System programming

2차 과제 - System Programming Assignment #1-3 (proxy server)

학 과 : 컴퓨터 공학과

담당교수 : 황호영

분 반 : 목34

학 번 : 2016722092

성 명 : 정동호

제출날짜 : 2018-04-12

**목차**

1 Introduction 3

2 Flowchart 3

3 Pseudo code 6

4 결과화면 12

5 결론 및 고찰 14

6 참고 레퍼런스 14

# Introduction

현재 만드는 프록시 서버는 여러 클라이언트로부터의 요청을 처리할 수 있어야 한다. 아직 서버/클라이언트는 구현하지 않고 대신, 멀티프로세싱을 구현한다.

# Flowchart

1-2에서도 그랬지만 1-3은 굉장히 많은 부분이 1-2와 겹쳤다. 그래서 실습 조교님께 여쭤보았고 중복되는 부분은 생략하는 대신 복사-붙여넣기 하라는 답변을 받았다. 따라서 이번에 새로 추가되거나 변경된 함수를 제외하곤 상당 부분 이전 보고서와 중복된 내용이 있을 수 있다.

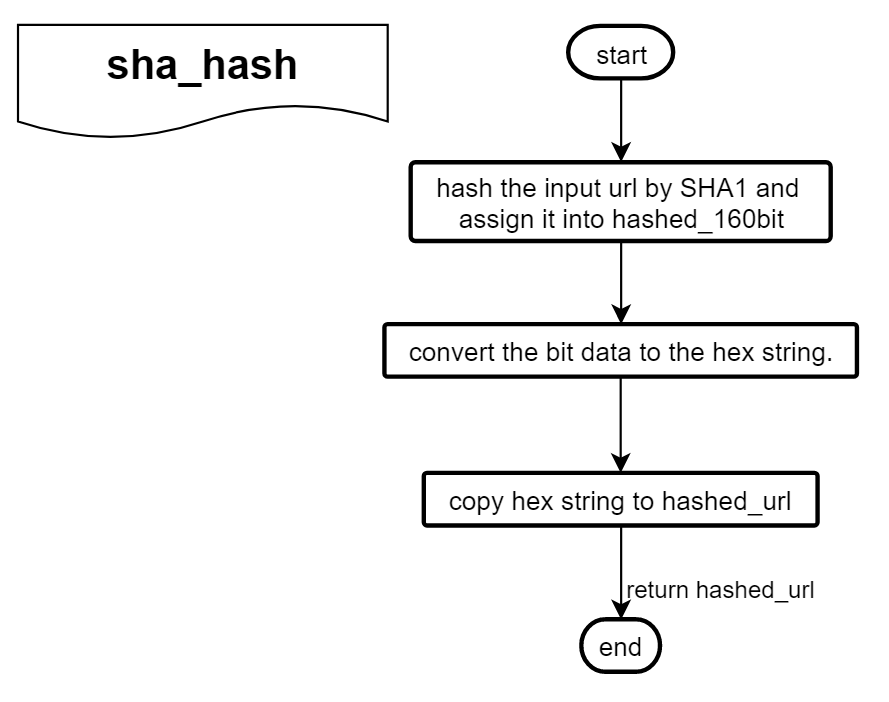
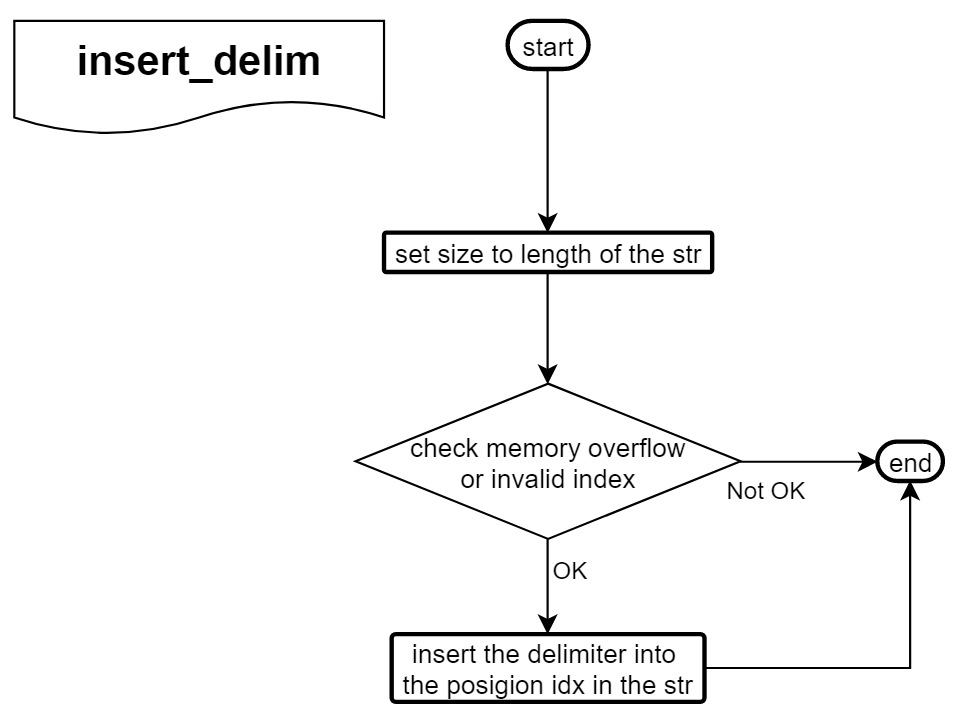
또한 각 순서도의 설명은 매우 짧고 간단하게 하며 자세한 내용은 Pseudo code 부분에서 설명한다.

