CSDS 325/425: Computer Networks
Project #3
Due: October 19, 11:59 PM

The third project of the semester involves writing a simple web server. The aim of this project is to think about and implement the server half of client-server applications.

Overview

The usage of your program—which will be called *proj3*—is as follows:

```
./proj3 -p port -r document_directory -t auth_token
```

Specifically:

- The "-p" option must be present when running your web server and will specify the server port number your project will use.
- The "-r" option must be present when running your web server and will specify the directory from which your web server will serve files. All files in this directory and all sub-directories (arbitrarily deep) will be made available via your web server.
- The "-t" option must be present when running your web server and will specify an "authentication token" that will be needed to terminate your web server (as explained below).
- The command line arguments may appear in any order.
- Unknown command line arguments must trigger errors.

-p option

The "-p" option must be present on all invocations of your web server. The argument to this option specifies the port number on which the server should listen for incoming connections from web clients.

Note: Port numbers up to 1024 are "reserved" for "well known" services. The operating system will therefore not allow normal user-level programs to use these ports. When testing you should pick random ports in the range $[1025, 65535]$.

-r option

The "-r" option must be present and the argument to this option specifies the root directory from which files will be served.

Example 1: Consider a web server invoked with "-r /home/mxa99/web-stuff" and a request arriving for the file "/foo.jpg". In this case, the web server will return the file "/home/mxa99/web-stuff/foo.jpg".

Example 2: Consider a web server invoked with "-r /home/mxa99/web-stuff" and a request arriving for the file "/foo/bar.jpg". In this case, the web server will return the file "/home/mxa99/web-stuff/foo/bar.jpg".

-t option

The "-t" option must be present and the argument to the option specifies an "authentication token" that the new HTTP "TERMINATE" method described below will use.[1]

[1] The "TERMINATE" method has been fabricated for this assignment and is not part of the actual HTTP protocol.

Operation

Your program will implement a minimal web server. In particular:

- Your server will accept TCP connections on the port given via the "-p" option.
- Your server will accept a standard HTTP request on an established TCP connection.
- The HTTP request must be well formed.
  - The request must start with a line of the form "`METHOD ARGUMENT HTTP/VERSION`".
  - Each line in the request must be terminated by a carriage-return and linefeed ("\r\n").
  - The HTTP request must end with a "blank" line that consists only of a carriage-return and linefeed ("\r\n").
  - The HTTP request may contain additional, arbitrary header lines. These must be accepted, but will be ignored by your server.

  When a request arrives that does not conform to all of the above rules the web server will return a minimal response of "HTTP/1.1 400 Malformed Request\r\n\r\n" to the client. At this point all processing of the request is finished.
- The last portion of the request line is "HTTP/VERSION". The VERSION part of this string is immaterial for this project and therefore should be ignored. However, you must verify the "HTTP/" portion is present (case sensitive). If the requested protocol does not start with "HTTP/" then your web server must return a minimal HTTP response of "HTTP/1.1 501 Protocol Not Implemented\r\n\r\n". At this point all processing of the request is finished.
- Your server will return "HTTP/1.1 405 Unsupported Method\r\n\r\n" if the METHOD portion of the request is not "GET" or "TERMINATE".
- When the "TERMINATE" method is requested the "ARGUMENT" is used to authenticate the client. Your server will take one of the following two approaches:
  - When the ARGUMENT given in the HTTP request matches the argument given via the "-t" option the server will (*i*) send a minimal HTTP response of "HTTP/1.1 200 Server Shutting Down\r\n\r\n" and (*ii*) terminate. Case sensitive matching must be used.
  - When the ARGUMENT given in the HTTP request does not match the argument given via the "-t" option the server will (*i*) send a minimal HTTP response of "HTTP/1.1 403 Operation Forbidden\r\n\r\n" and (*ii*) continue running and accepting further connections and requests. Case sensitive matching must be used.
- When the "GET" method is requested, the "ARGUMENT" will be a filename relative to the "document root" directory given via the "-r" command-line argument. Your server will use one of these responses:
  1. When the requested file does not begin with a "/", a minimal HTTP response of "HTTP/1.1 406 Invalid Filename\r\n\r\n" must be returned.
  2. When the requested file exists, a minimal HTTP response header consisting of "HTTP/1.1 200 OK\r\n\r\n" will be given, followed by the contents of the given file.
  3. When the requested file cannot be opened (e.g., because it does not exist), a minimal response header of "HTTP/1.1 404 File Not Found\r\n\r\n" will be returned (with no payload content).
  4. When the requested file is "/" the default file of "/homepage.html" will be used as the filename. The web server will then leverage either approach 2 or 3 above based on whether the file exists or not.

