Network Applications and Network Administration Project
November 12, 2007

Simple HTTP Proxy Server

Petr Zemek
xzemek02@stud.fit.vutbr.cz
Faculty of Information Technology, Brno

Abstract

This paper describes a simple HTTP proxy server implemented for purpose of our school course named "Network Applications and Network Administration". First an introduction to HTTP proxy servers is given and then application design and implementation details are presented. As for the implementation, reader is allowed to compile the project and try the proxy server by himself using given routine.

1 Introduction to HTTP proxy servers

A proxy server (in general) is a server which services the requests of its clients by forwarding requests to other servers. A client connects to the proxy server, requesting some service, such as a file, web page or other resource, available from a different server. The proxy server provides the resource by connecting to the specified server and requesting the service on behalf of the client. A proxy server may optionally alter the client's request or the server's response and sometimes it may serve the request without contacting the specified server.

An HTTP proxy server is a special type of a proxy server which focus on WWW trafic and it is often called a "web proxy". HTTP proxy servers can be used to block offensive web content, to enforce acceptable network use policies or to provide security and caching services.

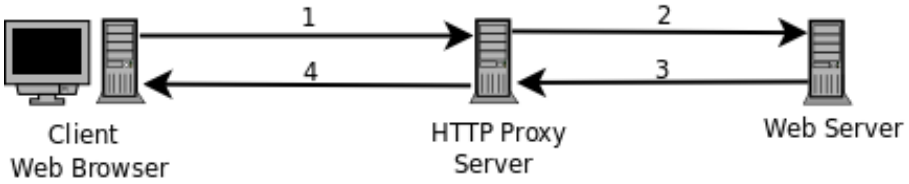

Figure 1: Client-server communication via HTTP proxy server

There is a simple scenario shown in the figure 1. For illustrative reasons lets assume that the HTTP proxy server's DNS name is `eva.fit.vutbr.cz`, web server's DNS name is `www.fit.vutbr.cz` and the client has set the proxy server address correctly in the web broswer. I will describe each step of the communication process separately:

1. Client wants to display `www.fit.vutbr.cz` content in its browser, so the web broswer creates and sends an HTTP request, but not to the "real" web server (`www.fit.vutbr.cz`), but to the HTTP proxy server (`eva.fit.vutbr.cz`).
2. HTTP proxy server receives the request, modifies its headers (or whatever it is set up to do) and forwards the modified request to "real" web server.
3. Web server receives the request and sends a response. Since the request was sent from the proxy server and the web server does not know anything about the "real" client, the response is sent to the proxy server instead of the client.
4. HTTP proxy server receives the response, modifies it (or not, depends on the rules, situation etc.) and forwards it to the client.

2 Simple HTTP proxy server

Project goal was to create a simple HTTP proxy server that will connect to the selected web server (according to the arguments) and continuously forward requests from clients to the web server. Note that this behavior (can connect to only one server) is not very useful, but it is sufficient for our school purposes.

2.1 Further description

Proxy server tracks files that the connected client wants to get (HTTP request type is `GET`) from the web server and object filenames (directories are omitted in the path) with client IP address are printed to standard output in the following format:

```
<client_ip>:<filename>
```

However, if the object file name matches some pattern specified in a file named `forbidden` (which is placed in the project root directory) and the client's IP address also matches, client will receive an `HTTP 404 Not Found` response and the record will be printed to standard error in the same format as specified above.

Records in the `forbidden` file must have the following format (one record per line). Note that the pattern is not a regular expression but only a simple text string.

```
<client_ip_1>:<pattern_1>
<client_ip_2>:<pattern_2>
...
```

2.1.1 Features

- server is able to serve more than one client at a time (*concurrent* behavior)
- supports HTTP/1.0 and HTTP/1.1 protocols (TCP)
- supports persistent connections (only HTTP/1.1)
- modifies HTTP/1.1 `Host` header when the original address is different from the host address the server is connected to (it causes problems if the address is not modified because of virtual hosts)
- allows administrator to forbid some files from being downloaded by some clients (see subsection 2.1)
- prints information about files that clients want to get (see subsection 2.1)

2.1.2 Limitations

This simple HTTP proxy server does not support "advanced" features of proxy servers like connection to more than one web server, caching, authentication, policies etc.

