# 시스템 프로그래밍 실습 2-2 과제

이름 : 이준휘

학번 : 2018202046

교수 : 최상호 교수님

강의 시간 : 화

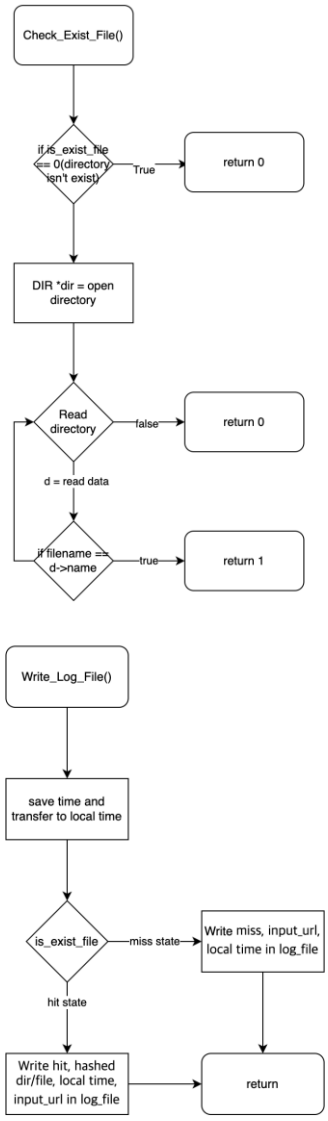
실습 분반 : 목 7, 8

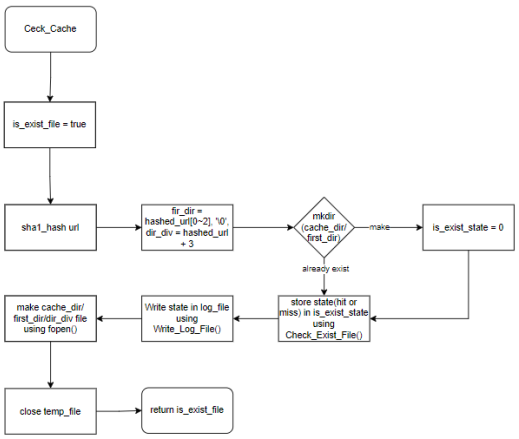
# 1. Introduction

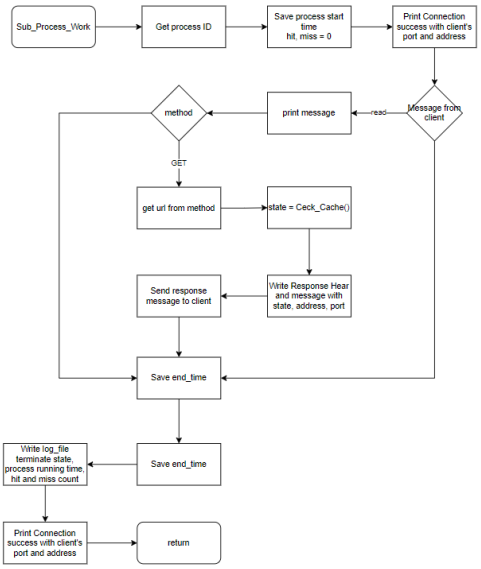
해당 과제는 Proxy server에서 server와 client 간의 기본적인 연결을 구현하게 된다. socket을 통해 다중의 client를 수용할 수 있는 서버를 구현한다. 해당 과제에서는 이전 2-1과 달리 firefox가 client가 된다. firefox로부터 URL 요청을 받을 경우 해당 Message를 분석하여 url을 추출한다. 추출한 url을 바탕으로 이전 1 과제에서 만든 내용을 수행하고, HIT 또는 Miss의 상태에 따라 HTML message를 작성하여 client에게 보낸다.

