광운대학교
KwangWoon University

2016년 2학기 **운영체제실습** 6주차

Module Programming, Wrapping

Dept. of Computer Engineering,
Kwangwoon Univ.

Contents

- **Module Programming**
  - 모듈의 이해
  - 특징
  - 모듈 프로그래밍 절차
  - 커널 모듈 구성
  - 커널 모듈의 추가 및 제거

- **실습**
  - 모듈 Load / Unload
  - Wrapping을 통한 Module Programming

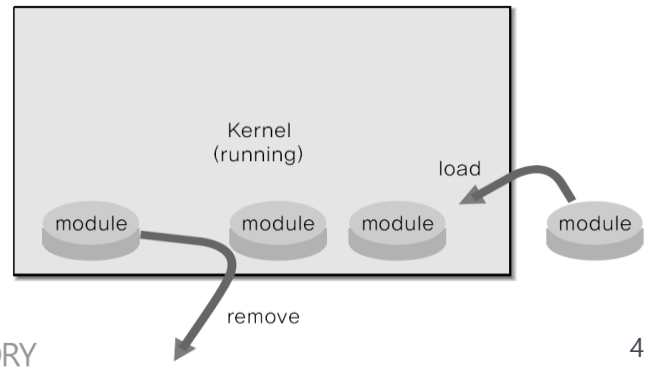
광운대학교
KwangWoon University

Module Programming

Dept. of Computer Engineering,
Kwangwoon Univ.

Kernel Module

- **Kernel Module**
  - 커널 코드의 일부를 커널이 동작하는 상태에서 로드 또는 언 로드 가능
  - 커널 크기 최소화, 유연성 제공
    - 커널이 실행 중에 동적으로 로딩하여 커널과 링크함으로써 커널의 기능을 확장하여 사용할 수 있다.
    - 불필요 시에 커널과의 링크를 풀고 메모리에서 제거 할 수 있다.
      ➔ 커널 재 컴파일 없이 커널 기능 확장 가능
  - 각종 디바이스 드라이버를 사용할 때 유용
    - 마우스, 키보드, 사운드카드 드라이버는 종류가 다양하고 상황에 따라 사용하지 않을 수 있기 때문
      ➔ 새로운 장치를 추가할 때마다 커널을 재 컴파일 한다면?
  - 파일시스템, 통신 프로토콜 및 시스템 콜 등도 모듈로 구현 가능

