# Proxy Lab

---

## 0. Introduction

코드에 주석을 자세히 달았으니 주석을 참고부탁드립니다.

A web proxy is a program that acts as a middleman between a Web browser and an end server. Instead of contacting the end server directly to get a Web page, the browser contacts the proxy, which forwards the request on to the end server. When the end server replies to the proxy, the proxy sends the reply on to the browser. This lab requires us to write a simple HTTP proxy that caches web objects.

csapp.h codes are included

/*Sockets interface wrappers */

int Socket(int domain, int type, int protocol);

int Setsockopt(int s, int level, int optname, const void *optval, int optlen);

void Bind(int sockfd, struct sockaddr *my_addr, int addrlen);

void Listen(int s, int backlog);

void Accept(int s, struct sockaddr *addr, socklen_t *addrlen);

void Connect(int sockfd, struct sockaddr *serv_addr, int addrlen);

/*Protocol independent wrappers*/

void Getaddrinfo(const char *node, const char *service, const struct addrinfo *hints, struct Daddrinfo **res);

void Getnameinfo(const struct sockaddr *sa, socklen_t salen, char *host, size_t hostlen, char *serv, size_t servlen, int flags);

void Freeaddrinfo(struct addrinfo *res);

void Inet_ntop(int af, const void *src, char *dst, socklen_t size);

void Inet_pton(int af, const char *src, void *dst);

Background Study

string.h functions

char *strstr(const char *str1, const char *str2); // str1 검색 대상 문자열, str2 찾는 문자열

//return: 찾고자 하는 문자열이 발견된 첫번째 위치의 포인터

Robust I/O package

Rio_writen(file descriptor, data buffer, buffer size)

write in buffer

stdio.h functions

int sscanf(const char *str, const char* format, …);

str에서 데이터를 형식 문자열에서 지정하는 바에 따라 읽어와 그 데이터를 부수적인 인자들이 가리키는 메모리 공간에 저장하게 된다.

<http://blog.naver.com/PostView.nhn?blogId=qbxlvnf11&logNo=221052344667&categoryNo=50&parentCategoryNo=0&viewDate=&currentPage=1&postListTopCurrentPage=1&from=postView&userTopListOpen=true&userTopListCount=5&userTopListManageOpen=false&userTopListCurrentPage=1 , https://studyc.tistory.com/12 , https://studyc.tistory.com/14?category=609946 , https://dojang.io/mod/page/view.php?id=294 참고>

‘/0’ is called null character. 문자열 끝에 자동적으로 붙는 특수문자. 널 문자로 인해 문자열 길이도 자동적으로 +1이 된다. 이를 문자의 형태로 출력할 경우 아무 출력도 발생하지 않는다. 널 문자의 아스키 코드는 0이다.

널 문자의 부재로 인해 발생할 수 있는 문제점을 예방하기 위해 문자열 배열을 사용할 때는 처음에 널 문자로 초기화를 하고 사용하는 것이 현명한 방법이다.

char str[10] = {0, };

//정상적으로 0을 넣고 그 이후에는 0으로 초기화 // 배열의 요소를 모두 0으로 초기화

char str[10] =‘\0’;

char str[10] =NULL;

**your proxy must ignore SIGPIPE signals and should deal gracefully with write operations that return EPIPE errors**

If a server writes to a connection that has already been closed by the client, the first write returns normally, but the second write causes the delivery of a SIGPIPE signal whose default behavior is to terminate the process. If the SIGPIPE signal is caught or ignored, then the second write operation returns -1 with error set to EPIPE. The strerr and perror functions report the EPIPE error as a Broken pipe.

The bottom line is that a robust server must catch these SIGPIPE signals and check write function calls for EPIPE errors.

**RFX 1945 ** complete specification for the HTTP/1.0 protocol

```
http_URL        = "http:" "//" host [ ":" port ] [ abs_path ]
host            = <A legal Internet host domain name
                          or IP address (in dotted-decimal form),
                          as defined by Section 2.1 of RFC 1123>
port            = *DIGIT
```

```
The first digit of the Status-Code defines the class of response.
The last two digits do not have any categorization role. There are 5
values for the first digit:

      o 1xx: Informational - Not used, but reserved for future use

      o 2xx: Success - The action was successfully received,
             understood, and accepted.

      o 3xx: Redirection - Further action must be taken in order to
             complete the request

      o 4xx: Client Error - The request contains bad syntax or
cannot
             be fulfilled

      o 5xx: Server Error - The server failed to fulfill an
apparently
             valid request
```

---

## 1. Implementing a sequential web proxy

This first part involves learning about basic HTTP operation and how to use sockets to write programs that communicate over network connections. It implements a basic sequential proxy that handles HTTP/1.0GET requests.

