TCP Client - Server

Contributors

- **Ahmed Essam Hassan Abd-ElRazek - 19015289**
- **Mohammed Ayman Saaed Mahmoud - 19016250**

Overview

An implementation for a basic TCP client/server written in C++. It can communicate with any other remote TCP clients/servers. Unix sockets were used for low-level networking logic.

TCP - Server

Pseudocode

1. Listen for requests on the predefined port.
2. Accept the incoming new connection.
3. Fork the process to handle the new connection without blocking receiving more new connections.
    a. Read the first 32kb chunk of data sent by the client.
    b. If the client sent zero, then close the connection.
    c. Extract method, path and HTTP version.
    d. If the method is GET, send the file requested if it exists after adjusting response headers or send a 404 page otherwise.
    e. If the method is POST, send a 200 OK message then read the payload sent by the client.
    f. Keep the connection open for (8 / total_clients_count) * 5 seconds. If after that period no more requests are received, then close the connection. Otherwise, go to step a.
4. Go to step 3

File Structure

- **main.c**
    - It contains the implementation of the pseudocode mentioned above.
    - It contains a dictionary for common mime types, so that the server can set the “Content-Type” header to the right value.

Timeout Heuristic

It’s very simple and efficient to calculate. Its formula is (8 / total_clients_count) * 5. Total number of clients is a value shared across all processes handling different connections.

TCP - Client

Pseudocode

1. Open socket
2. Connect with server with ip address taken from arguments
3. Read requests line in file in.txt line by line
4. Parse request and add header using header class
5. Add host attribute to the header
6. Get request
    a. Send header and wait until server send the file
    b. Receive the response header using class ResponseHeader
    c. Open new file to save file sent
    d. Save file
7. Post request
    a. Add content length and content type to the header
    b. Send the header
    c. Load the file that should be sent
    d. Sent it to the server
    e. Receive the request response from the server
8. Close the connection

File Structure

- **Header.h**
  - This file has the header parsing code
  - It contain Header class as parent for RequestHeader and ResponseHeader
  - RequestHeader class to class header when sending request
  - ResponseHeader to parse header when receiving a response for the request that I have created
  - Function to send content type based on file extension

- **Client.cpp**
  - The main file of the client side
  - It executed from terminal passing server ip and port number
  - Parse the requests from file in.txt
  - Send the requests based of method “GET”, “POST”

- **In.txt**
  - File of executed commands