FAST- National University of Computer and Emerging Sciences, Karachi.

FAST School of Computing

Assignment # 2(CLO-2), Fall 2023

CS3001- Computer Networks

ASSIGNMENT-II (HTTP Proxy)

Submission Guidelines:

- This is a group assignment. Each group can be of two members only.
- This is a programming assignment and the submission will be based on 4-5 minutes demo followed by a short viva on Wednesday 25th October, 2023.
- Each group member has to submit the code and screenshot of the working demo on Google classroom.
- The student ID, names and section must be mentioned clearly during the demo.
- Each group should mark their attendance during the demo.
- Plagiarism in any form will result in straight "F".

Assignment # 2 (100 points)

In this assignment, you will implement a web proxy that passes requests and data between multiple web clients and web servers, concurrently. The idea is to get to know one of the most popular application protocols on the Internet i.e. HTTP. 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.

For simplicity, in this assignment, we will be dealing only with version 1.0 of the HTTP protocol, defined in detail in RFC 1945. You may refer to that RFC while completing this assignment, but the instruction in this assignment should be self-contained.

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

The initial line and header lines are each followed by a "carriage-return line-feed" (\r\n) signifying the end-of-line. For most common HTTP transactions, the protocol boils down to a relatively simple series of steps.

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 requestline 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 touse (HTTP/1.0). The request line is followed by one or more header lines. The message body of the initial request is typically empty. (see 5.1-5.2, 8.1-8.3, 10, D.1 of the RFC 1945)
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 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 bythe client in the event of a successful request. (see 6.1-6.2, 9.1-9.5, 10 of the RFC 1945)
4. Once the server has returned the response to the client, it closes the connection.

HTTP Proxies
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).

Assignment Details

The Basics

Your task is to build a web proxy capable of accepting HTTP requests, forwarding requests to remote (origin) servers, and returning response data to a client. The proxy MUST handle concurrent requests by forking a process for each new client request using the fork() system call. You will only be responsible for implementing the GET method. All other request methods received by the proxy should elicit a "Not Implemented" (501) error (see RFC 1945 section 9.5 - Server Error).

Do not use a hard-coded port number for listening. 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 or interface. Each new client request is accepted, and a new process is spawned using fork() to handle the request. There should be a reasonable limit on the number of processes that your proxy can create (e.g., 100). Once a client has connected, the proxy should read data from the client and then check for a properly-formatted HTTP request and all other headers also needs to be properly formatted
In this assignment, client requests to the proxy must be in their absolute URI form (see RFC1945, Section 5.1.2) -- as your browser will send if properly configured to explicitly use a proxy. An invalid request from the client should be answered with an appropriate error code, i.e. "Bad Request" (400) or "Not Implemented" (501) for valid HTTP methods other than GET. Similarly, if headers are not properly formatted for parsing, your client should also generate a type-400 message.

Parsing Library

You also have to write a small routing that can do string parsing on the header of the request.

Parsing the URL

Once the proxy sees a valid HTTP request, it will need to parse the requested URL. The proxy needs at least three pieces of information: the requested host and port, and the requested path. If the hostname indicated in the absolute URL does not have a port specified, you should use the default HTTP port 80.

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 the HTTP request for the appropriate resource.

Returning Data to the Client

After the response from the remote server is received, the proxy should send the response message as-is to the client via the appropriate socket.

Configuring a Web Browser to Use a Proxy

In the end you will have to configure both Firefox and Chrome to use your proxy server.

Links:

- Guide to Network Programming Using Sockets
- HTTP Made Really Easy- A Practical Guide to Writing Clients and Servers
- Wikipedia page on fork()