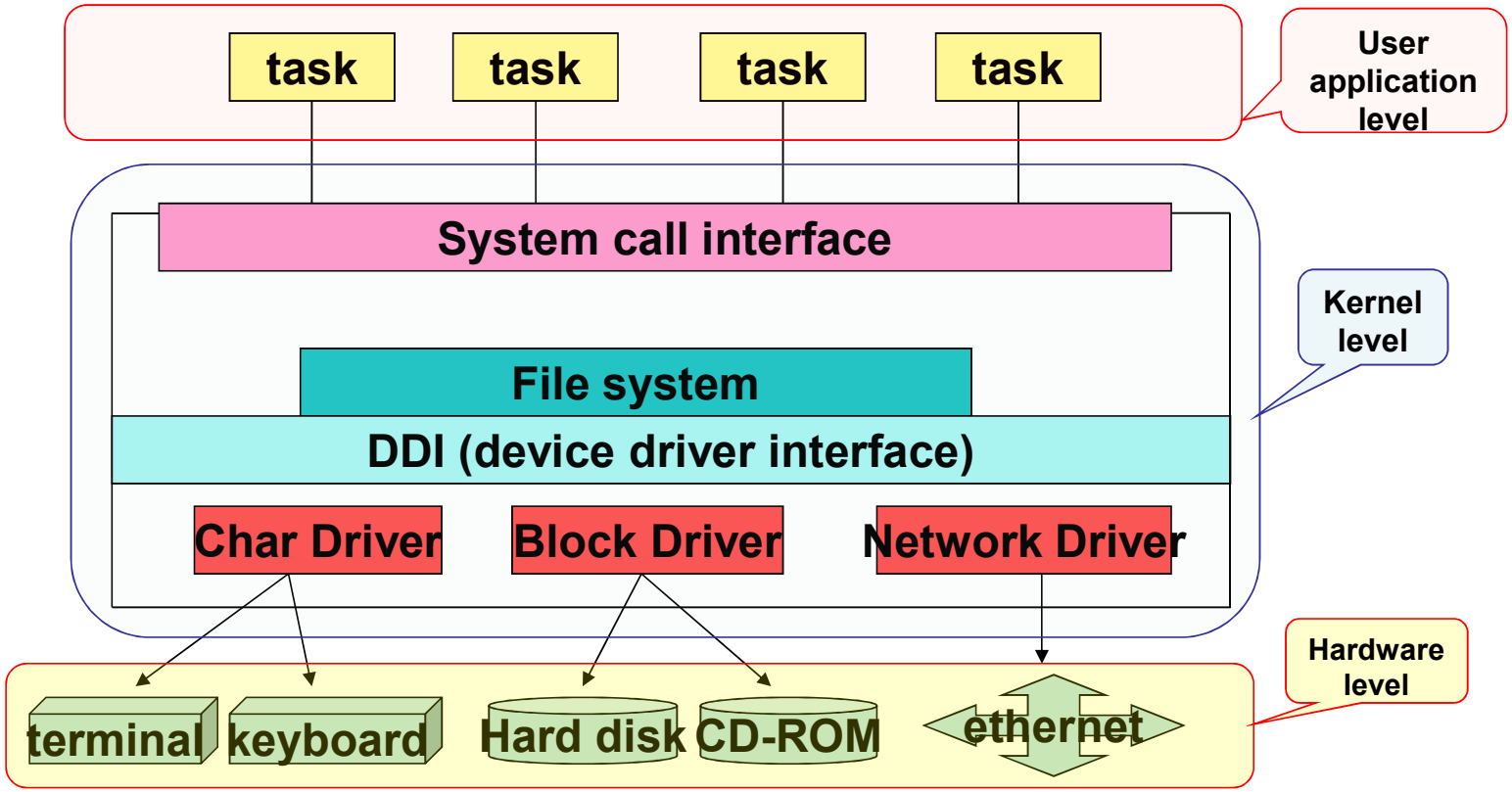
Device Driver

Device Driver

- 물리적인 hardware 장치를 다루고 관리하는 software
- user application이 driver에게 요청을 하면, driver는 hardware를 구동시켜서 목적을 달성
- major number와 minor number를 이용하여 각각의 devices을 구분하여 사용
- device와 system memory 간에 data의 전달을 담당하는 kernel 내부 기능

