Assignment 1
Introduction to Socket Programming in C/C++

Name: Ali Mohamed Ali Hamed

ID:40

Introduction

In this assignment, you will use sockets to implement a simple web client that communicates with a web server using a restricted subset of HTTP. The main objective of this assignment is to give you hands-on experience with UNIX sockets.

implementation

Class Parser:

- words opening; // emum to choose it's GET or POST.
- string directory; // store the directory of the file.
- string body; // store the body of the request in case of POST.
-
- vector<string> main_parse(string s,string delim)
  giving a string s and a delimiter, parse the string into a vector of strings by the delimiter.
- void parse_lines(vector<string> res)
  vector <string> res is the output of main_parse, the method parses the first line and checks if it GET or POST, then the directory of the file, it then gets the headers, and then if POST it stores the body of request.

Class Http:

- Parser parser
- string message
  to store the response

- void get_request(string directory)
  In case of GET request, pass the directory of the file, determine if the file exists, if it exists call the ok_response and read the file then store it message, if it doesn't exist call not_found_response.
- void post_request(string directory, string body)
  in case of POST request, pass the directory and the body of the request, create a file and store the body into it.
- void ok_response(string data)
  store the HTTP/1.1 200 OK and the data into the message response.

- void not_found_response()
  store HTTP/1.1 404 NOT FOUND

Server

- Create a winsock connection
- Resolve the local address and port to be used by the server
  iResult = getaddrinfo(NULL, DEFAULT_PORT, &hints, &result);
- Create a SOCKET for connecting to server
  ListenSocket = socket(result->ai_family, result->ai_socktype, result->ai_protocol);
- bind a socket that has already been created to an IP address and port,
  Setup the TCP listening socket
  iResult = bind(ListenSocket, result->ai_addr, (int)result->ai_addrlen);
- After the socket is bound to an IP address and port on the system, the server must then listen on that IP address and port for incoming connection requests.
  iResult = listen(ListenSocket, MAX_CLIENTS);
- Accept a client socket and assign this client to a thread

```
  while(true) {
      cout << "server is waiting\n";
      SOCKET ClientSocket = accept(ListenSocket, NULL, NULL);
      counter++;

      if (ClientSocket == INVALID_SOCKET) {
          printf("accept failed with error: %d\n", WSAGetLastError());
          closesocket(ListenSocket);
          WSACleanup();
          return 1;
      }

      thread(client_connect,ClientSocket).detach();
  }
```

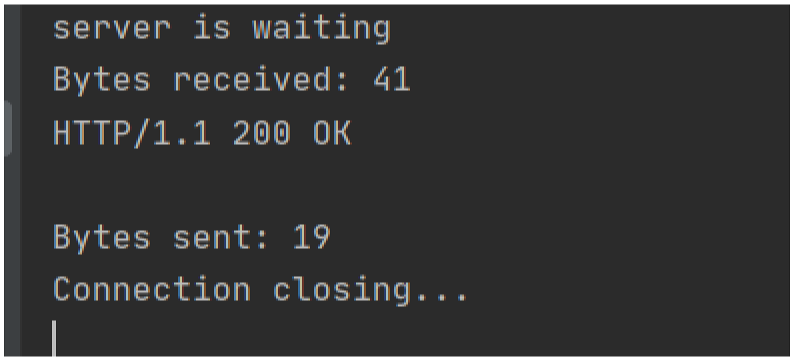
- Receive and send data with timeout
  iResult = recv(ClientSocket, recvbuf, recvbuflen, 0);
  int iSendResult = send(ClientSocket, send_buff, tmp.size(), 0);
- Timeout = number of max clients / number of active clients.

Client:

- Create a winsock connection
- Resolve the local address and port to be used by the server
  iResult = getaddrinfo(NULL, DEFAULT_PORT, &hints, &result);
- Attempt to connect to an address until one succeeds.
- Create a SOCKET for connecting to server
  ConnectSocket = socket(ptr->ai_family, ptr->ai_socktype, ptr->ai_protocol);
- Connect to the server.
  iResult = connect(ConnectSocket, ptr->ai_addr, (int)ptr->ai_addrlen);
- Receive and send data.

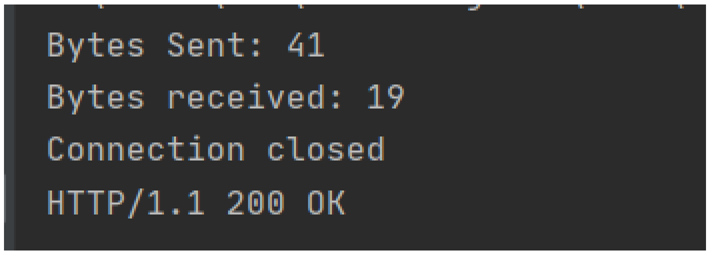
Screenshots:

Posting a file => "POST /test.txt HTTP/1.1\r\n\r\n hello i'm ali"
Server side:



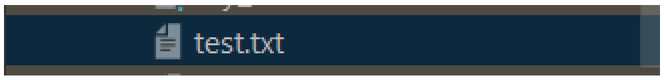
```
server is waiting
Bytes received: 41
HTTP/1.1 200 OK

Bytes sent: 19
Connection closing...
```

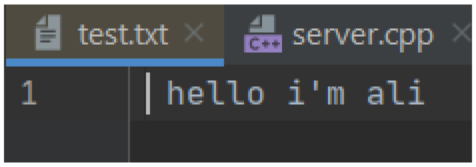
Client side:



```
Bytes Sent: 41
Bytes received: 19
Connection closed
HTTP/1.1 200 OK
```

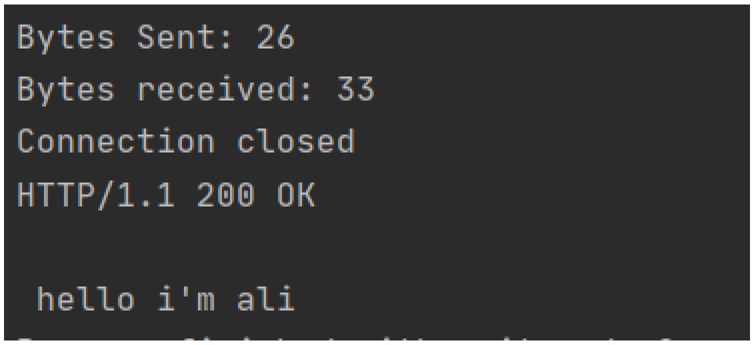
The file created successfully





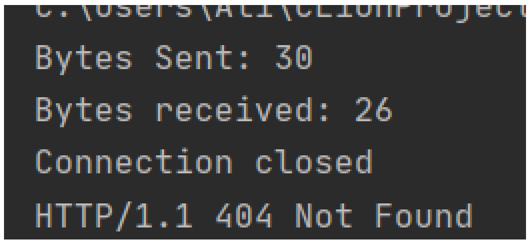
```
hello i'm ali
```

"GET /test.txt HTTP/1.1\r\n\r\n"



```
Bytes Sent: 26
Bytes received: 33
Connection closed
HTTP/1.1 200 OK

 hello i'm ali
```

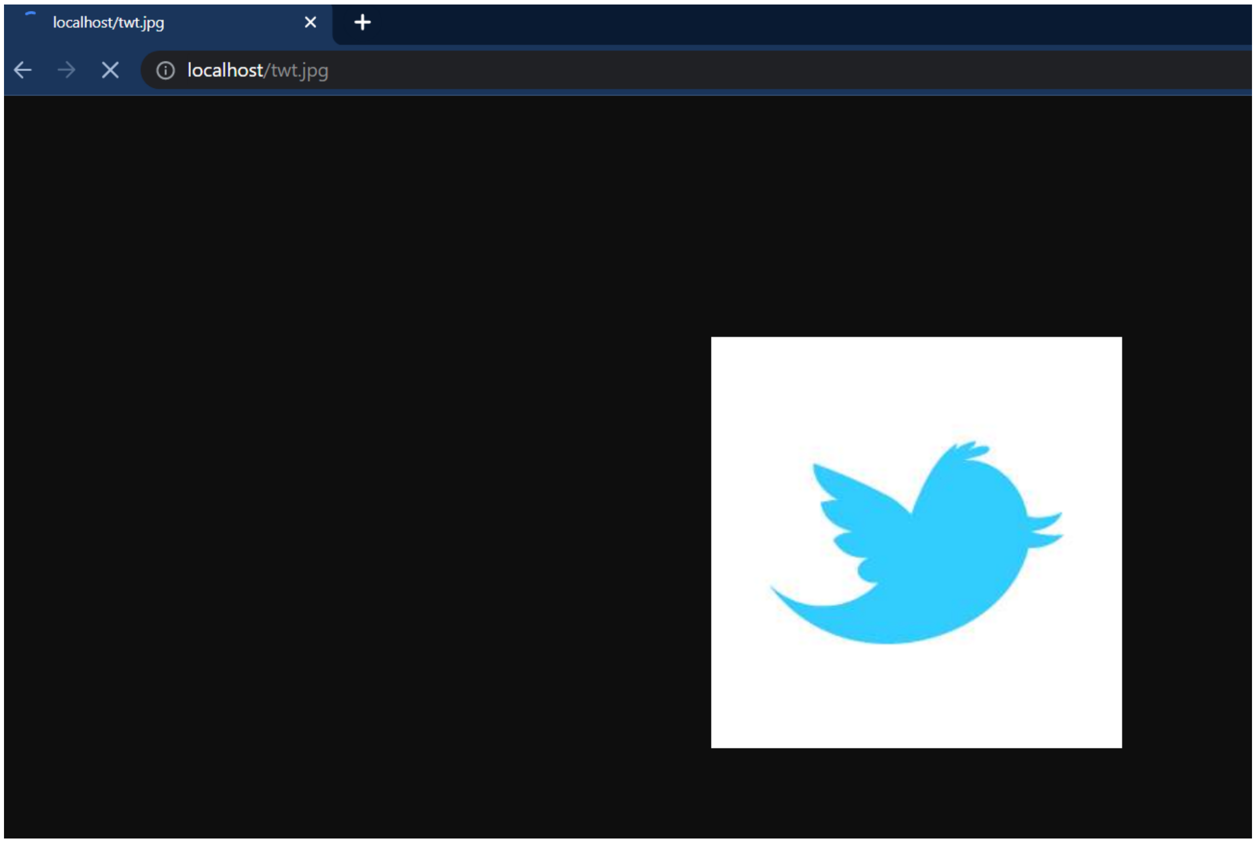
"GET /notfound.txt HTTP/1.1\r\n\r\n"



