Maiah Pardo, mapardo
11/23/21

WRITEUP for asgn3, httpproxy

Testing: For this program, my testing consisted of writing my own test files which:

- created files of different size (ranging from 1-10 MB) from dev/urandom
- First I began testing with one httpserver port, and went up to 20 different server ports.
- I tested that the result of GETs for different files had the right contents (using diff)
- I tested that the loadbalancing by writing tests to send GET requests to the proxy and determining beforehand how many GET requests should be written in the logfiles of the httpservers and compared the final result on the log files.
- I tested the caching by writing tests to send GET requests to the proxy and and checked the logfiles of the httpservers to make sure it was resulting in the correct number of HEAD and GET requests. I then compared the output of the file to the original file (using diff) to make sure my caching was writing the correct contents to the files.
- I ran the student provided asgn3-test.py file.

Questions:

- A more realistic implementation of distributing load (instead of choosing the least used server) would consider performance attributes from the machine running the server. This was not used in this assignment because we (the students) are all creating and testing our proxy on our own machines, so it would be impossible to consider performance attributes based on the machine which is testing our code that we do not have access to while implementing.

- Experiment 1: Create 8 400 MiB files and start eight separate instances of the client at the same time, one GETting each of the files and measure (using time(1)) how long it takes to get the files.
    - Time: 1.108 seconds
- After repeating the experiment with one server turned off, this took:
    - Time: 2.477 seconds
    - Yes there is a difference in performance. When turning one server off, it took over 2x longer to process the same requests.

- Experiment 2: Using one file created from the previous experiment, start the provided server with only one thread and (starting the proxy without caching) request the file 10 times.
    - Time: 1.362 seconds
- Repeat the experiment with caching on.
    - Time: 0.0687 seconds

- What I learned about system design:
    - Abstraction helped me simplify the design for the httpproxy in particular, since I reused my linked-list implementation that I created in CSE 101 in Fall 2020. This

helped me to handle complexity by hiding unnecessary details while implementation the caching. For example, when deleting an entry in the cache, what was happening behind the scenes (in the List object) was freeing data and re-routing the pointers between nodes in the linked list. I added a few changes in the List file to implement the caching the way I wanted, but I reused much of the code I had previously written. In my httpproxy file, I did not have to deal with or think about any of the 'behind the scenes' of the list, which also helped by simplifying the design, reducing the work I had to complete for this project, and made the project easier to design. I also created a simple queue data structure which helped to reduce my code's complexity by simplifying the design.

- I used modularity by creating separate files (one for httpproxy and it's functions, one for the queue, and one for the List implementation). This helped me break my code into separate parts which ultimately made my code easier to design, read and debug. I also broke my code up into portions (functions) when I knew certain portions could be reused in different parts of the program. In this way, I also had cleaner code and less to debug, but much less code in the file then would have otherwise been.

- Layering and hierarchy helped me by organizing the program into separate functional components that interact in different ways. For example, many of the functions I wrote only interact with the one that calls it. In this way, the path that the data travels in my program is much easier to track, and my code is clearer and easier to understand.