Michael Baptist
CE156 – Proxy Phase 1

Final Project Phase 1 – HTTP Proxy – Application Layer Protocol

In figure 1 below you can find the state diagram for the phase1 version of HTTP proxy. As shown below, the proxy first sets up a listening socket and binds it to the port passed in via the command line. Once an incoming connection takes place the main thread of the program with dispatch the work to a separate thread using the Posix thread library. The main thread passes the clients file descriptor or socket in this case to the new pthread, and the thread goes off and does the work.

In the thread, the HTTP request is read from the TCP receive buffer. Next is is parsed into lines and processed. The first line contains the request command, and it processed to make sure it is either a GET, POST, or HEAD command as per spec. If it passes that check the next line is processed which contains the hostname, which is used in the DNS search to get the ip address of the server. Before the DNS search is ran, the hostname is checked to make sure it is allowed but searching the parental controls file. If allowed then the DNS lookup takes place. If the DNS lookup comes back with a valid ip address, the next step is to create a socket and connect to the server. The proxy then forwards the clients request to the server, and awaits a response. The proxy then forwards all the data coming from the server to the client, which is rendered on the browser shortly after.

The proxy server manages its TCP connections by leaving them open while in the new dispatch thread, however when the request has been served by the proxy the thread closes the socket, and the thread exits. The concurrency is done using POSIX thread implementation in kernel space.
The design is good about error checking, however there are some bugs.

Known issues:

There is a few portability issues with Linux and mac osx. However I have tested the code running on Debian 7 kernel 3.2 and on Ubuntu 3.5 kernel. The code is untested on the Galileo machines in the 301a lab, however it should work on most Linux distributions running kernel 3.2. Debug #define is comment meaning that the build will print debug statements. To turn them off just uncomment the #define NDEBUG statement. Also there is no timer in the threads so sometimes they may be stuck in a read. Also something about my buffer arithmetic isn't perfect so everyone in a while some junk might get printed to stdout. Logging is commented out due to seg faults, something wasn't working the way I thought it would so that will have to wait for phase 2.

Any questions contact Michael Baptist. mbaptist@ucsc.edu.
Code: https://github.com/mbaptist23/basicHTTPproxy

