Linux Device Driver

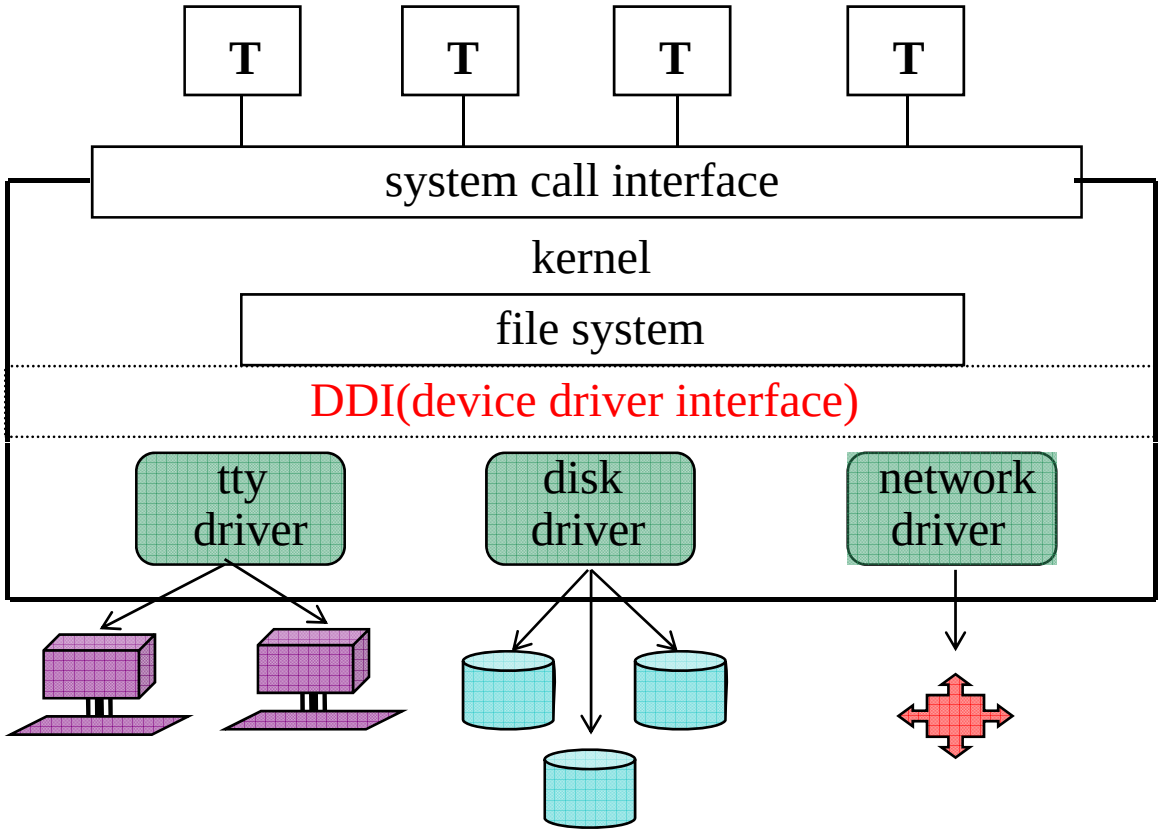
단국대학교
컴퓨터학과
2009
백승재
ibanez1383@dankook.ac.kr
http://embedded.dankook.ac.kr/~ibanez1383

강의 목표

- 디바이스 드라이버 구조 파악
- 리눅스의 디바이스 드라이버 관리 구조 이해
- 디바이스 드라이버 제작

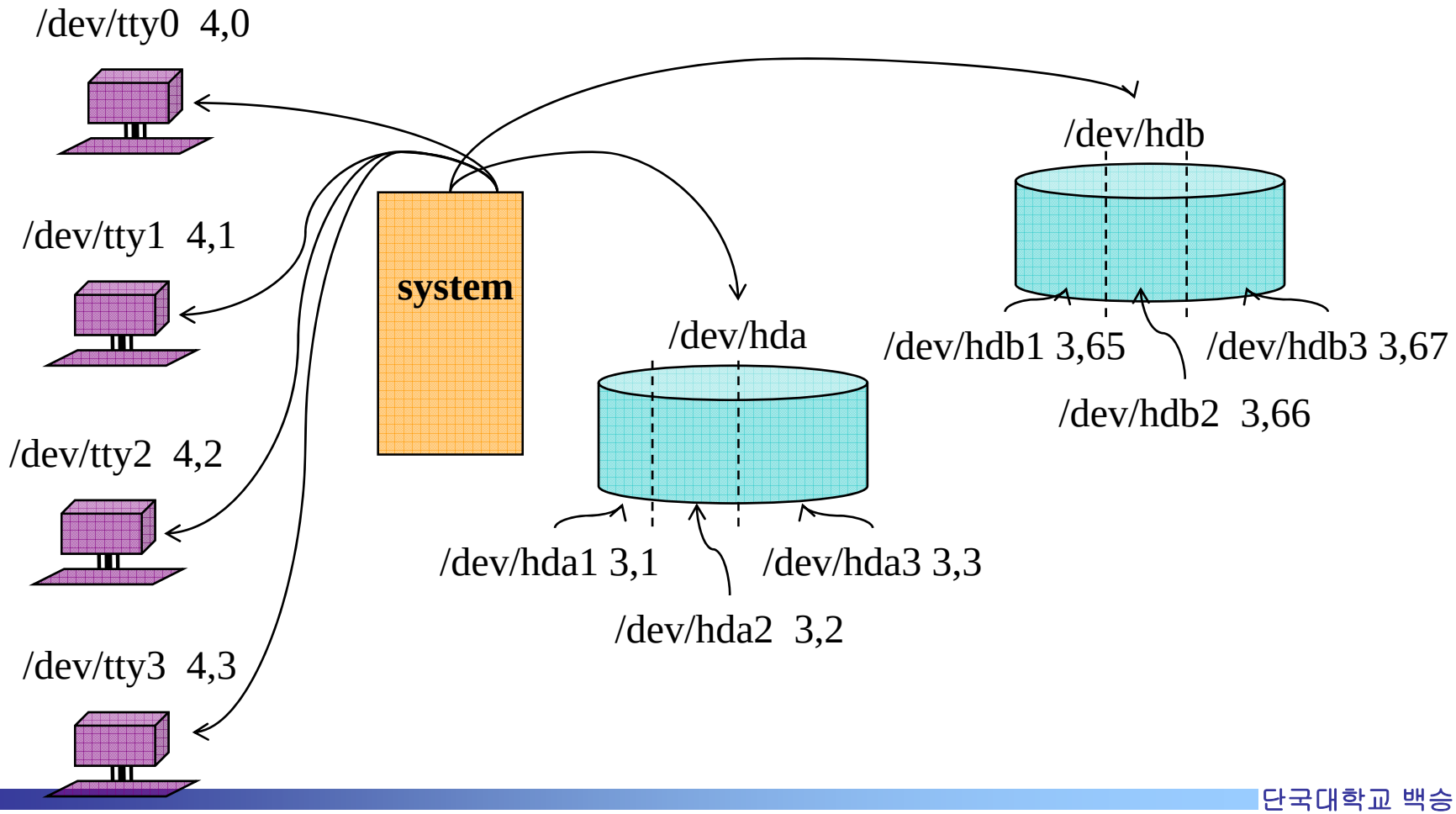
디바이스 드라이버의 역할

- 역할 : 주변 장치와 커널 간에 데이터 전달
- 3 유형의 디바이스 드라이버 : 문자 디바이스 드라이버, 블록 디바이스 드라이버, 네트웍 디바이스 드라이버,



