CSCI 340 HW 1

Collaboration Policy: Individual Assignment

Total Points: 100

HTTP Server Specification

Using the C server template code attached to the Dropbox, please modify the template code to become a simple HTTP server that responds to simple POST and GET requests sent from a standard web browser (e.g., Chrome, Safari, Firefox).

In general, your HTTP server shall:

- Fork a new child process for each client (i.e. web browser).
- Implement a child signal handler that reaps the child (i.e. remove the child process from the process table) when it has completed.
- Not include support for cookies
- Not include support for conditional GET
- Not include support for multipart request operations.
- Not include support for encryption/decryption capabilities.

In this assignment, you must modify the C code provided to you, and it must run on a Ubuntu 16.04 operating system. Furthermore, you may use any class or function supported in the C language specification, however you cannot change the Makefile (i.e. change the compiler from gcc to g++ etc.).

For most of you, this will be your first C program, or the first C program you've been written in several months. *I would give your self plenty of time*. Lastly, please do not copy paste existing code found on the web, your assignment will not be graded. I'm fully aware that there are plenty of solutions out on the web, and this is not a "search the web" exercise.

HTTP GET

From a web browser, if you enter the following URL

```
http://<server>:<port>/?name=CSCI&number=440&title=networks
```

where `<server>` and `<port>` must be replaced with the IP address and port number your HTTP sever is running on, then your server should return the following HTML shown in Appendix 1.0.

In general, your HTTP server shall parse any number of key/value pairs appended to the URL, i.e.

```
http://<server>:<port>/?key=value&key=value& ….
```

and then return a response message where the entity body echo's the key/value pairs in a HTML formatted table (See Appendix 2.0 for an example). For more information about the structure of the HTTP `request` message see Appendix 4.0, or more information about the structure of the HTTP `response` message see Appendix 5.0.

HTTP POST

From a web browser, if you enter the following URL

```
http://<server>:<port>/SimplePost.html
```

where `<server>` and `<port>` must be replaced with the IP address and port number your HTTP sever is running on, then your server should return a webpage that is similar to that seen in Appendix 3.0.

In general, when you submit the webpage with an HTTP `<form>` tag that uses a HTTP POST operation, your HTTP server shall parse any number of key/value pairs in the entity body of the request message, and then return a response message where the entity body echo's the key/value pairs in a HTML formatted table (See Appendix 3.0 for an example). For more information about the structure of the HTTP `request` message see Appendix 4.0, or more information about the structure of the HTTP `response` message see Appendix 5.0.

Additional Information

For more information about the HTTP protocol:

- The Internet engineering task force (IETF) request for comment (RFC) specification (https://tools.ietf.org/html/rfc7231). Essentially, this is official document that fully describes the protocol and how to implement it.

To help trouble shoot connection and protocol implementation problems your HTTP server may encounter, I suggest using Wireshark (https://www.wireshark.org/) to follow/analyze the TCP connections, and to inspect the packets sent between the client (i.e. the web browser) and your HTTP server.

Submission

Create a tarball that includes all the code required to compile and run your HTTP server (this includes your Makefile or CMake file). The name of the tarball file must be your last name. For example, *ritchie.tar.gz* would be correct if the original co-developer of UNIX (Dennis Ritchie) submitted the assignment. Assignments submitted in the correct format will be accepted. Please submit the tarball file (via OAKS) to the Dropbox setup for this assignment by the due date. You may resubmit the tarball file as many times as you like, Dropbox will only keep the newest submission.

Grading Rubric

HTTP server compiles	30 points
HTTP server runs without errors	10 points
HTTP GET test cases (3 cases 10 points each)	30 points
HTTP POST test cases (3 cases 10 points each)	30 points
	100 points

In particular, the assignment will be graded as follow, if the submitted solution

- Does not compile: 0 of 100 points
- Compiles but does not run: 30 of 100 points
- Compiles and runs with no errors: 40 of 100 points
- Passes all 6 test cases developed by instructor: 100 of 100 points.

Appendix

1.0 Example HTML returned by HTTP GET request

```
<html>
<body>

	<h1>GET Operation</h1>
	<table cellpadding=5 cellspacing=5 border=1>
		<tr>
			<td><b>first</b></td>
			<td>brent</td>
		</tr>
		<tr>
			<td><b>last</b></td>
			<td>munsell</td>
		</tr>
	<table>
</body>
</html>
```

2.0 Example SimplePost.html file

```
<html>
<head>
	<meta charset="UTF-8">
	<title>Simple Post</title>
</head>
<body>

<form method="POST" action="<server>">

	<b>first</b>&nbsp;</b><input type="text" name="first"><br>
	<b>last</b>&nbsp;</b><input type="text" name="last"><br>
	<input type="submit">

</form>

</body>
</html>
```

