Multi-threaded Socket-based Chatbot

Project Report

Spring 2023

CSE-204L Operating Systems Lab

Submitted by:

Shahzad Bangash(21PWCSE1980)

Suleman Shah(21PWCSE1983)

Ali Asghar(21PWCSE2059)

Class Section: **C**

"On my honor, as student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work."

Submitted to:

Engr. Madiha Sher

Date:

July 9, 2023

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

Problem Statement:

The problem addressed by this project is to develop a socket-based chatbot application that allows clients to communicate with a server and retrieve summarized information from Wikipedia. The goal is to enable users to send messages to the server and receive concise summaries of the corresponding Wikipedia articles in real-time.

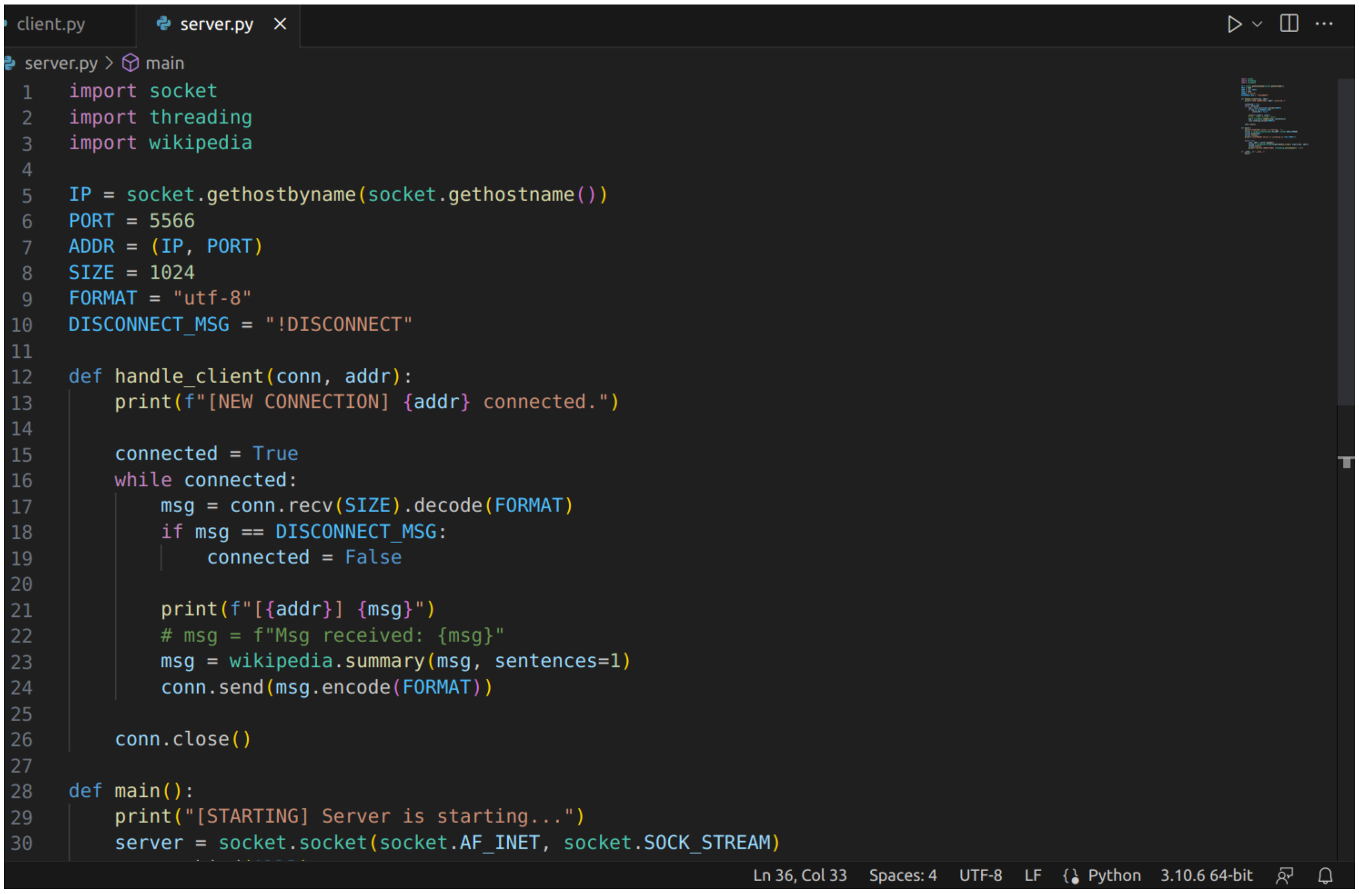
Introduction:

The project is a simple chatbot application that uses socket programming to establish communication between a client and a server. The server receives messages from the client, processes them, and retrieves a summary of the corresponding topic from Wikipedia. The server then sends the summary back to the client for display.