장치 파일 (1/3)

- 주 번호와 부 번호
  - ✓ 주 번호 : 디바이스 유형 (device type)
  - ✓ 부 번호 : 디바이스 단위 (device unit)



장치 파일 (2/3)

- 장치 파일 (device file)
  - ✓ 디바이스 드라이버를 접근하는 통로
  - ✓ 장치 파일의 inode는 장치 유형 (type), 주 번호 (major number), 부 번호 (minor number) 등으로 구성됨
- 장치 파일의 예

```
...
brw-r-----    1   root   disk  3  0   Oct   3    1993  hda
brw-r-----    1   root   disk  3  1   Oct   3    1993  hda1
brw-r-----    1   root   disk  3  2   Oct   3    1993  hda2
brw-r-----    1   root   disk  3  3   Oct   3    1993  hda3
...
crw--w--w--   1   card   tty   4  0   May   16   1993  tty0
crw--w--w--   1   card   tty   4  1   May   16   1993  tty1
crw--w--w--   1   card   tty   4  2   May   16   1993  tty2
...
```

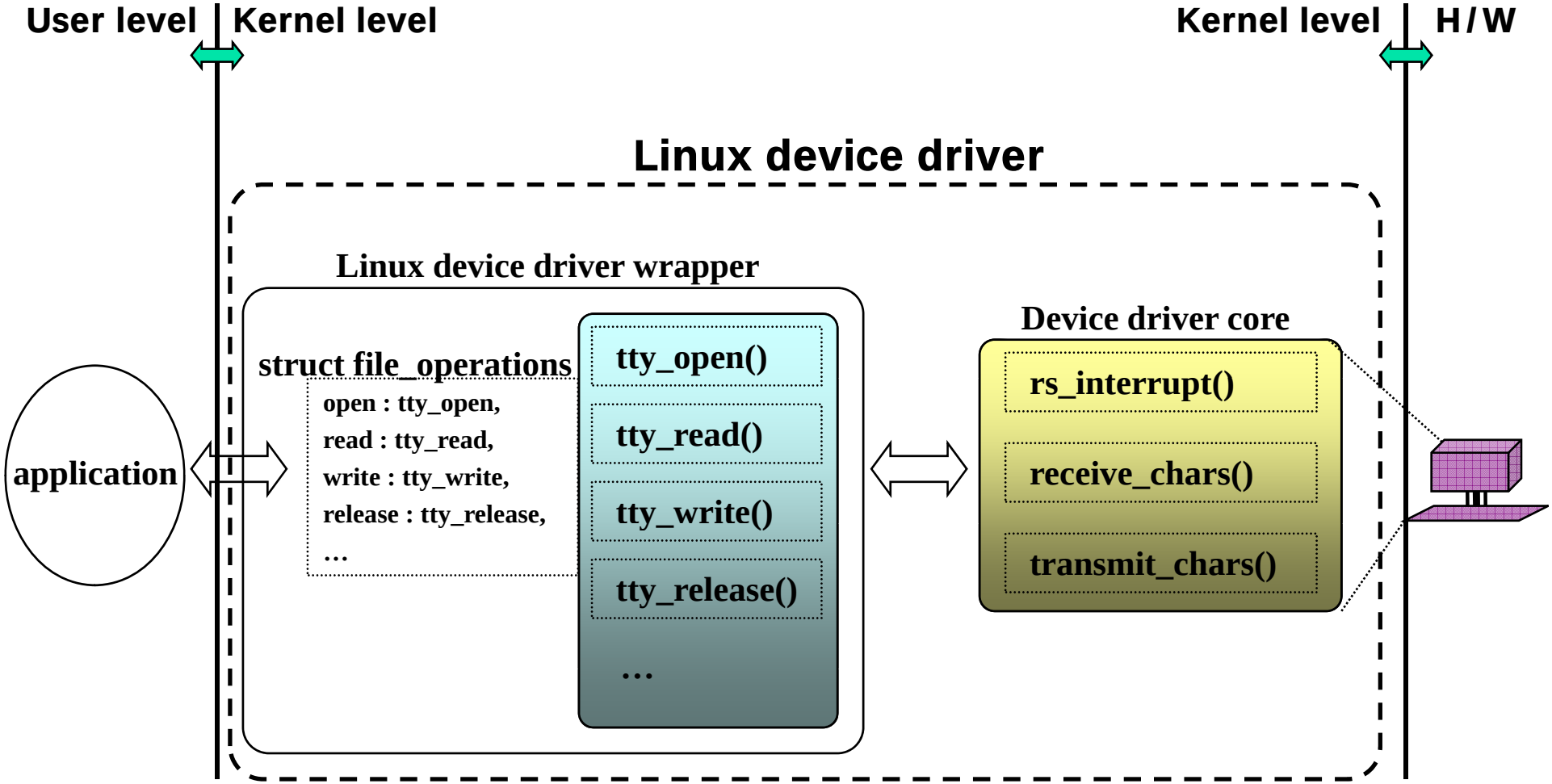
- 장치 파일의 생성 : mknod
  - ✓ mknod /dev/file_name [b|c] major_number minor_number

장치 파일 (3/3)

- 주 번호 할당 : 0~255 /* include/linux/major.h */
  - ✓ Linux에는 많은 장치들의 주 번호가 이미 정해져 있다.
  - ✓ 개발자는 현재 할당되지 않은 번호 사용 가능

Major	Character devices	Block devices
0		
1	mem	RAM disk
2		floppy (fd*)
3		IDE hard disk (hd*)
4	terminal	
5	terminal & AUX	
6	Parallel Interface	
7	virtual console (vcs*)	
8		SCSI hard disk (sd*)
9	SCSI tapes (st*)	
10	Bus mice(bm, psaux)	
11		SCSI CD-ROM(scd*)
……		
23		Mitsumi CD-ROM (mcd*)
….		

