시스템 프로그래밍 실습 3-1 과제

이름 : 이준휘

학번 : 2018202046

교수 : 최상호 교수님

강의 시간 : 화

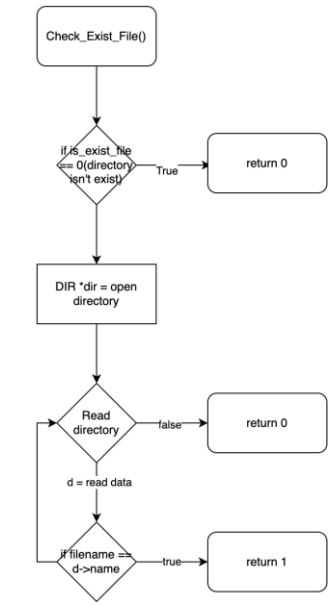
실습 분반 : 목 7, 8

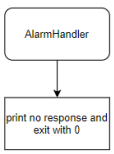
1. Introduction

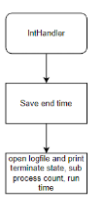
해당 과제는 기존에 2-4에서 만든 결과에서 추가적으로 덧붙어서 만들어진다. 해당 과제에서는 Semaphore를 사용하게 된다. Port Number에 맞는 Semaphore를 생성한 뒤 만약 cache 파일과 logfile에 접근하는 경우 Semaphore id를 통해 Critical zone에 접근하도록 한다. 해당 Critical zone에서 모든 작업을 마친 후에는 Critical zone에서 나오면서 자원을 다른 process에서 접근할 수 있도록 한다. 위의 작업들은 터미널창에 출력을 통해 표시하여 현재 어떤 process가 접근 중인지 알 수 있다.

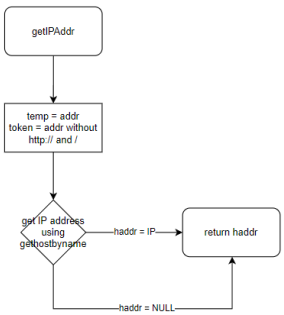
2. Flow Chart

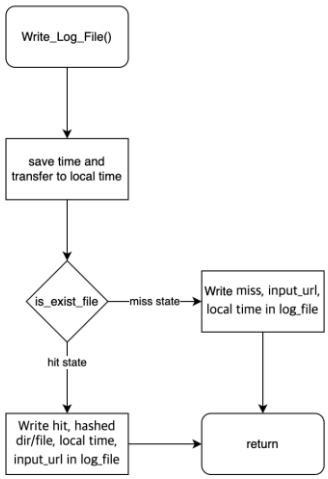
- proxy_cache.c

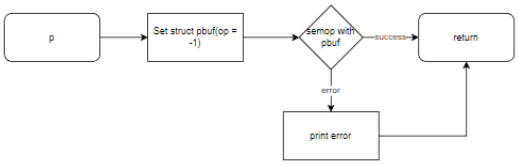


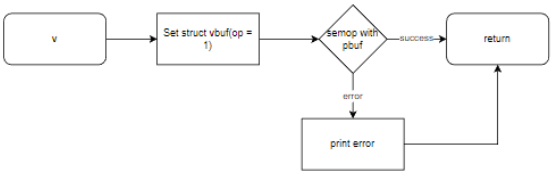


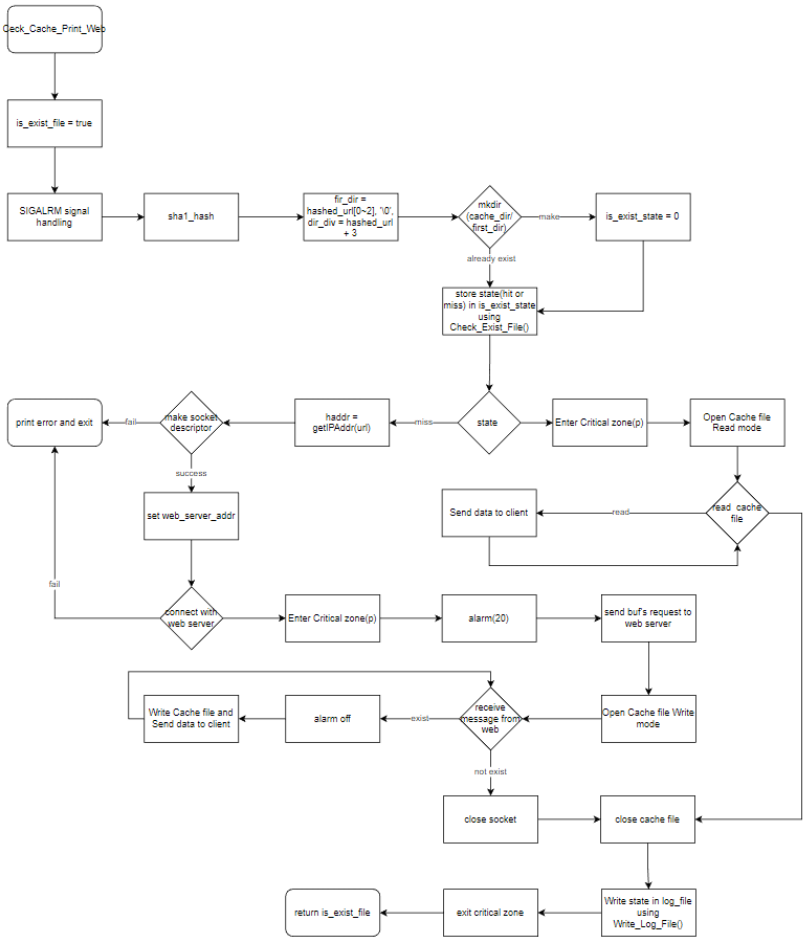


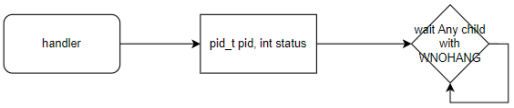

getIPAddr
temp = addr
token = addr without
http:// and /
get IP address
using
gethostbyname
haddr = IP
return haddr
haddr = NULL

