

University for Business and Technology in Kosovo

UBT Knowledge Center

UBT International Conference

2021 UBT International Conference

Oct 30th, 9:00 AM - 10:30 AM

Workplace Chat Application Using Socket Programming in Python

Egzon Salihu

University for Business and Technology - UBT, es44550@ubt-uni.net

Gentiana Blakaj

University for Business and Technology - UBT, gb46704@ubt-uni.net

Follow this and additional works at: <https://knowledgecenter.ubt-uni.net/conference>



Part of the [Computer Sciences Commons](#)

Recommended Citation

Salihu, Egzon and Blakaj, Gentiana, "Workplace Chat Application Using Socket Programming in Python" (2021). *UBT International Conference*. 388.

<https://knowledgecenter.ubt-uni.net/conference/2021UBTIC/all-events/388>

This Event is brought to you for free and open access by the Publication and Journals at UBT Knowledge Center. It has been accepted for inclusion in UBT International Conference by an authorized administrator of UBT Knowledge Center. For more information, please contact knowledge.center@ubt-uni.net.

Workplace Chat Application Using Socket Programming in Python

Egzon Salihu^{1*}, Gentiana Blakaj²

UBT- Higher Education Institution, Kosovo, Prishtina

¹es44550@ubt-uni.net ²gb46704@ubt-uni.net

Abstract. A chat is a real-time communication with one or more users connected to the internet. Chat applications are very necessary at this time in a workplace to communicate employees with each other for company purposes. In Python programming, this type of communication is possible to do using the library sockets, which enables connecting two nodes on a network to communicate with each other. We represent in this paper how to build a chat application that uses a server to connect multiple users and let them communicate with each other using the TCP protocol. Threads are used for parallel programming to send and receive messages in real-time which makes this chat application very useful even if there are a lot of clients connected at the same time.

Keywords: Client, Server, Sockets, Communication, Chat Application

1 Introduction

A chat is a text-based communication via keyboard or any input device. When talking to someone in a chat any typed text is received by other participants in real-time over the internet or on a local network [2,3]. Recently, there is a growing need to communicate with colleagues during working hours to answer questions, get support, or achieve greater efficiency in performing various tasks, so the idea came to us to create this simple application, which requires minimal resources and is easy to use from everyone[1,6]. To achieve that we used Python because it is a great programming language for computer networking also has a library called sockets where one node listens on a port at IP address, and the other node reaches out to the other so together will create an active connection[4]. Sockets were invented as a part of the BSD flavor of Unix in Berkeley and are used nearly everywhere because everything you need to know is the IP address of the server and the port number[5]. In this paper, we will explain how socket programming works and how we used it to create our application.

2 Design and Implementation

For the design of our application, we have used the Client-Server architecture, where a lot of clients can connect to the server, request and receive service, and communicate with each other[7]. In figure 1 we represent how client-server architecture will look like.

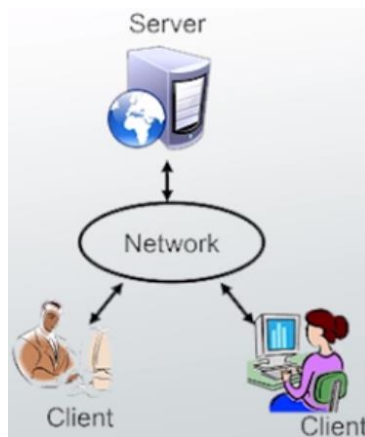


Fig. 1. Client-Server architecture[2]

To understand how socket programming works between the server and clients in Python several components are important, so we will explain below socket functions and methods:

- `socket()`, here we specify if we use IPv4 or IPv6 internet protocol, also TCP or UDP protocol.
- `bind()`, will assign an IP address and a port number.
- `listen()`, the server will listen if there is any connection from clients.
- `accept()`, when a client connects server calls this.
- `connect()`, client will initiate the three-way handshake with the server.
- `send()`, data is send to client/server.
- `recv()`, data is receive from client/server.
- `close()` both the server and clients close the connection[9].

In the figure 2 below we represent how the server and client communicate with each other using TCP protocol.

