

시스템 프로그래밍 실습 2-3 과제

이름 : 이준휘

학번 : 2018202046

교수 : 최상호 교수님

강의 시간 : 화

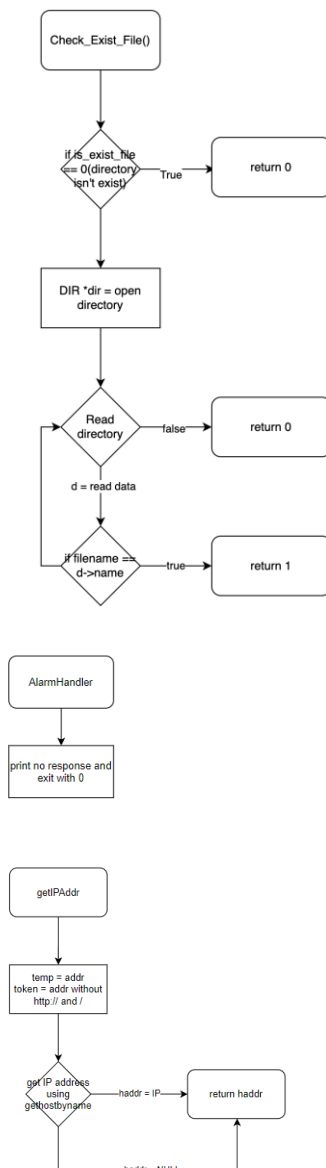
실습 분반 : 목 7, 8

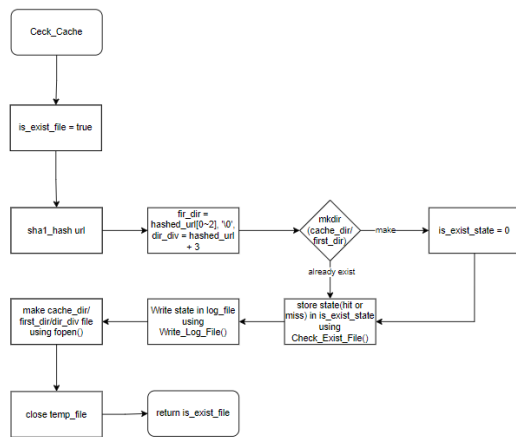
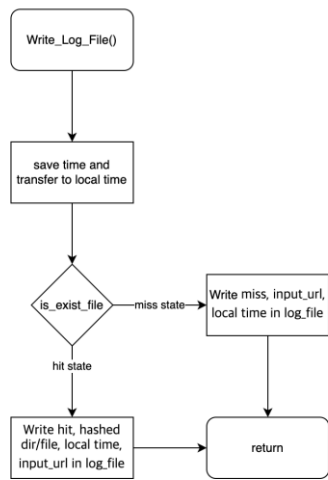
1. Introduction

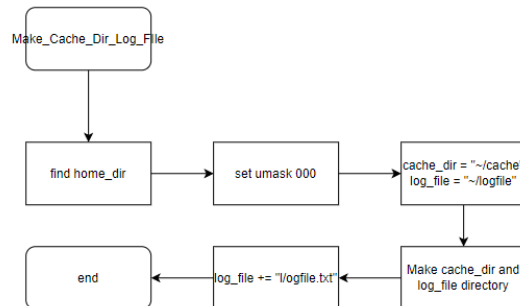
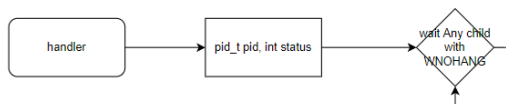
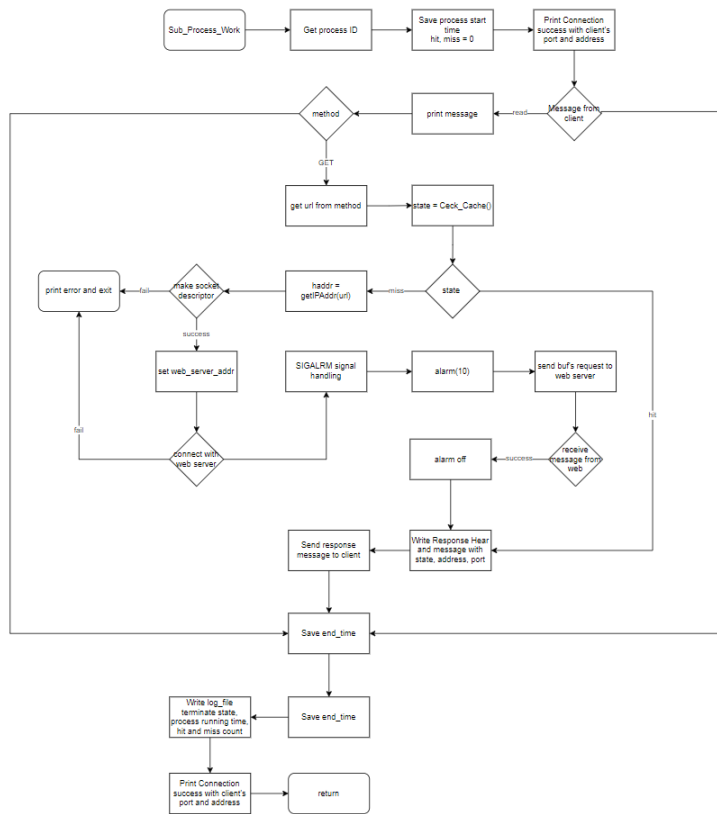
해당 과제는 기존에 2-2에서 만든 과제에서 확장된다. hit와 miss를 판별한 후 만약 miss 일 경우에는 web의 IP 주소를 알아낸다. 알아낸 주소를 바탕으로 web server와의 connection을 열고 연결을 시도한다. 만약 연결을 시도하였을 때 10초동안 반응이 없을 경우 SIGALARM이 발생하며 signal()함수와 자체적으로 만든 함수를 통해 error를 처리한다. 이외의 동작은 기존과 동일하게 유지한다.

