

# Setting up a HTTP Proxy Server

## Assignment - 3

C Gautam - CS17B047, Suhas Pai - CS17B116

### 1 Implementing the server

Socket programming is used extensively in the set-up of the HTTP server. Initially, a TCP socket is spawned, which acts as the proxy server. It is made to listen to incoming requests, on a port, which is taken in as a command line argument. Appropriate checks are made to ensure that the server has been set up correctly, and in case of failures, error messages are printed correspondingly. Upon receiving a request, the program forks a child, and calls a handler to handle that particular request. This ensures that multiple requests can be handled at the same time, parallelly. The handler reads the request from the buffer, till it finds the appropriate delimiter.

```
char request_msg[1024] = {0};
char * cur_msg = (char*)malloc(1024);
while(strstr(request_msg, "\r\n\r\n") == NULL)
{
    read( client_socket , cur_msg, 1024);
    strcat(request_msg,cur_msg);
}
```

Figure 1: Reading from the buffer, till the request contains the delimiter.

The request message is then parsed using the parser function from the proxy library. Bad requests are handled with appropriate error messages.

```
if (ParsedRequest_parse(req, request_msg, len) < 0) {
    printf("parse failed - %s \n ", request_msg);
    return ;
}
```

Figure 2: These errors are usually caused by invalid requests, that are not supported.

If a port isn't specified in the request, port 80 is set as a default. Using the request message, a new request message is made, combining appropriate parameters.

```

ParsedHeader_set(req,"Host",req->host);
ParsedHeader_set(req, "Connection", "close");
int tot = ParsedHeader_headersLen(req);
char *res = (char*)malloc(tot+1);
ParsedRequest_unparse_headers(req,res,tot);

char * final_msg = (char*)malloc(1024);
strcpy(final_msg,"");
strcpy(final_msg,req->method);
strcat(final_msg," ");
strcat(final_msg,req->path);
strcat(final_msg," ");
strcat(final_msg,req->version);
strcat(final_msg,"\r\n");
strcat(final_msg,res);
return final_msg;

```

Figure 3: This function generates the message that is sent to the server, from the proxy server.

A new TCP socket is created, to send this request message to the server. The domain name in the request is looked up, and the request is sent to the translated ip address, and the port.

```

char *IPbuffer;
struct hostent *host_entry;
host_entry = gethostbyname(host name);
IPbuffer = inet_ntoa(*(struct in_addr*)
    host_entry->h_addr_list[0]));

int sock = 0;
struct sockaddr_in serv_addr;

if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
{
    printf("\n Socket creation error \n");
    return ;
}
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = inet_addr(IPbuffer);
serv_addr.sin_port = htons(atoi(req->port));

```

Figure 4: Domain name lookup, and client socket setup.

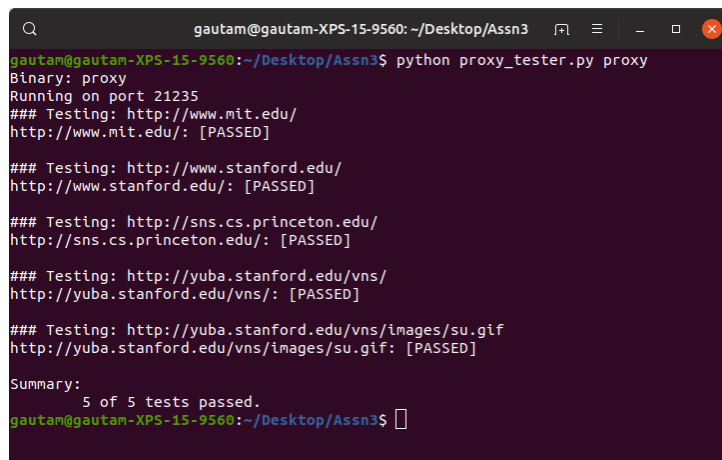
The program waits till the server responds, and then reads the response message, till the delimiter. It then sends this response to the client, and closes all open sockets.

## 2 Observations

The proxy server was tested on the two testing files("proxy\_tester.py" and "proxy\_tester\_conc.py").

### 2.1 Sequential

The first test file sends sequential requests to the proxy server. That is, the second request is sent only after the first response is received by the client. Only the second test case had an issue. The target address was "HTTP://WWW.stanford.edu". This was not passing as there was a difference in a login cookie. This was because the client was implicitly changing the destination address to lowercase. Thus for the comparison to remain fair the target address was changed to "http://www.stanford.edu". After this change, all the test cases passed.

A terminal window with a dark purple background and light green text. The window title is "gautam@gautam-XPS-15-9560: ~/Desktop/Assn3". The command "python proxy\_tester.py proxy" has been executed. The output shows five test cases, each with a "### Testing:" line and a "[PASSED]" result. The test cases are for http://www.mit.edu/, http://www.stanford.edu/, http://sns.cs.princeton.edu/, http://yuba.stanford.edu/vns/, and http://yuba.stanford.edu/vns/images/su.gif. A summary line at the bottom states "5 of 5 tests passed." followed by the prompt "gautam@gautam-XPS-15-9560:~/Desktop/Assn3\$".

```
gautam@gautam-XPS-15-9560:~/Desktop/Assn3$ python proxy_tester.py proxy
Binary: proxy
Running on port 21235
### Testing: http://www.mit.edu/
http://www.mit.edu/: [PASSED]

### Testing: http://www.stanford.edu/
http://www.stanford.edu/: [PASSED]

### Testing: http://sns.cs.princeton.edu/
http://sns.cs.princeton.edu/: [PASSED]

### Testing: http://yuba.stanford.edu/vns/
http://yuba.stanford.edu/vns/: [PASSED]

### Testing: http://yuba.stanford.edu/vns/images/su.gif
http://yuba.stanford.edu/vns/images/su.gif: [PASSED]

Summary:
5 of 5 tests passed.
gautam@gautam-XPS-15-9560:~/Desktop/Assn3$
```

Figure 5: Sequential Test Case

### 2.2 Concurrent

The second test file used different types of concurrent requests. Sometimes the message sent was split into parts. The proxy server handled these cases. In the **Apache** server test cases it was observed that the average time per request increases as the number of concurrent users increases. All the 11 test cases passed.

```
gautam@gautam-XPS-15-9560: ~/Desktop/Assn3
Connection Times (ms)
min mean[+/-sd] median max
Connect: 0 0 0.2 0 1
Processing: 493 604 195.4 519 1482
Waiting: 493 603 195.3 518 1482
Total: 494 604 195.4 519 1483

Percentage of the requests served within a certain time (ms)
50% 519
66% 525
75% 538
80% 627
90% 875
95% 1098
98% 1139
99% 1433
100% 1483 (longest request)
http://sns.cs.princeton.edu/ with args -n 1000 -c 50: [PASSED]

Summary:
Type multi-process: 11 of 11 tests passed.
```

Figure 6: Concurrent Test Case

### 3 Learning Outcomes

This assignment was really useful because we understood how proxy servers worked. We all have encountered blocked websites on the LAN, which probably is implemented by the proxy server set up on the network. So it was a good learning experience, since the proxy server is so ubiquitous.

Another really nice aspect was that we actually understood how the http requests are formatted, and what the various parameters in the requests represent.

We also became more familiar with socket programming, and handling requests from multiple clients at the same time.

Overall, this assignment was a very enjoyable experience, and it has encouraged us to learn more about how these servers are implemented on real systems, and how to scale them to handle millions of users at the same time.