Module 16: ioctl을 활용한 LED 제어 디바이스 드라이버

ESP30076 임베디드 시스템 프로그래밍
(Embedded System Programming)

조 윤 석

전산전자공학부

주차별 목표

- ioctl()을 활용법 배우기

- 커널 타이머와 ioctl을 활용하여 LED 제어용 디바이스 드라이브 작성하기

IOCTL을 이용한 드라이버 제어

- ioctl() 함수 활용
  - 어떤 경우에는 읽는 용도로만 쓰고, 어떨 때는 쓰는 용도로만 쓰다가 하나의 명령으로 읽고 쓰기를 하거나 혹은 어떠한 값을 어떤 방식으로 처리하고 싶은 경우
  - 각 디바이스마다 고유하게 선언하여 사용하는 전용 함수
- ioctl() 함수의 특징
  - read(), write() 함수와 같이 읽기와 쓰기 처리가 가능
  - 하드웨어를 제어하거나 상태값을 읽어올 수도 있음
  - 응용프로그램의 요구에 따라 디바이스 드라이버의 매개변수 해석이 달라짐

ioctl() 함수의 매개변수 전달 과정

```
ret = ioctl ( int fd, int request, char *argp);
        // #include <sys/ioctl.h>
```

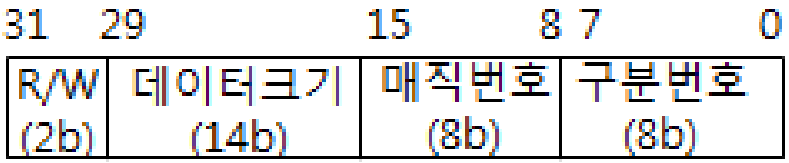
```
int dev_ioctl(struct inode *inode, struct file *filp,
                unsigned int cmd, unsigned long arg)
{
        ...
        return ret;
}
```

디바이스 드라이버 내의 ioctl() 구조

```
int dev_ioctl (struct inode *inode, struct file *filp,
               unsigned int cmd, unsigned long arg)
{
        switch( cmd )
        {
                case 구별상수1: 처리루틴1; break;
                case 구별상수2: 처리루틴2; break;
                ...
        }
        ...
        return 0;
}
```

ioctl()

- ioctl()
  - cmd(32-bit), arg
  - 32-bit cmd 구성



  - cmd 명령을 만드는 매크로

매크로 이름	기능
_IO(type, nr)	부가적인 데이터가 없는, 즉 매개변수가 없는 명령 생성
_IOR(type, nr, datatype)	디바이스 드라이버에서 데이터를 읽어오기(R) 위한 명령 생성
_IOW(type, nr, datatype)	디바이스 드라이버에서 데이터를 쓰기(R) 위한 명령 생성
_IOWR(type, nr, datatype)	디바이스 드라이버에서 데이터를 읽고 쓰기 위한 명령 생성

cmd 선언

- Macro 인자 의미: _IOR(type, nr, datatype)
  - type: 매직번호 (8-bit)
  - nr: 구분번호/명령어 번호 (8-bit)
  - datatype: 전송할 데이터 타입

- 실제 cmd 선언 예

```
#define MY_IOCTL_READ   _IOR(MY_IOCTL_MAGIC, 0, int)
#define MY_IOCTL_WRITE   _IOW(MY_IOCTL_MAGIC, 1, int)
#define MY_IOCTL_STATUS   _IO(MY_IOCTL_MAGIC, 2)
#define MY_IOCTL_READ_WRITE   _IOWR(MY_IOCTL_MAGIC, 3, int)
#define MY_IOCTL_NR 4 // 네 가지 명령 지정
```

ioctl(): cmd 명령을 해석하는 매크로

- cmd 명령을 해석하는 매크로
  - include/ioctl.h or include/asm-i386/ioctl.h

매크로 이름	기능
_IOC_NR(cmd)	구분 번호 필드 값을 읽는 매크로
_IOC_TYPE(cmd)	매직 번호 필드 값을 읽는 매크로
_IOC_SIZE(cmd)	데이터 크기 필드 값을 읽는 매크로
_IOC_DIR(cmd)	읽기와 쓰기 속성 필드 값을 읽는 매크로

ioctl()을 사용하여 LED 제어하기

❑ LED 제어 기능

기능	내용	인자
0	0이 입력되면, LED를 모두 OFF	arg 사용안함
1	1이 입력되면, LED를 모두 ON	arg 사용안함
2	2가 입력되고 arg값이 들어오면, arg값에 따라 LED를 토글	arg 사용

