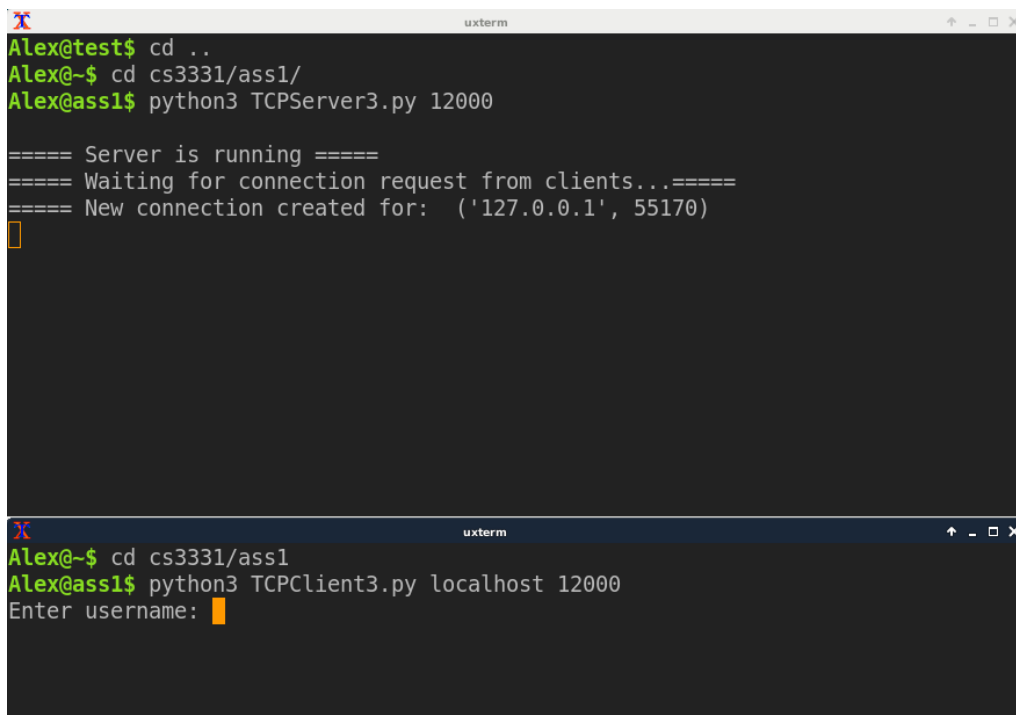


COMP3331 report
student ID: z5115499
name: Alex Piotrowski

I have written this program in python3.7. In order to run and test this program open up two terminals. In the first terminal run “**python3 TCPServer3.py <PORT NUMBER>**” where port number is any number greater than 1024 and is not currently being used by the machine. In the second terminal run the following command “**python3 TCPClient3.py localhost <PORT NUMBER>**” where port number is the same number you used in the first command with the Server.

Picture of how it should look (In this example I used port number 12000 since it is usually always free):



```
uxterm
Alex@test$ cd ..
Alex@~$ cd cs3331/ass1/
Alex@ass1$ python3 TCPServer3.py 12000

===== Server is running =====
===== Waiting for connection request from clients...=====
===== New connection created for: ('127.0.0.1', 55170)
[]

uxterm
Alex@~$ cd cs3331/ass1
Alex@ass1$ python3 TCPClient3.py localhost 12000
Enter username: []
```

Program design

Server: The server is designed to be able to interact with multiple clients at the same time. It creates a new thread for each client that it connects to.

Client: The client goal is to authenticate the user and if successful – pass user commands to the server to be handled. Authentication requests are still sent to the server to be processed.

Application layer message format

For this all messages are of a form “string” – of which it is encoded by the python method before being sent off. Important bits of information are separated by a space. For example

“MSG <THREADNAME> <MESSAGE>“

This makes it easy for the receiver to decode the message and separate important bits of information for processing. In the case for MSG where the message can contain multiple spaces. For example “This is a message” – The whole string would look like this “MSG <THREAD NAME> This is a message” – The message will still get appended to the thread correctly and whole fully.

The Server will only send Error messages or Confirmation message back to the client.

brief description of how the system works

- **The system supports Multithreading However UPD and DWN were not implemented.**
- The system will also not allow the same user to log in twice. This is down by having the server contain an array of users that are currently logged in.

The client has two main loops – the first loop is for authenticating the user. The only way to exit this loop is to receive a confirmation message from the server that your login is correct with credentials.txt. If you cannot login then you can use a new username and it will create a new line in credentials.txt to which you can authenticate from.

The second loop is for User commands. The user can use the following commands: CRT, MSG, DLT, EDT, LST, RDT, RMV, XIT. Any other commands – the server will not understand and will send an error back to the client.

design trade-offs

- Most commands the client will need to append the logged_in_username variable to the message to send off to the server. This is because the server does not know which user sent through the command in the case where there are multiple users logged into the server. This increases packet size by quite a bit.
- It might have been better to be doing argument checking on the client rather on the server. However I do not know if this is scalable. Since it could affect the performance of the client.

Where my program does not work.

- My program does not work with the UPD and DWN commands – I was not able to figure out how to correctly implement these commands in the final week of the assignment.

Declaration of external resources

- No external source code was used.
- Starter code was provided from COMP3331 assignment 1 page.