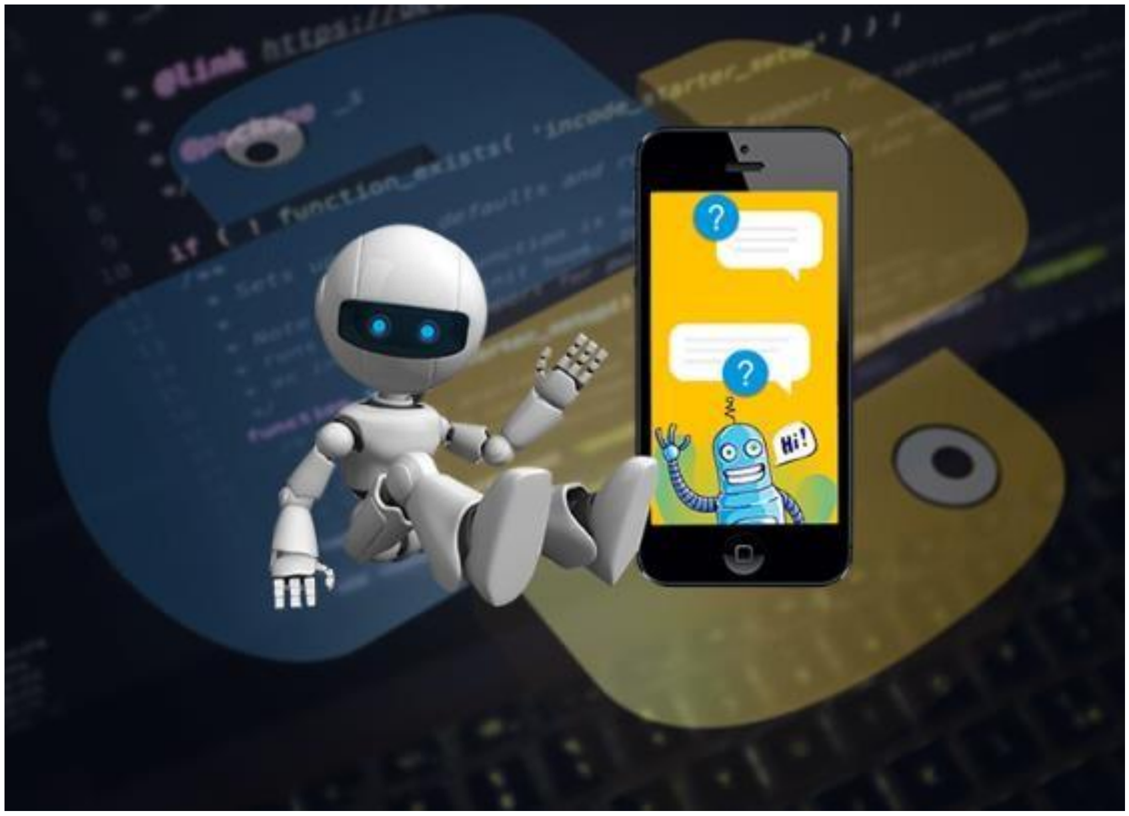
Multi-user Group Chat
GUI Application

1. Introduction

The Chat Application is very common today offered either via a web application or mobile application. Learning to write a Chat Application is good for understanding many network communication concepts and can be useful to build other network applications. Chat Application provides communication between two parties i.e. sender and receiver. The sender is someone who initiates and send a message to other known as receiver; receiver at other end, receives the message. The role of sender and receiver is not fixed and keep exchanging during communication, so in simple words, at a point, someone who sends the message is a sender and who receive the message is called receiver. In networking terms, sender and receiver are denoted as source and destination respectively. We are building a multi-user group chat application in Python. The group chat application will allow multiple users to connect to the server and chat with all other online users. The app works in a broadcast fashion. This means that messages from a user are broadcasted to other users. It is network programming (client/server application model) in Python.



2. Why Chat Application

The main reason for the development of the chat application was the growth it provided for the marketing and purchase channels. Recent times have shown the need for chat applications for mobile as 70% of the visits on the digital news comes from mobile devices. A perfect representation of mobile communication are messengers or chat apps.

3. History of Chat Apps

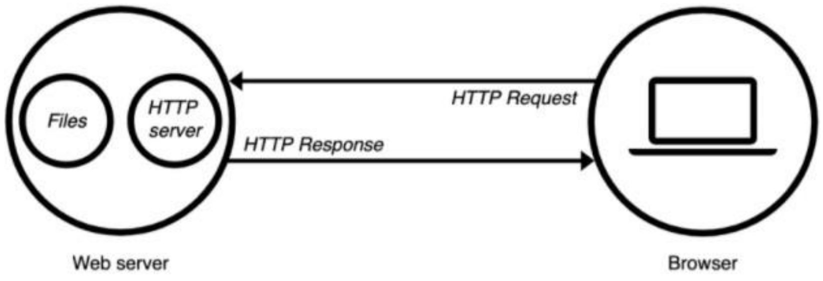
In the early 1980s, Quantum Link or Q-Link was introduced by Commodore which was an online service that enabled people to chat simultaneously along with playing games and sharing files. It further changed its name to AOL in the year 1991 and 1997, it launched the AOL instant messenger which proved to be a huge success. It was later challenged by other competitors in the early 2000's such as Skype and Yahoo chat room. The most famous chat application came in the late 2000s which is known as Blackberry Messenger and it revolutionized the world of modern chat applications.

Traditionally, instant messaging is thought to consist of one-to-one chat rather than many-to-many chat, which is called variously "groupchat" or "text conferencing". Groupchat functionality is familiar from systems such as Internet Relay Chat (IRC) and the chatroom functionality offered by popular consumer IM services. Multi user chat application is a python based application. Chat applications have been here for quite a long time, be it the ancient yahoo chat rooms or the advanced and modern chat applications such as WhatsApp, Facebook messenger, and Snapchat.

4. Description

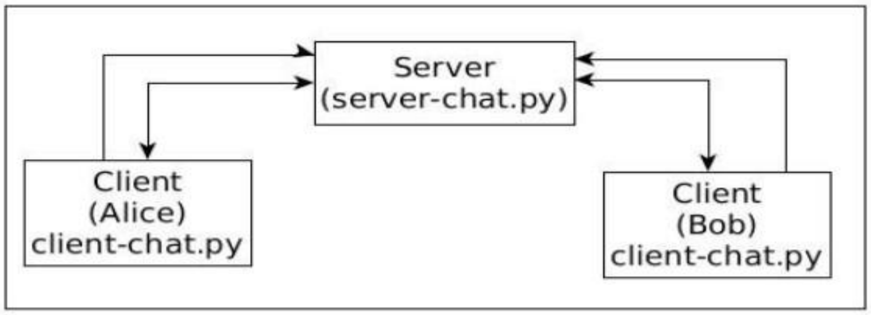
In a client-server system, the client will request for the data from the server and the server will respond by providing that particular information. For the communication between the client and the server, a connection needs to be established, which is performed by a socket.

The servers listen to a particular port and wait for the request from the client. The connection request by the client socket is allowed by the server and hence, the connection is made. For instance, a telnet server listens to port number 23. Similarly, the HTTP server listens to port number 80 and an FTP server listens to port number 21.



