Project 1 Report
Name: Sean Dong
NetID: sd1444

Step 2 Observations:

- In Step 2, I observed that the communication between the server and client was almost instantaneous, compared to when time.sleep(random.random() * 5) was added. The second line of code, time.sleep(5), simply introduces a delay before the "Done" message is sent. The main difference here is that the delay only affects when the "Done" message is sent, not the actual processing of the data.

- The first time.sleep() call (with the random delay) is a precaution to ensure that the server has enough time to fully set up before the client tries to connect. Without this delay, the client may attempt to connect immediately, which usually works, as the server tends to finish setting up just in time. However, if I tried to rerun the program immediately after stopping it, I encountered an error. This is because the port the server was using might not have been fully released by the operating system.

There are two key takeaways from this observation:

- The server might not be fully set up before the client tries to connect.
- The port may not be available if you try to rerun the program too quickly.

Issues and Solutions:

- **Closing the Client Socket:**
    - I encountered an issue where the server would not properly shut down because the client socket wasn't closed on the server side. This resulted in the server being stuck in the while loop, waiting for more data even though the client had stopped sending. I learned that it's important to explicitly close the client socket once the communication is finished to allow the server to exit the loop and terminate properly.
- **Handling Large Data with Byte Buffers:**
    - When receiving data from the recv() function, you need to specify the number of bytes to receive. This caused an issue when dealing with larger data, as it could get split across multiple packets. The solution I found was to initialize an empty string and continually append the data received from recv() until the entire message is received. This way, the data is gradually collected without losing any part of it.
- **Using split() and Line Separators:**
    - Another useful thing I learned is that the split() function removes newlines and spaces by default. While this is useful for splitting data into lines, I had to make sure to re-add newlines at the end when reconstructing the data, especially when handling files or multi-line strings.
- **Sending a signal that we are done sending data:**

    - My recent issue is that when the client finished sending data, it would close the socket however we need to not do that. That is where i found the shutdown method which will tell the server “hey I am done sending information” and it wouldn’t shut down the socket. The reason is that I still needed to receive the data from the server without the client from being closed so the only way I thought this was possible was the shutdown method.

Information used

https://docs.python.org/3/howto/unicode.html
https://www.w3schools.com/Python/ref_string_swapcase.asp
https://www.digitalocean.com/community/tutorials/python-string-encode-decode
https://docs.python.org/3/library/socket.html
https://builtin.com/software-engineering-perspectives/what-is-with-statement-python
https://stackoverflow.com/questions/16678363/how-do-i-declare-an-empty-bytes-variable
https://www.w3schools.com/python/ref_file_readlines.asp
https://stackoverflow.com/questions/409783/socket-shutdown-vs-socket-close