시스템프로그래밍

Assignment - Proxy server 3-2

보고서

학과: 컴퓨터정보공학부

담당교수: 최상호 교수님

분반: C

학번: 2021202077

성명: 최승환

1. Introduction

본 과제에서는 Proxy 3-1에서 구현한 semaphore를 이용해 로그파일에 접근을 제한하는 코드를 기반으로 로그파일 작성을 스레드에서 하는 기능을 구현하는 것을 목표로 한다.

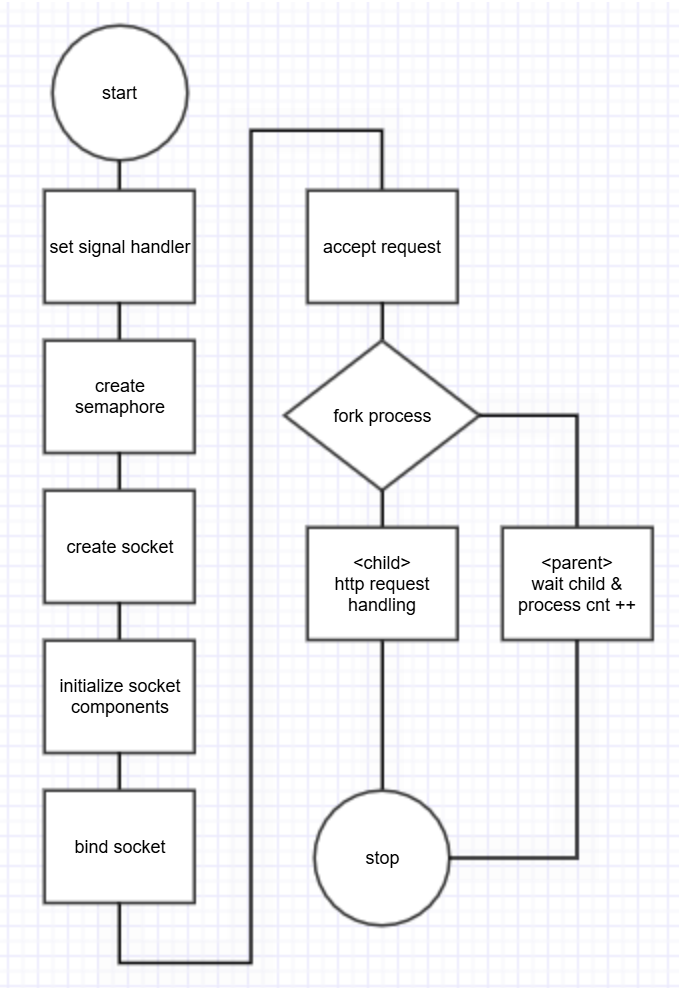
Server는 socket을 생성하고 client request를 대기한다. 이후 request를 받아들이고 url을 전달받아 HIT/MISS를 판별하여 HTTP client에 전송한다. HTTP Client는 Connect 이후, url을 server에 전송하고 HIT/MISS 플래그를 받아온다. 결과적으로 웹서버에서 받은 response를 웹 페이지에 표시하는 동작을 수행한다. Hit인 경우 캐시파일의 정보를 클라이언트에게 전달하고, miss인 경우 캐시파일을 생성하고 웹 서버에서 받은 정보를 캐시파일에 작성한다.

주요 구현 포인트는 스레드를 이용한 로그파일 작성과 세마포어를 이용해 한번에 하나의 수정만 가능하도록 하는 것이다. 이를 통해 Thread의 활용, http 통신, semaphore 활용과 프록시 서버 기초에 대한 종합적 학습을 할 수 있다.

2. proxy #3-2

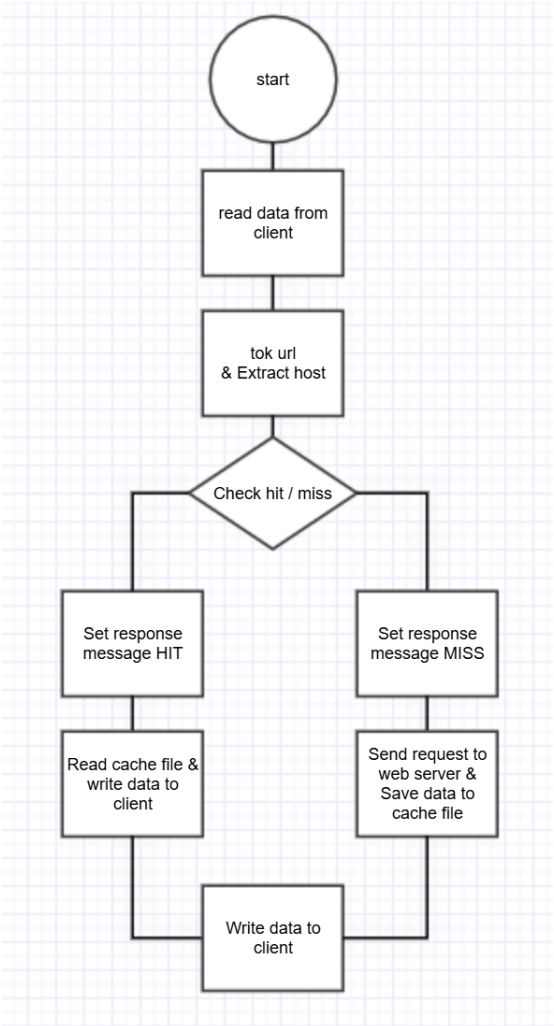
가. Flowchart

1) Flowchart - main()