디바이스 드라이버 코어와 래퍼의 분리된 구조



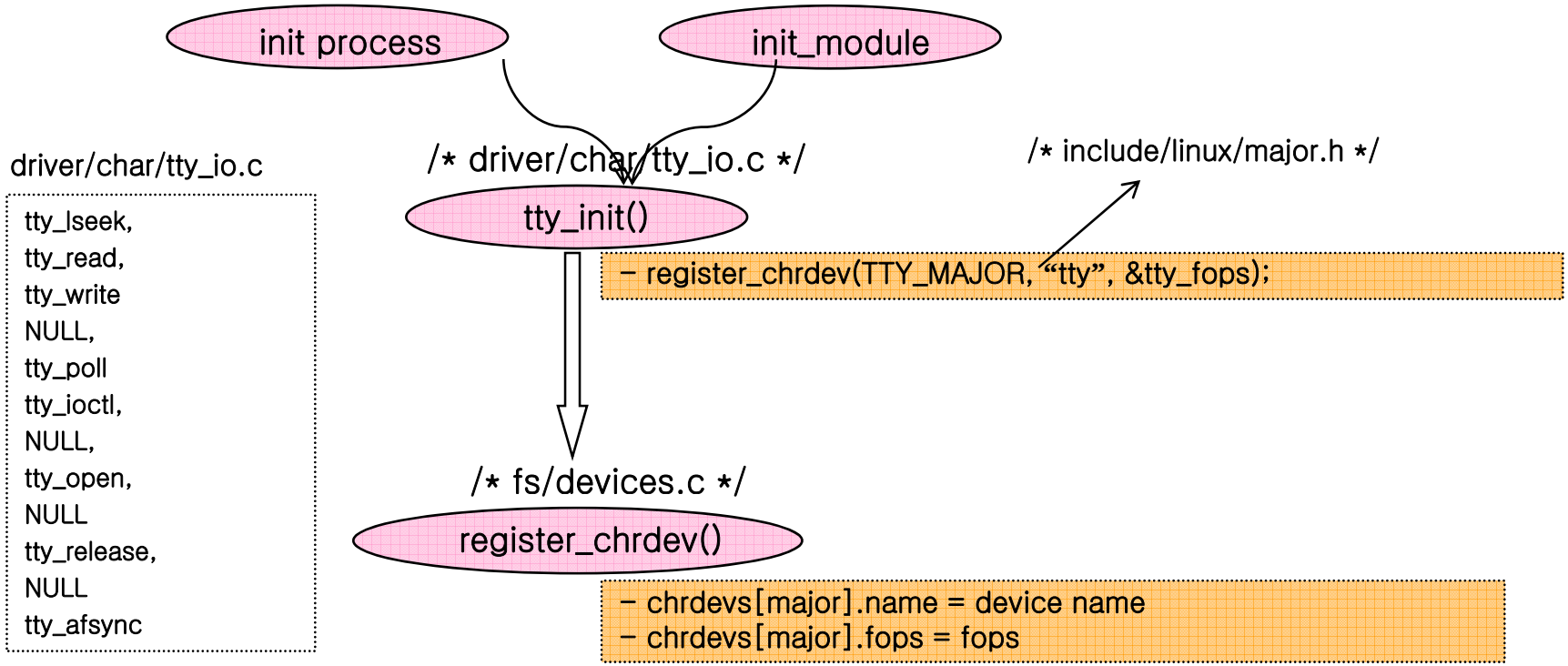
디바이스 드라이버 구조

- 디바이스 드라이버는 다음의 3 부분으로 구성
  - ✓ 초기화 인터페이스
    - init() 함수, init_module() 함수
  - ✓ 파일 시스템 인터페이스 부분
    - 잘 정의된 인터페이스 : 문자와 블록의 경우 file_operations 이용
    - 문자 드라이버 : open, release, read, write, ioctl
    - 블록 드라이버 : open, release, request, ioctl
    - 네트웍 드라이버 : open, close, transmit, receive, ioctl
  - ✓ 하드웨어 인터페이스
    - in(), out()

- 디바이스 드라이버와 커널간에 통신
  - ✓ 문자 드라이버 : chrdevs[] 테이블 이용
  - ✓ 블록 드라이버 : blkdevs[], blk_dev[] 테이블 이용
  - ✓ 네트웍 드라이버 : device 구조 이용

문자 디바이스 드라이버 구조 (1/4)

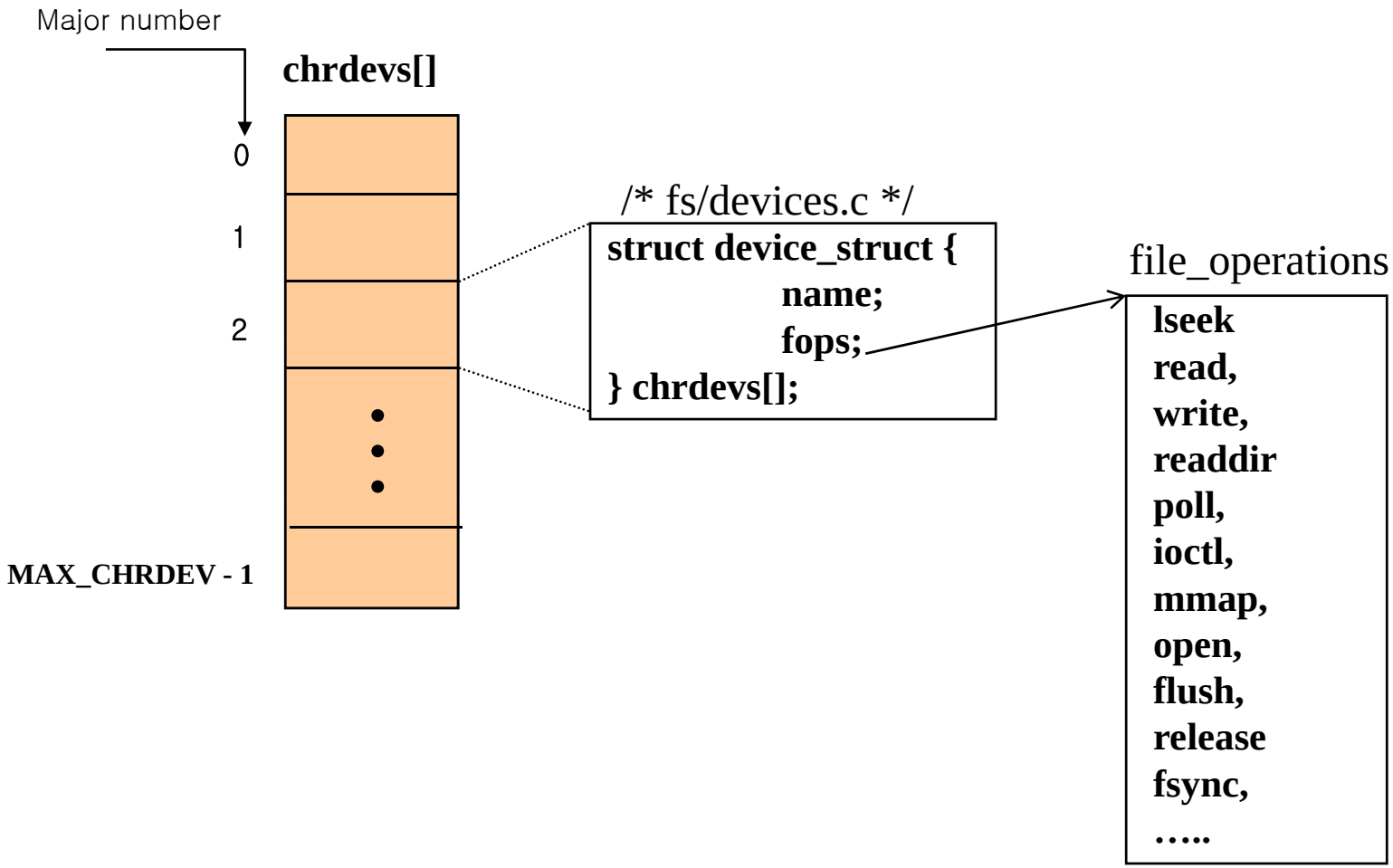
❑ 터미널 드라이버 초기화 : tty_init()



