Server client based Gui application

A COURSE PROJECT REPORT

By

SHIVANI JHA (RA2011003011256)
THATIPARTHY JAVALI (RA2011003011222)
NADELLA HAKESH (RA2011003011240)
YANGALA JAYASAIKUMAR(RA2011003011214)

Under the guidance of

<Mrs. Lavanya R>

In partial fulfilment for the Course

of

18CSC302J - COMPUTER NETWORKS

in <COMPUTER SCIENCE AND ENGINEERING t>

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

Kattankulathur, Chenpalpattu District

NOVEMBER 2022

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this mini project report "**Server client based gui application** " is the bonafide work of **Shivani JHA**and **T.Javali and N.hakesh and Y.Jayasaikumar** who carried out the project work under my supervision.

SIGNATURE

<R.Lavanya>
<Designation>
<CSE>
SRM Institute of Science and Technology

ABSTRACT

As we know, this chat system started early mid-1980 and was very popular at that time. Chat application refers to communication between two entities i.e. (sender) and (receiver). If we talk about security and internet penetration is increasing day by day. We focused on this and in this Application, we make a server and several client connection points in which the clients speak with the server utilizing an attachment module. These attachments are inside endpoints for sending and getting information. A solitary organization will have two attachments. This program is executed utilizing TCP attachment [TCP alludes to association oriented]. This attachment will be associated with some port in the machine or local host. On account of the client, we will interface an attachment to that server, on the very port that the server-side code is utilizing

ACKNOWLEDGEMENT

We express our heartfelt thanks to our honorable **Vice Chancellor Dr. C. MUTHAMIZHCHELVAN**, for being the beacon in all our endeavors.

We would like to express my warmth of gratitude to our **Registrar Dr. S. Ponnusamy,** for his encouragement

We express our profound gratitude to our **Dean (College of Engineering and Technology) Dr. T. V.Gopal,** for bringing out novelty in all executions.

We would like to express my heartfelt thanks to Chairperson, School of Computing **Dr. Revathi Venkataraman,** for imparting confidence to complete my course project

We wish to express my sincere thanks to **Course Audit Professor Dr.Annapurani Panaiyappan, Professor and Head, Department of Networking and Communications** and **Course Coordinators** for their constant encouragement and support.

We are highly thankful to our my Course project Faculty **<Faculty Name> , <Designation> , <Department>,** for his/her assistance, timely suggestion and guidance throughout the duration of this course project.

We extend my gratitude to our **HoD <Name> <Designation>, <Department>** and my Departmental colleagues for their Support.

Finally, we thank our parents and friends near and dear ones who directly and indirectly contributed to the successful completion of our project. Above all, I thank the almighty for showering his blessings on me to complete my Course project.

TABLE OF CONTENTS

CHAPTERS **CONTENTS**

INTRODUCTION

A multi-Client talk application runs utilizing an attachment programming, in this, we make a server-client application where different clients can speak with one another separately or it can speak with every one of the clients. By utilizing this, we can make a cut-off client application, such as clinical point of interaction, client report and administration association, and different applications that can run by utilizing this programming.As talk applications are vital in everyday life, they require a web to visit one another and assume a significant part of the correspondence. This application is utilized generally in every one of the fields, for example, IT organizations, schools, universities, individual texts, etc. As the talk application needs a web to visit it makes a significant disadvantage since there are numerous edges regions and numerous towns which don't have proper internet connections where they can convey even locally. As this application doesn't expect the web to talk when individuals are close by, it gives neighborhoods known as servers utilizing the strategy known as attachments where individuals known as clients can speak with one another. Here, the clients associated with the nearby host are known as a server and can speak with any remaining clients associated through the neighborhood.

PROBLEM STATEMENT

As the talk applications which are being used require a web to impart even the distance is short which make a significant downside in the edges region where the web association is poor, and the schools where we can't be given to the huge number of understudies, organizations which make the data more secret can't be messaged through the web. It needs an application that is expected to be introduced by each client who necessities to talk makes it troublesome. The greater part of the talk application doesn't give start to finish encryption which unveils the texts more.

