Aidan Smith

CSE 130

Design for Assignment 3

1.0 Introduction

This assignment is to work as an extension of our multithreaded server from assignment 2. It is a load balancer that allows multiple instances of httpserver running and connects them to incoming clients. This is to take another step towards increasing throughput, by increasing our servers we can handle more clients.

1.1 Goals and objectives

Load balancer will be able to connect multiple servers to many clients.

1.2 Statement of scope

Loadbalancer is an executable that runs a server that connects clients to httpservers. It can be run with different options, this is expanded on in the README. Clients will be able to connect to this server through curl commands and the port of the server as if they were interacting with a httpserver directly.

2.0 Structs used

I used two structs in order to send a group of information between functions so that each function could read from and write to it as needed.

2.1 struct servers servers

This struct is the main struct, which contains the following struct as an array. Servers holds all relevant information to the load balancer, including the number of httpservers, and the time and requests in between health checks. All information here is relevant to the entirety of the load balancer. It also contains mutexes and conditions so that the health checks are not performed too rapidly and that not too many threads are created. This struct is used throughout the program as it has lots of information needed for the load balancer.

2.2 struct serverStats serverStats

This struct is the secondary struct which is used to hold all information relative to each httpserver under the load balancer. It contains the port of the httpserver, a boolean for if it is alive or not, and the number of fails and requests it has completed. This struct is used when interacting with httpservers, which will be expanded on in the functions section.

3.0 Design

This design utilizes the template given.

3.1 loadbalancer.c

Loadbalancer.c is made up of seven functions. These will be expanded on in functions. They are used from taking in the client to sending its request to a httpserver to sending that response back to the client. The functions are also used to keep tabs on the readiness of servers.

3.2 Makefile

Makefile is used to run httpserver.c cleanly, its usage is described in the file.

4.0 Functions

Load balancer is a server and client, thus it needs to be able to make connections with clients of its own, and behave as a client to httpservers. These functions make sure the server can do so.

4.1 int main(int argc, char* argv)

This function begins by reading in the arguments from the httpserver initializer. It sorts them out using getopt() and saves them in the structs. It initializes the mutexes and conditionals, and the structs. It then creates the server that clients will connect to. It then creates the health checker thread which will run infinitely. After this it now hits an infinite while loop, used as the dispatcher thread. It will wait for a client to connect, and will give it a thread. If no threads are available, then it waits. It then loops back around and waits for another client to connect. Threads are created here and will be destroyed after they are done.

4.2 void* threadFunction(void* acceptfd)

This function is the thread function. It will be created by int main() and destroyed in its one server call life time. It will take the clients message and pick a server to send it to based on the criteria of best server. Criteria of best server is: least requests, if tied then least fails, if tied then random. Once it picks the server, it opens the communication channel between the server and client, allowing them to do as they need to. Once communications are finished, the thread dies off.

4.3 void *healthCheck() {

This function is a thread that is called in int main(). It periodically runs its code, either if waiting for X amount of time or R requests are made. It makes a request to every server to see if it's alive, and the number of requests and fails. It's lifetime is the same as the load balancer's lifetime.

4.4 int client_connect(uint16_t connectport) {

This function takes the port of each server and returns the socket number of it. This is used for communicating with any of the servers, called every time a client wants to make a request. The server socket is closed to save memory when not in use, hence it being called every time. This function is called in threadFunction() and healthCheck().

4.5 int server_listen(int port) {

This function is used to find the socket of the load balancer server. Using this socket, it is able to see all connections attempted to be made to it and accept them. This function is called in int main().

4.6 int bridge_connections(int fromfd, int tofd) {

This function is called by the following function. It is used to pass information between the server and the client.

4.7 void bridge_loop(int sockfd1, int sockfd2) {

This function creates the connection between the client and the server assigned to its request. It is called in the threadFunction() after the best server has been selected. This function then calls the previously mentioned function to pass information over this created connection.

5.0 User interface

This is described in README.md.