2.2 Application design

In the figure 2.2 you can see all main classes used in the project (operation signatures and other details are omitted because of simplicity).

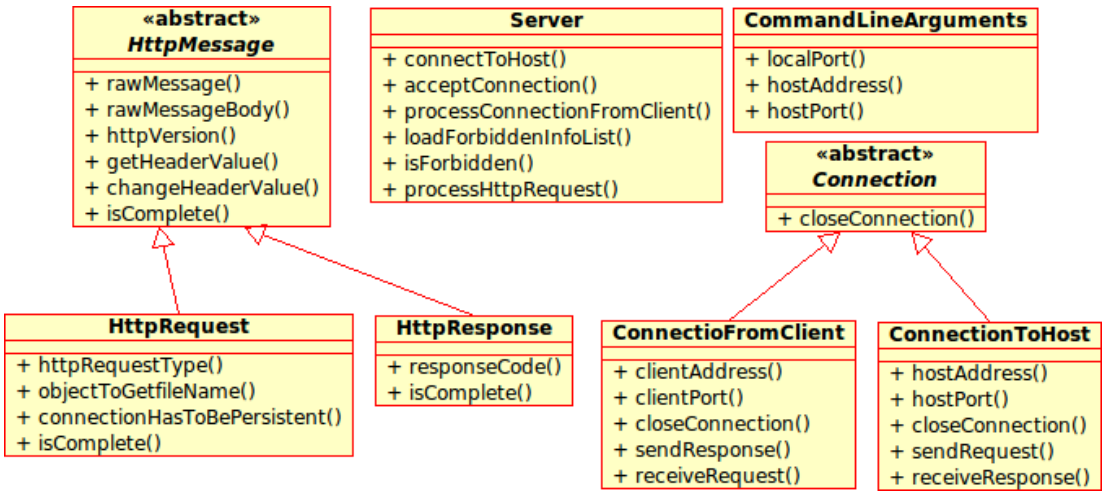

Figure 2: Simple UML class diagram with all main classes

2.3 Application flow

When a `proxyhttp` server is started, program arguments are parsed into a `CommandLineArguments` class instance and a `Server` class instace is created (it will listen on the selected port). After that the server accepts connections from clients (instances of the `ConnectionFromClient` class) and process them in an "endless" loop until it gets stopped.

Processing a connection means that a copy of the main process is created (concurrent server behavior) to handle that connection. This new process connects to the web server using the specified arguments (by creating a `ConnectionToHost` class instance) and then receives requests (`HttpRequest` class instances) from the client, forwards them to the web server and received responses (`HttpResponse` class instances) forwards to the client. After the connection to the client is closed, the process is terminated. Meanwhile, the main process accepts new connections.

2.4 Implementation details

Here are explained some implementation details that were not described in the previous sections but were considered important to mention.

- Application was written in C++ with STL and exceptions support, network communication is done by using BSD sockets and also some POSIX functions were used (`fork()`, `stat()`, etc.).
- Process copies (*childs*) are created by calling `fork()` in the main process and these childs are terminated (using `exit()` function) after the connection to a client gets closed. SIGCHLD signal is ignored to avoid zombie processes. Other signals (except SIGINT, SIGTERM, SIGSTOP and SIGKILL) are blocked.
- Message sendings are done via `send()` and `recv()` functions.
- If there is no filename specified in an HTTP request (like `"GET / HTTP/1.1"`), `index.html` is set as the filename.
- *URL searchpart*[1] (everything after the first '?' character) is removed from the object filename in HTTP requests (including '?'), so for example the object filename from this HTTP request: `"GET /path/index.php?test=true HTTP/1.1"` is only `index.php`.

