임베디드 시스템 실습 lab5

2016310936 우승민

goldfish driver 파일인 goldfish_iodev.c의 read, write 함수 코드입니다.

```
static ssize_t iodev_read (struct file *file, char __user *buf, size_t size, lof
f_t *loff) {
        int i;
        uint32_t lpn;

        if (*loff & (PAGE_SIZE - 1) || size != PAGE_SIZE )
                return -EINVAL;

        lpn = (uint32_t)(*loff >> PAGE_SHIFT);

        /*(iodev_read)
                1. Read status register (wait if busy)
                2. Write command register
                3. Read buf (check loff whether it aligned to 4K)
                4. Wait for completion (interrupt)
                5. copy_to_user()
         */

        while (readl(data->reg_base + STATUS_REG) != DEV_READY);

        spin_lock_irq(&wait_q_lock);
        condition = 0;

        writel(lpn,data->reg_base + LBA_REG);

        writel(READ_CMD, data->reg_base + CMD_REG);

        for(i=0; i < PAGE_SIZE / sizeof(uint32_t); i++){
                kbuf[i] = readl(data->reg_base + BUF_REG);

        }

        wait_event_lock_irq(wait_q, condition, wait_q_lock);    //5
        spin_unlock_irq(&wait_q_lock);

        copy_to_user(buf, (char*)kbuf, size);

        return size;
}

                                                        67,0-1        29%
```

1. while문을 통해 STATUS가 ready가 될 때까지 기다려줍니다.

2. irq에 lock을 걸어주고, lpn에 LBA_REG를 옮겨준 후 READ COMMAND를 write해줍니다.

3. for문을 사용하여 kbuf에 읽은 data를 옮겨줍니다.

4. interrupt를 발생시킨 후 irq lock을 풀어줍니다.

5. user에게 읽은 data를 전달합니다.

write 함수도 비슷하게 작동합니다.

```
static ssize_t iodev_write (struct file *file, const char __user *buf, size_t si
ze, loff_t *loff) {
        int i;
        uint32_t lpn;

        if (*loff & (PAGE_SIZE - 1) || size != PAGE_SIZE )
                return -EINVAL;

        lpn = (uint32_t)(*loff >> PAGE_SHIFT);
        copy_from_user((char*)kbuf, buf, size);

        /*(iodev_write)
                1. Read status register (until not DEV_READY)
                2. Write lpn to LBA_REG
                3. Write kbuf repeatedly (size of 4 bytes)f to BUF_REG
                4. Write WRITE_CMD to CMD_REG
                5. Wait for condition variable (condition)
                6. Increase offset
                7. Return size
         */
        while (readl(data->reg_base + STATUS_REG) != DEV_READY);        //1

        spin_lock_irq(&wait_q_lock);
        condition = 0;


        writel(lpn,data->reg_base + LBA_REG);

        //2

        for (i = 0; i < PAGE_SIZE / sizeof(uint32_t); i++ ) {
                //3
                writel(kbuf[i], data->reg_base + BUF_REG);
        }

        //4
        writel(WRITE_CMD, data->reg_base + CMD_REG);

        wait_event_lock_irq(wait_q, condition, wait_q_lock);    //5
        spin_unlock_irq(&wait_q_lock);

        (*loff) += size;        //6

        return size;    //7
}
                                                              149,0-1       51%
```

1. while문을 통해 STATUS가 ready가 될 때까지 기다려줍니다.

2. irq에 lock을 걸어주고, lpn에 LBA_REG를 옮겨줍니다.

3. for문을 사용하여 kbuf 옮겼던 data를 write해줍니다.

4. WRITE COMMAND를 CMD_REG에 옮겨줍니다.

5. interrupt를 발생시킨 후 irq lock을 풀어줍니다.

6. offset을 증가시키고 size를 반환합니다.