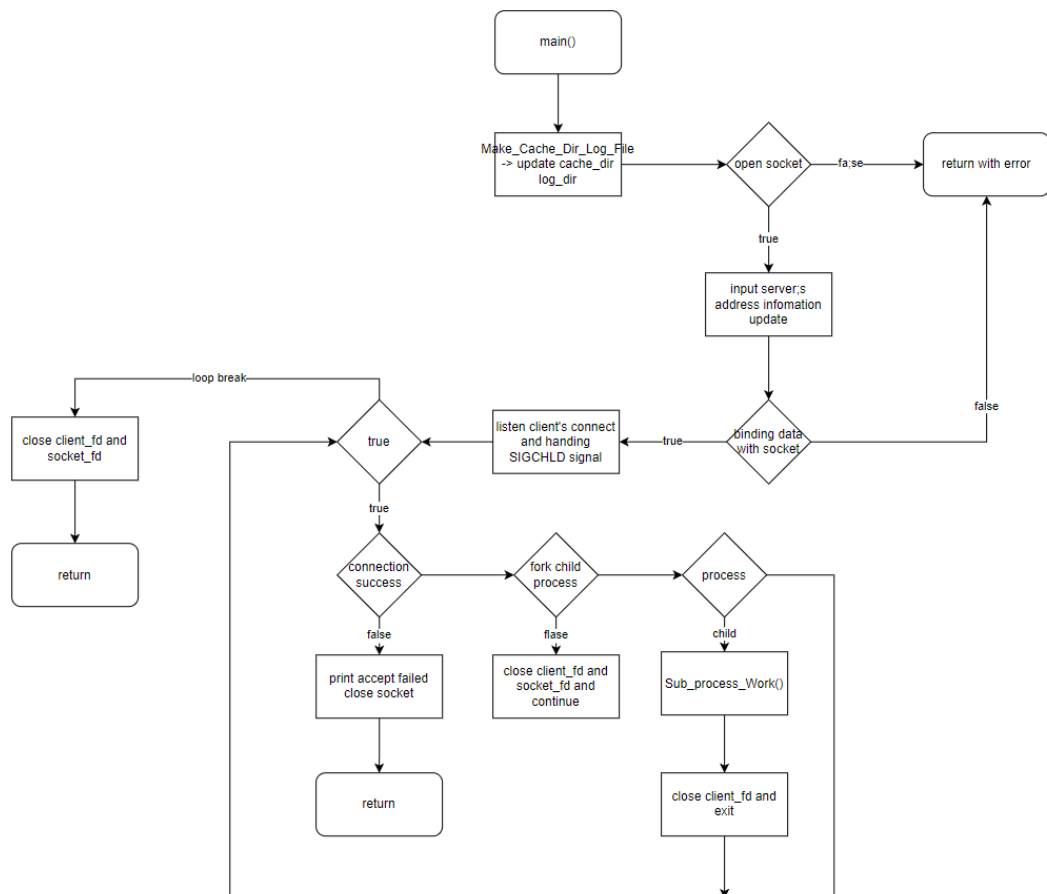
2. Flow Chart

- 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(char *url, char *cache_dir, char *log_file, int current_pid, int *hit, int *miss){

    char[60] hashed_url;

    char[4] first_dir;

    char *dir_div;

    char[100] temp_dir;

    int is_exist_file = 1;

    File *temp_file;

    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;

temp_file = make temp_dir file and open file;

temp_file close;

return is_exist_file;

