ACN Programming Assignment-2: WWW

The purpose of this www assignment is to learn about Web clients, Web servers, Web proxies, and the HyperText Transfer Protocol (HTTP). This is one of your major assignments in the Advanced Computer Networks course. TAs grouped students into teams of three students and they would inform you the names of other students in your group and your group-id shortly. **This www assignment is divided into four parts and it can be done in C/C++/Java/Python:**

- Part 1 is the implementation of basic web client and it is common for all groups
- Part 2 is the implementation of basic web server and it is common for all groups
- Part 3 is the implementation of basic web proxy and it is common for all groups
- Part 4 is where you enhance the features of basic web proxy with some interesting and novel extensions listed later in this document. You are free to choose which extension you want to work on. Again, inform TAs which extension-id you plan to work on. However, at maximum 5 groups are allowed to work on each of the extensions. So, if more than 5 groups choose any particular extension, TAs will intervene and assign these late comer group(s) to one of the other extensions with open slots.

PART-1: A Simple Web Client

In this part, you will develop a simple web client which will connect to the web server directly or through the intermediate web proxy using a TCP connection, send HTTP requests to the server, and display the server responses as output. You can assume that the HTTP requests sent are using the GET method. The client should take variable number of command line arguments specifying the IP address of the web proxy, host name or IP address of the web server, the port at which the proxy and web server are listening in case of indirect connection through the web proxy or simply IP address and port at which the web server is listening in case of direct connection to the web server and the path at which the requested object is stored at the web server. Run the client in one of the containers of the assigned VM.

If the requested object is the base HTML file, the web client should parse it for any reference to other objects and fetch them one after the other by establishing a non-persistent HTTP connection to the web proxy or web server.

PART-2: A Simple Web Proxy Server

In this part, you will implement and test a simple Web proxy in one of the containers of the assigned VM. This web proxy performs the first role (proxying) but not the second role (caching). The goals of the assignment are to build a properly functioning Web proxy for simple

web pages, and then to extend the functionality in certain ways in Part-4 to offer some novel web proxy features.

The most important HTTP command for your web proxy to handle is the "GET" request, which specifies the URL for an object to be retrieved. In the basic operation of your proxy, it should be able to parse, understand, and forward to the Web server a (possibly modified) version of the client request. Similarly, the proxy should be able to parse, understand, and return to the client a (possibly modified) version of the response that the Web server provided to the proxy. Your proxy should be able to handle response codes such as 200 (OK) and 404 (Not Found) correctly, notifying the client as appropriate. Reasonable handling of Conditional GET requests and 304 (Not Modified) responses is also required.

You will need at least one TCP (stream) socket for client-proxy communication, and at least one additional TCP (stream) socket for proxy-server communication. Your proxy should support multiple concurrent HTTP transactions, so you need to fork child processes (C/C++) or use threads (Java/Python) for request handling. Each child process or thread will use its own socket instances for its communications with the client and with the server.

You should be able to compile and run your web proxy in the assigned VM or even in your personal machine. You should be able to use your proxy from any Web browser (e.g., Internet Explorer, Chrome, and Mozilla Firefox), and from any machine (either on campus or at home). To test the proxy, you will have to configure your Web browser to use your specific Web proxy (e.g., look for menu selections like Edit, Preferences, Advanced, Proxies).

As you design and build your Web proxy, give careful consideration to how you will debug and test it. For example, you may want to print out information about requests and responses received and processed. Once you become confident with the basic operation of your Web proxy, you can toggle off the verbose debugging output. You can also use tools like **wireshark** to collect network packet traces for debugging. By studying the HTTP/TCP packets going to and from your proxy, you can convince yourself that it is working properly.

In your testing of the proxy, you may want to go through incremental steps similar to the following:

- Download a small ASCII text file such as IETF RFC on HTTP 1.1 (http://www.ietf.org/rfc/rfc2616.txt)
- Visit simple HTML pages such as https://people.iith.ac.in/tbr/teaching/teaching.html
- Visit a modest Web page with a few embedded objects such as https://www.iith.ac.in/
- Visit a more complicated Web page such as http://www.espncricinfo.com/

The primary test of correctness for your proxy is a simple visual test. That is, the content displayed by your Web browser should look the same regardless of whether you are using your Web proxy or retrieving content directly from the Web server. You need not be required to make your scripts to work for web servers that serve the content only through HTTPS.

