System programming

2차 과제 - System Programming Assignment #1-2 (proxy server)

학　　과 : 컴퓨터 공학과

담당교수 : 황호영

분　　반 : 목34

학　　번 : 2016722092

성　　명 : 정동호

제출날짜 : 2018-04-06

목차

1 INTRODUCTION..... 3

2 FLOWCHART..... 3

3 PSEUDO CODE..... 6

4 결과화면..... 12

5 결론 및 고찰..... 14

6 참고 레퍼런스..... 14

1 Introduction

이전 1번째 구현에서 URL을 입력받고 그에 해당하는 해시로 더미 캐시 파일을 생성했었다. 이번 2번째 구현에서는 로그 기능을 추가하는 것을 목표로 한다.

이 때 구현할 기능은 시스템으로부터 현재 시간을 구하고 로그 파일을 생성하여 입력 URL과 현재 시간 기록등을 수행하는 것이다.

2 Flowchart

1-1 에서 사용된 기존 함수들 중 일부가 변경된 점도 있고 순서도를 이해하기 위해 1번째 보고서와 번갈아 가며 보는것이 불편할 것 같다 판단하여 이전에 첨부한 순서도도 일부 중복하였다.

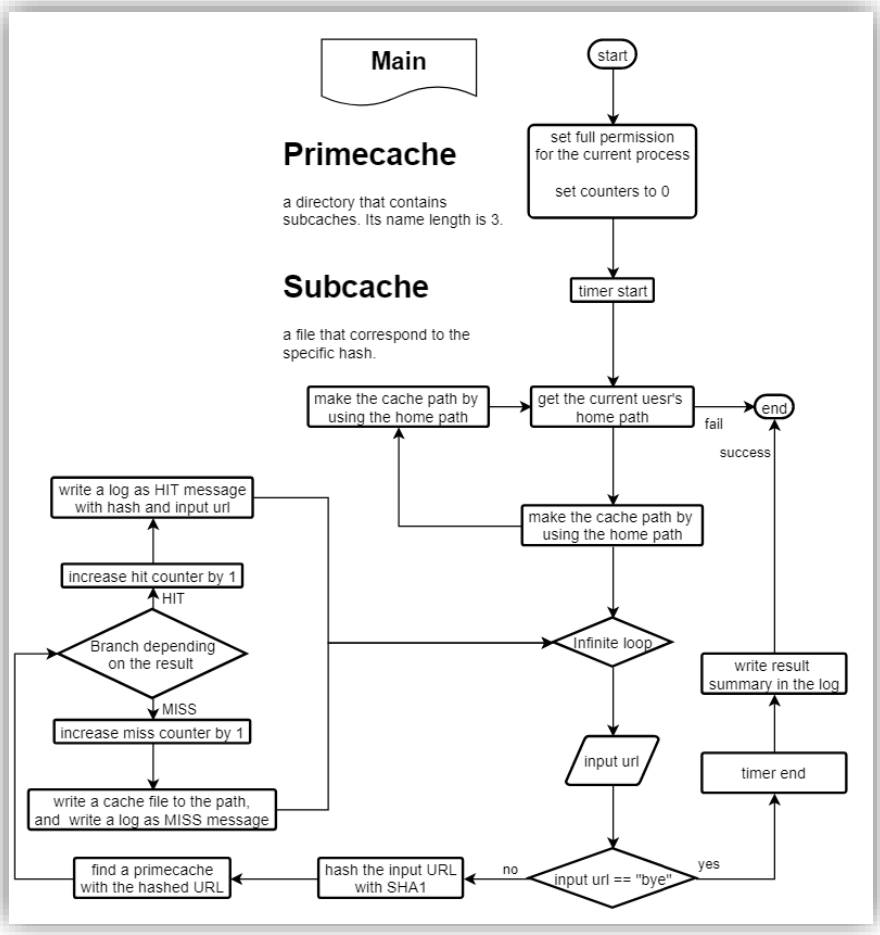

Figure 2.1 main 함수 순서도

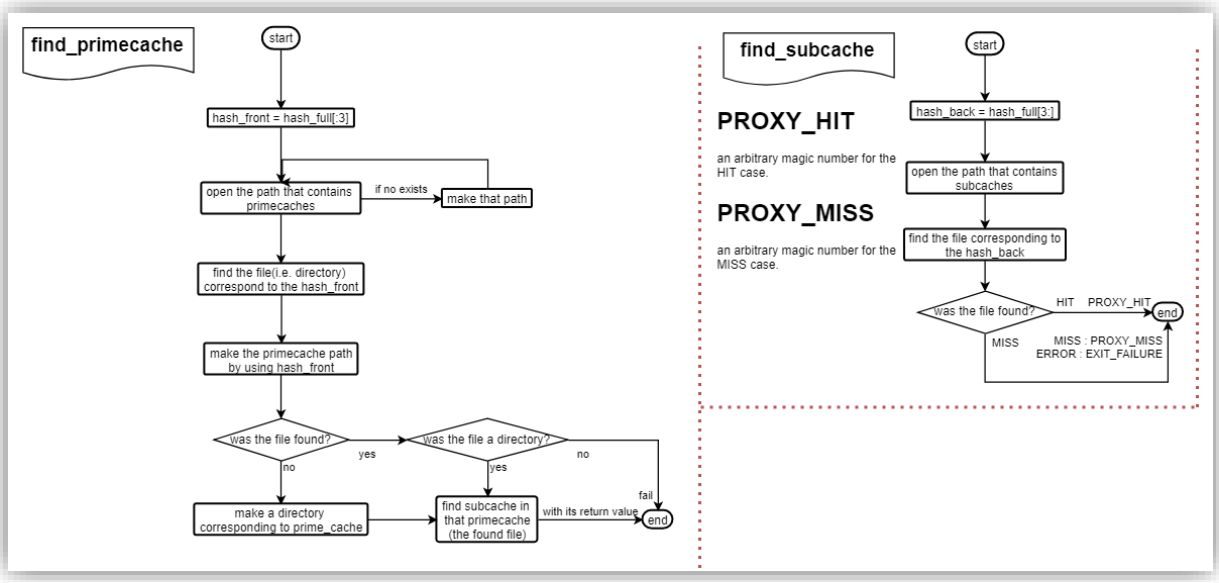

Figure 2.2 find_primecache & find_subcache 함수 순서도

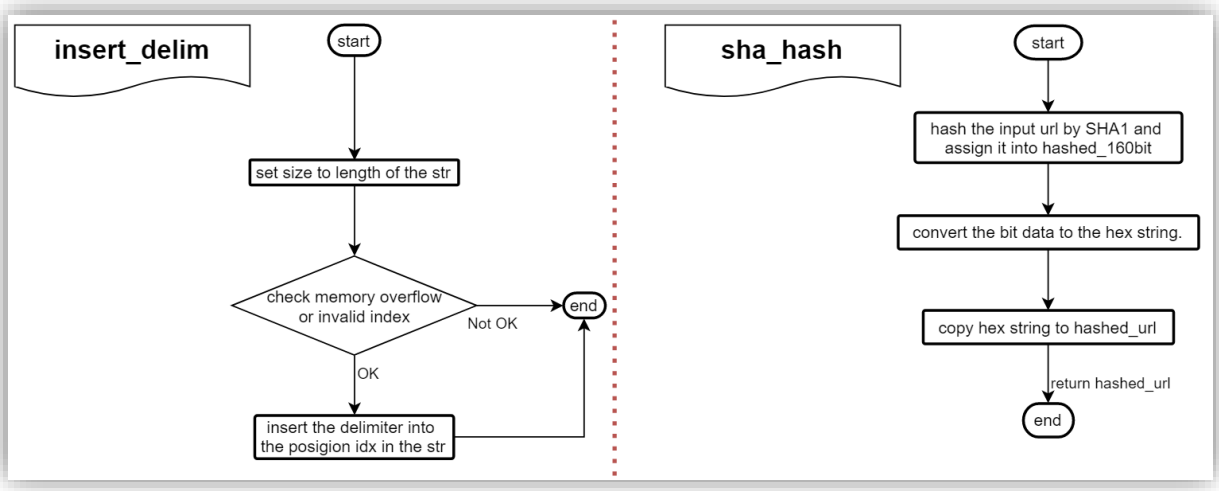

Figure 2.3 insert_delim & sha_hash 함수 순서도

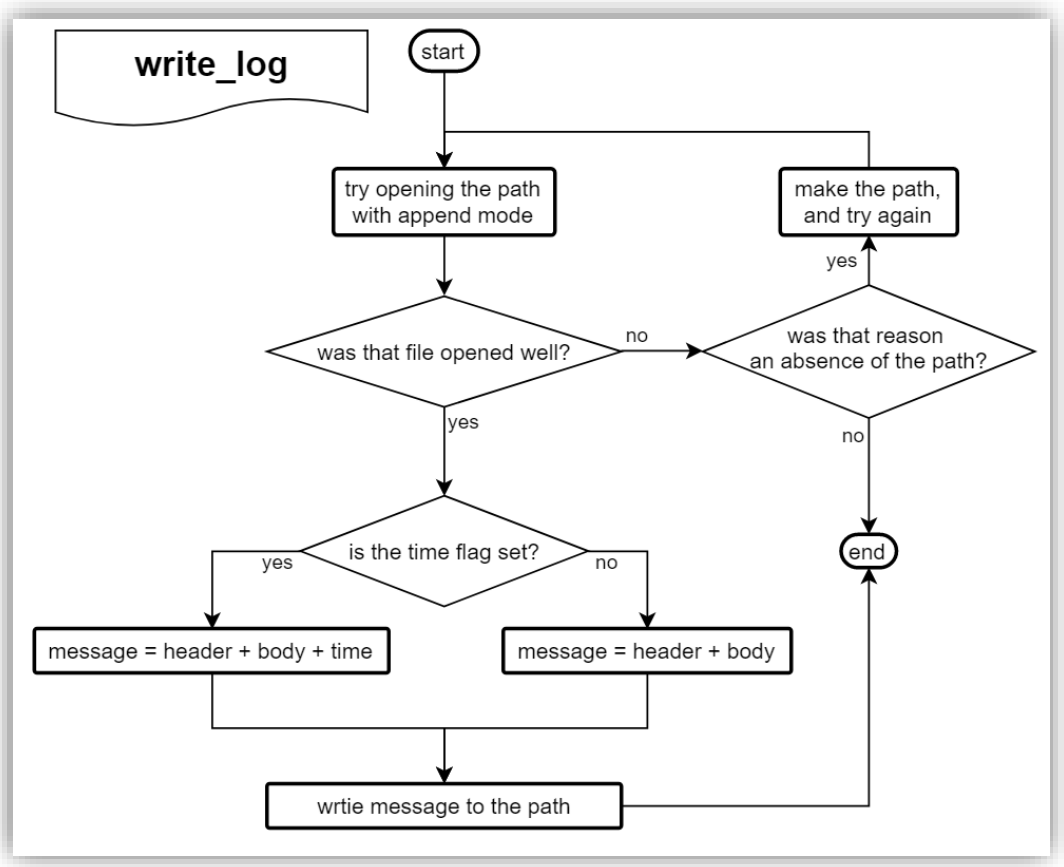

Figure 2.4 write_log 함수 순서도

기존의 1-1 에서 했던 내용은 생략하고 변경점만을 작성한다. 하지만 이는 어디까지나 부연설명일 뿐이므로 순서도를 통해 이해가 가능할 것이라 판단했다.

Figure 2.1 부터 보면 1 번째 구현에서와 다르게 main 함수의 역할이 2 개 추가되었다. 먼저 캐시 파일을 생성하는 역할을 main 함수에서 find_primecache 의 결과에 따라 MISS 인 경우에 수행한다. 또한 로깅 기능도 main 함수에 추가되었다. 마지막으로 경과시간 로깅을 위해 타이머 설정에 대한 블럭이 추가된것을 볼 수 있다.

Figure 2.2 의 find_primecache 와 find_subcache 에서 크게 달라진점은 없다. 대신 두 함수 모두 hash 전체 문자열을 받게 하였고 primecache(해시 앞 3 글자)나 subcache(해시 3 번째 이후 글자들)를 판별하는 역할은 호출되는 대상이 수행하도록 변경하였다. 그리고 HIT 와 MISS 의 경우 모두 그에 해당하는 상수값을 반환하도록 하여 파일 생성 역할은 호출자에게 전가했다. 이전 1-1 에서는 HIT 와 MISS 를 각각 EXIT_SUCCESS 와 EXIT_FAILURE 로 나타내고 예외 처리로 에러 코드를 반환하는 것은 음수로 했는데 직관적이지 못한것 같아 따로 상수를 만들어 처리하였다.

Figure 2.3 에선 새로 추가한 insert_delim 함수만 소개한다. 사실 일단 만들긴 만들었지만 생각보다 많이 활용되지 않았다. 이 함수가 맡은 역할은 임의의 문자열에 구분자를 추가하는 것이다. 본 과제에선 main 함수가 cache 파일 생성시 사용하는 경로를 만들기 위해 사용되었다.

