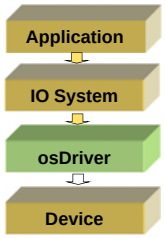
디바이스 드라이버 프로그래밍

Device Driver 개요

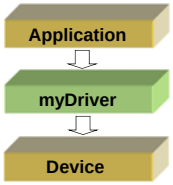
- Device
  - 네트워크 어댑터, LCD 디스플레이, PCMCIA, Audio, 터미널, 키보드, 하드디스크, 플로피디스크, 프린터 등과 같은 주변 장치들을 말함
  - 디바이스의 구동에 필요한 프로그램, 즉 디바이스 드라이버가 필수적으로 요구됨
- Device Driver
  - 실제 장치 부분을 추상화시켜 사용자 프로그램이 정형화된 인터페이스를 통해 디바이스를 접근할 수 있도록 해주는 프로그램
  - 디바이스 관리에 필요한 정형화된 인터페이스 구현에 요구되는 **함수**와 **자료구조**의 집합체
  - 표준적으로 동일 서비스 제공을 목적으로 커널의 일부분으로 내장
  - 응용프로그램이 H/W를 제어할 수 있도록 인터페이스 제공
  - 하드웨어 독립적인 프로그램을 작성을 가능하게 함

디바이스 드라이버 형태

- Device Driver Interface
  - Standard Device Driver Interface
    - UNIX compatible I/O system interface
      : open(), close(), read(), write(), ioctl()

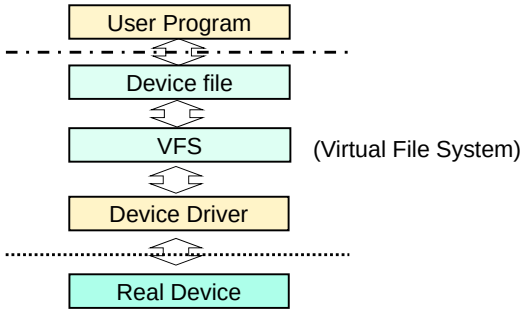


- Non-standard Device Driver Interface
  - Completely user-defined
  - Custom interface
  - May be more appropriate for some hardware



리눅스 디바이스 드라이버

- 사용자 관점에서의 디바이스 드라이버
  - 사용자는 디바이스 자체에 대한 자세한 정보를 알 필요 없음
  - device는 하나의 파일로 인식됨
  - file에 대한 접근을 통하여 real device에 접근 가능



리눅스 디바이스 드라이버(2)

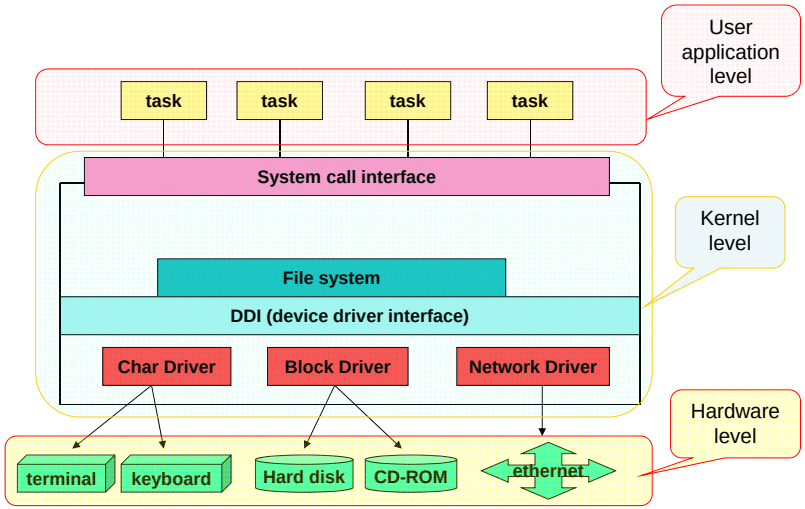
- 리눅스에서의 디바이스
  - Linux에서 device는 특별한 파일로 취급되고, access가 가능함.
    - 사용자(응용프로그램)는 file operation을 적용할 수 있음
  - 각 디바이스는 Major number와 Minor number를 가짐
    - Major number: 디바이스 장치를 구분
    - Minor number: 같은 종류의 디바이스들을 구분

