

# 과제 #2

## K-Board v2: over ProcFS

---

제출 기한: 6/16 (일) 23:59

제출 방법: KEDILMS “과제 #2”

# 질의응답 프로토콜

---

1. KEDILMS Q/A 게시판 확인
2. 인터넷 (구글) 검색
  1. 문제가 났을 때의 문장 혹은 키워드로 구글 검색
  2. 한글 자료는 많이 없으므로, 영어 사용 권장
3. KEDILMS Q/A 질문답변 게시판에 질문 올리기
  1. 가능한 자세한 정보를 제공할 것 (명령 수행 화면 등)
  2. 최대한 빠르게 답변하려고 노력 중이며,
  3. 질문 답변은 분반 간에 공유됨

# 과제 학습 목표 및 내용

---

- 학습 목표
  - 커널 모듈을 이용한 커널 개발 방법을 학습한다.
  - Proc FS를 이용한 유저-커널간 인터페이스를 학습한다.
  - 3가지 Reader-Writer 동기화 문제 해결 방법을 비교해본다.
- 내용
  - Proc FS를 이용한 K-Board v2 구현
    - 커널 모듈을 기반으로 ProcFS 구현
    - Proc FS를 기반으로 K-Board v2 구현
    - K-Board v2를 이용한 동기화 기법의 구현 및 비교
- 과제 총점수 30점 중 15점

# 과제 제출 내용

- 보고서와 소스코드를 압축해서 하나의 파일로 제출해야 함
- 보고서 (학번.pdf)
  - 제목, 학번, 이름, 캡처 화면 1개, 어려웠던 점 및 해결 방안, 소스코드
  - 유저 프로그램 결과 화면 캡처 1개 (분리 가능)
    - Copy, paste 를 연속해 실행한 결과 화면 캡처
    - Shared memory 동작 확인을 위해, 여러 차례 동작시킨 결과를 캡처
      - Full, empty 상태 확인을 포함하는, copy and paste 한 결과
  - 소스 코드는 수정한 부분, 핵심적인 부분만 포함할 것. 캡처 무방.
  - 캡처 화면, 소스 코드 외의 내용은 A4 3장을 절대 넘기지 말 것
  - (+) 동기화 관련 내용을 추가 수행한 경우, 해당 내용은 추가 3장 내에 작성
- 소스코드 (학번.zip)
  - 커널: 4개 파일 (수정 및 추가한 파일)
  - 유저: 3개 파일 (라이브러리, 소스 2개)

# 순서

---

1. Proc FS
2. Proc FS 모듈 예제 프로그램의 활용 및 이해
3. K-Board v2 구현
4. Reader-writer 동기화 문제 확인 및 해결



---

# Proc FS



# Proc File System

- /proc
  - 커널-유저 간 편리한 정보 전달을 위한 특수 파일 시스템
    - 파일 시스템 인터페이스를 이용해 커널 공간의 데이터를 접근
      - Open(), close(), read(), write(), ...
      - struct file\_operations 에 정의되어 있음
    - Pseudo file system
      - 일반 파일 시스템: 스토리지에 저장된 파일의 데이터에 접근
      - Pseudo file system: read(), write()는 편리한 인터페이스일 뿐, 어떤 동작을 하든 자유
        - Proc의 경우, 주로 커널의 데이터를 읽거나, 변경하는데 이용됨
        - CPU, 메모리 정보: /proc/cpuinfo, /proc/meminfo
        - 커널에 명령을 전달: echo 3 > /proc/sys/vm/drop\_caches
  - 참고
    - 리눅스 소스 디렉토리에서 Documentation/filesystems/proc.h
    - <https://www.kernel.org/doc/Documentation/filesystems/proc.txt>

