Embedded System Software [CSE4116]

실습 4주차(2) : Implementing Device Driver

Department of Computer Science and Engineering, Sogang University, Seoul, South Korea

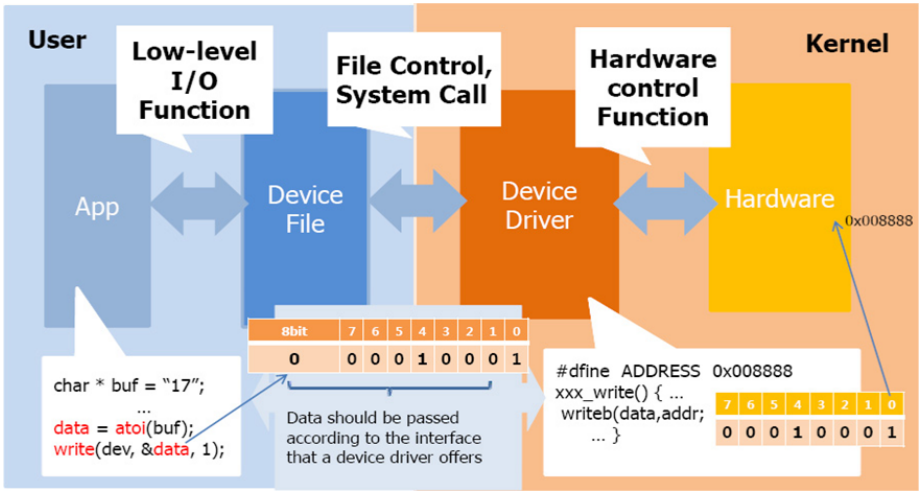
Data-Intensive and Computing and System Laboratory

1. Device Driver Control (Remind)

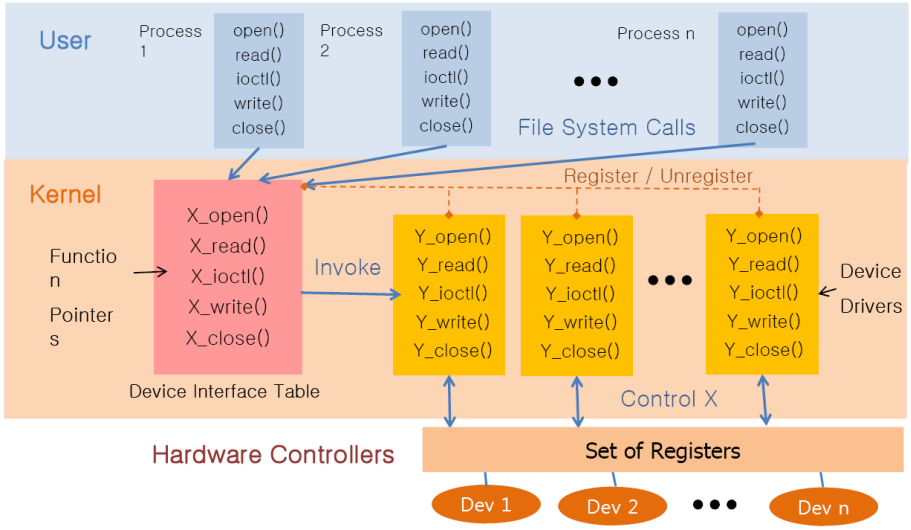
- User Program 이 Device 파일을 open 하여 read/write 와 같은 operation 을 통해 명령을 내리고, Device Drvier 가 이 명령을 해석해 device 를 제어함(값을 쓰거나 읽는 등).
- 따라서 mmap 과는 다르게 세부적인 mechanism(어떤 주소 값에 데이터를 쓰면 device 로 전송)등을 몰라도 사용 가능함.

