Application Software Track
Week 3 Assignment

Overview : In this assignment, we will create a simple HTTP server that responds to **HTTP GET** requests from a client. You will be reusing your code from week 2, and we have provided you with a skeleton code for the additional stuff that you will implement to process an HTTP request from a web client. The additional skeleton code can be found in the GithHub repository of the bootcamp.

Section A

A Simple Multi-Threaded HTTP Server

For this task, you will be adding some new data structures and functions to implement the functionality of a simple HTTP server on the server side. For this assignment, we will only deal with HTTP GET requests from a web client, requesting an HTML file at a given URL. A URL can resemble the path of a directory (ex.: */dir_a/sub_dir_b/*), or that of a file (ex.: */dir_a/sub_dir_b/index.html*). If the URL is that of a directory, we will be looking for an *index.html* file in that directory and serve that. The communication between the client and the server will be through TCP protocol. We will be using HTTP 1.0, in which the server closes the TCP connection immediately after responding to the client's request.

For this assignment, we have provided a skeleton code in the GitHub repository for the Bootcamp, which includes two files - *http_server.h* and *http_server.cpp* . You need to complete the code in these two files appropriately. You also need to modify your code in *server.cpp* from week 2 to accept HTTP requests from the client and respond with appropriate HTTP responses generated by your code.

For this assignment, assume that only HTML files are requested for. We have provided you with a sample directory structure in which resources (HTML files and other files if needed) are stored. The root directory (with the root url / for HTTP) is named *nested_html*. This will help you in testing your HTTP server by sending appropriate requests.

Your HTTP server must be waiting for connections at **port 80** on **localhost**.
You need to write a **Makefile** (with a target 'clean') which creates an executable file named *my_http_server*.

Reference Material :

- HTTP Made Really Easy (Read upto the 'Sample HTTP Exchange' section)
- HTTP Tutorial (An optional and more general reference)

Skeleton Code :

You are supposed to modify the given skeleton code by adding your code in the sections marked with TODO in comments.

http_server.h

- There are two cpp structs given to you : *HTTP_Request* and *HTTP_Response*
- You have been given some member variables for both the classes. You can add more fields to the structs if you wish to.
- The constructor for *HTTP_Request* and the member function *get_string()* of *HTTP_Response* are to be implemented in *http_server.cpp*

http_server.cpp

- You need to complete the constructor for the struct *HTTP_Request*, which extracts the relevant fields from a request in string form.
- You need to complete the member function *get_string()* for the class *HTTP_Response*, which returns the string form of the response with headers and body. (Go through the first link in the ‘reference material’ section to understand how an HTTP response is structured)
- The main function to be implemented is the function *handle_request()*. This function will be called from *server.cpp* to create an HTTP response for a given request. Go through the template code of this function to grasp the logic and fill the TODO sections appropriately.
- You have to modify the function *socketThread()* in your *server.cpp* file from week 2 to receive an HTTP request, call the *handle_request()* function to get the *HTTP_Response* object, and use *get_string()* function on the object, and send the string to the client, before closing the connection.

server.cpp

- You will need to include *http_server.h* in this code, to be able to use the data structures and functions
- You’ll have to modify the *socketThread()* function as described before.

- Note that you won't need any sort of synchronisation for this assignment. So, you need to get rid of the corresponding code, for the sake of efficiency and the natural independence of the HTTP GET requests (you would need synchronisation for some other types of HTTP methods which you will encounter in the upcoming weeks).

You can use appropriate library functions as and when required (don't forget to include the corresponding header files in your code).

Testing using a web browser : Open your favourite web browser (ex.: Mozilla Firefox), and type *http://localhost:80/<url>* in the address bar to request for the particular url. If your HTTP server works properly, then you should be able to see a corresponding webpage (if the requested url exists) or a 404 message (if the requested url doesn't exist).

Testing using wget : wget is a command-line tool using which you can download files from the web. If you use the command *wget http://localhost:80/<url> -O <output_file>*, then the file at the requested url should be downloaded and saved in the *output_file* .

Submission Directory :

.<id>_week3

|-- qa

|--server.cpp

|--http_server.cpp

|--http_server.h

|--nested_html/

|--Makefile