

2022년 2학기 운영체제실습 14주차

# File System

**System Software Laboratory**

School of Computer and Information Engineering  
Kwangwoon Univ.

# Contents

- 파일 시스템의 개요
- 파일 시스템 관련 자료 구조
- proc 파일 시스템
- flexible array member (참고자료)
- 실습

# 파일 시스템의 개요

- 파일

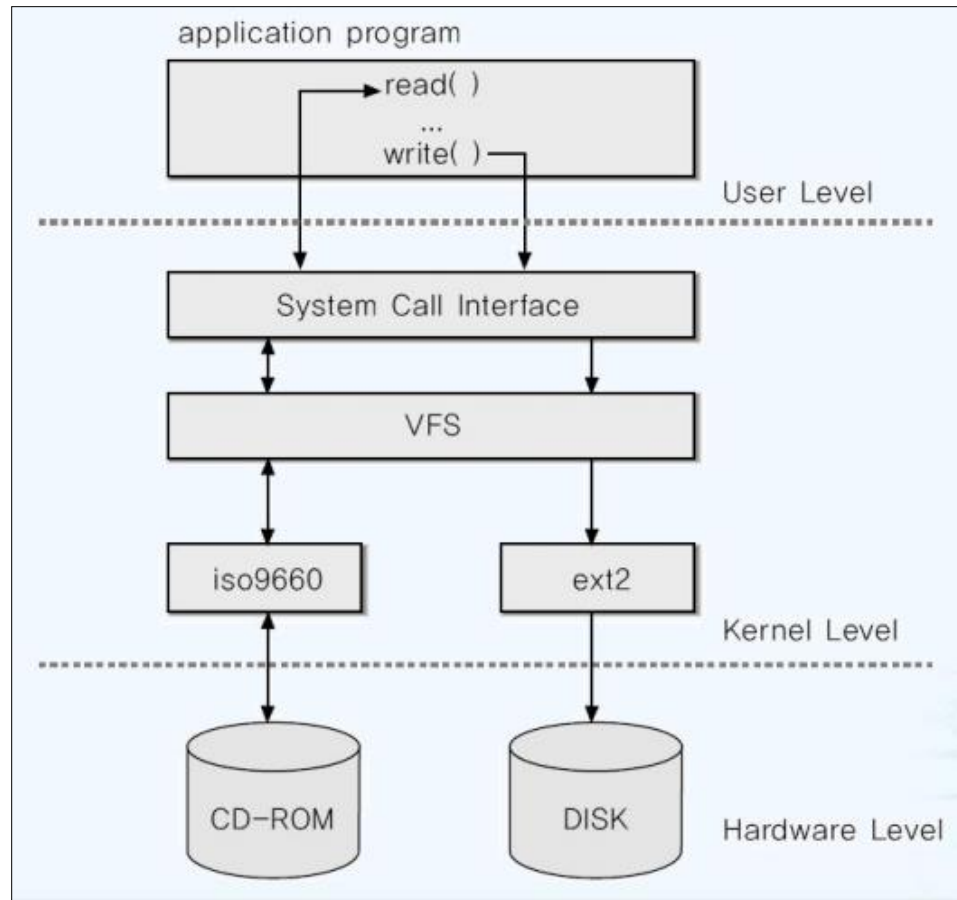
- 디스크와 같은 물리적인 저장 매체에 저장되는 프로그램
- 데이터 정보에 대한 논리적인 저장 단위

- 파일시스템

- 파일을 어떻게 관리할 것인가에 대한 정책
  - e.g. ext2, ext3, ext4, NTFS, ...
- 데이터를 특정 구조체에 담는 계층적인 저장소
- 네임스페이스(name space)라는 전역 계층 구조의 특정 지점에 마운트(mount)됨

