İhsan Doğramacı Bilkent University

Department of Computer Engineering

CS421 – Computer Networks Programming Assignment 2 *Instant Message App*

Instructor: Ezhan Karaşan

Name: Sebahattin Utku

Surname: Sezer

ID: 21802798

Contents

In this programming assignment, we were expected to implement server and client which are establishing TCP connection and messaging through sockets by using HTTP requests. Also, we are expected to implement client's messaging system through UDP connection, which is peer-to-peer connection. I have implemented this assignment using Python 3.9.0 on Windows 10 Pro operating system.

Implementation

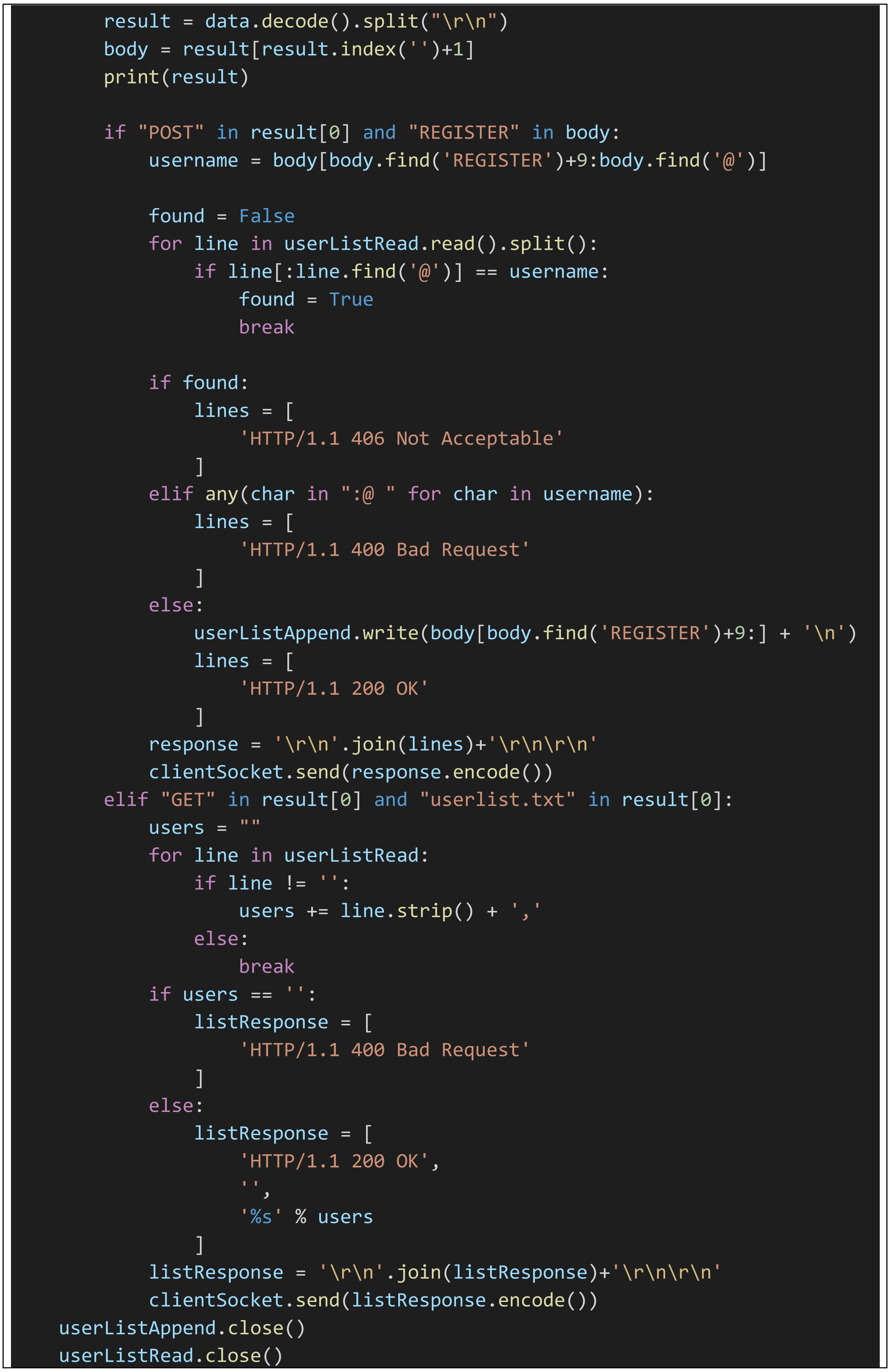
Message Server

For the client and server part, I have implemented server as responding to two specific HTTP requests, which are HTTP POST and HTTP GET. As it was in the documentation, HTTP POST request is for registering clients to the online users list ("userlist.txt"), and HTTP GET request is for getting the online user list from server.

Connections between server and clients are established through socket with IP address and port number are gotten from user's input. After serving binding his socket with IP and port number, it waits to receive HTTP messages from clients.