Methodology:

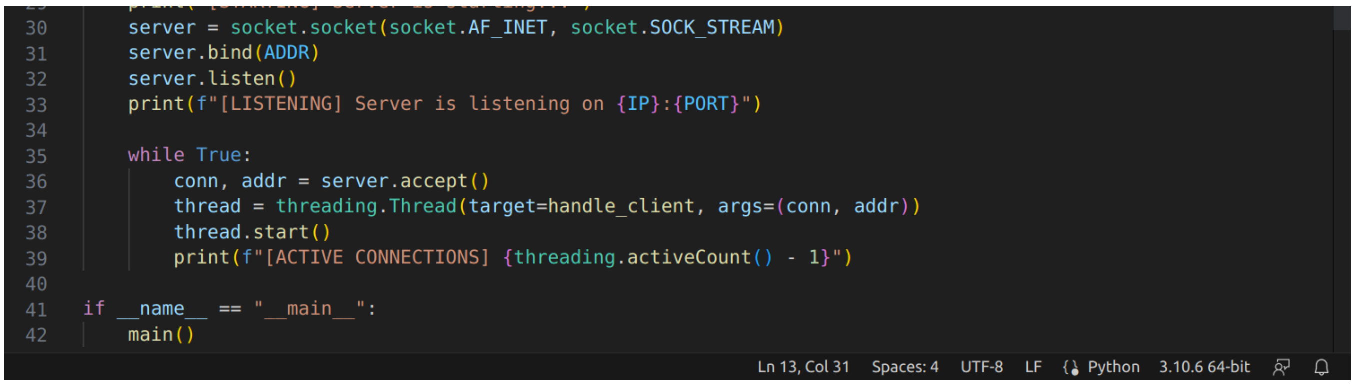
The project methodology comprises the following steps:

Code Screenshots:



```python
import socket
import threading
import wikipedia

IP = socket.gethostbyname(socket.gethostname())
PORT = 5566
ADDR = (IP, PORT)
SIZE = 1024
FORMAT = "utf-8"
DISCONNECT_MSG = "!DISCONNECT"

def handle_client(conn, addr):
    print(f"[NEW CONNECTION] {addr} connected.")

    connected = True
    while connected:
        msg = conn.recv(SIZE).decode(FORMAT)
        if msg == DISCONNECT_MSG:
            connected = False

        print(f"[{addr}] {msg}")
        # msg = f"Msg received: {msg}"
        msg = wikipedia.summary(msg, sentences=1)
        conn.send(msg.encode(FORMAT))

    conn.close()

def main():
    print("[STARTING] Server is starting...")
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

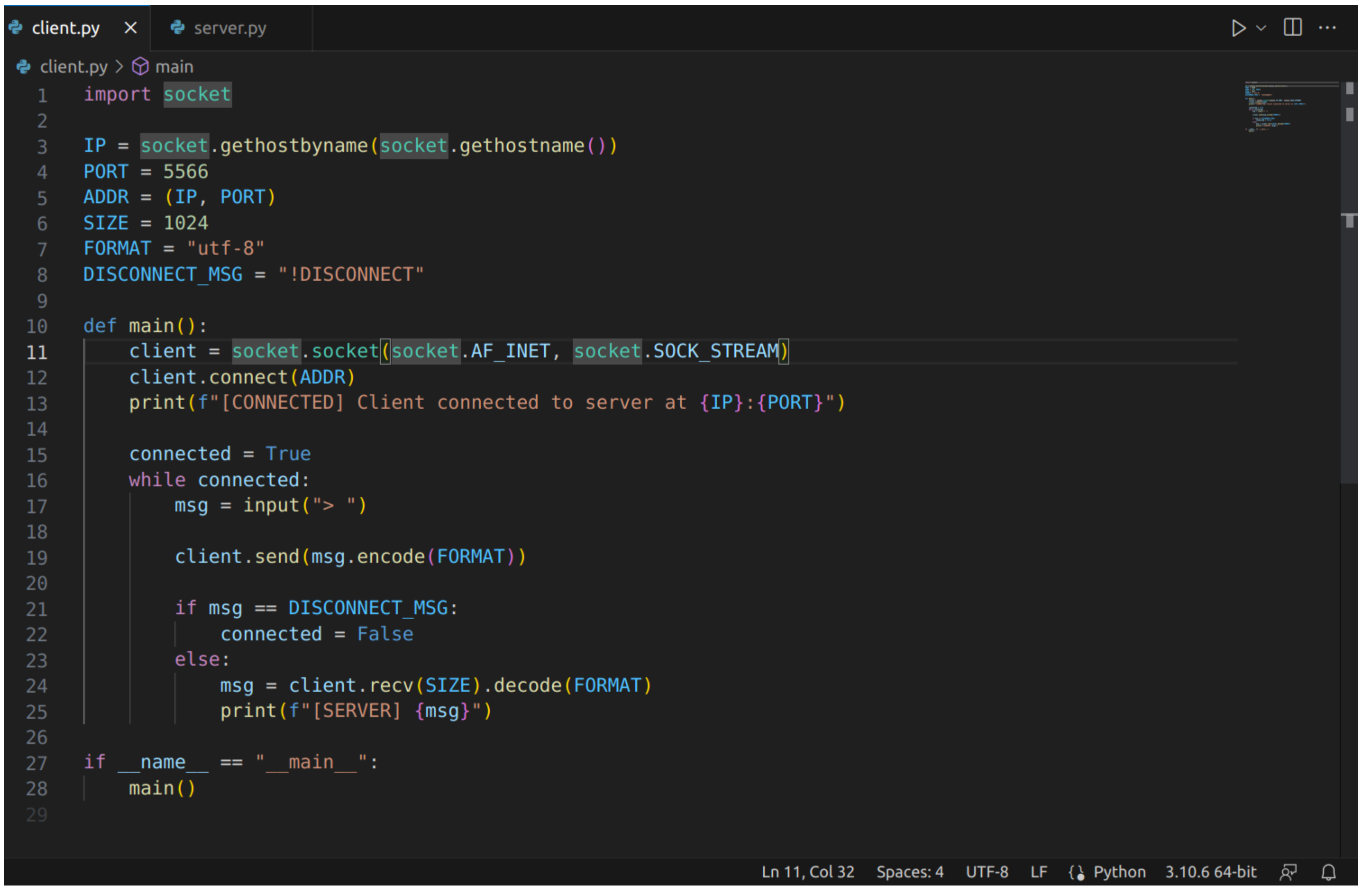
Figure 1-1a: Python Code for Server Implementation



