WRITEUP:

Testing:
**When I say text files, I don't mean .txt, but files that have text in them, instead of being binary files

1. The first thing I did was try to get multithreading done. Once I finished coding up my multithreading, I tested it in two ways.

    1.1 First, I added a sleep(20) at the end of my worker() function so that I could make sure that my thread was multithreading. I would call curl from different terminals, and make sure that different threads were receiving the different clients. This way, I was also able to make more client requests than there were threads, and ensure that my program would wait for a thread to be available if all the threads were busy.

    1.2 The second way I tested this was with scripts. I wrote scripts that would call curl multiple times with different requests, to see how my program would react if it got a bunch of client requests all at once. My program correctly assigned a different client to each thread, and would wait for threads to finish before assigning them a new client. The program would parse the PUT, GET and HEAD requests correctly. An interesting thing that I noticed however, was that because requests were running on different threads, even when I sometimes called a PUT for a file before a GET, as the PUT request takes longer initially, the GET would sometimes return that the file was found, but there was no content as the PUT had not finished writing data to a file. This was a consequence of multithreading that I decided to allow as you cannot control how fast a thread finishes a task, and I felt that putting a mutex such that requests can only be processed one at a time, would lead to slower multithreading, and somewhat defy the purpose of multithreading in the first place

2. After this, I worked on logging. The tests for logging were similar to tests in section 1.2. I would try a variety of files, text and binary, big and small, and would check if my program was logging them properly. I especially tried files that were greater than 4096 bytes, and binary files, as that was a spot that I struggled with in logging for some time. I finally was able to correctly log both big and small text and binary files after testing for sometime, and on various test cases.

3. After I finished multithreading and logging, I decided to test whether I was reading in arguments correctly before moving on to health check. I called my httpserver, with just a port number first. Then a port number and -N followed by the number of threads, in both orders. Then port number with -N not followed with number of threads. Then I did the same thing with port number and -l. Then I finally did port number with -N and -l, in a variety of combinations and orders. My program correctly responded to the above tests. The last thing I tried was providing an invalid header, such as an extra port number, or other random arguments. My program correctly gave an error.

4. Finally, I made my program accept a 'healthcheck', and tested it. First, I tested the health check without logging, and my program correctly responded with a 404. Then I tested a valid

health check with logging with a GET. I correctly got the right response. Then I tested a PUT and HEAD request on a health check, and I correctly got the 403 error. Health check was simply incorporated with my previous scripts.

5. I also ran my code on a test script provided by a TA, mintest.sh, and successfully passed all the tests. I also ran it on a test script that was a collaboration done by students, and passed all the tests, except for bad.request_4.test, which I believe hung because there was a netcat call that was never manually terminated, as when I ran it on my own, I got the correct response but had to manually terminate the netcat call. I also failed forbidden2.test in that, but that was because the 'correct response' was actually incorrect, and I'm pretty sure that my program gave the right response as the response they gave was supposed to be a 201 Created response, but the file already existed in the same directory, so that shouldn't have been right. Therefore, I believe my test actually passed all of the tests in that script, as the 2 I failed, I seemed to fail for reasons that did not involve my code.

Experimentation and Assignment Questions:

1. With the asgn1 httpserver, each curl request took an average time of 0.073s, with a total time of 0.778s.
2. The total time taken for asgn2 httpserver with 4 threads and no logging was 0.104s. The average time was therefore 0.013s. Though when I timed each curl execution separately, they all took more than 0.013s, when run using the multi threaded httpserver, the commands are able to run in parallel, allowing the total time to drop to 13% of the time for all 8 curl calls to run in the single threaded httpserver.
   The observed speedup is 0.778/0.104 = 7.48 times faster.
3. The first and main bottleneck in the system is the number of threads we can run parallel. This is determined by the caller, and therefore there is not much we can do to change it. The second bottleneck is the critical region where the worker threads receive a client socket descriptor. As I only allow one thread to accept a client at a time, we have to wait for a worker thread to finish receiving a client before another thread can receive the next one, reducing concurrency. Another bottleneck is that the dispatcher can only accept one client at a time, causing other clients that have requests to wait. In logging, the act of logging itself has concurrency, as multiple threads can perform logging at the same time. However, there is a loss of concurrency in the critical region where I get the log offset, and change it's value. This critical region is used to prevent threads from getting incorrect data, and is a bottleneck that I couldn't improve upon.
4. In this assignment, even though we log the data, there is no point to it as we never use the logged data. Therefore, logging the data is simply a waste of resources, as there is no benefit to it. There is nothing you can do with the stored data in this assignment, and therefore, it is pointless. The health check can also be implemented without logging. Logging in my program also reduces concurrency, as there is a critical region where only one thread may get the log offset at a time, and therefore that region acts as a bottleneck.