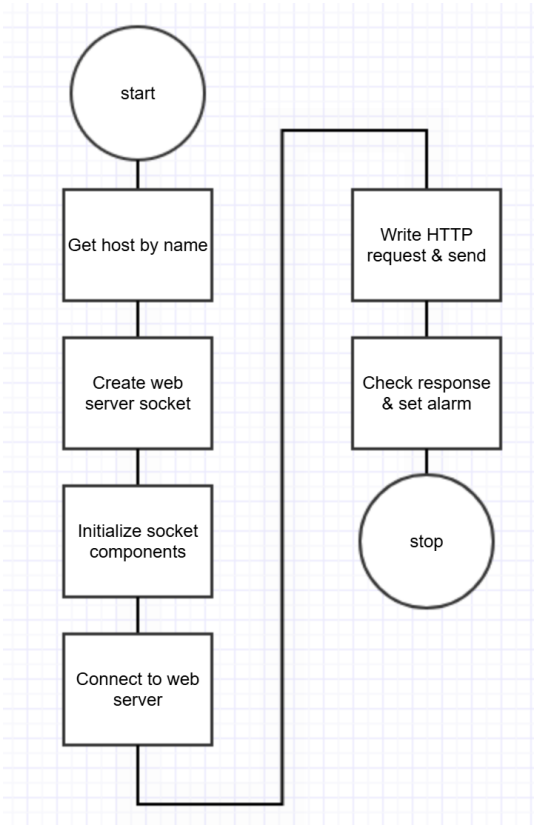
main() 함수는 먼저 시그널 핸들러 함수를 실행하여 시그널이 입력되었을 때 동작을 초기화한다. 그리고, 세마포어를 생성하여 세마포어 사용을 선언한다. 소켓을 생성하여 클라이언트 웹 브라우저로 부터의 request를 대기한다. 이후 소켓의 내부의 변수들을 초기화한다. 서버에 server_addr을 bind하며, client request가 입력될 때까지 대기한다. 웹 브라우저 클라이언트를 반복적으로 accept하여 child process를 생성하고 http request를 처리한다. Parent process는 child process의 종료를 대기하고 process 카운터를 증가한다.

2) Flowchart - http_client_request()



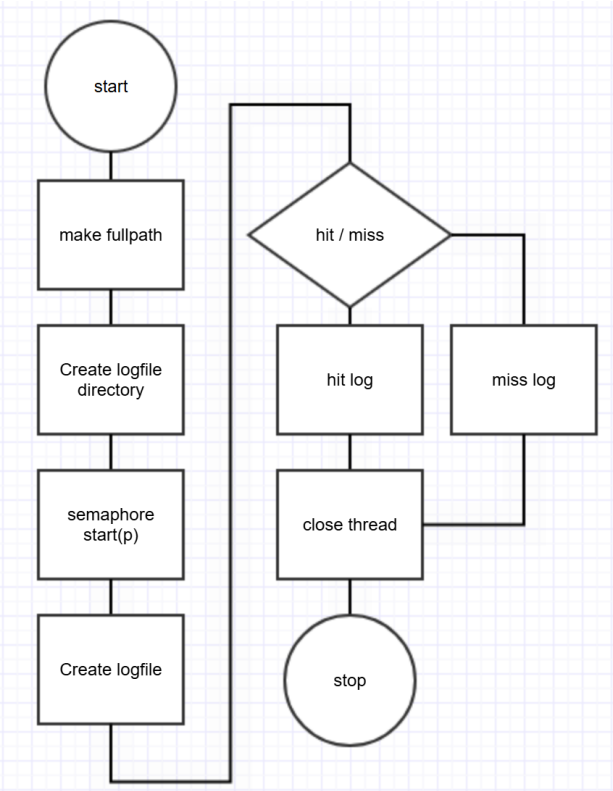
http_client_request 함수는 웹 브라우저로부터 입력된 request를 처리하는 함수이다. 클라이언트로부터 데이터를 읽고, host 정보를 추출한다. 이를 기반으로 hit/miss 판별을 하여 분기한다. hit인 경우 해당하는 캐시파일을 읽어 클라이언트에 전송한다. Miss인 경우 웹 서버에 request를 전송한다. 그리고 받은 정보를 캐시파일에 저장한다. 이후 로그파일을 생성하여 결과를 작성하고, 클라이언트 서버에 결과를 전송하여 화면에 출력한다.

3) Flowchart – send_http_wirh_timeout()



Send_http_with_timeout 함수는 miss일 때 웹 서버에 request를 전송하고 응답을 받는 함수이다. 소켓을 생성하고 각 부분을 초기화한다. 이후 웹서버에 연결하고, HTTP request를 작성하여 전송한다. 응답을 확인하고 alarm을 초기화하여 알림이 작동하지 않도록 한다. 10초 이상 웹서버의 응답을 받지 못하면 응답 없음 메시지를 출력한다.

4) Flowchart - LogfileThread()



LogfileThread() 함수는 스레드의 생성과 함께 수행되는 함수다. 기본적으로 Logfile() 함수의 동작과 같다. 먼저, logfile 디렉토리의 주소를 조합하고, 디렉토리를 생성한다. 그리고 세마포어를 생성하여 로그파일 입력을 하나씩 할 수 있도록 lock 한다. 로그파일을 생성하고 hit/miss를 판단하여 로그파일에 기록한다. 이후, 스레드를 종료하고 탈출한다.

