1. 과제 개요

Reader-Writer Problem은 여러 Reader와 Writer들이 공유 자원을 동시에 접근하려고 하는 상황에서 발생하는 문제이다.

시그널(Signal)은 프로세스 및 사용자 간에 통신을 위해 운영체제에서 제공하는 도구이다. 시그널은 비동기적인 이벤트의 발생을 통지하기 위한 용도(EX. SIGKILL, SIGSEGV … )와 이벤트를 동기화시키기 위한 용도(EX. SIGALRM …)로 사용된다.

FIFO(Named PIPE)는 이름이 있는 파일을 통하여 통신을 하는 기법으로 프로세스 간에 통신(IPC)을 할 때 주로 사용된다.

디몬 프로세스(Daemon Process)는 멀티태스킹 운영체제 환경에서 백그라운드 프로세스처럼 사용자의 직접 제어 없이 여러 작업을 수행하는 프로그램이다. 디몬 프로세스의 부모 프로세스는 보통 init 프로세스이다. 부모 프로세스가 자식 프로세스를 생성하고 즉시 자신의 수행을 종료하면 init 프로세스가 고아가 된 자식 프로세스를 입양하면서 디몬 프로세스가 만들어진다.

디몬 프로세스를 생성하기 위해서는 다음과 같은 규칙을 만족해야 한다.

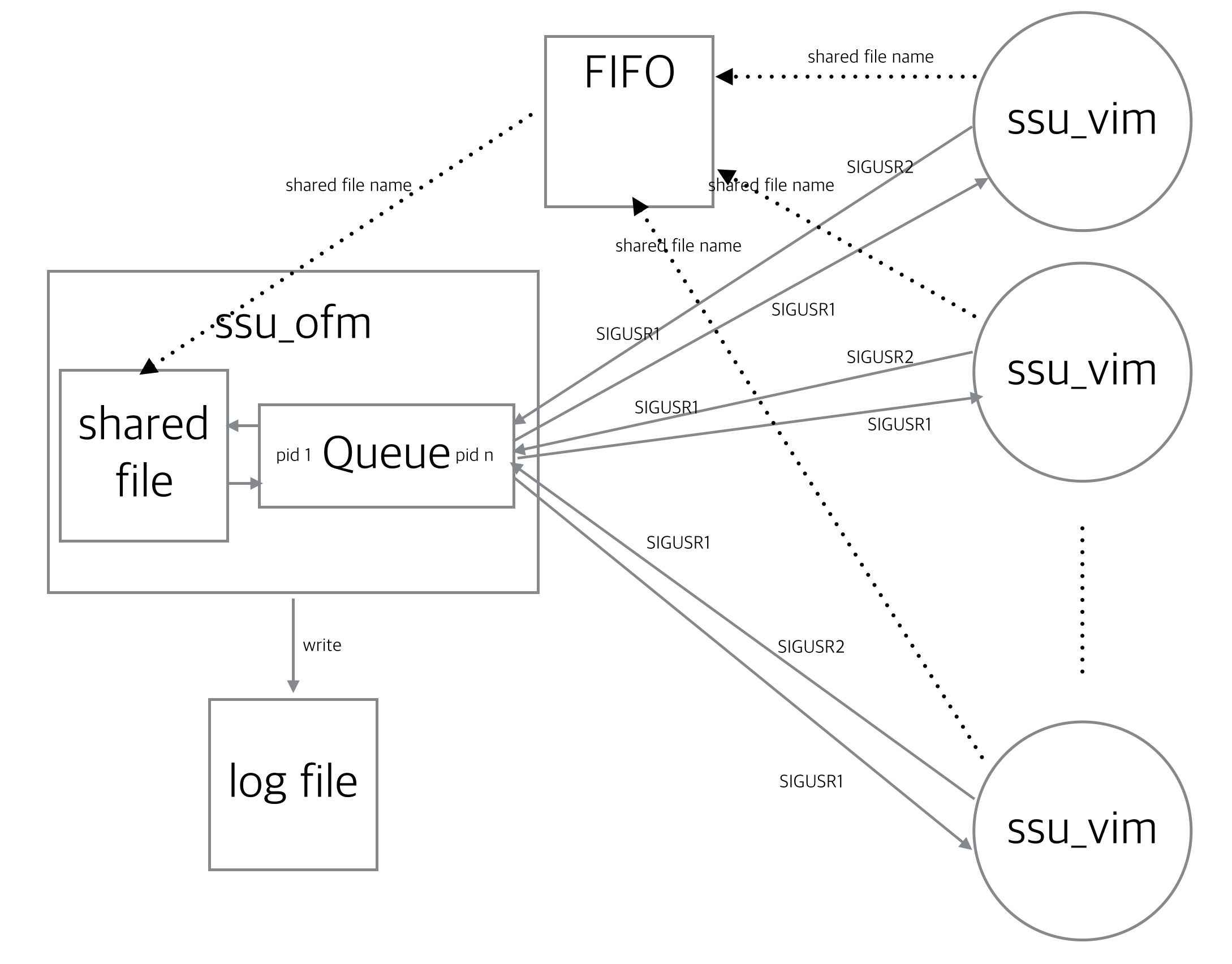
1. 백그라운드에서 수행되어야 한다.
2. 프로세스 그룹에서 탈퇴하고 새로운 프로세스 그룹을 생성한다.
3. 터미널 입출력 시그널을 무시한다.
4. 파일 모드 생성 마스크를 해제한다.
5. 현재 디렉터리를 루트 디렉터리로 설정한다.
6. 열려있는 모든 파일 디스크립터를 닫는다.
7. 표준 입출력과 오류출력을 /dev/null로 리다이렉션한다.

본 설계과제에서는 디몬 프로세스, 시그널, FIFO 파일을 이용하여 ssu\_vim 프로그램과 ssu\_ofm 프로그램을 구현함으로써 Reader-Writer의 동기화 문제, 공유 파일에 대한 접근 제어 문제를 이해하고 이를 해결하는 것을 목표로 한다.

* ssu\_vim : Reader와 Writer의 역할을 수행할 프로그램으로 자식 프로세스로 Vim 프로세스를 실행하여 에디터를 수행하는 명령어
* ssu\_ofm : 디몬 프로세스를 생성하고 생성한 디몬 프로세스로 FIFO 파일과 시그널을 이용하여 공유 파일의 Reader-Writer Problem을 제어하는 관리자로서의 역할을 수행하는 명령어

2. 설계

다음은 본 설계과제 프로그램의 개요도이다.



ssu\_ofm은 디몬 프로세스를 생성하여 공유 파일을 관리하고 로그 파일에 로그 기록을 남기는 프로그램이다. ssu\_vim은 공유 파일에 대하여 입출력을 수행하는 프로그램이다.

ssu\_vim은 read에 대해서는 ssu\_ofm에서 생성되는 디몬 프로세스와 무관하게 수행되는 반면 write에 대해서는 디몬 프로세스가 생성되기 전에는 수행될 수 없다. ssu\_vim에서 파일에 대해 write을 요청할 때 FIFO 파일을 통해 요청하려는 파일 이름을 전송하면 ssu\_ofm에서 해당 파일이 ssu\_ofm에서 관리하는 파일인지 확인한다. ssu\_vim은 SIGUSR1 시그널을 전송하고 ssu\_ofm에게 SIGUSR1 시그널을 전송받을 때까지 대기상태에 빠진다. ssu\_ofm은 ssu\_vim으로부터 SIGUSR1 시그널을 전송받으면 다른 프로세스가 공유 파일을 접근하고 있는지 확인한다. 공유 파일에 다른 프로세스가 접근하고 있으면 현재 파일 접근을 요청한 프로세스를 Queue에 저장한다. 공유 파일에 다른 프로세스가 접근하고 있지 않은 상태이고 Queue 또한 비어 있는 상태라면 현재 요청한 프로세스가 파일에 접근하게 되고 이때 ssu\_ofm이 ssu\_vim에게 SIGUSR1 시그널을 전송하게 된다. 프로세스가 공유 파일의 접근을 마치면 ssu\_vim에서 SIGUSR2 시그널을 ssu\_ofm에게 전송한다. SIGUSR2를 전송받은 ssu\_ofm은 만약 Queue에 다른 프로세스들이 저장되어 있다면 차례대로 SIGUSR1을 전송하면서 프로그램을 진행한다. ssu\_ofm은 ssu\_vim과 상호작용하는 과정에서 파일에 접근하려는 프로세스 id, 접근이 끝난 프로세스 id 등과 같은 로그를 ssu\_log.txt에 기록한다.

3. 구현

(1) ssu\_vim

1) void check\_option\_vim(int argc, char \*argv[])

- 기능 : ssu\_vim의 usage와 맞는지 검사하고 입력받은 옵션을 분류하는 작업을 수행하는 함수

- 매개변수

> argc : command line에서 입력받은 argument count

> argv[] : command line에서 입력받은 argument vector

- 반환값 : 없음

2) void do\_ssu\_vim(void)

- 기능 : -r, -w, -rw 옵션에 대한 실행을 관리하는 함수

- 매개변수 : 없음

- 반환값 : 없음

3) void read\_vim(void)

- 기능 : 공유 파일에 대한 read를 수행하는 함수

- 매개변수 : 없음

- 반환값 : 없음

4) void write\_vim(void)

- 기능 : 공유 파일에 대한 write을 수행하는 함수

- 매개변수 : 없음

- 반환값 : 없음

5) void read\_write\_vim(void)

- 기능 : 공유 파일에 대한 read, write을 수행하는 함수

- 매개변수 : 없음

- 반환값 : 없음

6) void execute\_vim(pid\_t daemon\_pid)

- 기능 : 공유 파일에 대해 write을 요청하는 함수. ssu\_ofm의 디몬 프로세스에 SIGUSR1 시그널을 전송하고 디몬 프로세스로부터 SIGUSR1 시그널을 전송받을 때까지 대기 상태에 빠진다.

- 매개변수

> daemon\_pid : ssu\_ofm의 디몬 프로세스 id

- 반환값 : 없음

7) bool is\_number(char \*str)

- 기능 : 매개변수로 입력받은 문자열이 숫자인지 판별하는 함수

- 매개변수

> str : 숫자인지 판별할 문자열

- 반환값 : 문자열이 숫자일 경우 true, 문자열이 숫자가 아닐 경우 false 반환

8) int find\_daemon(void)

- 기능 : 디몬 프로세스가 실행 중인지 찾는 함수

- 매개변수 : 없음

- 반환값 : 디몬 프로세스가 실행 중이라면 디몬 프로세스 id, 실행 중이 아니라면 0 반환

9) void execute\_t\_option(void)

- 기능 : -t 옵션에서 파일의 마지막 수정시간과 현재 시간을 출력해주는 함수

- 매개변수 : 없음

- 반환값 : 없음

10) struct tm \*get\_current\_time(void)

- 기능 : 현재 시간을 찾는 함수

- 매개변수 : 없음

- 반환값 : 현재 시간을 tm 구조체 타입으로 반환

11) void check\_modificaiton(time\_t before\_st\_mtime)

- 기능 : -t 옵션에서 파일이 수정되었는지 여부를 출력해주는 함수

- 매개변수

> before\_st\_mtime : 이전의 수정시간

- 반환값 : 없음

12) bool is\_modified(time\_t before\_st\_mtime)

- 기능 : 파일 접근 전 후의 수정 시간을 비교하여 파일이 수정되었지 여부를 알려주는 함수

- 매개변수

> before\_st\_mtime : 이전의 수정시간

- 반환값 : 파일이 수정되었을 경우 true, 수정되지 않았을 경우 false

13) void SIGUSR1\_handler(int signo, siginfo\_t \*info, void \*context)

- 기능 : ssu\_ofm으로부터 SIGUSR1 시그널을 전송받았을 때 실행되는 시그널 핸들러 함수

- 매개변수

> signo : 시그널 번호 (시그널 이름)

> info : siginfo\_t 구조체 변수

> context : context 변수

- 반환값 : 없음

14) void set\_signal(int signo, void (\*handler)(int, siginfo\_t \*, void \*))

- 기능 : sigaction의 인자로 들어갈 sigaction 구조체 변수를 초기화하고 sigaction 함수를 호출하는 함수

- 매개변수

> signo : 시그널 번호 (시그널 이름)

> handler : sigaction의 시그널 핸들러 함수 포인터

- 반환값 : 없음

15) void send\_message(void)

- 기능 : FIFO 파일에 write하는 함수

- 매개변수 : 없음

- 반환값 : 없음

16) int get\_file\_size(char \*file)

- 기능 : 파일 크기를 구하는 함수

- 매개변수

> file : 파일 이름

- 반환값 : 파일 크기 반환

17) void execute\_s\_option(int before\_size, int after\_size)

- 기능 : -s 옵션에서 파일 접근 전, 후의 파일 크기를 출력하는 함수

- 매개변수

> before\_size : 파일 접근 전의 파일 크기

> after\_size : 파일 접근 후의 파일 크기

- 반환값 : 없음

18) void copy\_file(char \*new\_file, char \*old\_file)

- 기능 : old\_file 파일 데이터를 new\_file 파일 데이터로 복사하는 함수

- 매개변수

> new\_file : 복사 내용을 저장할 파일 이름

> old\_file : 복사 내용을 담고 있는 파일

- 반환값 : 없음

19) void execute\_d\_option(char \*before\_file)

- 기능 : -d 옵션에서 diff 명령어를 실행시키는 함수

- 매개변수

> before\_file : diff 명령어 인자로 들어갈 파일 이름

- 반환값 : 없음

(2) ssu\_ofm

1) void check\_option\_ofm(int argc, char \*argv[])

- 기능 : ssu\_ofm의 usage와 맞는지 검사하고 입력받은 옵션을 분류하는 작업을 수행하는 함수

- 매개변수

> argc : command line에서 입력받은 argument count

> argv[] : command line에서 입력받은 argument vector

- 반환값 : 없음

2) bool is\_number(char \*str)