MessageServer.py code can be found below:

```
import socket
import sys

input = sys.argv[1]

serverURL = input[:input.find(':')]
serverPort = int(input[input.find(':')+1:])

userListAppend = open("userlist.txt", "a")
userListRead = open("userlist.txt", "r")

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((serverURL, serverPort))
s.listen(10)

clientPort = s.getsockname()

while True:
    clientSocket, address = s.accept()
    data = clientSocket.recv(2048)
    userListAppend = open("userlist.txt", "a")
    userListRead = open("userlist.txt", "r")

    if data:
```



```python
        result = data.decode().split("\r\n")
        body = result[result.index('')+1]
        print(result)

        if "POST" in result[0] and "REGISTER" in body:
            username = body[body.find('REGISTER')+9:body.find('@')]

            found = False
            for line in userListRead.read().split():
                if line[:line.find('@')] == username:
                    found = True
                    break

            if found:
                lines = [
                    'HTTP/1.1 406 Not Acceptable'
                ]
            elif any(char in ":@ " for char in username):
                lines = [
                    'HTTP/1.1 400 Bad Request'
                ]
            else:
                userListAppend.write(body[body.find('REGISTER')+9:] + '\n')
                lines = [
                    'HTTP/1.1 200 OK'
                ]
            response = '\r\n'.join(lines)+'\r\n\r\n'
            clientSocket.send(response.encode())
        elif "GET" in result[0] and "userlist.txt" in result[0]:
            users = ""
            for line in userListRead:
                if line != '':
                    users += line.strip() + ','
                else:
                    break
            if users == '':
                listResponse = [
                    'HTTP/1.1 400 Bad Request'
                ]
            else:
                listResponse = [
                    'HTTP/1.1 200 OK',
                    '',
                    '%s' % users
                ]
            listResponse = '\r\n'.join(listResponse)+'\r\n\r\n'
            clientSocket.send(listResponse.encode())
    userListAppend.close()
    userListRead.close()
    clientSocket.close()

s.close()

userListAppend.close()
userListRead.close()
```

Message Client

For the client side, there were many functionalities as it was well described in the assignment documentation. Since they were pre-described, I will not go through what each functionality does once more, but I will describe my frequently used two functions in client implementation.

Client also uses sockets to communicate with server and other clients. When client tries to communicate with server, it creates a TCP connection with socket, when it tries to connect with other client, it creates a UDP connection. TCP connection is established by using the following socket initialization:

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

UDP connection is established by using the following socket initialization:

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

Function to update user list

Since we were needed to update the online users list from the information, we get from the server many times in almost every function that client can do, I have made a function about it. This function basically sends a HTTP GET request for "userlist.txt" file that server keeps track of. According to server's response, it informs the user if there are no online users available, or it just updates the whole list. Functions code can be found below:

```
def updateUsers():
    # Updates the online users list -----
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((serverURL, serverPort))
    a = []
    listReq = [
        'GET %s HTTP/1.1' % ('/' + 'userlist.txt'),
        'Host: %s' % serverURL
    ]
    listPost = '\r\n'.join(listReq)+'\r\n\r\n'
    s.send(listPost.encode())
    listData = s.recv(2048)
    listResult = listData.decode().split("\r\n")
    if '200' in listResult[0]:
        listBody = listResult[listResult.index('')+1]
        usersSplitted = listBody.split(',')
        for user in usersSplitted:
            if user != '':
                a.append(user)
            else:
                break

    elif '400' in listResult[0]:
        print("There is no online users!")
    s.close()
    return a
```

Function to send message to another client

Since for every "send" mode functions we need to send a message to other client using UDP connection, I have made this functionality as function. What it does is simply adding username to the message (since there is no way from receiver client to know username), and send this message to indicated IP and port number. Code can be found below:

```
def sendMessageP2P(userAddressPort, message, username):
    message = username + ": " + message
    sendAddressPort = userAddressPort[userAddressPort.find('@')+1:]
    sendAddress = sendAddressPort[:sendAddressPort.find(':')]
    sendPort = sendAddressPort[sendAddressPort.find(':')+1:]

    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    s.sendto(message.encode(), (sendAddress, int(sendPort)))
    s.close()
    print("message is sent to", userAddressPort[:userAddressPort.find('@')])
```