나. Pseudo Code – proxy_cache.c

1) Main()

```
int main(){
    // SIGCHLD handler
    signal(SIGCHLD, sigchld_handler);
    signal(SIGALRM, my_alarm);

    Create semaphore

    Create Socket for listen web browser

    Initialize socket components

    Bind socketaddr to socket

    Listen Client request

    while(Get client request){
        Accept client
        if(Accept error){
            printf("Server : accept failed\n");
            return 0;
        }
        set client addr

        pid_t pid = fork();
        if(fork failed){
            printf("Fork failed\n");
            close(client_fd);
            continue;
        }
        if(child process){
            Handle http client request
            close(client_fd);
            exit(0);
        }
        else if(Parent process){
            close(client_fd);
            processcnt++;
        }
    }
    close(socket_fd);
    return 0;
} // end of main
```

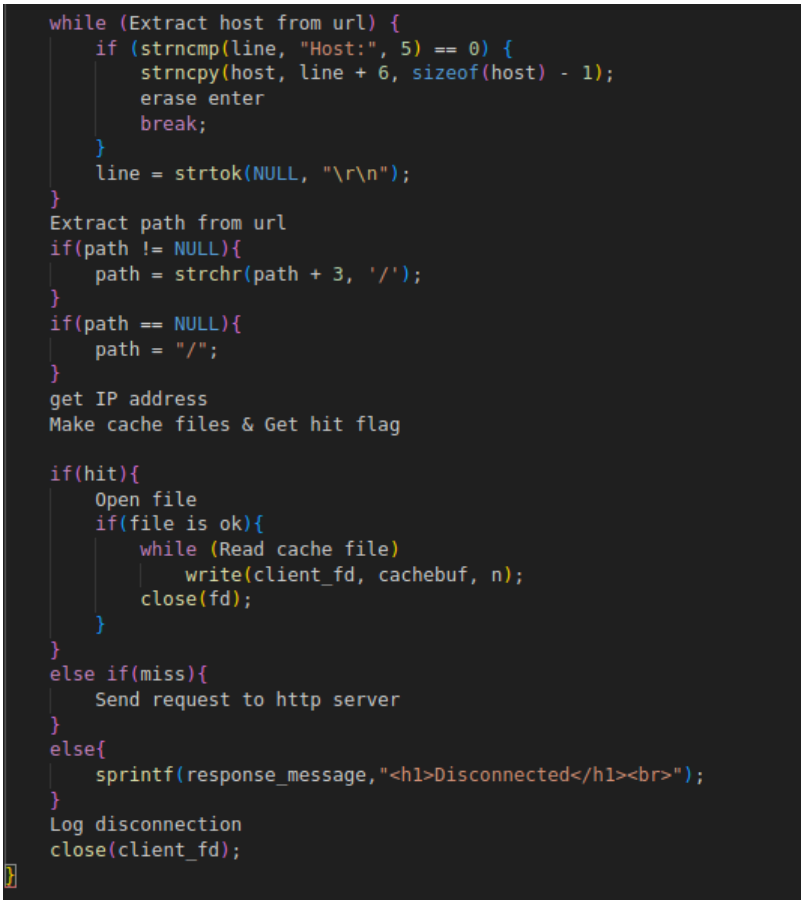
- Signal handler 함수를 호출하여 시그널 호출시 동작을 정의한다.
- Semaphore를 생성한다.
- 웹 브라우저 클라이언트의 연결을 받을 소켓을 생성한다.
- 소켓 구성 변수들을 초기화한다.
- 소켓을 바인딩한다. 바인딩 오류 시 에러코드를 출력한다.
- 웹 브라우저 클라이언트로부터 request를 대기한다.
- 클라이언트를 지속적으로 accept하면서 fork()로 child process를 생성한다.
- Pid의 값에 따라 child, parent process의 동작을 수행한다.
- Parent는 sigchld 에 따라 child 종료를 대기하지 않고 여러 프로세스를 실행

할 수 있도록 한다.

- Processcnt를 증가시켜 개수를 카운트한다.

2) http_client_request()

```
// Fuction for http client request handling
void http_client_request(int client_fd, struct in_addr inet_client_address, struct sockaddr_in client_addr){
    Read data from client
    if(read error){
        close(client_fd);
        return;
    }
    tok url
    if(tok error){
        close(client_fd);
        return;
    }
    tok method from http

    if(url is NULL){
        close(client_fd);
        return;
    }
    strcpy(url, tok);
    printf("url = %s\n",url);

    if(url has favicon.ico){
        printf("Favicon request ignored.\n");
        close(client_fd);
        return;
    }
    if(url has firefox.com){
        printf("Firefox request ignored.\n");
        close(client_fd);
        return;
    }
    tok by line
```



```
    while (Extract host from url) {
        if (strncmp(line, "Host:", 5) == 0) {
            strncpy(host, line + 6, sizeof(host) - 1);
            erase enter
            break;
        }
        line = strtok(NULL, "\r\n");
    }
    Extract path from url
    if(path != NULL){
        path = strchr(path + 3, '/');
    }
    if(path == NULL){
        path = "/";
    }
    get IP address
    Make cache files & Get hit flag

    if(hit){
        Open file
        if(file is ok){
            while (Read cache file)
                write(client_fd, cachebuf, n);
            close(fd);
        }
    }
    else if(miss){
        Send request to http server
    }
    else{
        sprintf(response_message,"<h1>Disconnected</h1><br>");
    }
    Log disconnection
    close(client_fd);
}
```

- 웹 브라우저로부터 데이터를 읽어온다. Read 오류 발생시 에러코드를 출력하고 client를 종료한다.
- 콘솔에 request data를 출력하여 디버깅에 용이하도록 한다.
- url을 파싱하여 method와 host와 path를 추출한다.
- Subprocess()함수를 통해 hit/miss 여부를 확인한다.
- Hit일 경우 : 해당하는 디렉토리의 캐시파일의 데이터를 클라이언트에 전송한다.
- Miss일 경우 : 웹 서버에 request를 전송하여 받은 데이터를 캐시파일을 생성하여 작성한다. 이후 클라이언트에 전송한다.
- 로그파일에 hit, miss 여부를 작성한다.

