System programming

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

학　　과 : 컴퓨터 공학과

담당교수 : 황호영

분　　반 : 목34

학　　번 : 2016722092

성　　명 : 정동호

목차

1 Introduction

본 과제에서 다룰 프로그램은 캐싱 기능을 수행하는 프록시 서버이다. 이번 1번째 구현 단계에선,

- 표준입력으로부터 URL 입력 받기
- SHA1 알고리즘으로 text URL을 hashed URL로 변환하기
- hashed URL을 이용하여 directory와 file 생성하기

이 3개를 구현한다.

2 Flowchart

본 과제에서 사용된 주요 함수 4개의 순서도는 다음과 같다:

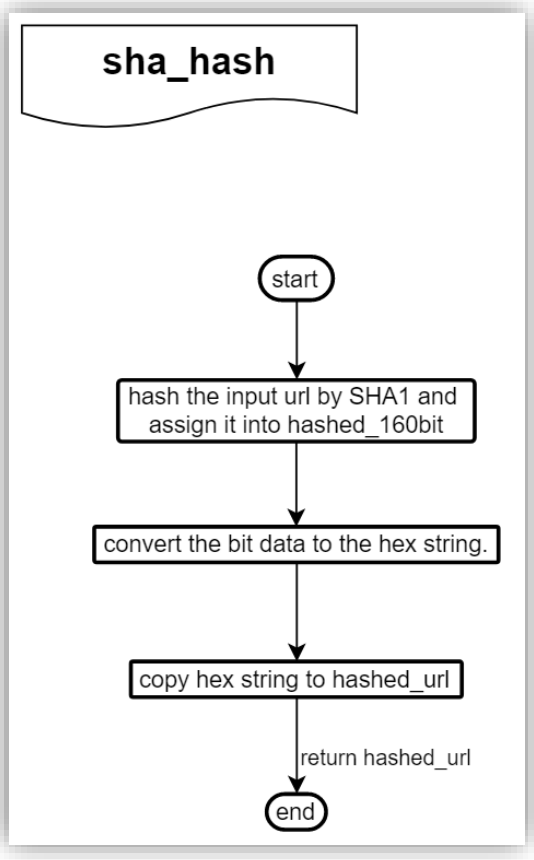

Figure 2.2 sha_hash 함수 순서도

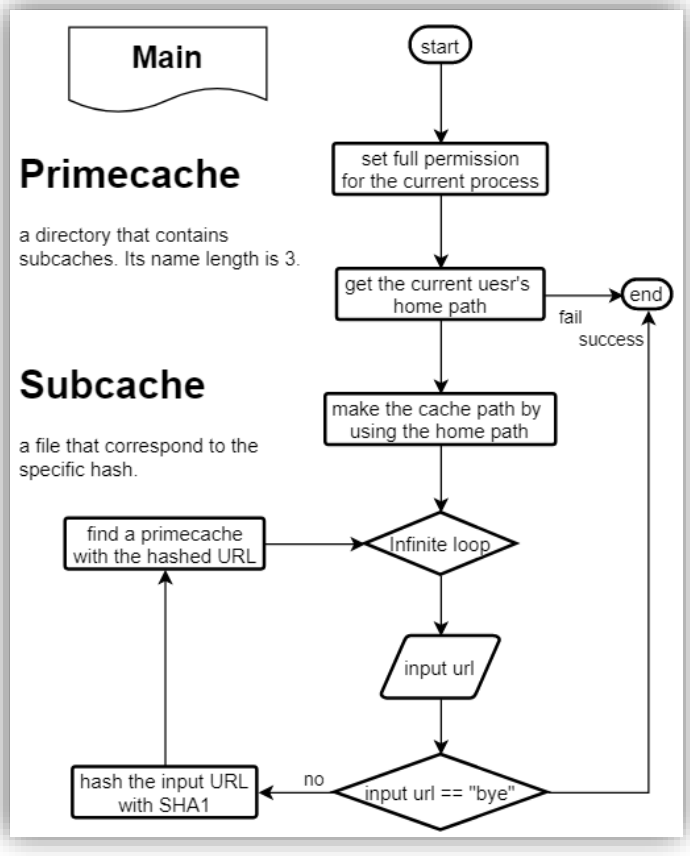

Figure 2.1 main 함수 순서도

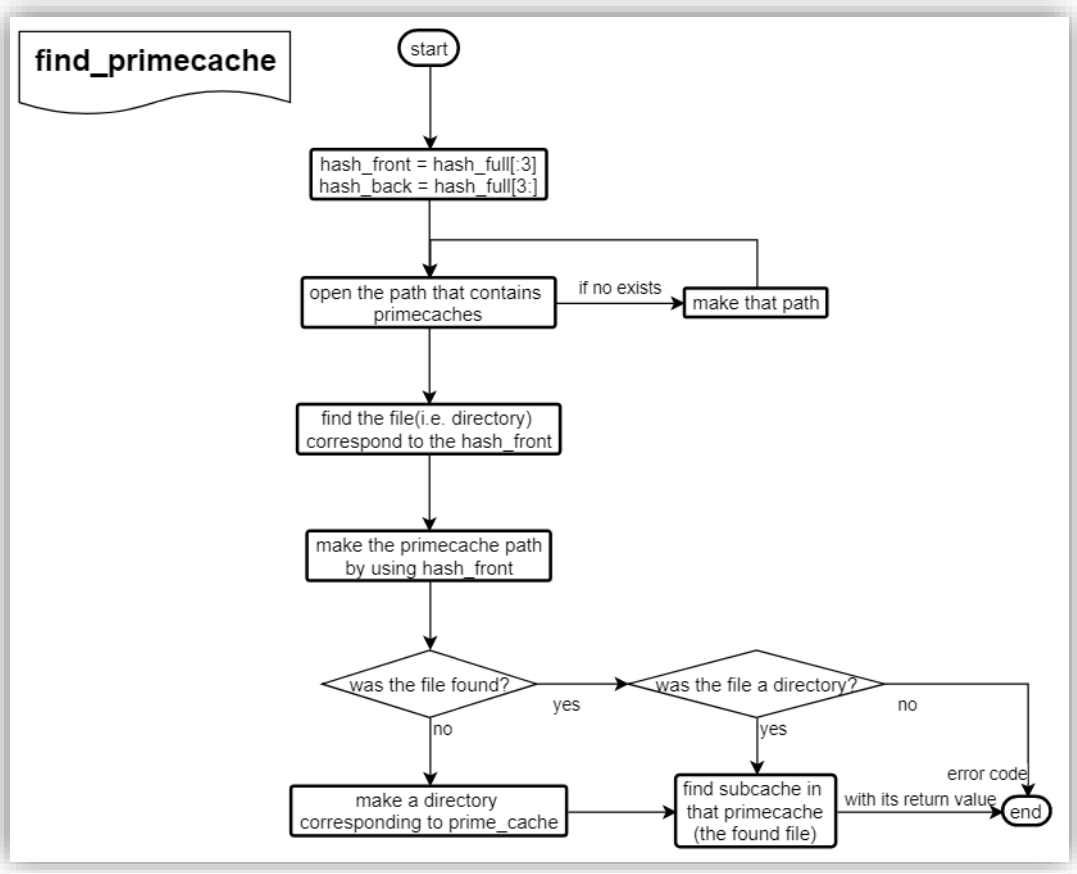