PART-3: A Simple Web Server

In this part, you will develop a multi-threaded web server that handles multiple HTTP requests simultaneously from your own web client(s) (PART-1), web browsers and your own proxy server (PART-2). Using threading, first create a main thread in which your modified server listens for clients at a fixed port. When it receives a TCP connection request from a client, it will set up the TCP connection through another port and service the client request in a separate thread. There will be a separate TCP connection in a separate thread for each HTTP request/response pair. Each thread at the 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, it should send an HTTP "404 Not Found" message back to the client.

Running the web server: Put HTML files (e.g., HelloWorld.html etc from the Internet sources) in the same directory that the server is in. Run the server program in one of the containers on the VM assigned to you by the TAs. Determine the IP address of the container that is running the server (e.g., 128.238.251.26). From another host, open a browser and provide the corresponding URL. For example:

http://128.238.251.26:6789/HelloWorld.html

'HelloWorld.html' is the name of the file you placed in the server directory. Note also the use of the port number after the colon. You need to replace this port number with whatever port you have used in the server code. In the above example, we have used the port number 6789. The browser should then display the contents of HelloWorld.html. If you omit ":6789", the browser will assume port 80 and you will get the web page from the server only if your server is listening at port 80. Then try to get a file that is not present at the server. You should get a "404 Not Found" message.

Instead of using a web browser, you can use the web client developed (PART-1) by running it in one of the containers of the assigned VM for issuing HTTP requests to the web server.

PART-4: Extensions to Simple Proxy Server or web client/web server

1. **Caching at web proxy:** A caching proxy server saves a copy of the files that it retrieves from remote servers. When another request comes in for the same resource, it returns the saved (or cached) copy instead of creating a new connection to a remote server. This saves a modest amount of time and CPU if the remote server is nearby and lightly trafficked, but can create more significant savings in the case of a more distant server or a remote server that is overloaded (it can also help reduce the load on heavily trafficked servers).

   Caching introduces a few new complexities as well. First of all, a great deal of web content is dynamically generated, and as such shouldn't really be cached. Second, we

need to decide how long to keep pages around in our cache. If the timeout is set too short, we negate most of the advantages of having a caching proxy. If the timeout is set too long, the client may end up looking at pages that are outdated or irrelevant.

There are a few steps to implementing caching behavior for your web proxy:

First, alter your proxy so that you can specify a timeout value (probably in seconds) on the command line.

Second, you'll need to alter how your proxy retrieves pages. It should now check to see if a page exists in the proxy before retrieving a page from a remote server. If there is a valid cached copy of the page, that should be presented to the client instead of creating a new server connection.

Finally, you will need to somehow implement cache expiration. The timing does not need to be exact (i.e. it's okay if a page is still in your cache after the timeout has expired, but it's not okay to serve a cached page after its timeout has expired), but you want to ensure that pages that are older than the user-set timeout are not served from the cache.

2. **URL Prefetch at web proxy:** The idea behind link prefetching is simple: if a user asks for a particular page, the odds are that he or she will next request a page linked from that page. Link prefetching uses this information to attempt to speed up browsing by parsing requested pages for links, and then fetching the linked pages in the background. The pages fetched from the links are stored in the cache, ready to be served to the client when they are requested without the client having to wait around for the remote server to be contacted.

   Parsing and fetching links can take an appreciable amount of time, especially for a page with a lot of links. For this reason, use a separate thread for URL prefetching. In this separate thread, the proxy will parse a page and extract the HTTP links, request those links from the remote server, and add them to the cache.

3. **URL and Content Filtering at web proxy and web usage stats at web proxy:**
   a. In this extension, you need to supply a black list of websites to the proxy who then restricts users accessing those websites and informs users accordingly. Further, proxy does content filtering where you specify some keywords and proxy deletes or shades those keywords whenever they appear in user visited webpages.
   b. **Web usage Statistics:** Here proxy keeps track of users browsing activities, i.e., links they are visiting. You could use MAC or IP addresses to classify links to any particular user. Finally, the proxy should provide aggregate web usage statistics of the institute on a daily/weekly/monthly basis in terms of pie charts, bar graphs, etc.

