HO CHI MINH UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE & ENGINEERING

COMPUTER NETWORK

ASSIGNMENT 1

Build a chat application according to the communication protocols defined by each group, using the TCP / IP protocols.

Lecturer: Nguyễn Phương Duy

Class: CC02

Student's name	Student ID
Trần Đinh Anh Hùng	2053066
Hoàng Đỗ Phương Nguyên	1952360
Tạ Bá Huy	2053052

Ho Chi Minh City – 2022

Table Of Contents

1. Requirement analysis

- Application allows to or more users on different machine to chat with each other
- The chat client is built on the hybrid model between the Client-Server and P2P models. The system has a central server for user registration and online user management, but clients chat directly with each other. (Students can refer to Napster for an illustration).
- One person has a list of friends retrieved from the central server. He/she can start some chats with different people at the same time. Each conversation is a P2P connection.
- Application allows file transfer during chats between clients through P2P connection.
-The application must use communication protocols already pre-defined by each group.

2. Client-Server and P2P model

2.1. What are servers and clients?

Servers are the providers of a resource or service whereas someone who requests for service is known as the client. Usually, communication between clients and servers happen through a computer network on separate hardware. However, it is also possible for the client and server to reside in the same system. A server host shares their resources with the clients byrunning one or more server programs. A client requests contents or service functions from a server but does not share any of its resources. Therefore, it can be said that the communication sessions are initiated by the clients to the servers, who waits for incoming requests.

2.2. Client-Server model

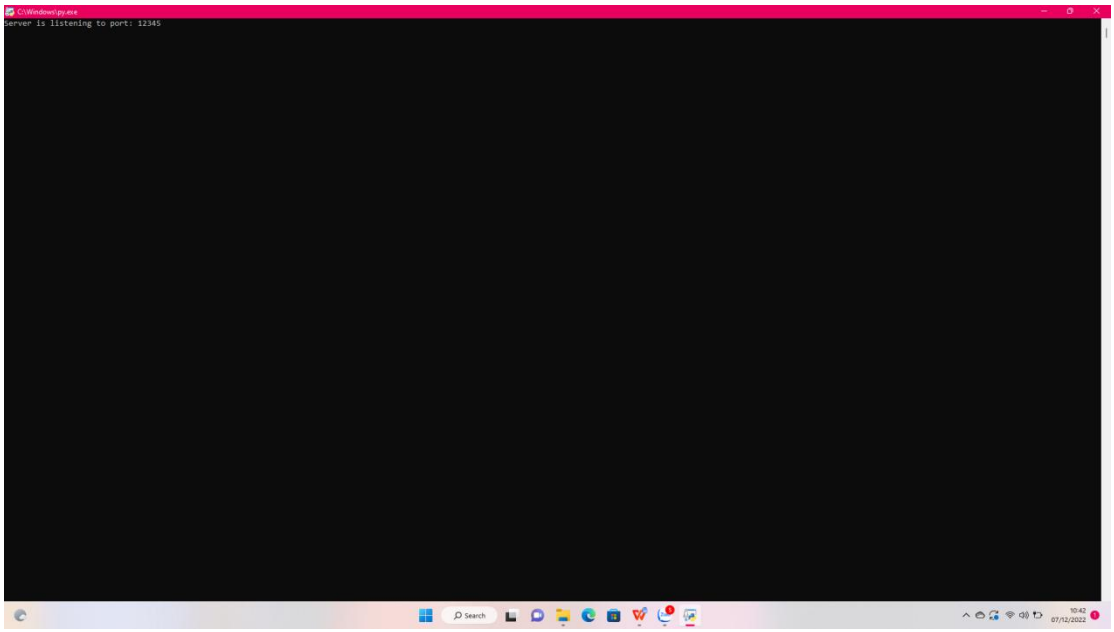
Is an architecture of application whereby a service or action is requested by the client to the server, which play the role of service provider. One good example to illustrate this model is he communication between a Web Browser and a Web server. Consider a Web browser and a Web server. When we type a URL address in the browser window, the Web browser (which plays the role of a client) will request a page from a Web server (which is the service provide or server). The Web server will send a html page to the client which will parse the page (also known as data) and it will be displayed on our computer screen.

