Adding a New Dev file

–김성영 , 이재혁 , 김남현

– 발표자 : 김남현

목차

Progress

- **4월 1일**
  - 프로젝트 방향 설정
- **4월 8일**
  - device file 추가 방법 조사
  - mem.c 파일 분석
- **4월 10일**
  - 알고리즘 제시
  - 필요한 함수 분석
- **4월 16일**
  - 발표자료 조사 및 만들기

진행 중 어려운 점

- Google 예제의 부족
- 디바이스 드라이버를 built-in 형태로 구현
- Linux 버전에 따른 코드의 수정
  - 2.x 버젼의 nopage가 없어지고 3.x 버젼의 fault로 통합
- Unmap의 방법

해야 하는 일

- Fault handling
  - PTE에서 physical page에 접근하는 것을 막는 방법
  - Fault가 일어나는 이유를 구분

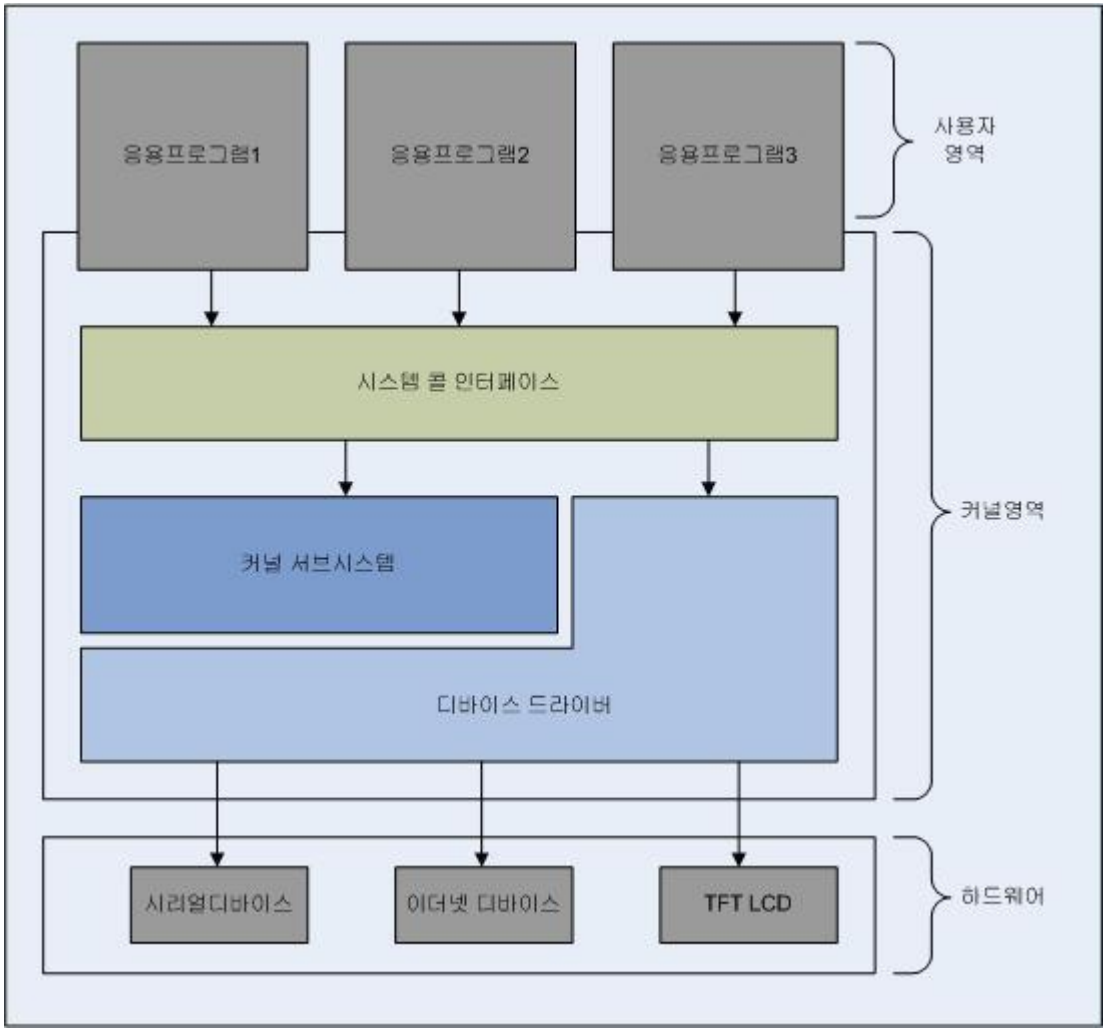
Device File System

디바이스 드라이버

- 시스템이 지원하는 하드웨어를 응용 프로그램에서 사용할 수 있도록 커널에서 제공하는 라이브러리
  - 하드웨어는 디바이스 드라이버를 통해 접근

- 응용 프로그램이 하드웨어를 제어하려면 커널에 자원을 요청하고, 커널은 이런 요청에 따라 시스템을 관리

디바이스 드라이버


응용프로그램1
응용프로그램2
응용프로그램3
사용자
영역
시스템 콜 인터페이스
커널 서브시스템
디바이스 드라이버
커널영역
시리얼디바이스
이더넷 디바이스
TFT LCD
하드웨어

디바이스 파일

- 유닉스의 철학 : “Everything is a file”
- 리눅스는 시스템에 있는 모든 자원을 파일 형식으로 표현
  - 램, 키보드, 보조기억장치인 하드디스크도 파일로 표현
  - 일반 파일의 목적이 데이터를 저장하는데 있다면 이들은 하드웨어 정보를 제공하는데 목적이 있다.

- /dev/ 디렉터리에 존재하며, 이 파일 하나 하나는 실질적인 하드웨어를 표현

디바이스 드라이버의 종류

- 문자 디바이스 드라이버
  - 임의의 길이를 갖는 문자열이나 자료의 순차성을 지닌 장치를 다루는 디바이스 드라이버