4. **English to Hindi Web PageTranslation and web usage stats at web proxy:**
    a. User types URLs of English language webpages and gets displayed webpage content in Hindi language in the browser. So, Proxy has to accept URL requests from the web browser, get the response from the respective web server, translate the response message to Hindi or any other Indian language using any available open-source translation libraries, and send the translated page to the web browser. Grammatical mistakes in the translation process are acceptable.
    b. **Web usage Statistics:** Here proxy keeps track of users browsing activities, i.e., links they are visiting. You could use MAC or IP addresses to classify links to any particular user. Finally, the proxy should provide aggregate web usage statistics of the institute on a daily/weekly/monthly basis in terms of pie charts, bar graphs, etc.

5. **Speedy Web client:** Extend your simple web client program (PART-1) so that it parses base HTML file received from the web server for any objects and fetches them by establishing parallel TCP connections to the web server. Use persistent HTTP connections to save latency incurred in fetching web pages. Quantify the savings in terms of latency reduction vis-a-vis the simple web client developed in PART-1.

6. **Speedy Web Proxy:** Extend your simple web proxy (PART-2) so that it parses the base HTML file received from the web server (no cache at the proxy) for any objects and fetches them by establishing parallel TCP connections to the web server. Use persistent HTTP connections to save latency incurred in fetching web pages. Quantify the savings in terms of latency reduction vis-a-vis the simple web proxy developed in PART-2.

Documentation should include a README file explaining what is your project and how you are implementing various major modules of the project, and names of various files (along with their purpose) created for the project.

15% of marks are allotted for documentation (code comments, inline explanations, function descriptions, README file, etc) and the report. Students should aim to make their code and explanations easy to understand for TA. Refer the following page for style guides of C/C++/Python/Java and adhere to these guidelines while coding so as not to lose the marks earmarked for documentation: https://google.github.io/styleguide/ and https://peps.python.org/pep-0008/

What to Hand in?

Deliverables in a tar ball or ZIP on GC

1. Client.py (Part 1)

2. Proxy.py (Part 2)

3. Server.py (Part 3)

4. ExtendedProxy.py or ExtendedClient.py (Part 4)

5. Readme.txt (Contains detailed Instructions to run the project)

6. Report.pdf (Detailed explanations and screenshots of the modules of the project)

Marking Scheme:

Documentation across all the parts - 15% (detailed comments in code files, report and readme)

Part 1 (Building the client) - 5%

Part 2 (Building the proxy) - 15%

Part 3 (Building the server) - 10%

Part 4 (Building the extended proxy, etc) - 20%

Viva and live-test - 35%. You should directly deploy client, proxy and server on the VMs assigned to you (on each of the 3 VMs) and demonstrate their working. Plus your proxy and web server should work with any browser.

Penalty for late submission: 10% per day

ANTI-PLAGIARISM Statement <Include it in your report>

We certify that this assignment/report is our own work, based on our personal study and/or research and that we have acknowledged all material and sources used in its preparation, whether they be books, articles, packages, datasets, reports, lecture notes, and any other kind of document, electronic or personal communication. We also certify that this assignment/report has not previously been submitted for assessment/project in any other course lab, except where specific permission has been granted from all course instructors involved, or at any other time in this course, and that we have not copied in part or whole or otherwise plagiarized the work of other students and/or persons. Additionally, we acknowledge that we may have used AI tools, such as language models (e.g., ChatGPT, Bard), for assistance in generating and refining my assignment, and we have made all reasonable efforts to ensure that such usage complies with the academic integrity policies set for the course. I pledge to uphold the principles of honesty and responsibility at CSE@IITH. In addition, we understand our responsibility to report honour violations by other students if we become aware of it.

Names <Roll Nos>:
Date:
Signatures: <keep your initials here>

References:

- https://en.wikipedia.org/wiki/Proxy_server
- https://gaia.cs.umass.edu/kurose_ross/programming/Python_code_only/Web_Proxy_programming_only.pdf
- https://gaia.cs.umass.edu/kurose_ross/programming/Python_code_only/WebServer_programming_lab_only.pdf
- HTTP 1.1 RFC: http://tools.ietf.org/html/rfc2616
- https://web.mit.edu/6.033/2000/www/lab/webproxy.html
- Sockets Tutorial: http://www.linuxhowtos.org/C_C++/socket.htm
- Beej's Guide to Network Programming Using Internet Socket: https://www.beej.us/guide/bgnet/
- Programming UNIX Sockets in C - Frequently Asked Questions: http://www.softlab.ntua.gr/facilities/documentation/unix/unix-socket-faq/unix-socket-faq.html