Project 2

Chris Church

Instrumentation

The project design described below is for evaluating the performance of a computer running a multithreaded, boss-worker GETFILE client implementation. Some metrics that can be used for evaluating the performance are the following: **Throughput** (number of requests / time) can be measured by inputting a certain amount of requests in the configuration and recording the time (all times can be recorded by using the system call, *time()*) when the boss thread creates the first request object to include the creating and loading in the evaluation. The stop time can then be recorded once all worker threads have finished and joined the waiting boss thread. The throughput will be the number of requests input divided by the stop time minus the start time. **Response time** (time to complete request / size of request in bytes) can be measured by recording the time for each request when the boss thread begins creating a request object for a particular request, and recording the time when the worker thread handling the request is finished receiving the request file and writing it to the output file. The size of the request can be found in the header response from the GETFILE server. The response time is then calculated by subtracting the start time from the stop time and dividing the result by the file size. For this experimental design, **response time** will be measured for client satisfaction.

Experimental Design

The question being answered is "How does multithreading affect the response time of a GETFILE client requesting several files of differing sizes?" This is the case for many computer users such as computer gamers who frequently need to download multiple files for games and updates of various sizes, as well as students who have to download various sized files for their assignments. The parameters for testing will the number of worker threads, workload request pattern, and the workload request range (in bytes). The test configurations are shown below.

Template: [Worker Threads, Request Pattern, Request Range Low End, High End]

1. [1, MIXED_SIZE, 1000, 1000000]
2. [10, MIXED_SIZE, 1000, 1000000]
3. [1, MIXED_SIZE, 1000, 1000000000]
4. [10, MIXED_SIZE, 1000, 1000000000]

To observe the benefits of multithreading, the configurations are set to either 1 or 10 worker threads. The number 10 was chosen because video game updates usually reside in only a few files (observed with the popular game, Overwatch). This value also works for students downloading assignment files where if the file count goes beyond 10, they are usually zipped. The size range of 1000 bytes to 1 MB is an estimation of the student download workload where the files are often PDFs and DOCX files. The larger range of 1000 bytes to 1 GB was selected based on the popular game Overwatch where new updates may require the download of data files that are up to 1 GB in size. In the case of an April 2018 update, entire game data had to be downloaded due to core changes that resulted in 20 GB total being downloaded.

Machine Specification

The hardware used is listed below

CPU Cores:	6 (no hyperthreading)
Cache Size:	9 MB
Memory:	8 GB
Bandwidth:	1 Gbps (between client and server)

The cores and cache size are based off of Intel's 8600K processor which was their highest selling processor in July 2018 according to the German computer parts supplier MindFactory (https://wccftech.com/intel-coffee-lake-amd-ryzen-cpu-market-share-july-2018/). The memory size was chosen based on 8 GB being commonly found in new PCs marketed to home consumers. The bandwidth was also chosen to be a size that is practical (new NICs are gigabit, most switches sold now are gigabit) and shouldn't create a significant bottleneck.

Results and Explanation

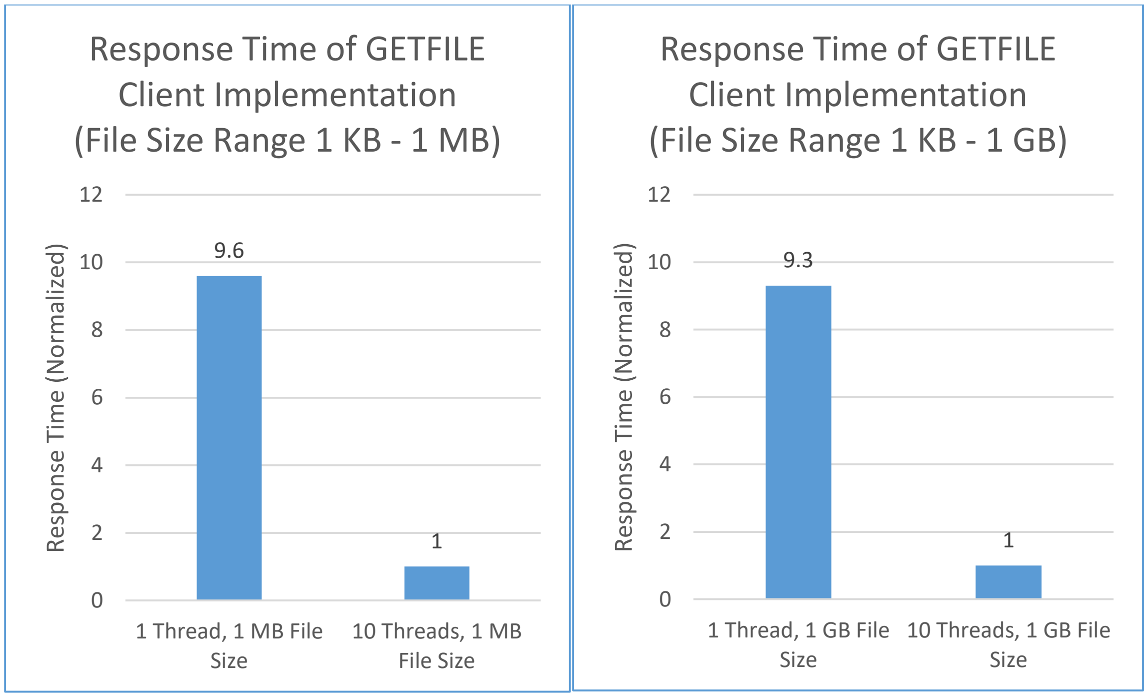

Figure 1: Response times of the 4 configurations where the times have been normalized to the response time of the 10 thread implementations

The results for the four configurations are shown in Figure 1. The workloads had greater than 10 requests which would lead to an expected ten times improvement between the single and multithreaded comparisons. In both, that behavior was observed. The improvement was not quite ten times due to the overhead of creating threads, context switches, and synchronization mechanisms.