

# 제 10 장

## 디바이스 드라이버

ACS30021

고급 프로그래밍

나보균 (bkna@kpu.ac.kr)

컴퓨터 공학과  
한국산업기술 대학교

# 학습 목표

---

- ❑ 디바이스 드라이버(장치 구동 프로그램) 정의
- ❑ 디바이스 종류
- ❑ 디바이스 드라이버와 커널의 인터페이싱
- ❑ 디바이스 제어
- ❑ 디바이스 드라이버 모듈화
- ❑ 디바이스 드라이버 작성 기초

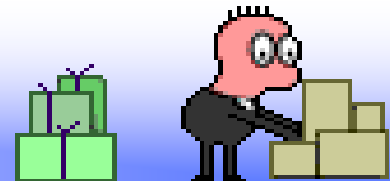
# 디바이스 드라이버 소개

## □ 디바이스 드라이버 정의

- ✓ 디바이스를 구동하는 커널 내부 S/W Component
- ✓ 시스템이 지원하는 H/W를 응용 프로그램에서 사용할 수 있도록 커널에서 제공하는 라이브러리

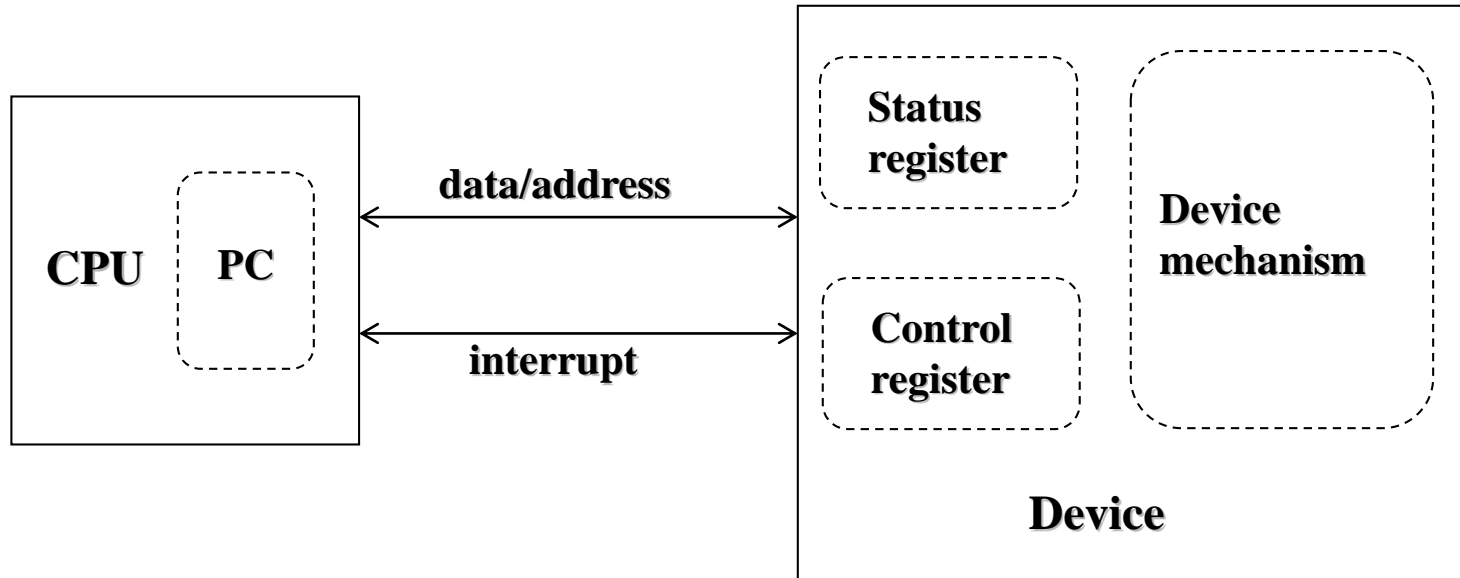
## □ 디바이스 드라이버 기능

- ✓ 디바이스 제어 (하드웨어 인터페이스)
- ✓ 커널과 디바이스간에 통로 (커널 인터페이스)
- ✓ Dynamic loading/unloading (초기화 인터페이스)
- ✓ 다중 접근 제어 (access control, queuing, ...)
- ✓ 사용자에게 디바이스 추상화 제공



# 디바이스 종류

## □ 디바이스 추상화



## □ 디바이스 제어

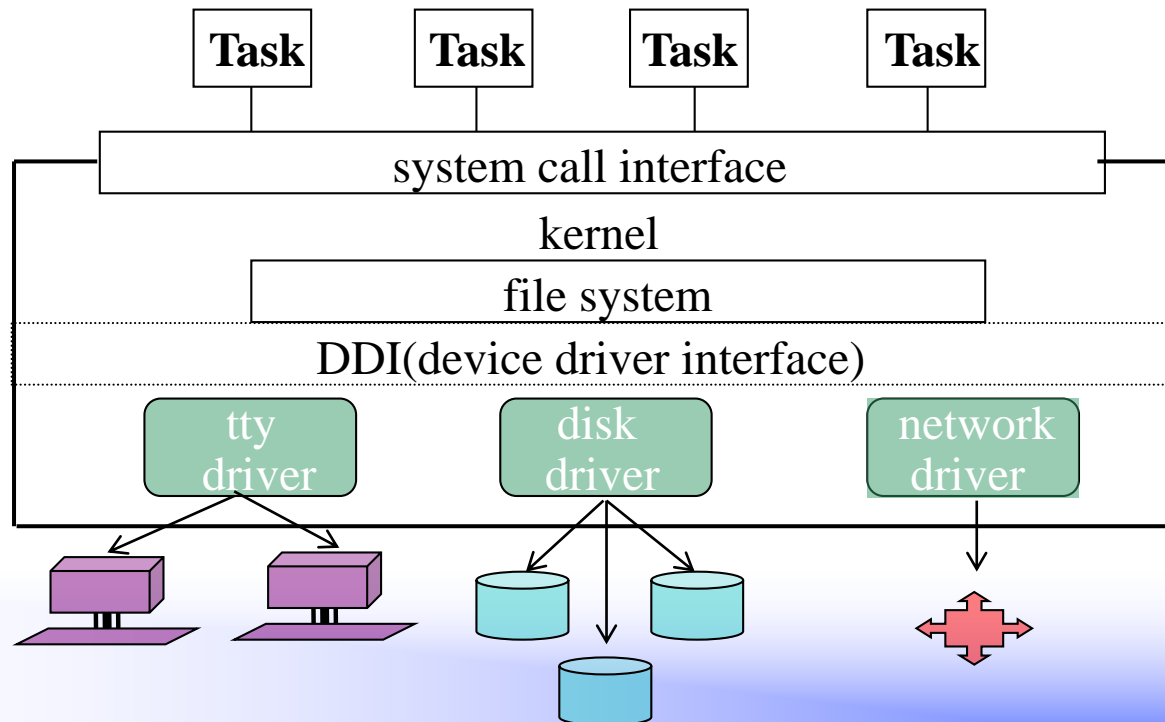
- ✓ 레지스터 접근 방법
- ✓ 이벤트 인식 방법
- ✓ 데이터 전달 방법
- ✓ 사용자 인터페이스 제공 방법

☞ **ARM에서는?**

# 디바이스 드라이버

## ❑ 디바이스 드라이버 in Linux

- ✓ 커널과 주변 장치간에 데이터 전달
- ✓ 문자 장치 드라이버, 블록 장치 드라이버, 네트워크 장치 드라이버
- ✓ 드라이버는 잘 정의된 인터페이스로 커널과 연결됨 (DDI, DKI)



