디바이스 드라이버

Suntae Hwang

Kookmin University

디바이스 드라이버 개요

디바이스 드라이버 개요

- 물리적인 하드웨어 장치를 다루고 관리하는 소프트웨어
- 커널의 일부분
- 주번호(major number) 와 부번호(minor number)를 이용하여 각각의 디바이스들을 구분하여 사용

디바이스 드라이버의 용도

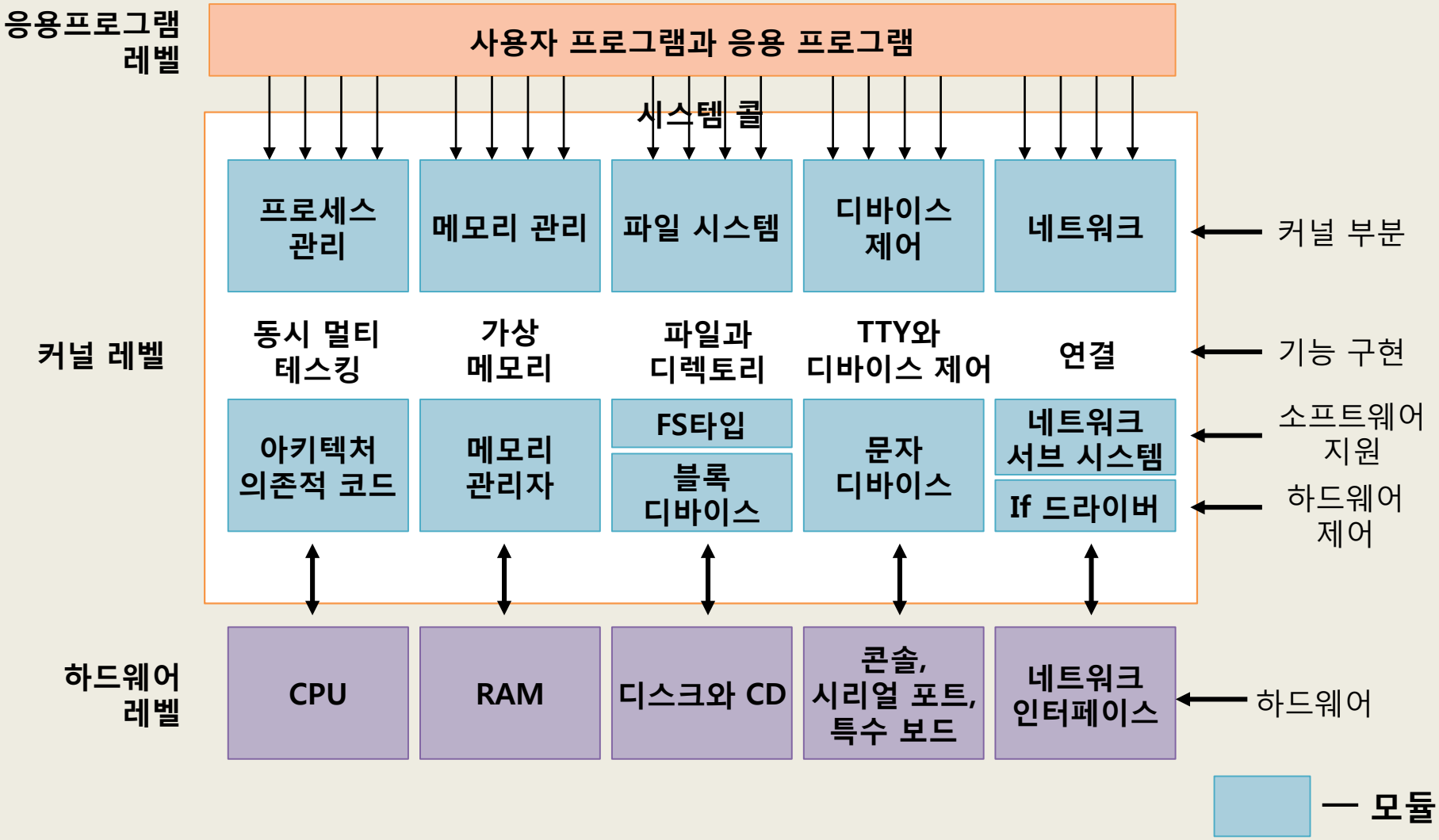
- 응용프로그램에서 하드웨어장치를 이용해서 데이터를 직접 읽고 쓰거나 제어해야 하는 경우에 디바이스 드라이버를 이용

디바이스 드라이버 종류 및 특징

⊙ 디바이스 드라이버의 종류 및 특징

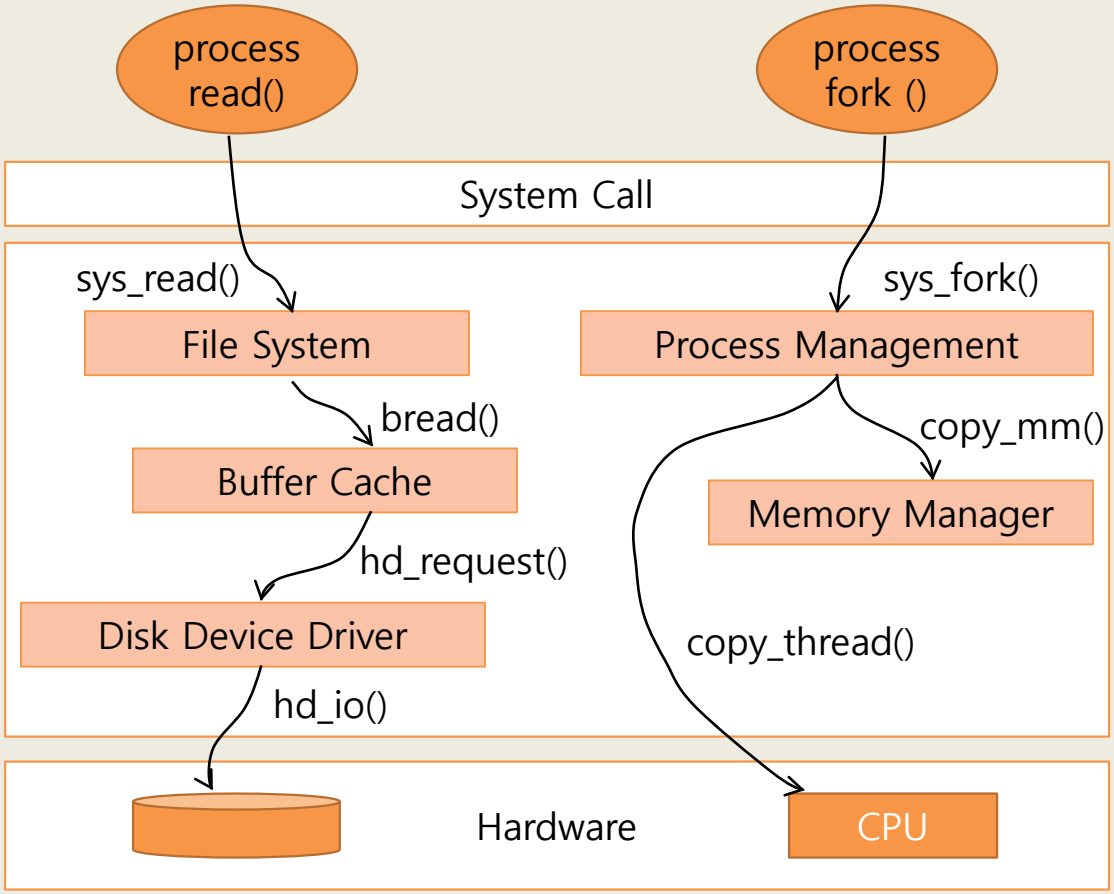
드라이버 종류	설 명	등록함수명
문자 드라이버	디바이스를 파일처럼 취급하고 접근하여 직접 읽기/쓰기를 수행, 데이터 형태는 스트림 방식으로 전송 EX) 콘솔, 키보드, 시리얼 포트 드라이버등	register_chrdev()
블록 드라이버	디스크와 같이 파일 시스템을 기반으로 일정한 블록 단위로 데이터 읽기/쓰기를 수행 EX) 플로피 디스크, 하드 디스크, CDROM 드라이버 등	register_blkdev()
네트워크 드라이버	네트워크의 물리 계층과 프레임 단위의 데이터를 송수신 EX) 이더넷 디바이스 드라이버(eth0)	register_netdev()