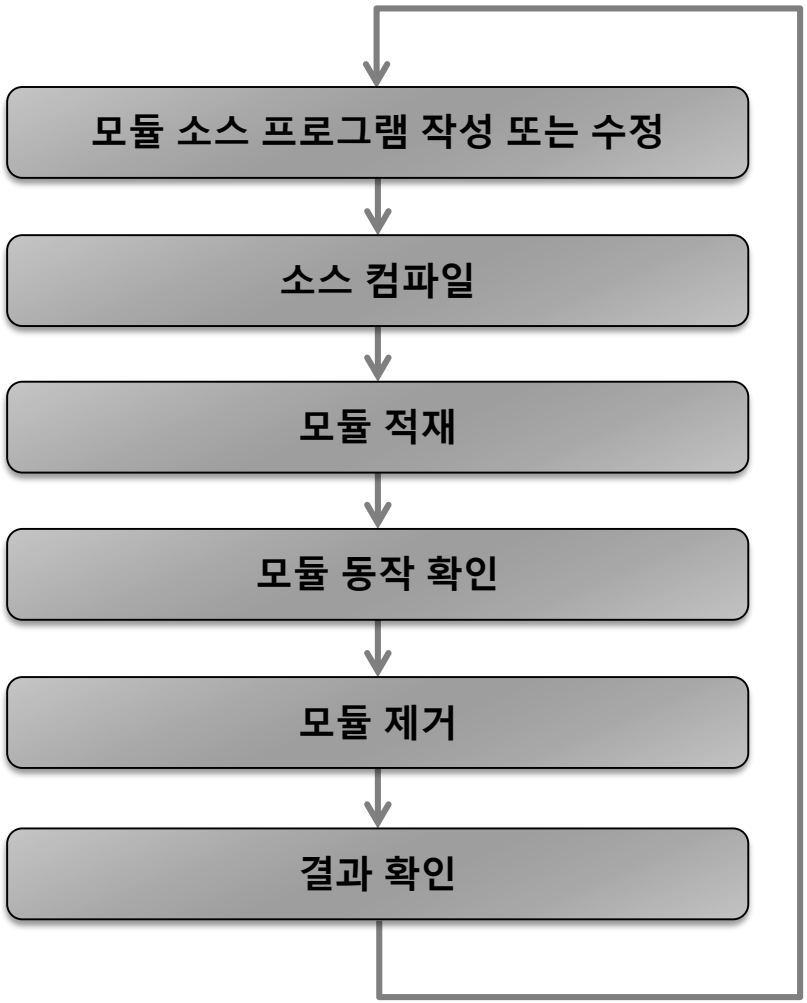


Kernel Module

- **특징**
  - 사건 구동형(event-driven program) 방식으로 작성
  - 내부 `main()` 함수 없음
  - 명시적인 커널 모듈 설치 및 제거 과정이 필요
    - `insmod/rmmod` 명령어
  - 외부로 공개 할 전역변수 사용에 주의
  - 디바이스 드라이버, 파일시스템, 네트워크 프로토콜 스택 등에 적용
    - 커널 경량화를 위해 반드시 필요
    - 임베디드 시스템의 경우, 제한적인 자원으로 인해 커널 등 시스템 소프트웨어의 최소화가 필요
  - 커널에 적재 모듈 프로그램은 무제한의 특권을 가지므로 신중하게 작성해야 함.

Kernel Module

- 모듈 프로그래밍 절차



커널 모듈의 추가 및 제거

- **커널 모듈 추가**
  - 커널 모듈이 적재되면 오브젝트 파일의 내용이 커널 영역으로 복사
  - `init_module()` 함수를 호출하여 적재된 커널 모듈 초기화
  - 커널 모듈의 초기화가 끝나면 커널 모듈 등록

- **커널 모듈을 제거**
  - 커널 모듈이 제거되면 `cleanup_module()` 함수를 호출
  - `init_module()` 함수에서 할당 받은 자원을 반환
  - 커널 모듈의 등록 해제
  - 커널 모듈의 오브젝트 코드를 위해 할당했던 메모리를 반환

커널 모듈의 추가 및 제거

- **커널 모듈 매크로**
  - 커널 ver 2.4 이상에서는 module_init, module_exit 매크로 지원
    - 함수 이름에 의한 종속 관계를 해결
    - module_init 매크로 : startup 함수 등록
    - module_exit 매크로 : cleanup 함수 등록

```
#include <linux/module.h>
/* global variables */
…
int module_start() { /* 모듈이 설치될 때에 초기화를 수행하는 코드 */ }
int module_end() { /* 모듈이 제거될 때에 반환작업을 수행하는 코드 */ }

module_init(module_start);
module_exit(module_end);
…
```

커널 모듈의 추가 및 제거

- **커널 모듈 구성 (make)**
  - 모듈 프로그램의 Makefile
    - 모듈 생성을 위한 일반적인 Makefile

```
1  obj-m := test.o #module object name
2
3  KDIR := /lib/modules/$(shell uname -r)/build #kernel module directory
4  PWD := $(shell pwd) #cwd
5
6  default:
7      $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules #-C is change directory opt.
8
9  clean:
10     $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) clean
```

- `obj-m := test.o` ➔ 모듈로 생성할 이름 정의(test)
- `KDIR` ➔ 커널 코드 디렉토리 위치 (symbolic link)
- `$(shell uname -r)` ➔ 현재 실행 중인 커널 버전
- `PWD` ➔ 컴파일 대상이 되는 모듈소스가 있는 위치(test.c 위치)
- `default` ➔ (target) 모듈을 컴파일 하는 명령
- `clean` ➔ (target) 컴파일 결과로 생성된 파일 모두 지움

