DESIGN HW3
Andi Zhao

Goals

The goal of this assignment is to make a load balancer that could manage incoming client connections and distribute the connections to a number of servers in a least-connected scheme. In the case of a tie, the connection should go to the server with the highest success rate. The load balancer should accomplish three things:

1. Handle multiple client requests at once
2. Perform periodic health check probes to all the servers
3. Assign a request to the server with the least total requests
4. Bridge the connection between actual server and client and send data appropriately

Design

Multithreading

Similar to assignment 2 the load balancer needs to be multithreaded. A queue implemented as a linked list. The load balancer threads will take work off the queue and distribute the work onto the servers. Because multiple threads could look at the server queue at the same time, I would have to lock the servers queue as well at each access.

Sending a Health Check

The load balancer must periodically send a health check to all servers and stop sending requests to servers that are down. The load balancer must send a health check to when either a request number is reached or a certain amount of time has passed. The health check will be done with a separate thread, which will use a timed wait function. If the wait is signaled before the timeout, an update to the server status will be issued meaning the number of requests was reached. Otherwise, it would mean that the timer was reached, so the update occurs regardless.

Health check to all servers

The load balancer will have a global table that only the healthcheck function can edit. When either R requests are processed or X time has passed, the health check function will broadcast healthcheck to all servers

and parse the replies. It will set all servers with valid responses to working and all others to down state.

Local access to all servers

The load balancer has access to a list of server structs. Each call to health check will refresh the server list and update the status of each of the servers.

Load Balancing Interactions with the client

To the client, the load balancer is just a server. As such, it will act very similarly to the server in HW2 to the client. The client will send a request to the load balancer and will receive the proper response. The load balancer would not need to log the client requests as the server in HW2 did for each client request. The load balancer will receive client requests in a queue

Load Balancing interaction with server

Assigning Client to Server

When the load balancer gets a client request, it will find the minimum connected server and send the request there. In the case of a tie, the client will send a health check request to the set of servers in question and compare their success rates. The load balancer will then send the request to the higher success rate server.

Getting the data from the server

Getting data from a server is done through the load balancer. The loadbalancer will act as a server to the actual client and a client to the actual server. It will gather the two file descriptors and use select to probe for data on any of the sockets. If there is any data from one socket, the load balancer will send the data to the other socket.

Bridging the connection

To bridge the connection, my program will have a function that takes in two socket file descriptors. Since the load balancer is a two way channel, there are two possibilities.

1. Connection sends data from server to load balancer
2. Connection sends data from load balancer to server

LoadBalancer to Server

In this case the client is sending a request to the load balancer. The load balancer needs to forward the same request to one of the servers. This is usually just the header information for a request. However, in the case of a PUT request, the client may send less than anticipated at once.

LoadBalancer to Client

The load balancer acts like a server to the clients. During a get or put request, the load balancer will read some data from the server and send that data back to the client in case the client is waiting for some response from the server before sending more data.

Data Structures

There are two data structures used in this project.

1. Queue
2. List to keep track of state

Queue

Similar to assignment 2, the multithreaded aspect of the load balancer is handled with a queue. This queue is shared between the dispatcher main thread and the worker threads. The nodes of this queue will contain the client file descriptor and the server fd that the request was assigned to

List

This list is used for the dispatcher thread to keep track of the health of all the servers. The list consists of server structs that contain the server fd, server status, failed connections and total connections.

Use Cases

The load balancer should be able to handle multiple requests similar to how the HTTP server did in the previous assignment. although the load balancer does not actually handle any of the requests, it must accept all types of requests from the client and send the server response to the client.

User sends GET request

In this case the load balancer will read the header and send the header to the designated server. When the server has data to read, a thread from the load balancer will read the data and write that data directly to the client.

User sends PUT request

In this case the load balancer will read the header and send the header to the designated server. It will send all data received from the client to the server. In this case, the program assumes that the client will send all the data at once for PUT requests.

User sends HEAD request

In this case the load balancer will read the header and send the header to the designated server. When the server has data to read, a thread from the load balancer will read the data and write that data directly to the client.

User sends other requests

The server will handle invalid requests as appropriate and the load balancer will forward the response to the client.

Functions

Thread function for health check

Input none
Output none
This thread function is used to keep track of when to run the healthcheck function. There are two conditions to run the health check function.

1. The number of requests is equal to the request threshold
2. Sufficient time has passed since the last healthcheck. This time is reset by the first condition

logic

There are two ways to implement waiting without busy waiting. One method is to use sem_timedwait, which uses a semaphore to keep track of the time. The other is to use a conditional variable timed_wait. As there is no difference in the two methods and the previous program was done using conditional variables, this program will follow suite.

In an infinite loop, the thread function calls cond_timedWait with a timeout set at 3 seconds. If the condition variable is signaled, meaning the first condition is satisfied, it will run the updateTable function. If the return value is -1, meaning the second condition is satisfied, it will run the update table function

Update Table

Input array of server structs
Output void
The update table function goes through all the servers and sends a health check to each. For each of the servers there are three possibilities

1. The server does not respond
2. The server responds appropriately
3. The server responds but with a malformed response

In the first and third case, update table will simply change the valid attribute for said server to false, and thus eliminate the server from balance contention. In the second case, it will update the server's failed connections and total connections. If the server was previously invalidated, the function will also change the valid bit to true.
Logic

After sending a health check to all servers, parse each response with a time out of 2 seconds. If the response is malformed, change server status to false. Otherwise, set server status to true.

Bridge_Connection

Input two sockets
Output void
Bridge connection connects two servers and reads from fromserver and sends the data to toserver
Logic

In a loop, read form fromserver and at the same time, write the contents of the read to toserver. Bridge connection will end when fromserver has no more information to read.

Thread function for handling requests

Input loadbalancerfd
Output infinite loop
Logic

This function works very similarly to the worker threads of HW2. In an infinite loop, try to take work off the queue. If there is no work go to sleep

and wait to be woken up. If there is work on the queue, call connection handler to handle the connection.

ConnectionHandler

Input client*
Output void
Logic

The connection handler function will take the pointer and extract the portnumber and clientfd from the pointer. It will then connect to the port number as a client. With the connection, the function will begin to transfer all information from the server to the loadbalancer.

FindMin

Input void
Output server struct
Logic

FindMin will find the best suited server currently up for a given request. FindMin does not need to know which request it is trying to find a server for. The function will loop through the scoreboard and look first for the server that is up and has the least number of connections. If there is a tie, min will compare its current error count with the server's error count. If the server's error count is smaller, min will set itself to that server. At the end of the loop, the function returns min.

Wait for response

Input server socket
Output integer flag
Logic

Wait for response uses select to determine if the server has anything to read after a set amount of time. If at any point there are things to read, the function returns 1, else it will return -1.

Parse buffer

Input char* buffer, size_t total size_t error
Output void
Logic

Parse buffer parses the buffer at buffer for two positive integers and stores them in error and total.

The Main Function

The main function sets up the number of threads and the number of requests to wait until the healthcheck probe. It uses getopt to look for arguments -N and -R. And assigns threads and requests appropriately. The main function will also create an array of structs representing a server object. It uses optind to find the servers that the loadbalancer forwards requests to and puts them in the array. When a request comes in, it calls findMin to assign the request to a server and puts that struct in a queue.