**Step 1.1**

We noticed that the server received the number the client sent. Upon response, the client received the same number the server received and sent back.

Link to *recording-step1.1*: <https://youtu.be/YDlI8OPZUOM>

**Step 1.2**

We noticed that the number sent to the server is outputted as 0 on the server side

Link to *recording-step1.2*: <https://youtu.be/W6b7yKsh1Q0>

**Step 1.3**

We noticed that the number sent to the server is outputted as 0 on the server side

Link to *recording-step1.3*: <https://youtu.be/187MFIgQtxI>

**Step 1.4**

We noticed that we are given the correct number of steps back

Link to *recording-step1.4*: <https://youtu.be/4Wl78oeDQ_g>

**Step 2.1**

We noticed that everything ran smoothly. The client sent 7 and received 16 from the server; just as well, the server received a 7 and sent a 16 back to the client.

Link to *recording-step2.1*: <https://youtu.be/jzwVHb5dHck>

**Step 2.2**

We notice the client is trying to send 7 over and over.

Link to *recording-step2.2*: <https://youtu.be/J9hCwJ3yX3M>

**Step 2.3**

Indeed, the Java server received the client and its data with little problems; one such problem being how the bytes received were not read correctly, causing them to be displayed as “0”. However, the client completely failed at reading any incoming integer from the server.

Link to *recording-step2.3*: <https://youtu.be/4njD0QzSqGY>

**Step 3.1**

We observed there was no problem connecting to the server from the client, and no difficulty sending and receiving data back. However, only the number 21 was sent to the server; indicating that the loop finished before any threads sent their corresponding task count.

Link to *recording-step3.1*: <https://youtu.be/ZgDBn10jtkc>

**Step 3.2**

We observed that the client was able to send numbers 1 through 20 to the server and receive the expected 16 back with no problem; consequently, the server received the expected 1 through 20 numbers and sent 16. Additionally, the processes did not report the sent and received values out of order.

Link to *recording-step3.2*: <https://youtu.be/mzRvlT2KVko>

**Step 3.3**

First thing we noticed was only a handful of connection requests were received by the server before having to close the socket because of a bad file descriptor; adjacently, the client encountered many network problems when reading from the socket. The client also sent numbers out of order.

Link to *recording-step3.3*: <https://youtu.be/y4kxKhyMgXs>

**Step 3.4**

While there were issues with how the interaction was displayed (i.e. due to the integers being sent out of order), such issues were minor and did not impede in the success of this client-server interaction; all numbers were successfully sent and received by the server, and the respective iterations of said numbers were successfully sent and received by the client. Now, the main reason why this works is how the server now has its own mechanism to ensure that the concurrent threads do not execute on the same client data. Indeed, through synchronization via mutexes on the server side, each incoming client connection is protected.

Link to *recording-step3.4*: <https://youtu.be/vFpGZCuIGGs>