COMP3331 Assignment report

Student name: Zhiyang Lin

Student id: z5142124

Program design

Structure of the program:

Files in the **server** folder

- server.py
  - Main server program
- checkCredential.py
  - Helper file for the server program to handle user login authentication checking
- messageProcess.py
  - Helper file for the server program to handle incoming and outgoing packets
- Credentials.txt
  - Text file where all the valid username and password reside

Files in the **client** folder

- client.py
  - A client program to interact with the server program

Application layer message format

The messages transferred between the client and the server are in JSON format
Generally, a ‘command’ attribute must be provided and the rest of the attributes vary among all the available commands. For example, upon the authentication process, the JSON formatted message payload will look like the following

{"command":**'authentication'**, "userName":**userName**,"password":**password**}

Another example will be ‘CRT” which indicates the user wants to create a thread on the server, the JSON message will look like the following

{"command":**'CRT'**, "arg0":**threadTitle** }

How the system works

Quick start guide as the following,
In terminal 1, ‘$’ means the current working directory which contains **client** and **server** folders.

```
$ cd server/

$ python3 server.py 5000 anyPasswordYouLike
```

Note: 5000 is the port number if the port number happened to crash, change it to a different one.

In terminal 2, ‘$’ means the current working directory which contains **client** and **server** folders.

```
$ cd client/

$ python3 client.py localhost 5000
```

Note that the port number must be the same as the server port number as the client uses it to connect to the server.

On the server-side, once the server starts, it will listen to every incoming connection from the client and create a thread for it. Basically, every message sent from the user will be processed by **messageProcess.py** file and the message reply is handled as well.

On the client-side, once the client program starts, it will establish a TCP connection to the server and create two threads, one thread acts like heartbeat signal transceiver periodically checking if the server is alive and, receive and handle the shut-down message sent from the server, another thread is the main thread that will prompt the user to enter the available commands which the application provides, then process them into JSON message and send to the server.

The functionality of multi-client interact with the server concurrently is implemented in this design.

Design trade-offs

1. Multi-threading module is used rather than “select” module or “multi-processing” module as multi-threading is more straightforward to handle the resources allocation among threads, especially for concurrency implementation.

2. JSON formatted message is used to be exchanged between the client and the server, the pros are
    - It is easier to understand and extract information from the message
    - It has straight forward implementation in some programming languages e.g. In python, “dict()” or “{}”

   However, when implementing the command for uploading and downloading files, especially binary files, such a strategy might not be a decent choice, as binary files are already in bytes and ready to be transferred.

3. Only one message process module for the server is implemented where each command could have been implemented as one module but it turns out that one module is sufficed.

Possible improvements

The efficiency of message processing might potential a problem with the design. As now the server will have an infinite loop for each client thread without pausing and update every 0.5 seconds. This is very wasteful of resources where could have been improved with an event-driven technique whenever the server receives a new incoming packet.

Extensions and realisation

Multi-client interact with the server concurrently is definitionally one of them and has already been realised. Another one would sound a bit fancy since now we are implementing command-line based discussion forum, we could think about how might we implement a GUI interface(front-end) of the forum. This can be realised via vue or Angular or any front-end framework.

Does not work under situations

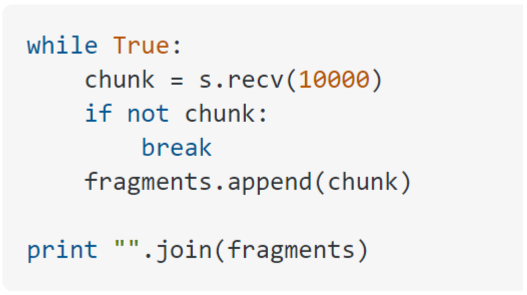
The system does not work in the following situations:

- Lack of credentials.txt in the server directory
- When uploading or downloading the file, it must be present in the directory of either client or server.
- in the event of not quitting the client correctly e.g. crtl + c or crtl +z will cause the server to malfunction.
- Given that the server.py is running in a while loop with time.sleep(0.1), there may occur a race condition and the user might not receive what they expected or even the client program is halted.

Otherwise, the system is working fine under CSE environment.

Segments of code referenced elsewhere

The server.py and client.py programs are designed based on the provided example files from COMP3331 named “Multi-threaded Code (Python)”.



```
while True:
    chunk = s.recv(10000)
    if not chunk:
        break
    fragments.append(chunk)

print "".join(fragments)
```

Referenced from https://stackoverflow.com/questions/17667903/python-socket-receive-large-amount-of-data.This code snippet is used for larger binary file transfer.