Project Report

Two Way Chat Application Program

(Using Socket Programming in Python)

EE-357

Computer and Communication Networks

Submitted by

Hira Najeeb	*268533*
Rushna Shah	*249747*
Muhammad Hamza Shakoor	*255633*
Karam Naveed	*265028*

Class and section: BEE 10B

Date of submission: 17th May,2021

Faculty Member: Dr. Hassaan Khalique

Table of Contents

1. Background

1.1 Socket Programming:

Socket programming enables communication between applications of programs running on computers in a network.

1.2 Sockets:

Sockets are Communication endpoint of computer that are built for sending and receiving data in a network. They are software objects with a combination IP address and port number.

1.3 Server socket and Client socket:

Server socket listens on a particular port at an IP address while client socket requests the connection to the server.

1.4 IP address and Port number:

IP address is a numerical label assigned to each computer connected to a network to identify the computer machine. For example: 192.168.1.10.

Port number directs the data to the correct application or process within the device (range 0-65535).

1.5 Server and Client:

Server is a software program or device which manages the resources required by the clients and it can be on a same computer or remote computer. Client is a software program or a device that requests data or services from servers.

2. Problem Statement:

In this project, we are required to create a chat box using socket programming in python. We have to create a webserver through which a minimum of 2 clients can communicate with each other. This chat box should be password protected and when the clients connect to the server, it will ask for the username and password which upon verification will share a list of usernames to the clients from which client will select a name and server will match the name/token from both clients. If matched, they both should receive a message that **"you are allowed to chat now"** and messages of the respective clients should pass through the server. The chat may end by sending some specific character or string.

3. Code Discussion:

client	12/06/2021 10:34 PM	PY File	7 KB
server	12/06/2021 10:34 PM	PY File	5 KB
users	13/06/2021 8:49 PM	Microsoft Excel C...	1 KB

- Our code folder consists of 3 files;
- ***client.py and server.py*** are python files for client and server side programming, respectively. Their working is explained separately.
- Users is a csv (or excel file) which contains database of our users,
  First column contains usernames and second column contains their passwords.

```
import threading
import socket
import json
import termcolor
import time
import numpy as np
import pandas as pd
```

- These are some important python libraries that we needed, used in both client and server files.

3.1 Server coding:

```
IP      = '127.0.0.1'   # server ip
# IP      = socket.gethostname()      # server ip
PORT    = 9999                      # server port
ADDRESS = IP, PORT
```

- Since we are using localhost to host our server that's why we set IP to 127.0.0.1
- We choose 9999 as our port, it could be anything else as well.

```
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Allows a socket to bind to an address and port already in use.

server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server.bind(ADDRESS) # binidng port ip with server
server.listen()
clients = []
aliases = []
```

- Creates a new socket using the given address family, socket type and protocol number.
- ipv4 family is used.
- TCP (SOCK_STREAM) is a connection-based protocol.
- The connection is established by this code so that the two parties can have a conversation until the connection is terminated by one of the parties or by a network error or manually closing the program.

3.2 Following is the brief discussion of server.py functions:

```
def specific_send(message):
    '''
```

- This function forwards messages to respective clients after checking the authenticity.

```
def get_users():
    '''
    Read username column from db and return
    '''
    df = pd.read_csv('users.csv')
    new = df['username'].to_numpy().tolist()
    return new
```

- Reads usernames column from data base file and return to the program.

```
def recv_all(target):
    '''
    Receive as long as there is something to receive,
    can receive more than 1024 bytes
    '''
    data = ''
    while True:
        try:
            # rstrip removes spaces at end
            data = data + target.recv(1024).decode().rstrip()
            return json.loads(data)
        except ValueError:
            continue
```

- This function receives packets (messages) as long as there is something to receive. It can receive more than 1024 bytes.
- Once all data is received this function comes out of the receiving loop.

```
def send_to_socket(target, data):
    '''
    Reliable send, json object encoded as string

    '''
    # json.dumps() takes in a json object and returns a string.

    jsondata = json.dumps(data)
    target.send(jsondata.encode())
```