# Struct file\_operations @ include/linux/fs.h

```
/ include / linux / fs.h
Search Identifier

1713 struct file_operations {
1714     struct module *owner;
1715     loff_t (*llseek) (struct file *, loff_t, int);
1716     ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
1717     ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
1718     ssize_t (*read_iter) (struct kiocb *, struct iov_iter *);
1719     ssize_t (*write_iter) (struct kiocb *, struct iov_iter *);
1720     int (*iterate) (struct file *, struct dir_context *);
1721     int (*iterate_shared) (struct file *, struct dir_context *);
1722     __poll_t (*poll) (struct file *, struct poll_table_struct *);
1723     long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
1724     long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
1725     int (*mmap) (struct file *, struct vm_area_struct *);
1726     unsigned long mmap_supported_flags;
1727     int (*open) (struct inode *, struct file *);
1728     int (*flush) (struct file *, fl_owner_t id);
1729     int (*release) (struct inode *, struct file *);
1730     int (*fsync) (struct file *, loff_t, loff_t, int datasync);
1731     int (*fasync) (int, struct file *, int);
1732     int (*lock) (struct file *, int, struct file_lock *);
1733     ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
1734     unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned long, unsigned long, unsigned long);
1735     int (*check_flags)(int);
1736     int (*flock) (struct file *, int, struct file_lock *);
1737     ssize_t (*splice_write)(struct pipe_inode_info *, struct file *, loff_t *, size_t, unsigned int);
1738     ssize_t (*splice_read)(struct file *, loff_t *, struct pipe_inode_info *, size_t, unsigned int);
1739     int (*setlease)(struct file *, long, struct file_lock **, void **);
1740     long (*fallocate)(struct file *file, int mode, loff_t offset,
1741                       loff_t len);
1742     void (*show_fdinfo)(struct seq_file *m, struct file *f);
1743 #ifdef CONFIG_MMU
1744     unsigned (*mmap_capabilities)(struct file *);
1745 #endif
1746     ssize_t (*copy_file_range)(struct file *, loff_t, struct file *,
1747                               loff_t, size_t, unsigned int);
1748     int (*clone_file_range)(struct file *, loff_t, struct file *, loff_t,
1749                             u64);
1750     ssize_t (*dedupe_file_range)(struct file *, u64, u64, struct file *,
1751                                 u64);
1752 } __randomize_layout;
```





# Proc File System

---

- 시스템콜과 비교하면?
  - Note.
    - 사실 시스템콜과 비교할 대상은 아님
    - ProcFS 는 read(), write() 와 같은 기존에 정의된 시스템콜을 활용하는 것
    - 따라서 성능 면에서는 거의 유사함
      - ProcFS 의 경우, VFS를 거치므로 아주 약간 더 경로가 길지만, 어차피 시스템콜 이라는 면에서 큰 차이 없음
  - 시스템콜
    - 새로운 시스템콜을 추가하기 위해 커널 수정, 컴파일, 설치, 재부팅 필요
  - ProcFS
    - 마찬가지로 커널 코드이므로 똑같은 과정이 필요하지만,
    - 모듈로 프로그래밍이 가능하므로 커널 수정을 하지 않아도 됨

# Useful Entries

---

- /proc/
  - meminfo
  - cpuinfo
  - partitions
  - kallsyms : 커널의 모든 함수 심볼 및 주소
  - devices : 현재 사용중인 장치 목록 (major and minor number)
  - filesystems : 현재 커널에 등록된 FS 목록
  - diskstats : 부팅 후 각 블록 장치에 이루어진 I/O 요청 정보

# Usage and example

---

- get : using Read() system call
  - E.g.) cat /proc/<filename>
- put : using Write() system call
  - E.g.) echo <data> > /proc/<filename>
- example
  - /proc/sys/kernel/pid\_max
    - 동시에 수행 가능한 최대 프로세스 개수

```
root@ostavbox:/# cat /proc/sys/kernel/pid_max
32768
root@ostavbox:/# echo 65535 > /proc/sys/kernel/pid_max
root@ostavbox:/# cat /proc/sys/kernel/pid_max
65535
root@ostavbox:/#
```

# Useful Entries

---

- /proc/<pid> : per-process information
  - cmdline: 실행 명령어 (프로그램 파일의 경로 및 이름, 실행 파라미터 등)
  - maps: memory map 정보 (mmap() 을 통해 공유중인 파일 리스트 및 영역)
  - fd/ : 사용 중인 파일 정보 (open())을 통해 열린 파일들)