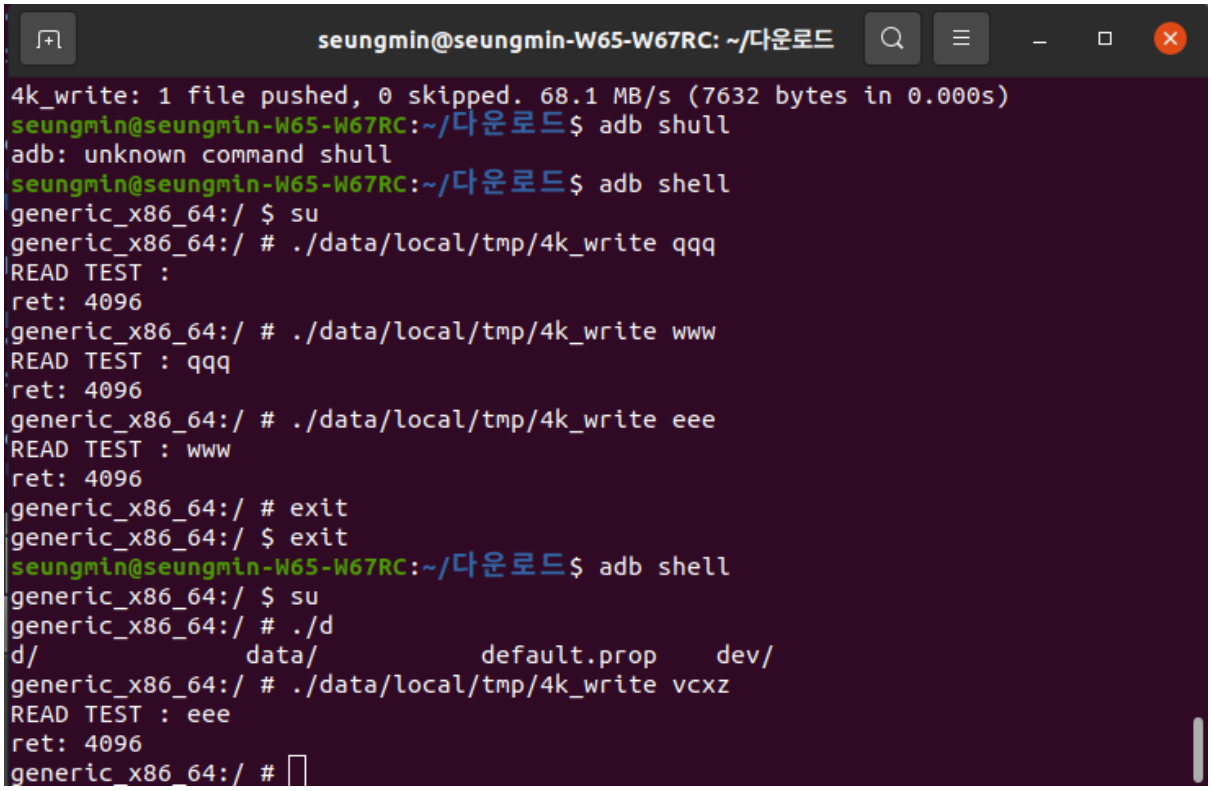
QEMU의 iodev.c 코드의 read 함수입니다. offset에서 BUF_REG일 경우를 추가하였습니다.

```
static uint32_t goldfish_iodev_read(void* opaque, hwaddr offset)
{
        struct goldfish_iodev_state* s = (struct goldfish_iodev_state*)opaque;
        uint32_t temp;

        if ( offset < 0 ) {
                cpu_abort(cpu_single_env, "iodev_dev_read: Bad offset %" HWADDR_
PRIx "\n", offset);
                return 0;
        }

        switch (offset) {
                case IODEV_STATUS_REG:
                        return s->status;
                case IODEV_BUF_REG:
                        temp = s->iodev_buf_pos;
                        s->iodev_buf_pos = (s->iodev_buf_pos+1) % 1024;
                        return s->iodev_buf[temp];
        };

        return 0;
}
                                                              104,11-32     48%
```

user code인 4k_write.c 코드입니다. 제 나름대로 확인을 하기 위해서 인자로 작성한 data를 write 하도록 수정하였습니다.

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/mman.h>

int main(int argc, char *argv[])
{
        int fd;
        int ret;
        char *buf = (char *)malloc(sizeof(char) * 4096);
        char *buf2 = (char *)malloc(sizeof(char) * 4096);
        int i;

        fd = open("/dev/iodev", O_RDWR | O_NDELAY);

        /*
          Access device by read and write
        */
        lseek(fd, 0, SEEK_SET);

        ret = read(fd, buf, 4096);
        printf("READ TEST : %s\n", buf);

        if(argc==2){
                strcpy(buf2, argv[1]);
        }

        i = 0;
        while(buf2[i]){
                buf[i]=buf2[i];
                i++;
        }
        buf[i]='\0';


        ret = write(fd, buf, 4096);

        printf("ret: %d\n", ret);

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

실행 사진입니다.



처음에는 disk가 비어져 있기 때문에 "READ TEST : " 의 값이 없고, 2번째부터는 이전에 인자로 주었던 data가 출력되는 것을 확인할 수 있습니다. 또한 종료하고 재 실행한 후에도 여전히 data가 남아있는 것을 확인할 수 있습니다.