2.3. P2P model

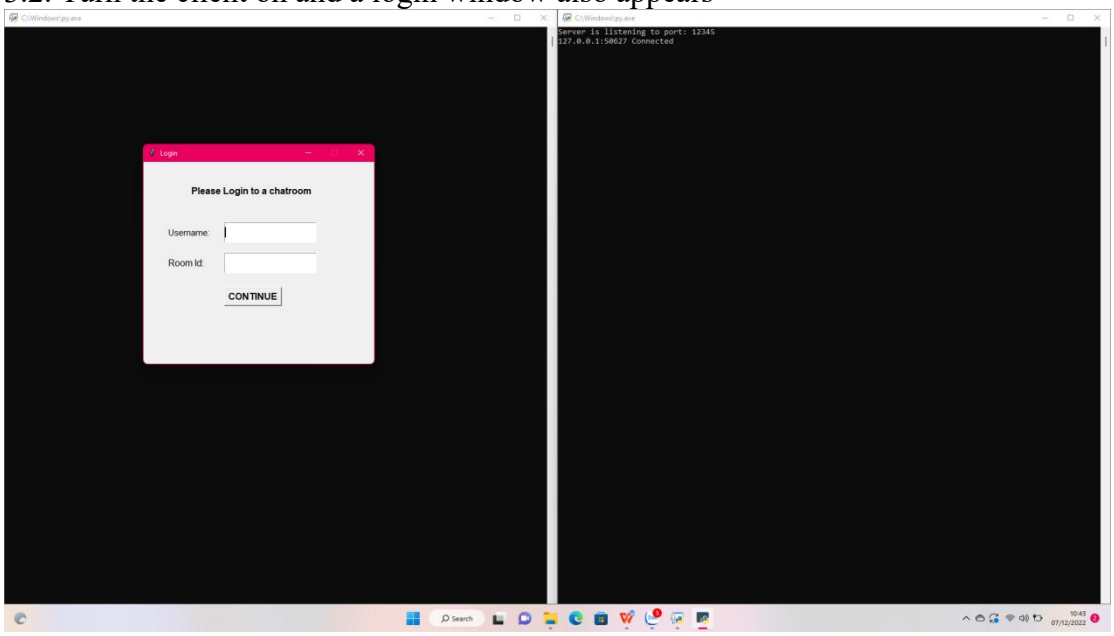
When a user downloads a file, the process is typically as follows: The user opens a web browser, visits the appropriate website and downloads the file. In this case, the website acts as a server, and the user's computer acts as a client that receives the data. This process is like a one-way street where the downloaded file transfers from point A, the website, to point B, the user's computer. If the user downloads the same file through a P2P network, however, the process occurs differently. The user must install P2P software on their computer, which creates a virtual network of P2P application users. When the user downloads a file, the file downloads in bits that come from various computers in the network that already have the file. Simultaneously, data also travels from the user's computer to other computers in the network that ask for it. This situation is like a two-way street -- the file is like numerous small bits of data that come to the user's computer but also leave when requested. In fact, the file transfer load distributes between the peer computers.