- 기능 : 매개변수로 입력받은 문자열이 숫자인지 판별하는 함수

- 매개변수

> str : 숫자인지 판별할 문자열

- 반환값 : 문자열이 숫자일 경우 true, 문자열이 숫자가 아닐 경우 false 반환

3) bool is\_dir(char \*str)

- 기능 : 디렉터리 이름인지 판별하는 함수

- 매개변수

> str : 디렉터리 이름인지 판별할 문자열

- 반환값 : -로 시작하면 옵션이므로 false, 너머지는 true 반환

4) void set\_log\_path(char \*path, char \*log, char \*dir)

- 기능 : log 파일의 경로를 설정해주는 함수

- 매개변수

> path : 경로를 저장할 변수

> log : log 파일의 이름

> dir : -p 옵션을 입력받은 경우 함께 입력받은 디렉터리 이름, -p 옵션을 입력받지 않은 경우 NULL

- 반환값 : 없음

5) int ssu\_daemon\_init(void)

- 기능 : 디몬 프로세스를 생성하는 함수

- 매개변수 : 없음

- 반환값 : 0 반환

6) void write\_log\_init(void)

- 기능 : 디몬 프로세스가 생성되자마자 써야 할 내용을 log 파일에 write하는 함수

- 매개변수 : 없음

- 반환값 : 없음

7) char \*get\_current\_time(void)

- 기능 : 현재 시간을 찾는 함수

- 매개변수 : 없음

- 반환값 : 현재 시간을 문자열 타입으로 반환

8) void set\_signal(int signo, void (\*handler)(int, siginfo\_t \*, void \*))

- 기능 : sigaction의 인자로 들어갈 sigaction 구조체 변수를 초기화하고 sigaction 함수를 호출하는 함수

- 매개변수

> signo : 시그널 번호 (시그널 이름)

> handler : sigaction의 시그널 핸들러 함수 포인터

- 반환값 : 없음

9) void SIGUSR1\_handler(int signo, siginfo\_t \*si, void \*context)

- 기능 : ssu\_vim으로부터 SIGUSR1 시그널을 전송받았을 때 실행되는 시그널 핸들러 함수

- 매개변수

> signo : 시그널 번호 (시그널 이름)

> info : siginfo\_t 구조체 변수

> context : context 변수

- 반환값 : 없음

10) void make\_fifo(void)

- 기능 : FIFO 파일을 생성하는 함수

- 매개변수 : 없음

- 반환값 : 없음

11) void shared\_file(char \*buf)

- 기능 : sender로부터 FIFO에 입력된 데이터를 read하는 함수

- 매개변수

> buf : read한 값을 저장

- 반환값 : 없음

12) void write\_log\_requested(pid\_t pid, char \*shared)

- 기능 : ssu\_vim으로부터 SIGUSR1 시그널을 전송받았을 때 log 파일에 써야할 데이터를 write하는 함수

- 매개변수

> pid : SIGUSR1 시그널을 보낸 프로세스의 id

> shared : 접근을 요청한 파일이름

- 반환값 : 없음

13) void execute\_id\_option(char \*buf)

- 기능 : -id 옵션에서 SIGUSR2를 전송한 프로세스의 유저이름, UID, GID를 log 파일에 write할 준비를 함수

- 매개변수

> buf : write할 데이터를 저장

- 반환값 : 없음

14) bool is\_running(void)

- 기능 : 현재 공유 파일이 다른 프로세스에 의해 접근 중인지 확인하는 함수

- 매개변수 : 없음

- 반환값 : 프로세스가 공유 파일에 접근 중인 경우에는 true, 접근 중이지 않은 경우에는 false 반환

15) void SIGUSR2\_handler(int signo, siginfo\_t \*info, void \*context)

- 기능 : ssu\_vim으로부터 SIGUSR2 시그널을 전송받았을 때 실행되는 시그널 핸들러 함수

- 매개변수

> signo : 시그널 번호 (시그널 이름)

> info : siginfo\_t 구조체 변수

> context : context 변수

- 반환값 : 없음

16) void write\_log\_finished(pid\_t pid)

- 기능 : ssu\_vim으로부터 SIGUSR2 시그널을 전송받았을 때 log 파일에 써야할 데이터를 write하는 함수

- 매개변수

> pid : SIGUSR2 시그널을 보낸 프로세스의 id

- 반환값 : 없음

17) void copy\_file(char \*new\_file, char \*old\_file)

- 기능 : old\_file 파일 데이터를 new\_file 파일 데이터로 복사하는 함수

- 매개변수

> new\_file : 복사 내용을 저장할 파일 이름

> old\_file : 복사 내용을 담고 있는 파일

- 반환값 : 없음

(3) queue

1) Node 구조체

typedef struct \_node {

pid\_t pid;

struct \_node \*next;

} Node;

- pid : 프로세스 id

- next : LinkedList로 구현하는 queue

2) Queue 구조체

typedef struct \_queue {

Node \*front, \*rear;

int count;

int max\_size;

} Queue;

- front : queue의 맨 앞 노드를 가리키는 포인터

- rear : queue의 맨 뒤 노드를 가리키는 포인터

- count : 노드의 개수

- max\_size : 최대로 저장할 수 있는 노드의 개수

3) Node\* newNode(pid\_t pid)

- 기능 : 새로운 노드를 생성하는 함수

- 매개변수

> pid : 노드 생성 시 추가할 데이터 (프로세스 id)

- 반환값 : 노드 포인터

4) void init\_queue(Queue \*q)

- 기능 : queue를 초기화하는 함수

- 매개변수

> q : queue 포인터

- 반환값 : 없음

5) void enqueue(Queue \*q, pid\_t pid)

- 기능 : queue의 rear에서 데이터를 추가하는 함수

- 매개변수

> q : queue 포인터

> pid : 추가할 데이터 (프로세스 id)

- 반환값 : 없음

6) pid\_t dequeue(Queue \*q)

- 기능 : queue의 front에서 데이터를 제거하는 함수

- 매개변수

> q : queue 포인터

- 반환값 : 프로세스id 반환

7) bool empty(Queue \*q)

- 기능 : 비어있는 queue인지 확인

- 매개변수

> q : queue 포인터

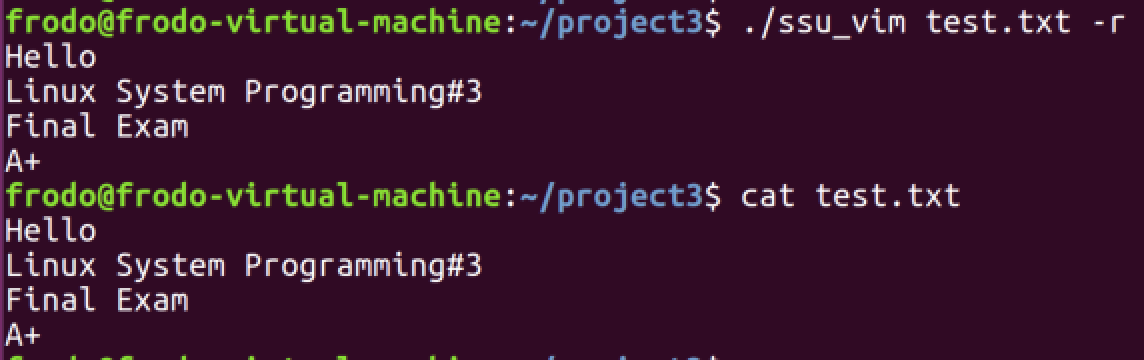
- 반환값 : 비어있으면 true, 비어있지 않으면 false 반환

4. 테스트 및 결과

(1) ssu\_vim

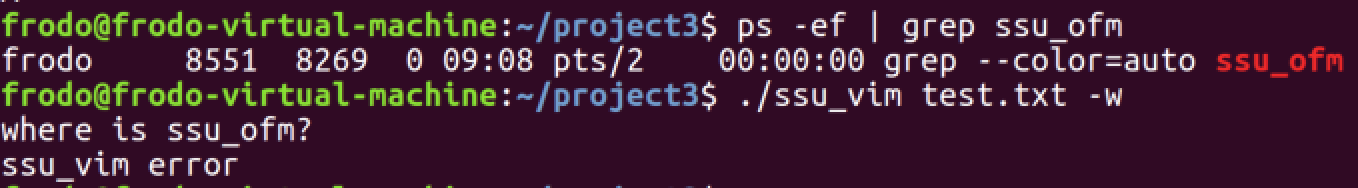
1) -r 옵션 실행

- ssu\_ofm의 디몬 프로세스가 생성되는 것과 무관하게 read는 수행 가능

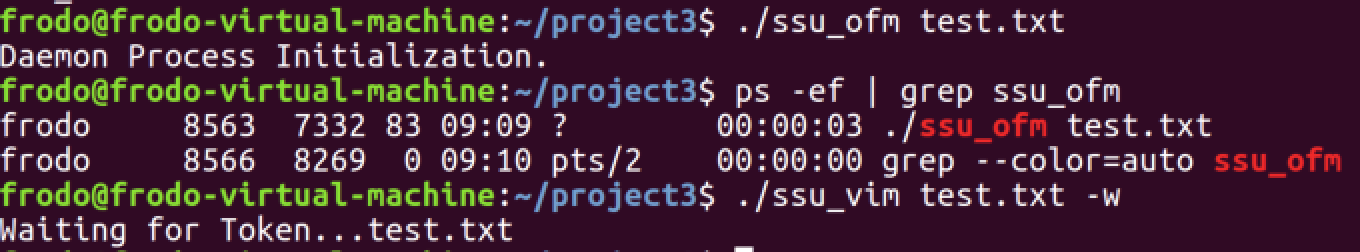


2) -w 옵션 실행

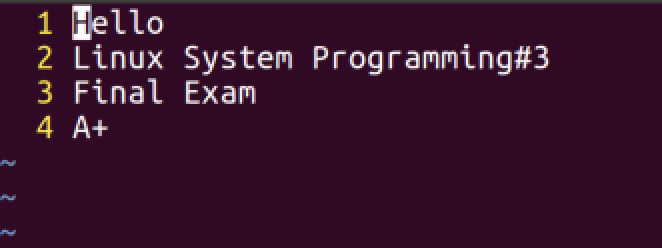
- ssu\_ofm이 실행되지 않은 경우 write는 수행되지 않는다.



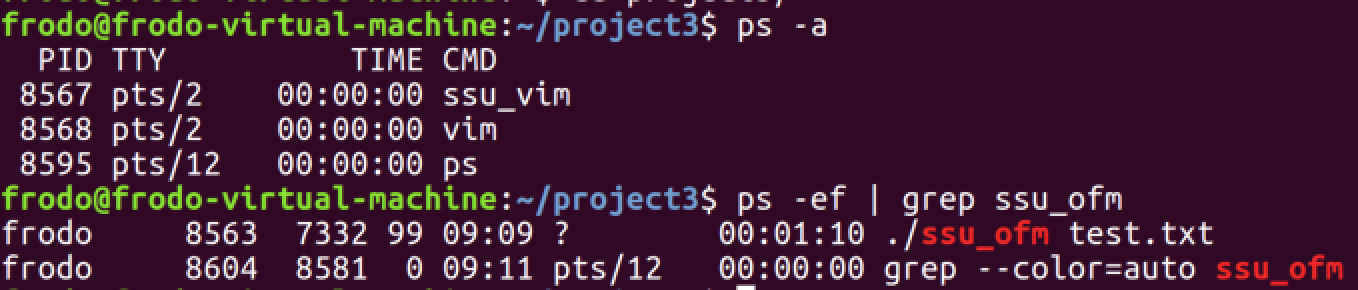
- ssu\_ofm이 실행한 경우 write가 수행된다.