Sends encoded json string data.

```
def handle_client(client):
    '''
    Works in different thread,
    handle every client connects with server
    checks header to call specific function
    '''
    while True:
        try:
            message = recv_all(client)
            if str(message[0])=='messaging':
                # messageing meant between specific clients
                specific_send(message)
            else:
                # if not broadcast
                broadcast(message)
```

```
    except:
        index = clients.index(client)
        clients.remove(client)
        client.close()
        alias = aliases[index]
        aliases.remove(alias)
        #remove client if disconnected from list
        break
```

- Main function to receive the client's connection
- Can work in multiple threads
- Handles every client connected with the server
- Checks header to call specific function.

```
def authenticate_user(name,password):
    '''
    Loading users from db and comparing with fed name and p
    '''
    #checks for password and username parameters
    df=pd.read_csv('./users.csv')
    row=df.loc[df['username']==name]
    isAuth=False
    row=np.array(row)
    if row.size:
        isAuth=row[0][1]==password
    return isAuth
```

- Loading users from db and comparing with fed name and password and returns true if authenticated.

```python
def broadcast(message):
    '''
    Broadcast messages ato all clients
    '''
    for client in clients:
        send_to_socket(client,message)
```

This function broadcasts messages to all clients.

```python
def receive():
    '''
    Once User connects it, checks user credentials with db and send auth=true Re
    It also asks for clients name and then calls sender and receiver threads.
    '''
    while True:
        print(termcolor.colored(f'[+] Server is Running on address {ADDRESS} \n
        client, address = server.accept()
        isAuth = False
        while not isAuth:
            data = recv_all(client)
            time.sleep(0.05)
            if data[0] == 'auth':
                isAuth = authenticate_user(data[1],data[2])
                users_list = get_users()
                time.sleep(0.050)
```

```
            users_list = get_users()
            time.sleep(0.050)
            send_to_socket(client, ['auth_res', isAuth, users_list])
            time.sleep(0.050)

        time.sleep(2)
        # print(termcolor.colored(str(address) + ' has connected!', 'green'))
        send_to_socket(client,'alias?')
        alias = recv_all(client)
        aliases.append(alias)
        clients.append(client)
        print(f'[{ address[0] }: { address[1] }] joined in as {alias}')
        send_to_socket(client,'you are now connected!')
        thread = threading.Thread(target= handle_client, args= (client,))
        thread.start()
```

Actually this is the main function of server.py that does all the work and calls other functions.

3.3 Client Code:

```
USER = '' # Username will be enterd by User
PASS = '' # Password Will be input bu user
is_auth = False # is user authorized ro neter into chat?
USERS = [] # users available for chat will be fetch from server
CHATTING_WITH = '' # User Selected for chat
is_selected = False # boolean is user selected for chat
```

Main variables for the necessary information.

```
SERVER_IP   = '127.0.0.1'
SERVER_PORT = 9999
```

```
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect((SERVER_IP, SERVER_PORT)) # connectting with serverip and port
```

- Create a new socket using the given address family, socket type and protocol number.
- ipv4 family is used
- TCP (SOCK_STREAM) is a connection-based protocol. The connection is established the two parties have a conversation until the connection is terminated by one.

```
def authenticator():
    '''
        Function to get inputs from users like username and passw
        Communicates with server.
        Sends Credentials to Server and If authorized by server,

    '''
    global USER
    global PASS
    global USERS
    isConn=False
    #iterate until not authroized
    while not isConn:
        USER = validate_striing('Enter username:')
        PASS = validate_striing('Enter password:')
        reliable_send(['auth',USER,PASS])
        time.sleep(0.050)
        data = reliable_recv()
```

```
        time.sleep(0.050)
        #check response from server
        if data[0]=='auth_res':
            #will set true if authorized
            isConn=data[1]
            #get users from db for chat
            USERS=list(data[2])
            if isConn:
                print(termcolor.colored('[+] You are authenticated ...', 'green'))
            else:
                print(termcolor.colored('[+] You have entered an invalid username
    return isConn
```