}

```

```

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

char response_header[BUFSIZE] = { 0 };

char response_message[BUFSIZE] = { 0 };

char temp[BUFSIZE] = { 0 };

char method[BUFSIZE] = { 0 };

char url[BUFSIZE] = { 0 };

char h_buf[BUFSIZE];

char* haddr;

char *token = NULL;

int len, h_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;

if read data from client_fd to buf{

    print Request message;

    method = Request message's method;

    if(method == GET){

        url = Request message's url

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

        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 10 sec;

            write buf's request to web server;

            if(receive response)

                alarm off;

        }

    }

}

write response message and header with client address, port, and state;

```

```

        Send message to client;
    }
}

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 SIGCHID 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;

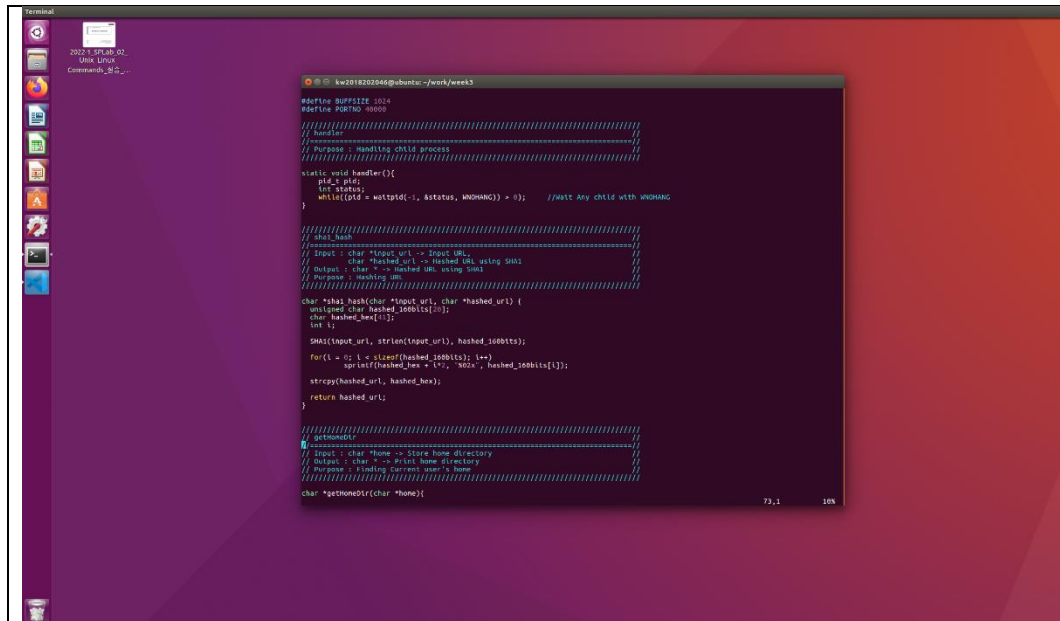
    }

    close socket file descriptor;

}

```

4. 결과 화면



```
#define SUPP1234 1024
#define PORTNO 40000

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

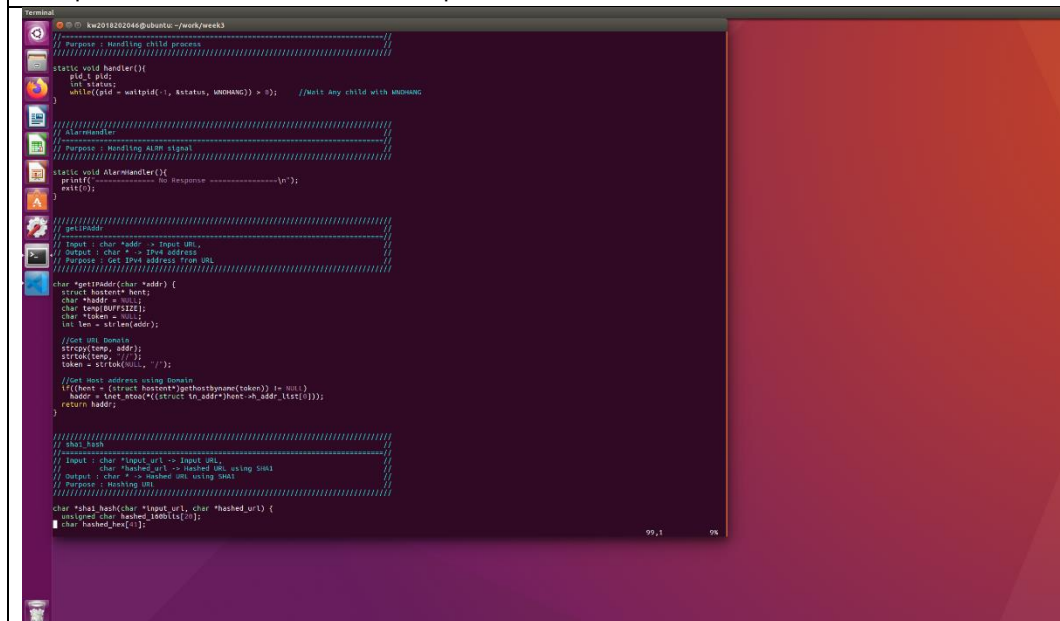
static void handler(){
    pid_t pid;
    int status;
    while(pid = waitpid(-1, &status, WNOHANG)) > 0; //Wait Any child with WNOHANG
}