디바이스 드라이버 종류 및 특징



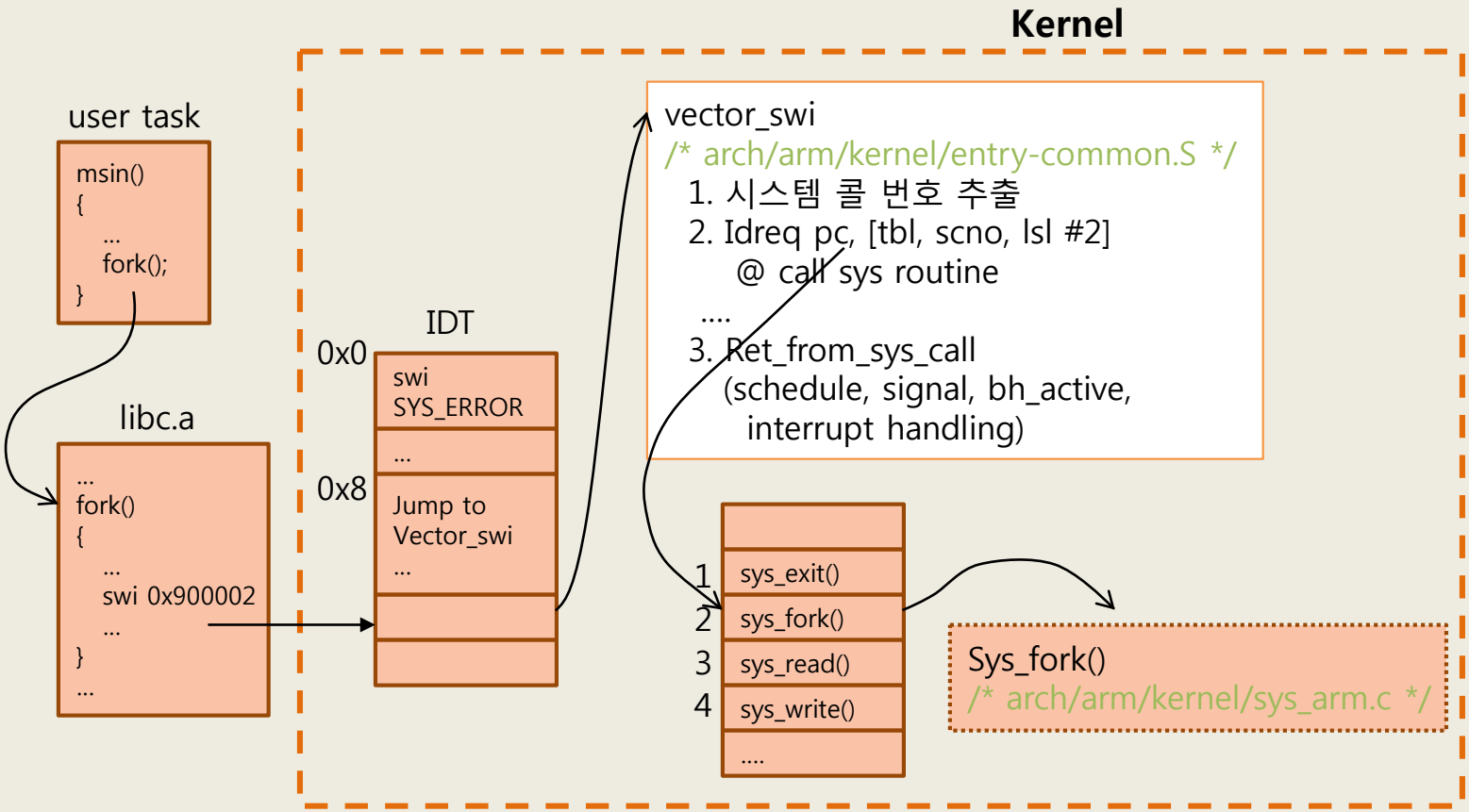
디바이스 드라이버 동작과 시스템 콜

시스템 콜 흐름도

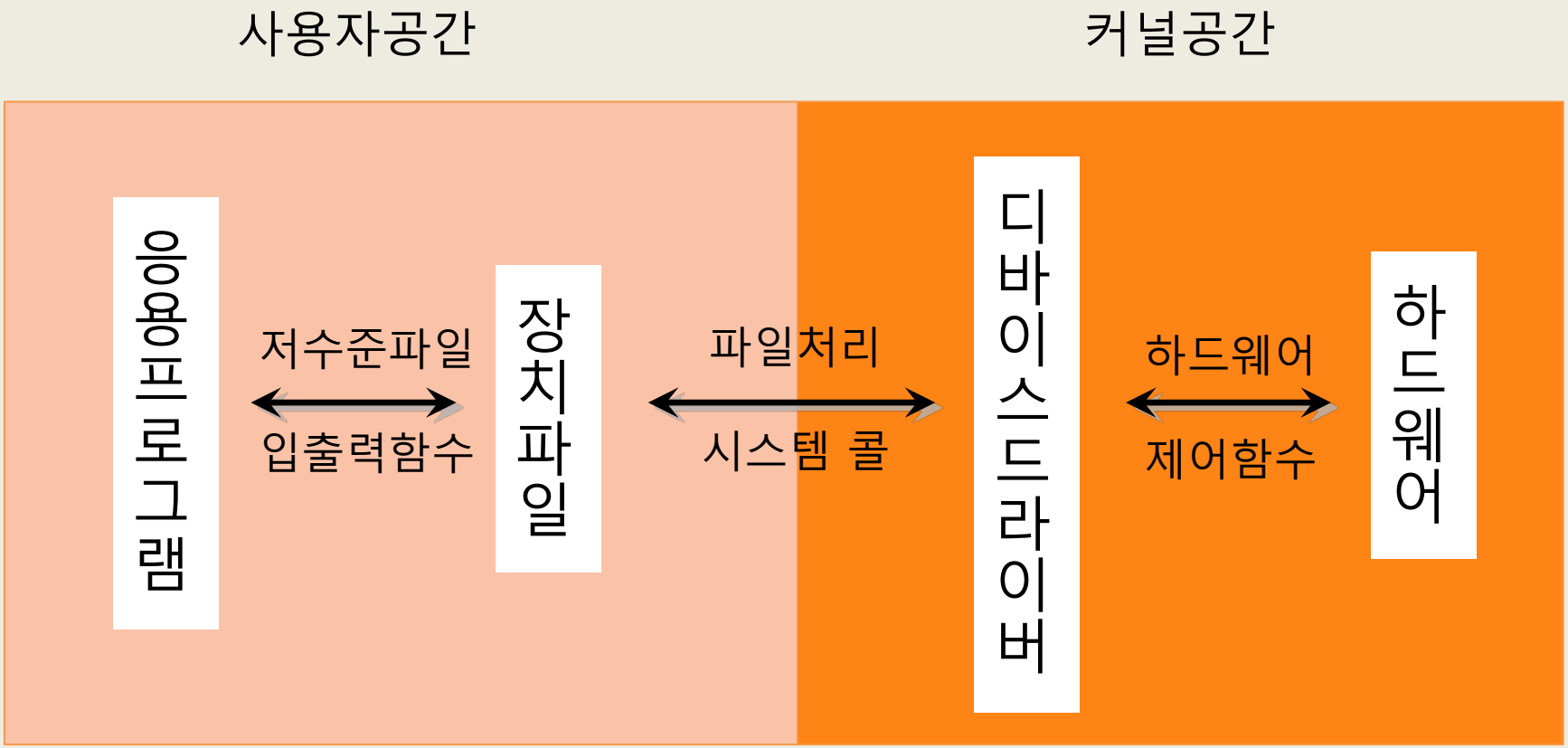


시스템 콜의 흐름 예: fork