- Function to get inputs from users like username and password.
- Communicates with server.
- Sends Credentials to Server and If authorized by server, returns True.

```
def select_user_for_chat():
    '''
    Asks user for for serial of users, displayed, available for chat.
    Selected user assigned to CHATTING with variable.
    Return True on selection.
    '''
    global USERS
    global CHATTING_WITH
    isSelected=False
    #iterate until not selected
    while not isSelected:
        #printing users with index
        for i,name in enumerate(USERS):
            print('{} >> {}\n'.format(i,name) )
        #get serial from user
        num=get_number('Enter username serial number from above to chat:')
        #update chatting with varibale
        CHATTING_WITH=USERS[num]
```

```
#on selection display you are chatting with
if len(str(CHATTING_WITH)):
    print(termcolor.colored('[+] You are now chatting with '+str(CHATTING_WITH)))
    isSelected=True
else:
    print(termcolor.colored('[+] You have selected an invalid username', 'red'))
urn isSelected
```

- Asks user for for serial of users, display, available for chat.
- Selected user assigned to chatting.
- Return True on selection.

```
def get_number(string):
    '''
    Helps get digit input from user and keeps asking on wrong entry.
    '''
    number=input(string)
    if not number.isdigit():
        validate_striing('Enter a number!' + string)
    return int(float(number))
```

- Helps get digit (index number) from user and keeps asking if user enters a wrong number.

```python
def client_receive():
    '''
    Receive function for client, will work in a thread
     1. When to end chat
     2. Sending alias or username to server
     3. checking received message for authorized client
    '''
    global USER
    global CHATTING_WITH
    global is_selected
    while True:
        try:
            message = reliable_recv()
            #check if server asks for username
            if message == 'alias?':
                reliable_send(USER)
            #check for end command
```

```python
        elif str(message[1])== CHATTING_WITH and str(message[2])==USER and str(m
            is_selected=False
            CHATTING_WITH=''
        #check if message is from authorized client
        elif str(message[1])== CHATTING_WITH and str(message[2])==USER :
            print('{} > {}'.format(message[1],message[3]))
        else:
            continue
    except:
        print('Error!')
        client.close()
        break
```

- **Receive function for client, will work in a thread and checks are ended:**

1. When to end chat
2. Sending alias or username to server
3. checking received message for authorized client for chat (message sender and receiver) and displaying them.

```
def client_send():
    '''
    Send function for client, will work in a separate thread and checks are en
        1. Asking User for person to chat with by calling select_user_for_chat
        2. checking message to be send, command @end, to end chat timely.
        3. message send formatted as [sender,receiver, message] list
    '''
    global CHATTING_WITH
    global is_selected
    while True:
        if not CHATTING_WITH and not is_selected:
            is_selected = select_user_for_chat()
        elif is_selected==True:
            input_msg=input('')
            if input_msg.lower()=='@end':
                # check for chat end command
                reliable_send(['messaging',USER,CHATTING_WITH,input_msg])
                is_selected=False
```

```
            CHATTING_WITH=''
        else:
            #sending formatted message
            reliable_send(['messaging',USER,CHATTING_WITH,input_msg])
```

- **Send function for client, will work in a separate thread and following checks are ended:**

1. Asking User for person to chat with, by calling select_user_for_chat function
2. checking message to be send, command @end, to end chat timely.
3. Send message with formating as [sender,receiver, message] list.

```python
def reliable_send(data):
    '''
    Reliable send, json object encoded as string

    '''
    #json.dumps() takes in a json object and returns a string.

    jsondata = json.dumps(data)
    client.send(jsondata.encode())
```

Reliably sends json object encoded as string.

```python
def reliable_recv():
    '''
    Receive as long as there is something to receive, can receive more
    '''

    data = ''
    while True:
        try:
            #receive as long as there is something to receive
            data = data + client.recv(1024).decode().rstrip()
            #json.loads() takes in a string and returns a json object.
            return json.loads(data)
        except ValueError:
            continue
```

- This function receives data as long as there is something to receive can receive more than 1024 bytes.

```python
def validate_striing(string):

    '''
    To get string from user and check if its or not
    if it is then ask again
    '''
    name = input(string)
    if not len(str(name)):
        #if empty ask again
        validate_striing('Empty!' + string)
    return name
```

- Gets string from user and checks if it is the authentic user or not,
- If it’s not the authentic user, then asks again.

```python
def initialize():
    '''
    check users entered details for auth from client, received by calling
    Once user auth remove user alias from users list and initiate multithr
    '''
    print(termcolor.colored('[+] Connected to the Server ...', 'green'))

    global is_auth
    global USER
    while not is_auth:
        is_auth = authenticator()
        print('is_auth', is_auth)
        if is_auth:
            USERS.remove(USER)
            receive_thread = threading.Thread(target=client_receive)
            receive_thread.start()
```

```
            send_thread = threading.Thread(target=client_send)
            send_thread.start()

def main():
    initialize()

#starting program
if __name__ == '__main__': main()
```

- Check users entered details for authorization from client
- This is received by calling authenticator function.
- Once user auth remove user alias from users list, it initiates multithreads of receiving and sending.
- Then main function of python calls initialize function, which basically calls every other function inside itself.

