I. Introduction

Implement a load balancer that will distribute connections over a set of servers.

Discussion: A shared mutex is ready. With the shared mutex locked, common resources will not be accessed by different threads at the same time. There's a lock around when distributing the incoming requests and a lock around when performing periodic health checks. Between threads, there is a global queue to store incoming requests and a shared variable to count the total processed requests (R) for periodically probing health checks. New HTTP messages are enqueued in main and dequeued in the worker thread; total processed requests (R) is updated(+1) once a message is processed and set to 0 once a health check is completed. Critical regions are the moments dealing with queue and performing health checks.

Detection of Server Connections: Health checks of the servers will be probed periodically. Connections to servers will be established via function *client_connect()* and a health check request will be sent to each server. If the server does not respond within 5 seconds or does not respond with a status code of 200 in a correct format, the server will be marked as down.

Usage	Functionality
./loadbalancer -N	number of threads
./loadbalancer -R	number of requests between two health check probes
./loadbalancer 1234 8080 8081...	port number of the load balancer and servers

II. Functional Design

Data Structure:

char arrays will be used as buffers to temporarily store HTTP requests and response.

Struct *Queue* will be used to store incoming HTTP requests.

Struct *ServerInfo* will be used to store the information of the each server.

Struct *Servers* will be used to store the information of all servers.

Function Design:

Functions *int client_connect(connectport), int server_listen(port), int bridge_connections(fromfd, tofd), void bridge_loop(sockfd1, sockfd2)* are from instructors' starter code, which are for connecting the load balancer and server.

void do_periodic_healthcheck():

- This will send a health check request to the server, receive health check response, and update the latest data on each server.

*void init_servers(argc, **argv, start_of_ports):*

- This will initialize all the servers information based on the command line argument, and perform an health check to each server to check the status.

int determine_port():

- The port with the lowest total requests and error requests is prioritized. If all servers are down, -1 is returned.

void worker_thread(*obj)*:

- This will be the worker thread of the program. Once it receives its condition variable from dispatcher thread, it will start to process the requests.
- This will get the client socket from the queue and process the request with the functions described above. Once it finishes the process, it will send a signal to the dispatcher.

void healthcheck_thread(*obj):*

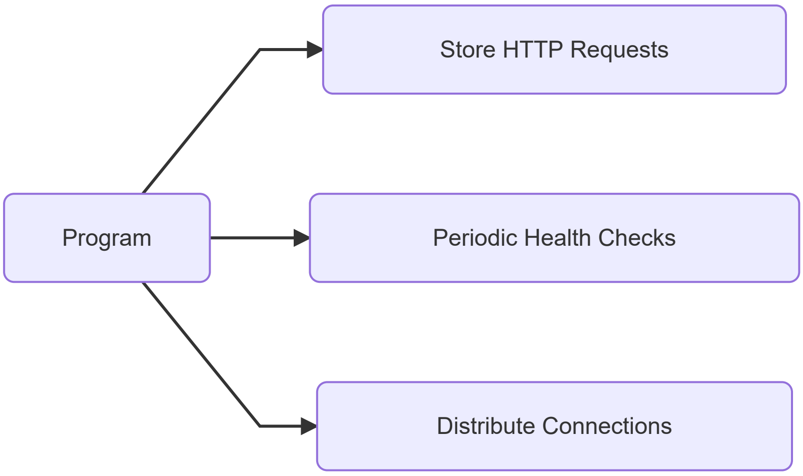
- This will be the dispatcher thread. Once no workers are available, it will wait until someone sends the condition variable back.
- If the message queue is not empty, it will dispatch message(s) to worker that is available by sending signal.