III. PROPOSED SYSTEM

This application runs utilizing an attachment programming which is a blend of an IP address and a port number. This module comprises inherent techniques that are expected for making attachments and assisting them with partnering with one another. In this, we make a server (meant to oversee network assets) and clients (client demand for administrations from a server).

This attachment and attachment Programming interface is utilized to send messages across different. These are given as IPC for example Between Cycle correspondence. The organization can be a coherent, neighborhood organization to the PC or one that is genuinely associated with the organization. We can show this application in the GUI [Graphical client interface] structure utilizing a module called Tkinter which gives an incredible arrangement and data on utilizing Tk from python and connections to other sources. This makes the proposed framework utilized better and causes the client to feel good while utilizing it. Attachments in python can be portrayed as it gives two degrees of organizational administration which can be gotten to. At the low level, we can get to the working framework by underlaying the essential attachment support which permits to execution of clients and servers for both association situated and connectionless conventions. Attachments additionally have a library to get to the more significant level modules in the, for example, FTP, HTTP, and so forth.

1. REQUIREMENTS

1.1 Requirement Analysis

Software Requirements :

From the given scenario, we draw the following requirements:

1)Internet

2)python 3.10

3)TCP/IP Network design with IP addressing

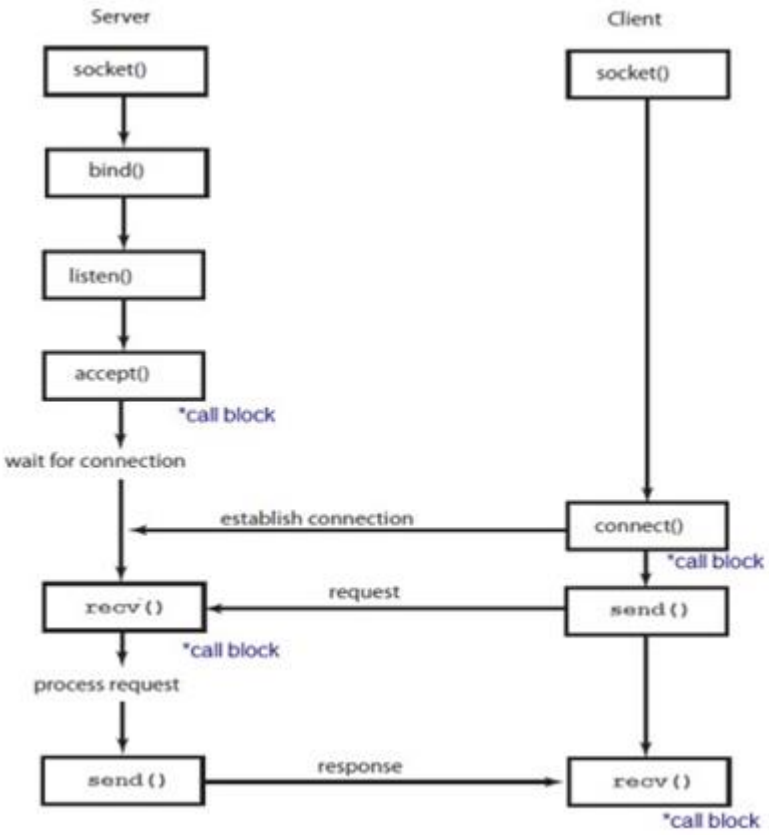
4)tkinter modules

5)socket modules

2. ARCHITECTURE AND DESIGN

2.1 Network Architecture

The network architecture is as follows:



1. **Create an attachment item and call the attachment work utilizing that article.**
2. **Bind the attachment object on the server-side which must utilize a residential area tie() prebuilt work.**
3. **Using tune-in () work we acknowledge the new associations and orchestrate them in the line design.**
4. **Using capacity interface() we clients associate with the neighborhood host of the server.**
5. **Accept the associations of clients utilizing the capacity acknowledge() from the server-side.**
6. **Accept different associations of clients by shutting the current client utilizing the capacity close().**

7. **Create an item from that read and compose the information on the server-side.**
8. **Repeat a similar interaction on the client-side.**
9. **Close the attachment of both server and client-side utilizing the capacity close().**

Modules_used_are:

a. Socket Module: These are the projects that sudden spike in demand for an organization utilizing a two-way correspondence endpoint connect among which structure an attachment.

b. Threading Module: Running as many strings is the same as running various projects simultaneously.

c. Tkinker Module: Tkinter is a standard library module utilized for making GUI in python. Tkinter gives a simple and quick approach to making a GUI application in python.

