시스템 프로그래밍 실습 2-1 과제

이름 : 이준휘

학번 : 2018202046

교수 : 최상호 교수님

강의 시간 : 화

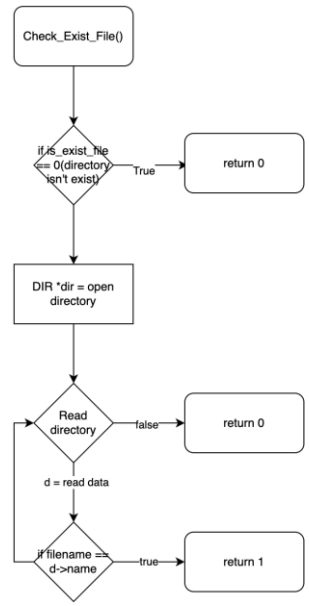
실습 분반 : 목 7, 8

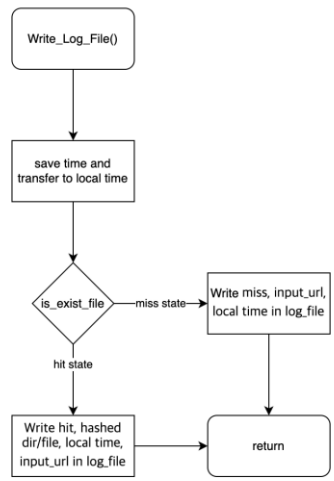
1. Introduction

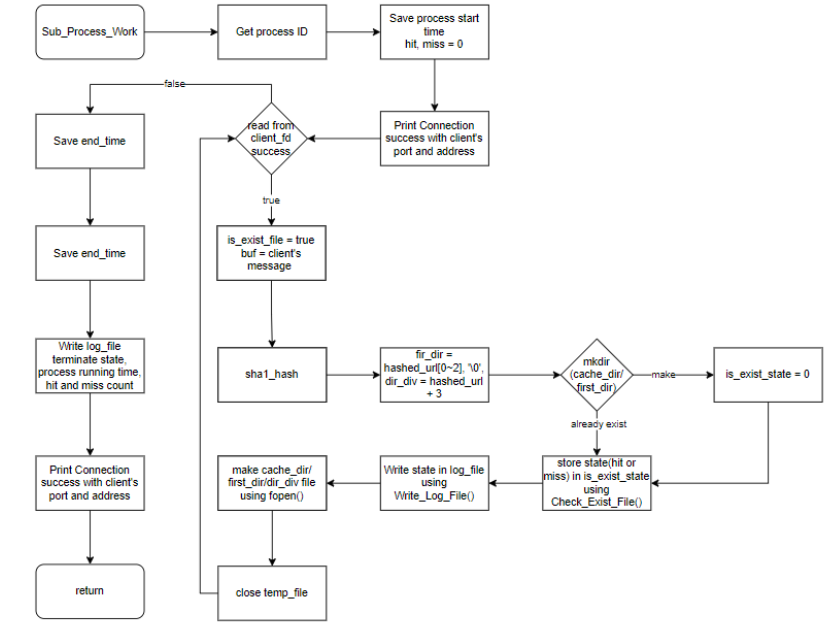
해당 과제는 Proxy server에서 server와 client 간의 기본적인 연결을 구현하게 된다. socket을 통해 다중의 client를 수용할 수 있는 서버를 구현한다. client 또한 socket을 열어 연결을 시도한다. 만약 연결이 성공할 경우 server는 child process를 만들어 다음 작업을 수행한다. client에서 전송한 URL을 받아 기존 1-3까지 구현했던 작업인 Cache, log 생성을 하고 client에게 HIT인지 MISS인지를 알린다. 해당 명령은 bye를 입력받을 때까지 반복한다.