# 디바이스 드라이버와 커널 모듈화

## □ 응용프로그램이 커널에 자원 처리를 요청하는 방법

### ✓ 시스템 호출

- 소프트웨어 인터럽트를 이용 응용프로그램에서 요청하는 작업을 커널이 처리

### ✓ 파일 입출력 형식

- 파일 입출력 함수로 H/W 제어하는 개념
- 디바이스 파일에 응용 프로그램의 입출력을 시도하면, 커널 내부의 디바이스 파일에 연결된 디바이스 드라이버 루틴들이 호출되어 디바이스에 대한 작업 처리, 처리 후 제어권은 응용프로그램에게로

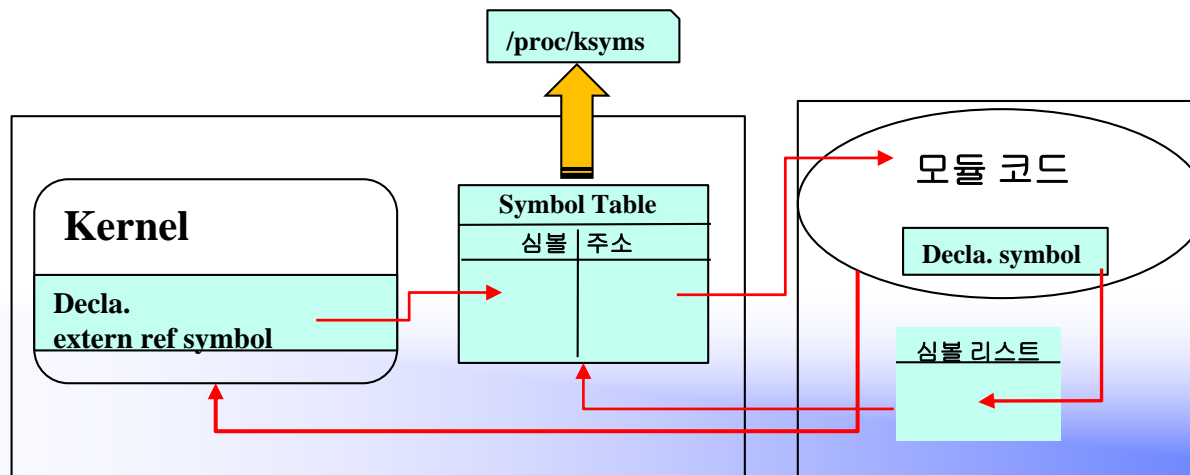
## □ 커널과 모듈

- ✓ 커널이 동작 중에도 디바이스 드라이버를 동적으로 추가/삭제
- ✓ MMU가 있는 CPU에서만 지원
- ✓ 커널 메모리의 효율적 사용
- ✓ 드라이버 개발 시 개발 시간 단축

# 커널과 모듈

## ❑ 모듈의 구현 원리

- ✓ 로딩: 논리 주소에 의해 각 함수나 변수의 주소가 주어짐
- ✓ 동적 라이브러리의 동작과 유사
  - 심볼릭 테이블 참조와 독립적 동적 라이브러리 저장 세그먼트
- ✓ 커널에 동적으로 링크
- ✓ 커널 심볼 테이블
  - /proc/ksyms에서 제공
  - 커널 내부의 함수나 변수 중 외부에서 참조할 수 있는 함수의 심볼과 주소를 담은 테이블
- ✓ 객체 형태의 커널 모듈 루틴이 참조할 커널 내부의 함수나 변수에 연결



# 커널과 모듈

## ❑ 모듈 유틸리티

- ✓ insmod: 모듈을 커널에 적재
- ✓ rmmod: 커널에서 모듈 제거
- ✓ lsmod: 커널에 적재된 모듈 목록 나열
- ✓ depmod: 모듈간 의존성 정보 생성
- ✓ modprobe: 모듈 및 연관 모듈을 커널에 적재/제거

insmod는 '/lib/modules/커널버전'의 디렉토리를 검색, 해당 모듈이 있으면 적재.  
modprobe는 depmod에 의해 생성된 modules.dep에서 찾아 모듈을 적재. 그리고 해당 모듈에 의존성이 있거나 해당모듈보다 선행되어야 하는 모듈이 있으면 또한 적재

## ❑ printk ()

- ✓ 디바이스 드라이버 작성 시 변수 값을 출력 가능.
- ✓ 이 경우, 반드시 “\n” 포함 - 출력 문자열에 개행문자!

## ❑ sudo less /proc/iomem

- ✓ 사용 중인 메모리 공간 출력



# 디바이스 파일

## □ 장치 파일 (device file)

- ✓ 디바이스 드라이버를 접근하는 통로
- ✓ Character device, block device
- ✓ 장치 파일의 inode 구성 요소
  - 장치 유형 (type) – character or block device 구별
  - 주 번호 (major number),
  - 부 번호 (minor number)

## □ 장치 파일의 예 (include/linux/major.h)

- ✓ /dev/ 에 있는 파일들

```
...  
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  
...
```

- ❑ 응용프로그램에서 디바이스 드라이버 구성 함수 호출 불가
  - ✓ 멀티 프로세싱에서 다른 프로세스 간 점유 경쟁
  - ✓ 응용프로그램이 커널에게 디바이스 제어 요청
  - ✓ 커널이 디바이스 드라이버 함수 호출
  
- ❑ 디바이스 파일
  - ✓ 디바이스 드라이버 타입 정보 (문자형/블록형)
  - ✓ 주번호(major number)
  - ✓ 부번호(minor number)

## □ 디바이스 주 번호

- ✓ 각종 장치 구동 함수들을 `file_operations` 구조체의 변수로 저장
- ✓ 문자형과 블록형에 따라 다른 배열 변수로 커널에 등록
- ✓ 커널에 등록할 때 배열 변수의 순번(index, 0에서 시작)
- ✓ 제어하려는 장치를 구분하기 위한 장치(device)의 ID

## □ 디바이스 부 번호

- ✓ 디바이스 드라이버가 다루는 실제 디바이스의 구분
- ✓ misc 계열 장치(입력 장치, RealTimeClock, Watch dog)의 용도에 따른 구분
- ✓ 블록 디바이스의 파티션 구분

