~~CSIS0234 Computer Networks~~

~~Programming Project~~

~~Total 15 points~~

Version 1.1

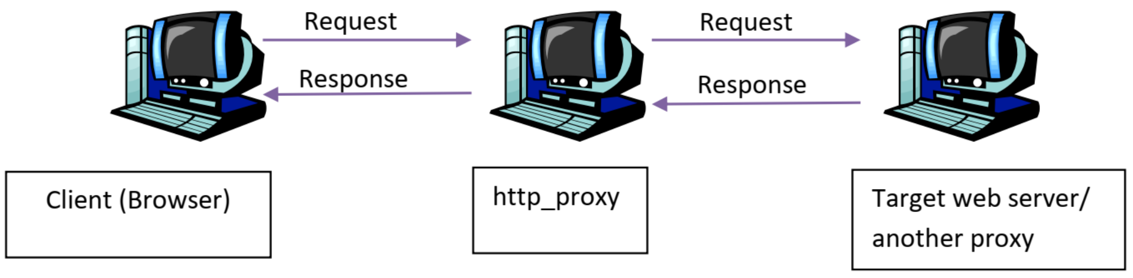
Due date: ~~5:00pm~~ April 29, 20xx

Programming Project ~~- HTTP~~ Proxy Server

Overview

For this programming project, you are going to implement a web proxy server. This proxy server gets request messages from clients (web browsers), forwards the requests to target web server or another proxy server, obtains the corresponding response messages and sends the responses back to the clients. This proxy server should be able to handle all kinds of objects, not just HTML pages, but also images and audio files. Moreover, this proxy server should support persistent connections; a timeout timer should be used to terminate idle connection from client. To reduce the complexity of this programming project, the proxy server does not support caching mechanism.

Figure 1. HTTP Proxy Server



~~Objectives~~

1. ~~Through this exercise, you should get a solid understanding of how the text-based networking protocol (such as HTTP) works as well as how to implement a standard networking protocol.~~
2. ~~This is a good opportunity to further enhance your Socket programming skill in C/C++ .~~
3. ~~A learning and assessment activity to support ILO2d and ILO4.~~

Background

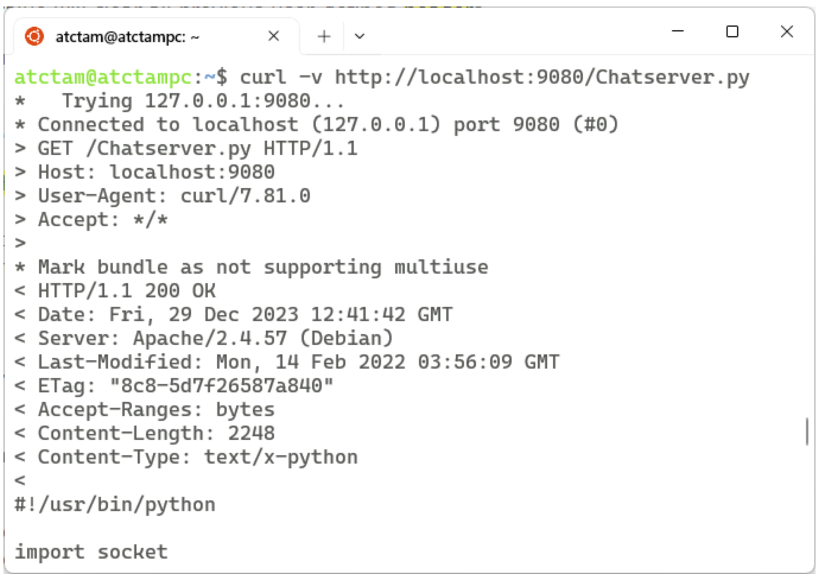
HTTP is the transfer protocol used throughout the WWW. The HTTP protocol is a request/response protocol. When a client opens a connection, it immediately sends its request for a file. The web server then responds with the file or an error message. Each interaction consists of one request and one reply. Both are MIME-like message (MIME: The Multipurpose Internet Mail Extensions).