Figure 2.3 find_primecache 함수 순서도

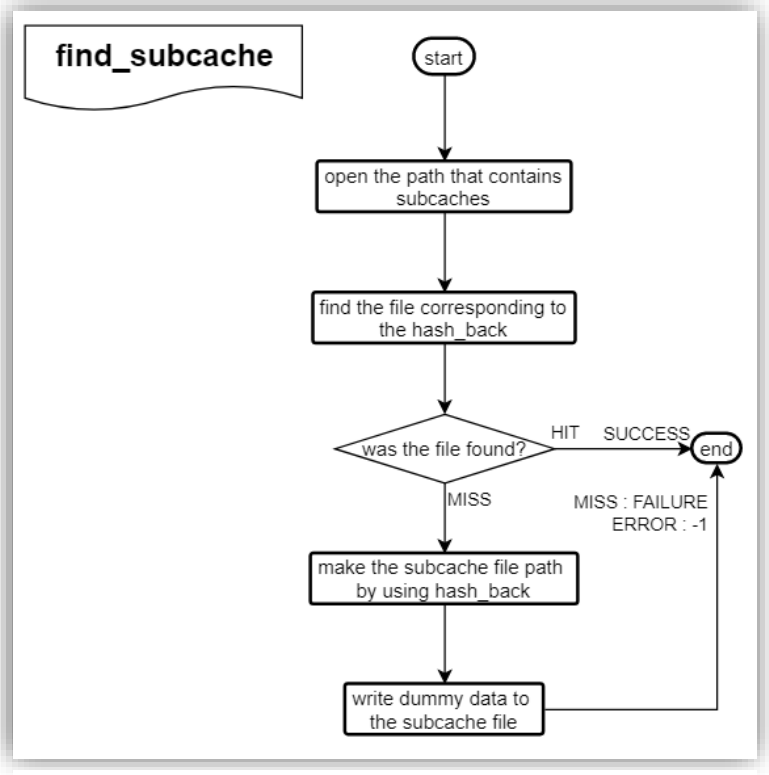

Figure 2.4 find_subcache 함수 순서도

먼저 main 함수 순서도에서 Primecache와 Subcache에 대해 정의했다. 실제로 있는 말은 아니지만 본 과제에서 SHA1 해시값이 두 부분으로 나뉘어 편의상 이렇게 구분하기로 하였다. 또 분기문이 아닌 블록에서 가끔 화살표가 두개로 나뉘는 데 이 부분은 예외 처리를 나타낸것이다. 따라서 예외가 없으면 일반 화살표 방향대로 가고 예외가 생길 경우에만 해당 블럭으로 넘어간다. 그리고 굳이 Infinite loop와 prime_cache의 반복을 구분한 이유는 실제 코드 상에서 유의미한 차이가 있기 때문이었다.

자세한 설명은 다음 절의 Pseudo code 부분과 상당 부분 겹치므로 일단 생략한다.

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.

Figure 3.1 상수값들

```
 * @see MAX_URL   : See https://stackoverflow.com/a/417184/7899226
 * @see MAN_PATH  : Not official but refer to the linux/limits.h and wikipedia.
typedef enum {
PROXY_LEN_PREFIX = 3,    ///< A length of the front hash.
PROXY_LEN_HASH = 41,     ///< A length of the full hash(in the SHA1).
PROXY_MAX_URL = 2048,    ///< A maximum length of an url.
PROXY_MAX_PATH = 4096    ///< A maximum length of a path.
} Proxy_constants;
```

전역 스코프에서 정의된 enum 구조체이다. 뜬금없기도 하고 가급적 쓰고 싶지 않았지만 C에서 스코프 충돌 없이, 디버깅 심볼을 이용가능하며, VLA 초기화 문제를 고려해 enum으로 위와 같은 상수들을 정의하였다. 각 값들은 과제에서 주어졌거나 따로 찾아 결정하였다.

