Aidan Smith

CSE 130

Writeup for Assignment 2

Testing:

In order to thoroughly test my program I tested all functionality related to assignment 1. To do this I used another terminal and ran random requests to make sure it would respond correctly. I also made sure to test binary files. I then moved on to getting multithreading to work. I did most of my testing for multithreading through the provided test cases because I could not manually create multiple requests at a time without a script. Once I passed all multithreading tests, I moved on to working on my log. I tested my log bit by bit, first writing the header and footer and making sure they worked. I then wrote the body and after testing in my terminal I got it to work. I then tested it on the provided tests to make sure it would work for multithreading. Once my logging worked, I worked on health checks. I played around with it in the terminal and once I felt finished I tested it on the provided tests and it worked.

Questions:

1. Repeat the same experiment after you implement multi-threading. Is there any difference in performance? What is the observed speedup?

   I was not able to get my server to handle 400MiB per file for all 8 files. So I downed the file sizes down to 4MB and the difference in performance is still very noticeable. Just for reference, I have 3 cores dedicated to my VM. When running on a single threaded server, still using logging, I get a time of 4.850s. Now switching to a multithreaded server, I gave it 8 threads, it ran in 0.076s. That is over 63 times faster!? I'm not sure if that is correct, I would have assumed it could've been at most 3 times faster while using 3 cores. I am unsure if there are some other delays caused that I am unaware of. I ran both of my assignment 2 httpserver, only difference being the number of threads I allowed, and I had logging on for both of them.

2. 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?

    On my VM, the bottleneck would be the 3 cores opposed to 8 threads, not all can be run concurrently. In the program itself, the bottleneck could be caused by the mutexes as only one thread may use them at a time.

    This leads into concurrency being haltered in the dispatcher and workers, and the logging between threads. Only one thread may have the client queue at a time, dispatcher vs worker 1 vs worker 2 vs … And when it comes to logging, only one worker thread may have the global offset variable at a time, worker vs worker. These bottlenecks hardly inhibit the program as they are rather small, but as you increase the number of threads, so does the number of threads waiting on the lock increase.

    Increasing concurrency in these areas would be hard since we would need to find atomic operations to replace the need for the locks. Without the need for locks we could run the threads concurrently without any issues. The only other way to increase concurrency would be to add more cores, but that is outside of the program.

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

    Because it takes a long time and really isn't necessary. And on the large scale, our server might go through millions of files, this would create an insanely large log file. The log file is made up of roughly 3 times the bytes of the files that are passed through. This would make the log file grow rapidly in size and it would need a ludicrous amount of storage just to keep it.