– 기능이 2이고 arg 값이 15인 경우, 하위 LED 4개가 모두 꺼지고 상위 LED 4개가 켜지며 이후 토글 동작으로 LED 점멸

LED 제어 디바이스 드라이버

root@esp:~# **mkdir /root/work/dd/ioctl_ioremap**
root@esp:~# **cd /root/work/dd/ioctl_ioremap**
root@esp:~/work/dd/ioctl_ioremap# **vi ioctl_led_driver.h**

```
#ifndef __LED_KERNEL_TIMER_DRIVER__
#define __LED_KERNEL_TIMER_DRIVER__

#include <linux/module.h>
#include <linux/fs.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <asm/io.h>
#include <asm/uaccess.h>
#include <linux/errno.h>
#include <linux/types.h>
#include <linux/fcntl.h>
#include <linux/slab.h> // kmalloc ()
#include <linux/timer.h>

typedef struct {
    struct timer_list timer;
    unsigned long led_state;
} __attribute__((packed)) KERNEL_TIMER_STRUCT;
#endif
```

LED 제어 디바이스 드라이버

root@esp:~/work/dd/ioctl_ioremap# **vi myled_ioctl.h**

```
#ifndef __MYLED_IOCTL__
#define __MYLED_IOCTL__

#define IOCTL_LED_MAGIC 'h'

typedef struct {
    unsigned long set_data;
    // unsigned long get_data;
} __attribute__((packed)) ioctl_led_data_t;

#define LED_OFF 0xFF
#define LED_ON  0x00

#define LED_MODE_OFF 0
#define LED_MODE_ON  1
#define LED_MODE_TW  2

#define LED_IOCTL_CMD_OFF _IO(IOCTL_LED_MAGIC, LED_MODE_OFF)
#define LED_IOCTL_CMD_ON  _IO(IOCTL_LED_MAGIC, LED_MODE_ON)
#define LED_IOCTL_CMD_TW  _IOW(IOCTL_LED_MAGIC, LED_MODE_TW, ioctl_led_data_t)

#define IOCTL_LED_MAXNR 3 // three commands; OFF, ON< TW

#define DEV_NAME "/dev/iom_ioctl_led"

#endif
```

LED 제어 디바이스 드라이버

root@esp:~/work/dd/fnd_ioremap# **vi ioctl_led_driver.c**

```
/* ioctl LED driver Example
   FILE: ioctl_led_driver.c */
#include "./myled_ioctl.h"
#include "./ioctl_led_driver.h"

#define IOM_LED_IOCTL_MAJOR_NUM 243
#define IOM_LED_ADDRESS 0xA8000000
#define TIME_STEP (5*HZ/10) /* 0.5 sec */

#define BLINK_MODE 1

static int ledport_usage = 0;
static unsigned char *iom_led_addr;
static KERNEL_TIMER_STRUCT *ptimermgr = NULL;
static unsigned char led_blink = 0;
static unsigned char led_tw_value = 0;

MODULE_LICENSE("GPL");
MODULE_AUTHOR("HGU");

int iom_ioctl_led_init(void);
void iom_ioctl_led_exit(void);
module_init(iom_ioctl_led_init);
module_exit(iom_ioctl_led_exit);

void led_timer_timeover(unsigned long arg); /* timer function */
void led_timer_register(KERNEL_TIMER_STRUCT *pdata, unsigned long timerover);
```