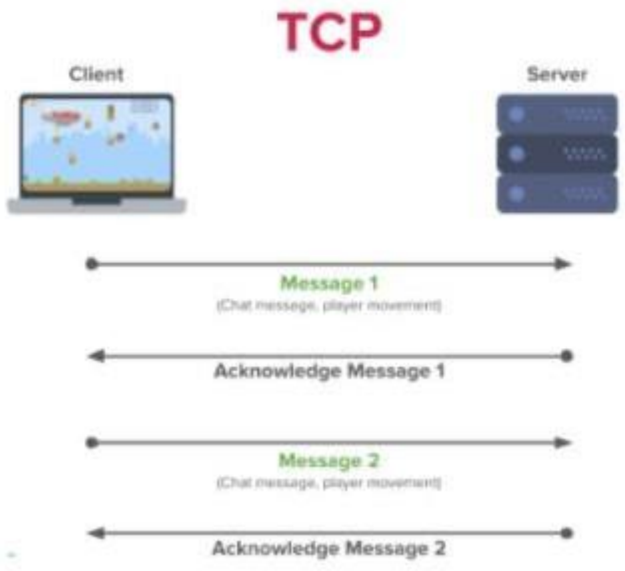
Client-Server Architecture

To create a Python Chat Application, one has to write a server program and client program/s (sender and receiver). Suppose, two parties Alice and Bob want to chat with each other and ask you to develop a chat application then being a developer you have to write a server program and a client program (different instance of the same program will be used by both Alice and Bob or even more users). Python has many modules which can help us to create network-related application, the socket is one of such popular default Python modules for low-level network programming.



4.1. Protocols

Communications over networks use what we call a protocol stack — building higher-level, more sophisticated conversations on top of simpler, more rudimentary conversations. Network layers can be represented by the OSI model and the TCP/IP model. Each network layer corresponds to a group of layer-specific network protocols. For the purpose of this application, we need not concern ourselves with many of the lower-level protocols. But we do need to know that we are using something called the Transmission Control Protocol (TCP). TCP is transport-layer protocols — which govern how data is sent from one point to another. TCP guarantees reliable delivery, without any information lost, duplicated, or out-of-order.



In our chat application, we don’t want to have to deal with lost packets because we always want to receive complete messages without any errors.

5. Multi-user Group Chat GUI Application

In this project we are building a group messaging application by using python with pyQt5.

5.1. PyQt5

There are so many options provided by Python to develop GUI application and PyQt5 is one of them. PyQt5 is cross-platform GUI toolkit, a set of python bindings for Qt v5. One can develop an interactive desktop application with so much ease because of the tools and simplicity provided by this library. A GUI application consists of Front-end and Back-end. PyQt5 has provided a tool called 'QtDesigner' to design the front-end by drag and drop method so that development can become faster and one can give more time on back-end stuff.

For installation, we need to install PyQt5 library. For this, we have typed the following command in the terminal or command prompt:

```
pip install pyqt5
```

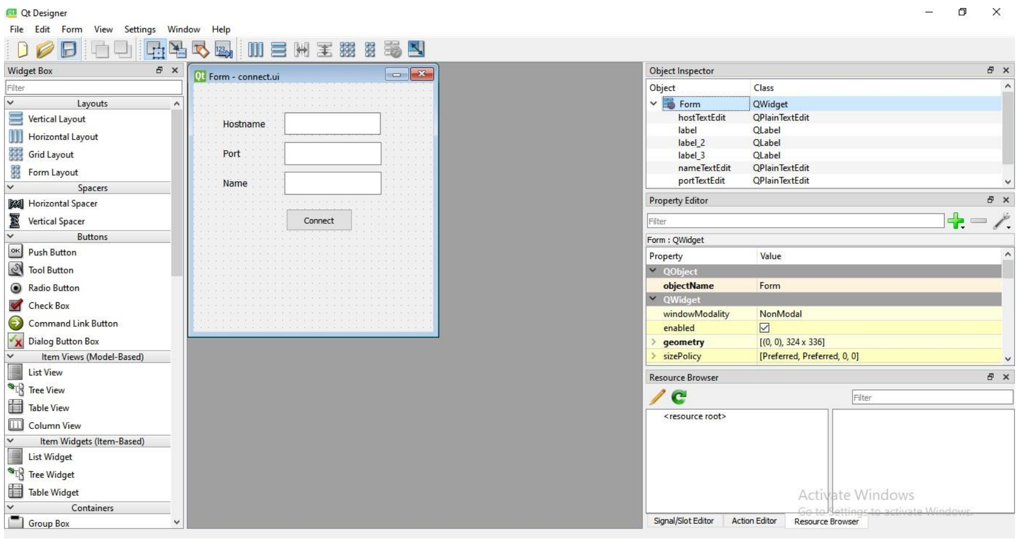
We will design the client pages in QT designer. PyQt5 provides lots of tools and QtDesigner is one of them. For this we run this command**:**

```
pip install PyQt5-tools
```

5.2. QtDesigner

Qt Designer is the Qt tool for designing and building graphical user interfaces (GUIs) with Qt Widgets. We can compose and customize our windows or dialogs in a what-you-see-is-what-you-get (WYSIWYG) manner, and test them using different styles and resolutions. Widgets and forms created with Qt Designer integrate seamlessly with programmed code, using Qt's signals and slots mechanism, so that we can easily assign behavior to graphical elements.

The client application will have two pages. First page is for connecting to the server and the second page is the messaging interface.

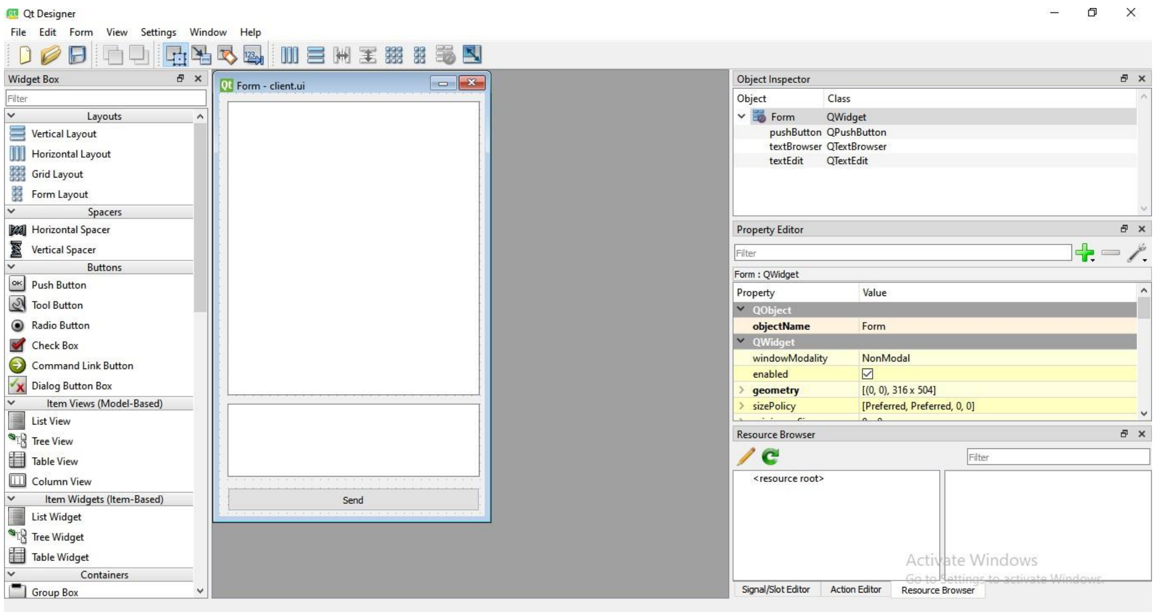


Object	Class
Form	QWidget
hostTextEdit	QPlainTextEdit
label	QLabel
label_2	QLabel
label_3	QLabel
nameTextEdit	QPlainTextEdit
portTextEdit	QPlainTextEdit

Form : QWidget

Property	Value
QObject	
objectName	Form
QWidget	
windowModality	NonModal
enabled	☑
geometry	[(0, 0), 324 x 336]
sizePolicy	[Preferred, Preferred, 0, 0]

So, once we open the application we are going to be presented with this window. For creating this window, we have used 3 labels, 3 Text Edit boxes and 1 Push Button. Here, first we will have to type hostname, port number and nickname. Then, we will be able to connect to the server after clicking ‘Connect’ button.



