

시스템 프로그래밍 실습 2-4 과제

이름 : 이준휘

학번 : 2018202046

교수 : 최상호 교수님

강의 시간 : 화

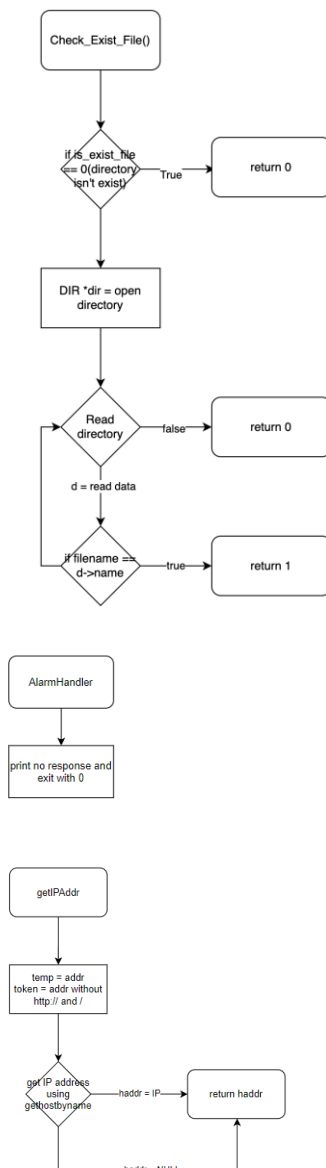
실습 분반 : 목 7, 8

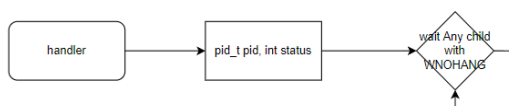
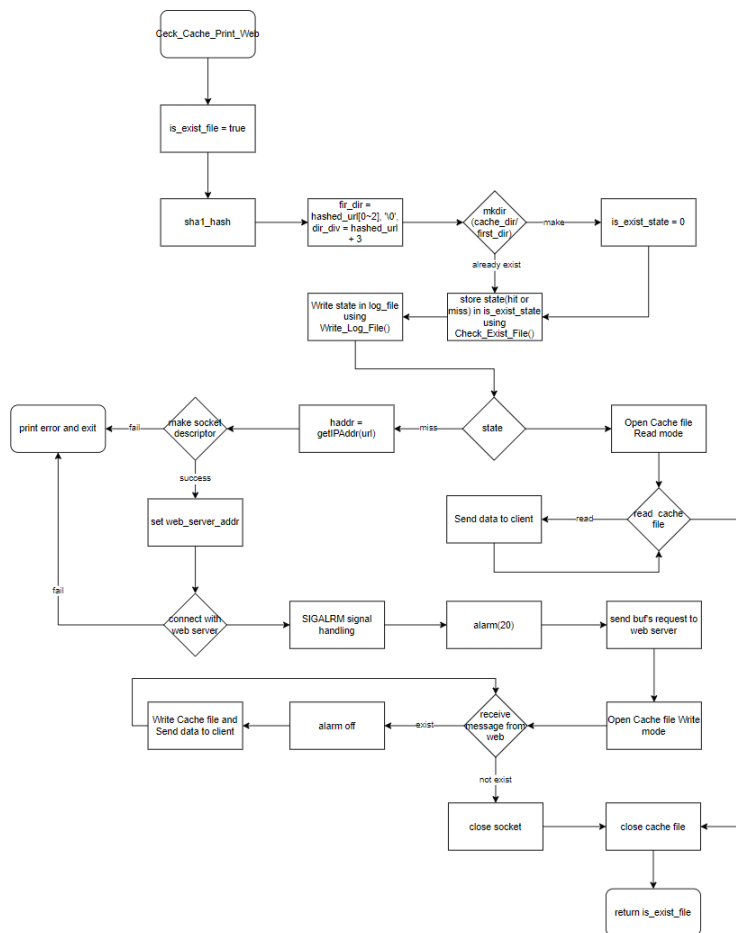
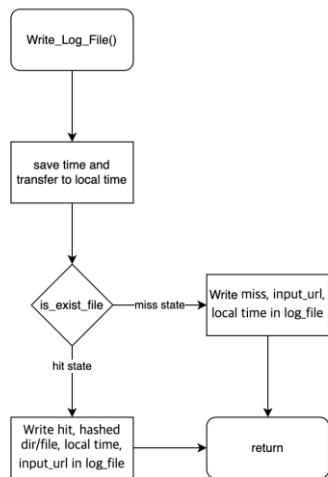
1. Introduction