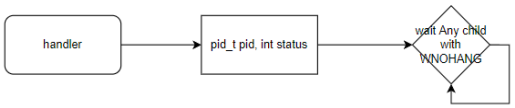
# 2. Flow Chart

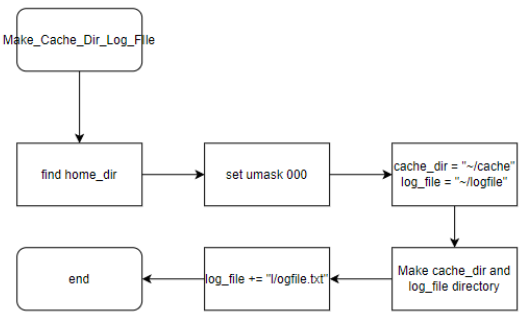
- proxy_cache.c

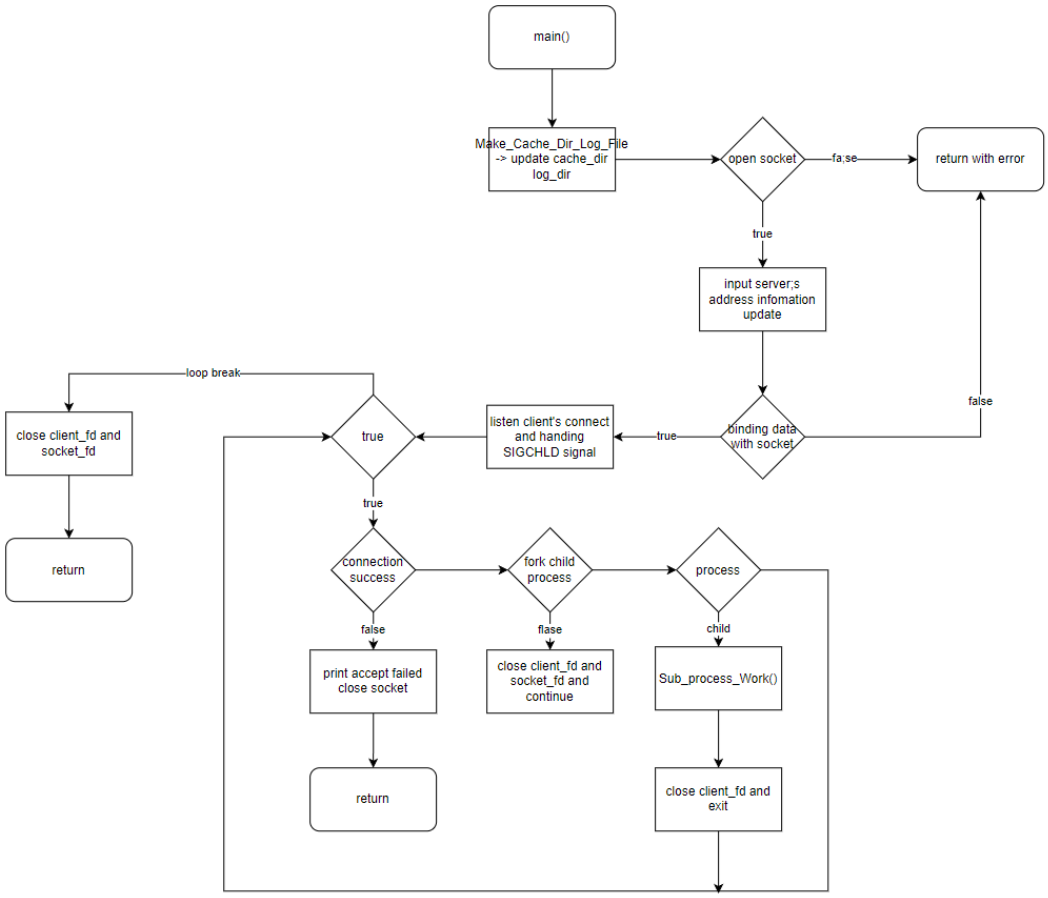



Ceck_Cache
is_exist_file = true
sha1_hash url
fir_dir = hashed_url[0~2], '\0', dir_div = hashed_url + 3
mkdir (cache_dir/ first_dir)
make
is_exist_state = 0
already exist
store state(hit or miss) in is_exist_state using Check_Exist_File()
Write state in log_file using Write_Log_File()
make cache_dir/ first_dir/dir_div file using fopen()
close temp_file
return is_exist_file


Sub_Process_Work
Get process ID
Save process start time
hit, miss = 0
Print Connection success with client's port and address
Message from client
read
print message
method
GET
get url from method
state = Ceck_Cache()
Write Response Hear and message with state, address, port
Send response message to client
Save end_time
Save end_time
Write log_file terminate state, process running time, hit and miss count
Print Connection success with client's port and address
return


handler
pid_t pid, int status
wait Any child with WNOHANG


Make_Cache_Dir_Log_File
find home_dir
set umask 000
cache_dir = "~/cache"
log_file = "~/logfile"
Make cache_dir and log_file directory
log_file += "l/ogfile.txt"
end




## 3. Pseudo Code

```
static void handler(){
    pid_t pid;
    int status;
    Wait Any child with WNOHANG
}

Make_Cache_Dir_Log_File(char* cache_dir, char* log_file){
    getHomeDirectory();
    cache_dir = ~/cache;
    log_file = ~/logfile;
```

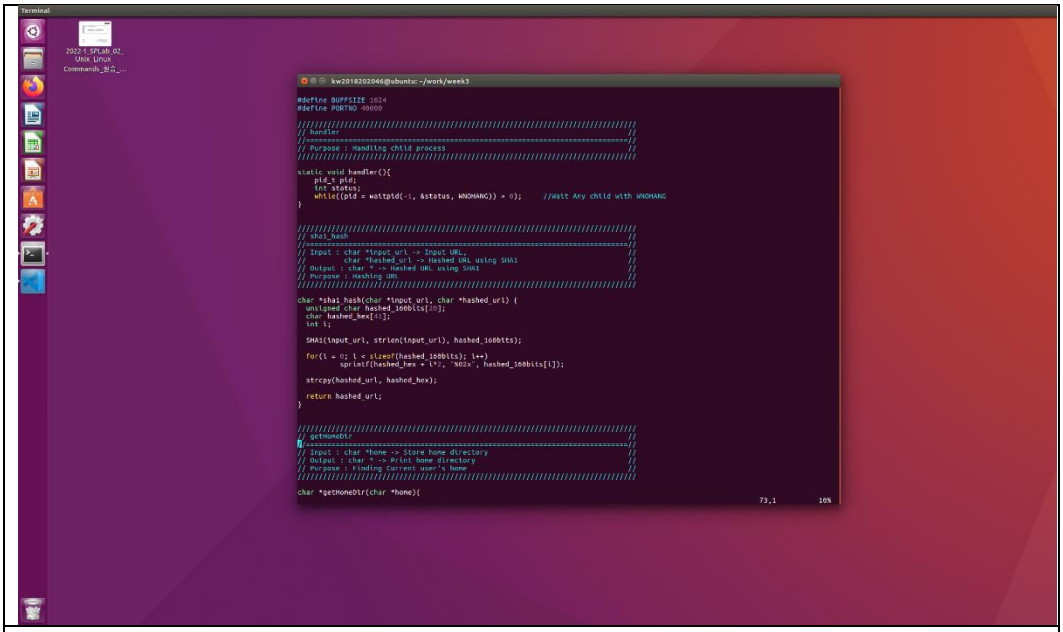
```
    set umask 000;
    make cache and log directory;
    log_file += /logfile.txt;
}

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], '₩0' };
    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[BUFFSIZE] = { 0 };
        char response_message[BUFFSIZE] = { 0 };
        char temp[BUFFSIZE] = { 0 };
        char method[BUFFSIZE] = { 0 };
        char url[BUFFSIZE] = { 0 };
        char *token = NULL;
        int len;
        int state, hit = 0, miss = 0;

        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)
            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
```

