CSE130 ASGN2 WRITEUP

Lang Li

May 2020

1 Testings

My testing involve testing modules individually and testing them together.

1.1 Testing Sockets

I first tested if the sockets are working properly. I used curl on localhost with two console windows opened to check for receiving requests. I then started to make parsers to identify each token of the headers so that I can execute the commands properly. I tried the code Professor Miller gave out on Piazza, but due to Segmentation faults. I abandoned his starter code. I found GeeksforGeeks have similar starter code as Miller's. I adopted that starter code instead and found out that it is easier to work with.

1.2 Testing Requests

I tested requests separately. The GET request is simple, just read the file from the server and write to the client socket, I used curl localhost:8080/test.txt to test if the client successfully retrieved the file. PUT has two parts, first if the file doesn't exist in the server and if the server does exist in the server. If the server doesn't have the file, I need to test both file creation and file writing. If the file already existed, just test if the overwriting process is working. I usec curl -T test.txt localhost:8080 to test if the file is updated or created in the server properly.

1.3 Testing Multi Threading

I tested multi threading in 4 different parts. I first test my arguments pointer is correctly packaged so that my modified GET and PUT can read them. I then tested the threads by having them print out statements individually to make sure the threads are created and functioning properly. I also tested if the dispatcher can distribute the requests to the array of worker threads. I tested the log with offsets to make sure they are writing where they are supposed to. Multithreading seems to work for GET flawlessly, but for PUT is a different

story. It can't seems to handle a certain race condition happening, causing the server to hang on a random file. Logging seems to work correctly, provided if the request is processed correctly as well. My offset calculation seems right from multiple tests.

1.4 Error handling

This section I did not have much trouble with because we know what type of errors we should handle. We need to handle bad header, denied access, and no such files. Server errors Which I found out which one specifically I can use. I used warnx similar to the Dog assignment, and just include the error message where appropriate.

2 Questions

Using either your HTTP client or curl and your original HTTP server from Assignment 1, do the following: (1) Place four different large files on the server. This can be done simply by copying the files to the server's directory, and then running the server. The files should be around 4 MiB long. (2)Start httpserver (3)Start eight separate instances of the client at the same time, one GETting each of the files and measure (using time(1)) how long it takes to get the files. Perhaps the best way to do this is to write a simple shell script (command file) that starts four copies of the client program in the background, by using & at the end. Repeat the same experiment after you implement multi-threading. Is there any difference in performance?

There is a difference in performance. The speedup of the multithreading server vs single-threaded server is around 6x.

What is likely to be the bottleneck in your system? How much concurrency is available in various parts, such as dispatch, worker, logging? Can you increase concurrency in any of these areas and, if so, how?

The most likely bottleneck in my system is the worker threads. Dispatcher only need to handle passing descriptors to workers and the log is passive, only workers perform actions on it. Dispatcher doesn't have concurrency as there is only one dispatcher. Workers have majority of the concurrency as there are multiple worker threads at different stages such as waking up, processing data, and logging. Logs have some concurrency but not much. Log can have multiple worker threads writing at the same time. We can increase the concurrency in the workers by creating more worker threads.

For this assignment you are logging the entire contents of files. In real life, we would not do that.
Why?

Too much overhead, it is unnecessary to log the content of the file when you have the file itself. If there is an error during the transfer of the file, the logging should be able to log the error of the transfer.