3. Demo

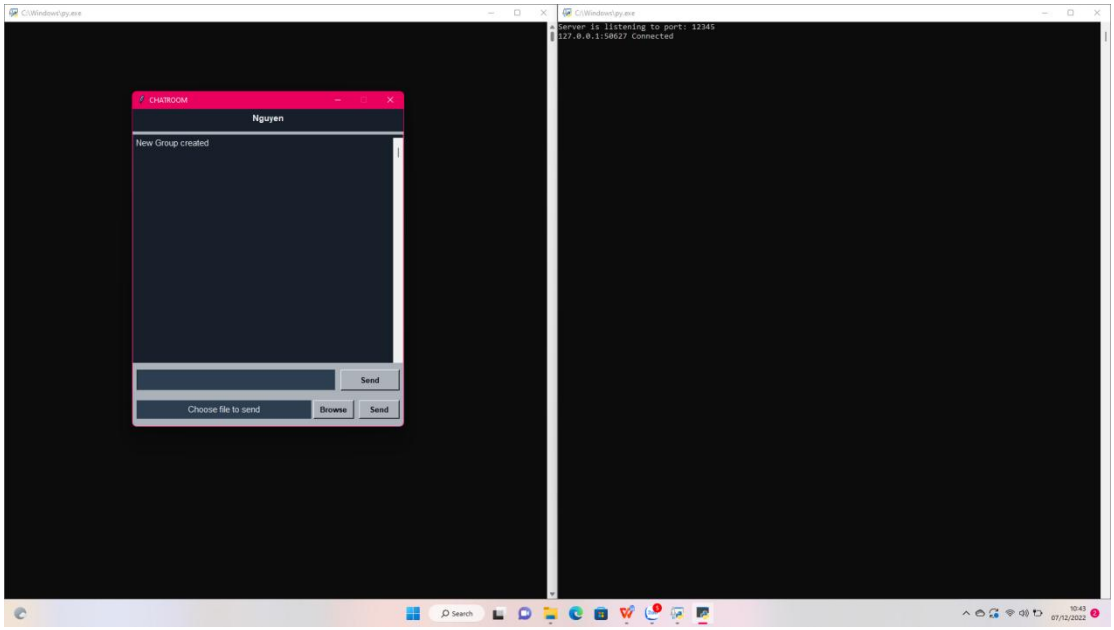
3.1. Turn the server on first



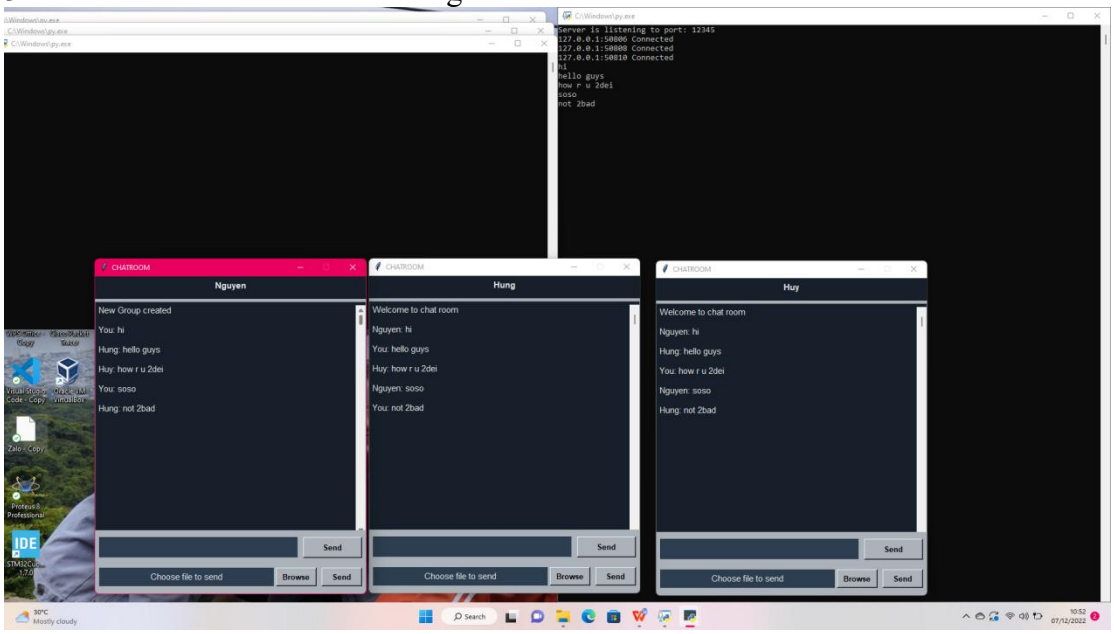
3.2. Turn the client on and a login window also appears



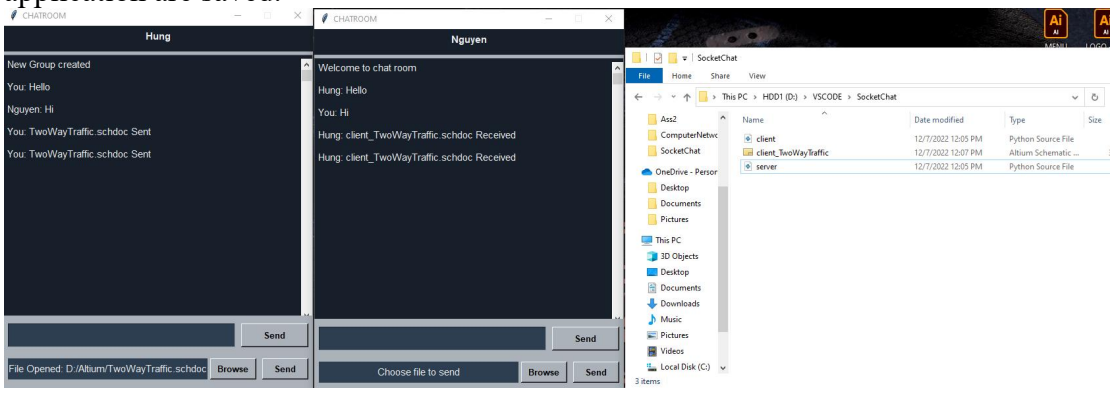
3.3. Enter the username and room ID, click CONTINUE button. At that time, in server terminal displays the user's address and IP address and the chat UI window also appears.



