M1522.000800
System Programming

- Lab. 2 Kernel Driver Lab. -

Assigned: Wed., March 25, Due: Tue., April 7, 11:59PM

Computer Systems and Platforms Laboratory
School of Computer Science and Engineering
Seoul National University

Before start…

- **It's time to start real system programming.**
- Before starting, we have to install a few tools in Linux Mint.
- **Commands**
  - *$ sudo apt-get update && sudo apt-get install build-essential*

What is A Kernel Module?

- A module is pieces of code that can be loaded and unloaded into the kernel upon demand.
- Can use privileged instructions without system calls, because a kernel module is loaded and executed within a kernel.
- **Module load / unload commands in Linux**
  - **Load** *$ insmod {module_name.ko}*
  - **Unload** *$ rmmod {module_name}*
  - **Module list** *$ lsmod*

Why Device Drivers?

- **Device Drivers** is one kind of kernel module, which provides the means to communicate with the (virtual / real) hardware.
- **Device Drivers**
  - **Character Device Drivers** : It could be called in user applications using a fixed length string without supporting a buffer.
  - **Block Device Drivers** : It has an inner buffer which is managed by the kernel. It communicates with applications using this buffer.
  - **Network Device Drivers** : It uses network protocols to communicate with others.

Why Device Drivers? (Cont.)

- **Each device driver has a major number and a minor number.**
- **for example: SATA Disk Drivers**

```
suwon@tux ~ $ ls -l /dev/sda*
brw-rw---- 1 root disk 8, 0 Mar 23 10:11 /dev/sda
brw-rw---- 1 root disk 8, 1 Mar 23 10:11 /dev/sda1
brw-rw---- 1 root disk 8, 2 Mar 23 10:11 /dev/sda2
brw-rw---- 1 root disk 8, 3 Mar 23 10:11 /dev/sda3
brw-rw---- 1 root disk 8, 4 Mar 23 10:11 /dev/sda4
suwon@tux ~ $
```

- **Major Number (Red square) :** The device specific number. Drivers has same major number if they use the same device.
- **Minor Number (Blue square) :** The driver identification number. Drivers has its own minor number within same device drivers.
- That is, each drivers has a unique combination of a major number and a minor number.

Character Device Drivers

- To register a character device driver in the kernel, the *init_module()* function must call the *register_chrdev()* function.

```
391 /*
392  * Initialize the module - Register the character device
393  */
394 int init_module()
395 {
396   int ret_val;
397   ret_val = register_chrdev(MAJOR_NUM, DEVICE_NAME, &Fops);
398
399   if (ret_val < 0) {
400     printk(KERN_ALERT "%s failed with %d\n",
401       "Sorry, registering the character device ", ret_val);
402     return ret_val;
403   }
```

- MAJOR_NUM : You must use a not-used major number not to intervene other devices.
  - $ cat /proc/devices -> prints all currently used major device number.

```
suwon@tux ~ $ cat /proc/devices
Character devices:
  1 mem
  4 /dev/vc/0
  4 tty
  4 ttyS
  5 /dev/tty
  5 /dev/console
```

Character Device Drivers (Cont.)

- DEVICE_NAME : The device name which will be printed as a device name in kernel.
- Fops : The device driver's function pointer structure.