# 디바이스 드라이버

## 장치 파일의 생성 : mknod

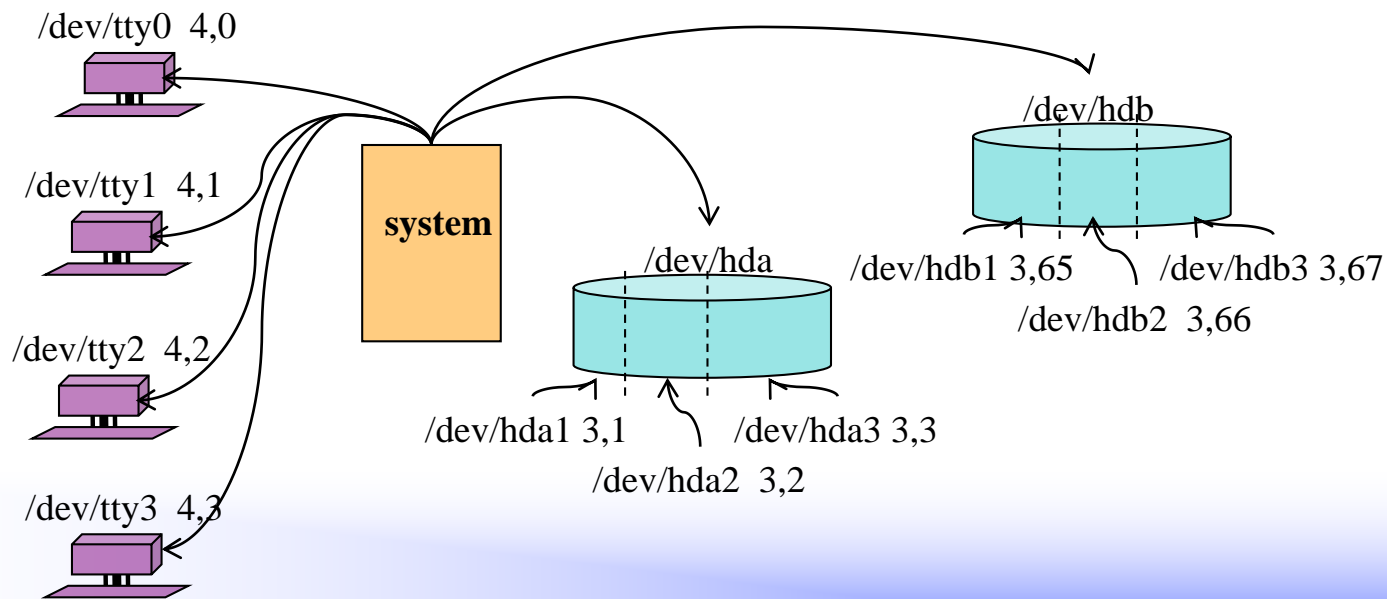
✓ `mknod /dev/file_name [b|c] major_number minor_number`

**convention**

**block or character device**

**device type**

**device unit**



# 디바이스 드라이버

---

## ❑ 장치(device)

- ✓ 장치는 일반적으로 제어기(controller)와 장치 자체(device itself)로 구성됨
- ✓ 제어기
  - 제어 레지스터 (Control Register), 상태 레지스터 (Status Register) 등의 레지스터와 internal buffer로 구성
  - 드라이버는 장치에 명령을 내리기 위해 제어 레지스터의 특정 비트를 설정하고, 명령 처리 결과와 에러 발생 여부를 확인하기 위해 상태 레지스터를 읽는다.
  - CPU/DMA driven
  - Memory mapped I/O, Special in/out instruction
  - 폴링(polling), 인터럽트(interrupt)
- ✓ 장치 자체(device itself)

# 디바이스 드라이버

## □ 디바이스 드라이버는 다음의 3 부분으로 구성

### ✓ 초기화 인터페이스

#### ➤ init() 함수 (또는 init\_module() 함수)

- register\_chrdev()
- register\_blkdev()
- request\_irq()
- ...

### ✓ 시스템 인터페이스

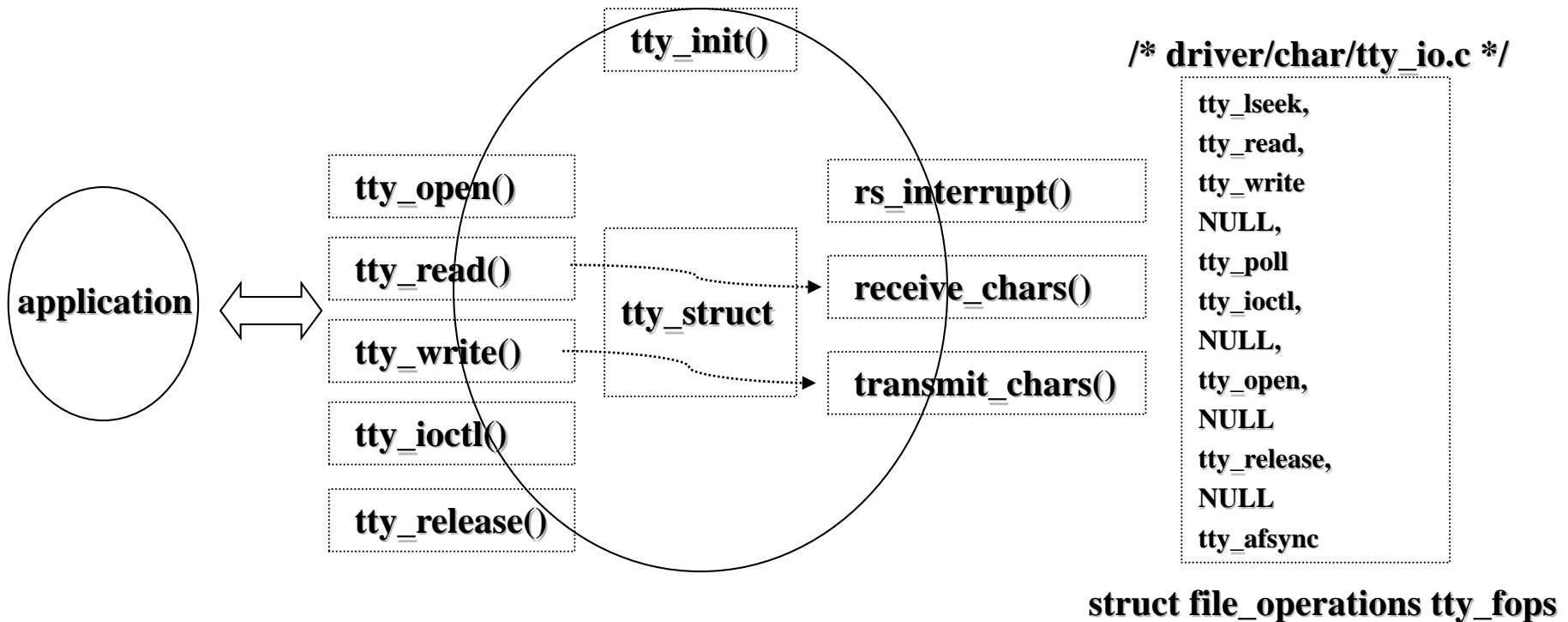
- 잘 정의된 인터페이스: 문자와 블록의 경우 block\_device\_operations 구조체 이용
- 문자 드라이버: open, release, read, write, ioctl
- 블록 드라이버: open, release, request, ioctl
- 네트워크 드라이버: open, close, transmit, receive, ioctl

### ✓ 하드웨어 인터페이스

- in(), out()

