03.ioctl(Input/Output control)

List

ioctl(Input/Output control)

DESCRIPTION

- **ioctl이란 하드웨어의 제어와 상태 정보를 얻기 위해 제공되는 오퍼레이션 입니다.**
  - read(), write() 오퍼레이션을 이용하여 데이터의 읽고 쓰기,등의 기능은 가능하지만 하드웨어의 제어 및 상태 정보까지는 확인할 수 없습니다.
  - 예를 들어 다음과 같은 제어를 위해 사용됩니다.
    - SPI 통신 속도 설정를 하거나, I2C 슬레이브 어드레스 설정,등의 작업은 read, write만으로는 할 수 없습니다.
    - CD-ROM 디바이스 드라이버는 실제 장치에서 디스크를 꺼내도록 지시할 수 있는 ioctl 요청 코드를 제공합니다.

SYNOPSIS

- **ioctl함수의 인자값 구성은 다음과 같습니다.**
  - 첫번재 인수는 fd는 open함수에서 얻은 파일 디스크립터를 전달합니다.
  - 두번째 인수는 request는 디바이스에 전달한 명령
  - 이 외에도 개발자의 설정에 따라 추가적인 인자를 생성할 수 있습니다.

ioctl()

```
#include <sys/ioctl.h>
int ioctl(int d, int request, ...);
```

- https://en.wikipedia.org/wiki/Ioctl
- http://man7.org/linux/man-pages/man2/ioctl.2.html

Using macros to generate ioctl command numbers

- **Linux 헤더 파일 "/usr/include/asm/ioctl.h" 는 ioctl의 커맨드 번호를 작성하는데 사용해야하는 매크로가 정의되어 있습니다.**
- **다음과 같은 커맨드 매크로 형태를 사용할 수 있습니다.**

Type

Macro	Description
_IO(int type, int number)	type, number 값만 전달하는 단순한 ioctl에 사용됩니다.
_IOR(int type, int number, data_type)	디바이스 드라이버에서 데이터를 읽는 ioctl에 사용됩니다
_IOW(int type, int number, data_type)	디바이스 드라이버에서 데이터를 쓰는 ioctl에 사용됩니다
_IORW(int type, int number, data_type)	디바이스 드라이버에서 데이터를 쓰고 읽는 ioctl에 사용됩니다.

- **매크로의 인자 값을 다음과 같은 형태로 구성되어 있습니다.**

Argument

Argument	Description
type	• 장치 드라이버에 고유하게 선택된 8 비트 정수입니다. • 같은 파일 기술자에 ''청취 할 수있는 다른 드라이버와 충돌하지 않도록 선택해야합니다. • 예를 들어 커널 내부에서 소켓 파일 설명 자로보낸 ioctl 은 두 스택에서 모두 검사 될 수 있기 때문에 TCP 및 IP 스택은 고유 한 번호를 사용 합니다.
number	• 8 비트 정수 • '드라이버 내 에서 드라이버가 서비스하는 서로 다른 종류의 ioctl 명령 마다 고유 번호를 선택해야 합니다.
data_type	• 클라이언트와 드라이버간에 교환되는 바이트 수를 계산하는 데 사용되는 유형 이름입니다.

- **다음과 같이 커맨드 매크로를 정의할 수 있습니다.**
  - "SET_DATA"의 경우 데이터의 입력, 출력, 쓰기가 가능한 매크로입니다.
  - "GET_DATA"의 경우 데이터의 입력, 출력, 읽기가 가능한 매크로입니다.

Example

```
struct ioctl_info{
       unsigned long size;
       unsigned int buf[128];
};

#define             IOCTL_MAGIC         'G'
#define             SET_DATA                        _IOW(IOCTL_MAGIC, 2 , ioctl_info )
#define             GET_DATA                        _IOR(IOCTL_MAGIC, 3 , ioctl_info )
```

- **다음 매크로를 이용하여 정의된 커맨드 매크로의 필드 값을 확인 할 수 있습니다.**

Macro to read field values

Function	Description
_IOC_NR()	number 필드 값을 읽는 매크로
_IOC_TYPE()	type 필드 값을 읽는 매크로
_IOC_SIZE()	data_type 필드 크기를 읽는 매크로
_IOC_DIR()	읽기와 쓰기 속성 필드 값을 읽는 매크로

- http://www.circlemud.org/jelson/software/fusd/docs/node31.html

Example

- **ioctl** .
  - 여기에서 ioctl()함수를 사용하여 하드웨어를 제어하는 부분은 다루지 않습니다.
- ioctl() 함수와 I2C, GPIO,등을 이용하여 하드웨어의 제어를 학습하고 싶은 분은 아래 사이트를 참고하시면 됩니다.