// shai_hash
// Purpose : Handling child process
// Input : char *input_url -> Input URL
// Output : char *hashed_url -> Hashed URL using SHA1
// Purpose : Hashing URL

char *shai_hash(char *input_url, char *hashed_url){
    unsigned char hashed_100bits[100];
    char hashed_hex[100];
    int i;
    SHA1(input_url, strlen(input_url), hashed_100bits);
    for(i = 0; i < sizeof(hashed_100bits); i++)
        sprintf(hashed_hex + i*2, "%02x", hashed_100bits[i]);
    strcpy(hashed_url, hashed_hex);
    return hashed_url;
}

// gethomeDir
// Input : char *home -> Store home directory
// Output : char * -> Print home directory
// Purpose : Finding current user's home
char *gethomeDir(char *home){
```

해당 함수는 signal함수에서 handling을 위해 만들어진 함수다. 해당 함수에서는 waitpid(WNOHANG)을 통해 child process의 종료를 확인시켜주는 역할을 수행한다.



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

// AlarmHandler
// Purpose : Handling child process
// Input : char *input_url -> Input URL
// Output : char *hashed_url -> Hashed URL using SHA1
// Purpose : Hashing URL

static void AlarmHandler(){
    printf("No Response\n");
    exit(0);
}

// getIPAddr
// Input : char *addr -> Input URL
// Output : char * -> IP address
// Purpose : Get IP address from URL

char *getIPAddr(char *addr){
    struct hostent *hent;
    char *haddr = NULL;
    char temp[BUFFSIZE];
    char *token = NULL;
    int len = strlen(addr);
    //Get IP Domain
    strcpy(temp, addr);
    strtok(temp, "/");
    token = strtok(temp, "/");
    //Get Host address using Domain
    if(hent = (struct hostent *)gethostbyname(token)) != NULL{
        haddr = inet_ntoa(*(struct in_addr *)hent->h_addr_list[0]);
        return haddr;
    }
}

// shai_hash
// Purpose : Handling child process
// Input : char *input_url -> Input URL
// Output : char *hashed_url -> Hashed URL using SHA1
// Purpose : Hashing URL

char *shai_hash(char *input_url, char *hashed_url){
    unsigned char hashed_100bits[100];
    char hashed_hex[100];
```

