Overview

The goal of this project was to implement a basic HTTP server that could support all requirements and run on a Linux server environment as proposed by the instructor. A HTTP server allocates a server socket, binds it to a port and then listens for incoming connections. The server receives and accepts an incoming client connection with an input data stream it must parse into an HTTP request. Looking at the parameters of the request, a response is formulated and sent back to the client.

The basic features required in this implementation are:

1. Both a Single and Multiple connection mode
   1. Single serial connection mode
   2. Multiple-thread parallelization mode
2. HTTP GET requests with query and header parsing
3. Automatic directory listing of the current working directory
4. Static file transport of files within the current working directory
5. Basic CGI support demonstrated by a sample server side CGI script

General testing is done through confirmation of access to a server through a Firefox browser. From the browser and monitoring of the responses in the command line, we observe different connections to the server and requests mirroring one of the features described above.

Design

The basic idea behind a simple HTTP web server consists of the server waiting to be sent an HTTP request, parsing that request, determining the content of request, generating and formatting the necessary data to be sent back in response.

The Initial foundation and barebones skeleton code for our program was some sort of RequestHandler class in which a do\_GET method parses requests and responds accordingly. With our approach to the Request Handler, we observed that parsing of the input data stream consisted of a message and header as shown in Figure 1, and then a few lines to invoke the running of our server

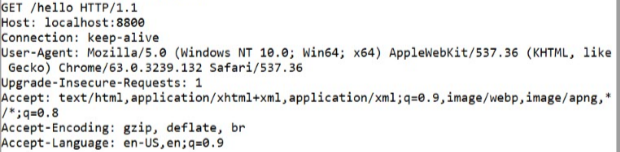


Figure 1

Each of our request handlers receives the call from the client and parses the request to determine the file extension type required and proceed to handle the mime type accordingly. When building our handlers to work for single and multiple threads we utilized the Python module of BaseHTTPServer to be used as a basis for building a functioning webserver. Listening at the HTTP socket to dispatch requests to our handler, this module serves as a strong foundation for creating on top of transmission control protocol. We extend and utilize this module to then serve files from the current directory and directly map the directory structure to HTTP requests.

When our webserver is accessed, the handler receives the request and looks at the full path to determine what actions should be taken or what is being asked of the server to perform. By default, when connecting to ‘localhost:portnumber’ the server will generate a directory listing through the use of our get\_dFiles() method. This method uses ox.listdir to scan the current working directory for files and also maintains a record of the initial starting root directory for troubleshooting issues encountered later. The directory contents are displayed in a pleasing way and as hyperlinks. When clicking a file or directory, another HTTP request by the client is sent to the server depending on which link was clicked. When receiving a request, our do\_GET method after parsing the header and determining the mimetype of the file extension within the request performs an action and sends a reply on a case by case basis. If the client is requesting to view an html file, the server will open that file and write the page to the client. If the client is requesting to execute a cgi script, the server reads in the .cgi file and executes it line by line to the client. We have already described above the generation of a directory listing if the client request access to a subdirectory.

**Paragraph about threaded server**

Functionality Walkthrough

To demonstrate the functionality of our program, we will walk through the testing process to confirm each requirement above was met satisfactorily.

Our sample working directory containing the code for our server is constructed as described below.

/

|---- cgi-bin

| |-------cgi script file

|-----http web server python file

|\_\_\_sub directory

|-------html file

|\_\_\_\_sample html file

Procedure

* To execute our simple HTTP web server simply enter the following into your machine’s command line.
  + For single serial connection, python http\_server.py –s
  + For multi thread connection, python http\_server.py –m
* SCREENSHOT
* By default, the port used is 8080 and we navigate to <http://localhost:8080> to confirm.
* SCREENSHOT
* The web browser displays the current working directory in which the server was initialized from our device.
* SCREENSHOT
* Next, we demonstrate directory navigation and static file transport, by selecting the subdirectory and a sample html file within.
* SCREENSHOT
* The next feature to be demonstrated is a the execution of a cgi script, located within the cgi-bin directory.
* SCREENSHOT OF SAMPLE SCRIPT
* SCREENSHOT OF EXECUTION
* Finally we will show our server running in multi-thread mode and multiple machines connecting to the server
* SCREENSHOT OF TERMINAL-description showing multiple IP addr have connected
* Screenshot of 3 different machines connecting and maybe cycling through directories in terminal.