  - 버퍼 캐쉬를 사용하지 않는다.
  - Ex) Serial, Console, Keyboard, Printer, Mouse ….
- 블록 디바이스 드라이버
  - 일정 크기의 버퍼를 통해 데이터를 처리하는 디바이스 드라이버
  - Ex) Hard disk, RAM ….
- 네트워크 디바이스 드라이버
  - 네트워크 계층과 연결되어 네트워크 통신을 통해 패킷을 송수신 할 수 있는 기능을 제공한다.
  - Ex) 이더넷, PPP ….

디바이스 파일 – 주번호와 부번호

- 주번호
  - 커널에서 디바이스 드라이버를 구분하고 연결하는데 사용
  - 주번호는 제어하려는 디바이스를 구분하기 위한 디바이스의 ID
- 부번호
  - 디바이스 드라이버 내에서 장치를 구분하기 위해 사용
  - 같은 종류의 디바이스가 여러 개 있을 때 그 중 하나를 선택하기 위해 사용

디바이스 파일 – 주번호와 부번호

- 시리얼 디바이스 파일을 살펴보면 아래와 같다.

```
root@ubunt:~$ ls -al /dev/ttyS*
crw-rw----   1   root   dialout   4,   64   Apr   17   02:31   /dev/ttyS0
crw-rw----   1   root   dialout   4,   65   Apr   17   02:31   /dev/ttyS1
crw-rw----   1   root   dialout   4,   74   Apr   17   02:31   /dev/ttyS10
crw-rw----   1   root   dialout   4,   75   Apr   17   02:31   /dev/ttyS11
crw-rw----   1   root   dialout   4,   76   Apr   17   02:31   /dev/ttyS12
……..
```

  - 시리얼 COM1 포트를 나타내는 /dev/ttyS0와 COM2 포트를 나타내는 /dev/ttyS1의 주 번호는 ‘4’로 같다. 즉 시리얼 포트라고 하는 같은 종류의 디바이스 파일이다. 하지만 이들을 각각을 구분해 부 번호는 다르다.

