CSE422 Computer Networks Fall 2020
Project 2: Web Proxy Server
Due: 23:59 Friday, November 6, 2020

1 Goals

Apply your knowledge of socket programming in order to implement a real-life application and gain some basic understanding of HTTP.

2 Overview

In this lab, you will implement a simple *proxy server* for HTTP that forwards requests from *clients* to *end servers* and returns responses from *end servers* to the *clients*.

This lab is worth for 150 points. This lab is due no later than 23:59 (11:59 PM) on Friday, November 6, 2020. No late submission will be accepted.

2.1 The HyperText Transfer Protocol, HTTP

The **HyperText Transfer Protocol (HTTP)** is the World Wide Web's application-layer protocol. HTTP operates by having a *client* (usually the browser) initiate a connection to a *server*, send some request, and then read the server's response. HTTP defines the structure of these messages and how the clients and servers exchange messages.

A *web object* is simply a file, such as an HTML file, a JPEG image, or a video clip. A *web page* usually consists of one HTML file with several referenced objects. A page or an object is addressed by a single *Uniform Resource Locator* (*URL*). When one wants to access a HTML page, the web browser initiates a request to the server and asks for the HTML file. If the request is successful, the server replies to the web browser with a response that contains the HTML file. The web browser examines the HTML file, identifies the referenced objects, and for each referenced object, initiates a request to retrieve the object.

An example of an HTTP request/response is shown in Figure 1. Both the request and response consist of a message header followed by a message body. The header is composed of several lines, separated by a carriage return line feed (CRLF, "`\r\n`"). For each message, the first line of the header indicates the type of the message. Zero or more header lines follow the first line; these lines specify additional information about this message. The end of header is marked by an empty line. The message body may contain text, binary data, or even nothing at all.

Request

```
GET /1MB.zip HTTP/1.1\r\n
Connection: close\r\n
Host: speedtest.tele2.net\r\n
If-Modified-Since: 0\r\n
\r\n
```

Response

```
HTTP/1.1 200 OK\r\n
Accept-Ranges: bytes\r\n
Connection: close\r\n
Content-Length: 1048576\r\n
Content-Type: application/zip\r\n
Date: Fri, 16 Oct 2020 07:54:51 GMT\r\n
ETag: "5c90e255-100000"\r\n
Last-Modified: Tue, 19 Mar 2019 12:36:37 GMT\r\n
Server: nginx\r\n
\r\n
[body]
```


Figure 1: Example HTTP request and response message. In the request, the client asks for `/1MB.zip` from the web server `speedtest.tele2.net` over HTTP/1.1. In the server's response, the server informs the client that the request was successful with the status code `200` and several additional header lines that carry information about this response. Note that each line is ended by a CRLF.

There are eight request methods that indicate what the client wants the server to do. In this lab, we consider only the `GET` method, which is used to request objects from the server. The `GET` request must include the `path` to the object the client wishes to download and the HTTP version. In the above example, the path is `/1MB.zip` and the HTTP version is `HTTP/1.1`. Some request methods (such as `POST`) that transmit data to the server include the data in a message body.i However, the `GET` method does not have a message body.

In its response, the server indicates the HTTP version, status code and status description. The status code and status description indicate whether the request was successful and, if not, why the request failed. Common status codes and status description include

- `200 OK`: Request succeeded
- `403 Forbidden`: The request failed because access to the resource is not allowed.
- `404 Not Found`: The request is failed because the referenced object could not be found.

- `500 Internal Server Error`: There is something wrong at the server side.

For a more detailed information about HTTP, please see:

- Computer Networking: A Top-Down Approach, Sixth Edition, page 97 - 105.
- Wikipedia Entry
- RFC 2612: HTTP/1.1 and RFC 1945: HTTP/1.0

2.2 Proxy Server [Wikipedia Entry]

As shown in Figure 2, a *proxy server* is a program that acts as a middleman between a *client* and an *end server*. Instead of requesting an object from the server directly, the client sends the request to the proxy, which forwards the request to the server. When the server replies to the proxy, the proxy returns the response to the requesting client.