커널 모듈의 추가 및 제거

- **사용 명령어**

이 름	용 도
insmod	Simple program to insert a module into the Linux Kernel (load)
rmmod	simple program to remove a module from the Linux Kernel (unload)
lsmod	program to show the status of modules in the Linux Kernel
depmod	program to generate modules.dep and map files (커널 내부에 적재된 모듈 간 의존성 검사)
modprobe	program to add and remove modules from the Linux Kernel (insmod와 유사하나, 모듈간 의존성을 검사하여 그 결과 누락된 다른 모듈을 찾아서 적재)
modinfo	program to show information about a Linux Kernel module

커널 모듈의 추가 및 제거

- **$ depmod -a**
  - /lib/modules/`uname -r`/modules.dep 파일과 맵 파일을 새롭게 생성
  - 개인이 개발한 드라이버를 해당 시스템에 설치할 경우 depmod 명령어 권장

```
# depmod -a
# cat modules.dep
kernel/arch/x86/kernel/cpu/mcheck/mce-xeon75xx.ko:
kernel/arch/x86/kernel/cpu/mcheck/mce-inject.ko:
kernel/arch/x86/kernel/cpu/cpufreq/e_powersaver.ko:
kernel/arch/x86/kernel/cpu/cpufreq/p4-clockmod.ko:
kernel/arch/x86/kernel/msr.ko:
kernel/arch/x86/kernel/cpuid.ko:
```

- **$ modprobe**
  - depmod로 생성된 modules.dep 파일에서 해당 모듈의 위치를 파악하고 모듈을 메모리에 적재
    - modinfo 명령어도 동일

예제 – Module Load / Unload

```c
//test.c
#include <linux/module.h>

int test_init(void)
{
    printk("insmod! %lld\n", get_jiffies_64());
    return 0;
}

void test_exit(void)
{
    printk("rmmod! %lld\n", get_jiffies_64());
}

module_init(test_init);
module_exit(test_exit);
MODULE_LICENSE("GPL");
```

```
obj-m := test.o #module object name

KDIR := /lib/modules/$(shell uname -r)/build #kernel module directory
PWD := $(shell pwd) #cwd

default:
    $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules #-C is change directory opt.

clean:
    $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) clean

```

Makefile

```
ssangkong@ssangkong-sslab:~/module$ sudo insmod test.ko
ssangkong@ssangkong-sslab:~/module$ lsmod | grep test
test                   12420  0
ssangkong@ssangkong-sslab:~/module$ sudo rmmod test
ssangkong@ssangkong-sslab:~/module$ sudo dmesg | tail -n 2
[  427.628920] insmod! 4294997243
[  440.247566] rmmod! 4295000398
```

모듈 적재, 확인 및 제거

```
ssangkong@ssangkong-sslab:~/module$ make
make -C /lib/modules/3.2.28-OSLAB/build  SUBDIRS=/home/ssangkong/module  modules
 #-C is change directory opt.
make[1]: Entering directory `/usr/src/linux-3.2.28'
  CC [M]  /home/ssangkong/module/test.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/ssangkong/module/test.mod.o
  LD [M]  /home/ssangkong/module/test.ko
make[1]: Leaving directory `/usr/src/linux-3.2.28'
```

모듈 컴파일

예제 – Module Load / Unload

- **Question**
  - 예제에서 insmod 후 몇 초 후에 rmmod를 했나?
    - (**4295000398** – **4294997243**)/250 = 12.62초
    - (**440.247566** – **427.628920**) = 12.618646초
    - dmesg에서 제공하는 시간과 jiffies로 계산한 결과가 일치함을 확인할 수 있음

```
ssangkong@ssangkong-sslab:~/module$ sudo insmod test.ko
ssangkong@ssangkong-sslab:~/module$ lsmod | grep test
test                   12420  0
ssangkong@ssangkong-sslab:~/module$ sudo rmmod test
ssangkong@ssangkong-sslab:~/module$ sudo dmesg | tail -n 2
[  427.628920] insmod! 4294997243
[  440.247566] rmmod! 4295000398
```

Wrapping

Dept. of Computer Engineering,
Kwangwoon Univ.