Create Dev File

디바이스 파일 생성 – mknod (쉘에서)

- 디바이스 파일은 일반 파일과 달리 create() 함수를 사용하지 않고, mknod 유틸리티에 의해서 생성
- 디바이스 파일은 주로 /dev/ 디렉터리에 생성
- mknod를 이용하여 디바이스 파일을 만드는 방법은 아래와 같다

```
root@ubuntu:# mknod /dev/ttyS4 c 4 68
```

  - **디바이스 파일명은 ttyS4이고, 문자 디바이스 드라이버이고, 주번호가 4, 부번호가 68인 디바이스 파일을 생성한다.**

디바이스 파일 생성 – mknod (소스코드에서)

- 응용 프로그램에서 디바이스 파일을 만들어야 할 경우

```
int mknod(const char *pathnmae, mode_t mode, dev_t dev)
```

  - Const char *pathname
    - 작성할 디바이스 파일명
  - mode_t mode
    - 접근 허가 및 디바이스 타입(OR 연산으로 설정)
  - dev_t dev
    - 디바이스 주번호 및 부번호 (시프트 연산으로 같이 입력)

디바이스 등록 – register_chrdev

- 커널 내부에 등록된 device driver를 관리하는 chrdev[] 배열 구조체에 하나의 배열을 할당 받는다.
- 배열안의 필드에 파일의 이름과 파일 오퍼레이션을 연결한다.

```
int register_chrdev(unsigned int major, const char *name,
                    struct file_operations *fops)
```

  - unsigned int major
    - 할당하고자 하는 주 번호
  - const char *name
    - 디바이스 파일의 이름
  - strtuct file_oprations *fops
    - 드라이버의 엔트리 포인트를 실행하는데 사용되는 함수 구조체 포인터

2가지 형태의 구현 방법

- External Module
  - 사용자에 의해 modprobe 혹은 insmod의 명령어로 동적으로 모듈에 적재된다.

- Built-in
  - 부팅되는 과정에서 커널에 적재된다.