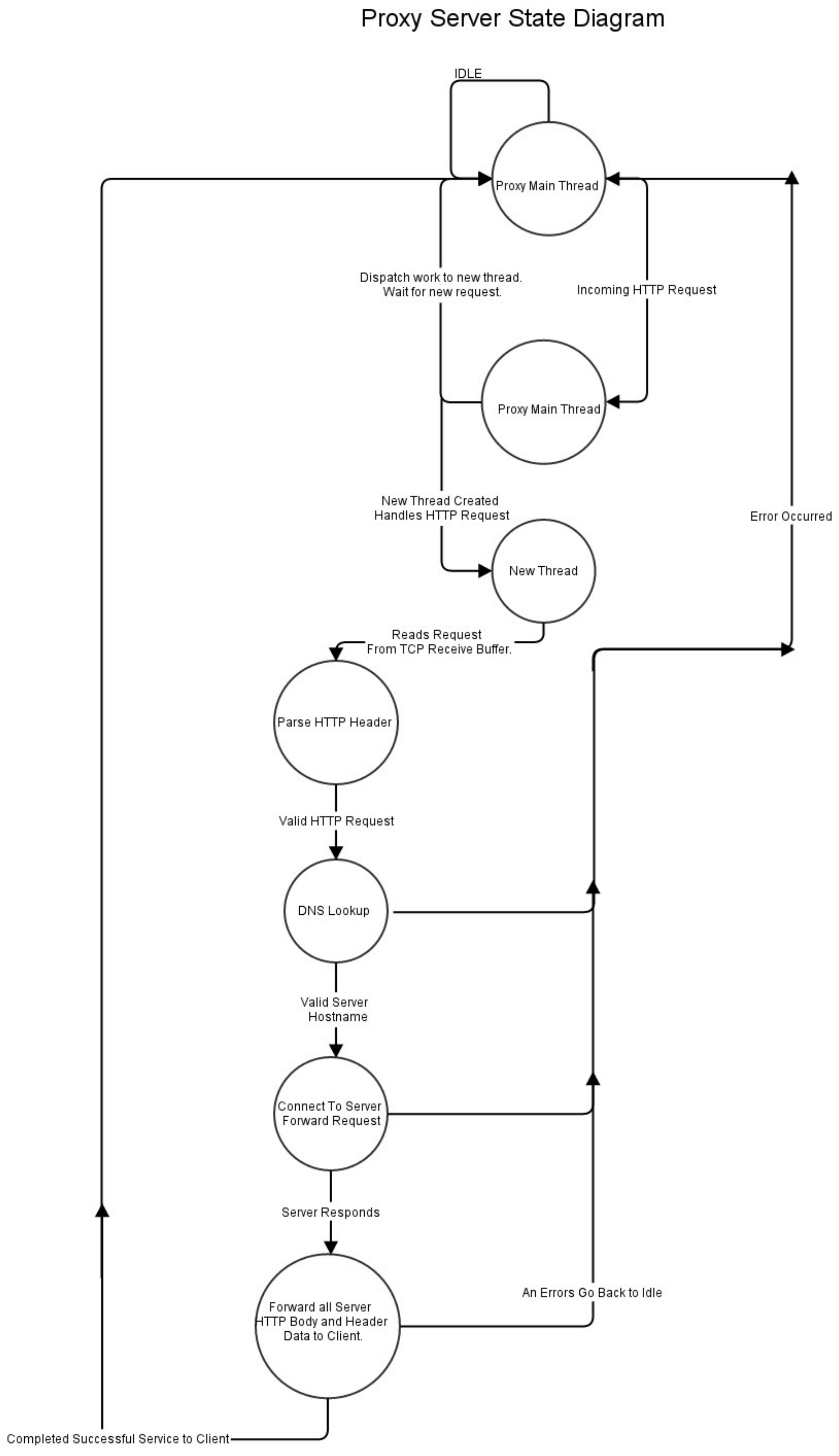
Proxy Server State Diagram
IDLE
Proxy Main Thread
Dispatch work to new thread. Wait for new request.
Incoming HTTP Request
Proxy Main Thread
New Thread Created Handles HTTP Request
Error Occurred
New Thread
Reads Request From TCP Receive Buffer.
Parse HTTP Header
Valid HTTP Request
DNS Lookup
Valid Server Hostname
Connect To Server Forward Request
Server Responds
Forward all Server HTTP Body and Header Data to Client.
An Errors Go Back to Idle
Completed Successful Service to Client

Figure 1. Proxy State Diagram.

Phase 2 Additions:

- Fixed slow loading on sites that request outside of “parental controls”. The problem was that is was incorrectly writing back HTTP responses. I have created an HTTP response module which currently only has 403, 501, and 500 implemented.
- Fixed error handling on forbidden sites as well as HTTP commands that are not allowed.
- Updated the Doxygen document to include the new HTTP response module.
- Added thread safe OpenSSL functions from the code provided on the forum. The code is left untouched and includes the original copyright. The code is not mine, however I am using it for this project.
- Implemented OpenSSL version of phase 1. I referred to the HP documentation. One thing I'd like to note is that there is no valid certificate validation, therefore is it a skeleton for an OpenSSL client, and is not truly secure in anyway.

OpenSSL Additions:

1. In the main thread the program now sets up the SSL libraries and algoritms.
    1. SSL_library_init();
    2. SLL_load_error_strings();
    3. ERR_load_BIO_strings();
    4. OpenSSL_add_all_algorithms();
    5. thread_setup;
2. Once in a new thread to handle a client request, the if the proxy verifies that the request is valid it sets up the SSL context.
    1. CCL_CTX *ctx = SSL_CTX_new(SSLv23_client_method());
    2. SSL *ssl = SSL_new(ctx);
    3. BIO *bio = BIO_new_socket(server_socket, BIO_NOCLOSE);
    4. SSL_set_bio(ssl, bio, bio);

3. Now the SSL connection is valid and the client can use BIO_read or BIO_write when communicating with the server, and the normal send/recv to communicate with the client unsecurly.

4. SSL_shutdown is called when the thread is about to exit, and all other sockets are closed that were created in that thread.