시스템 프로그래밍 실습 1-3 과제

이름 : 이준휘

학번 : 2018202046

교수 : 최상호 교수님

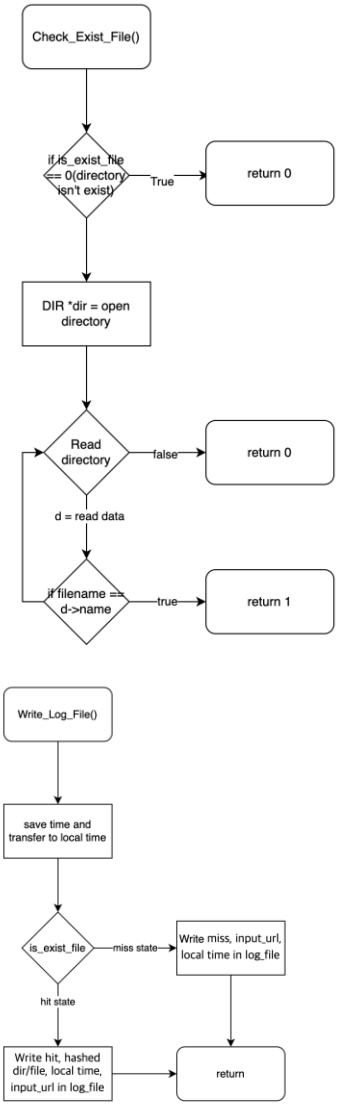
강의 시간 : 화

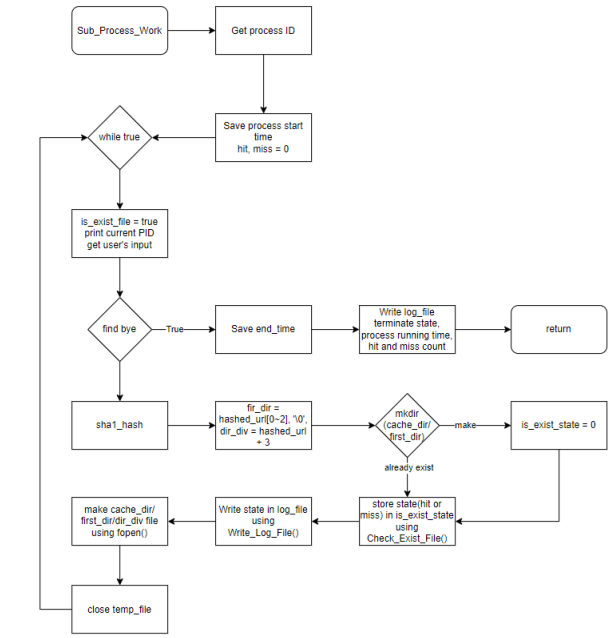
실습 분반 : 목 7, 8

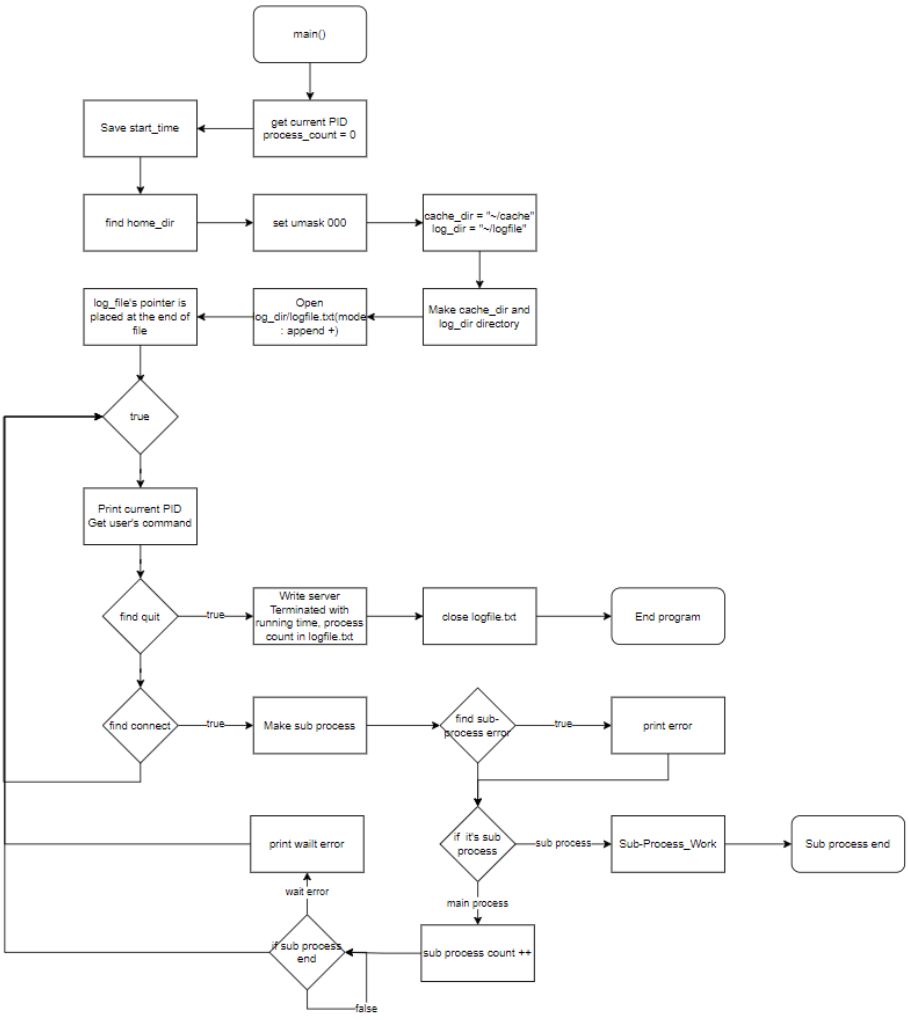
1. Introduction

해당 과제는 1-2 과제에 덧붙어서 진행된다. 프로그램 실행 시 현재 command를 입력받는다. connect를 입력을 받을 시 sub-process를 생성하여 이전에 1-2에서 만든 URL 입력 동작을 수행한다. quit를 입력할 경우 메인 process는 종료된다. terminal 상에는 입력을 받을 때 현재 process ID가 출력되어야 하고, 출력한 프로세스 종료 시 logfile.txt에 해당 기록을 남긴다.

2. Flow Chart




Sub_Process_Work
Get process ID
Save process start time
hit, miss = 0
while true
is_exist_file = true
print current PID
get user's input
find bye
True
Save end_time
Write log_file terminate state, process running time, hit and miss count
return
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


main()
get current PID
process_count = 0
Save start_time
find home_dir
set umask 000
cache_dir = "~/cache"
log_dir = "~/logfile"
Make cache_dir and log_dir directory
Open og_dir/logfile.txt(mode : append +)
log_file's pointer is placed at the end of file
true
Print current PID
Get user's command
find quit
true
Write server Terminated with running time, process count in logfile.txt
close logfile.txt
End program
find connect
true
Make sub process
find sub-process error
true
print error
if it's sub process
sub process
Sub-Process_Work
Sub process end
main process
print wailt error
wait error
if sub process end
sub process count ++
false

3. Pseudo Code

```
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(char *input, 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;
  while(true){
    print process id and get input;
    if(input is 'bye'){
      Save end process time;
      Write end – start process time, hit, miss count in logfile.txt;
      return;
    }
    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;
    }
}

main(void){
    char[100] input, home_dir, cache_dir, log_dir, temp_dir;
    int sub_process_count = 0;
    time_t start_time, end_time;
    FILE *log_file;
    pid_t pid, current_pid = Current Process ID;
    Save start time;

    home_dir = ~;
    cache_dir = ~/cache;
    log_dir = ~/logfile;
    set umask 000;
    make cache and log directory(permission = drwxrwxrwx);
    temp_dir = ~/logfile/logfile/txt;
    make log file(mode : a+);
    log_file's pointer is placed in the end of file;

    while(true){
        print current pid and Get command;
```

if(command == quit){

Save end time;

Write server terminated with running time, sub process count in log_file;

end program;

}

if(command = connect){

pid = make sub process;

if(failed to make sub process)

print error;

if(Current process is Sub process)