/kernel/include/linux/init.h

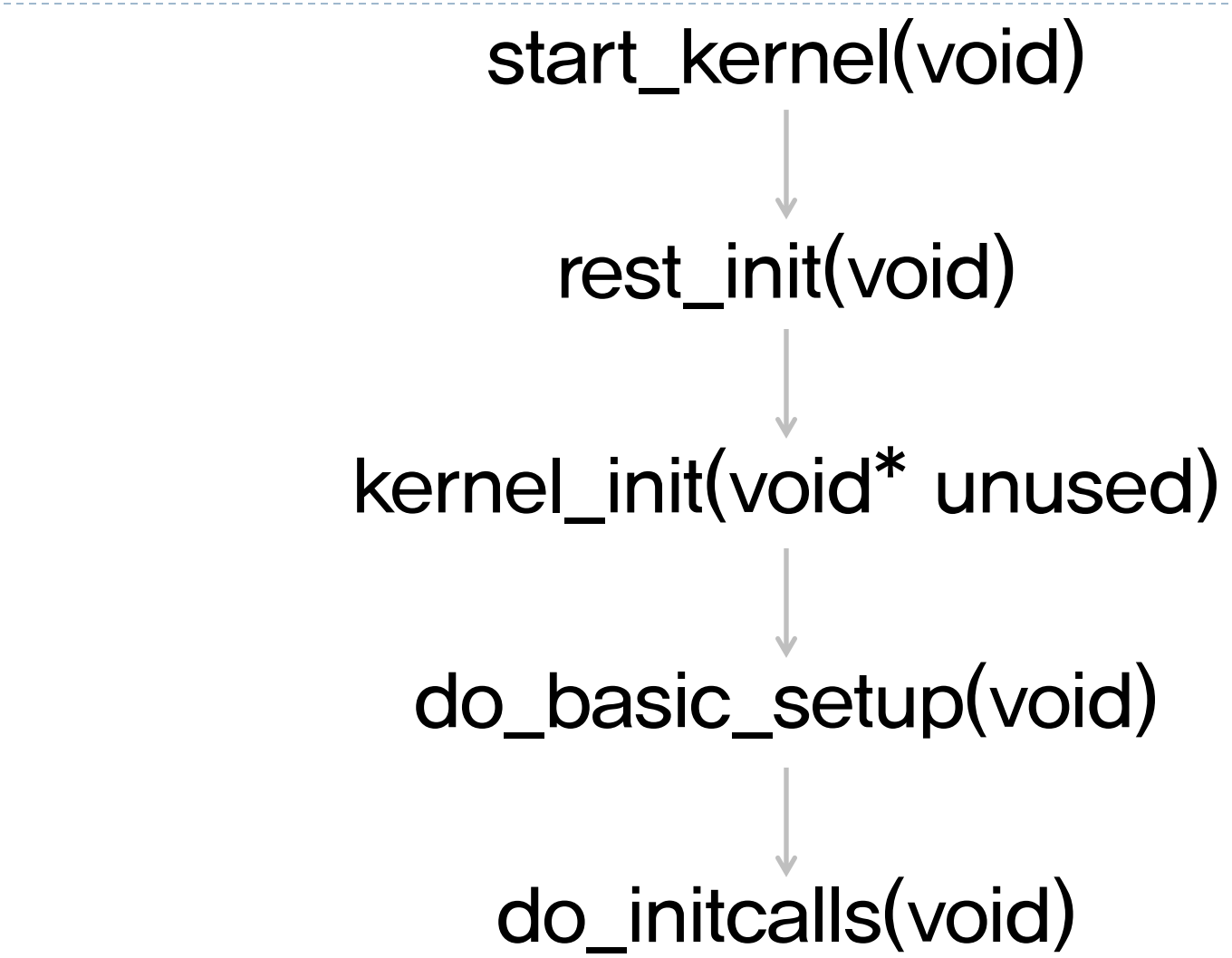
External Module방식

```
/* Each module must use one module_init(). */
#define module_init(initfn)                                    ₩
        static inline initcall_t __inittest(void)              ₩
        { return initfn; }                                     ₩
        int init_module(void) __attribute__((alias(#initfn)));
/* This is only required if you want to be unloadable. */
#define module_exit(exitfn)                                    ₩
        static inline exitcall_t __exittest(void)              ₩
        { return exitfn; }                                     ₩
        void cleanup_module(void) __attribute__((alias(#exitfn)));
```

Built-in 방식

```
#define device_initcall(fn)             __define_initcall(fn,6)

......

#define __initcall(fn) device_initcall(fn)

......

/**
 * module_init() - driver initialization entry point
 * @x: function to be run at kernel boot time or module insertion
 *
 * module_init() will either be called during do_initcalls() (if
 * builtin) or at module insertion time (if a module).  There can only
 * be one per module.
 */
#define module_init(x)  __initcall(x);
```

```
#define __define_initcall(level,fn,id) ₩
        static initcall_t __initcall_##fn##id __used ₩
        __attribute__((__section__(".initcall" level ".init"))) = fn
.....


#define device_initcall(fn)             __define_initcall(fn,6)
```


start_kernel(void)
rest_init(void)
kernel_init(void* unused)
do_basic_setup(void)
do_initcalls(void)

do_initcalls(void)

```
extern initcall_t __initcall_start[], __initcall_end[], __early_initcall_end[];
static void __init do_initcalls(void)
{
        initcall_t *call;
        for (call = __early_initcall_end; call < __initcall_end; call++)
                do_one_initcall(*call);
        /* Make sure there is no pending stuff from the initcall sequence */
        flush_scheduled_work();
}
```

do_one_initcalls(initcall_f fn)

```
int do_one_initcall(initcall_t fn)
{
        int count = preempt_count();
        ktime_t t0, t1, delta;
        char msgbuf[64];
        int result;
        if (initcall_debug) {
                printk("calling  %pF₩n", fn);
                t0 = ktime_get();
        }
        result = fn();

        ......
}
```

