CSCI 4131 – Internet Programming

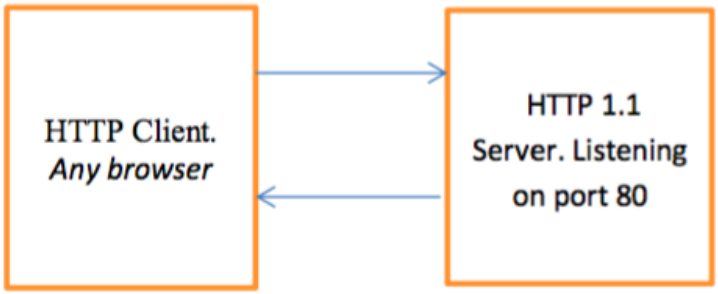
Assignment 6

Posted: 10/18/2016 – Due 11/1/2016 at 11:55pm

1 Description

The objective of this assignment is to learn HTTP protocol (HTTP1.1) and build a tiny HTTP server. In this assignment you would learn how to program, using **Python 3** and TCP sockets, the functionalities of an HTTP client and server. You will need to go through RFC 2616 for HTTP 1.1 protocol details.

When a web client (such as Google Chrome) connects to a web server (such as ***www.google.com***) the interaction between them happens through Hyper Text Transfer Protocol (HTTP).



The working of the basic HTTP server is described below in brief:

1. An HTTP client connects to the HTTPserver and makes a HTTP request to request a web resource.
2. The HTTP Server parses the request header fields. The request can be of type GET, POST, HEAD, etc. For a GET requests, the server identifies the requested resource (for example an HTML file) and checks if the resource exists.
3. The HTTP server then generates an appropriate HTTP response message. If the requested resource is found, the HTTP server includes successful (2xx) status code in the response headers along with other meta data such as the Content Type and Content Length and sends the resource data as the message body. If the requested resource is not found, then the HTTP Server sends response with status code 404.

2 Preparation and Provided Files

You server should be written in Python 3. While HTTP servers can be written in any language with socket programming available, Python is a language that is widespread in the web development world. Type the command ***idle3*** to get into the `Python3` development environment.

In the unlikely event that `Python 3` is not installed in your login environment on the CSElabs machines already, you may have to load a module to access `Python 3` on the CSELabs machines. You can use the command *`module load soft/python/3.5.2`* to do so, and see the operator if that does not work.

The following files are provided for this assignment:

- 403.html: this file should be sent to the client after forbidden header code.
- 404.html: this file should be sent to client when the requested file is not found.
- private.html: this file is the private file that triggers 403 forbidden code.
- calendar.html: replace this file with your own calendar.html file from Assignment 1.

To give above files proper permissions, please execute following commands after unzipping in your folder:

chmod 640 private.html

chmod 644 403.html

chmod 644 404.html

chmod 644 calendar.html

3 Functionality

When started, your server will establish a socket and bind to a port, listening for connections. You can send a request to the server with your web browser by pointing it at `<host>:<port>/calendar.html`. In most cases, `<host>` will be localhost and `<port>` should default to `9001`.

Your code should not use the *httplib* module of the Python standard library. You should do your own socket programming on this assignment. In class, we reviewed client and server programs for sending text messages, and posted them on moodle. You may use them as a basis for your solution.

When your server is called with no arguments, it will bind to port 9001 and serve requests. It should also accept one optional command line argument which specifies the port to bind to. Two example invocations are:

- *python3 server.py*
- *python3 server.py 9002*

Additionally, your server should log any incoming requests to STDOUT.

4 HTTP Protocol

HTTP is a protocol of non-trivial size. You will only implement a small, functional subset of HTTP: specifically: GET and HEAD requests.