- Device Driver의 종류
  - 문자 디바이스 드라이버
  - 블록 디바이스 드라이버
  - 네트워크 디바이스 드라이버

디바이스 드라이버 종류

Deriver 종류	설 명	등록함수명
문자 드라이버 (char)	device를 file처럼 접근하여 직접 read/write를 수행, data 형태는 stream 방식으로 전송 (ex) console, keyboard, serial port driver등	register_ chrdev()
블록 드라이버 (block)	disk와 같은 file system을 기반으로 block 단위로 data의 read/write 수행 (ex) hard disk, CD-ROM driver, floppy disk	register_ blkdev()
네트워크 드라이버 (network)	network의 물리계층과 frame 단위의 data 송수신 (ex) Ethernet device driver	register_ netdev()

디바이스 드라이버 종류(2)



Char Device (문자 디바이스)

- 문자 디바이스의 특징
  - 자료의 순차성을 지닌 장치
  - 버퍼 캐쉬를 사용하지 않음
  - 장치의 raw data를 사용자에게 제공
  - Terminal, Serial/Parallel, Keyboard, Sound Card, Scanner, Printer

- 리눅스에서의 문자 디바이스

null : black hole
tty* : virtual console
pt* : pseudo-terminal

```
crw--w--w-  0 root   root      5,   1  Oct  1  1998 console
crw-rw-rw-  1 root   root      1,   3  May  6  1998 null
crw-------  1 root   root      4,   0  May  6  1998 tty
crw-rw----  1 root   disk     96,   0  Dec 10  1998 pt0
crw-------  1 root   root      5,  64  May  6  1998 cua0
```

파일 관련 정보 중 첫 문자인 c는 char device를 의미

Block Device(블록 디바이스)

- Block device 특징
  - random access 가능
  - 블록 단위의 입출력이 가능한 장치
  - 버퍼캐쉬에 의한 내부 장치 표현
  - 파일 시스템에 의해 mount 되어 관리되는 장치
  - 디스크, RAM Disk, CD-ROM 등
- 리눅스에서의 Block device

fd* : Floppy disk
hd* : Hard disk
sda : SCSI disk

```
brw-------   1 root     floppy     2,   0 May  6  1998 fd0
brw-rw----   1 root     disk       3,   0 May  6  1998 hda
brw-rw----   1 root     disk       3,   1 May  6  1998 hda1
brw-rw----   1 root     disk       8,   0 May  6  1998 sda
brw-rw----   1 root     disk       8,   1 May  6  1998 sda1
```

파일 관련 정보 중 첫 문자인 b는 block device를 의미

Network Device(네트워크 디바이스)

- Network device 특징
  - 대응하는 장치파일이 없음
  - 네트워크 통신을 통해 패킷을 송수신할 수 있는 장치
  - 응용프로그램과의 통신은 표준 파일 시스템 관련 호출 대신에 socket() 이나 bind() 등의 시스템 호출 사용
  - Etherent, PPP, ATM, ISDN 등이 있음

Major & Minor Number

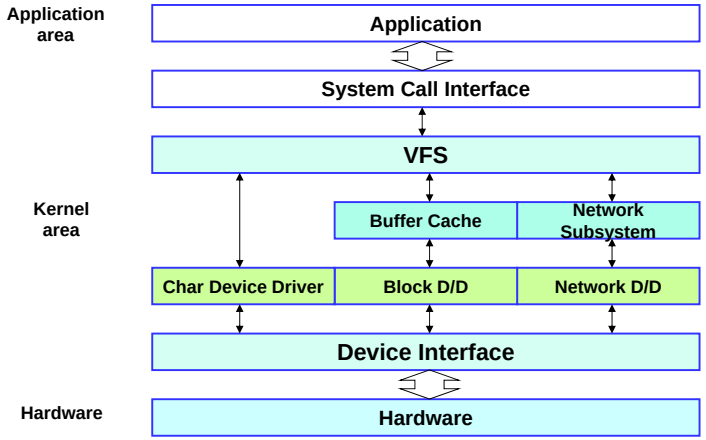
- Major number(주번호)
  - 커널에서 **디바이스 드라이버를 구분**/연결하는데 사용
  - 같은 Device의 종류를 지칭, 1Byte (0~255사이의 값)
- Minor number(부번호)
  - 디바이스 드라이버 내에서 **장치를 구분**하기 위해 사용
  - 각 Device의 부가적인 정보를 나타냄, 2Byte (부번호)
  - 하나의 디바이스 드라이버가 여러 개의 device 제어 가능
- $ ls -al /dev/hda*

```
brw-rw----   1 root     disk       3,   0 May  6  1998 hda
brw-rw----   1 root     disk       3,   1 May  6  1998 hda1
brw-rw----   1 root     disk       3,   2 May  6  1998 hda2
brw-rw----   1 root     disk       3,   3 May  6  1998 hda3
```