Built-in Device Driver

소스 코드 작성

↓

Kconfig 수정

↓

Make file 수정

↓

make menuconfig 수정

예제

- /drivers/char
  - my_driver.c 파일 생성

```
static int my_dev_init(void)
{
        int res;
        printk("initializer has been called₩n");
        res = register_chrdev(MY_DEV_MAJOR,MY_DEV_NAME,&mem_fops);
        if(res<0) return res;

        return 0;
}

module_init (my_dev_init);
```

- /driver/char
  - Kconfig 수정

```
menu "Character devices  “

source "drivers/tty/Kconfig  “

config MYDEVDRIVER
        bool "my_dev_driver"
        default y

.....
```

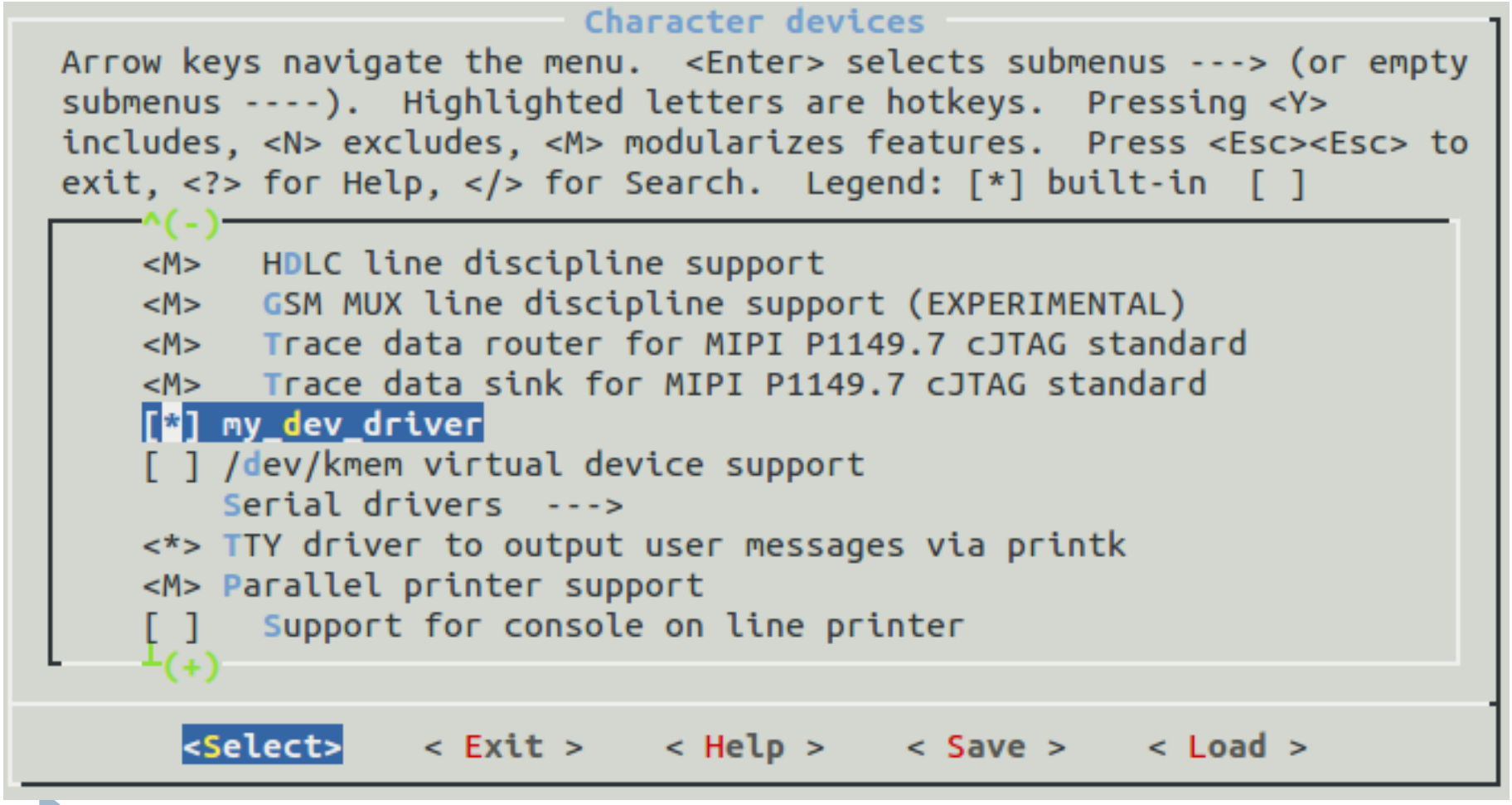
  - “y” 는 선택되었다는 것을 의미한다.
  - Make menuconfig 입력시, 메뉴 화면은 Kconfig의 내용을 파싱한 것임.