해당 함수 AlarmHandler()와 getIPAddr()함수는 이번 2-3에 새롭게 추가된 함수다. AlarmHandler()에서는 SIGALRM 에러가 왔을 경우 No response를 출력하고 exit()을 동작시킴으로써 error를 handling한다. getIPAddr()함수에서는 url을 입력으로 받으며 url에서 http://을 tokenize하고 /까지를 tokenize하여 순수한 주소를 가져온다. 이를 가지고 gethostbyname()함수를 통해 host의 정보를 가져와 IP를 추출한다.


```
void sub_process_work(int client_fd, struct sockaddr_in client_addr, char *buf, char *cache_dir, char *log_dir){
    char response_header[BUFFSIZE] = {0}; //response message / header
    char response_message[BUFFSIZE] = {0}; //response message / body
    char buf[BUFFSIZE] = {0}; //receive method
    char method[BUFFSIZE] = {0}; //receive method
    char url[BUFFSIZE] = {0}; //receive url
    char *token = NULL; //token
    int hit; //hit or miss
    int state; //state
    int hit = 0, miss = 0; //count hit and miss

    FILE *log_file; //log file's path
    time_t current_pid = getpid(); //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(buf, BUFFSIZE); //buffer clear
    printf("%s | %d | client was connected\n", inet_ntoa(client_addr.sin_addr), client_addr.sin_port); //Print which client is connected and pid number

    //read data from client file descriptor
    if((len = read(client_fd, buf, BUFFSIZE)) > 0){
        strcpy(tmp, buf); //Copy message and print it
        printf("Request from %s : %d\n", inet_ntoa(client_addr.sin_addr), client_addr.sin_port);
        printf("%s | %d |", buf);
        printf("-----\n");

        //Divide method and url from message
        token = strtok(tmp, " ");
        //Method is GET, use cache and send response
        if(strncmp(method, "GET", 4) == 0){
            token = strtok(tmp, " ");
            strcpy(url, token);
            state = Check_Cache(url, cache_dir, log_dir, current_pid, hit, miss);

            //response message
            sprintf(response_message,
                "HTTP/1.1 200 OK\n"
                "Content-Type: %s\n"
                "Content-Length: %d\n"
                "Content: %s\n",
                "text/html",
                strlen(response_message));

            //send data to client
            write(client_fd, response_header, strlen(response_header));
            write(client_fd, response_message, strlen(response_message));

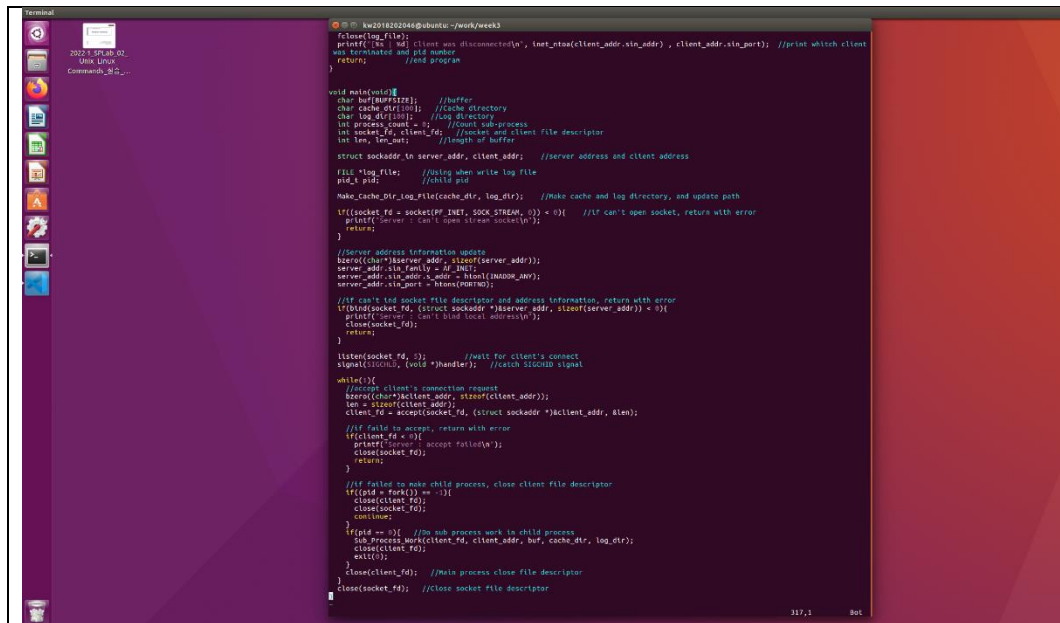
            time(&end_process_time); //check and process time
            log_file = fopen(log_dir, "a");
            printf("Log file : %s\n", log_file);
            printf("%s | %d | run time: %d sec | request hit : %d, miss : %d\n",
                current_pid, (int)(end_process_time - start_process_time), hit, miss); //write which client was terminated, process execute time, hit, miss in log file
            fclose(log_file);

            printf("%s | %d | client was disconnected\n", inet_ntoa(client_addr.sin_addr), client_addr.sin_port); //Print which client was terminated and pid number
            return; //end program
        }

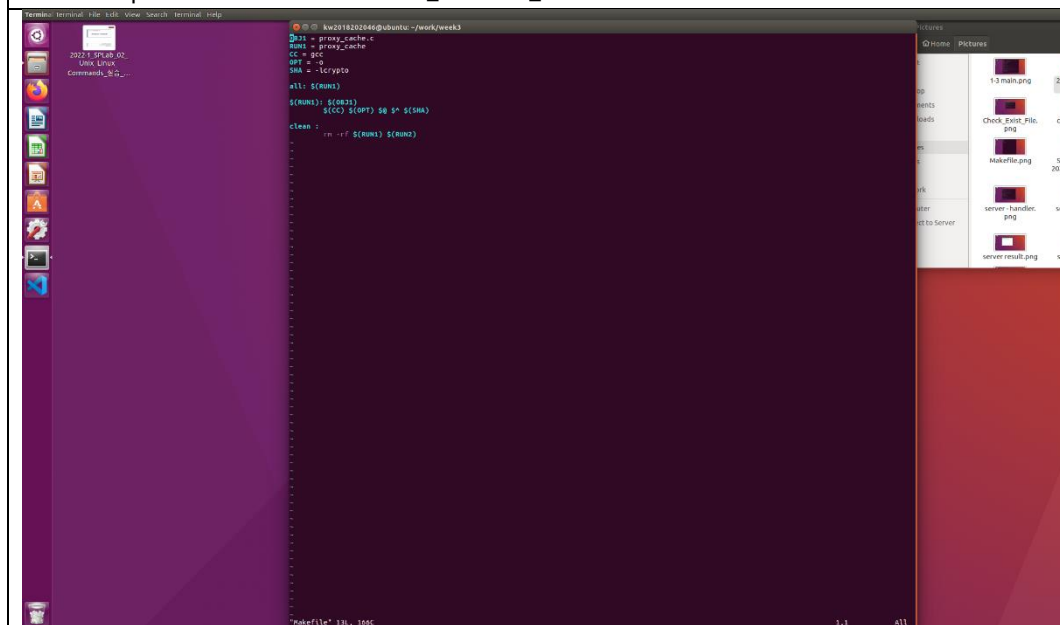
    }

    void main(void){
        char buf[BUFFSIZE]; //buffer
    }
}
```