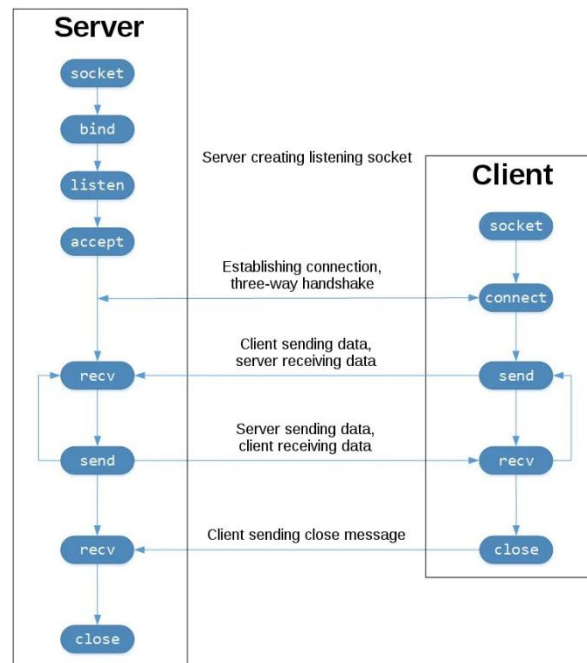
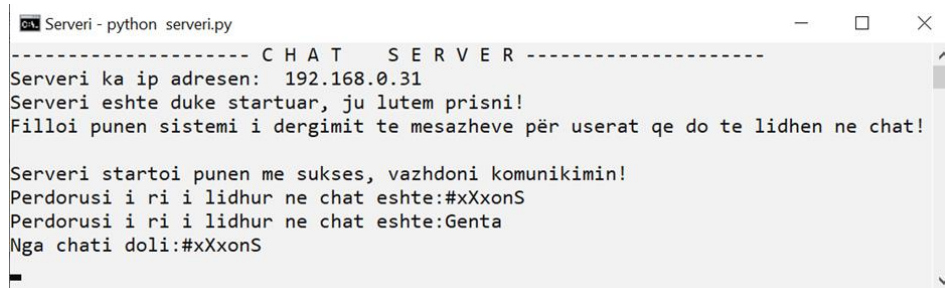


Fig. 2. Communication between client and the server using TCP protocol[8]

As we see in figure 2 both the server on the left and the client on the right has a socket because in the socket is stored the IP address and the port number. The entered IP address and the port number of the client should be the same as the server. In the Bind will be stored the IP address and the port number. Now the server will listen if is any connection from clients. When clients get connected, the server will immediately call accept, and the client will call connect, because we are using TCP in this step will happen three-way handshake between the server and a client so our connection is reachable over the network. When a client sends data to the server, in this case, a client will call send, the server will call recv, and vice-versa. If a client wants to terminate the connection it will send a close message, and the server will immediately call close. So, in this way is made the connection and transfer data between the server and the client in socket programming in python[10]. Our project is realized in two modules, server.py and the module chati.py, both modules are using Program Oriented Programming which are involved Abstract classes, Methods, Functions, Inheritance (Hierarchical), Encapsulation (private variable, protected variable), Polymorphism (Method overloading), and Threads.

2.1 Code snippets about Server part

In this section we will represent our server side that we implemented. In figure 3 is shown the module server.py after it is executed.



```
Serveri - python serveri.py
----- C H A T S E R V E R -----
Serveri ka ip adresen: 192.168.0.31
Serveri eshte duke startuar, ju lutem prisni!
Filloi punen sistemi i dergimit te mesazheve për userat qe do te lidhen ne chat!

Serveri startoi punen me sukses, vazhdoni komunikimin!
Perdorusi i ri i lidhur ne chat eshte:#xXxonS
Perdorusi i ri i lidhur ne chat eshte:Genta
Nga chati doli:#xXxonS
```

Fig. 3. Executed form of the module Serveri.py

When the server executes as in figure 3 it will show the IP address of the server, also it will display all the information's like:

Server has IP address: 192.168.0.31

Server is starting, please wait!