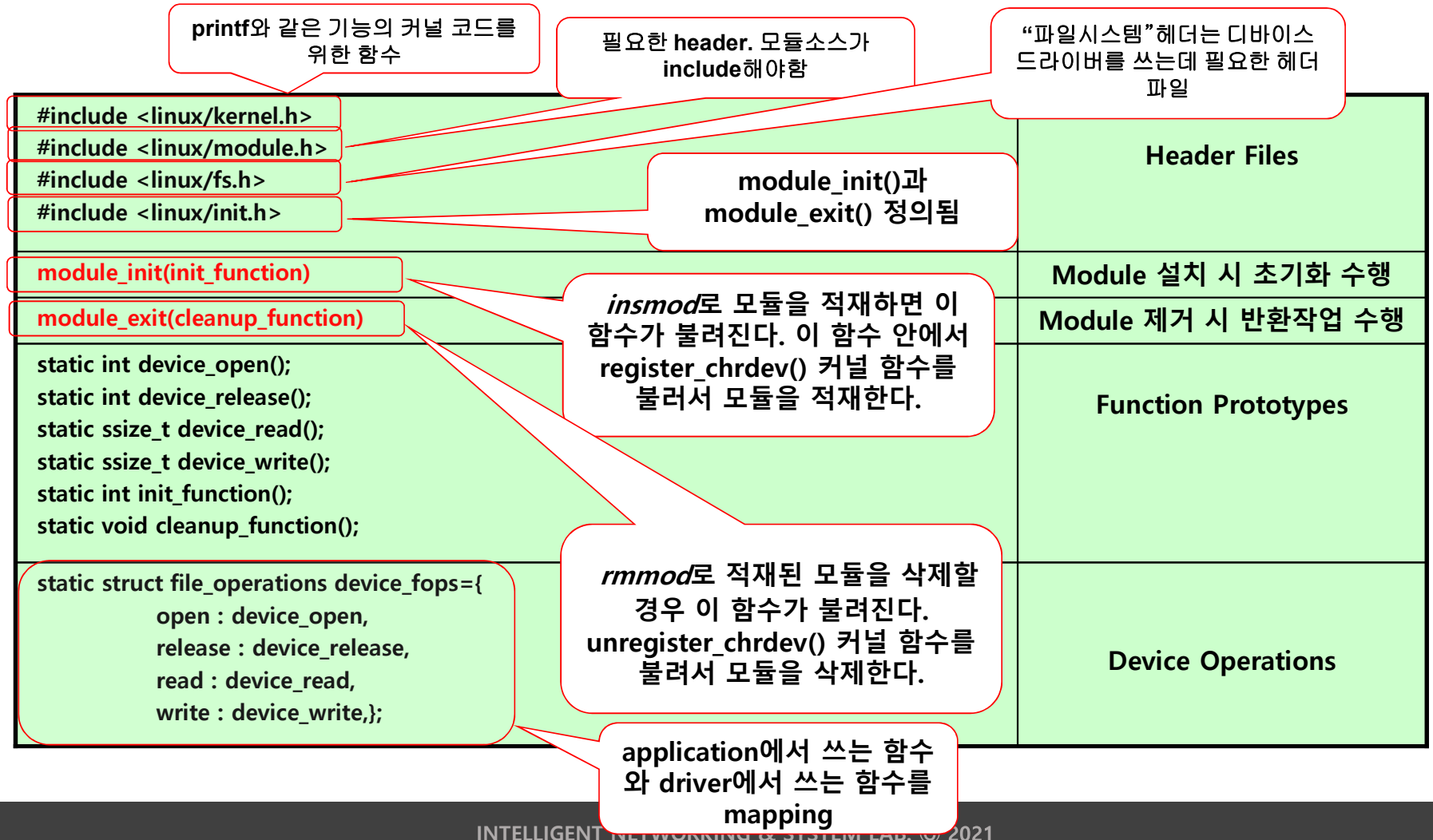
Device Driver



Device Driver 종류

Driver 종류	설 명	등록함수명
Char Driver	**device**를 **file**처럼 취급하고 접근하여 직접 **read/write**를 수행, **data** 형태는 **stream** 방식으로 전송 **EX) console, keyboard, serial port driver**등	**register_ chrdev()**
Block Driver	**disk**와 같이 **file system**을 기반으로 일정한 **block** 단위로 **data read/write**를 수행 **EX) floppy disk, hard disk, CD-ROM driver** 등	**register_ blkdev()**
Network Driver	**network**의 **physical layer**와 **frame** 단위의 데이터를 송수신 **EX) Ethernet device driver(eth0)**	**register_ netdev()**

기본적인 Device Driver



Device Driver 작성

test_dd.c

```
#include <linux/module.h>
#include <linux/fs.h>
#include <linux/kernel.h>
#include <linux/init.h>
#define DEV_NAME "testdd"

static int test_major = 0;

static int result;

////////////// 함 수 정 의 //////////////
static int device_open (struct inode *inode, struct file *filp);
static int device_release (struct inode *inode, struct file *filp);
static ssize_t device_read (struct file *filp, char *buf, size_t count, loff_t *f_pos);
static ssize_t device_write (struct file *filp, char *buf, size_t count, loff_t *f_pos);
static int device_ioctl(struct inode *inode, struct file *filp, unsigned int cmd, unsigned int arg);
static int sample_init();
static void sample_cleanup();
```

모듈을 커널에 적재하는 함수 register_chrdev()의 parameter로 써 0으로 설정하면 major값을 뒤에서부터 자동으로 빈자리로 등록시킨다

register_chrdev()함수가 return하는 값을 넘겨받음. 0보다 적으면 모듈 등록 실패

