Laszlo Koroleff
1839634

Directories:
Common:

Includes all file reading and writing functions. Also includes datagram parsing functions used for separating key parts of requests and responses between clients and servers. The header file Thread.h includes a lamda function to dynamically create threads with any number of arguments and objects. This was necessary to keep with the modularity of my program and ensure threads maintain separation from important parts of memory.

Server:

Where all important logic for my proxy server reside (see below).

The structure of my assignment has the typical folder organization expected with the document specifications. However, I include additional folders for header files (*../include*) and separate folders to maintain modularity between operations of my program (*../src/<server>* ***or*** *<common>*). The server folder is where both my main function and all proxy server logic reside. This is where I listen for connecting clients, accept those clients, and create unique pointers to structs for handling threads to use. This is also where I use openSSL libraries to establish a reliable certificate certifying connection between the proxy server and the destination server specified by the client. This certificate verifying logic is handled using OpenSSL's X509 functionality, chaining together the destination system certificates to ensure authenticity. I provide additional flag logic to maintain atomic reading and writing of a set that monitors forbidden sites defined in a "Forbidden Sites" text file. Finally, to ensure persistent connections between active clients and the proxy server, I implement mechanisms to efficiently manage and reuse connections, reducing overhead and maintaining stability for long-running sessions.

I ran these 5 tests to ensure that my program runs correctly:

1. `curl -i -x  http://127.0.0.1:9090/ http://www.example.com`

I ran this curl command to ensure that the basic functionality of my proxy server was correct. This of course ran as expected and returned a status code 200 to the client.

2. curl -i -x http://127.0.0.1:9090/ http://httpbin.org/stream/100

This command tests how my proxy server handles chunked streaming from the server. This passed like expected.

3. curl -i -x http://127.0.0.1:9090/ http://www.neverssl.com

This command was to see if my proxy could communicate with an additional server outside of simply example.com. It of course passed.

4. echo -e "GET http://www.example.com/ HTTP/1.1\r\nHost: www.example.com\r\nConnection: keep-alive\r\n\r\n" | nc 127.0.0.1 9090

This command was to see if my program could run when using a *netcat* request. It passed like expected.

5. printf "GET http://www.example.com/ HTTP/1.1\r\nHost: www.example.com\r\nConnection: keep-alive\r\n\r\n" | nc 127.0.0.1 9090

Since `printf` sends the request as plain text, this command will check if my proxy correctly parses and forwards HTTP headers without relying on higher-level libraries (like `curl` does).

The hardest part of this lab was ensuring that my program ran smoothly and efficiently. Getting the multithreading part to work was a pain. I also struggled with the certification verification of the SSL part of the assignment.