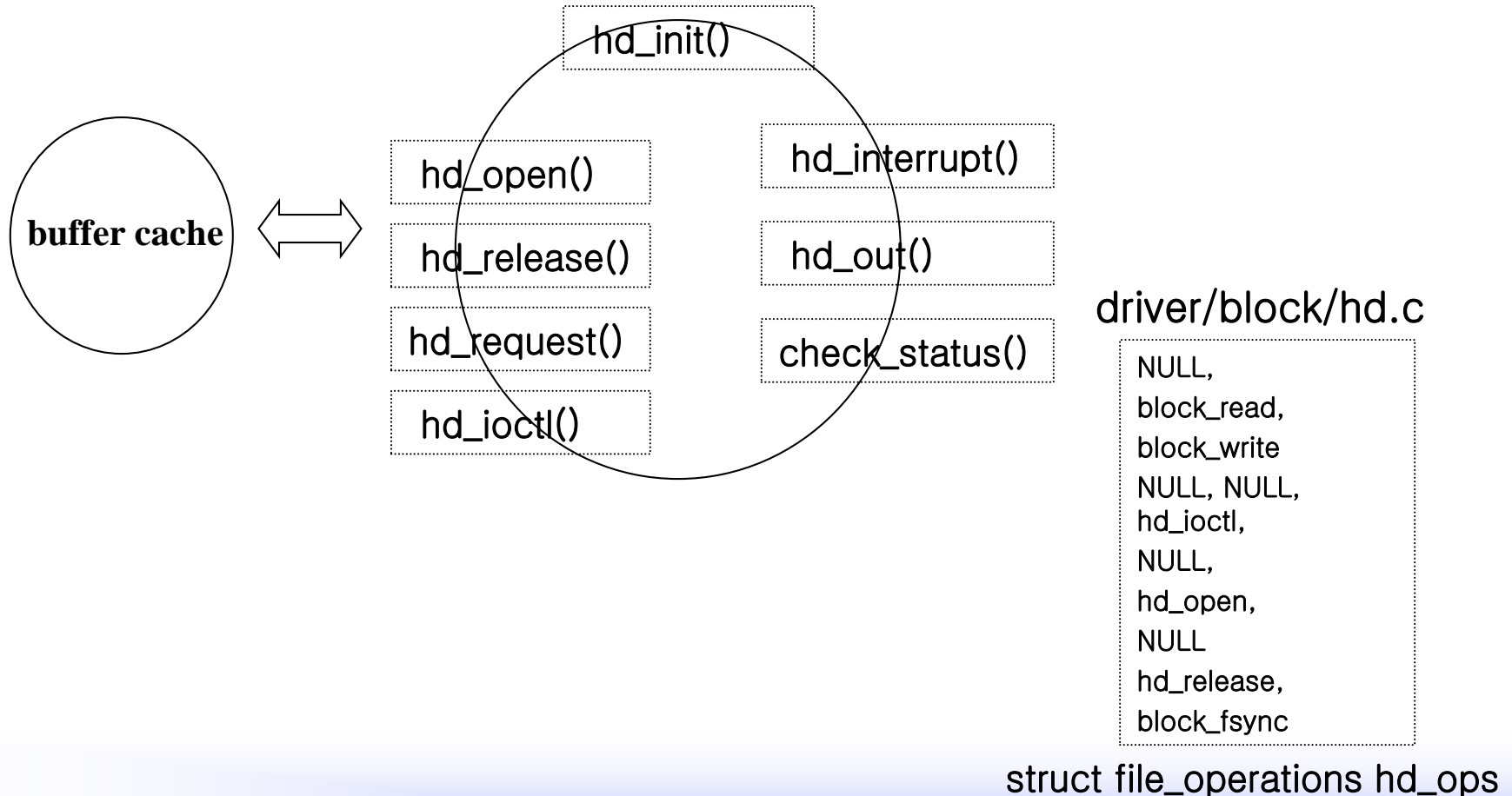
# 디바이스 드라이버

## 문자 디바이스 드라이버의 예 : 터미널 드라이버



# 디바이스 드라이버

□ 블록 디바이스 드라이버의 예 : IDE 디스크 드라이버

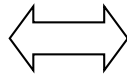
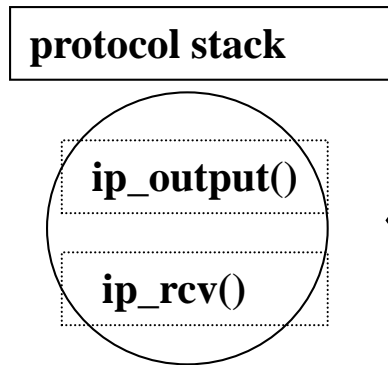