> test.txt 파일이 open된 상태

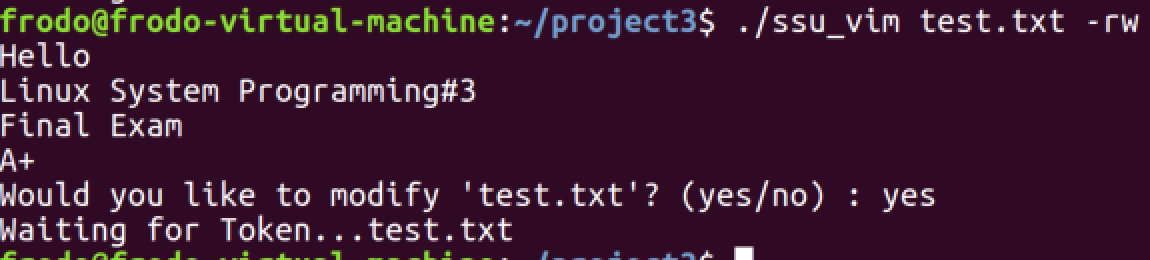


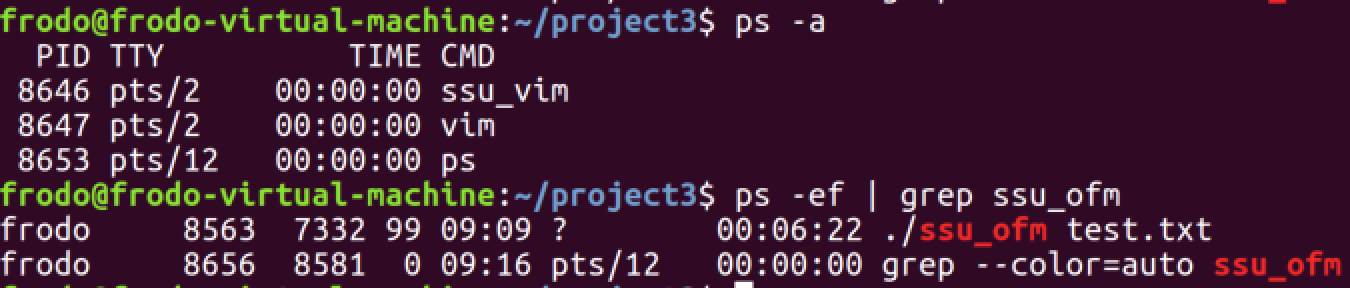
> ssu\_vim 실행 중인 상황



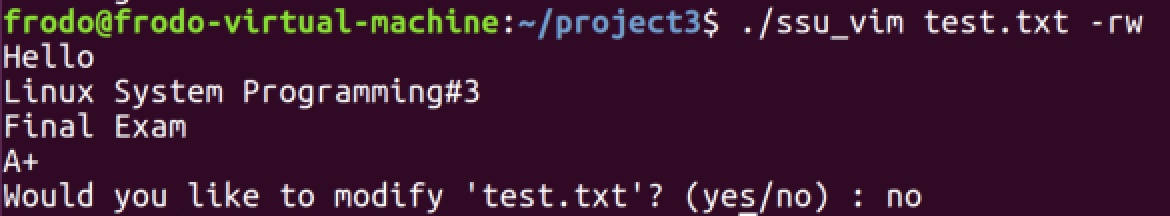
3) -rw 옵션 실행

- write에 대해 yes라고 응답한 경우 read 후 write 수행



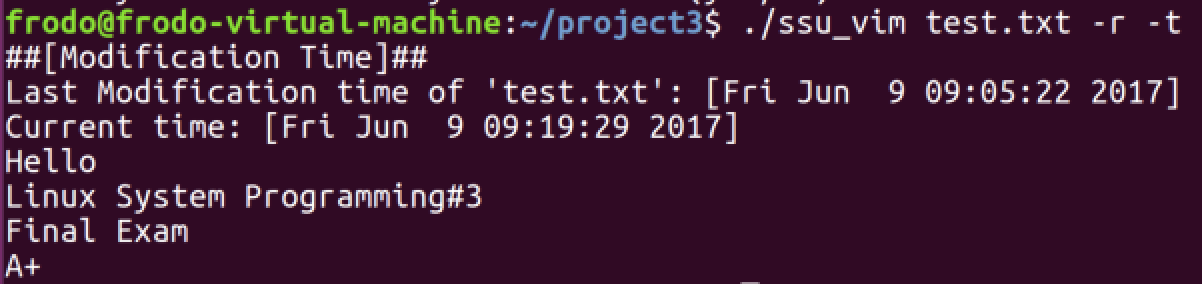


- write에 대해 no라고 응답한 경우 read만 수행

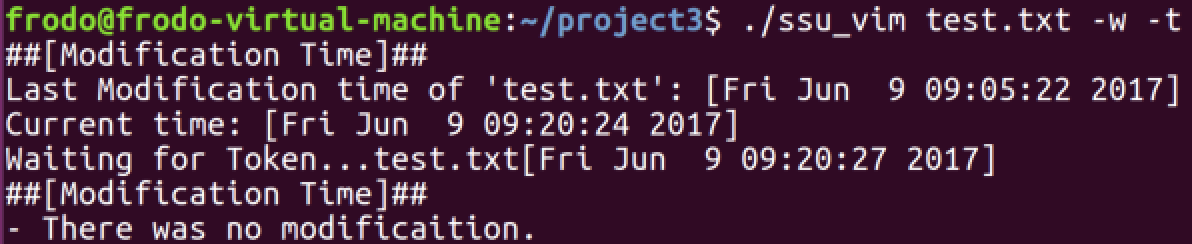


4) –t 옵션 실행

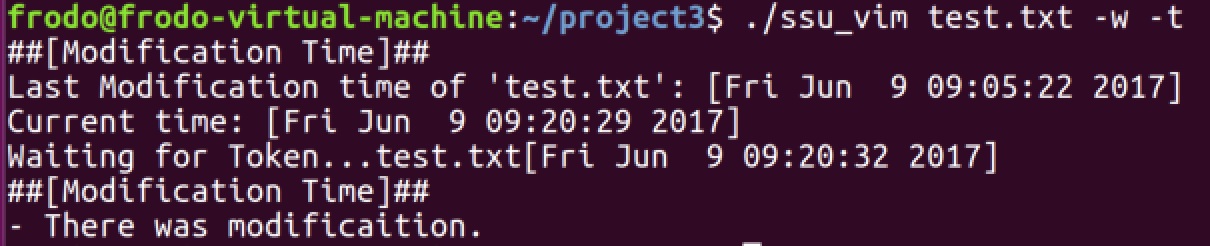
- -r 옵션



- vim을 q로 종료



- vim을 wq로 종료

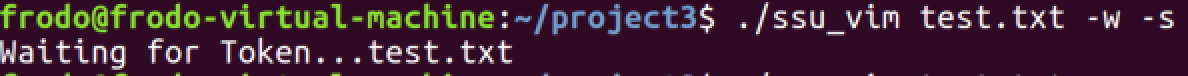


5) -s 옵션 실행