     Note: This only holds for "/" and not for other directories. E.g., "/foo/" should not return "/foo/homepage.html". To simplify your web server, it explicitly does not need to deal with requests for directory names (except for "/").

- A single space must separate every field / word in your response headers.
- There must be no whitespace at the beginning of the lines in the response headers.
- Response header lines must end with a "\r\n" per the HTTP standard. No whitespace may appear directly before the "\r\n".
- A blank line ("\r\n") must appear after all HTTP response header lines, per the HTTP standard.
- Your server will process one HTTP request per TCP connection. After the HTTP response has been sent, the TCP connection must be closed.
- Your web server need only process a single TCP connection at a time, but must process any number of TCP connections that appear serially (until a "TERMINATE" message is received).

The following is a prioritized list of HTTP responses. In other words, if multiple responses could be sent the first one encountered in this list is the one the web server must return.

1. `400 Malformed Request`
2. `501 Protocol Not Implemented`
3. `405 Unsupported Method`
4. `TERMINATE` requests:
   (a) `200 Server Shutting Down`
   (b) `403 Operation Forbidden`
5. `GET` requests:
   (a) `406 Invalid Filename`
   (b) `200 OK`
   (c) `404 File Not Found`

For instance, if the server receives a request line of "POST /foo ALTHTTP/0.2" the web server will return a 501 ("Protocol Not Implemented") because this takes priority over the 405 ("Unsupported Method") response.

Final Bits

1. The usual generic programming guidelines apply. A copy will be posted to the project 3 web page.
2. Submission specifications:
   (a) All project files must be submitted to Canvas in a gzip-ed tar file called "[CaseID]-proj3.tar.gz".
   (b) Your submission must contain all code and a Makefile that by default produces an executable called "`proj3`" (i.e., when typing "make").
   (c) Do not include executables or object files in the tarball.
   (d) Do not include sample input or output files in your tarball.
   (e) Do not include multiple versions of your program in your submission.
   (f) Do not include directories in your tarball.
   (g) Do not use spaces in file names.
   (h) Every source file must contain a header comment that includes (*i*) your name, (*ii*) your Case network ID, (*iii*) the filename, (*iv*) the date created and (*v*) a brief description of the code contained in the file.
3. Your project must be written using sockets-based code and implement the required parts of HTTP. You may not leverage a third-party library for these tasks. Projects that rely on extra libraries will be returned ungraded.
4. Your submission may include a "notes.txt" file for any information you wish to convey during the grading process. We will review the contents of this file, but not of arbitrary files in your tarball (e.g., "readme.txt").
5. There will be samples on the class web page by the end of the day on October 3.
6. Print only what is described above. Extra debugging information must not be included. Adding an extra option (e.g., "-v" for verbose mode) to dump debugging information is always fine.
7. Do not make assumptions about the size of the files or the number of files in the directory you are serving. Likewise, do not make assumptions about the directory structure.
8. If you encounter or envision a situation not well described in this assignment, please Do Something Reasonable in your code and include an explanation in the "notes.txt" file in your submission. If you'd like to ensure you're on the right track, please feel free to discuss these situations with me.
9. Hints / tips:
   (a) Needlessly reserving large amounts of memory in case it may be needed (e.g., for serving large content) is unreasonable.
   (b) Errors will be thrown at your project.
   (c) You may leverage the example sockets code we reviewed in class. This code is available from the class web page.
10. *WHEN STUCK, ASK QUESTIONS!*