# 파일 시스템의 개요

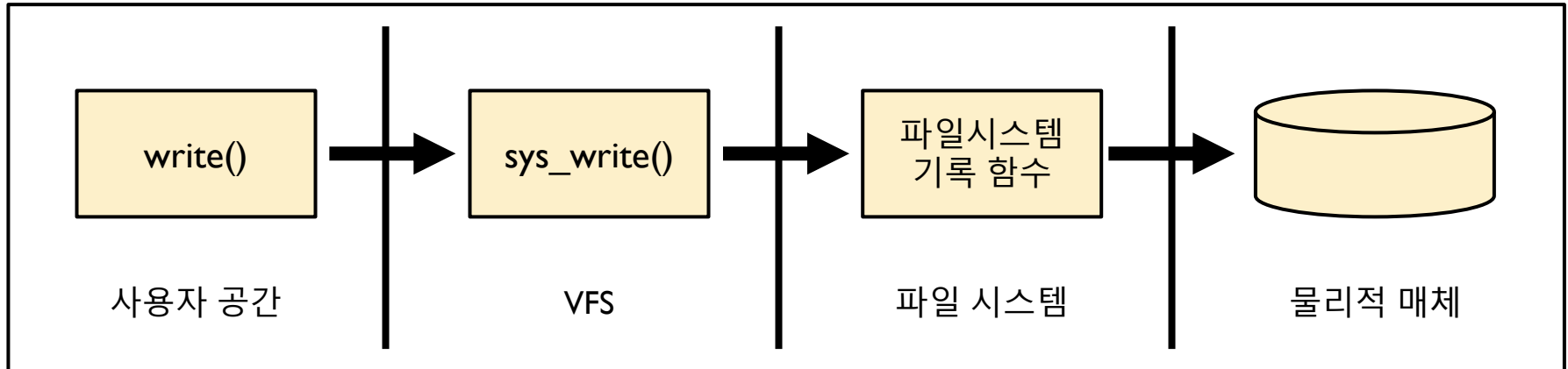
- 가상 파일 시스템 (VFS, Virtual File System)
  - 다양한 파일 시스템을 일관된 형태로 인식할 수 있도록 하는 기법



# 파일 시스템의 개요

- 가상 파일 시스템 (VFS, Virtual File System)

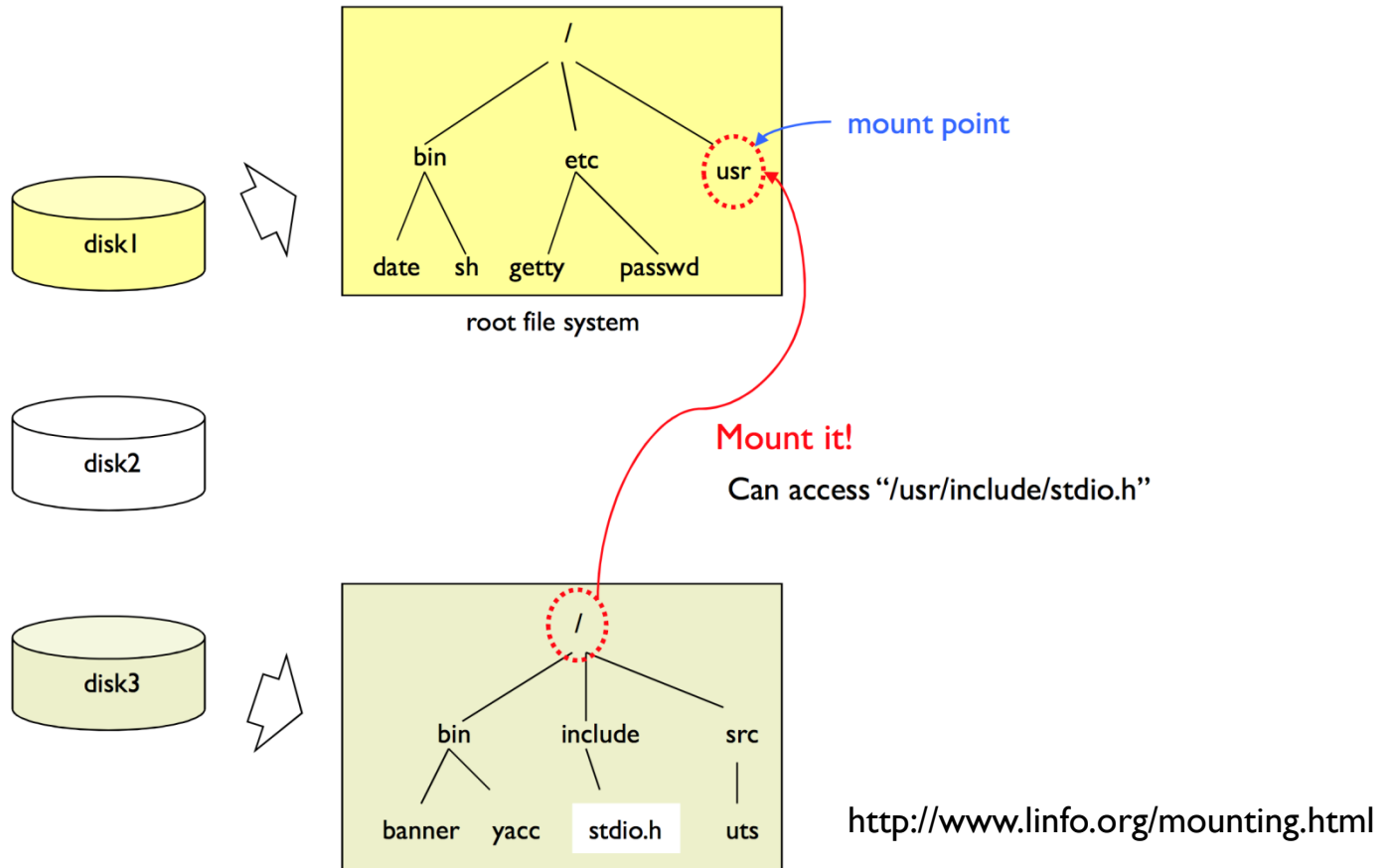
- 예) 물리적인 매체에 `write()`를 행한 경우
- VFS가 파일시스템의 일반적인 기능과 동작을 나타내는 공통 파일 모델 제공