ioctl_led_driver.c (2)

```
int iom_ioctl_led_open (struct inode *, struct file *);
int iom_ioctl_led_release (struct inode *, struct file *);
int iom_ioctl_led_ioctl (struct inode *, struct file *, unsigned int, unsigned long);

struct file_operations iom_ioctl_led_fops = {
    .owner = THIS_MODULE,
    .open = iom_ioctl_led_open,
    .ioctl = iom_ioctl_led_ioctl,
    .release = iom_ioctl_led_release,
};

int __init iom_ioctl_led_init(void)
{
    int major_num;

    major_num = register_chrdev(IOM_LED_IOCTL_MAJOR_NUM, DEV_NAME, &iom_ioctl_led_fops);

    if ( major_num < 0 ) {
        printk(KERN_WARNING"%s: can't get or assign major number %d₩n", DEV_NAME,
IOM_LED_IOCTL_MAJOR_NUM);
        return major_num;
    }

    printk("Success to load the device %s. Major number is %d₩n", DEV_NAME, IOM_LED_IOCTL_MAJOR_NUM);

    iom_led_addr = ioremap(IOM_LED_ADDRESS, 0x1);

    return 0;
}
```

ioctl_led_driver.c (3)

```
void led_timer_register(KERNEL_TIMER_STRUCT *pdata, unsigned long timeover)
{
   init_timer(&(pdata->timer));
   pdata->timer.expires = get_jiffies_64() + timeover;
   pdata->timer.function = led_timer_timeover; /* handler of the timeout */
   pdata->timer.data = (unsigned long)pdata; /* argument of the handler */
   add_timer(&(pdata->timer));
}


void led_timer_timeover(unsigned long arg)
{
   KERNEL_TIMER_STRUCT *pdata=NULL;

   pdata = (KERNEL_TIMER_STRUCT *) arg;

   if ( led_blink == BLINK_MODE ) {
      pdata->led_state = ~(pdata->led_state);
      if(pdata->led_state )
         outb(led_tw_value, (unsigned int)iom_led_addr);
      else
         outb(~led_tw_value, (unsigned int)iom_led_addr);
   } else {
      outb(LED_OFF, (unsigned int)iom_led_addr);
   }

   led_timer_register(pdata, TIME_STEP);
} /* end of led_timer_timeover() */
```

ioctl_led_driver.c (4)

```
int iom_ioctl_led_open (struct inode *inode, struct file *filp)
{
   if ( ledport_usage )
      return -EBUSY;
   ledport_usage = 1;
   return 0;
}

int iom_ioctl_led_release (struct inode *inode, struct file *filp)
{
   ledport_usage = 0;
   return 0;
}

int iom_ioctl_led_ioctl (struct inode *inode, struct file *filp, unsigned int cmd, unsigned long arg)
{
   ioctl_led_data_t led_data;
   int size, ret;


   if ( _IOC_TYPE(cmd) != IOCTL_LED_MAGIC ) return -EINVAL;
   if ( _IOC_NR(cmd) >= IOCTL_LED_MAXNR ) return -EINVAL;
   size = _IOC_SIZE(cmd);

   printk(KERN_ALERT"[_IOC_TYPE(cmd)]=%c, [_IOC_TYPE_NR(cmd)]=%d, [_IOC_SIZE(cmd)] = %d₩n", _IOC_TYPE(cmd),
_IOC_NR(cmd), size);
```

ioctl_led_driver.c (5)

```
switch (cmd) {
    case LED_IOCTL_CMD_OFF:
        led_blink = 0;
        if(ptimermgr != NULL ) {
            del_timer(&(ptimermgr->timer));
            kfree(ptimermgr);
            ptimermgr = NULL;
        }
        outb(LED_OFF, (unsigned int) iom_led_addr);
        break;
    case LED_IOCTL_CMD_ON:
        led_blink = 0;
        if(ptimermgr != NULL ) {
            del_timer(&(ptimermgr->timer));
            kfree(ptimermgr);
            ptimermgr = NULL;
        }
        outb(LED_ON, (unsigned int) iom_led_addr);
        break;
    case LED_IOCTL_CMD_TW:
        ret = copy_from_user( (void *)&led_data, (const void *)arg, sizeof(led_data));
        led_blink = 1;
        led_tw_value = led_data.set_data;
        ptimermgr = kmalloc ( sizeof(KERNEL_TIMER_STRUCT), GFP_KERNEL|__GFP_ZERO);
        if ( ptimermgr == NULL ) return -ENOMEM;
        led_timer_register(ptimermgr, TIME_STEP);
        break;
    default:            break;
  }
  return 0;
}
```