Figure 2.4 의 write_log 는 이번 과제에서 새롭게 추가된 로깅 기능을 수행하기 위해 만든 함수이다. 파일이 잘 열렸는지 확인하고 아니라면 적절한 예외처리를 거친뒤 그럼에도 불구하고 해결되지 않으면 EXIT_FAILRUE 를 반환하도록 했다. 잘 열렸다면 타임 플래그를 체크한다. 해당 플래그가 참이라면 메세지에 시간정보를 추가하고 아니라면 하지 헤더와 바디부분만 메세지에 추가한다. 이렇게 만들어진 메세지는 앞서 연 파일에 추가로 쓰여진다.

동적으로 문자열을 생성하는 것이나 버퍼의 크기를 그렇게 정한 이유 그리고 왜 무엇을 어떻게 사용하였는지와 같은 이유는 순서도에서는 알수가 없으니 다음 절의 Pseudo code 부분에서 자세히 설명한다.

3 Pseudo code

본 report에서 사용된 Psuedo code style은 다음과 같다.

1. 코드를 그대로 갖고온다.
2. 문장으로 치환할 수 있는 부분은 치환하고 굵은 노란색으로 표시한다.
3. 2번을 거친 뒤 나머지 1번에서 가져온 부분 중 언급되지 않은 것들은 제거한다.
4. 코드 블럭 밑에서 덧붙일 부가 설명과 중복된 부분도 제거한다.

그리고 2.의 경우 슬라이싱이나 집합등 기타 부분에서 python style을 사용할 것이다.

그럼 순서도에 나온 순서대로 코드를 설명한다.

※ 이전 1-1 에서 설명한 Proxy_constants 열거형 상수에 PROXY_HIT 와 PROXY_HIT 가 추가되었다. 하지만 이에 대한 설명은 2 장의 Figure 2.2 를 설명하는 과정에서 나온 것으로 충분하다 생각하여 생략한다.

Figure 3.1 main 함수 pseudo code

```
int main(int argc, char* argv[]){

    // char arrays for handling a url
    char url_input[PROXY_MAX_URL] = {0};
    char url_hash[PROXY_LEN_HASH] = {0};

    // counter for HIT and MISS cases
    size_t count_hit  = 0;
    size_t count_miss = 0;

    Timer start

    Set full permission for the current process.

    Try getting current user's home path
    and concatenate cache and log paths with it

    Receive inputs till the input is 'bye'{
        printf("input url> ");
        scanf("%s", url_input);

        If input is 'bye' then break loop

        Hash the input URL and find the cache with it

        Insert a slash delimiter at the 3rd index in the url_hash

        Make a path for fullcache

        switch(result){
            case PROXY_HIT:
                count_hit += 1;
                Write a log as url_hash and url_input in log path
                break;

            case PROXY_MISS:
                count_miss += 1;
                Create a dummy cache in the cache path
                Write a log as url_hash and url_input in log path
                break;

            default:
                break;
        }
    }

    Timer end
```

```
    Make a string for terminating the log and write it

    return EXIT_SUCCESS;
}
```

앞서 2 장의 순서도에서 보았듯이 Timer 에 대한 설정과 캐시 파일을 탐색한 결과에 따라 로그 및 더미파일을 생성하는 역할이 추가되었다. 이전 1-1 에는 MISS 인 경우 더미 캐시 파일 생성을 find cache 최종 단계인 find_subcache 에서 맡았지만 find_primecache 와 find_subcache 의 인자(hash_full)와 역할(hash_full 로부터 앞과 뒷부분을 분할)를 통일하는 과정에서 HIT 냐 MISS 냐에 따른 처리는 최초 호출자 main 에서 하도록 변경했다.

따라서 최종 캐시 파일 경로를 구하기 위해 뒤에 나올 insert_delim 으로 url_hash 를 경로로 변환한뒤 MISS 인 경우 로그에 MISS 메세지를 기록하면서 캐시 파일도 생성한다. HIT 인 경우도 이와 유사하게 로그에 HIT 메세지를 기록한다.

또 MISS 냐 HIT 냐에 따라 그에 해당하는 counter 를 증가시키고 bye 를 입력 받아 종료될때 처음에 설정한 Timer 와 현재 시간의 차이를 구해 로그에 counter 와 경과 시간을 기록한다.