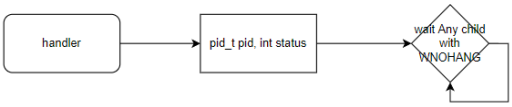
2. Flow Chart

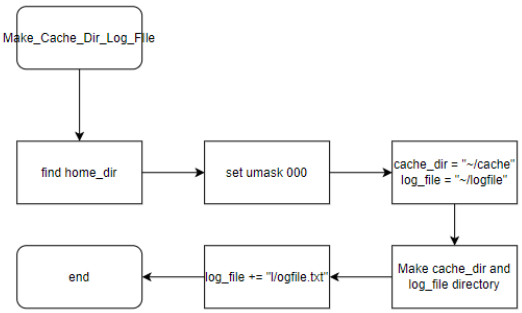
- server.c

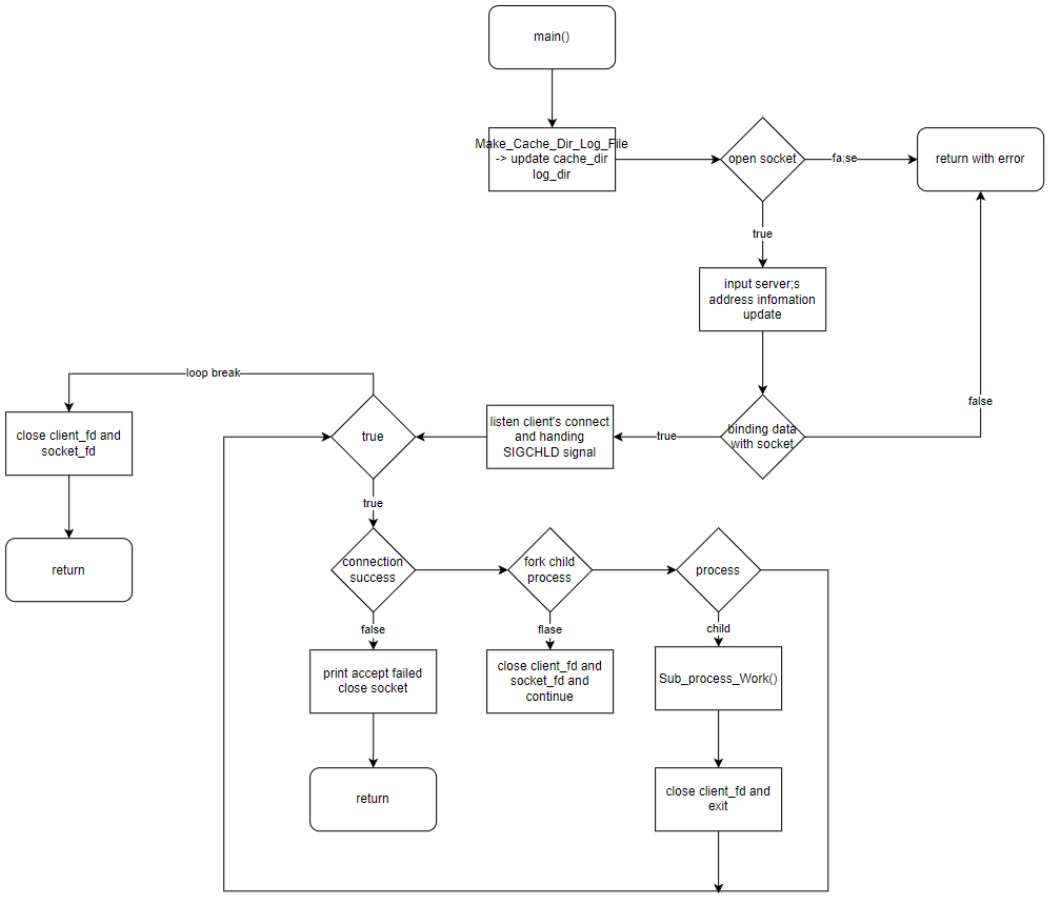




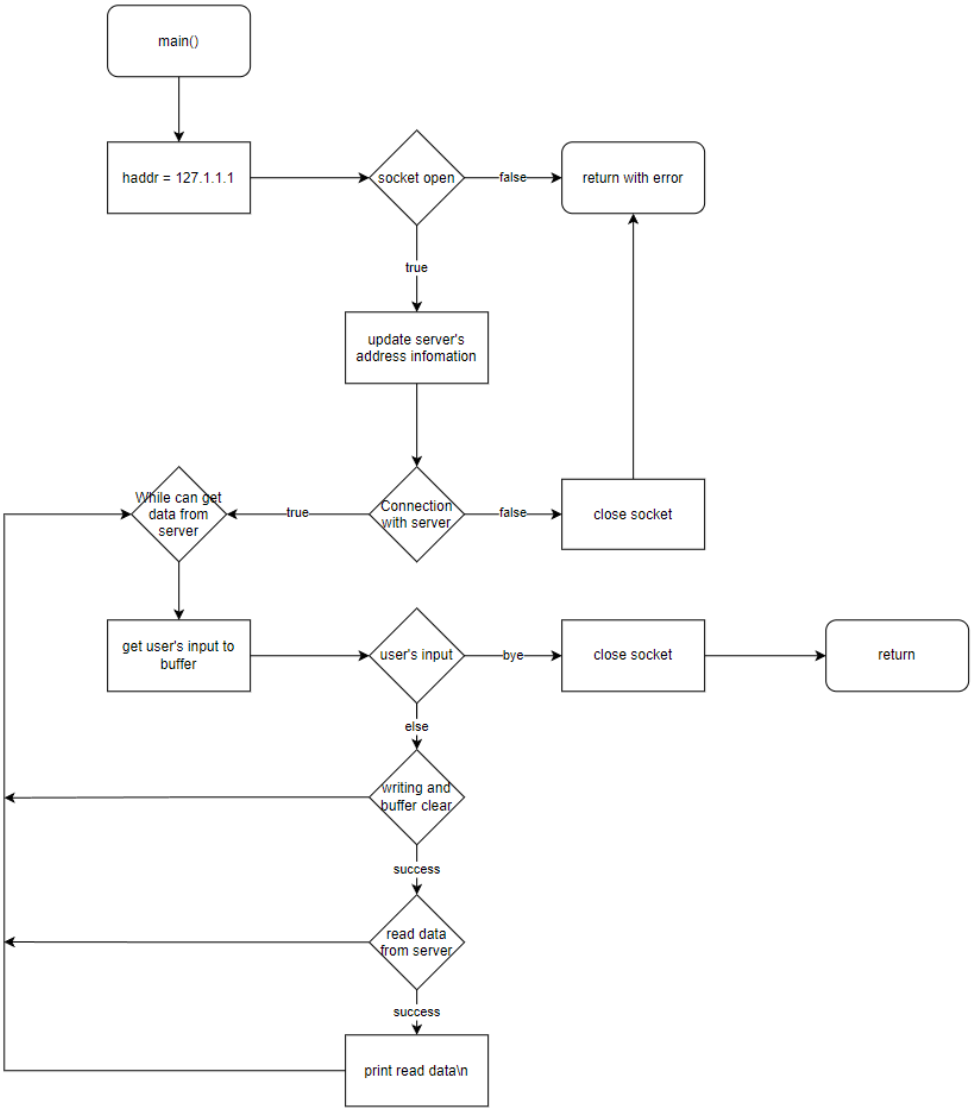

Sub_Process_Work
Get process ID
Save process start time
hit, miss = 0
Print Connection success with client's port and address
read from client_fd success
false
true
Save end_time
Save end_time
Write log_file terminate state, process running time, hit and miss count
Print Connection success with client's port and address
return
is_exist_file = true
buf = client's message
sha1_hash
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


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


main()
Make_Cache_Dir_Log_File
-> update cache_dir
log_dir
open socket
false
return with error
true
input server;s
address infomation
update
binding data
with socket
true
false
listen client's connect
and handing
SIGCHLD signal
true
loop break
close client_fd and
socket_fd
return
true
true
connection
success
fork child
process
process
false
flase
child
print accept failed
close socket
close client_fd and
socket_fd and
continue
Sub_process_Work()
return
close client_fd and
exit

