Proxy Server

CS425A
Computer Networks

Implemented Features :-

1) Implemented the GET method
2) Received data from the client and forwarded that to the server
3) Send data back from the server to client
4) Implemented the header of Connection: close
5) Implemented handling of multiple clients
6) Implemented the upper bound on the number of forks that we can make
7) Server Port Initialised at start up.

Running The code

$ make
$./proxy <port>
$make clean

Sai Kishan Pampana
13458
pkishan@iitk.ac.in

Design Choices

- Creating the header to send the sever

After parsing the header that we get from the client, we need to send a header to the server so far this purpose instead of using the functions that are present in the parsing code I have manually made the GET request to the server. By manual I don't mean I hard coded but instead taking the parsed input into account I have created the header to send to the server.

- Allow the port to be decided at the startup

For the port that we need to use in the proxy, I have given an input in the beginning of the code as a system argument, so that we can keep changing the port if any of the port is busy.

- Receiving the data from the client

For reading the request from the client instead of reading the whole request in one go i have place a while loop on top of the receive command and came out of the loop when i see "\r\n\r\n".

- Realloc

Used Realloc function in the case that the header is too big for the assigned buffer to hold.

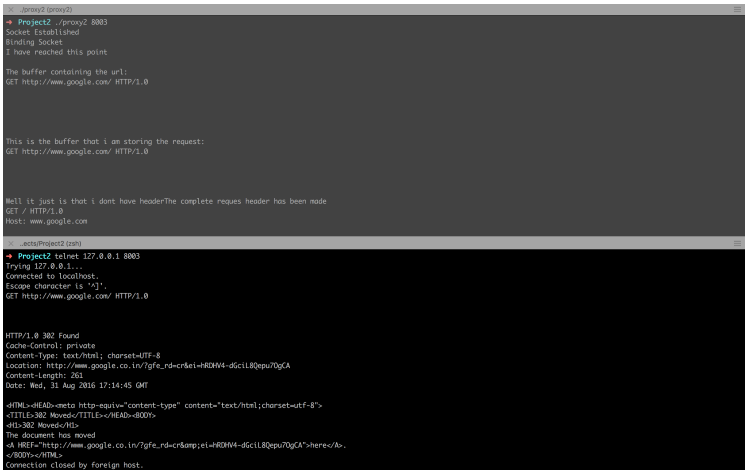
IMPLEMENTATION OF THE PROJECT

Browser Used:

The browser that was used for this project is Mozilla FireFox

Test Procedure Used:

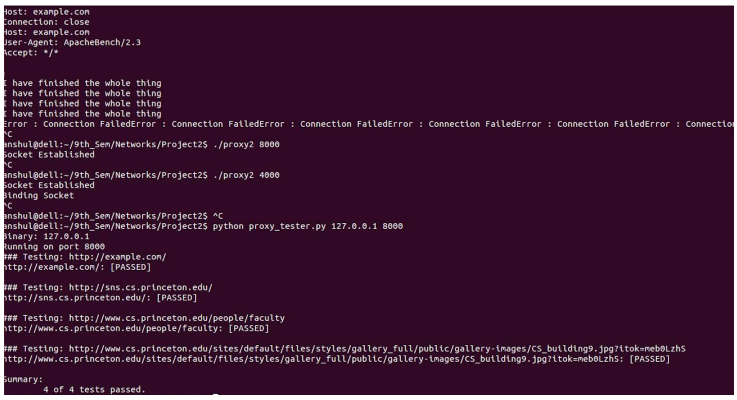
For the testing of the project this was done in different stages. Telnet, running the python scripts, Browser testing

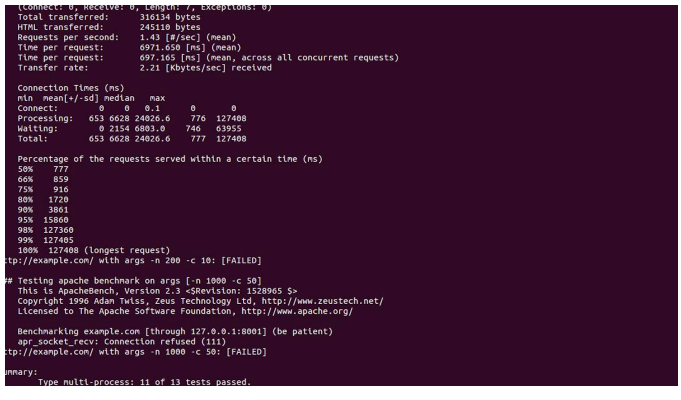


For the case of the telnet we first run the proxy on port and then use telnet to call the site.

