Navika Gupta
Design DOC

Intro and purpose:
The goal of this program is to create a load balancer that forwards requests to http servers. It will forward requests to the least loaded server. If two servers have handled the same amount of requests it will break the tie using whichever server has the most successful number of health checks.

Requests will be structures like:
./loadbalancer -N 7 1234 8080 -R 3 8081

You will send requests to the first port in the load balancer arguments. So for this example you will send requests to the port 1234 as such:

curl -s -T FILENAME http://localhost:1234/FILENAME

and they will be forwarded to parts 800 and 8081 that should have http servers running.

The other parameters in the argument -N represent the number of concurrent connections that can take place. While, -R flag represents the number needed to perform a health check.

Underlying Structure:
I used a similar approach as the last assignment to multithread my loadbalancer. The threads were implemented using the base structure of a queue where incoming threads were queued up and once there was a free thread it we dequeued it and assigned it to the thread. The mechanism of dequeuing and enqueuing threads were locked with a mutex so that two queues could not dequeue or enqueue the same request.

Mutex
In my program I used mutexes in a few places. Whenever a variable that needed to be used by other threads was modified it would be locked with a mutex. In my program this included dequeuing and enqueuing incoming requests keeping a global total of the total number of requests and fails and incrementing the offset when I printed to a log file.

Load Balancer:

For the load balancer, my main structure was an array of a struct of ports. This arry held a few key pieces of information about the port.

```
struct tuple
{
   int port;
   int count;
   int fail;
   bool alive;
   bool valid;
} ;
```

The port itself is the port number, the count is the amount of requests that it has handled, fail is the number of requests that the server has gotten a failed response from. Alive and valid both check if the server is responsive and is used to see if connections can be sent to that server.

1.0
Parse the arguments of load balancer using get opts

> **1.1**
> Create an array of tuples and assign values to the ports with the values parsed
>
> **1.2**
> Create N number of ports to handle incoming connections

2.0
Initialize values in the array of tuples such as count and valid

3.0
Queue request on the queue as they come in

> **3.1**
> Accept requests when a thread becomes available

4.0
Handle that request

> **4.1**
> Pick the best port to send the connections to using pickPort() functions
>
> **4.2**
> Check if connecting to the port is successful
>
> > **4.2.1**
> > If not call pickPort() again

4.3
Bridge a connection between the chosen port and the client.
4.4
Increment the number of connections handled
4.4.1
If the connection is divisible by the value of R then call for a health check of all servers

PickPort( ):
The way that I pick the best port to forward requests to is by giving them to a valid server that has received the least amount of requests. I check what is the smallest number of requests received by all the ports that are valid. I then cycled to see which ports the number of requests equaled the smallest. If there are multiple you compare the number of failed requests by each port and you pick the post that has the lowest amount of failed requests. If no ports are valid, pickPort() returns a -1 value wich will then trigger a 500 response.

Healthchecks:
The way the health checks are handled, you first check if the server is valid. If it is valid then perform a health check on it. To do so you send a request the will look like
GET /healthcheck HTTP/1.1\r\n\r\n
To the port in which you would like to perform a health check. You then check if a response was received in a set amount of time. If no response was received set the server as invalid. If not parse the response. If at any point parsing the response is wrong then also set the server to invalid.

500:
If at anypoint all servers are down and the pickPort() function is unable to return a valid a port, then send the client a 500 status code message the should look like:

"HTTP/1.1 500 Internal Server Error\r\nContent-Length: 0\r\n\r\n"