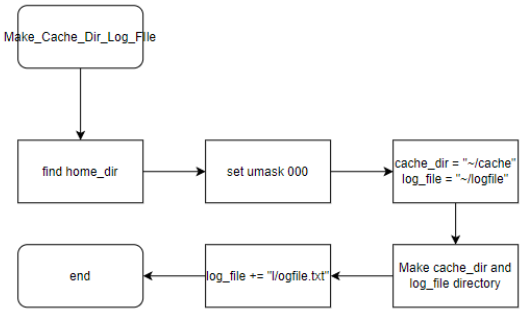

Write_Log_File()
save time and
transfer to local time
is_exist_file
miss state
Write miss, input_url,
local time in log_file
hit state
Write hit, hashed
dir/file, local time,
input_url in log_file
return

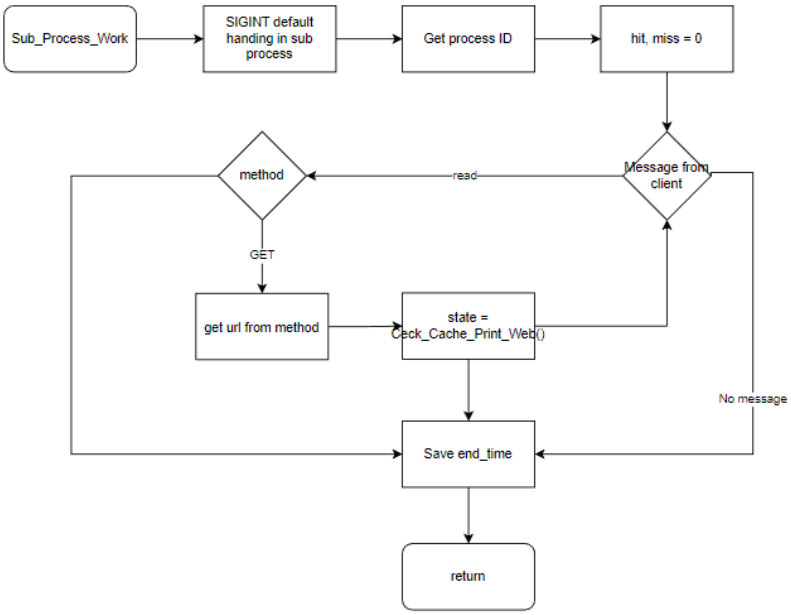

p
Set struct pbuf(op =
-1)
semop with
pbuf
success
return
error
print error

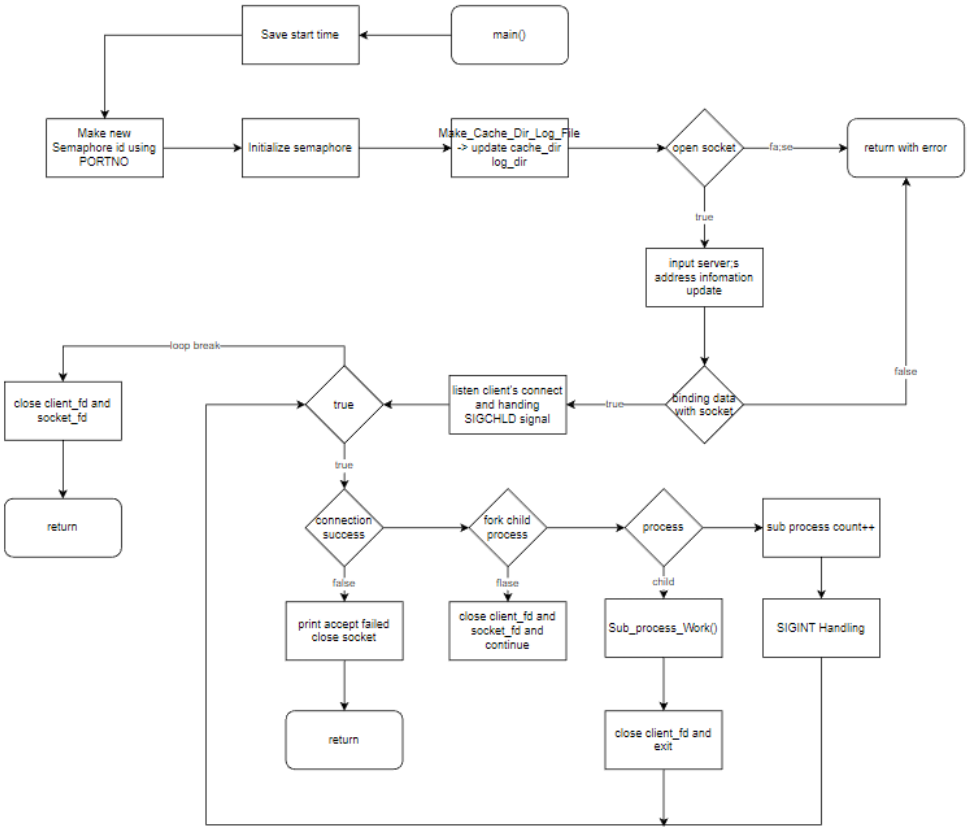

v
Set struct vbuf(op =
1)
semop with
pbuf
success
return
error
print error


Check_Cache_Print_Web
is_exist_file = true
SIGALRM signal handling
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()
state
miss
haddr = getIPAddr(url)
make socket descriptor
fail
print error and exit
success
set web_server_addr
connect with web server
fail
Enter Critical zone(p)
alarm(20)
send buf's request to web server
Open Cache file Write mode
receive message from web
exist
alarm off
Write Cache file and Send data to client
not exist
close socket
close cache file
Write state in log_file using Write_Log_File()
exit critical zone
return is_exist_file
Enter Critical zone(p)
Open Cache file Read mode
read_cache file
read
Send data to client


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
}

static void AlarmHandler(){
   print No Response and exit(0);
}

Static void IntHandler(){
   Save end time;
   Open logfile and write sub process count and run time;
   Close logfile and exit;
}

void p(int semid){
   struct sembuf pbuf;
   Set pbuf(op = -1);
   If(semop of semid failed)
      Print error;
}
void v(int semid){
   struct sembuf vbuf;
   Set pbuf(op = 1);
   If(semop of semid failed)
      Print error;
```

```
}

char *getIPAddr(char *addr){
    struct hostent* hent;
    char *haddr = NULL;
    char temp[BUFFSIZE] = addr;
    token = temp(delete http://, and tokenized with /;
    if (gethostbyname(token) is exist)
        haddr = inet_ntoa(hent's h_addr_list[0]);
    return haddr;
}

