Embedded Systems

Ch 4.
Introduction to Device Driver
Part B

Byung Kook Kim
Dept of EECS
Korea Advanced Institute of Science and Technology

Overview

- 1. Introduction to Linux Kernel
- 2. Kernel Compile
- 3. Kernel Compile Options
- 4. Module
- 5. Device
- 6. Module Build and Run
- 7. Skull Driver

6. Module Build and Run

- Module과 kernel programming에 관련된 기초적인 개념을 소개
  - PC Linux에서 동작함.
- HelloWorld module

```
#define MODULE
#include <linux/module.h>

Init_module(void) {
        printk("<1>Hello, world\n");
        Return 0;
}

Void cleanup_module(void) {
        printk("<1>Good bye cruel world\n");
}
```

Module Build and Run (II)

- Explanations
  - Printk: Linux kernel**에서 정의됨**. Printf**와 비슷하게 동작**.
  - <1>: message**의 우선권**. 1=Highest.

- Compile & run
  - Root privilege required!

```
Root# gcc –c helloworld.c          ; Compile

Root# insmod helloworld.o          ; Insert a module
Hello, world

Root# lsmod                        ; List modules
helloworld.o

Root# rmmod hello.o                ; Remove a module
Root#
```

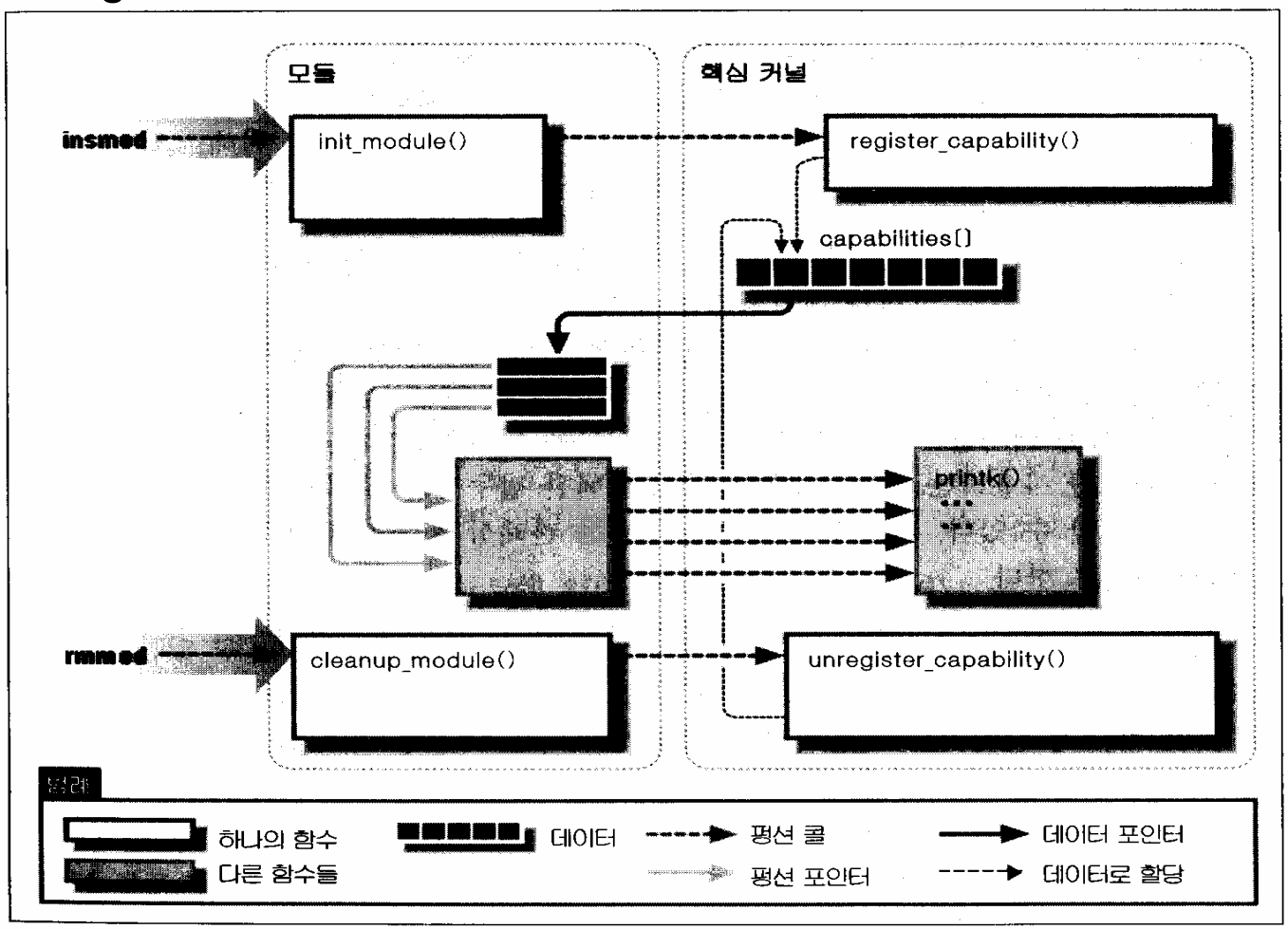
Module Build and Run (III)