4.1 GET Requests

GET requests are the most commonly encountered requests. When your web browser requests a webpage from a server, it is issuing a GET request. Here is an example GET request that will be received by your server:

```
GET /calendar.html HTTP/1.1
Host: localhost:9001
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:33.0) Gecko/20100101
Firefox/33.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

Note that the newlines in the HTTP header are actually CRLF (carriage return line feed) characters, **Your header must be followed by two CRLF characters to work.**

4.2 HEAD Requests

HEAD requests are almost identical to GET. Instead of returning a status code followed by the requested URL, your server should only send the status code in response.

4.3 Response payload for error conditions

You will need to handle error conditions, as specified on the next page.

Note, your server should also include an appropriate error message in the response body, specific to the error condition. If we do not provide an html file for the error code, include an appropriate error message plain text in the response to the client (i.e., Browser, Telnet, curl, etc.).

Your HTTP server should handle following error conditions: 403, 404, 405, and 406. It should send appropriate error responses as specified below.

1. If the requested resource does not have appropriate permissions (i.e. it is not world-readable), 403 error response should be sent

2. If the requested resource is not found, 404 error response should be sent

3. If the request is a POST request, then server should send 405 – method not allowed response.

4. If the request contains accept headers, and the content characteristics of the requested resource(s) is/are not acceptable according to the accept headers, then a 406 response should be sent. For example, if the accept headers in the request specify only html files will be accepted, and the requested entity is an image file, then 406 error response should be sent. Your server should support at least the following types of file extensions: .html, .jpeg, .gif, .pdf, .doc and .pptx.mod

4. 4 Redirection Response

Additionally, your server will also send redirection responses for certain URLs. If the request is for a resource named "csumn", then the client should be redirected to the following location: `https://www.cs.umn.edu/.`

For example if the URL requested is:

`http://comp1.cs.umn.edu:5555/csumn`

the browser should automatically be redirected to the URL: `https://www.cs.umn.edu/.`

To accomplish this, your server should send an appropriate response message with required headers. Your server will have to include appropriate location headers in the response. The redirection response would include "Permanently moved" status code. Please refer to section 10.3 of RFC 2616 for details.

5.Testing Guidelines

To run your HTTP server, you should pick a port number above 5000. You can test the HTTP server using a HTTP client, a browser, or telnet as follows:

a) *Suppose you are running the server on a machine named "silver.cs.umn.edu" and on port number 5555, then your URL would be http://silver.cs.umn.edu:5555/<resource-path>.*

b) *Use telnet in the linux terminal (http://www.blog.tonycode.com/tech-stuff/http-notes/making-http-requests-via-telnet/). Eg:*

```
$ telnet localhost 80
Trying 207.46.232.182...
Connected to microsoft.com.
Escape character is '^]'.
GET /index.html HTTP/1.1
Connection: close
```

c) *Write your own python HTTP client*

If you run the HTTP server on any of the CSE lab machines, then you will not be able to connect to your server from any machine outside the CSE domain due to firewall. Therefore, you should run both the client (or browser) and server on CSE machines only. Alternatively, you can run the HTTP server and any client on your home machine (but make sure your code runs correctly on the cselabs machines to ensure you get full credit).

Other unix/linux commands that you may find very useful for testing your server include the **cURL** command (which supports HTTP scripting) and the **wget** command.

6 Submission Instructions

- Submit your server program in a file named: <UMN x500> server.py
  For example, if your x.500 id is: *john1234* you should submit *john1234_server.py*
- You don't need to provide any extra files. We will use our own 403.html, 404.html, private.html, and calendar.html when we test your code.

7 Evaluation

Your submission will be graded out of 100 points on the following items:

• Server establishes a socket and binds to 9001 by default.	**10 points**
• Server accepts a parameter to change port.	**5 points**
• Server accepts connections from clients and reads incoming messages.	**10 points**
• Server correctly identifies GET and HEAD requests.	**10 points**
• Server correctly responds with 200 and serves requested page.	**15 points**
• Server correctly responds with 406.	**10 points**
• Server correctly responds with 405.	**5 points**
• Server correctly responds with 403.	**5 points**
• Server correctly responds with 404.	**5 points**
• Server redirection 301.	**10 points**
• Server logs requests to STDOUT.	**5 points**
• Source code is documented and readable.	**5 points**
• Submission instructions are followed.	**5 points**

Reminder: All assignments will be graded on the cselabs machines - so make sure to test

your server on a cselabs machine to ensure it functions correctly as specified in the evaluation criterial above.