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_Print_Web(int client_fd, int semid, char *url, char *cache_dir, char *log_file,
char *buf int current_pid, int len, int *hit, int *miss){
    char[60] hashed_url;
    char[4] first_dir;
    char *dir_div;
    char[100] temp_dir;
    char *haddr[BUFFSIZE;
```

```
int h_socket_fd, cache_fd, h_len;
int is_exist_file = 1;

struct sockaddr_in web_server_addr;

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;
temp_dir = ~/cache/first_dir/dir_div;
SIGALRM signal handling.

if(state == miss){
    haddr = getIPAddr from url;
    if (make socket fd failed)
        print error and exit();
        setting web_server_addr using haddr;
        if (connection with web failed)
            print error and exit();
        write buf's request to web server;
        alarm 20 sec;
        Enter Critical zone(p with semid);
        open temp_dir with write mode;
```

```
            while(receive response){
                alarm off;
                write response at cache file;
                write response to client_fd;
                h_buf clear;
            }
            close h_socket_fd;
        }
        else{
            Enter Critical zone(p with semid);
            open temp_dir with read mode;
            while read data is exist in cache file{
                write data to client_fd;
                h_buf clear;
            }
        }
    close fd;
    return is_exist_file;
    Write state, url, dir/file name, time in logfile.txt;
    Exit Critical zone(v with semid);
    }

    void Sub_Process_Work(int client_fd, struct sock_addr, int semid, char *buf, char *char_dir,
FILE *log_file){
        char temp[BUFFSIZE] = { 0 };
        char method[BUFFSIZE] = { 0 };
```

```
    char url[BUFFSIZE] = { 0 };
    char h_buf[BUFFSIZE];
    char* haddr;
    char *token = NULL;
    int len, h_socket_fd;
    int state, hit = 0, miss = 0;

    struct sockaddr web_server_addr;
    pid_t current_pid = Current process ID;
    Set sub process's SIGINT handler default;
    while read data from client_fd to buf is exist{
        method = Request message's method;
        if(method == GET){
            url = Request message's url
            state = Ceck_Cache's state(Hit or Miss);
        }
    }
}

