CMSC 491 Programming Assignment - HTTP

Due Date: Sunday April 24th, 2022 - 11:59pm

Description:

The goal of this assignment is to allow you to demonstrate your knowledge of socket programming for TCP connections (how to create a socket, bind it to a specific address and port), as well as HTTP requests and HTTP responses. You will develop a simple HTTP client and a simple HTTP server running version of HTTP/1.0. Your client should be able to construct multiple types of HTTP requests, send HTTP request to a webserver, receive and parse HTTP responses from the server. Your web server should accept and parse the HTTP request, get the requested file from the server's file system, create an HTTP response message consisting of the requested file preceded by header lines, and then send the response directly to the client. If the requested file is not present in the server, the server should send an HTTP "404 Not Found" message back to the client. 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, HTTPClient.py HTTPClient.c, etc.
- You are going to implement two methods of HTTP: **GET** and **PUT**.
- Your client takes command line arguments specifying the needed parameters as follows for each operation.
- **Get**
    - Use the HTTP client to request data (e.g., html base file) 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.
        - 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://egr.vcu.edu/index.html
        - 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://
    - Your client should connect to the host given in the URL at the port given in the URL thru a TCP connection.
        - Handle exceptions as you wish -- for example, if the host doesn't exist or the port given is not open
    - After connection, your client should submit a valid GET HTTP request for the path/file given in the supplied URL using HTTP/1.0. The optional header in your request must include the following:
        - The "**Host**" field with the proper corresponding entry
        - The "**Time**" field with the proper entry.
        - The "**Class-name**" field with the entry 'VCU-CMSC440-2022'.
        - The "**User-name**" field with ***your first last name*** as the entry.
        - No other request fields are needed.
        - Remember to end with extra CR/LF
    - Print the HTTP request you sent to the server.
        - Note: Print the actual string you sent to the server.
    - When you receive the HTTP response from the server, parse the HTTP response header and note the following information:
        - Response code (number)
        - Server type
        - If the response code is in the 200-level:
            - Last modified date (if included)
            - Number of bytes in the response data
            - Store the received file (for example: index.html from http://www.vcu.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 of the response you received
    - In case you successfully received the requested item/file, store the received file and then open the stored file using any browser to display 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.testingmcafeesites.com/
           java HTTPClient http://neverssl.com/
        2. Use it to get a file from your own developed server (see HTTPServer next). For example, if your server is running on 172.18.233.83 and port number 10003.
           java HTTPClient http://172.18.233.83:10003/index.html
           Note that you can create your own index.html file (doesn't have to be a real HTML webpage file) or store any of downloaded files.

- **PUT**
    - Use the HTTP client to put data only to your own developed 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://172.18.233.83:10010
        - 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 it doesn't exist, you exit after printing an error message of the form "ERR – FILE NOT FOUND"
        - 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 webserver given in the URL at the port given in the URL thru a 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. The optional header in your request must include the following:
        - The "**Host**" field with the proper entry.
        - The "**Time**" field with the proper entry.
        - The "**Class-name**" field with the entry 'VCU-CMSC440-2022'.
        - The "**User-name**" field with ***your first last name*** as the entry.
        - No other request fields are needed.
        - Remember to end with an extra CR/LF
    - Print the HTTP PUT request you sent to the server.
        - Note: Print the actual string you sent to the server.
    - If the server is successful in receiving and storing the file, the server should send 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 note the following information of the HTTP response header:
        - Response code (number)
        - Server type
    - Print the HTTP response header of the response you received.

HTTPServer

- Here, you will develop your own version of a simple web server
- Your server should be named HTTPServer.java, HTTPServer.py, HTTPServer.c, etc.
- It must accept a single command-line argument: port, which is the port that it will listen on for incoming HTTP requests.
    - 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 HTTP 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: 172.18.233.83:63307:GET
- Print each line in the HTTP request.
- Construct a valid HTTP response including the 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 content of 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 stored on your server machine, 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., chrome) to GET a file from your server
        - Note that you can download an index.html from any site and store in your server for the test.
    2. Use your HTTPClient to GET/PUT a file

VM Linux Machines:

- Open "terminal" window either on Windows, Mac, or Unix machine.
- On terminal, type "ssh 172.18.233.74 –l <eID>" then your VCU password to authenticate. Note that <eID> is your VCU user login. If success, you will be logged to machine "egr-v-cmsc440-1"
- You can also log to other machines "egr-v-cmsc440-2" and "egr-v-cmsc440-3". To do so:
    - Open a new terminal
    - Log first to machine "egr-v-cmsc440-1" as described above using the **ssh** command.
    - Then, once you logged to "egr-v-cmsc440-1", use "*ssh 10.0.0.2*" or "*ssh 10.0.0.3*" to log to "egr-v-cmsc440-2" or "egr-v-cmsc440-3" respectively.

Rules:

- ***The only programming networking classes allowed are the basic socket classes that we've used with the examples.*** For example, java.net.URL is not allowed and urllib2 in Python is not allowed.
- Your code should run on the VM Linux machines (e.g., 172.18.233.74). Note that only ports 10000-11000 are open for use
- 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 VM Linux 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/responses are formatted and terminated.
- If you are using Java, note that readLine() in Java strips off newline characters before returning a String.
- If you are using Java, 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 and running the server (e.g., on 172.18.233.74), the user opened a web browser (e.g., on 172.16.49.2) to the URL http://172.18.233.74:10003/my/url

```
egr-v-cmsc440-1> java HTTPServer 10003
172.16.49.2:33083:GET
GET /my/url HTTP/1.1
Host: 172.18.233.74:10003
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,i
mage/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.egr.vcu.edu/directory/tamer.nadeem/files/foo.txt
GET /directory/tamer.nadeem/files/foo.txt HTTP/1.0
Host: www.egr.vcu.edu
Time: Thu, 21 Feb 2019 20:53:29 GMT
Class-name: VCU-CMSC440-2022
User-name: Tamer Nadeem

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 VM Linux machines.
- Create a "Readme.txt" file for your program that lists how to compile and execute the program. Include your name and your V# as the first line in the Readme.txt.
- You must name your source programs HTTPClient.java/HTTPServer.java, HTTPClient.py/HTTPServer.py, or HTTPClient.cpp/HTTPServer.cpp.
- Submit all files necessary to compile your program.
- Zip all files of your program files in a single zip file and name it prog_assign.zip
- Submit through Canvas.

Submitting Assignments using Canvas

- Instructions from the official Canvas help pages

- https://community.canvaslms.com/t5/Student-Guide/How-do-I-submit-an-online-assignment/ta-p/503

- Step-by-step instructions:

1. Log in to the course Canvas page
2. Click the **Assignments** link on the left sidebar
3. Click the name of the intended assignment under the "**Programming Assignment**" group
4. To submit an assignment, click the **Start Assignment** button.
5. To upload a file from your computer as your assignment, select the **File Upload** tab.
6. You may upload as many files as needed.
7. When you've finished adding all files needed for the assignment, click **Submit Assignment**.

- After you have submitted your work, you will see information on the Sidebar about your submission with a link to your submission to download if necessary

Important: *If you submit your assignment and later realize you have made a mistake, you can resubmit another version of your assignment using the New Attempt button. The time of your last attempt will be counted as your submission date. You will only be able to view the details of your most recent submission on the Sidebar.*