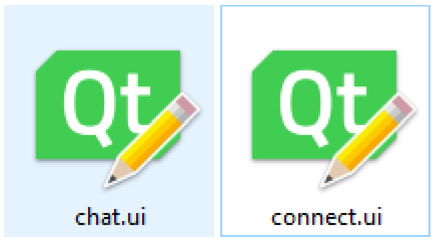
Object	Class
Form	QWidget
pushButton	QPushButton
textBrowser	QTextBrowser
textEdit	QTextEdit

Form : QWidget

Property	Value
QObject	
objectName	Form
QWidget	
windowModality	NonModal
enabled	☑
geometry	[(0, 0), 316 x 504]
sizePolicy	[Preferred, Preferred, 0, 0]

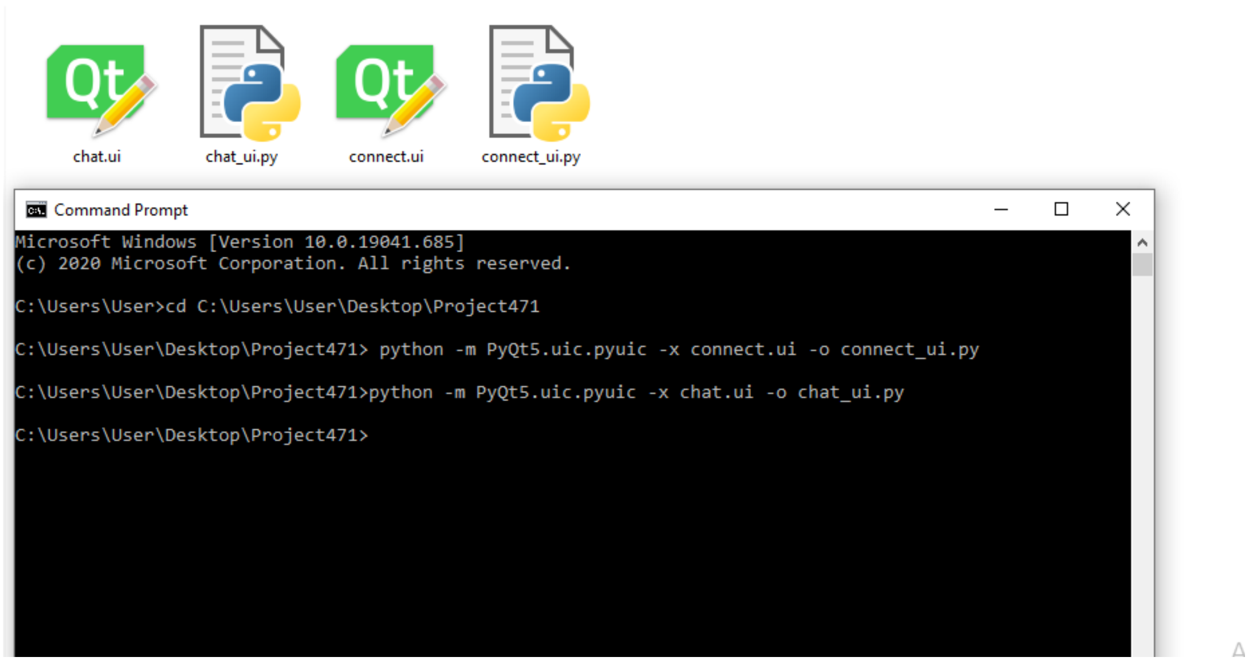
This is where we will see the messages that client have received and also the messages that we are sending and here we will be able to type up our messages. For creating this page we have used 1 Text browser box, 1 Text edit box and 1 Push button. After typing we can send message by clicking ‘Send’ button.

After designing the pages in Qt designer we will end up with two ui files.



To work with those files, we have to convert them into python files. For that, we run this command:

```
python -m PyQt5.uic.pyuic -x connect.ui -o connect_ui.py
python -m PyQt5.uic.pyuic -x chat.ui -o chat_ui.py
```



```
Microsoft Windows [Version 10.0.19041.685]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\User>cd C:\Users\User\Desktop\Project471

C:\Users\User\Desktop\Project471> python -m PyQt5.uic.pyuic -x connect.ui -o connect_ui.py

C:\Users\User\Desktop\Project471>python -m PyQt5.uic.pyuic -x chat.ui -o chat_ui.py

C:\Users\User\Desktop\Project471>
```