- Module vs **응용 프로그램**

	응용 프로그램	Module
Execution	Execution from start to finish	Init_module: **앞으로의 요청에 대비하여 모듈 함수를 등록** Cleanup module: module**을 삭제함**
사용 함수	Libc: printf etc	**외부에 공개된** kernel **함수**: printk etc
Header file	**일반적인** header file	/usr/include/linux /usr/include/asm
Namespace	Independent	Kernel**과 공유**: Namespace pollution All symbol static Prefix for global variable

Module Build and Run (IV)

- Linking a module to the kernel



Module Build and Run (V)

- **응용 프로그램과** module
  - **응용 프로그램**
    - User space**에서 수행**: **가장 낮은** level
    - Hardware**에 직접 접근하는 것과** memory**에 대한 허용하지 않은 접근의 제한**.
  - Module
    - Kernel space**에서 수행** (Kernel **기능의 확장**)
      - Supervisor mode
    - System call**과** hardware interrupt**를 통하여 진입**.
      - System call: run in process context
        - Run for the called process
        - Can access data in process space
      - Interrupt
        - No connection to processes
        - Run asynchronously with processes.

Module Build and Run (VI)

- Kernel의 동시성
  - Multitasking: performed by scheduler
  - Kernel operation:
    - 응용 프로그램을 위한 system call을 수행하기 위하여 비동기적으로 수행된다.
    - 시스템의 각 process를 위해 입출력을 책임지고 있다.
  - Kernel 함수는 single process context안에서 (Sleep mode로 전환되지 않는다면) 완전히 단일 thread로 수행된다.
    - 하나의 device driver는 여러 task들로부터의 요구를 동시에 수행할 수 있어야 한다.
    - 예: 하나의 device에 여러 process가 read call
  - Process 추적
    - Global structure variable 'task_struct current' : 현재 수행되고 있는 user process
    - printk("The process is %s (pid %i)₩n", current->comm, current->pid)

7. Skull Driver

- Sample driver skull (Simple Kernel Utility for Loading Localities)
  - **모듈 타입이 정의된** class **이외**

- Makefile
  - Set of commands to build the object code
  - #ifdef __KERNEL__
    - **외부 프로그램의 사용을 막기 위하여** header**를** include**하기 전에 사용**
  - #def __SMP__
    - SMP (Symmetric Multi-Processor) machine**을 사용할 경우**.
  - MODULE
    - Driver**를** kernel image**에 직접** link **시키지 않을 경우**
  - Compiler flag
    - –O: Inline **함수의 확장을 위해 필요**.
    - -g: enable debugging
    - -Wall: Warn all errors

Skull Driver (II)

- Example makefile

```
# Change it hee or specify it on the “make” commandline
INCLUDEDIR = /usr/include
CFLAGS = -D__KERNEL__ -DMODULE –O –Wall –I$(INCLUDEDIR)
# Extract version number from headers
VER = $(shell awk –F\” ‘/REL/ {print $$2}’
        $(INCLUDEDIR)/linux/version.h)
OBJS = skull.o
All: $(OBJS)
Skull.o: skull_init.o skull_clean.o
        $(LD) –r $^ -o $@
Install:
        install –d /lib/modules/$(VER)/misc /lib/modules/misc
        install –c skull.o /lib/modules/$(VER)/misc
        install –c skull.o /lib/modules/misc
Clean:
        rm –f *.o *~core
```

Skull Driver (III)

- Version 의존성
  - Module code가 link 시킬 각 kernel version마다 모두 재 compile하여야만 한다.
    - Module은 kernel의 일부분으로 동작하여야 하므로.
  - Version code
    - #define VERSION_CODE(vers,rel,seq) (((vers)<<16) | ((rel)<<8 | (seq))
    - Ex) 1.3.5 -> 0x10305 (66309)

Skull Driver (IV)

- Kernel symbol table
  - 공개된 symbol table: /proc/ksyms. Text style.
  - Module이 적재되었을때, 그 안에 선언한 global symbol은 전부 kernel symbol table의 일부가 되고 /proc/ksyms file에 기록된다.
  - 새로운 module은 여러분의 module이 외부 공개한 symbol을 사용할 수 있다.

  - 계층화된 module화
    - Module 쌓기: 복잡한 project에서 유용
    - 각 계층을 단순화시킴.
    - 개발 시간의 단축.

  - Register_symtab()
    - Symbol table 관리를 위한 공식적인 kernel interface
    - 명확하게 symbol table에 적혀있는 symbol들만 kernel에 외부공개 된다.

