Embedded System Software [CSE4116]

실습 4주차(1) : Module Programming

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

Data-Intensive Computing and System Laboratory

1. Module Device Driver

1.1. Module

- 리눅스 시스템이 부팅 된 후에 동적으로 load, unload 할 수 있는 커널의 구성 요소
  (커널을 다시 컴파일하거나 시스템을 재부팅 하지 않고도 커널의 일부분을 수정 가능)
- 모듈의 버전은 현재 실행되고 있는 커널 버전과 같아야 함
  (모듈의 버전 정보는 리눅스 커널소스/include/linux/module.h 에 정의 되어 있음)
- main() 함수가 없고, 커널에 loading 및 unloading 할 때 호출되는 int init_module(void) 함수와 void cleanup_module() 함수의 선언이 존재함

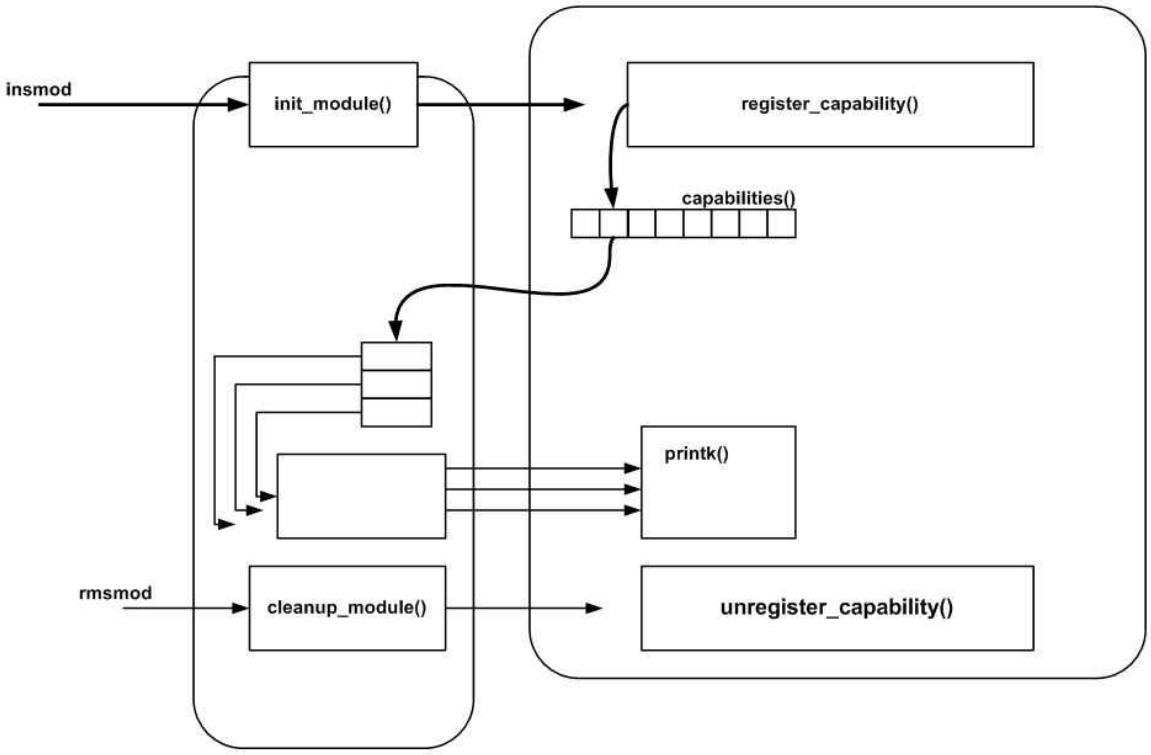


Figure 1 Module Linking Flow

1.2. Module Function & Macro

- MODULE_LICENSE() : 모듈에 대한 라이센스를 명시하는 커널 매크로
- MODULE_AUTHOR() : 모듈의 저작자를 명시하는 커널 매크로
- module_init() : 모듈이 시작 할때 실행되는 함수
- module_exit() : 모듈이 종료 될때 실행되는 함수