Figure 3.2 sha1_hash pseudo code

```
 * @param input_url A char pointer pointing an input URL to be hashed.
 * @param hashed_url A char pointer that the hashed URL'll be assigned into.
 * @return The hashed URL is also returned as a return value.
char *sha1_hash(char *input_url, char *hashed_url){
    unsigned char hashed_160bits[20];   // since the SHA1 has a 160 bits size and a
character is 8 bits
    char hashed_hex[41];                // since a hashed string is consists of
hexadecimal characters

    Hash the input_url by SHA1 and assign it into hashed_160bit
    Convert the bit data of input_url to the hex string
    Copy hex string to hashed_url

    return hashed_url;
}
```

input_url에 입력받은 URL이 담겨진 포인터를 전달하고 SHA1으로 해시된 결과가 담겨질 포인터를 hashed_url에 전달하면 가변 길이 input_url이 고정 길이 40자의 hashed_url에 담겨지게 된다.

Figure 3.3 main pseudo code

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

    Set full permission for the current process
    Try getting current user's home path and concatenate a cache path
    Receive an input till the input is 'bye' {
        Input url
        If input is 'bye' then break loop
        Hash the input URL and find the cache with it
    }

    return EXIT_SUCCESS;
}
```

먼저 과제에서 요구하는대로 파일 생성시 full permission을 주기 위해 umask를 사용했다. 처음엔 umask 함수에 대해 잘 몰라 시스템 전역에 영향을 미치면 어떡하지 하고 걱정했지만 다행히 현재 프로세스에만 적용되는 함수였다.

그리고 과제에서 주어진 getHomeDir 함수를 사용하여 cache 경로를 구한다.

그리고 유저로부터 'bye'를 입력받을때 까지 URL을 입력받고 해싱하여 그 값으로 캐시를 탐색한다.

Figure 3.4 find_primecache 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 primecache name.
 * @return [int] HIT:0, MISS:1, FAIL:-1
int find_primecache(const char *path_primecache, const char *hash_full){
    char hash_front[PROXY_LEN_PREFIX + 1]                = {0};
    char hash_back[PROXY_LEN_HASH - PROXY_LEN_PREFIX + 1] = {0};

    int result = 0;

    hash_front = hash_full[PROXY_LEN_PREFIX:]
    hash_back  = hash_full[:PROXY_LEN_PREFIX]

    If the path of primecache dosen't exist, create that path

    Find the primecache while traversing the path {
        If the primecache was found {
            If it is a directory {
                the primecache was found
            } else {
                exception handling
            }
        }
    }

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

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

    return result;
}
```

find_primecache는 인자로 주어진 path_primecache에서 hash_full의 앞 세글자 짜리 폴더(primecache)를 찾는 역할을 수행한다. 따라서 hash_front는 primecache의 이름으로 쓰이고 hash_back은 primecache아래 subcache의 이름으로 쓰이게 된다.

만약 primecache가 존재할 경로가 없다면(여기선 home path 밑의 cache directory) full permission으로 해당 경로를 생성한다.

그리고 그 경로를 순회하며 hash_front와 일치하는 primecache를 탐색한다. 일치하는 primecache를 발견했는데 directory가 아니라면 cache 폴더 구조에 문제가 생긴 것이므로 유저에게 이를 알려주고 예외 처리를 수행한다.

만약 primecache(the path of subcache)가 없다면 MISS인 경우로 이때는 해당 directory를 생성한다.

마지막으로 해당 primecache를 경로로하여 subcache를 탐색하는 함수를 호출하고 그 반환값을 다시 결과로써 반환한다.

Figure 3.5 find_subcache pseudo code

```
 * @param path_subcache A const char pointer to the path containing subcaches.
 * @param hash_back A const char pointer to be used as a subcache name.
 * @return [int] HIT:0, MISS:1, FAIL:-1
int find_subcache(const char *path_subcache, const char *hash_back){
    Find the subcache while traversing the path

    For MISS case {
        Try creating a file which contatins a dummy data to the path of subcache
        return EXIT_FAILURE;
    } else For HIT case {
        TODO
        return EXIT_SUCCESS;
    }
}
```

find_primecache에서 전달된 해당 subcache(hashed url의 4번째 이후 문자열)가 존재하는 경로를 순회하며 hash_back과 일치하는 subcache들을 탐색한다.

