Write Up: multi-threaded httpserver with aliases

Nixon Duong
CruzID: niduong

Unit Test:
handlePutRequest() **-** Tested individually to see if the subroutine could handle bad requests and proper requests. Tests were performed by passing the function with requests that were of status code 200, 201 and 400. Then checked if the response to the client was of the correct status code.

handleGetRequest() **-** Tested individually to see if the subroutine could handle bad request and proper request. Tests were performed by passing the function with requests that were of status code 200, 400 and 404. Then checked if the response to the client was of the correct status code.

handlePatchRequest() - Tested individually to see if the subroutine could handle bad request and proper request.

createNewAlias() - Tested individually to see if the subroutine could successfully add the new alias into the mapping file

followAliasToObject() - Tested individually to see if the subroutine could successfully read the mapping file to follow aliases to an object

Black Box Test:
Tested to see if the system behaved the way it should. In this test, I focused on behavior, rather than internals.

Explain the difference between fully resolving a name (to an httpname) when the name is created and the approach that you're taking for this assignment. Give an example of when it might be useful.

The difference between fully resolving a name to an httpname when their name is created is that httpnames add a layer of indirection. I added this layer of indirection in this assignment by keeping track of aliases in a mapping file. This could be particularly useful when you want multiple people to have a reference to your file. You could give them each different aliases to the same file without ever giving them your actual file. You could also give each reference a different level of privilege to increase security as well.

What did you learn about system design from this class? In particular, describe how each of the basic techniques (abstraction, layering, hierarch, and modularity) helped you by simplifying your design, making it more efficient, or making it easier to design.

I learned that computer systems are extremely complex and that it's impossible to create a perfect system. Everything about designing a system is a tradeoff. Although system design is complex, some techniques make it simpler. By keeping your design modular, you make it easier to modify one component without the fear of affecting others. By having abstractions in your code, you don't repeat yourself. Layering creates a hierarchical order in your program. My key takeaway of this class is that I need to get better at documentation and designing. Documentation and design makes coding easier and more efficient. Building a robust system does not come overnight. It takes a lot of time to think about all the possible conditions your system could be in.