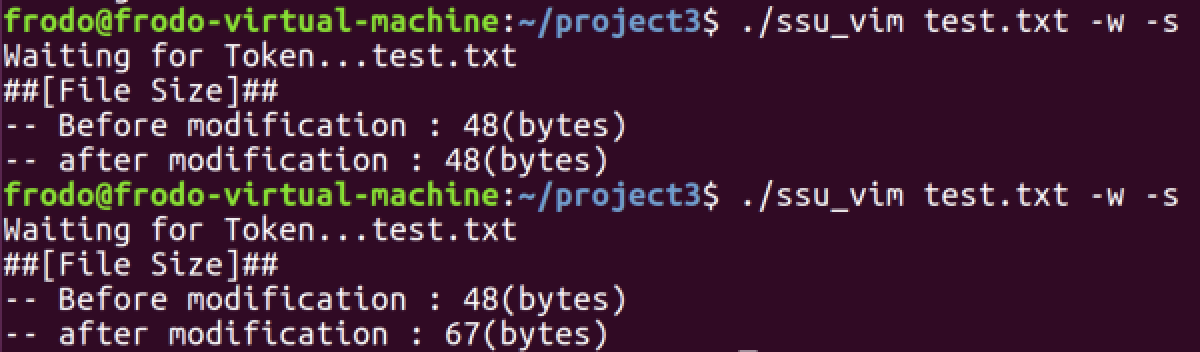
- -r 옵션



- vim을 q로 종료

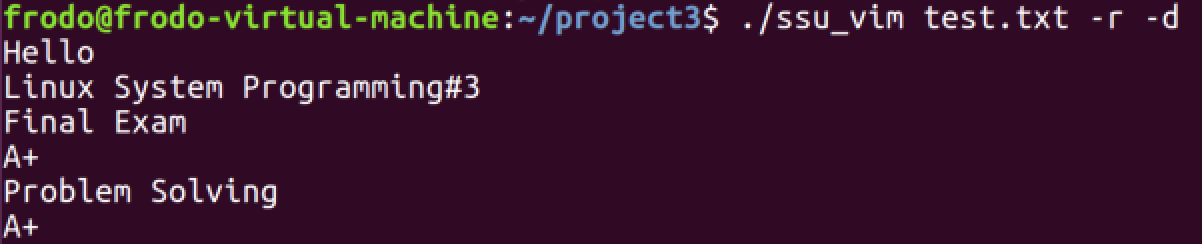


- vim을 wq로 종료

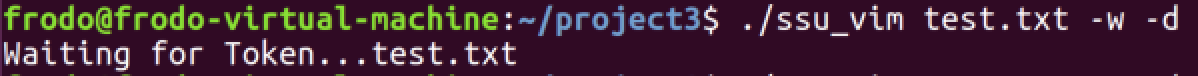


6) -d 옵션 실행

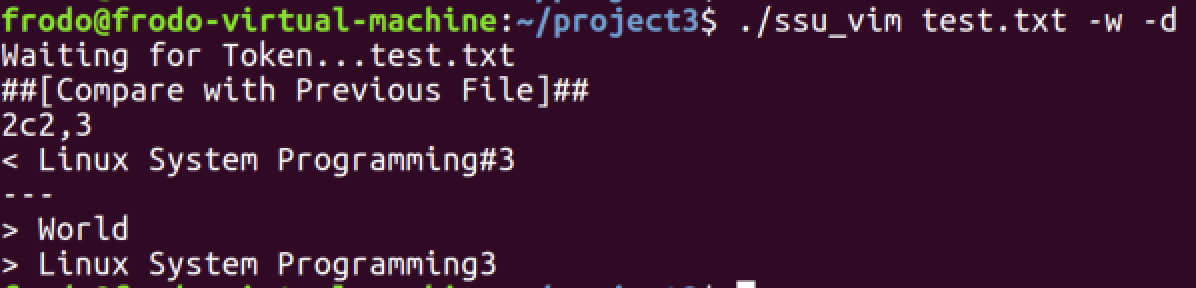
- -r 옵션



- vim을 q로 종료

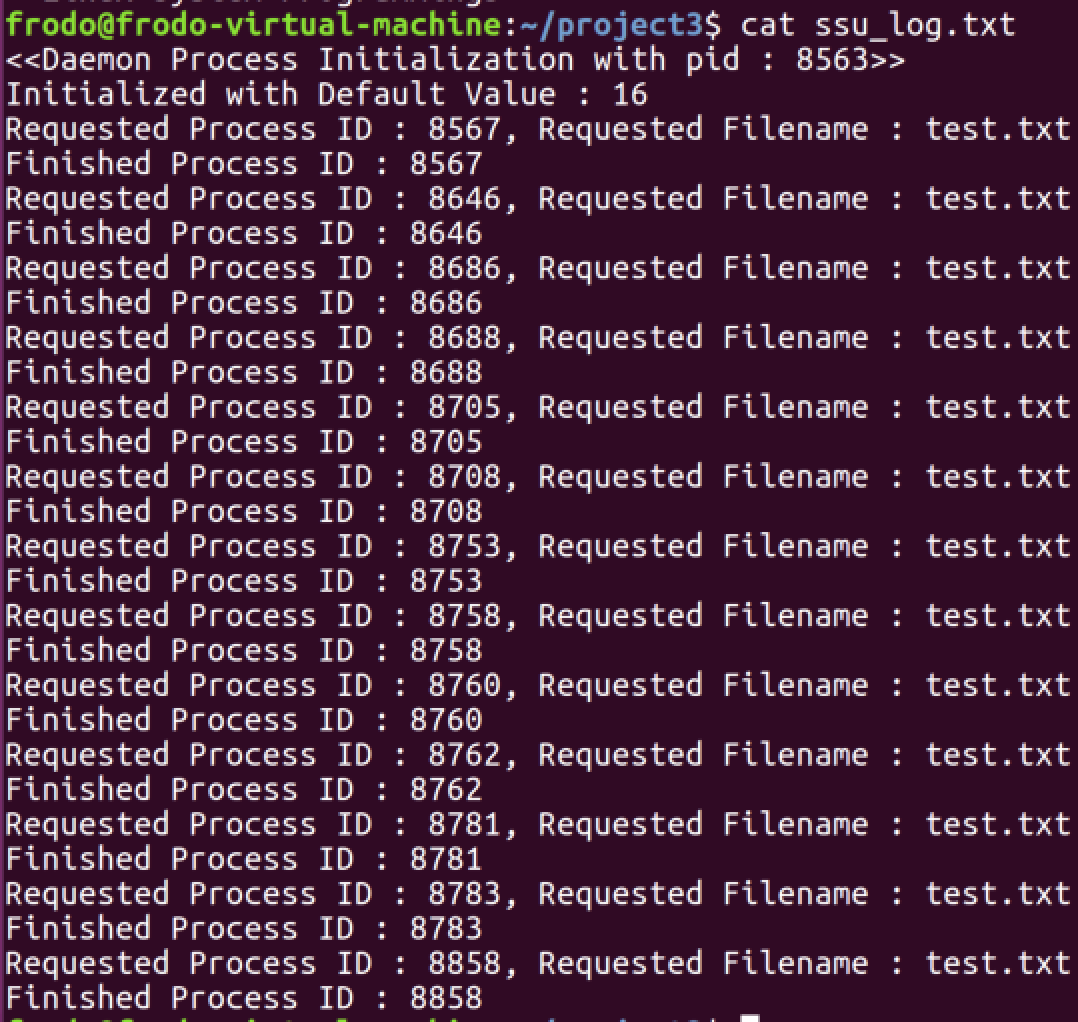


- vim을 wq로 종료

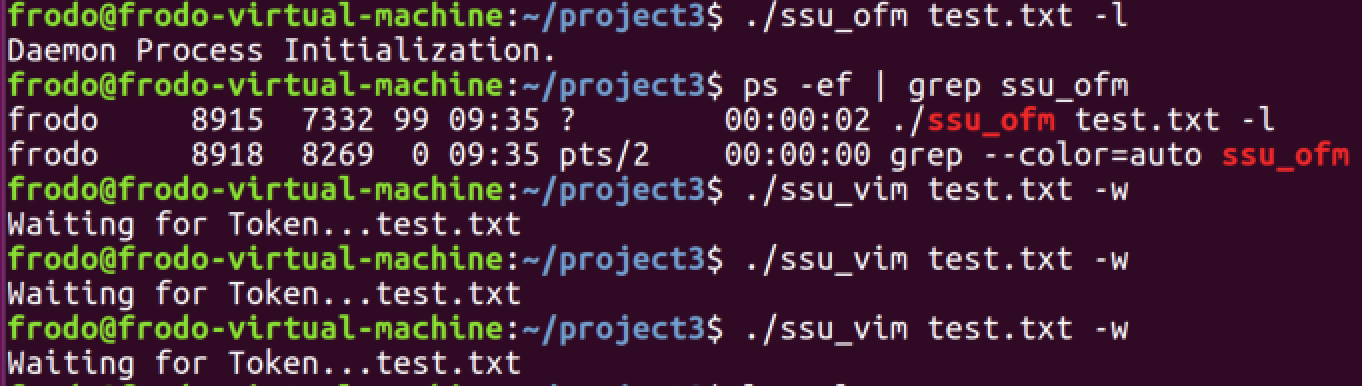


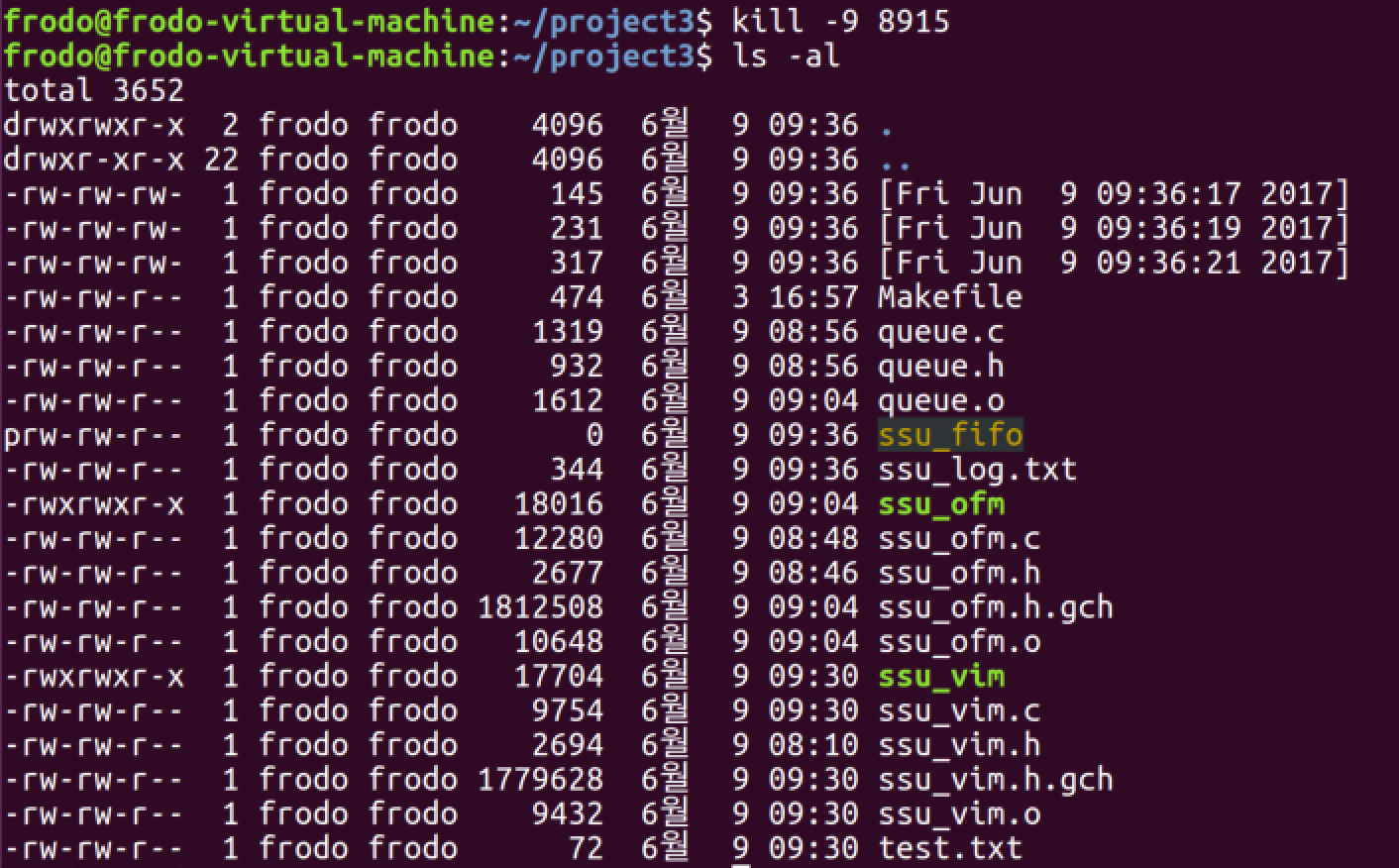
(2) ssu\_ofm

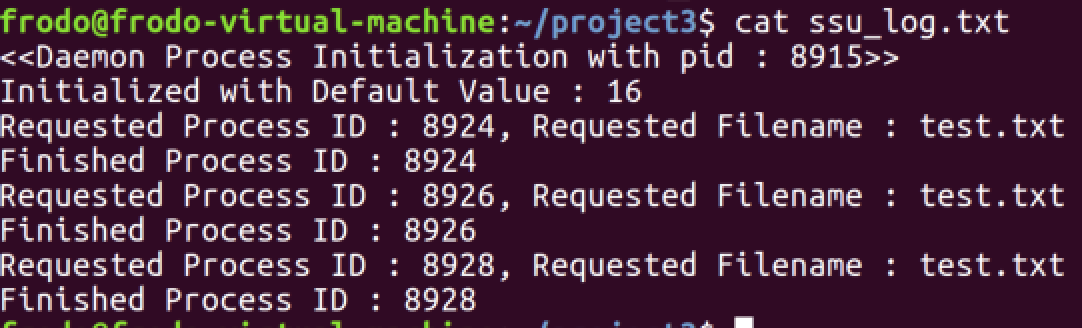
1) 옵션이 없는 경우



2) -ㅣ 옵션 실행

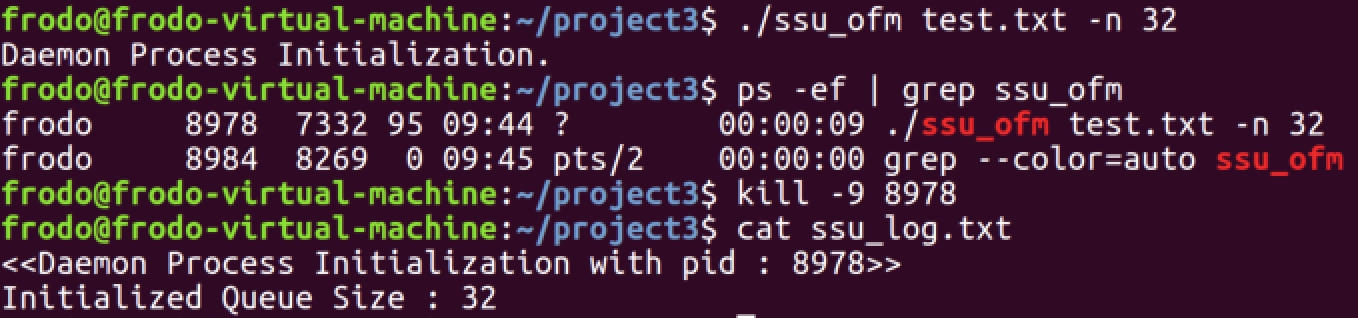




