4/22/2015

Kodjo Komedja, Durko Justin | Computer Networks | April 22 2015

**UNIVERSITY OF NORTH TEXAS**

**PROJET**

Multithreaded web proxy

**Table of Contents**

1. Problem being solved
2. Approach
3. Implementation

**A) Problem**

In this project, you will develop a web proxy. When your proxy receives an HTTP request for an object from a browser, it generates a new HTTP request for the same object and sends it to a remote server that is hosting the requested object. When the proxy receives the corresponding HTTP response with the object from the remote server, it creates a new HTTP response, including the object, and sends it to the client. The proxy will be multi-threaded, so it will be able to handle multIPle requests at the same time. Your proxy should have the following features:

1. Blocking web sites if they are found in a black list (www.facebook.com, www.youtube.com, www.hulu.com, [www.virus.com](http://www.virus.com))

2. Filtering out inappropriate language (http://www.hyperhero.com/en/insults.htm) from a requested site

3. Caching sites that have already been visited.

**B) Approach to solve the problem**

The goal of TCP proxy server is to act as an intermediary forwarding TCP based connection from external client applications onto remote server. The communication architecture diagram start from from the client to proxy which treat the request according to built-in parameters then contact the remote HTTP server and get the response back from server, then read the content to verify the matching of set of requirements before rendering a view (response) to client browsers.

// Up flow communication: Client ---> Proxy---> HTTP server +---------------+

// +---->------> |

// +-----------+ | | Remote Server |

// +---------> [x]--->----+ +---<---[x] |

// | | TCP Proxy | | +---------------+

// +-----------+ | +--<--[x] Server <-----<------+

// | [x]--->--+ | +-----------+

// | Client | |

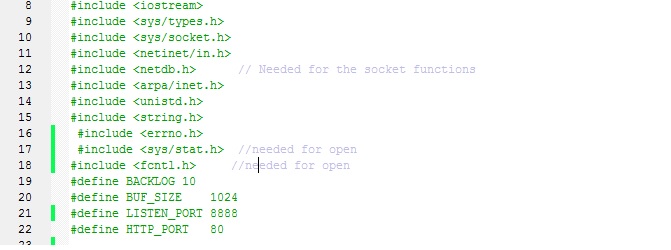
// | <-----<----+

// +-----------+

// Down flow communication: Client <--- Proxy <--- HTTP server

**Understanding socket programing**

On the proxy we start by defining the port number (8888) we wish to run our application on. We include built-in library headers, which are important for defining an end point to socket; open, read, and write files; multi-threading concepts, and hash-mapping (saving blocking specific websites).



They are followed by defined variables *int sock-send, sockaddr\_in, addr\_send* which are required to initialize a socket end point connection and to make the binding. The socket is created with *sock\_send=socket(PF\_INET,SOCK\_STREAM,…)*. We connect to server socket with *connect(sock\_send, &addr\_send,….)*

**Understanding multithread in C++**

On the proxy server, required variables are *pthread\_t, thread\_result*. We use a socket binding for client’s browser apps, and we listen to port number from client application and launch a thread to handle the client operation. A thread count is initialized to 10 and decrement each time a new client browser is connected (which means our server can only accept 10 users) which is enough for multithreading with a clients - proxy server setup.

**Understanding writing data to file as *ofstream* and reading from file as *ifstream , and both fstream***

This allows us the capability to save our data in file which is very useful for caching request. Because the proxy server application is executed on (UNT computer machines), we can only save our data in file and send the content to client browsers if they ask for a web content that already existed. We will clarify step by step how our code works in the sections to follow.