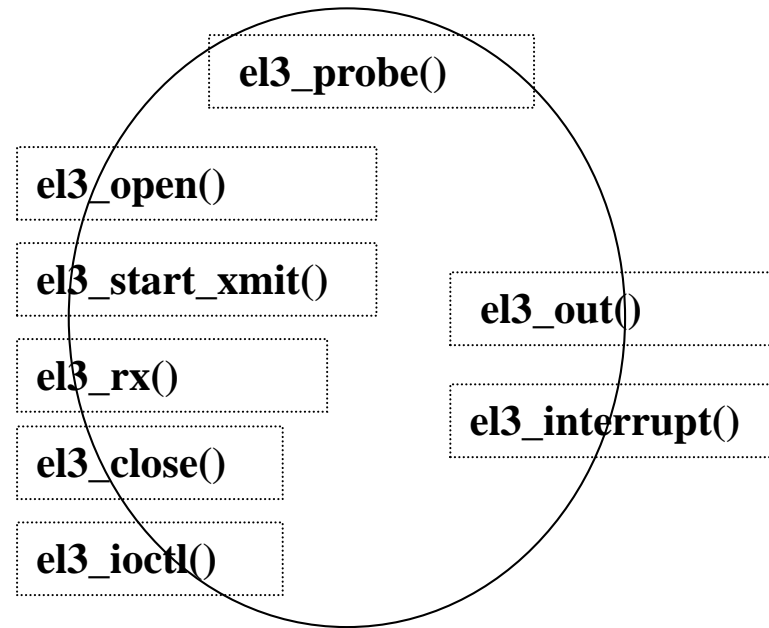
# 디바이스 드라이버

- 네트워크 디바이스 드라이버의 예: 3C509
  - ✓ 파일 시스템이 아닌 프로토콜 스택과 인터페이스

`/* net/ipv4/ip_input.c */`



`/* driver/net/3c509.c */`

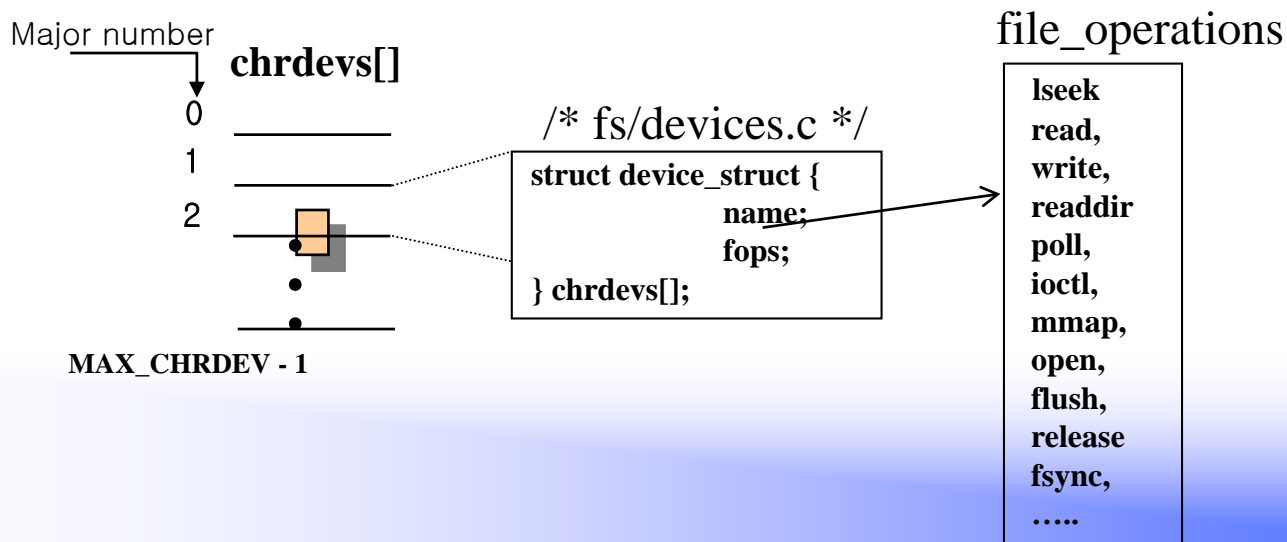


**struct device (no file operations)**

# 디바이스 드라이버

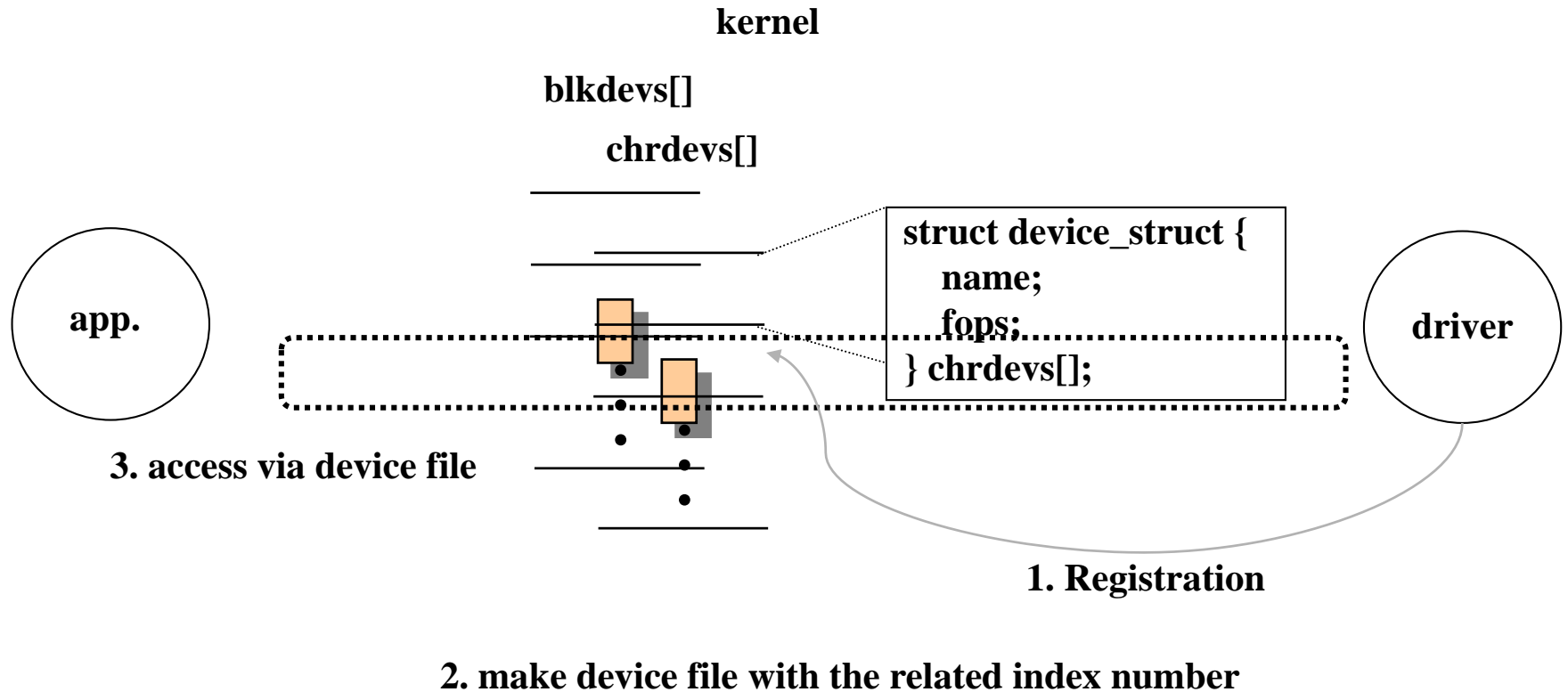
## ❑ 디바이스 드라이버와 커널간에 통신

- ✓ 문자 드라이버: chrdevs[] 테이블 이용
  - 버퍼없이 임의의 길이를 갖는 문자열 처리
  - open() read() write() close()
- ✓ 블록 드라이버: blkdevs[], blk\_dev[] 테이블 이용
  - 일정 크기의 내부적 버퍼를 통해 블록 데이터나 스트림 처리
  - open() read() write() close()
- ✓ 네트워크 드라이버: device 구조 이용
  - 네트워크 계층과 연결



# 디바이스 드라이버

## ❑ 드라이버와 커널 인터페이스



# 통합형 디바이스 드라이버

## □ 리눅스 커널 2.6 이후 규정

### ✓ device:

- 실존하는 하드웨어
- struct device에 정의하여 등록

### ✓ driver:

- 디바이스를 제어하는 소프트웨어
- struct device\_driver에 정의하고 등록

### ✓ bus:

- 하드웨어가 연결된 버스
- struct bus\_type에 정의하고 등록

### ✓ class:

- 하드웨어의 종류
- struct device\_class에 정의하고 등록

### ✓ interface:

- 입출력 처리의 논리적인 연결 방법
- struct device\_interface에 정의하고 등록

# 디바이스 드라이버 계층

---

- ❑ 인터페이스 구현부
  - ✓ 시스템과 연결되는 외부 시스템에 관련된 인터페이스(버스) 구현
- ❑ 인터페이스 접근 알고리즘 구현부
  - ✓ 외부 디바이스와 커널의 상호 통신을 구현
- ❑ 인터페이스에 연결되어 동작하는 디바이스 구현부
  - ✓ 인터페이스의 클라이언트 대응을 구현
  - ✓ 예, IDE에서 HDD
- ❑ 하드웨어 검출 구현부
  - ✓ 인터페이스 컨트롤러의 검출과 여기에 연결된 디바이스의 검출 구현
- ❑ proc 파일 시스템 구현부
  - ✓ /proc 파일 시스템 지원을 구현
- ❑ devfs 지원 구현부
  - ✓ 실존 디바이스의 디바이스파일을 드라이버에서 지원토록 구현
- ❑ 핵심 구현부
  - ✓ 각 디바이스에 맞는 디바이스 드라이버 등록

# 디바이스 드라이버

## □ 새로운 디바이스 드라이버 작성 단계

1. 디바이스 드라이버 커널 인터페이스 구현
  - entry point functions, file\_operations
2. 디바이스 드라이버 초기화 인터페이스 구현
  - 새로운 디바이스 드라이버를 위한 주 번호 할당 및 디바이스 드라이버 루틴 등록
3. 디바이스 드라이버 하드웨어 인터페이스 구현
  - 레지스터 접근
4. 장치 파일 생성
  - `# mknod /dev/mydrv [b|c] major_number minor_number`
5. 응용 프로그램 작성
6. 커널 컴파일 및 리부팅

☞ 리눅스의 모듈 인터페이스를 사용하면 단계 6에서 **insmod** 사용

# 모듈 프로그래밍

## □ 변수 선택

- ✓ 지역변수 사용 권장
- ✓ 전역변수: 커널 적재 이후 해제 시까지 유지해야 하는 정보 지정

## □ 중복 식별자 방지

- ✓ 파일 내로 접근 한정되는 함수명과 전역 변수명 (static variable과 다름) 사용

- ✓ 예,

```
static int checkout = 0; // 파일에서만 사용되는 변수 선언
static int dev_app_check (void)
{
    ...
}
```

