Project 2 Report

CS425: Computer Networks

Date: 31 AUG, 2016

Goutham Reddy Konduru

Roll No: 13334

Email: greddy@iitk.ac.in

Functionality implemented:

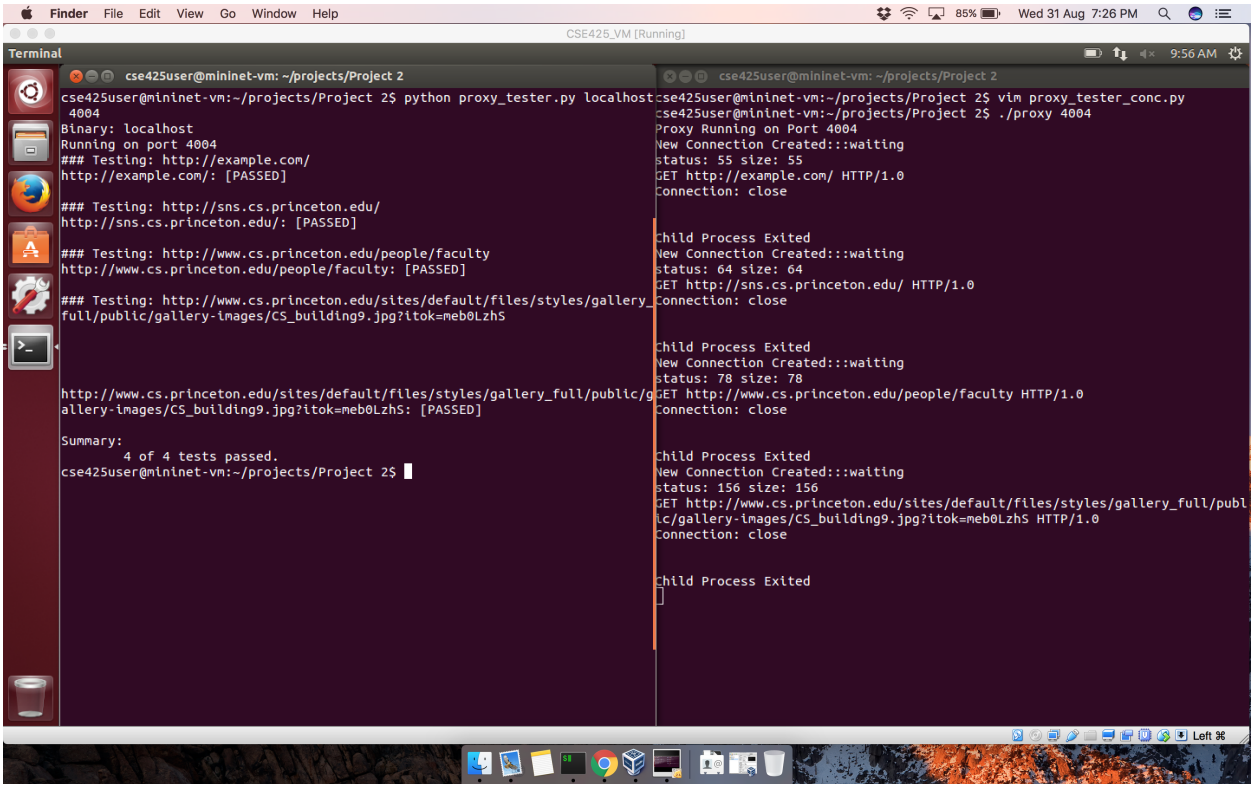
1. Parsing the request
2. Sending the request in required format to server
3. Receiving response from server and sending it to client
4. Connection: close in every request

Design Choices:

I implemented a simple/basic design where proxy receives the requests, parses it sends the request to server in the modified format, and sends back the response from the server to the client.

Test Results:

Scripts checking so many features:



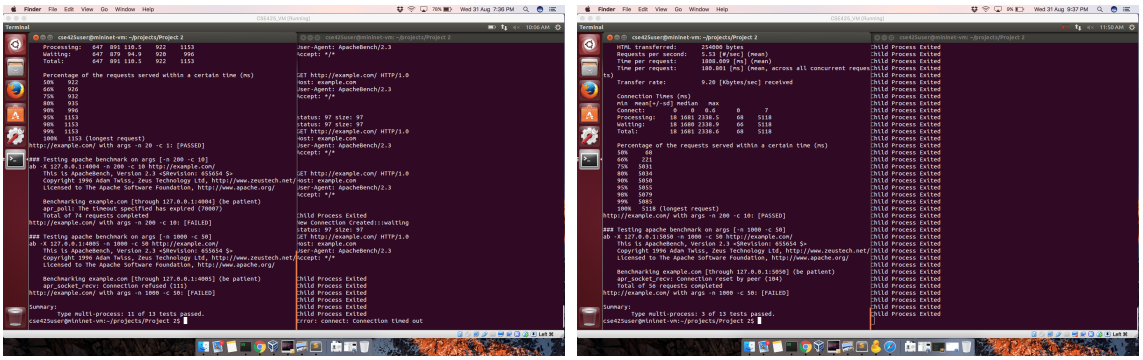


Figure 0.1: proxy_tester_conc.py

In the above screenshots we can observe that proxy_tester.py passes all cases but only after adding time.sleep(2) above read_all line in the http_exchange function. When checking proxy_tester_conc.py using fortinet gateway, we get 11 of 13 pass cases and the last two cases are being failed due to timeout. While checking in the CSE department building, where there is ironport gateway, we get only 5 of 13 passes. Here last two cases are passed but in the others the direct and proxy response differs by a single header value (Age). And there was a lot of abnormal behaviour like sometimes 200, 2000 requests are not gone before the timeout due to network problem.

Checking on Mozilla Firefox and Safari:

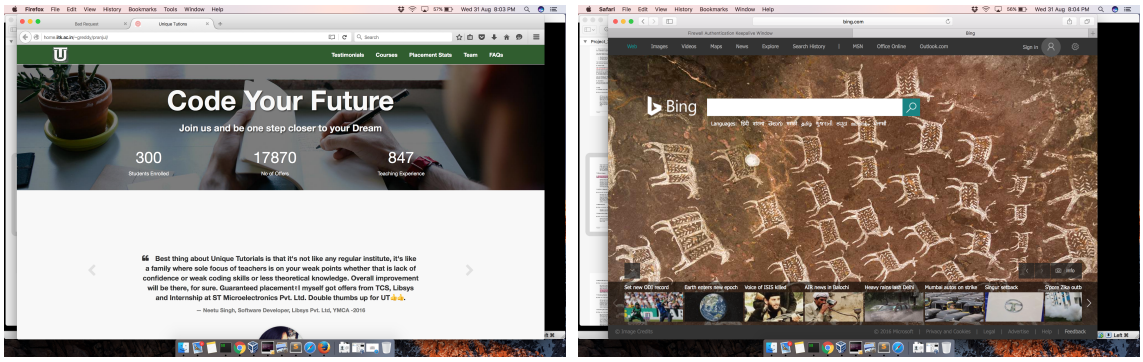


Figure 0.2: Firefox

In the above screenshots we can observe that sites are working fine on firefox and safari. But the thing is only http sites can be loaded and not https as we cannot parse the https request.

Source Code:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/uio.h>
#include <sys/wait.h>
#include <time.h>

#include "proxy_parse.h"


#define MAX_PENDING 10
#define MAX_SIZE    4096

void response_500(int sockid){
        char buffer[4096];

        strcpy(buffer, "HTTP/1.0 500 Internal Server Error\r\n")
        write(sockid, buffer, strlen(buffer));
        strcpy(buffer, "Content-length: 117\r\n");
        write(sockid, buffer, strlen(buffer));
        strcpy(buffer, "Connection: close\r\n");
        write(sockid, buffer, strlen(buffer));
        strcpy(buffer, "Content-Type: text/html\r\n\r\n");
        write(sockid, buffer, strlen(buffer));
        strcpy(buffer, "<html>\n<head>\n<title>Internal Server E
        write(sockid, buffer, strlen(buffer));
        strcpy(buffer, "<body>\n<p>500 Internal Server Error</p>
```

```
        write(sockid, buffer, strlen(buffer));
}

int main(int argc, char * argv[]) {
        int sockid, new_sockid;
        struct sockaddr_in sin;
        int status;
        pid_t  pid;
        char send_buf[MAX_SIZE], recv1_buf[MAX_SIZE];
        char* recv_buf = (char *) malloc(4096);
        socklen_t len;
        int number_active = 0;
        int max_active = 40;



        /*--- Checking number of arguments ---*/
        if(argc!=2){
                fprintf(stderr, "Run using ./proxy PORT\n");
                return 1;
        }

        /*--- Setting values of sockaddr_in ---*/
        bzero((char *)&sin, sizeof(sin));
        bzero((char *)send_buf, MAX_SIZE);
        bzero((char *)recv_buf, MAX_SIZE);
        sin.sin_family = AF_INET;
        sin.sin_port = htons(atoi(argv[1]));
        sin.sin_addr.s_addr = htons(INADDR_ANY);

        /*--- Creating a socket ---*/
        if((sockid = socket(AF_INET, SOCK_STREAM, 0)) < 0){
                perror("Error: socket");
                return 1;
        }