```python
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind(ADDR)
    server.listen()
    print(f"[LISTENING] Server is listening on {IP}:{PORT}")

    while True:
        conn, addr = server.accept()
        thread = threading.Thread(target=handle_client, args=(conn, addr))
        thread.start()
        print(f"[ACTIVE CONNECTIONS] {threading.activeCount() - 1}")

if __name__ == "__main__":
    main()
```

Ln 13, Col 31   Spaces: 4   UTF-8   LF   Python   3.10.6 64-bit

Figure 1-1b: Python Code for Server Implementation(Remaining Part)



client.py   server.py

client.py > main

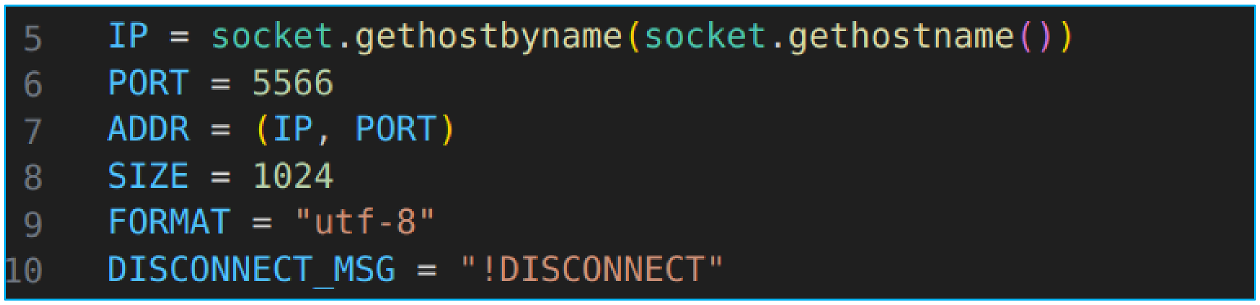
```python
import socket

IP = socket.gethostbyname(socket.gethostname())
PORT = 5566
ADDR = (IP, PORT)
SIZE = 1024
FORMAT = "utf-8"
DISCONNECT_MSG = "!DISCONNECT"

def main():
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect(ADDR)
    print(f"[CONNECTED] Client connected to server at {IP}:{PORT}")

    connected = True
    while connected:
        msg = input("> ")

        client.send(msg.encode(FORMAT))

        if msg == DISCONNECT_MSG:
            connected = False
        else:
            msg = client.recv(SIZE).decode(FORMAT)
            print(f"[SERVER] {msg}")

