CIS 457 Project 4: Mini HTTP Server

Objective: Implement a (subset of a) real protocol.

Deliverables: You must turn in your code on blackboard by 11:59 PM on the due date. You must turn in your documentation in by the next lecture after the due date. You must demo your client and server within two days of the due date. Lab time may be available for these demos, but if there is not sufficient time, you are responsible for arranging a time to complete the demo.

Teams: This project may be done in groups of 1 - 3 students.

Grading: This project is worth 100 points (90 for functionality, and 10 for documentation), as described below.

Introduction

In this project, we will be implementing a web (HTTP/1.1) server. HTTP, defined in RFC 2616 is a large and complex protocol. Therefore, you do not need to implement the entire HTTP/1.1 specification in the RFC. You must implement the subset of the protocol described below. Your server must be able to interact with a standard web browser (a recent version of chrome or firefox). Wireshark must be able to interperit the messages your server sends.

You must write this program in Java, C, or C++ (other languages may be allowed by request). You may not use any non-standard libraries without prior permission. You also may not use any libraries which automatically parse HTTP requests. You only need to write the server portion of this application, not the client. Network communication must be done using stream (TCP) sockets. Your server must run on the datacomm lab linux computers.

Invocation

Your server should accept the following command line arguments:

- **-p** *port*: Specify the port the server should run on. If no port is specified, the default should be 8080 (because without root access, we can not use the standard port 80).
- **-docroot** *directory*: Specify the directory from which the server should look for requested files. The default should be the directory the server was run from.
- **-logfile** *file*: Specify a file for log messages to be written out to. If no log file is

specified, then log messages should only be printed on standard out.

Behavior:

Your server must do the following:

- Your server must support multiple simultaneous clients or multiple connections from the same client.
- You only need to support the GET operation, other operations such as POST do not need to be supported. You must return the appropriate status code (501) in response to such requests to indicate this.
- You must send the *date* header.
- You must send the *last-modified* header.
- You must send a *content-type* header, but you only need to support the types for html, plain text, jpeg images, and pdf files. You may of course support others.
- You must send a header indicating the length of the content you are sending.
- You must return page contents and a 200 status code if the client requests a vaild document.
- You must return a 404 status code if the client requests and invalid document. Note that a 404 response has a body: you must include the error page to be displayed on the client.
- You must correctly interperit any *if-modified-since* header sent by the client, including returning the appropriate status code (304) if the page has not been modified, and detecting if the page has been modified on disk.
- You must support persistent connections. A browser may send several requests over the same HTTP connection (even before receiving any responses). If a recieved message contains a *connection: close* header, the connection must be closed immediately after handling the request. Otherwise, the connection must be kept open for 20 seconds, and closed if no messages have been recieved after this amount of time.
- Unsupported headers in requests must not cause your server to fail.
- Your server must print all requests it recieves to the log file, along with the headers (only) of responses it sends. Printing to the log must be thread-safe: a message printed in the handling of one request should not occur in the middle of a message from another.
- You must prevent your server from accessing files outside of its docroot directory, as this is a security risk.

Testing

Please test using html files, images, text files, and pdf files in the server's docroot directory. To test post, use the simple form linked here. Pressing the button should cause a post request. To test persistant connections, use the simple page linked here. You will also need to download the images and put them in the docroot. If viewed in wireshark, multiple images should be downloaded over the same connection (same pair of port numbers).

Documentation

Documentation of your program is worth 10 points. This should be a 2-3 page document describing the design of your program. This should not be a line by line explanation of your code, but should explain the overall structure and logic of the program, as well as what major challenges you encountered and how you solved them.

Grading:

There will be a total of 100 points, divided as follows:

Criteria	Points
Proper runtime error handling	10
Proper handling of valid documents (200)	20
Proper handling of HTTP errors (404, 501)	10
Multiple simultaneous clients	10
Persistent connections	10
If-modified-since	10
Command line options	10
Log file	5
Security (docroot)	5
Documentation	10