1.3. Module Command in Linux

명령어	용도
insmod	module 을 설치(install)
rmmod	실행중인 modules 을 제거(unload)
lsmod	Load 된 module 들의 정보를 표시
depmod	커널 내부에 적재된 모듈간의 의존성을 검사한다.
modprobe	모듈간 의존성을 검사하여 그 결과 누락된 다른 모듈을 찾아서 적재한 다.
modinfo	목적 파일을 검사해서 관련된 정보를 표시

- 실행 예시

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

2. Remind

2.1. printk 출력(보드-minicom 에서)

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

2.2. Host to Device 파일 전송

```
$ adb push [FILENAME] /data/local/tmp
```

3. 실습

3.1. hello_module

- 제공된 hello_module 디렉토리에서 make 수행
  (hello_module.c, Makefile 간단히 읽어볼 것)
- 컴파일 된 모듈 hello_module.ko 를 보드로 전송
- 모듈 insmod, rmmod 시 init, exit function 이 수행됨을 알 수 있음

```
root@achroimx:/data/local/tmp # insmod hello_module.ko
[ 1531.852517] start module
root@achroimx:/data/local/tmp # rmmod hello_module.ko
[ 1539.010928] end module
root@achroimx:/data/local/tmp #
```

3.2. communication_module

- 제공된 comm_module 디렉토리에서 make 수행
  (module1.c, module2.c 를 모듈간 연동 관계를 중점으로 이해할 것)
- 컴파일된 모듈 module1.ko, module2.ko 를 보드로 전송
- module1 에서 EXPORT_SYMBOL 을 통해 등록한 함수를 module2 에서 호출할 수 있음

4. 실습 과제 : Module Programing

- **module1.c 파일에 mynew_function 을 추가 및 심볼 등록**
  - mynew_function 은 4 자리 수를 Argument 로 입력받아 각 자리 수를 계산

- **module2 insmod 시 해당 함수에 본인 학번 뒷 4 자리 수를 넘겨 계산 된 값을 출력**
  - 과제 수행 시 모듈간의 dependency 를 고려하기(insmod, rmmod 순서)
  - [예시]
    입력 : 1234
    출력 : Thousands = 1 / hundreds = 2 / tens = 3 / units = 4

- **제출물**
  - module1 insmod 후 module 2 insmod 시 출력 되는 내용을 캡처한 이미지 파일
  - module1.c, module2.c, Makefile
  - 위 파일들을 tar 로 압축하여 학번_이름.tar 로 제출 (ex: 120221234_홍길동.tar)
    (이름은 영어로 해도 상관 없음)

5. 참고자료 (Device Driver I/O functions)

5.1. Function between kernel space and user space

A. copy_from_user()

기능	사용자 메모리 블록 데이터를 커널 메모리에 복사
형태	unsigned long copy_from_user (void *to, cont void _user* from, unsigned long n)
매개변수	to : 커널 메모리 시작 주소 (Destination) from : 사용자 메모리 시작 주소 (Source) n : 복사할 바이트 크기
반환값	성공 시 0 반환, 실패 시 복사하지 못한 바이트 크기 반환

B. copy_to_user()

기능	커널 메모리 블록 데이터를 사용자 메모리에 복사
형태	unsigned long copy_to_user (void *to, cont void _user* from, unsigned long n)
매개변수	to : 사용자 메모리 시작 주소(Destination) from : 커널 메모리 시작 주소 (Source)₩ n : 복사할 바이트 크기
반환값	성공 시 0 반환, 실패 시 복사하지 못한 바이트 크기 반환

5.2. Port mapped I/O vs. Memory mapped I/O

A. Port mapped I/O

- I/O 명령을 위한 instruction 이 따로 존재한다. CPU 의 extra I/O pin 또는 I/O 전용으로 할당된 별도의 bus 를 이용하여 I/O 명령을 처리한다.
- I/O 에 접근하기 위한 address space 가 memory 접근을 위한 address space 와는 별도로 존재한다.
- 주로 Intel 계열의 processor 에서 사용하는 방식이다.