3) send_http_with_timeout()

```
// Function for sending http request to web server
int send_http_with_timeout(const char* host, const char* path, int client_fd) {
    get host by name
    if (host error) {
        perror("gethostbyname err");
        return -1;
    }

    Create server Socket
    if (socket error) {
        perror("socket err");
        return -1;
    }

    Initialize server components
    set alarm after 10 sec

    connect to server
    if (connect error) {
        perror("connect err");
        close(sockfd);
        return -1;
    }

    while(Get data from web server){
        send data to client
        write data to cache file
    }
    if (write error) {
        perror("write err");
        close(sockfd);
        return -1;
    }

    Check response and set alarm
    if (!read error) {
        reset alarm
    }
    else {
        read error
    }
    close(sockfd);
    return 0;
}
```

- gethostbyname()으로 정보를 가져온다. 함수 오작동 시 에러코드를 출력한다.
- 서버 소켓을 생성하고 구성 변수들을 초기화한다.
- Alarm(10) 함수를 이용해 10초 이후 알람 시그널이 작동하도록 한다.
- 서버에 연결하고 연결 불량 시 에러코드를 출력한다.
- 웹서버에 보낼 http request를 작성하고 전송한다. 전송 오류 시 에러코드를 출력한다.
- While 반복을 이용하여 buffsize 단위로 통신한다.
- 응답을 확인했으면 알람을 초기화한다.

4) Sigint_handler()

```c
// Sigint handler
void sigint_handler(int signo){
    // Calculate runtime
    time_t end_time = time(NULL);
    int runtime = (int)(end_time - start_time);
    // Log terminate info
    Terminate(&runtime, &processcnt);
    printf("\n[Terminated] Run time: %d sec, #SubProcess: %d\n", runtime, processcnt);
    exit(0);
}
```

- Runtime을 계산하고 이를 terminate 함수에 전달한다.
- 콘솔창에 terminate 정보를 출력한다.

5) LogfileThread()

```c
// Function for log information
void* LogFileThread(void* arg){
    LogArgs* logArgs = (LogArgs*)arg;

    Make fullpath
    if(logfile not exist){
        mkdir(logdir, 0777);
    }
    umask(old_umask);

    Calculate time

    printf("*PID# %d is waiting for the semaphore.\n", getpid());
    p(semid);
    printf("*PID# %d create the *TID# %ld.\n", getpid(), pthread_self());
    Open logfile.txt
    if(file is ok){
        if(hit){
            Log hit info
        }
        else{
            Log miss info
        }
        fclose(fp);
    }
    else{
        perror("logfile open failed");
    }
    printf("*TID# %ld is exited\n", pthread_self());
    printf("*PID# %d exited the critical zone.\n", getpid());
    v(semid);

    free(logArgs); // Free dynamic alloc
    pthread_exit(NULL); // Close thread
}
```

- LogArg 변수를 가져온다.
- ~/logfile 경로를 만들어 변수에 저장한다.
- Logfile 디렉토리가 없을 경우 새로운 디렉토리를 생성한다.

- Umask를 원래대로 되돌린다.
- 시간을 계산하여 로그 파일 작성시 사용할 수 있도록 한다.
- P 함수를 호출하여 critical section을 잠근다.
- Hit, miss 여부에 따라 다른 값을 로그파일에 작성한다. Hit이면 파일 경로와 url, miss면 url을 작성한다.
- 파일을 종료하고 v함수를 호출하여 critical section의 잠금을 해제한다.

6) Hit()

```
// Function for log HIT & MISS
int hit(const char *subdir, const char *dir_name, const char *cache_file, const char *url, int client_fd){
    make fullpath

    Check hit or miss

    Log with Thread

    Create thread
    Execute thread function
    Close thread

    Return hit flag
}
```