The messaging system for users who will be connected in chat started working!

Server has successfully started working, proceed with communication!

When a new client is connected to the server it will show:

New user connected to the chat is: #xXxonS

New user connected to the chat is: Genta

When a client leaves the chat, it will show:

The Chat left: #xXxonS

In the background will always create a file log so we can see them if we want.

The Python code for the module Serveri.py is:

```
import threading
import socket
from abc import ABC, abstractmethod
import time
```

Above we import the library called threading, socket, and ABC which stands for Abstract Classes.

```
print("----- C H A T S E R V E R -----")
class Serveri(ABC):
    __nrportit = 16000
    __ip=socket.gethostbyname(socket.gethostname())
    #__ip="5.206.239.54"
    print("Serveri ka ip adresen: ",__ip)
    time.sleep(1.2)
```

```

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((__ip, __nrportit))
s.listen()
pseudonimet= []
@abstractmethod
def njoftimi(self):
    pass
perdoruesit = []

```

In the class Serveri(ABC) is defined the IP address of the server and the port number, in the socket we have used AF_INET which is IPv4, and the SOCK_STREAM stands for TCP protocol. Then bind will store the private variable IP address and the port number, where port number can be any port which is free to use, in our case is 16000. Also, in this class we declare two empty lists pseudonimet and perdoruesit which we will fill with data. This class has one abstract method called njoftimi(self) which will be our abstract class.

```

class DergoMesazhet(Serveri):
    def njoftimi(self):
        print("Serveri eshte duke startuar, ju lutem pris
ni!")

    def TransmetoMSG(self, mesazhi):
        for perdoruesi in self.perdoruesit:
            perdoruesi.send(mesazhi)

```

As you can see the class DergoMesazhet is Inheriting Serveri class which has two methods called njoftimit(self), and TransmetoMSG(self, mesazhi). First method notifies us when the server is starting, and second method will send a message to each perdoruesi that is connected in the perdoruesit list.

```

class ProcesoMesazhet(DergoMesazhet):
    def njoftimi(self):
        print("Filloi punen sistemi i dergimit te mesazhe
ve për userat qe do te lidhen ne chat!")
    def Trajto(self,perdoruesi):
        while True:
            try:
                mesazhi = perdoruesi.recv(2048)
                self.TransmetoMSG(mesazhi)
            except:
                index = self.perdoruesit.index(perdoruesi
)

                self.perdoruesit.remove(perdoruesi)
                perdoruesi.close()
                pseudo = self.pseudonimet[index]

```

```

        self.TransmetoMSG('Nga chati doli: {}'.format(pseudo).encode('utf-8'))
        self.pseudonimet.remove(pseudo)
        print("Nga chati doli:{}".format(pseudo))
        break

```

The class `ProcesoMesazhet` is inheriting `DergoMesazhet`, has two methods `njoftimit(self)`, and `Trajto (self, perdoruesi)`. First method notifies us the messaging system started working, and second method is in infinite loop and won't stop unless an exception occurs. This method receives messages from a client and share it to all clients. If any exception occurs it will remove the client that left from the server, so he can access again the chat, also will close connection with that client. In server will display the message who left the chat also in the client's chat.

```

class StartoServerin(ProcesoMesazhet):
    def njoftimi(self):
        print("Serveri startoi punen me sukses, vazhdoni komunikimin!")
    def PranoMSG(self):
        while True:
            perdoruesi, adresa = self.s.accept()
            perdoruesi.send('MARRESI'.encode('utf-8'))
            pseudo = perdoruesi.recv(2048).decode('utf-8')

            self.pseudonimet.append(pseudo)
            self.perdoruesit.append(perdoruesi)
            print("Perdoruesi i ri i lidhur ne chat eshte: {}".format(pseudo))
            self.TransmetoMSG("U lidh ne chat: {}".format(pseudo).encode('utf-8'))
            perdoruesi.send('Tani filloni biseden me postete!'.encode('utf-8'))
            th3 =threading.Thread(target=self.Trajto, args=(perdoruesi,))
            th3.start()