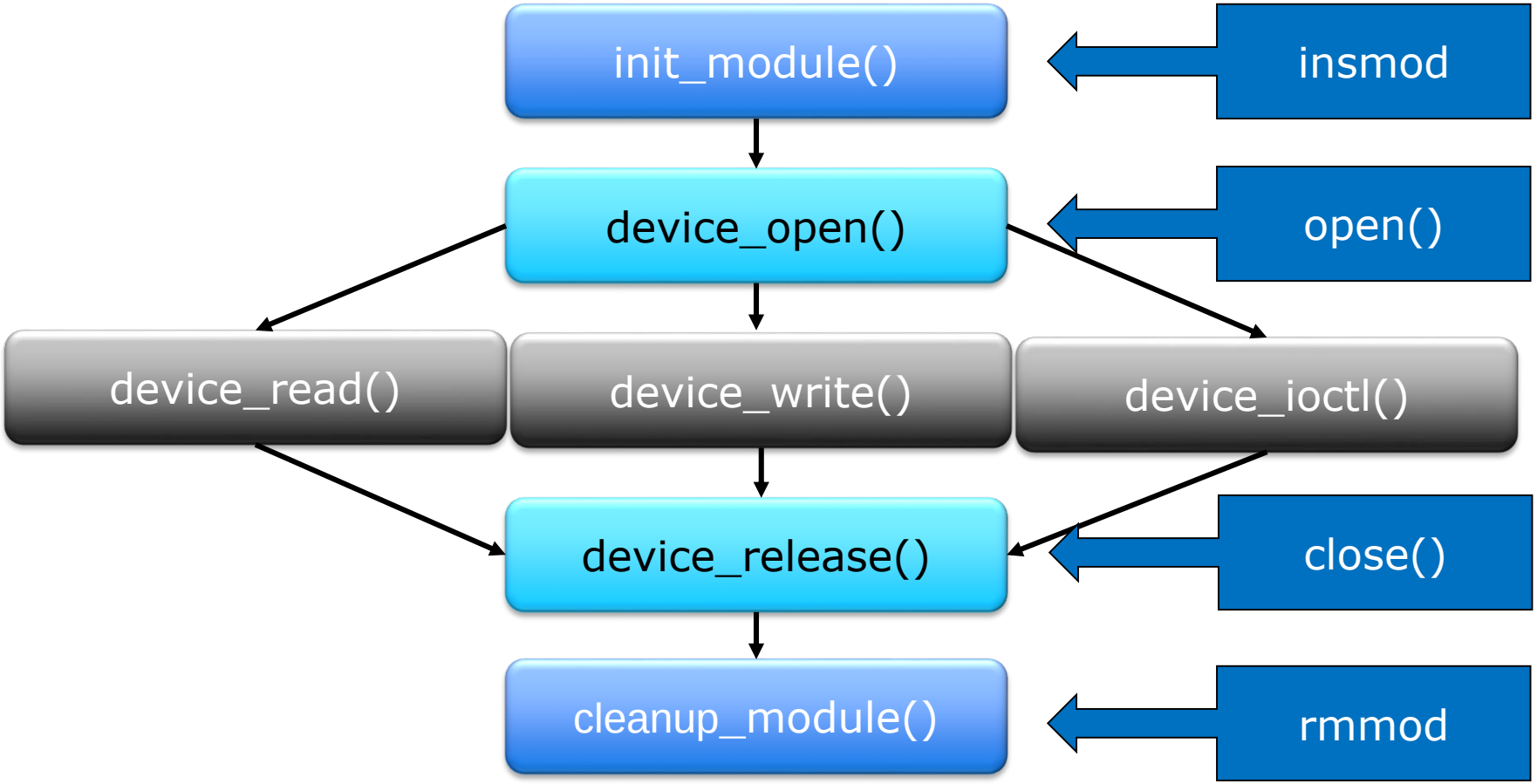
```
376 /*
377  * This structure will hold the functions to be called
378  * when a process does something to the device we
379  * created. Since a pointer to this structure is kept in
380  * the devices table, it can't be local to
381  * init_module. NULL is for unimplemented functions.
382  */
383 struct file_operations Fops = {
384   .read = device_read,
385   .write = device_write,
386   .unlocked_ioctl = device_ioctl,
387   .open = device_open,
388   .release = device_release,
389 };
390
```

- The device driver must be opened in a user application to communicate with it (remember everything is a "file" in Unix)

```
151   file_desc = open(DEVICE_FILE_NAME, 0);
152
153   if (file_desc < 0) {
154     printf("Error : Can't open device file : %s with %d\n",
155       DEVICE_FILE_NAME, errno);
156     exit(1);
157   }
158
```

Character Device Drivers (Cont.)

- Device Driver Progress Flow



Character Device Drivers (Cont.)

- Functions that must be provided by each driver
  - int init_module(void)
  - void cleanup_module(void)
  - static int device_open(structure inode *, struct file *)
  - static int device_release(struct inode *, struct file *)
  - static ssize_t device_read(struct file *, char __user *, size_t, loff_t)
  - static ssize_t device_write(struct file *, const char __user *, size_t, loff_t)
  - int device_ioctl(struct file *, unsigned int, unsigned long)

Assignment and Grading

1. (Warming up) Implement a kernel module that outputs the list of all parent processes up to the root when called through ioctl(). (30pts)

2. (Get serious) Implement a kernel module to manage the processor's performance monitoring unit (PMU) (60pts)

   (Alternative to 2) Implement a kernel module providing functionality of your choice (*pending prior approval by the TAs!*)

3. (Documentation is important) Report (10pts)

Assignment 1: Parent Process Tree Device

- Print all parent processes from the device calling process to the root.

