CS 252 Lab 5: Building a HTTP Server

CS 252 Lab 5: Building a HTTP Server

IMPORTANT: Your web server should be able to serve HTTP files. Directory listing and CGI will be considered extra credit.

Code review form

Lab Slides

Frequently asked questions

Your server should look like this: http://www.cs.purdue.edu/homes/grr/cs422-root-dir-test/htdocs/

This is the Grading Form

Purpose of the Lab

The objective of this lab is to implement a HTTP server that will allow a HTTP client (a web browser like FireFox or Internet Explorer) to connect to it and download files.

HTTP Protocol Overview

A HTTP client issues a `GET' request to a server in order to retrieve a file. The general syntax of such a request is given below :
GET <sp> <Document Requested> <sp> HTTP/1.0 <crlf>
{<Other Header Information> <crlf>}*
<crlf>
where :

- <sp> stands for a whitespace character and,
- <crlf> stands for a carraige return-linefeed pair. i.e. a carriage return (ascii character 13) followed by a linefeed (ascii character 10).
- `<crlf><crlf> is also represented as "\n\n".`

- <Document Requested> gives us the name of the file requested by the client. As mentioned in the previous lab, this could be just a backslash (/) if the client is requesting the default file on the server.
- {<Other Header Information> <crlf>}* contains useful (but not critical) information sent by a client. These can be ignored for this lab. Note that this part can be composed of several lines each seperated by a <crlf>.
- `* - kleene star // regular expressions`

Finally, observe that the client ends the request with two **carriage return linefeed character** pair: <crlf> <crlf>
The function of a HTTP server is to parse the above request from a client, identify the file being requested and send the file across to the client. However, before sending the actual document, the HTTP server must send a response header to the client. The following shows a typical response from a HTTP server when the requested file is found on the server:
HTTP/1.0 <sp> 200 <sp> Document <sp> follows <crlf>
Server: <sp> <Server-Type> <crlf>
Content-type: <sp> <Document-Type> <crlf>
{<Other Header Information> <crlf>}*
<crlf>
<Document Data>

where :

- <Server-Type> identifies the manufacturer/version of the server. For this lab, you can set this to CS 252 lab5.
- <Document-Type> indicates to the client, the type of document being sent. This should be "text/html" for an html document, "image/gif" for a gif file, "text/plain" for plain text, etc.
- {<Other Header Information><crlf>}* as before, contains some additional useful header information for the client to use. These may be ignored for this lab.
- <Document Data> is the actual document requested. Observe that this is separated from the response headers be two carraige return - linefeed pairs.

If the requested file cannot be found on the server, the server must send a response header indicating the error. The following shows a typical response:
HTTP/1.0 <sp> 404 File Not Found <crlf>
Server: <sp> <Server-Type> <crlf>
Content-type: <sp> <Document-Type> <crlf>
<crlf>
<Error Message>
where :

- <Document-Type> indicates the type of document (i.e. error message in this case) being sent. Since you are going to send a plain text message, this should be set to text/plain.
- <Error Message> is a human readable description of the error in plain text/html format indicating the error (e.g. Could not find the specified URL. The server returned an error).

Procedure and Algorithm Details

Stage 0:

Getting started

Login to a CS department machine (a lab machine or `data.cs.purdue.edu`), navigate to your preferred directory, and run

```
cd
cd cs252
git clone /homes/cs252/sourcecontrol/work/$USER/lab5-src.git
cd lab5-src
```

Then build the server by typing *make*. Run the server by typing *daytime-server* without arguments to get information about how to use the server. Run the server and read the sources to see how it is implemented. Some of the functionality of the HTTP server that you will implement is already available in this server.

Stage 1:

Basic Server

You will implement an iterative HTTP server that implements the following basic algorithm:

- Open Passive Socket.
- Do Forever
  - Accept new TCP connection
  - Read request from TCP connection and parse it.
  - Frame the appropriate response header depending on whether the URL requested is found on the server or not.
  - Write the response header to TCP connection.
  - Write requested document (if found) to TCP connection.
  - Close TCP connection

The server that you will implement at this stage will not be concurrent, i.e., it will not serve more than one client at a time (it queues the remaining requests while processing each request). The server should work as specified in the overview above. Implement your server in the file "**myhttpd.cc**". Use the file daytime-server.cc as base for your server.

