Computer Networks Mini Project : Web Server

29th March 2017

First Part (20% of the marks)

You will develop a web server that handles one HTTP request at a time. Your web server should accept and parse the HTTP request, get the requested file from the server's 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" message back to the client.

Running the Server

Put an HTML file (e.g., HelloWorld.html) in the same directory that the server is in. Run the server program. Determine the IP address of the host that is running the server (e.g., 128.238.251.26). From another host, open a browser and provide the corresponding URL. For example:

http://128.238.251.26:6789/HelloWorld.html

'HelloWorld.html' is the name of the file you placed in the server directory. Note also the use of the port number after the colon. You need to replace this port number with whatever port you have used in the server code. In the above example, we have used the port number 6789. The browser should then display the contents of HelloWorld.html. If you omit ":6789", the browser will assume port 80 and you will get the web page from the server only if your server is listening at port 80.
Then try to get a file that is not present at the server. You should get a "404 Not Found" message.

Second Part (30% of the marks)

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.
You should also handle uploading files using POST/PUT methods in this part.

Third Part (30% marks)

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. It should handle for any HTTP 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.

```
client.py server_host server_port filename
```

Fourth Part (20%)

Handle authentication to get the files requested. The authentication can be username/password based or cookies based.

Bonus

Handle HTTPS requests also.
Prevent DoS attack by blacklisting the client after repeated requests.

What to Hand in

You will hand in the complete server code (for single request and multithreaded separately), client code along with the screen shots of your client browser, verifying that you actually receive the contents of the HTML file from the server. Put them in two folders, one for the codes and README files and the other with the screenshots. Follow this format for submission: RollNo1_RollNo2.zip. The deadline for the project is 11th April 2017.