3. IMPLEMENTATION

The following are the executions of the undertaking multi-client visit application.

```
$ python3 server.py Zeyu
Listening at ('192.168.2.186', 1060)
q
Closing all connections...
Shutting down the server...
```

```
$ python3 client.py 192.168.2.186
Trying to connect to 192.168.2.186:1060...
Successfully connected to 192.168.2.186:1060

Your name:
```

codes:

server code:

```
#!/usr/bin/env python3

import threading
import socket
import argparse
import os
```

```
class Server(threading.Thread):
    """
    Supports management of server connections.

    Attributes:
        connections (list): A list of ServerSocket objects representing
the active connections.
        host (str): The IP address of the listening socket.
```

```
        port (int): The port number of the listening socket.
    """
    def __init__(self, host, port):
        super().__init__()
        self.connections = []
        self.host = host
        self.port = port

    def run(self):
        """
        Creates the listening socket. The listening socket will use the
SO_REUSEADDR option to
        allow binding to a previously-used socket address. This is a
small-scale application which
        only supports one waiting connection at a time.
        For each new connection, a ServerSocket thread is started to
facilitate communications with
        that particular client. All ServerSocket objects are stored in
the connections attribute.
        """
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        sock.bind((self.host, self.port))

        sock.listen(1)
        print('Listening at', sock.getsockname())

        while True:

            # Accept new connection
            sc, sockname = sock.accept()
            print('Accepted a new connection from {} to
{}'.format(sc.getpeername(), sc.getsockname()))

            # Create new thread
            server_socket = ServerSocket(sc, sockname, self)

            # Start new thread
            server_socket.start()

            # Add thread to active connections
            self.connections.append(server_socket)
            print('Ready to receive messages from', sc.getpeername())

    def broadcast(self, message, source):
```

```
        """
        Sends a message to all connected clients, except the source of
the message.

        Args:
            message (str): The message to broadcast.
            source (tuple): The socket address of the source client.
        """
        for connection in self.connections:

            # Send to all connected clients except the source client
            if connection.sockname != source:
                connection.send(message)

    def remove_connection(self, connection):
        """
        Removes a ServerSocket thread from the connections attribute.

        Args:
            connection (ServerSocket): The ServerSocket thread to
remove.
        """
        self.connections.remove(connection)


class ServerSocket(threading.Thread):
    """
    Supports communications with a connected client.

    Attributes:
        sc (socket.socket): The connected socket.
        sockname (tuple): The client socket address.
        server (Server): The parent thread.
    """
    def __init__(self, sc, sockname, server):
        super().__init__()
        self.sc = sc
        self.sockname = sockname
        self.server = server

    def run(self):
        """
        Receives data from the connected client and broadcasts the
message to all other clients.
        If the client has left the connection, closes the connected
```

```
socket and removes itself
        from the list of ServerSocket threads in the parent Server
thread.
        """
        while True:
            message = self.sc.recv(1024).decode('ascii')
            if message:
                print('{} says {!r}'.format(self.sockname, message))
                self.server.broadcast(message, self.sockname)
            else:
                # Client has closed the socket, exit the thread
                print('{} has closed the
connection'.format(self.sockname))
                self.sc.close()
                server.remove_connection(self)
                return

    def send(self, message):
        """
        Sends a message to the connected server.

        Args:
            message (str): The message to be sent.
        """
        self.sc.sendall(message.encode('ascii'))


def exit(server):
    """
    Allows the server administrator to shut down the server.
    Typing 'q' in the command line will close all active connections
and exit the application.
    """
    while True:
        ipt = input('')
        if ipt == 'q':
            print('Closing all connections...')
            for connection in server.connections:
                connection.sc.close()
            print('Shutting down the server...')
            os._exit(0)


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Chatroom Server')
```

```
    parser.add_argument('host', help='Interface the server listens at')
    parser.add_argument('-p', metavar='PORT', type=int, default=1060,
                        help='TCP port (default 1060)')
    args = parser.parse_args()

    # Create and start server thread
    server = Server(args.host, args.p)
    server.start()

    exit = threading.Thread(target = exit, args = (server,))
    exit.start()
```