struct file_operations tty_fops

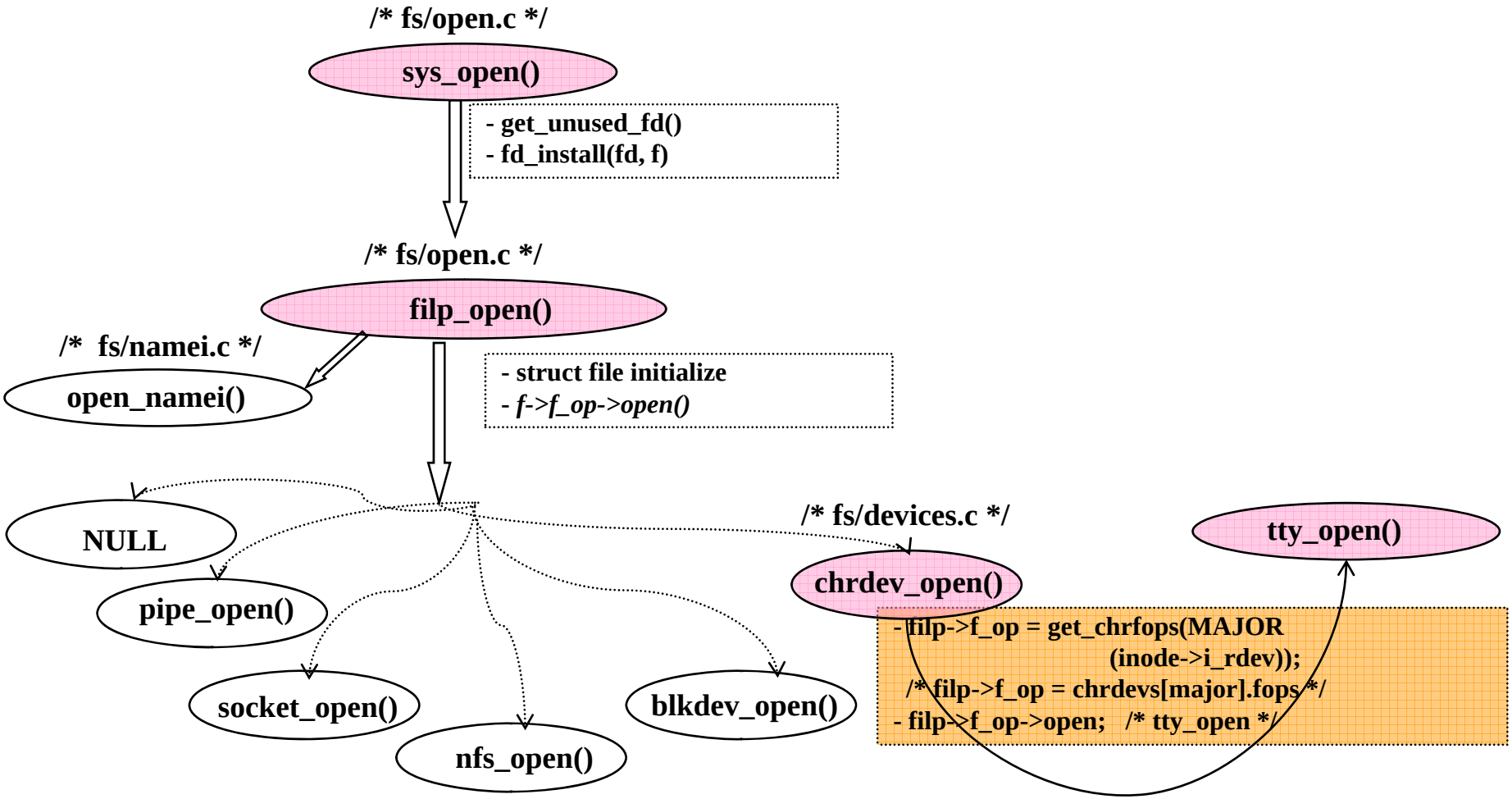
문자 디바이스 드라이버 구조 (2/4)

- chrdevs[] 테이블 구조



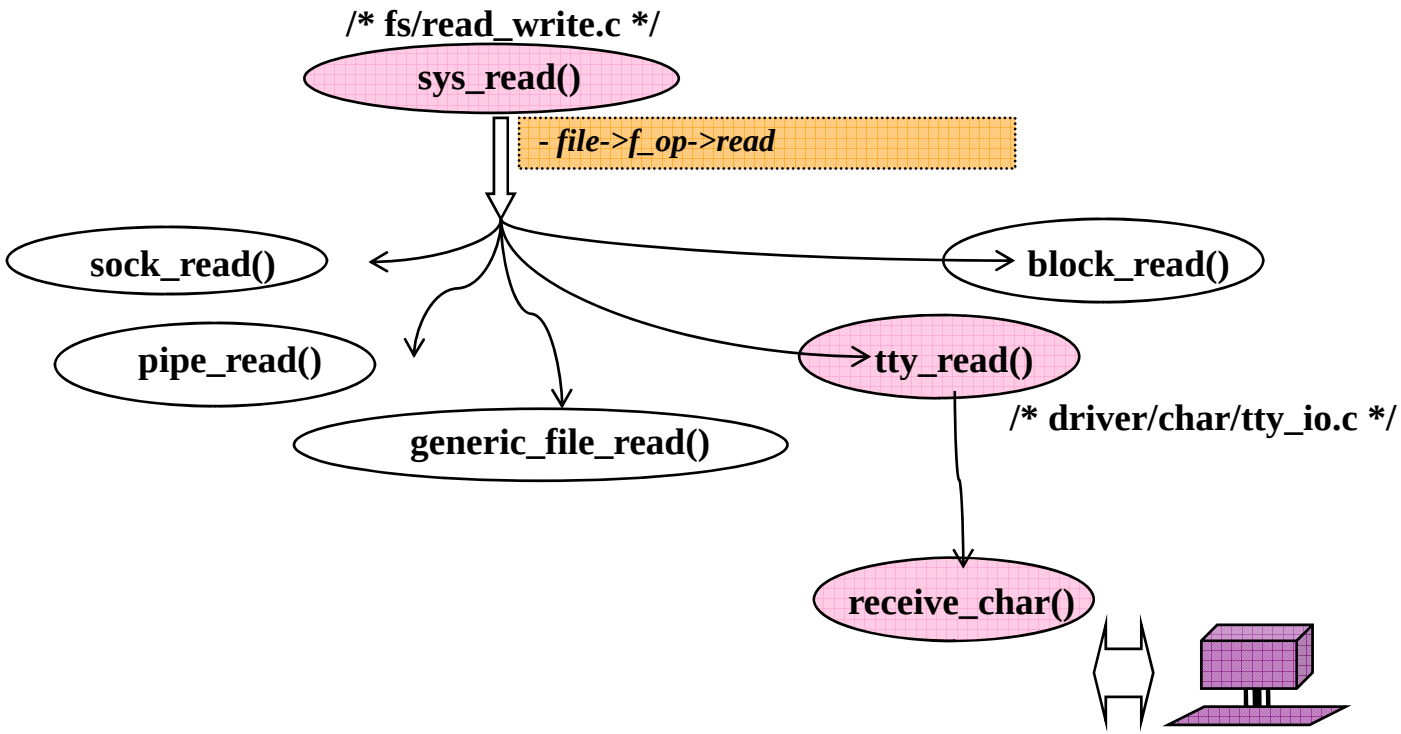
문자 디바이스 드라이버 구조 (3/4)

❑ 터미널 드라이버 열기 : tty_open()