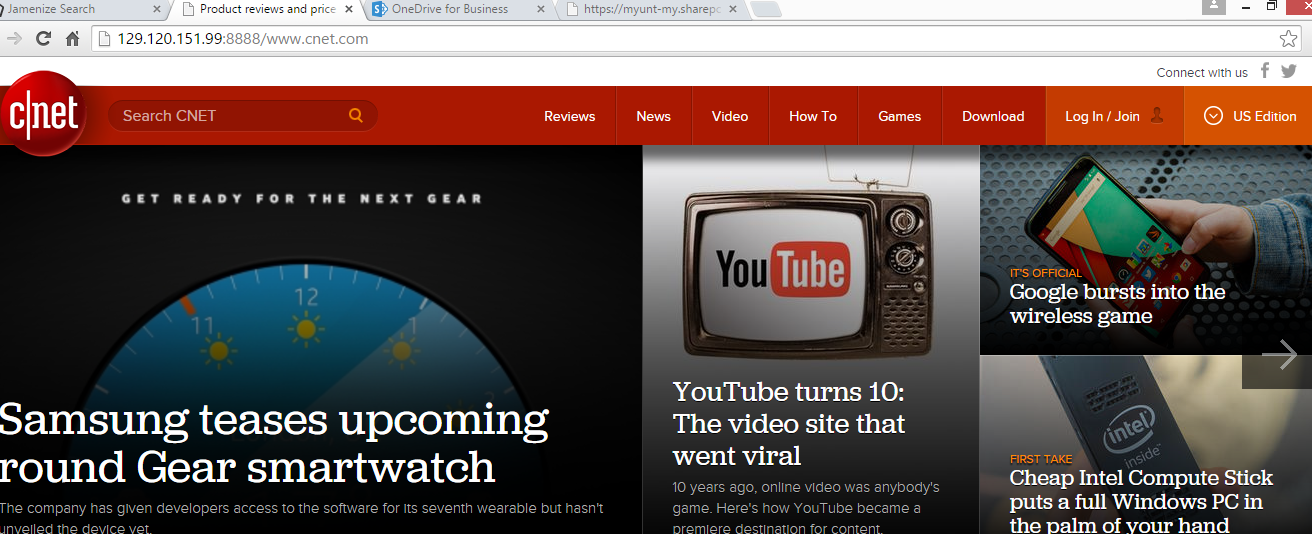
**Understanding regex() expression matching**

Regular expressions are a standardized way to express patterns to be matched against sequences of characters. The standard C++ library provides support for regular expressions in the <regex> header through a series of operations. All these operations make use of some typical regex parameters:In our case, we match bad word expression from list of bad words with the content of website.

**Understanding Hashmap data structure:** *Defines a data structure for clients records in file*

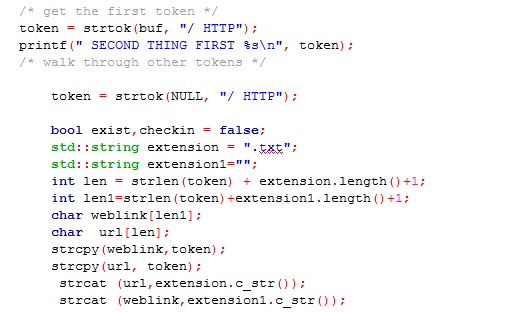
The solution involved the handling of blocking website contents which were handled with STL Map. The blocking content within that map class contain relevant information to clients to take place. In addition, using hashmap, make the parsing bad word faster, than using a regular read() method, or loop.

**C) Implementation**

****

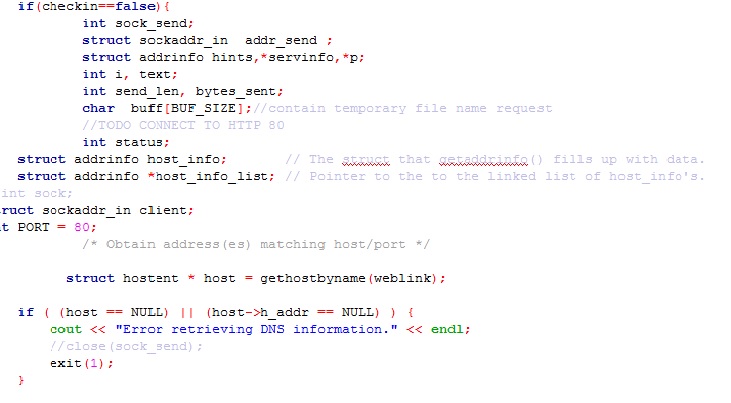
* **Flow from Client browser to Proxy Server:**

When the client browser enter [***http://IP\_address:port\_number/website***](http://ip_address:port_number/website), we get the content of website URL with recv() socket Linux method for receiving data into variable array of char [buf\_size]. The request is processed by parsing the URL website to get only the URL content (website) with the help of strtok(buf, delimiter) to split String character into 2 tokens. Therefore, we start by checking if the concatenation of URL and extension (.text) corresponds to filename existing in our folder for caching purpose. If the file is not in our cache folder we create a new socket to initialize a connection with HTTP remote server on port 80.



* **Flow from Proxy Server to Remote Server:**

In case of file not in cache, we check if the URL web link is on the blocking website list. In our code, we have used a hashmap collection to save all blocking URL links. Since, we want to efficiently to retrieve a data and faster, using hashmap look through data in constant time knowing the key. If the URL link matches with a a blocking link, we issued to Client browser a message << URL or age has been blocked>>. If it is not in the blocked list, we get the IP address of the hostname with gethostbyname(URL). Then we connect to host with IP address on port 80. Therefore, we start making a HTTP request (GET / HTTP/1.1/ hostname Accept:text/html) and we wait for response.