When started, the proxy listens for incoming connections on a port whose number will be specified on the command line.

```
int listenfd = Open_listenfd(argv[1]);
```

Once a connection is established,

```
int clientfd =
Accept(listenfd, (SA *)&clientaddr, &clientlen);
```

the proxy reads the entirety of the request from the client and parse the request.

```
Rio_readinitb(&rio_to_client, clientfd);
char request[MAXLINE];
```

```
parse_request(&rio_to_client, request, &req);
```

It determines whether the client has sent a valid HTTP request. If so, it establishes its own connection to the appropriate web server and request the object the client specified.

```
int serverfd =Open_clientfd(req.host_addr, req.port);
```

Finally, it reads the server's response and forward it to the client.

```
from_client_to_server(&rio_to_client,
                          &req, serverfd, clientfd);
```

1) HTTP/1.0GET requests

Notice that the web browser's request ends with HTTP/1.1, while the proxy's request line ends with HTTP/1.0. Modern web browsers will generate HTTP/1.1 requests, but my proxy handle them and forward them as HTTP/1.0 requests.

```
static char *server_version = "HTTP/1.0";
//in from_client_to_server
n = snprintf(req_final, MAXLINE, "%s %s %s\r\n",
                     req->method,
                     *req->path ? req->path : "/",
                     server_version);
```

2) Request headers

Request header for this lab are Host, User-Agent, Connection, and Proxy-Connection headers.

Host: describe the hostname of the end server. If web browser attach their own Host header to HTTP requests, proxy use the same host header as the browser.

User-Agent: identifies the client ( in terms of parameters such as the operating system and browser), and web servers.

Connection: close

Proxy-Connection: close

```
static char *user_agent_hdr = "User-Agent: Mozilla/5.0
(X11; " \
"Linux x86_64; rv:10.0.3) Gecko/20120305 Firefox/
10.0.3\r\n";
static char *connection_hdr = "Connection: close\r\n";
static char *proxy_connection_hdr = "Proxy-Connection:
close\r\n";
```

If browser send any additional request headers as part of an HTTP request, proxy forwards them unchanged.

```
//in from_client_to_server
```

```
else { //other header, just send it to the server

Rio_writen(serverfd,header_str,strlen(header_str));
            }
```

3) Port numbers

There are two significant classes of port numbers for this lab: HTTP request ports and proxy's listening port. Proxy properly function whether or not the port number is included in the URL.

```
static char *default_port = "80";
```

The listening port is the port on which the proxy listens for incoming connections. The proxy accepts a command line argument specifying the listening port number.

```
seri@ubuntu:~/Downloads/proxylab-handout$ ./port-for-user.pl seri
seri: 22108
seri@ubuntu:~/Downloads/proxylab-handout$ ./proxy 22108
bash: ./proxy: No such file or directory
seri@ubuntu:~/Downloads/proxylab-handout$ make
gcc -g -Wall -c proxy.c
gcc -g -Wall -c csapp.c
gcc -g -Wall -c cache.c
gcc -g -Wall proxy.o csapp.o cache.o -o proxy -lpthread
seri@ubuntu:~/Downloads/proxylab-handout$ ./proxy 22108
^C
seri@ubuntu:~/Downloads/proxylab-handout$ ./free-port.sh 22108
```

## 2. Dealing with multiple concurrent requests

The second part introduce us to dealing with concurrency, a crucial system concept.

```
while (1)
        {

            위 코드 생략 ...

           //Dealing with multiple concurrent requests
            pthread_t tid;
            Pthread_create(&tid, NULL,
            handle_client, (void *)(long)clientfd);
        }
```

The threads run in detached mode to avoid memory leaks.

```
void *handle_client(void *vargp)
{
    Pthread_detach(Pthread_self());
```

## 3. Caching web objects

In this part, I added a cache to proxy that stores recently-used web objects in memory.