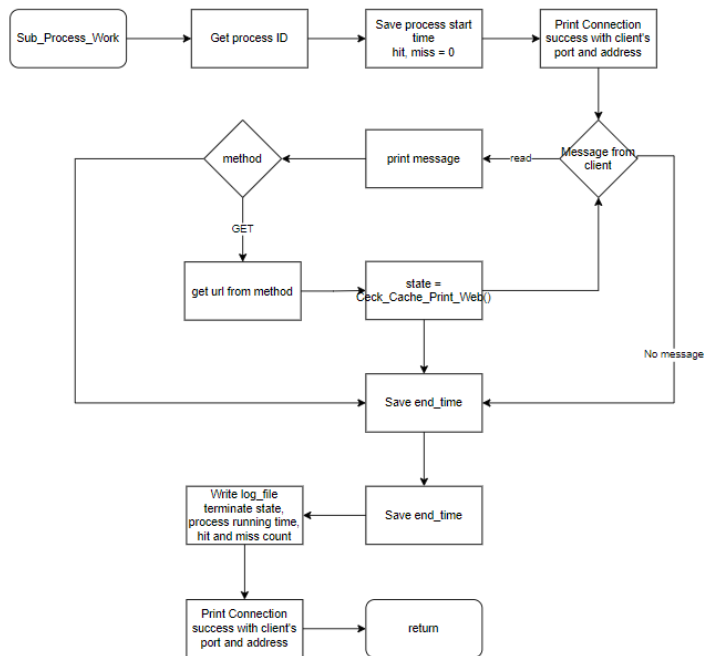
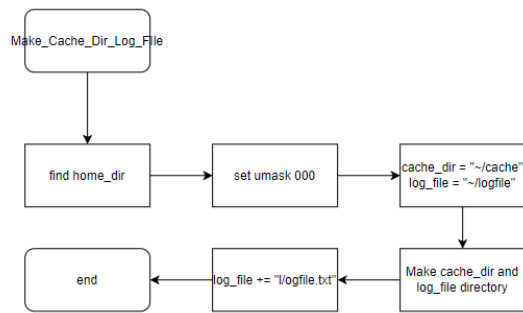
해당 과제는 기존에 2-3에서 만든 결과에서 추가적으로 덧붙여서 만들어진다. Cache파일의 유무를 파악한 뒤 miss 상태일 경우 web server와 연결하여 web server에 요청받은 request를 보낸다. 이후 response를 cache 파일에 저장하고 해당 데이터를 client에게 보낸다. hit의 경우 web과 연결하지 않고 cache파일에 있는 데이터를 client에게 보낸다. 이외의 부분은 기존과 동일하게 만들어진다.

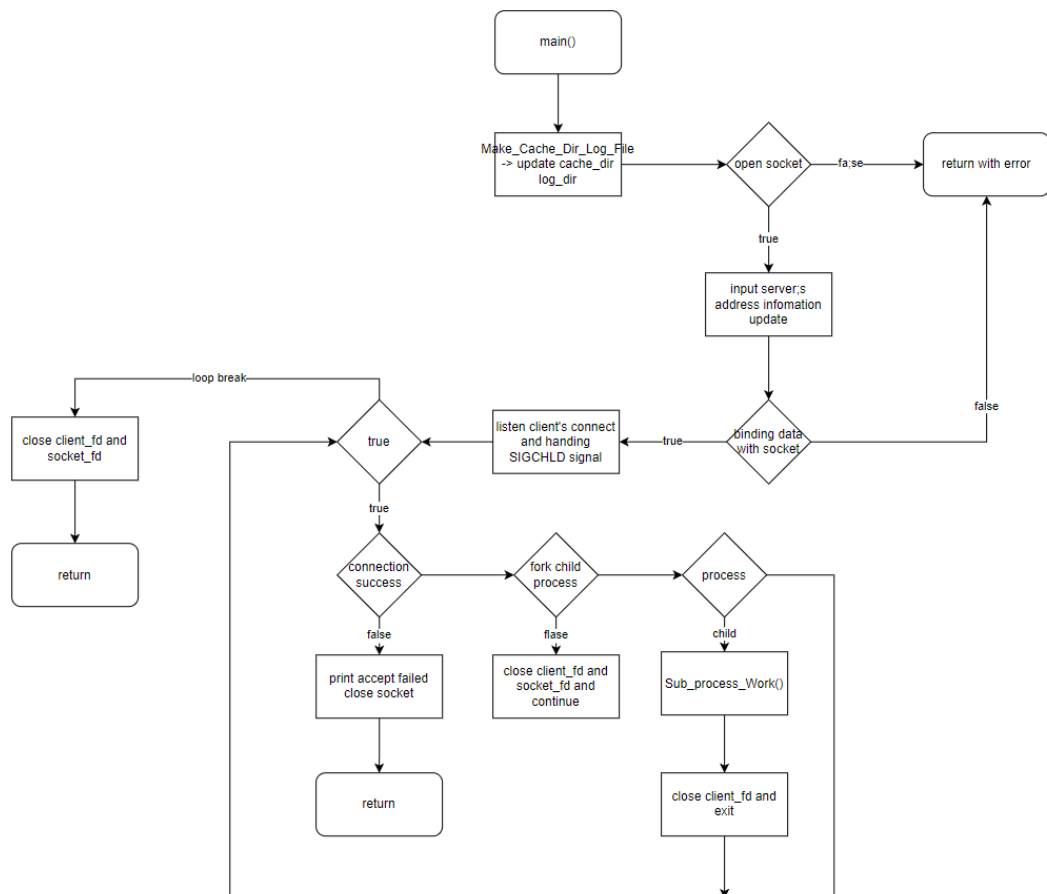
2. Flow Chart

- proxy_cache.c









3. Pseudo Code

```

static void handler(){

    pid_t pid;

    int status;

    Wait Any child with WNOHANG

}

```

```

static void AlarmHandler(){

    print No Response and exit(0);

}

```

```

char *getIPAddr(char *addr){

    struct hostent* hent;

    char *haddr = NULL;

    char temp[BUFSIZE] = addr;

    token = temp(delete http://, and tokenized with /;

    if (gethostbyname(token) is exist)

        haddr = inet_ntoa(hent's h_addr_list[0]);

    return haddr;

}

```

```

Make_Cache_Dir_Log_File(char* cache_dir, char* log_file){

    getHomeDirectory();

    cache_dir = ~/cache;

    log_file = ~/logfile;

    set umask 000;

    make cache and log directory;

    log_file += /logfile.txt;

}

```

```

Check_Exist_File(char *path, char *file_name, int is_exist_file){

    if(directory isn't exist)

        return 0;

    DIR *dir = Open path directory

    While(struct dirent *d = Read path directory){

        If(d->name == file name)