- KR
  - http://forum.falinux.com/zbxe/index.php?mid=lecture_tip&search_target=nick_name&search_keyword=%EC%9D%B4%EC%9A%B0%EC%98%81&page=3&document_srl=569969
  - http://forum.falinux.com/zbxe/index.php?mid=lecture_tip&search_target=nick_name&search_keyword=%EC%9D%B4%EC%9A%B0%EC%98%81&page=3
  - http://forum.falinux.com/pds/data-s2410/No14-1.pdf
- JP
  - https://qiita.com/take-iwiw/items/1fdd2e0faaaa868a2db2

Source code of module

chardev_ioctl

- **사용자 공간(User space)에서 해당 디바이스를 열어서 ioctl()함수를 호출하면 chardev_ioctl() 함수가 호출되며 다음 기능을 처리합니다.**
  - cmd의 값이 "SET_DATA" 일 경우 copy_from_user() 함수를 이용하여 사용자 공간으로 부터 전달받은 데이터를 info 구조체 변수에 복사합니다.
  - cmd의 값이 "GET_DATA" 일 경우 copy_to_user() 함수를 이용하여 Kernel 영역에 저장된 info 구조체 변수 데이터를 사용자 공간으로 복사합니다.

chardev.c

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

#include "chardev.h"
MODULE_LICENSE("Dual BSD/GPL");

#define DRIVER_NAME "chardev"

static const unsigned int MINOR_BASE = 0;
static const unsigned int MINOR_NUM  = 1;
static unsigned int chardev_major;
static struct cdev chardev_cdev;
static struct class *chardev_class = NULL;

static int     chardev_open(struct inode *, struct file *);
static int     chardev_release(struct inode *, struct file *);
static ssize_t chardev_read(struct file *, char *, size_t, loff_t *);
static ssize_t chardev_write(struct file *, const char *, size_t, loff_t *);
static long chardev_ioctl(struct file *, unsigned int, unsigned long);

struct file_operations s_chardev_fops = {
    .open    = chardev_open,
    .release = chardev_release,
    .read    = chardev_read,
```

```
    .write   = chardev_write,
    .unlocked_ioctl = chardev_ioctl,
};

static int chardev_init(void)
{
    int alloc_ret = 0;
    int cdev_err = 0;
    int minor = 0;
    dev_t dev;

    printk("The chardev_init() function has been called.");

    alloc_ret = alloc_chrdev_region(&dev, MINOR_BASE, MINOR_NUM, DRIVER_NAME);
    if (alloc_ret != 0) {
        printk(KERN_ERR  "alloc_chrdev_region = %d\n", alloc_ret);
        return -1;
    }
    //Get the major number value in dev.
    chardev_major = MAJOR(dev);
    dev = MKDEV(chardev_major, MINOR_BASE);

    //initialize a cdev structure
    cdev_init(&chardev_cdev, &s_chardev_fops);
    chardev_cdev.owner = THIS_MODULE;

    //add a char device to the system
    cdev_err = cdev_add(&chardev_cdev, dev, MINOR_NUM);
    if (cdev_err != 0) {
        printk(KERN_ERR  "cdev_add = %d\n", alloc_ret);
        unregister_chrdev_region(dev, MINOR_NUM);
        return -1;
    }

    chardev_class = class_create(THIS_MODULE, "chardev");
    if (IS_ERR(chardev_class)) {
        printk(KERN_ERR  "class_create\n");
        cdev_del(&chardev_cdev);
        unregister_chrdev_region(dev, MINOR_NUM);
        return -1;
    }

    device_create(chardev_class, NULL, MKDEV(chardev_major, minor), NULL, "chardev%d", minor);
    return 0;
}

static void chardev_exit(void)
{
    int minor = 0;
    dev_t dev = MKDEV(chardev_major, MINOR_BASE);

    printk("The chardev_exit() function has been called.");

    device_destroy(chardev_class, MKDEV(chardev_major, minor));

    class_destroy(chardev_class);
    cdev_del(&chardev_cdev);
    unregister_chrdev_region(dev, MINOR_NUM);
}

static int chardev_open(struct inode *inode, struct file *file)
{
    printk("The chardev_open() function has been called.");
    return 0;
}

static int chardev_release(struct inode *inode, struct file *file)
{
    printk("The chardev_close() function has been called.");
    return 0;
}
```

```
static ssize_t chardev_write(struct file *filp, const char __user *buf, size_t count, loff_t *f_pos)
{
    printk("The chardev_write() function has been called.");
    return count;
}

static ssize_t chardev_read(struct file *filp, char __user *buf, size_t count, loff_t *f_pos)
{
    printk("The chardev_read() function has been called.");
    return count;
}

static struct ioctl_info info;
static long chardev_ioctl(struct file *filp, unsigned int cmd, unsigned long arg)
{
    printk("The chardev_ioctl() function has been called.");

    switch (cmd) {
        case SET_DATA:
            printk("SET_DATA\n");
            if (copy_from_user(&info, (void __user *)arg, sizeof(info))) {
                return -EFAULT;
            }
            printk("info.size : %ld, info.buf : %s",info.size, info.buf);
            break;
        case GET_DATA:
            printk("GET_DATA\n");
            if (copy_to_user((void __user *)arg, &info, sizeof(info))) {
                return -EFAULT;
            }
            break;
        default:
            printk(KERN_WARNING "unsupported command %d\n", cmd);

        return -EFAULT;
    }
    return 0;
}

module_init(chardev_init);
module_exit(chardev_exit);
```

- **매크로 정의는 다음과 같습니다.**
    - SET_DATA은 _IOW(입력, 출력, 쓰기)로 설정하고 인자값의 형태는 "struct ioctl_info"로 설정하였습니다.
    - SET_DATA은 _IOR(입력, 출력, 읽기)로 설정하고 인자값의 형태는 "struct ioctl_info"로 설정하였습니다.

chardev.h