When the proxy receives a web object from a server, it caches it in memory as it transmits the object to the client.

```
//in handle_client function
static char response[MAX_OBJECT_SIZE];
    /**from server to client**/
int response_size = from_server_to_client(serverfd,
clientfd, response);

//then cache it in cache list
if (response_size < MAX_OBJECT_SIZE) {
//if the size of the buffer ever exceeds the maximum
object size, the buffer can be discarded
//write to cache
        write_cache(req.request, response,
response_size);
    }
```

If another client requests the same object from the same server, the proxy can simply resend the cached object.

```
char cached_obj[MAX_OBJECT_SIZE];
    //read cache and copy it into cached_obj
    //if n is 0 then it is not in the cache
    ssize_t n = read_cache(req.request, cached_obj);
    if (n) {

        Rio_writen(clientfd, cached_obj, n);
        return NULL;
```

MAX_CACHE_SIZE = 1 MiB

MAX_OBJECT_SIZE = 100KiB

Allocate a buffer for each active connection and accumulate data as it is received from the server.

```
//in this case, buffer is response
static char response[MAX_OBJECT_SIZE];
int response_size = from_server_to_client(serverfd,
clientfd, response);
int from_server_to_client(int serverfd, int clientfd,
char *response)
{
```

```
    코드 중간 생략 ...
    strcat(response, buf);
```

If the size of the buffer ever exceeds the maximum object size, the buffer can be discarded. If the entirety of the web server's response is read before the maximum object size is exceeded, then the object can be cached.

```
//this is implemented in handle_client
//then cache it in cache list
    if (response_size < MAX_OBJECT_SIZE) {
        //if the size of the buffer ever exceeds the
maximum object size, the buffer can be discarded
        //write to cache
        write_cache(req.request, response,
response_size);
    }

int from_server_to_client(int serverfd, int clientfd,
char *response)
{
  코드 중간 생략 ...
  while((n=rio_readn(serverfd, buf, MAXLINE)) != 0)
    {

        코드 중간 생략 ..
        if (response_size < MAX_OBJECT_SIZE)
        {
            코드 중간 생략 …
       }

        Rio_writen(clientfd, buf, n);

    }
```

Synchronization

Accesses to the cache is thread-safe, and ensuring that cache access is free of race conditions. Multiple threads must be able to simultaneously read from the cache. Only one thread is permitted to write to the cache at a time, but that restriction does not exist for readers.

```
static int readcnt;
sem_t mutex,w;

int read_cache(char *request, char *buf)
{
    int len = 0;
```

```
    P(&mutex);
    readcnt++;
    if (readcnt == 1) { //first in
        P(&w);
    }
    V(&mutex);

    //코드 생략

    P(&mutex);
    readcnt--;
    if (readcnt == 0) { //last out
        V(&w);
    }
    V(&mutex);

    return len;
}
void write_cache(char *request, char *block_data, int
block_size)
{
    struct cache_block *temp;
    P(&w);
    //코드 생략
    V(&w);
}
```

I used semaphores to implement some sort of first readers-writers solution.

