SER422 Spring 2014 Lab2 (worth ½ lab grade)  
Assigned 1/29/14, due 2/10/14 at 11:55pm via individual submission to Moodle

In this lab you are given 3 tasks. The first reinforces the concepts from the previous lab, the second introduces you to CGI-style dynamic execution, while the third part introduces you to SSI-style dynamic invocation.

**Task 1: Write a simple web server (40%)**

Write a program that implements a simple web server. The web server listens on a port specified on the command line for HTTP requests and returns an appropriate response. With this simple web server, only return to requestors static files that are found in the current working directory (where the java process is started). For this simple web server we will assume you will handle GET requests, e.g.

GET /sample.html HTTP/1.1

HTTP request headers parameter will be ignored (including the HTTP/1.1 above). You are provided an incomplete WebServer program that is missing socket and thread statements. Add the code to open a server socket, listen for requests, process them, and send a response back to the client. Each request should be handled in its own thread. In the code given to you, the constructor (that you will modify, see comment TASK) will call the createResponse() method and createResponse() calls readFileInBytes(File). Call your solution WebServer1.java

**Task 2: Dynamic CGI-like execution (30%)**

As we will cover in class, a Common Gateway Interface (CGI) dynamic execution involves the web server spawning a process and capturing the output of that process and shipping it back to the browser. Modify your Task 1 solution to do the following:

* For any filename ending in .cgi on the filename of the URL, assume that cgi file is an executable
* Modify the parsing logic to parse query string parameters. That is, parse a URL like [*http://localhost:8091/myexec.cgi?arg1=foo&arg2=bar&arg3=100*](http://localhost:8091/myexec.cgi?arg1=foo&arg2=bar&arg3=100) as an executable that needs to get passed 3 name-value pairs for arg1, arg2, and arg3. For simplicity we will assume that URL decoding is not required (I will not try to trick you with special characters).
* As we will talk about with CGIs, information should be passed to the CGI executable as environment variables. Fortunately there is a convenient Java API to do this.
* Instead of reading a file and writing the response back via readFileInBytes, you will now have to capture the output of the forked executable and write it back out your OutputStream.
* Again, try to anticipate and handle errors (e.g. like the cgi program crashing mid-execution)
* Name this solution WebServer2.java

**Task 3: Dynamic SSI-like execution (30%)**

As we will also discuss in class, server-side includes, or scripts, are another style of dynamic execution. However, this style inserts dynamically-generated content in the middle of static content. For this task, you will modify your solution further to process embedded server-side directives of the form:

$$$$$ <line to fork a process to execute>

that is, if the file requested has embedded in it a line starting with 5 consecutive dollar signs, then grab the rest of the line and execute it as a program, and copy the results into that place in the output stream. For example, if you hit URL <http://localhost:8091/foo.ssi?fname=Susie&lname=Queue> and the file content is:

$$$$$ . ./reverseecho.sh $fname $lname

is learning so much in this class from

$$$$$ . ./reverseecho.sh Joe Schmoe

then the output in your browser should be (presuming reverseecho.sh does what you think):

Queue Susie

is learning so much in this class from

Schmoe Joe.

This task is related to Task 2 but you will have to do some new things:

* You still have to parse out query parameters, but now you will replace them on the command line by name, where $<arg name> is replaced by the value of the corresponding <name> from the query string.
* Again, no tricks on URL encoding. But note you are not using environment variables either so you will use a slightly modified Java call to run the program.
* You may assume the filename ends in .ssi; modify the parsing loop appropriately.
* Note you will have to parse through the .ssi file to find any $$$$$ directives.
* Again, try to anticipate errors and handle in some graceful way.
* Name this solution WebServer3.java

**Submission:**

Put WebServer1.java, WebServer2.java, and WebServer3.java in a jarfile named <asurite>.lab2.jar. Please use the jar tool and the .jar extension, I will have test scripts specifically designed to work with this format! There will be no extensions for this lab! Please complete this lab on your own!