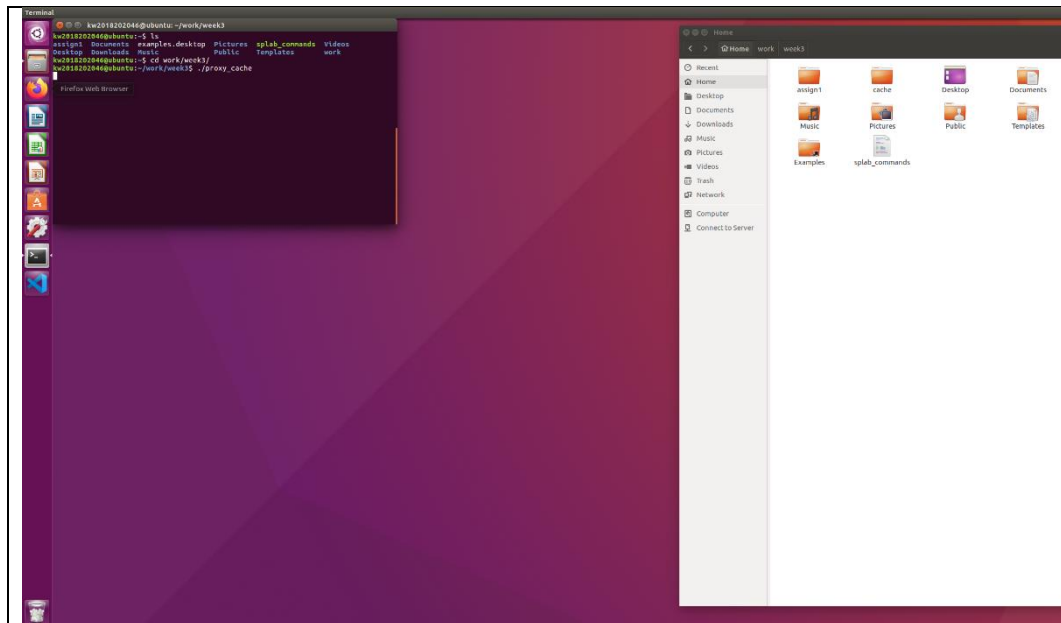
해당 함수는 2-2에서 주어진 함수에서 일부를 추가하였다. Ceck_Cache()함수를 통하여 hit, miss 상태를 판별한 후, miss인 상태일 때 추가적인 작업을 수행한다. getIPAddr()함수를 통해 URL의 IPv4 주소를 찾아낸 후 해당 주소에 대한 socket을 연결한다. 이 때 signal을 통해 SIGALRM에 대한 handler를 설치하고 10초동안 request에 대한 답이 없을 시 에러를 handling하도록 한다. 이외의 부분은 기존과 동일하게 수행한다.



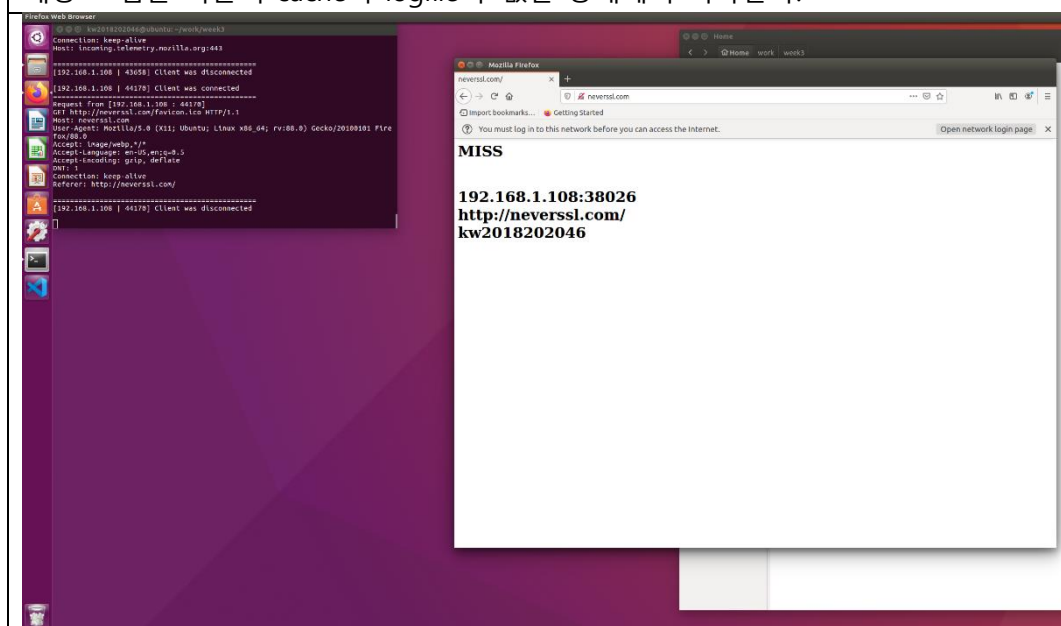
해당 그림은 server의 메인함수다. 해당 함수에서는 socket을 열고 binding을 통해 server의 정보를 묶는다. 그리고 listen을 통해 연결을 받는다. 연결을 accept할 경우 child process를 생성하여 Sub_Process_Work 작업을 수행한다.