# Useful Entries

---

- Example : cat /proc/filesystems

```
root@ubuntu:/home/m# cat /proc/filesystems
nodev    sysfs
nodev    rootfs
nodev    ramfs
nodev    bdev
nodev    proc
nodev    cpuset
nodev    cgroup
nodev    tmpfs
nodev    devtmpfs
nodev    debugfs
nodev    tracefs
nodev    securityfs
nodev    sockfs
nodev    bpf
nodev    pipefs
nodev    devpts
        ext3
        ext2
        ext4
        squashfs
nodev    hugetlbfs
        vfat
nodev    ecryptfs
        fuseblk
nodev    fuse
nodev    fusectl
nodev    pstore
nodev    mqueue
nodev    autofs
        xfs
        jfs
        msdos
        ntfs
        minix
        hfs
        hfsplus
        qnx4
        ufs
        btrfs
```



# Proc FS 사용 방법

---

- 생성 및 제거

- `struct proc_dir_entry *proc_mkdir(const char *, struct proc_dir_entry *)`;
  - /proc 아래에 새로운 디렉토리 생성
- `struct proc_dir_entry *proc_create(const char *name, umode_t mode, struct proc_dir_entry *parent, const struct file_operations *proc_fops)`
  - Parent 아래에 새로운 파일을 생성하고, 해당 파일에 대해 `proc_fops` 로 파일 오퍼레이션 연결
- `int remove_proc_subtree(const char *name, struct proc_dir_entry *parent)`;
  - Parent 이하 모든 파일들 제거 (파일의 삭제만)
- `void proc_remove(struct proc_dir_entry *de)`;
  - Proc directory entry 제거 (파일, 디렉토리와 연결된 커널 자료 구조들)

# Struct proc\_dir\_entry @ fs/proc/internal.h

```
/ fs / proc / internal.h
34 struct proc_dir_entry {
35     /*
36      * number of callers into module in progress;
37      * negative -> it's going away RSN
38      */
39     atomic_t in_use;
40     refcount_t refcnt;
41     struct list_head pde_openers; /* who did ->open, but not ->release */
42     /* protects ->pde_openers and all struct pde_opener instances */
43     spinlock_t pde_unload_lock;
44     struct completion *pde_unload_completion;
45     const struct inode_operations *proc_ops;
46     const struct file_operations *proc_fops;
47     union {
48         const struct seq_operations *seq_ops;
49         int (*single_show)(struct seq_file *, void *);
50     };
51     proc_write_t write;
52     void *data;
53     unsigned int state_size;
54     unsigned int low_ino;
55     nlink_t nlink;
56     kuid_t uid;
57     kgid_t gid;
58     loff_t size;
59     struct proc_dir_entry *parent;
60     struct rb_root subdir;
61     struct rb_node subdir_node;
62     char *name;
63     umode_t mode;
64     u8 namelen;
65     char inline_name[];
66 } __randomize_layout;
```



# Proc FS 사용 방법

- File operation 연결
  - Struct file\_operations 의 정의
    - 사용할 operations 들에 대해 실제 서비스 함수들을 정의함
  - 해당 파일에 대해 해당 operation 이 호출 시,  
해당 서비스 함수가 수행됨

```
static const struct file_operations foo_proc_ops = {  
    .owner      = THIS_MODULE,  
    .open       = foo_proc_open,  
    .read       = seq_read,  
    .llseek     = seq_lseek,  
    .release    = seq_release,  
};
```

```
static int foo_proc_open(struct inode *inode, struct file *file)  
{  
    return single_open(file, foo_simple_show, NULL);  
}
```





---

# Proc FS 모듈 예제 프로그램의 활용 및 이해



# LKM: Loadable Kernel Module

- 커널이 수행되고 있는 중에, 커널 코드를 추가할 수 있는 방법
  - 커널 코드를 고칠 때마다 커널 컴파일을 할 필요가 없음!