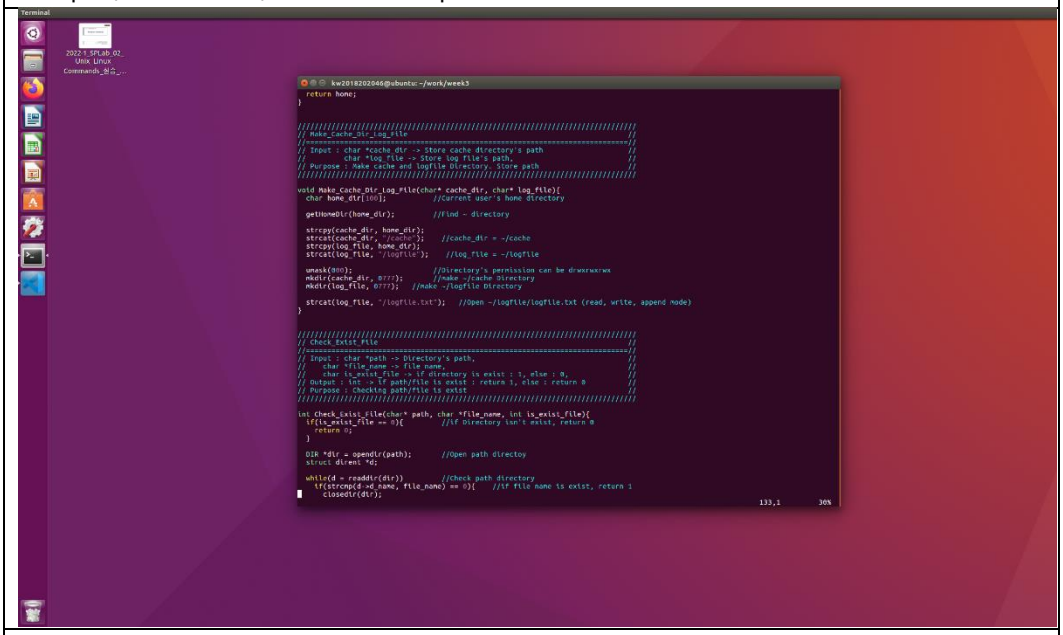
```
print Terminate connection with client address and port;
return;
}

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



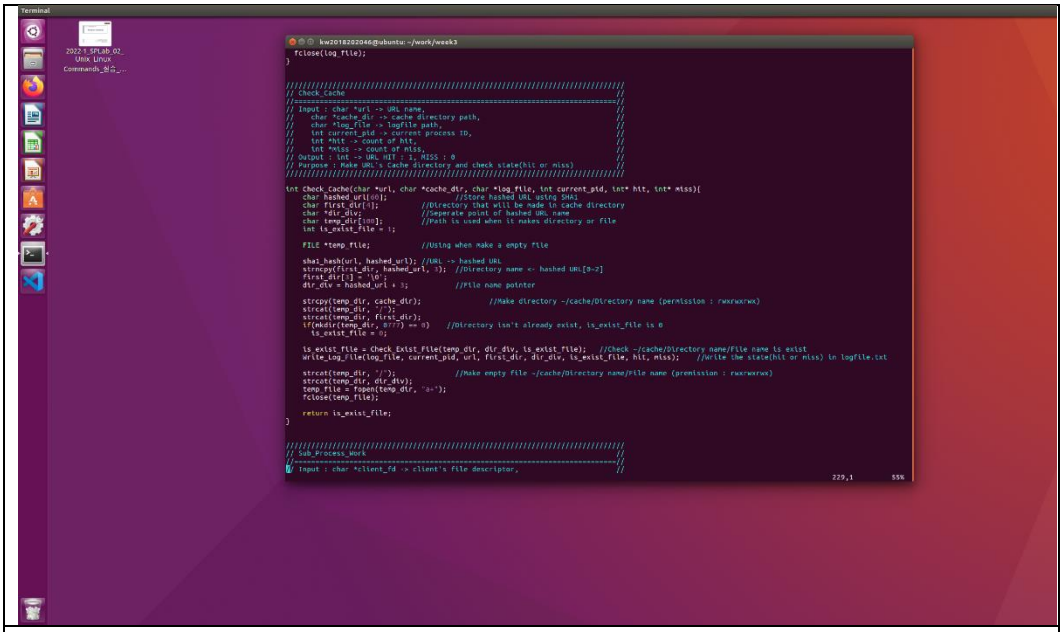
```
kw2018202046@ubuntu: ~/work/week3

#define BUFFSIZE 1024
#define PORTNO 40000

////////////////////////////////////////////////////////////////////////////////
// handler                                                                    //
//============================================================================//
// Purpose : Handling child process                                           //
////////////////////////////////////////////////////////////////////////////////

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


////////////////////////////////////////////////////////////////////////////////
// sha1_hash                                                                  //
//============================================================================//
// Input : char *input_url -> Input URL,                                      //
//         char *hashed_url -> Hashed URL using SHA1                          //
// Output : char *-> Hashed URL using SHA1                                    //
// Purpose : Hashing URL                                                      //
////////////////////////////////////////////////////////////////////////////////

char *sha1_hash(char *input_url, char *hashed_url) {
  unsigned char hashed_160bits[20];
  char hashed_hex[41];
  int i;