- client.c



3. Pseudo Code

- server

```
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 Sub_Process_Work(int client_fd, struct sock_addr, char *buf, char *char_dir, FILE *log_file){
        char[60] hashed_url;
        char[4] first_dir;
        char *dir_div;
        char[100] temp_dir;
        int is_exist_file, hit = 0, miss = 0;

        FILE *temp_file;
        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;
        while(read data from client_fd to buf){
            is_exist_file = 1;
            hashed_url = hashed input using sha1;
            first_dir = { hashed_url[0~3], '₩0' };
            dir_div = hashed_url + 3 address
            temp_dir = ~/cache/first_dir;
```

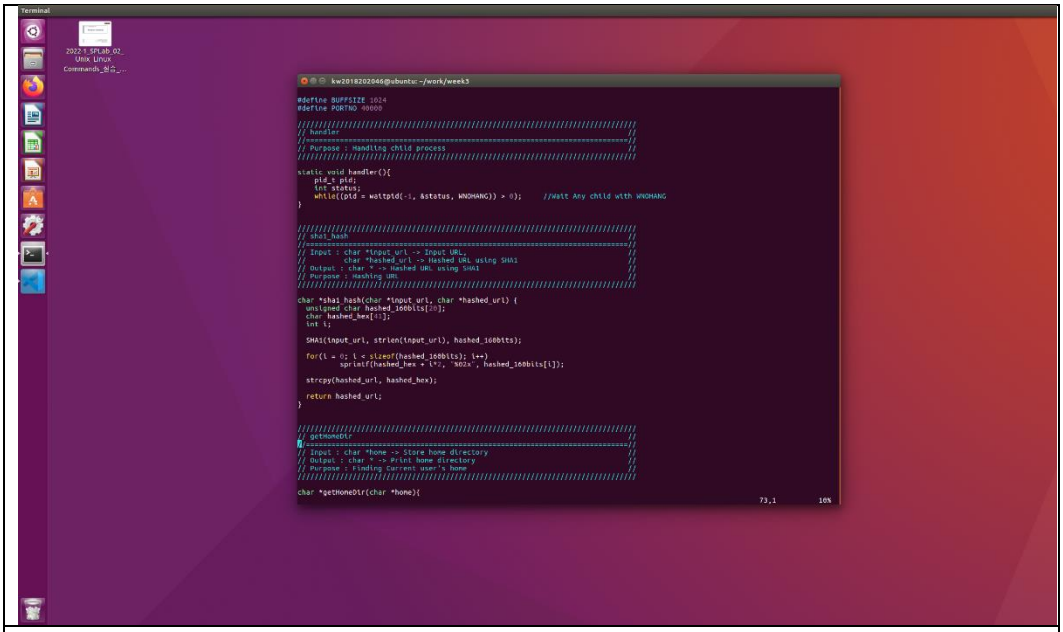
```
        if make temp_dir directory(permission = drwxrwxrwx)
            is_exist_file = 0;
        is_exist_file = hit or miss state;
        Write state, url, dir/file name, time in logfile.txt;
        temp_dir = ~/cache/first_dir/dir_div;
        temp_file = make temp_dir file and open file;
        temp_file close;
        write HIT or MISS state to Server_fd;
    }
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;
}
```

- client

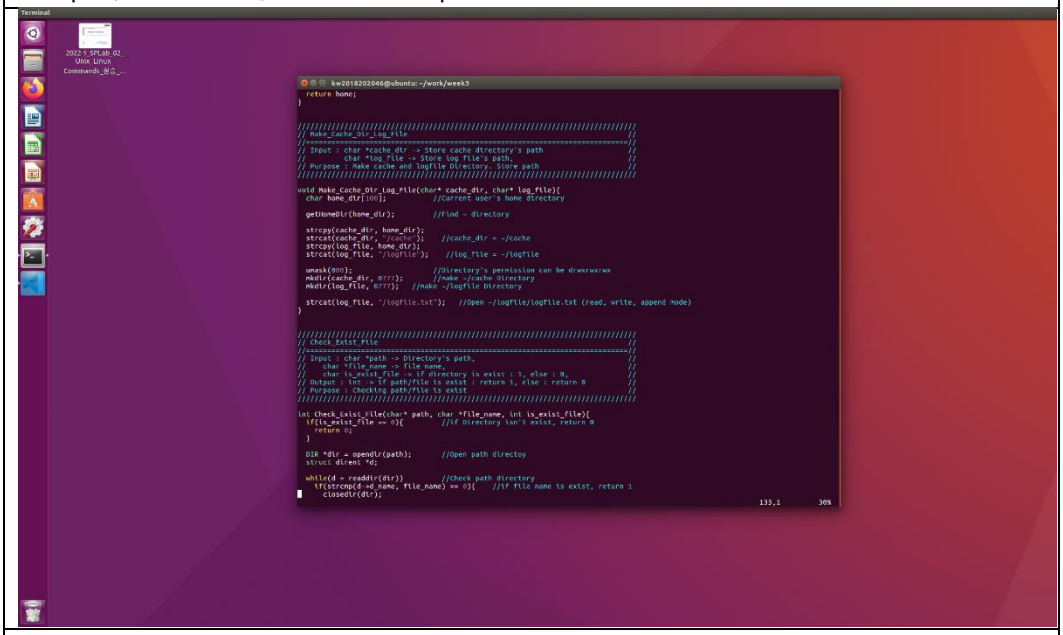
```
main(){
    if open socket is failed, return with error;
    update server's address information;
    if connection failed, close socket and print error, return;

    while(input data to buf){
        if buf has bye command, break;
        if write to server_fd successed{
            if receive data from server_fd successed{
                print receive data + ₩n;
            }
        }
    }
    close socket file descriptor;
    return
}
```

