Writeup for `loadbalancer.c`

Jay Montoya | jaanmont | 1742317

Testing Utilized:

- As with the previous assignments, a wide variety of curl commands were used with shell scripts to help test changes faster.
- The actual load balancing algorithm took a large amount of time to develop, as it required extensive testing of edge cases. This was more difficult to test, as I had to develop a plan for the amount of clients I was giving the load balancer, as well as validate the outcome of each experiment.
- A-lot of time was spent validating my terminal output, ensuring that events happened in an acceptable order within the load balancer.
- If I had more time, I would have developed my skills more on bash scripting to create more concise tests. However, I was still able to pass all tests given minimal bash scripting and terminal monitoring.

Questions:

Why didn't we consider performance attributes of the machine running the server for this assignment?

Since we don't have access to multiple machines with possibly different performance specs, we are limited in our implementation of health checking for this assignment. I'd also say it's because of the class time constraints: Implementing a health check in the previous assignment would have taken way longer if it was anything beyond the total number of requests to the total number of errors. That's not to say that considering these attributes shouldn't be done, as I think they should absolutely be considered in a real life implementation.

What improvements could you achieve by removing the restriction of not processing client requests. What would be the cost of those improvements?

If we removed this restriction, I would include some minimal HTTP processing to the client's request to ensure that at least something resembling HTTP is given to our slave servers. The cost of this improvement is obviously complexity within the load balancer, but I think it could be reasonably worth it if we were trying to limit the amount of garbage penetrating our system.

Experiment:

How long does it take to GET the 8 files with 2 http servers?

When starting 8 instances of the client in the experiment, both servers sigpipe error and exit. For each of the 8 files, only about 90% of the contents are transferred. After reviewing these results with a few classmates, this seems not uncommon, and perhaps may have something to do with the handling of sockets in our implementation, or simply how the given httpserver binaries perform under pressure. The total amount of time passed in the experiment is 24 seconds.

How long does it take to GET the files with one of the servers replaced with nc?

Since nc does not respond to the health checks issued by the load balancer, that port is marked problematic from the start, and the 1 instance of the httpserver is left to handle the clients. The same thing occurs, with only a fraction of the files being actually transferred. However, the sigpipe error occurs in half the time, 12 seconds.

What do you observe?

My results in this experiment speak to the concept of load balancer resilience. It doesn't matter what is listening on the server port, my load balancer responds appropriately in the face of an unresponsive or down server. If I had more time on this assignment, I'd look into the interface of connection forwarding more, as I think progress can be made in the area of truly ensuring that a server is unresponsive.

I'd like to additionally note that my server does handle the transfer of large files appropriately and passes all subset tests on GitLab.