main(void){
    if Make new Semid using PORTNO failed, return;
    if Semid initialize failed, return;
    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;
    if parent process{
        sub process count++;
        SIGINT Handling;
    }
    close client file descriptor;
  }
   close socket file descriptor;
}
```

4. 예상 시나리오

A. Process 종료 후에 다른 프로세스에서 semid에 접근하는 경우

PID# A is waiting for the semaphore

PID# A is in the critical zone

PID# A exited the critical zone

PID# B is waiting for the semaphore

PID# B is in the critical zone

PID# B exited the critical zone

B. Process 종료 전 다른 프로세스에서 semid에 접근하는 경우

PID# A is waiting for the semaphore

PID# A is in the critical zone

PID# B is waiting for the semaphore

PID# A exited the critical zone

PID# B is in the critical zone

PID# B exited the critical zone

C. Process 입장 전 다른 프로세스에서 semid에 접근하는 경우

PID# A is waiting for the semaphore

PID# B is waiting for the semaphore

PID# A is in the critical zone

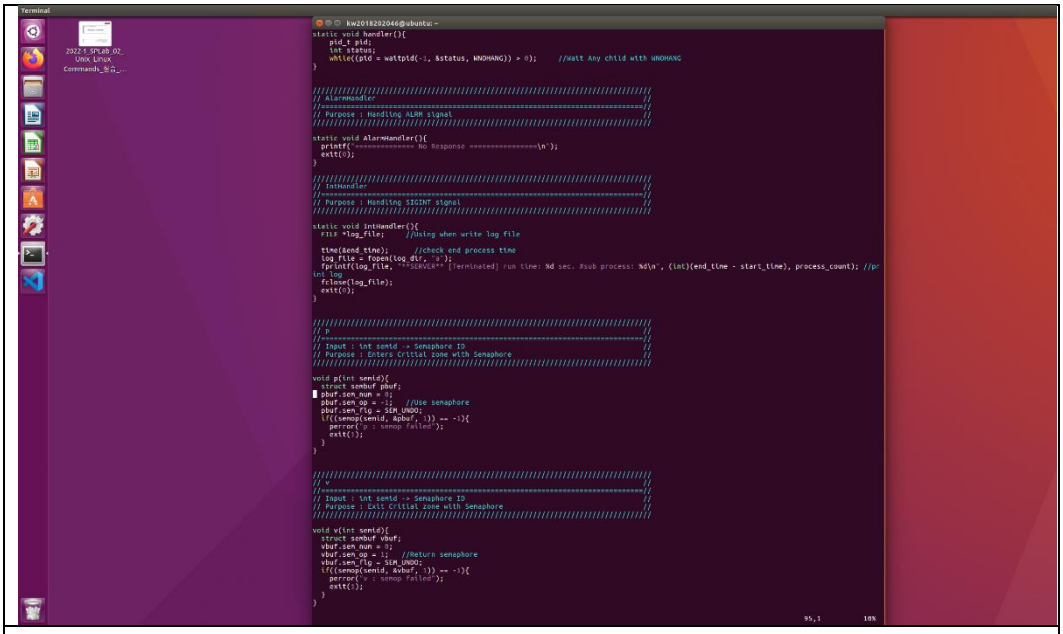
PID# A exited the critical zone

PID# B is in the critical zone

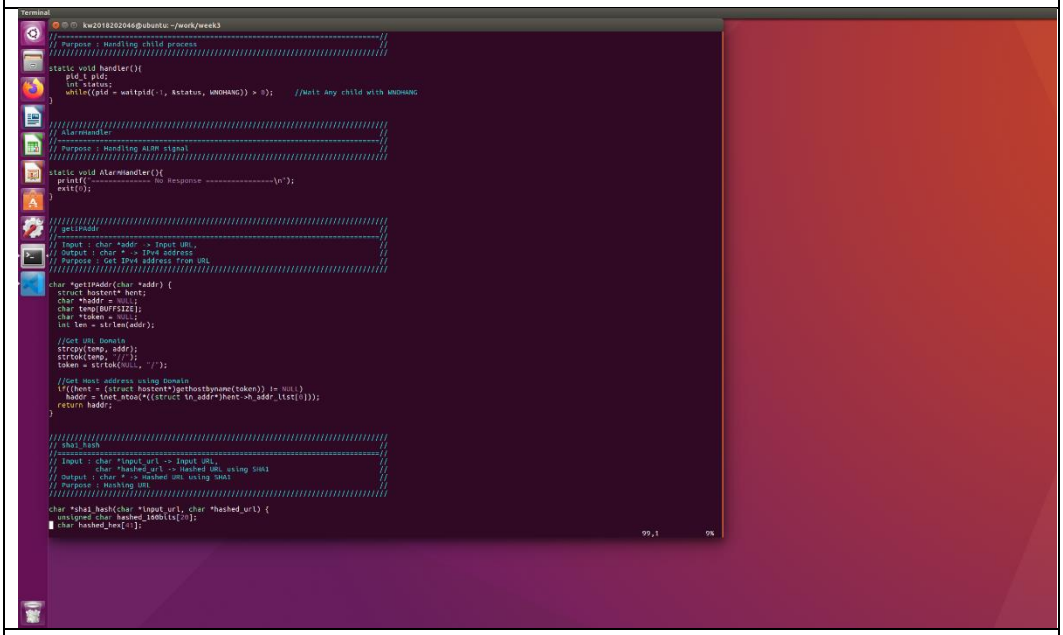
