Programming Assignment 2: Simple HTTP Client and Server

Using TCP sockets, you will write a simplified version of a HTTP client and server. The client program will use the HTTP protocol to download a file from the server using the HTTP **GET** method, and then subsequently use **conditional GET** operations to download the file only if it has been modified.

The HTTP client will perform the following functions:

1. Take in a single command line argument that specifies a web url containing the hostname and port where the server is running, as well as the name of the file to be downloaded, in the appropriate format. Example: localhost:12000/filename.html
2. Use a HTTP GET operation to download the file named in the URL
    a. Print out the contents of the file
3. Use a Conditional GET operation to download the file named in the URL again
    a. If the server indicates the file has not been modified since step 2, print output saying so (not necessary to print out file again)
    b. Otherwise, the behavior is the same as 2a)

The HTTP server will perform the following functions:

1. Open a TCP socket and listen for incoming HTTP Get and Conditional GET requests from one or more HTTP Clients
2. In the case of a HTTP Get request:
    a. Read the named file and return a HTTP GET Response, including the Last-Modified header field
3. In the case of a HTTP Conditional Get Request:
    a. If the file has not been modified since that indicated by If-Modified-Since, return the appropriate Not Modified response (return code 304)
    b. If the file has been modified, return the file contents as in step 2
4. In the case that the named file does not exist, return the appropriate "Not Found" error (return code 404)
5. The server must ignore all header fields in HTTP Requests it does not understand

Simplifying Assumptions:

- Only GET and Conditional GET requests need be supported in client and server
- Only a subset of header fields need to be supported in HTTP Requests and Responses (see Message Format section)
- However, the server must ignore all header fields it does not understand. For example, a "real' web browser will send many more header fields in GET requests than those expected to be implemented by the server. The server MUST ignore these fields and continue processing as if these fields were not part of the GET request. The server MUST NOT report an error in these cases.

Test Cases:

Enable wireshark during all the following test cases. One wireshark .pcap is fine for the test cases in this section.

1. Using your HTTP client, download the contents of a text-based html file named filename.html from your HTTP server using the appropriate URL. Example: localhost:12000/filename.html. The client must:
    a. Print out the contents of the header in the HTTP Request
    b. Print out the contents of the header in the HTTP Response
    c. Print out file contents (you can print "as is". No formatting is required)
2. File remains unmodified since Step 1. Using your HTTP Client, send a conditional GET request to your HTTP server. The client must:
    a. Print out the contents of the header in the HTTP Request
    b. Print out the contents of the header in the HTTP Response, indicating file is not modified.
3. Requested file does not exist. Using your HTTP Client, send a GET request for a filename that does not exist. The client must:
    a. Print out the contents of the header in the HTTP Request
    b. Print out the contents of the header in the HTTP Response

Test Cases for extra credit:

Using a web browser, such as Firefox (note: Safari web browser may not implement the conditional GET as expected), perform the same operations as above to test your server. Enable wireshark during all the following test cases. One wireshark .pcap is fine for the web browser test cases.

1. Enter the URL in the web browser search bar and press <return>. The web browser should print the contents of the file downloaded from your server.
2. Re-enter the URL in the web browser search bar and press <return>. The web browser should show the same web page contents as in step 1 (assuming the file has not been modified.), and the wireshark trace should show a Conditional Get and a "Not Modified" response.
3. Modify the file.
4. Re-enter the URL in the web browser search bar and press <return>. The web browser should now show the updated web contents.

Message Format:

HTTP messages are encoded in ASCII as strings in a specific format defined according to the HTTP specification[1,2,3]. References on the HTTP Message Format can be found as follows:

- Please see textbook (Section 2.2.3) or lecture slides web.pdf on Moodle for general format of HTTP Request and Response messages and some examples, specifically slides 16-17 and 20-22.
- See textbook section 2.2.5 and slide 32 for information on HTTP Conditional Get
- Also, look at the wireshark trace provided as part of the assignment (HTTP_Conditional_Get_Example.pcapng) for an example of HTTP GET and Conditional GET requests.