client code:

```
#!/usr/bin/env python3

import threading
import socket
import argparse
import os
import sys
import tkinter as tk


class Send(threading.Thread):
    """
    Sending thread listens for user input from the command line.

    Attributes:
        sock (socket.socket): The connected socket object.
        name (str): The username provided by the user.
    """
    def __init__(self, sock, name):
        super().__init__()
        self.sock = sock
        self.name = name

    def run(self):
        """
        Listens for user input from the command line only and sends it
to the server.
        Typing 'QUIT' will close the connection and exit the
application.
```

```
        """
        while True:
            print('{}: '.format(self.name), end='')
            sys.stdout.flush()
            message = sys.stdin.readline()[:-1]

            # Type 'QUIT' to leave the chatroom
            if message == 'QUIT':
                self.sock.sendall('Server: {} has left the
chat.'.format(self.name).encode('ascii'))
                break

            # Send message to server for broadcasting
            else:
                self.sock.sendall('{}: {}'.format(self.name,
message).encode('ascii'))

        print('\nQuitting')
        self.sock.close()
        os._exit(0)


class Receive(threading.Thread):
    """
    Receiving thread listens for incoming messages from the server.

    Attributes:
        sock (socket.socket): The connected socket object.
        name (str): The username provided by the user.
        messages (tk.Listbox): The tk.Listbox object containing all
messages displayed on the GUI.
    """
    def __init__(self, sock, name):
        super().__init__()
        self.sock = sock
        self.name = name
        self.messages = None

    def run(self):
        """
        Receives data from the server and displays it in the GUI.
        Always listens for incoming data until either end has closed
the socket.
        """
        while True:
```