        /*--- Reusing same port ---*/
```

```
    int enable = 1;
if (setsockopt(sockid, SOL_SOCKET, SO_REUSEADDR, &enable, si
    perror("setsockopt(SO_REUSEADDR) failed");

    /*--- Binding socket with ip and port ---*/
    if(bind(sockid, (struct sockaddr *)&sin, sizeof(sin)) <
            perror("Error: bind");
            return 1;
    }

    /*--- Socket listening for a connection ---*/
    if((status = listen(sockid, MAX_PENDING)) != 0){
            perror("Error: listen");
            return 1;
    }

    printf("Proxy Running on Port %d\n", atoi(argv[1]));

    while(1)
    {
            /*--- Waiting to accept connection from a client
            if((new_sockid = accept(sockid, (struct sockaddr
            {
                    perror("Error: accept");
                    continue;
            }

            pid_t w_pid;
            int number;
            printf("%d %d\n", number, number_active);

            for (; number_active >= max_active; number_activ

            w_pid = wait(&number);
            printf("%d\n", (int)w_pid);
            }
```

```
pid = fork();
number_active++;
if(pid == 0)
{
        // close(sockid);

struct timeval timeout;
        timeout.tv_sec = 5;
        timeout.tv_usec = 0;
        setsockopt(new_sockid, SOL_SOCKET, SO_RC

        printf("New Connection Created:::waiting
        bzero((char *)recv_buf, MAX_SIZE);

        status = 0;
        while(!strstr(recv_buf,"\r\n\r\n")){
                bzero((char *)recv1_buf, MAX_SIZ
                status = recv(new_sockid, recv1_
                if(strlen(recv_buf) + status > 4
                        recv_buf = (char *) real
                strcat(recv_buf, recv1_buf);
        }

        if(status < 0)
        {
                perror("Error: recv\n");
                continue;
        }
        else if(status == 0)
        {
                printf("Client Disconnected\n");
                continue;
        }

        if(strlen(recv_buf) >= 8192){
                response_500(new_sockid);
                printf("ParsedRequest_parse faile
```

```
                close(new_sockid);
                printf("Connection Closed\n");
                exit(0);
        }
        // strcat(recv_buf, "\r\n");

        /*--- Parsing ---*/
        struct ParsedRequest * parsed_req;
        parsed_req = ParsedRequest_create();
        int ret = ParsedRequest_parse(parsed_req
        if (ret == -1)
        {
                response_500(new_sockid);
                printf("ParsedRequest_parse faile
                close(new_sockid);
                printf("Connection Closed\n");
                exit(0);
        }

        if (parsed_req->port == NULL)
                parsed_req->port = (char*)"80";

        /*--- Making request to server ---*/
        struct sockaddr_in sin1;
        int sockid1;
        /*--- Setting values of sockaddr_in ---*
        bzero((char *)&sin1, sizeof(sin1));
        sin1.sin_family = AF_INET;
        sin1.sin_port = htons(atoi(parsed_req->p
        struct hostent *host_1 = gethostbyname(p
        bcopy((char *)host_1->h_addr, (char *)&s
        /*--- Creating a socket to connect to se
        if ((sockid1 = socket(AF_INET, SOCK_STREA
                perror("Error: socket");
                response_500(new_sockid);
                close(new_sockid);
                printf("Connection Closed\n");
```

```
        exit(0);
}
/*--- Connecting to Server Socket ---*/
if (connect(sockid1, (struct sockaddr *)&
        perror("Error: connect");
        close(sockid1);
        response_500(new_sockid);
        close(new_sockid);
        printf("Connection Closed\n");
        exit(0);
}

/*--- send_buf to send to server ---*/
bzero((char *)send_buf, 8096);
snprintf(send_buf, 8096, "GET %s HTTP/1.
bzero((char *)recv_buf, MAX_SIZE);
sprintf(recv_buf, "Host: %s\r\n", parsed
strcat(send_buf, recv_buf);
for (unsigned int i = 0; i < parsed_req-
{
        bzero((char *)recv_buf, MAX_SIZE
        if(!strcmp(parsed_req->headers[i
                continue;
        sprintf(recv_buf, "%s: %s\r\n",
        strcat(send_buf, recv_buf);
}
strcat(send_buf, "Connection: close\r\n\
send(sockid1, send_buf, strlen(send_buf)

/*--- Receiving and Sending response ---
while(1)
{
        bzero((char *)recv_buf, MAX_SIZE
        int length = recv(sockid1, recv_
        if (length <= 0) break;
        send(new_sockid, recv_buf, length
}
```

```
                        close(sockid1);
                        close(new_sockid);
                        printf("Child Process Exited\n");
                        exit(0);
                }
        close(new_sockid);
        }

  return 0;
}
```