CSCI.4210 Operating Systems
Fall, 2009
Programming Assignment 7
Writing an HTTP client and server

This is a two part assignment. You can do it in either Unix or Windows. You will write an HTTP (Hypertext Transfer Protocol) client and server for Unix and for Windows or NT. An HTTP client is commonly called a web browser and and HTTP server is commonly called a web server. Any of the standard web browsers (Firefox, Explorer, Chrome, Safari etc) should be able to communicate with your server and your client should be able to communicate with any web server on the planet.

The HTTP Protocol is very simple. The client can send one of only three commands, `GET`, `HEAD`, and `POST`. Your client and server only need to handle `GET` commands. These are requests for documents, and the server returns the requested document or an error message and code.

The format of a get command is `GET` *documentname* `HTTP/1.1` followed by zero or more header lines which provide additional information. Your server can ignore all of the header information (although you have to read it). The document string must start with a slash (which indicates the home directory of the server).

I ran my server on ashley at port 12345, and contacted it using firefox by typing this url.
`http://ashley.cs.rpi.edu:12345`
Here is what firefox sent to my server

```
GET / HTTP/1.1
Host: ashley.cs.rpi.edu:12345
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.1.5) Gecko/20091102
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

The format of the reply from the server to the client (browser) is
status line
headers (0 or more)
blank line
body

The status line consists of the keyword `HTTP/1.1` followed by a code, followed by a string indicating the result. There are only two codes that your server needs to provide

```
200 OK
404 NOT FOUND
```

When I pointed my web client to `www.rpi.edu` here is what was sent back.

```
HTTP/1.1 200 OK
Date: Fri, 26 Mar 2004 15:22:07 GMT
Server: Apache/1.3.27 (Unix)  (Red-Hat/Linux) mod_ssl/2.8.12 OpenSSL/0.9.6b mod_ldap_us
Last-Modified: Wed, 24 Mar 2004 21:03:51 GMT
ETag: "7a3a03a0-6c84-4061f7b7"
Accept-Ranges: bytes
Content-Length: 27780
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/ht
<html>
```

This waa s followed by the html file.

The Server

Your server should have several html files in its current directory. If the client sends a / as the document requested, return the names of all files in the working directory. (Hint: run `ls` or `dir`, redirecting the output to a file, then send the file, then delete the file. You can do this with the `system` command.). If the brower requests a file, return that file. If the browser requests a file which does not exist, return a 404 error code. Your server should always include the `Content-Length` and `Content-Type` headers. The argument for `Content-Length` is the number of bytes in the body. The argument for `Content-Type` can be either `text/html` if the document is an html document or `text/plain` for a plain text document (such as the table of contents). The HTTP protocol has lots of other document types as well, but your server does not need to provide them. You may provide other headers if you wish.

Your server should keep a log of all connections. This should be written to a file called `weblog`. It should contain the IP address of each client in dotted decimal form, and a verbatim transcript of everything which the client sent. The log does not need to record any information about what was returned to the client.

Your Unix server should run as a daemon, not connected to a terminal. Once a connection is accepted, the server should create a new thread to handle the reading and writing for each

connection. It should be a threaded server, that is, it should run the communication for each connection in a separate thread.

Alert: Make sure that you kill your server after you are through testing it, because it could be a security loophole.

Your server should take one argument, the port number.

The Client

The client should take one, two, or three arguments. The first argument is the url where the server is running. The second optional argument is a file name on that server, and the third is the port number. If the port number is missing, the default is 80. If both the second and third args are missing, the filename is / and the port is 80. Note that with this protocol, if you pass in a port number, you must have a second argument.

Your client should display a brief message on the terminal when a connection has been established. It should write everything which the server returns (including headers) to a file called `webscript` (emptying the file if it already exists, creating it otherwise). When the server is done, your client should display a message indicating how many bytes were written to `webscript` and terminate.

Alert: Do not use `remote` as your machine name, use the actual name `monica.cs.rpi.edu`, or whatever. You can get the name of the machine that you are on with the `hostname` command. The two programs will be graded independently. The server will be weighted 1.5 times as heavily as the client.

Grading:

Server	
Compiles (sincere attempt)	40%
Runs as a daemon	5%
Accepts connections	10%
Returns appropriate data	25%
Runs each connection in a separate thread	10%
Is well commented and well designed	10%
Client	
Compiles (sincere attempt)	40%
Parses command line correctly	25%
Establishes connections	20%
Writes data correctly	5%
Well commented, well designed	10%

The project is due at 11:59PM on Tuesday Dec 8.