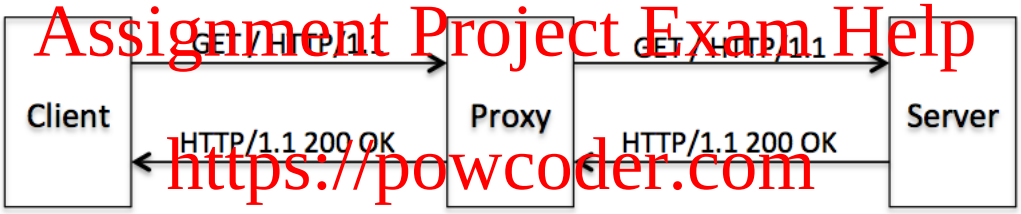

Figure 2: The client sends request to the proxy and the proxy forwards the request to the server. The proxy awaits the server's response and returns it to the client.

Proxies are used for many purposes. Sometimes proxies are used as firewalls, such that the proxy is the only way for a client behind a firewall to contact any server outside. Proxies are also used as anonymizers. By removing or modifying a request header, a proxy can make the client anonymous to the server. By examining the request header, a proxy can filter and block requests, for example, blocking any request where the URL contains the keyword "facebook".

An important application of proxies is to cache web objects by storing a copy when the first request is made, and then serving that copy in response to future requests rather than going to the server. For large business or ISPs, caching frequently requested object can reduce the communication cost.

3 A simple HTTP client

In order to understand the HTTP message exchange and to focus on the implementation of proxy, a simple command line HTTP client is provided to you along with the skeleton code

on Mimir. Several classes are provided along with this simple HTTP client, which can help you construct the proxy.

```
Usage: ./client [options]
The following options are available:
    -s host URL
    -p proxy URL
    -i client_id
    -h display help message
```

The URL to the desired web object must be specified by the argument `-s`. The arguments `-p` and `-i` are optional (the latter is only used to tag the proxy output for the test cases / debugging).

Example invocation without proxy:
`./client -s http://www.google.com/index.html`

This invocation does exactly the same thing as in Figure 1 and stores a copy of `index.html` in `Download` folder under current directory.

Example invocation with proxy running on `arctic.cse.msu.edu` at port `20987`:
`./client -s http://www.google.com/index.html -p arctic.cse.msu.edu:20987`

If the proxy port is not specified to the client, the client assumed it to be 8080. However, in this lab, the proxy port must be assigned by the operating system. If there is no proxy running on the address specified, the connection fails and the program is terminated. If there is a proxy running at the address specified, the download should be successful and store a local copy in subdirectory `Download`. Each invocation of this client program initiates a request and handles the response for that request.

3.1 Initiating a Request

To initiate a request, the HTTP client has to connect to the server, construct a request message and send the message to the server (or proxy, depending on how the client is invoked, in the following of this section, *server* means either end server or proxy server.). The `TCP_Socket` class provides the functionality for the communications and handles details of setting up the socket.

Processing HTTP messages requires a lot of string parsing and formatting. A *URL* class is provided to help you parse the given URL and store it as an object. The method `URL::parse` takes a string as the argument and returns the pointer to the parsed URL object if the string is a valid URL, or NULL otherwise.

An *HTTP_Request* class is provided to handle the construction of new HTTP requests, for sending/receiving of requests, and for parsing an incoming HTTP request (which is not needed by the client, but is needed by the proxy.)

We summarize the initiation and sending of requests as follows:

- Parse the server URL string by invoking `URL::parse`.
- Create a `TCP_Socket` object: `TCP_Socket client_sock`. The method `client_sock.Connect` connects to the corresponding server.
- Create an `HTTP_Request` object `request` by invoking `HTTP_Request::create_GET_request`.
- Configure this HTTP Request.
- The method `HTTP_Request::send(client_sock)` sends the request to the server.

3.2 Handle the Response

Next, the client expects the HTTP response from the server. We also provide an `HTTP_Response` class for sending/receiving of requests, for parsing of incoming HTTP response, and to handle the creation of new HTTP requests (which is not needed by the client, but is needed by the proxy).

