System Programming
Lab #5

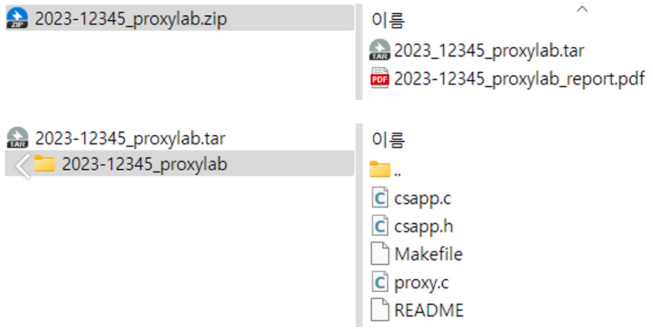
2023-05-23

sp-tas

Lab Assignment #5 : Proxy Lab

- Download skeleton code & pdf from eTL

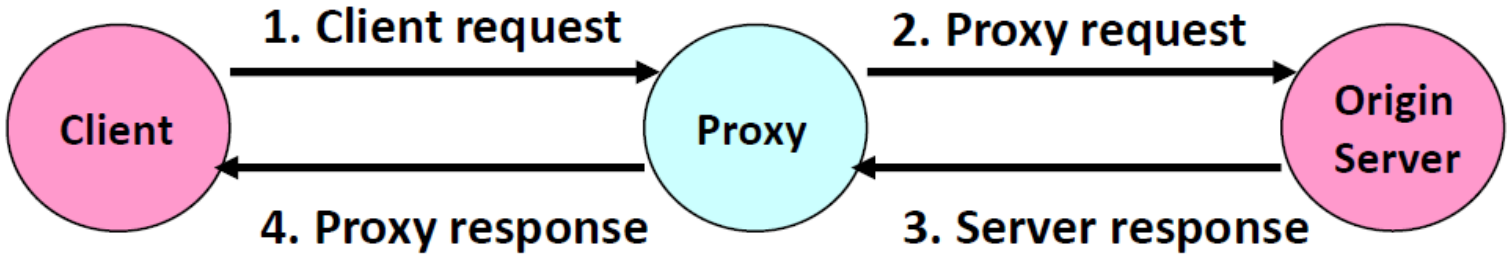
  proxylab-handout.tar, proxylab-handout.pdf



- Hand In
  - 구현 디렉토리 압축파일: 학번-proxylab.tar
  - Upload your files eTL
    - 압축파일 양식 : [학번]_proxylab.zip
    - Ex) 2023-12345_proxylab.zip
  - A zip file should include
  - (1) a tarball of your implementation directory (2) report
    - tarball 양식 : [학번]_proxylab.tar eg) 2023-12345_proxylab.tar
      Report 양식 : [학번]_proxylab_report.pdf
    - Please, READ the Hand-out and Lab material thoroughly!
- Assigned : May 23th
- Deadline : June 20th, 23:59:59 **(No late submission)**

