# Assignment 2 - A Multi-threaded Web Proxy

CSCI4430 - Data Communication and Computer Networks

Version 1.0: 2013 February 27.

**Abstract**

In this assignment, we will implement a multi-threaded web proxy, *MYPROXY*. MYPROXY implements the basic functions of a real-life web proxy (e.g., our department proxy: `proxy.cse.cuhk.edu.hk:8000`).

## 1 Overview

Figure 1 gives an overview of the architecture of MYPROXY. Different browsers (which could be from different machines) communicate with different web servers via MYPROXY. The goal of MYPROXY is to cache web objects received from web servers, so that if a browser later requests a cached web object, the proxy can simply return the web object to the browser without involving the web server.

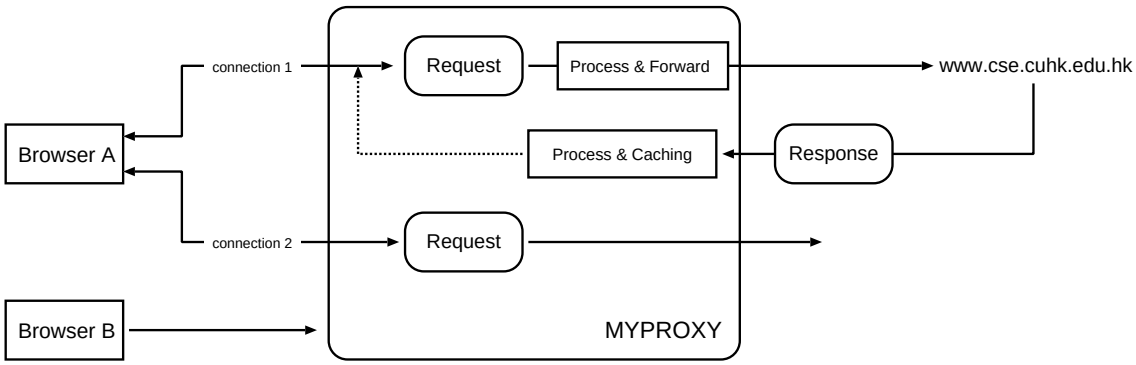


Figure 1: The MYPROXY architecture.

The high-level overview of the workflow is as follows.

1. A browser sends an HTTP request for a web object to MYPROXY.

2. If MYPROXY has cached the web object, then it simply returns the web object to the browser.

3. Otherwise, it forwards the HTTP request to the target web server. The target web server returns an HTTP response that contains the web object.

4. Last, MYPROXY caches the web object in its local storage and forwards the HTTP response (with the web object) to the browser.

We now discuss the implementation details as follows. First of all, you are required to deploy MYPROXY with real web browsers. Therefore, MYPROXY will receive genuine HTTP messages. We consider how MYPROXY handles the two types of messages: (i) HTTP request and (ii) HTTP response. Both the requests and the responses may come to MYPROXY with different favors:

- **Persistent connections**. Most browsers are adopting persistent connections. For the implementation of MYPROXY, you are therefore required to implement a **connection mapping functionality**.

- **Pipelining**. Most browsers are sending parallel, persistent connections while they are loading a page. Tabbed browsing is also another source of parallel connections. Hence, for the sake of performance, **POSIX thread** will be used in order to cope with the parallel requests coming from a browser.

- **Caching**. Most browsers have their local cache storage, and corresponding header fields will be embedded into the request messages. Later in the text, we will cover caching-related headers and values.

## 1.1 HTTP request from web browsers

Table 1 shows an example HTTP request message, and you may find this example useful while you are reading the text below.

### 1.1.1 Finding information of a request

When MYPROXY receives an HTTP request, it should first check for the following information:

1. **Request method**. In this assignment, MYPROXY only caches the web objects that are originated from `GET` and have the following file extensions: `html`, `jpg`, `gif`, and `txt`. Otherwise, MYPROXY simply forwards the HTTP requests to the web server.

| Sample Request Message | Descriptions |
|---|---|
| `GET http://www.cse.cuhk.edu.hk/ HTTP/1.1 \r\n` | Request method, URL, and Protocol version |
| `Host: www.cse.cuhk.edu.hk \r\n` | "`Host`" header field |
| `Proxy-Connection: keep-alive \r\n` | Declare that this is a persistent connection |
| `Cache-Control: no-cache \r\n` | "`Cache-Control header` |
| `..... \r\n` | Other headers. |
| `\r\n` | End of Request. |

Table 1: An example of a HTTP request message.

2. **Host Header**. When a browser is asked to connect through a proxy, the "`Host`" header will be added. This header field tells the proxy where should this request be forwarded to.

3. **URL and the file extension**. The URL is needed when MYPROXY needs to cache the web object; the url acts as an unique identifier of the web object. Also, the file extension (we will use "*file extension*" and "*file type*" interchangeably) of the web object is for deciding whether MYPROXY caches such an object or not. Note that there can be cases that an URL does not have an extension part.

   Note that we are not going to determine the file type using the "`Content-Type`" header in the response message.