[1] http://www.ietf.org/rfc/rfc1738.txt

- To check whether a complete HTTP request and response was received the following methods are used:
  - If a request or a response has a `Content-length` header, size of the message body is compared with the header value.
  - If a response has a `Transfer-encoding` header with its value equal to `chunked` the message body is searched for `"\r\n0\r\n\r\n"` (only HTTP/1.1 responses).
  - In other cases the message is searched for `"\r\n\r\n"` (end of a header).
- TCP connection from a client gets closed if this client requested a forbidden file (after an `HTTP 404 Not Found` response is sent to the client).

2.5 Compilation

To compile the project simply run `make` command in the project root directory. You can also compile `tester` (unit tests for all classes) by running `make test` command instead (but see `Readme` for requirements, namely `CPPUnit`[2]).

2.6 Usage

Now you can start the simple HTTP proxy server by running `proxyhttp`. Lets assume that you want to start the server listening on `localhost:36547` and you want to forward requests to `www.fit.vutbr.cz`. So, run `proxyhttp localhost:36547 www.fit.vutbr.cz` on your system and then try to connect to `localhost:36547` in your favorite web browser. If you see `www.fit.vutbr.cz` homepage, everything is working correctly.

Full `proxyhttp` command synopsis is the following:

```
proxyhttp <local_port> <host>[:<port>]

local_port - port number (1 - 65535)
host - host address (DNS name)
port - host port number (1 - 65535, 80 by default)
```

For more details please see `Readme` file or run `proxyhttp --help`. If you want to edit forbidden client address and filename combinations, please see subsection 2.1.

2.6.1 Advanced usage example

I will show an example how to forbid a client (`88.146.0.115`) from getting `.gif` and `.jpg` images when connecting through the proxy server. Proper `forbidden` file content is:

```
88.146.0.115:.jpg
88.146.0.115:.gif
```

Proxy server will run on the faculty server `eva.fit.vutbr.cz` on port 34535 and the web server will be `www.fit.vutbr.cz`. The redirection of standard output to `/dev/null` will cause that only forbidden IP:filename combinations will be printed.

```
eva$ ./proxyhttp 34535 www.fit.vutbr.cz > /dev/null
```

I will connect to this server via my web browser from my computer (lets suppose that is has the following IP address: `88.146.0.115`). This is the address I will use to connect to the proxy server:

```
http://eva.fit.vutbr.cz:34535/
```

[2] http://cppunit.sourceforge.net/

After the page is successfully loaded, output from the proxy server (on `eva.fit.vutbr.cz`) will be:

```
eva$ ./proxyhttp 34535 www.fit.vutbr.cz > /dev/null
88.146.0.115:fit_cz.gif
88.146.0.115:globus.gif
88.146.0.115:flag2_gb.gif
88.146.0.115:portal.gif
88.146.0.115:blue_home.gif
88.146.0.115:blue_bul.gif
88.146.0.115:rss.gif
88.146.0.115:fit_logo_cz.gif
88.146.0.115:IMG_4125x.jpg
```

You can clearly see that all `.gif` and `.jpg` images were forbidden. Note that the output can be different if you try this example.

3 Conclusion

All specifications from the assignment were observed and satisfied during the project development. Main parts were designed considering possible future development and extensibility, like connecting to more than one web server, modifying HTTP request/response headers or adding new functionality. The application was successfully tested on the faculty server `eva` (FreeBSD, x86) and on my system (GNU/Linux, x86). Unit testing suites (83 tests in total) for all classes were written using `CPPUnit` and run regularly. Memory leaks were also taken into account and tested with `valgrind`[3].

References

[1] RFC 1945, *Hypertext Transfer Protocol – HTTP/1.0*, `http://www.faqs.org/rfcs/rfc1945.html`

[2] RFC 2616, *Hypertext Transfer Protocol – HTTP/1.1*, `http://www.faqs.org/rfcs/rfc2616.html`

[3] RFC 1738, *Uniform Resource Locators (URL)*, `http://www.faqs.org/rfcs/rfc1738.html`

[4] James Marshall, *HTTP Made Really Easy*, `http://www.jmarshall.com/easy/http/`

[5] Wikipedia, *Internet Encyclopedia*, `http://en.wikipedia.org/wiki/Proxy_server`

[3] http://valgrind.org/