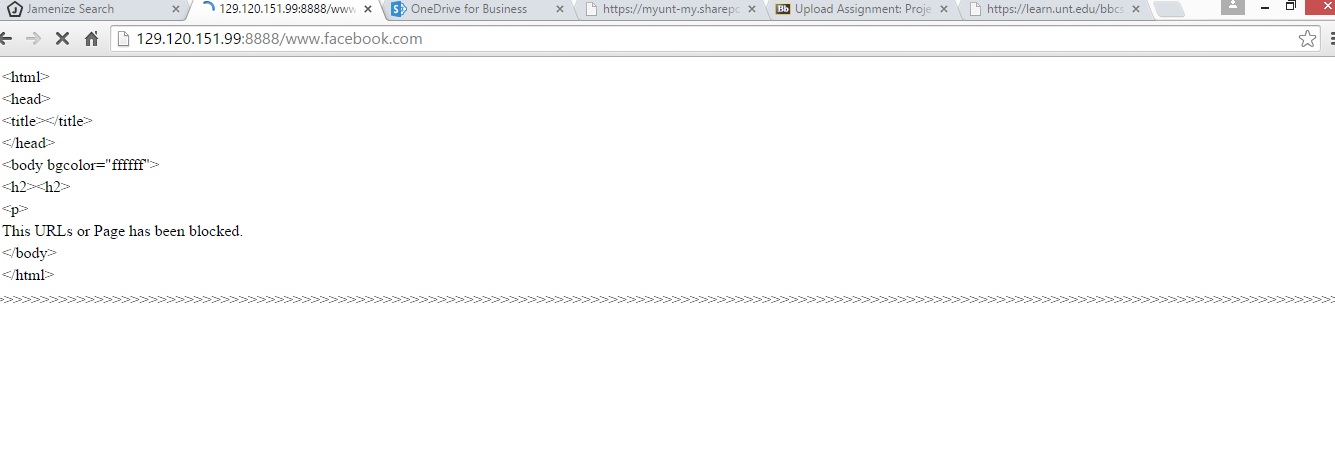


* **Flow from Remote Server to Proxy Server:**

The HTTP server send a reply on socket we have created. We begin par open a file in fstream , then we read the socket bit by bit at the same time both writing to file bit by bit and writing to client browser socket bit by bit. The checking of bad word is done at this level, we use regex() method which parse through the socket to find a matching bad word and replace by “B” then ,we continue the process until we reach end of socket descrIPtor file. Thus, we send the content to user browser.

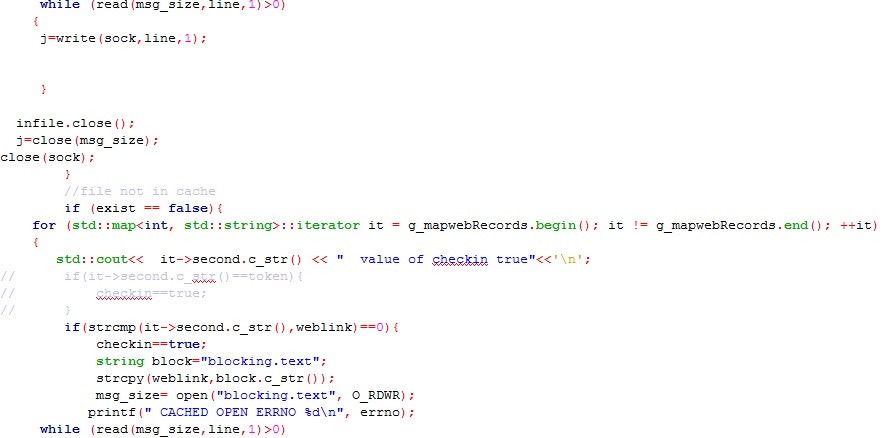


Launching [www.facebook.com](http://www.facebook.com) in browser is blocked by the proxy server

****

* **Flow from Proxy Server to Client browser:**

We can not use send() method socket linux to Client due to large file size. Because it required to define a char buffer to contain all contents from file, which will be impossible. We are addressing this issue by writing to Client socket bit at the time. Therefore the Client need to wait some period of time to have the rendering view on its browser.



**PATH TO GITHUB PROJECT:**

Path to GitHub: https://github.com/jdurko/web\_proxy