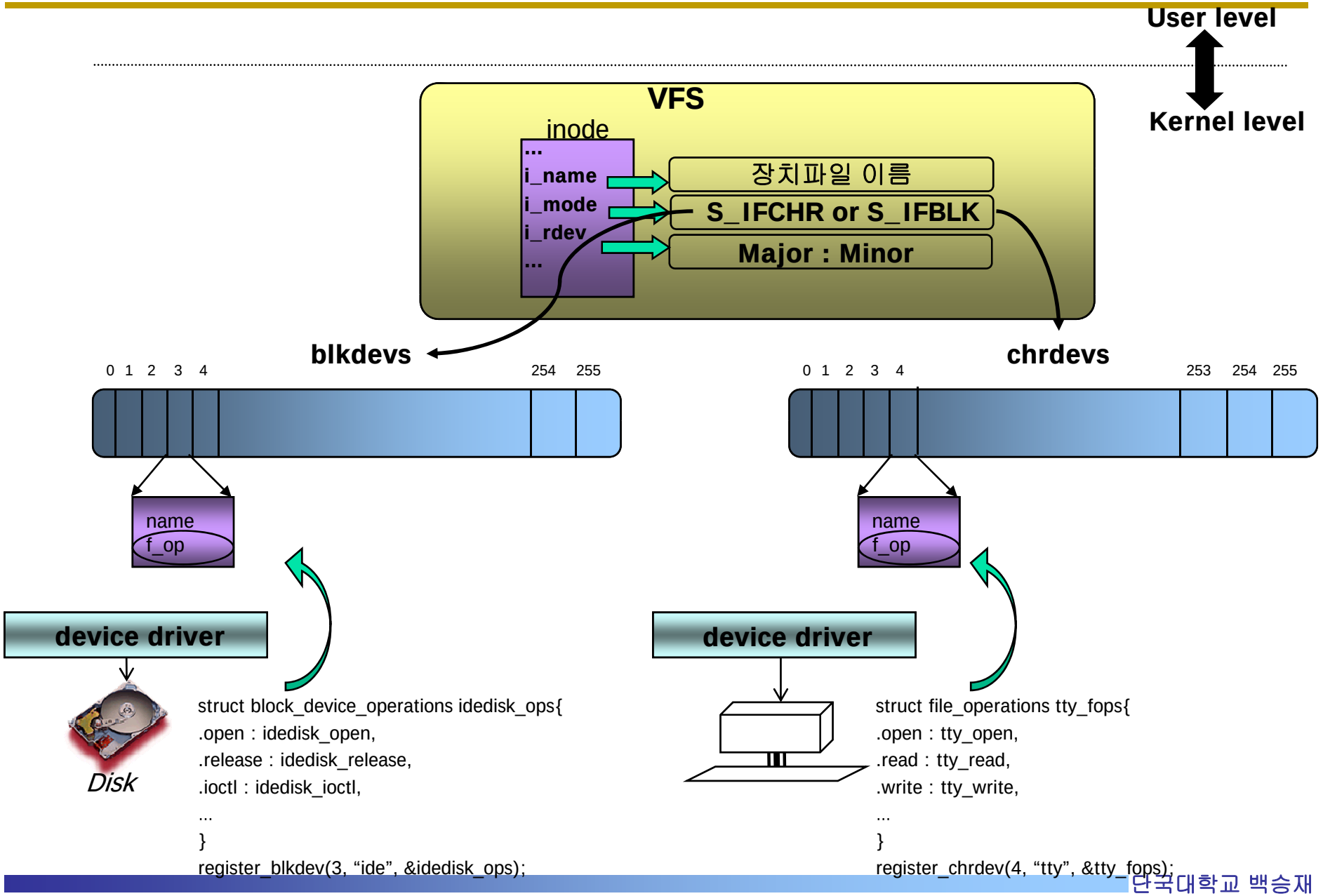


문자 디바이스 드라이버 구조 (4/4)

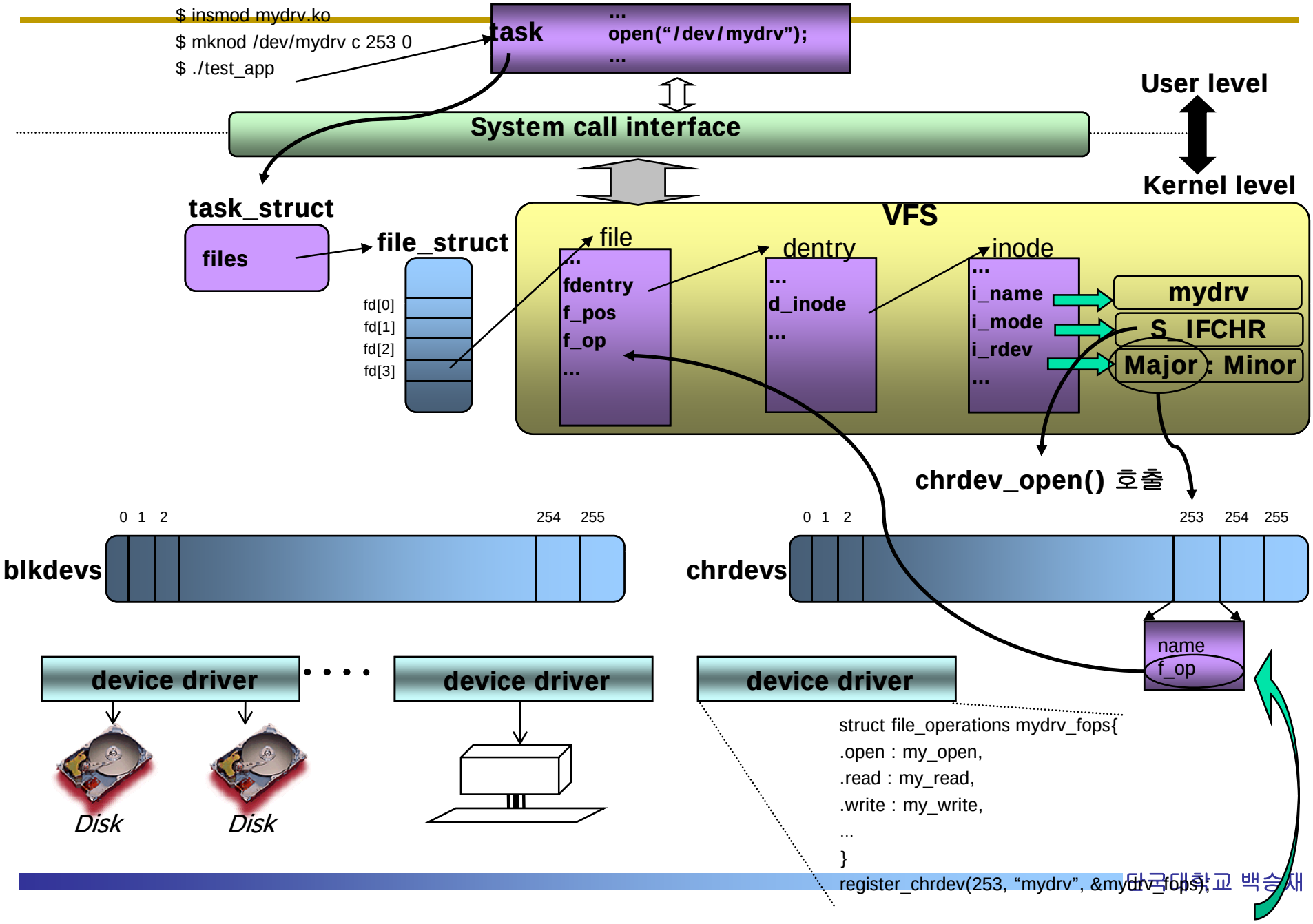
- 터미널 드라이버 읽기 : tty_read()



디바이스 드라이버 관리 구조



디바이스 드라이버와 장치파일 그리고 VFS



새로운 디바이스 드라이버 작성

- 새로운 디바이스 드라이버 작성 단계
  1. 디바이스 드라이버 구현 : entry point functions + file_operations
  2. 새로운 디바이스 드라이버를 위한 주 번호 할당
  3. 디바이스 드라이버 루틴 등록 : chrdevs[], blkdevs[]
  4. 장치 파일 생성

     # mknod /dev/mydrv [b|c] major_number minor_number
  5. 커널 컴파일 및 리부팅

모듈 프로그램 : Character Device Driver - 2.6

- 2.6 커널에서 character device driver

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/slab.h>
#include <linux/fs.h>
#include <asm/uaccess.h>