- /driver/char
  - Makefile 수정

```
obj-$(CONFIG_MYDEVDRIVER)        += my_driver.o
obj-y                            += mem.o random.o
obj-$(CONFIG_TTY_PRINTK)         += ttyprintk.o

.....
```

Make menuconfig

- Device Drivers –〉 Character devices



결과

```
[    32.221437] 00:07: ttyS1 at I/O 0x2f8 (irq = 3, base_baud = 115200) is a 16550
A
[    32.226929] initializer has been called
[    32.227600] Linux agpgart interface v0.103
[    32.227784] agpgart-intel 0000:00:00.0: Intel 440BX Chipset
```

Example

Google Example Code

```
#include <linux/init.h>

……

static char ker_buf[100];    //driver local  buffer
static int dev_open(struct inode *inod, struct file *fil);
static ssize_t dev_read(struct file *filep,char *buf,size_t len,loff_t *off);
static ssize_t dev_write(struct file *flip,const char *buff,size_t len,loff_t *off);
static int dev_release(struct inode *inod,struct file *fil);

//structure containing device operation

static struct file_operations fops=   {
    .read=dev_read,
    .write=dev_write,
    .open=dev_open,
    .release=dev_release,

};
```

Google Example Code

```
static int dev_open(struct inode *inod, struct file *fil)   {
    printk("KERN_ALERT device opened");
    return 0;
  }

static ssize_t dev_read(struct file *filep,char *buf,size_t len,loff_t *off)   {
    copy_to_user(buf,ker_buf,len);
  return len;
}

static ssize_t dev_write(struct file *flip,const char *buf,size_t len,loff_t *off)   {
    copy_from_user(ker_buf,buf,len);
    ker_buf[len]=0;
    return len;
}
```

Google Example Code

```
static int dev_release(struct inode *inod,struct file *fil)   {
    printk("KERN_ALERT device closed₩n");
  return 0;
}

static int hello_init(void)   {
    int t=register_chrdev(90,"mydev",&fops);
  if(t<0)    printk(KERN_ALERT "device registration failed.");
  else    printk(KERN_ALERT "device registred₩n");
  return 0;
}

static void hello_exit(void)   {
    unregister_chrdev(90,"mydev");
    printk(KERN_ALERT "exit");
}
module_init(hello_init);
module_exit(hello_exit);
```

감사합니다

– 김성영 , 이재혁 , 김남현