Write-up Document for Assignment 2 - Multi-threaded HTTP server

Testing:

The program has three new features: Multi-threading, logging, and healthcheck. A handful of different text files with multiple lines were created, including binary ones. Most test files were stored in the same directory of the executable file. Files in different directories were also created and tested, but the change of directory rarely created an error on its own.

It was decided that logging and health check will be tested first, followed by the health check, and then multi-threads. To test logs, a test like the following script was executed.
Below, assume that the target directory started empty, and our client's content for "sample.txt" is "Hello!"
Input:

```
curl http://localhost:8080/healthcheck.txt

curl -I http://localhost:8080/healthcheck.txt

curl -T client_input.txt http://localhost:8080/healthcheck.txt

curl -T client_input.txt http://localhost:8080/sample.txt

curl http://localhost:8080/sample.txt

curl -I http://localhost:8080/sample.txt

curl http://localhost:8080/nonexistent.txt

curl -I http://localhost:8080/nonexistent.txt

curl http://localhost:8080/healthcheck.txt
```

From reading this script, we can interpret the outcome and create a file with expected outcome.

1st line:	"0\n0\n".	(This was very important)
2nd line:	403 Forbidden	(HEAD on health check)
3rd line:	403 Forbidden	(PUT on health check)
4th line:	201 Created	(PUTS sample.txt in directory)
5th line:	200 OK	(GET the file we just put in)
6th line:	200 OK	(HEAD the file we just put in)
7th line:	404 File Not Found	(GET on non-existent file)
8rd line:	404 File Not Found	(HEAD on non-existent file)
9thst line:	"4\n8\n".	(From above, we have 4 passes and 4 fails)

Generate the following output text file:

```
HTTP/1.1 200 OK\r\nContent-Length: 4\r\n\r\n0\n0\n
HTTP/1.1 403 Forbidden\r\nContent-Length: 10\r\n\r\n Forbidden\n
HTTP/1.1 403 Forbidden\r\nContent-Length: 10\r\n\r\n Forbidden\n
HTTP/1.1 201 Created\r\nContent-Length: 6\r\n\r\n
HTTP/1.1 200 OK\r\nContent-Length: 6\r\n\r\nHello!
HTTP/1.1 200 OK\r\nContent-Length: 6\r\n\r\n
HTTP/1.1 404 File Not Found\r\nContent-Length: 15\r\n\r\n File Not Found\n
HTTP/1.1 404 File Not Found\r\nContent-Length: 15\r\n\r\n File Not Found\n
HTTP/1.1 200 OK\r\nContent-Length: 4\r\n\r\n4\n8\n
```

The logs, similarly, is expected to have the following:

```
GET\t/healthcheck\tlocalhost:8080\t4\t300a300a
FAIL\tHEAD /healthcheck HTTP/1.1\t403\n
FAIL\tGET /healthcheck HTTP/1.1\t403\n
PUT\t/sample.txt\tlocalhost:8080\t6\t48656c6c6f21
GET\t/sample.txt\tlocalhost:8080\t6\t48656c6c6f21
HEAD\t/sample.txt\tlocalhost:8080\t6\t48656c6c6f21
FAIL\tHEAD /healthcheck HTTP/1.1\t404\n
FAIL\tGET /healthcheck HTTP/1.1\t404\n
GET\t/healthcheck\tlocalhost:8080\t4\t340a380a
```

Diff the expected output.txt and the actual output.txt.

If diff returns no difference, then there is no error.

After we know this works, run the same tasks in a shell script at the same time.

When checking the logs again, the execution order may be different, and the health check logs may return a different result. But as long as the readings are consistent with the prior tasks performed, then all nine tasks have executed properly, and thus passing the tests

Question:
What was the outcome of the experiment comparing between single-thread and multi-thread, and what is likely to be the bottleneck.

When performing GET with eight files of 1 gigabyte, the time spent for single thread was 74.591 while multi-thread finished in 67.91, giving a speed up of 9.84%. Also, all tasks in multi-threaded server finished around the same time while some in single threaded finished way slower than others. There is improvement but not very significant. A likely bottleneck of my program could be when the threads are accessing the same files at the same time, while waiting behind a lock for other programs to finish. This may especially likely to happen for logging and healthcheck, as it requires all tasks to access the log file.