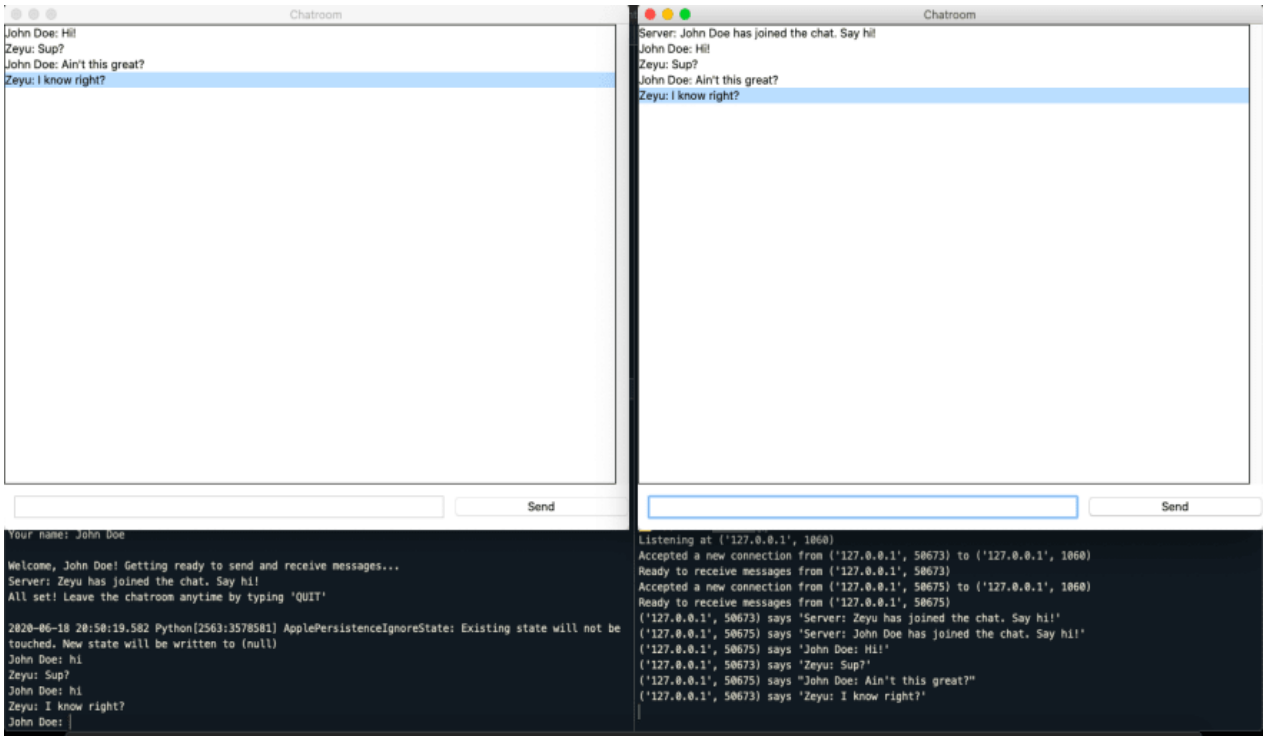
```
            message = self.sock.recv(1024).decode('ascii')

            if message:

                if self.messages:
                    self.messages.insert(tk.END, message)
                    print('hi')
                    print('\r{}\n{}: '.format(message, self.name), end
= '')

                else:
                    # Thread has started, but client GUI is not yet
ready
                    print('\r{}\n{}: '.format(message, self.name), end
= '')

            else:
                # Server has closed the socket, exit the program
                print('\nOh no, we have lost connection to the
server!')
                print('\nQuitting...')
                self.sock.close()
                os._exit(0)

class Client:
    """
    Supports management of client-server connections and integration
with the GUI.

    Attributes:
        host (str): The IP address of the server's listening socket.
        port (int): The port number of the server's listening socket.
        sock (socket.socket): The connected socket object.
        name (str): The username of the client.
        messages (tk.Listbox): The tk.Listbox object containing all
messages displayed on the GUI.
    """
    def __init__(self, host, port):
        self.host = host
        self.port = port
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.name = None
        self.messages = None

    def start(self):
```

```
        """
        Establishes the client-server connection. Gathers user input
for the username,
        creates and starts the Send and Receive threads, and notifies
other connected clients.

        Returns:
            A Receive object representing the receiving thread.
        """
        print('Trying to connect to {}:{}...'.format(self.host,
self.port))
        self.sock.connect((self.host, self.port))
        print('Successfully connected to {}:{}'.format(self.host,
self.port))

        print()
        self.name = input('Your name: ')

        print()
        print('Welcome, {}! Getting ready to send and receive
messages...'.format(self.name))

        # Create send and receive threads
        send = Send(self.sock, self.name)
        receive = Receive(self.sock, self.name)

        # Start send and receive threads
        send.start()
        receive.start()

        self.sock.sendall('Server: {} has joined the chat. Say
hi!'.format(self.name).encode('ascii'))
        print("\rAll set! Leave the chatroom anytime by typing
'QUIT'\n")
        print('{}: '.format(self.name), end = '')

        return receive

    def send(self, text_input):
        """
        Sends text_input data from the GUI. This method should be bound
to text_input and
        any other widgets that activate a similar function e.g.
buttons.
        Typing 'QUIT' will close the connection and exit the
```

```
application.

        Args:
            text_input(tk.Entry): A tk.Entry object meant for user text
input.
        """
        message = text_input.get()
        text_input.delete(0, tk.END)
        self.messages.insert(tk.END, '{}: {}'.format(self.name,
message))

        # Type 'QUIT' to leave the chatroom
        if message == 'QUIT':
            self.sock.sendall('Server: {} has left the
chat.'.format(self.name).encode('ascii'))

            print('\nQuitting...')
            self.sock.close()
            os._exit(0)

        # Send message to server for broadcasting
        else:
            self.sock.sendall('{}: {}'.format(self.name,
message).encode('ascii'))


def main(host, port):
    """
    Initializes and runs the GUI application.

    Args:
        host (str): The IP address of the server's listening socket.
        port (int): The port number of the server's listening socket.
    """
    client = Client(host, port)
    receive = client.start()

    window = tk.Tk()
    window.title('Chatroom')

    frm_messages = tk.Frame(master=window)
    scrollbar = tk.Scrollbar(master=frm_messages)
    messages = tk.Listbox(
        master=frm_messages,
        yscrollcommand=scrollbar.set
```

```
    )
    scrollbar.pack(side=tk.RIGHT, fill=tk.Y, expand=False)
    messages.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

    client.messages = messages
    receive.messages = messages

    frm_messages.grid(row=0, column=0, columnspan=2, sticky="nsew")

    frm_entry = tk.Frame(master=window)
    text_input = tk.Entry(master=frm_entry)
    text_input.pack(fill=tk.BOTH, expand=True)
    text_input.bind("<Return>", lambda x: client.send(text_input))
    text_input.insert(0, "Your message here.")

    btn_send = tk.Button(
        master=window,
        text='Send',
        command=lambda: client.send(text_input)
    )

    frm_entry.grid(row=1, column=0, padx=10, sticky="ew")
    btn_send.grid(row=1, column=1, pady=10, sticky="ew")

    window.rowconfigure(0, minsize=500, weight=1)
    window.rowconfigure(1, minsize=50, weight=0)
    window.columnconfigure(0, minsize=500, weight=1)
    window.columnconfigure(1, minsize=200, weight=0)

    window.mainloop()


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Chatroom Server')
    parser.add_argument('host', help='Interface the server listens at')
    parser.add_argument('-p', metavar='PORT', type=int, default=1060,
                        help='TCP port (default 1060)')
    args = parser.parse_args()

    main(args.host, args.p)
```