Each request message includes a request line, a few header lines, a blank line, and the possible message body. Similar is the response message.

```
Request-message = request-line *(message-header CRLF) CRLF [ message-body ]
Response-message = status-line *(message-header CRLF) CRLF [ message-body ]
```

For example, below are the request/response messages sent by a web client (curl) to the server to download a web document:

Figure 2. Sample Web Request/Response messages



```
atctam@atctampc:~$ curl -v http://localhost:9080/Chatserver.py
*   Trying 127.0.0.1:9080...
* Connected to localhost (127.0.0.1) port 9080 (#0)
> GET /Chatserver.py HTTP/1.1
> Host: localhost:9080
> User-Agent: curl/7.81.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Date: Fri, 29 Dec 2023 12:41:42 GMT
< Server: Apache/2.4.57 (Debian)
< Last-Modified: Mon, 14 Feb 2022 03:56:09 GMT
< ETag: "8c8-5d7f26587a840"
< Accept-Ranges: bytes
< Content-Length: 2248
< Content-Type: text/x-python
<
#!/usr/bin/python

import socket
```

Non-Persistent Connection and Persistent Connection

In HTTP 1.0, after the TCP connection was established, a single request was sent over and a single response was sent back. Then the TCP connection was released. Establishing a separate connection for each file request incurs expensive overhead. In view of this deficiency, HTTP 1.1 was designed to support persistent connections. In HTTP 1.1, it is possible to establish a TCP connection, sends a request and gets a response, and then sends additional requests and gets additional responses. Our proxy server should support persistent connections between web clients and itself as well as between itself and target web/proxy servers.

If an HTTP 1.1 client sends multiple requests through a single connection, the server should send responses back in the same order as the requests – this is all it takes for a server to support persistent connections. The server should close a connection after the connection has been idle for long enough duration (not defined in the specification; usually set to 10 – 15 seconds).

Requirements

You are going to write a simple concurrent HTTP proxy server (http_proxy) using C socket, which works as follows:

1. The http_proxy server listens for connection requests from clients.
2. When there is a new TCP connection request, the http_proxy server creates a new process** to handle the new connection.
3. The new process sends any HTTP requests to the (origin/proxy) server and reads the server's reply.
4. Then it sends the responses back to the requesting client.

** Instead of using multiple processes, you can implement a multiple threads proxy server; the http_proxy server creates one new thread to handle each connection.

Relay client requests and server responses

1. Directly connect to target web (origin) server:
    - Extract the "Host: " header line from the client request message to get the target server name
    - Connect and send the request to target server on port 80
    - Get the response and send back to client
2. When configured to connect to another proxy server
    - Connect and send the request to the specified proxy server and port
    - Get the response and send back to client

Persistent connections

With persistent connection, the proxy server leaves the connection open for serving more requests after handling a request. The server closes the connection as per the client instructs or if the timeout of idle connection has been reached. In order to let the client knows when is the end of a response message, all response messages on the persistent connection must have a self-defined message length using the Content-Length header field [except for the cases of dynamically generated web contents]. As our proxy server, in principle, is just a relay server, our proxy server does not need to handle that as the Content-Length headers should be generated by the origin servers.

To terminate an idle client connection, the proxy server can simply close the TCP connection and the client browser should detect this situation automatically. Similarly, a client browser can terminate an idle persistent connection by closing the TCP connection; the proxy server should detect this situation and should handle this correctly.

Support request types

For this programming project, HTTP requests generated by all test cases are GET request types. However, of unknown reasons, some browsers may generate POST requests under some situations (e.g., Chrome browser).

Caching proxy server

You are not required to implement the caching mechanism in this project. That means all requests received from the clients are forwarded to the target origin/proxy server without the needs to search the cache memory as well as manage the cache memory.

How to start the HTTP proxy process

The http_proxy should be invoked as follow:

```
./http_proxy myport [proxy_server:port]
```

The 1st parameter – myport is the port number that the http_proxy is listening on. The 2nd parameter – proxy_server:port is an optional argument. When proxy_server:port is specified, all the HTTP requests (responses) should be forwarded to (received from) proxy_server. The port in proxy_server:port is the port number that the proxy_server is listening on.

Output messages

To aid your programming and debugging, please print some log information to the screen. You are required to generate some output with major events:

- When receiving a new connection – identify the origin – display some useful information.
- When establishing a new connection – display the target server information
- When receiving a new request – display some information, e.g,, the request-line.
- When receiving a response message – display some information, e.g., message size
- When detecting the termination of persistent connections or server connection.

