1. 개요

리눅스 환경에서 모듈 프로그래밍을 활용하여 디바이스 드라이버 프로그램을

만들어보자

2. 프로그램 구조 설명

2.1 함수에 대한 설명

-mydrv.c ( 모듈 프로그램 )

void strupr2(char\* buf,size\_t count)

//유저 프로그래밍에서의 strupr의 기능을 모방한 함수입니다.

//char 포인터를 인자로 받아서 그 포인터 주소부터 count 수 만큼의 문자들을

//대문자는 소문자로, 소문자는 대문자로 만들어줍니다.

static ssize\_t mydrv\_write(struct file\* file,const char\* buf,size\_t count,loff\_t\* ppos)

//유저 프로그램에서 write를 실행하면 실행되는 함수입니다.

//유저 프로그램에서 받아온 데이터를 하드웨어에 저장합니다.

static ssize\_t mydrv\_read(struct file\* file, char\* buf, size\_t count, loff\_t\* ppos)

//유저 프로그램에서 read를 실행하면 실행되는 함수입니다.

//하드웨어에서 읽은 값을 유저프로그램에 전달합니다.

int mydrv\_init(void)

//모듈 프로그램이 insmod에 의해 커널에 올라갔을 때 처음 실행되는 함수입니다.

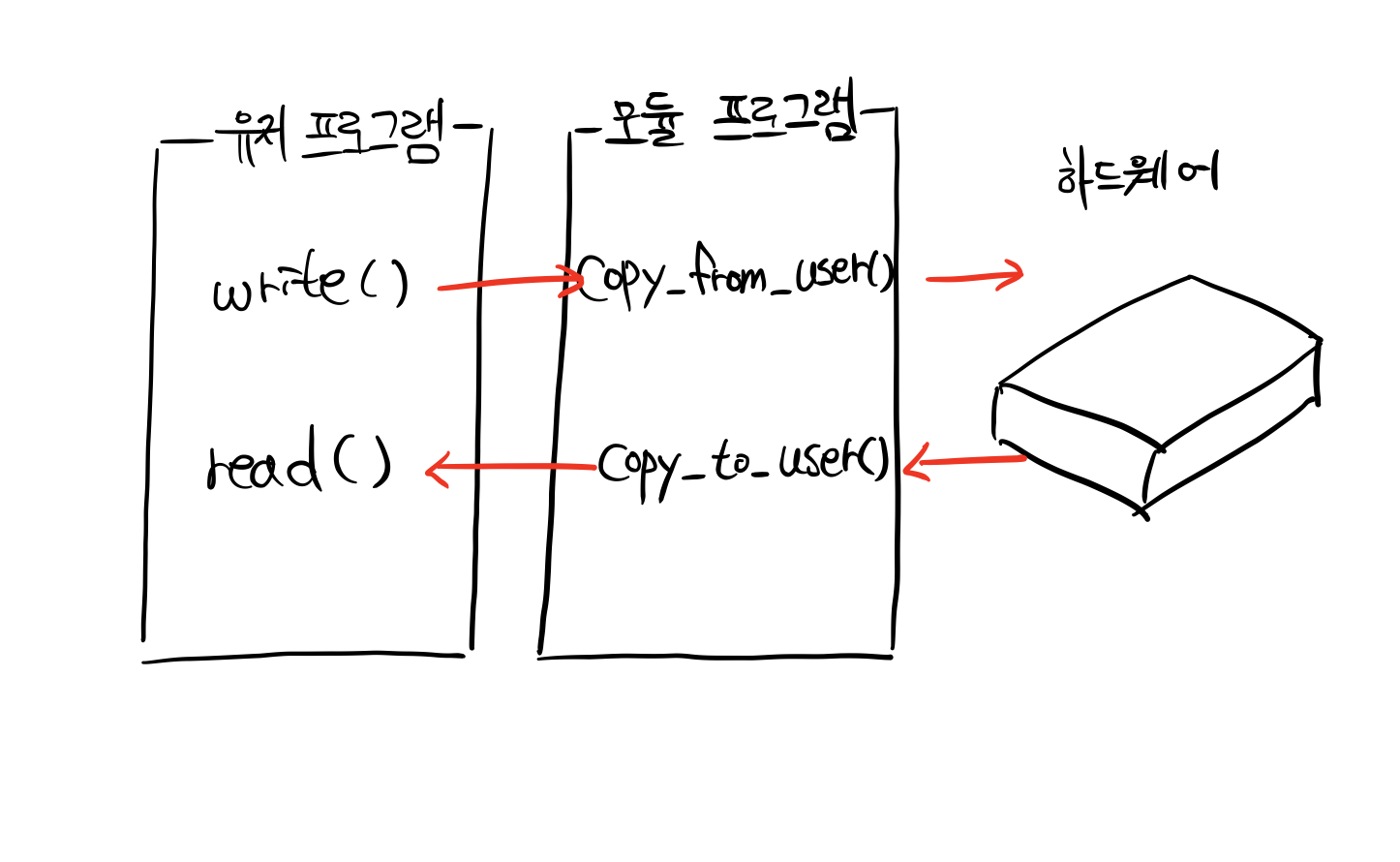
//모듈 프로그램의 초기 설정을 담당합니다.