만약 못 찾았다면 MISS 인 경우로 해당 subcache의 이름으로 더미 데이터를 생성한다.
만약 찾았다면 HIT인 경우로 아직 구현은 되지 않았다.

4 결과화면

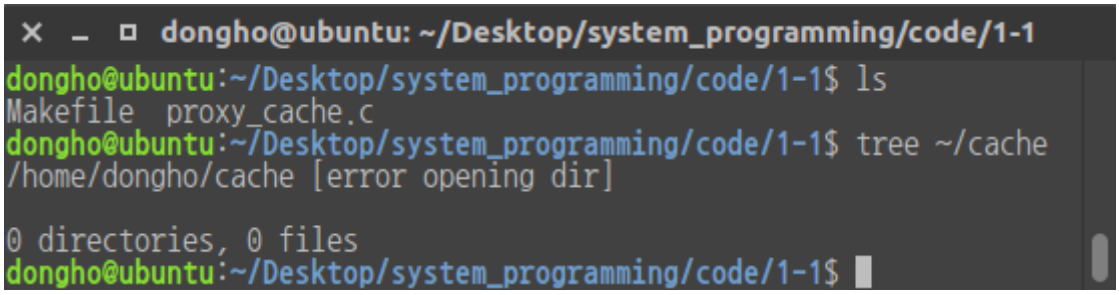

Figure 4.1 초기 상태

맨 처음 초기 상태는 위와 같다.

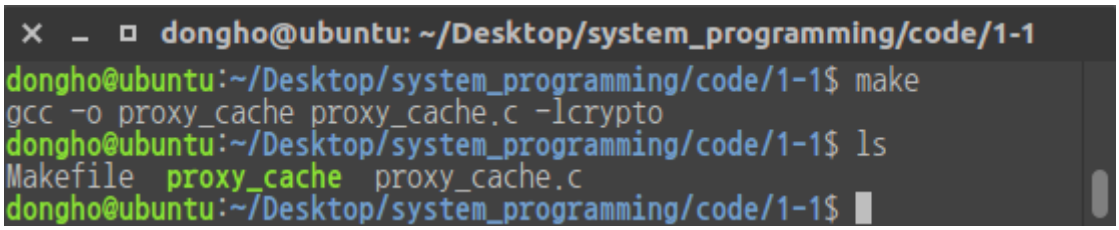

Figure 4.2 make

make를 해주어 실행파일을 생성한다.

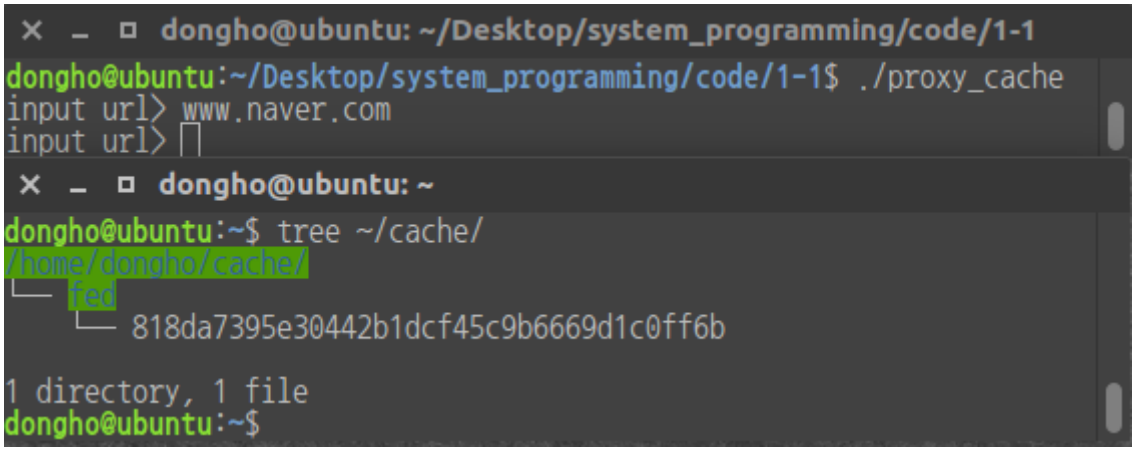

Figure 4.3 1번째 URL 입력

www.naver.com을 입력하자 cache directory가 생성되고 그 아래에 캐시가 생성된것을 확인하였다.