Sample runs:

```
atctam@LinuxPC: > ./http_proxy 54321
http_proxy listening on port 54321
http_proxy got a NEW connection (1) from 147.8.175.181
Received Request from user: 349 bytes
Set up a connection to server i.cs.hku.hk
Requesting server:
GET http://i.cs.hku.hk/course/c0324/test/test-1.htm HTTP/1.1
Host: i.cs.hku.hk
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:14.0)
Gecko/20100101 Firefox/14.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Proxy-Connection: keep-alive
...

Received Response from Server: 263 bytes
Received Response from Server: 357 bytes
Received Response from Server: 0 bytes
Connection close by server : i.cs.hku.hk

Timeout - Persistent Connection (1) Closed
```

OR

```
atctam@LinuxPC: > ./http_proxy 54321 proxy.cs.hku.hk:8282
http_proxy listening on port 54321
http_proxy got a NEW connection (1) from 147.8.175.181
Received Request from user: 349 bytes
Set up a connection to server proxy.cs.hku.hk
Requesting server:
GET http://i.cs.hku.hk/course/c0324/test/test-1.htm HTTP/1.1
Host: i.cs.hku.hk
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:14.0)
Gecko/20100101 Firefox/14.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Proxy-Connection: keep-alive
...

Received Response from Server: 761 bytes

Timeout - Persistent Connection (1) Closed
```

Test cases

We will use (1) Firefox and (2) Chrome to connect to the following web sites for testing of your proxy program.

```
(1) http://i.cs.hku.hk/course/c0324/test/test-1.htm
(2) http://i.cs.hku.hk/~atctam/view/SimpleWebpage.html
(3) http://i.cs.hku.hk/course/c0324/test/test-2.htm
(4) http://www.simpleweb.org/
```

Submission

A single file named http_proxy.c or http_proxy.cpp with in-line documentation. You should hand-in the program via the Moodle course Website.

Computer Platform to Use

For this assignment, you are expected to work on any UNIX-like workstation, but you need to test your program on the latest distribution of Ubuntu before submission. You can use C/C++ to implement the program, and it should be successfully compiled with g++ (or gcc).

Format for the documentation

1. At the head of the submitted file, state clearly the
   - Student name:
   - Student No. :
   - Date and version:
   - Development platform:
   - Development language:
   - Compilation:
2. Inline comments (try to be detailed so that your code could be understood by others easily)

Grading Policy

Documentation (1 point)	High Quality [0.6/1] • Include necessary documentation to clearly indicate the logic of the program • Include necessary output messages for debugging and tracing of the execution flow
	Standard Quality [0.4/1] • Include required program and student's info at the beginning of the program • Include minimal inline comments
Connect to target origin Web server [4 points]	• By setting Firefox or Chrome to use http_proxy as the proxy server, after removing all cache contents in browser's cache, the browser can successfully download the documents/objects related to the four test cases. [2/4]
	• Without clearing browser's cache, the browser can successfully display the four test cases by just using the reload button to initiate the download requests. [2/4]
Connect to another proxy server [2.5 points]	• By setting Firefox or Chrome to use http_proxy as the proxy server, which in turn, is requesting the department proxy server "proxy.cs.hku.hk:8282" to download all Web documents/objects. After removing all cache contents in browser's cache, the browser can successfully download the documents/objects related to the test cases (2), (3), & (4). [1.5/2.5]
	• Without clearing browser's cache, the browser can successfully display the the test cases (2), (3), & (4) by just using the reload button to initiate the download requests. [1/2.5]
Persistent connection [1.5 points]	• The http_proxy server supports persistent connection with the browser for downloading test cases (2), (3), & (4). [1/1.5]
	• The http_proxy server can successfully terminate idle persistent connection. [0.5/1.5]

Readings

1. Chapter 2.2 of Computer Networking – A Top-Down Approach Featuring the Internet, 6th edition by J. Kurose et. al
2. Lecture 9
3. Workshop 5 - Using WireShark to examine and understand SMTP and HTTP

Plagiarism

Plagiarism is a very serious offence. Students should understand what constitutes plagiarism, the consequences of committing an offence of plagiarism, and how to avoid it. **Please note that we may request you to explain to us how your program works as well as we may make use of software tools to detect software plagiarism.**