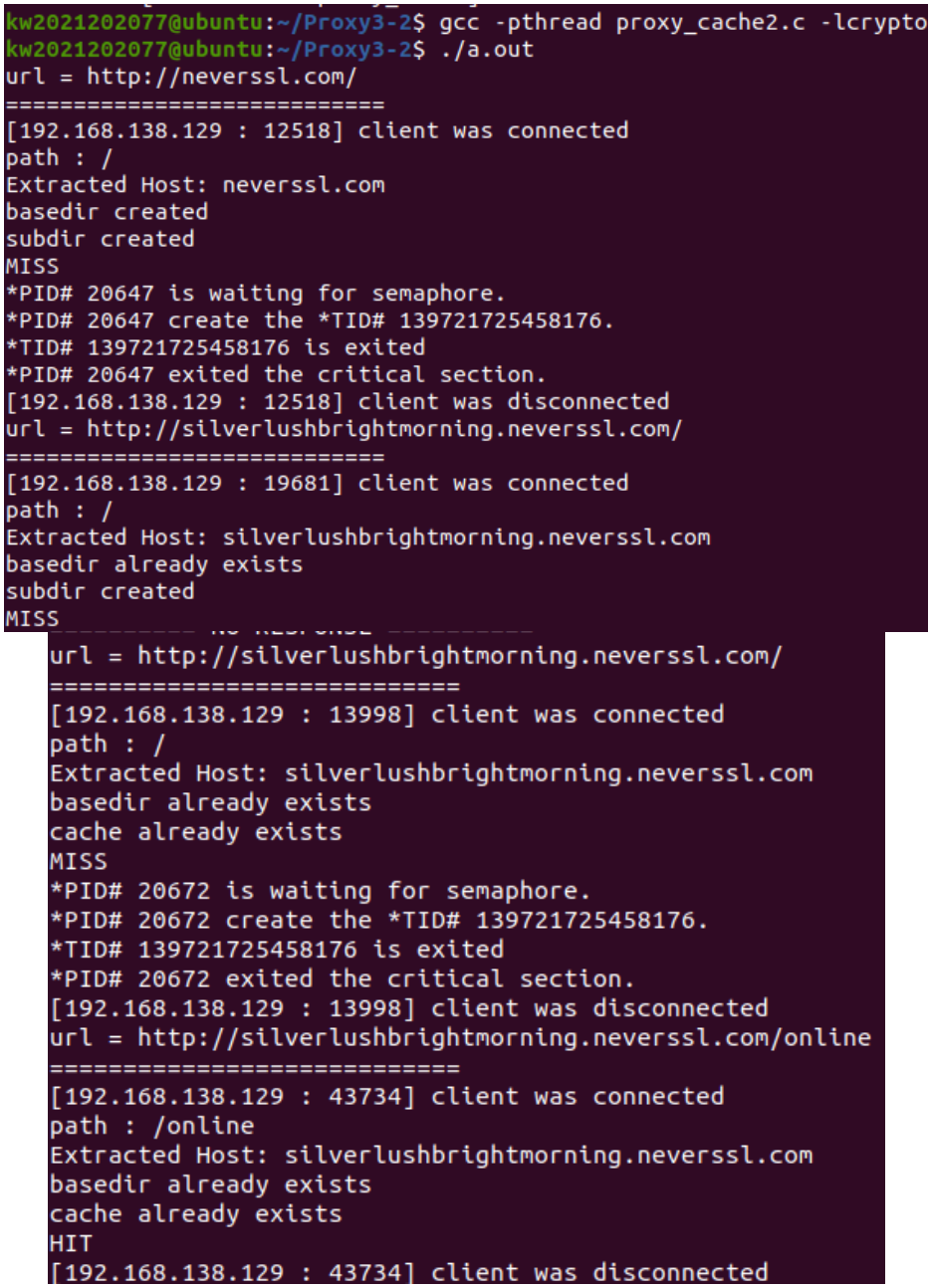
- 파일 주소를 구성하는 인자를 입력 받는다.
- 캐시 파일의 절대 주소를 조합한다.
- 해당 파일의 hit/miss 여부를 판단한다.
- 로그할 변수들을 조작한다. 이후 스레드를 생성한다.
- 스레드 함수를 수행하며 로그파일에 기록을 작성한다.
- hit인지 아닌지 판별할 수 있는 플래그를 리턴한다.

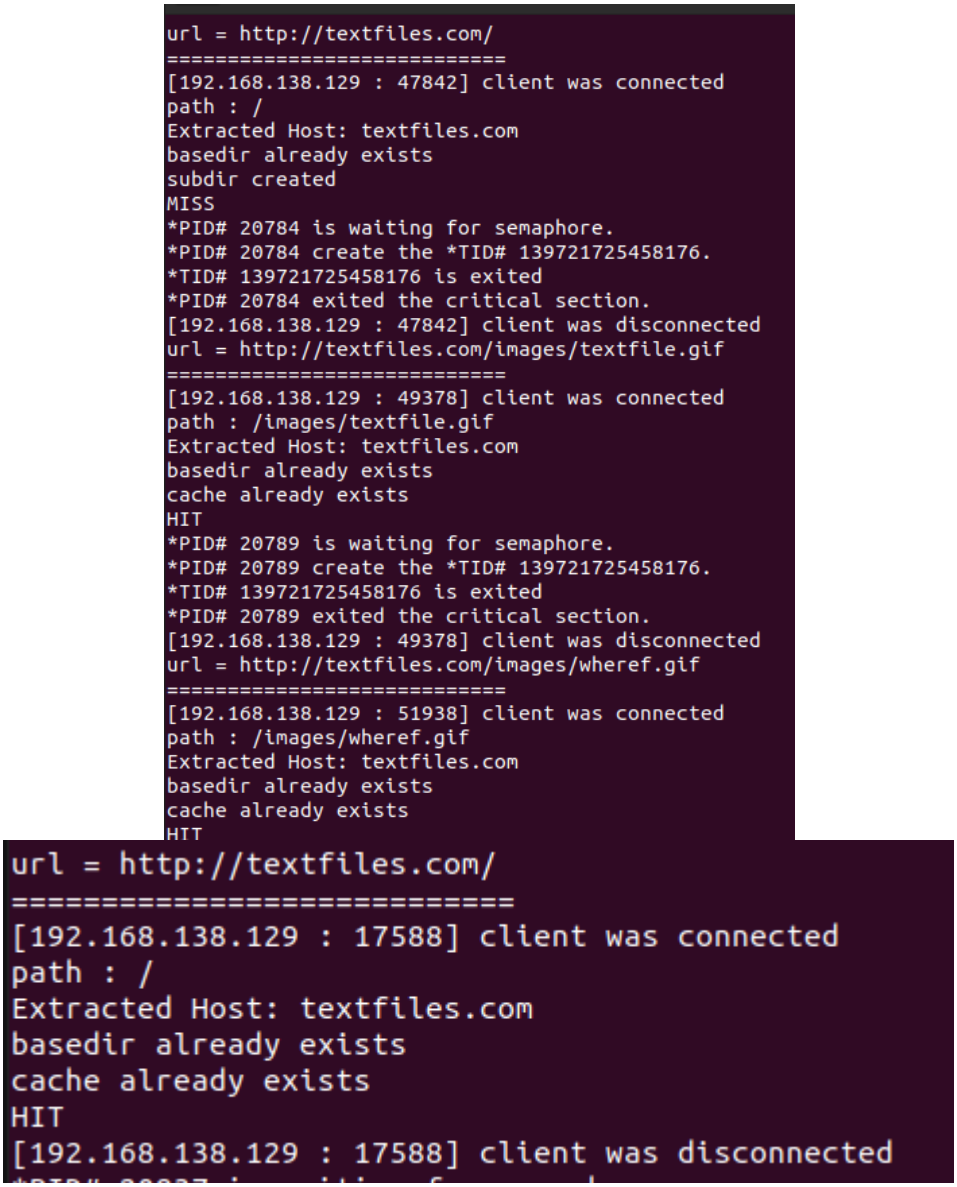
7) Semaphore 관련 함수(p, v)

```
// Semaphore p function
void p(int semid){
    Initialize Sem variables
    if(Lock section & check fail){
        exit(1);
    }
}
// Semaphore v function
void v(int semid){
    Initialize Sem variables
    if(Unlock section & check fail){
        exit(1);
    }
}
```

- Semaphore 변수들을 초기화한다.
- P 함수는 Critical section을 잠그고, V 함수는 잠금을 해제한다.
- 실행에 실패한 경우 종료한다.

