Xilinx Zynq FPGA, TI DSP, MCU 기반의 프로그래밍 전문가 과정

<리눅스 커널>

2018.04.13 - 37 일차

강사 - 이상훈

gcccompil3r@gmail.com

학생 - 안상재

sangjae2015@naver.com

<Chapter 8 디바이스 드라이버>

1. 디바이스 드라이버 일반
- 유닉스 계열 시스템에서 모든 것은 파일이다.
- 사용자 태스크 입장에서는 접근 하려는 파일이 어떤 것인지 신경 쓰지 않고 open(), read(), write(), close() 등의 일관된 함수 인터페이스를 통해서 다양한 파일에 접근함.

가. 사용자 입장에서 디바이스 드라이버

1) 사용자 태스크가 장치 파일을 접근 할때는 아래의 file_operations 구조체의 함수포인터를 이용함.

```
struct file_operations{
struct module *owner;
int (*open) (struct inode *, struct file *);
ssize_t (* read) (struct file *, char __user *, size_t, loff_t *);
ssize_t (* write) (struct file *, const char __user *, size_t, loff_t *);
int (*release) (struct inode *, struct file *);
int (*mmap) (struct file *, struct vm_area_struct *);
int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
loff_t (*llseek) (struct file *, loff)t, int);
...
};
```

2) 여러 개의 디바이스 드라이버를 구분하기 위해 각 디바이스 드라이버마다 주번호를 할당한다.
- inode 구조체에 i_rdev 가 존재하고 i_rdev 에 주 번호와 부 번호를 저장한다.
- 주 번호 : 장치의 종류, 주 번호를 통해서 디바이스 드라이버를 선택함.
- 부 번호 : 같은 디바이스 드라이버를 사용하는 장치가 복수 개 있을 때 이들을 서로 구분하기 위해 사용됨.

나. 개발자 입장에서 디바이스 드라이버

1) 디바이스 드라이버 구성
- 리눅스 디바이스 드라이버는 특정 하드웨어를 위한 디바이스 드라이버 코어 와, 코어를 리눅스에서 사용가능한 형태로 만들어 주기 위한 일종의 래퍼로 구성됨.
- 디바이스 드라이버 코어 : 하드웨어 메뉴얼을 참조해 해당 하드웨어 특성에 맞게 작성됨.
- 래퍼 : 작성된 디바이스 드라이버 코어를 커널에 등록시키고, 사용자 태스크가 장치 파일을 통해 접근할 수 있게 하기 위해 사용자 태스크가 호출할 함수들과 코어의 함수를 연결 시켜줌.

2) 디바이스 드라이버의 관리 구조
- 사용자 태스크는 특정 디바이스 드라이버가 어떤 함수를 제공하는지 일일이 알 필요 없이 file_operations 구조체에 정의되어 있는 함수를 통해 일관되게 장치를 접근할 수 있음.
- 디바이스 드라이버 개발자는 file_operations 구조체의 일관된 인터페이스만을 고려한 됨.
- 디바이스 드라이버 개발자는 file_operations 구조체에 정의되어 있는 함수를 디바이스 드라이버 내에 구현해 줌으로써 간단히 개발을 완료할 수 있다.

2. 문자 디바이스 드라이버 구조
소스 코드

<커널 프로그램>

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

#define DEVICE_NAME "mydrv"
#define MYDRV_MAX_LENGTH 4096
#define MIN(a,b) (((a) < (b)) ? (a) : (b))

struct class *myclass;
struct cdev *mycedv;
struct device *mydevice;
dev_t mydev;

static char *mydrv_data;
static int mydrv_read_offset, mydrv_write_offset;

static int mydrv_open(struct inode *inode, struct file *file)
{
        printk("%s\n", __FUNCTION__);
        return 0;
}

static int mydrv_release(struct inode *inode, struct file *file)
{
        printk("%s\n", __FUNCTION__);
        return 0;
}

