KWANGWOON UNIVERSITY

2024년 2학기 **운영체제실습** 4주차

Module Programming, Wrapping

System Software Laboratory
School of Computer and Information 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

KWANGWOON UNIVERSITY

Module Programming

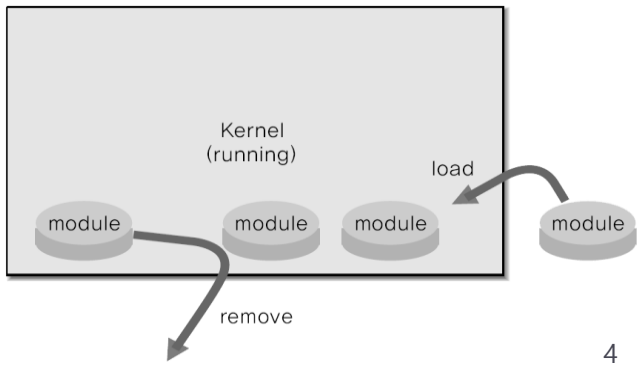
System Software Laboratory
School of Computer and Information Engineering
Kwangwoon Univ.

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

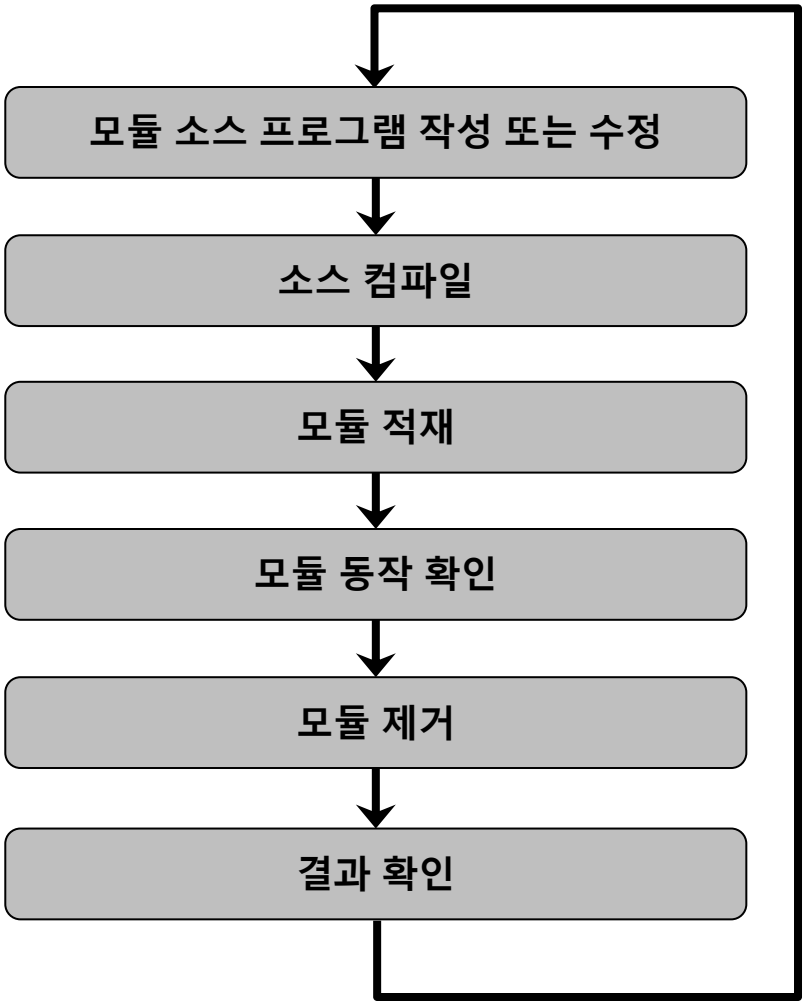


Kernel Module

- **특징**
  - 사건 구동형(event-driven program) 방식으로 작성
  - 명시적인 커널 모듈 설치 및 제거 과정이 필요
    - `insmod, rmmod` 명령어 등
  - 내부 `main()` 함수 없음
  - 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
  - 외부로 공개할 전역변수 사용에 주의
  - 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
  - `module_init()` 에 명시한 함수를 호출하여 커널 모듈 등록
  - 설치된 모듈은 '/proc/modules' 파일에 기록
    - $ cat /proc/modules 로 확인 가능