```

```

        Close directory and return 1;
    }

    Close directory and return 0;
}

```

```

Void Write_Log_File(File *log_file, char *input_url,
char *hashed_url_dir, char* hashed_url_file, int is_exist_file){

    time_t now;

    struct tm *ltp;

    ltp = current local time;

    if(miss state)

        Write miss, input_url, local time in log_file;

    Else

        Write hit, hashed dir/file, local time, input_url in log_file;

}

```

```

void Check_Cache_Print_Web(int client_fd, char *url, char *cache_dir, char *log_file, char *buf
int current_pid, int len, int *hit, int *miss){

    char[60] hashed_url;

    char[4] first_dir;

    char *dir_div;

    char[100] temp_dir;

    char *haddr[BUFSIZE];

    int h_socket_fd, cache_fd, h_len;

    int is_exist_file = 1;

```

```

struct sockaddr_in web_server_addr;

hashed_url = hashed url using sha1;

first_dir = { hashed_url[0~3], 'W0' };

dir_div = hashed_url + 3 address

temp_dir = ~/cache/first_dir;

if make temp_dir directory(permission = drwxrwxrwx)

is_exist_file = 0;

is_exist_file = hit or miss state;

Write state, url, dir/file name, time in logfile.txt;

temp_dir = ~/cache/first_dir/dir_div;

if(state == miss){

    haddr = getIPAddr from url;

    if (make socket fd failed)

        print error and exit();

    setting web_server_addr using haddr;

    if (connection with web failed)

        print error and exit();

    SIGALRM signal handling.

    alarm 20 sec;

    write buf's request to web server;

    open temp_dir with write mode;

    while(receive response){

        alarm off;

        write response at cache file;

```



```

        write response to client_fd;

        h_buf clear;

    }

    close h_socket_fd;

}

else{

    open temp_dir with read mode;

    while read data is exist in cache file{

        write data to client_fd;

        h_buf clear;

    }

}

close fd;

return is_exist_file;

}

```

```

void Sub_Process_Work(int client_fd, struct sock_addr, char *buf, char *char_dir, FILE
*log_file){

```

```

    char temp[BUFSIZE] = { 0 };

    char method[BUFSIZE] = { 0 };

    char url[BUFSIZE] = { 0 };

    char h_buf[BUFSIZE];

    char* haddr;

    char *token = NULL;

    int len, h_socket_fd;

    int state, hit = 0, miss = 0;

```

```

struct sockaddr web_server_addr;

pid_t current_pid = Current process ID;

time_t start_process_time, end_process_time;

Save start process time;

print Connect success with client address and port;

while read data from client_fd to buf is exist{

    print Request message;

    method = Request message's method;

    if(method == GET){

        url = Request message's url

        state = Ceck_Cache's state(Hit or Miss);

    }

}

check end time;

print terminate state, running time, hit or miss state in logfile.txt

print Terminate connection with client address and port;

return;

}

main(void){

    Make Cache and log directory and store path's information;

    if open socket is failed, print error and return;

    update server's address information;

    if binding socket and server's address data is failed, print error and return;

```

Waiting Connection and Collect SIGCHLD signal using handler;

```
while true{
```

```
    if connection didn't occur, print error and return;
```

```
    if make child process failed, close file descriptor and socket and continue;
```

```
    if child process, Do Sub_Process_Work() and exit;
```

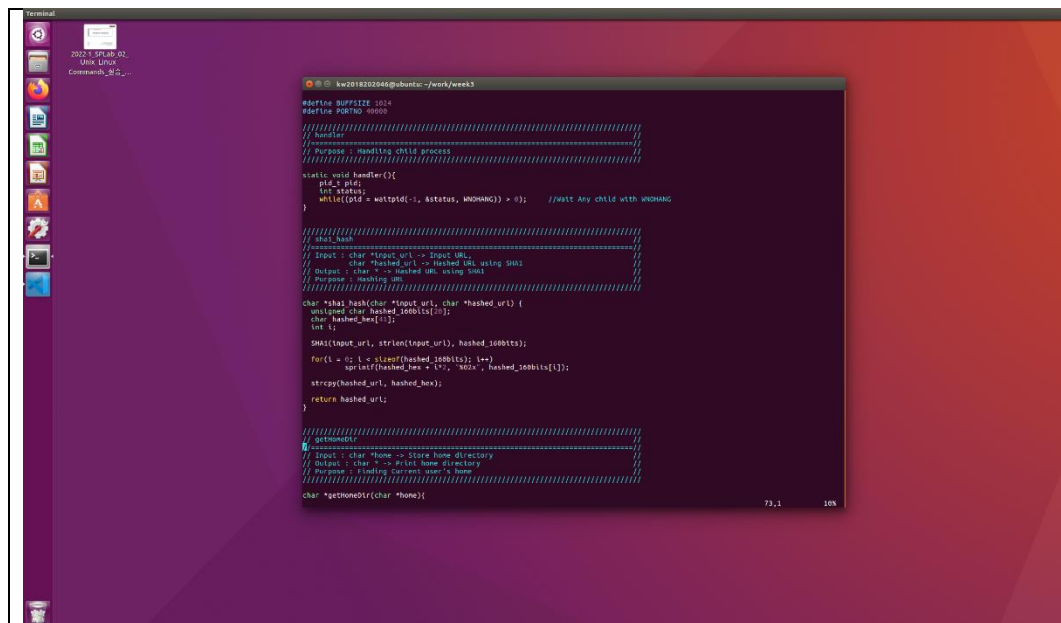
```
    close client file descriptor;
```

```
}
```

```
close socket file descriptor;
```

```
}
```