application에서 open()함수로 driver를 호출할때 실행되는 함수

application에서 close()함수로 driver를 닫을때 실행되는 함수

Device Driver 작성

```
int device_open (struct inode *inode, struct file *filp)
{
    printk("test_open start!!!\n");
    MOD_INC_USE_COUNT;
    return 0;
}
int device_release (struct inode *inode, struct file *filp)
{
    MOD_DEC_USE_COUNT;
    return 0;
}
ssize_t device_read (struct file *filp, char *buf, size_t count, loff_t *f_pos)
{
    printk("read() ^^\n");
    return 0;
}
ssize_t device_write (struct file *filp, char *buf, size_t count, loff_t *f_pos)
{
    printk("write() ^^\n");
    return 0;
}
```

이 함수가 호출될때마다 count가 증가한다.

이 함수가 호출될때마다 count가 감소한다.

Application에서 read함수를 호출하면 terminal에 문자 출력

Application에서 write함수를 호출하면 terminal에 문자 출력

Device Driver 작성

```
int device_ioctl(struct inode *inode, struct file *filp, unsigned int cmd, unsigned int arg)
{
        printk("ioctl() ^^\n");
        return 0;
}
```

ioctl()호출이 성공하면 문자 출력

```
struct file_operations test_fops = {
        open:           device_open,
        read:           device_read,
        write:          device_write,
        ioctl:          device_ioctl,
        release:        device_release,
};
```

파일연산구조체는 char device driver에 대한 모든 연산을 포함한다. 즉 여기에 정의된 함수를 통해서 커널이 접근하게 된다. 즉 application에서 open함수를 호출하면 이는 driver의 device_open()과 mapping되어있어 이를 실행하게 된다.

```
module_init(sample_init);
module_exit(sample_cleanup);
```