- 관련 명령

- insmod: 모듈 삽입 (로드 or 탑재)
  - insmod mod\_proc.ko (모듈 파일명 입력)
- rmmod: 모듈 제거
  - rmmod mod\_proc (이때는 모듈 이름만)
- lsmod: 현재 삽입된 모듈 리스트 확인

```
root@hcpark:~/proj2/module# dmesg -c
root@hcpark:~/proj2/module# rmmod mod_proc
rmmod: ERROR: Module mod_proc is not currently loaded
root@hcpark:~/proj2/module# insmod mod_proc.ko
root@hcpark:~/proj2/module# dmesg -c
[ 8087.983642] add_data 10, 20
[ 8087.983645] add_data 30, 40
[ 8087.983659] Created /proc/foo-dir/foo
root@hcpark:~/proj2/module# lsmod | grep mod_proc
mod_proc                16384  0
root@hcpark:~/proj2/module# rmmod mod_proc
root@hcpark:~/proj2/module# dmesg -c
[ 8097.946975] Removed /proc/foo-dir/foo
root@hcpark:~/proj2/module#
```

- 모듈의 삽입, 제거는 printk() 를 이용해 출력 후, dmesg로 확인

# LKM: Loadable Kernel Module

- 모듈의 기본 형태

```
#include <linux/module.h>
#include <linux/init.h>

MODULE_LICENSE( "GPL" );
MODULE_AUTHOR( "Module Author" );
MODULE_DESCRIPTION( "Module Description" );

static int __init mod_entry_func( void )
{
    return 0;
}

static void __exit mod_exit_func( void )
{
    return;
}

module_init( mod_entry_func );
module_exit( mod_exit_func );
```

Module  
macros

Module  
constructor /  
destructor

Entry / exit  
macros



# mod\_proc.c

---

- 제공하는 모듈: mod\_proc.c
  - ProcFS의 등록, 해제 및 간단한 기능들의 예제 포함
    - 모듈이 등록될 때
      - /proc/foo-dir 디렉토리 및 /proc/foo-dir/foo 파일 생성
      - 두 개의 샘플 foo\_info 구조체를 커널 메모리를 할당받아 작성하고, list로 유지함
    - 모듈이 해제될 때는 위의 두 파일과 관련 내용을 삭제함
    - foo 파일을 읽으면, list 를 순회하며 foo\_info 구조체의 내용을 출력함
- 모듈 코드 참고 사이트
  - <http://jake.dothome.co.kr/proc/>
    - 자세한 설명도 있으니 꼭 한번 읽어볼 것.
    - (감사합니다...)

# LKM: 커널 모듈 작성

- 작업 디렉토리

- 커널 소스와 무관한 디렉토리를 따로 만들어서 사용

(Makefile 내용)

- 모듈 컴파일 방법

- Makefile 작성 →
  - “make” 수행

- 결과물

- .ko 파일

```
obj-m += mod_ssm.o
```

```
all:
```

```
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
```

```
clean:
```

```
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

```
root@hcpark:~/proj2/module# make clean; make
make -C /lib/modules/4.18.20-hcpark/build M=/root/proj2/module clean
make[1]: Entering directory '/usr/src/linux-source-4.18.0/linux-source-4.18.0'
  CLEAN   /root/proj2/module/.tmp_versions
  CLEAN   /root/proj2/module/Module.symvers
make[1]: Leaving directory '/usr/src/linux-source-4.18.0/linux-source-4.18.0'
make -C /lib/modules/4.18.20-hcpark/build M=/root/proj2/module modules
make[1]: Entering directory '/usr/src/linux-source-4.18.0/linux-source-4.18.0'
  CC [M]   /root/proj2/module/mod_proc.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC       /root/proj2/module/mod_proc.mod.o
  LD [M]   /root/proj2/module/mod_proc.ko
make[1]: Leaving directory '/usr/src/linux-source-4.18.0/linux-source-4.18.0'
root@hcpark:~/proj2/module#
```



# 예제 프로그램 수행

- 모듈 컴파일 → 등록 → read() 수행 → 모듈 해제
- 코드를 잘 분석해서 사용 방법을 익힐 것