5.3. Server.py

- **Import Libraries**

At first, we have done all our imports.

```
import socket
import threading
```

Socket: Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server. They are the real backbones behind web browsing. In simpler terms there is a server and a client. Socket programming is started by importing the socket library and making a simple socket.

```
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Thread: A thread is an entity within a process that can be scheduled for execution. Also, it is the smallest unit of processing that can be performed in an OS (Operating System). In simple words, a thread is a sequence of such instructions within a program that can be executed independently of other code. For simplicity, you can assume that a thread is simply a subset of a process.

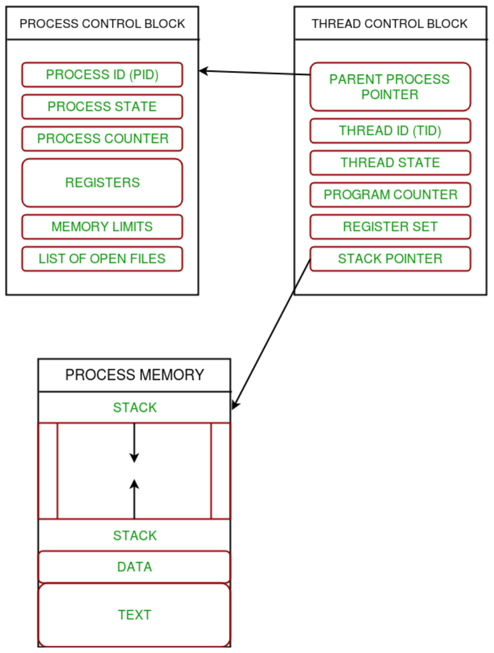


Figure: Understand the relation between process and its thread

Multithreading: Multiple threads can exist within one process where:

- Each thread contains its own register set and local variables (stored in stack).
- All threads of a process share global variables (stored in heap) and the program code.

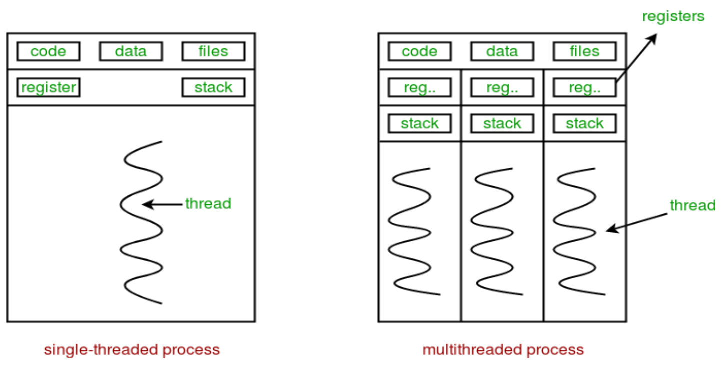
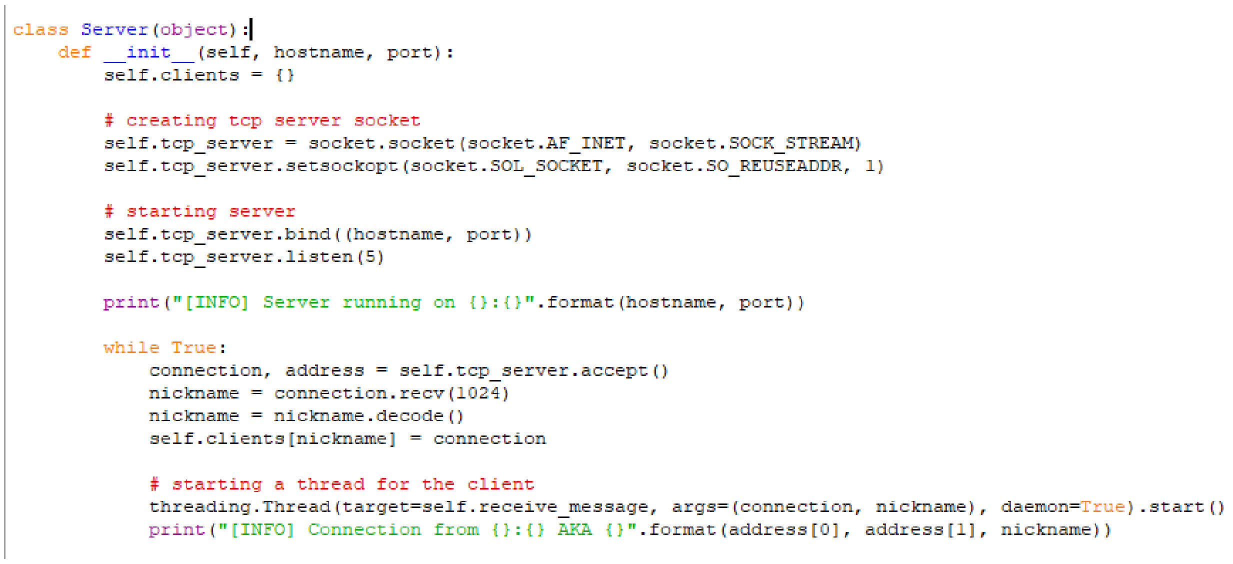


Figure: Understand how multiple threads exist in memory.

- **Server class**

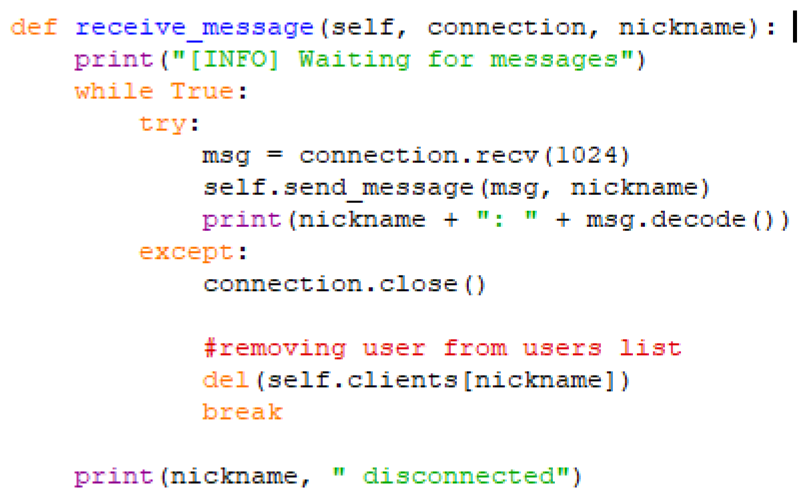
We have created a class called server. The constructor for the class will take two arguments the host name and the port number that the server will run on. Then we have created a TCP socket for our server and set a flag that allows for the application to reuse the same host name and port number. Then we have used the bind function to bind the socket to the hostname and port number and then started listening for a connection. In the main thread we are keeping on accepting connections from clients. Whenever a client connects, it will store the client's details and also spin off a thread for that client.



```
class Server(object):
    def __init__(self, hostname, port):
        self.clients = {}

        # creating tcp server socket
        self.tcp_server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.tcp_server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

        # starting server
        self.tcp_server.bind((hostname, port))
        self.tcp_server.listen(5)

        print("[INFO] Server running on {}:{}".format(hostname, port))

        while True:
            connection, address = self.tcp_server.accept()
            nickname = connection.recv(1024)
            nickname = nickname.decode()
            self.clients[nickname] = connection

            # starting a thread for the client
            threading.Thread(target=self.receive_message, args=(connection, nickname), daemon=True).start()
            print("[INFO] Connection from {}:{} AKA {}".format(address[0], address[1], nickname))
```

- **After receiving message from client**

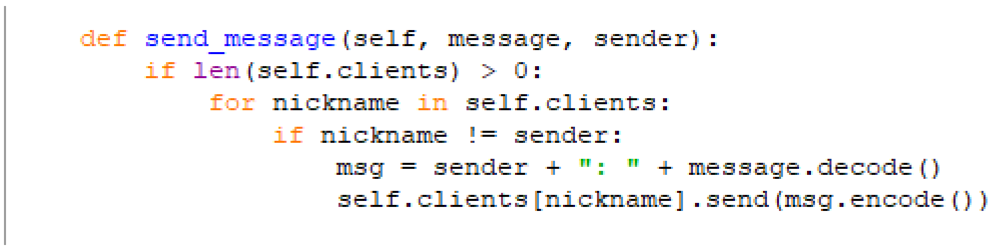
We have implemented receive_message() method. The receive message method will take two arguments. In an infinite loop we will try to receive messages from a client. If an error occurs it means that the client has disconnected from the server. If a client has disconnected will remove them from the list of clients.



```
def receive_message(self, connection, nickname):
    print("[INFO] Waiting for messages")
    while True:
        try:
            msg = connection.recv(1024)
            self.send_message(msg, nickname)
            print(nickname + ": " + msg.decode())
        except:
            connection.close()

            #removing user from users list
            del(self.clients[nickname])
            break

    print(nickname, " disconnected")
```

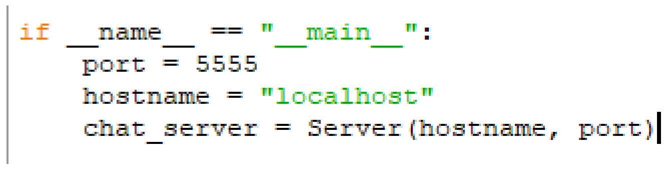
- **<u>Sending message to other clients</u>**

We have also created a method to broadcast messages that have been received by the server. The method will send message to all other clients expect the person that sent the message.



```
def send_message(self, message, sender):
    if len(self.clients) > 0:
        for nickname in self.clients:
            if nickname != sender:
                msg = sender + ": " + message.decode()
                self.clients[nickname].send(msg.encode())
```

The server will run on local host and on port 5555**.**



```
if __name__ == "__main__":
    port = 5555
    hostname = "localhost"
    chat_server = Server(hostname, port)
```

.

We should change localhost in server.py to 0.0.0.0 to be able to accept connections from other computers.

5.4. Client.py

- <u>**Import libraries**</u>

To create the client application at first, we took all our imports.

