CS536 Lab1: Build Your Own Concurrent Web Server

Due Date: Friday, Feb 21 2020 (23:59:59 PM), Total points: 100 points

1 Goal

In this project, we are going to develop a Web server in **Python 2.7**. Also, we will take the chance to learn a little bit more about how Web browser and server work behind the scene.

2 Instructions

1. Read Chapter 2.2 carefully. The textbook will help you to understand how HTTP works.
2. HTTP Protocol is defined by IETF RFC1945. You can refer the below link to know more about HTTP. `https://tools.ietf.org/html/rfc1945section-8.1`
3. You must work individually on this assignment. You will write Python code that compiles and operates correctly on the XINU machines (xinu1.cs.purdue.edu, xinu2.cs.purdue.edu, etc.) found in HAAS 257.
4. You are going to implement a web server as follows

   - The web server handles one HTTP request at a time. Your web server should accept and parse the HTTP request, get the requested file from the servers file system, create an HTTP response message consisting of the requested file preceded by header lines, and then send the response directly to the client. If the requested file is not present in the server, the server should send an HTTP "404 Not Found" response message back to the client. If a file is present but the proper permissions are not set, a permission denied error is returned.
   - You should also note that web servers typically translate "GET /" to "GET /index.html". That is, index.html is assumed to be the filename if no explicit filename is present.
   - For the debug purpose, your web server should dump the request messages to the console. This is a good chance to observe how HTTP works.
   - Please name your source file webserver.py. Your program should accept the port number as a command line argument. Instead of using port 80 or 8080 or 6789 for the listening socket, you should pick your own to avoid conflicts. It is suggested not to use port numbers 0-1024. The following is an input command format to run the server.
     `webserver.py server_port`
   - For this assignment, you will need to support enough of the HTTP protocol to allow an existing web browser (Firefox, Safari or Chrome) to connect to your web server and retrieve the contents of sample pages from your server. (Of course, this will require that you copy the appropriate files to your server?s document directory - please name the server document directory as Upload). We will be mainly checking for txt files and images.
   - Instead of using a browser, write your own HTTP client to test your server. Your client will connect to the server using a TCP connection, send an HTTP request to the server, and display the server response as an output. You can assume that the HTTP request sent is a GET method. The client should take command line arguments specifying the server IP address or host name, the port at which the server is listening, and the path at which the requested object is stored at the server. The following is an input command format to run the client.
     `webclient.py server_host server_port filename`
     The client should save the files in the folder named as Download.

- At a high level, your web server will be structured somewhat like the following:
  Forever loop:

  Listen for connection

  Accept new connection from incoming client

  Dump HTTP request to the console and Parse HTTP request

  Ensure well-formed request (return error otherwise)

  Determine if target file exists and if permissions are set properly (return error otherwise)

  Transmit contents of file to connect

  Close the connection
- Currently, the web server handles only one HTTP request at a time. Implement a **multithreaded** server that is capable of serving multiple requests simultaneously. Using threading, first create a main thread in which your modified server listens for clients at a fixed port. When it receives a TCP connection request from a client, it will set up the TCP connection through another port and services the client request in a separate thread. There will be a separate TCP connection in a separate thread for each request/response pair.

5. Pay attention to the following issues when you are implementing the project:
   - If you are running the browser and server on the same machine, you may use localhost or 127.0.0.1 as the name of the machine.
   - Make sure your *"Content-Type"* function support text/html, JPEG and GIF images.
6. To test your server, you first put a html file in the directory of your server, or more exactly, where you run your server. Connect to your server from a browser with the URL of http://<machine name>:<port number>/<path-to-html-file> and see if it works. Your browser should be able to show the content of the requested html file. You then can use any images to test your program. You can use "diff" or "md" to ensure that the transferred file is the same as the original one.

3 Materials to turn in

You will submit your assignment on blackboard. You submission should zip all the source files, here, webserver.py, and name it as "lab1_UID*.zip"

Note:

You do not have to write a lot of details about the code, but just adequately comment your source code, especially when any part of your assignments are incomplete.

Questions about the assignment should be posted on blackboard and Piazza.

4 Useful Reference

A tutorial for socket programming and http server for Python 2.7 can be found in:

- `https://docs.python.org/2/howto/sockets.html`
- `https://docs.python.org/2/library/simplehttpserver.html`