## □ 변수의 데이터형

- ✓ 이종 플랫폼간 이식성을 고려
- ✓ `#include <asm/types.h>`

부호 있는 정수		부호 없는 정수	
__s8, s8	8비트	__u8, u8	8비트
__s16, s16	16비트	__u16, u16	16비트
__s32, s32	32비트	__u32, u32	32비트
__s64, s64	64비트	__u64, u64	64비트

## □ 구조체

- ✓ 보편적으로 int형 크기의 배수로 선언
- ✓ 문제 시, “packed” 키워드 사용 실제 크기로 선언 가능

```
typedef struct attr {  
    u16 index;  
    u16 data;  
    u8  msg;  
} __attribute__((packed)) testctl_t;
```

## □ I/O 메모리 접근 변수

- ✓ `u32 *ptr = (u32) 0xe000300; //`
- ✓ `volatile u32 *ptr = (volatile u32 *) 0xe000300; /* 컴파일러 최적화 옵션에도 정상 보장, 레지스터에 저장 안 됨 */`



## ❑ 동적 메모리 할당

- ✓ 사용자 영역이 아닌 커널 영역 (kernel space v.s. user space)
- ✓ `kmalloc ()`, `kfree ()`
  - `32*PAGE_SIZE` 크기로 할당 가능
  - 처리방식
- ✓ `__get_free_pages ()`, `free_pages ()`
  - 페이지 단위 할당 함수
- ✓ `vmalloc ()`, `vfree ()`
  - 가상 공간이 허용하는 범위 내에서 할당 가능
  - 할당 속도가 `kmalloc ()`보다 느림
  - 인터럽트 처리(Interrupt handling) 함수 안에서 사용 불가

## ❑ 메모리 풀(memory pool) – linux kernel v2.6 이상

- ✓ 대용량 데이터 처리에 적합
- ✓ `mempool_create ()`, `mempool_destroy ()`
- ✓ `mempool_alloc ()`, `mempool_free ()`

## ❑ struct file\_operations

- ✓ 문자 디바이스 드라이버와 응용 프로그램 연결
- ✓ 저수준 파일 입출력 함수를 사용하여 디바이스 파일에 접근
  - 커널이 등록된 구조체 정보를 참고하여 대응함수를 호출
- ✓ 문자 디바이스 드라이버 등록은 이 구조체 멤버 변수를 등록하는 의미
- ✓ 예,  
    open () : xxx\_open ()  
    read (): xxx\_read ()

## ❑ 문자 디바이스 드라이버 등록/해제

- ✓ insmod 명령으로 커널에 삽입
- ✓ rmmod 명령

## ❑ 응용프로그램의 디바이스 파일 개/폐

- ✓ open (): file\_operations.open
- ✓ close (): file\_operations.release

# I/O Accesses

## □ I/O 주소 지정 방식에 따라

### ✓ I/O mapped I/O – Intel 계열

- I/O 주소와 메모리 주소가 다르게 지정
- 입출력 회로가 메모리 보다 속도가 느리고 공간이 작다는 가정에 따라서 이 두 공간을 분리하여 속도와 공간의 효율을 높임
- 주소 공간을 분리하려면 기계어 명령어를 분리해야 함
- `inb()`, `inw()`, `inl()`, `insb()`, `insw()`, `inw()`
- `outb()`, `outw()`, `outl()`, `outsb()`, `outsw()`, `outl()`

### ✓ Memory mapped I/O – ARM 계열

- I/O 주소와 메모리 주소의 구분 없이 하나의 메모리 공간에 입출력 장치 지정
- `readb()`, `readw()`, `readl()`, `memcpy_fromio()`
- `writeb()`, `writew()`, `writel()`, `memcpy_toio()`

# 디바이스 제어

---

## ❑ 디바이스 제어(Device Control)

- ✓ ioctl ()
- ✓ ioperm (): 장치 접근 여부 제어
- ✓ fcntl (): 장치 상호배제
- ✓ fsync (): 파일에 쓴 데이터와 실제 H/W의 동기화

## ❑ 저수준 입출력 함수 ioctl()를 디바이스에 적용시키면 디바이스 파일에 연결된 디바이스 드라이버의 file\_operations 구조체에서 멤버 변수(필드), ioctl에 선언된 함수가 호출

## ❑ 디바이스 파일 전용 함수

## ❑ 특징:

- ✓ read(), write()와 같이 읽기 쓰기 처리 가능
- ✓ H/W 제어나 상태 정보 얻기 가능

# 입출력 장치 제어 - ioctl()

- ❑ 입출력(Input/Oupt)장치의 제어(Control)
- ❑ 장치 정보를 얻거나 장치의 값을 변경
  - ✓ 터미널, 소켓, cdrom, floppy, 프린터, 사운드카드.. 등 모든 입출력과 관련된 장치를 제어
  - ✓ 예, 오디오 CD를 재생하는 애플리케이션을 제작하려면 ioctl()을 이용해서 장치를 제어하고 CD에 있는 데이터를 읽어와야 함

```
#include <sys/ioctl.h>
```

```
int ioctl(int fd, int request, ...);
```

fd: open 함수 실행 결과로 반환된 입출력 지정자 (일반 파일, 소켓, 장치)

request: 디바이스파일에 연동된 디바이스 드라이버에 취해야 할 명령을 정의

... : 세번째 인자 부터는 request에 해당하는 명령에 대한 보조적인 정보값

- ✓ request를 통해서 얻은 정보를 저장하기 위한 버퍼

```

#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <linux/lp.h> // <----- 프린트 포트 관련 명령이 담겨있는 헤더파일

int main( int argc, char ** argv)
{
    int fd, prnstate, lp;
    unsigned char buff[128];
    fd = open ("/dev/lp0", O_RDWR | O_NDELAY); // <----- 프린트 포트 디바이스 열기
    if ( fd< 0) {
        perror ("open error"); // <----- 열기 실패시 에러 메세지 출력
        exit (1);
    };
    while (1) {
        ioctl (fd, LPGETSTATUS ,&prnstate); // <-- 프린트 포트의 상태를 요청. (프린트 포트에 대한 명령은
                                           //          LPGETSTATUS, 프린트 포트의 상태가 prnstate변수에 저장)
        if (prnstate & LP_PSELECD) printf ("ON\n");
        else printf ("OFF\n");
        usleep (50000);
    }
    close (fd);
    return 0;
}

```

# 인터럽트

---

- ❑ `request_irq()`: 디바이스 드라이버가 IRQ #와 핸들러 등록
- ❑ `do_IRQ()`: 리눅스 커널에서 IRQ 처리, 핸들러 호출
- ❑ `free_irq()`: 인터럽트 핸들러 함수 제거

# 메모리 매핑 – 디바이스 데이터 전송

- ❑ 저수준 메모리 접근 – `inb()`, `inw()`, `outb()`, `outw()`
- ❑ Memory mapping
  - ✓ 메모리 접근 명령으로 장치에 데이터 전달
  - ✓ **Page-based data access**
  - ✓ 물리주소와 커널 가상 주소 변환
    - `ioremap()`, `ioremap_nocache()`
    - `iounmap()`
  - ✓ 응용프로그램에서
    - `#include <sys/mman.h>`
    - `mmap ()`
    - `munmap ()`
  - ✓ 디바이스 드라이버의 `mmap` 구현
    - `file_operations` 구조체 변수에 등록된 `mmap()`가 호출 됨
  - ✓ Frame buffer 등 대량의 데이터 전송에 이용