4. 결과 화면



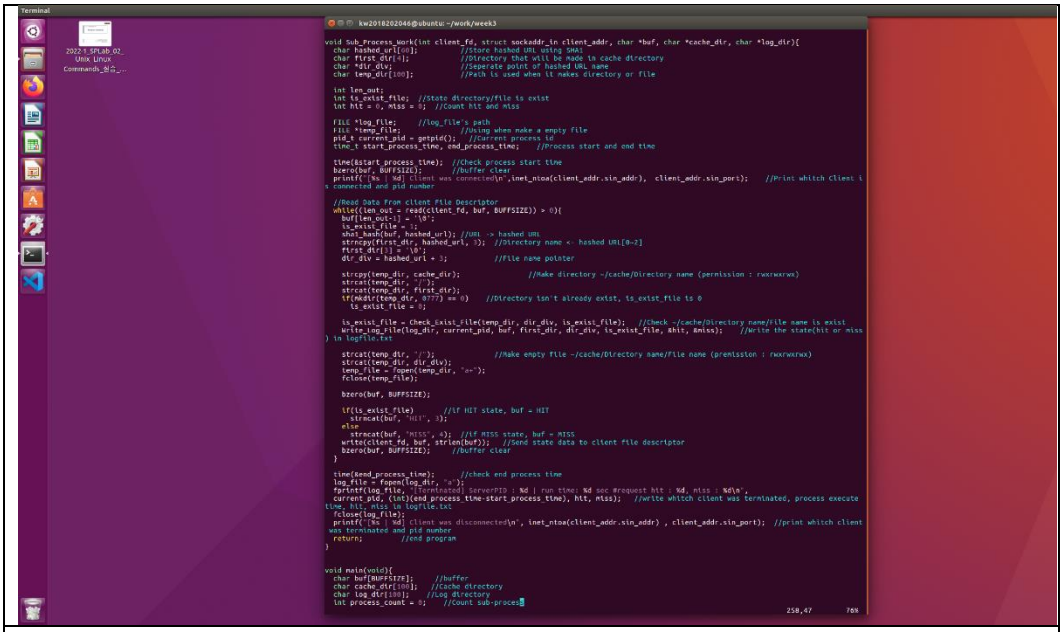
```
kw2018202046@ubuntu: ~/work/week3

#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.



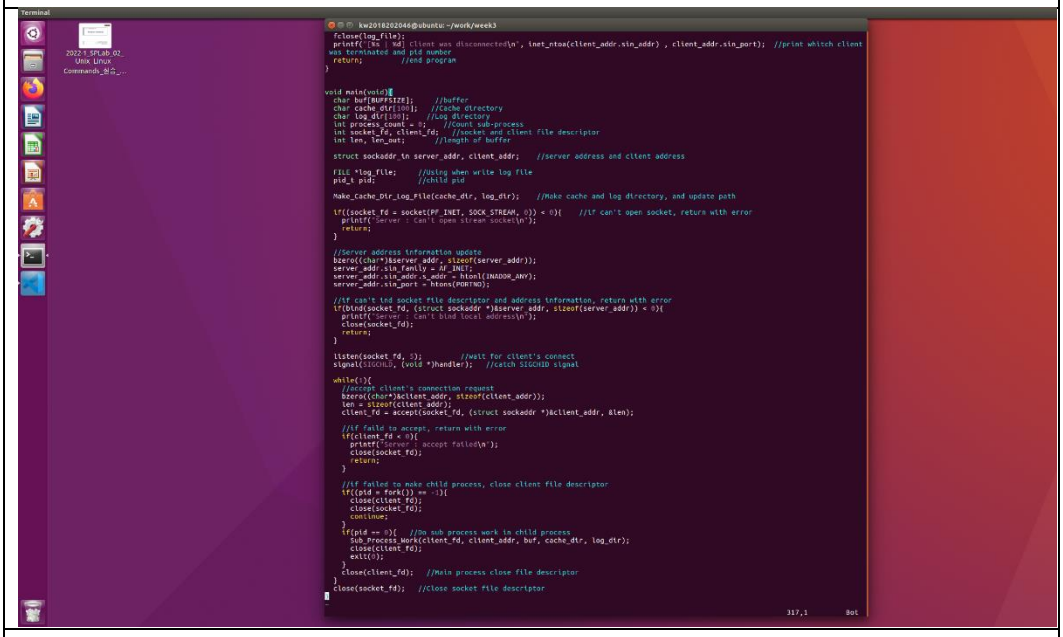
```
kw2018202046@ubuntu: ~/work/week3
  return home;
}


////////////////////////////////////////////////////////////////////////////////
// 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.