# 파일 시스템의 개요

## ■ 마운트 (Mount)

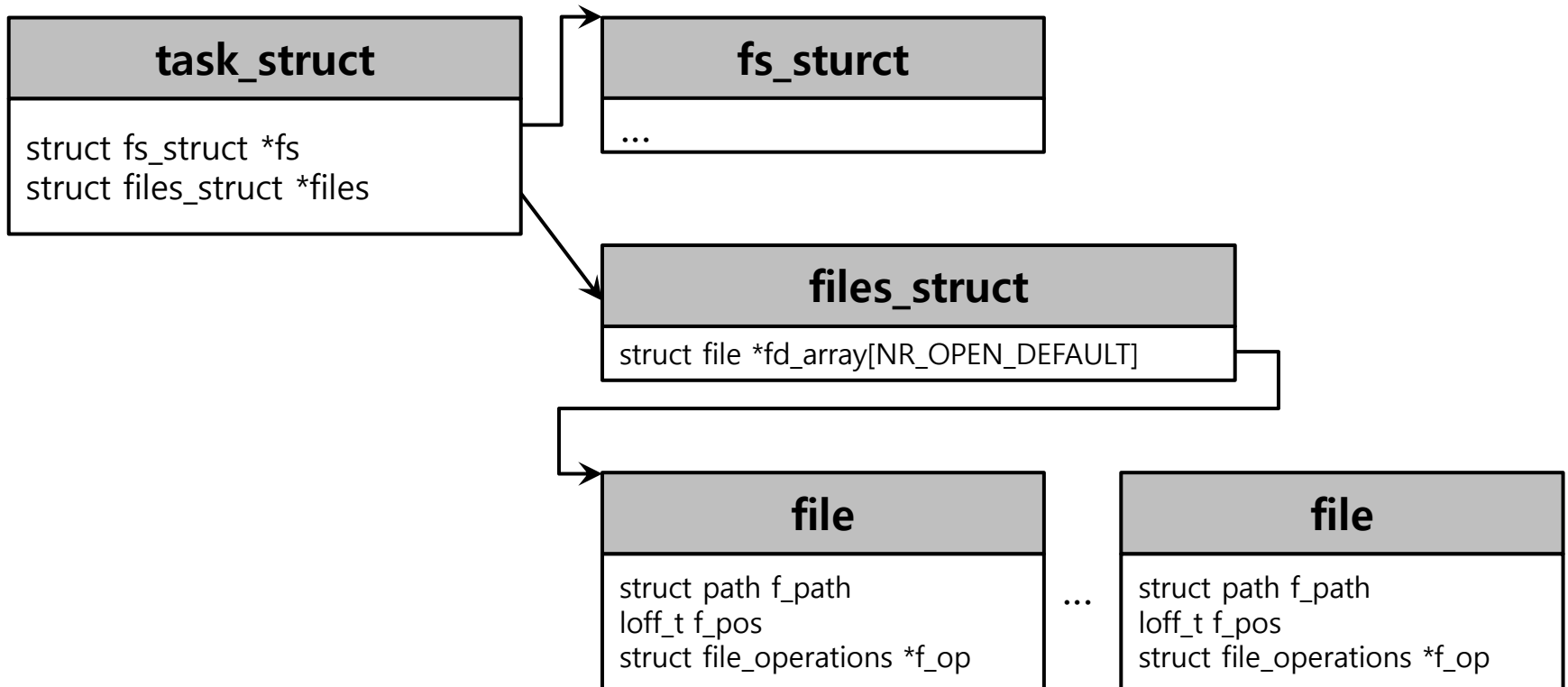
- 마운팅은 현재 접근 가능한 파일시스템에 추가적인 파일시스템을 붙이는 작업