1.1. Overview

A. **Call Path**



B. **Device Driver Overview**



1.2.Using Device Driver (On the Device-side, Minicom)

- Device Driver 모듈 삽입

```
$ insmod [module]
```

```
[root@ACHRO module]# insmod fpga_led_driver.ko
[ 4464.076231] init module, fpga_led major number : 260
[root@ACHRO module]#
```

성공 시 major number 출력

- 삽입된 모듈 확인

```
$ lsmod
```

```
[root@ACHRO module]# lsmod
Module                  Size  Used by
fpga_dot_driver         1539  0
fpga_fnd_driver         1685  0
fpga_led_driver         1593  0
[root@ACHRO module]#
```

```
[root@ACHRO module]# cat /proc/modules
fpga_dot_driver 1539 0 - Live 0xbf018000
fpga_fnd_driver 1685 0 - Live 0xbf014000
fpga_led_driver 1593 0 - Live 0xbf010000
[root@ACHRO module]#
[root@ACHRO module]#
```

cat /proc/modules 로도 가능함

- (참고) 삽입된 모듈 제거

```
$ rmmod [module]
```

- Device 파일 생성

```
$ mknod /dev/[device name] [type] [major] [minor]
```

```
[root@ACHRO module]# mknod /dev/fpga_led c 260 0
[root@ACHRO module]# ls /dev/fpga*
/dev/fpga_led
```

ls -l /dev 명령어를 통해 device 파일 확인 가능

- 로그 레벨 변경

```
$ echo "7 6 1 7" > /proc/sys/kernel/printk
```

로그 레벨을 변경하지 않을 시 printk 가 출력되지 않음
cat /proc/sys/kernel/printk 로 로그 레벨 확인 가능

```
$ insmod MODULE_NAME.ko
$ lsmod
$ rmmod MODULE_NAME.ko
```

2. Device Driver Register

2.1. Flow

1) Driver 에서 register 할 때 함수의 첫번째 parameter 에 0 을 넘긴다. (시스템이 major # 결정)
2) Return value 로 major #를 받아 printk 를 통해 화면에 출력한다.
3) 이 driver 를 사용하기 위해서 device file 이 존재하는지 확인한다.
4) File 이 존재하면 device file 의 major #가 출력된 값과 동일한지 확인한다.
5) File 이 없던지, major # 다르면, printk 된 major #를 이용해서 새로운 device file 을 생성한다. (mknod)
6) Driver 에서 init 시 printk 를 통하여 major #가 화면에 나오지 않을 경우 /proc/modules 를 통해서도 확인 가능하다.

2.2. 관련 코드

```
#define DEVICE_DRIVER_NAME "driver_test"
long number = 0;
int major_number;
```

```
int __init device_driver_init(void)
{

   printk("device_driver_init\n");

   major_number = register_chrdev(0, DEVICE_DRIVER_NAME, &device_driver_fops);
   if(major_number <0) {
      printk( "error %d\n",major_number);
      return major_number;
   }

   printk( "dev_file: /dev/%s , major: %d\n", DEVICE_DRIVER_NAME, major_number);

   printk("init module\n");
   return 0;
}

void __exit device_driver_exit(void)
{
   printk("device_driver_exit\n");

   unregister_chrdev(major_number, DEVICE_DRIVER_NAME);
}
```

2.3.실행 예시

```
root@achroimx:/data/local/tmp # insmod device_driver_module.ko
[13517.713595] device_driver_init
[13517.716685] dev_file: /dev/driver_test , major: 246
[13517.721609] init module
root@achroimx:/data/local/tmp # mknod /dev/driver_test c 246 0
```

모듈 등록 시 반환 받은 Device file name / Major number

3.Device Driver Operations

- Assigning File Operations

```
struct file_operations xxx_fops {
        .owner  = THIS_MODULE;
        .write  = ...;
        . ...   = ...;
        ...
};
```

→ 파일 operation system call 과 device driver 의 operation 을 mapping 한다.

- 관련 코드

```
int test_device_driver_open(struct inode *, struct file *);
int test_device_driver_release(struct inode *, struct file *);
ssize_t test_device_driver_write(struct file *, const char *, size_t, loff_t *);

static struct file_operations device_driver_fops =
{ .open = test_device_driver_open, .write = test_device_driver_write, .release =
test_device_driver_release };

int test_device_driver_open(struct inode *minode, struct file *mfile) {
   printk("test_device_driver_open\n");
   return 0;
}

int test_device_driver_release(struct inode *minode, struct file *mfile) {
   printk("test_device_driver_release\n");
   return 0;
}

ssize_t test_device_driver_write(struct file *inode, const char *gdata, size_t
length, loff_t *off_what) {
   const char *tmp = gdata;
   char kernel_buff[4];


   printk("Write\n");
   if (copy_from_user(&kernel_buff, tmp, 1)) {
      return -EFAULT;
   }
```

```
    number = simple_strtol(kernel_buff, NULL, 10);
    printk("Copy number to kernel buffer : %d \n",number);

    return 1;
}
```