Complete implementation of InstantMessenger .py can be found below:

```
import socket
import sys

def updateUsers():
    # Updates the online users list -----
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((serverURL, serverPort))
    a = []
    listReq = [
        'GET %s HTTP/1.1' % ('/' + 'userlist.txt'),
        'Host: %s' % serverURL
    ]
    listPost = '\r\n'.join(listReq)+'\r\n\r\n'
    s.send(listPost.encode())
    listData = s.recv(2048)
    listResult = listData.decode().split("\r\n")
    if '200' in listResult[0]:
        listBody = listResult[listResult.index('')+1]
        usersSplitted = listBody.split(',')
        for user in usersSplitted:
            if user != '':
                a.append(user)
            else:
                break

    elif '400' in listResult[0]:
        print("There is no online users!")
    s.close()
    return a

def sendMessageP2P(userAddressPort, message, username):
    message = username + ": " + message
    sendAddressPort = userAddressPort[userAddressPort.find('@')+1:]
    sendAddress = sendAddressPort[:sendAddressPort.find(':')]
    sendPort = sendAddressPort[sendAddressPort.find(':')+1:]

    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    s.sendto(message.encode(), (sendAddress, int(sendPort)))
    s.close()
    print("message is sent to", userAddressPort[:userAddressPort.find('@')])

username = sys.argv[1]
addrPort = sys.argv[2]
mode = sys.argv[3]

if any(char in ":@ " for char in username):
    print("Invalid username!")
    exit()

serverURL = addrPort[:addrPort.find(':')]
serverPort = int(addrPort[addrPort.find(':')+1:])


availableUsers = []

if mode == "listen": # Listen mode
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((serverURL, serverPort))
    clientIP = s.getsockname()[0]
    clientPort = s.getsockname()[1]
    # Prepares HTTP POST request
    register = 'REGISTER %s' % (username + '@' + clientIP + ':' +
str(clientPort))
    lines = [
        'POST %s HTTP/1.1' % ('/' + 'userlist.txt'),
        'Host: %s' % serverURL,
        'Content-Type: %s' % 'text/*',
        'Content-Length: %s' % len(register),
        '',
        '%s' % register
    ]
    post = '\r\n'.join(lines)+'\r\n\r\n'
    s.send(post.encode()) # Sends HTTP POST
    data = s.recv(2048)
    result = data.decode().split("\r\n")

    if "200" in result[0]: # 200 OK response
        s.close()
        s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        s.bind((clientIP, clientPort))

        while True:
            message = s.recv(2048).decode()
            print(message)
    elif "400" in result[0]: # 400 Bad Request response
        print("Invalid username character!")
        s.close()
        exit()
    elif "406" in result[0]: # 406 Not Acceptable response
        print("This user already exists!")
        s.close()
        exit()

elif mode == "send": # Send mode
    while True:
        sendCommand = input("")

        if "list" in sendCommand: # 'list' command in "send" mode
            # Updates the online users list -----
            availableUsers = []
            availableUsers = updateUsers()

            print("The online users are:")
            for user in availableUsers:
                print(user[:user.find('@')])

        elif "unicast" in sendCommand: # 'unicast <username> "message"'
command in "send" mode
            msg = sendCommand.split()
            sendUser = msg[1]
            sendMessage = msg[2]
            sendMessage = sendMessage[1:-1]

            # Updates the online users list -----
            availableUsers = []
            availableUsers = updateUsers()

            # Searches for the user in the online users list
            sendAddressPort = ""
            for users in availableUsers:
                if users[:users.find('@')]  == sendUser:
                    sendAddressPort = users
                    break

            if sendAddressPort != "": # If the user is found
                sendMessageP2P(sendAddressPort, sendMessage, username)
            else:
                print("user", sendUser, "is not found")

        elif "broadcast" in sendCommand: # 'broadcast "message"' command in
"send" mode
            sendMessage = sendCommand.split()[1][1:-1]

            # Updates the online users list -----
            availableUsers = []
            availableUsers = updateUsers()

            if availableUsers: #If there are some online user(s)
                for user in availableUsers:
                    sendMessageP2P(user, sendMessage, username)
            else:
                print("there is no online user")

        elif "multicast" in sendCommand: # 'multicast [user1, user2, ...
userN] "message"' command in "send" mode
            userListStr =
sendCommand[sendCommand.find('[')+1:sendCommand.find(']')]
            userList = [s.strip() for s in userListStr.split(',')] # Creates
a list from user input
            sendMessage =
sendCommand[sendCommand.find('"')+1:sendCommand.rfind('"')]

            # Updates the online users list -----
            availableUsers = []
            availableUsers = updateUsers()

            if userList:
                for sendUser in userList: # Searches for the user input list
```

```
                    flag = False
                    for avaUser in availableUsers: # Searches for the online
users
                        if sendUser == avaUser[:avaUser.find('@')]:
                            flag = True
                            sendMessageP2P(avaUser, sendMessage, username)

                    if not flag:
                        print("user", sendUser, "is not found")
            else:
                print("please enter a user in []")

        elif "exit" in sendCommand:
            exit()
        else:
            print("Invalid command!")

else:
    print("Invalid mode!")
s.close()
```

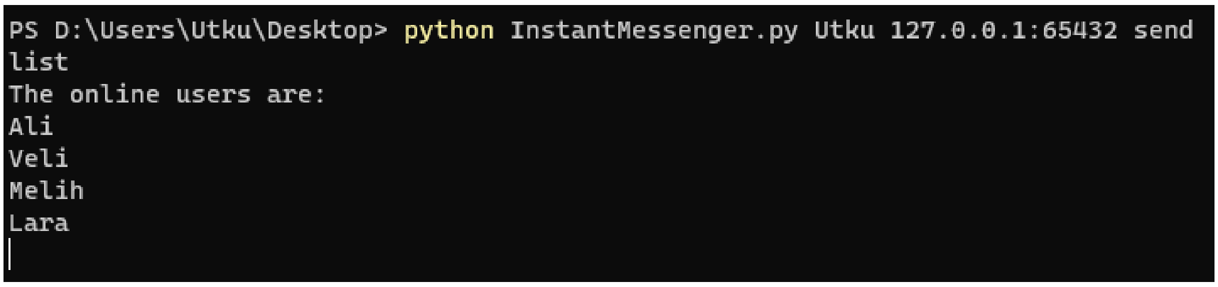
Sample Runs

For sample runs, I have created four clients which are in “listen” mode and two clients which are in “send” mode to test all the functionalities. “listen” mode clients’ names are Ali, Veli, Melih, Lara while “send” mode clients’ names are Utku and Defne.