# VFS 객체와 자료구조

- VFS는 객체지향적
  - VFS 객체의 4가지 유형
    - 슈퍼블록(super block) 객체
      - 파일시스템을 기술하는 정보를 저장
    - 아이노드(i-node) 객체
      - 커널이 파일이나 디렉토리를 관리하는데 필요한 모든 정보를 가짐
    - 덴트리(dentry) 객체
      - 파일을 포함한 경로 상의 모든 항목
    - 파일(file) 객체
      - 열린 파일은 메모리 상에 나타낸 것

# 파일 시스템 관련 자료 구조





# 파일 시스템 관련 자료 구조

- **struct task\_struct**

- struct fs\_struct \*fs
  - 현재 프로세스가 사용되고 있는 프로세스의 수, 루트 디렉토리, 현재 작업 디렉토리와 같은 정보를 담고 있음
- struct files\_struct \*files
  - 프로세스가 사용중인 파일과 파일 디스크립터에 대한 정보 등을 담음

- **struct files\_struct**

- 열려 있는 파일을 관리하기 위한 테이블
- struct file \*fd\_array[NR\_OPEN\_DEFAULT]

# 파일 시스템 관련 자료 구조

- **struct file**

- **f\_path**

- path 구조체
    - struct vfsmount \*mnt
      - 마운트 정보
    - struct dentry \*dentry
      - 이 구조체 내의 d\_inode 변수가 inode를 가리킴

- **f\_pos**

- 현재 파일에서 읽거나 쓸 위치를 나타냄
    - 파일이 처음 열리면 0으로 설정
    - 읽기 또는 쓰기 연산이 수행됨에 따라 위치 이동

- **f\_op**

- file\_operations 자료구조를 가리키는 포인터
      - read/write 등의 함수들이 어떤 작업을 수행해야 하는지 함수에 대한 포인터들로 구성

