Laiba Maab

CID: DEP2248

Task3: Building a Multi-Threaded Web Server

```
#include <iostream>
#include <thread>
#include <vector>
#include <cstring>
#include <arpa/inet.h>
#include <unistd.h>
#include <fstream>

const int PORT = 8080;
const int BUFFER_SIZE = 1024;
void handle_client(int client_socket);
std::string get_file_content(const std::string &path);

int main()
{
   int server_fd, new_socket;
   struct sockaddr_in address;
   int opt = 1;
   int addrlen = sizeof(address);
   if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
   {
      perror("socket failed");
      exit(EXIT_FAILURE);
   }
```

```
if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR |
      SO_REUSEPORT, &opt, sizeof(opt)))
{
   perror("setsockopt");
   exit(EXIT_FAILURE);
}
address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons(PORT);

if (bind(server_fd, (struct sockaddr *)&address, sizeof(address))
      < 0)
{
   perror("bind failed");
   exit(EXIT_FAILURE);
}

if (listen(server_fd, 3) < 0)
{
   perror("listen");
   exit(EXIT_FAILURE);
}

std::vector<std::thread> threads;
while (true)
{
   if ((new_socket = accept(server_fd, (struct sockaddr
      *)&address, (socklen_t*)&addrlen)) < 0)
   {
```

```
            perror("accept");
            exit(EXIT_FAILURE);
        }
        threads.emplace_back(std::thread(handle_client,
            new_socket));
    }
    for (auto &t : threads)
    {
        if (t.joinable())
        { t.join();}
    }
    return 0;
}
void handle_client(int client_socket)
{
    char buffer[BUFFER_SIZE] = {0};
    read(client_socket, buffer, BUFFER_SIZE);

    std::string request(buffer);
    std::cout << "Request:\n" << request << std::endl;

    std::string delimiter = " ";
    size_t pos = request.find(delimiter);
    std::string method = request.substr(0, pos);
    request.erase(0, pos + delimiter.length());
    pos = request.find(delimiter);
    std::string path = request.substr(0, pos);

    if (path == "/")
```

```
    {
        path = "/index.html";
    }
    std::string response;
    std::string content = get_file_content("." + path);

    if (content.empty())
    {
        response = "HTTP/1.1 404 Not Found\r\nContent-Length:
            0\r\n\r\n";
    } else
    {
        response = "HTTP/1.1 200 OK\r\nContent-Length: " +
            std::to_string(content.size()) + "\r\n\r\n" + content;
    }
    send(client_socket, response.c_str(), response.size(), 0);
    close(client_socket);
}

std::string get_file_content(const std::string &path)
{
    std::ifstream file(path, std::ios::in | std::ios::binary);
    if (!file)
    {  return ""; }
    std::string content((std::istreambuf_iterator<char>(file)),
            std::istreambuf_iterator<char>());
    return content;
}
```