CS425:Mini Project 2

HTTP Proxy Server

Do not share code or review anyone else's code. Work on this project is to be your own.

Submit your project via as a single compressed and zipped file (using `tar` and `gzip`); include only source code and documentation files (if any).

To package up your submission, use `tar` and `gzip`. More specifically, create a compressed `tar` file , as in `16111029.tar.gz`, that contains your source files (e.g. `main.c` and `file2.c`); include a `readme.txt` file only if necessary. Here's an example:

```
bash$ tar cvf 16111029.tar main.c file2.c readme.txt
main.c
file2.c
readme.txt
bash$ gzip -9 16111029.tar
```

Be sure to comment your code and include your name at the top of each file submitted.

Write an HTTP proxy server in C that supports filtering based on server domain name. Your proxy server must correctly handle GET, HEAD, and POST requests. Further, your server must refuse to process any request that specifies any other HTTP request method (or any other unknown request) sent by a client speaking HTTP version 1.0 or 1.1.

Your proxy must work with any HTTP client (browser); assume that we will test your server with a custom HTTP client designed to validate messages sent to (and find problems with) your proxy server.

Domain Filtering:

Your proxy server must be capable of filtering out requests to Web servers within some DNS domains. More specifically, you must support both prefix and suffix filtering. For example, your proxy could be instructed to filter out any request made to a server whose name ends in `advertisements.com` or whose name starts with `www.cs`. When your server detects a request that should be filtered, your server should return an HTTP error 403 (forbidden), which means you need to send back an HTTP status line that indicates an error.

Your proxy server obtains, via the command line, the list of domains to be filtered. The first command-line argument will be the port number to use to receive connections (the listener); the remaining arguments (if any) are the domains (or domain suffixes) that should be filtered. Below is an example of a command line that could be used to run your proxy server on port 8888 and filter out requests to `doubleclick.com` and `yimg.com`:

```
./proxy 8888 doubleclick.com yimg.com
```

The filtering you must implement is based entirely on strings, which could include domain names and/or IP addresses. For example, if you are filtering `128.113`, the following request line should result in a 403 error.

```
GET http://128.113.126.13/index.html HTTP/1.1
```

Key Requirements:

1. Your proxy server must filter based on domain name (or IP address) prefixes/suffixes as described above. All command-line arguments following the port number are domain or IP address prefixes/suffixes that must be filtered. There can be zero or more of these; and there is no limit to the number of arguments.
2. Your server must handle GET, HEAD, and POST request methods.
3. Your server must refuse to process any HTTP request method other than GET, HEAD, and POST. In such cases, you should send back an HTTP status code of 405 (Method not allowed) or 501 (Not Implemented) if you receive any other request method.
4. Your server must forward the appropriate HTTP request headers to the requested server, then send the responses back to the client.
5. Your server must send an error to the client whenever appropriate, including such cases as the request line being invalid, a `Host:` header is not found, or a `POST` request does not include a `Content-Length:` header. In these cases, your server must send a 400 (Bad Request) as a result.
6. Your server does must be a concurrent server (i.e. do not use an iterative server). If you decide to use `fork()`, be sure you don't leave zombie processes behind.
7. We must not be able to kill your server simply by sending an invalid request.
8. We must not be able to kill your server by stopping or killing the client.
9. We must not be able to kill your server by sending it a `SIGINT` signal.
10. Your server cannot get larger (use more memory) every time it processes a request (i.e. no memory leaks!).
11. Your server does not need to support port numbers in URIs. We will not test this.
12. Your server does not need to support the `https` protocol (though you can support it if you'd like to).
13. Your server does not need to support all of the fancy features of HTTP version 1.1 (i.e. persistent connections, pipelining, etc.). See the note below about HTTP persistence.

Server Output:

To track requests, your proxy server must print exactly one line (to `stdout`) for each request serviced. The line should include the host name or IP address of the *client*, plus the original request-line sent by the client (not including any headers that accompanied the request). Also print one line for each filtered request, indicating that your proxy server did not process the request. For example, the following might be the output generated by your proxy server if it received requests from a client running on `monica.cs.rpi.edu`:

```
bash$ ./proxy 8888 doubleclick.com slashdot.org
monica.cs.rpi.edu: GET http://www.cs.rpi.edu/
monica.cs.rpi.edu: GET http://www.cs.rpi.edu/gfx/backg5.jpg
monica.cs.rpi.edu: GET http://www.cs.rpi.edu/gfx/logo.jpg
monica.cs.rpi.edu: GET http://www.slashdot.org/foo/blah [FILTERED]
monica.cs.rpi.edu: GET http://www.yahoo.com/images/annakornikova.jpg
monica.cs.rpi.edu: GET http://fred.com/purchase.asp?prod=17&qty=102
monica.cs.rpi.edu: HEAD http://www.slashdot.org/ [FILTERED]
monica.cs.rpi.edu: POST http://www.fbi.gov/insecuresubmission.cgi
```

Signal Handling:

When your server receives a `SIGUSR1` signal, your server should output a summary of what prefixes/suffixes it is filtering, as well as the set of statistics shown in the example below:

```
bash$ ./proxy 8888 doubleclick.com slashdot.org
monica.cs.rpi.edu: GET http://www.cs.rpi.edu/
monica.cs.rpi.edu: GET http://www.cs.rpi.edu/gfx/backg5.jpg
monica.cs.rpi.edu: GET http://www.cs.rpi.edu/gfx/logo.jpg
monica.cs.rpi.edu: GET http://www.slashdot.org/foo/blah [FILTERED]
monica.cs.rpi.edu: GET http://www.yahoo.com/images/annakornikova.jpg
monica.cs.rpi.edu: GET http://fred.com/purchase.asp?prod=17&qty=102
monica.cs.rpi.edu: HEAD http://www.slashdot.org/ [FILTERED]
monica.cs.rpi.edu: POST http://www.fbi.gov/insecuresubmission.cgi
Received SIGUSR1...reporting status:
-- Processed 6 requests successfully
-- Filtering: doubleclick.com; slashdot.org
-- Filtered 2 requests
-- Encountered 0 requests in error
```

As shown above, display the number of requests successfully processed, meaning the number of requests that were not filtered and successfully pass through the proxy server. Also show the number of requests filtered, as well as the number of requests for which your proxy server responded with an error (e.g. HTTP error 403).

When your server receives a `SIGUSR2` signal, your server should exit gracefully. Therefore, to shutdown your server, you would need to send it a signal via the shell as follows (assuming the server is process id 5555):

```
bash$ kill -12 5555
```