Jesse Sterchi

Professor Boxiang Pan

CSS 432 A

December 11th, 2022

TFTP Project Report

Steps Completed

For this project, I worked with my partner Narapady, until he later dropped the course in the last few weeks of the quarter. Of course, he helped up until that point, primarily with steps 1, 2 and 3. However, I also worked on these steps with him, and completed the remaining steps as well afterwards. I was able to implement and test every step. I have attached screenshots below for all of them.

Step 1

The first step was to transmit a simple small file. The screenshots below show the server and client output for simple a simple write request.

A screenshot of a computer

Description automatically generated with low confidence

Next, here is a screenshot showing the success of this write request.

A screenshot of a computer

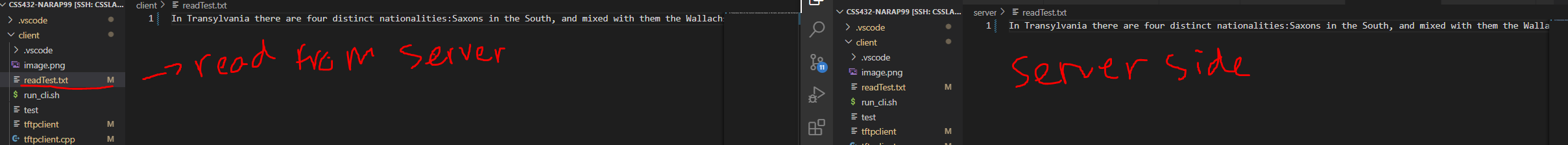
Description automatically generated with medium confidence

For a simple read request, here is the output on both the server and client side.

A screenshot of a computer screen

Description automatically generated with low confidence

Lastly, here is the resulting success of this write request attached below.



Step 2

The next step was to allow for large file transfers. I had some trouble with this step at first, as it took me a bit to figure out the main transmission loop logic for this to be possible. However, it was successful in the end. Here are screenshots to show this success. First, the write request.

A screenshot of a computer

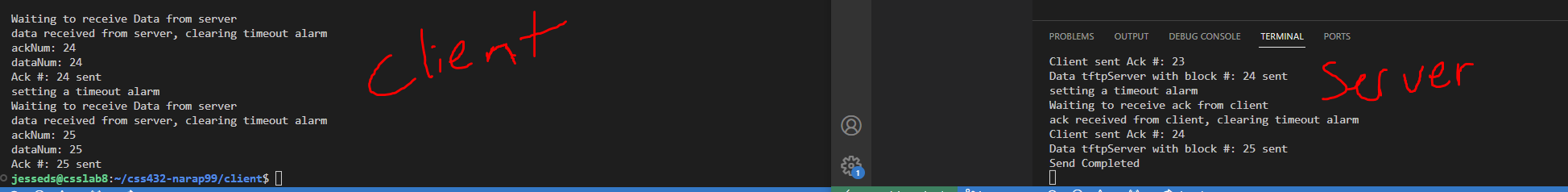
Description automatically generated with medium confidence

Here is the output for the successful write request

A picture containing text

Description automatically generated

For the read request, here are the client and server output.



Here is the output for the successful read request.

Graphical user interface

Description automatically generated

Two clients recovering after the server is suspended while processing their requests.

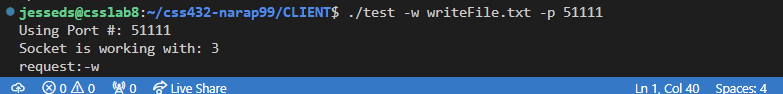
Text

Description automatically generated

Steps 3 & 4

For step 3, we needed to include the option to manually set the port number through the command line. Here are my screenshots for this step.

The server-side:



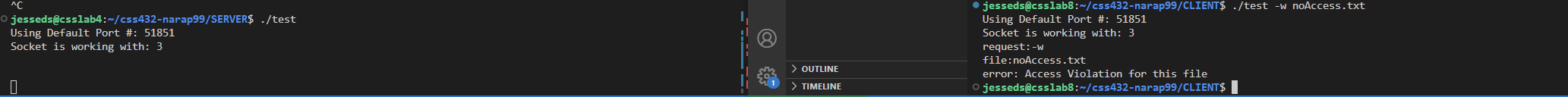
The client-side:

Text

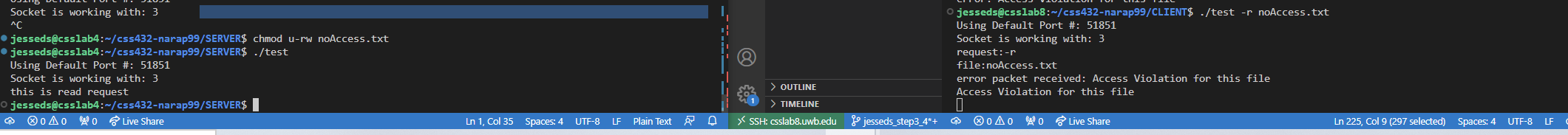
Description automatically generated

I extensively tested all error cases I could think of for step 4. Here are the screenshots for this step.