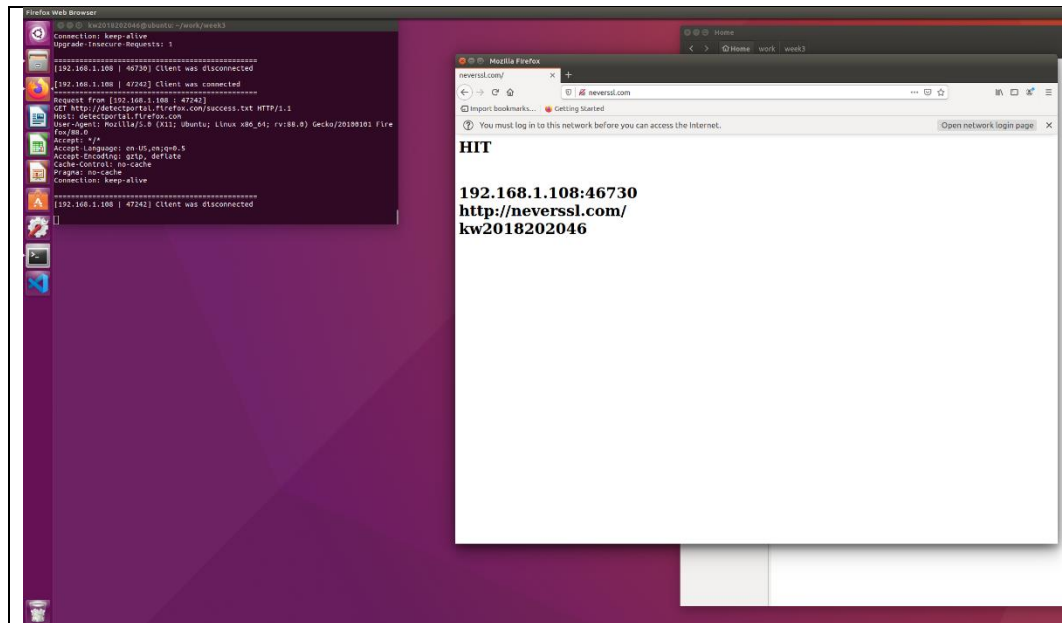
해당 파일은 Makefile이다. proxy_cache.c의 파일을 proxy_cache 실행파일로 만드는 역할을 수행한다.



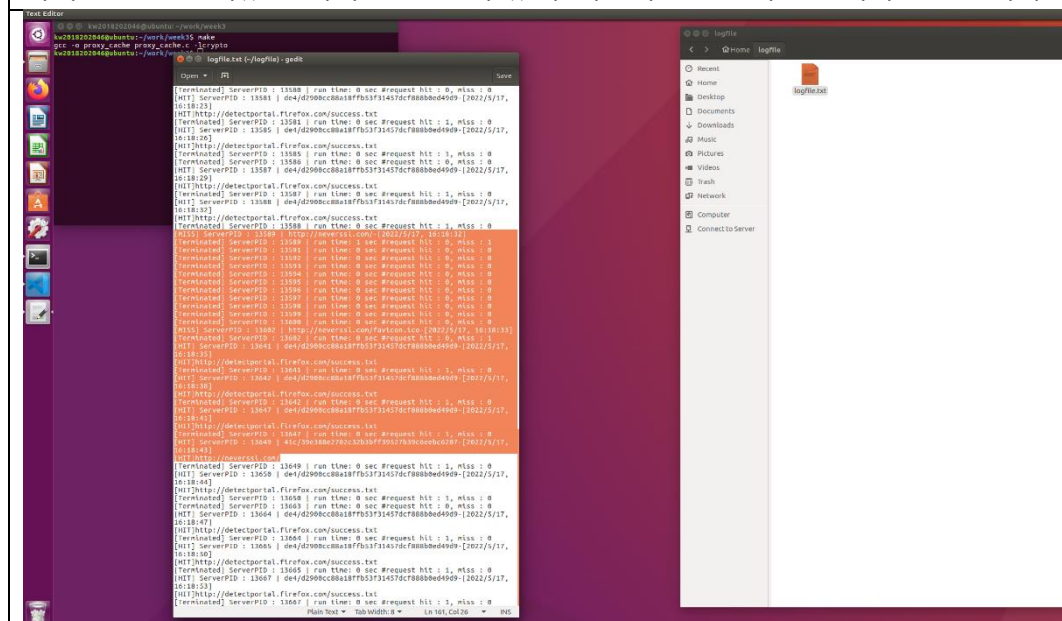
해당 그림은 기존의 cache와 logfile이 없는 상태에서 시작한다.



