Most of my testing involved unit testing with each portion of the code. The way I would test my code was through “curling” multiple responses to my server. I would curl for GET requests and PUT requests initially by having files on each side of the socket and checking each side for the specific file that was supposed to be put there respectively. I would then curl for things such as if a PUT request had no content length and also for error statuses which involved using improper ports and addresses, as well as improper resource names. The one thing that really troubled me was when there were more clients than threads. My code wouldn’t stop reading as it could not obtain the EOF. The way I was able to debug this was through running it through the Xcode debugger. The Xcode debugger was also helpful for threading as I could see what each thread was doing and I could suspend threads whenever I wanted to just look at one. I was able to figure it out and fix it by putting another \r\n at the end of the status messages and then sending those status messages after I’ve written back to my server. Debugging for logging was fairly simple for GET and Error logs since the lines were basically always around the same length. Since PUT logs had varying lengths they were a little trickier. To debug them I would take incremental steps. First I would check if it was properly printing hex numbers and then check if I could print simple files in the required length. Lastly I would check if I could do large files as well as print files that are longer than what was read in from the read calls in my handlePut function.

When I tried to curl four clients to my assignment 1 server vs multithreaded server I get a little bit of speedup. The time for completing the first curl was the same for both but around the 3rd or 4th it was around 1.5-2x faster for the 4th curl client. But overall it wasn’t really that much faster. I think this has to do with how our computer handles doing reading and writing. The

bottleneck of our server is file input and output. The entirety of our server's performance is heavily based on how fast we can read and write to files so if we can find a way to make that faster, then we can make our server much faster. One way would be to use caching. If we cache files that are popular then if that file is requested again, we don't have to travel as far to obtain that file again. The way my server is written, my logging has the most concurrency while my dispatcher and my worker threads have the least. This is because my dispatcher and worker threads have a shared resource which is the queue of client file descriptors. This will obviously slow down my server since threads have to wait before they can manipulate the queue. Logging is a little more concurrent due to the fact that all the threads can log whenever as long as they don't share a space. I could increase concurrency by having multiple threads listening for requests. If we can make it so that the threads have less down time between the queue being empty and clients being pushed into the queue, then we could increase performance.