List Function:

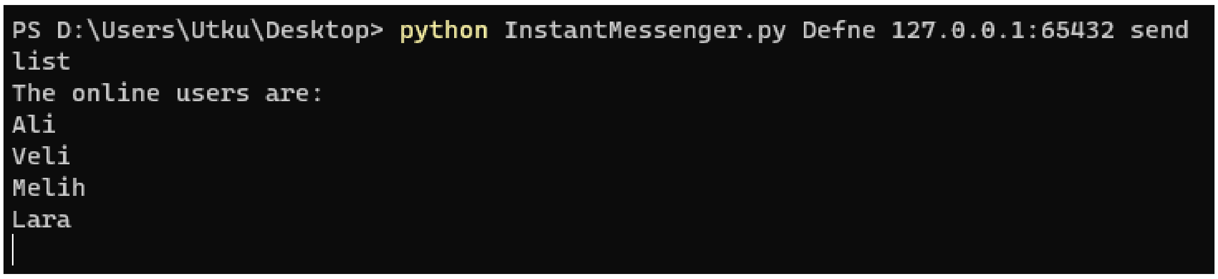
For Utku and Defne, if we use “list” function we get:

- For Utku:



```
PS D:\Users\Utku\Desktop> python InstantMessenger.py Utku 127.0.0.1:65432 send
list
The online users are:
Ali
Veli
Melih
Lara
```

- For Defne:



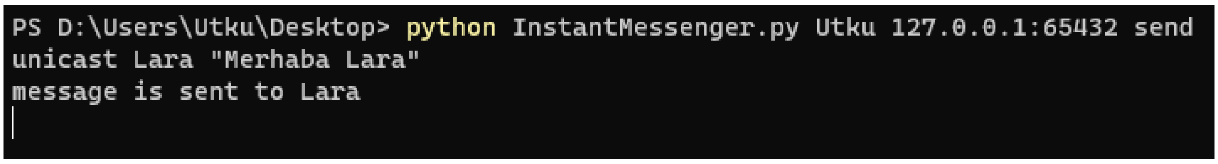
```
PS D:\Users\Utku\Desktop> python InstantMessenger.py Defne 127.0.0.1:65432 send
list
The online users are:
Ali
Veli
Melih
Lara
```

as we have expected. Both Utku and Defne can see the online “listen” mode users, and not themselves since “read” mode clients are not registered to server.

Unicast Function:

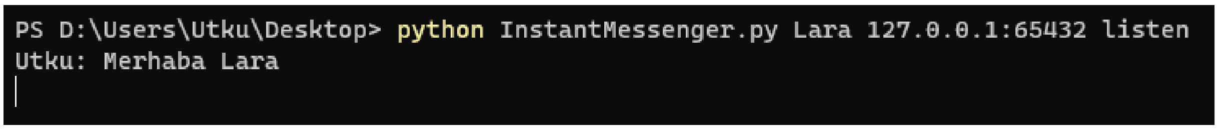
For Utku, if we try to send a message to Lara, we get the following:

- Utku sees:



```
PS D:\Users\Utku\Desktop> python InstantMessenger.py Utku 127.0.0.1:65432 send
unicast Lara "Merhaba Lara"
message is sent to Lara
```

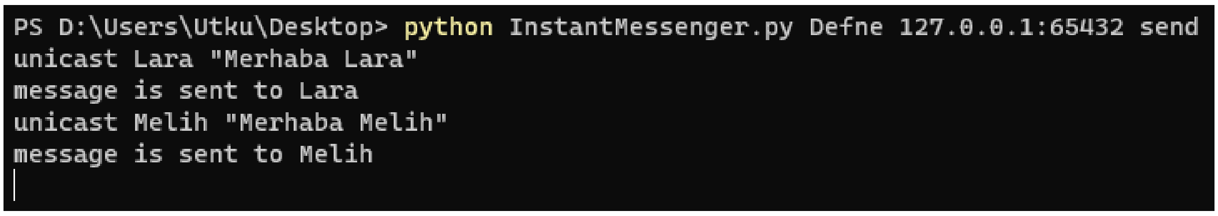
- Lara sees:



```
PS D:\Users\Utku\Desktop> python InstantMessenger.py Lara 127.0.0.1:65432 listen
Utku: Merhaba Lara
```