Figure 2.1 insert\_delim function flowchart

Figure 2.2 sha\_hash function flowchart

**Figure 2.1**의 sha\_hash 함수는 전달받은 문자열을 SHA1으로 해싱하여 그 값을 반환한다.

**Figure 2.2**의 insert\_delim 함수는 인자로 전달된 문자열 포인터에 idx번쨰 위치에 문자 delim을 삽입하여 그 값을 반환한다.

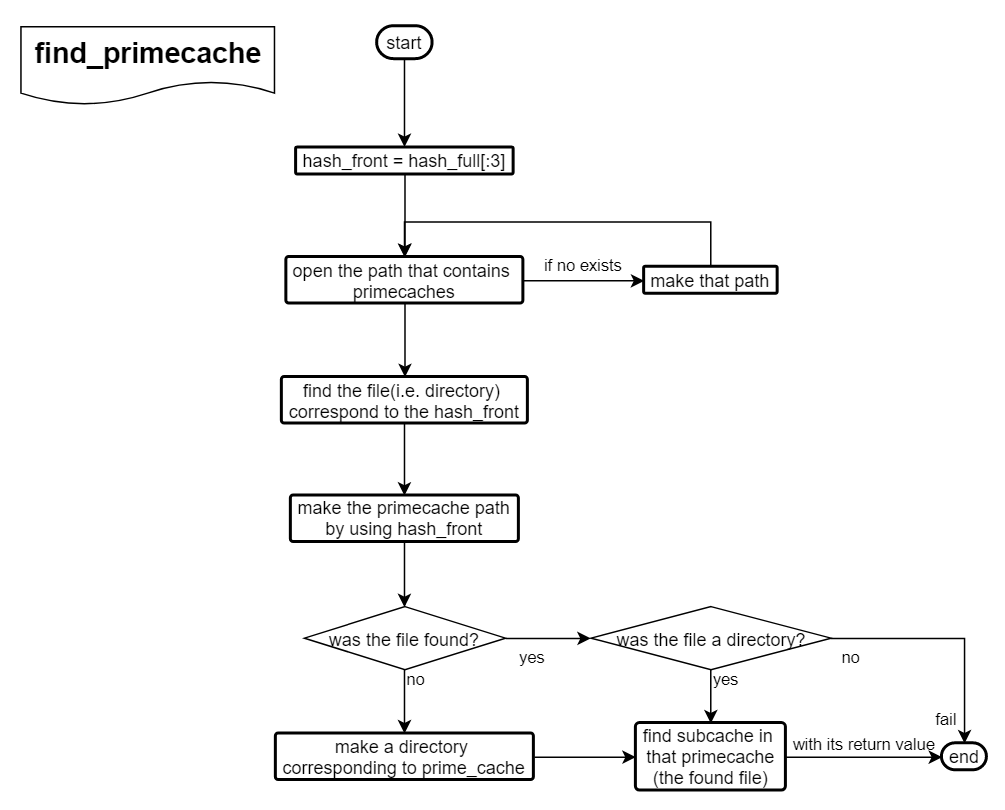
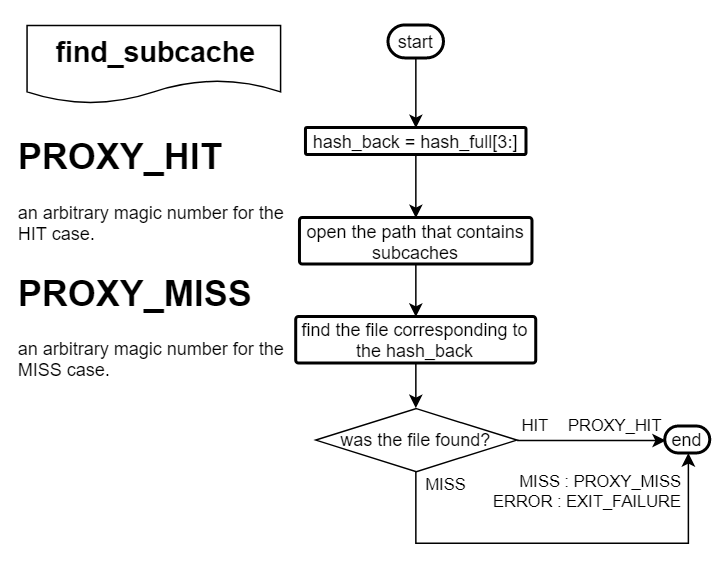


Figure 2.3 find\_subcache function flowchart