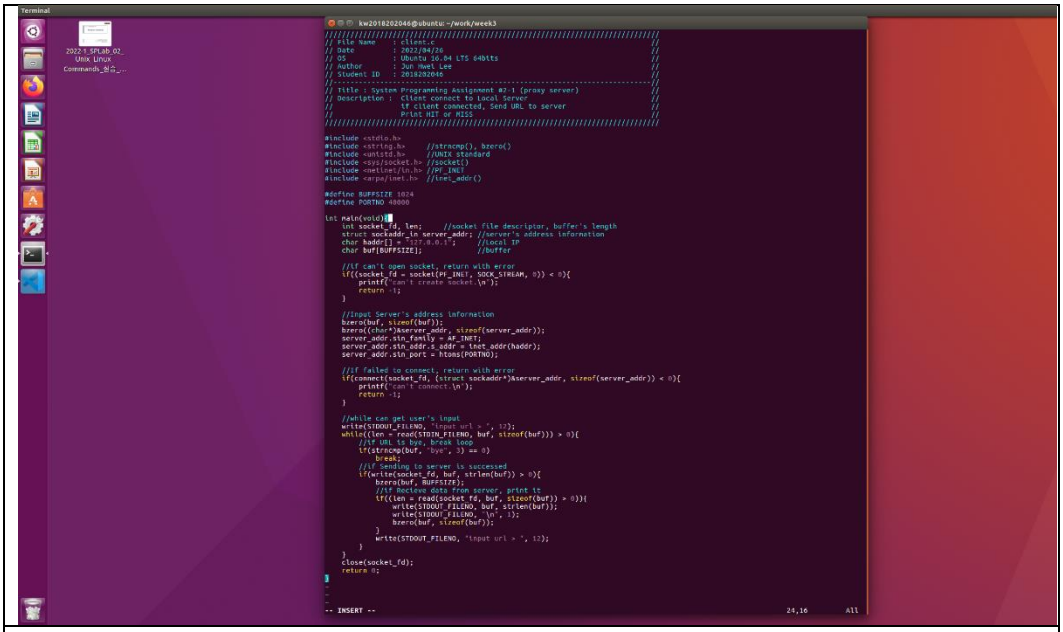
```
void Sub_Process_Work(int client_fd, struct sockaddr_in client_addr, char *buf, char *cache_dir, char *log_dir){
  char hashed_url[80];          //Store hashed URL using SHA1
  char first_dir[4];            //Directory that will be made to cache directory
  char *dir_div;                //Seperate point of hashed URL name
  char temp_dir[100];           //Path is used when it makes directory or file

  int len_out;
  int is_exist_file; //State directory/file is exist
  int hit = 0, miss = 0;  //Count hit and miss

  FILE *log_file;     //log_file's path
  FILE *temp_file;          //Using when make a empty file
  pid_t current_pid = getpid();   //Current process id
  time_t start_process_time, end_process_time;    //Process start and end time

  time(&start_process_time);  //Check process start time
  bzero(buf, BUFFSIZE);       //buffer clear
  printf("[%s | %d] Client was connected\n",inet_ntoa(client_addr.sin_addr),  client_addr.sin_port);    //Print whitch Client i
s connected and pid number

  //Read Data from client file Descriptor
  while((len_out = read(client_fd, buf, BUFFSIZE)) > 0){
    buf[len_out-1] = '\0';
    is_exist_file = 1;
    sha1_hash(buf, hashed_url); //URL -> hashed URL
    strncpy(first_dir, hashed_url, 3);  //Directory name <- hashed URL[0~2]
    first_dir[3] = '\0';
    dir_div = hashed_url + 3;           //File name pointer

    strcpy(temp_dir, cache_dir);              //Make directory ~/cache/Directory name (permission : rwxrwxrwx)
    strcat(temp_dir, "/");
    strcat(temp_dir, first_dir);
    if(mkdir(temp_dir, 0777) == 0)    //Directory isn't already exist, is_exist_file is 0
      is_exist_file = 0;

    is_exist_file = Check_Exist_File(temp_dir, dir_div, is_exist_file);   //Check ~/cache/Directory name/file name is exist
    Write_Log_File(log_dir, current_pid, buf, first_dir, dir_div, is_exist_file, &hit, &miss);     //Write the state(hit or miss
) in logfile.txt

    strcat(temp_dir, "/");              //Make empty file ~/cache/Directory name/File name (premission : rwxrwxrwx)
    strcat(temp_dir, dir_div);
    temp_file = fopen(temp_dir, "a+");
    fclose(temp_file);

    bzero(buf, BUFFSIZE);

    if(is_exist_file)           //if HIT state, buf = HIT
      strncat(buf, "HIT", 3);
    else
      strncat(buf, "MISS", 4);  //if MISS state, buf = MISS
    write(client_fd, buf, strlen(buf));  //Send state data to client file descriptor
    bzero(buf, BUFFSIZE);       //buffer clear
  }

  time(&end_process_time);      //check end process time
  log_file = fopen(log_dir, "a");
  fprintf(log_file, "Terminated| ServerPID : %d | run time: %d sec #request hit : %d, miss : %d\n",
  current_pid, (int)(end_process_time-start_process_time), hit, miss);  //write whitch client was terminated, process execute
time, hit, miss in logfile.txt
  fclose(log_file);
  printf("[%s | %d] Client was disconnected\n", inet_ntoa(client_addr.sin_addr) , client_addr.sin_port);  //print whitch client
was terminated and pid number
  return;         //end program
}


void main(void){
  char buf[BUFFSIZE];     //buffer
  char cache_dir[100];    //Cache directory
  char log_dir[100];      //Log directory
  int process_count = 0;    //Count sub-proces
                                                                                    258,47        78%
```

해당 함수는 기존의 Sub_Process_Work를 일부 변경하여 연결된 포트와 주소까지 출력할 수 있도록 작성되었다.

