CSSE432
Spring 2013

Project Description

Your project is to design and implement a **functioning HTTP/1.1** compliant server which **supports multiple clients simultaneously**. The webserver should have a **security component (your choice)** and should **support persistent** and **stateful connections** (using cookies or otherwise).

Programming Guidelines

Your web server may be written in any language (although C is preferred). Do not use any custom socket classes or libraries except for libssl (if you are implementing SSL support).

Invoking the server

Your web server program must accept the following command line options, in any arbitrary order. If an argument is left out when running the server, then use the specified default.

- -p port: Overrides the default port the server should run on. The default is port 8000 (note that ports under 1024 require root access to use, so we cannot use port 80 as default).
- -root path: Sets the root directory of the web server. This is where the files are served from. For example, if the server is run with
- -root ./networks/website, then a request for http://www.myserver.com/index.html will result in serving the file ./networks/website/index.html. If this option is not present, than use ./www as the default root directory.
- -sslport: Listens for an SSL connection on the specified port. Your server should be able to listen on a normal port and on an SSL port at the same time, and handle both types of connections. The default SSL port is 8001.
- -debug: When this option is present, you may print out debugging messages to the console (stdout). When this option is not present, your program may not print any output to the console.

HTTP Operations

You may refer to RFC 2616 for further information on the HTTP/1.1 specifications. You will need to support the GET, HEAD, and POST request methods. You will need to implement only the 200, 301, 400, 404, 500, and 501 error codes. Your server should also respond to the Host, User-Agent, Connection, Accept, and at least two other request headers. It should also produce appropriate response headers including but not limited to the Date, Server, Content-Length, Connection and Content-Type headers.

Persistent Connections
If an HTTP/1.1 client sends multiple requests through a single connection, the server MUST keep the connection open and send responses back in the same order as the requests. If a request includes the \Connection: close" header, then that request is the final one for the connection and the server should close the connection after sending the response. Also, the server should close an idle connection after some timeout period (can be anything, but yours should be 15 seconds). Your server must support these persistent connections. Please remember that a single client may issue additional requests while your server is still reading and the first request. In this case, your server must read in and process all requests before closing the connection.

Multiple Connections
A web server that accepts only one connection at a time is probably impractical and definitely not very useful. As such, your server should also be written to accept multiple connections (usually from multiple hosts). It should be able to simultaneously listen for incoming connections, as well as keep reading from the connections which are already open. Note that today's web browsers may open two connections to your server (as per RFC 2068), so your server should be able to handle these multiple connections.

Stateful Connections and the Security Component
You will be responsible for choosing your own method of implementing state and security in your web server. Your design should be described in your initial design document. Suggestions include implementing cookies or a database to maintain state or implementing SSL or firewalls within the web server for the security component.

Milestones

1. **Team Selection (End of Week 4)**: Teams should include 3-4 members.
2. **Initial Design Document (End of Week 5)**: Teams should submit their initial design document in the svn folder called *Project*. The document should include:
    a. The programming language to be used
    b. The socket infrastructure provided by the language (socket/bind/listen/accept).
    c. Description of your proposed implementation for stateful connections and your security component.
3. **Prototype Demonstration #1 (End of Week 7)**: Teams will demonstrate their server. Teams should demonstrate that their server starts with the right command line arguments, handles, GET and HEAD requests, and returns the correct error codes.
4. **Prototype Demonstration #2 (End of Week 8)**: Teams will demonstrate that their server supports multiple connections.
5. **Prototype Demonstration #3 (End of Week 9)**: Teams will demonstrate that their server has a security component and supports stateful connections.

6. **Final Demonstration and Final Report (End of Week 10)**: Teams will do a demonstration in class. Teams will also submit a final report to their Project folder on svn. The final report will include a final description of the server implementation and code.

Grading

Milestone	**% of grade**
2	10
3	10
4	30
5 – Demo	40
5 – Report	10