if __name__ == "__main__":
    main()
```

Ln 11, Col 32   Spaces: 4   UTF-8   LF   Python   3.10.6 64-bit

Figure 1-2: Python Code for Client Implementation

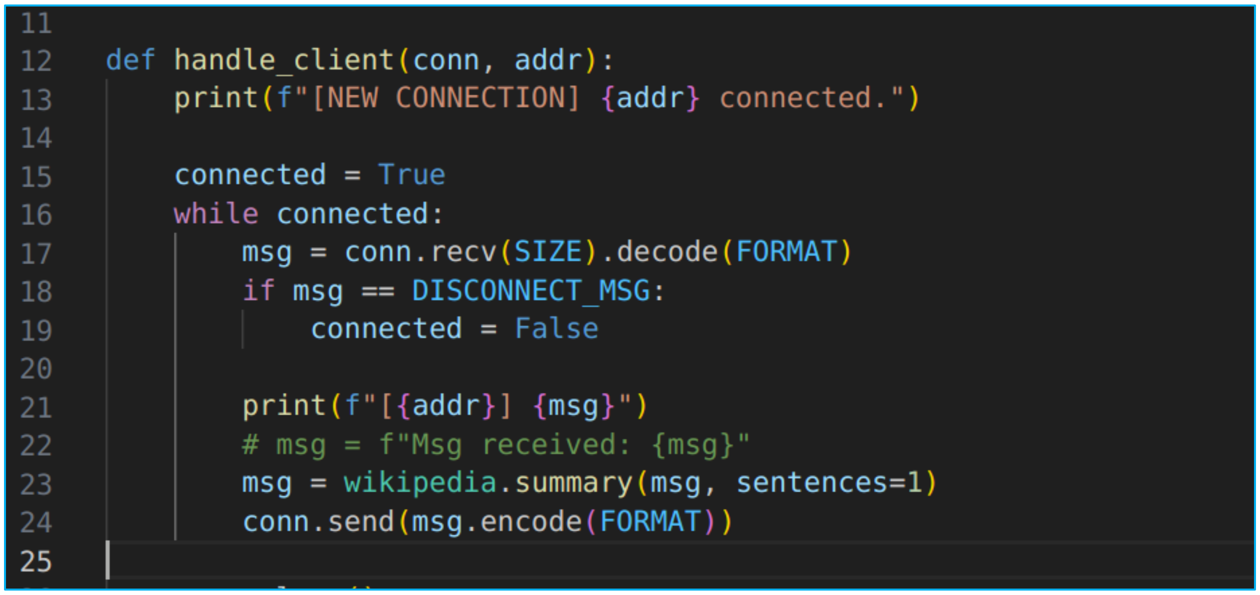
Code Explanation:

Server:



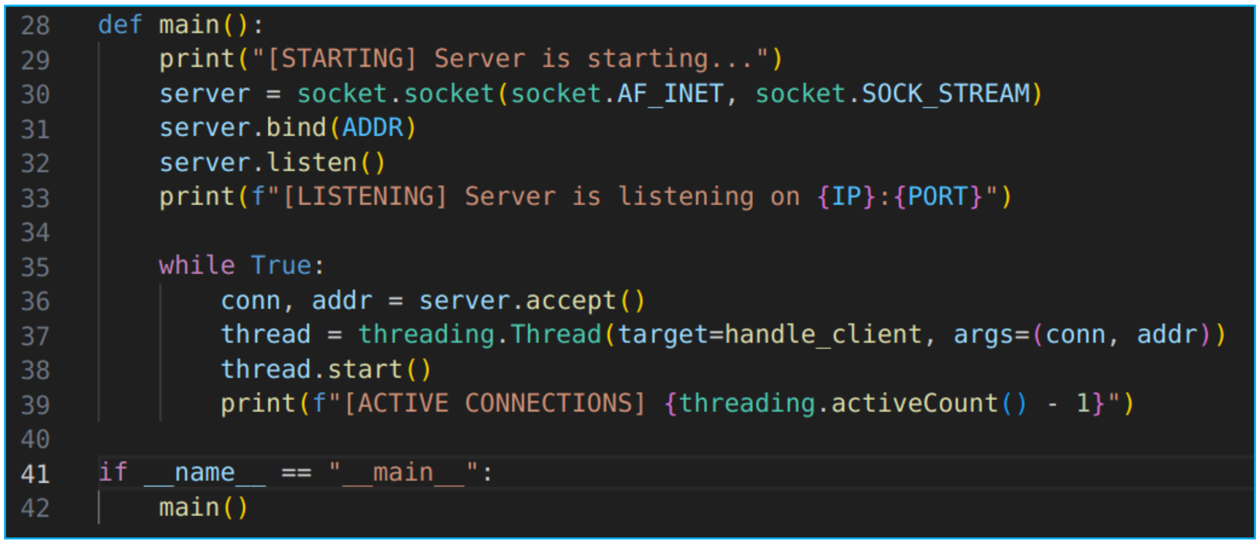
```
IP = socket.gethostbyname(socket.gethostname())
PORT = 5566
ADDR = (IP, PORT)
SIZE = 1024
FORMAT = "utf-8"
DISCONNECT_MSG = "!DISCONNECT"
```

The above lines of code initialize and assign values to important variables used in a client-server chatbot application. The **IP** variable retrieves the IP address of the host machine, while the **PORT** variable defines the port number for communication. The **ADDR** variable combines the IP address and port number into a tuple, representing the server address. The **SIZE** variable sets the maximum size of messages that can be sent or received. The **FORMAT** variable specifies the character encoding format for message encoding and decoding. Finally, the **DISCONNECT_MSG** variable defines a specific message indicating the client's intention to disconnect from the server. These variables play a crucial role in configuring and establishing communication between the client and server in the chatbot application.