주번호 부번호

디바이스 드라이버 구조

- 리눅스 시스템 구조 상의 디바이스 드라이버



커널 모듈(kernel Module)

- 커널 모듈
  - 시스템 부팅 후에 동적으로 loading 할 수 있는 커널 구성요소
  - 커널을 다시 컴파일 하거나 시스템 재부팅할 필요 없이 커널의 일부분을 교체하는 것이 가능
  - 디바이스 드라이버, 파일 시스템, 네트워크 프로토콜 등이 모듈로 제공됨
- 일반 응용 프로그램과의 차이점
  - main() 함수가 없음
  - 커널에 로딩 및 제거 될 때 불러지는 함수가 존재
    - Loading 시 : module_init()로 지정된 함수 호출
    - Unloading 시 : module_exit()로 지정된 함수 호출

Linux Device Driver 특성

- 커널 코드
  - 디바이스 드라이버는 커널의 한 부분이므로, 커널의 다른 코드와 마찬가지로 잘못되면 시스템에 치명적인 피해를 줄 수 있다
- 커널 인터페이스
  - 디바이스 드라이버는 리눅스 커널이나 자신이 속한 서브시스템에 표준 인터페이스를 제공해야 한다.
- 커널 메커니즘과 서비스
  - 디바이스 드라이버는 메모리 할당, 인터럽트 전달, wait queue같은 표준 커널 서비스를 사용할 수 있다.
- Loadable
  - 대부분의 리눅스 디바이스 드라이버는 커널 모듈로서, 필요할 때 load 하고 더 이상 필요하지 않을 때 unload 할 수 있다.
- 설정가능(Configurable)
  - 리눅스 디바이스 드라이버를 커널에 포함하여 컴파일 할 수 있다. 어떤 장치를 넣을 것인지는 커널을 compile 할 때 설정할 수 있다

간단한 커널 모듈 작성 - Hello

- 예제 프로그램
  - 커널에 모듈이 로딩될 때 "Hello, linux kernel module"를 출력
  - 모듈이 제거될 때 "Good bye"를 출력
- Source file : hello.c
  - 필요한 include 파일 포함
  - module license 지정: MODULE_LICENSE(...);
    - license 종류: GPL, GPL v2, Dual BSD/GPL, Proprietary 등
    - 커널 2.6부터 반드시 지정해야 함
  - module_init 함수 작성 및 등록 : module_init(*init_func*);
  - module_exit 함수 작성 및 등록 : module_exit(*exit_func*);

hello.c

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

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Gildong Hong");
MODULE_DESCRIPTION("Hello");

static int __init module_begin(void)
{
        printk("Hello, linux kernel module.\n");
        return 0;
}