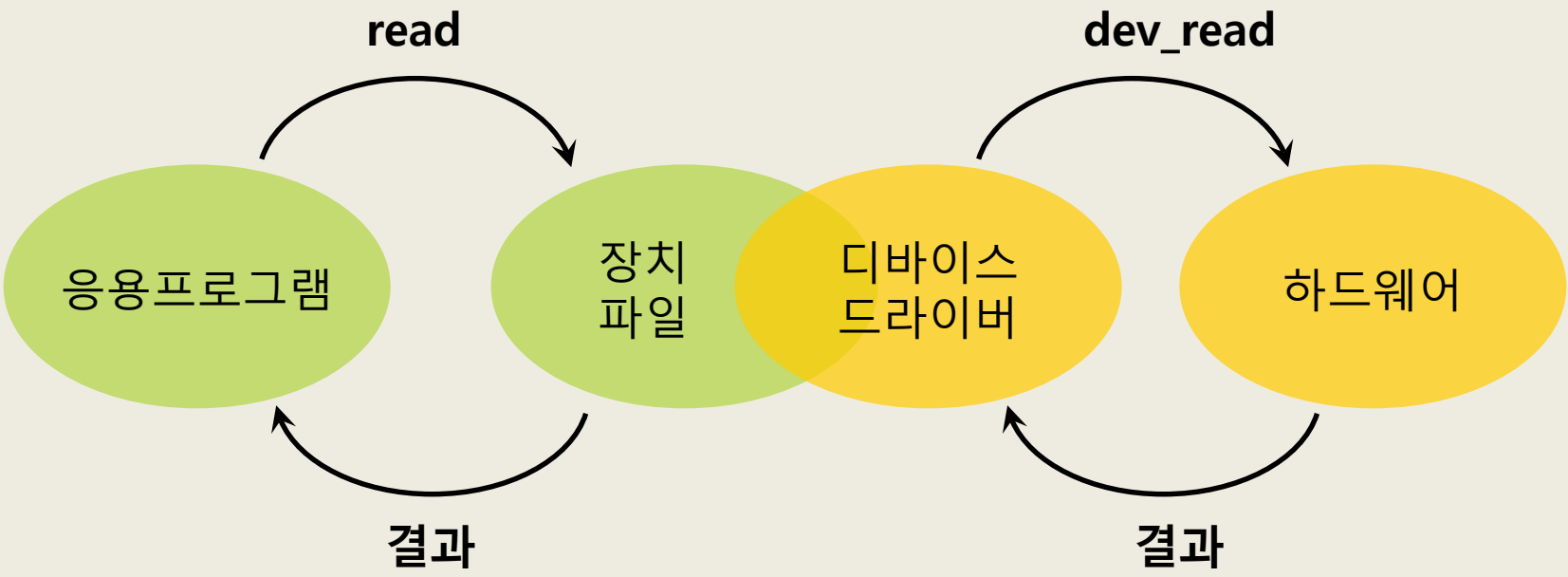
- **System call processing**



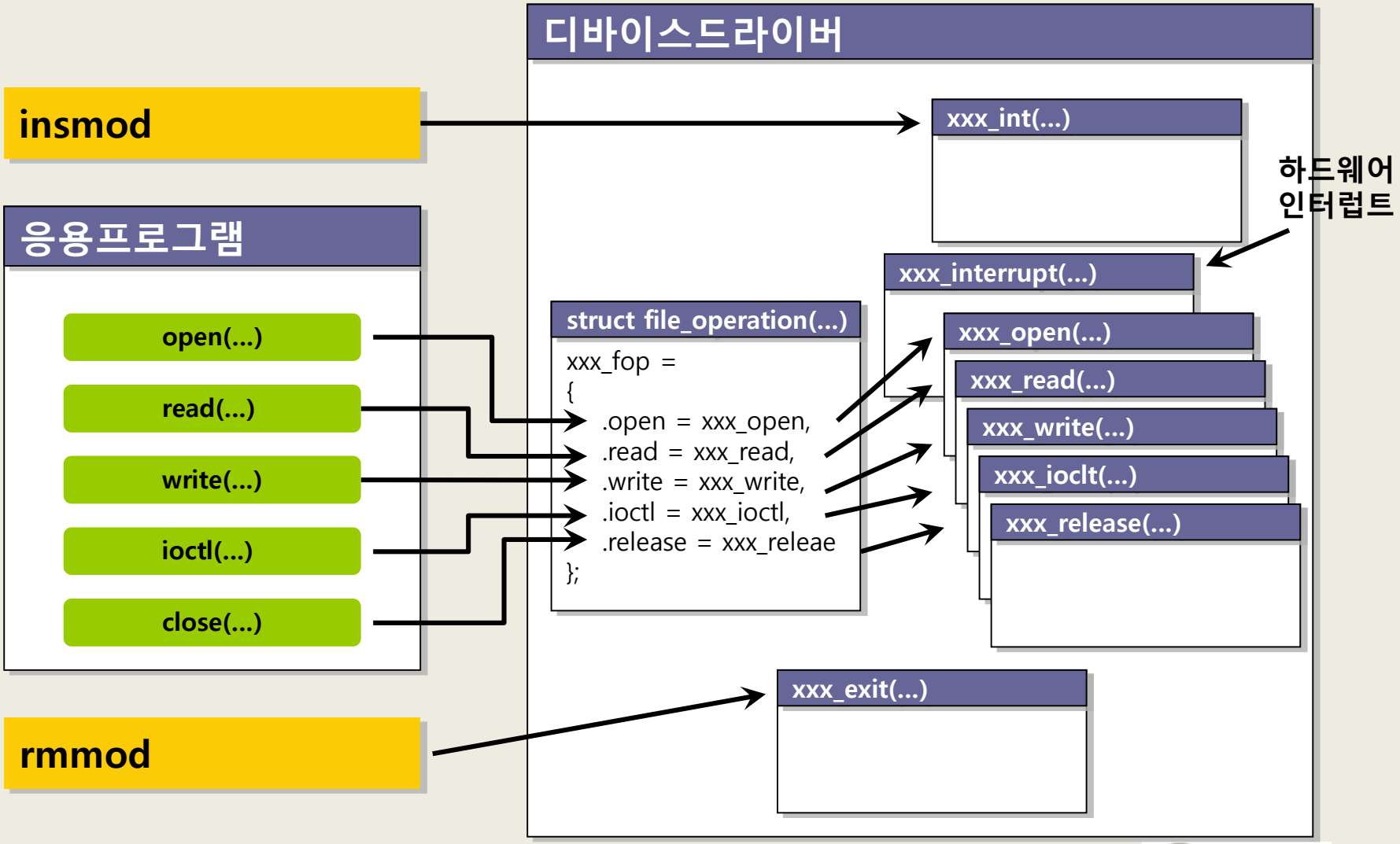
User space and Kernel space



Data exchange between Kernel and User space.

