Assessment 6 - Report

- George Wen

What parts were the most challenging across both Assessment 2 and this assessment?

- Error handling is a common challenge across both implementations, the chat program requires robust error handling and there is a need to check every step for error.
- Blocking vs. Non-Blocking Sockets: Blocking calls wait until the operation completes, potentially causing the application to hang if the network is slow or unresponsive. Non-blocking sockets, while addressing this, introduce complexity in managing state and ensuring data is fully sent or received.
- Concurrency: Handling multiple connections simultaneously is a common requirement for server applications. Implementing an efficient, scalable concurrency model using threads, processes, or asynchronous I/O is challenging. Both languages provide primitives for concurrency, but designing a system that avoids common pitfalls like race conditions, deadlocks, and resource contention requires careful planning and testing. It is also worth pointing out that Python as a single threaded language, sharing variables between threads is actually easier than c, albeit it might affect the program's ability to handle high concurrent requests.

Was writing the second client easier, or more difficult? Why?

- Writing in c is more difficult due to the need for explicit memory management, detailed error handling, and the manual setup for data structures for storing the user messages. The lower-level nature of C programming provides more control over system resources and performance but at the cost of increased complexity and more code to achieve the same functionality as in Python. There is also the need to use mutex to lock shared variables across threads.

Did you have any compatibility issues using different languages? Why or why not?

- As long as a client and server adhere to the same protocol (e.g., TCP/IP for socket communication), a client written in Python can communicate seamlessly with a server written in C, and vice versa. The underlying TCP/IP stack ensures that data sent from one end is correctly understood on the other, irrespective of the programming language used to implement the client or server. However one needs to be careful how different languages handle string input and buffers etc. For example one needs to flush the buffer explicitly in c but there is no need to do that in python.