PROXY SERVER
CS653 Project

Manish Kumar
Roll : 150379

March 2018

About the Project

Proxy Server is a computer system or an application that acts as an intermediary for client requesting for some resource from other servers. Web proxies provide anonymity and can be used to bypass IP address blocking.

Web proxies forward HTTP requests to the destination server and return the response to the client. They can also be used to filter out requests to some sites and can make stricter policies on the kind of requests that are entertained.

Deliverable

A Web proxy application that could serve HTTP / HTTPS requests but allows the administrator to predefine (in the code) any HTTP methods that should not be allowed.

In addition to that, it should do the following:

- Allowed HTTP methods include GET, POST, HEAD, CONNECT.
- **Domain Filtering:** Filter out domain names / ip address that are provided in arguments at the time of launch. In such cases the client should receive "Bad Request" response.
- In case the code is made to neglect certain HTTP methods, HTTP status code 405 (Method not allowed) is to be sent to the client.
- It should forward appropriate HTTP requests to the destination server and the responses back to the client.
- If "Host: xyz" is not found then send a "Bad Request" back to client.
- It should be a **concurrent server**.
- It should be resistant to malicious activity from client side to shut it down.
- It should not be killed by SIGINT signal. And upon encountering the signal SIGUSR1, it should print out the statistics about the number of requests successfully processed, filtered, resulted in error, etc. Also it should exit gracefully when seeing a SIGUSR2 signal.

Task Time Estimates

- 1st week of March $\longrightarrow$ build basic server
- 2nd week of March $\longrightarrow$ add domain filtering options
- 3rd week of March $\longrightarrow$ add https method and chunk parsing
- 1st week of April $\longrightarrow$ make it a concurrent server
- 2nd week of April $\longrightarrow$ add statistics feature
- 3rd week of April $\longrightarrow$ handle signals to direct specific functions

Implementation

All the promised features were fulfilled with their implementation decribed below:

Server.hs

It is the main module of the application. It sets up a server that listens for the incoming requests from the (new) clients forever until explicitly shut down. Every time a new client is accepted, we create a new thread which is handled by ThreadProxy module.

Also it has procedures to handle signals SIGUSR1 and SIGUSR2. For gracefully exiting the program upon encounter of SIGUSR2, we have a MVar semaphore “tidList” that is used to keep track of the currently active thread and activer sockets (client and destination server) under it.

ThreadProxy.hs

The client socket is read for the request and the destination server it wants to connect to. This is done by HttpParser module. It checks for any sort of error or filter in the request packet. It then sets up a socket for the destination server.

Associated with each socket is a “TerminalInfo” which consists of buffer, socketID, httpParser’s information, isClosed flag, httpPacket object, and stillToProcess data. Next, it calls the “requestResponseCycle” that polls for the socket which is ready to read or send data. In case, any socket wants to send data, a thread is created that fills the incoming data into other socket’s buffer. And in case, any socket is ready to read, we just flush its buffer.

HttpParser.hs

This module is responsible for parsing the client request. It can parse GET, HEAD, POST, CONNECT requests. For the POST method it could either parse the requests have header “Content-Length” or “Transfer-encoding : Chunked”. This module provides the hostname on which the destination socket is to be build up by the ThreadProxy module and a mechanism to detect admin defined errors and filters.

Stats.hs

This module provides data structures and functions to keep track of number of success, filters and errors from the beginning of the program.

Performance

The program is considerably fast with relatively low memory utilization. It might use of significant amount of memory when there are lot of clients connected to it simultaneously. This is because allocation of memory to new threads fills up the main memory. It would not have been the case if ‘select’ was used as it would had reduced the number of threads by more than thrice. Unfortunately, I did not find any such system call in haskell and hence was forced to do polling.