4. 결과 화면



```
Terminal
2022.1.10. 10:02
Unix/Linux
C++/C/C#

kw2018202046@ubuntu: ~/work/week3

#define SUPP_SIZE 1024
#define PORTNO 8080

// Purpose : Handling child process
// Purpose : Handling child process

static void handler(){
    pid_t pid;
    int status;
    while((pid = waitpid(-1, &status, WNOHANG)) > 0); //Wait Any child with WNOHANG
}

// Hash
// Input : char *input url -> Input URL
// Output : char * -> Hashed URL using SHA1
// Purpose : Hashing url

char *hash(char *input url, char *hashed_url){
    unsigned char hashed_160bits[160];
    char hashed_hex[320];
    int i;

    SHA1(input url, strlen(input url), hashed_160bits);

    for(i = 0; i < sizeof(hashed_160bits); i++)
        sprintf(hashed_hex + "%2", "%02x", hashed_160bits[i]);

    strcpy(hashed_url, hashed_hex);

    return hashed_url;
}

// HomeDir
// Input : char *home -> Store home directory
// Output : char * -> Print home directory
// Purpose : Finding current user's home

char *gethomeDir(char *home){
    73.1 10%
```

해당 함수는 signal함수에서 handling을 위한 함수로 waitpid(WNOHANG)을 통해 child process의 종료를 확인시켜주는 역할을 수행한다.


```
terminal
2023.10.09.02.
UNIX Linux
Command...

int check_Cache_Protocol(int client_fd, char *url, char *cache_dir, char *log_file, char *buf, int current_pid, int len, int* hit, int* miss){
    char hashed_url[1024]; //store hashed url using md5
    char *first_dir[1]; //Directory that will be made in cache directory
    char *dir_div; //separate point of hashed url name
    char *temp_dir[1]; //Path is used when it makes directory or file
    char *haddr = NULL;
    int h_socket_fd, cache_fd, h_len;
    int is_exist_file = 0;

    struct sockaddr_in web_server_addr;
    bzero(h_buf, sizeof(h_buf));

    strcpy(first_dir, hashed_url); //URL = hashed url
    strcpy(temp_dir, hashed_url); //Directory name = hashed URL[0-2]
    first_dir[0] = '\0';
    dir_div = hashed_url + 1; //file name pointer
    strcpy(temp_dir, cache_dir); //Make directory ~/cache/Directory name (permission : rwxrwxrwx)
    structstat temp_dir;
    structstat temp_dir;
    if(stat(temp_dir, 0777) == 0) //Directory isn't already exist, is_exist_file is 0
        is_exist_file = 0;
    is_exist_file = check_exist_file(temp_dir, dir_div, is_exist_file); //Check ~/cache/Directory name/file name is exist
    while(log_write_file(temp_dir, current_pid, url, first_dir, dir_div, is_exist_file, hit, miss); //write the state(hit or miss) in logfile.txt
        is_exist_file = 0;
    structstat temp_dir; //Make empty file ~/cache/Directory name/file name (permission : rwxrwxrwx)
    structstat temp_dir, dir_div;

    //If state is miss, get connection with web server
    if(is_exist_file == 0){
        haddr = getIPAddr(url); //Get host's IP address
        if(h_socket_fd = socket(AF_INET, SOCK_STREAM, 0) < 0){ //Make socket descriptor
            perror("can't create socket.\n");
            close(h_socket_fd);
            exit(0);
        }

        //Web server's address
        bzero((char *)web_server_addr, sizeof(web_server_addr));
        web_server_addr.sin_family = AF_INET;
        web_server_addr.sin_addr.s_addr = inet_addr(haddr);
        web_server_addr.sin_port = htons(HTTP_PORT);

        //Connect with web server
        if(connect(h_socket_fd, (struct sockaddr *)&web_server_addr, sizeof(web_server_addr)) < 0){
            perror("can't connect to web server.\n");
            close(h_socket_fd);
            exit(0);
        }

        //SIGALRM Handler
        alarm(10); //Alarm(10sec, alarmhandler);
        alarm(0); //Wait 10 sec
        //Sleep(5); //Connection error case

        cache_fd = open(temp_dir, O_WRONLY | O_CREAT | O_APPEND, 0777); //Create file
        write(cache_fd, buf, sizeof(buf)); //Send request to web server
        while(h_len = read(h_socket_fd, h_buf, HTTP_SIZE)){ //Read data from web server
            alarm(0); //If get data from web server, alarm off
            write(cache_fd, h_buf, h_len); //Write data in cache file
            write(client_fd, h_buf, h_len); //Send data to client
            bzero(h_buf, sizeof(h_buf));
        }
        close(h_socket_fd);
    } else //If state is hit, read cache file and send it to client
    {
        cache_fd = open(temp_dir, O_RDONLY); //Open read mode
        while(h_len = read(cache_fd, h_buf, HTTP_SIZE)){ //Read all of data in cache file
            write(client_fd, h_buf, h_len); //Send data to client
        }
        close(cache_fd);
    }
}

//Sub Process Work
// Sub Process Work
// Input : char *client_fd => client's file descriptor,
// struct sockaddr_in *client_addr => client's address info,
// char *buf => input buffer,
// char *cache_dir => cache path,
// FILE *log_dir => Write ~/logfile/logfile.txt,
// Output : void
// Purpose : receive message from client and check URL from Cache
// =====
void Sub_Process_Work(int client_fd, struct sockaddr_in client_addr, char *buf, char *cache_dir, char *log_dir){
    char h_buf[HTTP_SIZE];
    char method[HTTP_SIZE] = {0}; //receive method
    char url[HTTP_SIZE] = {0}; //receive url
    char *token = NULL; //tokenize
    int len;
    int state; //HIT or MISS exist
    int hit_or_miss = 0; //Count HIT and MISS

    FILE *log_file; //Log file's path
    FILE *temp_file; //Writing when make a empty file
    pid_t current_pid = getpid(); //Get current process ID
    time_t start_process_time, end_process_time; //Process start and end time

    time(&start_process_time); //Check process start time
    bzero(h_buf, HTTP_SIZE); //buffer clear

    //Read data from client file descriptor
    while((len = read(client_fd, h_buf, HTTP_SIZE)) > 0){
        strcpy(temp_buf, h_buf); //Copy message and print it
        printf("Received message from [%s : %d]\n", inet_ntoa(client_addr.sin_addr), client_addr.sin_port);
        printf("Method: %s, URL: %s\n", method, url);
        printf("=====");
        token = strtok(temp_buf, " ");
        strcpy(method, token);
        if(strlen(method) > 0){
            //If method is GET, Make Cache and send response
            if(strlen(url) > 0){
                token = strtok(NULL, " ");
                state = check_Cache_PrintWeb(client_fd, url, cache_dir, log_dir, buf, current_pid, len, &hit, &miss);
            }
        }
        time(&end_process_time); //Check end process time
        log_file = fopen(log_dir, "a");
        fprintf(log_file, "Terminated ServerPID : %d | Run time: %d sec | Request hit : %d, miss : %d\n",
            current_pid, (int)(end_process_time - start_process_time), hit, miss); //write which client was terminated, process execute time, hit, miss in logfile.txt
        fclose(log_file);
    }
    return; //end program
}

void main(void){
    char buf[HTTP_SIZE]; //buffer
    char web_dir[1024]; //Cache directory
    char log_dir[1024]; //Log directory
    int process_count = 0; //Count sub process
    int socket_fd, client_fd; //Socket and client file descriptor
    int len, len_buf; //length of buffer
    int opt = 0;

    struct sockaddr_in server_addr, client_addr; //server address and client address
    FILE *log_file; //using when write log file
    pid_t pid; //child PID
}
```