```
from PyQt5 import QtCore, QtWidgets

import client_ui
import connect_ui

import sys
import socket
import random
```

QtCore, QtWidgets: We import QtCore, QtWidgets from PyQt5. Qt Widgets module provides a set of UI elements to create classic desktop-style user interfaces and the **QtCore** module contains the core classes, including the event loop and Qt's signal and slot mechanism, regular expressions, and user and application settings.

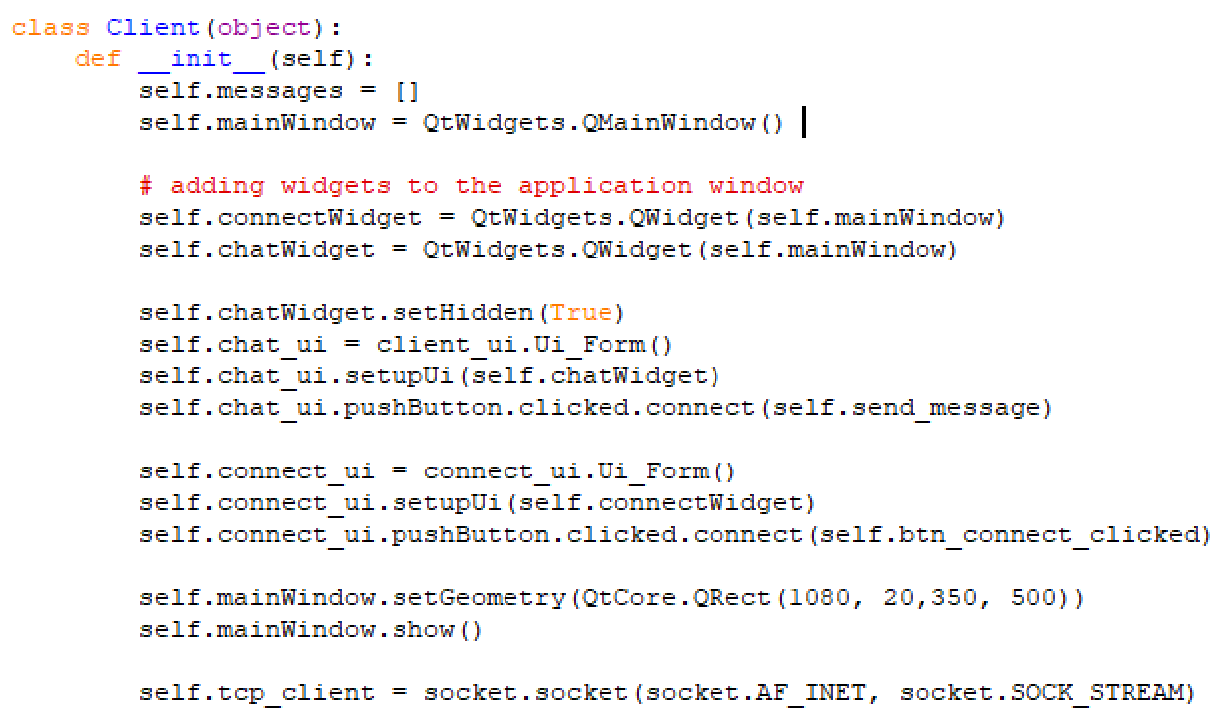
Random: Python offers random module that can generate random numbers. random.randint(a,b) returns a random integer between a and b inclusive.

Sys: The python sys module provides functions and variables which are used to manipulate different parts of the Python Runtime Environment. It lets us access system-specific parameters and functions. sys.argv() function returns a list of command line arguments passed to a Python script. The name of the script is always the item at index 0, and the rest of the arguments are stored at subsequent indices.

We also import client_ui, connect_ui. We also import socket to connect to the client with server.

- <u>**Client class**</u>

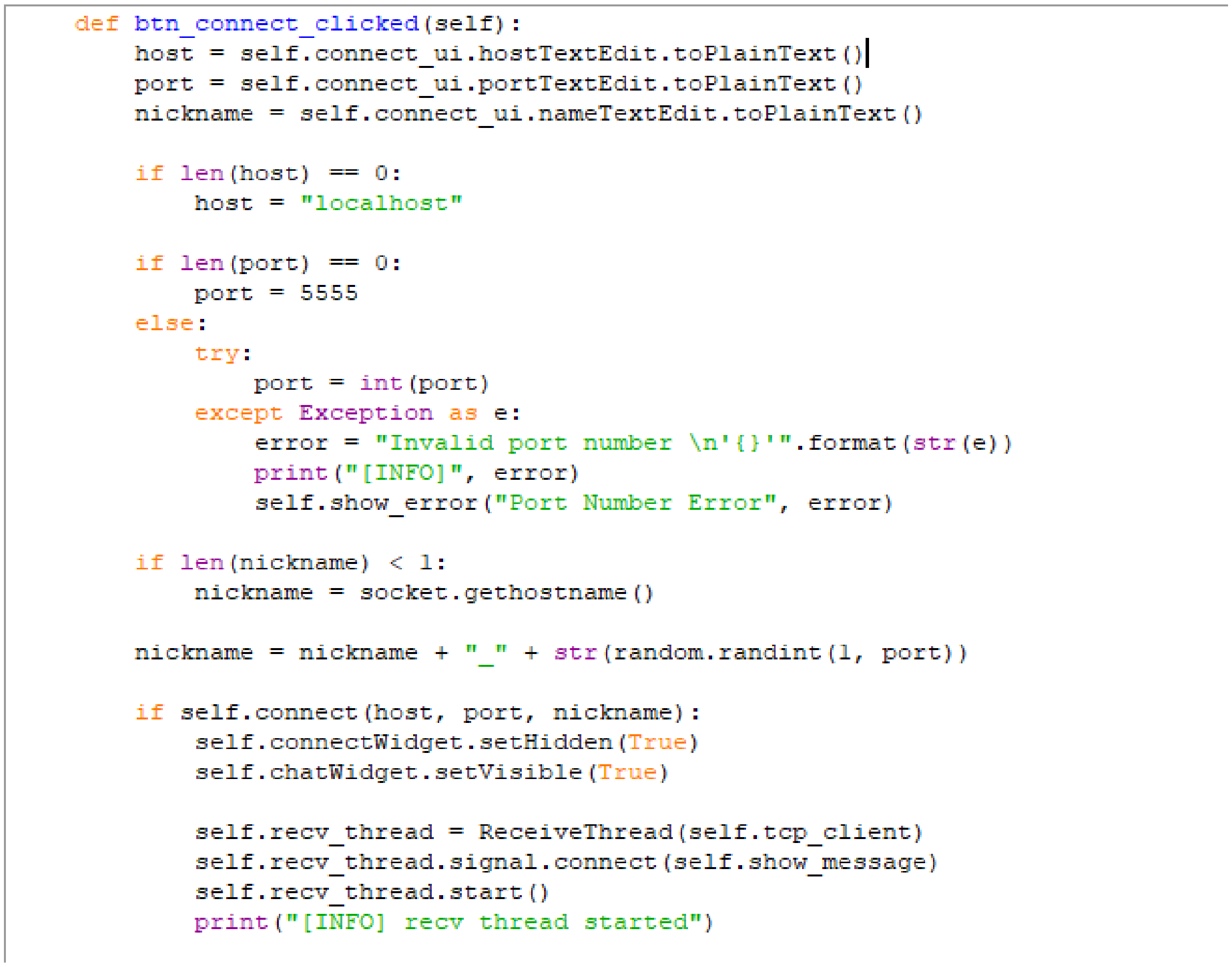
We have created a class called client and it have a couple of methods. In this class there are some regular class function and also some slot function that gets called whenever an event is triggered from the interface. Events could be pressing a button or clicking a button. So, we have created the main window for our application and also widgets for the pages that we have designed. Since only one page will be shown at a time, we hide the messaging page until the user is successfully connected to the server. We have registered send_message() function to the send button signal. We add the pages that we designed to their corresponding widgets. We also link a method to one of our buttons. When the connect button is clicked the btn_connect_clicked() function method gets called. We have also resized the application window, that's because we didn't specify the window size for the main window. Then we showed our main window. After that's done, we create a TCP socket for our client.