3.4. We create one more user to begin the chat



3.5. The user called Hung browsed a file in his computer and and send to the other user. The user who receive the file can find that file in the folder in which the application are saved.



4. Server side

- class Server jobs is to handle tasks from the server including create a new server, establish connections and send files

- `__INIT__(self)`: creating a new server with AF_INET(IP v4) and SOCK_STREAM(TCP or UDP…).

```
def __init__(self):
    self.rooms = df(list)
    self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
```

- `ACCEPT_CONNECTION(self, ip address, port)`: link newly created self.server to the IP address and port, which in TCP protocol. The maximum listening number of port is 100.

- `CLIENTTHREAT(self, connection)`: if the message is a file then it jumps to `broadcastFile()`, else the message jumps to `broadcast()`.

```
def accept_connections(self, ip_address, port):
    self.ip_address = ip_address
    self.port = port
    self.server.bind((self.ip_address, int(self.port)))
    self.server.listen(100)

    print("Server is listening to port: " + str(port))
    while True:
        connection, address = self.server.accept()
        print(str(address[0]) + ":" + str(address[1]) + " Connected")

        start_new_thread(self.clientThread, (connection,))

    self.server.close()
```

- `BROADCASTFILE(self, connection, room_id, user_id)`: send to client processed file

```
def broadcastFile(self, connection, room_id, user_id):
    file_name = connection.recv(1024).decode()
    lenOfFile = connection.recv(1024).decode()
    for client in self.rooms[room_id]:
        if client != connection:
            try:
                client.send("FILE".encode())
                time.sleep(0.1)
                client.send(file_name.encode())
                time.sleep(0.1)
                client.send(lenOfFile.encode())
                time.sleep(0.1)
                client.send(user_id.encode())
            except:
                client.close()
                self.remove(client, room_id)

    total = 0
    print(file_name, lenOfFile)
    while str(total) != lenOfFile:
        data = connection.recv(1024)
        total = total + len(data)
        for client in self.rooms[room_id]:
            if client != connection:
                try:
                    client.send(data)
                    # time.sleep(0.1)
                except:
                    client.close()
                    self.remove(client, room_id)
    print("Sent")
```

- `BROADCAST(self, message to send, connectin, room_id)`: send to client processed message

```
def broadcast(self, message_to_send, connection, room_id):
    for client in self.rooms[room_id]:
        if client != connection:
            try:
                client.send(message_to_send.encode())
            except:
                client.close()
                self.remove(client, room_id)
```

- `REMOVE(self, connection, room_id)`: close client if error happens, then functions above will packaged into try except

```
def remove(self, connection, room_id):
    if connection in self.rooms[room_id]:
        self.rooms[room_id].remove(connection)
```

5. Client side

At first, we design the UI interface for the chat application by using Tkinter - a standard GUI library for Python, Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

```
import socket
import tkinter as tk
from tkinter import font
from tkinter import ttk
from tkinter import filedialog
import time
import threading
import os

class GUI:

    def __init__(self, ip_address, port):
        self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.server.connect((ip_address, port))

        self.Window = tk.Tk()
        self.Window.withdraw()

        self.login = tk.Toplevel()

        self.login.title("Login")
        self.login.resizable(width=False, height=False)
        self.login.configure(width=400, height=350)

        self.pls = tk.Label(self.login,
                            text="Please Login to a chatroom",
                            justify=tk.CENTER,
                            font="Helvetica 12 bold")

        self.pls.place(relheight=0.15, relx=0.2, rely=0.07)

        self.userLabelName = tk.Label(self.login, text="Username: ", font="Helvetica 11")
        self.userLabelName.place(relheight=0.2, relx=0.1, rely=0.25)

        self.userEntryName = tk.Entry(self.login, font="Helvetica 12")
        self.userEntryName.place(relwidth=0.4 ,relheight=0.1, relx=0.35, rely=0.30)
        self.userEntryName.focus()

        self.roomLabelName = tk.Label(self.login, text="Room Id: ", font="Helvetica 12")
        self.roomLabelName.place(relheight=0.2, relx=0.1, rely=0.40)

        self.roomEntryName = tk.Entry(self.login, font="Helvetica 11", show="*")
        self.roomEntryName.place(relwidth=0.4 ,relheight=0.1, relx=0.35, rely=0.45)

        self.go = tk.Button(self.login,
                            text="CONTINUE",
                            font="Helvetica 12 bold",
                            command = lambda: self.goAhead(self.userEntryName.get(), self.roomEntryName.get()))

        self.go.place(relx=0.35, rely=0.62)

        self.Window.mainloop()
```