```
root@hpcpark:~/proj2/module# dmesg -c
[14248.617208] Removed /proc/foo-dir/foo
root@hpcpark:~/proj2/module# dmesg -c
root@hpcpark:~/proj2/module# rmmod mod_proc; make clean; make && insmod mod_proc.ko
rmmod: ERROR: Module mod_proc is not currently loaded
make -C /lib/modules/4.18.20-hcpark/build M=/root/proj2/module clean
make[1]: Entering directory '/usr/src/linux-source-4.18.0/linux-source-4.18.0'
  CLEAN   /root/proj2/module/.tmp_versions
  CLEAN   /root/proj2/module/Module.symvers
make[1]: Leaving directory '/usr/src/linux-source-4.18.0/linux-source-4.18.0'
make -C /lib/modules/4.18.20-hcpark/build M=/root/proj2/module modules
make[1]: Entering directory '/usr/src/linux-source-4.18.0/linux-source-4.18.0'
  CC [M]  /root/proj2/module/mod_proc.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /root/proj2/module/mod_proc.mod.o
  LD [M]  /root/proj2/module/mod_proc.ko
make[1]: Leaving directory '/usr/src/linux-source-4.18.0/linux-source-4.18.0'
root@hpcpark:~/proj2/module# dmesg -c
[14267.141422] add_data 10, 20
[14267.141425] add_data 30, 40
[14267.141436] Created /proc/foo-dir/foo
root@hpcpark:~/proj2/module# cat /proc/foo-dir/foo
30 + 40 = 70
10 + 20 = 30
root@hpcpark:~/proj2/module# rmmod mod_proc
root@hpcpark:~/proj2/module# dmesg -c
[14324.987718] Removed /proc/foo-dir/foo
root@hpcpark:~/proj2/module#
```

---

# K-Board v2 구현



# K-Board v2 작성

---

- 구현 내용

- os\_kboard.c 에서 구현한 내용을 모듈로 이전
  - 예제 프로그램을 참고하여, 필요한 부분 수정, 확장하여 진행
- Enqueue, dequeue 인터페이스를 Proc FS의 write(), read()로 변경
  - /proc/kboard/writer
- Queue 내용에 대해 Read 만 수행할 수 있는 인터페이스 제공
  - /proc/kboard/reader
- Count 값과 버퍼 상태를 볼 수 있는 인터페이스도 구현
  - /proc/kboard/count, /proc/kboard/dump
- 참고
  - K-Board의 초기화는 모듈을 내리고 새로 로드할 때 수행 가능함



---

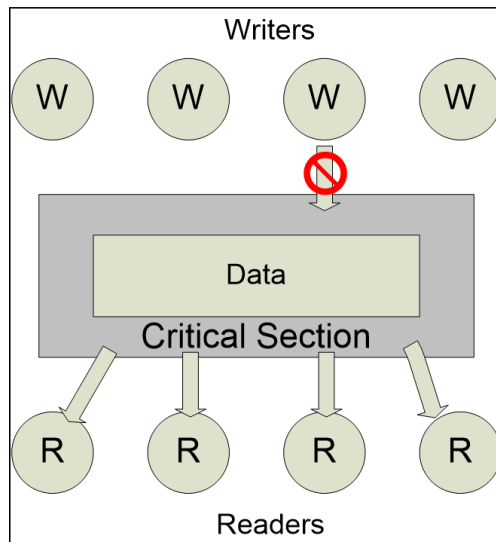
# Reader-writer 동기화 문제

## 확인 및 해결

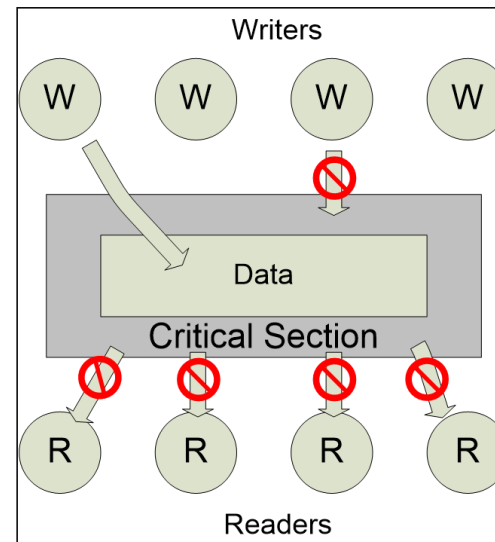


# Readers-Writers Problem

- 여러 Readers와 Writers가 하나의 공유 데이터를 읽거나 쓰기 위해 접근
- Readers - 공유 데이터를 읽는다.
  - 여러 Reader는 동시에 데이터를 읽을 수 있다.
- Writers – 공유 데이터에 쓴다.
  - Writer가 데이터를 수정하기 위해서는, reader혹은 다른 writer가 작업을 하고 있지 말아야 한다.



<여러 reader는 동시에 read가능>



<하나의 writer만 write가능>

# Readers-Writers on K-Board

- Readers-writers problem 에 대한 3가지 solution 을 비교 분석함
  - Readers-Writers on K-Board
    - Readers: Queue 의 임의 위치에 대해 읽기만을 진행함
    - Writers: Queue 에 대해 enqueue, dequeue를 수행함
  - 과제 수행 내용
    - 3가지 솔루션의 구현
      - Spinlock 과 semaphore 사용 (kernel/locking/spinlock.c, semaphore.c)
    - Readers 와 writers를 병렬 수행함
      - 프로세스를 사용하여도 되고,
      - **쓰레드를 사용하는 경우 추가 점수 2점 부여 (pthread 이용)**
    - 분석: 성능 분석을 통한 Starvation 문제의 확인
      - Performance: 단위 시간 당 읽고, 쓰는 데이터(item)의 횟수
      - Issue: 어떻게 여러 프로세스/쓰레드의 성능 결과를 합산할 것인가?

# Semaphore in Linux kernel

- Semaphore definition and its init code