# proc 파일 시스템

- 저장장치 대신 메모리 상에 위치하는 파일 시스템
  - 커널이 직접 관리
  - 시스템의 정보나 커널 및 현재 실행 중인 태스크의 여러 정보 포함
  - e.g.
    - /proc/meminfo : 메모리 사용의 세부사항
    - /proc/cpuinfo : 현재 CPU의 정보들(클럭, 지원 명령어 등)
    - /proc/version : 커널 버전(\$ uname 을 통해 확인 가능)
    - /proc/devices : 설정되어 있는 장치의 목록
    - /proc/dma : DMA채널과 관련된 정보
    - /proc/filesystems : 설정된 파일시스템의 목록
    - /proc/kmsg : 커널이 출력하는 메시지(\$ dmesg 를 통해 확인가능)
    - /proc/modules : 현재 사용 중인 커널 모듈 목록 (\$ lsmod 를 통해 확인가능)
    - /proc/net : 네트워크 프로토콜들의 상태 정보
    - /proc/stat : 시스템의 상태에 관련된 정보
    - /proc/self : 현재 실행중인 프로세스에 관한 정보 (/proc/(PID)의 링크)

# proc 파일 시스템

- 저장장치 대신 메모리 상에 위치하는 파일 시스템

- e.g. (cont'd)

- /proc/PID (ex. /proc/1/)

- 해당 PID의 태스크의 관련 정보를 담고 있음
    - cmdline : 명령행 옵션
    - cwd : 작업 디렉토리 링크
    - exe : 태스크를 실행시킨 명령어 링크
    - fd : 파일 디스크립터 목록을 가지는 디렉토리
    - maps : 태스크의 메모리 맵
    - mem : 태스크가 사용 중인 메모리
    - root : 태스크의 루트 디렉토리
    - stat : 태스크의 상태
    - statm : 태스크의 메모리 상태
    - status : 태스크의 각종 정보 및 메모리 정보

```
sslab@ubuntu:~$ sudo ls /proc/1
attr      cpuset    limits    net        personality  smaps_rollup  timerslack_ns
autogroup  cwd       loginuid  ns          projid_map   stack          uid_map
auxv       environ   map_files numa_maps   root         stat          wchan
cgroup     exe       maps      oom_adj     sched        statm
clear_refs fd         mem       oom_score   schedstat    status
cmdline    fdinfo    mountinfo oom_score_adj sessionid     syscall
comm       gid_map   mounts    pagemap     setgroups    task
coredump_filter io         mountstats patch_state  smaps        timers
```

# proc 파일 시스템

- 디렉토리 생성 함수

- proc\_mkdir()

```
struct proc_dir_entry *proc_mkdir ( const char *name, struct proc_dir_entry *parent)
```

- name : 생성할 디렉토리 이름
    - parent : 디렉토리가 생성 될 부모 디렉토리 ( NULL일 때 /proc )

- proc\_dir\_entry 구조체

- 생성된 디렉토리에 관련된 정보를 가지는 구조체
  - include/linux/proc\_fs.h

- 디렉토리 생성 과정

- proc\_dir\_entry 구조체의 변수 선언 후 proc\_mkdir() 함수로 디렉토리 생성
  - e.g. /proc 하위에 exam 디렉토리 생성

```
struct proc_dir_entry *proc_dir = NULL;  
proc_dir = proc_mkdir("exam", NULL);
```

# proc 파일 시스템

- 파일 생성 함수

- proc\_mkdir()

```
struct proc_dir_entry *proc_create(const char *name, umode_t mode,  
                                   struct proc_dir_entry *parent,  
                                   const struct file_operations *proc_fops);
```

- name : proc 파일 이름
    - mode : 생성할 파일의 접근 권한
    - parent : 생성될 파일이 위치할 디렉토리 , NULL     proc 폴더에 생성
    - proc\_fops : 생성될 파일의 file operations

- e.g. exam 디렉토리 내에 file이라는 파일을 생성

```
static const struct file_operations my_proc_fops = {  
    ...  
};  
  
struct proc_dir_entry *proc_fp = NULL;  
proc_fp = create_proc_entry("file", 0644, proc_dir, &my_proc_fops );
```

# | proc 파일 시스템

- 파일 또는 디렉토리 제거 함수

- remove\_proc\_entry()

```
void remove_proc_entry (const char *name, struct proc_dir_entry *parent)
```

