COMP3331 Report on assignment

By John Dao z5258962

*This assignment was run using **python3** on vlab. It is also shown to be working in screenshots.*

The design

For this assignment, a simple design based around the use of JSON messages was used. I had chosen this method due to the fact that the use of JSON allowed for much easier parsing and the reduction of extra code such as HTTP headers. My own experience with using JSON also weighed in on this decision and as a result, the design relied much on the ease of use of JSON and was used in both communicating from the client to the server and vice versa.

The overall construction of the program was very simple;

1. A client.py
    - Backed with the help of a helper function requestHandler.py
2. A server.py
    - Relying heavily on the help of clientHandler.py

The simple design of the code structure allowed for swiffer code development, though it resulted in increasing amounts of repetitive and long code due to the fact that most of the code was contained in 4 files. Both the client and server were not coupled or dependent on each other in any way as required by the spec and could function by themselves, though with not much purpose.

Ways of communication

Exploring further with my own use of JSON messages, a very basic and familiar use of status codes was introduced to ease the development of both the client and the server. These codes were chosen because of their prevalence, with them already being used in today's transport layers. They include;

1. 404: not found
2. 401: unauthorised
3. 200: OK
4. 100: continue
5. 500: Internal Error/server error

Such statuses were also paired up with 'types' which were unique to my own program. These were used to distinguish certain requests from others and to sort commands out more efficiently. They were labeled much like the commands required by the spec for simplicity and can be found to be used most in the function *clientHandler* within the file *server.py*.

How it works

As what can be implied from the above contents, the program relied much on the use of JSON objects. The majority of the time, the process of sorting functions such like commands went somewhat like the following process;

$$Request \rightarrow Conversion\ into\ JSON \rightarrow Transmission\ via\ TCP \rightarrow Decryption/filtering \rightarrow Function$$

After the use of a function, the receiver of the request would undergo the exact same process but now transmitting to the sender to confirm status. This idea much mimics the original concept of TCP in general with the process of ACKs, SYN and FIN functions and doesn't differ too much from experiences when using APIs much like API's serviced by banks and the like.

The tradeoffs

Though very easy to use and develop, this architecture doesn't come without its flaws. Naturally, due to its need to send many packets, a higher level of latency is increased and combined with TCP's 3 stage handshake and like processes, can hinder performance and reduce the quality of the user's experience when using this application. I have also found that developing with JSON can lead to significant amounts of code repetition during development due to the nature of developing with JSON. The use of JSON also means that there may be a lot of dead weight during the transfer of packets due to null and/or unused keys.

Personally, during the development of this assignment, I have found that the use of JSON has greatly helped with not just the readability and understandability of my own code, but has allowed me to understand TCP at a greater level than it would compared to struggling with parsing XML or using strings.

The shortcomings

With the conclusion of the assignment, I have found myself to be satisfied with the quality of work, however do see some places for improvement. One of such improvements I would make would be the smoother execution of the closure of concurrent connections between multiple clients. This was mainly caused due to the blocking nature of the input method I had used, input(). The function prevented the use of multithreading within the client as it overshadowed all other functions, making even exit() an impossibility during an input stream request. To my knowledge, this is a known fault and I have explored other avenues of attempting to work myself around this, with the use of timers and the like but have come up empty. Within the source code, my efforts may be seen in *client.py* commented and left in case a solution is found. It seems as though that even when the threads are closed, the input() function prevents the client from exiting and leaves the terminal in a state of forever requesting user input (though the use of cntr-c and cntr-d work but that wouldn't be a correct implementation). Some good news is that the SHT function however, works correctly on singular clients.

Conclusion

Development on this assignment has been interesting. With many confusing bugs and great strides made in the understanding of application protocols. Overall, I am satisfied with the experience I have had with this project and hope to further my understanding of networks in the future.

Bonus screenshots of vlab terminal with sample interactions with credentials and files as proof it works