static void __exit module_end(void)
{
        printk("Good bye.\n");
}

module_init(module_begin);
module_exit(module_end);
```

커널 모듈 프로그램을 위한 Makefile

```
obj-m += hello.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
```

```
# make
make -C /lib/modules/3.13.0-00293-g1664d72/build M=/home/root/my/hello modules
make[1]: Entering directory `/usr/src/3.13.0-00293-g1664d72'
  CC [M]  /home/root/my/hello/hello.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/root/my/hello/hello.mod.o
  LD [M]  /home/root/my/hello/hello.ko
```

- 컴파일 과정
  - hello.o 파일을 먼저 생성
  - modpost를 C 소스파일에 적용해 .ko에서 요구되는 추가정보 부착하여 hello.mod.c를 생성한 후 컴파일 → hello.mod.o 생성
  - 두 개의 .o 파일을 링크하여 .ko (kernel object) 파일 생성

모듈 적재(loading), 제거(unloading)

- 생성된 모듈(hello.ko)을 로딩
  - insmod *hello.ko*
- 커널에 적재된 모듈 목록 보기
  - lsmod
- 모듈 제거
  - rmmod *hello* (.ko가 붙지 않음)
- hello 모듈 동작 확인
  - 모듈 적재와 제거 시에 메시지들이 출력되는 지를 확인
    # dmesg 또는
    # tail –f /var/log/kern.log (-f 옵션: 계속적인 변화 출력)

디바이스 드라이버의 작성 방법

- 디바이스 드라이버 함수 작성
  - struct file_operations 정의
  - open, release, read, write, ioctl 함수 구현
  - init, exit 함수 구현
- 커널에 디바이스 드라이버 등록 – init 함수에서 수행
  - register_chrdev(), register_blkdev(), register_netdev()
- 컴파일/로딩
  # make ... Makefile 작성 후 실행
  # insmod ... 생성된 .ko 파일 load
- 디바이스 파일 생성
  # mknod 디바이스파일이름 드라이버타입 주번호 부번호
  (예) mknod /dev/LED c 239 0
  - 필요하면 속성변경
    # chmod ug+w /dev/LED
- 디바이스 파일에 입출력하는 응용프로그램 작성 및 테스트

디바이스 드라이버 골격

#include <linux/kernel.h> #include <linux/module.h> #include <linux/fs.h> #include <linux/init.h>	Header Files
int device_open(…) { … } int device_release(…) { … } ssize_t device_write(…) { … } ssize_t device_read(…) { … }	Function Prototypes & Definitions

static struct file_operations device_fops = {
.open = device_read,
.release = device_release,
.write = device_write,
.read = device_read
.ioctl = … // deprecate
};
	File Operation

module_init(init_func); module_exit(exit_func);	**모듈 설치 시 초기화 수행** **모듈 제거 시 반환 작업수행**

Device Driver 관련 자료구조

```
struct  char_device_struct {     /* fs/char_dev.c */
    ....
    const char * name;                      // name field
    struct file_operations * fops;          // file_operations
} *chrdev[MAX_PROBE_HASH];

struct file_operations {            /* <linux/fs.h> */
            struct module *owner;
            loff_t (*llseek) (struct file *, loff_t, int);
            ssize_t (*read) (struct file *, char *, size_t, loff_t *);
            ssize_t (*write) (struct file *, const char *, size_t, loff_t *);
            …
            int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
            int (*mmap) (struct file *, struct vm_area_struct *);
            int (*open) (struct inode *, struct file *);
            int (*flush) (struct file *);
            int (*release) (struct inode *, struct file *);
            …
}
```


디바이스 드라이버 작성 – LED 디바이스 드라이버

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/init.h>
#include <linux/interrupt.h>
#include <asm/io.h>

#include <linux/fs.h>
#include <linux/uaccess.h>
#include <linux/types.h>
#include <linux/ioport.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("SangKyun Yun");
MODULE_DESCRIPTION("Seven Segment LEDs");

#define base_lwFPGA      0xFF200000
#define len_lwFPGA       0x200000

#define addr_LED         0

#define LED_DEVMAJOR     239
#define LED_DEVNAME      "led"

static void  *mem_base;
static void  *led_addr;
```

필요한 헤더 포함

LED 디바이스 드라이버(2)

- open, release 연산
  - 여기서는 특별한 동작을 하지 않음

```
static int led_open(struct inode *minode, struct file *mfile)
{
        return 0;
}

static int led_release(struct inode *minode, struct file *mfile)
{
        return 0 ;
}
```


LED 디바이스 드라이버(3)