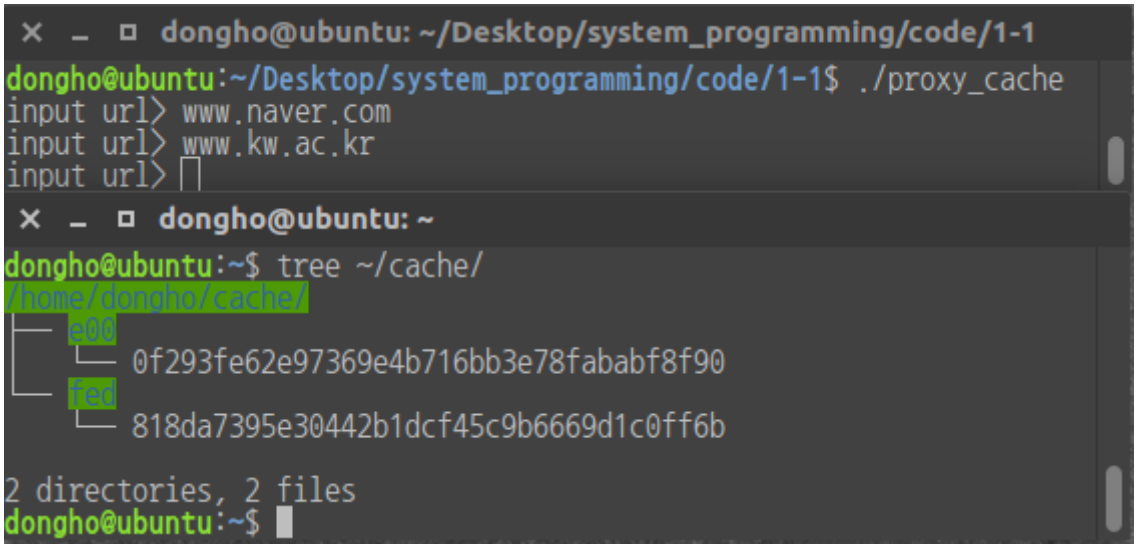

Figure 4.5 2번째 URL 입력

이번엔 www.kw.ac.kr을 입력하였다. 새로운 cache가 생성된것을 확인하였다.