Skull Driver (V)

- Init_module에서의 error 처리
  - Utility를 등록할 때 error를 만나면 기존의 작업을 취소해야 한다.
  - Init_module의 수행 중 error가 생기면 모든 것을 스스로 제거해야 한다.
    - Kernel이 불안정한 상태로 남아있게 되므로.
  - Error 복구를 위해 goto 문장을 사용하기를 권장한다.
    - 유일한 유용한 상황이라 생각됨.

Skull Driver (VI)

- Error 처리 example

```
int init_module(void)
{
int err;
    /* Registered: pointer nad name          */
    err = register_this(ptr1, “skull”);
    if (err) goto fail_this;
    err = register_that(ptr2, “skull”);
    if (err) goto fail_that;
    err = register_those(ptr3, “skull”);
    if (err) goto fail_those;

    return 0;               /* Success */

    fail_those: unregister_that(ptr2, “skull”);          /* Reverse order      */
    fail_that: unregister_this(ptr1, “skull”);
    fail_this: return err;                               /* Return error       */
}
```

Skull Driver (VII)

- Cleanup_module error **처리**
  - Init_module**이 등록했던 모든 것을 원래의 상태로 되돌려 놓아야 한다**.

```
Void cleanup_module(void)
{
   unregister_those(ptr3, “skull”);
   unregister_that(ptr2, “skull”);
   unregister_this(ptr1, “skull”);
}
```

Skull Driver (VIII)

- **사용수 카운트**
  - System은 각 module을 사용하고 있는 프로세스의 수를 유지하고 있다.
    - Module을 안전하게 삭제할 수 있는지 검사하기 위함.
  - Macros (in linux/module.h)
    - MOD_INC_USER_COUNT: 현 module의 count를 증가시킨다
    - MOD_DEC_USE_COUNT: 현 module의 count를 감소시킨다
    - MOD_IN_USE: Count가 0이 아니면 true.
  - 사용수 count의 현재값은 /proc/modules의 각 모듈 항목의 세번째 field
    - floppy 45960 1 (autoclean) ; MemSize, ModCount
    - Ipv6 75164 -1 ; No use count

Skull Driver (IX)

- Module **삭제**
  - Rmmod **명령**
    - Call delete_module
      - Use_count**가 0이면 모듈안에 정의된** cleanup_module**을 호출한다**.
    - Cleanup_module
      - Module**이 등록한 모든 것을 제거**
      - Symbol table**을 제거**.

Skull Driver (X)

- Resource Usage
  - Module은 system resource를 사용하지 않고 자신의 과업을 수행할 수 없다.
  - Resource:
    - Memory, I/O port, interrupt, DMA channel, etc.
  - Memory
    - 획득: kmalloc (malloc과 동일. 우선권 인자)
      - 우선권: GFP_KERNEL
    - 해제: kfree (free와 동일)
  - I/O port
    - 각 port는 개별적인 역할을 가지고 있다.
    - 각 driver는 특정 port를 대상으로 작업할 필요가 있다.

Skull Driver (XI)

- 사용자 공간에서의 driver 작업
  - 장점
    - C library를 모두 link 할 수 있다. Driver는 외부 프로그램에 의존하지 않고 다양한 작업을 수행할 수 있다.
    - 일반적인 debugger가 driver code를 운영할 수 있다.
    - Driver가 정지한다면 간단히 kill 할 수 있다.
    - 사용자 memory는 kernel memory와는 달리 swap 될 수 있다.
    - 제대로 설계된 driver program은 device에 대한 동시 접근을 허용 할 수 있다.
  - 사용자 공간 driver의 예: X server
    - 사용가능한 hardware가 어떤 것인지 정확히 알고 있으며, 모든 X client에게 graphic resource를 제공한다.
    - Kernel로부터 hardware 제어에 책임이 있는 단일관리자로써 역할을 넘게 받은 server process를 구현한다.

Skull Driver (XII)

- 사용자 공간에서의 driver 작업 (cont'd)
  - 단점
    - 사용자 공간에서는 interrupt를 쓸 수 없다.
    - Memory 직접 접근은 /dev/mem에 mapping되어 있고, 특권 사용자 일 때만 가능하다.
    - 입출력 port에 대한 접근은 ioparm이나 iopl을 호출한 다음에만, 그리고 특권 사용자일 때만 가능하다.
    - Client와 hardware 간에 정보를 전달하거나 동작을 수행하기 위한 context switch가 필요하기 때문에 응답시간이 느리다.
    - Driver가 disk로 swap 되었다면 응답시간이 허용범위 이외로 커진다.
    - 가장 중요한 device들은 사용자 공간에서 처리할 수 없다.
  - 평범하지 않은 hardware를 다룰 경우 사용자 공간 software를 작성함으로써 일을 시작해도 좋다. 그후 kernel module에 이 software를 집어넣는 것이 고통스럽지 않은 작업이 될 것이다.

References

- Module & skull
  - Alessandro Rubini, “Linux Device Drivers”, O’Reilly, 1998.