The `tkinter.ttk` module provides access to the Tk themed widget set, at this point, we apply this to style the button, label and texts.
Then, we start to create some functions for the chat application such as creating a chat room, sending messages and transferring files.
The `goAhead()` is used to receive the username and chat room ID from the input field and create the chat room. Client re-send to server the username and room ID entered from UI by `self.server.send(str.encode(username))` and `self.server.send(str.encode(username))`. After client get the username and room ID successfully, the `self.login.destroy()` will close the login UI alongside with the `self.layout()` to open the chatting UI.

```
def goAhead(self, username, room_id=0):
    self.name = username
    self.server.send(str.encode(username))
    time.sleep(0.1)
    self.server.send(str.encode(room_id))

    self.login.destroy()
    self.layout()

    rcv = threading.Thread(target=self.receive)
    rcv.start()
```

The function `layout()` below is used for organizing the structure and style of this application's user interface

```
def layout(self):
    self.Window.deiconify()
    self.Window.title("CHATROOM")
    self.Window.resizable(width=False, height=False)
    self.Window.configure(width=470, height=550, bg="#17202A")
    self.chatBoxHead = tk.Label(self.Window,
                                bg = "#17202A",
                                fg = "#EAECEE",
                                text = self.name ,
                                font = "Helvetica 11 bold",
                                pady = 5)

    self.chatBoxHead.place(relwidth = 1)

    self.line = tk.Label(self.Window, width = 450, bg = "#ABB2B9")

    self.line.place(relwidth = 1, rely = 0.07, relheight = 0.012)

    self.textCons = tk.Text(self.Window,
                            width=20,
                            height=2,
                            bg="#17202A",
                            fg="#EAECEE",
                            font="Helvetica 11",
                            padx=5,
                            pady=5)

    self.textCons.place(relheight=0.745, relwidth=1, rely=0.08)

    self.labelBottom = tk.Label(self.Window, bg="#ABB2B9", height=80)

    self.labelBottom.place(relwidth = 1,
                                                        rely = 0.8)

    self.entryMsg = tk.Entry(self.labelBottom,
                            bg = "#2C3E50",
                            fg = "#EAECEE",
                            font = "Helvetica 11")
    self.entryMsg.place(relwidth = 0.74,
                                                  relheight = 0.03,
                                                  rely = 0.008,
                                                  relx = 0.011)
    self.entryMsg.focus()
```

```
self.buttonMsg = tk.Button(self.labelBottom,
                                                text = "Send",
                                                font = "Helvetica 10 bold",
                                                width = 20,
                                                bg = "#ABB2B9",
                                                command = lambda : self.sendButton(self.entryMsg.get()))
self.buttonMsg.place(relx = 0.77,
                                            rely = 0.008,
                                            relheight = 0.03,
                                            relwidth = 0.22)


self.labelFile = tk.Label(self.Window, bg="#ABB2B9", height=70)

self.labelFile.place(relwidth = 1,
                                                rely = 0.9)

self.fileLocation = tk.Label(self.labelFile,
                        text = "Choose file to send",
                        bg = "#2C3E50",
                        fg = "#EAECEE",
                        font = "Helvetica 11")
self.fileLocation.place(relwidth = 0.65,
                        relheight = 0.03,
                        rely = 0.008,
                        relx = 0.011)

self.browse = tk.Button(self.labelFile,
                                                text = "Browse",
                                                font = "Helvetica 10 bold",
                                                width = 13,
                                                bg = "#ABB2B9",
                                                command = self.browseFile)
self.browse.place(relx = 0.67,
                                            rely = 0.008,
                                            relheight = 0.03,
                                            relwidth = 0.15)

self.sengFileBtn = tk.Button(self.labelFile,
                                                text = "Send",
                                                font = "Helvetica 10 bold",
                                                width = 13,
                                                bg = "#ABB2B9",
                                                command = self.sendFile)
self.sengFileBtn.place(relx = 0.84,
                                            rely = 0.008,
                                            relheight = 0.03,
                                            relwidth = 0.15)


self.textCons.config(cursor = "arrow")
scrollbar = tk.Scrollbar(self.textCons)
scrollbar.place(relheight = 1,
                                        relx = 0.974)

scrollbar.config(command = self.textCons.yview)
self.textCons.config(state = tk.DISABLED)
```