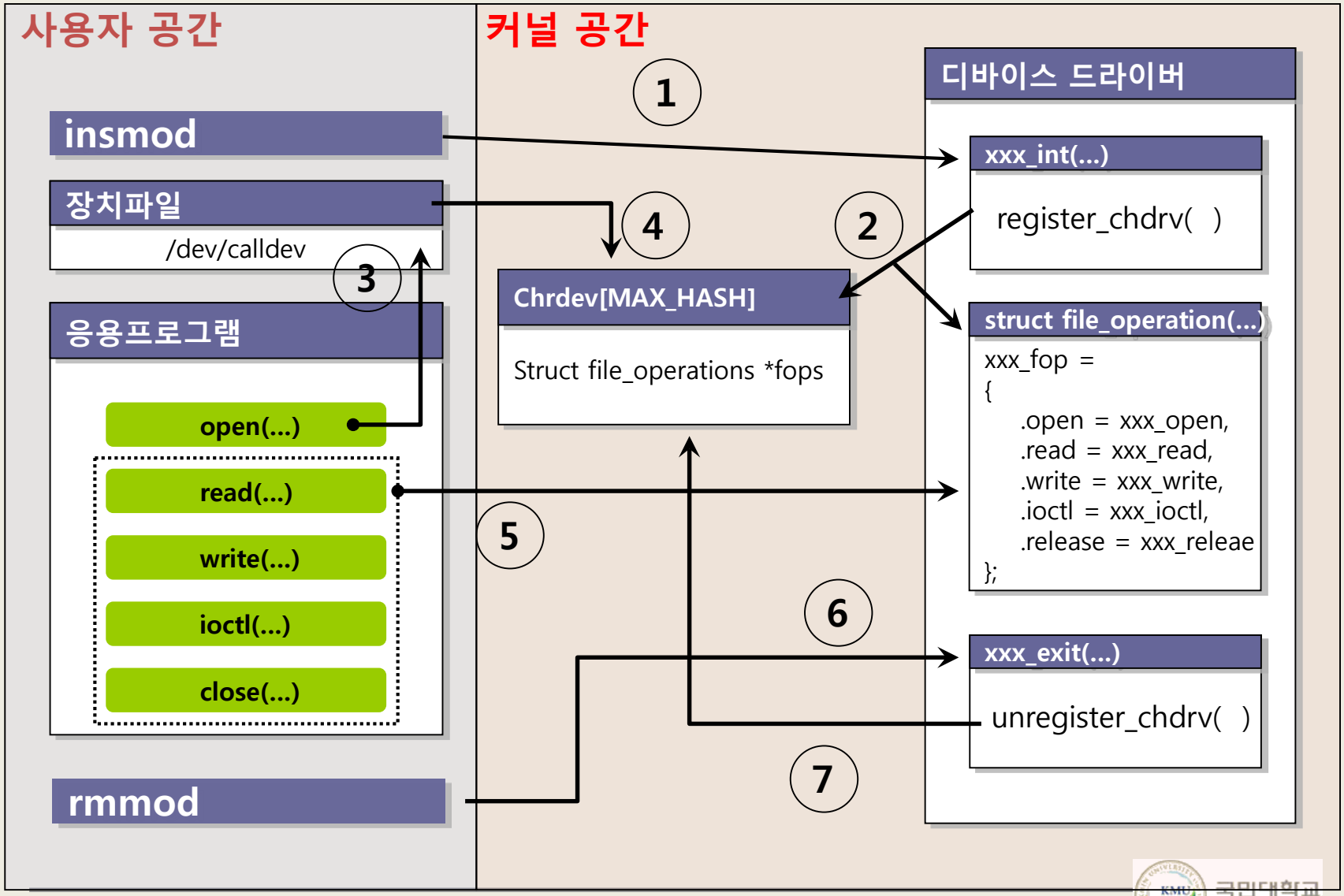


Device Driver as Kernel Module



※ 커널버전 2.6

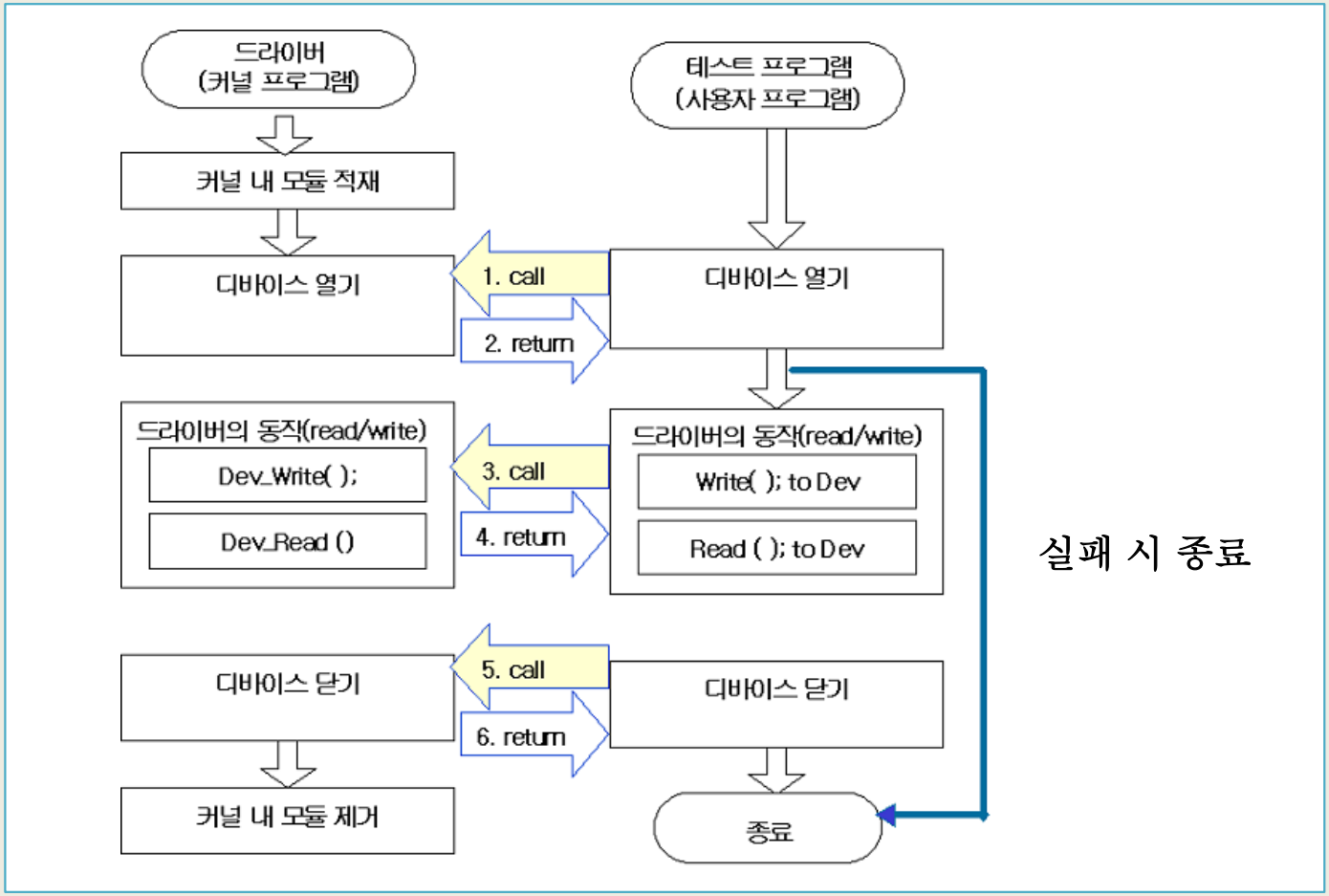
Device Driver as Kernel Module (cont'd)



