running = True

    gui_thread = threading.Thread(target=self.gui_loop)
    receive_thread = threading.Thread(target=self.receive)

    gui_thread.start()
    receive_thread.start()

def gui_loop(self):
    self.win = tkinter.Tk()
    self.win.configure(bg="black")

    self.chat_label = tkinter.Label(self.win, text="chat: ", bg="black")
    self.chat_label.config(font=("Arial", 12))
    self.chat_label.pack(padx=20, pady=5)

    self.text_area = tkinter.scrolledtext.ScrolledText(self.win)
    self.text_area.pack(padx=20, pady=5)
    self.text_area.config(state='disabled')

    self.msg_label = tkinter.Label(self.win, text="Message: ", bg="black")
    self.msg_label.config(font=("Arial", 12))
    self.msg_label.pack(padx=20, pady=5)

    self.input_area = tkinter.Text(self.win, height=3)
    self.input_area.pack(padx=20, pady=5)

    self.send_button = tkinter.Button(self.win, text="Send", command=self.write)
    self.send_button.config(font=("Arial", 12))
    self.send_button.pack(padx=20, pady=5)

    self.gui_done = True

    self.win.protocol("WM_DELETE_WINDOW", self.stop)

    self.win.mainloop()

client = Client(HOST, PORT)

```

Figure 5: Client Code (3)

- After we run the code using command prompt, we can see that the chat box asks for the name of the Client before opening the chat box. The same is done before any client connects to the Server. After the name has been entered the chat box opens and the clients can send messages to each other via the Server (Figure 5 and 6). Once a client connects the server shows the network host IP Address and the port number to confirm that the TCP network connection with the client has been made (Figure 7). All the messages that the clients share through the server are monitored by the server as we can see in Figure 7. Finally, whenever a client leaves the chat then the message stating that the client has left the chat and is no longer available to communicate is shown to both the server and the client (Figure 9).

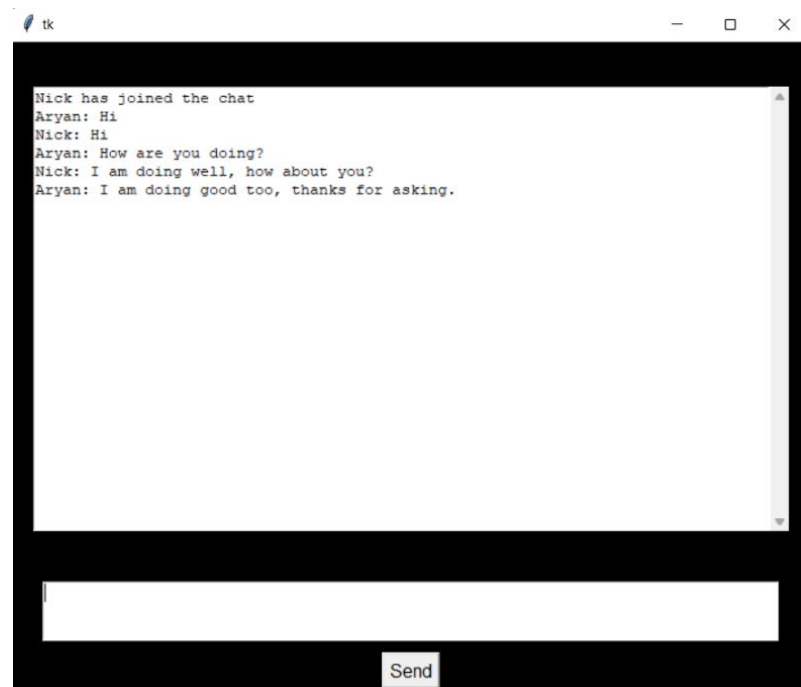
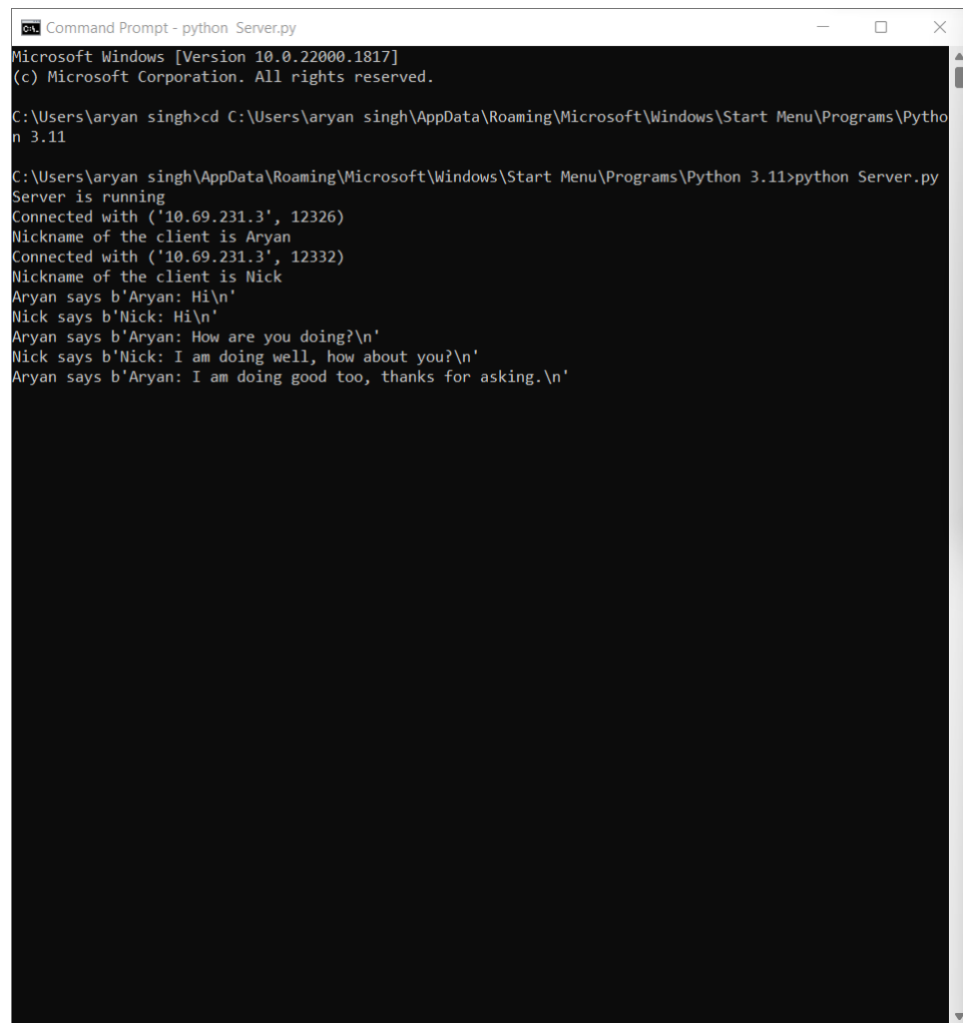


Figure 5: Chatbox for Client 1



Figure 6: Chatbox for Client 2

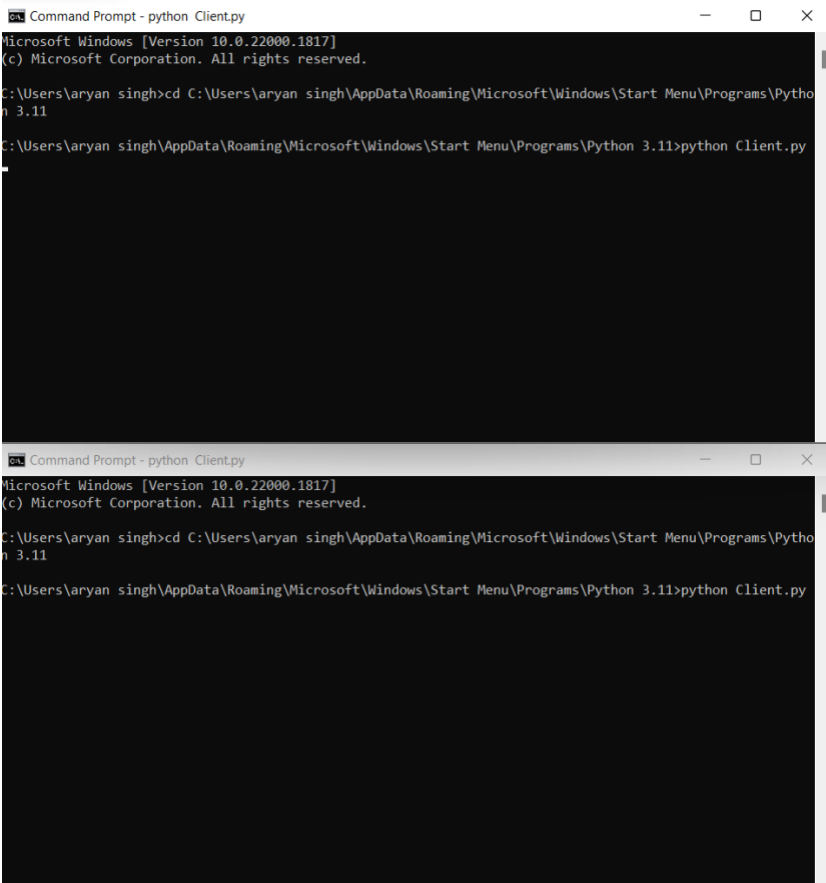