Wrapping을 통한 Module Programming

ssangkong — ssangkong@ssangkong-sslab: ~/hooking — ssh — 80×53

```
 1 #include <linux/module.h>
 2 #include <linux/highmem.h>
 3 #include <asm/unistd.h>
 4
 5 #define __NR_add 349
 6
 7 void **syscall_table = (void**)SYSCALL_TABLE;
 8
 9 asmlinkage int (*real_add)(int, int);
10
11 asmlinkage int sub(int a, int b)
12 {
13     printk("kernel hooked func %d %d\n", a, b);
14     return a-b;
15 }
16
17 /* Make the page writable */
18 void make_rw(void *address)
19 {
20     unsigned int level;
21     pte_t *pte = lookup_address((unsigned long)address, &level);
22     if( pte->pte &~ _PAGE_RW )
23         pte->pte |= _PAGE_RW;
24 }
25
26 /* Make the page write protected */
27 void make_ro(void *address)
28 {
29     unsigned int level;
30     pte_t *pte = lookup_address((unsigned long)address, &level);
31     pte->pte = pte->pte &~ _PAGE_RW;
32 }
```

```
34 int hooking_init(void)
35 {
36     make_rw(syscall_table);
37     real_add = syscall_table[__NR_add];
38     syscall_table[__NR_add] = sub;
39     return 0;
40 }
41
42 void hooking_exit(void)
43 {
44     syscall_table[__NR_add] = real_add;
45     make_ro(syscall_table);
46 }
47
48 module_init(hooking_init);
49 module_exit(hooking_exit);
50 MODULE_LICENSE("GPL");
51
hooking.c
:wq
```

```
325 /*
326  * Lookup the page table entry for a virtual address. Return a pointer
327  * to the entry and the level of the mapping.
328  *
329  * Note: We return pud and pmd either when the entry is marked large
330  * or when the present bit is not set. Otherwise we would return a
331  * pointer to a nonexisting mapping.
332  */
333 pte_t *lookup_address(unsigned long address, unsigned int *level)
"arch/x86/mm/pageattr.c" [readonly] 1383 lines --22%--       308,1-4       22%
```

Wrapping을 통한 Module Programming

ssangkong — ssangkong@ssangkong-sslab: ~/hooking — ssh — 80×24

```
 1 #include <linux/unistd.h>
 2 #include <stdio.h>
 3
 4 int main(int argc, const char *argv[])
 5 {
 6     int a=5, b=3;
 7     printf("%d op %d = %d\n", a, b, syscall(__NR_add, a, b));
 8     return 0;
 9 }
~
~
~
~
~
~
~
~
~
~
~
~
~
hooking_test.c                                    3,0-1         All
:wq
```

Wrapping을 통한 Module Programming

ssangkong — ssangkong@ssangkong-sslab: ~/hooking — ssh — 80×26

```
1  SRCS := hooking_test.c
2  obj-m := hooking.o
3
4  SYSCALL_ADDRESS = 0x$(subst R sys_call_table,,$(shell grep sys_call_table /b
   oot/System.map-$(shell uname -r)))
5  CFLAGS_hooking.o += -DSYSCALL_TABLE=$(SYSCALL_ADDRESS)
6
7  KDIR := /lib/modules/$(shell uname -r)/build
8  PWD := $(shell pwd)
9
10 default:
11     $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules
12
13 run:default hooking_test
14     sudo insmod hooking.ko
15     ./$(SRCS:.c=)
16     sudo rmmod hooking
17     ./$(SRCS:.c=)
18
19 $(SRCS:.c=):$(SRCS:.c=.o)
20
21 clean:
22     $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) clean
23     $(RM) -rf hooking_test.o hooking_test
Makefile                                              1,1           Top
:wq
```

Wrapping을 통한 Module Programming

```
ssangkong — ssangkong@ssangkong-sslab: ~/hooking — ssh — 80×24
ssangkong@ssangkong-sslab:~/hooking$ sudo make run
make -C /lib/modules/3.2.28-OSLAB/build SUBDIRS=/home/ssangkong/hooking modules
make[1]: Entering directory `/usr/src/linux-3.2.28'
  CC [M]  /home/ssangkong/hooking/hooking.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/ssangkong/hooking/hooking.mod.o
  LD [M]  /home/ssangkong/hooking/hooking.ko
make[1]: Leaving directory `/usr/src/linux-3.2.28'
cc    -c -o hooking_test.o hooking_test.c
cc   hooking_test.o   -o hooking_test
sudo insmod hooking.ko
./hooking_test
5 op 3 = 2
sudo rmmod hooking
./hooking_test
5 op 3 = 8
ssangkong@ssangkong-sslab:~/hooking$
```