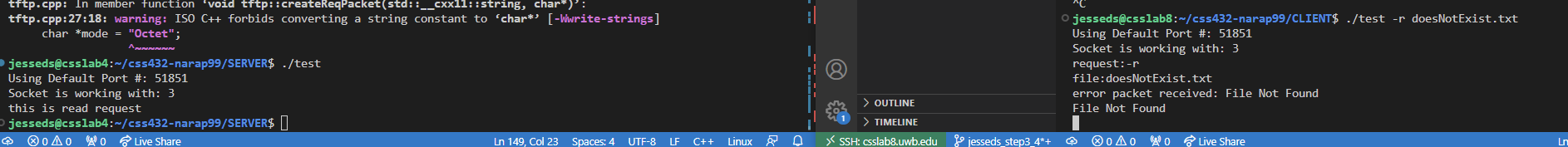
Here is the test case for a client-side access violation.



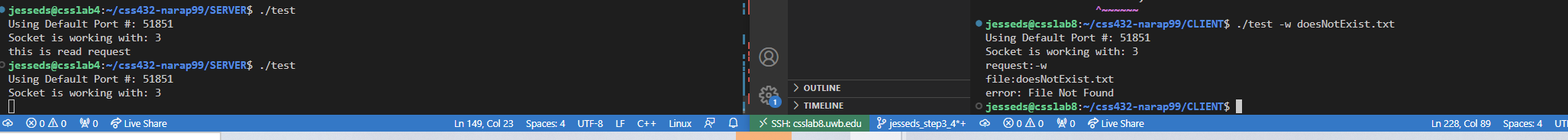
Here is the test case for a server-side access violation.



Next, here is the test case for a read request resulting in a file not found error.



Also, here is the test case for a write request resulting in a file not found error.



Next, here is the test case for a read request that resulted in a file already exists error.

A screenshot of a computer

Description automatically generated with medium confidence

Finally, here is the test case for a write request resulting in a file already exists error.

Text

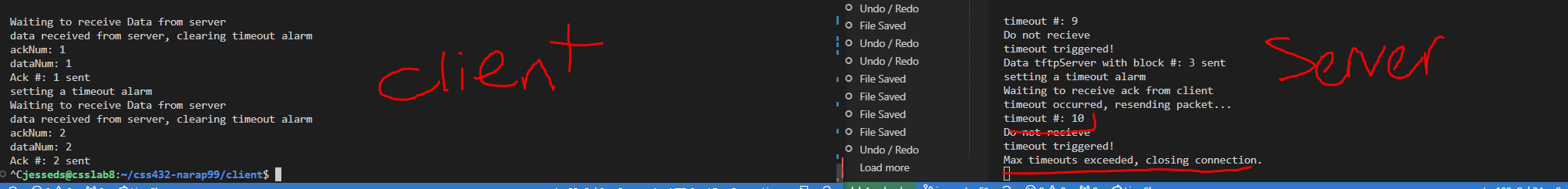
Description automatically generated

Step 5

This step involved implementing proper packet loss handling, allowing for timeouts, and packet retransmission. This functionality is best shown in video form. This video has been submitted to the “Step 5” assignment on canvas and has also been included as an attachment for this submission. This step is fully implemented and working.

I have attached screenshots showing both the server closing the connection after 10 timeouts, and the client terminating after timeouts.

This is the server closing the connection after the client was closed for 10 timeouts.



This is the client closing after the server was closed for 10 timeouts.

A screenshot of a computer

Description automatically generated with medium confidence

Step 6

I had a significant challenge implementing step 6, which was to enable the server to allow multiple clients simultaneously. At first, I tried using p\_threads, but I keep receiving many ambiguous error messages from the compiler, and each time I resolved one, a new one would appear. After fixing all of them, it still didn’t work as intended, and so I tried out using the fork() system call instead.

One issue I had was not assigning my serv\_addr and servlen parameters as references within my receive functions, as it made it impossible for the client to update these values with the new port assigned in the new socket designated to handle that request. I figured this out after checking the address information between these function and system calls, and after making this small change, it appeared to work just fine as intended.

This proved to work much better, and I was able to obtain the desired functionality across multiple Linux machines.

Here is a screenshot showing multiple clients (one with a read request, one with a write request) being connected to the server simultaneously.

Text

Description automatically generated

Additional Test Cases

I ran both the client and the server under many different circumstances various times for testing purposes. Some of the more notable additional test cases I considered and their output is shown below in the following screenshots:

Multiple read requests at once:

Graphical user interface, text

Description automatically generated

Multiple write requests at once:

Text

Description automatically generated

Multiple write requests at once, with one write request file being inaccessible. I wanted to ensure that an invalid request would not interrupt the successful client. My error handling prevented it from ever sending the request to the server, and ultimately acted as expected.

A screenshot of a computer

Description automatically generated with medium confidence

Known Bugs

The only bug I noticed through my testing is that sometimes, my program has a hard time telling the difference between a file that it does not have access to, and one that does not exist. I investigated this, but have been unable to reliably replicate it, and therefore I had a tough time debugging it. This is the only bug I identified and was not able to pinpoint and fix.

Learnings from the Project

Overall, I learned a great deal from this project. It is one experience to learn about a protocol from readings and lectures, and a totally different experience to implement the protocols yourself. I have learned how to create and use UDP sockets from this project, and how to create/manipulate packets. I feel that actually implementing the protocol firsthand has also made it easier to remember and understand important details about the TFTP protocol, as well as the UDP protocol.

I also learned how the transmission flow works between a server and a client using TFTP. Additionally, I also learned a lot about packet loss handling, and signal handling. This was a concept that was very new to me. Finally, I also learned a great deal about how to use threads (theoretically) to enable a multithreaded server to handle many requests, and how to use forking to do the same thing with multiprocessing.

GitHub Link

<https://github.com/Narapady/css432-narap99>