*int main (int argc, char** argv):*

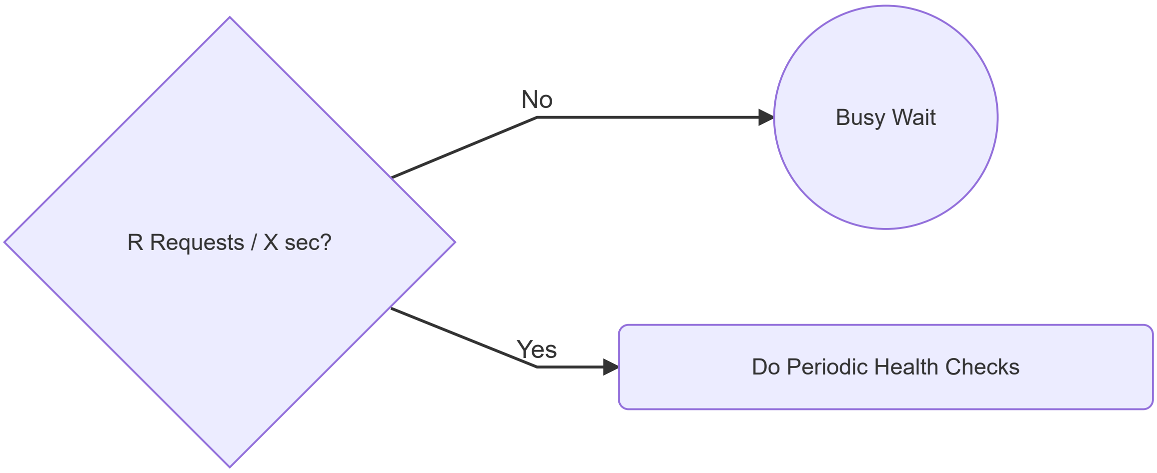
- Main will take command line arguments via *getopt()*. -N indicates number of threads and -R indicates number of requests to do before the next health check.
- Worker threads and health check thread will be created.
- HTTP requests will be continuously received in a *while(1)* loop, and requests will be pushed to the queue.

III. Architectural Design

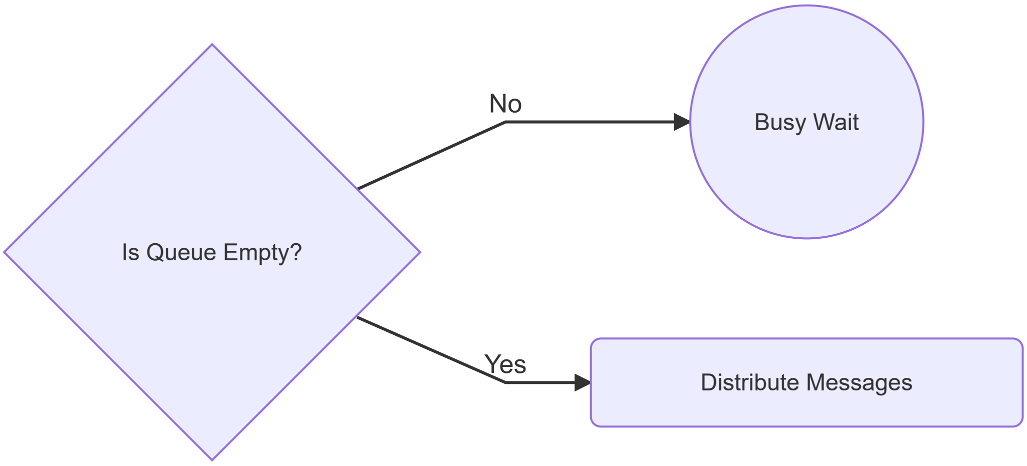
Overall, requests are distributed by the Worker Threads. Periodic health checks are done by the Healthcheck Thread on a regular basis. Once the threads are created in main(), they start running simultaneously.



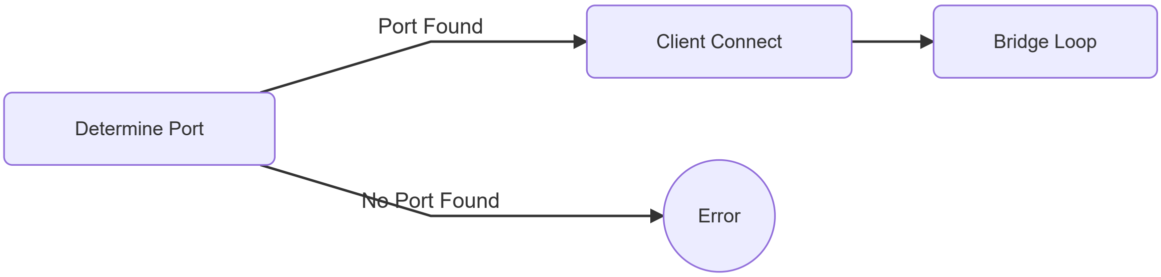
Below is the implementation of the Healthcheck Thread:



Below is the implementation of the Worker Thread:



Below is the implementation of Distribute Messages:



IV. Testing

Open multiple terminals, which includes one for client, one for load balancer, and several for servers. On the client sides, *curl* multiple simultaneous requests to the load balancer, then check if the health check has determined the correct port to sent requests and if correct responses are sent back.

```
./loadbalancer -N 7 1234 8080 -R 3 8081
```