void mydrv\_cleanup(void)

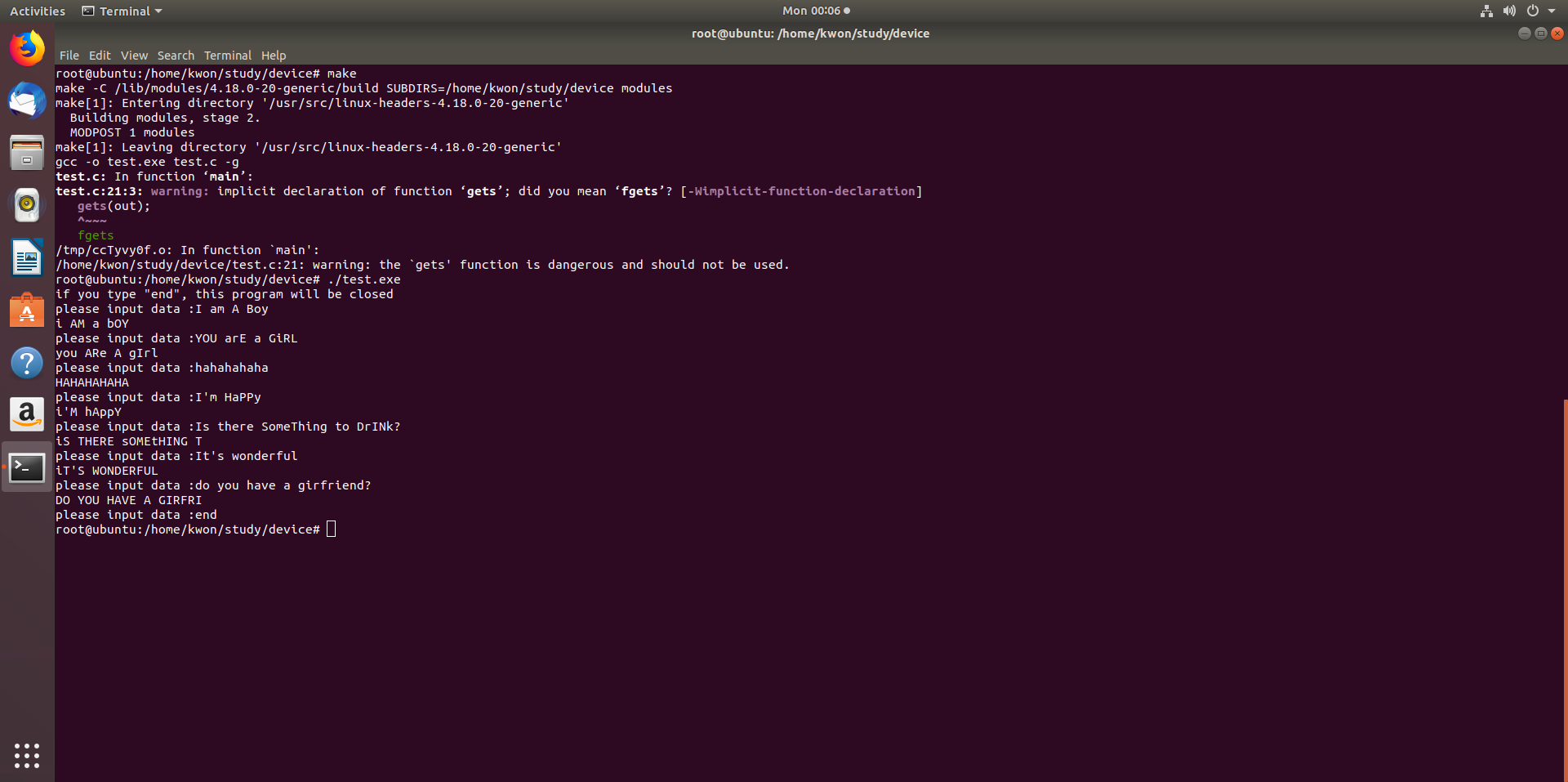
//모듈 프로그램이 rmmod에 의해 커널에서 내려왔을 때 실행되는 함수입니다.

