- Start four 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?
    - For my multi threaded server, sending four requests at the same time to get a 4MB file takes 14.949s. My single threaded server, only does 1 request and it takes it 6.6881s so, if I multiply that by 4 (number of requests) it would roughly equal 27s. So I can indeed see a big difference in performance. The multi threaded server has a much higher throughput than my single threaded server.

- What is likely to be the bottleneck in your system?
    - Definitely handling large files would be a bottleneck in my system. Also, if there are only a few threads created and a large number of requests are sent at the same time, that is a likely bottleneck in my system.

- How much concurrency is available in various parts, such as dispatch, worker, logging?
    - The workers can handle up to N requests at the same time, because that is the total number of available threads. My dispatcher on the other hand, can put a huge number of requests on the queue that my workers will handle N at a time. Logging is done concurrently since I don't block the writing to the logfile for an individual process.

- Can you increase concurrency in any of these areas and, if so, how?
    - Yes, I could perhaps set up some kind of priority queue which could handle all the small files first and then process all the bigger files.