Project 2 Group Project
Implementing a Simple HTTP Web Proxy

Overview:

This is a group project. Students are encouraged to form a group of 2-3 students (It is ok if you want to take it as an individual project). This project aims to implement a simple web proxy using HTTP 1.0, and it consists of two steps, each of which has a submission deadline. The last step includes demonstration and presentation.

Background:

HTTP

The Hypertext Transfer Protocol or (HTTP) is the protocol used for communication on the web. That is, it is the protocol which defines how your web browser requests resources from a web server and how the server responds. For simplicity, in this project we will be dealing only with version 1.0 of the HTTP protocol, defined in detail in RFC 1945. You should read through this RFC and refer back to it when deciding on the behavior of your proxy.

HTTP communications happen in the form of transactions, a transaction consists of a client sending a request to a server and then reading the response. Request and response messages share a common basic format:

- An initial line (a request or response line, as defined below)
- Zero or more header lines
- A blank line (CRLF)
- An optional message body.

For most common HTTP transactions, the protocol boils down to a relatively simple series of steps (important sections of RFC 1945 are in parenthesis):

1. A client creates a connection to the server.
2. The client issues a request by sending a line of text to the server. This **request line** consists of a HTTP *method* (most often GET, but POST, PUT, and others are possible), a *request URI* (like a URL), and the protocol version that the client wants to use (HTTP/1.0). The message body of the initial request is typically empty. (5.1-5.2, 8.1-8.3, 10, D.1)
3. The server sends a response message, with its initial line consisting of a **status line**, indicating if the request was successful. The status line consists of the HTTP version (HTTP/1.0), a *response status code* (a numerical value that indicates whether or not the request was completed successfully), and a *reason phrase*, an English-language message providing description of the status code. Just as with the request message, there can be as many or as few header fields in the response as the server wants to return. Following the CRLF field separator, the message body contains the data requested by the client in the event of a successful request. (6.1-6.2, 9.1-9.5, 10)
4. Once the server has returned the response to the client, it closes the connection. It's fairly easy to see this process in action without using a web browser. From a Unix/LINUX prompt, type:
   ```
   telnet www.yahoo.com 80
   ```

This opens a TCP connection to the server at www.yahoo.com listening on port 80- the default HTTP port. You should see something like this:

```
Trying 209.131.36.158...
Connected to www.yahoo.com (209.131.36.158).
Escape character is '^]'.
```

type the following:

```
GET / HTTP/1.0
```

and hit enter twice. You should see something like the following:

```
HTTP/1.0 200 OK
Date: Fri, 10 Nov 2006 20:31:19 GMT
Connection: close
Content-Type: text/html; charset=utf-8
<html><head>
<title>Yahoo!</title>
(More HTML follows)
```

There may be some additional pieces of header information as well-setting cookies, instructions to the browser or proxy on caching behavior, etc. What you are seeing is exactly what your web browser sees when it goes to the Yahoo home page: the HTTP status line, the header fields, and finally the HTTP message body- consisting of theHTML that your browser interprets to create a web page.

HTTP Proxy

Ordinarily, HTTP is a client-server protocol. The client (usually your web browser) communicates directly with the server (the web server software). However, in some circumstances it may be useful to introduce an intermediate entity called a proxy. Conceptually, the proxy sits between the client and the server. In the simplest case, instead of sending requests directly to the server the client sends all its requests to the proxy. The proxy then opens a connection to the server, and passes on the client's request. The proxy receives the reply from the server, and then sends that reply back to the client. Notice that the proxy is essentially acting like both a HTTP client (to the remote server) and a HTTP server (to the initial client).

Why use a proxy? There are a few possible reasons:

• **Performance:** By saving a copy of the pages that it fetches, a proxy can reduce the need to create connections to remote servers. This can reduce the overall delay involved in retrieving a page, particularly if a server is remote or under heavy load.

• **Content Filtering and Transformation:** While in the simplest case the proxy merely fetches a resource without inspecting it, there is nothing that says that a proxy is limited to blindly fetching and serving files. The proxy can inspect the requested URL and selectively block access to certain domains, reformat web pages (for instances, by stripping out images to make a page easier to display on a handheld or other limited-resource client), or perform other transformations and filtering.

• **Privacy:** Normally, web servers log all incoming requests for resources. This information typically includes at least the IP address of the client, the browser or ther client program that they are using (called the User-Agent), the date and time, and the requested file. If a client does not wish to have this personally identifiable information recorded, routing HTTP requests through a proxy is one solution. All requests coming from clients using the same proxy appear to come from the IP address and User-Agent of the proxy itself, rather than the individual clients. If a number of clients use the same proxy (say, an entire business or university), it becomes much harder to link a particular HTTP transaction to a single computer or individual.

