CS425: Computer Networks

Project-2: Http Proxy

Shubham Agrawal
13674, agshubh191@gmail.com

August 31, 2016

1. Features Implemented

- Mandatory Features:
    1. Proxy server which receives request from client, modifies request, send to server, receive response, modifies response and send response back to client.
    2. Proxy server port can be initialized from command line
    3. Proxy server responses to GET request and send 500, Internal Error on another methods
    4. Proxy server requires client to send request in absolute uri form
    5. Proxy server supports HTTP/1.0 Protocol and send 500, Internal Error on other protocols
    6. Concurrent requests are handled using forked chilren. Maximum number of forked children is 20. After 20, maximum 100 clients can wait for children to exit.
- Optional Features
    1. Proxy server handles request which are not send completely at a time by receiving until CRLF is received.
    2. Proxy server handles responses of any size.

2. Testing Results (Browser Used: Firefox)

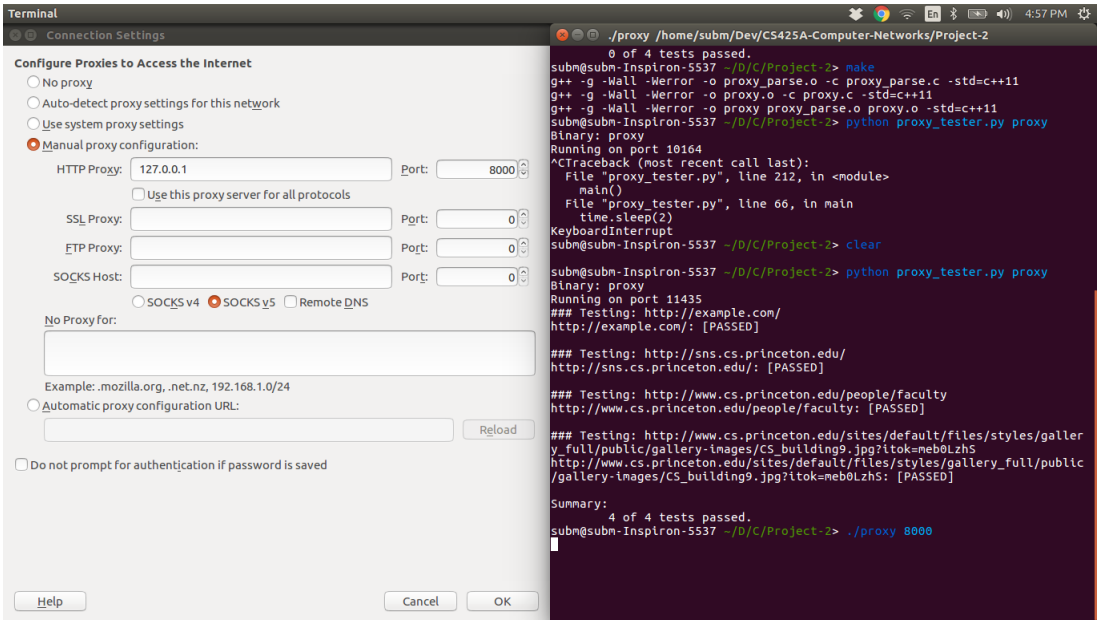

Figure 1: Firefox proxy settings

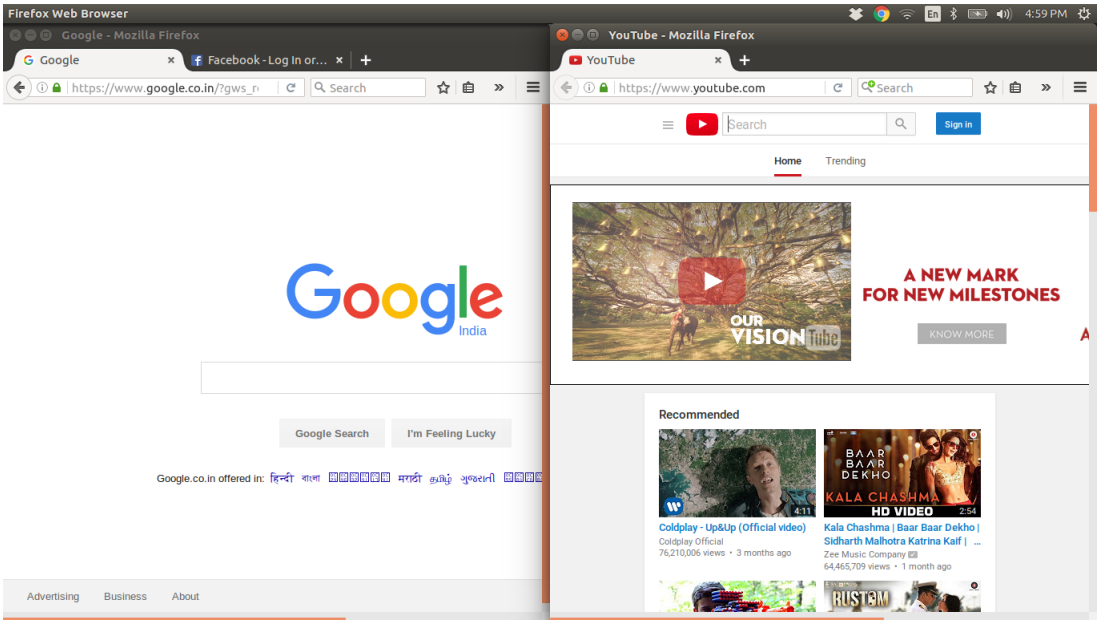

Figure 2: Multiple page request working on firefox with proxy

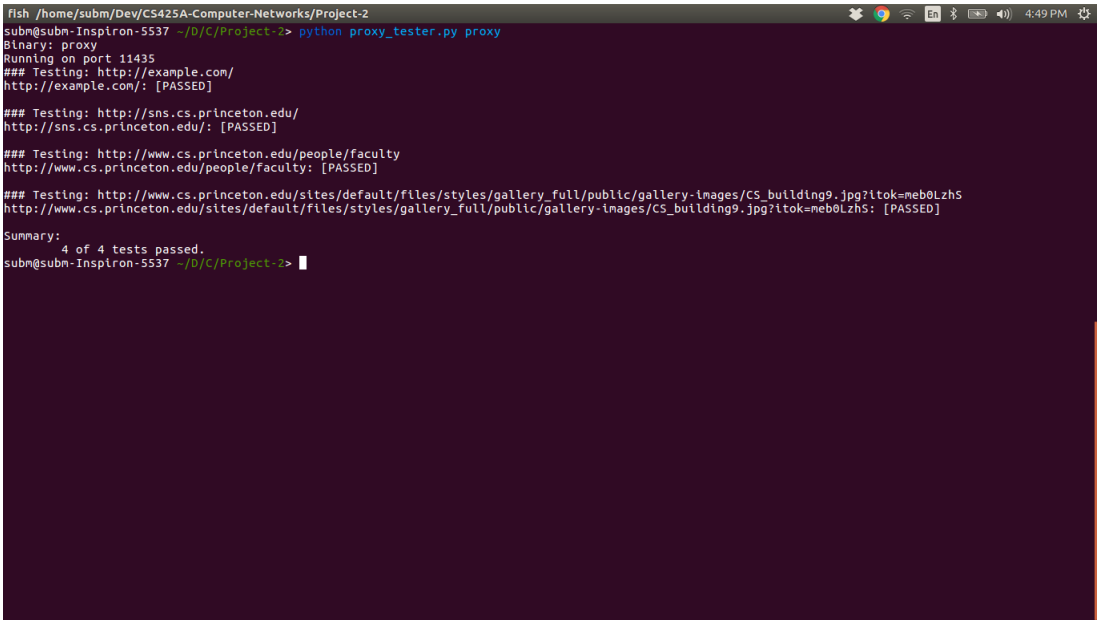

Figure 3: Results on *proxy_tester.py*

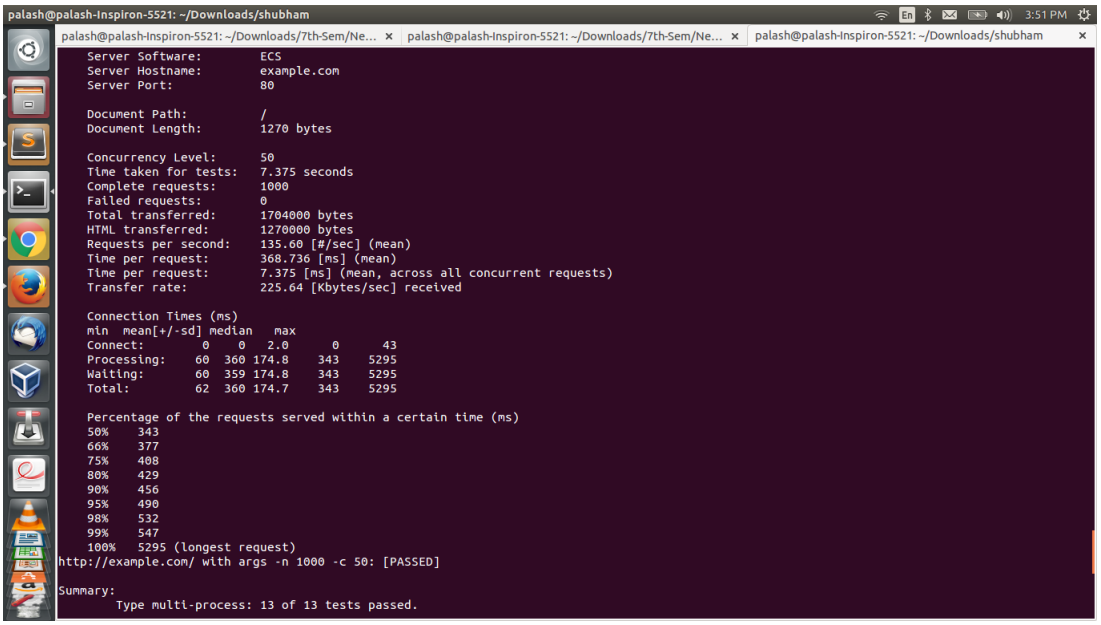

Figure 4: Results on *proxy_tester_conc.py*

3. Summary

- All the mentioned features seems to work
- All the test-cases mentioned pass with changing "read_all" to "read_some" as discussed on slack
- Sometimes test-cases does not pass due to slow internet connection