Figure 3.2 find_primecache 함수 pseudo code

```
 * @param path_primecache A const char pointer to the path containing primecaches.
 * @param hash_full A const char pointer to be used as a part of a cache.
 * @return [int] HIT:PROXY_HIT, MISS:PROXY_MISS, FAIL:EXIT_FAILURE
int find_primecache(const char *path_primecache, const char *hash_full){
    char hash_front[PROXY_LEN_PREFIX + 1] = {0};

    int result = 0;

    Extract the front part of the hash
    and assign the result into hash_front

    Check whether the path of primecache exist or not
    If not exist{
        create that path, and try opening it again
    }

    Make the full path of the directory which contains subcaches

    Find the primecache that matches with hash_front
    while traversing the path{
        If the primecache was found{
            Check whether it is a directory or not
            If the file was regular, then something's wrong in the cache directory   {
                Throw an error
            }
```

```
        }
    }

    If there isn't the path of subcache,
    then create that path with full permission

    Find the subcache in the path of the current primecache
    and assign the return value into the result

    return result;
}
```

find_primecache 가 바뀐 부분은 딱 하나, subcache(hash 의 뒷부분)을 구하는 역할을 빼고 find_subcache 를 호출할 때 subcache 를 전달하지 않고 hash_full 을 전달하도록 변경했다.

먼저 인자로 받은 전체에서 primecache(해시 앞부분)을 추출한다. 그리고 인자로 받은 경로를 순회하며 해당하는 primecache 를 탐색한다. 만약 경로를 순회하려는데 해당 경로가 없으면 해당 경로를 생성하고 다시 시도한다.

이 때 찾은 파일이 디렉터리가 아니라 파일이면 cache 디렉터리 구조에 문제가 생긴것이므로 이를 알리고 EXIT_FAILRUE 를 반환한다.

파일을 못 찾았았으면 해당 primecache 로 시작하는 캐시가 아직 생성되지 않은 것이므로 이를 생성하고 find_subcache 를 호출한다.

그리고 이에 대한 HIT 냐 MISS 냐에 대한 결과를 반환한다.

Figure 3.3 find_subcache 함수 pseudo code

```
 * @param path_subcache A const char pointer to the path containing subcaches.
 * @param hash_full A const char pointer to be used as a part of a cache.
 * @return [int] HIT:PROXY_HIT, MISS:PROXY_MISS
int find_subcache(const char *path_subcache, const char *hash_full){
    struct dirent *pFile = NULL;
    DIR           *pDir  = NULL;

    char hash_back[PROXY_LEN_HASH - PROXY_LEN_PREFIX + 1] = {0};

    Extract the back part of the hash
    and assign the result into hash_back

    Find the subcache while traversing the path

    If the file is found {
        return PROXY_MISS;
```

```
    } else {
        return PROXY_HIT;
    }
}
```

find_subcache 에는 계산된 subcache(해시의 뒷부분)값이 들어오는 대신 hash_full 이 들어오면서 함수 내에 subcache 를 계산하는 부분이 추가되었다.

그리고 primecache 와 했던것과 유사하게 인자로 전달받은 경로를 순회하며 subcache 들을 탐색하고 캐시를 찾았냐 못찾았냐에 따라 PROXY_HIT 또는 PROXY_MISS 를 반환한다.

Figure 3.4 insert_delim 함수 pseudo code

```
 * @param str A char array that a delimiter'll be inserted into.
 * @param size_max The size of str buffer.
 * @param idx The location where the delimiter to be inserted into.
 * @param delim A delimiter character.
 * @return [int] Success:EXIT_SUCCESS, Fail:EXIT_FAILURE
int insert_delim(char *str, size_t size_max, size_t idx, char delim){

    Check buffer overflow or invalid index

    Insert the delimiter into the position idx in the str.

    return EXIT_SUCCESS;
}
```

insert_delim 은 하는 역할만큼이나 단순한 함수로 전달받은 str 의 idx 위치에 문자 delim 을 추가한다. 딱히 실패할 일은 없지만 이미 str 이 할당받은 크기만큼 꽉 차있거나 idx 가 str 의 길이 보다 큰 경우엔 EXIT_FAILURE 를 반환한다.

