**CSE 4074 COMPUTER NETWORKS PROJECT**

**AHMET ONAT ÖZALAN – 150118054**

**ALİ ONUR ASLAN – 150117040**

**GÜNEŞ YÜZAK – 150117004**

**Project Summary**

In this project, we’ve created an application that uses the internet infrastructure for the first time. We’ve learned how to deal with sockets, what local and public ip’s are, and provide communication between threads and many more. But the most useful and interesting thing we’ve learned while doing the project was taking the input even before the user presses the ‘enter’ key.

**Solution Approach**

We haven’t thought about the structure of the program in a very detailed way, before starting to write the code. We’ve started coding and tried to solve the problems we’ve came across. We’ve searched for the internet for possible libraries we could use and tried them in test files, and after that, implemented them in our real code.

**Encountered Problems & Their Solutions**

1. Our computers has been using a different IP address from our local IP addresses. We’ve figured that the program named VirtualBox was the one had been causing this problem. We’ve disabled its network related adapter and the problem has been solved.
2. While the users has been communicating, incoming messages have been interrupting the input process of the receiving user. They have been getting appended to the input the receiving user has been currently typing. We’ve solved this by using Python’s msvcrt library. It helped us take the input one by one, and even without the need of waiting for the user to press the ‘enter’ key.
3. Peer-side program opens a new thread for sending messages when a new chat starts. If one wants to leave the chat, they can type !quit and leave the chat. This process closes the send-msg-thread for the peer that typed quit, but the other peer’s send-msg-thread is remained open. To solve this problem, we’ve again used the msvcrt library. Instead of using built-in input function, we’ve writed our code for taking input and made it check if the chat was over.
4. Msvcrt was very helpful for us, but we couldn’t use it immediately. It just wasn’t working on our IDE. We’ve searched on the internet and found that we needed to “emulate the terminal in output console” for it to work.
5. We made msvcrt work on our IDE, but it caused another problem for us (actually it is not its problem). One of the computers was running the server and the peer program simultaneously. Server was running on PyCharm, and peer was running on terminal. When taking input to send it to the user we chat, a decoding problem occured. msvcrt’s getche() method was taking the inputs as bytes, and it raised an error when we typed Turkish characters. We changed the encoding to a one that supports Turkish characters, but this time weird symbols appeared when we typed Turkish characters. We found that the reason of this was terminal. We ran the registry on terminal and peer on PyCharm, and our problem solved.

**Unresolved Issues**

1. We’ve not implemented the UDP socket feature.
2. We’ve not provided error checks for the most of the part of the code. (Unless it is stated in the programming assigment pdf)

**Usage Explanation**

First, the main menu welcomes the user. In this stage, the user can register with a new account, login to his/her existing account, or close the application by typing the corresponding numbers and pressing the ‘enter’ key.

After logging in successfully, the user can send a chat request to another user, log-out from his/her account, and answer the chat request. ‘Answer call’ option is only used when there is an incoming chat request from a user. If the user selects ‘send chat request to someone’ option, he/she can enter the username of the user he/she wants to chat and send them to a chat request. If the user receiving the request accepts, chat starts and they both are redirected to chat screen; else, they both are redirected to the after-login screen.

**PROTOCOL BETWEEN PEER AND REGISTRY**

1. **The messages peer can send**

* REGISTER <username> <password>
* LOGIN <username> <password>
* EXIT: For closing the server thread of peer
* SEARCH <username>
* LOGOUT

1. **The messages registry can send**

* 100 USERNAME\_TAKEN
* 101 REGISTER\_SUCCESSFUL
* 102 USER\_NOT\_FOUND
* 103 WRONG\_PASSWORD
* 104 ALREADY\_LOGGED
* 105 LOGIN\_SUCCESSFUL
* 106 LOGOUT\_SUCCESSFUL
* 107 LOGOUT\_FAILED
* 108 USER\_OFFLINE
* 109 SEARCH\_SUCCESSFUL <ip>
* 113 EXIT\_CONFIRMED: This may be unnecessary
* EXIT: For closing the server thread of peer

**PROTOCOL BETWEEN PEERS**

* CHAT <username>: Username belongs to the peer that sends the message
* MESSAGE <username> <message>: Username belongs to the peer that sends the message
* 110 CHAT\_REQUEST\_ACCEPTED
* 111 CHAT\_REQUEST\_REJECTED
* 112 BUSY