### 1.1.2 `If-Modified-Since` and `Cache-Control`

MYPROXY handles two specific header fields that may appear in an HTTP request:

- `If-Modified-Since` **header**. The `If-Modified-Since` header is followed by a time string (let us call it the *IMS time*). An example is shown below:

  ```
  If-Modified-Since: Sat, 1 Jan 2000 00:00:00 GMT
  ```

  The client browser will include the `If-Modified-Since` header if (1) it has cached the same web object locally and (2) wants to confirm whether its cached web object is up-to-date.

  If the requested object has not been modified on the server side since the IMS time, then no data (only the response headers) would be returned from the server; a 304 (not modified) response will be returned without any message-body.

- `Cache-Control` **header**. The `Cache-Control` header specifies the caching option. Here, we only focus on the `no-cache` option. MYPROXY will always contacts the web server for any latest copy of the web object when this header appears in the request message, even though it may already cache the web object.

  The client browser will include "`Cache-Control: no-cache`" if it wants to request the web object from the web server *anyway*. This happens when you hold down the "`Shift`" key and click the "`Refresh`" button in the browser. For more details about the header fields, please refer to `http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html`.

Suppose that the **requested web object is cached in MYPROXY**. Then, MYPROXY will handle the following four cases:

- *Case (i): No* `If-Modified-Since` *and no* `Cache-Control`. MYPROXY returns the web object directly back to the client. Specifically, it constructs an HTTP response with status code 200 and includes the web object in the response.

- *Case (ii): With* `If-Modified-Since` *and no* `Cache-Control`. MYPROXY checks if the IMS time is later than the last modified time of the cached web object (see the hints below for how to obtain the last modified time). If yes, MYPROXY returns a 304 (not modified) response to the client; else it returns the cached object to the client.

  **Hints:** To retrieve the last modified time of a local file, we may use the function `stat()`. Call "`man 2 stat`" to find out how it is used.

- *Case (iii): No* `If-Modified-Since` *and with* `Cache-Control: no-cache`. MYPROXY will forward the request to the web server and it will also insert the `If-Modified-Since` header to the request, where the IMS time is set to be the last modified time of the cached web object.

- *Case (iv): With* `If-Modified-Since` *and with* `Cache-Control: no-cache`. MYPROXY will also forward the request to the web server. However, if the last modified time of the cached web object is *after* the IMS time, then the IMS time will be overwritten with the last modified time of the cached web object.

If the **requested web object is not stored in MYPROXY**, then MYPROXY will simply forward the *exactly same* request to the web server.

### 1.2 HTTP Response

Suppose that the web server returns a HTTP response. MYPROXY will check the status code of the response and do the following.

- *Case (i): Status code = 200.* It means that the HTTP response contains a web object. Then, MYPROXY will cache the web object in the local storage. It will overwrite the previously cached web object that corresponds to the same URL. Then it will forward the HTTP response back to the client.

  To store locally the web object, you need to provide a filename. Here, we construct the filename using the function `crypt()`. The function `crypt()` takes two parameters: (i) `key` and (ii) `salt`. The `key` is the unique identifier, and we set it to be the URL of the web object (e.g., `http://www.cse.cuhk.edu.hk/~pclee/web.txt`). You should generate a filename using the following code:

```
// "url" is some character array that stores the URL
int i;
char filename[23];
strncpy(filename, crypt(url, "$1$00$")+6, 23);
for( i = 0; i < 22; i++) {
    if (filename[i] == '/')
        filename[i] = '_';
}
```

  This hashes url with MD5 and salt "`00`" and returns a 28-byte string starting with "`$1$00$`". We should discard the first 6 bytes and take the remaining 22 bytes. Also, we should replace any '/' in the result with '_'. Note important that the "`url`" in the above code should include the **complete URL in the HTTP GET request** (i.e., do not use the `Host` header").

  You may assume that this string is unique for different URLs. For more details about the usage of `crypt()`, you can visit here: `http://linux.die.net/man/3/crypt`. Make sure you compile with the option `-lcrypt`.

- *Case (ii): Status code = 304.* It implies two possibilities: either the web object that is locally cached in the client browser is up-to-date, or the web object that is cached in MYPROXY is up-to-date. MYPROXY will do the following:

  - If the corresponding HTTP request has both the header fields `If-Modified-Since` and `Cache-Control: no-cache` (see Case (iv) in Section 1.1.2)) and MYPROXY has replaced the IMS time with the last modified time of its cached web object, then it means that MYPROXY has cached the most up-to-date web object. Thus, it retrieves the web object and returns it to the client browser with status code 200.
  - Otherwise, it implies that the client browser has cached the most up-to-date object. Thus, MYPROXY simply forwards the 304 HTTP response.

- *Case (iii): Status code = otherwise.* MYPROXY simply forwards the response to the client browser and lets the browser handle the response.