Figure 3.5 write_log 함수 pseudo code

```
 * @param path A const char pointer pointing the path to write the log.
 * @param header The header of a log message.
 * @param body The body of a log message.
 * @param time_ If it is true, write the log with current time. otherwise, don't.
 * @return [int] Success:EXIT_SUCCESS, Fail:EXIT_FAILURE
int write_log(const char *path, const char *header, const char *body, bool time_){
    // 32 is a moderately large value to save time information
    char time_str[32] = {0};

    Try opening the path with the append mode
    If the try goes fail {
        If its reason is absence of the log path {
            Make the path, and try again
```

```
        } else {
            Throw an error
        }
    }

    Assign new memory block to msg_total by enough space

    If time flag is true,
    then time_str has format as "-[%Y/%m/%d, %T]"
    ※ time information is set to current local time

    Join all parts of the message together

    Write message to the path

    return EXIT_SUCCESS;
}
```

write_log 는 insert_delim 과 함께 이번 2 번째 구현에서 추가된 함수다. 인자는 path, header, body, time_ 으로 이루어져 있고 이름과 자료형에서 알 수 있다시피 path 는 로그를 작성할 위치, header 와 body 는 메세지의 앞과 뒷부분, time_은 메세지 뒤에 시간 정보를 추가할지 말지 결정하는 플래그이다. 굳이 header 와 body 로 나눈 이유는 호출측에서 굳이 추가적인 문자열 연산 없이도 Literal string 과 동적으로 변하는 문자열을 그대로 인자로 넘길 수 있게끔 하기 위해서였다.

보면 time_str 이 32 크기인데 - dash 나 [] square brackets 또는 공백 컴마 같은 것들과 연월일 시분초 정보를 수용할 수 있는 적당한 크기라 생각해서 결정했다.

로그 파일을 열때 해당 경로의 부모 디렉터리가 없으면 생성하고 다시 시도해본다. 근데 디렉터리의 부재같은 문제가 아니면 해당하는 에러를 알리고 EXIT_FAILURE 를 반환한다. 그리고 time_ 플래그에 따라 header + body 에 시간 정보를 추가할지 말지 결정하고 그 결과를 log 파일에 작성한다.

4 결과화면

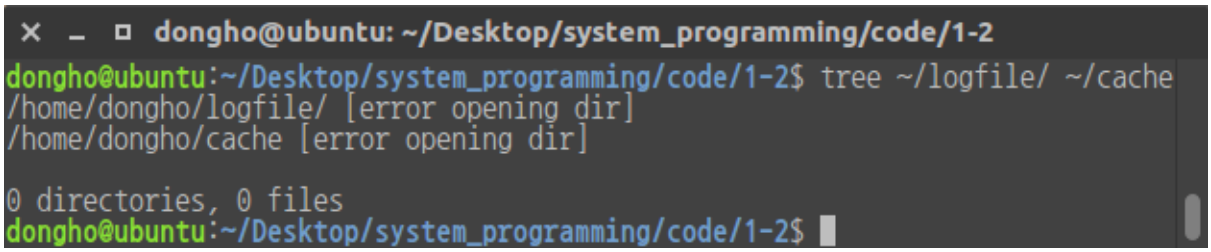

Figure 4.1 log & cache 디렉터리 초기 구조

처음에 두 디렉터리 ~/logfile 과 ~/cache를 tree로 조회해 본 결과이다. 아직 아무것도 수행하지 않았으므로 아무것도 없다.

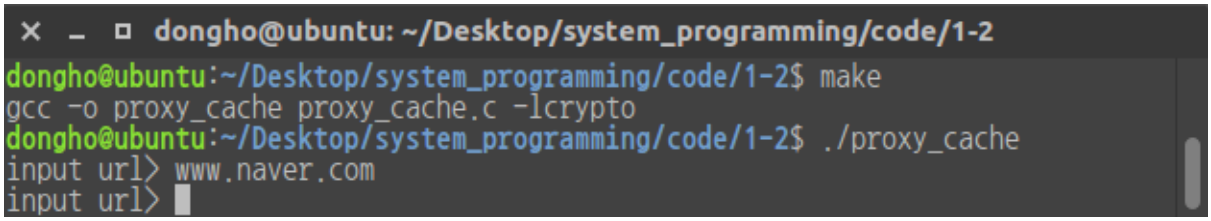

Figure 4.2 컴파일 & URL 입력

make로 컴파일을 하고 실행시킨 뒤 URL www.naver.com을 입력했다.