PID# B exited the critical zone

위와 같이 3가지의 시나리오가 예상된다.

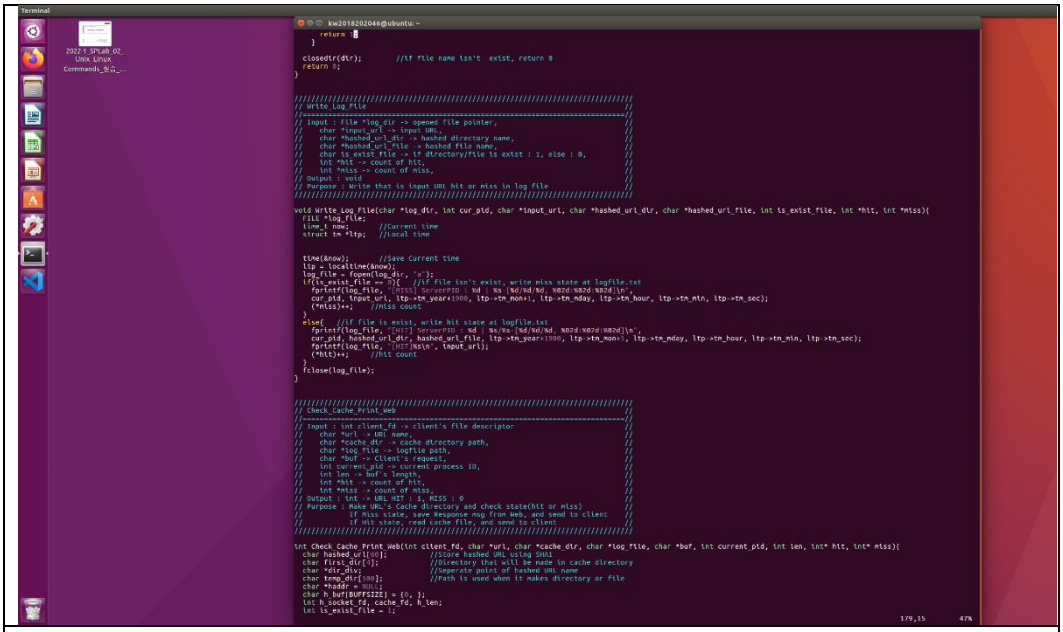
5. 결과 화면



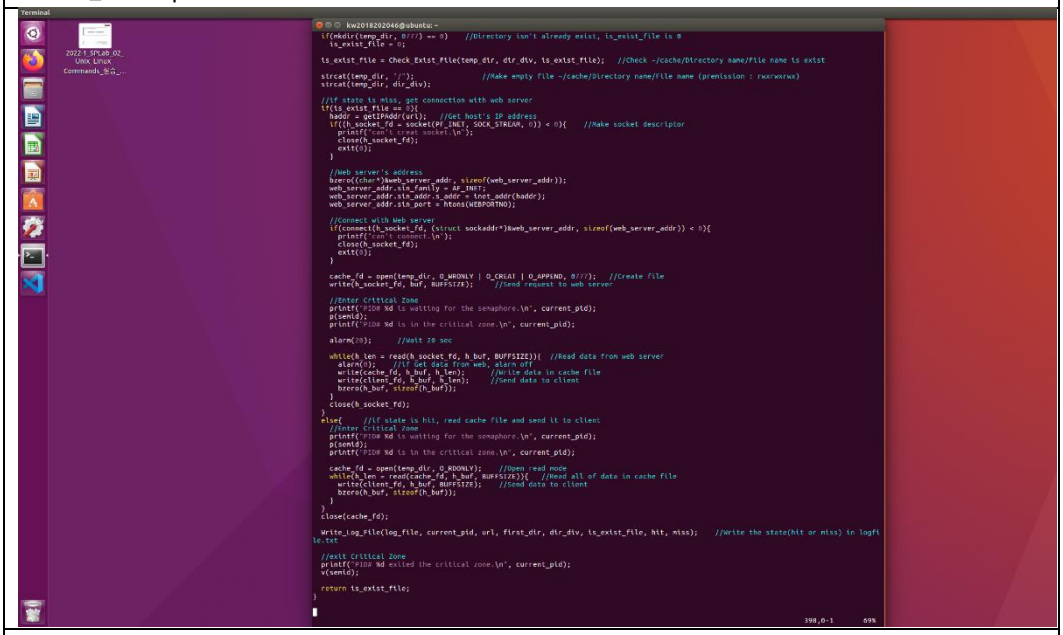
Signal 함수에서 handling을 위한 함수들과 이번에 semaphore를 위해 추가된 함수가 존재한다. Handler는 SIGCHLD, AlarmHandler는 SIGALRM, IntHandler는 SIGINT를 위한 handler이다. IntHandler는 전역변수로 선언된 sub_process_count와 log_file의 주소를 이용하여 종료 로그를 작성 후 terminate한다. P, v 함수는 각각 Critical zone 진입, 탈출 시 사용하는 함수로 segid에 해당하는 val을 컨트롤함으로써 semaphore를 조정한다.



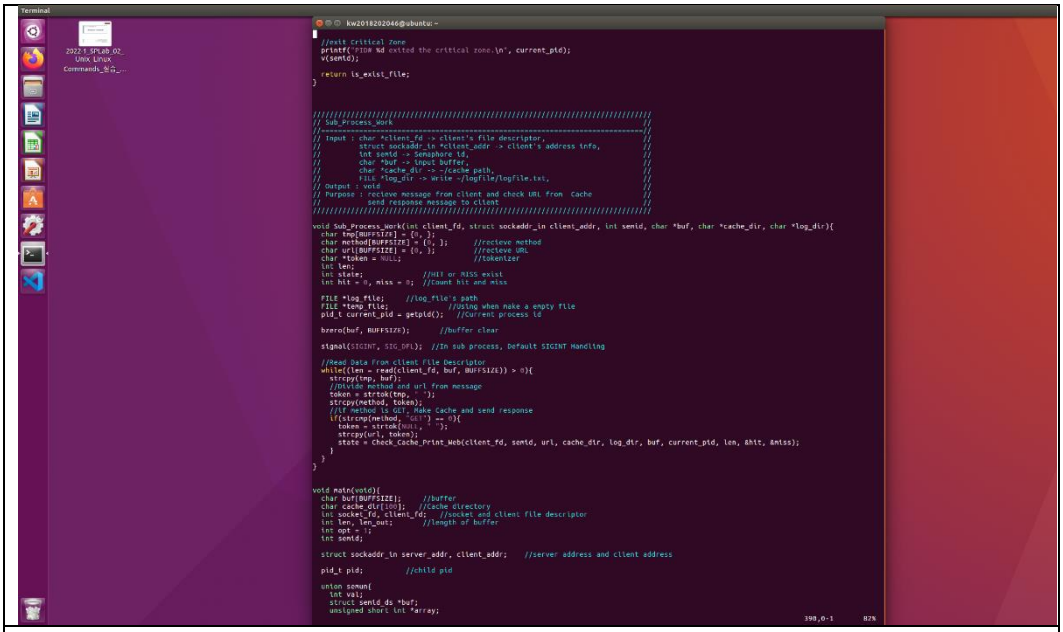
AlarmHandler()에서는 SIGALRM 에러가 왔을 경우 No response를 출력하고 exit()을 동작시킴으로써 error를 handling한다. getIPAddr()함수에서는 url을 입력으로 받으며 url에서 http://을 tokenize하고 /까지를 tokenize하여 순수한 주소를 가져온다. 이를 가지고 gethostbyname()함수를 통해 host의 정보를 가져와 IP를 추출한다.



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

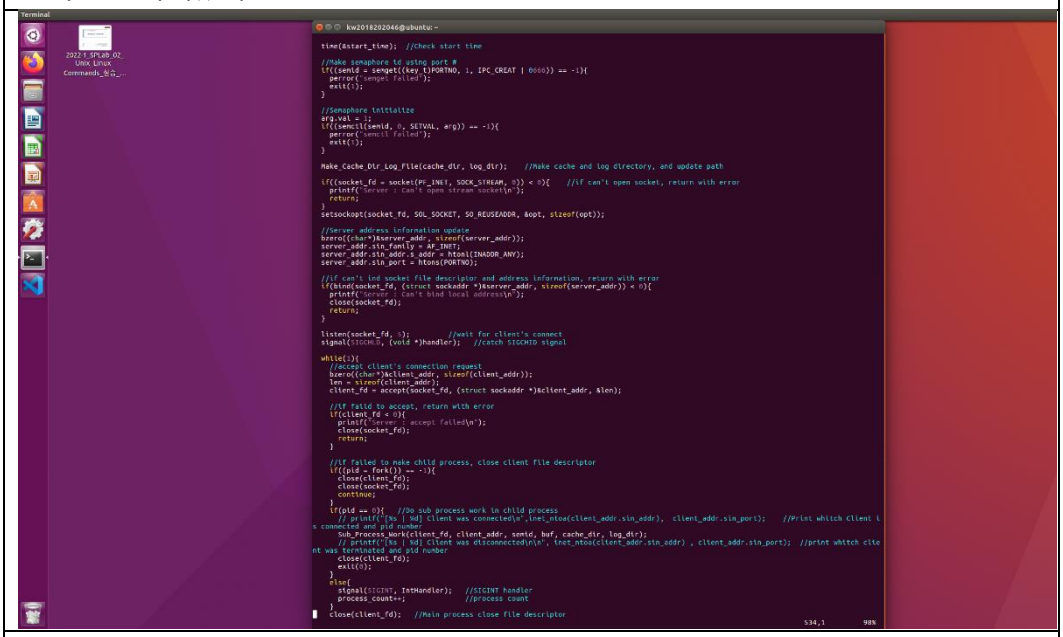


해당 함수는 이전의 함수에서 일부가 수정되었다. Semid를 인자로 추가로 받는다. 이전의 부분에서 cache 파일 읽기, 쓰기, logfile 쓰기 부분이 p와 v 사이의 critical zone에 위치하도록 설정하여 하나의 프로세스만 접근하도록 조정하였다. 또한 진입 여부를 판단하기 위한 출력도 추가되었다.



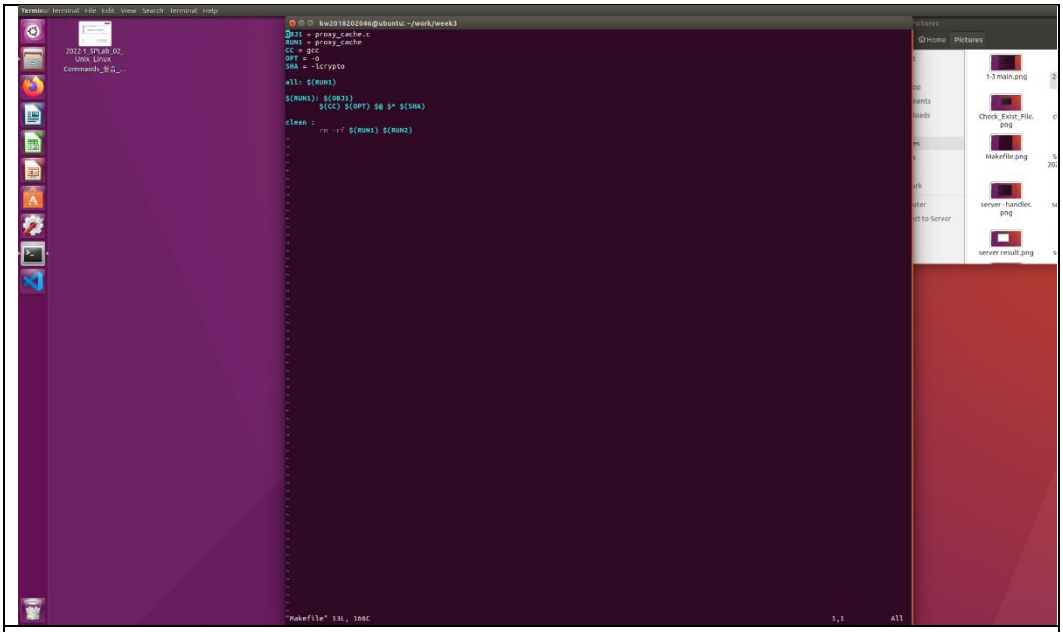
```
//exit Critical Zone
printf("PID# %d exited the critical zone.\n", current_pid);
v(semid);