#define DEVICE_NAME    "mydrv"
#define MYDRV_MAX_LENGTH      4096
#define MIN(a, b) (((a) < (b)) ? (a) : (b))
static int MYDRV_MAJOR;
static char * mydrv_data;
static int mydrv_read_offset, mydrv_write_offset;

static int mydrv_open(struct inode *inode, struct file *file)
{
      if( MAJOR(inode->i_rdev) != MYDRV_MAJOR )
            return -1;
      return 0;
}

static int mydrv_release(struct inode * inode, struct file *file)
{
      if( MAJOR(inode->i_rdev) != MYDRV_MAJOR )
            return -1;
      return 0;
}

static int mydrv_ioctl(struct inode *inode, struct file *file, unsigned int cmd, unsigned long arg)
{
      printk("<0> mydrv_ioctl is invoked₩n");
      return 0;
}
```

모듈 프로그램 : Character Device Driver - 2.6

```
static ssize_t mydrv_read(struct file *file, char *buf, size_t count, loff_t *ppos)
{
	if( (buf == NULL) || (count < 0 ) )
		return -EINVAL;
	if( (mydrv_write_offset - mydrv_read_offset) <= 0 )
		return 0;
	count = MIN( (mydrv_write_offset - mydrv_read_offset), count );
	if( copy_to_user(buf, mydrv_data + mydrv_read_offset, count) )
		return -EFAULT;
	mydrv_read_offset += count;
	return count;
}

static ssize_t mydrv_write(struct file *file, const char *buf, size_t count, loff_t *ppos)
{
	if( (buf==NULL) || (count<0) )
		return -EINVAL;
	if( count+mydrv_write_offset >= MYDRV_MAX_LENGTH) {
		/* driver space is too small */
		return 0;
	}
	if( copy_from_user(mydrv_data + mydrv_write_offset, buf, count) )
		return -EFAULT;
	mydrv_write_offset += count;
	return count;
}
```

모듈 프로그램 : Character Device Driver - 2.6

```
struct file_operations mydrv_fops = {
	.owner = THIS_MODULE,
	.read = mydrv_read,
	.write = mydrv_write,
	.ioctl = mydrv_ioctl,
	.open = mydrv_open,
	.release = mydrv_release,
};

int mydrv_init(void)
{
	if( (MYDRV_MAJOR = register_chrdev(0, DEVICE_NAME, &mydrv_fops)) < 0 ){
		printk("<0> can't be registered₩n");
		return MYDRV_MAJOR;
	}
	printk("<0> major NO = %d₩n", MYDRV_MAJOR);
	if( (mydrv_data = (char *) kmalloc(MYDRV_MAX_LENGTH * sizeof(char), GFP_KERNEL)) == NULL ){
		unregister_chrdev(MYDRV_MAJOR, DEVICE_NAME);
		return -ENOMEM;
	}
	mydrv_read_offset = mydrv_write_offset = 0;
	return 0;
}

void mydrv_cleanup(void)
{
	kfree(mydrv_data);
	unregister_chrdev(MYDRV_MAJOR, DEVICE_NAME);
}

module_init(mydrv_init);
module_exit(mydrv_cleanup);
```

모듈 프로그램 : Character Device Driver – 2.6

- 2.6 커널에서 character device driver 모듈을 컴파일하기 위한 Makefile

```
MY_TARGET      := chr_test.ko
obj-m          := chr_test.o

KERNEL_DIR     := /lib/modules/$(shell uname -r)/build
MODULE_DIR     := /lib/modules/$(shell uname -r)/kernel/chr_test
PWD            := $(shell pwd)

default :
	$(MAKE) -C $(KERNEL_DIR) SUBDIRS=$(PWD) modules
install :
	mkdir -p $(MODULE_DIR)
	cp -f $(MY_TARGET) $(MODULE_DIR)
clean :
	$(MAKE) -C $(KERNEL_DIR) SUBDIRS=$(PWD) clean
```

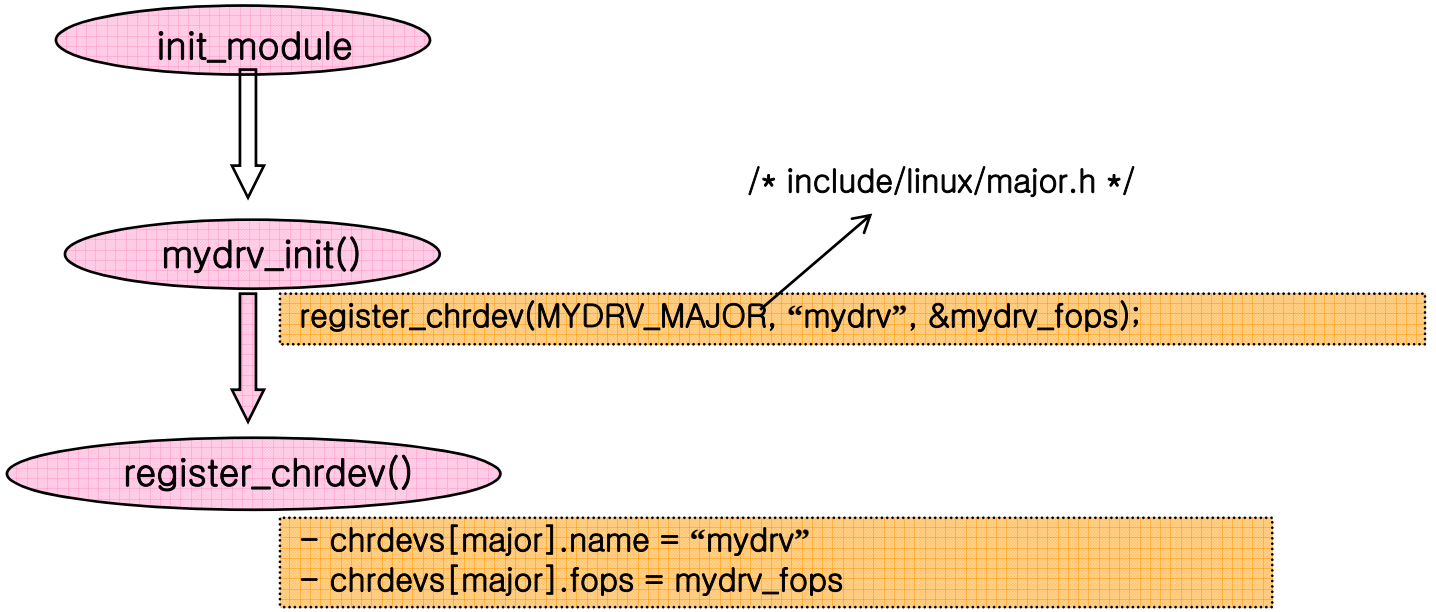
모듈 프로그램 : Character Device Driver – 2.6

- 2.6 커널에서 character device driver 테스트를 위한 명령어 수행