The `browseFile()` will search for the location of the file which user wants to transfer and `sendFile()` generates the sending-file process. Client gets and sends the file's name to the server by `self.server.send(str("client_" + os.path.basename(self.filename)).encode())`. Client side continually gives server the size of file by `self.server.send(str(os.path.getsize(self.filename)).encode())`.

```
def browseFile(self):
    self.filename = filedialog.askopenfilename(initialdir="/",
                                title="Select a file",
                                filetypes = (("Text files",
                                            "*.txt*"),
                                            ("all files",
                                            "*.*")))
    self.fileLocation.configure(text="File Opened: "+ self.filename)


def sendFile(self):
    self.server.send("FILE".encode())
    time.sleep(0.1)
    self.server.send(str("client_" + os.path.basename(self.filename)).encode())
    time.sleep(0.1)
    self.server.send(str(os.path.getsize(self.filename)).encode())
    time.sleep(0.1)

    file = open(self.filename, "rb")
    data = file.read(1024)
    while data:
        self.server.send(data)
        data = file.read(1024)
    self.textCons.config(state=tk.DISABLED)
    self.textCons.config(state = tk.NORMAL)
    self.textCons.insert(tk.END, "You: "
                                + str(os.path.basename(self.filename))
                                + " Sent\n\n")
    self.textCons.config(state = tk.DISABLED)
    self.textCons.see(tk.END)
```

The `sendButton()` processes the action that when we click the button “SEND” to send messages or files. When the “SEND” button is clicked, a thread is created with which the target is the function sending messages to server.

```
def sendButton(self, msg):
    self.textCons.config(state = tk.DISABLED)
    self.msg=msg
    self.entryMsg.delete(0, tk.END)
    snd= threading.Thread(target = self.sendMessage)
    snd.start()
```

The `receive()` is to receive the messages from server and display them on UI.

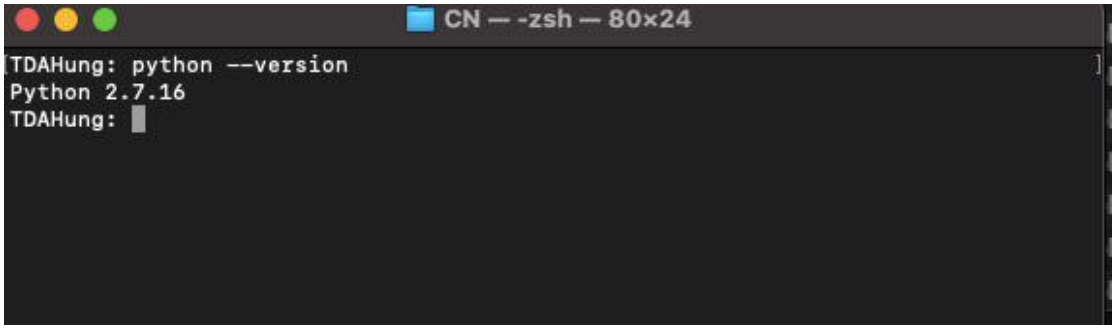
```
def receive(self):
    while True:
        try:
            message = self.server.recv(1024).decode()

            if str(message) == "FILE":
                file_name = self.server.recv(1024).decode()
                lenOfFile = self.server.recv(1024).decode()
                send_user = self.server.recv(1024).decode()

                if os.path.exists(file_name):
                    os.remove(file_name)

                total = 0
                with open(file_name, 'wb') as file:
                    while str(total) != lenOfFile:
                        data = self.server.recv(1024)
                        total = total + len(data)
                        file.write(data)

                self.textCons.config(state=tk.DISABLED)
                self.textCons.config(state = tk.NORMAL)
                self.textCons.insert(tk.END, str(send_user) + ": " + file_name + " Received\n\n")
                self.textCons.config(state = tk.DISABLED)
                self.textCons.see(tk.END)

            else:
                self.textCons.config(state=tk.DISABLED)
                self.textCons.config(state = tk.NORMAL)
                self.textCons.insert(tk.END,
                                     message+"\n\n")

                self.textCons.config(state = tk.DISABLED)
                self.textCons.see(tk.END)

        except:
            print("An error occured!")
            self.server.close()
            break
```

Lastly, the `sendMessage()` sends the messages to the server.