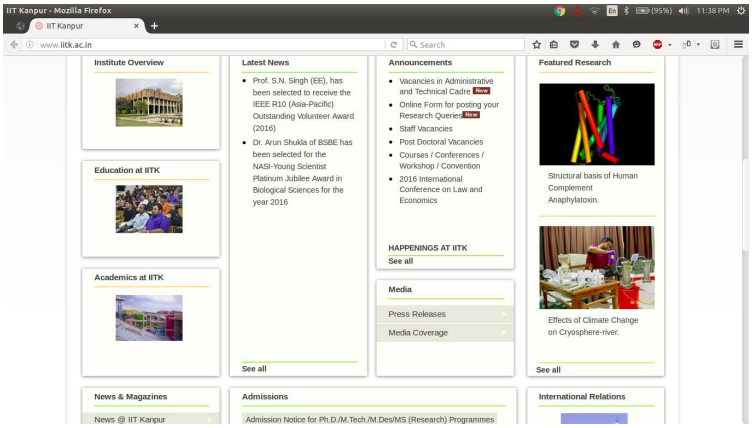
As you can see in the image in the telnet we have called the google.com and as per that the proxy has returned the corresponding data.

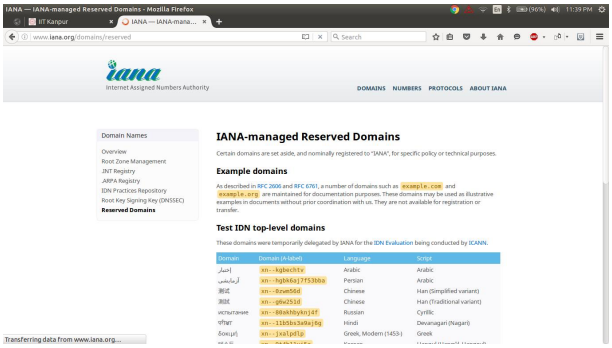
Now in the python script for the we have two python scripts and in one we have 4 test and in another we have 13th test. In the second python script only 11 pass since the net is not fast enough.