B. Memory mapped I/O

- memory 에 접근하기 위한 address space 와 I/O 에 접근하기 위한 address space 가 같다.
- memory 의 특정 영역이 I/O 를 위한 주소로 미리 예약되어 있다.
- 주로 ARM 계열의 processor 에서 사용하는 방식이다.

5.3. I/O function type

- **Memory mapping function**

  특정 물리 주소 공간을 커널 주소 공간으로 mapping 하는 함수 (asm-generic/io.h 에 선언됨)
- **I/O Resource Management function (경쟁 처리 함수)**

  동일한 I/O 영역(port or memory)을 다른 목적으로 사용하는 driver 들이 동시에 경쟁할 경우, 충돌이 생기지 않도록 영역의 사용 권한을 허가 또는 제한하는 함수 (linux/ioport.h 에 선언됨)
- **I/O function**

  다른 architecture 와의 호환성을 위하여 일반적인 pointer 연산을 사용하지 않고, I/O 를 위하여 정의된 macro 함수들을 사용한다.

5.4. Port mapped I/O function

A. I/O Resource Management function(경쟁 처리 함수)

- **int check_region(unsigned long start, unsigned long len)**
  check_region : 등록할 수 있는 I/O port 영역인지 확인한다, 가능하지 않을 경우, 0 보다 작은 값 - EBUSY 를 반환한다.
- **struct resource * request_region(unsigned long start, unsigned long len, char * name)**
  I/O port 영역을 등록한다.
- **void release_region(unsigned long start, unsigned long len)**
  등록된 I/O port 영역을 해제한다.

B. I/O 처리 함수

- unsigned char inb(unsigned short port);
- unsigned short inw(unsigned short port);
- unsigned long inl(unsigned short port);
- void outb(unsigned char data, unsigned short port);
- void outw(unsigned short data, unsigned short port);
- void outl(unsigned long data, unsigned short port);
- void insb(unsigned short port, void *addr, unsigned long count);
- void outsb(unsigned short port, void *addr, unsigned long count);

5.5. Memory mapped I/O function

A. Memory mapping function

- **void *ioremap(unsigned long addr, unsigned long size)**
  물리 주소 공간을 커널 주소 공간으로 mapping 한다.
- **void *iounmap(unsigned long addr, unsigned long size)**
  mapping 한 커널 주소 공간을 해제한다.

B. I/O Resource Management function (경쟁 처리 함수)

- **int check_mem_region(unsigned long start, unsigned long len)**
  check_region : 등록할 수 있는 I/O memory 영역인지 확인한다, 가능하지 않을 경우, 0 보다 작은 값 - EBUSY 를 반환한다.
- **struct resource * request_mem_region(unsigned long start, unsigned long len, char * name)**
  I/O memory 영역을 등록한다.
- **void release_mem_region(unsigned long start, unsigned long len)**
  등록된 I/O memory 영역을 해제한다.

C. I/O 처리 함수

- unsigned char readb(unsigned int addr);
- unsigned short readw(unsigned int addr);
- unsigned long readl(unsigned int addr);
- void writeb(unsigned char data, unsigned short addr);
- void writew(unsigned short data, unsigned short addr);
- void writel(unsigned long data, unsigned short addr);
- void readsb(unsigned short addr, void *addr, unsigned long count);
- void writesb(unsigned short addr, void *addr, unsigned long count);