*insmod*로 불려지는 함수

*rmmod*로 불려지는 함수

Device Driver 작성

```
int sample_init(void)
{
    result = register_chrdev(test_major, DEV_NAME, &test_fops);

    if (result < 0) {
        printk(KERN_WARNING "%s: can't get major %d\n", DEV_NAME, test_major);
        return result;
    }

    printk("<1> init module success!!.. %s major number : %d\n", DEV_NAME, result);
    return 0;
}
void sample_cleanup(void)
{
    if( !unregister_chrdev(result, DEV_NAME) )
        printk("%s cleanup_module success...\n", DEV_NAME);
    else
        printk("%s cleanup_module fail...\n", DEV_NAME);
}
```

driver를 커널에 모듈로 등록하는 함수, result에 major 값이 넘어옴

driver를 커널에서 삭제하는 함수

Device Driver 검증 파일 작성

```
#include <stdio.h>
#include <fcntl.h>
#include <string.h>
#include <errno.h>

int fd;

int main(int argc, char **argv)
{

    if( (fd = open("/dev/testdd", O_RDWR)) < 3) {
            fprintf(stderr, "/dev/testdd device file open error!! .. %s\n",strerror(errno));
            return 0;
    }
     read(fd, 0, 0);
     ioctl(fd, NULL,0);
     close(fd);
     return 0;
}
```

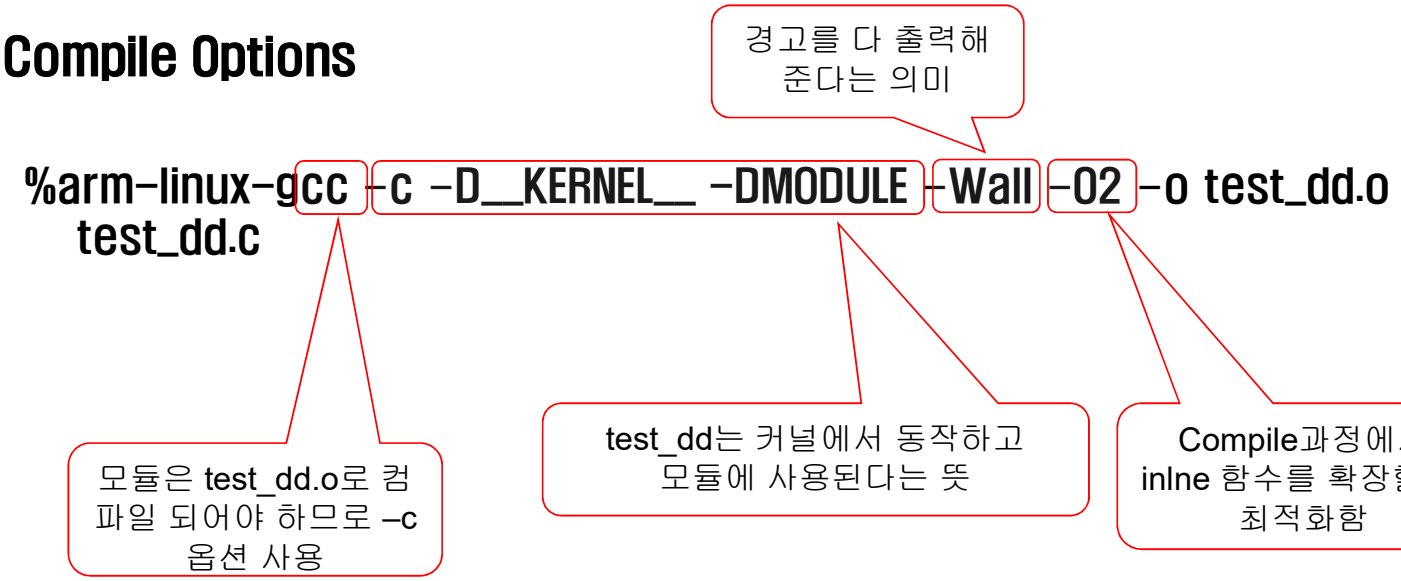
File에 대한 제어와 설정

Error에 관한 header file

mknod를 통하여 /dev에 만들어진 node(file)을 open한다.

