Natalie Petrosian
CSE 130 Principles of Computer Systems Design
Prof. Peter Alvaro

Assignment 1: Writeup

Testing:

In this assignment, I have implemented *whole-system testing*, which involves running the code in specific scenarios.

The following are several examples of testing the single-threaded HTTP server:

1) GET a zero sized file
2) GET a small text file
3) GET a small binary file
4) GET a large text file
5) GET a large binary file
6) PUT a small text file
7) PUT a small binary file
8) PUT a large binary file
9) Test HEAD request for a proper response header with 200 OK
10) Test if PUT requests receive response code 201 (Created) in response header
11) Test for 400 error on bad resource (request-target) name
12) Test for 403 error on resource without permission (using PUT)
13) Test for 403 error on resource without permission (using GET)
14) Test for 403 error on resource without permission (using HEAD)
15) Test for 404 error for resource that doesn't exist (using GET)
16) Test for 404 error for resource that doesn't exist (using HEAD)
17) PUT effectively overwrites an existing file (generates 201 Created)
18) PUT a zero-length file
19) HEAD a zero-length file
20) Test for 400 error on file exceeding 27 ASCII characters
21) Test for a file consisting of the *allowed* characters
22) Test for 400 error on file without slash
23) Test for a file with more than one Content-Length
24) Test for a file with no Content-Length

Assignment 1 Questions

1. **What fraction of your design and code are there to handle errors properly? How much of your time was spent ensuring that the server behaves "reasonably" in the face of errors?**

About 50% of my design and code implementation accommodate handling errors appropriately. Consequently, I spent roughly 50% of my time ensuring that the server behaves “reasonably” in the face of errors.

2. **<u>List the “errors” in a request message that your server must handle. What response code are you returning for each error?</u>**

   The “errors” in a request message that my server handles are: 400 (Bad Request), 403 (Forbidden), 404 (Not Found), and 500 (Internal Server Error). A 400 (Bad Request) is generated when an error is detected in the command name, an error is detected in the file name, or an error is detected in getting the content-length. A 403 (Forbidden) is generated when the file descriptors for PUT, GET, and HEAD are -1 and the errno is classified as EISDIR or EACCES. Essentially, we are trying to manipulate files that do not give us permissions to read or write to them, thus our commands fail to execute. A 404 (Not Found) is generated when the file descriptors for PUT, GET, and HEAD are -1 and errno is classified as ENOENT. Essentially, we cannot ask the server to generate information with GET and HEAD when the file does not exist. All other errors that do not meet the criteria of the previously listed error response codes are handled with 500 (Internal Server Error).

3. **<u>What happens in your implementation of, during a PUT with Content-Length, the connection is closed, ending the communication early?</u>**

   If the connection handling a PUT with Content-Length is ended prematurely, a 500 (Internal Server Error) is generated.

4. **<u>Does endianness matter for the HTTP protocol? Why or why not?</u>**

   Endianness matters when you read() and write() to an external socket, especially when binary data communication is happening between machines with different architectures. Therefore, it does matter for the HTTP protocol.