  SHA1(input_url, strlen(input_url), hashed_160bits);

  for(i = 0; i < sizeof(hashed_160bits); i++)
          sprintf(hashed_hex + i*2, "%02x", hashed_160bits[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){
                                                                    73,1          10%
```

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

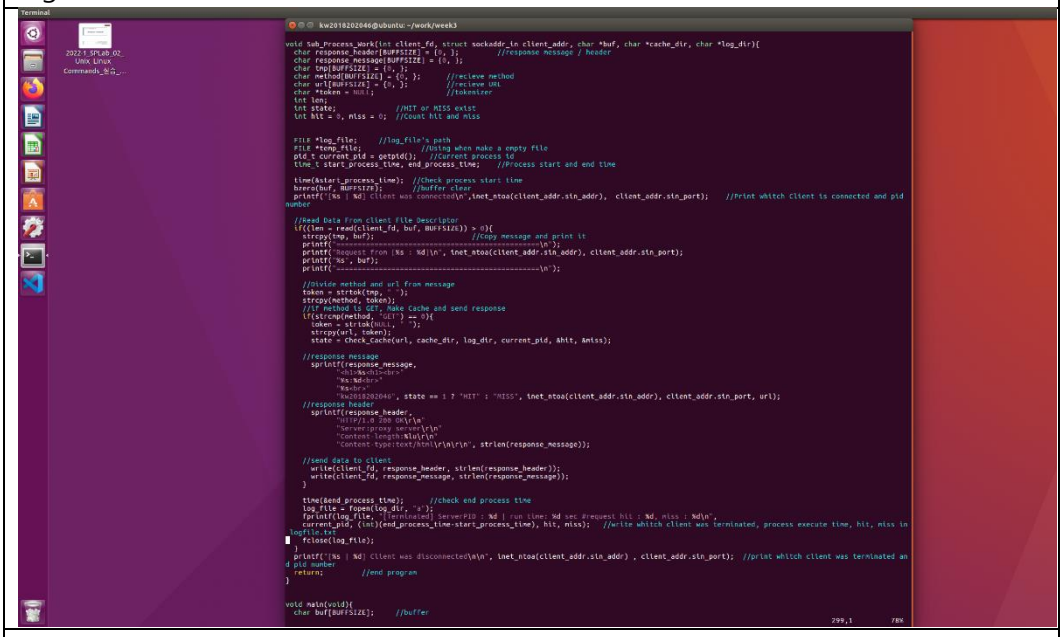


```
kw2018202046@ubuntu: ~/work/week3
  return home;
}


////////////////////////////////////////////////////////////////////////////////
// Make_cache_Dir_Log_File                                                    //
//============================================================================//
// Input : char *cache_dir -> Store cache directory's path                    //
//         char *log_file -> Store log file's path,                           //
// Purpose : Make cache and logfile Directory, Store path                     //
////////////////////////////////////////////////////////////////////////////////

void Make_Cache_Dir_Log_File(char* cache_dir, char* log_file){
  char home_dir[100];           //Current user's home directory

  getHomeDir(home_dir);         //Find ~ directory

  strcpy(cache_dir, home_dir);
  strcat(cache_dir, "/cache");    //cache_dir = ~/cache
  strcpy(log_file, home_dir);
  strcat(log_file, "/logfile");   //log_file = ~/logfile

  umask(000);                   //Directory's permission can be drwxrwxrwx
  mkdir(cache_dir, 0777);       //make ~/cache Directory
  mkdir(log_file, 0777);   //make ~/logfile Directory

  strcat(log_file, "/logfile.txt");   //Open ~/logfile/logfile.txt (read, write, append mode)
}


////////////////////////////////////////////////////////////////////////////////
// Check_Exist_File                                                           //
//============================================================================//
// Input : char *path -> Directory's path,                                    //
//    char *file_name -> file name,                                           //
//    char is_exist_file -> if directory is exist : 1, else : 0,              //
// Output : int -> if path/file is exist : return 1, else : return 0          //
// Purpose : Checking path/file is exist                                      //
////////////////////////////////////////////////////////////////////////////////

int Check_Exist_File(char* path, char *file_name, int is_exist_file){
  if(is_exist_file == 0){         //if Directory isn't exist, return 0
    return 0;
  }

  DIR *dir = opendir(path);       //Open path directoy
  struct dirent *d;