4. Appendix

4.1 Code *proxy.c*

```
#include "proxy_parse.h"
#include<iostream>
#include<stdio.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<unistd.h>
#include<string.h>
#include<iostream>
#include<string>
#include<csignal>
#include<netdb.h>
#include<unistd.h>
#include<sys/types.h>
using namespace std;

#define maxClientsWaiting 100  // Max number of waiting clients
#define bufferSize 9000
#define smallBufferSize 1000
#define maxForkedChilds 20
char proxyRequestFormat[] = "GET %s HTTP/1.0\r\n%s";
char defaultProxyPort[] = "80\0";
int numForkedChild = 0;
char errResponseFormat[] = "HTTP/1.0 %d %s\r\n\r\n";

struct status{
        int status;
        char msg[1000];
};
typedef struct status status;
```

```
status S500 = {500, "Internal Error\0"};

// To get html for response error
void sendError(int sockid, status resStatus){
        char buffer[smallBufferSize];
        bzero(buffer, smallBufferSize);
        snprintf(buffer, smallBufferSize, errResponseFormat, resStatus.status, resS
        send(sockid, buffer, strlen(buffer),0);
        return;
}

//To update number of forked children
void updateNumForkedChild(int update){
        //TODO: Make it shared and race free in parent and child
        numForkedChild = numForkedChild + update;
        return;
}

int hostname_to_ip(char * hostname , char* ip)
{
    struct hostent *he;
    struct in_addr **addr_list;
    int i;

    if ( (he = gethostbyname( hostname ) ) == NULL)
    {
        // get the host info
        //herror("gethostbyname");
                //printf("get host by name error");
        return 1;
    }

    addr_list = (struct in_addr **) he->h_addr_list;

    for(i = 0; addr_list[i] != NULL; i++)
    {
        //Return the first one;
        strcpy(ip , inet_ntoa(*addr_list[i]) );
        return 0;
    }
    //printf("%d\n", i) ;
    return 1;
}
```

```
int getProxyRequest(ParsedRequest* req, char* buffer){
        //TODO: Fill headers

        // Set a specific header (key) to a value. In this case,
        //we set the "Last-Modified" key to be set to have as
        //value  a date in February 2014

        if(ParsedHeader_set(req, "Connection", "close") < 0){
                //printf("Set header key not work\n");
                return -1;
        }

        if(ParsedHeader_set(req, "Host", req->host) < 0){
                //printf("Set header key not work\n");
                return -1;
        }
        // Turn the headers from the request into a string.
        int rlen = ParsedHeader_headersLen(req);
        char buf[rlen+1];
        if (ParsedRequest_unparse_headers(req, buf, rlen) < 0) {
                //printf("unparse failed\n");
                return -1;
        }
        buf[rlen] ='\0';
        //printf("Header String........\n%s\n.......\n", buf);
        //print out buf for text headers only

        snprintf(buffer, bufferSize, proxyRequestFormat, req->path, buf);
        return 0;
}

int main(int argc, char* argv[]){
        if(argc<2){
                //printf("Usage ./proxy portNumber");
                return 0;
        }
        int port = atoi(argv[1]);
        int sockid = socket(AF_INET,SOCK_STREAM, 0);
        if(sockid<0){
                //printf("Unable to get sockid");
                return 0;
        }
```

```
struct sockaddr_in serverAdd;
serverAdd.sin_family = AF_INET;
serverAdd.sin_port=htons(port);
serverAdd.sin_addr.s_addr=INADDR_ANY;
int bind_status = bind(sockid, (struct sockaddr*)&serverAdd, sizeof(serverA
if(bind_status==0){
        //printf("bind successful\n");
}
else{
        //printf("bind failed\n");
        return 0;
}
int listen_status = listen(sockid, maxClientsWaiting);
if(listen_status==0){