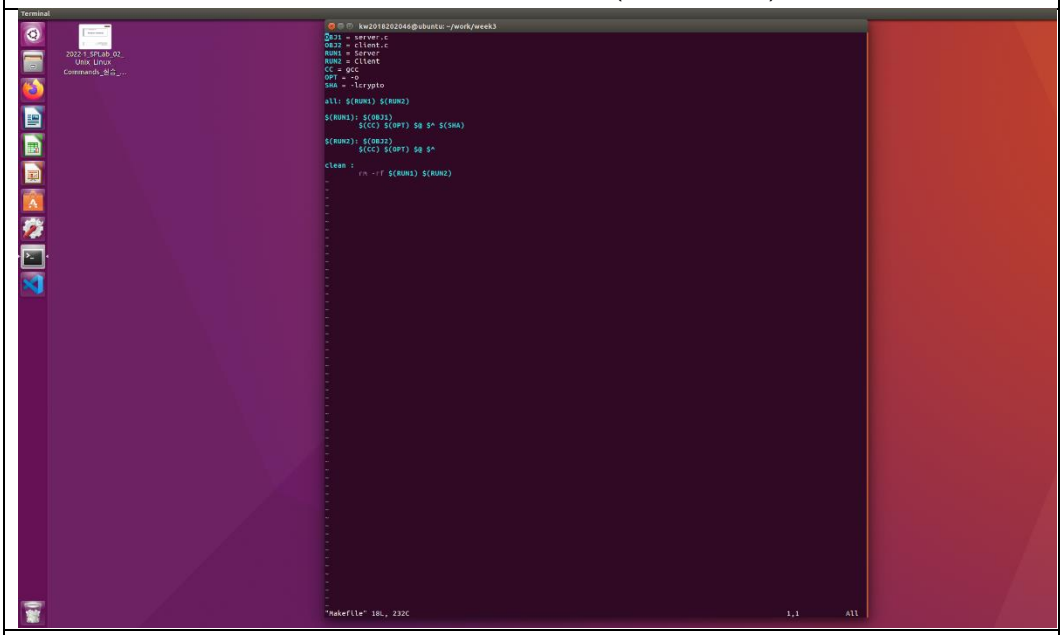


```
  fclose(log_file);
  printf("[%s | %d] Client was disconnected\n", inet_ntoa(client_addr.sin_addr) , client_addr.sin_port);  //print whitch client
was terminated and pid number
  return;         //end program
}


void main(void){
  char buf[BUFFSIZE];     //buffer
  char cache_dir[100];    //Cache directory
  char log_dir[100];      //Log directory
  int process_count = 0;    //Count sub-process
  int socket_fd, client_fd;   //socket and client file descriptor
  int len, len_out;           //length of buffer

  struct sockaddr_in server_addr, client_addr;    //server address and client address

  FILE *log_file;     //Using when write log file
  pid_t pid;          //child pid

  Make_Cache_Dir_Log_File(cache_dir, log_dir);    //Make cache and log directory, and update path

  if((socket_fd = socket(PF_INET, SOCK_STREAM, 0)) < 0){    //if can't open socket, return with error
    printf("Server : Can't open stream socket\n");
    return;
  }

  //Server address information update
  bzero((char*)&server_addr, sizeof(server_addr));
  server_addr.sin_family = AF_INET;
  server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
  server_addr.sin_port = htons(PORTNO);

  //if can't bid socket file descriptor and address information, return with error
  if(bind(socket_fd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0){
    printf("Server : Can't bind local address\n");
    close(socket_fd);
    return;
  }

  listen(socket_fd, 5);           //wait for client's connect
  signal(SIGCHLD, (void *)handler);   //catch SIGCHID signal

  while(1){
    //accept client's connection request
    bzero((char*)&client_addr, sizeof(client_addr));
    len = sizeof(client_addr);
    client_fd = accept(socket_fd, (struct sockaddr *)&client_addr, &len);

    //if faild to accept, return with error
    if(client_fd < 0){
      printf("Server : accept failed\n");
      close(socket_fd);
      return;
    }

    //if failed to make child process, close client file descriptor
    if((pid = fork()) == -1){
      close(client_fd);
      close(socket_fd);
      continue;
    }
    if(pid == 0){   //Do sub process work in child process
      Sub_Process_Work(client_fd, client_addr, buf, cache_dir, log_dir);
      close(client_fd);
      exit(0);
    }
    close(client_fd);   //Main process close file descriptor
  }
close(socket_fd);   //Close socket file descriptor
                                                                                    317,1         Bot
```

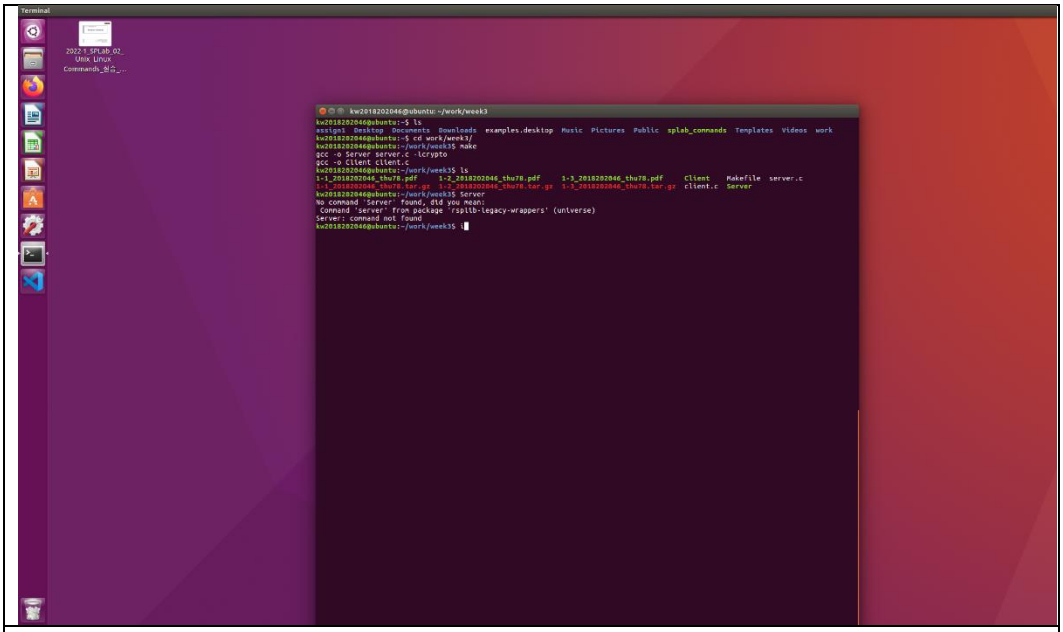
해당 그림은 server의 메인함수다. 해당 함수에서는 socket을 열고 binding을 통해 server의 정보를 묶는다. 그리고 listen을 통해 연결을 받는다. 연결을 accept할 경우 child process를 생성하여 Sub_Process_Work 작업을 수행한다.