Figure 2.4 find\_primecache function flowchart

**Figure 2.3**의 find\_primecache는 해시의 앞부분(primecache)을 인자로 전달받은 경로에서 탐색한다. 만약 해당 폴더가 없다면 그 문자열로 시작하는 해시가 아직 한번도 안나온 것이므로 해당 폴더를 생성해준다. 그 다음 find\_subcache를 호출한다.

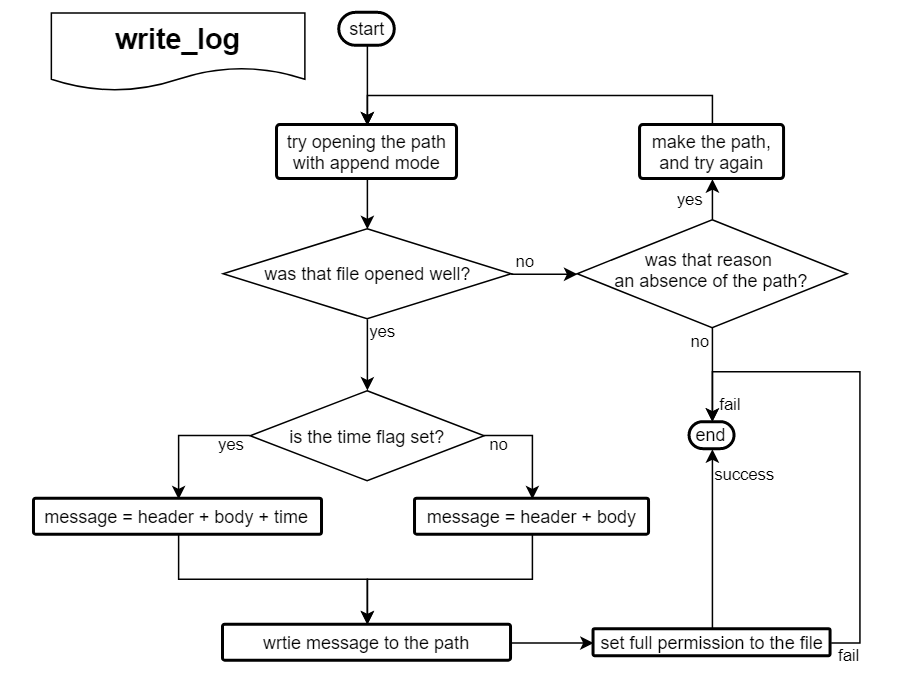
**Figure 2.4**의 find\_subcache는 위의 find\_primecache로부터 호출되는 함수다. 여기서 한번 더 subcache를 탐색하고 찾았다면 HIT를 못찾았다면 MISS를 반환한다.

