University of Science and Technology of Hanoi

Network Programming

Final Project Report

Load Balancer Programming

Group 2

Hanoi, June 2021

Contents

1 What is our project

1.1 About our group

- Dỗ Quang Hiếu - BI10-063 : programmer
- Nguyễn Khang Thái - BI10-158: report writer
- Lê Duy Anh - BI9-034: slide maker

1.2 Load balancer

Load balancer is used in the online storing network to control the flow of connections from Clients to Storage and Authentication servers. The main functionality is to forward messages from client to authentication and storage, and from those servers back to client.

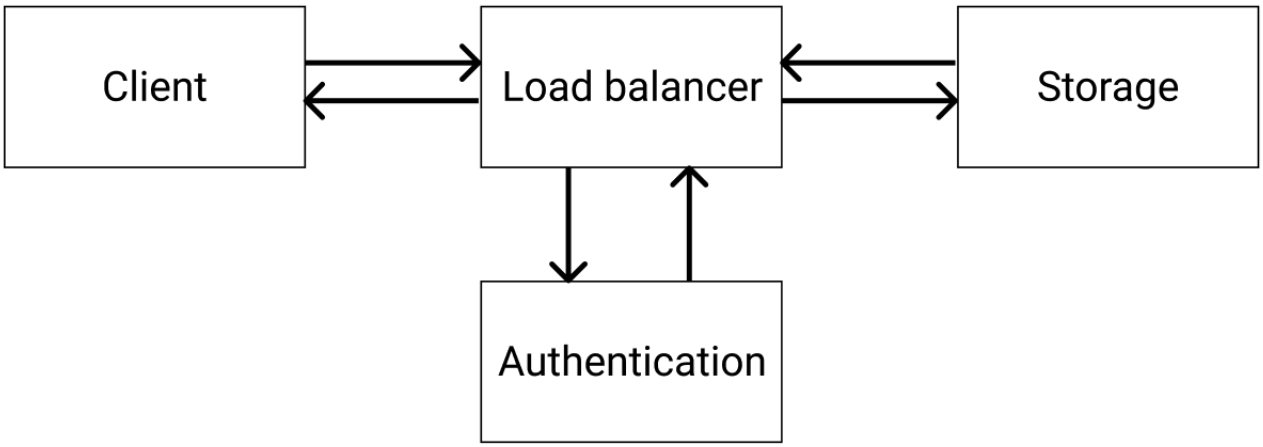

Figure 1: Online storing network

1.3 Our load balancer

Our load balancer is simpler than a typical load balancer. It still can be used to forward messages from client server exactly as they were sent and to perform validity check on the server so that the requests are handled correctly. However, our load balancer works with only one client at a time and performs only one action at a time.

2 Why people need our project

Since our load balancer does have all the functions of a typical one, but in general, we still have:

- **Abstraction:** We can make clients "think" that they connect to the system.
- **Verbosity:** All of the transaction is logged.

3 How we created it

3.1 Approach

In this project, the following tools will be used:

- **Programming Language:** C
- **Virtual Machine:** DigitalOcean

Also, we created this project with:

- **Protocol:** TCP
- **Socket method:** Blocking
- **Mechanism:** Transfer exact messages
- A util file (**util.c**) contains all functions
- A main file (**main.c**) describes what the load balancer does.

3.2 util.h

```
#ifndef __UTIL_H
#define __UTIL_H

#define NONBLOCK 1
#define BLOCK 0

int accept_connection(int fd, int nonblock);
int make_server(short serv_port, int maxqueue, int nonblock);
int make_client(char *serv_hostname, short serv_port, int nonblock);
void send_wrapper(int fd, char *msg, char *errmsg);
void recv_wrapper(int fd, char *buffer, int size, char *errmsg);

void make_file(char *buffer, char *filename);
void make_buffer(char *buffer, char *filename);

#endif
```