Adding Concurrency

You will also add concurrency to the server. You will implement three concurrency modes. The concurrency mode will be passed as argument. The concurrency modes you will implement are the following:

-f : Create a new process for each request

In this mode your HTTP server will fork a child process when a request arrives. The child process will process this request while the parent process will wait for another incoming request. You will also have to prevent the accumulation of inactive zombie processes.

-t : Create a new thread for each request

In this mode your HTTP server will create a new thread to process each request that arrives. The thread will go away when the request is completed.

-p: Pool of threads

In this mode your server will put first the master socket in listen mode and then it will create a pool of 5 threads where each thread will execute a procedure that has a while loop running forever which calls accept() and dispatches the request. The idea is to have an iterative server running in each thread. Having multiple threads calling accept() at the same time will work but it creates some overhead under Solaris (See [4]). To avoid having multiple threads calling accept() at the same time, use a MUTEX lock around the accept() call.
If you want a review of threads see Introduction to Threads.
The format of the command should be:

myhttpd [-f|-t|-p] [<port>]

If no flags are passed the server will be an iterative server like in the Basic Server section. **If <port> is not passed, you will choose your own default port number**. Make sure it is larger than **1024** and less than **65536**.

URL to File Mapping

In Apache and other web servers, the mapping from URL documents to directories in the file system is done through a configuration file. In your project you will do this mapping programmatically in the following way:

If document is icons/document in the URL, then it will serve the document from http-root-dir/icons/document.

Otherwise, the document requested will be served from http-root-dir/htdocs. The document could be a subdirectory or a file in a subdirectory in htdocs.

IMPORTANT: A URL for your server should not contain http-root-dir or htdocs in it.

IMPORTANT: Make sure that a user should not be able to request/browse files above the htdocs/ cgi-bin/ or icons/ directories. You can use the "realpath" function to translate paths with ".." and other relative paths to absolute paths.

IMPORTANT: The default page when requesting http://host:port should be the index.html in htdocs.

Extra credit

CGI-BIN

If the document is cgi-bin/script, then your server will execute the script in http-root-dir/cgi-bin.

You will implement cgi-bin **.** When a request like this one arrives:
GET <sp> /cgi-bin/<script>?{<var>=<val>&}*{<var>=<val>}<sp> HTTP/1.0 <crlf>
{<Other Header Information> <crlf>}*
<crlf>
the child process that is processing the request will call *execv* on the program in *cgi-bin/<script>*. There are two ways the variable-value pairs in {<var>=<val>&}*{<var>=<val>} are passed to the cgi-bin script: the GET method and the POST method. You will implement the GET method.
In the GET method the string of variables {<var>=<val>&}*{<var>=<val>} is passed to the <script> program

as an environment variable QUERY_STRING. It is up to the <script> program to decode this string. Also if this string of variables exists, you should set the REQUEST_METHOD environment variable to "GET". The output of <script> will be sent back to the client.
For more information on how cgi-bin works see The Common Gateway Interface <http://www.oreilly.com/openbook/cgi/ch01_01.html>.

Browsing Directories

You will add to your server the capacity to browse directories. If the <Document Requested> in the request is a directory, your HTTP server should return an HTML document with hyperlinks to the contents of the directory. Also, you should be able to recursively browse subdirectories contained in this directory. An example of how a directory should look like is indicated in http-root-dir. Check the man pages for *opendir* and *readdir*.
Also implement sorting by name, size, and modification time.

Turning in your project

1. You will presenting your projects to your lab instructor during lab time. If you will not be able to attend your lab, you are responsible for arranging another time with him for the presentation.

2. Make sure that your server uses the http-root-dir and it loads by default the index.html from this directory. Test the simple, complex test. Your lab instructors will use this directory during the presentation.
3. Write a short README file that includes:
    a) Features in the handout that you have implemented av
    b) Features in the handout that you have not implemented
    c) Extra features
Include this file in your server's directory lab5-src/
4. You still need to turnin your project electronically.
Write your program in a directory called lab5-src. Make sure that your server can be built by typing "make" in one of the lab machines.
IMPORTANT: Do not include the http-root-dir in your submitted files.
You will turn in this part electronically by typing the following command from a lab machine before your presentation:

```
turnin -c cs252 -p lab5 lab5-src
```

The deadline for the project turnin is on Friday November 11th at 11:59pm, The presentations will take place that week during your lab section.
The grade will be based on how well your server works, the organization of your code, as well as the extra features you include to your project. Do not forget the README file.

- [1] "UNIX Network Programming Vol 1" by Richard Stevens