Figure 2.5 write\_log function flowchart

**Figure 2.5**의 write\_log 함수는 인자로 전달받은 경로에 메세지들을 지정된 서식에 맞게 조합하여 그 내용을 덧쓰는 역할을 한다. 주 목적은 로깅이지만 본 프로그램에서는 캐시파일을 생성하는데도 사용하였다.

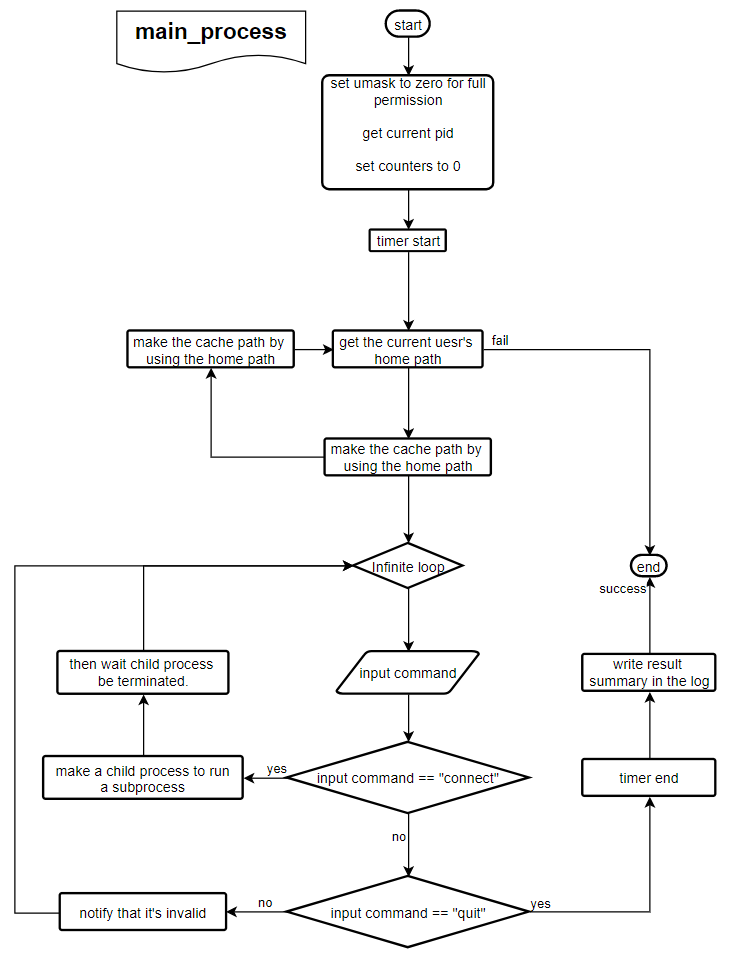


Figure 2.6 main\_process function flowchart

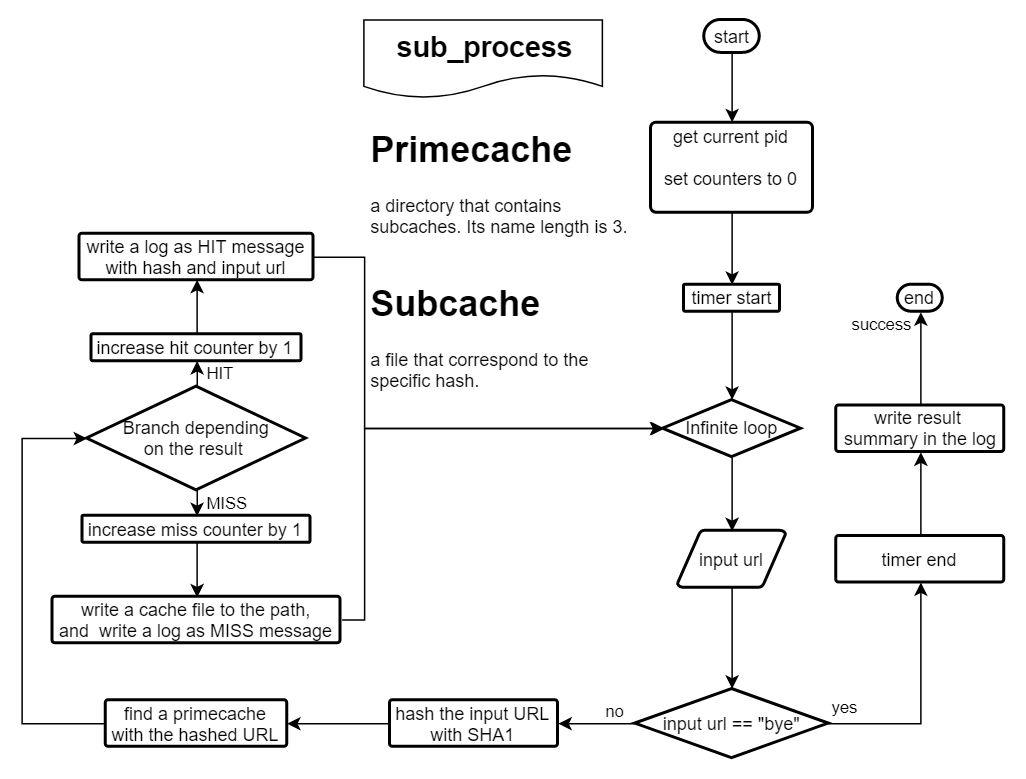


Figure 2.7 sub\_process function flowchart

일단 main은 **Figure 2.6**의 main\_process를 호출하는 역할만 하므로 이번 보고서에는 생략하였다. main\_process는 사용자의 입력을 처리하며 connect일 경우 자식 프로세스를 fork하여 **Figure 2.7**의 sub\_process를 수행하도록 한다. sub\_process는 이전과제의 main과 상당 부분 유사한 역할을 맡고 있으며 URL을 유저로부터 입력받아 캐시를 만들고 로그를 기록하는 역할을 한다.

# 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 insert\_delim function 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;  } |

Figure 3.2 write\_log function 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;  } |

Figure 3.3 find\_primecache function 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;  } |

Figure 3.4 find\_subcache function 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;  }  } |