해당 함수는 기존의 Ceck_Cache 함수에서 추가된 내용을 포함하는 함수다. 이전과 같이 hit와 miss를 판별한 후 miss일 경우 이전 Sub_Process_Work에서 Web과 연결한 부분을 수행한다. 이 때 alarm은 사진을 받는데 오래 걸리기 때문에 20초로 설정하며 while문을 통해 반복적으로 response message를 받아 이를 cache file에 저장하고 client_fd에게 전달한다. miss일 경우 cache file을 읽고 해당 내용을 client_fd에게 전달한다.

```
terminal
2023.10.09.02.
UNIX Linux
Command...

// Sub Process Work
// Sub Process Work
// Input : char *client_fd => client's file descriptor,
// struct sockaddr_in *client_addr => client's address info,
// char *buf => input buffer,
// char *cache_dir => cache path,
// FILE *log_dir => Write ~/logfile/logfile.txt,
// Output : void
// Purpose : receive message from client and check URL from Cache
// =====
void Sub_Process_Work(int client_fd, struct sockaddr_in client_addr, char *buf, char *cache_dir, char *log_dir){
    char h_buf[HTTP_SIZE];
    char method[HTTP_SIZE] = {0}; //receive method
    char url[HTTP_SIZE] = {0}; //receive url
    char *token = NULL; //tokenize
    int len;
    int state; //HIT or MISS exist
    int hit_or_miss = 0; //Count HIT and MISS

    FILE *log_file; //Log file's path
    FILE *temp_file; //Writing when make a empty file
    pid_t current_pid = getpid(); //Get current process ID
    time_t start_process_time, end_process_time; //Process start and end time

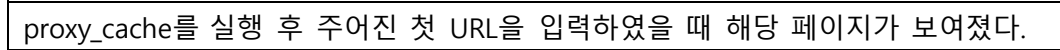
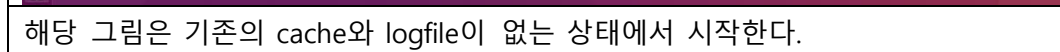
    time(&start_process_time); //Check process start time
    bzero(h_buf, HTTP_SIZE); //buffer clear

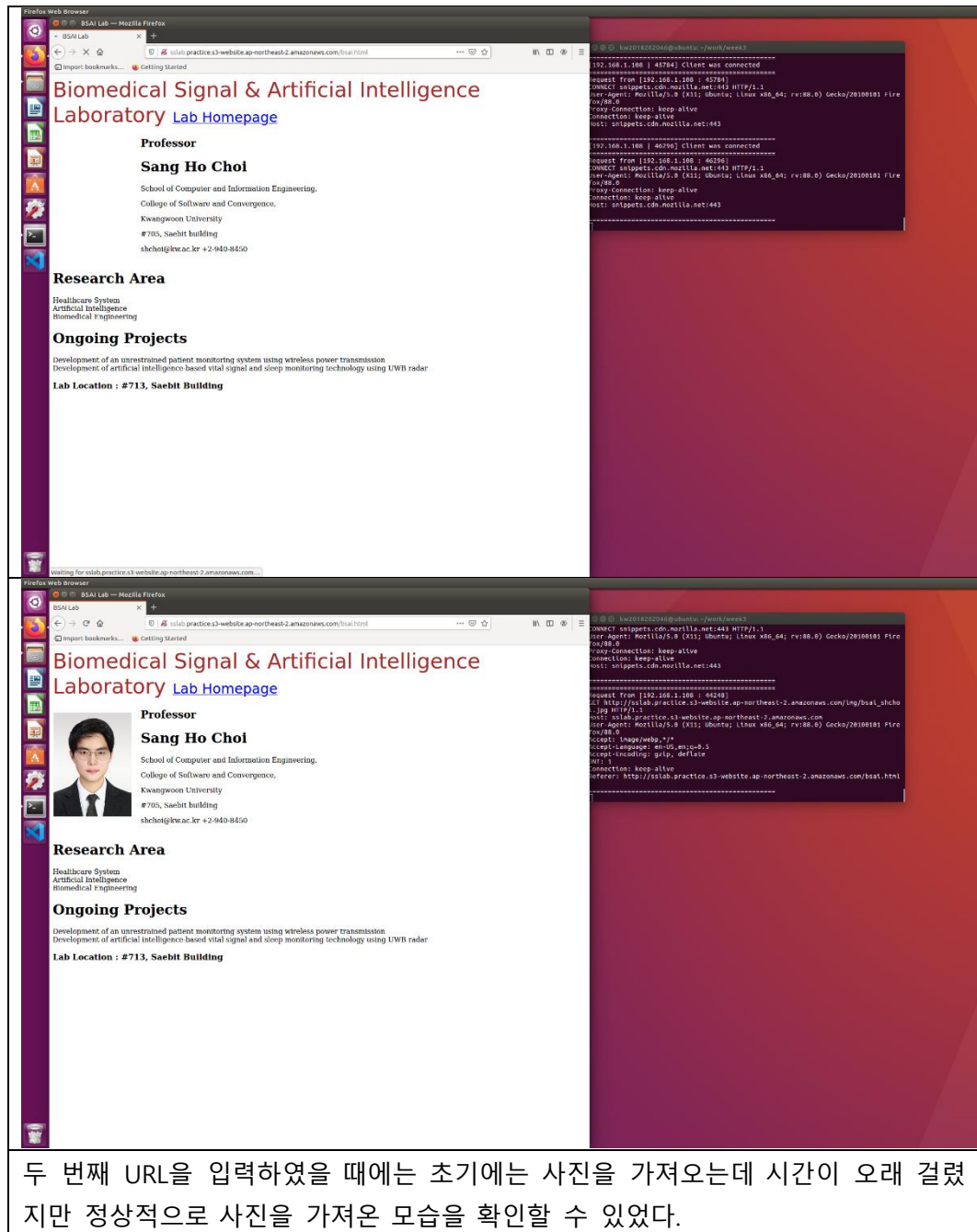
    //Read data from client file descriptor
    while((len = read(client_fd, h_buf, HTTP_SIZE)) > 0){