※ 커널버전 2.6

문자 디바이스 드라이버 동작

◉ 디바이스 드라이버의 동작과정



※ 커널버전 2.6

문자 디바이스 제작방법

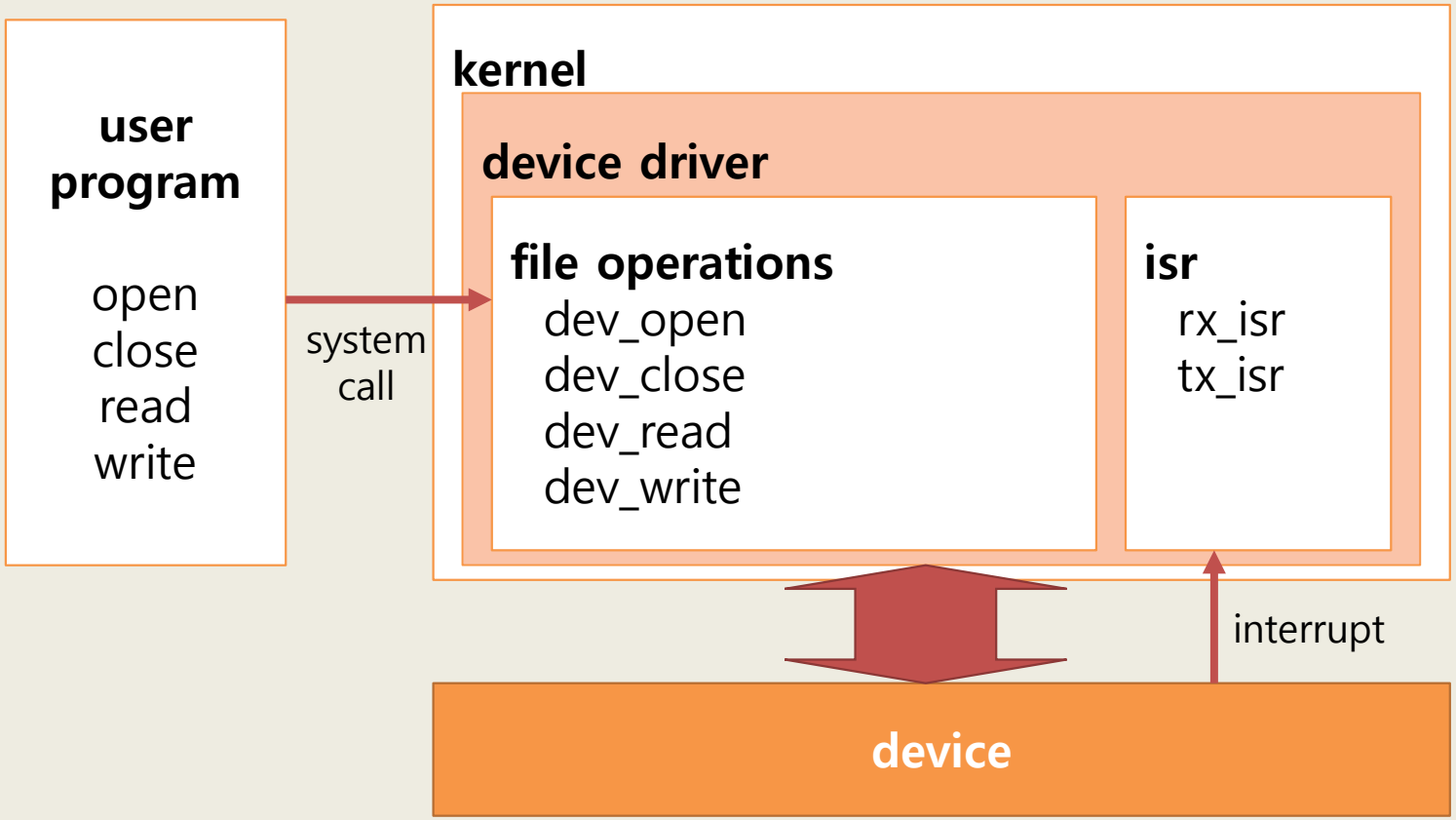
⊙ 문자 디바이스 드라이버 제작법

- **외부와 디바이스 드라이버는 파일 인터페이스를 통해서 연결**
- **디바이스 드라이버는 file_operations를 제공함으로써 구현**
- **디바이스 드라이버는 자신을 구별하기 위해 고유의 major number를 사용**

⊙ 장치의 등록과 해제

- **등록 : int register_chrdev(unsigned int major, const char *name, struct file_operations *fops)**
  - major : 등록할 major number. 0이면 사용하지 않는 번호 중 동적으로 할당
  - name : 장치의 이름
  - fops : 장치에 대한 파일 연산 함수들
- **해제 : int unregister_chrdev(unsigned int major, const char *name)**

문자 디바이스 제작방법



※ 커널버전 2.6

문자 디바이스 제작방법

major number와 minor number

- 장치를 구분하는 방법으로 둘을 이용하여 특정 장치를 구별
- major number : 커널에서 디바이스 드라이버를 구분하는데 사용
- minor number : 디바이스 드라이버 내에서 필요한 경우 장치를 구분하기 위해 사용
- 새로운 디바이스는 새로운 major number를 가져야 함
- linux/Documentation/devices.txt에 모든 장치의 major number 정의
- register_chrdev()로 장치를 등록할 때 major number를 지정
- 같은 major number가 등록되어 있으면 등록 실패

문자 디바이스 제작방법

- mknod 명령으로 디바이스 드라이버에 접근할 수 있는 장치 파일 생성
  - mknod [device file name] [type] [major] [minor]
- mdev_t : 장치의 major, minor number를 표현하는 자료구조
  - MAJOR() : kdev_t에서 major number를 얻어내는 매크로
  - MINOR() : kdev_t에서 minor number를 얻어내는 매크로
  - MKDEV(ma, mi) : major number, minor number를 가지고 kdev_t를 만듬
  - cat /proc/devices 명령으로 현재 로드된 디바이스 드라이버 확인

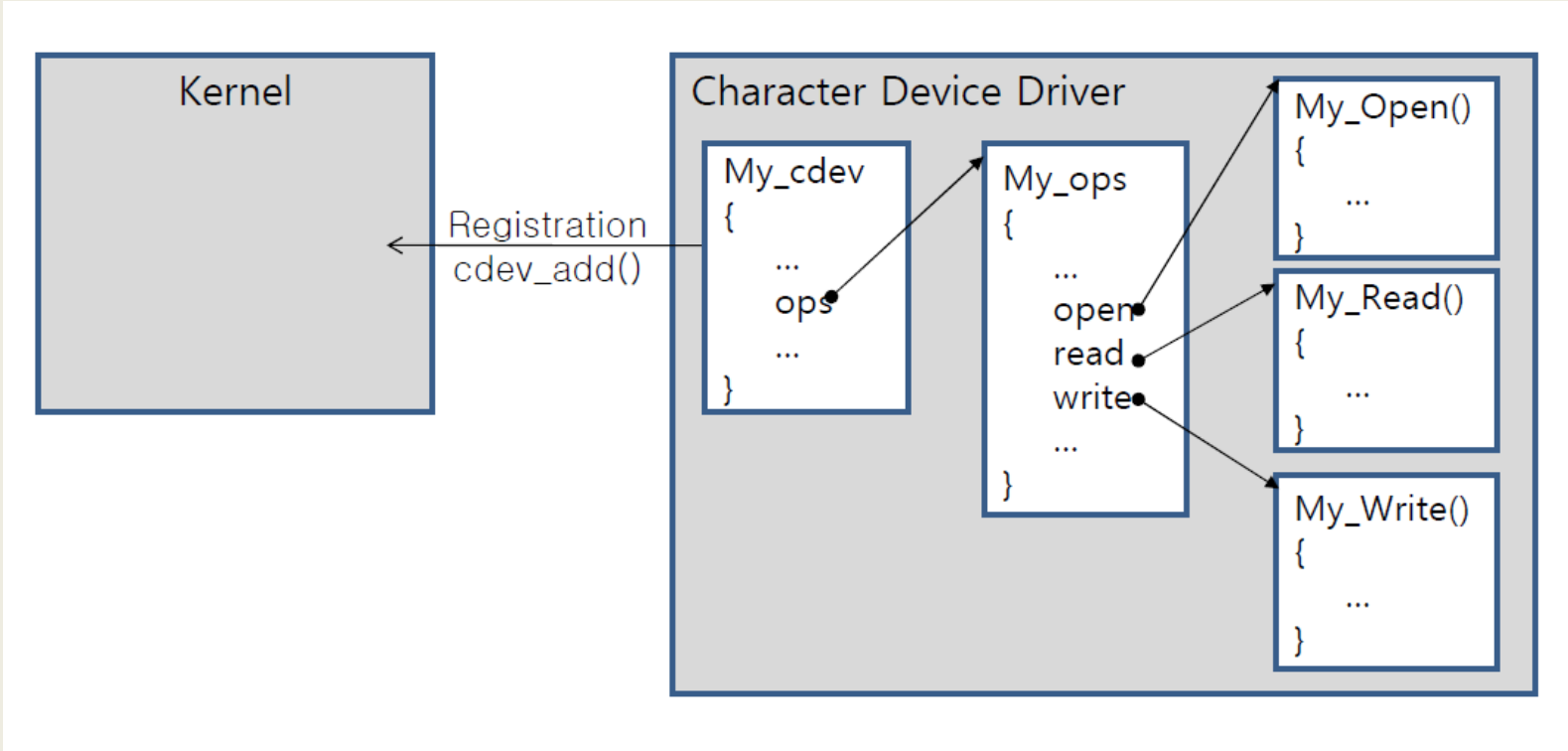
Driver Programming

file_operations Structure

- **Defines driver's operations (system calls)**
- **Collection of function pointers**
  - Leaves NULL for unsupported operations

```
struct file_operations {
  struct module *owner;
  …
  ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
  ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
  …
//int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
  long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
  long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
  …
  int (*open) (struct inode *, struct file *);
  …
};
```