```
$ make
$ insmod  chr_test.ko
Major NO = 252
$mknod /dev/mydrv c 252 0
$echo “test string” >> /dev/mydrv
$cat /dev/mydrv
test string
$...
```

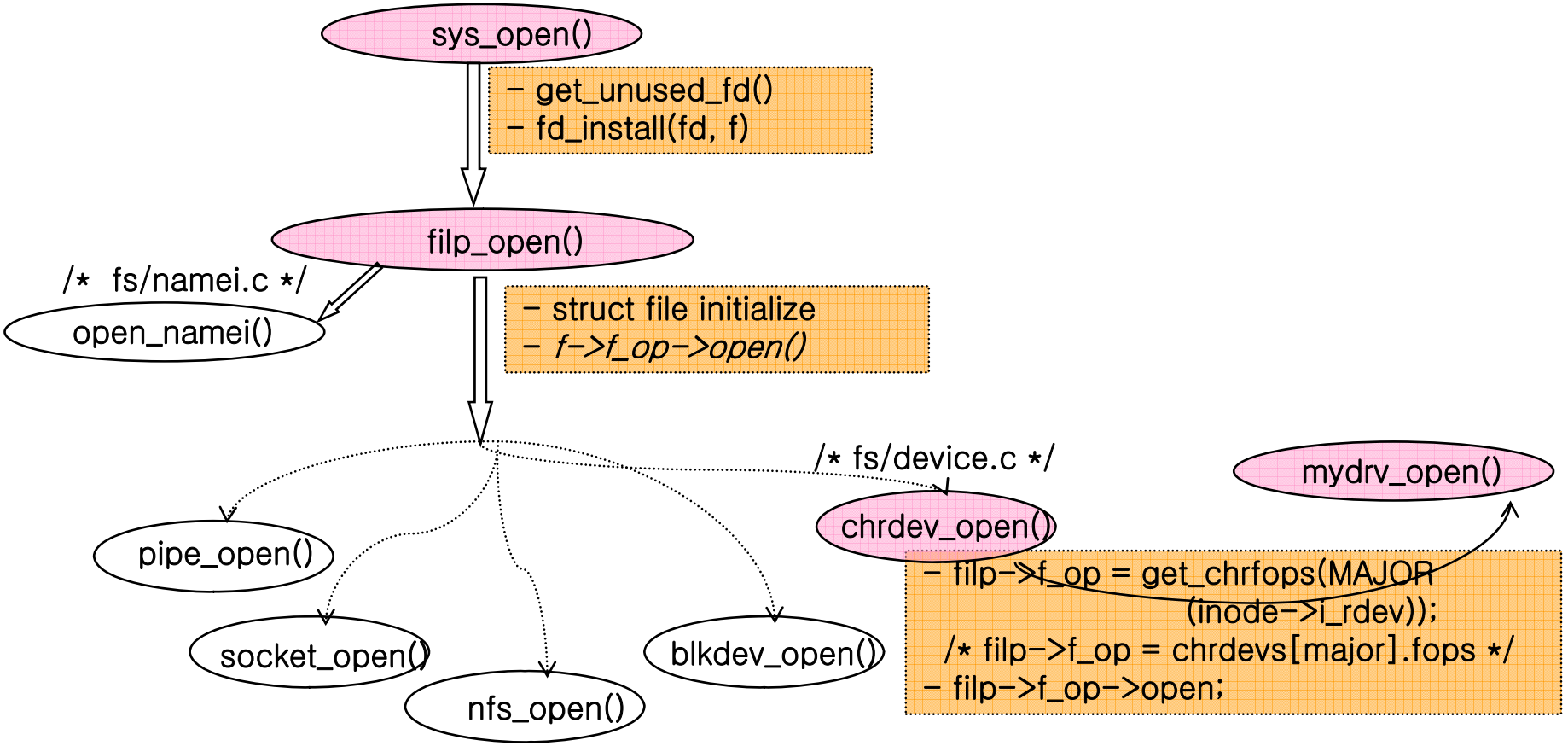
모듈 프로그램 : Character Device Driver

- 모듈 초기화



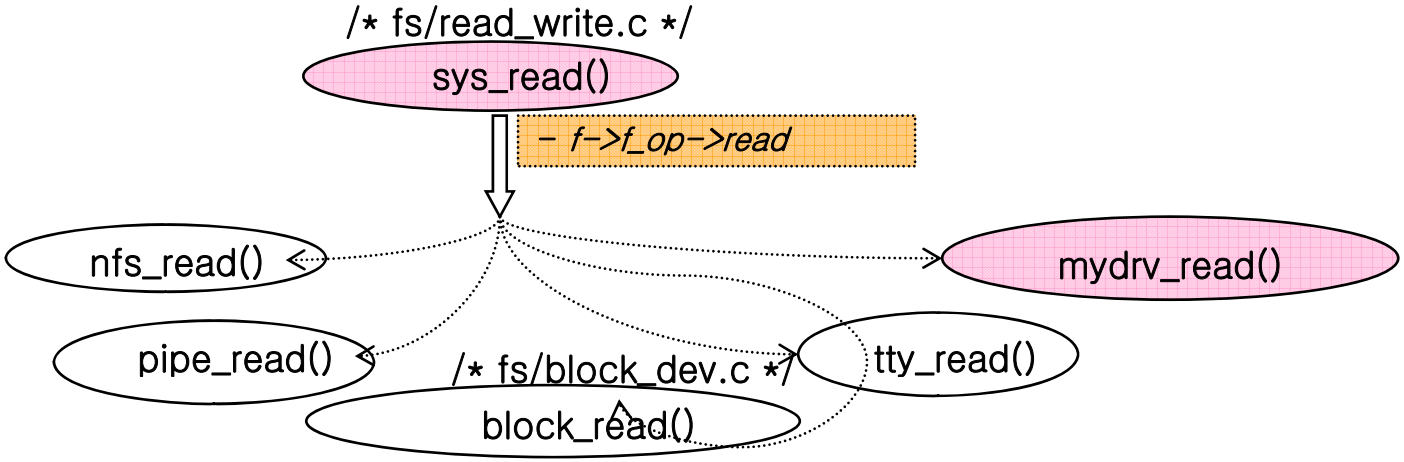
모듈 프로그램 : Character Device Driver

- 열기



모듈 프로그램 : Character Device Driver

❑ 읽기



모듈 프로그램 : Block Device Driver – 2.6

- 2.6 커널에서 block device driver

```
#include <linux/string.h>
#include <linux/slab.h>
#include <asm/atomic.h>
#include <linux/bio.h>
#include <linux/module.h>
#include <linux/moduleparam.h>
#include <linux/init.h>
#include <linux/pagemap.h>
#include <linux/blkdev.h>
#include <linux/genhd.h>
#include <linux/buffer_head.h>
#include <linux/backing-dev.h>
#include <linux/blkpg.h>
#include <linux/writeback.h>

#include <asm/uaccess.h>

#define DEVICE_NAME          "mydrv"
#define MYDRV_MAX_LENGTH      (8*1024*1024)
#define MYDRV_BLK_SIZE        512
#define MYDRV_TOTAL_BLK       (MYDRV_MAX_LENGTH/MYDRV_BLK_SIZE)

static int MYDRV_MAJOR = 0;
static char * mydrv_data;
struct request_queue *mydrv_queue;
struct gendisk      *mydrv_disk;
```

모듈 프로그램 : Block Device Driver – 2.6

```
static int mydrv_make_request(request_queue_t *q, struct bio *bio)
{
	struct block_device *bdev = bio->bi_bdev;
	int i;
	char *data;
	char *buffer;
	sector_t sector = bio->bi_sector;
	unsigned long len = bio->bi_size >> 9;
	struct bio_vec *bvec;
	int rw = bio_data_dir(bio);
	if (sector + len > get_capacity(bdev->bd_disk))		goto fail;
	if(rw == READA)		rw = READ;
	data  = mydrv_data + (sector * MYDRV_BLK_SIZE);
	bio_for_each_segment(bvec, bio, i){
		buffer = kmap(bvec->bv_page) + bvec->bv_offset;
		switch(rw){
			case READ  : memcpy(buffer, data, bvec->bv_len);
				break;
			case WRITE : memcpy(data, buffer, bvec->bv_len);
				break;
			default    : kunmap(bvec->bv_page);
				goto fail;
		}
		kunmap(bvec->bv_page);
		data += bvec->bv_len;
	}
	bio_endio(bio, bio->bi_size, 0); //until 2.6.22
	//bio_endio(bio, 0); //after 2.6.24
	return 0;
fail:
	bio_io_error(bio, bio->bi_size); //until 2.6.22
	//bio_io_error(bio); //after 2.6.24
	return 0;
}
```

모듈 프로그램 : Block Device Driver – 2.6

