CS2910 Lab 4: HTTP Server

Lab Assignment

This is a team assignment; each team should be two members unless a different size is approved by the instructor.

Introduction

The goal of this lab is to write a short Python program, to respond to HTTP requests and return web resources, acting as an HTTP server.

Procedure

1. Download the skeleton Python template: httpserver.py
2. Edit the header of the file to include your team members' names and usernames. (Note: the header format has changed from prior labs.)
3. Complete the **handle_request** method to parse a request and respond by returning the designated resource. You will want to add other helper methods, but do not change the any other code provided in the template.
    - Note that this method will be invoked on a separate thread for each request received. This means that there may be multiple copies of this method running simultaneously, if the web client opens more than one connection at a time (e.g., to download resources that are referenced in a main HTML file).
    - For this reason, you should not rely on any global variables, but instead pass data as arguments to related functions. Each thread will have its own execution stack.
4. Add comments *at the end* of your Python file, with the following information:
    - A description of the functionality you implemented and the results of your testing.
    - Comments on your experience in completing the lab, including any problems you encountered. Briefly explain what you learned.
    - Questions and suggestions.

You **may not** use a prebuilt library like **Lib/BaseHTTPServer**; the point of this lab is for you to understand the low-level implementation of the HTTP protocol.

You **may** and **should** use the utility functions that are included near the end of the skeleton template file. Read the description for each function and ask the instructor if you have questions about them.

Additional implementation details will be discussed in class. If you have questions about these requirements, ask in class or lab.

If this base functionality turns out to be too easy, you may experiment with adding additional functions, but be sure the basic requirements are still met.

Try to divide up the primary responsibility for parts of the program in an equitable way.

Assignment details

- Your server is only expected to handle "file" resources, so that you can service a client request by returning the contents of a file associated with the resource identifier.
    - You must be able to serve at least the following resources. Download each of them from the table below: (You can find this table in the upload page, where the right-click works.)

Relative URL	**File path, relative to the directory with your Python server code file**	**File to serve (right-click to download)**
/ (default)	./index.html	index.html
index.html	./index.html	index.html
sebern1.jpg	./sebern1.jpg	sebern1.jpg
style.css	./style.css	style.css
e-sebern2.gif	./e-sebern2.gif	e-sebern2.gif

- You must parse the request **Request-Line** and all request header lines, storing the key/value pairs in a Python dictionary.
    - Unless you implement additional functionality, it is unlikely that you will need to make use of any of the request headers, but you should store and print them after the entire request is received, so you can verify that you are handling the request correctly.
- You must return an appropriate response **Status-Line** and header lines to the requesting client.
- Use a Python dictionary to store the response header lines, and then send them all at once at an appropriate time.
- The response header lines must include:
    - A **Date** header (in proper RFC format) indicating the time that the request was satisfied.
    - A **Connection** header indicating that a persistent connection will NOT be used.
    - A **Content-Type** header with an appropriate MIME type (you should use the provided function to get the MIME type).

    - A **Content-Length** header to specify the size of the resource being returned (if there is one to return).
        - You are not required to use chunked encoding for any file type.
        - Optionally, you may use chunked encoding for text/html resources. If you do so, you must include the appropriate **Transfer-Encoding** header instead of **Content-Length**, and format the resource data appropriately in the response.

Hints and Notes

- As in the HTTP client lab, you will have to both send and receive on the TCP connection to the HTTP client. On the receiving side, since we will only be handling GET requests with no entity bodies, there will likely be only one kind of data that needs to be processed:
    1. "Textual" data, organized as a sequence of characters followed by the CR/LF pair. Data in this category includes:
        - The HTTP **Request-Line**.
        - Request header lines.
        - "Blank" lines (e.g., to terminate a header). You should probably have a "read line" function from the HTTP client lab, which you can likely use here.
- Remember that when you read from the network stream with a function like **recv**, or from a file with a function like **read**, you can only control the **maximum** number of bytes that will be returned. You will always get at least one byte, unless there is no more data (in the case of a file or a socket that has been closed), but there is no way to predict absolutely in advance how many bytes will be available when you make the recv or read call.
    - At times, you may get fewer than the number of bytes needed (e.g., in a block of "binary byte" data). If this happens, you will have to make another recv call to get additional data.
- When serving resource data from a file, open the file in binary ('rb') mode to avoid problems with line-ending modification on Windows.
- Getting the proper HTTP "Date" value can be a little tricky. You can try something like this:

```
timestamp = datetime.datetime.utcnow()
timestring = timestamp.strftime('%a, %d %b %Y %H:%M:%S GMT')
                    #Sun, 06 Nov 1994 08:49:37 GMT
```

- When you have questions you can't resolve, consult the instructor as soon as possible, in person or by email.

Submission (Due Friday, Week 6, 11PM)

One team member should submit your Python file by uploading it to the upload page (which is also linked from the Schedule).

(Acknowledgements: The original version of this lab written by Dr. Sebern.)