==========

CS 50011 INTRODUCTION TO SYSTEMS II Summer 2017

Lab #3

==========

Name: _____

==========

Part 0: Introduction

NOTE: A significant portion of this lab assignment is based on a laboratory projected by Prof. Gustavo Rodriguez-Rivera.

1. Copy this week’s lab files (all files in ~jeff/labfiles/Lab03) into your Lab03 subdirectory. Be sure to use “cp -r” to recursively copy the directories as well.

==========
Part 1: Basic Web Server (50 points)

For this portion of the lab, you will build a functional HTTP/1.0 server. That is, a webserver that implements the HTTP/1.0 specification.

We suggest that you carefully study the daytime server and client examples provided in lecture 10.

You will need to become familiar with the socket API including functions like: getservbyname(), getprotobyname(), bind(), listen(), accept(), etc.

Basic GET

An HTTP client issues a “GET” request to a server when it wishes to retrieve a file. The general syntax of such a request appears below:

```
GET /path/to/file HTTP/1.0\r\n
{Additional Header Information\r\n}
\r\n
```

Note the whitespace, and the trailing carriage return and linefeed character. For this lab, the additional header information may safely be ignored.

The file does not need to be explicitly specified. It if is omitted, it should default to “index.html”. For more advanced webservers, there is generally a DirectoryIndex definition that includes one or more index files to be attempted in order.

Note that a request is ended with two carriage return and linefeed character pairs.

Response

The HTTP server parses a request in the above format, identifies and identifies the file to transmit. Before sending the file, however, the HTTP server must send a response header to the client. The following illustrates a typical response from an HTTP server, assuming the requested file is found and accessible:

```
HTTP/1.1 200 OK\r\n
Server: server-type\r\n
Content-type: content-type\r\n
{Additional Header Information\r\n}
\r\n
<Requested File Data>
```

server-type often specifies the platform and OS (eg, “Apache/2.4.6 (CentOS)”). For this lab, set it to “CS50011/1.0”

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.

The additional header information may again be ignored for this lab.

<Requested File Data> is obviously the requested file’s contents.

Errors

Should the requested file not exist, the server replies with:

```
HTTP/1.1 404 File Not Found\r\n
Server: server-type\r\n
Content-type: content-type\r\n
\r\n
<Error Message>
```

Content-type in this case refers to the format of the error message. For this lab, set it “text/plain”. Error message is a human readable description of the error in plain text/html.

URL to File Mapping

Webservers map URLs to local files. This mapping is generally specified in a configuration file in the form of a “DocumentRoot” all URLs are relative to the DocumentRoot. Eg, if one issues “GET /index.html” and DocumentRoot is “/var/www/html”, the actual file fetched is located at “/var/www/html/index.html”.

Webservers also ensure that requests do not fall outside of the DocumentRoot. Consider the request “GET /../../../etc/passwd”.

Your DocumentRoot should be hard coded.

You should also specially handle requests for “icons/file”. These requests should be served out of “http-root-dir/icons”.

All other URLs should be served from “http-root-dir/htdocs”.

Actual URLs should not include these paths.

HINT: Look at realpath().

Again, if no file is specified, it should default to “index.html”.

Sample Program

You have been provided with a tarball for this part – lab3.tar.gz – in the ~jeff/labfiles/Lab03 directory. Create your own Lab03 subdirectory and untar its contents inside of it.

```
tar -zxvf lab3.tar.gz
```

You can execute “make” to build the daytime-server. Play with it, and use it as a starting point to help build your webserver. There is also a simple client, client.c.

The Server

Create an iterative HTTP server that implements the following basic algorithm:

```
  Open a passive socket
  while (forever) {
    accept() new TCP connection;

    read request from socket and parse it;

    frame the appropriate response header depending on whether the URL
      is valid and accessible;

    write the response header to the TCP connection;

    if appropriate, write the requested document;

    close() the TCP connection;
  }
```

This server is not concurrent. That is, it is not capable of serving more than one client at a time. The remaining requests are queued.

Make a copy of the daytime server named “myhttpd.c”. Modify the Makefile to automatically build it. If you wish, you may create this program using C++.

Part of the score for this part includes a functioning Makefile.

As an aside, RFC 1945 defines the HTTP/1.0 specification. It may be worth a look.

==========
Part 2: Fork It (25 points)

Add concurrency to the above webserver. We will go with simple concurrency in this part – use fork() to create a child process to handle each incoming request. The parent should immediately resume waiting for more incoming requests while the child takes care of parsing and delivering the response to the client.

Be sure that you do not accumulate a large number of zombie processes. You are responsible for reaping the dead children.

There are a number of online resources available that help with this.

For a hardcover resource, Chapter 11 in Doug Comer’s “Internetworking with TCP/IP – Vol 3” may be particularly useful.

Name your new server “myhttpdconc.c”. Again, alter the Makefile to compile it.

==========
Part 3: Directory Browsing (25 points)

For this part, should the requested document be a directory, your HTTP server should return an HTML document with hyperlinks to the contents of the directory. You should also be able to recursively browse subdirectories contained in this directory. An example of how this might look is available here:

https://www.cs.purdue.edu/homes/grr/cs422-root-dir-test/htdocs/

You may ignore the cgi-bin directories for our purposes.