        //printf("Listening ...\n");
}
else{
        //printf("Listening failed\n");
        return 0;
}
// accept connections in loop

while(1){
        struct sockaddr clientAdd;
        socklen_t addrLen;
        int newSockid = accept(sockid, &clientAdd, &addrLen);
        // Keep waiting until
        for (; numForkedChild >= maxForkedChilds; --numForkedChild){
                //printf("Waiting for some(%d) children to exit\n", numFork
                wait();
        }
        if(fork()==0){
                close(sockid); //Child will not listen
                char buffer[bufferSize];
                char bufferTemp[bufferSize];
                bzero(buffer, strlen(buffer));
                int n;
                do{
                        bzero(bufferTemp, strlen(bufferTemp));
                        n = recv(newSockid, bufferTemp, bufferSize, 0);
                        strcat(buffer, bufferTemp);
                }while((strstr(buffer, "\r\n\r\n")==NULL));
                //printf("--------Request Recieved(%d)------\n%s\n---------
```

```
//Parse request
int len = strlen(buffer);
//Create a ParsedRequest to use. This ParsedRequest
//is dynamically allocated.
ParsedRequest *req = ParsedRequest_create();
if (ParsedRequest_parse(req, buffer, len) < 0) {
        //printf("Request parsing failed\n");
        sendError(newSockid, S500);
        return 0;
}
//printf("Request parsing completed\n");

if(strcmp(req->method, "GET") != 0 ){
        //printf("Method %s not implemented\n", req->method
        sendError(newSockid, S500);
        return 0;
}
if(req->port==NULL){
        req->port = (char*)malloc(6);
        bzero(req->port, 6);
        strcpy(req->port, defaultProxyPort);
}

char ip[50];
bzero(ip, 50);
hostname_to_ip(req->host, ip);
int fSockid = socket(AF_INET,SOCK_STREAM, 0);
struct sockaddr_in fServerAdd;
fServerAdd.sin_family = AF_INET;
fServerAdd.sin_port=htons(atoi(req->port));
fServerAdd.sin_addr.s_addr=inet_addr(ip);
int fAddrLen = sizeof(fServerAdd);
int conn_status = connect(fSockid,(struct sockaddr*)&fServer
if(conn_status==0){
        //printf("Connection to host established\n");
}
else{
        //printf("Connection to host failed: %d\n ", conn_s
        sendError(newSockid, S500);
        return 0;
}
char proxyRequest[bufferSize];
```

```
                        bzero(proxyRequest, bufferSize);
                        if(getProxyRequest(req, proxyRequest)!= 0){
                                //printf("Proxy request unable to create\n");
                                sendError(newSockid, S500);
                                return 0;
                        }
                        //printf("--------Proxy sending request------\n%s\n--------
                        n = send(fSockid,proxyRequest,strlen(proxyRequest),0);
                        bzero(buffer, strlen(buffer));
                        int bytes = 0;
                        //printf("--------Response from host recieved------\n");
                        while((n = recv(fSockid, buffer, bufferSize, 0)) != 0){
                                //printf("%s",buffer);
                                // Send to client
                                send(newSockid,buffer,strlen(buffer),0);
                                bytes = bytes+n;
                                bzero(buffer, strlen(buffer));
                        }
                        //printf("\n-----------------\n");
                        close(newSockid);
                        return 0;
                }
                else{
                        updateNumForkedChild(1);
                        close(newSockid);
                }
        }
        return 0;
}
```