```
Command Prompt - python Server.py
Microsoft Windows [Version 10.0.22000.1817]
(c) Microsoft Corporation. All rights reserved.

C:\Users\aryan singh>cd C:\Users\aryan singh\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Python
n 3.11

C:\Users\aryan singh\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Python 3.11>python Server.py
Server is running
Connected with ('10.69.231.3', 12326)
Nickname of the client is Aryan
Connected with ('10.69.231.3', 12332)
Nickname of the client is Nick
Aryan says b'Aryan: Hi\n'
Nick says b'Nick: Hi\n'
Aryan says b'Aryan: How are you doing?\n'
Nick says b'Nick: I am doing well, how about you?\n'
Aryan says b'Aryan: I am doing good too, thanks for asking.\n'
```

Figure 7: Server command prompt



The image shows two identical screenshots of a Windows Command Prompt window. The title bar reads "Command Prompt - python Client.py". The window content shows the following text:

```
Microsoft Windows [Version 10.0.22000.1817]
(c) Microsoft Corporation. All rights reserved.

C:\Users\aryan singh>cd C:\Users\aryan singh\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Python
Python 3.11

C:\Users\aryan singh\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Python 3.11>python Client.py
```

Figure 8: Command prompt for Clients

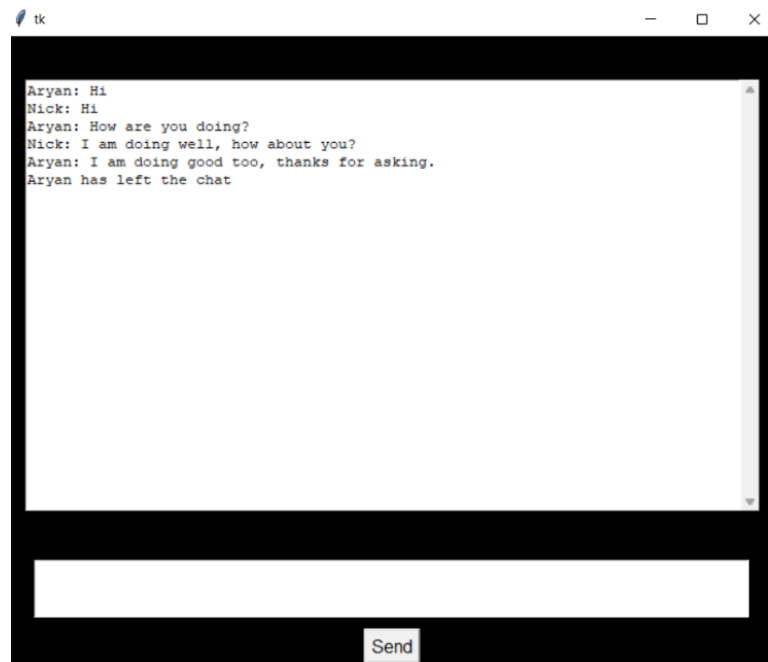


Figure 9: Client 2 chatbox when Client 1 has left

Conclusion

The main purpose of this project was to implement a messaging system between multiple clients using a TCP connection over a Server. This objective was fulfilled by implementing a client-side script in Python using the concepts of socket programming and threading. Through this project we were able to understand the use of socket programming and threading in order to create a chatting application. We used the programming language python to create a server and client module and implemented a Graphical User Interface for more optimal interaction of the user with the client computer. Another benefit of this project was that we got a basic understanding of Python and how to create modules that can be connected to each other and can interact over a TCP connection.

Appendix

Server Code

