Programming for Kernel Module

Suntae Hwang
Kookmin University

Kernel Module

- **Piece of code that can be added to the kernel at runtime**
  - **Can minimize the kernel image**

- **Usages**
  - **Device drivers**
    - Because it is not possible to permanently compile drivers for all possible devices into the base kernel
  - **Extensions of the kernel**

모듈 프로그래밍의 필요성

- **monolithic Kernel**
  - 모든 기능이 커널 내부에 구현
  - 사소한 커널 변경에도 커널 컴파일과 리부팅 필요
  - 커널의 크기가 너무 큼
  - 일부 사용되지 않는 기능이 메모리 상주
- **micro Kernel**
  - 높은 확장성과 동적 재구성이 용이
  - VxWorks, pSOS, VRTX, QNX, SROS

- **리눅스 커널은 모노리딕 커널, 마이크로 커널 방식이 아님**

모듈 프로그래밍의 역할

- 모듈 프로그래밍으로 모노리딕의 단점을 해결
  - 커널의 일부 기능을 빼고 모듈로 구현
  - 필요할 때만 적재시키므로 효과적인 메모리 사용

- 커널 영역에서 프로그래밍 방법 제공

Kernel Modules vs. Applications

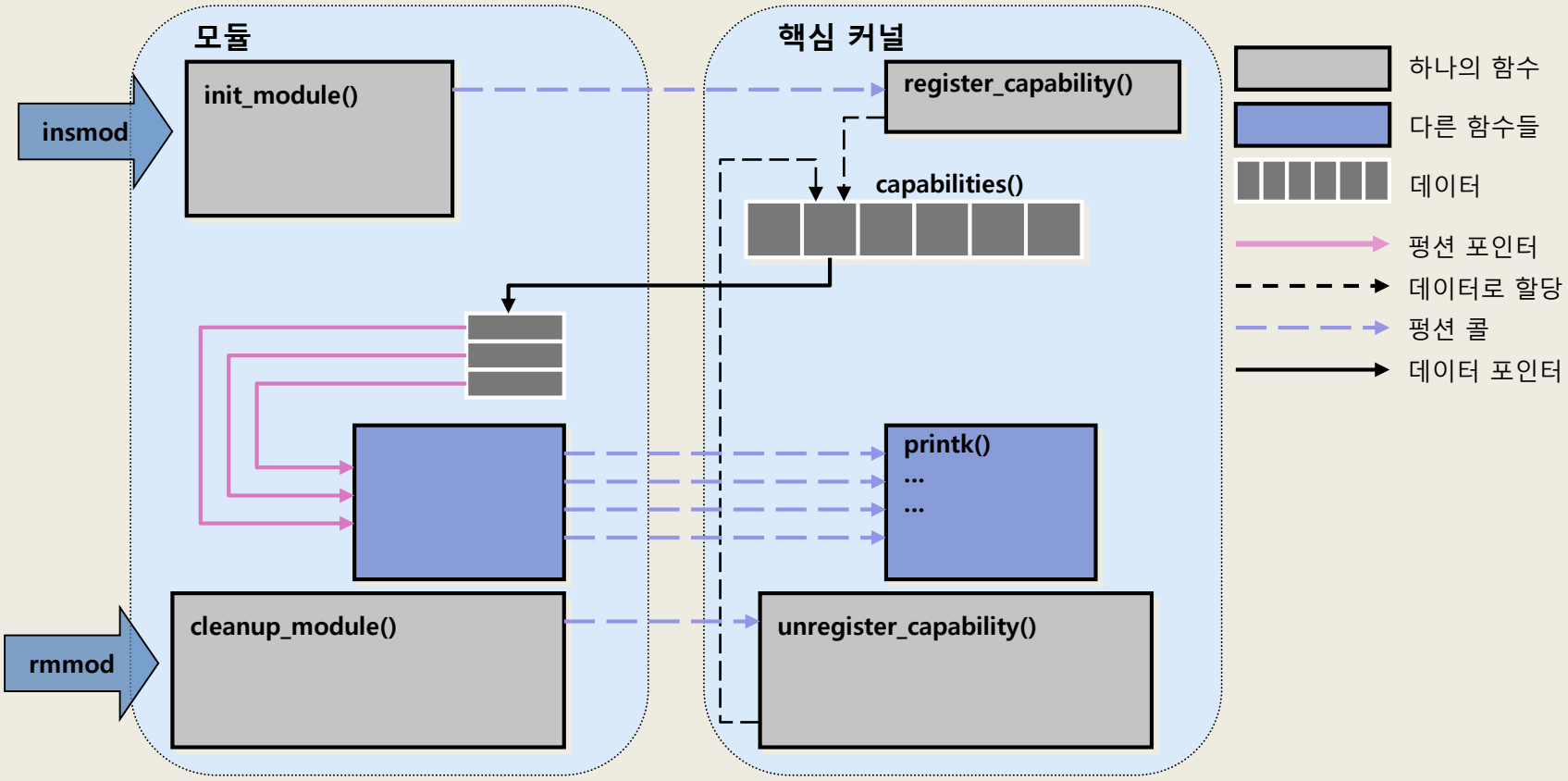
Applications

- **Can access various functions in user-level libraries (e.g., printf in C library)**

Kernel modules

- **No user-level libraries**
- **printk is defined within the kernel**
  - Exported to modules
- **Should include only header files defined within the kernel source tree**

