Writeup

Rishab Jain
CruzID: rjain9

CSE:130, Fall 2019

1 Testing

To test my program for the multi-threading I created a shell.h which had a mix of GET and PUT commands. I tried to make the files different sizes ranging from really big ones to empty files. I also put in requests that should fail like 400 and 404 errors.

For logging I first tested just the header for both the Success and FAIL requests. Once those worked I then tested converting the ascii text to hex. Lastly, I tested it with the mutex to make sure it was logged contiguously.

2 Comparing Servers

	Single Threaded	Multi-threaded
File 1	Real: 0m0.068s User:0m0.008s Sys:0m0.004s	Real: 0m0.065s User:0m0.004s Sys:0m0.007s
File 2	Real: 0m0.083s User:0m0.007s Sys:0m0.004s	Real: 0m0.077s User:0m0.011s Sys:0m0.000s
File 3	Real: 0m0.087s User:0m0.011s Sys:0m0.000s	Real: 0m0.082s User:0m0.012s Sys:0m0.000s
File4	Real: 0m0.085s User:0m0.010s Sys:0m0.000s	Real: 0m0.083s User:0m0.006s Sys:0m0.005s

.

The multi-threaded server is more efficient as the real clock time is slower than than single threaded. This because multithreading allows the server to process multiple client requests at

the same time. For a single threaded server it would have to wait for the entire process to finish before accepting a new request. The bottleneck of my system is basically the critical region. This is basically when multiple threads are trying to access a shared variable and need one thread to finish before processing.

Dispatch acts like a buffer that controls the amount of threads being used. This can slow the process down because threads have to wait for a thread to exit a critical region. In logging there is a mutex that can cause threads to wait for one thread to finish writing to a file. You can increase concurrency by having a small critical region so that threads don’t have to wait a long time to continue with their process.