- name : proc 파일 이름
    - parent : 생성될 파일이 위치할 디렉토리
      - /proc 폴더에 생성할 시 NULL 삽입
  - e.g. 생성한 디렉토리 exam과 파일 file을 제거

```
struct proc_dir_entry *proc_fp = NULL;  
proc_fp = remove_proc_entry("file", proc_dir);
```

# proc 파일 시스템

- 읽기/쓰기 함수

- 해당 함수 구현 후 proc\_dir\_entry 구조체에 있는 read\_proc, write\_proc 함수 포인터에 대입
- e.g. 생성한 exam 디렉토리 내의 file에 데이터 읽고 쓰기

```
static const struct file_operations my_proc_fops = {  
    .owner = THIS_MODULE  
    .read = my_read_operations,  
    .write = my_write_operations,  
};
```

- 해당 함수의 원형

```
ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);  
ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
```

- struct file \*: current file
- char \_\_user \* : data for getting from user space
- size\_t : size of data for getting from user space
- loff\_t : offset when processing a file



# proc 파일 시스템

- 읽기 함수
  - 사용자 영역의 태스크로부터 요청이 들어오면 커널 메모리 영역에서 데이터를 읽어와 적정 포맷으로 변경한 후 태스크로 되돌려 줌
- 쓰기 함수
  - 사용자 영역의 태스크로부터 데이터를 받아 커널에게 넘겨줌
  - 통상적으로 `copy_from_user()` 함수를 이용
- proc 파일 시스템에서의 데이터는 커널 메모리 상에 저장

# 실습

- 모듈을 이용하여 proc 파일시스템에 "proc\_학번" 디렉토리(i.e. proc\_학번)를 생성하고, B파일을 생성해 보자.
  - "B" 파일
    - Permission : 소유자, 그룹, 일반 유저 모두가 읽기/쓰기 가능
    - Output : B파일의 맨 뒤에는 jiffies가 출력

# 실습

- 코드 (oslab\_proc.c)

```
1 #include <linux/syscalls.h>
2 #include <linux/proc_fs.h>
3 #include <linux/string.h> /* strlen() */
4 #include <linux/uaccess.h> /* copy_to_user(), copy_from_user() */
5 #include <linux/jiffies.h> /* get_jiffies_64() */
6
7 struct proc_dir_entry *oslab_fp_dir;
8 struct proc_dir_entry *oslab_fp_B;
9
10 char oslab_buffer[500];
11
12 int oslab_is_processed = 0;
13
14 int oslab_open(struct inode *inode, struct file *file)
15 {
16     oslab_is_processed = 0;
17     return 0;
18 }
```

- Line 7 : for "proc\_본인학번" proc directory
- Line 8 : for "proc\_본인학번 /B" proc file
- Line 10 : 사용자에게 입력 받은 데이터 저장
- Line 12 : 연산 종료 지시자
- oslab\_is\_processed : 연산이 종료되었는지 알려주기 위한 지시자
- Line 14 ~ 18 : "/proc/proc\_본인학번/B" 파일을 열 때 마다 수행
  - 즉, 해당 파일에 대해 open() 시스템 콜을 요청할 때 마다 수행

# 실습

## ■ 코드 (oslab\_proc.c)

```
20 ssize_t oslab_read(struct file *f, char __user *data_usr, size_t len, loff_t *off)
21 {
22     char buf[512] = {'\0', };
23     ssize_t len_written;
24
25     if(oslab_is_processed) return 0;
26
27     sprintf(buf, "%s%llu", oslab_buffer, get_jiffies_64());
28     len_written = strlen(buf)+1;
29     copy_to_user(data_usr, buf, len_written);
30
31     oslab_is_processed++;
32
33     return len_written;
34 }
35
```

- Line 20~34 : “/proc/proc\_본인학번/B” 파일에 읽기 연산을 요청할 때 마다 수행
  - 즉, 해당 파일에 대해 read() 시스템 콜을 요청할 때 마다 수행하는 연산
- Line 22 : jiffies 시간 값을 덧붙여서 호스트에게 반환할 데이터를 만들 임시 공간
- Line 27 : “buf”에 사용자가 입력한 데이터에 jiffies를 덧붙임
- Line 28 : 응용 프로그램에게 반환해주어야 하는 “읽은 데이터의 byte 크기”
- Line 29 : 사용자 영역(data\_usr)에 커널 내 데이터(buf)를 len\_written만큼 복사
- Line 25, 31 : 연산 종료 지시자