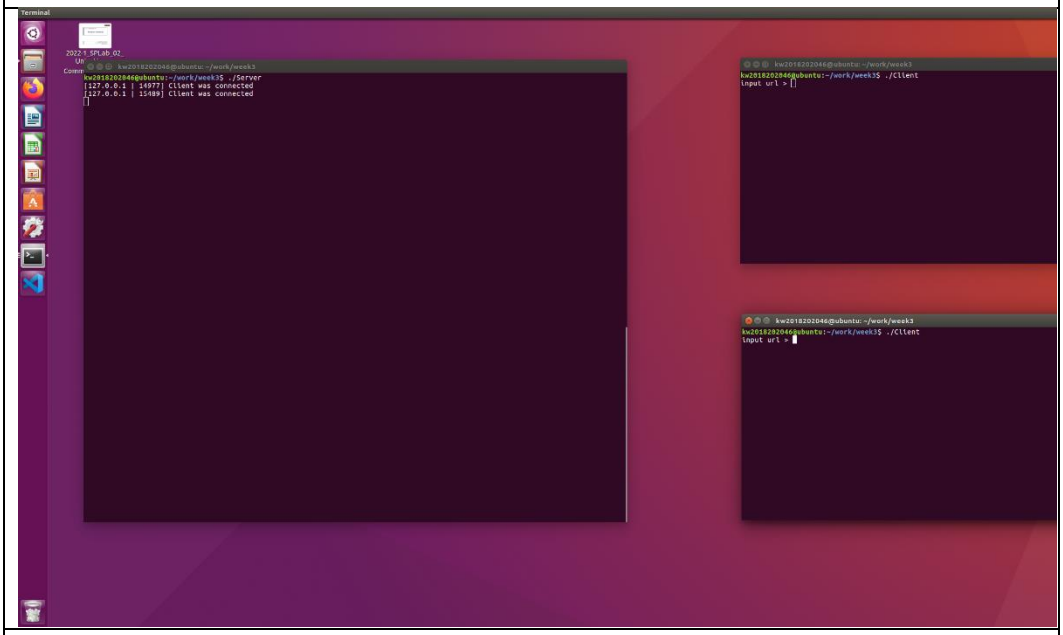
해당 그림은 client.c에서의 메인함수다. 해당 함수에서도 동일하게 socket을 열고 준비된 주소로 server address를 update한다. 그 후 connect 함수를 통해 server에 연결한다. 그리고 사용자의 입력이 있는 동안에는 다음 작업을 반복한다. 입력은 buffer에 저장되어 있고 bye 명령을 입력받은 경우 loop를 탈출한다. 만약 데이터를 server로 보내고 답장을 받을 경우 받은 답장(HIT or MISS)를 출력한다.



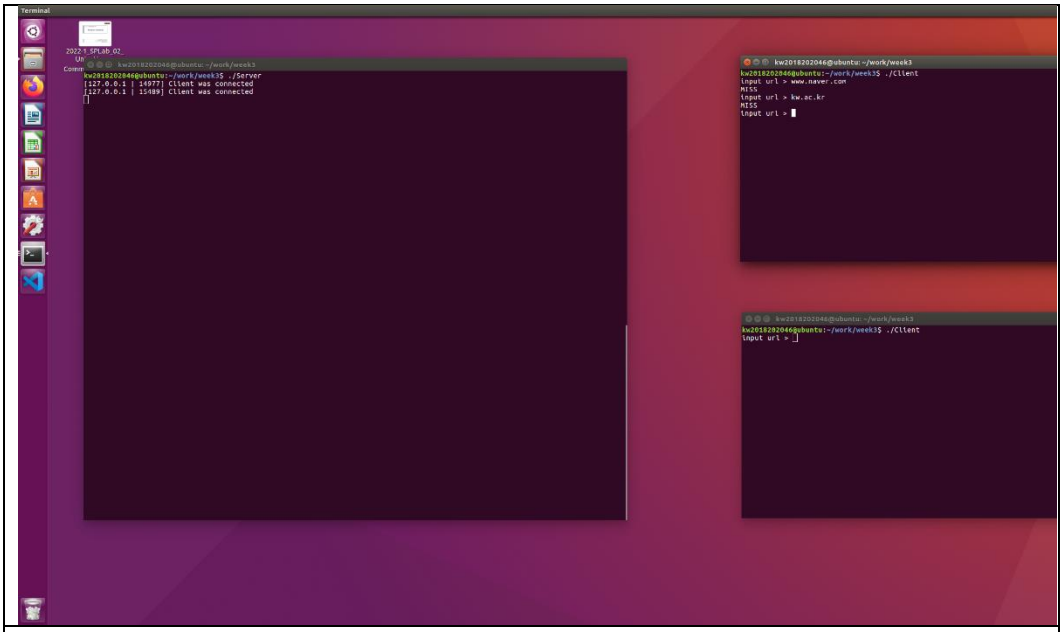
해당 파일은 변경된 Makefile이다. server.c와 client.c에 대한 실행파일을 만드는 작업을 수행한다.



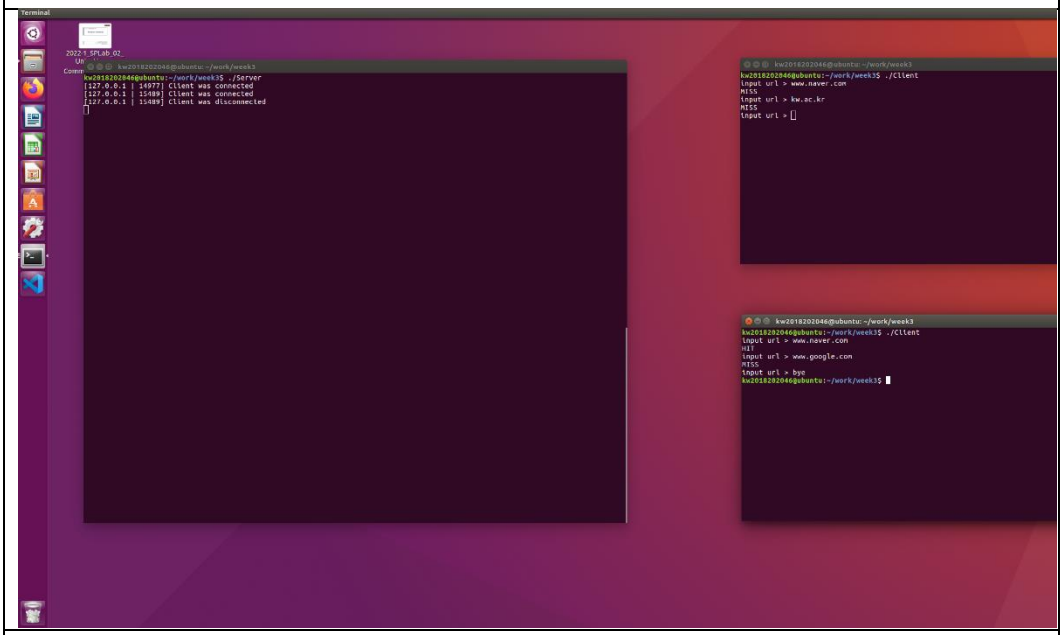
해당 그림은 기존의 cache와 logfile이 없는 상태에서 시작한다. make 명령을 통해 실행파일이 정상 생성되는 것을 볼 수 있다.