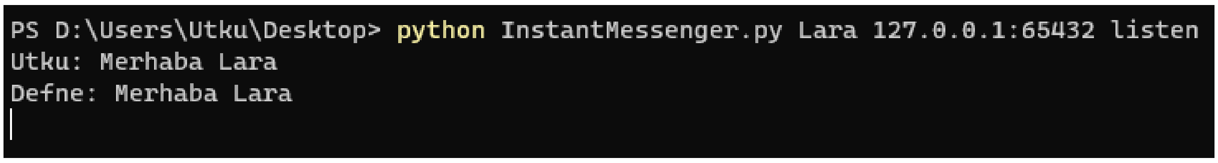
If we also try to send message from Defne to both Lara and Melih by using “unicast” function, we get:

- Defne sees:



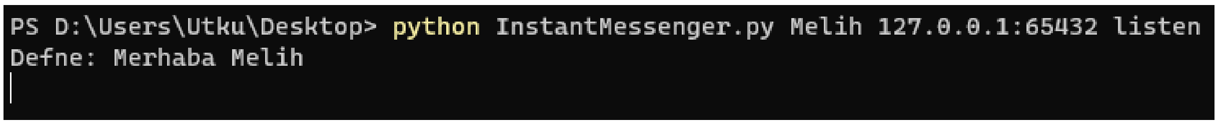
```
PS D:\Users\Utku\Desktop> python InstantMessenger.py Defne 127.0.0.1:65432 send
unicast Lara "Merhaba Lara"
message is sent to Lara
unicast Melih "Merhaba Melih"
message is sent to Melih
```

- Lara sees:



```
PS D:\Users\Utku\Desktop> python InstantMessenger.py Lara 127.0.0.1:65432 listen
Utku: Merhaba Lara
Defne: Merhaba Lara
```

- Melih sees:



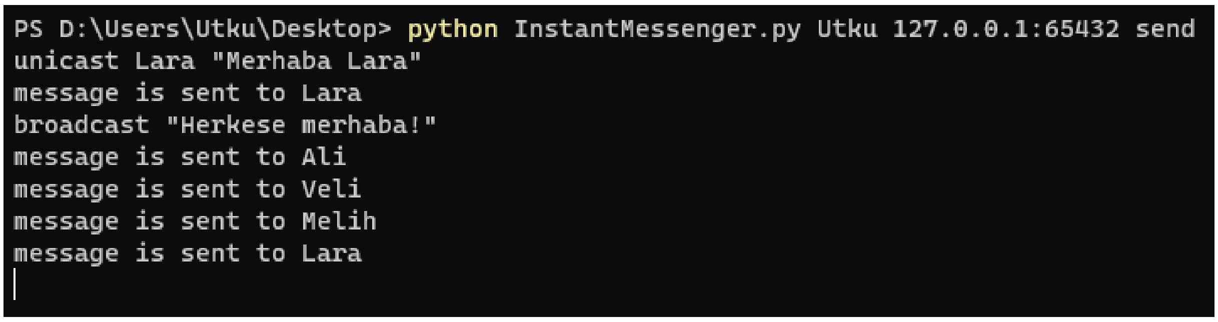
```
PS D:\Users\Utku\Desktop> python InstantMessenger.py Melih 127.0.0.1:65432 listen
Defne: Merhaba Melih
```

As it can be seen above, “unicast” function works as desired.

Broadcast Function:

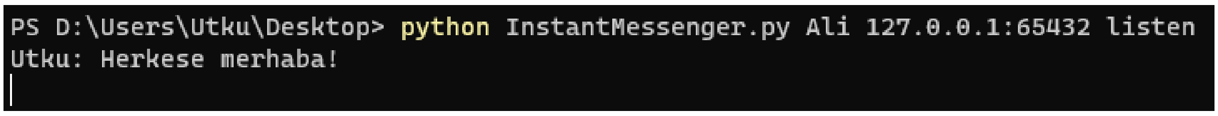
Since “broadcast” function sends messages to all available users, when Utku uses “broadcast” we get the same message from all clients who are in “listen” mode:

- Utku sees:

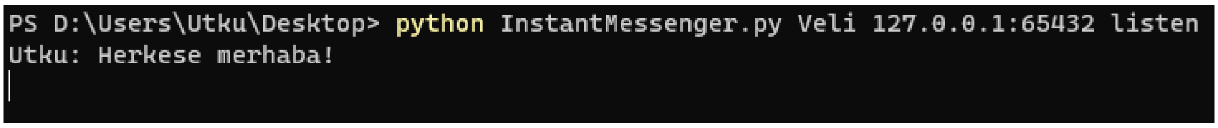


```
PS D:\Users\Utku\Desktop> python InstantMessenger.py Utku 127.0.0.1:65432 send
unicast Lara "Merhaba Lara"
message is sent to Lara
broadcast "Herkese merhaba!"
message is sent to Ali
message is sent to Veli
message is sent to Melih
message is sent to Lara
```