```
import socket
import threading

HOST = '10.69.231.3'
PORT = 9090

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind((HOST, PORT))
server.listen()

clients = []
nicknames = []

def broadcast(message):
    for client in clients:
        client.send(message)

def handle(client):
    while True:
        try:
            message = client.recv(1024)
            print(f'{nicknames[clients.index(client)]} says {message}')
            broadcast(message)
        except:
            index = clients.index(client)
            clients.remove(client)
            client.close()
            nickname = nicknames[index]
            broadcast(f'{nickname} has left the chat '.encode('utf-8'))
            nicknames.remove(nickname)
            break

def receive():
    while True:
```

```

client, address = server.accept()
print(f'Connected with {str(address)}')

client.send("NICK".encode('utf-8'))
nickname = client.recv(1024).decode('utf-8')
nicknames.append(nickname)
clients.append(client)

print(f'Nickname of the client is {str(nickname)}')
broadcast(f'{nickname} has joined the chat \n'.encode('utf-8'))
client.send("Connected to the server ".encode('utf-8'))

thread = threading.Thread(target=handle, args=(client,))
thread.start()

print("Server is running ")
receive()

```

Client Code

```

import socket
import threading
import tkinter
import tkinter.scrolledtext
from tkinter import simpledialog

HOST = '10.69.231.3'
PORT = 9090

class Client:

    def write(self):
        message = f'{self.nickname}: {self.input_area.get('1.0', 'end')}'
        self.sock.send(message.encode('utf-8'))
        self.input_area.delete('1.0', 'end')

    def stop(self):

```

```

self.running = False
self.win.destroy()
self.sock.close()
exit(0)

def receive(self):
    while self.running:
        try:
            message = self.sock.recv(1024).decode('utf-8')
            if message == 'NICK':
                self.sock.send(self.nickname.encode('utf-8'))
            else:
                if self.gui_done:
                    self.text_area.config(state='normal')
                    self.text_area.insert('end', message)
                    self.text_area.yview('end')
                    self.text_area.config(state='disabled')
        except ConnectionAbortedError:
            break
        except:
            print("Error")
            self.sock.close()
            break

def __init__(self, host, port):

    self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.sock.connect((host, port))

    msg = tkinter.Tk()
    msg.withdraw()

    self.nickname = simpledialog.askstring("Nickname", "Please choose a Nickname",
parent=msg)

    self.gui_done = False
    self.running = True

```



```
gui_thread = threading.Thread(target=self.gui_loop)
receive_thread = threading.Thread(target=self.receive)
```

```
gui_thread.start()
receive_thread.start()
```

```
def gui_loop(self):
```

```
    self.win = tkinter.Tk()
    self.win.configure(bg="black")
```

```
    self.chat_label = tkinter.Label(self.win, text="chat: ", bg="black")
    self.chat_label.config(font=("Arial", 12))
    self.chat_label.pack(padx=20, pady=5)
```

```
    self.text_area = tkinter.scrolledtext.ScrolledText(self.win)
    self.text_area.pack(padx=20, pady=5)
    self.text_area.config(state='disabled')
```

```
    self.msg_label = tkinter.Label(self.win, text="Message: ", bg="black")
    self.msg_label.config(font=("Arial", 12))
    self.msg_label.pack(padx=20, pady=5)
```

```
    self.input_area = tkinter.Text(self.win, height=3)
    self.input_area.pack(padx=20, pady=5)
```

```
    self.send_button = tkinter.Button(self.win, text="Send", command=self.write)
    self.send_button.config(font=("Arial", 12))
    self.send_button.pack(padx=20, pady=5)
```

```
    self.gui_done = True
```

```
    self.win.protocol("WM_DELETE_WINDOW", self.stop)
```

```
    self.win.mainloop()
```

```
client = Client(HOST, PORT)
```

References

- [1] “Project 1: Chatting Program using Python/C++”, ECE 146 Project 1 Spring 2023.
Lyles College of Engineering, California State University, Fresno, CA, Spring 2023