Figure 2: util.h

3.3 util.c

This util file has functions to create server and client, accept connections, make files, make buffers and send wrappers.

```c
int make_server(short port, int maxqueue, int nonblock)
{
    int fd = socket(AF_INET, SOCK_STREAM, 0);
    if (fd < 0) {
        perror("Socket creation error");
        exit(err_base_val);
    }
    fprintf(stderr, "Socket made.\n");

    if (nonblock) {
        setsockopt(fd, SOL_SOCKET, SO_REUSEADDR, &(int){ 1 }, sizeof(int));
        fcntl(fd, F_SETFL, fcntl(fd, F_GETFL, 0) | O_NONBLOCK);
    }

    struct sockaddr_in serv_addr;
    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    serv_addr.sin_port = htons(port);

    if (bind(fd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0) {
        perror("Binding error");
        exit(err_base_val + 1);
    }
    fprintf(stderr, "Socket binded.\n");

    if (listen(fd, maxqueue) < 0) {
        perror("Listening error");
        exit(err_base_val + 2);
    }
    fprintf(stderr, "Socket listened.\n");

    return fd;
}
```

Figure 3: Function to create server

```c
int accept_connection(int fd, int nonblock)
{
    struct sockaddr_in cli_addr;
    socklen_t cli_len;

    int clientfd = accept(fd, (struct sockaddr *) &cli_addr, &cli_len);
    if (clientfd < 0) {
        perror("Accepting error");
        exit(err_base_val + 3);
    }
    printf("Connection accepted.\n");

    if (nonblock) {
        setsockopt(clientfd, SOL_SOCKET, SO_REUSEADDR, &(int){ 1 }, sizeof(int));
        fcntl(clientfd, F_SETFL, fcntl(clientfd, F_GETFL, 0) | O_NONBLOCK);
    }

    return clientfd;
}
```

Figure 4: Function to accept connection between client, load balancer, storage and authentication

```c
int make_client(char *serv_hostname, short serv_port, int nonblock)
{
    struct hostent *entry = gethostbyname(serv_hostname);
    if (entry == NULL) {
        herror("Host resolve error");
        exit(err_base_val + 4);
    }
    fprintf(stderr, "Host resolved.\n");

    int serv_fd;
    if ((serv_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("Socket creation error");
        exit(err_base_val + 5);
    }
    fprintf(stderr, "Socket created.\n");

    if (nonblock) {
        setsockopt(serv_fd, SOL_SOCKET, SO_REUSEADDR, &(int){ 1 }, sizeof(int));
        fcntl(serv_fd, F_SETFL, fcntl(serv_fd, F_GETFL, 0) | O_NONBLOCK);
    }

    struct sockaddr_in serv_addr;
    socklen_t serv_len = sizeof(serv_addr);

    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    memcpy((char *) &serv_addr.sin_addr.s_addr, entry->h_addr_list[0], entry->h_length);
    serv_addr.sin_port = htons(serv_port);

    if (connect(serv_fd, (struct sockaddr *) &serv_addr, serv_len) < 0) {
        perror("Connect error");
        exit(err_base_val + 6);
    }

    fprintf(stderr, "Connected.\n");

    return serv_fd;
}
```

Figure 5: Function to create client

```c
void send_wrapper(int fd, char *msg, char *errmsg)
{
    fprintf(stderr, "%s: Sending...\n", errmsg);
    fprintf(stderr, "%s: Content: %s\n", errmsg, msg);
    if (send(fd, msg, strlen(msg), 0) < 0) {
        perror(errmsg);
        exit(err_base_val + 7);
    }
    fprintf(stderr, "%s: Sent.\n", errmsg);
}

void recv_wrapper(int fd, char *msg, int size, char *errmsg)
{
    fprintf(stderr, "%s: Receiving...\n", errmsg);
    memset(msg, 0, size);
    int status;
    if ((status = recv(fd, msg, size, 0)) < 0) {
        perror(errmsg);
        exit(err_base_val + 8);
    } else if (status == 0) {
        shutdown(fd, SHUT_RDWR);
        close(fd);
        msg = NULL;
        fprintf(stderr, "Socket hung up.\n");
        exit(err_base_val + 9);
    }

    fprintf(stderr, "%s: Content: %s\n", errmsg, msg);
    fprintf(stderr, "%s: Received.\n", errmsg);
}
```

