In this program, I create a loadbalancer that accepts clients, and sends it to one of the provided httpservers. The best server is chosen by probing each server with a healthcheck. The one with the least load (# entries) is chosen. I use N worker threads so that there can be N parallel connections between by loadbalancer and the server, and one thread just for healthcheck. This healthcheck thread executes every 3 seconds or every R requests (whichever happens first).

I use getopt to extract the provided port numbers of the servers and the port number used for client connections. I insert the server ports in a global array named serverport[num], where num is a large number. I also have a global variable numServers to keep track of the number of servers.

getBestServer()

This function is executed at the start of the loadbalancer and inside the healthcheck thread. It connects and then sends and reads healthcheck request and response from the server. I use a for loop to probe the healthcheck resource of each server, and update the global int variables bestServer and serverReady.

processHealthRequest(connfd, port)

connfd is the socket of the port used by the server. If the server is up, connfd =/= -1. In this case, processHealthCheck is run. I build a string of healthcheck request: "GET /healthcheck HTTP/1.1\r\n\r\n", load it in a bffer, and send() that request to connfd. Then I call **healthTimer(connfd).** healthTimer() simply takes the server socket and returns 1 if there is data ready to be sent by the server within 2 seconds. If not, -1 is returned. If 1, then I start reading the response from the server until recv() returns 0. I use sscanf() to parse the response, checking the status codes, correct http format, and the total number of errors and entries returned.

Initially, bestServer is set to 0. If this is the first healthcheck, then upon the successful extraction of numErrors and numEntries of healthcheck from the server, I set the global var **bestServer** to this server port, **healthEntries** to numEntries, and **healthErrors** to numErrors.

If the bestServer has already been selected, I check if numEntries < healthEntries. If so, I set the port as bestServer and healthEntries as numEntries and healthErrors as numErrors.

If numEntries == healthEntries, then I check if this server has less errors than the one currently set. If so, I update my internal bestServer, healthEntries, and healthErrors.

Returns 1 if successful update, and -1 if server doesn't respond in time, or server sends a bad response.

processHealthCheck()

Thread function run by the healthcheck thread. Forever, getBestServer() is run every 3 seconds or every R requests. I use the timespec and timeval structs to set the timer. I referred to TA Daniel's section video on how to accomplish this.

```
While(true){
        Set the time struct ts to wait for 3 seconds
        Sem_timedwait(&signal, &ts)
        Sem_wait(&sem)
        resetData()       // resets the globals serverReady, serverState[], and bestServer, healthErrors
                and healthEntries to 0
        success = getBestServer()
        sem_post(&sem)
}
```

I have chosen a 3 second timer for the healthcheck because I thought 5 was too long and anything less than 3 was too short.
Here, signal is a semaphore that is signaled by the worker threads after processing R requests. getBestServer() is guarded by the locks that are also used by the worker threads. This is done to ensure synchronization as I want the workers to wait until the bestServer has been selected.

Bridger(void* threadId)

In main(), I create N threads to that run the bridger function. The bridger first extracts a client connection from the share array as in the last httpserver assignment. It then waits on the seamaphore sem, until the healthcheck thread signals the seamaphore that the servers have been checked. If the the global variable serverReady == 1, then I get the bestServer and use client_connect(bestServer). If client_connect is successful, I use the provided bridge_loop(acceptfd, connfd) function, where acceptfd is the client socket, and connfd is the httpserver socket. I then close connfd and acceptfd. Bridge_loop() does the transfer of data from client to server and vice versa. If during the select process the server doesn't respond, I send the error message (500) to the client and mark the global serverReady as -1 so that other threads don't have to wait. At the end of bridger(), I increment the global variable numRequests. If numRequest % R is 0, then I signal the healthcheck thread using sem_post(&signal) to peform another healthcheck.

```
for(i = 0; i < numServers; i++){

   port = serverport[i];
   connfd = client_connect(port);
   if(connfd != 1){
      success = processHealthRequest(connfd, port);
      if(success == 1)
           serverState[i] = 1;
       else
           serverState[i] = -1;
   else
        serverState[i] = -1;
    }
    Close(connfd);
}
```

serverState[num] is a global variable that stores the states of the servers: serverport[num]. If there is at least 1 serve that is up, the variable serverReady = 1. 1 server is ready for action. If not, serverReady = -1. There are no servers ready for action. I return the value of serverReady in getBestServer().

The execution in **main()** goes like this:

getopt()

getBestServer()

Run the healthchek thread

Run the N worker threads

```
While(1){
    Accept() the client connections and put them in the shared array (same as asgn2)
}
```