4. RESULTS AND DISCUSSION



5. CONCLUSION AND FUTURE ENHANCEMENT

Future enhancement:

Numerous applications can be made involving these strategies wherein the server holds all the data that clients can speak with that.

It tends to be made by putting away the data in the servers where the client texts consequently getting an answer by making a basic chatbot. These can be utilized broadly in many schools or universities where educators can cooperate with understudies or direct a test.

Conclusion:

This application can assist a large number of the expert establishments and Colleges with preferred schools, universities, and IT organizations. Thus, we expect to plan this application for the LAN of these associations. The individuals could utilize many highlights of this visit application to impart and conceptualize inside a LAN. Thus, essentially server-client applications can be utilized to do different kinds of quires, for example, clinical helpline administrations, and client report associations, which can be utilized in various situations. This furnishes the effective approach to speaking with the server, where nowadays it very well may be utilized to make a talk cycle with no UI or other man activity.

REFERENCES

[1] https://socket.io/docs/v4/ [2] https://realpython.com/python-sockets/ [3] “Multi-users communicating system in client/server mode based on www”, by Shen Ruiming, Computer Engineering, 1998(2):53-58 [4] \"The design of instant communication system in server-side\", by Huan Kai, Tao Hongcai, Journal of Chengdu University of Information Technology, 2006(4):535-538. [5] https://en.wikipedia.org/wiki/Online_chat#:~:text=References-,History,-The%20first%20online [6] Ahmed, Mohammed A., Sara Ammar Rafea, Lara Moufaq Falah, and Liqaa Samir Abd Ullah. \"Design and Implement Chat Program Using TCP/IP.\" Iraqi Journal for Computers and Informatics ijci 44, no. 1 (2018): 42-47. [7] The Python Tutorial — Python 3.10.4 documentation [8] Socket Programming in Python - GeeksforGeeks