```python

def handle_client(conn, addr):
    print(f"[NEW CONNECTION] {addr} connected.")

    connected = True
    while connected:
        msg = conn.recv(SIZE).decode(FORMAT)
        if msg == DISCONNECT_MSG:
            connected = False

        print(f"[{addr}] {msg}")
        # msg = f"Msg received: {msg}"
        msg = wikipedia.summary(msg, sentences=1)
        conn.send(msg.encode(FORMAT))

```

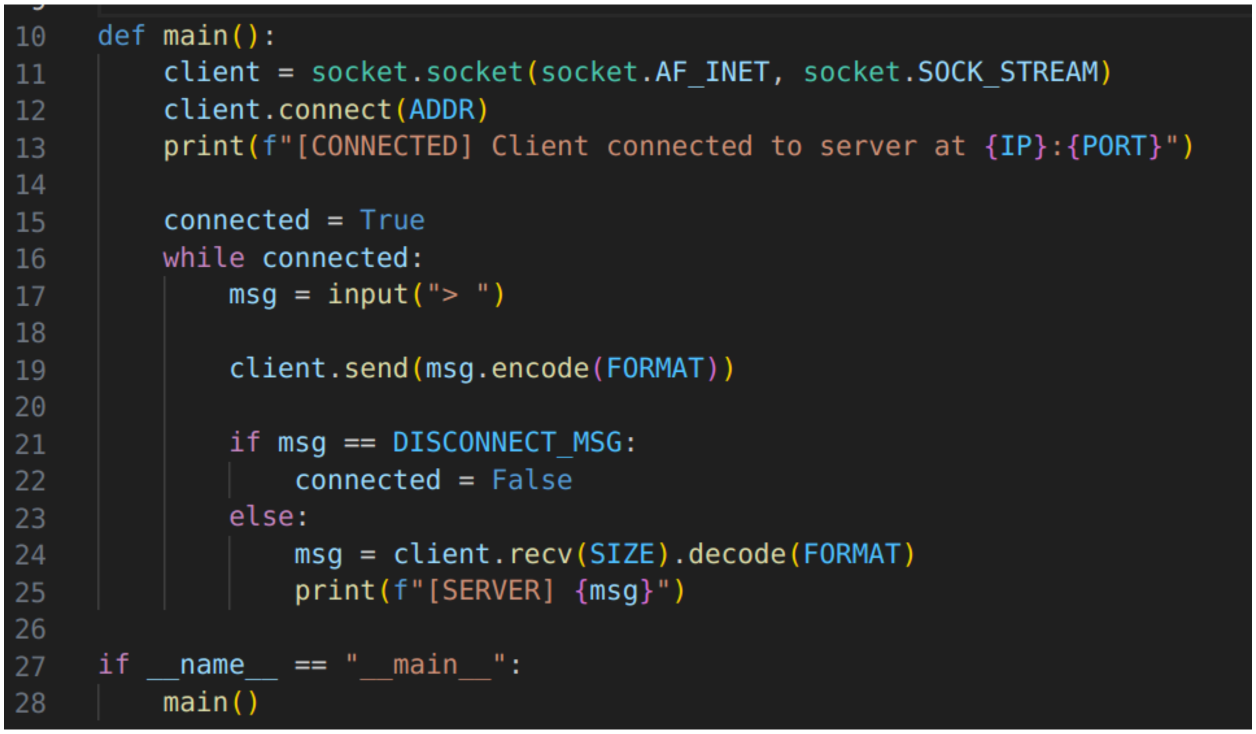
handle_client function is responsible for managing communication with a connected client in the server-side code. It receives messages from the client, checks if the client wants to disconnect, prints the received message and client's address, retrieves a summary of the message from Wikipedia, and sends the summary back to the client. This function continues to run as long as the client remains connected, and it closes the connection once the client decides to disconnect.