  while((d = readdir(dir)))       //Check path directory
    if(strcmp(d->d_name, file_name) == 0){    //if file name is exist, return 1
      closedir(dir);
                                                                    133,1         30%
```

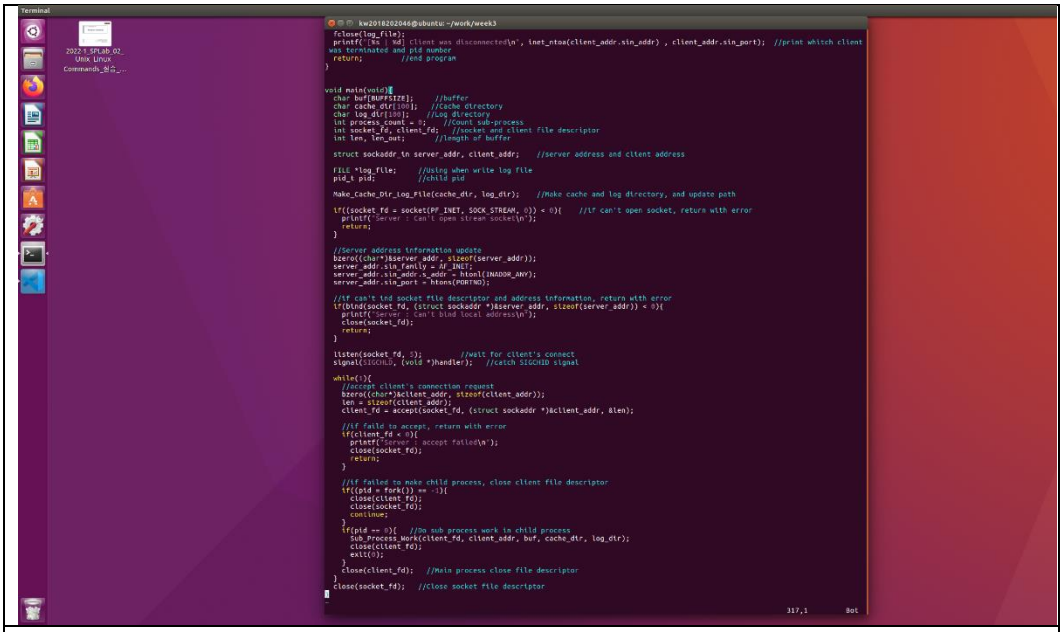
해당 함수는 기존의 cache directory와 log directory를 생성하고 log_file과 cache_dir의 path를 저장하는 역할을 수행한다.



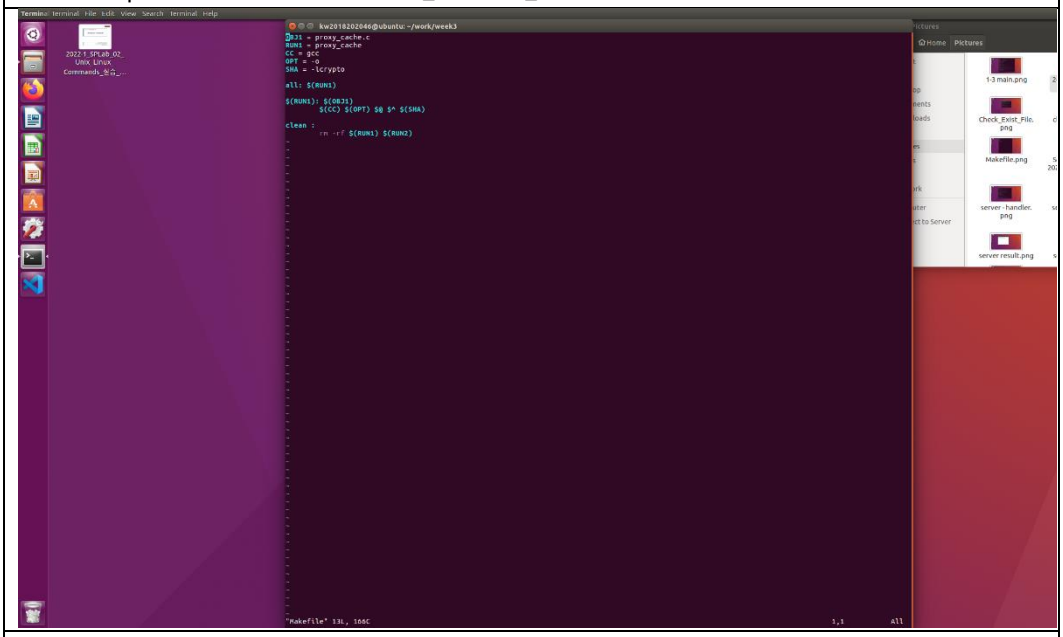
해당 함수는 기존의 Sub_Process_Work 함수에서 URL을 받아 hashing, cache 생성, logfile 작성을 맡고 있었던 부분을 분리하여 작성한 함수다.