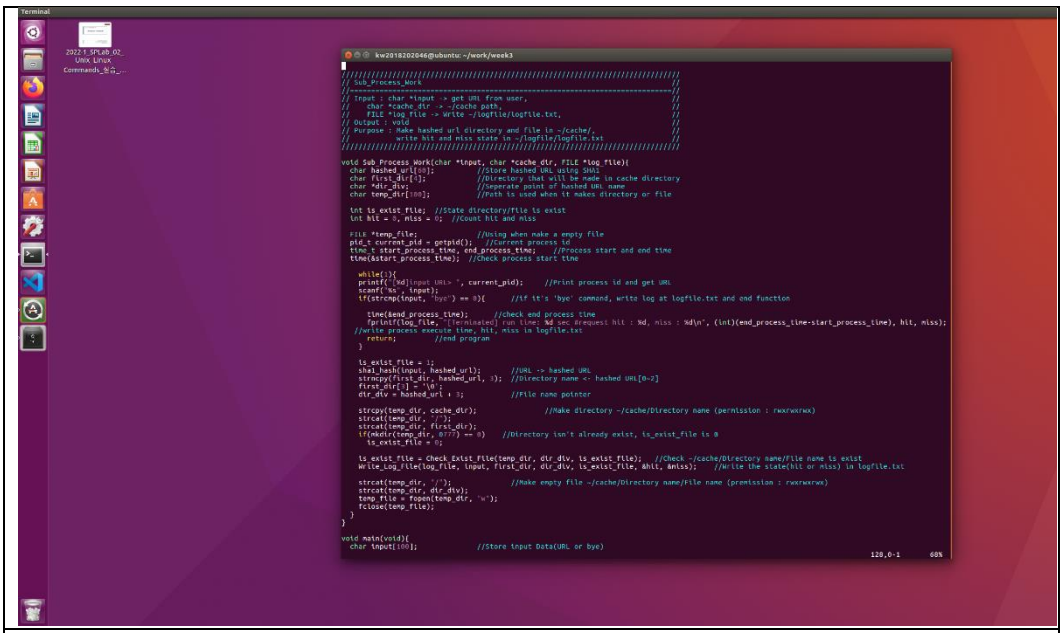
Do Sub_Process_Work and end process;

if(Current process is main process)

Process count++ and wait until Sub_Process end;