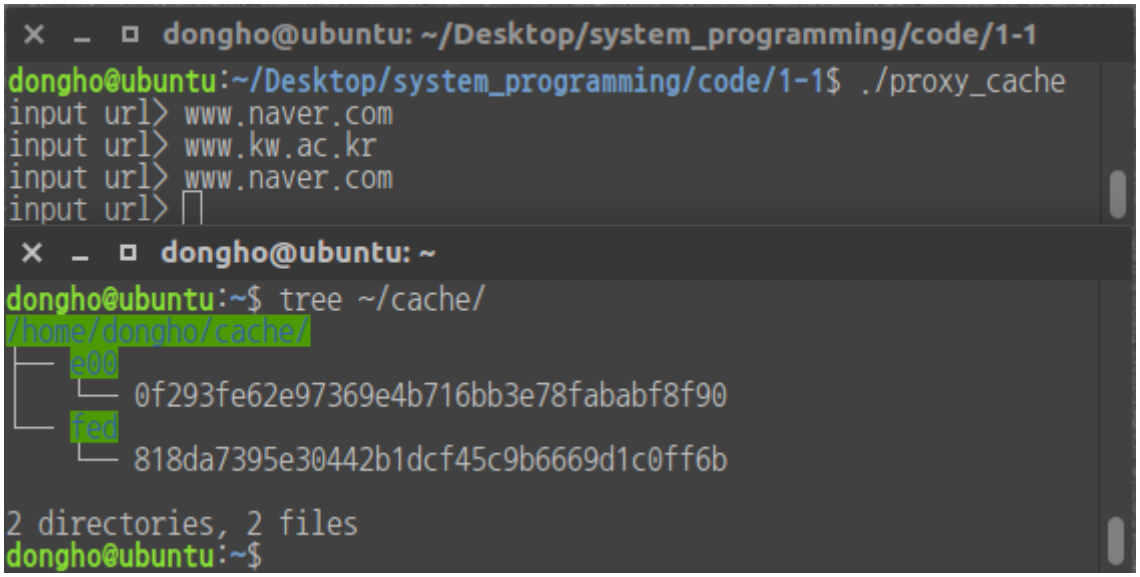

Figure 4.4 중복된 URL 입력

중복된 URL을 입력하여 HIT인 경우를 만들어보았다. 아직 HIT인 경우는 TODO 상태므로 아무일도 일어나지 않았다.

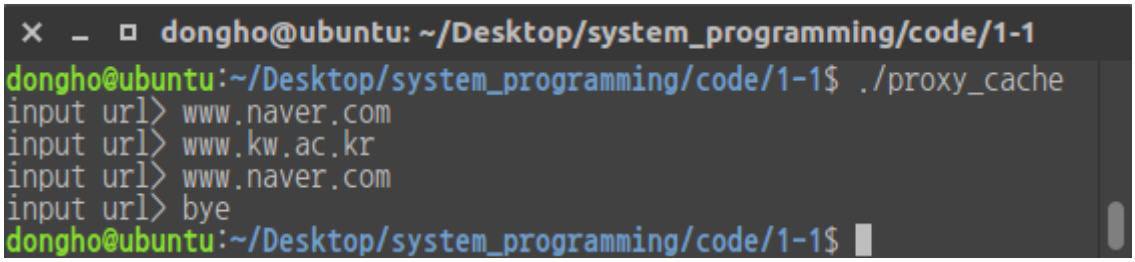

Figure 4.6 bye 입력

bye가 입력되면 종료되는 것 까지 확인하였다.

5 결론 및 고찰

unix에서 C로 무언가를 코딩해보는건 처음이라 왜 안되지(VLA 초기화, 네임스페이스 설정 등등)와 무엇이 더 나은 방법일까(static const vs define vs enum, 함수 관계 등등)를 많이 고민했었다. 확실히 너무 불편하고 이해안되고 힘든 점도 있었지만 이러한 제약 덕분에 컴파일 시간과 런타임 시간에서 우위에 있을 수 있던 것 같다.

위와 별개로 프록시니 캐시니 하는 것도 모두 처음이라 흥미롭다. 아직 전체 3단계에서 1단계를 구현하였고 남은 2단계도 기대된다.

사실 더 쓰고 싶은 말이 좀 있긴한데 과제와 별 관련이 없을것 같아 다음으로 참고문헌들을 소개하며 마무리 하겠다.

6 참고 레퍼런스

http://yhcting.tistory.com/entry/Order-of-file-list-of-the-directory-in-ext4
readdir가 파일 탐색 결과를 반환하는 순서에 대해 상세한 설명이 나와있다.
Stackoverflow에서도 제대로 된 답변을 못찾았는데 티스토리에서 찾을 수 있었다.

https://stackoverflow.com/questions/417142/what-is-the-maximum-length-of-a-url-in-different-browsers 최대 url 버퍼 길이값 선정에 대한 근거

https://en.wikipedia.org/wiki/Comparison_of_file_systems,
https://askubuntu.com/a/859953 UNIX에서 PATH의 최대 경로 버퍼 길이값 선정에 대한 근거 (근거들이라고 했지만 공식적인 것들은 아니다.)

http://pubs.opengroup.org/onlinepubs/7908799/xsh/pwd.h.html 헤더 요약 참고