return is_exist_file;
}


////////////////////////////////////////////////////////////////////////////////
// Sub_Process_Work                                                           //
//----------------------------------------------------------------------------//
// Input : char *client_fd -> client's file descriptor,                       //
//         struct sockaddr_in *client_addr -> client's address info,          //
//         int semid -> Semaphore id,                                         //
//         char *buf -> input buffer,                                         //
//         char *cache_dir -> ~/cache path,                                   //
//         FILE *log_dir -> Write ~/logfile/logfile.txt,                      //
// Output : void                                                              //
// Purpose : recieve message from client and check URL from  Cache            //
//           send response message to client                                  //
////////////////////////////////////////////////////////////////////////////////

void Sub_Process_Work(int client_fd, struct sockaddr_in client_addr, int semid, char *buf, char *cache_dir, char *log_dir){
  char tmp[BUFFSIZE] = {0, };
  char method[BUFFSIZE] = {0, };      //recieve method
  char url[BUFFSIZE] = {0, };         //recieve URL
  char *token = NULL;                 //tokenizer
  int len;
  int state;              //HIT or MISS 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

  bzero(buf, BUFFSIZE);       //buffer clear

  signal(SIGINT, SIG_DFL);  //In sub process, Default SIGINT Handling

  //Read Data from client File Descriptor
  while((len = read(client_fd, buf, BUFFSIZE)) > 0){
    strcpy(tmp, buf);
    //Divide method and url from message
    token = strtok(tmp, " ");
    strcpy(method, token);
    //if method is GET, Make Cache and send response
    if(!strcmp(method, "GET") == 0){
      token = strtok(NULL, " ");
      strcpy(url, token);
      state = Check_Cache_Print_Web(client_fd, semid, url, cache_dir, log_dir, buf, current_pid, len, &hit, &miss);
    }
  }
}


void main(void){
  char buf[BUFFSIZE];     //buffer
  char cache_dir[100];    //Cache directory
  int socket_fd, client_fd;   //socket and client file descriptor
  int len, len_out;       //length of buffer
  int opt = 1;
  int semid;

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

  pid_t pid;          //child pid

  union semun{
    int val;
    struct semid_ds *buf;
    unsigned short int *array;
                                                                  398,0-1       82%
```

해당 함수는 2-4에서 만든 내용에서 기존 터미널 창에서의 출력을 없애서 과제의 스팩을 만족시켰다.

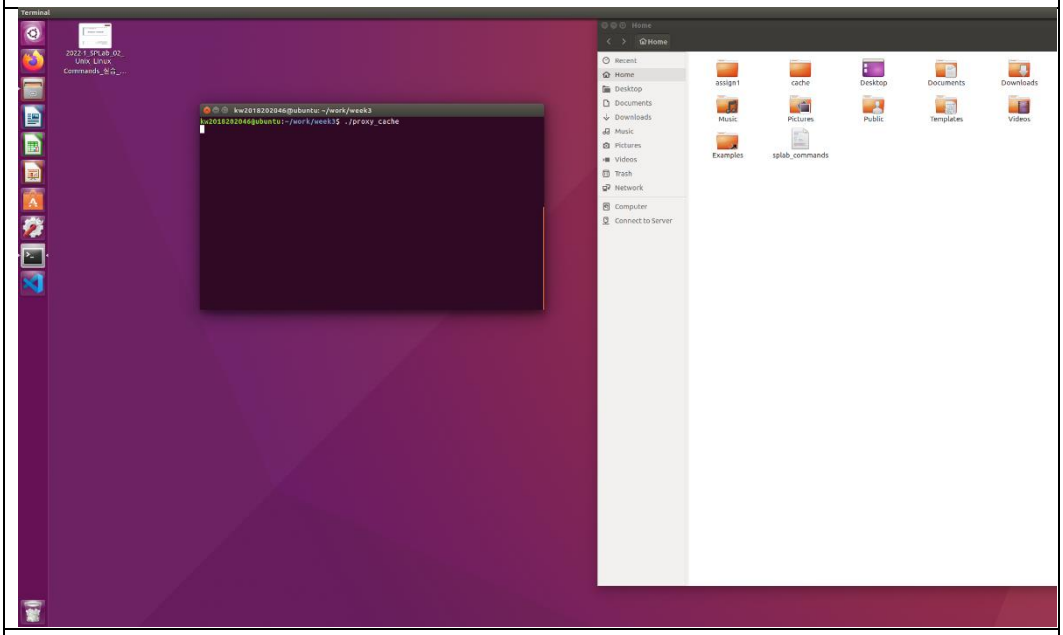


```
  time(&start_time);  //Check start time

  //Make semaphore to using port #
  if((semid = semget((key_t)PORTNO, 1, IPC_CREAT | 0666)) == -1){
    perror("semget failed");
    exit(1);
  }

  //Semaphore initialize
  arg.val = 1;
  if((semctl(semid, 0, SETVAL, arg)) == -1){
    perror("semctl failed");
    exit(1);
  }

  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;
  }
  setsockopt(socket_fd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));