ioctl_led_driver.c (6)

```
void __exit iom_ioctl_led_exit(void)
{
    if ( ptimermgr != NULL ) {
        printk(KERN_ALERT"[Driver][EXIT]... ptimermgr is not NULL...₩n");
        del_timer(&(ptimermgr->timer));
        kfree(ptimermgr);
        ptimermgr = NULL;
    }

    outb(LED_OFF, (unsigned int)iom_led_addr);

    iounmap(iom_led_addr);
    unregister_chrdev(IOM_LED_IOCTL_MAJOR_NUM, DEV_NAME);
    printk("Success to unload the device %s...₩n", DEV_NAME);
}
```

LED 구동 응용 프로그램 (ioctl_led_test.c)

root@esp:~/work/dd/fnd_ioremap# **vi ioctl_led_test.c**

```
/* IOCTL Test program */
#include "myled_ioctl.h"
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <fcntl.h>

int main(int argc, char **argv)
{
    int led_fd;
    ioctl_led_data_t led_data;
    unsigned char get_cmd, get_arg;

    led_fd = open(DEV_NAME, O_RDWR);
    if ( led_fd < 0 ) {
        printf("LED ioctl driver open failure..₩n");
        return -1;
    }

    if ( !(argc == 2 || argc == 3) ) {
        printf("%s [cmd(0 ~ 2)] or [arg(0~255)]₩n", argv[0]);
        printf(" cmd: 0 - OFF₩n");
        printf(" cmd: 1 - ON₩n");
        printf(" cmd: 2 - Twinkle with the value of arg(0~255)₩n");
        exit(1);
    }
```

ioctl_led_test.c (2)

```
get_cmd = (unsigned char) atoi(argv[1]);
  if ( argc == 3 ) {
     get_arg = (unsigned char) atoi(argv[2]);
     if ( get_arg < 0 || get_arg > 255 ) get_arg = 255;
  }

  switch (get_cmd) {
     case LED_MODE_OFF:
        printf("[TEST]: OFF mode₩n");
        ioctl(led_fd, LED_IOCTL_CMD_OFF);
        break;
     case LED_MODE_ON:
        printf("[TEST]: ON mode₩n");
        ioctl(led_fd, LED_IOCTL_CMD_ON);
        break;
     case LED_MODE_TW:
        printf("[TEST]: TW mode₩n");
        led_data.set_data = get_arg;
        ioctl(led_fd, LED_IOCTL_CMD_TW, &led_data);
        break;
     default:
        break;
  }
  close(led_fd);
  return 0;
}
```

컴파일

% vi Makefile

```
CC = arm-linux-gcc
obj-m = ioctl_led_driver.o

KDIR = /root/download/kernel-2.6.35
PWD = $(shell pwd)

TEST_TARGET = ioctl_led_test
TEST_SRCS = $(TEST_TARGET).c

all: module test_pgm

module:
	$(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules
test_pgm:
	$(CC) $(TEST_SRCS) -o $(TEST_TARGET)
clean:
	rm -rf *.ko
	rm -rf *.o *.mod.*
	rm -rf *.symvers *.order .*.cmd
	rm -rf $(TEST_TARGET)
```

% make

실행

- 타겟보드에서 실행
  - Host 컴퓨터의 /root 디렉토리를 NFS로 연결

```
# cd /root/nfs/work/dd/ioctl_ioremap
# insmod ioctl_led_driver.ko
# mknod /dev/ioctl_iom_led c 243 0
# ./ioctl_led_test 1
# ./ ioctl_led_test 0
# ./ ioctl_led_test 2 15
# rmmod ioctl_led_driver
```