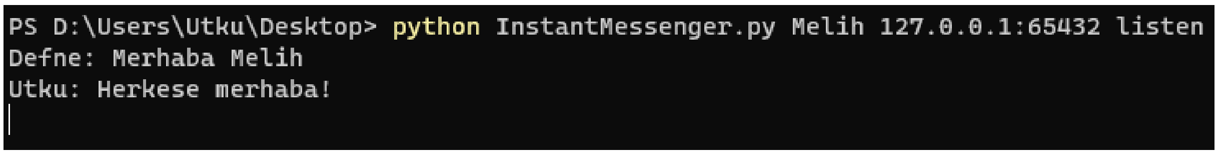
- Ali, Veli, Lara, and Melih sees respectively:



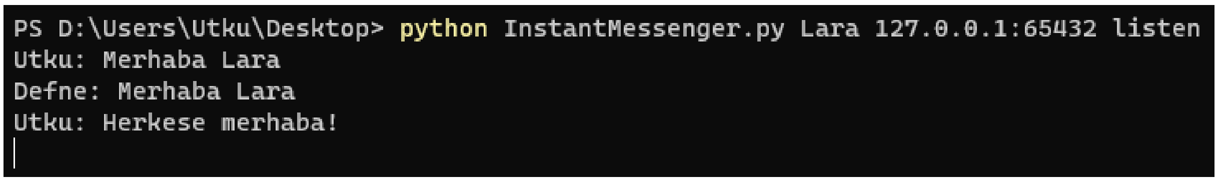
```
PS D:\Users\Utku\Desktop> python InstantMessenger.py Ali 127.0.0.1:65432 listen
Utku: Herkese merhaba!
```



```
PS D:\Users\Utku\Desktop> python InstantMessenger.py Veli 127.0.0.1:65432 listen
Utku: Herkese merhaba!
```



```
PS D:\Users\Utku\Desktop> python InstantMessenger.py Melih 127.0.0.1:65432 listen
Defne: Merhaba Melih
Utku: Herkese merhaba!
```



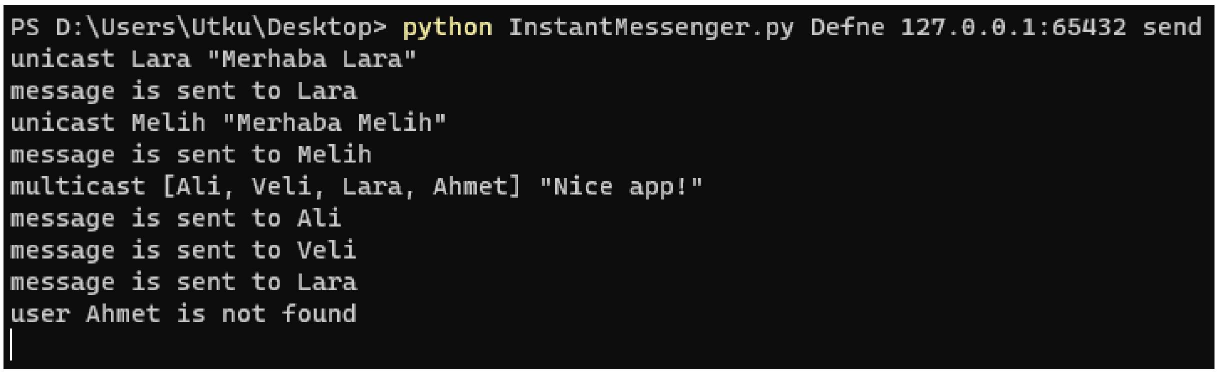
```
PS D:\Users\Utku\Desktop> python InstantMessenger.py Lara 127.0.0.1:65432 listen
Utku: Merhaba Lara
Defne: Merhaba Lara
Utku: Herkese merhaba!
```

As it can be seen, all “listen” mode clients are getting the same message from the sender, Utku.

Multicast Function

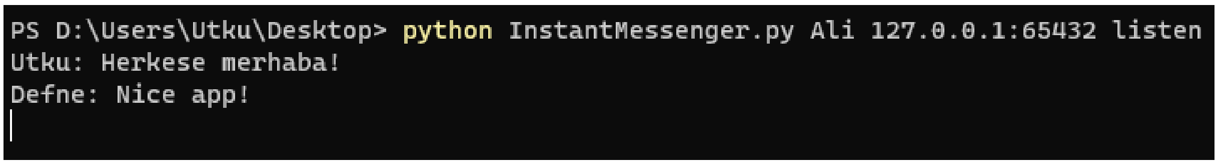
In "multicast", we can send a message to several "listen" mode clients as we choose. Let us assume that Defne sends a message to Ali, Veli, Lara, and Ahmet (which is not an available user).