Makefile

- **Compile Options**



Makefile

```
CC   =arm-linux-gcc

INCLUDEDIR = /usr/local/pxa255/linux-2.4.19-pro3_usb20/include

CFLAGS = -D__KERNEL__ -DMODULE -Wall -O2

CFLAGS2 = -I$(INCLUDEDIR)

All: test_dd test_app

test_dd : test_dd.c
	$(CC) $(CFLAGS) $(CFLAGS2) -c -o test_dd.o test_dd.c

test_app : test_app.c
	$(CC) -o test_app test_app.c

clean :
	rm -rf test_dd.o test_app
```

target

command

주의할 점은 이 공백은
반드시 Tab 공백

Macro 부분
C에서 define 역할

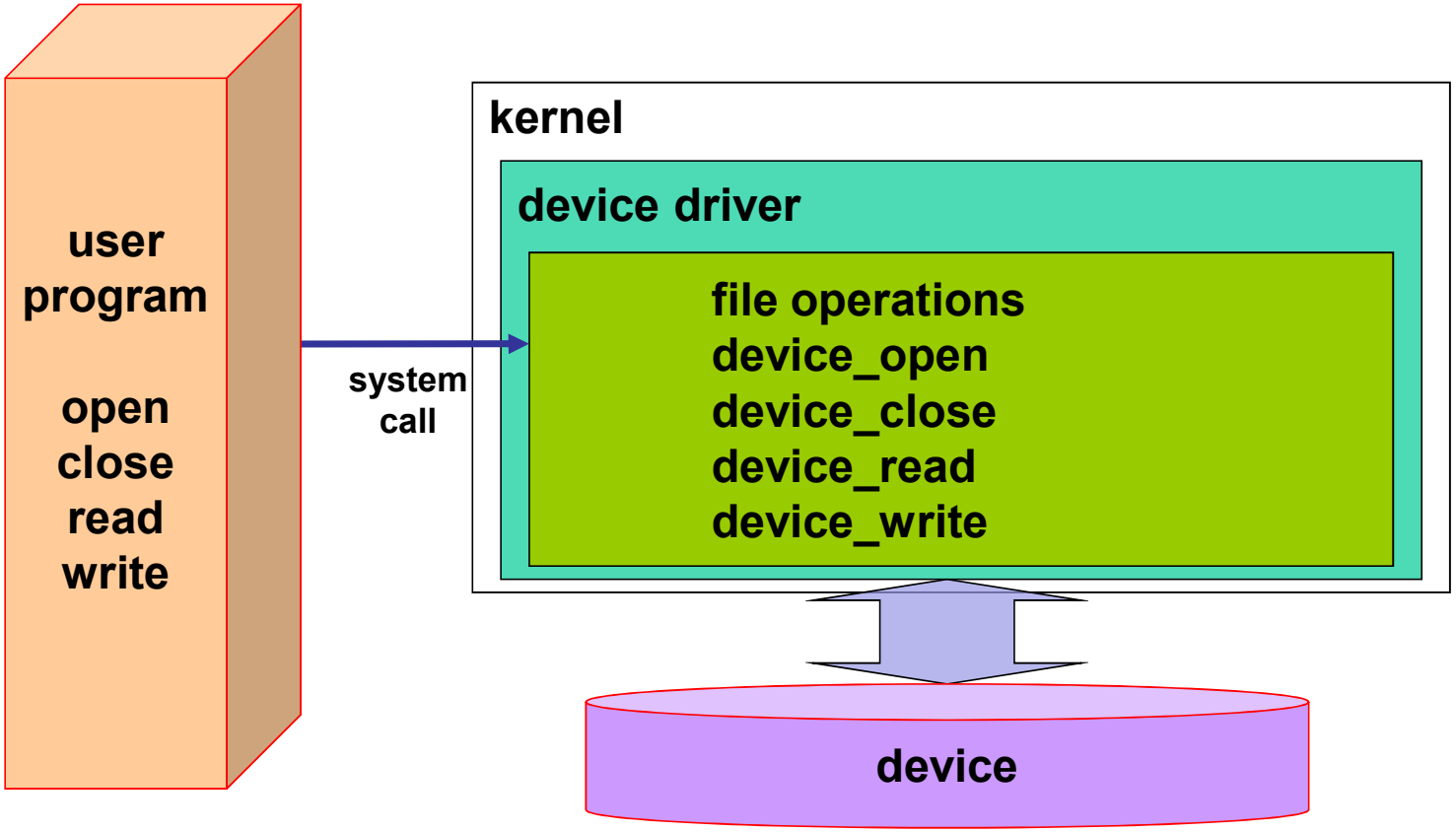
Command 부분
파일들의 dependency를 설정

Driver를 커널에 모듈로 등록 및 해제

- **Char Device Driver 등록 방법**
  - 외부와 device driver는 file interface (node를 의미)를 통해 연결
  - Device driver는 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 자동으로 할당
    - Name : device의 이름
    - Fops : device에 대한 file 연산 함수들
  - 해제 : int unregister_chrdev(unsigned int major, const char *name)

Driver를 커널에 모듈로 등록 및 해제



Driver를 커널에 모듈로 등록 및 해제

- **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
  - ✓ register_chrdev()로 장치를 등록할 때 major number를 지정
  - ✓ 같은 major number가 등록되어 있으면 등록 실패
  - ✓ Major와 minor 번호는 파일의 정보를 담고 있는 inode의 i_rdev에 16bit로 저장된다. 상위 8bit는 major, 하위 8bit는 minor이다.

Driver를 커널에 모듈로 등록 및 해제

- **mknod 명령으로 디바이스 드라이버에 접근할 수 있는 장치 파일 생성**
  - ✓ mknod [device file name] [type] [major] [minor]

    Ex] %mknod test_dd c 252 0

    C는 char device drive의미,
    block device drive는 b를 사용
- **mdev_t : 장치의 major, minor number를 표현하는 자료구조**
  - ✓ MAJOR() : kdev_t에서 major number를 얻어내는 매크로