        strcpy(temp_buf, h_buf); //Copy message and print it
        printf("Received message from [%s : %d]\n", inet_ntoa(client_addr.sin_addr), client_addr.sin_port);
        printf("Method: %s, URL: %s\n", method, url);
        printf("=====");
        token = strtok(temp_buf, " ");
        strcpy(method, token);
        if(strlen(method) > 0){
            //If method is GET, Make Cache and send response
            if(strlen(url) > 0){
                token = strtok(NULL, " ");
                state = check_Cache_PrintWeb(client_fd, url, cache_dir, log_dir, buf, current_pid, len, &hit, &miss);
            }
        }
        time(&end_process_time); //Check end process time
        log_file = fopen(log_dir, "a");
        fprintf(log_file, "Terminated ServerPID : %d | Run time: %d sec | Request hit : %d, miss : %d\n",
            current_pid, (int)(end_process_time - start_process_time), hit, miss); //write which client was terminated, process execute time, hit, miss in logfile.txt
        fclose(log_file);
    }
    return; //end program
}

void main(void){
    char buf[HTTP_SIZE]; //buffer
    char web_dir[1024]; //Cache directory
    char log_dir[1024]; //Log directory
    int process_count = 0; //Count sub process
    int socket_fd, client_fd; //Socket and client file descriptor
    int len, len_buf; //length of buffer
    int opt = 0;

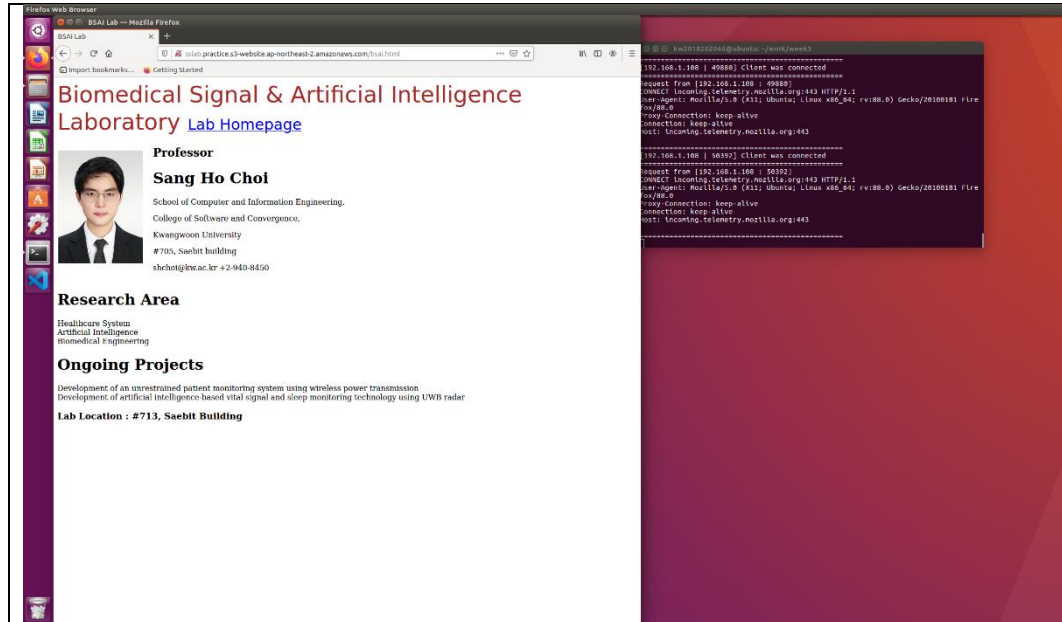
    struct sockaddr_in server_addr, client_addr; //server address and client address
    FILE *log_file; //using when write log file
    pid_t pid; //child PID
}
```