Figure 3.5 sub\_process function pseudo code

|  |
| --- |
| /\*\*  \* A subprocess running in a child process.  \* **@param** path\_log The path containing a logfile.  \* **@param** path\_cache The path containing primecaches.  \* **@return** [int] Success:EXIT\_SUCCESS  \*/  int sub\_process(const char \*path\_log, const char \*path\_cache){  // pid number for current process  pid\_t current\_pid = getpid();  // char arrays for handling a url  char url\_input[PROXY\_MAX\_URL] = {0};  char url\_hash[PROXY\_LEN\_HASH] = {0};  // char arrays for fullcache  char path\_fullcache[PROXY\_MAX\_PATH] = {0};  // result for finding cache  int res = 0;  // counter for HIT and MISS cases  size\_t count\_hit = 0;  size\_t count\_miss = 0;  // time structs for logging elapsed time  time\_t time\_start = {0}, time\_end = {0};  // temporary buffer for misc  char buf[BUFSIZ] = {0};  // timer start  time(&time\_start);  // set full permission for the current process  umask(0);  // receive inputs till the input is 'bye'  **while**(true){  printf("[%d]input URL> ", current\_pid);  fgets(url\_input, PROXY\_MAX\_URL, stdin);  url\_input[strlen(url\_input)-1] = '\0';  // if input is 'bye' then break loop  **if**(strcmp("bye", url\_input) == 0){  **break**;  }  // hash the input URL and find the cache with it  sha1\_hash(url\_input, url\_hash);  res = find\_primecache(path\_cache, url\_hash);  // insert a slash delimiter at the 3rd index in the url\_hash  insert\_delim(url\_hash, PROXY\_MAX\_URL, 3, '/');  // make a path for fullcache  snprintf(path\_fullcache, PROXY\_MAX\_PATH ,"%s/%s", path\_cache, url\_hash);  **switch**(res){  // If the result is HIT case,  **case** PROXY\_HIT:  count\_hit += 1;  write\_log(path\_log, "[Hit]", url\_hash, true);  write\_log(path\_log, "[Hit]", url\_input, false);  **break**;  // If the result is MISS case,  **case** PROXY\_MISS:  count\_miss += 1;  write\_log(path\_fullcache, "[TEST]", url\_input, true);  write\_log(path\_log, "[Miss]",url\_input, true);  **break**;  **default**:  **break**;  }  }  // timer end  time(&time\_end);  // make a string for terminating the log and write it  snprintf(buf, BUFSIZ, " run time: %d sec. #request hit : %lu, miss : %lu", (int)difftime(time\_end, time\_start), count\_hit, count\_miss);  write\_log(path\_log, "[Terminated]", buf, false);  **return** EXIT\_SUCCESS;  } |

Figure 3.6 main\_process function pseudo code

|  |
| --- |
| /\*\*  \* A main process running a subprocess  \* **@return** [int] Success:EXIT\_SUCCESS, Fail:EXIT\_FAILURE  \*/  int main\_process(){  // pid numbers of processes  pid\_t current\_pid = getpid();  pid\_t child\_pid = 0;  // status of a child process  int child\_status = 0;  // char arrays for handling paths  char path\_home[PROXY\_MAX\_PATH] = {0};  char path\_log[PROXY\_MAX\_PATH] = {0};  char path\_cache[PROXY\_MAX\_PATH] = {0};  // command input by user  char cmd\_input[BUFSIZ] = {0};  // counter for number of subprocesses  size\_t count\_subprocess = 0;  // time structs for logging elpased time  time\_t time\_start = {0}, time\_end = {0};  // temporary buffer for misc  char buf[BUFSIZ] = {0};  time(&time\_start);  // try getting current user's home path and concatenate cache and log paths with it  **if**(getHomeDir(path\_home) == **NULL**){  fprintf(stderr, "[!] getHomeDir fail\n");  **return** EXIT\_FAILURE;  }  snprintf(path\_cache, PROXY\_MAX\_PATH, "%s/cache", path\_home);  snprintf(path\_log, PROXY\_MAX\_PATH, "%s/logfile/logfile.txt", path\_home);  // receive inputs till the input is 'quit'  **while**(true){  printf("[%d]input CMD> ", current\_pid);  fgets(cmd\_input, BUFSIZ, stdin);  cmd\_input[strlen(cmd\_input)-1] = '\0';  // if input is 'connect',  **if**(strcmp(cmd\_input, "connect") == 0){  // then make a child process  **if**((child\_pid = fork()) < 0){ // if it goes fail,  fprintf(stderr, "[!] make sub\_process fail\n");  **continue**;  } **else** **if**(child\_pid == 0){ // in a child process  sub\_process(path\_log, path\_cache);  **return** EXIT\_SUCCESS;  } **else** { // in an original process  count\_subprocess += 1;  wait(&child\_status);  }  } **else** **if**(strcmp(cmd\_input, "quit") == 0){ // if input is 'quit'  **break**; // then escape loop  } **else** { // if input is not valid,  fprintf(stderr, "[!] Invalid command \n");  **continue**;  }  }  // timer end  time(&time\_end);  // make a string for terminating the log and write it  snprintf(buf, BUFSIZ, " run time: %d sec. #sub process: %lu", (int)difftime(time\_end, time\_start), count\_subprocess);  write\_log(path\_log, "\*\*SERVER\*\* [Terminated]", buf, false);  **return** EXIT\_SUCCESS;  } |

