Experiment II: Set up an HTTP Proxy

Submission deadline: 20th Feb 2013 (Wednesday) 5:30PM

Submit your base program as per above deadline. Next exam will be based on this base program.

In this assignment, you will implement a simple web proxy that passes requests and data between a web client and a web server. This will give you a chance to get to know one of the most popular application protocols on the Internet- the Hypertext Transfer Protocol (HTTP) v. 1.1. When you're done with the assignment, you should be able to configure your web browser to use your personal proxy server as a web proxy.

Introduction: The Hypertext Transfer Protocol

The Hypertext Transfer Protocol or (HTTP) is the protocol used for communication on this 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 assignment we will be dealing only with version 1.1 of the HTTP protocol, defined in detail in RFC 2616. You should read through this RFC and refer back to it when deciding on the behaviour 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 2616 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/1.1). The message body of the initial request is typically empty.
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/1.1), 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.
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 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/1.1
```

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

```
HTTP/1.1 200 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 behaviour, 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 the HTML that your browser interprets to create a web page.

HTTP Proxies

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).

Links:

- RFC 2616 The Hypertext Transfer Protocol, version 1.1

Assignment Details

The Basics

Your first task is to build a basic web proxy capable of accepting HTTP requests, making requests to remote servers, and returning data to a client.

This assignment can be completed in either ANSI C or C++ to give a binary called `proxy` that takes as its first argument a port to listen from. Don't use a hard-coded port number.

You shouldn't assume that your server will be running on a particular IP address, or that clients will be coming from a pre-determined IP.

Listening

When your proxy starts, the first thing that it will need to do is establish a socket connection that it can use to listen for incoming connections. Your proxy should listen on the port specified from the command line, and wait for incoming client connections.

Once a client has connected, the proxy should read data from the client and then check for a properly-formatted HTTP request. An invalid request from the client should be answered with an appropriate error code.

Parsing the URL

Once the proxy sees a valid HTTP request, it will need to parse the requested URL. The proxy needs at most three pieces of information: the requested host and port, and the requested path. See the `URL` manual page for more info.

Getting Data from the Remote Server

Once the proxy has parsed the URL, it can make a connection to the requested host (using the appropriate remote port, or the default of 80 if none is specified) and send a HTTP request for the appropriate file. The proxy then sends the HTTP request that it received from the client to the remote server.

Returning Data to the Client

After the response from the remote server is received, the proxy should send the response message to the client via the appropriate socket. Once the transaction is complete, the proxy should close the connection.

Testing Your Proxy

Run your client with the following command:

`./proxy <port>`, where `port` is the port number that the proxy should listen on. As a basic test of functionality, try requesting a page using telnet:

```
telnet localhost <port>
Trying 127.0.0.1...
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.
GET http://www.google.com HTTP/1.1
```

If your proxy is working correctly, the headers and HTML of the Google homepage should be displayed on your terminal screen.

For a slightly more complex test, you can configure your web browser to use your proxy server as its web proxy. See the section below for details.

Configuring a Web Browser to Use a Proxy

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. **That's OK**. You are not required to use threading in this assignment.

As long as your proxy works correctly for a simple HTML document (like, for instance, this assignment page) and follows the RFC, you can still receive all the points for this assignment.

Firefox

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.

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