```
class Client(object):
    def __init__(self):
        self.messages = []
        self.mainWindow = QtWidgets.QMainWindow()

        # adding widgets to the application window
        self.connectWidget = QtWidgets.QWidget(self.mainWindow)
        self.chatWidget = QtWidgets.QWidget(self.mainWindow)

        self.chatWidget.setHidden(True)
        self.chat_ui = client_ui.Ui_Form()
        self.chat_ui.setupUi(self.chatWidget)
        self.chat_ui.pushButton.clicked.connect(self.send_message)

        self.connect_ui = connect_ui.Ui_Form()
        self.connect_ui.setupUi(self.connectWidget)
        self.connect_ui.pushButton.clicked.connect(self.btn_connect_clicked)

        self.mainWindow.setGeometry(QtCore.QRect(1080, 20,350, 500))
        self.mainWindow.show()

        self.tcp_client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

- **<u>After clicking the connect button</u>**

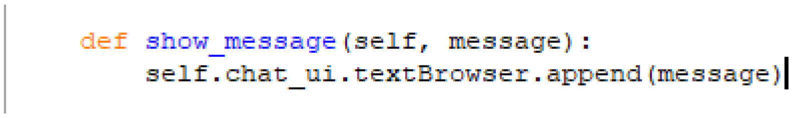
In btn_connect_clicked() method we undo connecting to the server. So, we have retrieved the server's host name, port number that the server application is running on and also the nick name that the client wants to use. Then we have handled cases when the user doesn't provide any information. For the host name the default will be local host. For port number it's going to be 5555 and nick name will be the host name of the client with some added randomness. After getting all the information we have connected to the server. If the connection is successful, we have hidden the connection page and go to the messaging page. Here we start the message receiving thread right after the connecting to the server. We also set the slot for the signal in the message receiving thread. So, whenever a message is received the show_message() function method will get called from the main thread and the message that was received will get passed as an argument.



```
    def btn_connect_clicked(self):
        host = self.connect_ui.hostTextEdit.toPlainText()
        port = self.connect_ui.portTextEdit.toPlainText()
        nickname = self.connect_ui.nameTextEdit.toPlainText()

        if len(host) == 0:
            host = "localhost"

        if len(port) == 0:
            port = 5555
        else:
            try:
                port = int(port)
            except Exception as e:
                error = "Invalid port number \n'{}'".format(str(e))
                print("[INFO]", error)
                self.show_error("Port Number Error", error)

        if len(nickname) < 1:
            nickname = socket.gethostname()

        nickname = nickname + "_" + str(random.randint(1, port))

        if self.connect(host, port, nickname):
            self.connectWidget.setHidden(True)
            self.chatWidget.setVisible(True)

            self.recv_thread = ReceiveThread(self.tcp_client)
            self.recv_thread.signal.connect(self.show_message)
            self.recv_thread.start()
            print("[INFO] recv thread started")
```

- **show_message()**

In the show_message() methods we just append the message that we received to the message history box by using append function.



```
    def show_message(self, message):
        self.chat_ui.textBrowser.append(message)
```

- **connect()**

In connect() function, we send the clients nickname to the server after connecting to the server. So that messages from that client can be tagged appropriately. We make the connect method return true to be able to test switching between pages of the application.

```
def connect(self, host, port, nickname):
    try:
        self.tcp_client.connect((host, port))
        self.tcp_client.send(nickname.encode())
        print("[INFO] Connected to server")
        return True

    except Exception as e:
        error = "Unable to connect to server \n'{}'".format(str(e))
        print("[INFO]", error)
        self.show_error("Connection Error", error)
        self.connect_ui.hostTextEdit.clear()
        self.connect_ui.portTextEdit.clear()
        return False
```

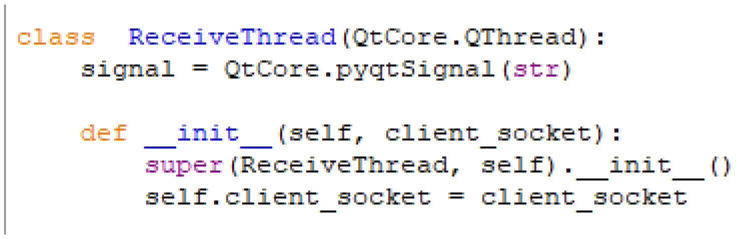
- **<u>Send message to the server</u>**

The method send_message() function will get called whenever we click the send button. This method will get the text, the users typed and send it to the server. We add some code to be able to see to what we have sent in the message history box. We also delete the typing box once the message has been sent.

```
def send_message(self):
    message = self.chat_ui.textEdit.toPlainText()
    self.chat_ui.textBrowser.append("Me: " + message)
    print("sent: " + message)

    try:
        self.tcp_client.send(message.encode())
    except Exception as e:
        error = "Unable to send message '{}'".format(str(e))
        print("[INFO]", error)
        self.show_error("Server Error", error)
    self.chat_ui.textEdit.clear()
```

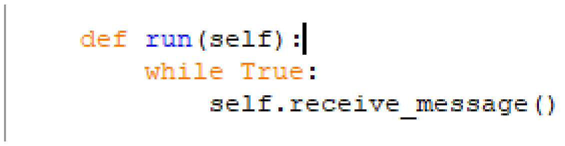
- **<u>ReceiveThread()</u>**

In ReceiveThread() class, to make the clients receive messages, the client will create a thread that will receive messages from the server. After receiving the message, we have appended it to the message history box by using append function. We have used QThread and the signal slot mechanism. We create a new class that receives a socket as the constructor’s argument.



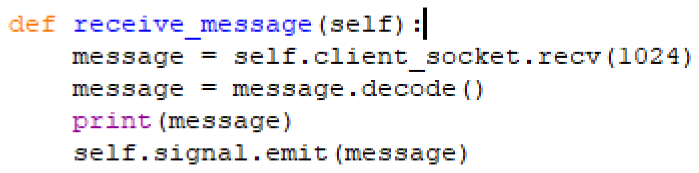
```python
class  ReceiveThread(QtCore.QThread):
    signal = QtCore.pyqtSignal(str)

    def __init__(self, client_socket):
        super(ReceiveThread, self).__init__()
        self.client_socket = client_socket
```

Then we create a Py-qt signal. The run() method will get called when the thread is started.



```python
    def run(self):
        while True:
            self.receive_message()
```

Our receive_message() method will handle receiving messages and sending signals.

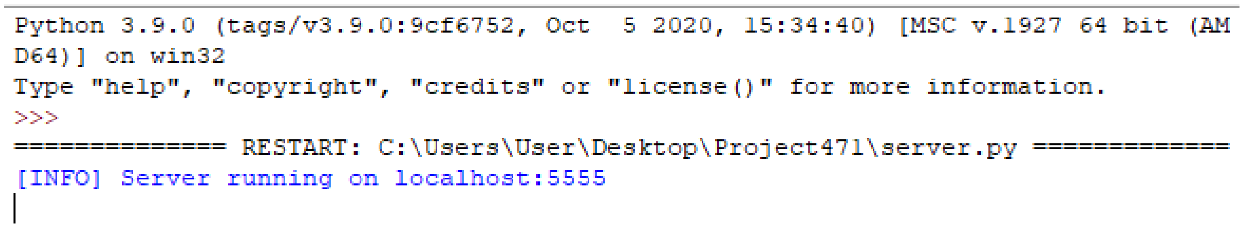


```python
    def receive_message(self):
        message = self.client_socket.recv(1024)
        message = message.decode()
        print(message)
        self.signal.emit(message)
```

Then finally all the client can send and receive messages.