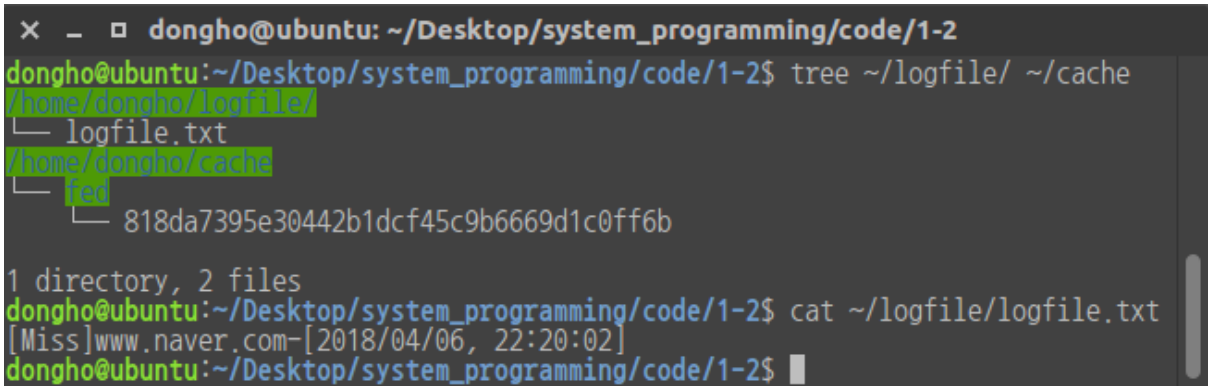

Figure 4.3 logfile 내용

logfile과 cache 경로에 모두 파일이 잘 생성되었고 logfile.txt에 내용도 잘 기록되었음을 확인하였다.

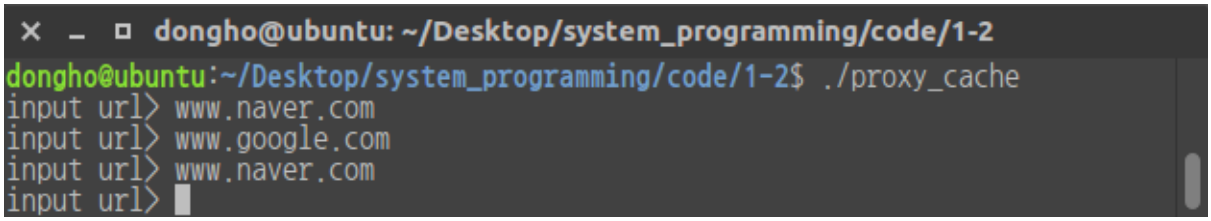

Figure 4.5 추가 URL 입력

이번엔 MISS인 경우와 HIT인 경우를 모두 입력했다. 결과는 다음과 같다.

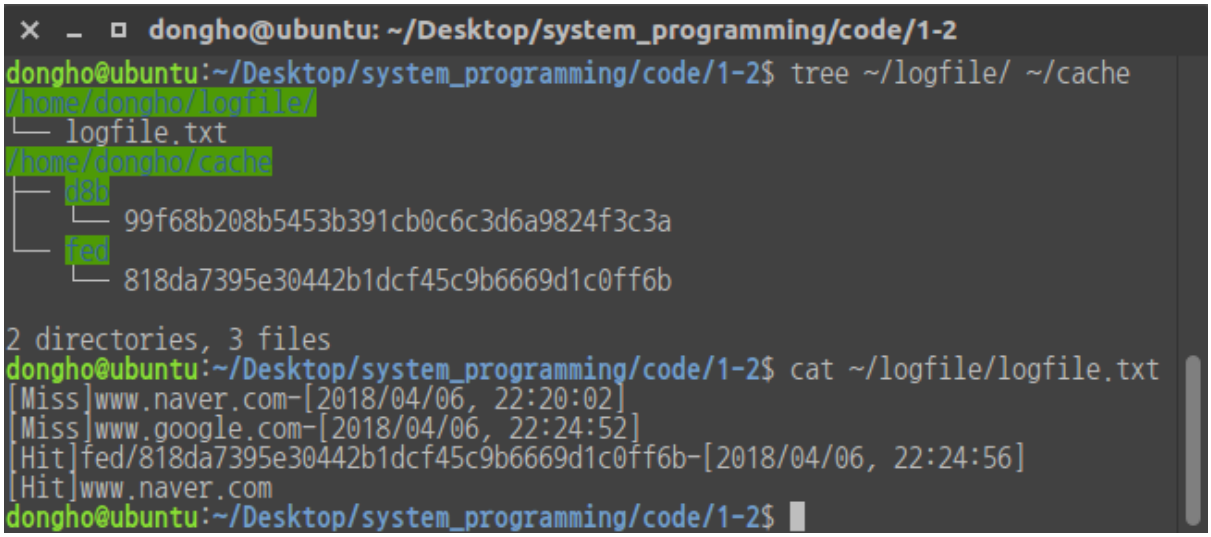

Figure 4.4 추가 URL 입력후

입력 받은 URL의 종류는 2개므로 캐시도 2개가 생겼고 로그 파일에도 MISS와 HIT에 대한 경우 모두 기록이 잘 이루어 졌음을 확인하였다.