3. 결과화면


kw2021202077@ubuntu:~/Proxy3-2$ gcc -pthread proxy_cache2.c -lcrypto
kw2021202077@ubuntu:~/Proxy3-2$ ./a.out
url = http://neverssl.com/
============================
[192.168.138.129 : 12518] client was connected
path : /
Extracted Host: neverssl.com
basedir created
subdir created
MISS
*PID# 20647 is waiting for semaphore.
*PID# 20647 create the *TID# 139721725458176.
*TID# 139721725458176 is exited
*PID# 20647 exited the critical section.
[192.168.138.129 : 12518] client was disconnected
url = http://silverlushbrightmorning.neverssl.com/
============================
[192.168.138.129 : 19681] client was connected
path : /
Extracted Host: silverlushbrightmorning.neverssl.com
basedir already exists
subdir created
MISS
url = http://silverlushbrightmorning.neverssl.com/
============================
[192.168.138.129 : 13998] client was connected
path : /
Extracted Host: silverlushbrightmorning.neverssl.com
basedir already exists
cache already exists
MISS
*PID# 20672 is waiting for semaphore.
*PID# 20672 create the *TID# 139721725458176.
*TID# 139721725458176 is exited
*PID# 20672 exited the critical section.
[192.168.138.129 : 13998] client was disconnected
url = http://silverlushbrightmorning.neverssl.com/online
============================
[192.168.138.129 : 43734] client was connected
path : /online
Extracted Host: silverlushbrightmorning.neverssl.com
basedir already exists
cache already exists
HIT
[192.168.138.129 : 43734] client was disconnected

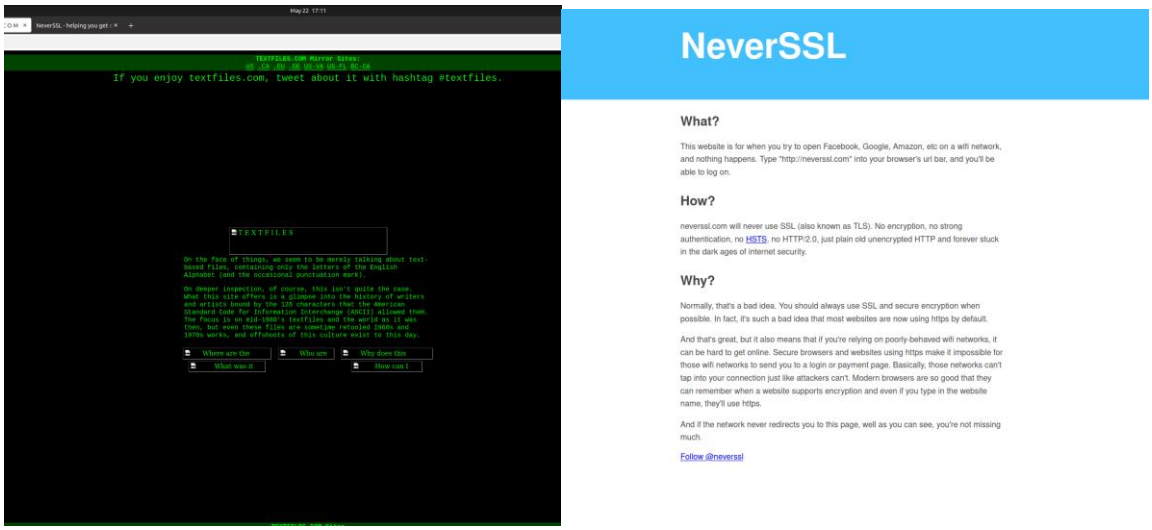


<Firefox Browser / console 창>

Proxy_cache.c 파일을 실행하면 listen 상태로 client request를 기다린다. 주소창에 url을 입력하면 콘솔창에 http 정보들이 출력된다. 디버깅 코드를 통해 url과 추출된 host가 제대로 처리되었는지 확인할 수 있다. http://neverssl.com , http://textfiles.com/ 를 2개의 콘솔창에 입력하여 hit/miss 여부를 확인하였다. 2개의 url이 스레드를 생성하며, 로그파일을 기록하는 것을 확인할 수 있다.

```
Favicon request ignored.
url = http://textfiles.com/favicon.ico
Favicon request ignored.
^C
[Terminated] Run time: 109 sec, #SubProcess: 49

[Terminated] Run time: 109 sec, #SubProcess: 61
kw2021202077@ubuntu:~/Proxy3-2$ tree ~/cache
/home/kw2021202077/cache
├── 0cc
│   └── 00fe1de958bc84c3bea15fb39582d90fab6ee
└── 830
    └── 41679ce90aecf4a849cb9612ada825db29392
```