Handling the response is a two-step procedure, first handling the response header and then the response body. Two steps are needed because the length of the message body varies, and the client does not know in advance *when to stop receiving incoming data*. When a process invokes the `read/recv` system call, the system call returns the number of bytes received or the process is blocked and waits for incoming bytes. Without knowing the length of the message body, the client does not know when to stop calling `read/recv`. Therefore, a client has to receive the header first and examine the header fields to determine the number of bytes to expect in the body.

There are several transfer encoding mechanisms in HTTP and in this lab, we only care about two transfer encoding mechanisms, identity encoding and chunked transfer encoding. The message header comprises several lines, each ending with a CRLF, and the end of the header is marked by a blank line. The client keeps reading one line of data until two consecutive CRLFs are found in the buffer; the rest of incoming data belong to response body. The `read_header` method is provided in both `HTTP_Request` and `HTTP_Response`. If you wish to handle the data yourself, the method `read_line` is provided in `TCP_Socket`.

3.2.1 Identity Encoding

Identity encoding is the default transfer encoding mechanism defined in HTTP. The `Transfer-Encoding` line is not present in the header. The `Content-Length` line specifies the length of the response body explicitly. The client simply receives this specified amount of data and stores it as the response body.

3.2.2 Chunked Transfer Encoding

Chunked transfer encoding, defined in HTTP version 1.1, enables a web object to be sent from the server as a series of "chunks." The advantage of chunked transfer encoding is that the server does not need to know the length of the response body before starting to send parts of it to the client.

Each chunk is separated by a CRLF and begins with a hexadecimal chunk size followed by an extra CRLF. After reading the header, if this response is in chunked transfer encoding, the client reads one more line, which indicates the length of the first chunk. The client receives data until the chunk is completely downloaded. The client reads two more lines, the first line is the blank line between chunks and the second line is the size of next chunk. The client continues this process until it receives a zero chunk size, which indicates the end of the transfer.

3.3 Response Summary

The process of receiving the response is summarized as follows:

- Create an `HTTP_response` object.
- Receive the response header by invoking `HTTP_Response::receive_header` and parse it.
- Receive the response body. You can check if this response is chunked by invoking `HTTP_Response::is_chunked`.
- Store the received data as a file.

4 Specification

In this lab, you are required to implement a proxy that forwards `GET` requests from a client to the server and returns the responses from the server back to the client. The port for

listening to incoming request is assigned by the operating system. This lab only addresses non-persistent connections. The proxy is expected to be able to handle multiple requests by forking an instance for each request. Both default encoding and chunked transfer encoding must be handled by this proxy.

To help with debugging, you are required to add/modify a field in the response header, saying that this response is returned by your proxy. Specifically, you are required to add (or modify) the field `Server` with a string, such as your MSU NetID, showing the header has been modified. The method `HTTP_Response::set_header_field` is able to do this.

The proxy is expected to respond with error messages to bad requests. For a request that tries to download an object from a host that does not exist, the proxy returns a `404 Not Found` response. As long as the end server exists, the case when the requested web object that does not exist is handled by the end server. The proxy simply forwards the request and returns the response.

The proxy is expected to perform simple filtering. Specifically, the proxy rejects any request to any host that contains the keyword "facebook," but allows request to "path" that contains the keyword "facebook." The proxy returns a `403 Forbidden` response for the former request. For example, the proxy rejects the requests to `www.facebook.com` and forwards requests to

The proxy is required to handle both default transfer encoding and chunked transfer encoding. For default transfer encoding, the proxy is required to display (print to the console) the content length. For chunked transfer encoding, the proxy is required to display (print to the console) the length of all chunks.