```

The class `StartoServerin` is inheriting `ProcesoMesazhet`, also this class has two methods `njoftimit(self)`, and `PranoMSG(self)`, the first method will notify us that server started working successfully, and the second method is in infinite loop because all the time accept new connection from clients. If a client is connected it sends `MARRESI` and will tell the client his nickname is required. In this method we will encode messages with `utf-8`, and we are using 2048 byte for the length of the message. Also, in this method we are using a thread `th3` which has the target `Trajto` and the argument `perdoruesi`.

```

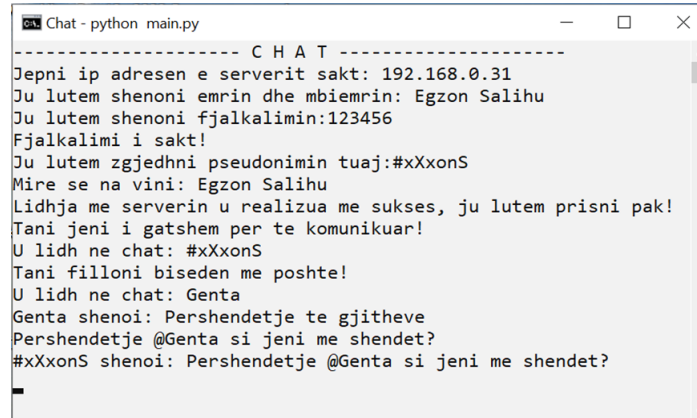
def fillo_ekzekutimin():
    b_obj=DergoMesazhet()
    c_obj=ProcesoMesazhet()
    d_obj=StartoServerin()
    b_obj.njoftimi()
    time.sleep(2)
    c_obj.njoftimi()
    time.sleep(2)
    d_obj.njoftimi()
    d_obj.PranoMSG()
if __name__=='__main__':
    fillo_ekzekutimin()

```

In the end of the server.py we have two functions fillo_ekzekutimin() and `__name__ == '__main__'`, first function will create objects for the classes we created above and initialize them, and the second one will call the function fillo_ekzekutimin().

2.2 Code snippets about Chat part

In this section we will represent our chat side that we implemented. In figure 4 is shown the module chati.py after it is executed.



```

----- C H A T -----
Jepni ip adresen e serverit sakt: 192.168.0.31
Ju lutem shenoni emrin dhe mbiemrin: Egzon Salihu
Ju lutem shenoni fjalkalimin:123456
Fjalkalimi i sakt!
Ju lutem zgjedhni pseudonimin tuaj:#xXxonS
Mire se na vini: Egzon Salihu
Lidhja me serverin u realizua me sukses, ju lutem prisni pak!
Tani jeni i gatshem per te komunikuar!
U lidh ne chat: #xXxonS
Tani filloni biseden me poshte!
U lidh ne chat: Genta
Genta shenoi: Pershendetje te gjithëve
Pershendetje @Genta si jeni me shendet?
#xXxonS shenoi: Pershendetje @Genta si jeni me shendet?

```

Fig. 4. Executed form of the module Chat.py

When the chat executes as in figure 4 user must:

Enter the IP address of the server, if the IP address is wrong the chat will be closed.

Enter Name and Surname, if user doesn't fill this field it will show Welcome user!

Enter the password, if the password is wrong user has to write it again.

Enter the nickname, this field is obligatory, otherwise u cant start the chat without filling.

When all the fields above are filled it will display:

```
Welcome: Egzon Salihu.
Connection with the server successfully started, please wait!
Now you are ready to communicate!
Connected to chat: #xXxonS
Now start chatting below:
Connected to chat: Genta, as u can see new user is connected now.
Genta says: Hello to everyone
#xXxonS says: Hello @Genta how are you with your health?
Genta left the chat, if any user will leave it will appear.
```

The Python code for the module Chati.py is:

```
import socket
from threading import *
from abc import ABC, abstractmethod
import time
```

Above we imported the library called socket, threads, ABC which stands for Abstract Classes, and time.

```
print("----- C H A T -----")
__ip=input("Jepni ip adresen e serverit sakt: ")
perdoruesi = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
    perdoruesi.connect((__ip, 16000))
except:
    print("Keni shenuar ip adresen e serverit gabim!")
    print("Chat do te mbyllet automatikisht")
    time.sleep(4)
    exit()
```

