CS 455/555 Programming Assignment #3

Due Date: Friday November 18th, 2016 - 11:59pm

Description:

The goal of this assignment is to allow you to demonstrate your knowledge of HTTP requests and HTTP responses. We'll be implementing a simple HTTP client and a simple HTTP server running version of HTTP/1.0. The goal is to get familiar with requests generated by your favorite web browser and responses from real web servers.

HTTPClient

- Your client should be named HTTPClient.java should or HTTPClient.py
- You are going to implement two methods of HTTP: **GET** and **PUT**.
- Your client take command line arguments specifying the needed parameters as follow for each operation.
- **Get**
    - Use the HTTP client to request data from a web server. Your client should be able to send requests and receive responses from any web server on the Internet. There is no need to write a server program to communicate with the client with GET command.
    - Your client should accept a single command-line argument: URL, which is the URL that it will request. There is no need to define GET command in the command line here.
        - The URL will be given in the following format: http://hostname[:port][/path] (only lowercase letters will be used)
            - hostname - the web server's hostname
            - :port - an optional port, if not present, use port 80.
            - path - the path from the web server's main directory to the requested file, if this is not present, then the path is '/'
            - Example: http://www.odu.edu/index.html
        - As with the Programs 1 and 2, if any of the arguments are incorrect, exit after printing an error message of the form ERR - arg x, where x is the argument number.
        - Treat any URL as valid as long as it starts with http://
    - Connect to the host given in the URL at the port given in the URL thru TCP connection.
        - Handle exceptions as you wish -- for example, if the host doesn't exist or the port given is not open
    - Submit a valid GET request for path/file given in the supplied URL using HTTP/1.0.
        - Your request must include the Host field (with the proper entry).

        - Your request must also include the User-agent field with the entry 'ODU-CS455/555'.
        - No other request fields are needed.
        - Remember to end with extra CR/LF
    - Print the HTTP request sent to the server.
        - Note: Print the actual string sent to the server. Do not send a request and then re-type the request for output.
    - Parse the HTTP response header received from the server and print the following information:
        - Response code (number)
        - Server type
        - If the response code is in the 200-level:
            - Last modified date
            - Number of bytes in the response data
            - Store the received file (for example: index.html from http://www.odu.edu/) in the current directory.
        - If the response code is in the 300-level:
            - The URL where the file is located
    - Print the HTTP response header.
    - In case you received a file successful and store it, open the stored file using any browser to check it.
    - Once you have this part of the client working, you should test it with the following two test cases:
        1. Use it to get a file of your choosing from a "real" web server on the Internet. For example,
           java HTTPClient http://www.cnn.com/index.html
        2. Use it to get a file from your own server program (next). For example, your server is running on atria.cs.odu.edu, port number 10003.
           java HTTPClient http:// atria.cs.odu.edu:10003/index.html
           Note that you can create your own index.html file (doesn't to have a real HTML webpage file) or store any of a downloaded index.html files such as cnn index.html file.

- **PUT**
    - Use the HTTP client to put data only to your own web server (described next).
    - Your client should accept a three command-line argument: PUT command, URL to where the files should be stored in the server, and the local path/filename of the file to transmit.
        - The URL will be given in the following format: http://hostname[:port] (only lowercase letters will be used)
            - hostname - a CS Unix machine (use either atria or sirius)
            - :port - only ports 10000-11000 are open for use.
            - Example: http://atria.cs.odu.edu:10010
        - The path/name of local file has the format: [path/]<filename>
            - path - the path of the location of the file to be transmitted at the current machine ,if this is not present, then the path is '/'

                - <filename> is the name of the file that you intend to transmit to the server. If doesn't exit, you
            - Similarly, if any of the arguments are incorrect, exit after printing an error message of the form ERR - arg x, where x is the argument number.
    - Connect to the host given in the URL at the port given in the URL thru TCP connection.
        - Handle exceptions as you wish -- for example, if the file doesn't exist, host doesn't exist, or the port given is not open
    - Submit a valid PUT request for path/file to be transferred to the webserver given in the supplied URL using HTTP/1.0.
        - Your request must include the Host field (with the proper entry).
        - Your request must also include the User-agent field with the entry 'ODU-CS455/555'.
        - No other request fields are needed.
        - Remember to end with extra CR/LF
    - Print the HTTP PUT request sent to the server.
        - Note: Print the actual string sent to the server. Do not send a request and then re-type the request for output.
    - If the server is successful in receiving and storing the file, the server sends back a "200 OK File Created" response. Otherwise, the server sends back a "606 FAILED File NOT Created" response.
    - Parse the HTTP response header received from the server and print the following information:
        - Response code (number)
        - Server type
    - Print the HTTP response header.

HTTPServer

- Your server should be named HTTPServer.java should or HTTPServer.py
- It must accept a single command-line argument: port, which is the port that it will listen on for incoming HTTP requests.
    - As with the Programs 1 and 2, if any of the arguments are incorrect, exit after printing an error message of the form ERR - arg x, where x is the argument number.
    - The only error-checking that needs to be done on the port is to ensure it is a positive integer less than 65536.
    - Remember that only ports 10000-11000 are open for use.
- The server should able to handle both HHTP commands: **GET** and **PUT**.
- For each new HTTP request, print the client's IP address, port, and the request type in the format IP:port:request
  Example: 128.82.4.98:63307:GET
- Print each line in the HTTP request.
- Construct a valid HTTP response including status line, any headers you feel are appropriate, and, of course, the requested file in the response body.
- For **GET**, If the server receives for example the "GET index.html HTTP/1.0" request,
    - Sends out "200 OK" to the client, followed by the file ./index.html.