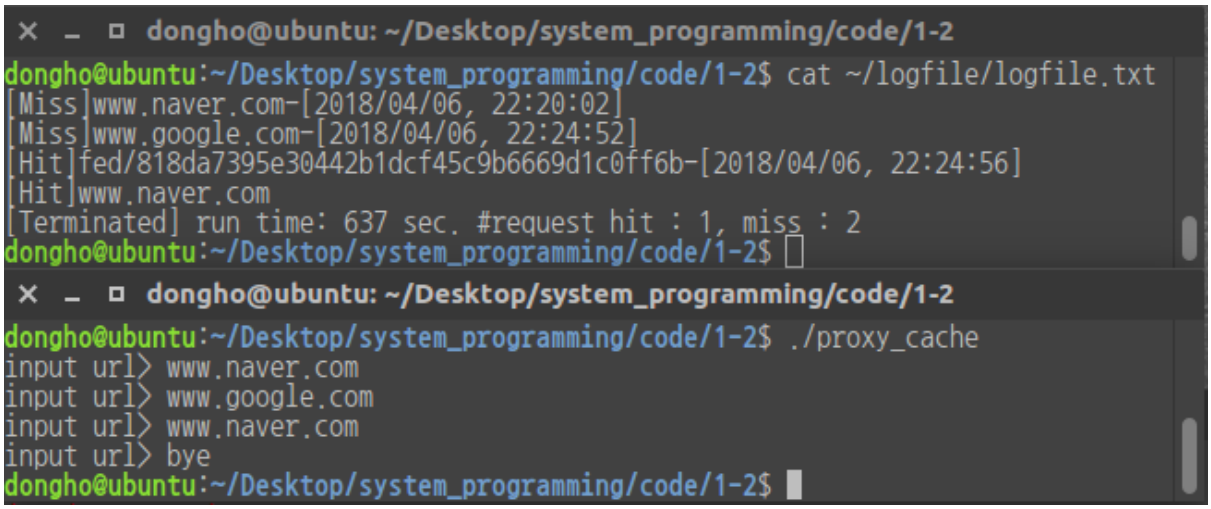

Figure 4.6 종료

bye를 입력받아 종료시에 경과 시간과 요청 통계에 대한 값도 잘 기록되었음을 확인하였다.

5 결론 및 고찰

이번 과제에는 1번째 구현처럼 아무것도 없는 상태에서 시작한게 아니라서 그런지 조금 수월했다. 이번에 새롭게 요구된 기능은 크게 로깅 기능인데 이를 통해 파일을 open하는 모드들과 파일이 존재하지 않을때 반환되는 에러같은 파일에 대한 c 프로그래밍 지식들을 익힐 수 있었다. 뿐만 아니라 시간정보를 기록하는 과정에서 시간 구조체와 이것들을 문자열로 반환하고 차를 계산하고 하는 시간에 대한 c 프로그래밍 지식들 또한 익힐 수 있었던 유익한 과정이었다.

또한 이전 1번째 구현에서 맘에 들지 않았던 비직관적인 반환 코드들을 바꾸는 과정에서 많은 고민을 할 수 있었다.

다음이 마지막 구현인데 멀티프로세싱을 구현한다고 한다. 하나도 모르는건데 기대된다.

6 참고 레퍼런스

https://stackoverflow.com/a/9285679 문자열에 구분자를 삽입할때 memmove를 사용하는 법을 참고했다.

https://linux.die.net/man/3/dirname 전달받은 경로에 파일을 생성하려는데 해당 경로가 없을 경우 해당 경로를 생성하기 위해 전달받은 경로로부터 디렉터리 부분을 뽑기 위해 dirname을 사용하는 과정에서 참고했다.

강의자료실의 4.+time+and+date.pdf 와 18-1_SPLab_week04_Proxy+1-2.pdf 에서 시간관련한 함수들을 많이 참고할 수 있었다.