# 실습

- 코드 (oslab\_proc.c)

```
35  
36 ssize_t oslab_write(struct file *f, const char __user *data_usr, size_t len, loff_t *off)  
37 {  
38     ssize_t len_copied;  
39  
40     if(oslab_is_processed) return 0;  
41  
42     len_copied = len;  
43     copy_from_user(oslab_buffer, data_usr, len_copied);  
44  
45     oslab_is_processed++;  
46  
47     return len_copied;  
48 }  
49
```

- Line 36~48 : "/proc/proc\_본인학번/B" 파일에 쓰기 연산을 요청할 때 마다 수행
  - 즉, 해당 파일에 대해 write() 시스템 콜을 요청할 때 마다 수행하는 연산
- Line 42 : 응용 프로그램에게 반환해주어야 하는 "쓴 데이터의 byte 크기"
  - 응용 프로그램이 쓰도록 요청한 데이터 크기(len)를 모두 기록
- Line 29 : 사용자 영역 데이터를 커널 공간(oslab\_buffer)으로 len\_copied 만큼 복사
- Line 40, 45 : 연산 종료 지시자

# 실습

- 코드 (oslab\_proc.c)

```
49
50 const struct file_operations oslab_ops = {
51     .owner = THIS_MODULE,
52     .open = oslab_open,
53     .read = oslab_read,
54     .write = oslab_write,
55 };
56
57 int __init oslab_init(void)
58 {
59     oslab_fp_dir = proc_mkdir("proc_본인학번", NULL);
60     oslab_fp_B = proc_create("B", 권한, oslab_fp_dir, &oslab_ops);
61
62     return 0;
63 }
64
65 void __exit oslab_exit(void)
66 {
67     remove_proc_entry("B", oslab_fp_dir);
68     remove_proc_entry("proc_본인학번", NULL);
69 }
70
71 module_init(oslab_init);
72 module_exit(oslab_exit);
73 MODULE_LICENSE("GPL");
74
```

- Line 50~55 : "proc\_본인학번/B" 파일에 시스템 콜 요청 시 실제 호출되는 함수 지정
- Line 59 : 모듈 적재 시, /proc/proc\_본인학번 디렉토리 생성
- Line 60 : 모듈 적재 시, /proc/proc\_본인학번/B 파일 생성
- Line 67~68 : 모듈 해제 시, 파일 → 디렉토리 순서대로 제거 (순서에 유의)

# 실습

- 실행 결과 예시

- 컴파일

```
obj-m := oslab_proc.o

KDIR := /lib/modules/$(shell uname -r)/build
PWD := $(shell pwd)

all:
    $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules

clean:
    $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) clean
```

- 모듈 적재 및 해제

```
sslab@ubuntu:~/filesystem$ sudo insmod oslab_proc.ko
sslab@ubuntu:~/filesystem$ cat /proc/proc_OSLAB/B
4296041987sslab@ubuntu:~/filesystem$
sslab@ubuntu:~/filesystem$ echo "OSLAB!" > /proc/proc_OSLAB/B
sslab@ubuntu:~/filesystem$ cat /proc/proc_OSLAB/B
OSLAB!
4296049933sslab@ubuntu:~/filesystem$
sslab@ubuntu:~/filesystem$
sslab@ubuntu:~/filesystem$ sudo rmmod oslab_proc
sslab@ubuntu:~/filesystem$ cat /proc/proc_OSLAB/B
cat: /proc/proc_OSLAB/B: No such file or directory
sslab@ubuntu:~/filesystem$
```

# Assignment 5

- 제출 기한: 2022. 11.24(목) ~ 2022.12.9(금) 23:59:59
- Delay 없음
- 업로드 양식에 어긋날 경우 감점 처리
- Hardcopy 제출하지 않음