- Defne sees:



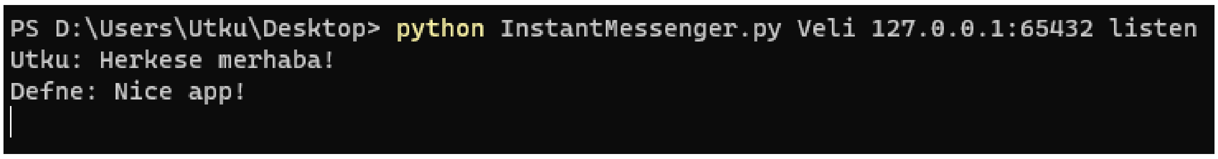
```
PS D:\Users\Utku\Desktop> python InstantMessenger.py Defne 127.0.0.1:65432 send
unicast Lara "Merhaba Lara"
message is sent to Lara
unicast Melih "Merhaba Melih"
message is sent to Melih
multicast [Ali, Veli, Lara, Ahmet] "Nice app!"
message is sent to Ali
message is sent to Veli
message is sent to Lara
user Ahmet is not found
```

- Ali sees:



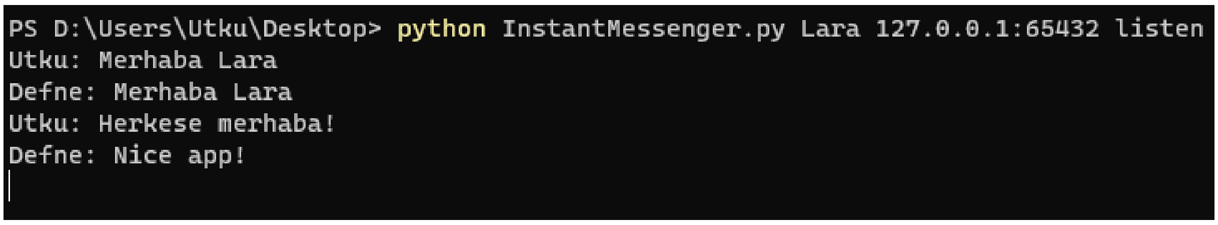
```
PS D:\Users\Utku\Desktop> python InstantMessenger.py Ali 127.0.0.1:65432 listen
Utku: Herkese merhaba!
Defne: Nice app!
```

- Veli sees:



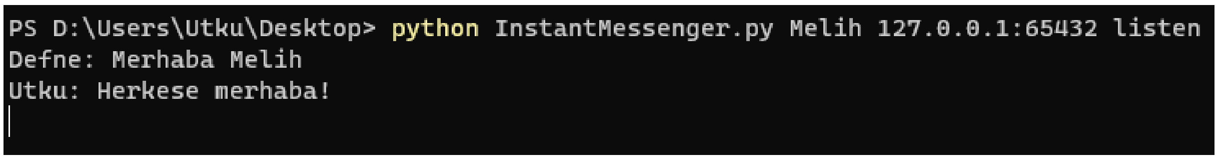
```
PS D:\Users\Utku\Desktop> python InstantMessenger.py Veli 127.0.0.1:65432 listen
Utku: Herkese merhaba!
Defne: Nice app!
```

- Lara sees:



```
PS D:\Users\Utku\Desktop> python InstantMessenger.py Lara 127.0.0.1:65432 listen
Utku: Merhaba Lara
Defne: Merhaba Lara
Utku: Herkese merhaba!
Defne: Nice app!
```

- Melih sees:



```
PS D:\Users\Utku\Desktop> python InstantMessenger.py Melih 127.0.0.1:65432 listen
Defne: Merhaba Melih
Utku: Herkese merhaba!
```

As it can be seen, Melih cannot see the message Defne sent since Defne did not specify him in "multicast" function.

Exit Function

Exit function simply ends the "read" mode clients' session and exits the code.

Bonus

For bonus functionality, I have added an option to leave the "listen" mode for clients who are in this mode. Clients in "listen" mode who receives more than 5 consecutive messages are asked if they want to leave the application or not. If they choose to leave the application, they send HTTP POST request to server indicating that their user record needs to be deleted from "userlist.txt". HTTP POST request's body is as follows:

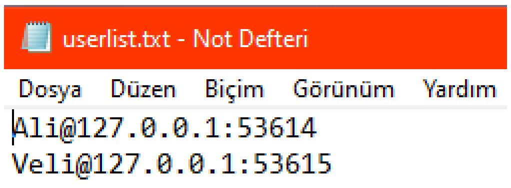
LOGOUT <username>@<IP>:<Port> (e.g. "LOGOUT hamza@192.168.1.13:54321")

With this way, user who logs out from the session will be deleted from "userlist.txt" preventing clients in "send" mode to list the clients who are logged out. When clients in "send" mode cannot list their names, they cannot send messages to these logged out users as if they have never existed. If the user does not want to logout, it asks the same question for every 5 messages the "listen" client receives.

I have named bonus part files as "MessageServerBonus.py" and "InstantMessenger.py" in an extra folder named "Bonus".

For sample run of this code, I have created "listen" mode users named Ali and Veli. I have created a "send" mode user, named Utku. When Utku sends 3 messages to Ali and 5 messages to Veli, Veli gets a question regarding whether he wants to logout and he chooses "y" (indicating "yes").