- **커널 모듈을 제거**
  - 커널 모듈이 제거되면 `module_exit()` 에 명시한 함수를 호출
  - `module_init()` 에서 호출한 함수에서 할당한 자원을 반환
  - 커널 모듈의 등록 해제
  - 커널 모듈의 오브젝트 코드를 위해 할당했던 메모리를 반환

커널 모듈의 추가 및 제거

- **커널 모듈 추가 및 제거 예시**
  - module_init, module_exit 매크로 지원
    - module_init() : startup 함수 (모듈을 로드하면 해당 매크로에 명시된 함수를 호출)
    - module_exit() : cleanup 함수 등록 (언 로드시 등록된 함수 호출)

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

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

insmod or modprobe

rmmod

커널 모듈의 추가 및 제거

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

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

KDIR := /lib/modules/$(shell uname -r)/build    #Kernel module Directory
PWD := $(CURDIR)        #Current Worcking Directory

all:
        $(MAKE) -C $(KDIR) M=$(PWD) modules     # -C is change directory option.
clean:
        $(MAKE) -C $(KDIR) M=$(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 모듈 코드가 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**

이름	용도
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 (디스크 내 적재된 커널 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 파일을 새로 생성
  - $ cat modules.dep

```
kernel/net/ax25/ax25.ko:                                    0개 의존
kernel/net/can/can.ko:
kernel/net/can/can-raw.ko: kernel/net/can/can.ko            1개 의존
kernel/net/can/can-bcm.ko: kernel/net/can/can.ko
kernel/net/can/can-gw.ko: kernel/net/can/can.ko
kernel/net/bluetooth/bluetooth.ko: kernel/crypto/ecdh_generic.ko
kernel/net/bluetooth/rfcomm/rfcomm.ko: kernel/net/bluetooth/bluetooth.ko kernel/
crypto/ecdh_generic.ko  2개 의존
kernel/net/bluetooth/bnep/bnep.ko: kernel/net/bluetooth/bluetooth.ko kernel/cryp
to/ecdh_generic.ko
kernel/net/bluetooth/cmtp/cmtp.ko: kernel/drivers/isdn/capi/kernelcapi.ko kernel
/net/bluetooth/bluetooth.ko kernel/crypto/ecdh_generic.ko  3개 의존
kernel/net/bluetooth/hidp/hidp.ko: kernel/drivers/hid/hid.ko kernel/net/bluetoot
h/bluetooth.ko kernel/crypto/ecdh_generic.ko
```

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

예제 – Module Load / Unload

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

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

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

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

test.c

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

KDIR := /lib/modules/$(shell uname -r)/build    #Kernel module Directory
PWD := $(CURDIR)        #Current Worcking Directory

all:
        $(MAKE) -C $(KDIR) M=$(PWD) modules     # -C is change directory option.
clean:
        $(MAKE) -C $(KDIR) M=$(PWD) clean
```

Makefile

예제 – Module Load / Unload

```
os2019110613@ubuntu:~/moduleTest$ sudo insmod test.ko
os2019110613@ubuntu:~/moduleTest$ lsmod |grep test
test                   16384  0
os2019110613@ubuntu:~/moduleTest$ sudo rmmod test
os2019110613@ubuntu:~/moduleTest$ sudo dmesg |tail -n -2
[12291.896358] insmod! 4297965144
[12304.603767] rmmod! 4297968320
os2019110613@ubuntu:~/moduleTest$
```

모듈 적재, 확인 및 제거

```
os2019110613@ubuntu:~/moduleTest$ sudo make
make -C /lib/modules/4.19.67-2019110613/build SUBDIRS=/home/os2019110613/moduleT
est modules
make[1]: Entering directory '/home/os2019110613/linux-4.19.67'
  CC [M]  /home/os2019110613/moduleTest/test.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/os2019110613/moduleTest/test.mod.o
  LD [M]  /home/os2019110613/moduleTest/test.ko
make[1]: Leaving directory '/home/os2019110613/linux-4.19.67'
```

모듈 컴파일

예제 – Module Load / Unload

- **Question**
  - 예제에서 insmod 후 몇 초 후에 rmmod를 했나?
    - (**4297968320** – **4297965144**)/250 = 12.824초
    - (**12304.603767** – **12291.896358**) = 12.707409초

    - dmesg에서 제공하는 시간과 jiffies로 계산한 결과가 일치함을 확인할 수 있음