해당 함수는 2-3에서 만든 내용을 Ceck_Cache_Print_Web 함수로 옮겼기 때문에 내용이 적어졌다. 단 이번의 함수에서는 반복문을 통해 Request가 있는 동안에는 지속적으로 받는 것으로 변경되었다.

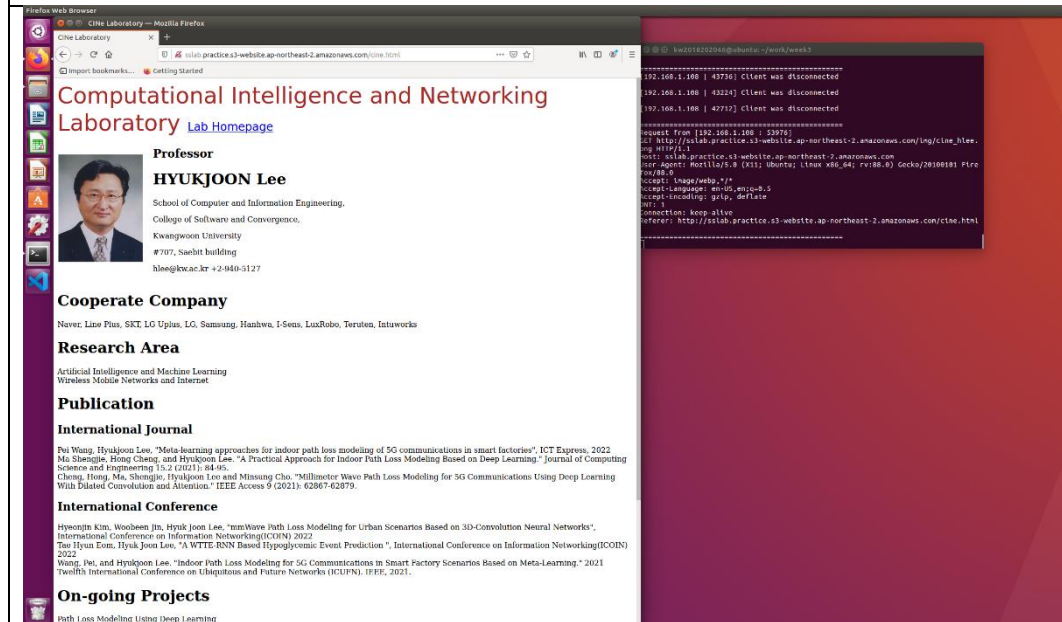





두 번째 URL을 입력하였을 때에는 초기에는 사진을 가져오는데 시간이 오래 걸렸지만 정상적으로 사진을 가져온 모습을 확인할 수 있었다.



firefox를 닫고 다시 해당 URL을 입력했을 때에는 아까보다 빠르게 사진을 가져오는 모습을 볼 수 있었다.



System Software Laboratory



Professor

TAESEOK KIM

School of Computer and Information Engineering,
College of Software and Convergence,
Kwangju National University
#963, Saebit building
tskim@kw.ac.kr +2-940-5774

Research Area

Operating System
Storage System
Embedded Software

Publication

Seongmin Kim, Kyusik Kim, Heeyoung Shin, and Taeseok Kim, "Practical Enhancement of User Experience in NVMe SSDs," Applied Sciences, 10(14), 2020.
Kyusik Kim, Seongmin Kim, and Taeseok Kim, "HMB I/O: Fast Track for Handling Urgent I/Os in Non-Volatile Memory Express Solid State Drives," Applied Sciences, 10(12), 2020.
Kyusik Kim and Taeseok Kim, "HMB in DRAM-less NVMe SSDs: Their usage and effects on performance," PLOS One, Mar. 3, 2020.

Ongoing Project

Collaborative Machine Learning Framework for Multidimensional Optimization of SSDs

Lab Location : #912, Saebit Building

NeverSSL - helping you get online - Mozilla Firefox

NeverSSL - helping you get online

What?

This website is for when you try to open Facebook, Google, Amazon, etc on a wifi network, and nothing happens. Type "http://neverssl.com" into your browser's url bar, and you'll be able to log on.

How?

neverssl.com will never use SSL (also known as TLS). No encryption, no strong authentication, no HTTPS, no HTTP2.0, just plain old unencrypted HTTP and forever stuck in the dark ages of internet security.

Why?