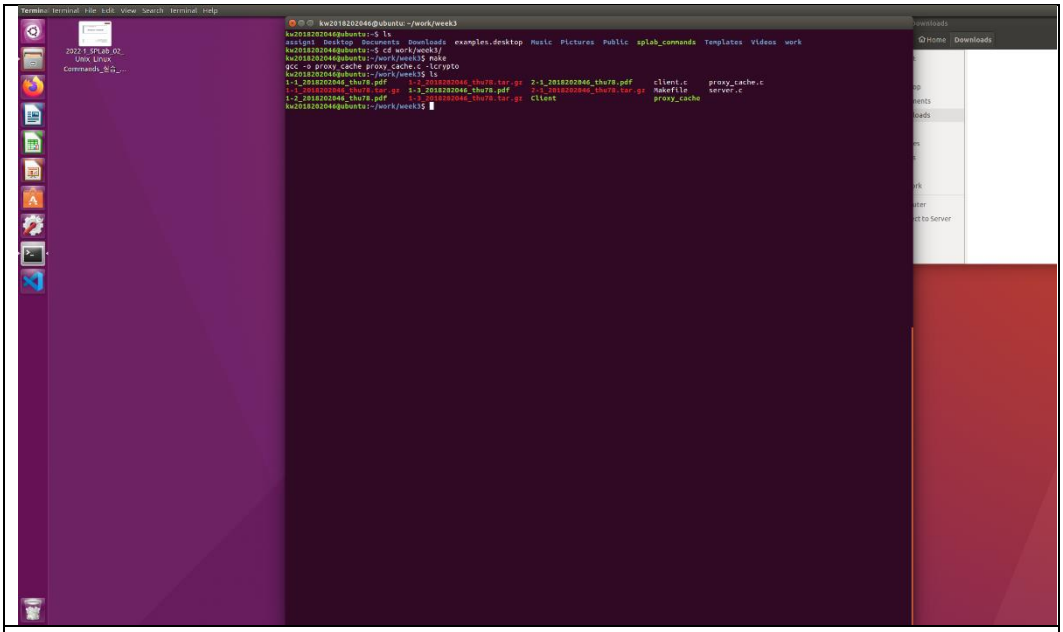
해당 함수는 2-1에서 존재하던 Sub_Process_Work 함수에서 일부 수정하였다. 우선 큰 변경점으로는 read()를 받을 경우 해당 message를 출력하고 해당 read()의 method와 url을 분리한다. method가 get인 경우에 한해 url을 Ceck_Cache()함수를 통해 처리하며 state를 통해 HIT/MISS 상태를 기억한다. 기억한 상태는 response message를 작성하는데 사용하며 header와 함께 message를 client에게 전송한다. 이외의 부분은 이전 함수와 동일하다.



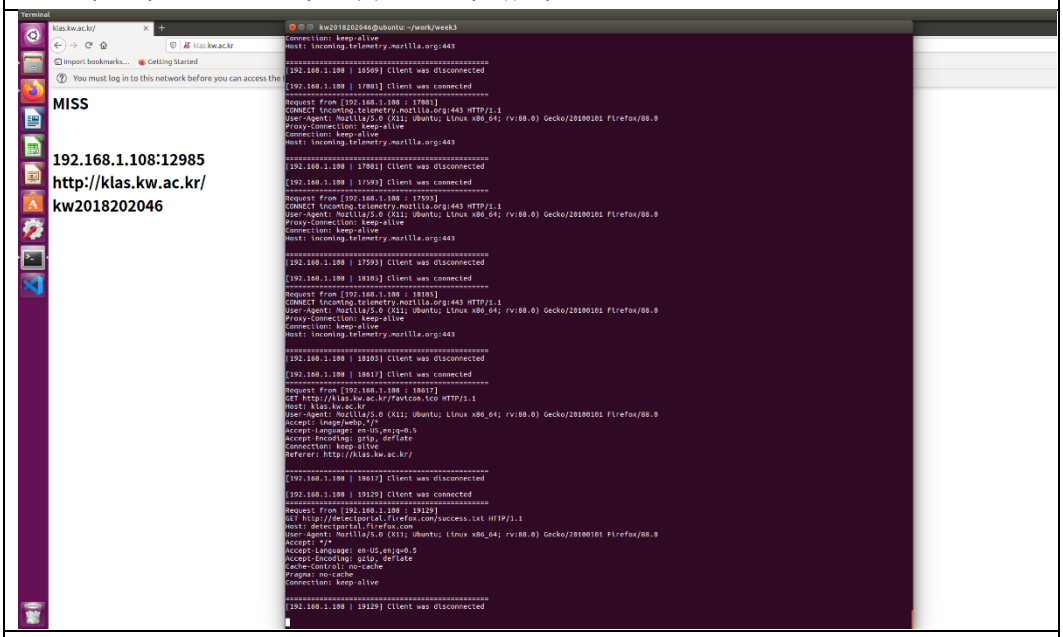
해당 그림은 server의 메인함수다. 해당 함수에서는 socket을 열고 binding을 통해 server의 정보를 묶는다. 그리고 listen을 통해 연결을 받는다. 연결을 accept할 경우 child process를 생성하여 Sub_Process_Work 작업을 수행한다.



