CS 422 Lab1: Build a simple HTTP web server

Due date: Tuesday, March 3 2020, 23:59:59 pm

1 Goal

In this lab, you will use Python 3.6 or 3.7 to build a web server. You will learn some knowledge about socket programming as well as how web browser and server work behind the scene.

2 Implementation

- Programming environment: Your code should be able to compile and run correctly on the XINU machines (*xinu1.cs.purdue.edu, xinu2.cs.purdue.edu,...*). To SSH into those machine, in terminal type *ssh {your purdue username}@xinu{1-10}.cs.purdue.edu*.
- Write your own HTTP server and name it as "myserver.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.

  ```
  myserver.py server_port
  ```

- First you need to support 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.
  - In order to achieve this, one necessary thing is that you copy the appropriate files to your server's document directory (put its name as "Upload"). You may hardcode a simple *index.html* file in which we can see a page showing the link list of files on the Upload folder, and also we can see the content of a file when we click its link.
  - Your browser should work for txt files and images, which means it should be able to show the file content when you enter the URL of the file. The format of an URL is

    ```
    {ip_address}:{port_number}/filename
    ```

  - 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.

- Write your own HTTP client and name it as "myclient.py". 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.

  - Learn the format of an HTTP request and HTTP response. The format of an HTTP request is

    ```
    <method><request-uri><http-version>
    ```

    and the format of an HTTP response is

    ```
    <http-version><status-code><reason-phrase>
    ```

  - HTTP/1.1 defines seven methods: GET, HEAD, POST, PUT, DELETE, TRACE, and OPTIONS. In this project, you will only need to support the first two, GET and HEAD. 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.
  - There at at least three cases that you need to implement: (1) if the request is accepted, just return the requested object; (2) if the request file is not present in the server, the server should send an HTTP "404 Not Found" response message back to the client; (3) if a file is present but the proper permissions are not set, a permission denied error is returned. To test this, you can modify the file permission in your Python code.
  - The following is an input command format to run the client:

    ```
    $ myclient.py serverIP serverPort filename command(GET or HEAD)
    ```

    Note that when implementing the GET method, you should not only return the corresponding message, but also save the requested file in a folder named "Download".

- HTTP/1.1 enables browsers to send several HTTP requests to the server on a single TCP connection. In this lab, you should implement your server which can handle a persistent connection. If a client sends multiple requests through a single connection, the server MUST keep the connection open. If a request includes the "Connection: close" header, then that request is the final one for the connection and the server should close the connection after sending the response. Also, the server should close an idle connection after some timeout period (in this lab, just choose 15 seconds).

  This requirement should work both in browsers and your client. When implementing your client, you may need to modify the input command format to send the requests at the same time:

  ```
  $ myclient.py serverIP serverPort filename1 command1 filename2 command2
  filename3 command3 ...
  ```

or build the connection at first:

```
$ myclient.py serverIP serverPort
```

and then allow users to input requests. Denote how to run your client to test the persistent connections in the README file.

- 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.

3 Submission and Grading

For the submission, you need to submit a zipped file named as "*YourPurdueID_lab1.zip*" which contains:

- A README file that contains instruction to run your code.
- *myserver.py* and *myclient.py.* The function of each part in your code should be commented.
- A folder named Upload, which should contain one text file (.txt) and 5 image files (.jpg or .png). You can choose your own files. Please note each file should not be larger than 1MB.
- A folder named Download, which should be empty.
- 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.

Please note we will carefully check code plagiarism behavior. You may discuss with classmates. However, copying code from the Internet will lead to 0 score for you, and copying others will lead to 0 scores for both.

4 References

- HTTP protocol:

  ```
  https://tools.ietf.org/html/rfc1945#section-8
  https://tools.ietf.org/html/rfc1945#section-6
  https://www.tutorialspoint.com/http/http_responses.htm
  https://www.tutorialspoint.com/http/http_requests.htm
  ```

- Socket programming:

  `https://docs.python.org/3/library/http.server.html`
  `https://docs.python.org/3/howto/sockets.html`