<캐시파일 디렉토리 목록>



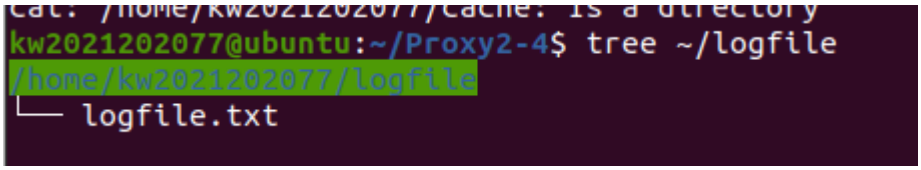
<textfile.com, neverssl.com 화면>

```
HTTP/1.1 200 OK
Date: Wed, 04 Jun 2025 13:05:16 GMT
Server: Apache/2.4.62 ()
Upgrade: h2,h2c
Connection: Upgrade, close
Last-Modified: Wed, 29 Jun 2022 00:23:33 GMT
ETag: "f79-5e28b29d38e93"
Accept-Ranges: bytes
Content-Length: 3961
Vary: Accept-Encoding
Content-Type: text/html; charset=UTF-8

<html>
        <head>
                <title>NeverSSL - Connecting ... </title>
                <style>
                body {
                        font-family: Montserrat, helvetica, a
                        font-size: 16x;
                        color: #444444;
                        margin: 0;
                }
                h2 {
                        font-weight: 700;
                        font-size: 1.6em;
                        margin-top: 30px;
                }
                p {
                        line-height: 1.6em;
                }
                .container {
                        max-width: 650px;
                        margin: 20px auto 20px auto;
```

```
HTTP/1.1 200 OK
Date: Wed, 04 Jun 2025 13:06:28 GMT
Server: Apache/2.4.58 (FreeBSD) OpenSSL/1.1.1o-freebsd
Last-Modified: Sun, 03 Nov 2024 02:54:33 GMT
ETag: "2958-625f949a948e2"
Accept-Ranges: bytes
Content-Length: 10584
Connection: close
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<HTML><TITLE>T E X T F I L E S D O T C O M</TITLE>

<!-- TEXTFILES.COM is a site dedicated to the Textfiles of the BBS World -->
<!-- from the 1980's. Naturally, this role expands into the continuation -->
<!-- of the textfile-writing world into the 1990's, and goes back to the -->
<!-- 1960's and 1970's as well. If you have any questions, feel free to  -->
<!-- mail them to jason@textfiles.com. Submissions are always wanted.    -->

<!-- Unlike a lot of sites, this Page Source has a lot of comments. This -->
<!-- is because this site's main purpose is education, and a user should -->
<!-- have the opportunity to see how this site was created and what the  -->
<!-- thought process was behind the look. Plus, the site weighs in at a  -->
<!-- very low size, so giving an explanation with the download is not a  -->
<!-- very bandwidth-intensive activity, and might inspire someone.       -->
```

<캐시파일 내부 데이터>

캐시파일을 확인한 스크린샷이다. 총 2가지의 URL이 입력되었고 2개의 파일이 생성되었다. 캐시파일 내부에는 웹서버에서 전송받은 http response가 작성되어 있다.



로그파일에는 terminate 정보와 함께, hit/miss 로그가 기록되어 있는 것을 확인할 수 있다.

4. 고찰

이번 Proxy server 3-1 과제는 2-4 과제에서 구현한 firefox proxy server와 소켓 통신에서 로그파일에 대한 semaphore 제어를 구현하는 것이었다. Semaphore 관련 구조체와 변수, 함수의 위치 설정이 주요 구현 포인트였다.

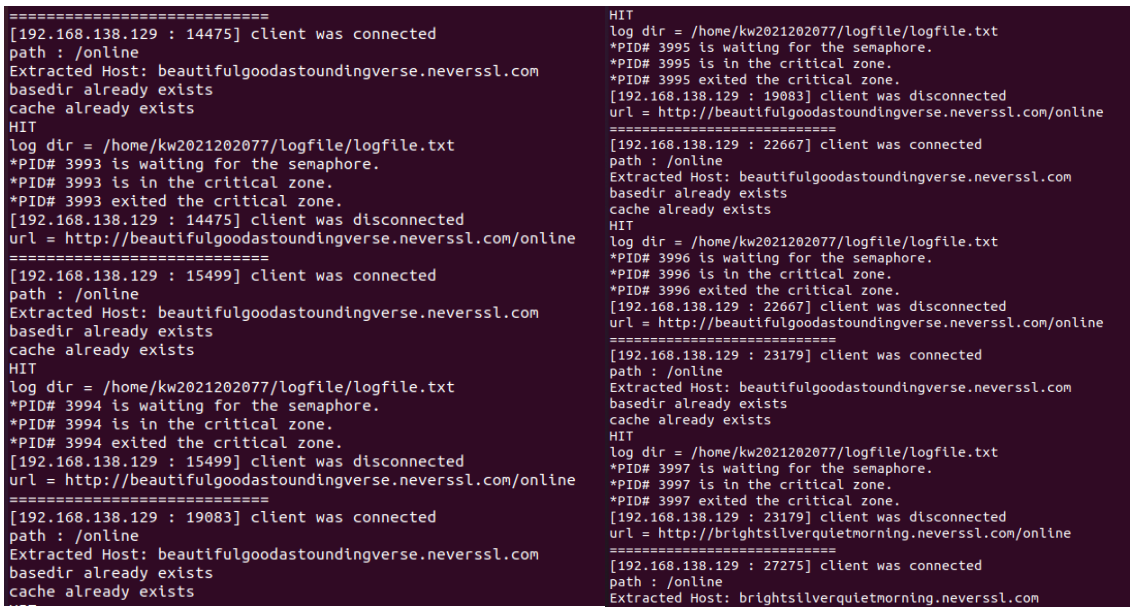
1) Logfile 작성 함수의 스레드화

Proxy3-2 과제는 로그파일을 스레드를 통해 작성하도록 코드를 구성하는 것이었다. 이를 해결하기 위해 LogFile()을 스레드 함수 형태로 다시 구성하고, 로그 기록에 필요한 정보들을 구조체(LogArgs)로 묶어 스레드 함수의 인자로 전달하는 구조를 적용하였다. 이를 통해 로그 기록은 메인 프로세스와 독립적으로 병렬 실행될 수 있게 되었고, 시스템의 응답성도 개선되었다.