```python
def main():
    print("[STARTING] Server is starting...")
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind(ADDR)
    server.listen()
    print(f"[LISTENING] Server is listening on {IP}:{PORT}")

    while True:
        conn, addr = server.accept()
        thread = threading.Thread(target=handle_client, args=(conn, addr))
        thread.start()
        print(f"[ACTIVE CONNECTIONS] {threading.activeCount() - 1}")

if __name__ == "__main__":
    main()
```

The **main** function in the server-side code is responsible for setting up the server, listening for incoming client connections, and spawning new threads to handle each client. It creates a server socket, binds it to the specified IP address and port, and enters a continuous loop to accept connections. For each connection, it accepts the client, starts a new thread to handle communication with that client, and keeps track of the active connections. The function continuously runs until the program is terminated, allowing the server to handle multiple client connections concurrently.

Client:



```python
def main():
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect(ADDR)
    print(f"[CONNECTED] Client connected to server at {IP}:{PORT}")

    connected = True
    while connected:
        msg = input("> ")

        client.send(msg.encode(FORMAT))

        if msg == DISCONNECT_MSG:
            connected = False
        else:
            msg = client.recv(SIZE).decode(FORMAT)
            print(f"[SERVER] {msg}")

if __name__ == "__main__":
    main()
```

The **main** function in the client-side code establishes a connection with the server, allows the user to input messages, sends those messages to the server, and displays the server's responses. It continues to run until the client initiates a disconnection by sending a predefined disconnect message. This function forms the core of the client-side code, facilitating the interaction between the client and the server in a chat-like manner.

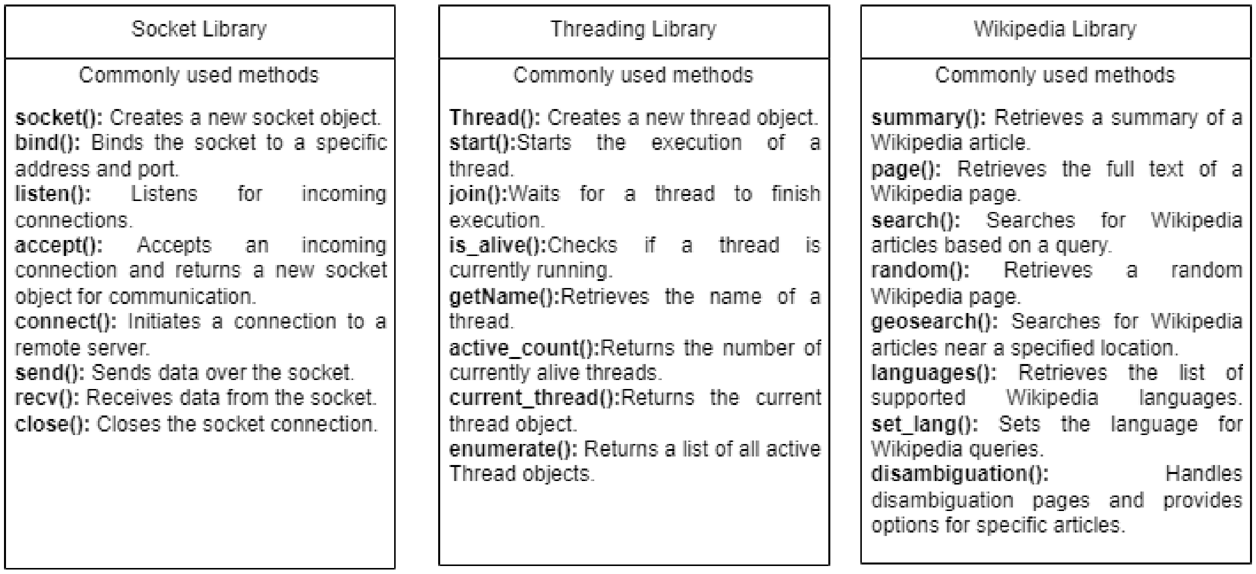
Modules/Libraries Used:

Here are the key libraries used:

Socket:
The socket library provides the necessary classes and methods for creating and managing socket connections. It enables communication between the client and server by establishing network connections and sending/receiving data over the network.

Threading:
The threading library is used to create and manage multiple threads within the server. Each thread handles communication with a specific client, allowing concurrent processing of multiple client requests.

Wikipedia:
The wikipedia library is used to retrieve information from Wikipedia. It provides a convenient interface to query and retrieve articles, summaries, and other details from the Wikipedia database. In this project, it is used to generate summarized information based on user queries.



Socket Library

Commonly used methods

socket(): Creates a new socket object.
bind(): Binds the socket to a specific address and port.
listen(): Listens for incoming connections.
accept(): Accepts an incoming connection and returns a new socket object for communication.
connect(): Initiates a connection to a remote server.
send(): Sends data over the socket.
recv(): Receives data from the socket.
close(): Closes the socket connection.

Threading Library

Commonly used methods

Thread(): Creates a new thread object.
start(): Starts the execution of a thread.
join(): Waits for a thread to finish execution.
is_alive(): Checks if a thread is currently running.
getName(): Retrieves the name of a thread.
active_count(): Returns the number of currently alive threads.
current_thread(): Returns the current thread object.
enumerate(): Returns a list of all active Thread objects.

Wikipedia Library

Commonly used methods

summary(): Retrieves a summary of a Wikipedia article.
page(): Retrieves the full text of a Wikipedia page.
search(): Searches for Wikipedia articles based on a query.
random(): Retrieves a random Wikipedia page.
geosearch(): Searches for Wikipedia articles near a specified location.
languages(): Retrieves the list of supported Wikipedia languages.
set_lang(): Sets the language for Wikipedia queries.
disambiguation(): Handles disambiguation pages and provides options for specific articles.

Figure 2-1: UML Diagram of Libraries Used

Flow of Program:

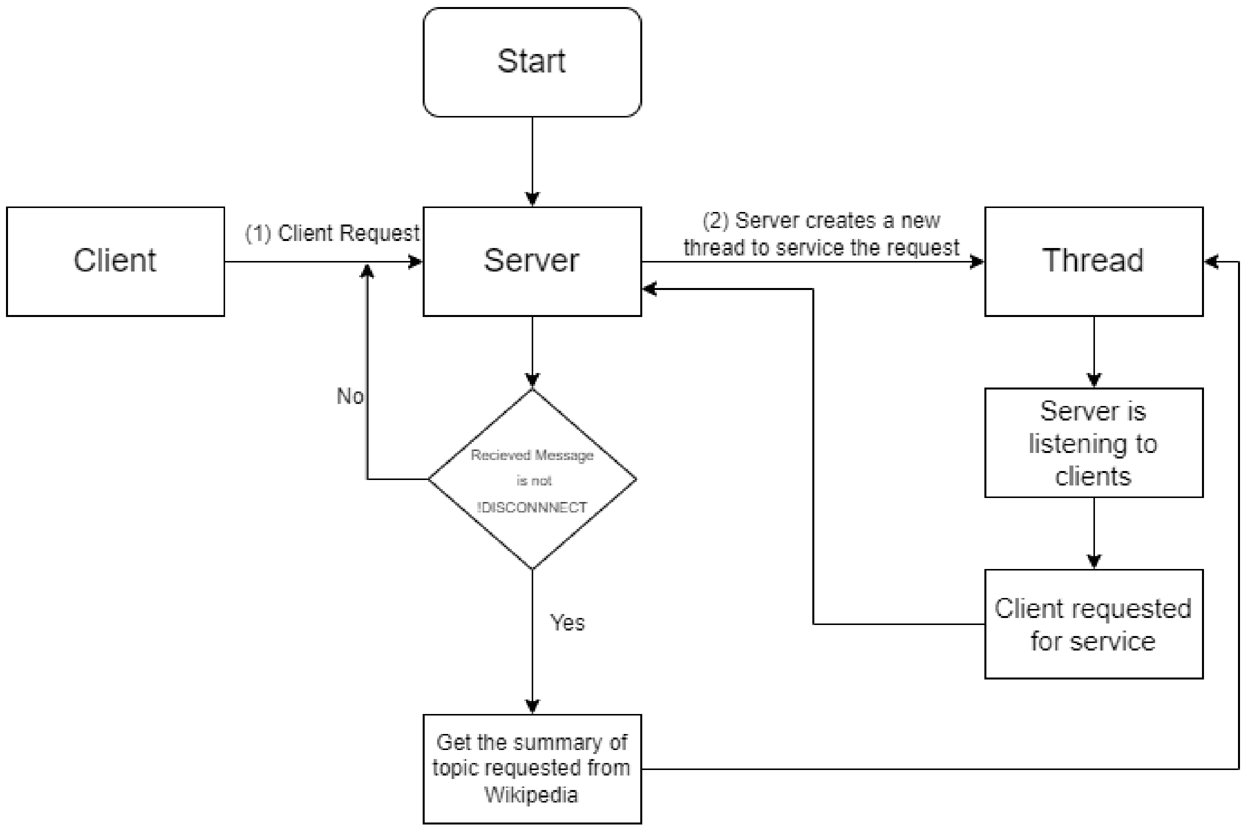

Start
Client
(1) Client Request
Server
(2) Server creates a new thread to service the request
Thread
No
Recieved Message is not !DISCONNNECT
Server is listening to clients
Yes
Client requested for service
Get the summary of topic requested from Wikipedia

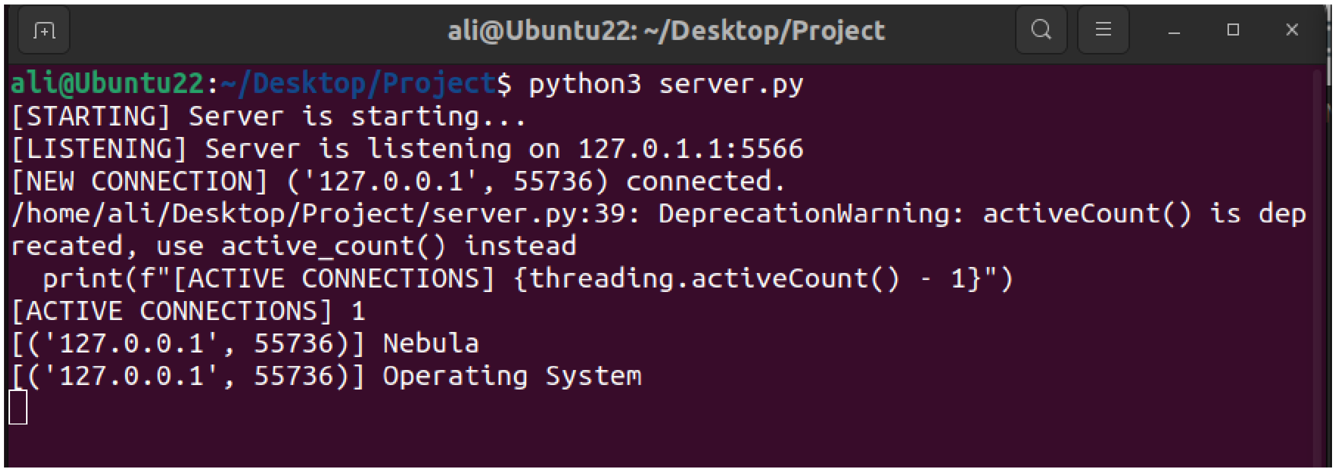
Figure 3-1: Working Flow Chart of this project

How to Use:

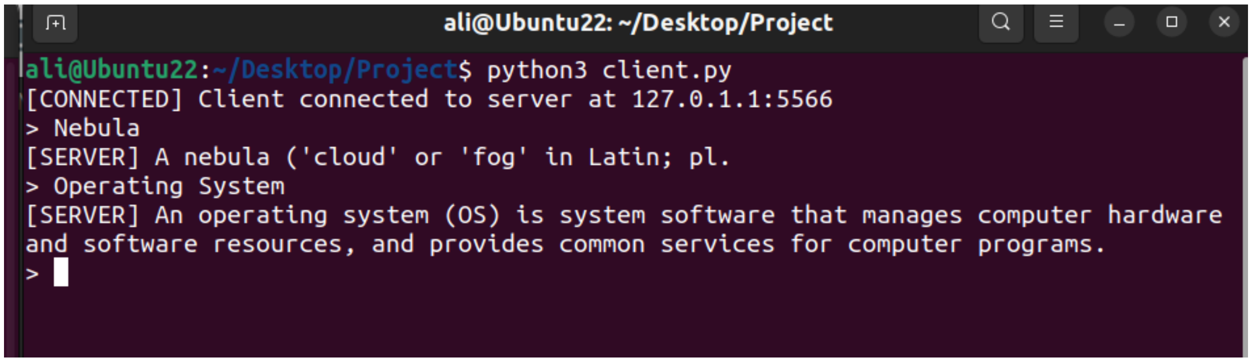
To use the chatbot application:

- Run the server script on a machine accessible on the network.
- Run the client script on a different machine, providing the server's IP address and port.
- Enter queries or messages in the client console.
- The server will retrieve the summary from Wikipedia and send it back to the client.
- The client will display the summary in its console.
- To disconnect, enter the predefined **!DISCONNECT** message.

Results:



```
ali@Ubuntu22:~/Desktop/Project$ python3 server.py
[STARTING] Server is starting...
[LISTENING] Server is listening on 127.0.1.1:5566
[NEW CONNECTION] ('127.0.0.1', 55736) connected.
/home/ali/Desktop/Project/server.py:39: DeprecationWarning: activeCount() is dep
recated, use active_count() instead
  print(f"[ACTIVE CONNECTIONS] {threading.activeCount() - 1}")
[ACTIVE CONNECTIONS] 1
[('127.0.0.1', 55736)] Nebula
[('127.0.0.1', 55736)] Operating System
```



```
ali@Ubuntu22:~/Desktop/Project$ python3 client.py
[CONNECTED] Client connected to server at 127.0.1.1:5566
> Nebula
[SERVER] A nebula ('cloud' or 'fog' in Latin; pl.
> Operating System
[SERVER] An operating system (OS) is system software that manages computer hardware
and software resources, and provides common services for computer programs.
> 
```

Conclusion:

The project demonstrates the use of socket programming to create a simple chatbot that retrieves information from Wikipedia. While it is a basic implementation, it provides a foundation for building more sophisticated chatbot applications. The project can be extended by integrating additional natural language processing capabilities, improving the user interface, and enhancing security measures.

References:

Abraham Silberschatz Operating System Concepts 10th Edition

https://github.com/nikhilroxtomar/multithread-client-server-in-python

https://www.geeksforgeeks.org/socket-programming-python