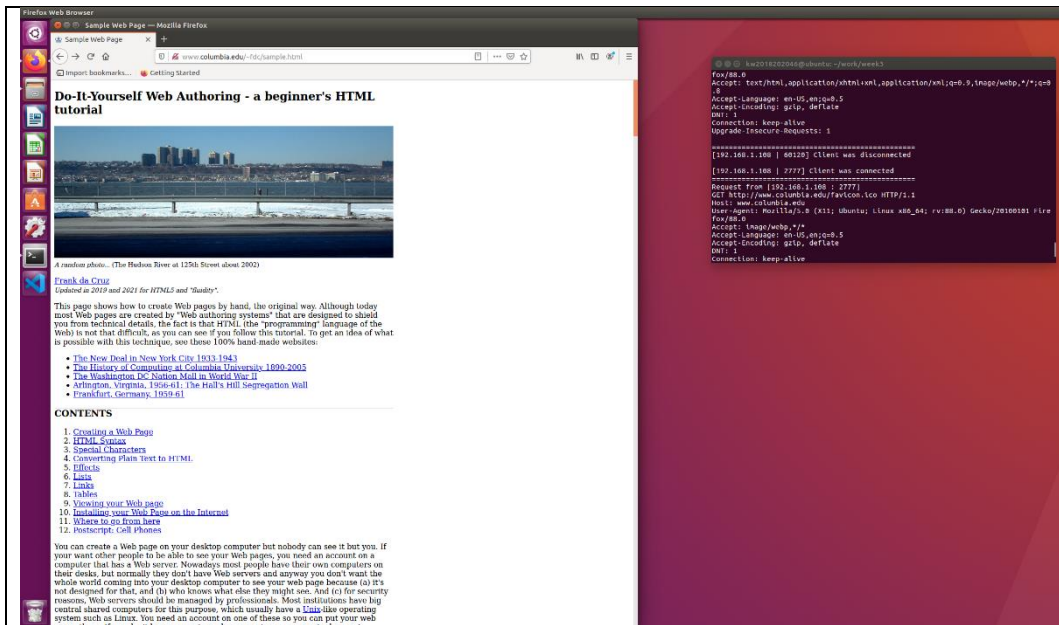
Normally, that's a bad idea. You should always use SSL and secure encryption when possible. In fact, it's such a bad idea that most websites are now using https by default.
And that's great, but it also means that if you're relying on poorly-behaved wifi networks, it can be hard to get online. Secure browsers and websites using https make it impossible for these wifi networks to send you to a login or payment page. Basically, these networks can't tap into your connection just like attackers can't. Modern browsers are so good that they can remember when a website supports encryption and even if you type in the website name, they'll use https.
And if the network never redirects you to this page, well as you can see, you're not missing much.
[Follow @neverssl](#)

```
kw2018022040@ubuntu:~/work/week3$
script language: en-us, encoding: 8.5
script encoding: glib, deflate
HTTP/1.1
Connection: keep-alive
Upgrade-Insecure-Requests: 1

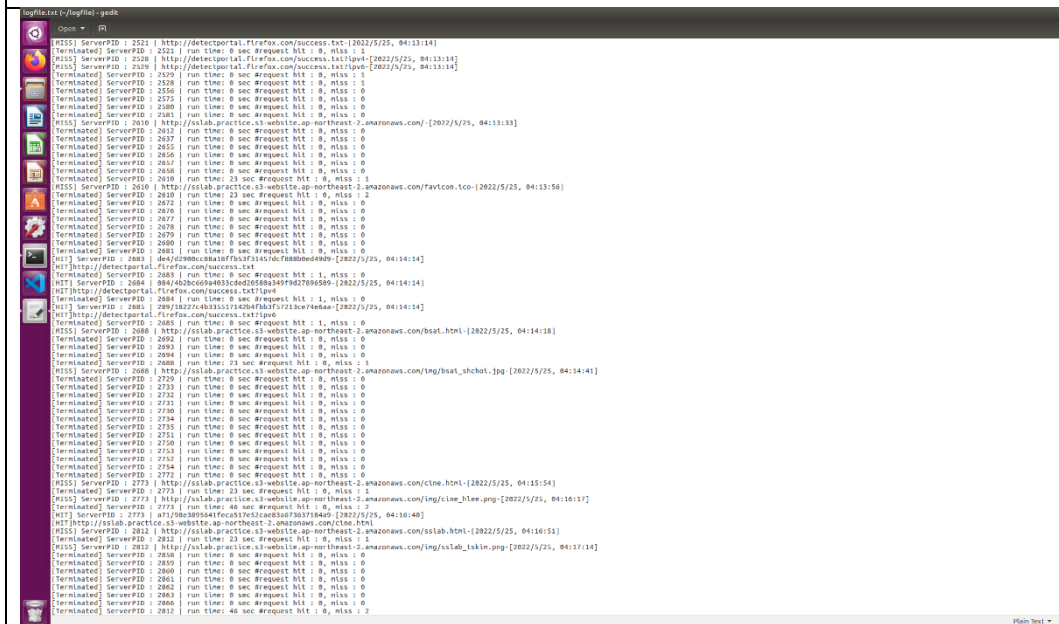
=====
Request from 192.168.1.108 | 55512
Host: http://ssl.practice.s3-website-ap-northeast-2.amazonaws.com/ssl/ssl_tk1
Host: ssl.practice.s3-website-ap-northeast-2.amazonaws.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:68.0) Gecko/20100801 Firefox/68.0
Accept: */*
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://ssl.practice.s3-website-ap-northeast-2.amazonaws.com/ssl/ssl.htm

=====
HTTP/1.1
Connection: keep-alive
Referer: http://neverssl.com/
Upgrade-Insecure-Requests: 1

192.168.1.108 | 61144 Client was disconnected
192.168.1.108 | 62168 Client was connected
=====
Request from 192.168.1.108 | 62168
Host: http://shiningastoundingwonderfulmorning.neverssl.com/feicon.ico HTTP/1.1
Host: shiningastoundingwonderfulmorning.neverssl.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:68.0) Gecko/20100801 Firefox/68.0
Accept: */*
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://shiningastoundingwonderfulmorning.neverssl.com/online
```



다른 페이지들도 사진이 있으면 비교적 오래 걸리긴 했지만 정상적으로 response를 가져오는 모습을 보였다.



log_file을 보면 hit와 miss 모두 잘 받아지며, 한 번의 프로세스에서 html과 사진을 가져오는 경우에는 해당 숫자가 request를 요청한 만큼 뜬다.