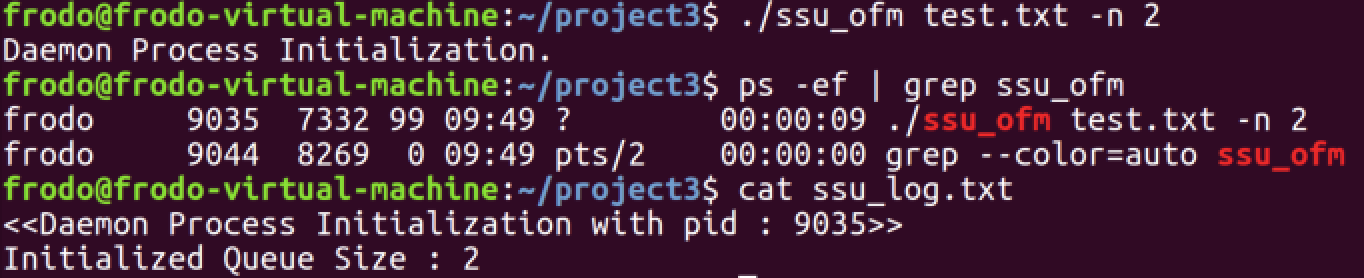


3) -n 옵션 실행

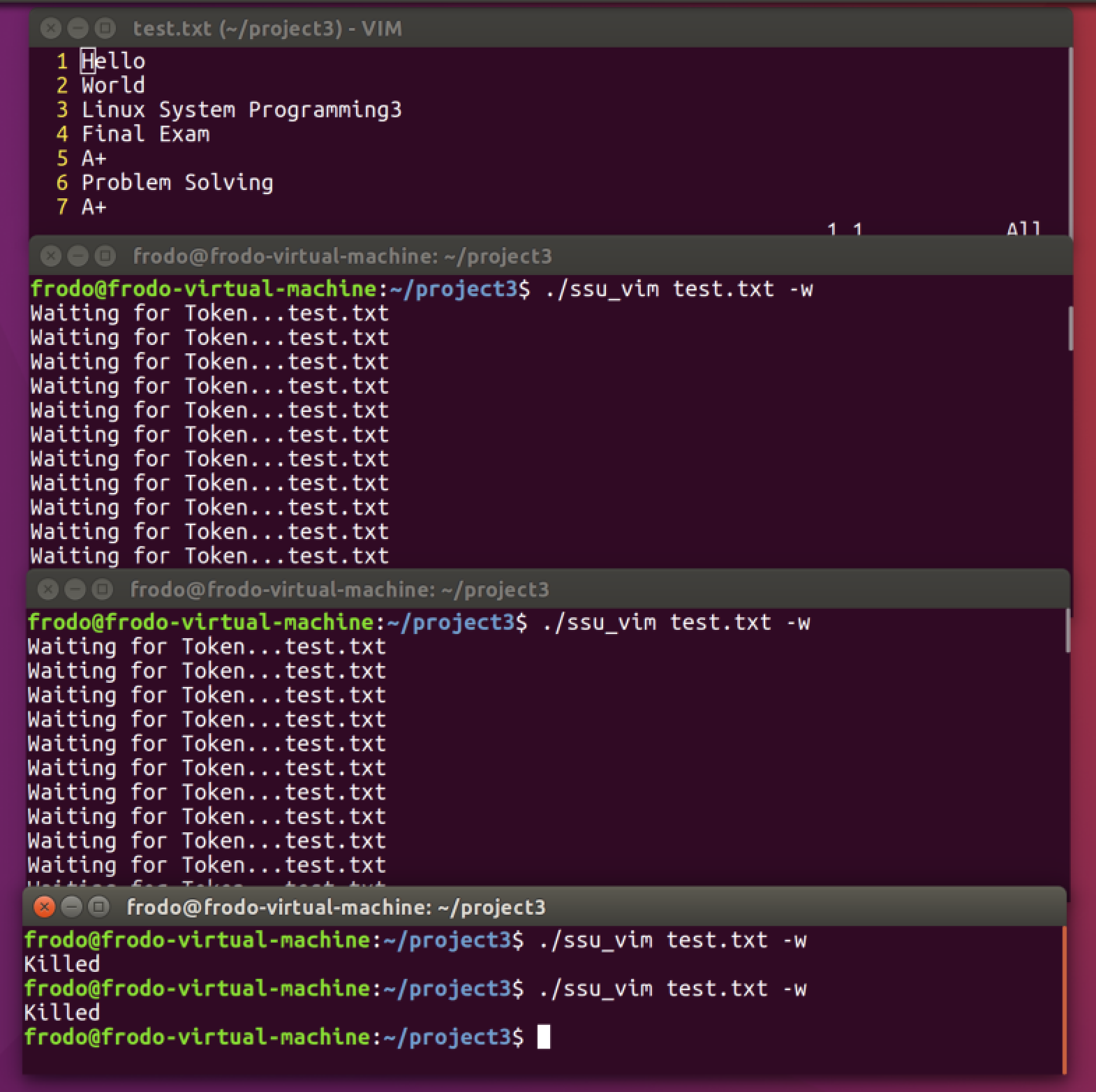
- queue의 크기가 32인 경우



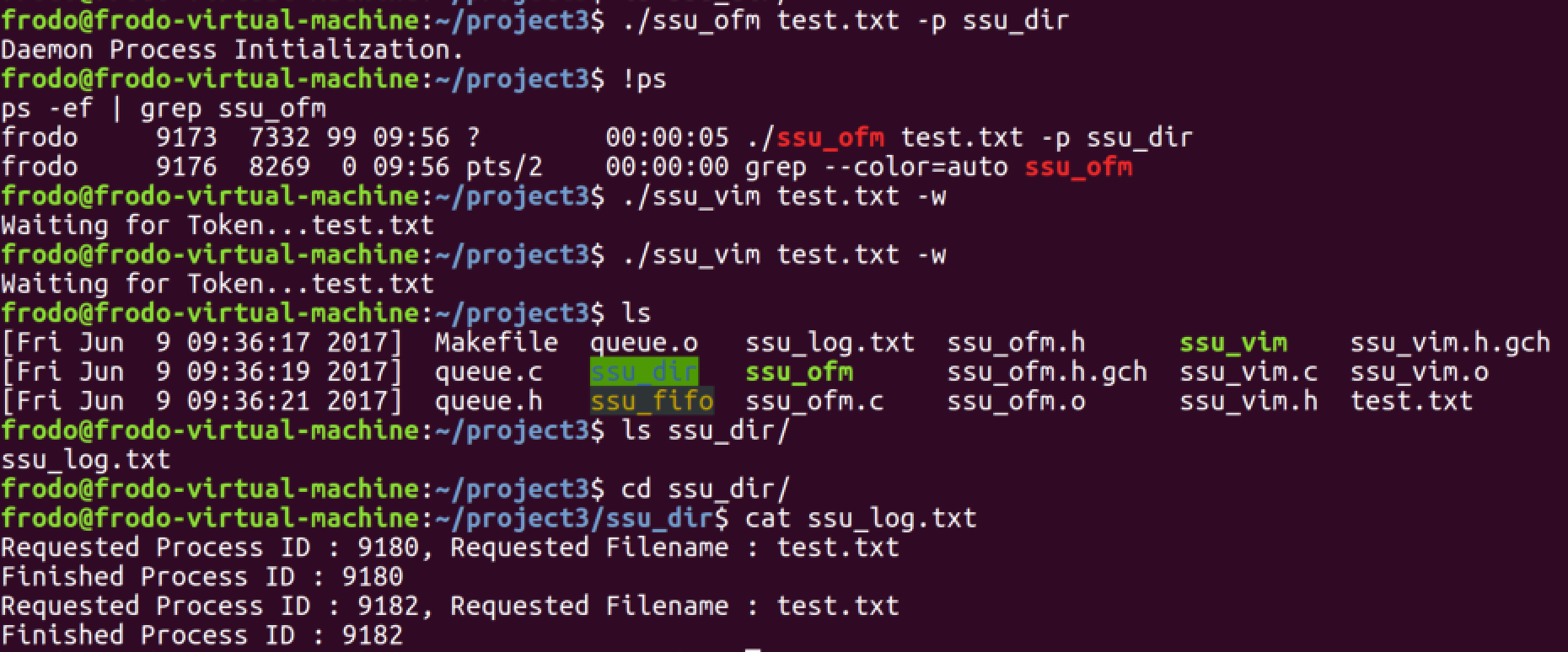
- queue의 크기가 2인 경우



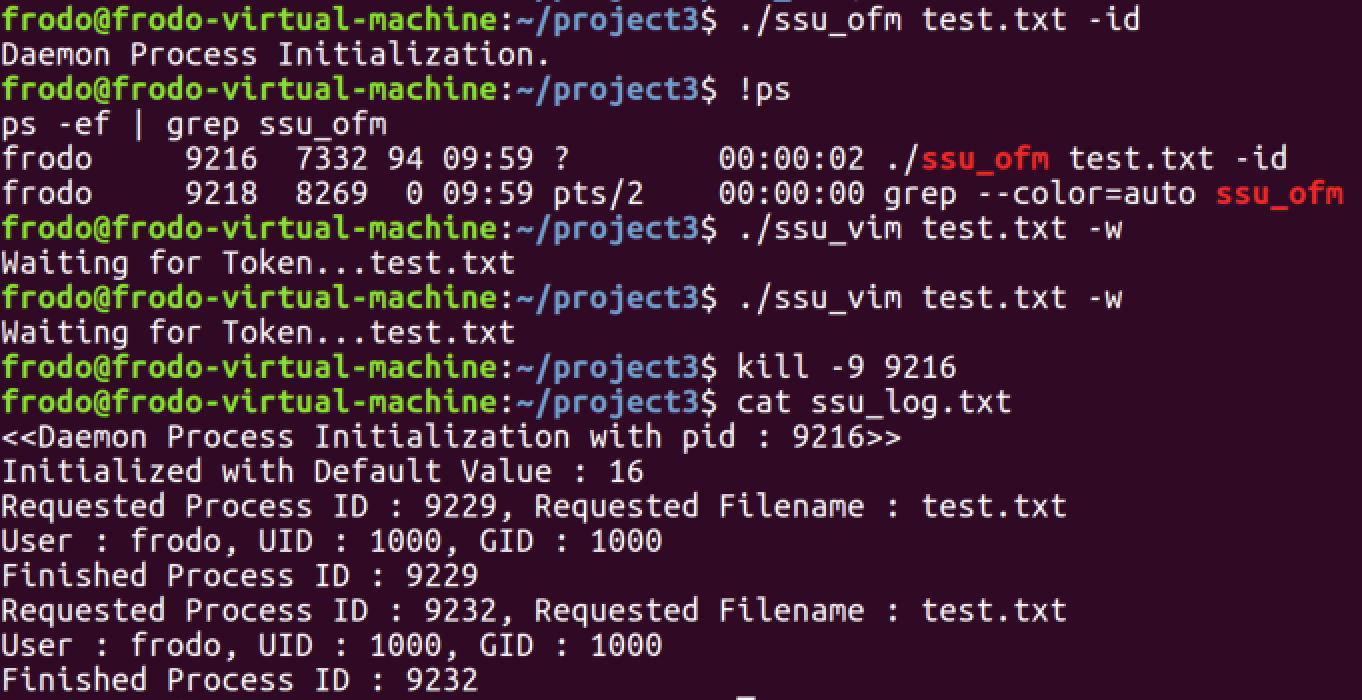
> queue의 최대 크기보다 더 많은 프로세스가 공유 파일에 접근을 요청하는 경우 최대 크기를 초과하여 요청을 한 프로세스들에게 SIGKILL 시그널을 보내도록 처리하였다.



4) -p 옵션 실행



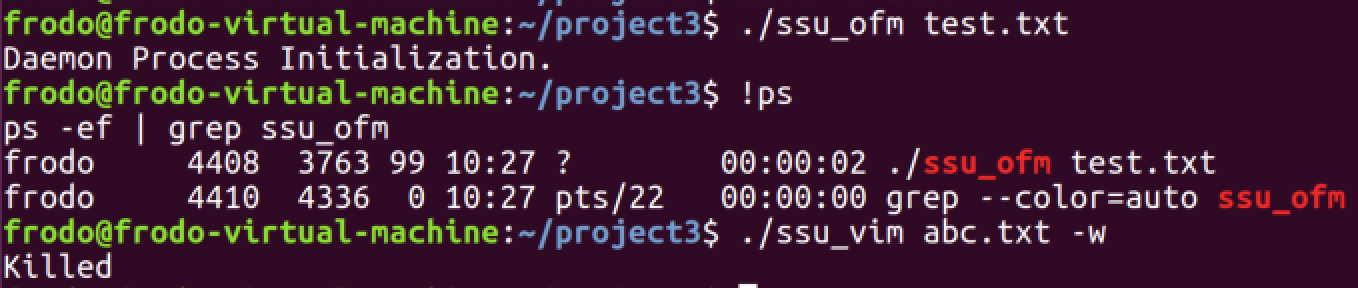
5) –id 옵션 실행



(3) 예외처리

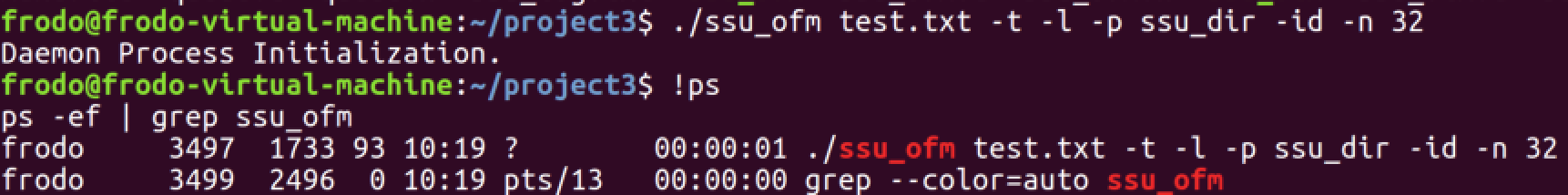
1) ssu\_vim에서 접근을 요청하는 파일이 ssu\_ofm에서 관리하는 공유 파일이 아닌 경우