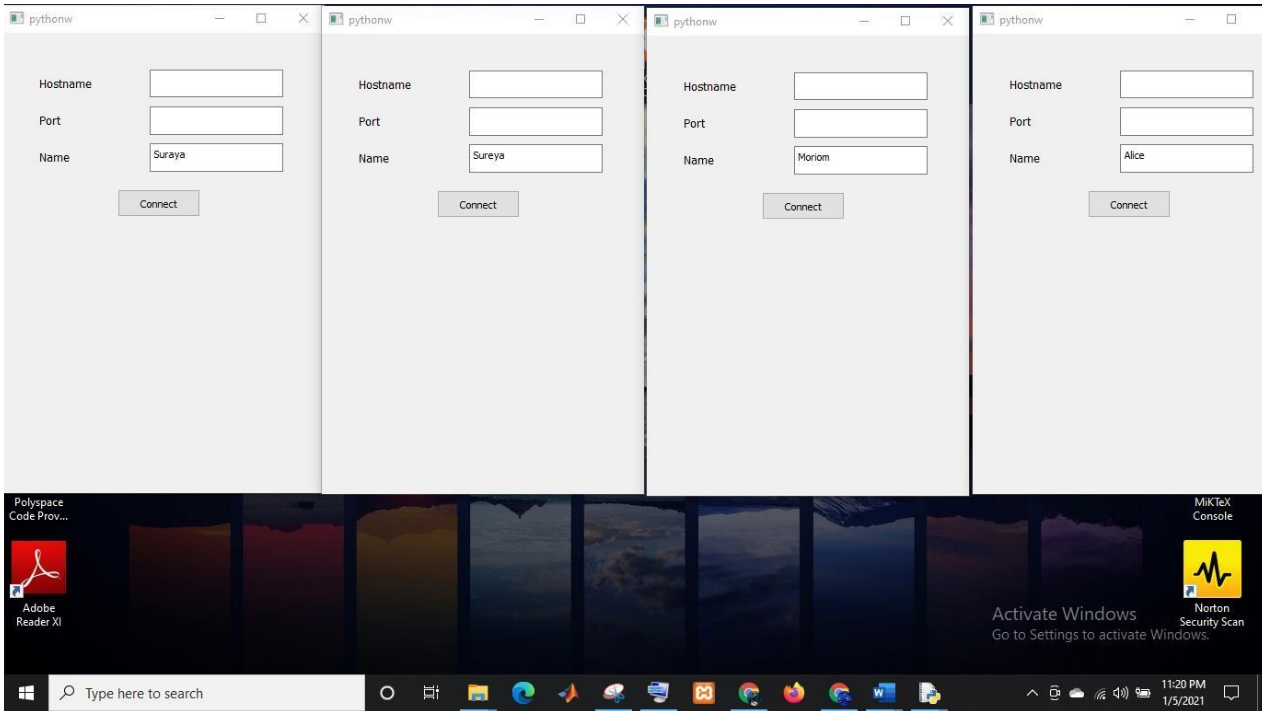
6. After running the application

After running the server.py, it is waiting for client response



```
Python 3.9.0 (tags/v3.9.0:9cf6752, Oct  5 2020, 15:34:40) [MSC v.1927 64 bit (AM
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: C:\Users\User\Desktop\Project471\server.py =============
[INFO] Server running on localhost:5555
```

After running multiple clients, the application will be started and we will type the nicknames



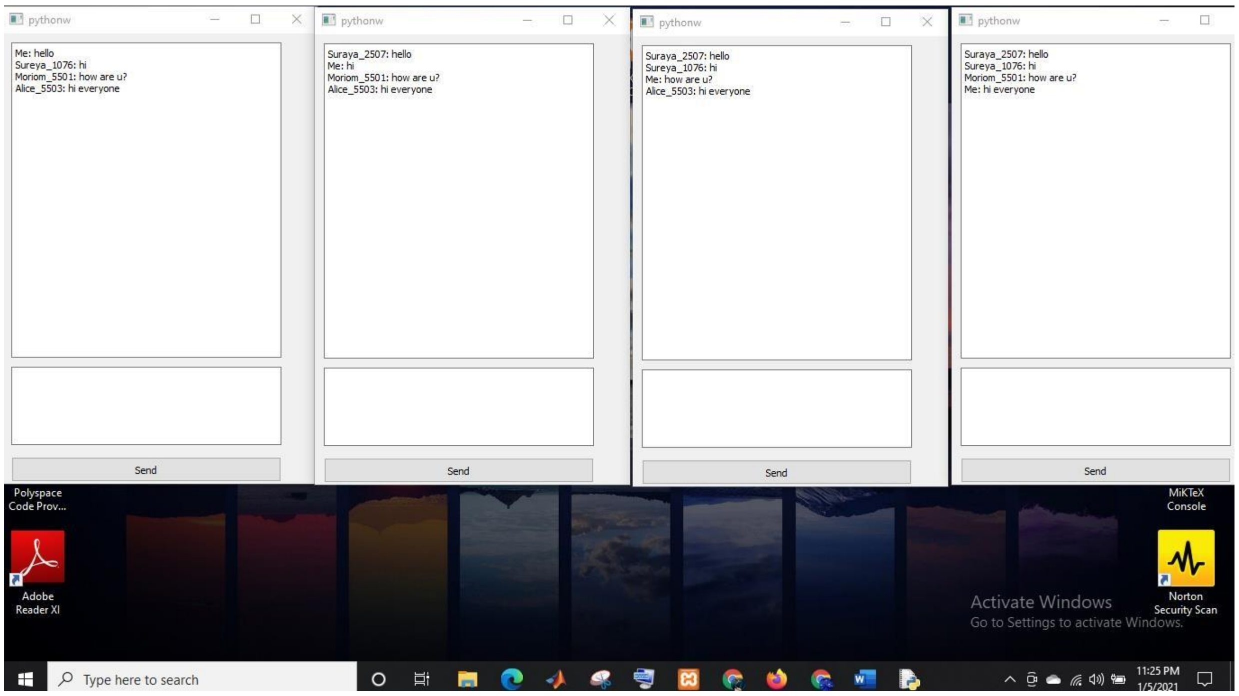
Name: Suraya

Name: Sureya

Name: Moriom

Name: Alice

After clicking the connect button, we can type messages and send it to others by send button.



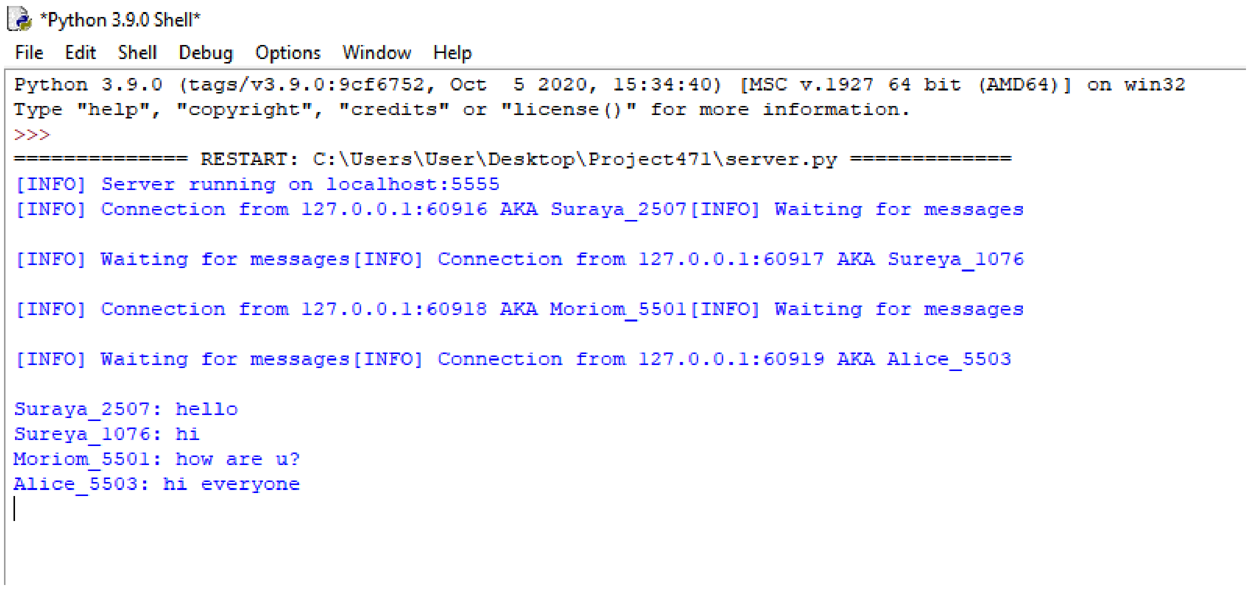
```
Me: hello
Sureya_1076: hi
Moriom_5501: how are u?
Alice_5503: hi everyone
```

```
Suraya_2507: hello
Me: hi
Moriom_5501: how are u?
Alice_5503: hi everyone
```

```
Suraya_2507: hello
Sureya_1076: hi
Me: how are u?
Alice_5503: hi everyone
```

```
Suraya_2507: hello
Sureya_1076: hi
Moriom_5501: how are u?
Me: hi everyone
```