```
#ifndef CHAR_DEV_H_
#define CHAR_DEV_H_
#include <linux/ioctl.h>

struct ioctl_info{
       unsigned long size;
       char buf[128];
};

#define             IOCTL_MAGIC         'G'
#define             SET_DATA            _IOW(IOCTL_MAGIC, 2 ,struct ioctl_info)
#define             GET_DATA            _IOR(IOCTL_MAGIC, 3 ,struct ioctl_info)

#endif
```

Makefile

```
obj-m := chardev.o

all:
	make -C /lib/modules/$(shell uname -r)/build M=$(shell pwd) modules
clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(shell pwd) clean
```

Source code of Test program

- **테스트 프로그램은 다음과 같이 동작합니다.**
    - open() 함수를 이용하여 "/dev/chardev0" 파일을 열어 fd값을 얻습니다.
    - ioctl() 함수를 이용하여 사용자 공간에 저장된 데이터를 커널 영역에 데이터를 복사하기 위해 fd, "SET_DATA", 커널에 저장할 데이터를 인자 값(&set_info)으로 전달합니다.
    - ioctl() 함수를 이용하여 커널 영역에 저장된 데이터를 사용가 공간으로 복사하기 위해 fd, "GET_DATA", 사용자 공간에에 데이터를 저장할 변수를 인자 값(&get_info)으로 전달합니다.
    - printf() 함수를 이용하여 사용자 공간에 복사된 데이터를 출력합니다.
    - close() 함수를 이용하여 fd를 닫습니다.

test.c

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/ioctl.h>
#include "chardev.h"

int main()
{
    int fd;
    struct ioctl_info set_info;
    struct ioctl_info get_info;

         set_info.size = 100;
        strncpy(set_info.buf,"lazenca.0x0",11);

    if ((fd = open("/dev/chardev0", O_RDWR)) < 0){
        printf("Cannot open /dev/chardev0. Try again later.\n");
    }

    if (ioctl(fd, SET_DATA, &set_info) < 0){
        printf("Error : SET_DATA.\n");
    }


    if (ioctl(fd, GET_DATA, &get_info) < 0){
        printf("Error : SET_DATA.\n");
    }

         printf("get_info.size : %ld, get_info.buf : %s\n", get_info.size, get_info.buf);

    if (close(fd) != 0){
        printf("Cannot close.\n");
    }
    return 0;
}
```

Make & Run

- 해당 모듈을 빌드하고 테스트 프로그램을 실행하면 다음과 같이 전달된 데이터들이 정상적으로 복사되는 것을 확인할 수 있습니다.

Build & Run

```
lazenca0x0@ubuntu:~/Kernel/Module/ioctl$ make
make -C /lib/modules/4.18.0-11-generic/build M=/home/lazenca0x0/Kernel/Module/ioctl modules
make[1]: Entering directory '/usr/src/linux-headers-4.18.0-11-generic'
Makefile:982: "Cannot use CONFIG_STACK_VALIDATION=y, please install libelf-dev, libelf-devel or elfutils-libelf-devel"
  CC [M]  /home/lazenca0x0/Kernel/Module/ioctl/chardev.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/lazenca0x0/Kernel/Module/ioctl/chardev.mod.o
  LD [M]  /home/lazenca0x0/Kernel/Module/ioctl/chardev.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.18.0-11-generic'
lazenca0x0@ubuntu:~/Kernel/Module/ioctl$ sudo insmod chardev.ko
[sudo] password for lazenca0x0:
lazenca0x0@ubuntu:~/Kernel/Module/ioctl$ ./test
get_info.size : 100, get_info.buf : lazenca.0x0
lazenca0x0@ubuntu:~/Kernel/Module/ioctl$ dmesg |tail
[   53.358158] rfkill: input handler disabled
[   97.190519] chardev: loading out-of-tree module taints kernel.
[   97.190571] chardev: module verification failed: signature and/or required key missing - tainting kernel
[   97.191019] The chardev_init() function has been called.
[  117.777326] The chardev_open() function has been called.
[  117.777328] The chardev_ioctl() function has been called.
[  117.777329] SET_DATA
[  117.777330] info.size : 100, info.buf : lazenca.0x0
[  117.777331] The chardev_ioctl() function has been called.
[  117.777331] GET_DATA
lazenca0x0@ubuntu:~/Kernel/Module/ioctl$
```

References

- https://www.wdic.org/w/TECH/ioctl
- http://www.circlemud.org/jelson/software/fusd/docs/node31.html
- https://hunn1979.blogspot.com/2016/04/ioctl.html
- http://guswnsla1223.tistory.com/126
- https://qiita.com/take-iwiw/items/ade0a73d4c05fc7961d3