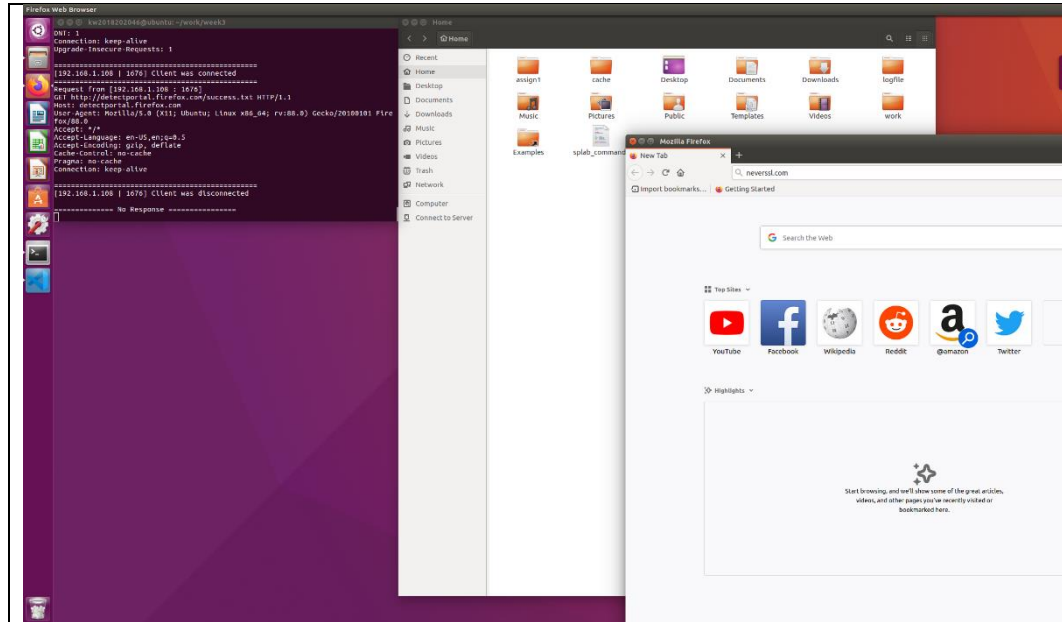
proxy_cache를 실행 후 <http://neverssl.com>을 입력하였을 때 기존과 동일하게 miss 상태를 출력하는 것을 알 수 있다.



다시 한번 실행하였을 때에는 cache가 있기 때문에 HIT 상태를 출력하는 모습이다.



logfile.txt에서도 또한 기존과 같이 로그가 정상적으로 작성된 것을 확인할 수 있었다.



같은 환경에서 sleep(5)를 통해 약간의 지연을 시킨 후 인터넷을 끊었을 때의 결과다. 해당 결과에서는 인터넷이 끊겨 No Response가 출력된 것을 확인 할 수 있었다. 이는 web server에 접근을 하였지만 실패했다는 이야기로 정상적으로 과제의 목표를 완수했다는 의미다.

5. 고찰

해당 과제를 통해서 proxy server와 web server간의 연결을 기존 client.c를 통해 만들었다. 이를 통해 이전 내용을 복습할 수 있는 과제였다. 또한 기존에 signal()함수에 대해 왜 사용하는지 몰랐지만, 이번 과제를 통해 해당 함수가 특정 signal에 대해 지속적으로 확인하면서 error를 처리해줄 수 있다는 사실을 알게 되었다. 그리고 gethostbyname()이라는 함수를 통해 url로부터 IP주소를 알아낼 수 있다는 사실 또한 알게 되었다. 이처럼 기존의 모르고 넘어간 점을 다시 알고, 새로운 사실 또한 알 수 있었던 과제였다.

6. Reference

강의 자료만을 참고