- ssu\_ofm에서 ssu\_vim으로 SIGKILL 시그널을 전송하도록 처리

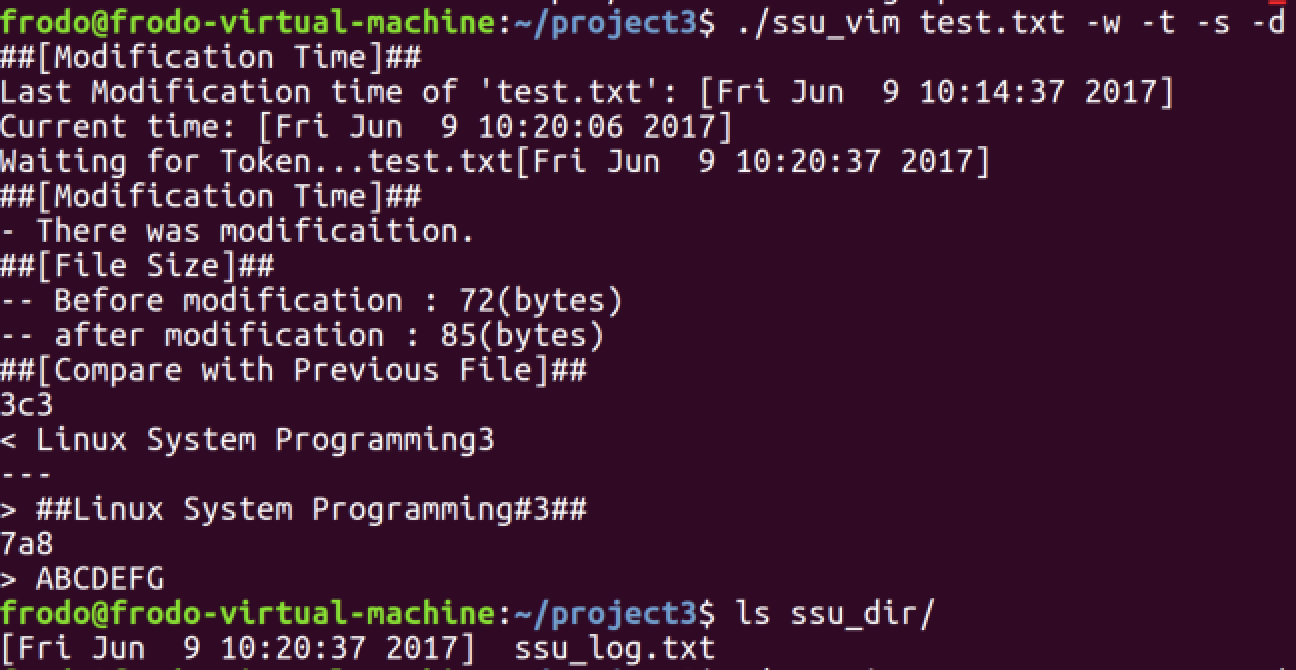


(4) 옵션들 복합 실행

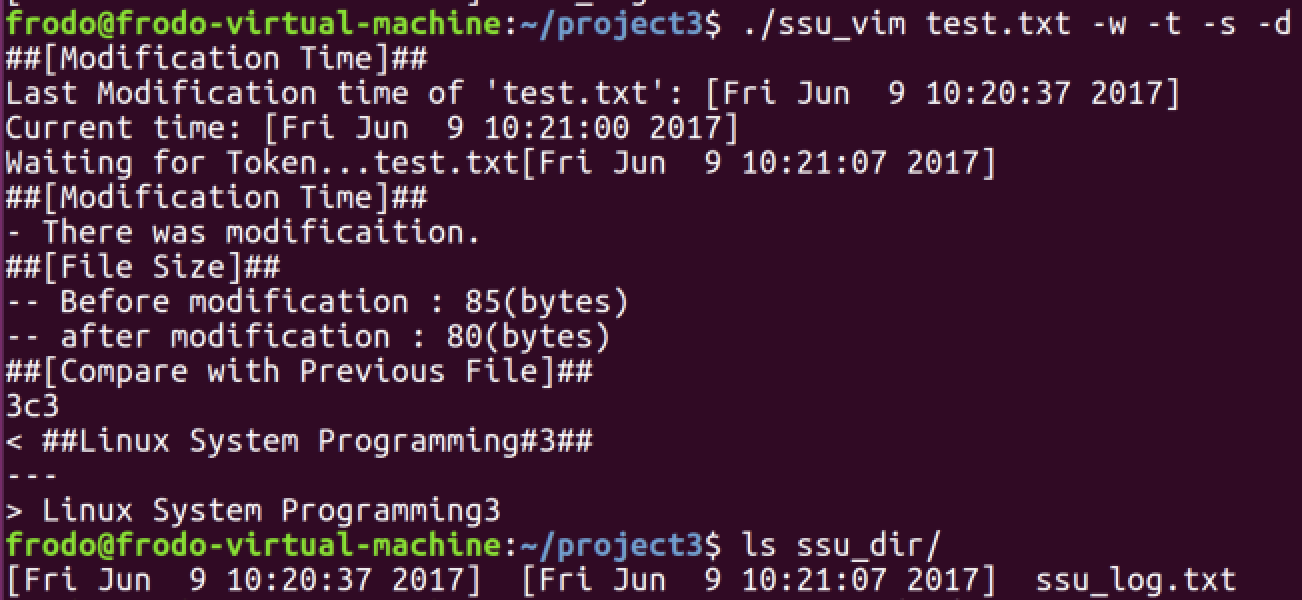
- ssu\_ofm의 모든 옵션 실행



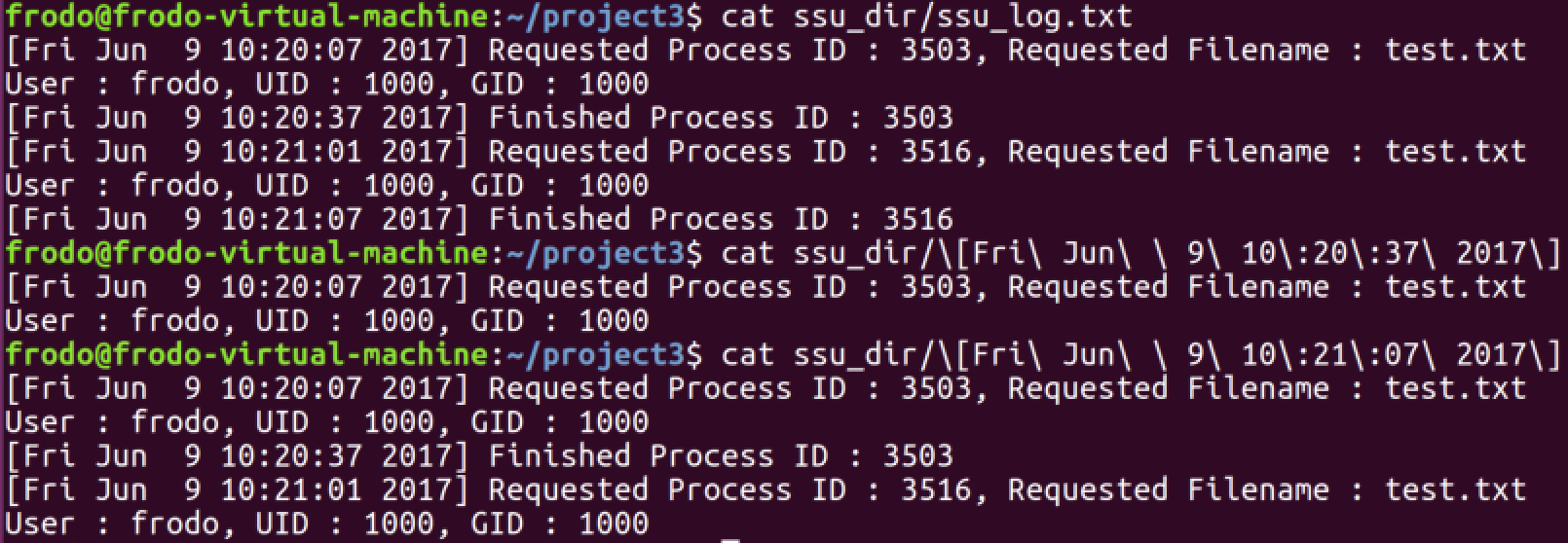
- ssu\_vim의 모든 옵션 실행



- ssu\_vim 다시 한 번 실행



- log 파일 보기



5. 소스코드

1) ssu\_vim.h

#ifndef \_\_SSU\_VIM\_H\_\_

#define \_\_SSU\_VIM\_H\_\_

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <fcntl.h>

#include <unistd.h>

#include <stdbool.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <sys/stat.h>

#include <ctype.h>

#include <dirent.h>

#include <time.h>

//-r, -w, -rw 옵션 플래그 값

#define R\_FLAG 0x1

#define W\_FLAG 0x2

#define RW\_FLAG 0x4

//-t, -s, -d 옵션 플래그 값

#define VIM\_T\_FLAG 0x1

#define VIM\_S\_FLAG 0x2

#define VIM\_D\_FLAG 0x4

#define BUF\_SIZE 1024

#define TIME\_SIZE 64

//FIFO 파일 이름

#define FIFO\_FILE "ssu\_fifo"

//ssu\_vim의 usage와 맞는지 검사하고 입력받은 옵션을 분류하는 작업을 수행하는 함수

void check\_option\_vim(int argc, char \*argv[]);

//-r, -w, -rw 옵션에 대한 실행을 관리하는 함수

void do\_ssu\_vim(void);

//공유 파일에 대한 read를 수행하는 함수

void read\_vim(void);

//공유 파일에 대한 read를 수행하는 함수

void write\_vim(void);

//공유 파일에 대한 read, write을 수행하는 함수

void read\_write\_vim(void);

//공유 파일에 대해 write을 요청하는 함수

void execute\_vim(pid\_t daemon\_pid);

//매개변수로 입력받은 문자열이 숫자인지 판별하는 함수

bool is\_number(char \*str);

//디몬 프로세스가 실행 중인지 찾는 함수

int find\_daemon(void);

//-t 옵션에서 파일의 마지막 수정시간과 현재 시간을 출력해주는 함수

void execute\_t\_option(void);

//현재 시간을 찾는 함수

struct tm \*get\_current\_time(void);

//-t 옵션에서 파일이 수정되었는지 여부를 출력해주는 함수

void check\_modificaiton(time\_t before\_st\_mtime);

//파일 접근 전 후의 수정 시간을 비교하여 파일이 수정되었지 여부를 알려주는 함수

bool is\_modified(time\_t before\_st\_mtime);

//ssu\_ofm으로부터 SIGUSR1 시그널을 전송받았을 때 실행되는 시그널 핸들러 함수

void SIGUSR1\_handler(int signo, siginfo\_t \*info, void \*context);

//sigaction의 인자로 들어갈 sigaction 구조체 변수를 초기화하고 sigaction 함수를 호출하는 함수

void set\_signal(int signo, void (\*handler)(int, siginfo\_t \*, void \*));

//FIFO 파일에 write하는 함수

void send\_message(void);

//파일 크기를 구하는 함수

int get\_file\_size(char \*file);

//-s 옵션에서 파일 접근 전, 후의 파일 크기를 출력하는 함수

void execute\_s\_option(int before\_size, int after\_size);

//old\_file 파일 데이터를 new\_file 파일 데이터로 복사하는 함수

void copy\_file(char \*new\_file, char \*oldfile);

//old\_file 파일 데이터를 new\_file 파일 데이터로 복사하는 함수

void execute\_d\_option(char \*before\_file);

#endif

2) ssu\_vim.c

#include "ssu\_vim.h"

//-r, -w, -rw 옵션의 입력 여부를 판별하는 플래그, 비트연산

int rw\_flag = 0;

//-t, -s, -d 옵션의 입력 여부를 판별하는 플래그, 비트연산

int vim\_flag = 0;

//SIGUSR1 시그널 수신 여부를 판별하는 플래그

bool SIGUSR1\_flag = false;

//접근을 요청하고자 하는 파일 이름을 저장

char \*fname = NULL;

int main(int argc, char \*argv[]) {

//SIGUSR1 시그널 등록

set\_signal(SIGUSR1, SIGUSR1\_handler);

//옵션 처리

check\_option\_vim(argc, argv);

//-r, -w, -rw 옵션 별 실행

do\_ssu\_vim();

exit(EXIT\_SUCCESS);

}

//ssu\_vim의 usage와 맞는지 검사하고 입력받은 옵션을 분류하는 작업을 수행하는 함수

void check\_option\_vim(int argc, char \*argv[]) {

int i;

char \*opt;

if(argc < 3) {

fprintf(stderr, "USAGE: ssu\_vim <FILENAME> <-r | -w | -rw> [OPTION]\n");

exit(EXIT\_FAILURE);

}

//접근을 요청하고자하는 파일저장

fname = argv[1];

//-r, -w, -rw 옵션 중 입력받은 옵션을 플래그에 저장

if(!strcmp(argv[2], "-r"))

rw\_flag = R\_FLAG;

else if(!strcmp(argv[2], "-w"))

rw\_flag = W\_FLAG;

else if(!strcmp(argv[2], "-rw"))

rw\_flag = RW\_FLAG;

else {

fprintf(stderr, "ERROR: Invalid option\n");

exit(EXIT\_FAILURE);

}

//-t, -s, -d 옵션 중 입력받은 옵션을 플래그에 저장

for(i = 3; i < argc; i++) {

opt = argv[i];

if(!strcmp(opt, "-t"))

vim\_flag |= VIM\_T\_FLAG;

else if(!strcmp(opt, "-s"))

vim\_flag |= VIM\_S\_FLAG;

else if(!strcmp(opt, "-d"))

vim\_flag |= VIM\_D\_FLAG;

else {

fprintf(stderr, "ERROR: Invalid option %s\n", opt);

exit(EXIT\_FAILURE);

}

}

}

//-r, -w, -rw 옵션에 대한 실행을 관리하는 함수

void do\_ssu\_vim() {

if(rw\_flag == R\_FLAG)

read\_vim();

else if(rw\_flag == W\_FLAG)

write\_vim();

else if(rw\_flag == RW\_FLAG)

read\_write\_vim();

else {

fprintf(stderr, "ERROR: Invalid option\n");

fprintf(stderr, "USAGE: ssu\_vim <FILENAME> <-r | -w | -rw> [OPTION]\n");

exit(EXIT\_FAILURE);

}

}

//공유 파일에 대한 read를 수행하는 함수

void read\_vim() {

FILE \*fp;

char buf[BUF\_SIZE];

int fsize;

fp = fopen(fname, "r");

//-t 옵션을 입력받은 경우

if(vim\_flag & VIM\_T\_FLAG)

execute\_t\_option();

//파일 데이터를 표준 출력으로 출력

fsize = fread(buf, sizeof(char), BUF\_SIZE - 1, fp);

buf[fsize] = 0;

fwrite(buf, sizeof(char), strlen(buf), stdout);

}

//공유 파일에 대한 read를 수행하는 함수

void write\_vim() {

pid\_t pid;

struct stat statbuf;

char \*tmp\_file = "tmp.c";

int before\_size, after\_size;

//디몬 프로세스가 실행하고 있지 않은 경우 예외처리

if((pid = find\_daemon()) == 0) {

printf("where is ssu\_ofm?\n");

printf("ssu\_vim error\n");

exit(EXIT\_FAILURE);

}

lstat(fname, &statbuf);

//-t 옵션을 입력받은 경우

if((vim\_flag & VIM\_T\_FLAG) && (rw\_flag != RW\_FLAG))

execute\_t\_option();

//-s 옵션을 입력받은 경우

if(vim\_flag & VIM\_S\_FLAG)

before\_size = get\_file\_size(fname);

//-d 옵션을 입력받은 경우

if(vim\_flag & VIM\_D\_FLAG)

copy\_file(tmp\_file, fname);

execute\_vim(pid);

//-t 옵션을 입력받은 경우

if(vim\_flag & VIM\_T\_FLAG)

check\_modificaiton(statbuf.st\_mtime);

//-s 옵션을 입력받은 경우

if(vim\_flag & VIM\_S\_FLAG) {

after\_size = get\_file\_size(fname);

if(is\_modified(statbuf.st\_mtime))

execute\_s\_option(before\_size, after\_size);

}

//-d 옵션을 입력받은 경우

if(vim\_flag & VIM\_D\_FLAG) {

if(is\_modified(statbuf.st\_mtime))

execute\_d\_option(tmp\_file);

remove(tmp\_file);

}

}

//공유 파일에 대한 read, write을 수행하는 함수

void read\_write\_vim() {

char y\_or\_n[32];

read\_vim();//read 수행

while(1) {//yes인 경우에만 write 수행

printf("Would you like to modify '%s'? (yes/no) : ", fname);

fgets(y\_or\_n, 31, stdin);

y\_or\_n[strlen(y\_or\_n) - 1] = 0;

if(!strcmp(y\_or\_n, "yes")) {

write\_vim();

break;

}

else if(!strcmp(y\_or\_n, "no"))

break;

else

printf("try again...\n");

}

}

//공유 파일에 대해 write을 요청하는 함수

void execute\_vim(pid\_t daemon\_pid) {

struct tm \*cur\_timeinfo;

//디몬 프로세스에 SIGUSR1 시그널을 전송한다.

kill(daemon\_pid, SIGUSR1);

//FIFO 파일에 요청하고자 하는 파일이름을 write한다.

send\_message();

//ssu\_ofm으로부터 SIGUSR1 시그널을 수신할 때까지 대기상태에 빠진다.

while(!SIGUSR1\_flag) {

sleep(1);

printf("Waiting for Token...%s", fname);

//-t 옵션을 입력받은 경우

if(vim\_flag & VIM\_T\_FLAG) {

cur\_timeinfo = get\_current\_time();

printf("[%.24s]\n", asctime(cur\_timeinfo));

}

else

printf("\n");

}

}

//매개변수로 입력받은 문자열이 숫자인지 판별하는 함수

bool is\_number(char \*str) {

int i;

for(i = 0; i < strlen(str); i++) {

if(!isdigit(str[i]))

return false;

}

return true;

}

//디몬 프로세스가 실행 중인지 찾는 함수

int find\_daemon(void) {

DIR\* dirp;

struct dirent\* ent;

pid\_t pid;

char path[BUF\_SIZE], buf[BUF\_SIZE];

FILE \*fp;

int dsize;

// /proc 디렉터리 open

if((dirp = opendir("/proc")) == NULL) {

fprintf(stderr, "opendir error\n");

exit(EXIT\_FAILURE);

}

while((ent = readdir(dirp)) != NULL) {

//프로세스id가 숫자가 아닌 경우 예외처리

if(!is\_number(ent->d\_name))

continue;

memset(path, 0, sizeof(buf));

snprintf(path, BUF\_SIZE, "/proc/%s/cmdline", ent->d\_name);

if((fp = fopen(path, "r")) != NULL) {

memset(buf, 0, sizeof(buf));

dsize = fread(buf, sizeof(char), BUF\_SIZE - 1, fp);

buf[dsize] = 0;

// /proc/pid/cmdline에 ./ssu\_ofm이 저장되어 있는 경우 디몬 프로세스가 실행 중이라는 의미

if(!strcmp(buf, "./ssu\_ofm")) {

fclose(fp);

closedir(dirp);

pid = atoi(ent->d\_name);

return pid;

}

fclose(fp);

}

}

closedir(dirp);

return 0;

}

//-t 옵션에서 파일의 마지막 수정시간과 현재 시간을 출력해주는 함수

void execute\_t\_option(void) {

struct stat statbuf;

struct tm \*cur\_timeinfo, \*mod\_timeinfo;

lstat(fname, &statbuf);

mod\_timeinfo = localtime(&statbuf.st\_mtime);

printf("##[Modification Time]##\n");

printf("Last Modification time of \'%s\': [%.24s]\n", fname, asctime(mod\_timeinfo));

cur\_timeinfo = get\_current\_time();

printf("Current time: [%.24s]\n", asctime(cur\_timeinfo));

sleep(1);