```
int mydrv_open(struct inode *inode, struct file *filp)
{
	printk("<0>%s:%s:%d₩n", __FILE__, __FUNCTION__, __LINE__);
	return 0;
}

int mydrv_release (struct inode *inode, struct file *filp)
{
	printk("<0>%s:%s:%d₩n", __FILE__, __FUNCTION__, __LINE__);
	return 0;
}

int mydrv_ioctl(struct inode *inode, struct file *filp, unsigned int cmd, unsigned long arg)
{
	printk("<0>%s:%s:%d₩n", __FILE__, __FUNCTION__, __LINE__);
	printk("<0>CMD = %d₩n", cmd);
	return 0;
}

static struct block_device_operations mydrv_fops =
{
	.owner   = THIS_MODULE,
	.open    = mydrv_open,
	.release = mydrv_release,
	.ioctl   = mydrv_ioctl,
};
```

모듈 프로그램 : Block Device Driver – 2.6

```
int mydrv_init(void)
{
	if( (MYDRV_MAJOR = register_blkdev(MYDRV_MAJOR, DEVICE_NAME)) < 0 ){
		printk("<0> can't be registered₩n");
		return -EIO;
	}
	printk("<0> major NO = %d₩n", MYDRV_MAJOR);
	if( (mydrv_data = vmalloc(MYDRV_MAX_LENGTH)) == NULL ){
		unregister_chrdev(MYDRV_MAJOR, DEVICE_NAME);
		printk("<0> vmalloc failed₩n");
		return -ENOMEM;
	}
	if( (mydrv_disk = alloc_disk(1)) == NULL ){
		printk("<0> alloc_dirk failed₩n");
		unregister_chrdev(MYDRV_MAJOR, DEVICE_NAME);
		vfree(mydrv_data);
		return -EIO;
	}
	if( (mydrv_queue = blk_alloc_queue(GFP_KERNEL)) == NULL ){
		printk("<0> blk_alloc_queue failed₩n");
		unregister_chrdev(MYDRV_MAJOR, DEVICE_NAME);
		vfree(mydrv_data);
		put_disk(mydrv_disk);
		return -EIO;
	}
	blk_queue_make_request(mydrv_queue, &mydrv_make_request);
	blk_queue_hardsect_size(mydrv_queue, MYDRV_BLK_SIZE);
	mydrv_disk->major = MYDRV_MAJOR;
	mydrv_disk->first_minor = 0;
	mydrv_disk->fops = &mydrv_fops;
	mydrv_disk->queue = mydrv_queue;
	sprintf(mydrv_disk->disk_name,  "mydrv");
	set_capacity(mydrv_disk, MYDRV_TOTAL_BLK);
	add_disk(mydrv_disk);
	return 0;
}
```

모듈 프로그램 : Block Device Driver – 2.6

```
void mydrv_exit(void)
{
        vfree(mydrv_data);
        del_gendisk(mydrv_disk);
        put_disk(mydrv_disk);

        unregister_blkdev(MYDRV_MAJOR, DEVICE_NAME );
}

module_init(mydrv_init);
module_exit(mydrv_exit);

MODULE_LICENSE("GPL");
```

모듈 프로그램 : Block Device Driver – 2.6

- 2.6 커널에서 block device driver 테스트를 위한 명령어 수행

```
$ make
$ insmod  blk_test.ko
Major NO = 252
$mknod /dev/mydrv b 252 0
$mke2fs /dev/mydrv
$mkdir test
$mount –t ext2  -o loop /dev/mydrv ./test
$vi hello.c
$ls
.   ..   hello.c
$gcc –O2 –o hello hello.c
$ls
.   ..    hello    hello.c
$
```

모듈 프로그램 : Network Device Driver – 2.6

- 2.6 커널에서 network device driver

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/netdevice.h>
#include <linux/ethtool.h>
#include <linux/pci.h>

struct net_device virtnet;
struct pci_dev *pdev = NULL;
struct net_device_stats virt_stats;

int virtnet_open(struct net_device *dev)
{
	netif_start_queue(dev);
	return 0;
}
int virtnet_release(struct net_device *dev)
{
	netif_stop_queue(dev);
	return 0;
}
static int virtnet_ioctl(struct net_device *dev, struct ifreq *rq, int cmd)
{
	return 0;
}
static int virtnet_xmit(struct sk_buff *skb, struct net_device *dev)
{
	dev_kfree_skb(skb);
	return 0;
}
```

모듈 프로그램 : Network Device Driver – 2.6

```
static int virtnet_probe(struct net_device *dev, struct pci_dev *pdev)
{
	int ret;
	unsigned char pci_rev;

	pdev = pci_find_device(PCI_VENDOR_ID_REALTEK, PCI_DEVICE_ID_REALTEK_8139, pdev);

	if(pdev) printk("<0>probed for rtl 8139₩n");
	else	 printk("<0>Rtl8139 card not present₩n");

	pci_read_config_byte(pdev, PCI_REVISION_ID, &pci_rev);

	if(ret = pci_enable_device(pdev)){
		printk("<0>Error enabling the device₩n");
		return ret;
	}
	if(pdev->irq < 2){
		printk("<0>Invalid irq number₩n");
		ret = -EIO;
	}else{
		printk("<0>Irq Obtained is %d", pdev->irq);
		dev->irq = pdev->irq;
	}

	return 0;
}

static void virtnet_get_drvinfo(struct net_device *dev, struct ethtool_drvinfo *info)
{
	strcpy(info->driver, "virtnet");
}
```

모듈 프로그램 : Network Device Driver – 2.6

```
static const struct ethtool_ops virtnet_ethtool_ops = {
        .get_drvinfo    = virtnet_get_drvinfo,
};

static struct net_device_stats *virtnet_get_stats(struct net_device *dev)
{
        return &virt_stats;
}

int virtnet_init(struct net_device *dev)
{
        int ret;
        if((ret = virtnet_probe(dev, pdev)) != 0)
                return ret;

        dev->open = virtnet_open;
        dev->hard_start_xmit = virtnet_xmit;
        dev->stop = virtnet_release;
        dev->get_stats = virtnet_get_stats;
        dev->do_ioctl = virtnet_ioctl;
        dev->ethtool_ops = &virtnet_ethtool_ops;

        return 0;
}
```

모듈 프로그램 : Network Device Driver – 2.6


```
int virtnet_init_module(void)
{
	int result;
	virtnet.init = virtnet_init;

	printk("<0>%s:%s:%d₩n", __FILE__, __FUNCTION__, __LINE__);

	strcpy(virtnet.name, "virtnet");
	if((result = register_netdev(&virtnet))){
		printk("virtnet: Error %d initializing card virtnet card", result);
		return result;
	}
	return 0;
}

void virtnet_cleanup(void)
{
	printk("<0> Cleaning Up the Module₩n");
	unregister_netdev(&virtnet);
	return;
}

module_init(virtnet_init_module);
module_exit(virtnet_cleanup);

MODULE_LICENSE("GPL");
```

모듈 프로그램 : Network Device Driver

- Network device driver 테스트를 위한 명령어 수행

```
$ make
$ insmod  net_test.ko
$ ifconfig  virtnet 192.168.13.83 up
$ ifconfig
...
$ ping 192.168.122.1
...
$
```