**Name**: Sooseok Kim

**SID**: 50283751

**Email**: sooseokk@buffalo.edu

**Socket setup**

Socket setup process takes basically the same process for both client and server except that client specifies the IP address to connect to, and server takes the host IP adrress to host on.

During the setup, we create a addrinfo struct that contains essential information needed to set up connection. Before assigning any values to the addrinfo, we make sure values in it are empty with memset(). Once done so, we use getaddrinfo() from which we use the first available IP address in the struct to create a socket file descriptor. Once we have a socket file descriptor, we are ready to create a connection between client and server.

**Connect/accept loop**

On the client side, we use connect() to connect to the server. Whereas, on the server side, we use bind() to map a port to the socket, and create a queue that intakes connection requests from clients with listen() function. Once we have established a connection, we can free the address info. Then, client waits for any input with read() function. On the server side, I have provided sigaction functionality which cleans out any zombie processes. After then, the server enters blocking mode where read() function waits for any data sent through the connected socket.

**Chunked read/write**

Both client and server use a buffer of size 4096 bytes to ingest input. Since each connection in this project is meant for one-time use, once we reach EOF (when read() returns 0) we break out of the current loop and ends the connection. As long as there is data to read, it stays within the loop, and starts to send or write data.

**Partical send/write handling**

Client uses a custom function called sendall() to handle partical send. This function takes the socket connection, input buffer, and length of input. Then, as send() returns the size of data sent, we repeatedly compare the size of data sent and the size of, if any, remaining data to send. If there is more data than the buffer can take, it will be handled in the next read(). Likewise, server uses output buffer to calculate the size of written data, and the remaining size of to-be-written data. Then, once the size matches, the job is done, and it closes the connection.

**EINTR handling**

EINTR occurs due to system calls that may interrupts processes in the middle of some task. We need to handle this because while waiting for input, or while reading or writing out output, a system call can

occur any time, thus interrupting the busy process. Thus, we need to make sure that the process continues the job it has been working on to avoid losing or missing any data.

**Testing**

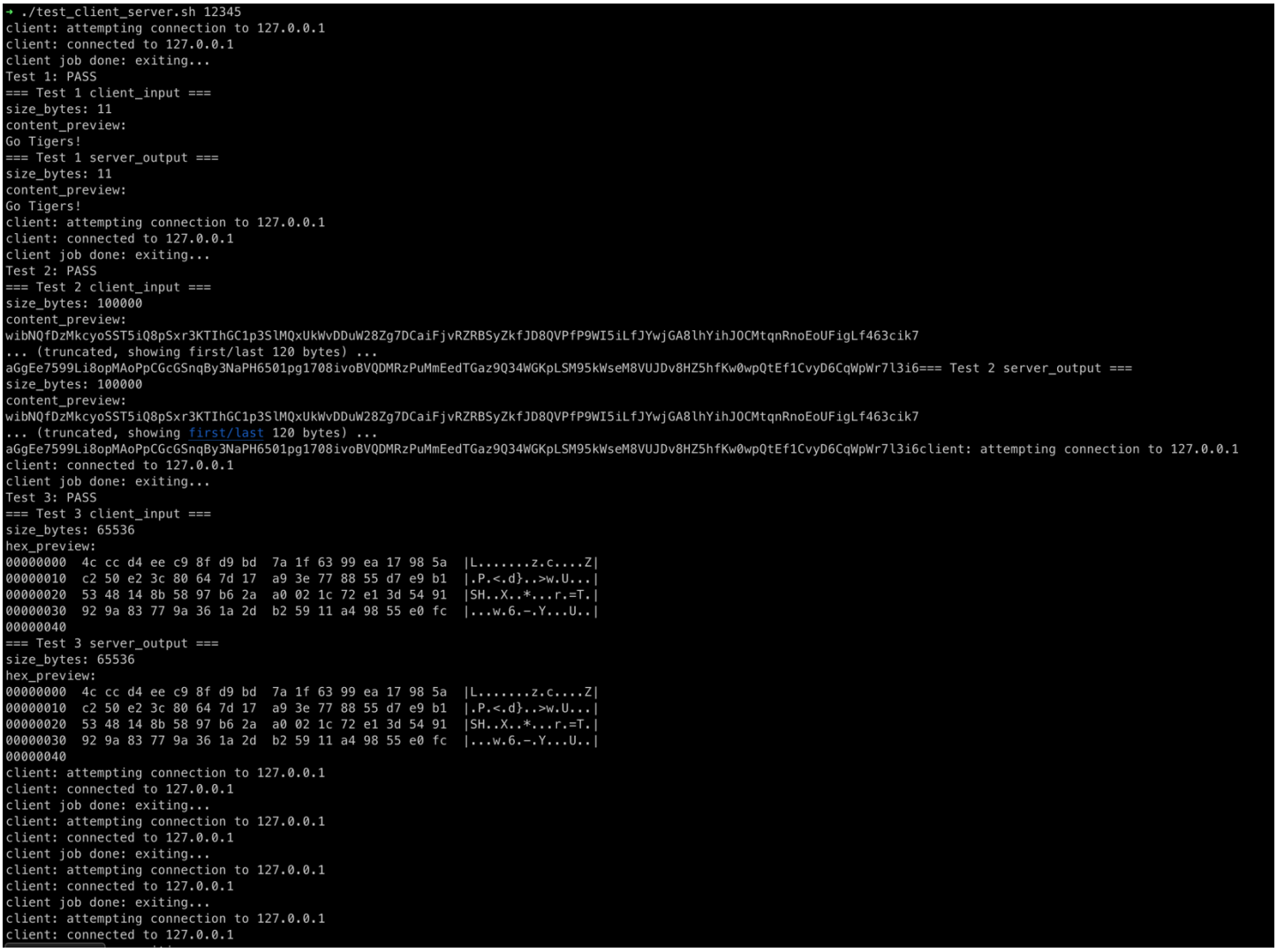
1. Initial socket connection test and transmitting a message

I followed the provided document for most of my project's basic setup and structure, and for establishing the first initial connection.

2. Hello input to print into output.txt

Provided in the project's README.md. Success.

3. test script



```
client: attempting connection to 127.0.0.1
client: connected to 127.0.0.1
client job done: exiting...
client: attempting connection to 127.0.0.1
client: connected to 127.0.0.1
client job done: exiting...
Test 4: PASS
=== Test 4 client_input ===
size_bytes: 30
content_preview:
seq-1
seq-2
seq-3
seq-4
seq-5
=== Test 4 server_output ===
size_bytes: 30
content_preview:
seq-1
seq-2
seq-3
seq-4
seq-5
client: attempting connection to 127.0.0.1
client: connected to 127.0.0.1
client job done: exiting...
client: attempting connection to 127.0.0.1
client: attempting connection to 127.0.0.1
client: attempting connection to 127.0.0.1
client: attempting connection to 127.0.0.1
client: connected to 127.0.0.1
client job done: exiting...
client: connected to 127.0.0.1
client: connected to 127.0.0.1
client job done: exiting...
client job done: exiting...
client: connected to 127.0.0.1
client job done: exiting...
Test 5: PASS
=== Test 5 client_input ===
size_bytes: 95
content_preview:
concurrent-message
concurrent-message
concurrent-message
concurrent-message
concurrent-message
=== Test 5 server_output ===
size_bytes: 95
content_preview:
concurrent-message
concurrent-message
concurrent-message
concurrent-message
concurrent-message
```