}

//현재 시간을 찾는 함수

struct tm \*get\_current\_time(void) {

time\_t cur\_time;

time(&cur\_time);

return localtime(&cur\_time);

}

//-t 옵션에서 파일이 수정되었는지 여부를 출력해주는 함수

void check\_modificaiton(time\_t before\_st\_mtime) {

printf("##[Modification Time]##\n");

if(!is\_modified(before\_st\_mtime))

printf("- There was no modificaition.\n");

else

printf("- There was modificaition.\n");

}

//파일 접근 전 후의 수정 시간을 비교하여 파일이 수정되었지 여부를 알려주는 함수

bool is\_modified(time\_t before\_st\_mtime) {

struct stat statbuf;

lstat(fname, &statbuf);

return before\_st\_mtime != statbuf.st\_mtime;

}

//ssu\_ofm으로부터 SIGUSR1 시그널을 전송받았을 때 실행되는 시그널 핸들러 함수

void SIGUSR1\_handler(int signo, siginfo\_t \*info, void \*context) {

pid\_t pid;

char \*argv[] = {"vim", fname, NULL};

SIGUSR1\_flag = true;

if((pid = fork()) < 0) {

fprintf(stderr, "fork error\n");

exit(EXIT\_FAILURE);

}

else if(pid > 0) {

wait(NULL);//자식 프로세스가 종료될 때까지 대기

kill(info->si\_pid, SIGUSR2);//vim 프로세스 종료 후 ssu\_ofm에 SIGUSR2 전송

}

else {//자식 프로세스에서 exec 함수를 호출하여 vim 프로세스 실행

execvp("vim", argv);

exit(EXIT\_SUCCESS);

}

}

//sigaction의 인자로 들어갈 sigaction 구조체 변수를 초기화하고 sigaction 함수를 호출하는 함수

void set\_signal(int signo, void (\*handler)(int, siginfo\_t \*, void \*)) {

struct sigaction act;

memset(&act, 0, sizeof(act));

act.sa\_sigaction = handler;

act.sa\_flags = SA\_SIGINFO;

sigemptyset(&act.sa\_mask);

sigaction(signo, &act, NULL);

}

//FIFO 파일에 write하는 함수

void send\_message(void) {

int fd;

char buf[BUF\_SIZE];

if((fd = open(FIFO\_FILE, O\_WRONLY)) < 0) {

fprintf(stderr, "open error\n");

exit(EXIT\_FAILURE);

}

strcpy(buf, fname);

write(fd, buf, strlen(buf));

close(fd);

}

//파일 크기를 구하는 함수

int get\_file\_size(char \*file) {

FILE \*fp = fopen(file, "r");

int size;

fseek(fp, 0L, SEEK\_END);

size = ftell(fp);

fclose(fp);

return size;

}

//-s 옵션에서 파일 접근 전, 후의 파일 크기를 출력하는 함수

void execute\_s\_option(int before\_size, int after\_size) {

printf("##[File Size]##\n");

printf("-- Before modification : %d(bytes)\n", before\_size);

printf("-- after modification : %d(bytes)\n", after\_size);

}

//old\_file 파일 데이터를 new\_file 파일 데이터로 복사하는 함수

void copy\_file(char \*new\_file, char \*old\_file) {

FILE \*nfp = fopen(new\_file, "w");

FILE \*ofp = fopen(old\_file, "r");

char buf[BUF\_SIZE];

memset(buf, 0, sizeof(buf));

fread(buf, sizeof(char), BUF\_SIZE, ofp);

fwrite(buf, sizeof(char), strlen(buf), nfp);

fclose(ofp);

fclose(nfp);

}

//old\_file 파일 데이터를 new\_file 파일 데이터로 복사하는 함수

void execute\_d\_option(char \*before\_file) {

pid\_t pid;

char \*argv[] = {"diff", before\_file, fname, NULL};

if((pid = fork()) < 0) {

fprintf(stderr, "fork error\n");

exit(EXIT\_FAILURE);

}

else if(pid > 0)

wait(NULL);//자식 프로세스가 종료할 때까지 대기

else {//자식 프로세스에서 exec을 호출해서 diff 명령어를 수행

printf("##[Compare with Previous File]##\n");

execvp("diff", argv);

exit(EXIT\_SUCCESS);

}

}

3) ssu\_ofm.h

#ifndef \_\_SSU\_OFM\_H\_\_

#define \_\_SSU\_OFM\_H\_\_

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <signal.h>

#include <syslog.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <signal.h>

#include <stdbool.h>

#include <ctype.h>

#include <string.h>

#include <time.h>

#include <pwd.h>

#include <dirent.h>

#include "queue.h"

//-l, -t, -n, -p, -id 옵션 플래그 값

#define OFM\_L\_FLAG 0x01

#define OFM\_T\_FLAG 0x02

#define OFM\_N\_FLAG 0x04

#define OFM\_P\_FLAG 0x08

#define OFM\_ID\_FLAG 0x10

#define BUF\_SIZE 1024

//FIFO 파일 이름

#define FIFO\_FILE "ssu\_fifo"

//log 파일 이름

#define LOG\_FILE "ssu\_log.txt"

//ssu\_ofm의 usage와 맞는지 검사하고 입력받은 옵션을 분류하는 작업을 수행하는 함수

void check\_option\_ofm(int argc, char \*argv[]);

//매개변수로 입력받은 문자열이 숫자인지 판별하는 함수

bool is\_number(char \*str);

//디렉터리 이름인지 판별하는 함수

bool is\_dir(char \*str);

//log 파일의 경로를 설정해주는 함수

void set\_log\_path(char \*path, char \*log, char \*dir);

//디몬 프로세스를 생성하는 함수

int ssu\_daemon\_init(void);

//디몬 프로세스가 생성되자마자 써야 할 내용을 log 파일에 write하는 함수

void write\_log\_init(void);

//현재 시간을 찾는 함수

char \*get\_current\_time();

//sigaction의 인자로 들어갈 sigaction 구조체 변수를 초기화하고 sigaction 함수를 호출하는 함수

void set\_signal(int signo, void (\*handler)(int, siginfo\_t \*, void \*));

//ssu\_vim으로부터 SIGUSR1 시그널을 전송받았을 때 실행되는 시그널 핸들러 함수

void SIGUSR1\_handler(int signo, siginfo\_t \*si, void \*context);

//FIFO 파일을 생성하는 함수

void make\_fifo(void);

//sender로부터 FIFO에 입력된 데이터를 read하는 함수

void shared\_file(char \*buf);

//ssu\_vim으로부터 SIGUSR1 시그널을 전송받았을 때 log 파일에 써야할 데이터를 write하는 함수

void write\_log\_requested(pid\_t pid, char \*shared);

//-id 옵션에서 SIGUSR2를 전송한 프로세스의 유저이름, UID, GID를 log 파일에 write할 준비를 함수

void execute\_id\_option(char \*buf);

//현재 공유 파일이 다른 프로세스에 의해 접근 중인지 확인하는 함수

bool is\_running();

//ssu\_vim으로부터 SIGUSR2 시그널을 전송받았을 때 실행되는 시그널 핸들러 함수

void SIGUSR2\_handler(int signo, siginfo\_t \*info, void \*context);

//ssu\_vim으로부터 SIGUSR2 시그널을 전송받았을 때 log 파일에 써야할 데이터를 write하는 함수

void write\_log\_finished(pid\_t pid);

void copy\_file(char \*new\_file, char \*old\_file);

#endif

4) ssu\_ofm.c

#include "ssu\_ofm.h"

//-l, -t, -n, -p, -id 옵션의 입력 여부를 판별하는 플래그, 비트연산

int ofm\_flag = 0;

//공유파일

char \*fname = NULL;

//-n 옵션의 인자 = queue의 최대 크기

char \*NUMBER = NULL;

//-p 옵션의 인자 = 디렉터리 이름

char \*DIRECTORY = NULL;

//프로세스id queue

Queue pid\_queue;

int main(int argc, char \*argv[]) {

//queue 초기화

init\_queue(&pid\_queue);

//queue 크기를 디폴트로 16 저장

pid\_queue.max\_size = 16;

//옵션 처리

check\_option\_ofm(argc, argv);

//FIFO 파일 생성

make\_fifo();

//SIGUSR1, SIGUSR2 시그널 등록

set\_signal(SIGUSR1, SIGUSR1\_handler);

set\_signal(SIGUSR2, SIGUSR2\_handler);

printf("Daemon Process Initialization.\n");

//디몬 프로세스 생성

if(ssu\_daemon\_init() < 0) {

fprintf(stderr, "ssu\_daemon\_init failed\n");

exit(EXIT\_FAILURE);

}

//디몬 프로세스 생성 후 log 파일에 써야할 내용 기록

write\_log\_init();

//디몬 프로세스가 SIGKILL 시그널을 받을 때까지 실행

while(1);

exit(EXIT\_SUCCESS);

}

//ssu\_ofm의 usage와 맞는지 검사하고 입력받은 옵션을 분류하는 작업을 수행하는 함수

void check\_option\_ofm(int argc, char \*argv[]) {

int i;

char \*opt;

if(argc < 2) {

fprintf(stderr, "USAGE: ssu\_ofm <FILENAME> [OPTION]\n");

exit(EXIT\_FAILURE);

}

//공유 파일 이름 저장

fname = argv[1];

//-l, -t, -n, -p, -id 옵션 중 입력받은 옵션을 플래그에 저장

for(i = 2; i < argc; i++) {

opt = argv[i];

if(!strcmp(opt, "-l"))

ofm\_flag |= OFM\_L\_FLAG;

else if(!strcmp(opt, "-t"))

ofm\_flag |= OFM\_T\_FLAG;

else if(!strcmp(opt, "-n")) {

if(i + 1 < argc && is\_number(argv[i + 1])) {

ofm\_flag |= OFM\_N\_FLAG;

NUMBER = argv[i + 1];//queue 크기 업데이트

pid\_queue.max\_size = atoi(NUMBER);

i++;

}

else {

fprintf(stderr, "Invalid option\n");

fprintf(stderr, "-n USAGE: -n <NUMBER>\n");

exit(EXIT\_FAILURE);

}

}

else if(!strcmp(opt, "-p")) {

ofm\_flag |= OFM\_P\_FLAG;

if(i + 1 < argc && is\_dir(argv[i + 1])) {

DIRECTORY = argv[i + 1];//디렉터리 이름 저장

i++;

}

else {

fprintf(stderr, "Invalid option\n");

fprintf(stderr, "-p USAGE: -p <DIRECTORY>\n");

exit(EXIT\_FAILURE);

}

}

else if(!strcmp(opt, "-id"))

ofm\_flag |= OFM\_ID\_FLAG;

else {

fprintf(stderr, "ERROR: Invalid option %s\n", opt);

exit(EXIT\_FAILURE);

}

}

}

//매개변수로 입력받은 문자열이 숫자인지 판별하는 함수

bool is\_number(char \*str) {

int i;

for(i = 0; i < strlen(str); i++) {

if(!isdigit(str[i]))

return false;

}

return true;

}

//디렉터리 이름인지 판별하는 함수

bool is\_dir(char \*str) {

if(\*str == '-')

return false;

return true;

}

//log 파일의 경로를 설정해주는 함수

void set\_log\_path(char \*path, char \*log, char \*dir) {

struct stat statbuf;

//-p 옵션을 입력받은 경우

if(ofm\_flag & OFM\_P\_FLAG) {

lstat(dir, &statbuf);

//dir 이름의 디렉터리가 존재하지 않는 경우

if(access(dir, F\_OK) || !S\_ISDIR(statbuf.st\_mode))

mkdir(dir, 0777);//디렉터리 생성

snprintf(path, BUF\_SIZE, "%s/%s", dir, log);

}

else

snprintf(path, BUF\_SIZE, "%s", log);

}

//디몬 프로세스를 생성하는 함수

int ssu\_daemon\_init(void) {

pid\_t pid;

int fd, maxfd;

if((pid = fork()) < 0) {

fprintf(stderr, "fork error\n");

exit(1);

}

else if(pid != 0)//백그라운드에서 수행되어야 한다.

exit(0);

pid = getpid();

setsid();//프로세스 그룹에서 탈퇴하고 새로운 프로세스 그룹을 생성한다.

//터미널 입출력 시그널을 무시한다.

signal(SIGTTIN, SIG\_IGN);

signal(SIGTTOU, SIG\_IGN);

signal(SIGTSTP, SIG\_IGN);

maxfd = getdtablesize();

//열려있는 모든 파일 디스크립터를 닫는다.

for(fd = 0; fd < maxfd; fd++)

close(fd);

//파일 모드 생성 마스크를 해제한다.

umask(0);

//현재 디렉터리를 루트 디렉터리로 설정한다.

/\* 본 설계과제에서는 사용자 레벨에서 디몬 프로세스가 수행되어야 하므로

\* 루트 디렉터리로 설정하는 부분은 생략한다.