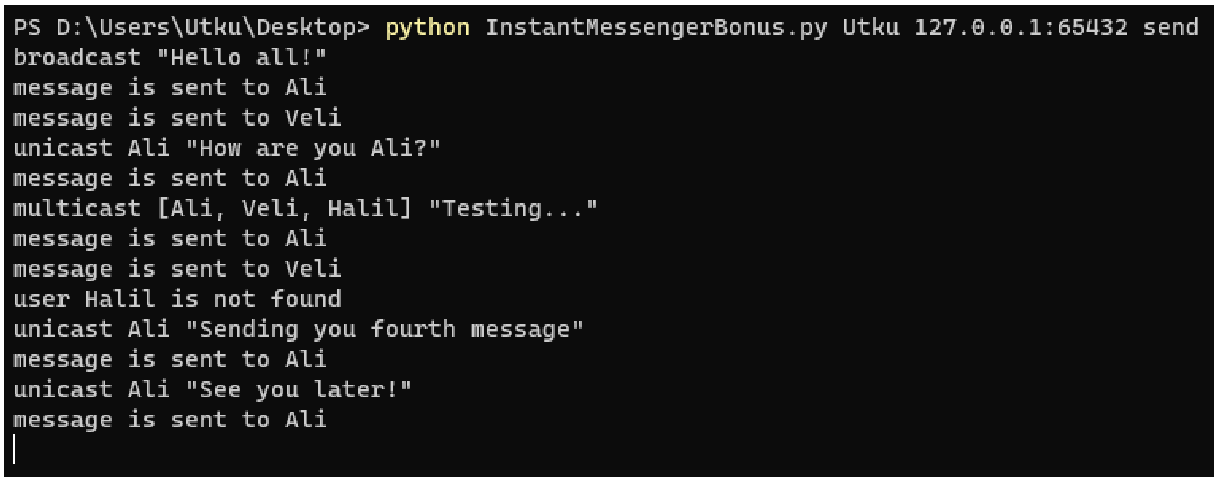
"userlist.txt" in server after everyone signs in:



```
Ali@127.0.0.1:53614
Veli@127.0.0.1:53615
```

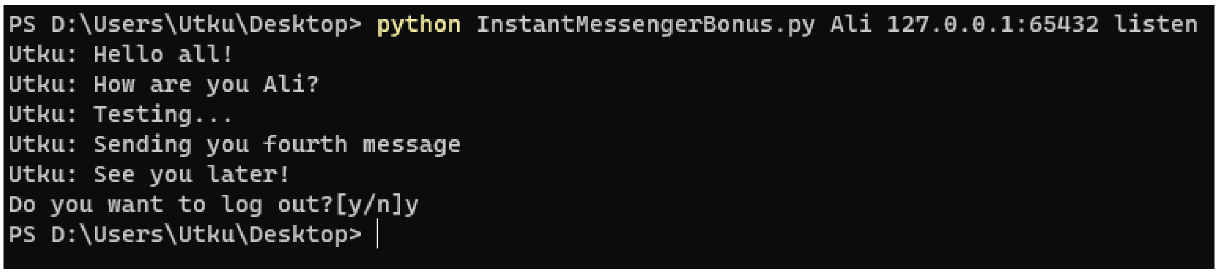
After some messaging,

- Utku sees:



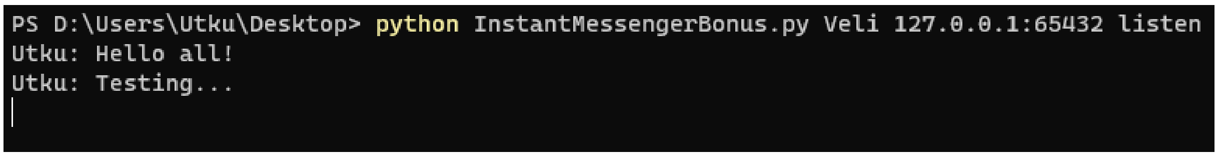
```
PS D:\Users\Utku\Desktop> python InstantMessengerBonus.py Utku 127.0.0.1:65432 send
broadcast "Hello all!"
message is sent to Ali
message is sent to Veli
unicast Ali "How are you Ali?"
message is sent to Ali
multicast [Ali, Veli, Halil] "Testing..."
message is sent to Ali
message is sent to Veli
user Halil is not found
unicast Ali "Sending you fourth message"
message is sent to Ali
unicast Ali "See you later!"
message is sent to Ali
```

- Ali sees:



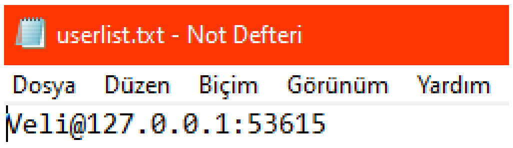
```
PS D:\Users\Utku\Desktop> python InstantMessengerBonus.py Ali 127.0.0.1:65432 listen
Utku: Hello all!
Utku: How are you Ali?
Utku: Testing...
Utku: Sending you fourth message
Utku: See you later!
Do you want to log out?[y/n]y
PS D:\Users\Utku\Desktop>
```

- Veli sees:



```
PS D:\Users\Utku\Desktop> python InstantMessengerBonus.py Veli 127.0.0.1:65432 listen
Utku: Hello all!
Utku: Testing...
```

“userlist.txt” in server after Ali logs out:



```
Veli@127.0.0.1:53615
```

As it can be seen, when Ali decides to log out, he is not at “userlist.txt” anymore.