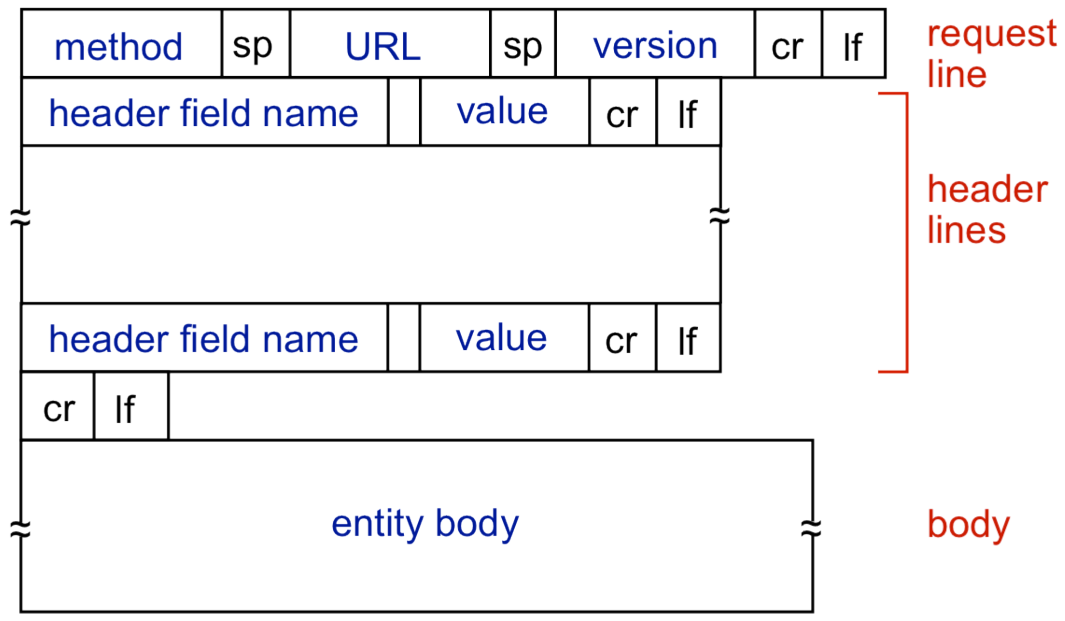
Where `<server>` must be replaced with the IP address of the machine your HTTP server is running on.

3.0 Example HMTL returned by HTTP POST request

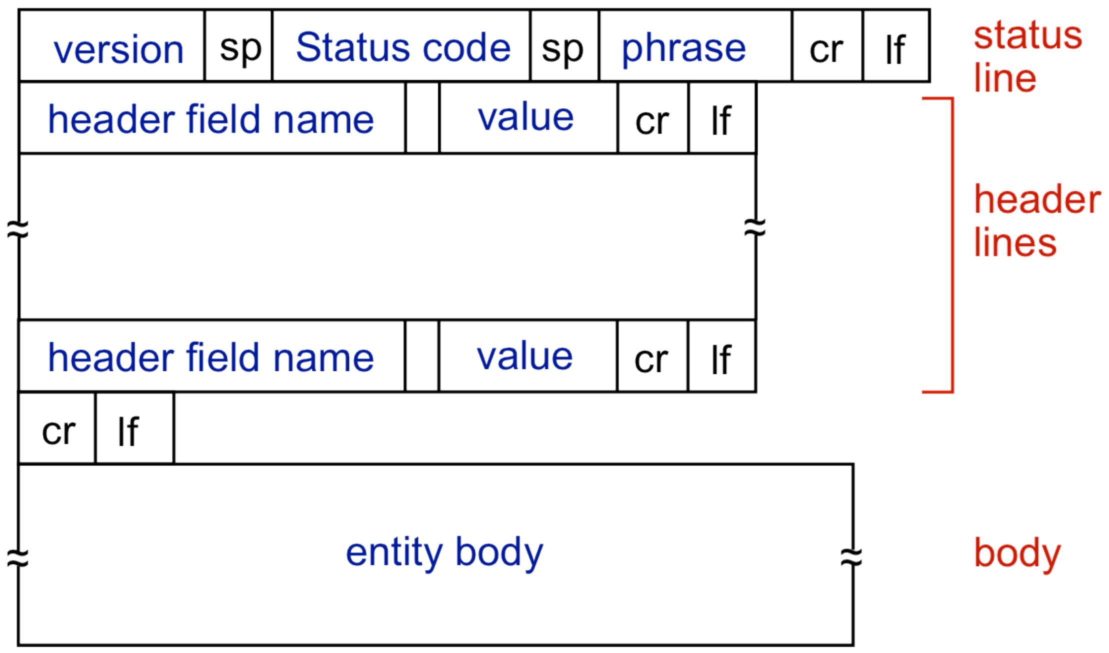
```
<html>
<body>

     <h1>POST Operation</h1>
     <table cellpadding=5 cellspacing=5 border=1>
          <tr>
               <td><b>first</b></td>
               <td>brent</td>
          </tr>
          <tr>
               <td><b>last</b></td>
               <td>munsell</td>
          </tr>
     <table>
</body>
</html>
```

4.0 HTTP Request Message Format



5.0 HTTP Response Message Format


version
sp
Status code
sp
phrase
cr
lf
status line
header field name
value
cr
lf
header lines
header field name
value
cr
lf
cr
lf
entity body
body