5.6. FND Device Driver

- **Registering / Unregistering a Device**

  #include <linux/fs.h>

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

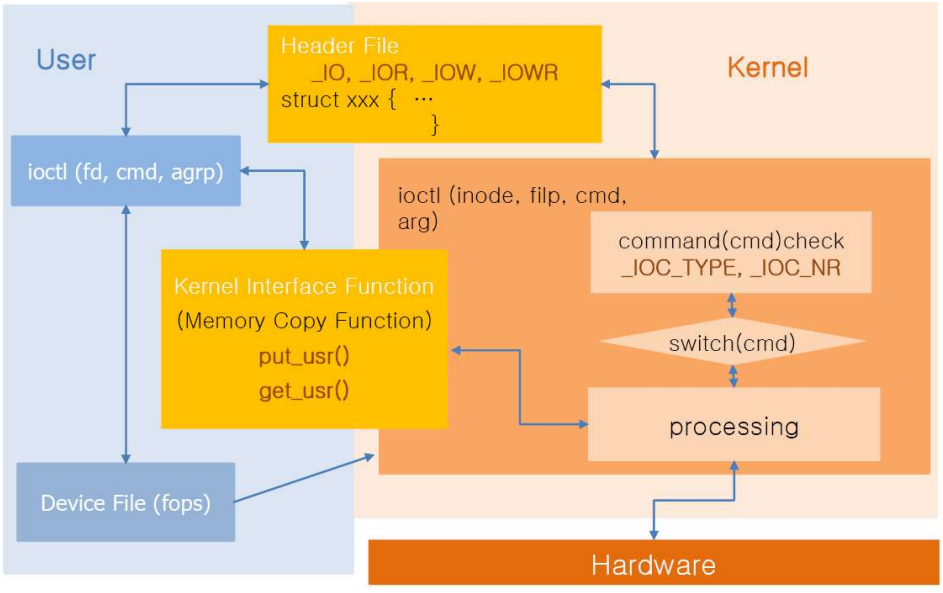
  unregister_chrdev(unsigned int major, const char *name);

- **Assigning File Operations**

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

5.7. ioctl

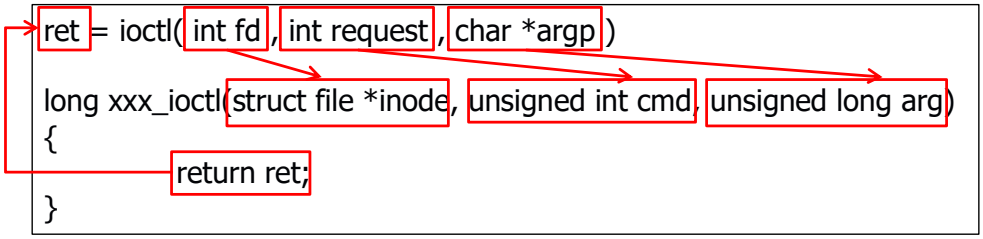
- Flow



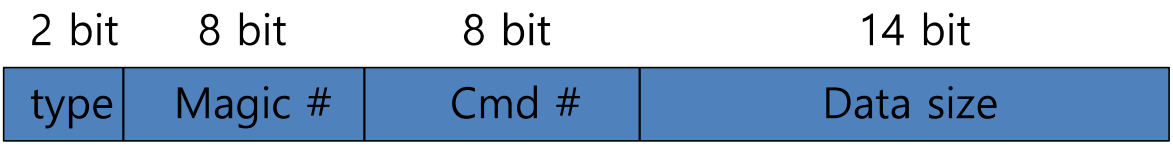
- 사용방법 : Device Driver 내 file_operations 에 사용 함수 선언

```
//file_operation structure
struct file_operations driver_fops=
{
    owner:   THIS_MODULE,
    open:    new_driver_open,
    unlocked_ioctl: new_driver_ioctl,
    release:     new_driver_release,
};
```

- iotcl() fuction



- Command Structures



  - Example

    User 가 ioctl(fd , 2151694592,….) 를 호출했을 때, 그 command 의 의미는

    2151694592 = 10 / 00000001 / 00000001 / 00000100000000 (2 진수)

    Type = 2, magic # = 1, cmd # = 1, data size = 256

    → Command 를 만들기 위한 macro 가 있음

- Macro to encode / decode a command (asm/ioctl.h 에 선언)
  - Encoding

    _IO(type, number)

    _IOR(type, number, datatype)

    _IOW(type, number, datatype)

    _IOWR(type, number, datatype)
  - Decoding

    _IOC_TYPE(command)

    _IOC_NR(command)

    _IOC_DIR(command)

    _IOC_SIZE(command)