As part of this assignment, your HTTP Client and HTTP Server programs are only expected to handle the following header fields:

HTTP Client GET Request Message:

Your GET Request must include the following:

- Request line containing method (GET) , object (from URL) and version (HTTP1.1)
- Host: includes hostname (and port, if specified, separated by ':')
- Blank line: signifies ends of header, expressed by "\r\n\"

Example:

```
GET   /filename.html   HTTP/1.1\r\n
Host:  localhost:12000\r\n
\r\n
```

HTTP Server Response to Client GET Request (assuming file exists):

The response from the HTTP Server must include the following:

- Status line including version (HTTP1.1), status code (200), and status phrase (OK)
- Date: header field containing current date and time in the following format (must be UTC/GMT time zone):
    - Example of Date format: `Mon, 23 Jan 2017 15:55:47 GMT`
- Last-Modified: header field containing date and time file was last modified. Must follow same format as Date: above
- Content-Length: length of data in bytes
- Blank line: signifies ends of header
- Body: Contents of requested file

Example:

```
HTTP/1.1 200 OK\r\n
Date: Sun, 04 Mar 2018 21:24:58 GMT\r\n
Last-Modified: Sun, 04 Mar 2018 21:24:58 GMT\r\n
Content-Length: 75\r\n
Content-Type: text/html; charset=UTF-8\r\n
\r\n
<html><p>First Line<br />Second Line<br />Third Line<br />COMPLETE<p><html>
```

HTTP Client Conditional GET Request Message:

Your GET Request must include the following:

- Request line containing method (GET) , object (from URL) and version (HTTP1.1)
- Host: Same as in GET request
- If-Modified-Since: Echo back value of "Last-Modified" time in HTTP GET Response
- Blank line: signifies ends of header

Example:

```
GET   /filename.html   HTTP/1.1\r\n
Host:  localhost:12000\r\n
If-Modified-Since: Fri, 02 Mar 2018 21:06:02 GMT\r\n
\r\n
```

HTTP Server Conditional Response Message (Not Modified):

```
HTTP/1.1 304 Not Modified\r\n
Date: Sun, 04 Mar 2018 21:24:58 GMT\r\n
\r\n
```

HTTP Server Response when file not found:

```
HTTP/1.1 404 Not Found\r\n
Date: Sun, 04 Mar 2018 21:24:58 GMT\r\n
\r\n
```

Programming Hints:

See references [4,5,6]

Get current time in UTC/GMT time zone and convert to string in HTTP format:

```
import datetime, time
t = datetime.datetime.now(timezone.utc)
date = time.strftime("%a, %d %b %Y %H:%M:%S %Z\r\n", t)
```

Determining a file's modification time (in seconds since 1 Jan, 1970 on Unix machines)

```
import os.path
secs = os.path.getmtime(filename)
```

Convert above time to UTC /GMT (returns a time tuple):

```
import time
t = time.gmtime(secs)
```

Convert above time tuple to a string in HTTP format:

```
last_mod_time = time.strftime("%a, %d %b %Y %H:%M:%S %Z\r\n", t)
```

Convert a date/time in string format back to time tuple and seconds since 1 Jan, 1970

```
t = time.strptime(last_mod_time, "%a, %d %b %Y %H:%M:%S %Z\r\n")
secs = time.mktime(t)
```

References:

1. Kurose & Ross Textbook, Section 2.2.3
2. Hypertext Transfer Protocol (HTPP1.1): Message Syntax and Routing, https://tools.ietf.org/html/rfc7230
3. Hypertext Transfer Protocol (HTPP1.1): Semantics and Content, https://tools.ietf.org/html/rfc7231
4. Reading file modification time, testing whether file exists, and other file routines, https://docs.python.org/2/library/os.path.html
5. Time access and conversion routines, https://docs.python.org/2/library/time.html
6. Getting current date and time in Python,https://www.saltycrane.com/blog/2008/06/how-to-get-current-date-and-time-in/

Submission Guidelines:

Please submit the following individual files to Moodle by due date. **Please, NO zip files.**

- ✓ Submit the HTTP client and server source program files (please include full name, UCID (user name), section number in comments at top of source files)
- ✓ Submit screenshots in .pdf format showing the output for each test case with your HTTP client (be sure the .pdf is legible)
- ✓ Submit a wireshark .pcap file captured while running each of the 3 test cases. A single .pcap file for all 3 test cases is fine. .pcap file names should have the following form: ***"<user-name>-http-client.pcap***", where <user-name> is the NJIT login/email name.
- ✓ If you have tested your HTTP Server with a web browser, submit .pdf screenshots and corresponding wireshark trace for all test cases. Name the wireshark trace ***"<user-name>-web-browser.pcap***" to distinguish from above.
- ✓ Submit the README file using the submission format on Moodle). Include in your README which test cases are working or not working.