---

## 4. Result & Conclusion

```
Fetching ./tiny/home.html into ./.noproxy directly from Tiny
Fetching ./tiny/home.html into ./.proxy using the proxy
Checking whether the proxy fetch succeeded
Success: Was able to fetch tiny/home.html from the proxy.
Killing tiny, proxy, and nop-server
concurrencyScore: 15/15

*** Cache ***
Starting tiny on port 3611
Starting proxy on port 3118
Fetching ./tiny/tiny.c into ./.proxy using the proxy
Fetching ./tiny/home.html into ./.proxy using the proxy
Fetching ./tiny/csapp.c into ./.proxy using the proxy
Killing tiny
Fetching a cached copy of ./tiny/home.html into ./.noproxy
Success: Was able to fetch tiny/home.html from the cache.
Killing proxy
cacheScore: 15/15

totalScore: 70/70
```

```
stu224@sp2:~/proxylab-handout$ curl --proxy localhost:22108 http://snu.ac.kr/ind
ex.html
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.or
g/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="ko" xml:lang="ko">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<title>서 울 대 학 교 </title>
<meta name="author" content="SEOUL NATIONAL UNIVERSITY" />
<meta name="robots" content="all" />

<link rel="icon" href="/favicon.ico" type="image/x-icon" />
<link rel="shortcut icon" href="/favicon.ico" type="image/x-icon" />

<link rel="apple-touch-icon-precomposed" href="/mobileicon.png" />


stu224@sp2:~/proxylab-handout$ curl --proxy localhost:22108 http://csapp.cs.cmu.
edu
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.or
g/TR/html4/loose.dtd">

<html>
<head>
<title>CS:APP3e, Bryant and O'Hallaron</title>
<link href="http://csapp.cs.cmu.edu/3e/css/csapp.css" rel="stylesheet" type="tex
t/css">
</head>

<!-- Page-independent header  -->
<div id="cover_box">
<a href="http://csapp.cs.cmu.edu/3e/home.html">
<img src="http://csapp.cs.cmu.edu/3e/images/csapp3e-cover.jpg" width=70>
</a>
</div>



stu224@sp2:~/proxylab-handout$ curl --proxy localhost:22108 http://www.sk.co.kr



<!doctype html>
<meta http-equiv="X-UA-Compatible" content="IE=edge"/>
<html lang="ko">
<head>
        <meta charset="UTF-8">
        <meta name="Author" content="">
        <meta name="description" content="SK공 식  홈 페 이 지 입 니 다 . 기 업 문 화 , 역 사 ,
 SUPEX추 구 협 의 회  및  계 열 사  정 보 , 채 용 안 내 , 사 회 공 헌  활 동  소 개 ">
        <meta name="viewport" content="width=device-width, initial-scale=1.0, mi
nimum-scale=1.0, maximum-scale=1.0, user-scalable=no">
        <script src="/lib/plugin/jquery-1.11.1.min.js"></script>
```

```
stu224@sp2:~/proxylab-handout$ curl --proxy localhost:22108 http://www.culture.g
o.kr/index.do
<!DOCTYPE html>
<html lang="ko" xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0, maximu
m-scale=1.0, minimum-scale=1.0, user-scalable=no">
    <title>문화포털 - 메인페이지</title>
    <link rel="stylesheet" type="text/css" href="/assets/css/main.css">
    <script type="text/javascript" src="/assets/js/jquery-1.11.1.min.js"></scrip
t>
    <script type="text/javascript" src="/assets/js/slick.min.js"></script>
    <script type="text/javascript" src="/assets/js/common_ui.js"></script>
    <script type="text/javascript" src="/assets/js/layout.js"></script>

    <!-- kakao map 관련  -->
    <script type="text/javascript" src="//dapi.kakao.com/v2/maps/sdk.js?appkey=2
```

사실 GET method만 구현하면 되는 굉장히 간단해보이는 과제, (막상 manually testing을 하면 정말 simple한 단 한가지 동작밖에 하지 못하는 과제)였지만 이렇게 간단해 보이는 것을 구현하는 데 이렇게 많은 시간이 들 줄 몰랐다. 특히 request를 parse 하는 과정에서 strncpy, strstr, strcat 함수를 사용하는데 자꾸 오류가 생겨 시간을 많이 소모해야 했다. (‘\0’을 더해서 비교해야 한다는 사실을 깨닫게 되었다 ..)

cache 부분은 오히려 수업시간에 배운 first readers-writers problem의 solution을 약간 변형한 mutex를 그대로 사용하고 그 외에는 자료구조 시간에 배운 linked-list를 구현하는 과정과 크게 다른 것은 없어서 많이 헤메지 않고 구현할 수 있었다. 현재는 수업시간에 배운 multi-threaded 방식으로 proxy가 구현되어있다. (그렇게 해도 점수가 잘 나오고 manual testing을 해도 문제가 있어 보이진 않는다.) 그러나 시간이 더 허용된다면 수업시간에 배운 pre-threaded 방식으로도 proxy를 구현해보고 싶다.

소개원실 수업을 들으면서 web에서 response와 request가 오가는 것들을 많이 다뤘었지만 Header도 훨씬 복잡했고 request의 구조에 대해서도 하나하나 따져가면서 생각해본 적이 없었기 때문에 request와 response의 문장 구조를 가장 기본적인 것부터 구성해보고 파싱해본 것이 좋은 경험이었다. 그러나 에러 핸들링하기가 무척 까다롭긴 했다. (‘\r\n’등의 실수를 하는 바람에)