static ssize_t mydrv_read(struct file *file, char *buf, size_t count, loff_t *ppos)
{
        if( (buf == NULL) || (count < 0) )
                        reyurn -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)
                return 0;

        if( copy_form_user(mydrv_data + mydrv_write_offset, buf, count) ) /* 유저영역으로부터 커널
                                                                영역으로 buf 값을 count 만큼 복사 */
                return -EFAULT;
        mydrv_write_offset += count;        // 다음에 write할 위치를 이동시킴
        return count;
}

struct file_operations mydrv_fops = { // file_operations 의 함수포인터 다발에 커스텀 함수를 물림
        .owner = THIS_MODULE,
        .read = mydrv_read,
        .write = mydrv_write,
        .open = mydrv_open,
        .release = mydrv_release,
};

int mydrv_init(void)
{
        if( alloc_chrdev_region(&mydev, 0, 1, DEVICE_NAME) < 0 )
  // 캐릭터 디바이스의 주번호를 할당 받고 커널에 등록함
                return -EBUSY;

        myclass = class_create(THIS_MODULE, "mycharclass");  /* 어떤 종류의 장치로 할 것인지
                                                                정함 */
        if(IS_ERR(myclass))
        {
                unregister_chrdev_region(mydev, 1);
                return PTR_ERR(myclass);
        }

        mydevice = device_create(myclass, NULL, mydev, NULL, "mydevicefile");
                                                // 실제 장치를 물림
        if(IS_ERR(mydevice))
        {
                clasS_destroy(myclass);
                unregister_chrdev_region(mydev, 1);
                return PTR_ERR(mydevice);
        }

        mycdev = cdev_alloc();
        mycdev->ops = &mydrv_fops;
        mycdev->owner = THIS_MODULE;  // 이 모듈에 권한이 있음

        if( cdev_add(mycdev, mydev, 1) < 0)
        {
                device_destroy(myclass, mydev);
                class_destroy(myclass);
                unregister_chrdev_region(mydev, 1);
                return - EBUSY;
        }
```

```
        mydrv_data = (char *)kmalloc(MYDRV_MAX_LENGTH *sizeof(char), GFP_KERNEL);
        // GFP_KERNEL 옵션은 반드시 이 작업을 수행하고 넘어가야함 (blocking)
        mydrv_read_offset = mydrv_write_offset = 0;
        return 0;
}

void mydrv_cleanup(void)
{
        kfree(mydrv_data);
        cdev_del(mycdev);
        device_destroy(myclass, mydev);
        class_destroy(myclass);
        unregister_chrdev_region(mydev, 1);
}

module_init(mydrv_init);    // insmod가 동작할 때 어떤 함수가 동작할지 정함
module_exit(mydrv_cleanup)  // rmmod가 동작할 때 어떤 함수가 동작할지 정함
MODULE_LICENSE("GPL");

////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

<사용자 태스크>

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

#define MAX_BUFFER 26
char buf_in[MAX_BUFFER];
char buf_outp[MAX_BUFFER];

int main(void)
{
        int fd, i, c = 65;
        if( (fd = open("/dev/mydevicefile", O_RDWR)) < 0 ) /* open을 하는 순간 file_operations의 함
                                                              수들이 바뀜 */
        {
                perror("open error");
                return -1;
        }

        for(i=0;i<MAX_BUFFER;i++)
        {
                buf_out[i] = c++;
                buf_in[i] = 65;
        }

        for(i=0;i<MAX_BUFFER;i++)
        {
                fprintf(stderr, "%c", buf_int[i]);
        }

        fprintf(stderr, "\n");

        write(fd, buf_out, MAX_BUFFER);  // copy from user
        read(fd, buf_in, MAX_BUFFER);    // copy to user

        for(i=0;i<MAX_BUFFER;i++)
```

```
                fprintf(stderr, "%c", buf_in[i]);

        fprintf(stderr, "\n");

        close(fd);
        return 0;
}
```