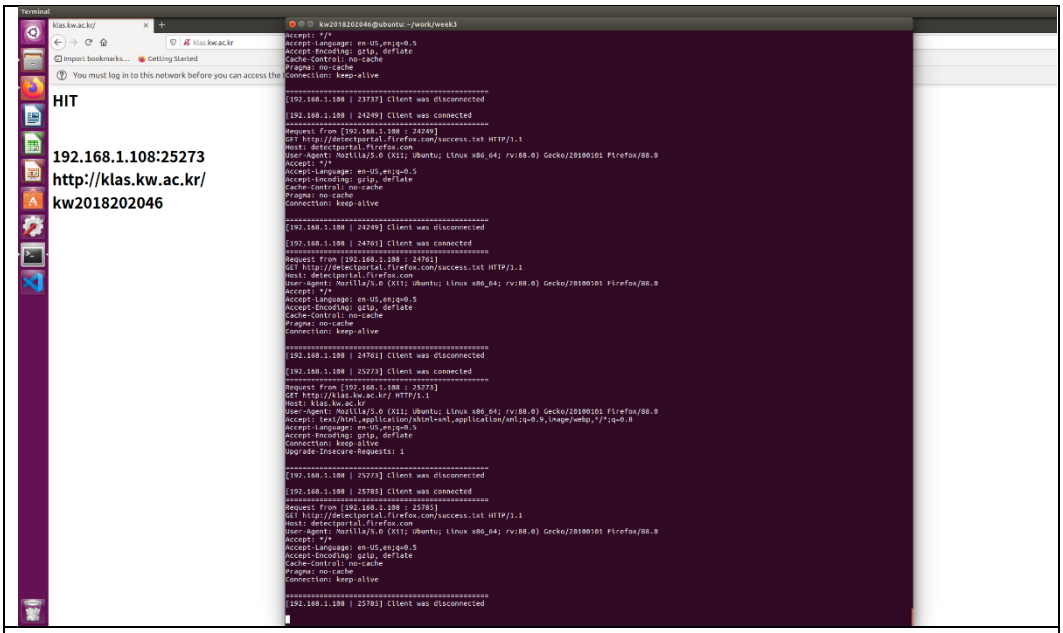
해당 파일은 변경된 Makefile이다. proxy_cache.c의 파일을 proxy_cache 실행파일로 만드는 역할을 수행한다.



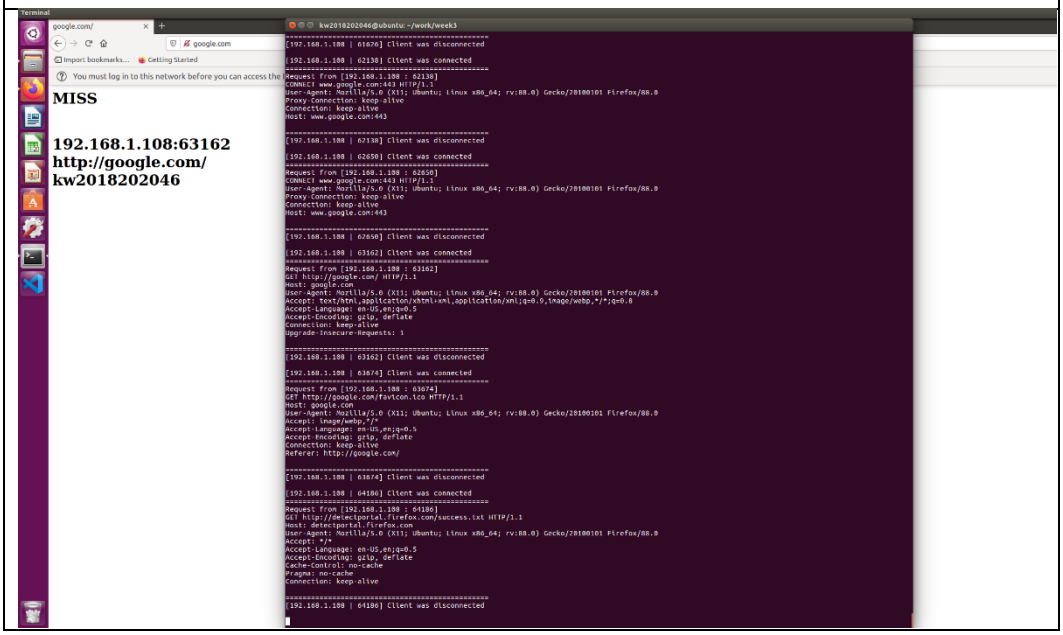
해당 그림은 기존의 cache와 logfile이 없는 상태에서 시작한다. make 명령을 통해 실행파일이 정상 생성되는 것을 볼 수 있다.

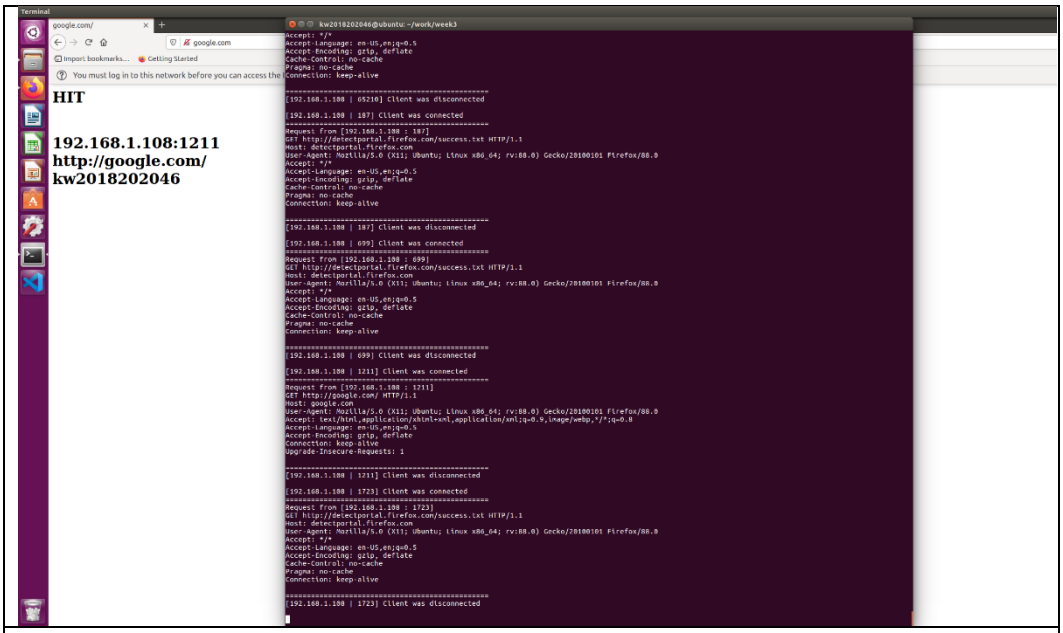



proxy_cache를 실행 후 klas.kw.ac.kr을 입력하였을 때 기존 cache가 없었기 때문에 MISS가 뜬 모습을 볼 수 있다. 많은 출력이 나오는 것은 firefox에서 지속적으로 신호를 보내기 때문에 그렇다.

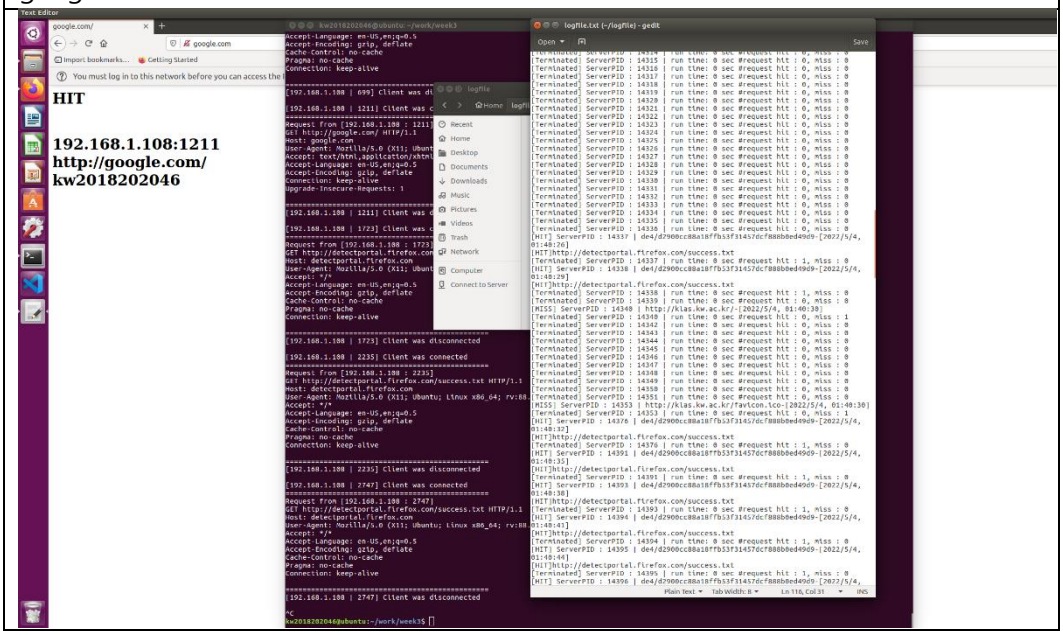


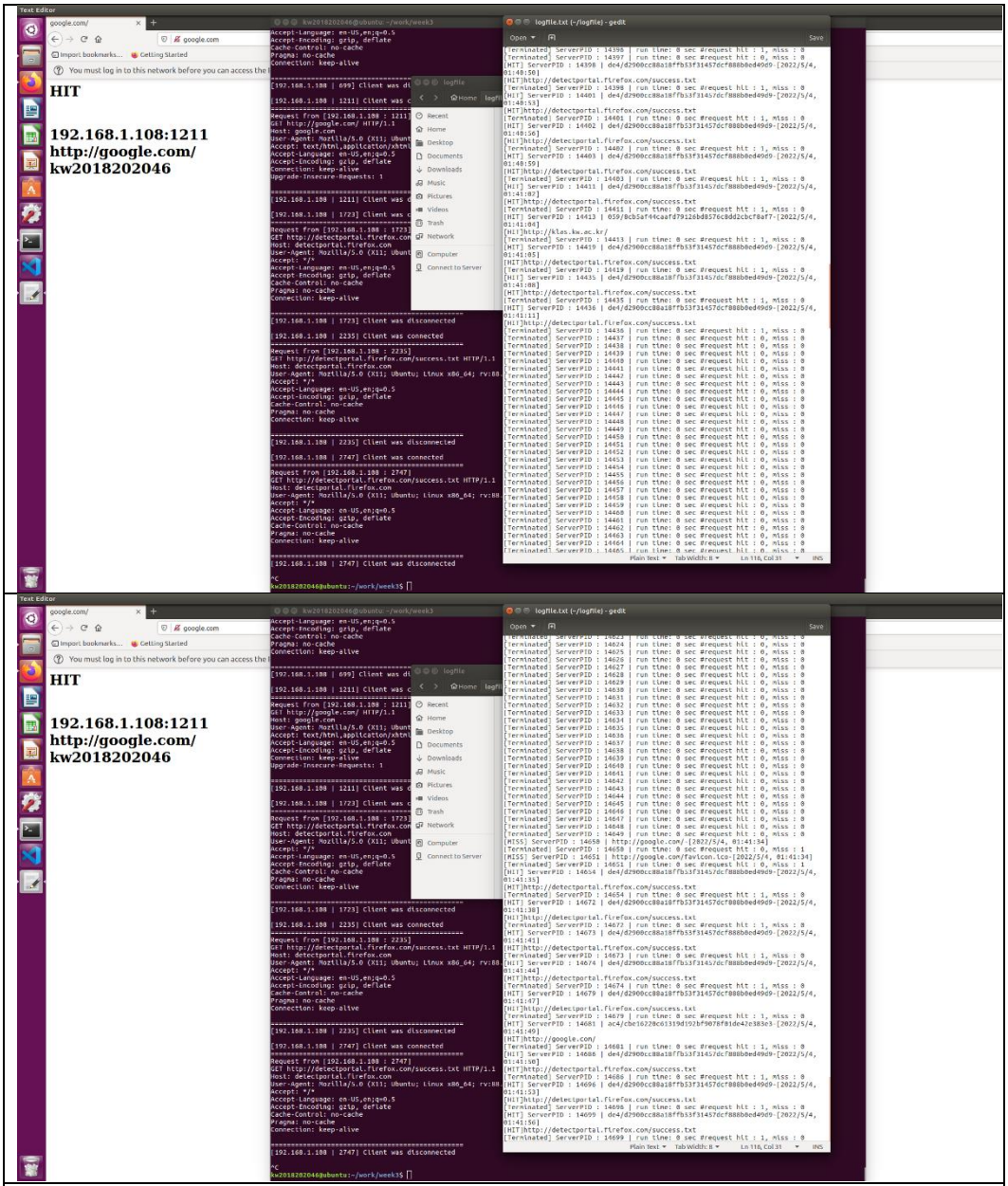

다시 한번 실행하였을 때에는 cache가 있기 때문에 HIT 상태를 출력하는 모습이다.






google.com을 통해서도 동일한 결과를 얻을 수 있었다.






logfile을 살펴보면 URL을 받았을 때 url 정보와 해당 URL에 대한 HIT 또는 MISS 상태가 기존 출력대로 제대로 출력되는 모습을 볼 수 있다. 이를 통해 해당 프로그램이 정상적으로 구현되었음을 확인하였다.

## 5. 고찰

해당 과제를 통해서 단순하게 server와 client간에 message를 전달하는 것이 아니라 Request 요청을 받아서 처리하고 Response Message를 보내는 일련의 과정을 이해할 수 있었다. 또한 컴퓨터 네트워크 강의에서 배웠던 HTTP Protocol을 실제적으로 눈으로 확인할 수 있던 과제였다. 또한 Response Message를 작성하면서 해당 Header와 MTML 양식에 대해서도 공부할 수 있었던 과제였다.

## 6. Reference

강의 자료만을 참고