Advanced Programming Coursework

Abidon Jude Fernandes 001013672, Ronnelle Muia 001019466, Ibrahim Anjum 001011658 and Sean Daly 001013392
COMP1549: Advanced Programming
University of Greenwich
Old Royal Naval College
United Kingdom

I. Introduction

Following the specification provided to us, our group set out to achieve the requirements as best we could to create a client-server application using Python. To better aid our understanding of the coursework and specifications, we looked to existing social media applications that closely follow the concept we were provided. Applications such as WhatsApp, Discord and Instagram all have client to server messaging features where data is sent from one client to another via the server. This proved somewhat useful in obtaining a direction to work toward, as this area of programming was new to us and required a level of understanding greater than initially told.

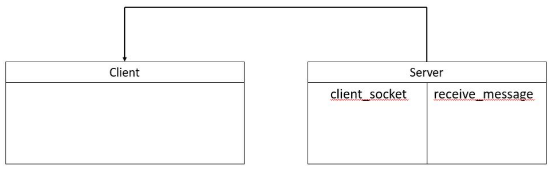
The content of this paper will include Design/implementation which will clarify all the coursework requirement we had to meet, what programming language we used, why we used it, how we got the program running, how each of the components function and why they function.
Analysis and Critical Discussion section will be an in-depth analysis of our program result on how it met all our coursework required tasks. Explaining modularity, fault tolerance, testing and through which component we used. Prior to the conclusion, there will be an overview of limitations and weakness that arose and still persist within our code at the time of writing this report. Lastly, the paper will conclude with a general overview of the report, the program itself and what we intend to do moving forward with this program.

II. Design/implementation

As aforementioned, a client-server application needed to be designed with various additional features implemented. We set out to do so in Python due to the level of simplicity in developing the code for the server and client. The libraries it offers aided greatly in the development as it allowed us to focus more on the other features other that establishing communication between multiple client instances. A summary of additional features required in implementation:

1) A server to client connection-based program with a client and server class.
2) When connected a unique ID assigned to each user, a port it will listen into, IP address number and a port and IP number of one of the existing members.
3) Allows every user to message each of the connected members or the whole group in the server.
4) Display all the currently connected members in your server.
5) Allow every user to exit the client using CTRL+C.

We managed to include all the requirements that were needed. Below is a UML diagram illustrating the general structure and interactions between classes and objects within the program. The client class inherits from the server class, allowing it to send messages to and from. both server and client are composed of global variable S, used to store connections of all users within the server.



The Classes work as follows:

Server has a main class called receive_messsage which allows the listening of any specified port that the user inputs, we had used port 8801 as it was free. The program then is able to parse the user's inputs as an IPV4 address and initialises a TDP connection to be used, UDP could have been used but TDP is more reliable as it confirms the connection before sending information.

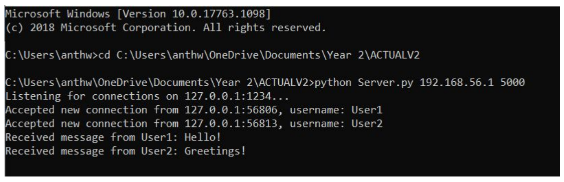
III. Analysis and Critical Discussion

In this section we will explain the results of our program and how it meets the coursework specification. Afterwards we explain how we achieved modularity, fault tolerance, testing and through which components. Finally, I will conclude by stating the weakness and limitations of our implementation choices.
We approached this problem via implementing two main classes: Client, and Server.

Results

Our code managed to create a client that handles all connections to it, and in turn the client can accept strings from each user and send them to the server which displays it to all users currently connected to it.
we had used UTF-8 encoding to test this as the UTF-8 set included all commonly used characters of the English alphabet, numbers, and punctuation.

Instance of running the server (command: ‘python Server.py “<server-ip>” “<server-port>”’)

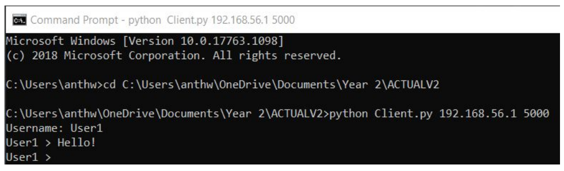


```
Microsoft Windows [Version 10.0.17763.1098]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\anthw>cd C:\Users\anthw\OneDrive\Documents\Year 2\ACTUALV2

C:\Users\anthw\OneDrive\Documents\Year 2\ACTUALV2>python Server.py 192.168.56.1 5000
Listening for connections on 127.0.0.1:1234...
Accepted new connection from 127.0.0.1:56806, username: User1
Accepted new connection from 127.0.0.1:56813, username: User2
Received message from User1: Hello!
Received message from User2: Greetings!
```

First client (ran with server IP address and server port as connection parameters)

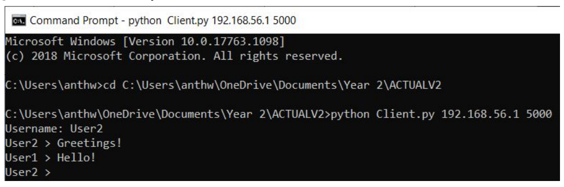


```
Command Prompt - python  Client.py 192.168.56.1 5000
Microsoft Windows [Version 10.0.17763.1098]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\anthw>cd C:\Users\anthw\OneDrive\Documents\Year 2\ACTUALV2

C:\Users\anthw\OneDrive\Documents\Year 2\ACTUALV2>python Client.py 192.168.56.1 5000
Username: User1
User1 > Hello!
User1 >
```