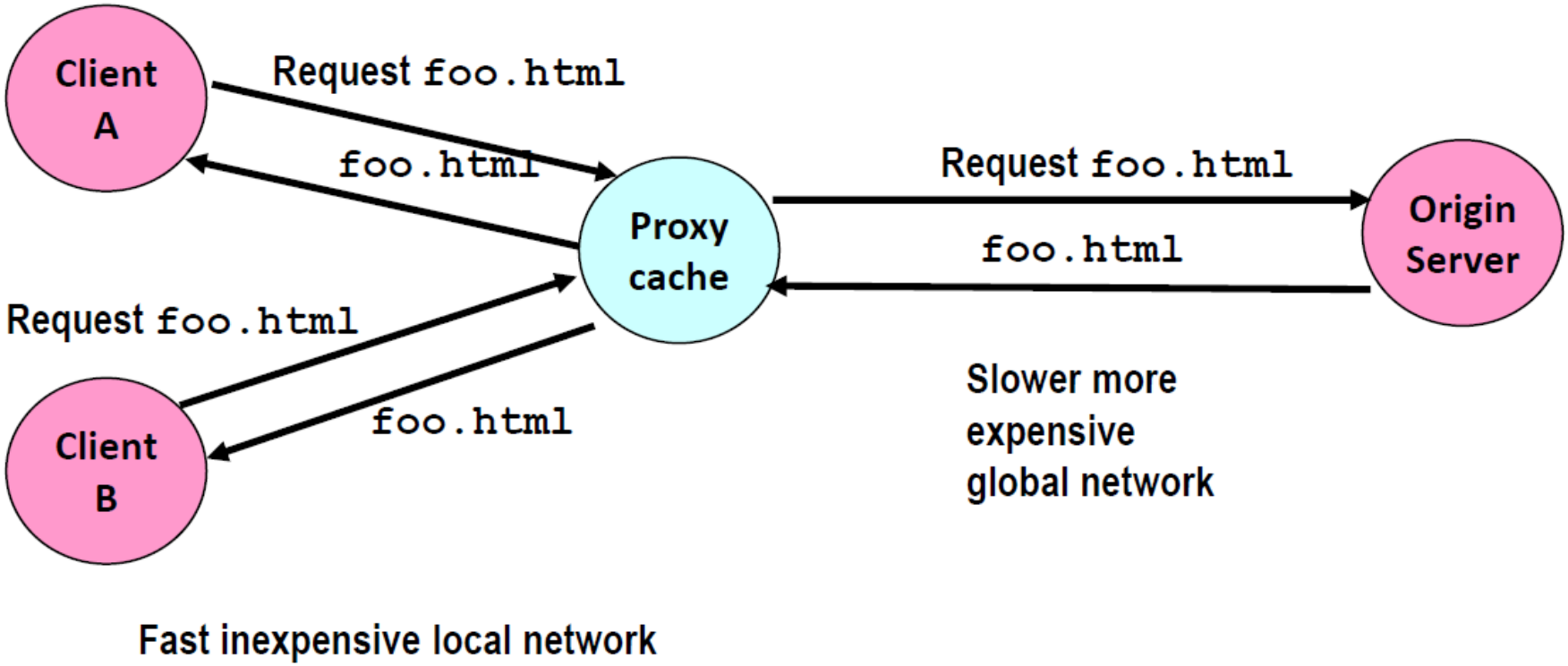
Proxies

- A *proxy* is an intermediary between a client and an *origin server*
  - To the client, the proxy acts like a server
  - To the server, the proxy acts like a client



Why Proxies?

- **Can perform useful functions as requests and responses pass by**
  - Examples: Caching, logging, anonymization, filtering, transcoding



How the Web Really Works

- In reality, a single HTML page today may depend on 10s or 100s of support files (images, stylesheets, scripts, etc.)
- Builds a good argument for concurrent servers
  - Just to load a single modern webpage, the client would have to wait for 10s of back-to-back request
  - I/O is likely slower than processing, so back
- Caching is simpler if done in pieces rather than whole page
  - If only part of the page changes, no need to fetch old parts again
  - Each object (image, stylesheet, script) already has a unique URL that can be used as a key

You will implement

- Write a simple HTTP proxy that caches web objects

- Part 1: Implementing a sequential web Proxy
  - **Basic HTTP operation & socket programming**
    - set up the proxy to accept incoming connections
    - read and parse requests
    - forward requests to web servers
    - read the servers' responses
    - forward those responses to the corresponding clients

- Part 2: Dealing with multiple concurrent requests
  - upgrade your proxy to deal with multiple **concurrent** connections
  - multi-threading

- Part 3: Caching web objects
  - add caching to your proxy using a simple main memory cache of recently accessed web content
  - cache individual objects, not the whole page
  - **Use an LRU eviction policy**
  - your caching system must allow for concurrent reads while maintaining consistency

Guide to start your implementation

- int main(int argc, char *argv[])
  - initialize everything such as data structure
  - checking port number
  - establish listening requests
  - when a client connects, spawn a new thread to handle it

```c
#include <stdio.h>

/* Recommended max cache and object sizes */
#define MAX_CACHE_SIZE 1049000
#define MAX_OBJECT_SIZE 102400

/* You won't lose style points for including this long line in your code */
static const 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";

int main()
{
    printf("%s", user_agent_hdr);
    return 0;
}
```