```
os2019110613@ubuntu:~/moduleTest$ sudo insmod test.ko
os2019110613@ubuntu:~/moduleTest$ lsmod |grep test
test                   16384  0
os2019110613@ubuntu:~/moduleTest$ sudo rmmod test
os2019110613@ubuntu:~/moduleTest$ sudo dmesg |tail -n -2
[12291.896358] insmod! 4297965144
[12304.603767] rmmod! 4297968320
os2019110613@ubuntu:~/moduleTest$
```

KWANGWOON UNIVERSITY

Wrapping

System Software Laboratory
School of Computer and Information Engineering
Kwangwoon Univ.

System Call wrapping을 위한 Module Programming

- **Hooking**
  - 운영체제, 응용 프로그램 등에서 동작하는 함수 등을 다른 것으로 대체하는 행위
  - 본 수업에서는 Linux 커널 내 존재하는 기존의 시스템 콜을 다른 행동을 하는 시스템 콜로 hooking
    - 시스템 콜 hooking 작업을 kernel module에서 수행

System Call wrapping을 위한 Module Programming

- **Example**
  - hooking.c
    - 11: `void **syscall_table;`
      - 기존의 add 시스템 콜 주소를 저장할 포인터
    - 24: `__SYSCALL_DEFINEx(2, sub, …`
      - 기존의 add 시스템 콜을 대체할 sub 시스템 콜 정의
      - 기존에 수행한 덧셈 연산 대신 뺄셈 연산을 수행
      - `__SYSCALL_DEFINEx()`
        - `SYSCALL_DEFINE`*n*`()` 매크로 (단, $0 \le n \le 6$) 에서 사용하며, 시스템 콜을 실제 생성하는 매크로.
        - 단순히 함수 정의 뿐만 아니라 시스템 콜 호출을 위한 관련 작업을 일괄 수행해주는 매크로로, 이를 통해 정의해야 후킹 가능

```
 1 #include <linux/module.h>
 2 #include <linux/highmem.h>
 3 #include <linux/kallsyms.h> /* kallsyms_lookup_name() */
 4 #include <linux/syscalls.h> /* __SYSCALL_DEFINEx() */
 5 #include <asm/syscall_wrapper.h> /* __SYCALL_DEFINEx() */
 6
 8
 9 void **syscall_table;
10
11 void *real_add;
12
13 /**
14   * Function to be hooked
15   *  - Low-level macro of the "SYSCALL_DEFINEn" (n: 1-6)
16   *  - Parameters of this macro
17   *     --> 2   : Number of parameters of this system call
18   *     --> sub : Name of the new system call hooked
19   *     --> int : First parameter type of the "sub" system call
20   *     --> a   : First parameter name of the system call
21   *     --> int : Second parameter type
22   *     --> b   : Second parameter name
23 **/
24 __SYSCALL_DEFINEx(2, sub, int, a, int, b)
25 {
26     printk("Kernel hooked func: %d - %d\n", a, b);
27     return a-b;
28 }
```

System Call wrapping을 위한 Module Programming

- **Example**
  - hooking.c
    - 30: `void make_rw(void *addr)`
      - addr이 속해 있는 페이지의 읽기 및 쓰기 권한을 부여하는 함수
      - 기본적으로, system call table은 쓰기 권한이 존재하지 않음
        - 본 함수를 호출하여 쓰기 권한이 없는 system call table에 쓰기 권한을 부여