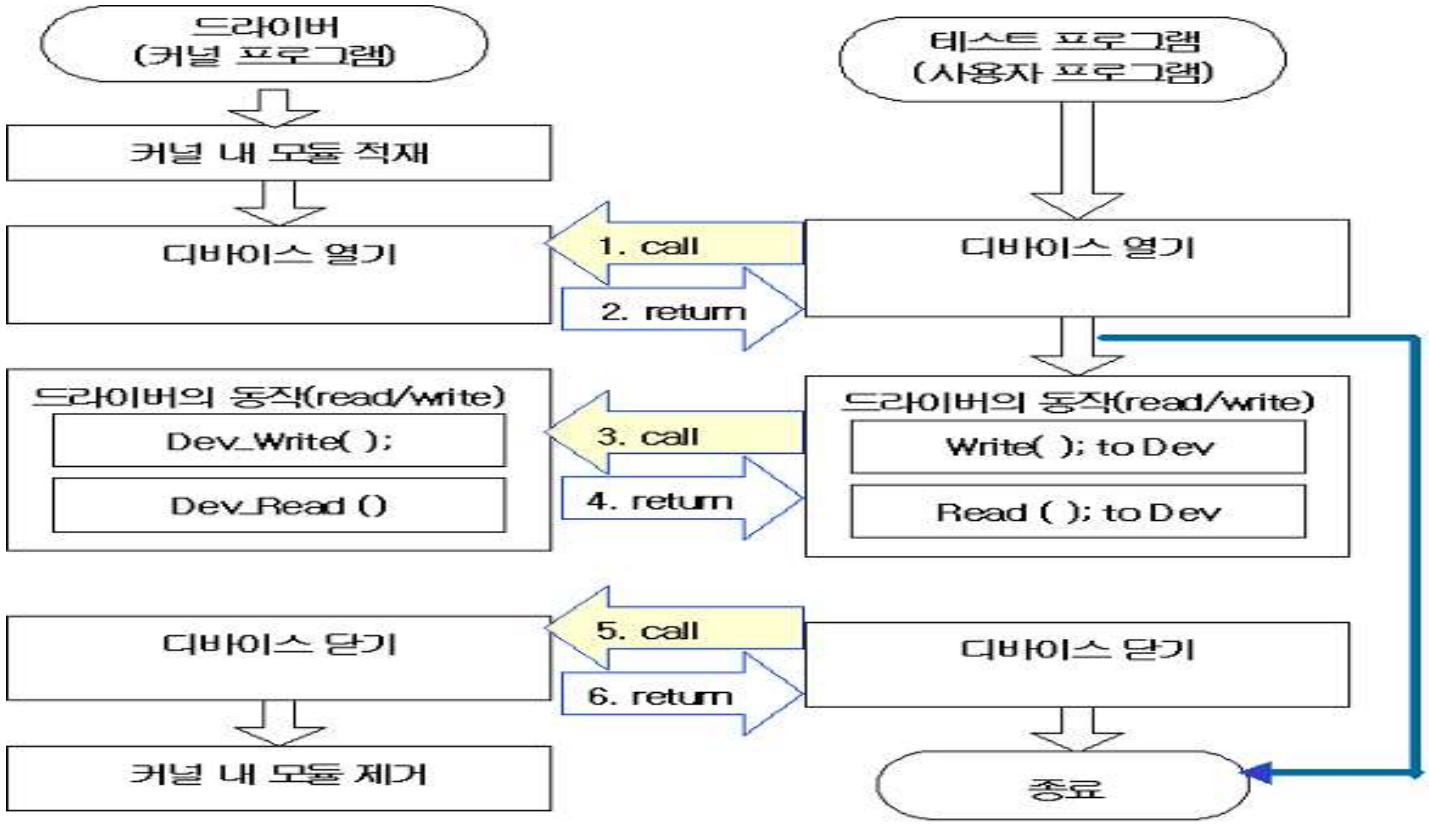
    Ex] MAJOR(inode->i_rdev);
  - ✓ MINOR() : kdev_t에서 minor number를 얻어내는 매크로
  - ✓ cat /proc/devices 명령으로 현재 로드된 디바이스 드라이버 확인

Driver를 커널에 모듈로 등록 및 해제

이 름	용 도
insmod	**module**을 설치**(install)**
rmmod	실행중인 **modules**을 제거**(unload)**
lsmod	**Load**된 **module**들의 정보를 표시
depmod	**Module**들의 **symbol**들을 이용하여 **Makefile**과 유사한 **dependency file**을 생성
modprobe	**depmod**명령으로 생성된 **dependency**를 이용하여 지정된 디렉토리의 **module**들과 연관된 **module**들을 자동으로 **load**
modinfo	목적화일을 검사해서 관련된 정보를 표시

Driver를 커널에 모듈로 등록 및 해제

- Device Driver의 동작 과정



Driver를 커널에 모듈로 등록 및 해제 (실습)

- 만든 드라이버 소스를 test_dd.c, test application은 test_app.c, 그리고 makefile은 Makefile로 작성 후 저장한다.