4. 실습

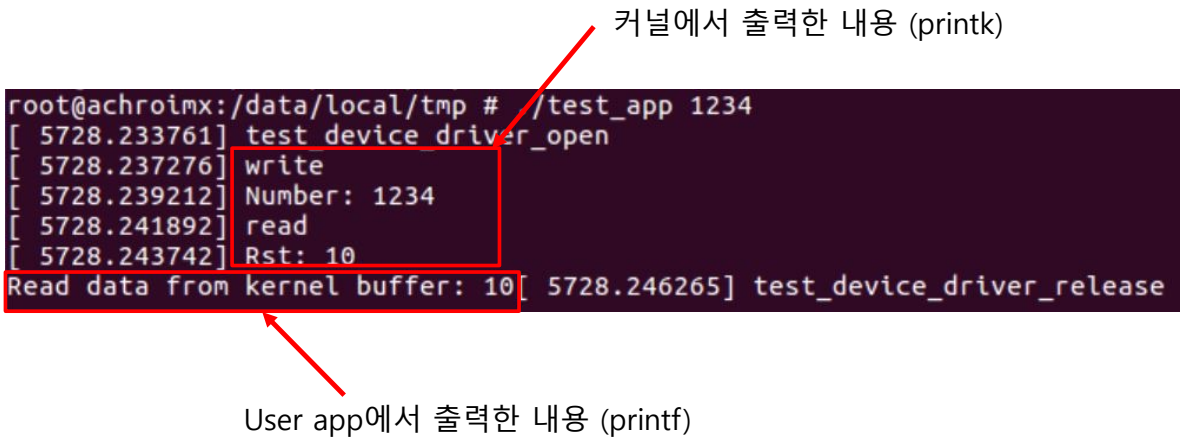
- 제공된 device_driver_module 디렉토리에서 make 수행
  (test_app.c 와 device_driver_module.c 를 읽어볼 것)
- teat_app.c(4 자리 수를 입력받아 커널 버퍼에 write 하는 예제 프로그램.)를 크로스 컴파일
- 컴파일 된 모듈 device_driver_module.ko 과 teat_app.c 의 바이너리 파일을 보드로 전송
- User application 에서 파일 시스템 콜이 device driver 의 operation 들과 매핑되는 것을 확인할 수 있음.

```
int main(int argc, char **argv)
{
    int test_fd;
    char get_number[4];

    if(argc != 2) {
        printf("Usage : [Number]\n");
        return -1;
    }

    test_fd = open("/dev/driver_test", O_RDWR);
    if (test_fd<0){

        printf("Open Failured!\n");
        return -1;
    }
    memcpy(get_number, argv[1], sizeof(get_number));

    write(test_fd, &get_number, sizeof(get_number));
    close(test_fd);

    return 0;
}
```

```
root@achroimx:/data/local/tmp # ./test_app 1234
[ 3431.564353] test_device_driver_open
[ 3431.567868] write
[ 3431.569805] Number: 1234
[ 3431.572657] test_device_driver_release
```

5. 실습 과제

- device_driver_module.c 와 test_app.c 를 수정하여 read 기능을 추가한다.
- Device driver operation set 에 아래 함수를 등록한다.
  - ssize_t test_device_driver_read(struct file *, char *, size_t, loff_t *);
- Device drive 의 read operation 이 수행하는 내용은 다음과 같다.
  - write 를 통해 받은 4 자리 수를 모두 더한 후 char (1 byte)로 변환하여 user 버퍼로 copy 한다. (copy_to_user 사용)
- 수정 된 test_app.c 에서 read 시스템 콜을 호출하여 커널 버퍼에서 계산한 값을 읽어 출력한다.
- [실행 예시]



- **제출물**
  - Test application 수행 시 출력 되는 내용을 캡처한 이미지 파일
  - 수정된 device_driver_module.c, test_app.c 파일
  - 위 파일들을 tar 로 압축하여 학번_이름.tar 로 제출 (ex: 120221234_홍길동.tar)