File Operations Initialization



File Operations

- module owner : 모듈 사용자 이름
- llseek : 파일에서 현재 읽고 쓰는 위치 이동
- read : 장치에서 데이터를 읽어 들임
- write : 장치에 데이터를 기록
- readdir : 장치에서 사용하지 않고 디렉토리에서 사용하는 함수
- poll : 장치에 읽고 쓸 수 있는지 예외 상황이 발생했는지 확인하는 함수
- unlocked_ioctl : 읽기/쓰기가 아닌 장치마다 필요한 명령을 내리는데 사용하는 함수
  - ioctl에서 변경됨. 기존엔 자동으로 lock이 수행되었으나 비효율적이라 변경됨.
- compat_ioctl : 32비트, 64비트 호환성을 갖도록 설계된 ioctl.
- mmap : 장치의 메모리와 프로세서의 메모리를 mapping함
- open : 장치 열기
- flush : 장치를 해제
- release : 장치를 닫음
- fsync : 장치를 flush함
- readv : 파일 기술자 fd 에서 데이터를 읽고 그 결과를 vector 가 가리키는 버퍼에 삽입
- writev : vector 가 가리키고 있는 버퍼에서 파일 기술자 fd 에 데이터를 씀

문자 디바이스 제작방법

Open에서 할 일:

- 처음으로 장치를 연 경우 장치 초기화
- Minor번호를 확인하고 필요한 경우 f_op 포인터 수정
- 필요한 경우 메모리를 할당받아 filp->private_data삽입
- 참조 횟수(usage count)를 증가

Release에서 할 일:

- 참조 횟수를 감소
- filp->private_data에 할당된 데이터가 있으면 삭제
- Close하는 경우 장치 종료

문자 디바이스 제작방법

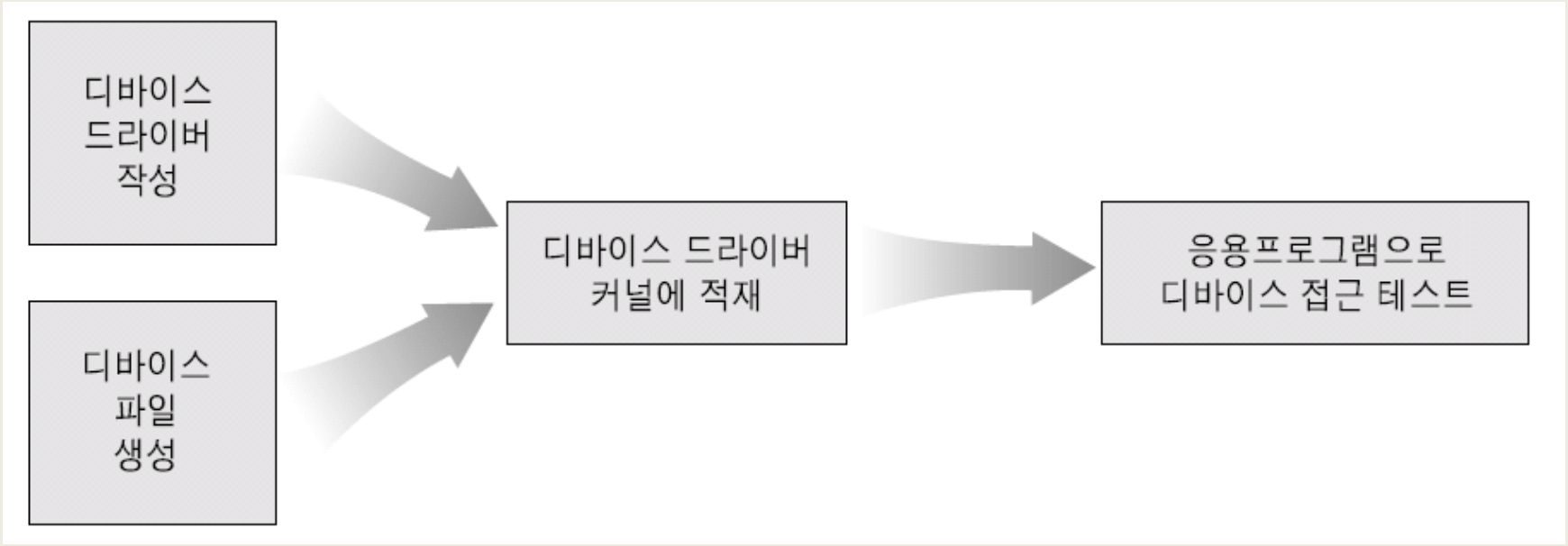
Read의 return value

- 요구한 만큼 읽은 경우 count와 같은 값을 리턴
- 요구한 값보다 작을 경우 count보다 작은 양수 리턴
- EOF(end of file)인 경우 0을 리턴
- 에러가 발생하면 음수 리턴

Write

- 요구한 만큼 기록한 경우 count와 같은 값을 리턴
- 요구한 크기보다 적게 쓴 경우 count보다 작은 양수 리턴
- 하나도 쓰지 못한 경우 0 리턴, write()함수 재시도
- 에러가 발생한 경우 음수 리턴

디바이스 드라이버 구현 및 테스트 과정



가상 문자 디바이스 드라이버 프로그램 작성 (chr_dev.c)

```c
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/init.h>

#define CHR_DEV_NAME "chr_dev“   // 디바이스 파일 이름
#define CHR_DEV_MAJOR 240        // 디바이스 파일의 주번호

int chr_open(struct inode *inode, struct file *filp)
{
        int number = MINOR(inode->i_rdev); // 부번호를 number에 저장
        printk("Virtual Character Device Open: Minor Number is %d\n", number);
        return 0;
}
ssize_t chr_write(struct file *filp, const char *buf, size_t count, loff_t *f_pos)
{
        printk("write data: %s\n", buf);    // 응용프로그램 write()
                                            // 함수의 buf 값을 커널 메시지에 출력
        return count;
}
ssize_t chr_read(struct file *filp, const char *buf, size_t count, loff_t *f_pos)
{
        printk("read data: %s\n", buf);     // 응용프로그램 read()
                                            // 함수의 buf 값을 커널 메시지에 출력
        return count;
}
```

```c
int chr_ioctl(struct inode *inode, struct file *filp, unsigned int cmd, unsigned long arg)
{
        switch(cmd) {            // ioctl 함수로 전달된 cmd 값을 출력
                case 0: printk("cmd value is %d\n", cmd); break;
                case 4: printk("cmd value is %d\n", cmd); break;
        }
        return 0;
}

int chr_release(struct inode *inode, struct file *filp)
{
        printk("Virtual Character Device Release\n");
        return 0;
}

struct file_operations chr_fops =
{
        owner: THIS_MODULE,
        unlocked_ioctl: chr_ioctl,
        write: chr_write,
        read: chr_read,
        open: chr_open,
        release: chr_release
};
```

```c
int sample_init(void)// 디바이스를 커널에 모듈로 적재시 수행되는 함수
{
        int registration;;   // registration에 주번호나 반환값을 저장
        printk("Registration Character Device to Kernel\n");
        registration = register_chrdev(CHR_DEV_MAJOR,CHR_DEV_NAME, &chr_fops);
        if(registration < 0)
                return registration;
        printk("Major Number:%d\n", registration);
        return 0;
}

void sample_cleanup(void)// 커널에서 디바이스를 제거할 때 수행되는 함수
{
        printk("Unregistration Character Device to Kernel\n");
        unregister_chrdev(CHR_DEV_MAJOR, CHR_DEV_NAME);
}

MODULE_LICENSE("GPL");
module_init(sample_init);
module_exit(sample_cleanup);
```