모듈 프로그램의 동작 설명



※ 커널버전 2.4

모듈 프로그래밍 명령어 요약

- **insmod**
  - module을 설치(install)
- **rmmod**
  - 실행중인 modules을 제거(unload)
- **lsmod**
  - Load된 module들의 정보를 표시
- **depmod**
  - Module들의 symbol들을 이용하여 Makefile과 유사한 dependency file을 생성
- **modprobe**
  - depmod명령으로 생성된 dependency를 이용하여 지정된 디렉토리의 module들과 연관된 module들을 자동으로 load
- **modinfo**
  - 목적화일을 검사해서 관련된 정보를 표시

Dynamic Linking

- **Each module is made up of object code that can be dynamically linked to the running kernel**
  - **When loading the kernel module, unresolved symbols in the module are linked to the symbol table of the kernel**
  - **The symbol table contains the addresses of global functions and variables that are needed to implement device drivers**

Module Programming

Entry and Exit Functions

- **module_init() and module_exit()**
  - The macros specifies initialization and cleanup functions of the module
  - static int __init my_init (void) { … }
    - Returns 0 if it succeeds
  - static void __exit my_cleanup (void) { … }

- **Kernel version 2.4: init_module() and cleanup_module()**

Module Programming

⊙ Parameter Passing

- **Load-time configuration is possible**
  - Parameter values can be assigned by insmod or modprobe
  - More flexible than compile-time configuration
- **module_param()**

```
insmod mymod count=10
============================
static int count = 0;
module_param(count, int, S_IRUGO);
```

- **Kernel version 2.4: MODULE_PARM()**

Module Programming

Module Usage Count

- **Kernel version 2.6:**
  - The kernel manages this by itself

- **Kernel version 2.4: MOD_IN_USE**
  - Increased and decreased by
  - MOD_INC_USE_COUNT and
  - MOD_DEC_USE_COUNT

Module License

- MODULE_LICENSE()
  - Mandatory in 2.6

License	Description
GPL	GNU Public License v2 or later
GPL v2	GNU Public License v2
GPL and additional rights	GNU Public License v2 rights and more
Dual BSD/GPL	GNU Public License v2 or BSD license choice
Dual MPL/GPL	GNU Public License v2 or Mozilla license choice
Proprietary	Non free products

- 현재 설치되어 있는 모듈을 확인
  - /sbin/lsmod
- 모듈 제작-> 오브젝트 파일 생성
  - /sbin/insmod my_driver.o
  - 재확인: /sbin/lsmod
- 모듈제거
  - /sbin/rmmod my_driver
- my_driver.c 파일 예문

```
int init_module(void)
        extern char kernel_version[];
        printk("module test\n");
return 0;
```

Exporting Symbols

Exported Kernel Symbols

- **Can be checked with**
  - /proc/kallsyms
  - Kernel version 2.4: ksyms

- **Exported by**
  - EXPORT_SYMBOL() or EXPORT_SYMBOL_GPL()
  - Symbols Exported by Modules

- **A module also can export its symbols to other modules**

- **Become part of the kernel symbol table when the module is loaded**

⊙ 커널의 심볼 테이블에 등록

- **커널 내부에서 별도로 관리**
- **[커널 소스 디렉토리]/kernel/ksyms.c 에 등록**
  - 라즈베리파이는 /lib/modules/3.18.10+/source/kernel
- **#cat /proc/ksyms 명령으로 등록된 심볼의 목록을 확인**

⊙ 심볼 추가 등록에 사용되는 매크로 명령

심볼 등록 매크로	설 명
EXPORT_SYMTAB	심볼 테이블을 등록하기 위해 소스코드에서 'linux/module.h'전에 정의
EXPORT_SYMBOL	심볼을 등록하는 매크로로써 커널 소스의 kernel/ksyms.c 파일을 통해 정의
EXPORT_NO_SYMBOL	모듈 프로그램에서 아무런 심볼 등록을 원하지 않을 경우 이 매크로를 삽입

Example

```
#include <linux/init.h>
#include <linux/module.h>
MODULE_LICENSE("Dual BSD/GPL");
static int hello_init(void)
{
    printk(KERN_ALERT "Hello, World #2\n");
    return 0; /* return zero on successful loading */
}
static void hello_exit(void)
{
    printk(KERN_ALERT "Goodbye\n");
}
module_init(hello_init);
module_exit(hello_exit);
```

Makefile Example

```
obj-m := gymod2.o

KERNELDIR := /lib/modules/$(shell uname -r)/build
PWD := $(shell pwd)

default:
	$(MAKE) -C $(KERNELDIR) M=$(PWD) modules

clean:
	$(MAKE) -C $(KERNELDIR) M=$(PWD) clean
```

Compiling Modules

- **obj-m := mymod.o**
  - **Means that there is a module to be built from the object file mymod.o**
  - **If your module is generated from several files**
    - obj-m := mymod.o
    - mymod-objs := myfile1.o myfile2.o
- **make –C <path_to_kernel>**
  - **M=<module_src_dir> modules**
    - Builds a kernel module
    - The target refers to the list of modules found in the obj-m variable

Kernel Dependency

- **Modules are strongly tied to the data structures and functions defined in a particular kernel version**
- <linux/version.h>
  - LINUX_VERSION_CODE
    - 132633 for 2.6.25
  - KERNEL_VERSION(major, minor, release)
    - KERNEL_VERSION(2.6.25)
    - returns 132633

```
#if LINUX_VERSION_CODE >= KERNEL_VERSION(2,6,0)
    /* Codes for recent kernel versions */
#else
    /* Codes for old kernel versions */
#endif
```