\*/

//chdir("/");

//표준 입출력과 오류출력을 /dev/null로 리다이렉션한다.

fd = open("/dev/null", O\_RDWR);

dup(0);

dup(0);

return 0;

}

//디몬 프로세스가 생성되자마자 써야 할 내용을 log 파일에 write하는 함수

void write\_log\_init(void) {

FILE \*fp;

pid\_t pid;

char path[BUF\_SIZE], buf[BUF\_SIZE], queue\_size\_buf[BUF\_SIZE];

set\_log\_path(path, LOG\_FILE, DIRECTORY);

if((fp = fopen(LOG\_FILE, "w")) == NULL)

exit(EXIT\_FAILURE);

pid = getpid();

memset(buf, 0, sizeof(buf));

if(ofm\_flag & OFM\_T\_FLAG)//-t 옵션을 입력받아 시간을 추가

snprintf(buf, BUF\_SIZE, "[%.24s] <<Daemon Process Initialization with pid : %d>>\n", get\_current\_time(), pid);

else

snprintf(buf, BUF\_SIZE, "<<Daemon Process Initialization with pid : %d>>\n", pid);

memset(queue\_size\_buf, 0, sizeof(buf));

if(ofm\_flag & OFM\_N\_FLAG)//-n 옵션을 입력받아 queue의 최대 크기 변경

snprintf(queue\_size\_buf, BUF\_SIZE, "Initialized Queue Size : %s\n", NUMBER);

else

snprintf(queue\_size\_buf, BUF\_SIZE, "Initialized with Default Value : %d\n", 16);

strcat(buf, queue\_size\_buf);

fwrite(buf, sizeof(char), strlen(buf), fp);

fclose(fp);

fp = NULL;

}

//현재 시간을 찾는 함수

char \*get\_current\_time() {

time\_t cur\_time;

struct tm \*cur\_timeinfo;

time(&cur\_time);

cur\_timeinfo = localtime(&cur\_time);

return asctime(cur\_timeinfo);

}

//sigaction의 인자로 들어갈 sigaction 구조체 변수를 초기화하고 sigaction 함수를 호출하는 함수

void set\_signal(int signo, void (\*handler)(int, siginfo\_t \*, void \*)) {

struct sigaction act;

memset(&act, 0, sizeof(act));

act.sa\_sigaction = handler;

act.sa\_flags = SA\_SIGINFO;

sigemptyset(&act.sa\_mask);

sigaction(signo, &act, NULL);

}

//ssu\_vim으로부터 SIGUSR1 시그널을 전송받았을 때 실행되는 시그널 핸들러 함수

void SIGUSR1\_handler(int signo, siginfo\_t \*info, void \*context) {

//SIGUSR1을 보낸 프로세스의 id

pid\_t pid = info->si\_pid, next\_pid;

char buf[BUF\_SIZE];

memset(buf, 0, sizeof(buf));

shared\_file(buf);//ssu\_vim에서 접근을 요청하는 파일

//요청 상황을 log 파일에 기록

write\_log\_requested(pid, buf);

//공유 파일과 요청한 파일이 다르면

if(strcmp(fname, buf)) {

kill(pid, SIGKILL);//ssu\_vim에 SIGKILL 전송

return;

}

if(!is\_running()) {//공유 파일에 접근 중인 프로세스가 없는 경우

if(!empty(&pid\_queue)) {//queue가 비어 있지 않은 경우

next\_pid = dequeue(&pid\_queue);//queue에서 다음으로 실행할 pid를 가져옴

enqueue(&pid\_queue, pid);//가장 최근에 요청받은 프로세스를 queue에 저장

kill(next\_pid, SIGUSR1);//ssu\_vim에 SIGUSR1 전송

}

else

kill(pid, SIGUSR1);//ssu\_vim에 SIGUSR1 전송

}

else//공유 파일에 접근 중인 프로세스가 있는 경우

enqueue(&pid\_queue, pid);//가장 최근에 요청받은 프로세스를 queue에 저장

}

//FIFO 파일을 생성하는 함수

void make\_fifo(void) {

//FIFO 파일과 동일한 이름의 파일이 존재한다면

if(access(FIFO\_FILE, F\_OK) == 0)

remove(FIFO\_FILE);//해당 파일 삭제

//FIFO 파일 생성

if(mkfifo(FIFO\_FILE, 0666) < 0) {

fprintf(stderr, "mkfifo error\n");

exit(EXIT\_FAILURE);

}

}

//sender로부터 FIFO에 입력된 데이터를 read하는 함수

void shared\_file(char \*buf) {

int fd;

//FIFO 파일 open

if((fd = open(FIFO\_FILE, O\_RDWR)) < 0) {

fprintf(stderr, "open error\n");

exit(EXIT\_FAILURE);

}

//buf에 파일의 데이터를 read

read(fd, buf, BUF\_SIZE);

close(fd);

}

//ssu\_vim으로부터 SIGUSR1 시그널을 전송받았을 때 log 파일에 써야할 데이터를 write하는 함수

void write\_log\_requested(pid\_t pid, char \*shared) {

FILE \*fp;

char buf[BUF\_SIZE], id\_buf[BUF\_SIZE], path[BUF\_SIZE];

set\_log\_path(path, LOG\_FILE, DIRECTORY);

memset(buf, 0, sizeof(buf));

if(ofm\_flag & OFM\_T\_FLAG)//-t 옵션을 입력받아 시간을 추가

snprintf(buf, BUF\_SIZE, "[%.24s] Requested Process ID : %d, Requested Filename : %s\n", get\_current\_time(), pid, shared);

else

snprintf(buf, BUF\_SIZE, "Requested Process ID : %d, Requested Filename : %s\n", pid, shared);

memset(id\_buf, 0, sizeof(id\_buf));

if(ofm\_flag & OFM\_ID\_FLAG) {//-id 옵션을 입력받아 id 데이터 추가

execute\_id\_option(id\_buf);

strcat(buf, id\_buf);

}

if((fp = fopen(path, "a+")) == NULL) {

fprintf(stderr, "fopen error\n");

exit(EXIT\_FAILURE);

}

//log 파일에 write

fwrite(buf, sizeof(char), strlen(buf), fp);

fclose(fp);

}

//-id 옵션에서 SIGUSR2를 전송한 프로세스의 유저이름, UID, GID를 log 파일에 write할 준비를 함수

void execute\_id\_option(char \*buf) {

uid\_t uid = getuid();//유저 아이디

struct passwd \*pw\_info;

//uid를 이용하여 passwd 구조체 얻음

if((pw\_info = getpwuid(uid)) == NULL) {

fprintf(stderr, "getpwnam error\n");

exit(EXIT\_FAILURE);

}

//유저 이름 uid, gid 값 저장

snprintf(buf, BUF\_SIZE, "User : %s, UID : %d, GID : %d\n", pw\_info->pw\_name, pw\_info->pw\_uid, pw\_info->pw\_gid);

}

//현재 공유 파일이 다른 프로세스에 의해 접근 중인지 확인하는 함수

bool is\_running() {

DIR\* dirp;

struct dirent\* ent;

char path[BUF\_SIZE];

FILE \*fp;

char argv0[BUF\_SIZE], argv1[BUF\_SIZE];

int size;

// /proc 디렉터리 open

if((dirp = opendir("/proc")) == NULL) {

fprintf(stderr, "openddir error for /proc\n");

exit(EXIT\_FAILURE);

}

while((ent = readdir(dirp)) != NULL) {

if(!is\_number(ent->d\_name))

continue;

memset(path, 0, sizeof(path));

snprintf(path, BUF\_SIZE, "/proc/%s/cmdline", ent->d\_name);

if((fp = fopen(path, "r")) != NULL) {

size = fread(argv0, sizeof(char), BUF\_SIZE - 1, fp);

argv0[size] = 0;

if(size > 0) {//vim 프로세스가 실행 중인 경우

if(!strcmp(argv0, "vim")) {

//vim을 통해 open한 공유 파일도 같은 경우 => 실행

strcpy(argv1, argv0 + strlen(argv0) + 1);

argv1[strlen(argv1)] = 0;

if(!strcmp(argv1, fname)) {

fclose(fp);

closedir(dirp);

return true;

}

}

}

fclose(fp);

}

}

closedir(dirp);

return false;

}

//ssu\_vim으로부터 SIGUSR2 시그널을 전송받았을 때 실행되는 시그널 핸들러 함수

void SIGUSR2\_handler(int signo, siginfo\_t \*info, void \*context) {

pid\_t pid = info->si\_pid, next\_pid;

//SIGUSR2 시그널을 받았을 때 log 파일에 써야할 데이터 기록

write\_log\_finished(pid);

//queue가 비어 있는 경우

if(empty(&pid\_queue))

return;

//다음에 실행할 프로세스를 queue에서 가져옴

next\_pid = dequeue(&pid\_queue);

//ssu\_vim에 SIGUSR1 전송

kill(next\_pid, SIGUSR1);

}

//ssu\_vim으로부터 SIGUSR2 시그널을 전송받았을 때 log 파일에 써야할 데이터를 write하는 함수

void write\_log\_finished(pid\_t pid) {

FILE \*fp, \*fp2;

char \*cur\_time = NULL;

char buf[BUF\_SIZE], new\_file[BUF\_SIZE], tmp[BUF\_SIZE], path[BUF\_SIZE], data[BUF\_SIZE];

int len;

set\_log\_path(path, LOG\_FILE, DIRECTORY);

cur\_time = get\_current\_time();

memset(buf, 0, sizeof(buf));

if(ofm\_flag & OFM\_T\_FLAG) //-t 옵션을 입력받아 시간을 추가

snprintf(buf, BUF\_SIZE, "[%.24s] Finished Process ID : %d\n", cur\_time, pid);

else

snprintf(buf, BUF\_SIZE, "Finished Process ID : %d\n", pid);

if((fp = fopen(path, "a+")) == NULL) {

fprintf(stderr, "fopen error\n");

exit(EXIT\_FAILURE);

}

fwrite(buf, sizeof(char), strlen(buf), fp);

if(ofm\_flag & OFM\_L\_FLAG) {//-l 옵션을 입력받아 시간을 파일이름으로 하는 log 파일 생성

snprintf(tmp, BUF\_SIZE, "[%.24s]", cur\_time);

set\_log\_path(new\_file, tmp, DIRECTORY);

copy\_file(new\_file, path);

}

fclose(fp);

}

//old\_file 파일 데이터를 new\_file 파일 데이터로 복사하는 함수

void copy\_file(char \*new\_file, char \*old\_file) {

FILE \*nfp = fopen(new\_file, "w");

FILE \*ofp = fopen(old\_file, "r");

char buf[BUF\_SIZE];

memset(buf, 0, sizeof(buf));

fread(buf, sizeof(char), BUF\_SIZE, ofp);

fwrite(buf, sizeof(char), strlen(buf), nfp);

fclose(ofp);

fclose(nfp);

}

5) queue.h

#ifndef \_\_QUEUE\_H\_\_

#define \_\_QUEUE\_H\_\_

#include <stdlib.h>

#include <stdbool.h>

#include <sys/types.h>

#include <signal.h>

//Node 구조체

typedef struct \_node {

pid\_t pid;//프로세스 id

struct \_node \*next;//LinkedList로 구현하는 queue

} Node;

//LinkedList로 구현하는 queue

typedef struct \_queue {

//front : queue의 맨 앞 노드를 가리키는 포인터

//rear : queue의 맨 뒤 노드를 가리키는 포인터

Node \*front, \*rear;

//노드의 개수

int count;

//최대로 저장할 수 있는 노드의 개수

int max\_size;

} Queue;

//새로운 노드를 생성하는 함수

Node\* newNode(pid\_t pid);

//queue를 초기화하는 함수

void init\_queue(Queue \*q);

//queue의 rear에서 데이터를 추가하는 함수

void enqueue(Queue \*q, pid\_t pid);

//queue의 front에서 데이터를 제거하는 함수

pid\_t dequeue(Queue \*q);

//비어있는 queue인지 확인

bool empty(Queue \*q);

#endif

6) queue.c

#include "queue.h"

//새로운 노드를 생성하는 함수

Node\* newNode(pid\_t pid) {

Node \*node = (Node \*)malloc(sizeof(Node));

node->pid = pid;

node->next = NULL;

return node;

}

//queue를 초기화하는 함수

void init\_queue(Queue \*q) {

q = (Queue \*)malloc(sizeof(Queue));

q->front = NULL;

q->rear = NULL;

q->count = 0;

}

//queue의 rear에서 데이터를 추가하는 함수

void enqueue(Queue \*q, pid\_t pid) {

//최대 크기에 도달하면 해당 프로세스에게 SIGKILL 시그널 전송

if(q->max\_size <= q->count) {

kill(pid, SIGKILL);

return;

}

//새 노드 생성

Node \*temp = newNode(pid);

q->count++;

//처음으로 노드를 추가하는 경우

if (q->rear == NULL) {

q->front = temp;

q->rear = temp;

return;

}

//rear에 새 노드 추가

q->rear->next = temp;

q->rear = temp;

}

//queue의 front에서 데이터를 제거하는 함수

pid\_t dequeue(Queue \*q) {

pid\_t pid;

if (q->front == NULL)

return -1;

//front에서 노드 삭제

Node \*temp = q->front;

q->front = q->front->next;

q->count--;

//노드를 전부 삭제한 경우

if (q->front == NULL)

q->rear = NULL;

pid = temp->pid;

free(temp);

temp = NULL;

//삭제된 노드 pid 반환

return pid;

}

//비어있는 queue인지 확인

bool empty(Queue \*q) {

return (q->count == 0);

}

7) Makefile

CC := gcc

FILENAME1 := ssu\_vim

FILENAME2 := ssu\_ofm

OPT1 := -o $(FILENAME1) -W -Wall

OPT2 := -o $(FILENAME2) -W -Wall

all: ssu\_vim ssu\_ofm

ssu\_vim: ssu\_vim.o

$(CC) $(OPT1) ssu\_vim.o

ssu\_vim.o: ssu\_vim.c ssu\_vim.h

$(CC) -c ssu\_vim.c ssu\_vim.h

ssu\_ofm: ssu\_ofm.o queue.o

$(CC) $(OPT2) ssu\_ofm.o queue.o

ssu\_ofm.o: ssu\_ofm.c ssu\_ofm.h

$(CC) -c ssu\_ofm.c ssu\_ofm.h

queue: queue.c queue.h

$(CC) -c queue.c

clean:

rm -f ssu\_vim

rm -f ssu\_ofm

rm -f \*.o

rm -f \*.h.gch