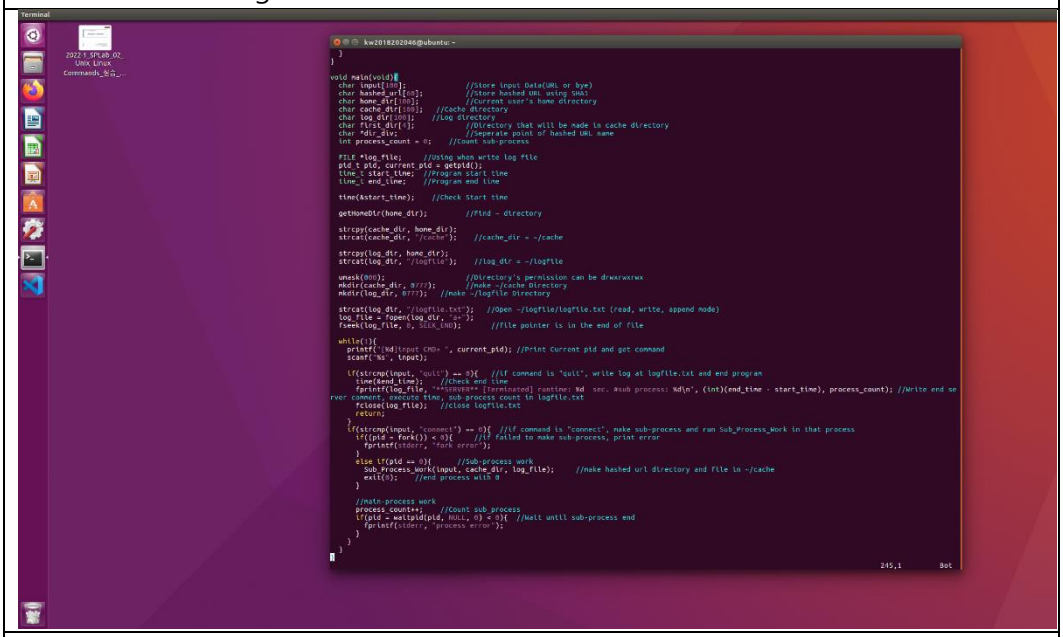
}

4. 결과 화면

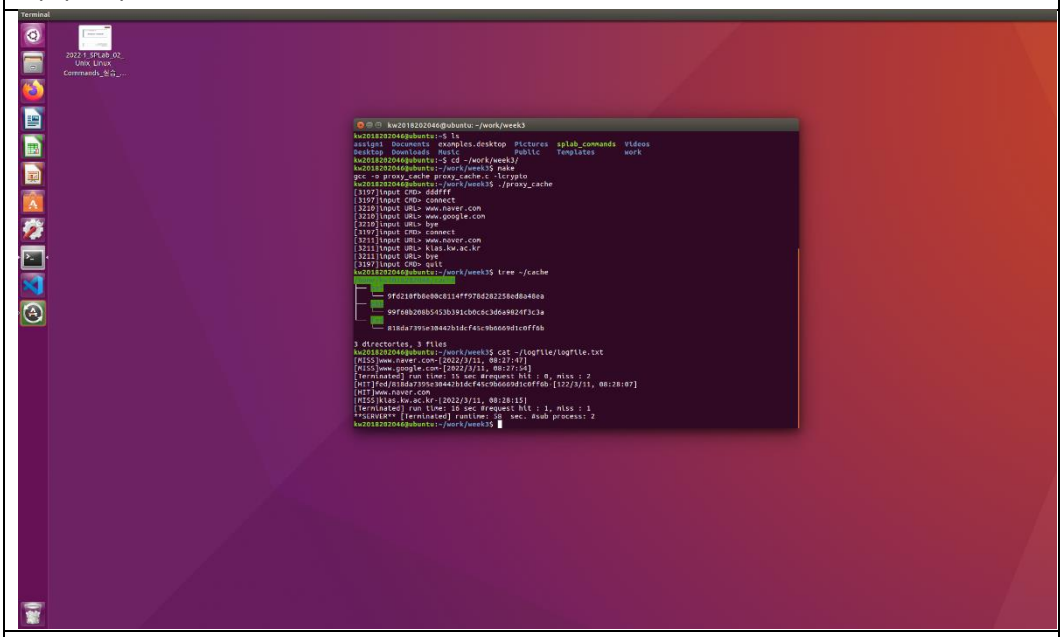


해당 사진은 Sub_Process_Work를 찍은 것이다. 해당 함수는 기존의 1-2에서 반복되는 부분을 함수로 바꾸고 현재의 PID를 추가적으로 출력하는 것으로 바뀌었으며,

또한 해당 함수가 종료할 경우 함수의 시간을 기록함으로써 sub process가 얼마동안 동작하였는지 logfile.txt에 기록한다.



위의 사진은 메인 함수를 표시한 것이다. 해당 함수는 logfile.txt를 여는 동작까지는 기존과 동일하지만 이후의 동작은 달라진다. 이후에 while문을 통해 반복적으로 현재의 PID를 출력하고 입력을 받는다. 만약 입력이 quit일 경우에는 종료 시간을 측정, logfile.txt에 Server가 terminate됨을 알리고 동작 시간, sub process 작동 횟수를 기록한다. 만약 connect를 입력을 받은 경우 fork() 함수를 통해 sub process를 만들고 sub process는 1-2의 작업에 해당하는 Sub_Process_Work() 함수를 수행한 뒤 마친다. main process는 process count를 1 증가시키고, sub process가 끝날 때까지 기다린다.



위의 사진은 해당 프로그램을 동작시킨 사진이다. 기존에 cache와 logfile이 없는 상태에서 해당 함수를 수행하였다. 처음 동작하였을 때 현재 프로세스와 CMD를

입력 받는 동작이 제대로 수행되었고 connect를 입력 받은 경우 Sub process를 발생시키기 때문에 pid가 다르게 출력된다. URL 2개를 입력받고 해당 프로세스를 종료시키면 다시 원래 pid가 출력된다. 입력의 경우 첫 번째 프로세스에서는 miss 2개, 두 번째 프로세스에서는 hit 1개, miss 1개를 입력하였다. logfile.txt를 살펴본 결과 2개의 프로세스가 위의 언급한 상태 그대로 기록되었고 Server를 종료할 경우 해당되는 상태 또한 제대로 기록된 것을 볼 수 있다. 이를 통해 해당 프로그램이 목적에 맞게 구현된 것을 확인하였다.

5. 고찰

해당 과제를 통해서 getpid()함수를 통해 현재의 프로세스에 대한 정보를 알 수 있다는 사실을 알게 되었다. 또한 fork()함수를 사용할 경우 2개의 return값을 통해 각각의 프로세스가 동작한다는 메커니즘을 이해할 수 있었다. wait() 함수나 waitpid()를 직접 사용해보면서 어떻게 하면 child process가 끝날 때까지 main process가 동작하지 않을지에 대해 고민해보았으며, child process가 main process의 자원을 복사하여 사용한다는 사실 또한 알게 되었다.

6. Reference

강의 자료만을 참고