3.4 Output:

- Since we are using **Multithreading**
  [***definition:*** *specifically refers to the concurrent execution of more than one sequential set/thread of instructions. In our case, server and multiple clients' file running at the same time in the same PC*]

- We have to run each file in a separate terminal of the IDE.
- Here we are using one terminal for server file and 2 terminals for two separate clients.

Terminal 1:

```
Terminal:  Local  Local (2)  Local (3)  +
Microsoft Windows [Version 10.0.19042.1052]
(c) Microsoft Corporation. All rights reserved.

C:\Users\dell\Desktop\CCN project>python server.py
←[32m[+] Server is Running on address ('127.0.0.1', 9999)
    Waiting for incoming connections ...←[0m
[127.0.0.1: 55952] joined in as hamza
←[32m[+] Server is Running on address ('127.0.0.1', 9999)
    Waiting for incoming connections ...←[0m
[127.0.0.1: 55953] joined in as karam
←[32m[+] Server is Running on address ('127.0.0.1', 9999)
    Waiting for incoming connections ...←[0m
```

Terminal 2:

```
Microsoft Windows [Version 10.0.19042.1052]
(c) Microsoft Corporation. All rights reserved.

C:\Users\dell\Desktop\CCN project>python client.py
←[32m[+] Connected to the Server ...←[0m
Enter username:hamza
Enter password:123
←[32m[+] You are authenticated ...←[0m
is_auth True
0 >> karam

1 >> hammad

2 >> hello

3 >> max

Enter username serial number from above to chat:0
[+] You are now chatting with karam←[0m
```

Terminal 3:

```
C:\Users\dell\Desktop\CCN project>python client.py
←[32m[+] Connected to the Server ...←[0m
Enter username:karam
Enter password:naveed
←[32m[+] You are authenticated ...←[0m
is_auth True
0 >> hamza

1 >> hammad

2 >> hello

3 >> max

Enter username serial number from above to chat:0
[+] You are now chatting with hamza←[0m
```

3.5 Chatting:

Hamza's window

```
[+] You are now chatting with karam←[0m
hello
karam > hi
how are you??
karam > i am fine, wby?
I am doing CCN project
karam > Oh man, i guess it must be hard to work on it. right?
yess, but it's worth it, i have learnt a lot about socket programming
```

Karam's Window:

```
[+] You are now chatting with hamza←[0m
hamza > hello
hi
hamza > how are you??
i am fine, wby?
hamza > I am doing CCN project
Oh man, i guess it must be hard to work on it. right?
hamza > yess, but it's worth it, i have learnt a lot about socket programming
```

Underlined names indicates the sender, that sent the message to the user.

4. Future Advancements:

There is always a room for improvements in any software package, however good and efficient it maybe done. But the most important thing is that it should be flexible enough to accept further modification. Right now we are just dealing with

Text communication. In future, this software maybe extended to include features such as:

- **File transfer**
  This will enable the user to send files of different formats to others via the chat application.
- **Voice chat**
  This will enhance the application to a higher level where communication will be possible via voice calling as in telephone.
- **Video chat**
  This will further enhance the feature of calling into video communication.
- **Hosting on online server**
  So that the clients can access the chat box remotely using internet.

5. Conclusion:

Chatting is a method of using technology to bring people and ideas together despite the geographical barriers. This technology has been available for years, but its acceptance is quite recent. Our project is an example of a chat server. It is made up of two application the client application which runs on the users PC and server application which runs on any PC in the network.

Communication over a network is one field where this chat box finds wide range of applications. This tool can be used for large scale communication and conferencing in an organization or campus of vast size, thus increasing the standard of operation.

Our objective was to use socket programming to build a computer network and we successfully managed to achieve our goals.