//모듈 프로그램에 의해 할당된 메모리를 free 시켜주는 역할을 합니다.

2.다이어그램



3. 실행 결과

****

메모리 드라이버의 총 메모리 크기는 100byte입니다.

한번 입력을 받을 때 20byte씩 입력을 받습니다.

하지만 메모리 크기가 100byte임에도 5번 입력 후에 계속 메모리에

쓰는 것이 가능한 것을 볼 수 있습니다.

메모리 최대 크기를 넘어가면 다시 처음부터 overwrite하게 코딩했기 때문에

계속 입력이 가능합니다.

**(실행결과는 자신의 컴퓨터환경에서 실행한 결과 캡쳐(그림파일)해서 문서에 첨부하여 제출합니다. 실행결과가 성공한 경우뿐만 아니라 실패한 경우도 포함되어야 함.**

**만약 과제로 올려서 실행한 결과와 실행 결과로 캡쳐한 결과가 다를 경우 감점합니다.)**

4. 고찰 (제일 중요함 - **점수배점이 가장 높음!** 자세히 쓸 것)

모듈 프로그래밍이 제게 생소했기 때문에 이번 과제에서 저는 굉장히 헤맸습니다.

특히 저는 유저 프로그램으로부터 받아온 인자를 모듈 프로그램에서 바로

다룰 수 있을 것이라 생각했던 것이 가장 큰 착오였습니다. Copy\_from\_user(); 함수가

있다는 것은 알고 있었지만 저는 그 함수가 메모리에 저장을 하는 함수로 오해를 하고 있었

기 때문에, 유저프로그램에서 받아온 인자를 바로 대소문자 변환을 하고 Copy\_from\_user()로 넣

어야 된다는 생각을 가지고 있었습니다. 하지만 stackoverflow라는 외국 프로그래밍 정보 공유 사 이트에서, 커널 프로그래밍의 큰 목적 중 하나가 메모리 보호이고, 커널의 메모리 region과

Userspace의 메모리 region은 다르기 때문에 kernel에서 Userspace의 메모리에 접근을 할 수

없다는 사실을 알게되었고, 그때서야 Copy\_from\_user() 함수의 진정한 기능을 알게되었습니다.

커널 프로그래밍에 대해 자세히 공부를 하지 않고 바로 코딩에 덤벼든 저의 불찰이었습니다.

이번 과제를 하면서 다음번에는 이론부터 충실히 해야겠다는 생각을 가지게 되었습니다.