가상 문자 디바이스를 사용하는 응용프로그램 작성 (chr_appl.c)

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <unistd.h>

#define DEVICE_FILE_NAME "/dev/chr_dev“     // 디바이스 파일

// argv 값을 받아 디바이스 파일의 IOCTL cmd 값으로 사용
int main(int argc, char *argv[]) {
     int device;
     char wbuf[128] = "Write buffer data";
     char rbuf[128] = "Read buffer data";
     int n = atoi(argv[1]);

     device = open(DEVICE_FILE_NAME, O_RDWR | O_NDELAY);
     if (device >= 0) {
            printf("Device file Open\n");
            ioctl(device, n);                // argv 값을 디바이스 파일에 cmd 값으로 전달
            write(device, wbuf, 10);         // wbuf 값을 디바이스 파일에 전달
            printf("Write value is %s\n", wbuf);
            read(device, rbuf, 10);
            printf("read value is %s\n", rbuf);
     } else
            perror("Device file open fail");

     return 0;
}
```

모듈 컴파일하는 Makefile 작성 (Makefile)

```
obj-m := chr_dev.o

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

All: test_dd test_app

test_dd: chr_dev.c
	$(MAKE) -C $(KERNELDIR) SUBDIRS=$(PWD) modules

test_app : chr_appl.c
	gcc -o chr_appl chr_appl.c


clean:
	$(MAKE) -C $(KERNELDIR) M=$(PWD) clean
	rm -rf gyapp2
```

Make 유틸리티를 실행해 모듈 생성

```
pi@raspberrypi ~ $ make
make -C /lib/modules/3.18.10+/build M=/home/pi modules
make[1]: Entering directory '/home/pi/linux-7afb1c5b7cf33a3182c97ac9be7379394b9b462a'
  CC [M]  /home/pi/chr_dev.o
/home/pi/chr_dev.c:42:2: warning: initialization from incompatible pointer type [enabled by
    default]
  unlocked_ioctl: chr_ioctl,
  ^
/home/pi/chr_dev.c:42:2: warning: (near initialization for 'chr_fops.unlocked_ioctl')
    [enabled by default]
/home/pi/chr_dev.c:44:2: warning: initialization from incompatible pointer type [enabled by
    default]
  read: chr_read,
  ^
/home/pi/chr_dev.c:44:2: warning: (near initialization for 'chr_fops.read') [enabled by
    default]
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/pi/chr_dev.mod.o
  LD [M]  /home/pi/chr_dev.ko
make[1]: Leaving directory '/home/pi/linux-7afb1c5b7cf33a3182c97ac9be7379394b9b462a‘
pi@raspberrypi ~/t $
```

디바이스 드라이버 모듈을 적재하고 적재 여부 확인

```
pi@raspberrypi ~ $ sudo insmod chr_dev.ko
pi@raspberrypi ~ $ lsmod
Module                  Size  Used by
chr_dev                 2060  0
snd_bcm2835            21157  0
snd_pcm                90778  1 snd_bcm2835
snd_seq                61097  0
snd_seq_device          7209  1 snd_seq
snd_timer              23007  2 snd_pcm,snd_seq
snd                    66285  5
    snd_bcm2835,snd_timer,snd_pcm,snd_seq,snd_seq_device
uio_pdrv_genirq         3666  0
uio                     9897  1 uio_pdrv_genirq
pi@raspberrypi ~ $
```

디바이스 파일을 만들고 확인

```
pi@raspberrypi ~ $ sudo mknod /dev/chr_dev c 240 0
pi@raspberrypi ~ $ ls -l /dev/chr_dev
crw-r--r-- 1 root root 240, 0 Apr 10 14:14 /dev/chr_dev
pi@raspberrypi ~ $
```

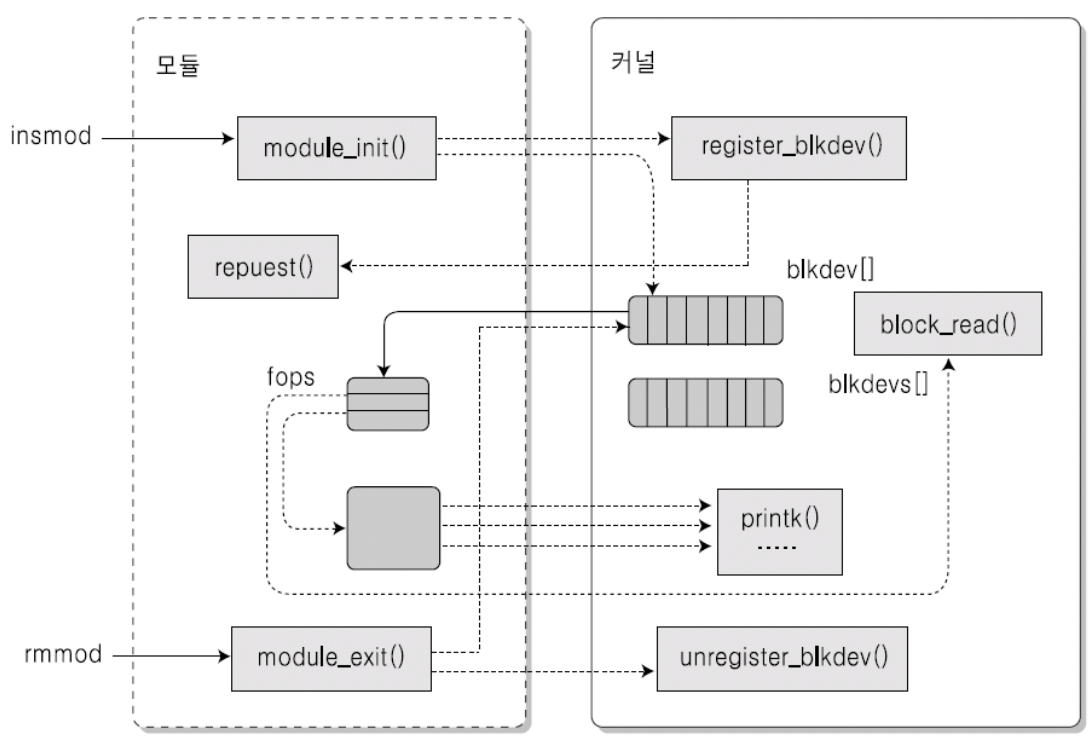
응용 프로그램을 실행하여 디바이스드라이버 테스트

```
pi@raspberrypi ~ $ sudo ./chr_appl 1
Device file Open
Write value is Write buffer data
read value is Read buffer data
pi@raspberrypi ~ $
```

디바이스 드라이버 모듈 제거