Use csapp.[ch] functions

- Also, csapp.[ch] codes are included! yeah!

```
/* Sockets interface wrappers */
int Socket(int domain, int type, int protocol);
void 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);
int 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 addrinfo **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);

/* DNS wrappers */
struct hostent *Gethostbyname(const char *name);
struct hostent *Gethostbyaddr(const char *addr, int len, int type);
```

```
/* Rio (Robust I/O) package */
ssize_t rio_readn(int fd, void *usrbuf, size_t n);
ssize_t rio_writen(int fd, void *usrbuf, size_t n);
void rio_readinitb(rio_t *rp, int fd);
ssize_t rio_readnb(rio_t *rp, void *usrbuf, size_t n);
ssize_t rio_readlineb(rio_t *rp, void *usrbuf, size_t maxlen);

/* Wrappers for Rio package */
ssize_t Rio_readn(int fd, void *usrbuf, size_t n);
void Rio_writen(int fd, void *usrbuf, size_t n);
void Rio_readinitb(rio_t *rp, int fd);
ssize_t Rio_readnb(rio_t *rp, void *usrbuf, size_t n);
ssize_t Rio_readlineb(rio_t *rp, void *usrbuf, size_t maxlen);

/* Reentrant protocol-independent client/server helpers */
int open_clientfd(char *hostname, char *port);
int open_listenfd(char *port);

/* Wrappers for reentrant protocol-independent client/server helpers */
int Open_clientfd(char *hostname, char *port);
int Open_listenfd(char *port);
```

```
/* Pthreads thread control wrappers */
void Pthread_create(pthread_t *tidp, pthread_attr_t *attrp,
                    void * (*routine)(void *), void *argp);
void Pthread_join(pthread_t tid, void **thread_return);
void Pthread_cancel(pthread_t tid);
void Pthread_detach(pthread_t tid);
void Pthread_exit(void *retval);
pthread_t Pthread_self(void);
void Pthread_once(pthread_once_t *once_control, void (*init_function)());
```

Checking Your Work

- Auto grader
  - `./driver.sh` will run the tests:
    - Ability to pull basic web pages from a server
    - Handle a (concurrent) request while another request is still pending
    - Fetch a web page again from your cache after the server has been stopped
  - This should help answer the question:
    "Is this what my proxy is supposed to do?"
  - Please don't use this grader to definitively test your proxy; there are many things not tested here
  - If you have a permission problem, use the command below
    - chmod +x *

Checking Your Work

- Get your port
  - Generate a port for yourself with `./port-for-user.pl [sp ID]`
  - Generate more ports for web servers and such with `./free-port.sh`
- Run proxy server
  - ./proxy port_id & (background)
- Request to server
  - (without proxy) >> curl http://www.sk.co.kr
  - (with proxy) >> curl --proxy localhost:port_id http://www.sk.co.kr

Evaluation

- Total Score: 100 points
- Basic Correctness (40 points)
  - basic proxy operation (auto graded)
- Concurrency (15 points)
  - handling concurrent requests (auto graded)
- Cache (15 points)
  - working cache (auto graded)
- Real Pages (20 points)
  - correctly serving the real pages
- Report (10 points)
  - describes the goal of proxy lab and how to implement for each part
  - what you learn in this lab
  - what was difficult, surprising, and so on

Don’t forget!

Last year's FAQ

- Q1. Do I need to implement GET request only?
  - A1. Yes. Other requests (e.g., POST) are optional.
- Q2. Do I have to consider chunked responses?
  - A2. No, this is also optional.
- Q3. May I assume that the URI of a GET request is an absolute path? (e.g., `http://example.com/index.html`)
  - A3. Yes, relative paths(e.g., `/index.html`)are not tested.
- Q4. Which size is used for calculating the cache size?
  - A4. The size of a response message from the server is used.

Last year's FAQ

- Q5. Does the response of the same request can be changed?
  - A5. In our evaluation, the response will be the same.
- Q6. Timeout bug in `driver.sh`?
  - A6. Check if `python` and `netstat` are installed properly.
- Q7. Does the ordering of the header fields affect the response?
  - A7. No, it does not affect the response of a request.

Fin.

- Questions
  - eTL Q&A Board
- Read the handout thoroughly & start early!
- This is our last LAB session
- The QnA session will 6/13