# 결과화면

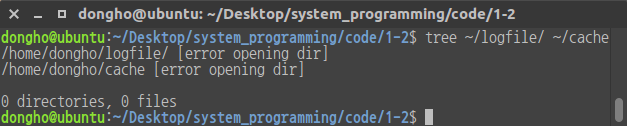


Figure 4.1 log & cache 디렉터리 초기 구조

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

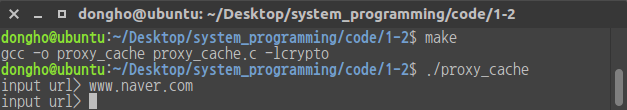


Figure 4.2 컴파일 & URL 입력

make로 컴파일을 하고 실행시킨 뒤 URL [www.naver.com](http://www.naver.com)을 입력했다.

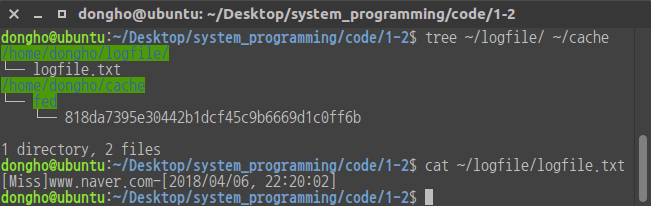


Figure 4.3 logfile 내용

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

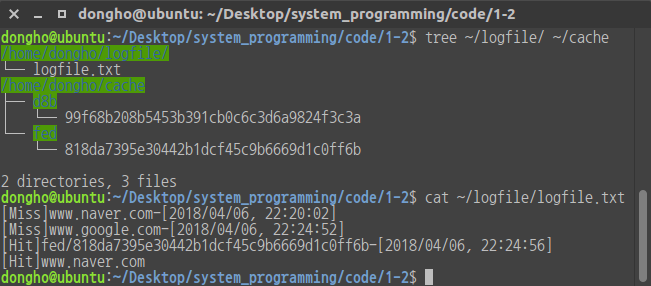
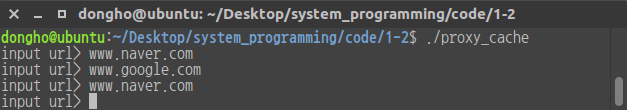
이번엔 MISS인 경우와 HIT인 경우를 모두 입력했다. 결과는 다음과 같다.

Figure 4.4 추가 URL 입력후

Figure 4.5 추가 URL 입력

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

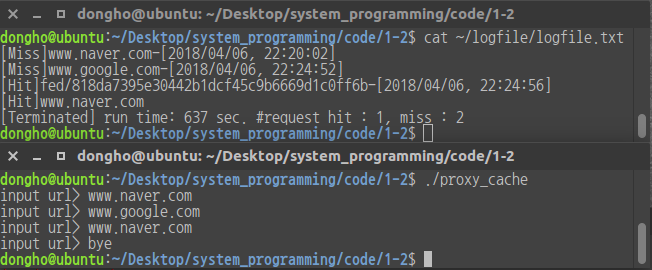


Figure 4.6 종료

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

# 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.

# 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.