In the section of the code above we have private variable called IP which says to enter the IP address of the server, another variable declared is perdoruesi which will store socket (AF_INET, SOCK_STREAM). We also can see above an exception, this will check if the connection is starting, if not then will show the message IP address is wrong and close the chat automatically in 4 seconds.

```
class Personi():
    def __init__(self, __emrimbiemri=input("Ju lutem sheno
ni emrin dhe mbiemrin: ")):
        self.__emrimbiemri=__emrimbiemri
        if __emrimbiemri != "":
            print("Mire se na vini:", __emrimbiemri)
```

```

else:
    print("Mire se na vini: Perdorues")

```

In the class Personi is a constructor with private variable emrimbiemri. In this variable the user need enter name and surname, if the user doesn't fill this field than message will be displayed Welcome: user, here we achieved Polymorphism.

```

class Komunikimi(ABC):
    __fjalkalimi=input("Ju lutem shenoni fjalkalimin:")
    while True:
        if __fjalkalimi != "123456":
            print("Fjalkalimi i dhene eshte i pasakt!")
            __fjalkalimi=input("Ju lutem shenoni fjalkalimi sakt per te vazhduar ne chat")
        else:
            print("Fjalkalimi i sakt!")
            break
    pseudo=input("Ju lutem zgjedhni pseudonimin tuaj:")
    while True:
        if pseudo=="":
            print("Ju duhet te shenoni patjeter nje pseudonim per te filluar komunikimin!")
            pseudo=input("Ju lutem shenoni nje pseudonim:")
        else:
            break
    @abstractmethod
    def njoftimi(self):
        pass

```

The class Komunikimi is abstractclass with an abstract method called njoftimi(self), in this class is declared private variable fjalkalimi which will be store our password. Also, above is declared pseudonimi which stores nickname of the user, which should be filled anyway.

```

class KomunikimeServer(Komunikimi):
    def njoftimi(self):
        print("Lidhja me serverin u realizua me sukses, ju lutem prisni pak!")
    def PranimiMSG(self):
        while True:
            try:
                mesazhi = perdoruesi.recv(2048).decode('utf-8')
                if mesazhi == 'MARRESI':
                    perdoruesi.send(self.pseudo.encode('utf-8'))
            else:

```

```

        print(mesazhi)
    except:
        print("Lidhja me serverin u nderpre!")
        perdoruesi.close()
        break

```

The class `KomunikoMeServer` is inheriting `Komunikimi` has two methods called `njoftimi(self)` and `PranimiMSG(self)`, the first method will show as Connection with the server is successful, and the second method have infinite loop because constantly tries to receive messages and show them on the screen. If there is any exception it will close the connection. Also, if the message is `MARRESI` it doesn't print on the screen but sends its nickname to the server. As we can see in this class all the messages decode when we receive and encode when we send them with `utf-8`. Length of the message is 2048 bytes, which will be enough for large messages.

```

class MesazhetNeServer(KomunikoMeServer):
    def njoftimi(self):
        print("Tani jeni i gatshem per te komunikuar!")
    def DergoMSG(self):
        while True:
            mesazhi = '{} shenoi: {}'.format(self.pseudo,
            input(''))
            perdoruesi.send(mesazhi.encode('utf-8'))

```

The class `MesazhetNeServer` is inheriting `KomunikoMeServer`, this class has two methods `njoftimi(self)` and `DergoMSG(self)`, first method shows the notification Now you are ready to communicate, and the second one has infinite loop because is always waiting for a user input. If the user input something than this function will combinate message with the nickname and will send to the server.

```

def starto_ekzekutimin():
    p_obj=Personi()
    kms_obj=KomunikoMeServer()
    mns_obj=MesazhetNeServer()
    kms_obj.njoftimi()
    time.sleep(2)
    th1 = Thread(target=kms_obj.PranimiMSG)
    th1.start()
    mns_obj.njoftimi()
    time.sleep(2)
    th2 =Thread(target=mns_obj.DergoMSG)
    th2.start()
    time.sleep(2)
if __name__=='__main__':
    starto_ekzekutimin()

```

In the end we have two functions called `starto_ekzekutimin()`, and `__name__ == '__main__'`, first function will create objects for the classes we created above and initialize them, in this function we have used two threads `th1` will receive messages and has no arguments and `th2` will send messages also has no arguments. The second function will call the function `starto_ekzekutimin()`.