- write 연산
  - 8개의 LED에 8-bit 값을 출력함
  - get_user(): user space 데이터를 kernel space로 복사
    - 첫번째 인수의 크기에 따라서 복사할 데이터 크기가 정해짐

```
static ssize_t led_write (struct file *file, const char __user *buf, size_t count,
    loff_t *f_pos){

        unsigned int led_data = 0;

        get_user(led_data, (unsigned char *)buf);
//      copy_from_user(led_data, (unsigned char*)buf, count);

        iowrite32(led_data, led_addr);
        return count;
}
```

LED 디바이스 드라이버(4)

- read 연산
  - led에 출력된 값을 읽음
  - put_user : kernel data를 user 공간에 복사

```
static ssize_t led_read (struct file *file, char __user *buf, size_t count,
    loff_t *f_pos){
        unsigned int led_data;

        led_data = ioread32(led_addr);
        put_user(led_data, buf);
        return 4;
}
```


LED 디바이스 드라이버(5)

- file operations에 대한 연산 초기화

```
static struct file_operations led_fops = {
        .read           =       led_read,
        .write          =       led_write,
        .open           =       led_open,
        .release        =       led_release,
//      .ioctl          =       led_ioctl,
};
```


LED 디바이스 드라이버(6)

- 드라이버 init/exit 함수

```
static int __init led_init(void)
{
        int res;

        res=register_chrdev(LED_DEVMAJOR, LED_DEVNAME, &led_fops);
        if(res < 0) {
                printk(KERN_ERR " leds : failed to register device.\n");
                return res;
        }

        mem_base = ioremap_nocache(base_lwFPGA, len_lwFPGA);
        if( !mem_base) {
                printk("Error mapping memory\n");
                release_mem_region(base_lwFPGA, len_lwFPGA);
                return -EBUSY;

        }
        led_addr = mem_base + addr_LED;

        printk("Device: %s MAJOR: %d\n", LED_DEVNAME, LED_DEVMAJOR);
        return 0;
}
```


LED 디바이스 드라이버(7)

```
static void __exit led_exit(void){
        unregister_chrdev(LED_DEVMAJOR, LED_DEVNAME);
        printk(" %s unregisterd.\n",LED_DEVNAME);
        iounmap(mem_base);
}

module_init(led_init);
module_exit(led_exit);
```

- 드라이버 등록
  - register_chrdev(): 문자 디바이스 드라이버 등록
  - ioremap_nocache(): 입출력장치 주소를 커널공간의 가상주소로 맵핑
- 드라이버 해제
  - unregister_chrdev(): 문자 디바이스 드라이버 등록 해제
  - iounmap(): 커널공간의 가상주소로 맵핑된 장치의 공간을 해제

디바이스 파일 생성 및 모듈 설치

- 디바이스 파일(노드) 생성
  - 디바이스를 사용하기 위해서는 노드(파일)을 생성해야 함

    # mknod /dev/led c **239** 0

    ... 디바이스 드라이버의 major 번호와 같아야 함

    형식: mknod /dev/파일이름 드라이버타입 주번호 부번호

- 디바이스 파일을 통해서 입출력 수행

  파일 이름 /dev/led

  → 문자장치 major 번호 c 239

  → chrdev 테이블 참조 p = chrdev[239]

  → 파일 연산 함수 p->fops->read();

- 디바이스 드라이버 모듈 설치

  # insmod led.ko

디바이스 드라이버 응용 프로그램

- 인수값을 LED로 출력하는 응용 프로그램: "app_led.c"

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>

#include <sys/types.h>
#include <sys/ioctl.h>
#include <sys/stat.h>

int main(void)
{
        int dev, data, rdata;

        dev = open("/dev/led", O_RDWR);
        if (dev < 0) {
                fprintf(stderr, "cannot open LED device\n");
                return 1;
        }
```


디바이스 드라이버 응용 프로그램 (2)

```c
// write
        printf("Input LED data (hex) : ");
        scanf("%x", &data);
        write(dev, &data, 4);
// read
        read(dev, &rdata, 4);
        printf("led data = %x\n", rdata);
        return 0;
}
```

- 입출력 순서
  - open()
  - read() 또는 write()
  - close()

응용 프로그램 컴파일 및 실행

- 컴파일하기

  # cc –o app_led app_led.c

- 실행하기

  # ./app_led