5. 프로그램 소스 파일 (주석을 꼭 달 것)

-mydrv.c (모듈 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 <linux/uaccess.h>

#include <linux/string.h>

#define DEVICE\_NAME "mydrv"

#define MYDRV\_MAX\_LENGTH 100 // 100byte

#define DATA\_SIZE 20

struct class\* myclass;

struct cdev\* mycdev;

struct device\* mydevice;

dev\_t mydev;

static char\* mydrv\_data;

static int mydrv\_read\_offset,mydrv\_write\_offset;

void strupr2(char\* buf,size\_t count) { // 포인터가 가르키는 주소부터 count만큼의 문자를

// 대소문자 변환합니다.

int i = 0;

while (i < count) {

if (buf[i] >= 'A' && buf[i] <= 'Z') {

buf[i] = buf[i] + 32;

}else if(buf[i] >= 'a' && buf[i] <= 'z'){

buf[i] = buf[i] - 32;

}

i++;

}

return;

}

static ssize\_t mydrv\_write(struct file\* file,const char\* buf,size\_t count,loff\_t\* ppos){

// 유저 프로그램에서 받아온 데이터를 메모리에 write 하는 함수입니다.

if(mydrv\_write\_offset >= MYDRV\_MAX\_LENGTH){

mydrv\_write\_offset = 0;

}

if(copy\_from\_user(mydrv\_data+mydrv\_write\_offset,buf,count)){

return -1;

}

strupr2(mydrv\_data+mydrv\_write\_offset,count);

mydrv\_write\_offset += count;

return count;

}

static ssize\_t mydrv\_read(struct file\* file, char\* buf, size\_t count, loff\_t\* ppos){

//메모리에서 읽어온 데이터를 유저프로그램에 전달하는 역할을 합니다.

if(mydrv\_read\_offset >= MYDRV\_MAX\_LENGTH){

mydrv\_read\_offset = 0;

}

if(copy\_to\_user(buf,mydrv\_data+mydrv\_read\_offset,count)){

return -1;

}

mydrv\_read\_offset += count;

return count;

}

struct file\_operations mydrv\_fops={

.owner = THIS\_MODULE,

.read = mydrv\_read,

.write = mydrv\_write,

// .open = mydrv\_open,

// .release = mydrv\_release,

};

int mydrv\_init(void){

//모듈프로그램의 초기화 함수입니다.

printk("Hello~~~\n");

if( alloc\_chrdev\_region(&mydev,0,1,DEVICE\_NAME)<0){

return -EBUSY;

}

myclass = class\_create(THIS\_MODULE,"mycharclass");

if(IS\_ERR(myclass)){

unregister\_chrdev\_region(mydev,1);

return PTR\_ERR(myclass);

}

mydevice = device\_create(myclass,NULL,mydev,NULL,"mydevicefile");

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;

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);

mydrv\_read\_offset = mydrv\_write\_offset = 0;

return 0;

}

void mydrv\_cleanup(void){

//모듈 프로그램의 메모리 할당 해제 함수입니다.

kfree(mydrv\_data);

cdev\_del(mycdev);

device\_destroy(myclass,mydev);

class\_destroy(myclass);

unregister\_chrdev\_region(mydev,1);

printk("Goodbye\n");

}

module\_init(mydrv\_init);

module\_exit(mydrv\_cleanup);

MODULE\_LICENSE("GPL");

}

-test.c (사용자 프로그램)

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <string.h>

#define MAX\_SIZE 20

int main(void){

int fd;

char\* in = (char\*)malloc(sizeof(char)\*MAX\_SIZE);

char\* out = (char\*)malloc(sizeof(char)\*MAX\_SIZE);

if( (fd=open("/dev/mydevicefile",O\_RDWR))<0){ //디바이스를 open합니다.

perror("open error");

return -1;

}

printf("if you type \"end\", this program will be closed\n");

while(printf("please input data :")){ //반복적으로 입력을 받아서 메모리에 저장합니다.

gets(out);

if(!strcmp(out,"end")){

break;

}

write(fd,out,MAX\_SIZE);

read(fd,in,MAX\_SIZE);

printf("%s\n",in);

}

//메모리 할당 해제

free(in);

free(out);

close(fd);

return 0;

}

6. 자료 출처 (어떤 책을 참고했으며 어떤 웹사이트를 참고했는지 출처를 적습니다)

<https://stackoverflow.com/questions/45978645/crash-when-trying-to-access-const-char-buffer>

<https://wiki.kldp.org/KoreanDoc/html/EmbeddedKernel-KLDP/device-understanding.html>

등등을 참고했습니다.