Figure 6: Functions to send and receive wrappers

```c
void make_file(char *buffer, char *filename)
{
    FILE *file = fopen(filename, "w");
    fprintf(file, "%s\n", buffer);
    fclose(file);
}

void make_buffer(char *buffer, char *filename)
{
    FILE *file = fopen(filename, "r");
    char line[1024] = { 0 };
    while (fgets(line, sizeof(line), file)) {
        strncat(buffer, line, strlen(line));
        memset(line, 0, sizeof(line));
    }
    fclose(file);
}
```

Figure 7: Function to make files and buffers

3.4 main.c

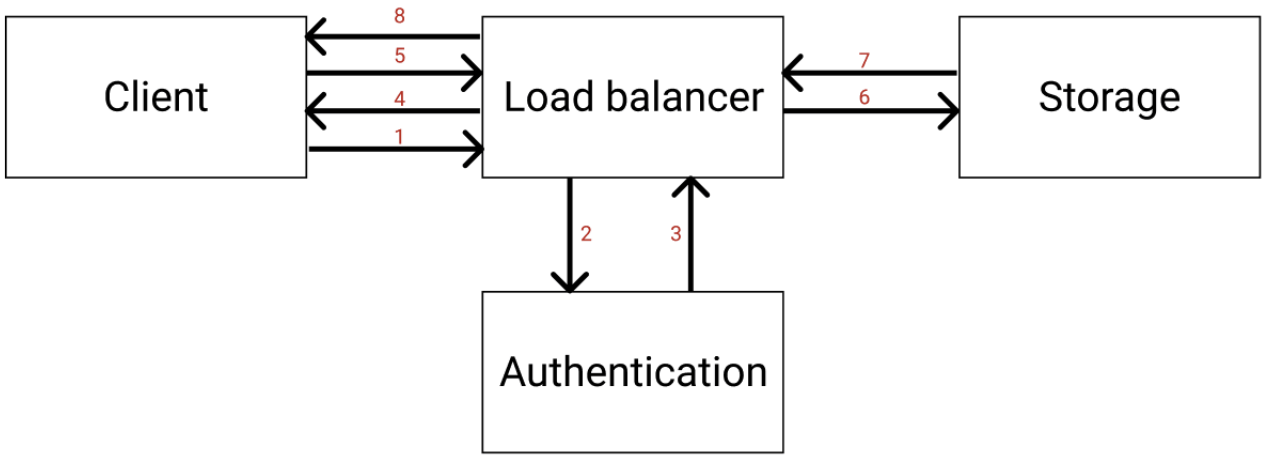

Figure 8: Functions of a load balancer

This load balancer has some basic functions:

- **auth_exchange (1, 2, 3, 4):** send U/P from the client to the authentication server and then send status valid/invalid from the authentication back to the client.
- **storage_exchange (5, 6, 7, 8:** send command from client to storage with the protocol of client or storage: send file's name or command's name, then send and receive their sizes, and lastly send and receive their content.

```c
int  auth_exchange(int clientfd, int authfd, char *buf, int bufsz)
{
    recv_wrapper(clientfd, buf, bufsz, "lb-recv-creds");
    send_wrapper(authfd, buf, "lb-send-creds");

    recv_wrapper(authfd, buf, bufsz, "lb-recv-auth-status");
    send_wrapper(clientfd, buf, "lb-send-auth-status");

    if (strcmp(buf, "invalid") == 0) {
        return 0;
    }

    return 1;
}
```