# DMA(Direct Memory Access)

---

## ❑ Programmed I/O

- ✓ kernel memory space
- ✓ user memory space

## ❑ DMA

### ✓ 알고리즘

1. 프로세서가 메모리의 시작주소와 디바이스 시작주소, 그리고 데이터 크기를 DMAC에게 전달
2. DMAC가 버스 제어권 요구
3. 버스 제어권을 프로세서에서 DMAC가 넘겨 받음
4. 데이터 전송
5. 프로세서에게 버스 제어권 반납
6. DMAC가 프로세서에게 전송 종료를 인터럽트로 전달

### ✓ request\_dma(), free\_dma()

## ❑ 발전 방향

- ✓ Third-party DMA: ISA 버스 지원, 리눅스 지원
- ✓ Bus mastering: PCI 버스 지원
- ✓ Ultra DMA

# 디바이스 드라이버 예제

## ■ 모듈 프로그램을 이용한 문자 디바이스 드라이버 구현

```
#include <linux/module.h> /* chr_drv.c 모듈 프로그램 */
#include <linux/kernel.h>
#include <linux/slab.h>
#include <asm/uaccess.h>

int init_module (void);
void cleanup_module (void);
int mydrv_init (void);
static int mydrv_open (struct inode *, struct file *);
static int mydrv_release (struct inode *, struct file *);
static int mydrv_ioctl (struct inode *, struct file *, unsigned int, unsigned long);
static ssize_t mydrv_read (struct file *, char *, size_t, loff_t *);
static ssize_t mydrv_write (struct file *, const char *, size_t , loff_t *);

#define DEVICE_NAME "mydrv"          /* /proc/devices 에 등록될 이름 */
#define MYDRV_MAJOR    251
#define MYDRV_MAX_LENGTH  4096
#define MIN(a, b) (((a) < (b)) ? (a) : (b))
static char *mydrv_data;
static int mydrv_length1, mydrv_length2;
```

# 디바이스 드라이버 예제

## ■ 초기화

```
struct file_operations mydrv_fops = { NULL, NULL, mydrv_read, mydrv_write, NULL, NULL, mydrv_ioctl,  
    NULL, mydrv_open, NULL, mydrv_release, NULL, NULL, NULL, NULL };  
  
int mydrv_init (void)  
{  
    int result, i;  
  
    result = register_chrdev(MYDRV_MAJOR, "mydrv", &mydrv_fops);  
  
    if (result < 0) {  
        printk(KERN_WARNING, "mydrv: %d character device driver can't be registered\n", MYDRV_MAJOR);  
        return result;  
    }  
  
    mydrv_data = (char *) kmalloc(MYDRV_MAX_LENGTH * sizeof(char), GFP_KERNEL);  
    if (mydrv_data == NULL) {  
        unregister_chrdev(MYDRV_MAJOR, "mydrv");  
        return -ENOMEM;  
    }  
  
    mydrv_length1 = mydrv_length2 = 0;  
    return 0;  
}
```

# 디바이스 드라이버 예제

## ■ 모듈 등록/해제

```
int init_module(void) {
    return (mydrv_init());
}

void cleanup_module(void) {
    kfree(mydrv_data);
    unregister_chrdev(MYDRV_MAJOR, "mydrv");
}

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

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

# 디바이스 드라이버 예제

## ■ file operation 함수

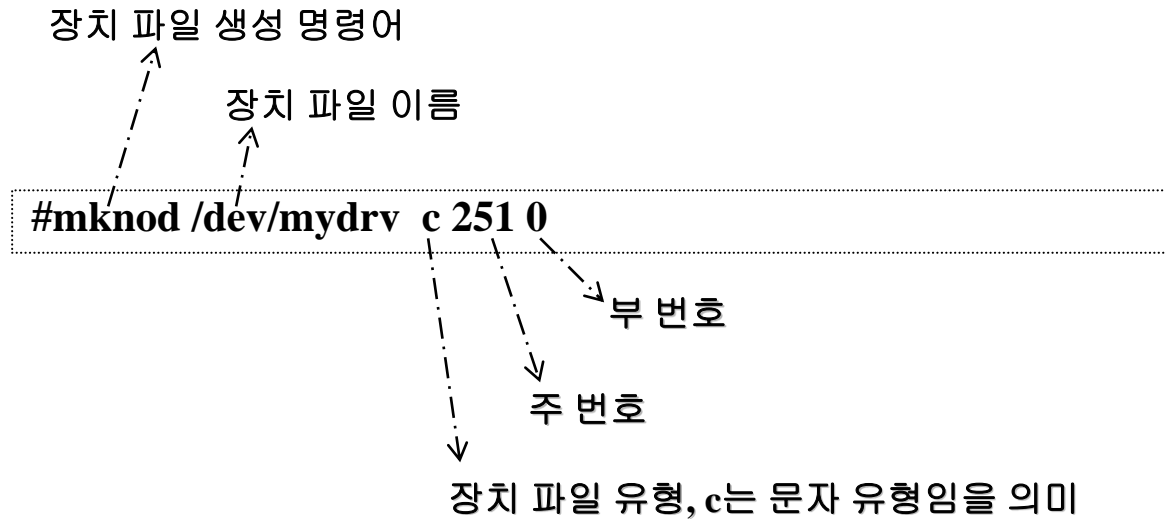
```
static int mydrv_ioctl (struct inode * inode, struct file * file, unsigned int cmd, unsigned long arg) {
    printk("mydrv_ioctl\n");
    return 0;
}

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_length2 - mydrv_length1) <= 0)
        return 0;
    count = MIN((mydrv_length2 - mydrv_length1), count);
    if (copy_to_user(buf, mydrv_data + mydrv_length1, count))
        return -EFAULT;
    mydrv_length1 += 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_length2 > MYDRV_MAX_LENGTH)
        count = MYDRV_MAX_LENGTH - mydrv_length2;
    if (copy_from_user(mydrv_data + mydrv_length2, buf, count))
        return -EFAULT;
    mydrv_length2 += count;
    return count;
}
```

# 디바이스 드라이버 예제

## ■ mydrv를 위한 장치 파일



# 디바이스 드라이버 예제

## ■ 드라이버를 사용하는 응용 예

```
#include <unistd.h> /* user_app.c, 응용 프로그램 */
#include <fcntl.h>
#include <stdio.h>

#define MAX_BUFFER 26

// 응용 수행 전에 장치 파일 생성 필요

int main ()
{
    int fd, i, c = 65;
    char buf_in[MAX_BUFFER], buf_out[MAX_BUFFER];

    fd = open ("/dev/mydrv", O_RDWR);
    if (fd < 0) {
        printf ("Can't open /dev/mydrv file\n");
        exit(0);
    }