### 1.3 Multi-Threading

MYPROXY is a multi-threaded program. Each HTTP connection will be handled by a single thread. Each thread should do the following:

1. Accept a new connection and handle this new connection in a new thread.
2. Receive the entire HTTP request from the client browser.
3. Parse the HTTP request.
4. Either return the cached web object to the client with a 200 response, a 304 response, or forward the HTTP request to the web server.
5. If the HTTP request is forwarded to the web server:
   (a) wait for the entire HTTP response from the web server.
   (b) Parse the HTTP response.
   (c) If necessary, cache the web object in the HTTP response.
   (d) Either return the cached web object to the client browser, or forward the HTTP response to the client browser.
6. Repeat steps 2 - 5 if the same HTTP connection has more available HTTP requests (because HTTP 1.1 allows *persistent connections* with more than one HTTP request). If no more request is available, the thread quits.

There is a twist when MYPROXY is caching web objects. It is possible that two HTTP connections request for the same web object. While MYPROXY is saving the web object in its

local disk for one HTTP connection, another HTTP connection may want to retrieve the same web object.

To avoid such a problem (or the race condition), we use a *global mutex* to control all I/O operations within MYPROXY. That is, MYPROXY will do the following:

```
Lock(mutex)
Read/write all web objects from/to disk
Unlock(mutex)
```

You should understand that this is *not* an efficient approach, since a thread will be blocked even through it tries to access a different web object from the thread that currently holds the mutex. A more efficient approach is to lock individual files (e.g., using `flock()`), but we do not delve into the details here.

### 1.4 Assumptions and Requirements

You may pay attention to the following issues in your implementation.

- MYPROXY should handle all byte-ordering issues properly, since it communicates with the browsers and servers running on different operating systems.
- You do not need to clear up the cached objects in MYPROXY.
- Only HTTP is required. You do not need to worry about HTTPS or other protocols.
- MYPROXY only handles GET requests. You can assume that MYPROXY will not see other types of requests (e.g., POST).
- By default, all web servers are running on port 80. However, it is possible that they run on different ports. You should check the `Host` header field to see if it is running on a different port. For example, it may look like `Host: www.cse.cuhk.edu.hk:82`.
- You must cache all the web objects into the directory "`[current working directory]/cache`".
- There is a chance that:
  - A HTTP response is a large web object.
  - The client may choose to stop the download, e.g., press "`Esc`" while a page is loading. Then, MYPROXY will receive FIN from the browser and the connection is supposed to be closed.

- When MYPROXY keeps writing data to the closed client, a signal `SIGPIPE`, which means the process is writing to something closed, will be received.
- However, the default handler of the `SIGPIPE` is **termination**.

When you face such a scenario, you must not allow MYPROXY to be terminated accidentally. Rather, MYPROXY should handle that signal properly.

### 1.5 How to Test

MYPROXY works like a regular web proxy. Thus, you can use any currently available web browser to test. For simplicity, however, *we assume that only the Firefox browser is tested.* In Firefox, you can follow the procedures below:

> Step (1) Tools → Options → Advanced → Network Tab → Settings.
> Step (2) In "Manual proxy configuration", set the IP address and the port number that MYPROXY is running on.

MYPROXY can connect directly to any web server within the CSE department. However, you have to go through the department web proxy to reach the Internet. For simplicity, *we assume that the proxy is only connected to the websites within ".cse.cuhk.edu.hk".* Last, MYPROXY should be invoked as follows:

```
[tutor@localhost]$ ./myproxy [Port] [Milestone]
```

The option `Port` is the port number to which we bind MYPROXY, and the option `Milestone` is expected to be a number specifying the milestones, which will be described later.

## 2 Milestones

Note that multi-threaded implementation is a must for all the following milestones.

### 2.1 Milestone 1: no persistent connections & no caching (30%)

You have to handle HTTP requests and responses properly without caching, as described in Section 1.1.2. Also, you have to modify every request message in the following way:

| |
|---|
| "`Connection: keep-alive`" → "`Connection: close`" |
| "`Proxy-Connection: keep-alive`" → "`Proxy-Connection: close`" |

Then, every connection will become non-persistent and this keeps the processing of the response message easy.

### 2.2 Milestone 2: persistent connections & no caching (30%)

Again, you have to handle HTTP requests and responses properly without caching, as described in Section 1.1.2. However, you cannot modify the "`Connection`" or the "`Proxy-Connection`" header.

### 2.3 Milestone 3: persistent connections & caching (40%)

Together with the persistent connection handling, you have to include the caching control described in Section 1.1.2.

## 3 Submission Guidelines

You *must* at least submit the following files, though you may submit additional files that are needed:

- *myproxy.c*: the MYPROXY program
- *Makefile*: a makefile to compile myproxy.c

Your programs must be implemented in C/C++. They must be compilable on Linux virtual machines in the department. Please refer to the course website for the submission instructions. We will arrange demo sessions to grade your assignments. Have fun! :)

## Deadline. 23:59, March 24, 2013 (Sunday).

## Change Log

| Time | Affected Version(s) | Details |
|---|---|---|
| 2013 Feb 27 | NIL | Release version 1.0. |