4.1 The Work Flow of the Proxy

We outline the work flow of this proxy in this section. The proxy starts running and waits for incoming connections. For each connection, the proxy has to do the following:

- Get the request string from the client, check if the request is valid by parsing it (the method `HTTP_Request::receive` receives the request and parse it at once).
- From the parsed `HTTP_request` object, obtain the server address by invoking `HTTP_Request::get_host`. Also check the validity of this server address by invoking `URL::parse`. If this server is invalid (returned NULL), respond this request by a `404 Not Found`. If we are blocking this server, respond this request by a `403 forbidden`.
- Forward this request to the server by invoking `HTTP_Request::send`.
- Receive the response header and modify `Server` header field (`HTTP_Response::set_header_field`).

- Receive the response body. Handle both default and chunked encoding transfer.
- Return the modified response to the client.

The proxy returns a `404 Not Found` response to the client as long as it cannot reach the server specified in the request, such as

- The parsed server URL is NULL, which means the URL is not valid.
- Fail to connect to the server.
- Unable to resolve the server URL.
- Server does not exist.
- ... etc.

When the servers respond `403` or `404` messages, the content/body of the response is usually a webpage showing related information. However, the provided `HTTP_Response` class constructor only constructs responses with header only. In this lab, your proxy servers do not need to provide response content/body. It is fine as long as the header is correct.

A skeleton file is provided to you along with the simple client on Mimir.

5 Deliverables

You will submit your lab using Mimir.

Please submit all files in your project directory. If you start your lab with the skeleton code, submit all files, even for files that are not modified.

This lab is due no later than 23:59 (11:59 PM) on Friday, November 6, 2020. No late submission will be accepted.

The compilation must be done using a makefile (one is provided in the skeleton code). The code should compile and link on Mimir (the Mimir IDE is extremely similar to that of the test cases. You will not be awarded any point if your submission does not compile using the command "make".

A README file is required. Please fill out your name, an approximation of the time it took you (prefereably a measure of the number of hours you spent working on it and how many days they were spread between, e.g. '4 hrs over 10 days' or '6 hrs over 1 day').

6 Grading

You will not be awarded any point if your submission does not compile. Furthermore, you will be awarded points based on the results of the Mimir test cases. Make sure to test your code early to make sure it adhears to the test case formatting – further clarification can be given if one asks while there is a reasonable amount of time left before the project is due.

```
General requirements: 10 points
_____  5 pts Coding standard, comments ... etc
_____  1 pts README file
_____  4 pts Descriptive messages/Reasonable output.
             Display the headers and (content length or chunk sizes)

Proxy basic functions: 110 points
_____ 10 pts Closes properly on SIGINT
_____ 10 pts Forward the request
_____ 30 pts Return the default encoding transfer responses.
_____ 30 pts Return the chunked encoding transfer responses.
_____ 15 pts Handle multiple requests (Multiprocessing)
_____ 15 pts Add/Modify the Server header field

Proxy handling special cases: 30 points
_____ 10 pts Respond with 404 not found to request for non-exist URL
_____ 15 pts Filter out requests to any "host" that contains "facebook"
             and return a 403 forbidden.
_____  5 pts Allow requests to a "path" contains "facebook"
```


7 Notes

- The given skeleton code has comments formatted to be parsed by doxygen and includes the html output. Feel free to inspect these docs by opening up html/index/html in your favorite browser – this may be easier than reading the code if you just want an overview.
- Please BE CAREFUL when running / testing your code to avoid fork-bombs. If done on the Mimir IDEs, you apparently get locked out for 20 minutes, but make sure this does not happen on the CSE servers. Note that a child process exists only to serve the one client interaction, it should NEVER get a chance to spawn other processes. It is a good practice to make this clear in the accept loop by putting all of the child code in a few separate functions. For example, consider the following code snippet.

```
// loop to handle each connection
while (1) {
    ...
    pid = fork();

    if (pid == 0) {
        run_child_code();

        // get out of the loop
        exit(1);  // or break or return
    }
    ...
}
```