  //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 ind 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
      // 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
      Sub_Process_Work(client_fd, client_addr, semid, buf, cache_dir, log_dir);
      // printf("%s | %d] Client was disconnected\n\n", inet_ntoa(client_addr.sin_addr) , client_addr.sin_port);  //print whitch clie
nt was terminated and pid number
      close(client_fd);
      exit(0);
    }
    else{
      signal(SIGINT, IntHandler);   //SIGINT handler
      process_count++;              //process count
    }
  close(client_fd);   //Main process close file descriptor
                                                                  534,1         98%
```

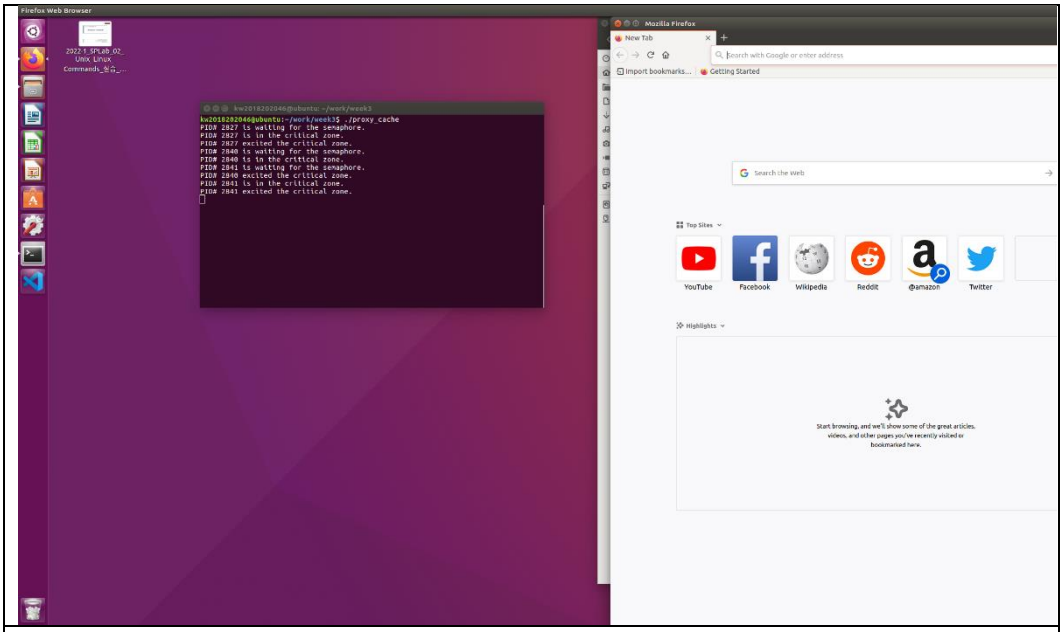
해당 그림은 메인함수다. 기존과 다른 점으로는 semaphore를 생성하고 초기화하는 부분이 추가되었으며 sub_process_work의 인자로 semid가 추가된 부분이 있다.



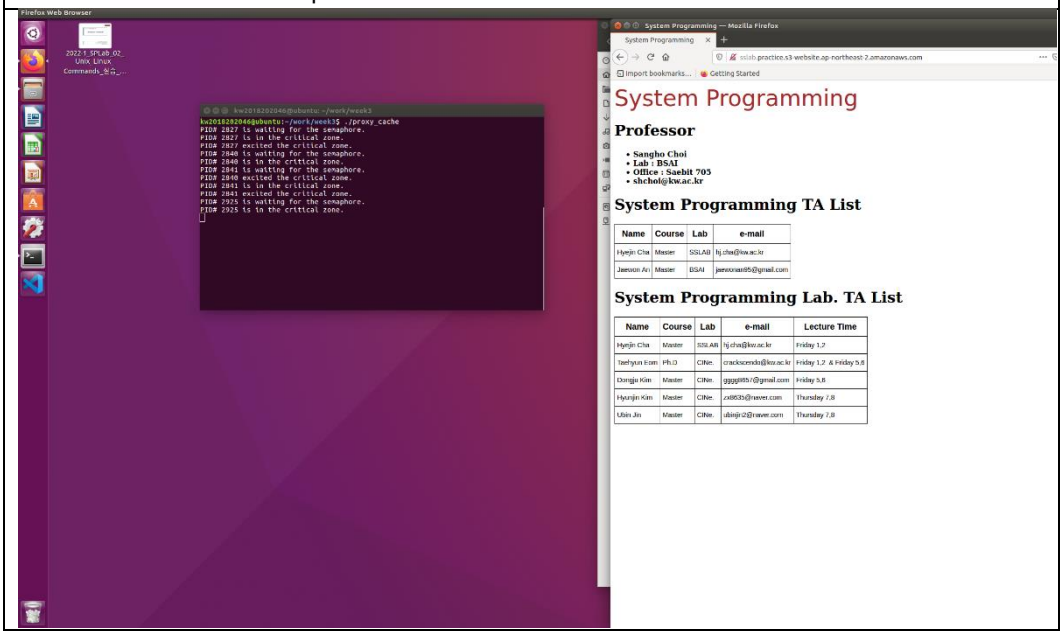