Second client instance (ran with server IP address and server port as connection parameters)



```
Command Prompt - python  Client.py 192.168.56.1 5000
Microsoft Windows [Version 10.0.17763.1098]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\anthw>cd C:\Users\anthw\OneDrive\Documents\Year 2\ACTUALV2

C:\Users\anthw\OneDrive\Documents\Year 2\ACTUALV2>python Client.py 192.168.56.1 5000
Username: User2
User2 > Greetings!
User1 > Hello!
User2 >
```

Modularity

Modularity was mainly implemented via the use of the reveice_message function made in the server program, this allows the program to listen to the specified port that the user inputs and decipher the information bytes into the UTF-8 character set.

As well as the net code being modular, the Graphical User Interfaces that were planned to be used in the implementation were built to be reusable. They were created as separate classes that could be imported and then instantiated whenever needed. Both classes have getter and setter methods that allow for interaction with the instance of the object.

Fault Tolerance

Fault tolerance is the mitigation of components failing and preventing operation. Within our code, components preventing the project from working is limited to either running the program or whilst communicating with other clients. Firstly, the server must be started before any clients can join. Any editing of the server address can be done within the program's code. While this may seem susceptible to additional input, parameters entered will not have an added effect as they aren't processed. Similarly, when running the client, any parameter input will not be recognised as everything the program needs to connect to the server (i.e. the server's address would be stated in the code itself). When inputting a client name, the system checks and will re-prompt users if their name isn't a valid alphanumeric input. Similarly, if any message they send isn't with the standard UTF-8 encoding, it will not be processed, preventing any foreign text input from crashing the system. Lastly and importantly, to break clients leaving from affecting the server, the handler method within the server class runs a loop that sends and receives data for each connection. If someone leaves, their connection loop breaks, nothing else.

We also enabled the use of variable parsing: if the user inputs a wrong number of variables required to initiate the program, the program outputs a prompt with the correct syntax implementation

Testing

To test functionality of the project, we implemented a unit testing file. Python has a built in unit testing framework that allows for this to be performed, which is named by the file 'unittest'. This tool was originally inspired by Junit and it supports test automation, sharing of setup and shutdown code for tests, aggregation of tests into collections and independence of the tests from the reporting framework. It also supports important object-oriented frameworks such as test fixture, text case, text suite and test runner.
These are all the tests we have done on our Python network program. (Python, n.d.)
First, we created a small program to automatically test a few aspects of the project to ensure they would receive, and process inputs as intended. The program tests against specific outputs to determine if methods fully work. There were four major areas that we were able to test based on the main code: server address, messages, default sockets and users connected. Since these methods and classes can be tested separately, they proved to be most ideal for unit testing.
The server class obtains the IP address of the device at the time of testing and compares it to the IP address at the time to determine if the main program correctly obtains the correct IP address for connections to the server.
The second method tests the client's ability to send messages to the server as they would appear in chat logs. The message "Hello Server" is sent from the method as an object and if the test is successful, the message tested will match the message sent.
The third method runs the client method which joins the server using a default port "1000". This test will only run unsuccessfully if a user joins a server that has a customised port address.
Lastly, the fourth method simply obtains the list of peers (based on their IP address), in an array and prints the array. The default peer of the user will always be the local host.

Weakness and Limitations

The first major weakness of the communication system is the centralised server implementation. While a peer to peer netcode was developed for the program, it required security access from the network to run as a peer to peer program. This obviously means that a sudden interruption of the server instance will shut down communication between all clients. Additionally, there's no means for users to automatically determine whether they've joined the correct server. However, there are no real means to avoid this as this is based on user input without an expected input.

Another weakness of our system is the lack of a Graphical User Interface. With our system being command line based ,it is less welcoming and less intuitive for novice computer users to pick up and use. We had intentions and have created interfases for entering a screen name and chatting with the other members. However, due to complications with implementing the classes into the net code, problems with the chat window interface not responding and due to time pressure, we decided to stick to a Command Line Interface.

IV. Conclusion

In conclusion, our code is able to allow multiple clients to connect to a server. Unit tested, with adequate levels of fault tolerance implemented with modularity. The implementation of the netcode meant the program had to be implemented as a client-server program. Regardless, the program overall acts as a simple, reliable client-server communication service. The server is able to receive and display messages sent to all active clients and With UTF-8 encoding, it ensures that all communication is standardised and understandable to all clients
In future, a peer to peer implementation could be made to better meet the specifications as it proved to be a necessity for various requirements; a better method of error handling user inputs could be implemented also.
Whenever the user uses wrong syntax to launch the program, the program breaks, we could have implemented try-catch statements to process this better.

If we were to rewrite the program again, we would most likely implement better error handling, implement a GUI, and make the program peer to peer.

Acknowledgement
We would like to thank Danny Devito for being an inspiration, specifically in the episode of one of his most famous works "It's Always Sunny in Philadelphia" Season 6 Episode 13.
We also stan Twice's Yoo Jeongyeon and Kim Dahyun, thanking them for the 3+ years of happiness they brought to our lives.

References

Python, n.d. *unittest — Unit testing framework.* [Online]
Available at:
https://docs.python.org/3/library/unittest.html
[Accessed 09 03 2020].