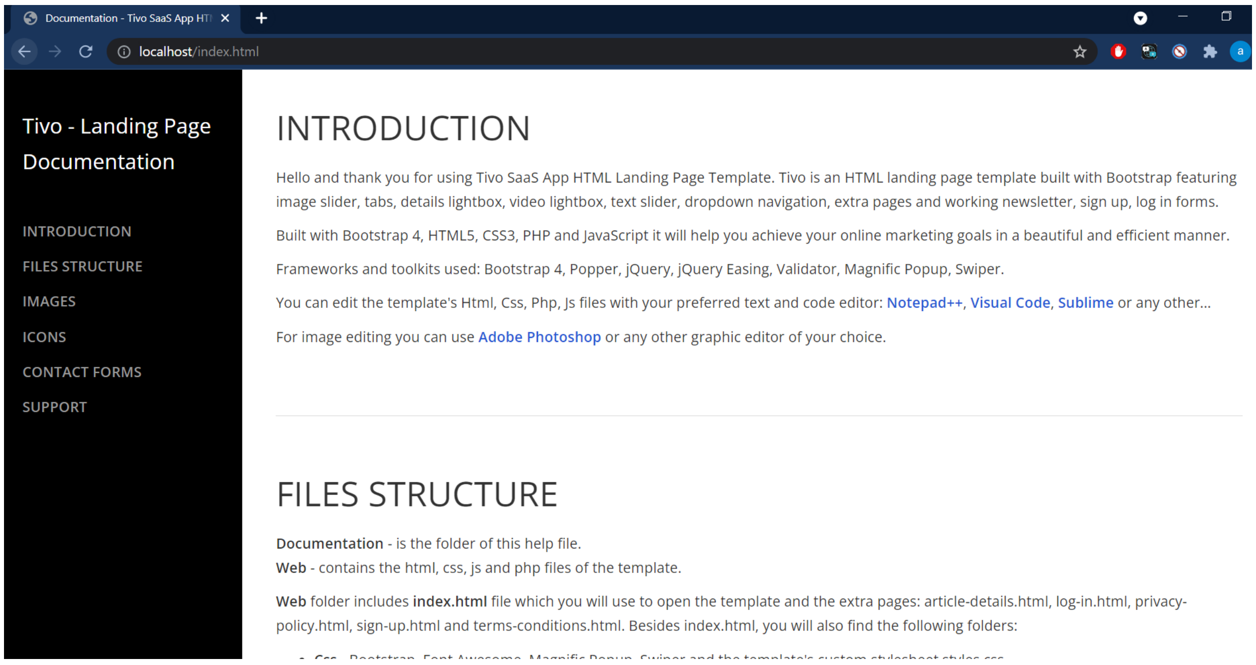
```
Bytes Sent: 30
Bytes received: 26
Connection closed
HTTP/1.1 404 Not Found
```

Using browser:

Get a picture



GET html and css files



INTRODUCTION

Hello and thank you for using Tivo SaaS App HTML Landing Page Template. Tivo is an HTML landing page template built with Bootstrap featuring image slider, tabs, details lightbox, video lightbox, text slider, dropdown navigation, extra pages and working newsletter, sign up, log in forms.

Built with Bootstrap 4, HTML5, CSS3, PHP and JavaScript it will help you achieve your online marketing goals in a beautiful and efficient manner.

Frameworks and toolkits used: Bootstrap 4, Popper, jQuery, jQuery Easing, Validator, Magnific Popup, Swiper.

You can edit the template's Html, Css, Php, Js files with your preferred text and code editor: Notepad++, Visual Code, Sublime or any other...

For image editing you can use Adobe Photoshop or any other graphic editor of your choice.

FILES STRUCTURE

Documentation - is the folder of this help file.
Web - contains the html, css, js and php files of the template.

Web folder includes **index.html** file which you will use to open the template and the extra pages: article-details.html, log-in.html, privacy-policy.html, sign-up.html and terms-conditions.html. Besides index.html, you will also find the following folders: