시스템 프로그래밍 proxy lab 보고서

2017-12751 컴퓨터공학부 이동학

이번 lab에서는 web object를 cache하는 간단한 HTTP proxy를 구현하였습니다. 우선 sequential web proxy를 구현하여 기본적인 HTTP 동작과 socket programming을 익힌 후, multiple concurrent request에 대응하고 web object를 caching하도록 업그레이드 하였습니다.

1. proxy.c

```c
#include <stdio.h>
#include "csapp.h"
#include "cache.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";
static const char *init_version = "HTTP/1.0\r\n";
static const char *connection = "Connection: close\r\n";
static const char *proxy_connection = "Proxy-Connection: close\r\n";

/* function definitions */
void server_connection(void *vargp);
int parse_line(char* host, char* filename, char* URI, char* port);
int forward_to_server(char *host, char* port, int *server_fd, char *request_str);
int read_and_forward_response(int server_fd, int client_fd, char* uri);
int request_client(rio_t *rio_client, char *str, int client_fd, char* port, char* host, char* filename, char *method, char* version);
```

각종 문자열과 함수들을 선언해둡니다.

```c
int main(int argc, char **argv)
{
    int listenfd, connfd, port;
    unsigned int clientlen;

    listenfd = Open_listenfd(argv[1]);
    struct sockaddr_in client_addr;
    Signal(SIGPIPE, SIG_IGN);
    init();

    while(1) {
        clientlen = sizeof(struct sockaddr_in);

        pthread_t tid;
        int *connfdp = Malloc(sizeof(int));
        *connfdp = -1;
        *connfdp = Accept(listenfd, (SA *) &client_addr, (socklen_t *)&clientlen);
        Pthread_create(&tid, NULL, (void *)server_connection, (void *)connfdp);
    }
}
```

server와 client의 connection을 accept하는 proxy를 setup합니다.

```c
void server_connection(void *vargp)
{
    Pthread_detach(Pthread_self());

    int clientfd = *((int *) vargp);
    char port[MAXLINE];
    char line[MAXLINE], host[MAXLINE], filename[MAXLINE], method[MAXLINE], URI[MAXLINE], version[MAXLINE];
    char client_str[MAXLINE];
    rio_t r;

    Rio_readinitb(&r, clientfd);
    Rio_readlineb(&r, line, MAXLINE);
    sscanf(line, "%s %s %s", method, URI, version);

    printf("request : %s\n", line);
    printf("method : %s\n", method);
    printf("uri : %s\n", URI);
    printf("http version : %s\n", version);

    if(strcasecmp(method, "GET"))
    {
        return;
    }

    if(!parse_line(host, filename, URI, port))
    {
        return;
    }

    cnode *temp;
    if( (temp = find(URI)) != NULL )
    {
        printf("Cache Hit \n");
        Rio_writen(clientfd, temp->object, strlen(temp->object));
        printf("forward to client : ok now on \n");
        if(clientfd >=0)
            Close(clientfd);
        return;
    }
```

server로부터 line을 받아 get request를 받고 문자열을 parsing 해줍니다. cache hit이면 바로 return 해줍니다.

```c
    request_client(&r, client_str, clientfd, port, host, filename, method, version);
    printf("read request : ok now on \n");
    printf("%s \n", client_str);

    int serverfd;
    forward_to_server(host, port, &serverfd, client_str);
    printf("forwarding ok : ok now on \n");

    read_and_forward_response(serverfd, clientfd, URI);

    printf("forward to client : ok now on \n");
    if(clientfd >=0)
        Close(clientfd);

    if(serverfd >=0)
        Close(serverfd);

}
```

client로부터 받은 request를 parsing한 후 server에 forward합니다. 그 후 client에 forward response를 보내고 모든 fd를 close합니다.

```c
int request_client(rio_t *rio_client, char *str, int client_fd, char* port, char* host,
 char* filename, char *method, char* version)
{
    char tmpstr[MAXBUF];
    char tmpport[MAXBUF];

    if (strstr(method, "GET")) {
        strncpy(str, method, strlen(method) + 1);
        strcat(str, " ");
        strcat(str, filename);
        strcat(str, " ");
        strcat(str, init_version);


        if(strlen(host))
        {
            strncpy(tmpstr, "Host: ", strlen("Host: ") + 1);
            strcat(tmpstr, host);
            strcat(tmpstr, ":");
            strcat(tmpstr, port);
            strcat(tmpstr, "\r\n");
            strcat(str, tmpstr);
        }

        strcat(str, user_agent_hdr);
        strcat(str, connection);
        strcat(str, proxy_connection);

        printf("\\\\\\\\\\\\\\\\\\\\\\\\\n");
        while(Rio_readlineb(rio_client, tmpstr, MAXBUF) > 0) {
            printf("%s\n", tmpstr);
            if (!strcmp(tmpstr, "\r\n")){
                strcat(str,"\r\n");
                break;
            }
            else if(strstr(tmpstr, "User-Agent:") || strstr(tmpstr, "Connection:") ||
                strstr(tmpstr, "Proxy Connection:") || strstr(tmpstr, "Host:")  )
                continue;
            else
                strcat(str, tmpstr);
        }

        return 0;
    }
    return 1;
}
```

client로부터 받은 request를 parsing하는 과정입니다.

```c
int read_and_forward_response(int server_fd, int client_fd, char *uri) {


    char tmp_str[MAXBUF], content[MAX_OBJECT_SIZE];
    unsigned int size = 0, len = 0;
    int valid_size = 1;

    content[0] = '\0';

    rio_t rio_server;
    Rio_readinitb(&rio_server, server_fd);

    while ((len = Rio_readlineb(&rio_server, tmp_str, MAXLINE)) != 0) {
        printf("%s\n", tmp_str);
        size += len;
        if (size <= MAX_OBJECT_SIZE)
            strcat(content, tmp_str);
        Rio_writen(client_fd, tmp_str, len);
    }

    add_cache(uri, content);
    printf("Added Cache\n");

    return 0;
}
```

client에 response를 forward하는 과정입니다.

```c
int forward_to_server(char *host, char *port, int *server_fd, char *request_str)
{
    *server_fd = Open_clientfd(host, port);

    if (*server_fd < 0) {
        if (*server_fd == -1)
            return -1;
        else
            return -2;
    }
    Rio_writen(*server_fd, request_str, strlen(request_str));

    return 0;
}
```

request를 server에 forward하는 과정입니다.

```c
int parse_line(char* host, char* filename, char* URI, char* port)
{
    if(sscanf(URI, "http://%[^/]%s", host, filename) != 2)
        return 0;

    if(strstr(host, ":")) {
        if(sscanf(host, "%*[^:]:%s", port) != 1) return 0;
        if(sscanf(host, "%[^:]", host) != 1) return 0;
    }
    else strcpy(port, "80");
    return 1;
}
```

uri로부터 host, filename, port를 parsing하는 과정입니다.

2. cache.h

```c
#include "csapp.h"

#define MAX_CACHE_SIZE 1049000
#define MAX_OBJECT_SIZE 102400

int sum_of_cache;

typedef struct cnode {
    char uri[MAXLINE];
    char object[MAX_OBJECT_SIZE];
    int content_size;
    struct cnode *next;
    struct cnode *prev;
} cnode;

cnode *rear, *front;

void init();
cnode *find(char *uri);
void replacement(int length);
void add_cache(char *uri,  char *content);
```

cache_block structure를 이용해 linked list를 구현하고 LRU policy를 이용하기 위한 node를 선언해줍니다.

3. cache.c

```c
#include "cache.h"
#include <stdlib.h>
#include <string.h>

void init()
{
    sum_of_cache = 0;
    rear = (front = NULL);
}

cnode *find(char *uri)
{
    cnode *temp = front;

    for(temp = front; temp != NULL; temp = temp->next)
    {
        printf("uri: %s \n THIS IS CACHE", temp-> uri);
        if(!strcmp(temp->uri, uri))
        {
            if(temp == front)
            {
                if(temp->next != NULL)  temp->next->prev = NULL;
                if(rear != NULL)    rear->next = temp;
                temp->prev = rear;
                rear = temp;
                front = temp->next;
                temp->next = NULL;
            }
            else if(temp == rear);
            else
            {
                temp->prev->next = temp->next;
                temp->next->prev = temp->prev;
                temp->prev = rear;
                temp->prev->next = temp;
                rear = temp;
            }
            return temp;
        }
    }
    return NULL;
}
```

cache를 hit하면 처리 후 리턴하고 아니라면 아무것도 하지 않습니다.

```c
void replacement(int length)
{
    cnode *temp;

    while(front != NULL && sum_of_cache + length > MAX_CACHE_SIZE) {
        sum_of_cache -= front->content_size;
        temp = front->next;
        Free(front);
        front = temp;
    }

    if(front == NULL) rear = NULL;
}
```

lru policy에 따라 replacement를 해줍니다. 아무 블록도 없는경우 따로 처리를 해줍니다.

```c
void add_cache(char *uri, char *content)
{
    cnode *new_block;

    if(strlen(content) > MAX_OBJECT_SIZE) return;

    if(find(uri) != NULL) return;

    replacement(strlen(content));

    new_block = (cnode *) Malloc(sizeof(cnode));

    strncpy(new_block->uri, uri, strlen(uri) + 1);
    memcpy(new_block->object, content, strlen(content));
    new_block->content_size = strlen(content);
    new_block->next = NULL;
    new_block->prev = NULL;

    sum_of_cache += strlen(content);

    if(rear == NULL) front = (rear = new_block);
    else {
        rear->next = new_block;
        new_block->prev = rear;
        rear = new_block;
    }

    printf("Cache Addition Finished");
    for(cnode *t = front; t!=NULL; t= t->next)
    {
        printf("%s\n", t->uri);
    }
}
```

cache를 추가할 때 content의 크기, cached 여부, 빈공간 확인 후 proxy cache와 linked list에 새로운 cache block을 추가해줍니다.

후기: 평소에도 자주 사용하던 프록시를 직접 구현해보고 작동하는 것이 재미있었습니다. 각종 테스트 및 디버깅 도구와 csapp.c 파일의 존재 덕분에 저번 kernel lab보다는 수월하게 과제를 마칠 수 있었습니다. 본래 프록시의 기능만을 알고 있었다면, 이번 랩을 통해서 프록시의 동작 원리까지 자세하게 알 수 있었습니다.