In the browser





APPENDIX

```
#include "proxy_parse.h"
#include "proxy_parse.c"
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <pthread.h>
#include <assert.h>
#include <time.h>
#include <netdb.h>
```

```
#include <sys/mman.h>
#include <sys/wait.h>

static int *count;

int main(int argc, char * argv[])
{
	int act_server_socket, actual_client_socket, act_clinet_socket;

	socklen_t addrlen;

	char buf[1024],head[1024];
	struct sockaddr_in client_address, server_address;
	struct hostent *name;

	char request[1024];

	if ((act_server_socket = socket(AF_INET, SOCK_STREAM, 0)) < 0)     //
Creating the server socket which will listen for client requests
	{
	  printf("The socket was not created\n");
	  exit(1);
	}

	printf("Socket Established \n");

	client_address.sin_family = AF_INET;
	client_address.sin_addr.s_addr = INADDR_ANY;
	client_address.sin_port = htons(atoi(argv[1]));   //Setting up the port on which
the server will be listening

	if (bind(act_server_socket, (struct sockaddr *) &client_address,
sizeof(client_address)) == 0)     //Binding the server sockets to the adress that was
defined above
	{
	  printf("Binding Socket\n");
	}
```

```
        if (listen(act_server_socket, 10) < 0)        //Make the server to start listening
to the requests of the clients
        {
          perror("server: listen");
          exit(1);
        }

        count = mmap(NULL, sizeof *count, PROT_READ | PROT_WRITE,
                MAP_SHARED | MAP_ANONYMOUS, -1, 0);
        *count = 0;

        while(1)
        {
                if((actual_client_socket = accept(act_server_socket, (struct sockaddr *)
&client_address, &addrlen)) < 0)
                {
                        perror("Could not create new socket");
                        break;
                }
                pid_t pid;

                if(*count <20)
                {
                        pid = fork();
                }
                else
                {
                        wait(NULL);
                        pid = fork();
                }
                if(pid !=0)
                {
                        close(actual_client_socket);
                        continue;
                }

                *count = *count + 1;

                char  tmp_buf[1024];
```

```
bzero((char *)buf, sizeof(buf));
while(1)
{
        bzero((char *)tmp_buf, sizeof(tmp_buf));
        if(recv(actual_client_socket, tmp_buf, 1024, 0)<= 0 )
        {
                printf("No message is received form the client");
                *count = *count -1;
                exit(1);
        }
        strcat(buf, tmp_buf);
        if(strstr(tmp_buf, "\r\n\r\n") != NULL)
                break;
}

printf("I have reached this point\n");

printf("\nThe buffer containing the url:\n%s\n\n", buf);

struct ParsedRequest *pr;
pr = ParsedRequest_create();       //Here we are just making an struct

int ret = ParsedRequest_parse(pr, buf, 1024);

if(ret != 0)
{
        char resp[1024];
        bzero((char*) resp, sizeof(resp));
        printf("Something went wrong while parsing\n");
          strcat(resp, "HTTP/1.1 500 Internal Server Error\r\n");
     strcat(resp, "Content-Length: 155");
     strcat(resp, "\r\n");
     strcat(resp, "Content-Type: text/html");
     strcat(resp, "\r\n");
     strcat(resp, "Connection: close");
     strcat(resp, "\r\n");
     strcat(resp, "\r\n");
```

```
                strcat(resp, "<html><head><TITLE>500 Internal Server Error</
TITLE></head> \r\n<body><H1>500 Internal Server Error</H1>\r\nThere is some
error on this server.\r\n</body></html>");

                        write(actual_client_socket,resp,strlen(resp));
                        *count = *count - 1;
                        exit(1);
                }

        bzero((char *)request, sizeof(request));

                strcat(request, pr->method);
                strcat(request, " ");
                strcat(request, pr->path);
                strcat(request, " ");
                strcat(request, pr->version);
                strcat(request, "\r\n");
                strcat(request, "Host: ");
                strcat(request, pr->host);
                strcat(request, "\r\n");
                strcat(request, "Connection: close\r\n");

                if(pr->headers[0].key == NULL)
                {
                        printf("Well it just is that i dont have header");
                }

                else
                {       printf("Well what do you know i have headers\n");
                        int list = 0;
                        while(1)
                        {
                                if(pr->headers[list].key == NULL)
                                        break;
                                if((strcmp(pr->headers[list].key, "Connection")!=0))
                                {
                                        strcat(request, pr->headers[list].key);
                                        strcat(request, ": ");
                                        strcat(request, pr->headers[list].value );
```

```
                        strcat(request, "\r\n");
                    }
                    list = list + 1;
                }
            }
            strcat(request, "\r\n");
            printf("The complete reques header has been made\n");
            printf("%s\n", request);

            act_clinet_socket=socket(AF_INET,SOCK_STREAM,0);
            server_address.sin_family = AF_INET;
        server_address.sin_port = htons(80);

        name = gethostbyname(pr->host);

        bcopy((char *)name->h_addr, (char *)&server_address.sin_addr.s_addr,
name->h_length);


             if(connect(act_clinet_socket,(struct
sockaddr*)&server_address,sizeof(struct sockaddr)))
            {
                printf("Error : Connection Failed");
                *count = *count -1;
                exit(1);
            }
            int b;
            int a;
            int c;
            char buff[4096];


            c = write(act_clinet_socket,request,1024);
            while(1)
            {
                bzero((char *)buff, sizeof(buff));
                b = recv(act_clinet_socket, buff, 4096, 0);

                if( b<=0)
```

```
                {
                        close(act_clinet_socket);
                        break;
                }
                a = write(actual_client_socket,buff,b);
        }


        printf("I have finished the whole thing\n");

        close(actual_client_socket);
        *count = *count -1;
        exit(0);
    }
    return 0;
}
```