

- 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 4 requests at the same time to get a 4MB file takes 20.658s. My single threaded server, only does 1 request and it takes it 8.688s so, if I multiply that by 4 (number of requests) it would roughly equal 34s. So I can indeed see a big difference in performance. The multi threaded server is 10 seconds faster processing the same 4 requests.
- 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. Finally, logging is a huge bottleneck since I only allow one request to write to the log file at a time.
- How much concurrency is available in various parts, such as dispatch, worker, logging?
 - The dispatcher can handle up to N requests at the same time, then these requests are sent to the worker which can also handle up to N request. Both can handle N request at most because that is the total number of available threads. On the other hand, logging only handles one request at a time since I put a mutex around it that blocks as soon as one process starts logging information.
- Can you increase concurrency in any of these areas and, if so, how?
 - Yes, I could make logging available to other processes instead of blocking it for the other processes as soon as one process starts writing to it. I could do this by getting the total size I need to write a request before I actually start writing, this way I know I where to reserve space and where to start writing no matter if there are other processes writing to the file.