    - If the requested file doesn't exist, the server sends out "404 Not Found" response to the client.
- For **PUT**, if the server receives the "PUT index.html HTTP/1.0" request, it will save the file as ./index.html.
    - If the received file from client is successfully created, the server sends back a "200 OK File Created" response to the client.
    - Otherwise, the server sends back a "606 FAILED File NOT Created" response.
- The HTTP server program should remain running until the user closes it with Ctrl-C. The program should close the client socket after each request has been received (even if the browser is using HTTP/1.1), but leave the welcoming/listening socket open.
- Once you have your server working, you could test it with the following two test cases:
    1. Use a real browser (e.g., firefox) to GET a file from your server
        - Note that you can download an index.html of any site and store in your server for the test.
    2. Use your HTTPClient to GET/PUT a file
- Since the CS servers only allow connections from on-campus, make sure that the browser you are testing with is on-campus (or you won't be able to reach your server program). You can run 'firefox &' from any of the CS Unix machines if you're working on this from home and need to test (make sure that you have X-Win32 or another XWindows server running at home first).

Rules:

- Your code should run on the CS department Linux machines (use either atria or sirius). Remember that only ports 10000-11000 are open for use
- The only networking classes allowed are the basic socket classes that we've used with the examples and Program 2. For example, java.net.URL is not allowed and urllib2 (Python) is not allowed.
- As with all projects, you are not permitted to work with anyone else (even students not in the class) - all of the coding and documentation must be your own.
- Your program must compile (if Java/C++) and run on the CS Unix machines.
- You must write neat code and document it well. You will lose points for sloppy programs that contain little or no comments.

Hints:

- Look back in your notes to recall how HTTP requests are formatted and terminated.
- Note that readLine() in Java strips off newline characters before returning a String.
- Use the equals() method in Java to compare two Strings.

Testing:

A large part of your program's grade will be determined by how well it handles a set of inputs. You should test your program rigorously before submitting. Because your programs will be run and tested using a script, you must format your output exactly as I have described or you will lose points.

The examples below are just examples. I will test your programs rigorously. In particular, I will test your HTTP Client on a wide range of URLs.

Example 1

java HTTPServer
Usage: java HTTPServer port

Example 2

In this example, after setting up the server, the user opened a web browser to the URL http://atria.cs.odu.edu:10003/my/url

atria> java HTTPServer 10003
128.82.4.118:33083:GET
GET /my/url HTTP/1.1
Host: atria.cs.odu.edu:10003
User-Agent: Mozilla/5.0 (X11; U; SunOS sun4u; en-US; rv:1.7) Gecko/20070606
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
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

Example 3

java HTTPClient
Usage: java HTTPClient URL or java HTTPClient PUT URL path/<filename>

Example 4

java HTTPClient http://www.cs.odu.edu/~nadeem/files/foo.txt
GET /~nadeem/files/foo.txt HTTP/1.0
Host: www.cs.odu.edu
User-agent: ODU-CS455/555

200
Apache/2.2.17 (Unix) PHP/5.3.5 mod_ssl/2.2.17 OpenSSL/0.9.8q
Thu, 19 May 2011 19:23:43 GMT
92

HTTP/1.1 200 OK

Date: Thu, 21 Mar 2013 20:53:29 GMT
Server: Apache/2.2.17 (Unix) PHP/5.3.5 mod_ssl/2.2.17 OpenSSL/0.9.8q
Last-Modified: Thu, 19 May 2011 19:23:43 GMT
ETag: "5c-4a3a5f178cdd0"
Accept-Ranges: bytes
Content-Length: 92
Connection: close
Content-Type: text/plain

Submission Materials:

- Make sure your program compiles and executes on Dept's Linux machines.
- Create a "Readme.txt" file for your program that lists how to compile and execute the program. Include your name and your UIN as the first line in the Readme.txt.
- You must name your source programs PingClient.java/PingServer.java, PingClient.py/PingServer.py, or PingClient.cpp/PingServer.cpp.
- For Java and C++, make sure that you submit all files necessary to compile your program. But, do not submit compiled files
- Zip all files of your program and name them prog_assign_3.
- Submit through Blackboard.

Submitting Assignments using Blackboard

Instructions from the official Blackboard 9.1 help pages (with screenshots)

My step-by-step instructions:

- Log in to the course Blackboard page
- Click the **Programming Assignments** link in the left sidebar
- Select the current assignment
- Under **Assignment Submission**, there is an **Attach File** section.
  Click **Browse My Computer**. Your web browser will open a window in which you can select the file to attach.
- Once you have successfully attached a file, its name should appear in the **Attached files** list.
- You may attach as many files as needed.
- When you've finished adding all of the files that are needed for the assignment, click **Submit** at the bottom of the page.
  Important: If you click Save as Draft instead of Submit, your assignment will not be turned in.
- After you've clicked **Submit**, an assignment receipt page will display.
  Click **OK**.

Important: *If you submit your assignment and later realize you have made a mistake, Blackboard will allow you to re-submit the assignment. The time of your last attempt will be counted as your submission date. Re-submissions submitted after the assignment deadline are not guaranteed to be graded. If I have started grading your assignment, I will not grade another version submitted after the due date.*