컴파일

← 모듈 삽입후 테스트(뺄셈)

← 모듈 제거후 테스트(덧셈)

Wrapping을 통한 Module Programming

```
7 void **syscall_table = (void**)SYSCALL_TABLE;
```

➔ 선언된 곳이 없다 ?

ssangkong — ssangkong@ssangkong-sslab: ~/hooking — ssh — 80×26

```
1  SRCS := hooking_test.c
2  obj-m := hooking.o
3
4  SYSCALL_ADDRESS = 0x$(subst R sys_call_table,,$(shell grep sys_call_table /b
   oot/System.map-$(shell uname -r)))
5  CFLAGS_hooking.o += -DSYSCALL_TABLE=$(SYSCALL_ADDRESS)
6
7  KDIR := /lib/modules/$(shell uname -r)/build
8  PWD := $(shell pwd)
9
10 default:
11     $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules
12
13 run:default hooking_test
14     sudo insmod hooking.ko
15     ./$(SRCS:.c=)
16     sudo rmmod hooking
17     ./$(SRCS:.c=)
18
19 $(SRCS:.c=):$(SRCS:.c=.o)
20
21 clean:
22     $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) clean
23     $(RM) -rf hooking_test.o hooking_test
Makefile                                    1,1           Top
:wq
```

컴파일시 -D옵션으로 #define 한 것과 같은 효과

ex)
-DA=3 옵션을 주면,
#define A 3 과 같은 효과

Wrapping을 통한 Module Programming

ssangkong — ssangkong@ssangkong-sslab: ~/hooking — ssh — 80×26

```
1  SRCS := hooking_test.c
2  obj-m := hooking.o
3
4  SYSCALL_ADDRESS = 0x$(subst R sys_call_table,,$(shell grep sys_call_table /b
   oot/System.map-$(shell uname -r)))
5  CFLAGS_hooking.o += -DSYSCALL_TABLE=$(SYSCALL_ADDRESS)
6
7  KDIR := /lib/modules/$(shell uname -r)/build
8  PWD := $(shell pwd)
9
10 default:
11     $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules
12
13 run:default hooking_test
14     sudo insmod hooking.ko
15     ./$(SRCS:.c=)
16     sudo rmmod hooking
17     ./$(SRCS:.c=)
18
19 $(SRCS:.c=):$(SRCS:.c=.o)
20
21 clean:
22     $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) clean
23     $(RM) -rf hooking_test.o hooking_test
Makefile                                              1,1           Top
:wq
```

이 것은 무엇일까요?

Wrapping을 통한 Module Programming

```
SYSCALL_ADDRESS = 0x$(subst R sys_call_table,,$(shell grep sys_call_table /b
oot/System.map-$(shell uname -r)))
```

```
sslab@sslab-VirtualBox:/usr/src/linux-3.2.28$ uname -r
3.2.28-OSLAB-ASSISTANT
sslab@sslab-VirtualBox:/usr/src/linux-3.2.28$
```

```
sslab@sslab-VirtualBox:/usr/src/linux-3.2.28$ cat /boot/System.map-3.2.28-OSLAB
-ASSISTANT | grep sys_call_table
c1595000 R sys_call_table
sslab@sslab-VirtualBox:/usr/src/linux-3.2.28$
```

/boot/System.map-$(uname -r) 파일은 커널에서 사용하는 심볼 테이블.

위 결과는 이 테이블에서 sys_call_table(시스템 콜 테이블)이 포함된 라인이다.
➔ 주소(0xc1599000), 권한(Read), 이름(sys_call_table)을 알 수 있다.

$(shell grep sys_call_table /boot/System.map-$(shell uname -r))
= c1599000 R sys_call_table

$(subst a,b,c)는 치환이다. c에서 a를 b로 바꿔라.
➔ 즉 'c1599000 R sys_call_table'에서 'R sys_call_table'을 ''로 바꿔라.

$(subst R sys_call_table,,c1599000 R sys_call_table)
= c1599000

16진수이기 때문에, 앞에 0x를 붙여주었다.
= 0xc1599000