```
/ include / linux / semaphore.h
15  /* Please don't access any members of this structure directly */
16  struct semaphore {
17      raw_spinlock_t    lock;
18      unsigned int      count;
19      struct list_head  wait_list;
20  };
21
32  static inline void sema_init(struct semaphore *sem, int val)
33  {
34      static struct lock_class_key __key;
35      *sem = (struct semaphore) __SEMAPHORE_INITIALIZER(*sem, val);
36      lockdep_init_map(&sem->lock.dep_map, "semaphore->lock", &__key, 0);
37  }
```

- Wait(S) and Signal (S) = down(sem) and up(sem)

```
/ kernel / locking / semaphore.c
43  /**
44   * down - acquire the semaphore
45   * @sem: the semaphore to be acquired
46   *
47   * Acquires the semaphore. If no more tasks are allowed to acquire the
48   * semaphore, calling this function will put the task to sleep until the
49   * semaphore is released.
50   *
51   * Use of this function is deprecated, please use down_interruptible() or
52   * down_killable() instead.
53   */
54  void down(struct semaphore *sem)
55  {
56      unsigned long flags;
57
58      raw_spin_lock_irqsave(&sem->lock, flags);
59      if (likely(sem->count > 0))
60          sem->count--;
61      else
62          __down(sem);
63      raw_spin_unlock_irqrestore(&sem->lock, flags);
64  }
65  EXPORT_SYMBOL(down);
66

/ kernel / locking / semaphore.c
172  /**
173   * up - release the semaphore
174   * @sem: the semaphore to release
175   *
176   * Release the semaphore. Unlike mutexes, up() may be called from any
177   * context and even by tasks which have never called down().
178   */
179  void up(struct semaphore *sem)
180  {
181      unsigned long flags;
182
183      raw_spin_lock_irqsave(&sem->lock, flags);
184      if (likely(list_empty(&sem->wait_list)))
185          sem->count++;
186      else
187          __up(sem);
188      raw_spin_unlock_irqrestore(&sem->lock, flags);
189  }
190  EXPORT_SYMBOL(up);
```

# 참고자료

---

- Google search
  - 가급적 최근 문서들을 참고하고, 버전 정보를 꼭 확인할 것.  
리눅스 커널은 버전에 따라 내용이 많이 달라짐
- Documentation: In your kernel source tree
  - Documentation 디렉토리 내에 문서들이 잔~~뜩 있음
- Web-based Cross Reference
  - <https://elixir.bootlin.com/linux/v4.18/source>