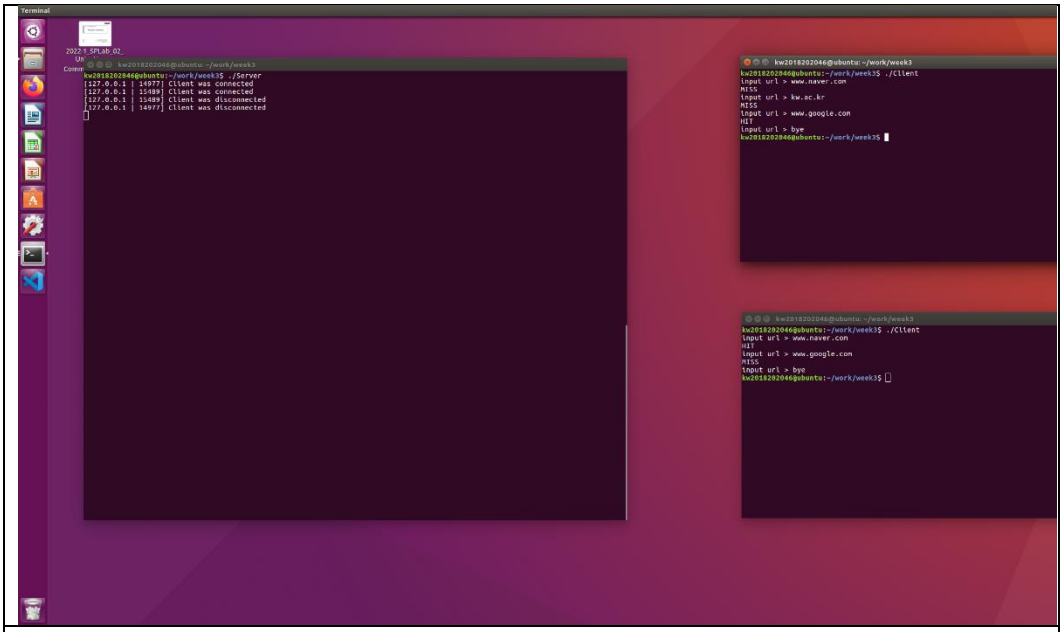
server를 열고 2개의 client의 접속이 성공하였다.



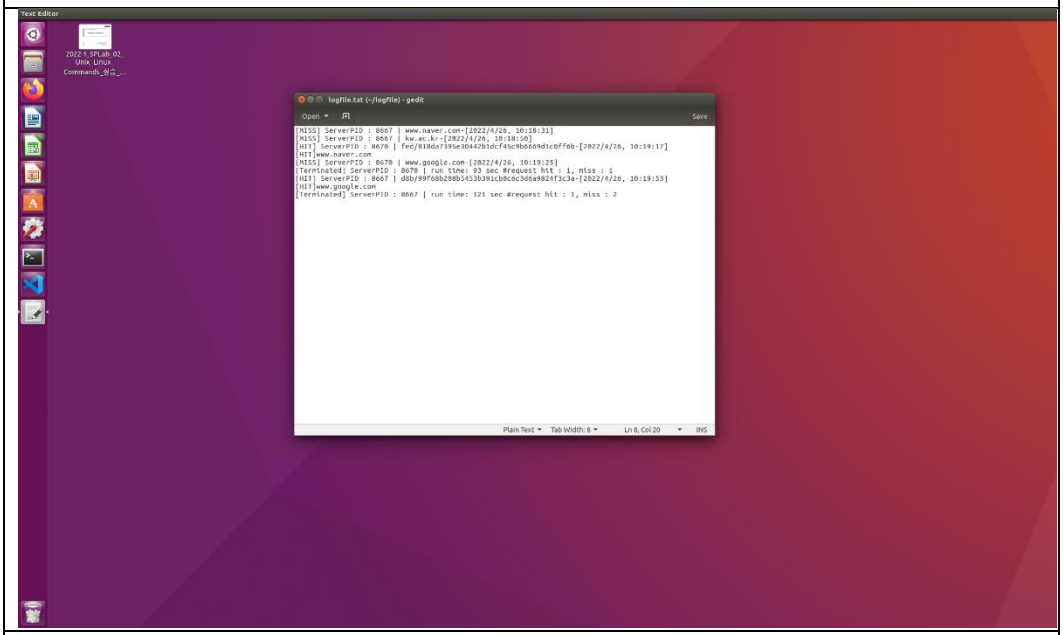
2번의 입력을 받았을 때 MISS state를 정확히 출력하였다.



다른 client에서 2번의 입력을 하였을 때 다른 client에서 중복된 입력이 HIT 상태로 정상 출력되고 새로운 입력은 MISS 상태로 정상 출력된다. 그리고 해당 client를 닫았을 때 server에 종료가 표시된다.



마지막으로 아직 열린 client에서 닫은 client에서 입력한 URL을 입력하였을 때 HIT 상태를 출력한다. 해당 client 또한 bye를 입력할 경우 정상적으로 종료되고 server에서 disconnected된 것이 표시된다.



해당 결과를 살펴보면 로그에서 각 client의 입력들이 시간에 따라 저장되어있는 것을 볼 수 있다. 또한 해당 client 종료시 해당 시간과 정보 또한 정상 저장되었는 것을 볼 수 있다. 이를 통해 해당 과제를 성공적으로 수행한 것을 알 수 있다.

5. 고찰

해당 과제를 통해서 sever와 client 사이에서 어떻게 message를 주고받는지 알 수 있는 과제였다. socket을 열고 해당 socket을 통해 file descritor를 생성하여 소통하는 과정에서

쓰이는 함수들을 공부할 수 있는 과제였다. 또한 signal 함수를 통해 child process의 termination을 받을 수 있다는 사실을 알게되었다. 그리고 기존에 과제에서 시간과 관련하여 실수한 부분을 발견하여 이를 고치면서 더욱 더 집중해서 검증을 수행해야겠다는 생각을 하였다.

6. Reference

강의 자료만을 참고