해당 파일은 Makefile이다. proxy_cache.c의 파일을 proxy_cache 실행파일로 만드는 역할을 수행한다.

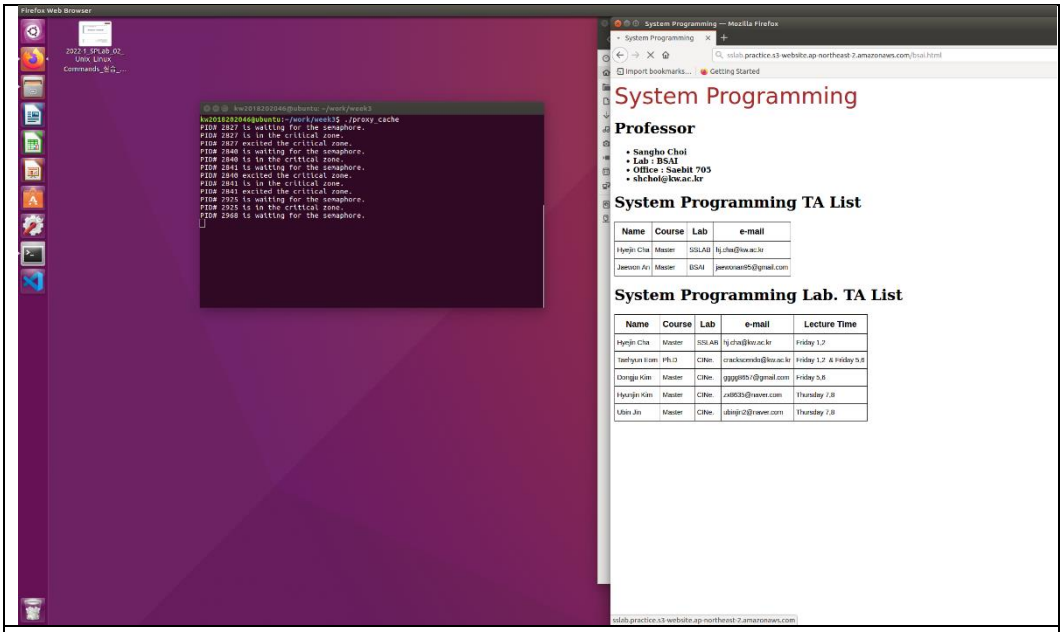


해당 그림은 기존의 cache와 logfile이 없는 상태에서 시작한다.

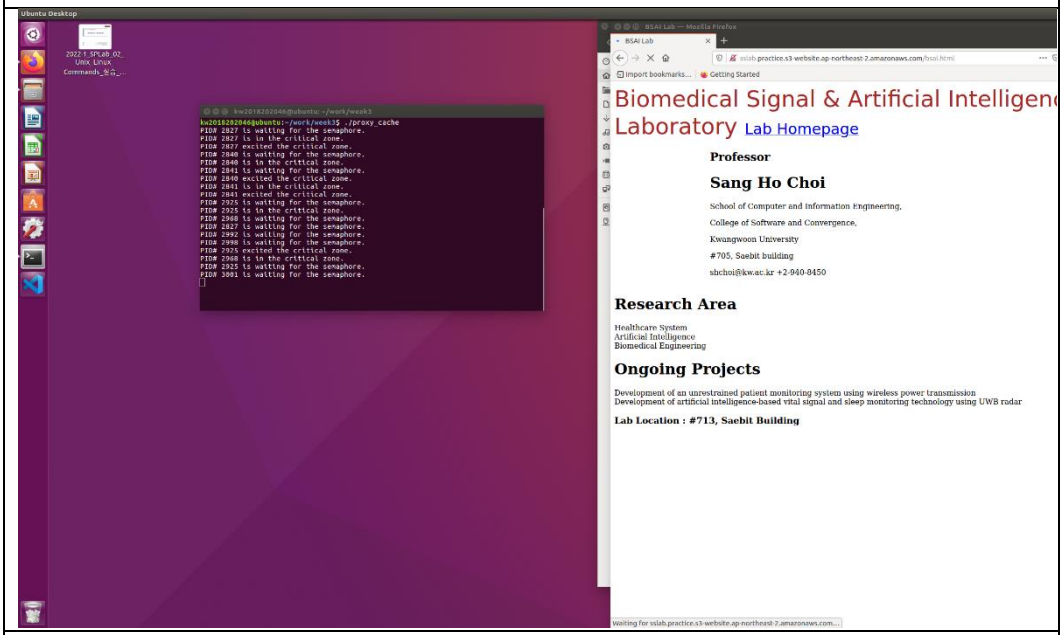


다음과 같이 우선 semaphore의 출력은 확인할 수 있다.

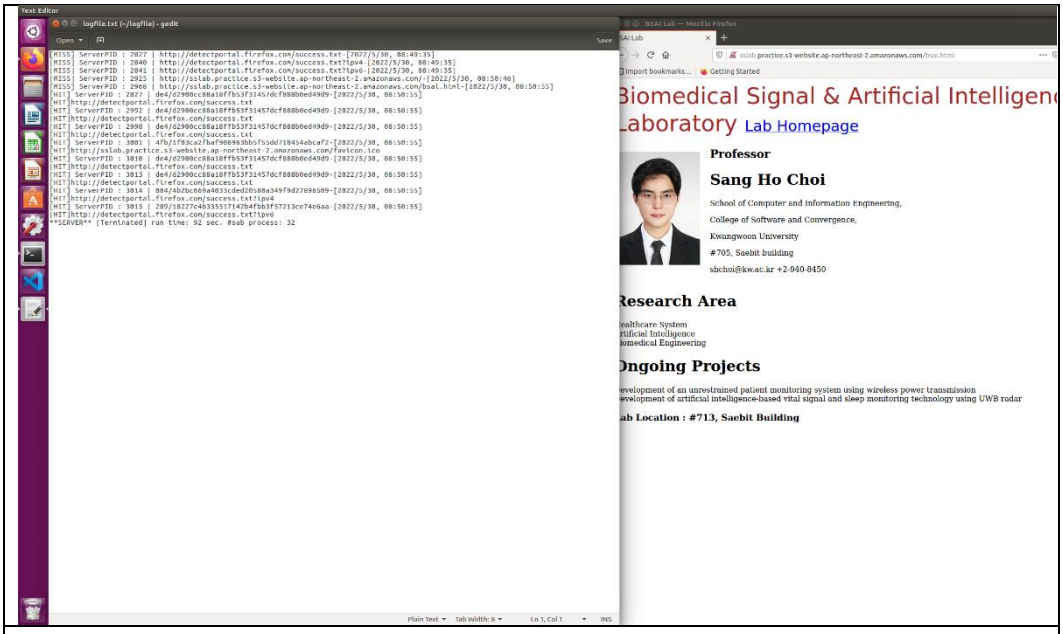




다음 그림은 일부로 sleep 5초를 걸어 critical zone에서 오래 머무를 때 다른 process에서 접근 시 다른 프로세스는 기다리도록 하여 하나의 프로세스만 접근하는 모습을 확인할 수 있다.



다음의 모습 또한 하나의 프로세스만의 접근을 허용하고 있다. 이는 이전의 예상한 시나리오와 동일한 패턴의 모습으로 semaphore가 정상적으로 작동하는 것을 알 수 있다.



해당 결과를 보면 Miss와 hit의 상태를 알 수 있으며 server 종료 시에 동작 시간과 sub process의 실행 개수까지 출력하는 것을 볼 수 있다. 이를 통해 해당 과제가 성공적으로 완수된 것을 확인할 수 있다.

6. 고찰

해당 과제를 통해서 Semaphore라는 생소한 개념에 대하여 공부할 수 있는 과제였다. Semaphore를 통해 특정 id값에 접근하는 프로세스의 개수를 제한하고, 조절할 수 있다는 사실을 알 수 있던 과제였다. 또한 이를 실제적으로 semget, semop, setctrl 등의 함수를 사용하여 구현함으로써 되새김질할 시간을 가질 수 있었다. 그리고 이전 과제에서 실수로 잊고 구현하지 못하였던 sigint control를 마저 만들면서 과제의 스팩을 다시 확인하는 습관을 가지도록 다짐하였다.

7. Reference

강의 자료만을 참고