Step 1 – Making Your Proxy Work in a Linux Shell (10 points):

Requirements:

- You need to implement the proxy program called “proxy.c/cpp”. Assume the executable file for this proxy program is “proxy”, then run your proxy with the following command:

  ```
  ./proxy hostname [port]
  ```

  Where hostname is the web server name and port is the port number that the proxy listens on. The behavior of your proxy should be like that of “telnet hostname [port]”, as demonstrated in class. You proxy should display the following messaging (the IP address corresponds to the hostname):

  ```
  Trying 124.129.12.3...
  Connected to 124.129.12.3.
  Ctrl-C to escape.
  ```

- And, the proxy waits for the HTTP request message(ending with a blank line, do not forget!), for example:

  ```
  GET   /index.html  HTTP/1.0
  ```

- If your proxy is working correctly, the headers and HTML of the requested object should be displayed on your terminal screen.

Submisson:

- Due by: 6:00PM 04/30/2009, Thursday
- Make your files into a tar package, named project2.tar, which includes Makefile, all source files, README file.
- Howto: **/home/fac/zhuy/spring09/SubmitHW593 p2 project2.tar**
- Run the above command in cs1.seattleu.edu

Step 2 – Configuring a Web Browser to Use Your Proxy (20 points):

A Caveat

If you write a single-threaded proxy server, you will probably see some problems when you use your proxy with a standard web browser. Because a web browser like Firefox or IE issues multiple HTTP requests for each URL you request (for instance, to download images and other embedded content), a single-threaded proxy will likely miss some requests, resulting in missing images or other minor errors.

Firefox

Version 2.0:

1. Select **Tools->Options** from the menu.
2. Click on the '**Advanced**' icon in the **Options** dialog.
3. Select the '**Network**' tab, and click on '**Settings**' in the '**Connections**' area.
4. Select '**Manual Proxy Configuration'** from the options available. In the boxes, enter the hostname and port where proxy program is running.

Earlier Versions:

1. Select **Edit->Preferences** from the menu.
2. On the '**General**' tab, click 'Connection Settings'.
3. Select '**Manual Proxy Configuration'** and enter the hostname and port where your proxy is running.

To stop using the proxy server, select 'Direct connection to the Internet' in the connection settings dialog.

Configuring Firefox to use HTTP/1.0

Because Firefox defaults to using HTTP/1.1 and your proxy speaks HTTP/1.0, there are a couple of minor changes that need to be made to Firefox's configuration. Fortunately, Firefox is smart enough to know when it is connecting through a proxy, and has a few special configuration keys that can be used to tweak the browser's behavior.

1. Type 'about:config' in the title bar.
2. In the search/filter bar, type 'network.http.proxy'
3. You should see three keys: network.http.proxy.keepalive, network.http.proxy.pipelining, and network.http.proxy.version.
4. Set keepalive to false. Set version to 1.0. Make sure that pipelining is set to false.

Internet Explorer

Take a look at http://support.microsoft.com/kb/135982 for complete instructions on enabling a proxy for various versions in Internet Explorer.

You should also do the following to make Internet Explorer work in a HTTP 1.0 compatible mode with your proxy:

1. Under Internet Options, select the 'Advanced' tab.
2. Scroll down to HTTP 1.1 Settings. Uncheck 'Use HTTP 1.1 through proxy connections'.

Requirements:

- You must make your proxy a multi-client program by using multi-threading (PThread). Thread synchronization is needed.
- A brief class presentation is required. All the teams need to talk about the system architecture, main data structures, design issues & challenges, solutions, lessons learned, etc.
- Class demonstration is required. All the teams need to test their proxy via their web browsers.

Submisson:

- Due by: 6:00PM 5/12/2009, Tuesday
- Make your files into a tar package, named project2.tar, which includes Makefile, all source files, README file.
- Howto: **/home/fac/zhuy/spring09/SubmitHW593 p2 project2.tar**
- Run the above command in cs1.seattleu.edu

Grading:

- Grading will be based on software, presentation & demo. Performance will be one of important metrics to evaluate your project. E.g., how fast can your proxy relay the pages (in my experience, this is directly related to your proxy design including thread synchronization)? How well can your proxy handle a large number of requests sent by web browsers?