```
pi@raspberrypi ~ $ sudo rmmod chr_dev
pi@raspberrypi ~ $
```

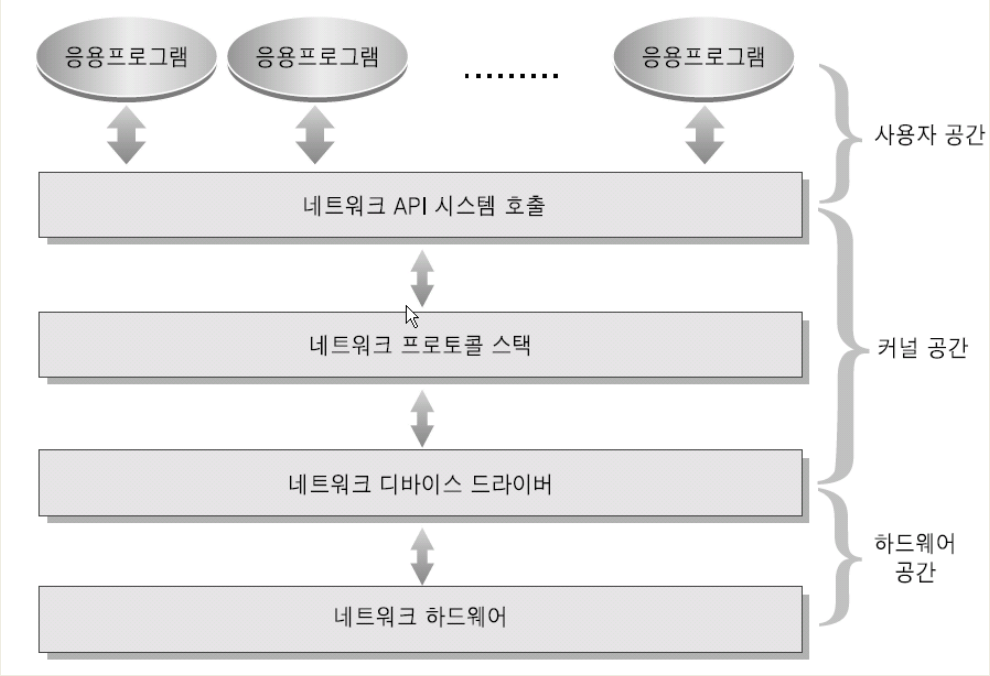
블록 디바이스 드라이버 작성



- 블록 디바이스 특성, 주번호와 부번호 및 디바이스 이름 정의
- register_blkdev() 함수를 사용한 블록 디바이스 드라이버의 등록
- 블록 디바이스 연산 구조체 처리
- 요구 큐에 관련된 처리 및 함수를 선언
- 블록 디바이스 추가를 위한 gendisk 구조체 생성 및 등록
- 블록 디바이스의 크기 설정 및 기타 속성을 처리

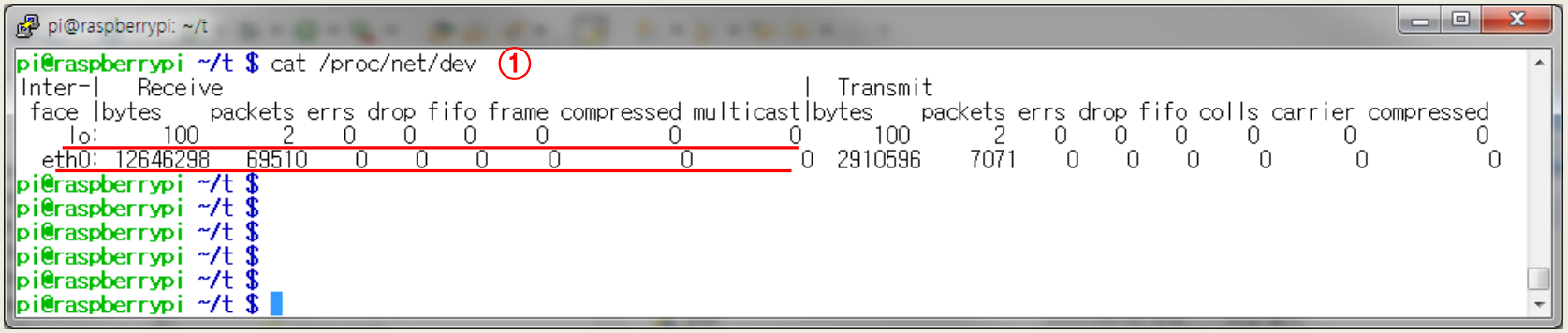
네트워크 디바이스 드라이버 작성

- 모듈 적재 혹은 커널 부팅에 의한 초기화 처리
- 모듈 제거를 위한 마무리 처리
- 네트워크 디바이스 검출
- 네트워크 디바이스 초기화 및 등록
- 네트워크 디바이스 열기 및 닫기
- 네트워크 데이터 전송 및 수신
- 인터럽트 처리
- 네트워크 디바이스 제어를 위한 ioctl() 함수 구현
- 멀티 캐스트 처리
- 설정 정보의 재설정 처리



등록된 네트워크 디바이스 드라이버 정보 보기

- Proc 파일시스템에서 커널에 등록된 디바이스 드라이버 정보 확인



REFERENCES: 커널 인터페이스 함수

커널 인터페이스 함수

Memory

- **kmalloc(unsigned int len, int priority)**
  - 커널 메모리 할당. 128~131056byte까지 가능
  - priority:GFP_BUFFER, GFP_ATOMIC, GFP_USER, GFP_KERNEL
- **kfree(void *obj)**
  - kmalloc()에서 할당받은 커널 메모리(obj)를 반납
- **vmalloc(unsigned int len), vmfree(void *addr)**
  - 커널 메모리 할당/반납, 크기 제한 없음
- **memcpy_xxfs(void *to, const void *from, unsigned long n)**
  - 커널주소공간과 사용자주소공간 사이에 memory copy
  - Xx=from/to
- **memset(void *s, char c, sizt_t count)**
  - 메모리 s에 c를 count만큼 복사

커널 인터페이스 함수

⊙ 동기화

- **sleep_on(struct wait_queue **q)**
  - q의 번지를 event로 sleep하며, uninterruptible
- **sleep_in_interruptible(struct wait_queue **q)**
  - q의 번지를 event로 sleep하며, interruptible
- **wake_up(struct wait_queue **q)**
  - sleep_on(q)에 의해 sleep한 task를 wakeup
- **wake_up_interruptible(struct wait_queuq **q)**
  - sleep_on_interruptible(q)에 의해 sleep한 task를 wakeup

커널 인터페이스 함수

- **printk(const char *fmt,….)**
  - **printf의 커널 버전**
  - **printk(LOG_LEVEL_ message)**
    - LOG_LEVEL:KERN_EMERG, KERN_ALERT, KERN_ERR, KERN_WARNING, KER_INFO, KERN_DEBUG
  - **예**
    - printk(“<1>Hello, World”);
    - printk(KERN_WARNING“warning…₩n”);
  - **sprintf(char *str, const char *fmt, …)**
    - print to string

커널 인터페이스 함수

driver register

- **register_xxxdev(unsigned int major, const char *name, struct file_operations *fops)**
  - character/block driver를 xxxdev[major]에 등록
  - xxx:blk/chr
- **unregister_xxxdev(unsigned int major, const char *name)**
  - xxxdevs[major]에 등록되 있는 device driver를 제거
- **major(a)/minor(a)**
  - 장치번호a로부터 major/minor 번호를 구함

커널 인터페이스 함수

Port I/O

- **inb(unsigned short port),inb_p(unsigned port)**
  - port로부터 1byte읽음
- **outb(char value, unsigned short port)**
- **outb_p(char value, unsigned short port)**
  - port에 1byte의 value를 출력함
- **inb_p(), out_p:inb()/outb() and pause의 의미**
- **inw(), inw_p(), outw(), outw_p()**
  - port로부터 short int(2byte)읽거나 출력함
- **int(), inl_p(), outl(), outl_p()**
  - port로부터 long int(4byte)읽거나 출력함
- **insb(unsigned int port, void *to, int len)**
  - port로부터 len bytes를 읽어, to가 가리키는 메모리에 저장
  - insb_p(), outsb(), outsb_p(), insw(), insw_p(), outsw(),
  - outsw_p(), insl(), outsl_p()

커널 인터페이스 함수

Interrupt

- **cli()/sti()**
  - clear/set interrupt enable
- **save_flags(), restore_flags()**
  - status register의 내용을 저장하고 복원

```
register unsigned long flags;

save_flags(flags);
cli();
…
restore_flags(flags);
```

- **requst_irq(unsigned int irq, void (*handler)(int),**
- **unsigned long flags, const char *device)**
  - 커널로부터 IRQ를 요청하여, IRQ interrupt handler를 install
- **free_irq(unsigned int irq)**
  - request_irq()에서 획득한 irq를 반납함