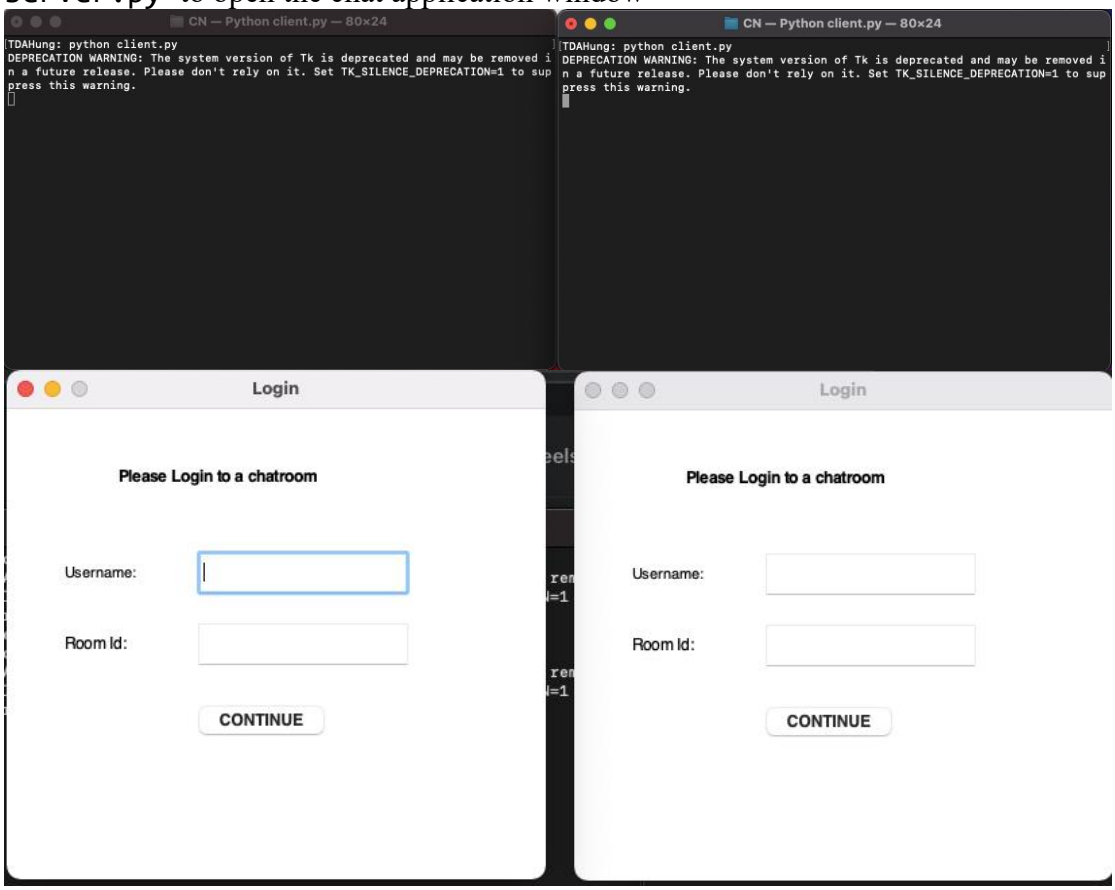
```
def sendMessage(self):
    self.textCons.config(state=tk.DISABLED)
    while True:
        self.server.send(self.msg.encode())
        self.textCons.config(state = tk.NORMAL)
        self.textCons.insert(tk.END,
                             "You: " + self.msg + "\n\n")

        self.textCons.config(state = tk.DISABLED)
        self.textCons.see(tk.END)
        break
```

6. Usage

- Download and install Python from version 2.7.0



- Turn on the cmd of your desktop and run `python client.py` and `python server.py` to open the chat application window



i