- This lab only uses non-persistent connections. The constructor of `HTTP_Request` class sets the `Connection` header to `close` for you already.
- Please spend some time tracing the code in the provided classes. One should be able to build the entire proxy using those classes. Tracing the client code would be a good start.
- Obviously, the default transfer encoding is easier to implement than chunked transfer encoding. For your convenience, the requests to the following URLs are guaranteed to reply with default encoding responses. In fact, responses to most web objects that are not HTMLs should be default transfer encoding.
  - `https://speedtest.tele2.net/1MB.zip`
  - `https://www.google.com/images/srpr/logo3w.png`
  - `https://releases.ubuntu.com/20.04/ubuntu-20.04.1-desktop-amd64.iso` It might take up to a minute. It is a big file (2.6 GB).
- This lab does not require the proxy to work with real browsers. However, if the functions required in this lab are implemented correctly, this proxy should be able to work with real browsers and should be able to display most web pages.

Please feel free to email TA Jonathon Fleck, at fleckjo1@msu.edu, for questions or clarifications. Additional notes and FAQ will be posted on Piazza as well.

7.1 Office Hours

Office hours are planned for the following times for this project. All times are given in EST.

Week	Monday	Tuesday	Wednesday	Thursday	Friday
10/19 - 10/23	-	2-3 pm	-	Mid-Term	1-2 pm
10/26 - 10/30	2-3 pm	2-3 pm	-	-	1-2 pm
11/03 - 11/06	-	2-3 pm	2-3 pm	-	1-3 pm

Here is the Zoom [link]. The password is "422ta".

If you cannot meet during these times or if you are having trouble being helped during office hours, please email me to let me know and to set up additional zoom meetings.

8 Examples

The following examples show output from the client or proxy for various scenarios.

8.1 Client without using proxy


```
>./proj2_client -s http://www.google.com
Request sent...
============================================================
GET / HTTP/1.1
ClientID: https://www.google.com
Connection: close
Host: www.google.com
If-Modified-Since: 0
============================================================

Response header received
============================================================
HTTP/1.1 200 OK
Accept-Ranges: none
Cache-Control: private, max-age=0
Connection: close
Content-Type: text/html; charset=ISO-8859-1
Date: Sat, 17 Oct 2020 06:05:01 GMT
Expires: -1
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Server: gws
Set-Cookie: NID=204=cpO1pblaLfgyFEUrkUGr6HwzWDcbP7NpDOiScU-kjf67n7...
Transfer-Encoding: chunked
Vary: Accept-Encoding
```

```
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 0
============================================================

Downloading rest of the file ...
Chunked encoding transfer
chunk length: 17344
chunk length: 1032
chunk length: 30257
chunk length: 0

Download complete (48633 bytes written)
```

8.2 Client with proxy

```
>./proj2_proxy
Proxy running at 49365...
[?] New connection established.
[?] New proxy child process started.
[?] Getting request from client...
[?]
[0] Received request:
[0] ============================================================
[0] GET / HTTP/1.1
[0] Connection: close
[0] Host: www.google.com
[0] If-Modified-Since: 0
[0] ============================================================
[0]
[0] Checking request...
[0] Done. The request is valid.
[0]
[0] Forwarding request to server...
[0] Response header received.
[0]
[0] Receiving response body...
[0] Chunked encoding transfer
[0] chunk length: 18942
[0] chunk length: 416
[0] chunk length: 29239
[0] chunk length: 0
```

```
[0]
[0] Returning response to client ...
[0] ============================================================
[0] HTTP/1.1 200 OK
[0] Accept-Ranges: none
[0] Cache-Control: private, max-age=0
[0] Connection: close
[0] Content-Type: text/html; charset=ISO-8859-1
[0] Date: Sat, 17 Oct 2020 06:11:40 GMT
[0] Expires: -1
[0] P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
[0] Server: MSU/CSE422/FS20
[0] Set-Cookie: NID=204=tQcOKAF1PoIb7-BTDCPnb9ajosT-ioh3pZEaWtKek6A...
[0] Transfer-Encoding: chunked
[0] Vary: Accept-Encoding
[0] X-Frame-Options: SAMEORIGIN
[0] X-XSS-Protection: 0
[0] ============================================================
[0]
[0] 49279 bytes sent
[0] Connection served. Proxy child process terminating.
Child process terminated.
```



```
>./proj2_client -s https://www.google.com -p localhost:40365 -i 0
Request sent...
============================================================
GET / HTTP/1.1
ClientID: https://www.google.com
Connection: close
Host: www.google.com
If-Modified-Since: 0
============================================================

Response header received
============================================================
HTTP/1.1 200 OK
Accept-Ranges: none
Cache-Control: private, max-age=0
Connection: close
Content-Type: text/html; charset=ISO-8859-1
Date: Sat, 17 Oct 2020 06:11:40 GMT
```

```
Expires: -1
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Server: MSU/CSE422/FS20
Set-Cookie: NID=204=tQcOKAF1PoIb7-BTDCPnb9ajosT-ioh3pZEaWtKek6A...
Transfer-Encoding: chunked
Vary: Accept-Encoding
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 0
============================================================

Downloading rest of the file ...
Chunked encoding transfer
chunk length: 18942
chunk length: 416
chunk length: 29239
chunk length: 0

Download complete (48597 bytes written)
```