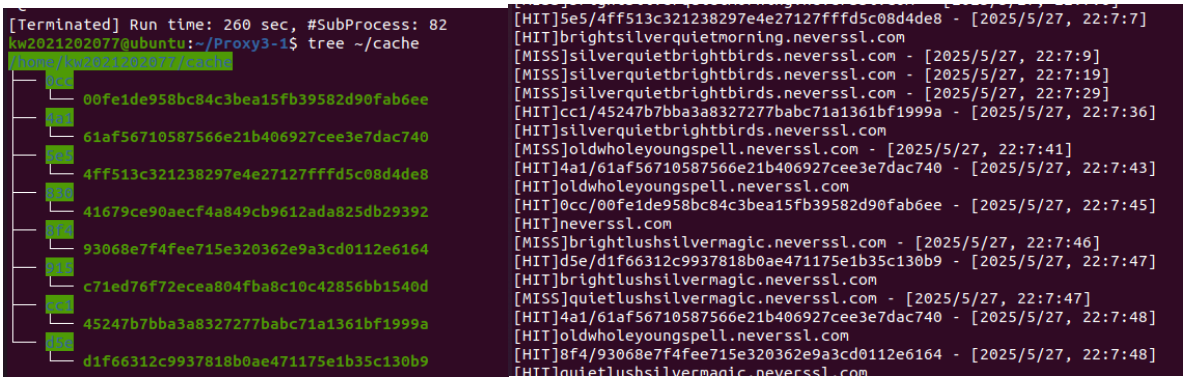
스레드화 과정에서 중요한 부분은 Critical Section 보호였다. 여러 스레드가 동시에 로그파일에 접근할 수 있기 때문에, 세마포어를 활용하여 LogFile() 내부를 Critical Section 으로 설정하였다. 이로 인해 세마포어가 한번에 한 개의 로그 파일 수정만 수행하여 문제 없이 안정적으로 기록되도록 만들 수 있었다.

이 과정을 통해 단순히 코드를 비동기적으로 분리하는 것이 끝이 아니라, 세마포어와 함수를 통한 스레드 간 데이터 전달 방식을 고려해야 한다는 점을 학습할 수 있었다. 또한, pthread_detach()를 통해 불필요한 리소스 누수를 방지하는 방법도 이해하게 되었다.

2) 리디렉션 링크 처리



url 테스트 중, http://neverssl.com 테스트를 진행하는데, 입력하지 않은 url이 콘솔창에 출력되는 것을 확인할 수 있었다. 이 링크들은 neverssl.com 을 포함하고 있었지만, 각기 다른 링크로 연결되고 있었다. 이는 url이 자동으로 리디렉션으로 연결하는 결과로 보인다. 문제는 이것들이 새로운 입력으로 판별되어 새로운 캐시파일을 만드는 것이었다.



위 그림처럼 여러 개의 캐시파일이 생성된 것을 확인할 수 있다. 하지만, 프록시 서버는 리디렉션까지 저장해두는 것이 구현 방향에 있어 바람직하다고 생각되어 입력을 제거하는 로직을 추가하지는 않았다.

3) Subprocess 개수 세기 - 부모, 자식 프로세스 간 통신의 필요성

과제 제안서에 따르면 Subprocess의 개수를 세는 작업에서는 miss로 판별되었을 때 세는 것처럼 보인다. 처음에는 miss로 판단된 요청마다 processcnt++을 수행하려 했으나, 해당 처리를 자식 프로세스 내부에서 수행할 경우, 자식 프로세스가 종료되면서

카운터 값이 초기화되어 반영되지 않는 문제가 발생했다. 이 문제를 해결하기 위해 부모-자식 프로세스 간 통신 방식인 pipe() 시스템 호출에 대해 학습하였다. pipe()를 통해 자식 프로세스에서 계산된 정보를 부모 프로세스로 전달할 수 있다.

4) Firefox 브라우저의 구동 방식

Firefox 브라우저를 이용하여 http url을 입력하면 콘솔창에 url 뒤에 /favicon.ico 및 firefox.com 가 붙어 GET 하는 것을 볼 수 있었다. 이 url의 host는 이전에 입력한 값과 같았기 때문에 캐시파일 생성 등의 심각한 오류는 발생하지 않았지만, 콘솔창이 보기 복잡한 형태로 출력되는 문제가 있었다. 하지만, 이는 잘못된 동작이 아니며 firefox 브라우저 구동 중 자동으로 웹사이트의 파비콘을 요구하는 단계가 포함되어 발생하는 것임을 알 수 있었다. 본 과제에서 과제에서는 이 단계를 무시하는 코드를 추가하여 subprocess 생성 이전에 client_fd를 종료하도록 설계하여 이 부분을 방지할 수 있었다.

5. Reference

- 방지민, “Proxy 3-2”, 광운대학교, 2025.
- 방지민, “Construction Proxy Connection”, 광운대학교, 2025.
- 최상호, “System Programming: Semaphore”, 광운대학교, 2025.
- 최상호, “System Programming: Thread”, 광운대학교, 2025.
- IBM, “Mozilla Firefox로 favicon.ico 파일 처리, 2024, https://www.ibm.com/docs/ko/sva/10.0.8?topic=configuration-handling-faviconico-file-mozilla-firefox
- Die.net, “pipe”, Linux man page, https://linux.die.net/man/2/pipe
- IBM, “ping 명령”, 2024, https://www.ibm.com/docs/ko/aix/7.2.0?topic=p-ping-command