Figure 9: Function to exchange wrappers between authentication and client

```c
void storage_exchange(int clientfd, int storefd, char *buf, int bufsz)
{
    while (1) {
        // Always send 1
        recv_wrapper(clientfd, buf, bufsz, "lb-recv-cmd");
        send_wrapper(storefd, buf, "lb-send-cmd");

        char *result = strstr(buf, "send");

        if (result && (result - buf == 0)) {
            // Send 2 more if the file is being sent
            recv_wrapper(storefd, buf, bufsz, "lb-recv-resp");
            send_wrapper(clientfd, buf, "lb-send-resp");

            recv_wrapper(storefd, buf, bufsz, "lb-recv-resp");
            send_wrapper(clientfd, buf, "lb-send-resp");
        } else {
            // But receive 2 if the file is being received
            recv_wrapper(storefd, buf, bufsz, "lb-recv-resp");
            send_wrapper(clientfd, buf, "lb-send-resp");

            recv_wrapper(storefd, buf, bufsz, "lb-recv-resp");
            send_wrapper(clientfd, buf, "lb-send-resp");
        }
    }
}
```

Figure 10: Function to exchange wrappers between storage and client

```c
int main(int argc, char *argv[])
{
    if (argc != 3) {
        fprintf(stderr, "Usage: %s auth storage\n", argv[0]);
        exit(255);
    }

    int sockfd = make_server(8000, 5, BLOCK);
    char buffer[1024];

    while (1) {
        int clientfd = accept_connection(sockfd, BLOCK);
        int authfd = make_client(argv[1], 8001, BLOCK);

        int valid = auth_exchange(clientfd, authfd, buffer, sizeof(buffer));
        close(authfd);

        if (valid) {
            int storefd = make_client(argv[2], 8002, BLOCK);

            while (1) {
                storage_exchange(clientfd, storefd, buffer, sizeof(buffer));
            }

            close(storefd);
        } else {
            fprintf(stderr, "Invalid authentication. Connection rejected.\n");
        }

        close(clientfd);
    }

    close(sockfd);

    return 0;
}
```

Figure 11: Main function

4 Demonstration

With the projects of three other groups that created Client, Authentication server and Storage server, here is the demonstration of how our load balancer works in the online storing network.

```
zer0warm@network-programming:~/loba$ ./LB localhost localhost
Socket made.
Socket binded.
Socket listened.
Connection accepted.
Host resolved.
Socket created.
Connected.
lb-recv-creds: Receiving...
lb-recv-creds: Content: hung    123
lb-recv-creds: Received.
lb-send-creds: Sending...
lb-send-creds: Content: hung    123
lb-send-creds: Sent.
lb-recv-auth-status: Receiving...
lb-recv-auth-status: Content: valid
lb-recv-auth-status: Received.
lb-send-auth-status: Sending...
lb-send-auth-status: Content: valid
lb-send-auth-status: Sent.
Host resolved.
Socket created.
Connected.
lb-recv-cmd: Receiving...
```

Figure 12: Demonstration of exchanging between client and authentication server

```
Host resolved.
Socket created.
Connected.
lb-recv-cmd: Receiving...
lb-recv-cmd: Content: ls
lb-recv-cmd: Received.
lb-send-cmd: Sending...
lb-send-cmd: Content: ls
lb-send-cmd: Sent.
lb-recv-resp: Receiving...
lb-recv-resp: Content: 132
lb-recv-resp: Received.
lb-send-resp: Sending...
lb-send-resp: Content: 132
lb-send-resp: Sent.
lb-recv-resp: Receiving...
lb-recv-resp: Content: LB
Makefile
authen.c
authen.out
client.txt
guess.rb
hello.txt
main.c
```

Figure 13: Demonstration of exchanging between client and storage server

5 Summary

5.1 What we did

We made a "reverse proxy" that can act as the load balancer, because we used blocking method, not no-blocking.

5.2 What we could have done

This project creates a simple version of the load balancer. We could have upgraded this load balacer to:

- Handle multiple connections.
- Proper signal transfer protocol.