3331 Assignment Report

z5208639, Ali khan

<u>Program Design:</u>

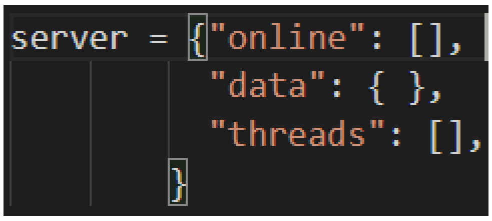
- <u>Language written in</u>: *Python 3.7*
- <u>server.py</u>
    - Usage: python3 server.py server_port
    - Need to be accompanied by *helper.py* in the same working directory. *helper.py* contains utility functions to facilitate the functionality of *server.py*
- <u>client.py</u>
    - Usage: python3 client.py server_port

<u>Application layer message:</u>

The client receives input from the terminal. Important details of the input such as the command inputted, and its arguments are packed into a message of type string. This message is encoded in utf-8 format and sent to the server.
The server will receive this message and split the string into an array to gather all arguments. These gathered arguments will be decoded.

<u>Data Structures:</u>

When the client sends information to the server, and information such as threads creates, users logged in, files uploaded etc… are stored in a dictionary. This dictionary is named *server* as its variable name, and has keys *online* (hashes to an array of users online), *data* (hashes to dictionaries containing names of threads files have been uploaded to) and *threads* (hashes to an array of threads). If a user logs off, their name is removed from the *online* array, and if a thread is deleted, the thread name is removed from the *threads* array.



```
server = {"online": [],
          "data": { },
          "threads": [],
         }
```

<u>How system works:</u>

<u>UDP Commands:</u>

- Authentication
    - Input from client sent to server where the server will open credentials file as read mode to check if the input is present or not. This is the manner in which authentication occurs
- CRT
    - Server will check the data structure to ensure whether a thread (input from client) can be created or not
- LST

    - Server will check data structure and send all elements in the server[“threads”] array to the client. If no threads in the array, a message conveying this will be sent instead to the client.
- XIT
    - Server will remove the user (input sent from client) from server[“online”] array. Server then sends message to client to confirm this occurred.
- MSG
    - Server will check if thread exists in server[“threads”] array. If so a message (corresponding to the input sent by client to server) will be appended to the thread file.
- RDT
    - Server will check if thread exists in server[“threads”] array. If so, server will open up the thread file, and package all of the contents excluding the first line, and then send this package to the client in which it will be displayed at the client terminal.
- EDT
    - Server will check if thread exists in server[“threads”] array. If so, server will then open the thread file and see if a message was sent by the user. If so, server will open the file in writing mode and edit the thread file. Confirmation message sent back to client.
- DLT
    - Server will check if thread exists via the data structure. If so, server opens the file and checks if the message exists and was sent by the user. If so, server deletes this line, moves all lines up one line, and reorganises the numbering if it needs to be done.
- RMV
    - Server checks to see if thread exists via the data structure. If so, server removes this thread in the data structure server[“threads”] as well as initiates a command to delete that file that would be present in the current working directory of the server.

<u>TCP commands:</u>

- UPD
    - Client takes input of a thread file and a file to upload. Firstly server checks if the thread exists and file hasn’t already been uploaded. If so, client collects the data as bytes of the file, sends all of this data to the server in which the server stores it in the data structure.
- DWN
    - Client takes input of a thread file and file to download. Firstly server checks if thread exists, and also if the file to download lies in this thread file. If so, server accesses its data structure to gather the bytes of this file and send all of these bytes over to the client.
    - The client will now receive all of these bytes. Following this, the client opens up a file in writing bytes mode and writes all of these bytes to the file. The filename here will correspond to the filename that was inputted in the command line argument at the client end of the terminal.

Issues with functionality

Issues occurred with the UPD and DWN commands. Whilst these commands work perfectly fine in terms of storing all bytes to the server, and downloading all of the stored bytes at the client and forming the file, it appears that sometimes, when the UPD and DWN commands are inputted multiple times consecutively at the same, client a socket error will occur. I suspect this to occur due to the manner in which I set up my TCP connection, where the same line of code will be read (such as binding socket) if a client executes these commands multiple times, thus causing the error.

All else was tested thoroughly and seemed to work perfectly.

Handling multiple concurrent clients

Multiple clients can be handled concurrently perfectly with the server. This is because whenever a client initiates a command, any important information will be stored at the server end in the dictionary data structure. Any other clients that decide to connect with the server will have shared access of the information. For example, if a client logs in and decided to execute LST, any existing threads created by any other clients can be displayed. The client can then choose to create any other threads, download any files existing in the server etc…..

Ultimately, the server will communicate with multiple clients concurrently and this evidently works well.

Assumptions:

I have assumed that if a correct command is entered in at the client terminal, it will have the correct number of arguments.
If an unknown command is entered in, the client terminal just simply prompts to enter a command again (as a loop) .