After running the application, the server response



```
Python 3.9.0 (tags/v3.9.0:9cf6752, Oct  5 2020, 15:34:40) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: C:\Users\User\Desktop\Project471\server.py =============
[INFO] Server running on localhost:5555
[INFO] Connection from 127.0.0.1:60916 AKA Suraya_2507[INFO] Waiting for messages

[INFO] Waiting for messages[INFO] Connection from 127.0.0.1:60917 AKA Sureya_1076

[INFO] Connection from 127.0.0.1:60918 AKA Moriom_5501[INFO] Waiting for messages

[INFO] Waiting for messages[INFO] Connection from 127.0.0.1:60919 AKA Alice_5503

Suraya_2507: hello
Sureya_1076: hi
Moriom_5501: how are u?
Alice_5503: hi everyone
```

7. Limitations

There are some limitations in our application. First it is not nicely designed. And we can only send text messages. No emojis or files or pictures or videos.

8. Future Work

Right now, we are just dealing with text communication in future the application may be extended to include features such as:

- **File transfer:** This will enable the user to send files to different formats to others riyada chat application
- **Voice chat:** This will enhance the application to a higher level via communication will be possible via voice calling in telephone
- **Video chat:** This will further enhance the feature of calling into video communication.

9. Conclusion

Chatting is a very common used application among the users. General users use the instant messaging services to communicate with other individual users. Though our application has limitations; after adding some features, people could have a better experience of chatting with our application. We hope the app will run fine if the application is on different machines.

10. APPENDIX

Server

```
import socket
import threading


class Server(object):
    def___init_(self, hostname, port):
        self.clients = {}
        # creating server socket
        self.tcp_server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.tcp_server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        # starting server
        self.tcp_server.bind((hostname, port))
        self.tcp_server.listen(5)
        print("[INFO] Server running on {}:{}".format(hostname, port))
        while True:
            connection, address = self.tcp_server.accept()
            nickname = connection.recv(1024)
            nickname = nickname.decode()
            self.clients[nickname] = connection
            # starting a thread for the client
            threading.Thread(target=self.receive_message, args=(connection, nickname),
daemon=True).start()
            print("[INFO] Connection from {}:{} AKA {}".format(address[0], address[1], nickname))


    def receive_message(self, connection, nickname):
        print("[INFO] Waiting for messages")
        while True:
            try:
                msg = connection.recv(1024)
                self.send_message(msg, nickname)
                print(nickname + ": " + msg.decode())
            except:
                connection.close()
                #removing user from users list
                del(self.clients[nickname])
                break
        print(nickname, " disconnected")

    def send_message(self, message, sender):
        if len(self.clients) > 0:
            for nickname in self.clients:
                if nickname != sender:
                    msg = sender + ": " + message.decode()
                    self.clients[nickname].send(msg.encode())

if__name__== "_main_":
    port = 5555
    hostname = "localhost"
    chat_server = Server(hostname, port)
```

Client

```
from PyQt5 import QtCore, QtWidgets

import client_ui

import connect_ui

import sys

import socket

import random


class ReceiveThread(QtCore.QThread):

    signal = QtCore.pyqtSignal(str)

    def___init_(self, client_socket):

        super(ReceiveThread, self)._init_()

        self.client_socket = client_socket

    def run(self):

        while True:

            self.receive_message()

    def receive_message(self):

        message = self.client_socket.recv(1024)

        message = message.decode()

        print(message)

        self.signal.emit(message)


class Client(object):

    def___init_(self):

        self.messages = []

        self.mainWindow = QtWidgets.QMainWindow()

        # adding widgets to the application window

        self.connectWidget = QtWidgets.QWidget(self.mainWindow)
        self.chatWidget = QtWidgets.QWidget(self.mainWindow)
        self.chatWidget.setHidden(True)
        self.chat_ui = client_ui.Ui_Form()
        self.chat_ui.setupUi(self.chatWidget)
        self.chat_ui.pushButton.clicked.connect(self.send_message)
        self.connect_ui = connect_ui.Ui_Form()
        self.connect_ui.setupUi(self.connectWidget)
        self.connect_ui.pushButton.clicked.connect(self.btn_connect_clicked)
        self.mainWindow.setGeometry(QtCore.QRect(1080, 20,350, 500))
        self.mainWindow.show()
        self.tcp_client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    def btn_connect_clicked(self):
        host = self.connect_ui.hostTextEdit.toPlainText()
        port = self.connect_ui.portTextEdit.toPlainText()
        nickname = self.connect_ui.nameTextEdit.toPlainText()
        if len(host) == 0:
            host = "localhost"
        if len(port) == 0:
            port = 5555
        else:
            try:
                port = int(port)
            except Exception as e:
                error = "Invalid port number \n'{}'".format(str(e))
                print("[INFO]", error)
                self.show_error("Port Number Error", error)
        if len(nickname) < 1:
            nickname = socket.gethostname()
        nickname = nickname + "_" + str(random.randint(1, port))
        if self.connect(host, port, nickname):
            self.connectWidget.setHidden(True)
            self.chatWidget.setVisible(True)
            self.recv_thread = ReceiveThread(self.tcp_client)
            self.recv_thread.signal.connect(self.show_message)
            self.recv_thread.start()
            print("[INFO] recv thread started")

    def show_message(self, message):
        self.chat_ui.textBrowser.append(message)

    def connect(self, host, port, nickname):
        try:
            self.tcp_client.connect((host, port))
            self.tcp_client.send(nickname.encode())
            print("[INFO] Connected to server")
            return True
        except Exception as e:
            error = "Unable to connect to server \n'{}'".format(str(e))
            print("[INFO]", error)
            self.show_error("Connection Error", error)
            self.connect_ui.hostTextEdit.clear()
            self.connect_ui.portTextEdit.clear()
            return False

    def send_message(self):
        message = self.chat_ui.textEdit.toPlainText()
        self.chat_ui.textBrowser.append("Me: " + message)
        print("sent: " + message)
        try:
            self.tcp_client.send(message.encode())
        except Exception as e:
            error = "Unable to send message '{}'".format(str(e))
            print("[INFO]", error)
            self.show_error("Server Error", error)
        self.chat_ui.textEdit.clear()

    def show_error(self, error_type, message):
        errorDialog = QtWidgets.QMessageBox()
        errorDialog.setText(message)
        errorDialog.setWindowTitle(error_type)
        errorDialog.setStandardButtons(QtWidgets.QMessageBox.Ok)
        errorDialog.exec_()

if__name__== "_main__":
    app = QtWidgets.QApplication(sys.argv)
    c = Client()
    sys.exit(app.exec())
```