    for (i=0; i<MAX_BUFFER; i++) buf_in[i] = c++;
    write (fd, buf_in, MAX_BUFFER);
    close (fd);
}
```

# 디바이스 드라이버 예제

## ■ 드라이버를 사용하는 응용 예

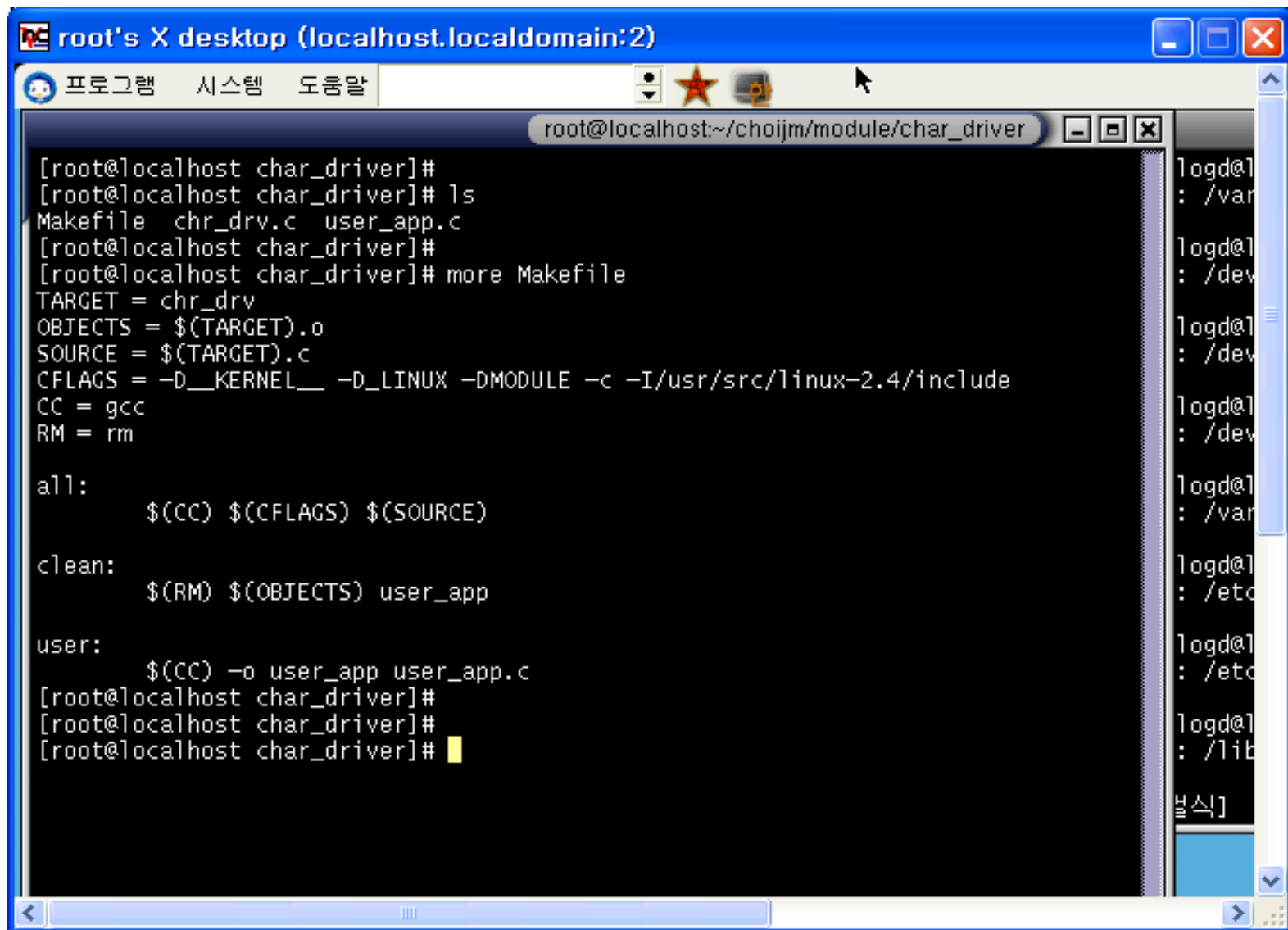
```
/* user_app.c 응용 프로그램 */  
  
for (i=0; i<MAX_BUFFER; i++) buf_out[i] = 0;  
  
fd = open ("/dev/mydrv", O_RDWR);  
read (fd, buf_out, MAX_BUFFER);  
  
for (i=0; i<MAX_BUFFER; i++) fprintf (stderr, "%c", buf_out[i]);  
fprintf (stderr, "\n");  
  
close (fd);  
return 0;  
}
```

☞ 사용자가 **read()**를 요청했을 때 어떻게 **mydrv\_read()**가 호출될 수 있을까?



# 디바이스 드라이버 예제

## 수행 결과: Makefile



The screenshot shows a terminal window titled "root's X desktop (localhost.localdomain:2)". The terminal is running a shell session in the directory `~/choijm/module/char_driver`. The user has executed `ls` and `more Makefile` to view the contents of the `Makefile` file. The Makefile defines the target `chr_drv`, the objects `chr_drv.o`, the source file `chr_drv.c`, and the compilation flags `-D__KERNEL__ -D_LINUX -DMODULE -c -I/usr/src/linux-2.4/include`. The compiler is `gcc` and the linker is `rm`. The `all` target compiles the source file into the object file. The `clean` target removes the object file and the `user_app` file. The `user` target compiles the `user_app.c` file into the `user_app` executable.

```
[root@localhost char_driver]#  
[root@localhost char_driver]# ls  
Makefile chr_drv.c user_app.c  
[root@localhost char_driver]#  
[root@localhost char_driver]# more Makefile  
TARGET = chr_drv  
OBJECTS = $(TARGET).o  
SOURCE = $(TARGET).c  
CFLAGS = -D__KERNEL__ -D_LINUX -DMODULE -c -I/usr/src/linux-2.4/include  
CC = gcc  
RM = rm  
  
all:  
    $(CC) $(CFLAGS) $(SOURCE)  
  
clean:  
    $(RM) $(OBJECTS) user_app  
  
user:  
    $(CC) -o user_app user_app.c  
[root@localhost char_driver]#  
[root@localhost char_driver]#  
[root@localhost char_driver]#
```

# 디바이스 드라이버 예제

## 수행 결과

```
choijm@embedded:/home/user/choijm/syspro/exam_module/4_chardriver
TARGET = chr_drv
OBJECTS = $(TARGET).o
SOURCE = $(TARGET).c
CFLAGS = -DMODULE -D__KERNEL__ -DLINUX -I/usr/src/linux-2.4/include -c
CC = gcc
RM = rm

# rules
all :
    $(CC) $(CFLAGS) $(SOURCE)

clean:
    $(RM) $(OBJECTS) user_app

user :
    $(CC) -o user_app user_app.c

~
~
~
"Makefile" 18L, 266C 저장 했습니다
[영어] [완성] [두벌식]
```

```
choijm@embedded:/home/user/choijm/syspro/exam_module/4_chardriver
[root@embedded 4_chardriver]# ls
Makefile chr_drv.c user_app.c
[root@embedded 4_chardriver]# ls /dev/*drv*
ls: /dev/*drv*: 그런 파일이나 디렉토리가 없음
[root@embedded 4_chardriver]# mknod /dev/mydrv c 251 0
[root@embedded 4_chardriver]# ls -l /dev/*drv*
crw-r--r-- 1 root root 251, 0 11월 4 22:10 /dev/mydrv
[root@embedded 4_chardriver]# make
gcc -DMODULE -D__KERNEL__ -DLINUX -I/usr/src/linux-2.4/include -c chr_drv.c
[root@embedded 4_chardriver]# make user
gcc -o user_app user_app.c
[root@embedded 4_chardriver]# ls
Makefile chr_drv.c chr_drv.o user_app user_app.c
[root@embedded 4_chardriver]# insmod chr_drv.o
Module chr_drv loaded, with warnings
[root@embedded 4_chardriver]# ./user_app
ABCDEFGHIJKLMNOPQRSTUVWXYZ
[root@embedded 4_chardriver]# ./user_app
ABCDEFGHIJKLMNOPQRSTUVWXYZ
[root@embedded 4_chardriver]# rmmod user_app
rmmod: module user_app is not loaded
[root@embedded 4_chardriver]# rmmod chr_drv
[root@embedded 4_chardriver]#
[영어] [완성] [두벌식]
```