Networks

Assignment 1

HTTP Client-Server

Name : ***Seif Mohamed Mahmoud Gneedy***

ID : 18010834

- **Project structure :**
  - **Server folder : server.cpp and client_handler.cpp**
  - **Client folder : client.cpp**
  - **Utils folder : file_util.cpp and parser.cpp**

- **Data structure used:**
  - **Used a char array with size=BUFFER_SIZE = 4 KB to store the received pieces of message.**
  - **Used atomic variable to prevent race condition which counts the number of clients currently in the server.**

```
    atomic<int> clients_count;
```

- **How Multi-threaded server works :**

Used multi-threaded approach as dealing with threads has less overhead than multi-process so we can serve the clients quickly.

The server starts with creating a socket, binds it to the given address, starts listening and accepting the incoming clients.
With every incoming client attach a thread to serve its requests by calling thread_func() which calls client_handler.handl() to handle this client.
Handling the client : by setting timeout with every request according to the number of clients in the server to make the connection persistence and in the same time doesn't take resources while making nothing by providing heuristic :

```
MAX_TIMEOUT -   (MAX_TIMEOUT-1) * ((clients_count-1)/MAX_THREADS)
```

Then it parses the request ,saves the file if POST, gets the file to be sent if GET and prepares the response in the HTTP format.
Then it waits for a timeout or another request.

- ## How Client works :

The client starts with creating the socket,connects to the given server (as arguments) and then starts to request.
How to handle request : it starts by reading the requests file(provided as an arg) , taking every request ,put it in HTTP request format(with file in the body if POST) ,sends it to the server ,waits for the response, , parse the response,print the response and save the file if GET successed.

- ## In file_util.cpp :

```
/*
   open file
   read it line by line with every line : get the request in 2 parts:
method file_path
   */
   vector<vector<string>> get_requests(string path);
   /*
       create http response in format that client can understand
   */
   string create_http_response(string method_line,string body);
   /*
       create http request in the format as the server can understand it
   */
   string create_http_request(string method,string file_path);
   /*
       open the file specified in POST and returns its content
   */
   pair<char*,int> get_file_content(string file_path);
   /*
       create directories (if exists) for the specified file_path
   */
   void create_dirs(string file_path);
   /*
       save the content in the file path
   */
```

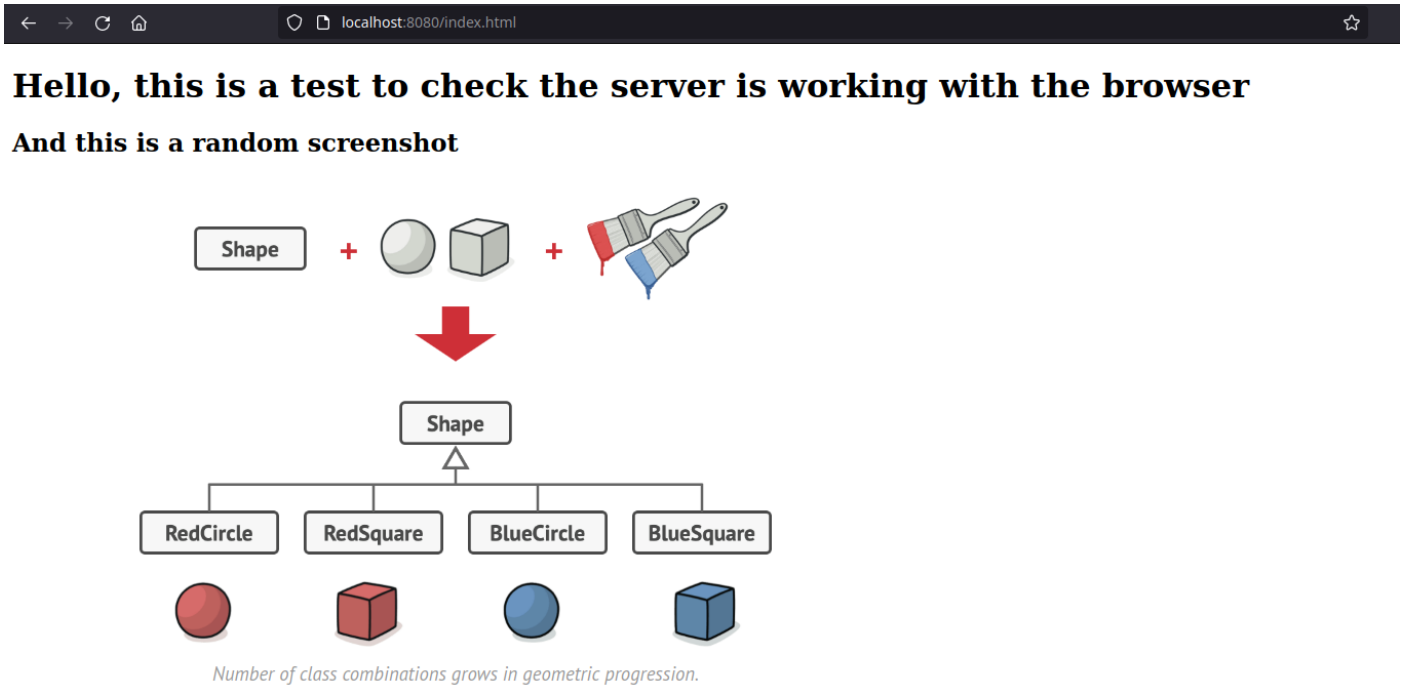
```
    void save_file(string path,string content);
    /*
        send message to the specified socket
    */
    void send_message(string& msg,int sock_id);
    /*
        receive message from the server to the sent requests
    */
    pair<string,string> receive_message(int sock_id);
    /*
get the requests from the file,determine the type of every request, handle
the required request, send the request receive resoponse and handle it

    */
void handle_requests(string file_path, int clntSock);
```

- **In parser.cpp :**

```
// parse header and return <request_line,<rest of body,content-length>>
    pair<string,pair<string,int>> parse_header(char* received,int rec_sz);
```

- **Testing the server with firefox :**



Adding new shape types and colors to the hierarchy will grow it exponentially. For example, to add a triangle shape you'd need to introduce two subclasses, one for each color. And after that, adding a new color would require creating three subclasses, one for each shape type. The further we go, the worse it becomes.