3 Testing and Validation

To test our application, you can download it in our GitHub profile, where the link is described below in Appendix A. First of all, you need to install python 3.9.1 or greater so you can open all of our python scripts, also you should have an active internet connection. File `serveri.py` is needed to run all the time in a local server or in the cloud server, `serveri.py` automatically creates logs so you can see the IP addresses and nicknames of everyone who was connected. File `chati.py` should be started in all employ's computers, users have to input the IP address of the server, password of the application which in our case password is "123456", name and surname, and their nickname which is the key to send messages. Where the employees are online in the chat they can send and receive messages in real-time from everyone who is connected.

4 Related work

Nowadays are many chat applications that use socket programming because are the bests for real-time applications. Social giant Facebook use sockets so users don't have to refresh the page to see if there are any new messages or any activity in their account. We use a lot of instant messaging applications like WhatsApp, Viber, Kick, etc. In the past, they were very simple just for text messaging, and now can send files, can make video calls, etc. All of those mentioned above are implemented in other program languages using the same logic but not in Python as we did.

5 Conclusions and Future Work

When developing a chat application one of the difficulties is to ensure secure communication between the client and the server[4]. Messages that are sent to each other must be ensured that they will be accepted by all online users. For the best user experience and secure connection is to use domain `AF_INET` which stands for IPv4, and the type of connection `SOCKET_STREAM` which stands for TCP protocol. Packets that are dropped in the network immediately are detected and will be retransmitted by the sender so no data will be lost, one more important thing is because all the data will be read by our application in the order it was written by the sender. Also, it is mandatory for all messages that the user sends to encode them using ASCII, Unicode, UTF-8, or UTF-16 and to decode when we receive them. The main reason for that is because in socket programming you cannot send strings just bytes[8]. The most important thing we conclude is if we don't use threads for parallel

programming, we have to wait to receive data from the server and then send our messages to the server so this process is not an instant message. For this reason, we have used threads to enable us to constantly receive data from the server and at the same time send our messages to the server. The file sending and windows form application is our next subject to ongoing developments. Based on the user's feedback, our system will be further enhanced and evaluate.

References

1. Abhishek Ratan, Eric Chou, Pradeeban Kathiravelu, Faruque Sarker, Learning Path-Python Network Programing, 31 January, 2019.
2. John Goerzen, Tim Bower, Brandon Rhodes, Foundations of Python Network Programming, Second Edition, 21 December 2010.
3. SINGH, AJIT, PYTHON SOCKET PROGRAMMING, 2019.
4. Adam Bash, Python Programming, 31 October, 2019.
5. John M. Zelle, Python Programming: An Introduction to Computer Science (3rd Edition), 1 January 2016.
6. Faruque Sarker, Sam Washington, Learning Python Network Programming, 2015.
7. Wajid Hassan, Python ScriPting for Network Engineers, 2019.
8. Jessica McKellar, Abe Fettig, Twisted Network Programming Essentials, 2013.
9. Hemant Kumar Srivastava, Rounak Sinha, Sumita Gupta, Implementation of Socket Programming and RMI Using Simulating Environment, International Journal of Scientific & Engineering Research, Volume 4, Issue 5, May-2013.
10. Avinash Bamane, Parikshit Bhoyar, Ashish Dugar & Lineesh Antony, Enhanced Chat Application, Global Journal of Computer Science and Technology Network, Web & Security Volume 12, Issue 11, Version 1.0, June 2012.

Appendix: A

Everyone interested in our chat application to test it or to use it for his company you can get this app in our GitHub account:

https://github.com/Geentiana/Projekti_Python_SC