```
30 void make_rw(void *addr)
31 {
32     unsigned int level;
33     pte_t *pte = lookup_address((u64)addr, &level);
34
35     if(pte->pte &~ _PAGE_RW)
36         pte->pte |= _PAGE_RW;
37 }
38
39 void make_ro(void *addr)
40 {
41     unsigned int level;
42     pte_t *pte = lookup_address((u64)addr, &level);
43
44     pte->pte = pte->pte &~ _PAGE_RW;
45 }
```

    - 39: `void make_ro(void *addr)`
      - addr이 속해 있는 페이지의 읽기 및 쓰기 권한을 회수

System Call wrapping을 위한 Module Programming

- **Example**
  - hooking.c
    - 47: …hooking_init(void)
      - 모듈 적재 시 호출되는 함수
    - 50: syscall_table = …
      - System call table의 주소를 찾는 함수
        - “sys_call_table” 이라는 전역 변수로 선언되어 있는 시스템 콜 위치를 찾음
    - 56: make_rw(syscall_table);
      - 쓰기 금지되어 있는 시스템 콜 테이블에 쓰기 권한 부여
    - 58: real_add = syscall_table[__NR_add];
      - 모듈 해제 시 기존의 시스템 콜을 원상 복구 하기 위해, 기존 시스템 콜 주소 저장

```
47 static int __init hooking_init(void)
48 {
49     /* Find system call table */
50     syscall_table = (void**) kallsyms_lookup_name("sys_call_table");
51
52     /*
53      * Change permission of the page of system call table
54      * to both readable and writable
55      */
56     make_rw(syscall_table);
57
58     real_add = syscall_table[__NR_add];
59     syscall_table[__NR_add] = __x64_syssub;
60
61     return 0;
62 }
63
64 static void __exit hooking_exit(void)
65 {
66     syscall_table[__NR_add] = real_add;
67
68     /* Recover the page's permission (i.e. read-only) */
69     make_ro(syscall_table);
70 }
71
72 module_init(hooking_init);
73 module_exit(hooking_exit);
74 MODULE_LICENSE("GPL");
```

System Call wrapping을 위한 Module Programming

- **Example**
  - hooking.c
    - 59: syscall_table[__NR_add]…
      - 후킹할 “sub” 시스템 콜을 add 시스템 콜 대신 대체하는 과정
      - Line 24의 결과로 __x64_syssub 함수가 생성되며, 이를 삽입
    - 64: …hooking_exit(void)
      - 모듈 해제 시 호출되는 함수
    - 66: syscall_table[__NR_add]…
      - 후킹했던 시스템 콜을 원래대로 복원하는 작업

```
47 static int __init hooking_init(void)
48 {
49     /* Find system call table */
50     syscall_table = (void**) kallsyms_lookup_name("sys_call_table");
51
52     /*
53      * Change permission of the page of system call table
54      * to both readable and writable
55      */
56     make_rw(syscall_table);
57
58     real_add = syscall_table[__NR_add];
59     syscall_table[__NR_add] = __x64_syssub;
60
61     return 0;
62 }
63
64 static void __exit hooking_exit(void)
65 {
66     syscall_table[__NR_add] = real_add;
67
68     /* Recover the page's permission (i.e. read-only) */
69     make_ro(syscall_table);
70 }
71
72 module_init(hooking_init);
73 module_exit(hooking_exit);
74 MODULE_LICENSE("GPL");
```

System Call wrapping을 위한 Module Programming

- **Example**
  - Makefile (for module)

```
obj-m := hooking.o

KDIR := /lib/modules/$(shell uname -r)/build
PWD := $(CURDIR)

all:
	$(MAKE) -C $(KDIR) M=$(PWD) modules
clean:
	$(MAKE) -C $(KDIR) M=$(PWD) clean
```

  - app.c (test application)

```c
#include <stdio.h>

#include <unistd.h> /* syscall() */
#include <sys/syscall.h> /* syscall() */

int main(void)
{
    int a, b;
    long ret;

    a = 7;
    b = 4;

    ret = syscall(549, a, b);
    printf("%d op. %d = %ld\n", a, b, ret);

    return 0;
}
```

System Call wrapping을 위한 Module Programming

- **Example**
  - Results
    - $ ./app

```
sslab@ubuntu:~/test$ ./app
7 op. 4 = 11
```

    - $ sudo insmod hooking.ko
    - $ ./app

```
sslab@ubuntu:~/test$ ./app
7 op. 4 = 3
```

    - $ sudo rmmod hooking
    - $ ./app

```
sslab@ubuntu:~/test$ ./app
7 op. 4 = 11
```