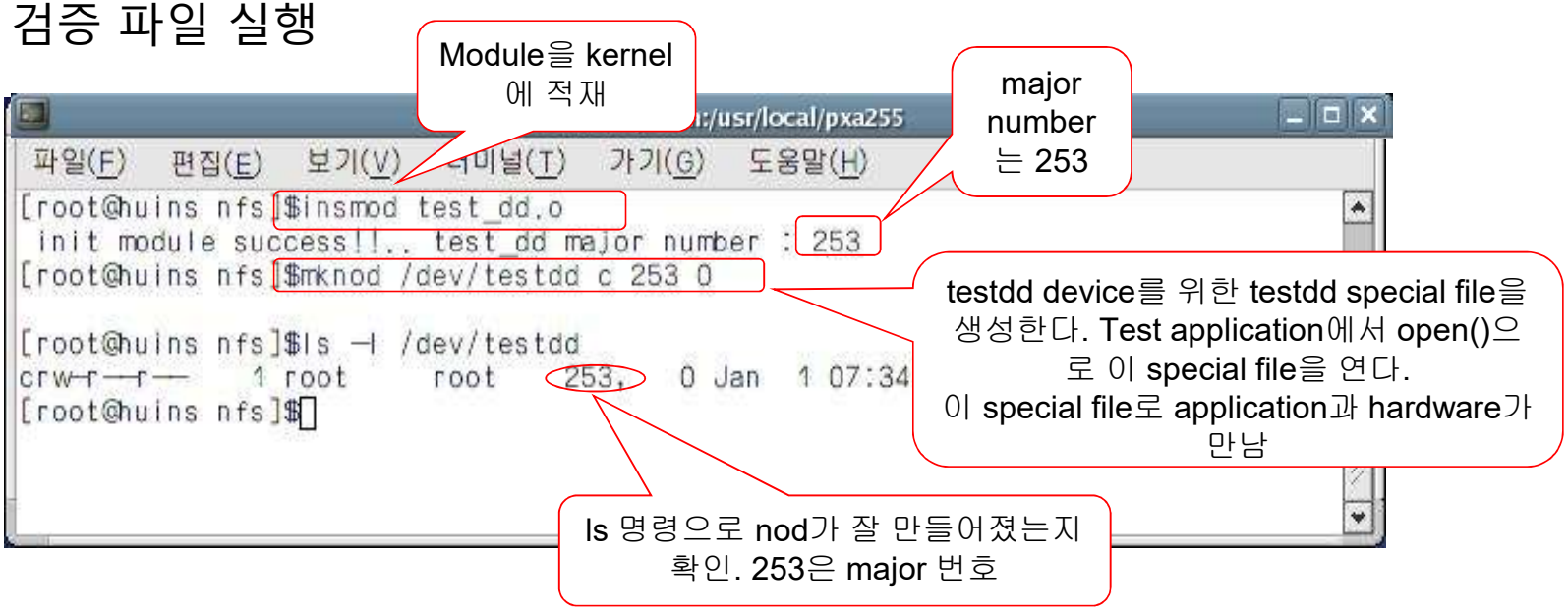
- make 명령어를 이용하여 위의 2 files을 컴파일한다.

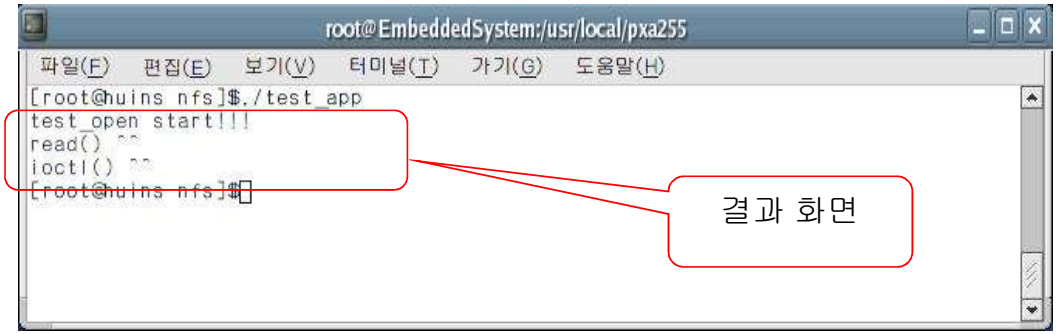
  % make

- 생성된 test_dd.o 와 test_app를 target board (Raspberry Pi) 으로 전송한다.

Driver를 커널에 모듈로 등록 및 해제 (실습)

- 모듈 적재 및 검증 파일 실행





E N D