8.3 Request URLs that are blocked

```
>./proj2_proxy
Proxy running at 48647...
[?] New connection established.
[?] New proxy child process started.
[?] Getting request from client...
[?]
[0] Received request:
[0] ============================================================
[0] GET / HTTP/1.1
[0] Connection: close
[0] Host: www.facebook.com
[0] If-Modified-Since: 0
[0] ============================================================
[0]
[0] Checking request...
[0] Request to URL contains 'facebook'
[0]
[0] Returning 403 to client ...
[0] ============================================================
[0] HTTP/1.1 403 Forbidden
```

```
[0] Connection: close
[0] Content-Length: -1
[0] Content-Type: text/html
[0] Date: Sat, 17 Oct 2020 06:38:53 GMT
[0] Server: MSU/CSE422/FS20
[0] ============================================================
Child process terminated.


>./proj2_client -s http://www.facebook.com -p localhost:49467 -i 0
Request sent...
============================================================
GET / HTTP/1.1
ClientID: 0
Connection: close
Host: www.facebook.com
If-Modified-Since: 0
============================================================

Response header received
============================================================
HTTP/1.1 403 Forbidden
Connection: close
Content-Length: -1
Content-Type: text/html
Date: Sat, 17 Oct 2020 06:38:53 GMT
Server: MSU/CSE422/FS20
============================================================

Downloading rest of the file ...
Default encoding transfer
Content-length: -1

Download complete (0 bytes written)
403 Forbidden
```


8.4 Requesting an URL that does not exist

```
>./proj2_proxy
Proxy running at 41323...
[?] New connection established.
[?] New proxy child process started.
```

```
[?] Getting request from client...
[?]
[0] Received request:
[0] ============================================================
[0] GET / HTTP/1.1
[0] Connection: close
[0] Host: www.cse.msu123
[0] If-Modified-Since: 0
[0] ============================================================
[0]
[0] Checking request...
[0] Done. The request is valid.
[0]
[0] Forwarding request to server...
[0] TCP_Socket Exception: could not resolve hostname
[0]
[0] Returning 404 to client...
[0] ============================================================
[0] HTTP/1.1 404 Not Found
[0] Connection: close
[0] Content-Length: -1
[0] Content-Type: text/html
[0] Date: Sat, 17 Oct 2020 06:13:04 GMT
[0] Server: MSU/CSE422/FS20
[0] ============================================================
Child process terminated.


>./client -s http://www.cse.msu123 -p localhost:40753 -i 0
Request sent...
============================================================
GET / HTTP/1.1
ClientID: 0
Connection: close
Host: www.cse.msu123
If-Modified-Since: 0
============================================================

Response header received
============================================================
HTTP/1.1 404 Not Found
Connection: close
```

```
Content-Length: -1
Content-Type: text/html
Date: Sat, 17 Oct 2020 06:13:04 GMT
Server: MSU/CSE422/FS20
============================================================

Downloading rest of the file ...
Default encoding transfer
Content-length: -1
Download complete (0 bytes written)
404 Not Found
```