- Example output

```
swapper/0(0)
\-init(1)
  \-sshd(16712)
    \-sshd(7000)
      \-sshd(7010)
        \-bash(7015)
          \-caller(7101)
```

- Hint: useful system calls / data structures
  - getpid, task_struct, parent, pid

Assignment 2: PMU Management Device

1. Read the documentation on the Intel PMU (references at the end of this assignment)
2. Implement the following functionality:
   - Reset PMU counter
   - Select PMU event to monitor
   - Start PMU counter
   - Read PMU counter
   - Read the TSC (Time Stamp Counter) register
3. Hints:
   - program the PMU using the MSR (Model Specific Register).
   - use inline assembly programming within driver source
   - Instructions / Registers (see Intel Instruction Set Reference on how to use them)

     RDMSR, WRMSR, RDTSC, PERFEVTSEL, PERFCTR
   - the PMU is not virtualized by default in VMWare

     Go to Virtual Machine Settings > Processors and check ‘Virtualize CPU performance Counters’

Assignment 2: PMU Management Device

- Example output

  (output format is free, but should be functional (i.e., easy to read & understand))

```
before wrmsr: eax=0x00530080, ecx=0x00000187
after wrmsr: eax=0x00530080, ecx=0x00000187
Instruction Fetch Unit
delay by access an array
stop the counter, eax=0x00130080
rdmsr: ecx=0x000000c2
rdmsr: Instruction fetches is 158696291748 (0x24f3093da4)
time stamp : 2311702538
```

Assignment 2: Your Own Topic

- You can choose your own topic to implement using a kernel driver.
- Before you start, you have to get approval from the professor or the TAs
  - Submit short description and functionality to be implemented by email
  - Deadline: Friday, March 27
- After getting approval you can start implementing it.

HOWTO: Makefile

- **‘make’** is one of the stand-alone program which keeps track of how to make your program from the sources (hence the name).
- Makefile is the file that contains the instructions for ‘make’.
- *$ make* -> automatically building your program from sources.

```
suwon@tux ~/Desktop/SystemProgramming/labs/kernellab/src $ make
gcc -o module_trigger -I./module module_trigger.c
cd benchmarks; make; mv *_benchmark ../
make[1]: Entering directory `/home/suwon/Desktop/SystemProgramming/labs/kernellab/src/benchmarks'
gcc -o cpu_benchmark cpu_benchmark.c -lpthread
cpu_benchmark.c: In function ‘thread_initialize’:
cpu_benchmark.c:28:11: warning: cast from pointer to integer of different size [-Wpointer-to-int-cast]
cpu_benchmark.c:30:3: warning: format ‘%d’ expects argument of type ‘int’, but argument 3 has type ‘pthread_t’ [-Wformat]
cpu_benchmark.c: In function ‘thread_calculate’:
cpu_benchmark.c:52:11: warning: cast from pointer to integer of different size [-Wpointer-to-int-cast]
cpu_benchmark.c:54:3: warning: format ‘%d’ expects argument of type ‘int’, but argument 3 has type ‘pthread_t’ [-Wformat]
cpu_benchmark.c: In function ‘main’:
cpu_benchmark.c:75:59: warning: cast to pointer from integer of different size [-Wint-to-pointer-cast]
cpu_benchmark.c:84:58: warning: cast to pointer from integer of different size [-Wint-to-pointer-cast]
```

HOWTO: Makefile (Cont.)

- Simple example

```
1  all: module_trigger benchmark driver
2
3  module_trigger:
4    gcc -o module_trigger -I./module module_trigger.c
5
6  benchmark:
7    cd benchmarks; make; mv *_benchmark ../
8
9  driver:
10   cd module; make
11
12 clean:
13   rm module_trigger *_benchmark; cd module; make clean
```

- all : …
  - Defines targets. Each targets must be defined in the followings.
  - When you just command ‘make’, it produce all targets in <all>.
- clean:
  - Defines typically removal all built object files.
  - You can call it with commanding ‘make clean’

HOWTO: Makefile (Cont.)

- Module Compile Example

```
1 obj-m := prof_dev.o
2
3 all:
4   make -C /lib/modules/$(shell uname -r)/build M=$(PWD) -I. modules
5
6 clean:
7   make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

- Kernel module is not compiled with general ‘gcc’. It needs for kernel specific compile tools.
- The example shows a simple makefile for kernel module build.
- You should change the module name in the red box.

HOWTO: Makefile (Cont.)

- **You must implement makefile supporting the following features.**
  - **<all>**
    - Your makefile must build one executable binary file and other object files from your sources when I command just ‘make’.
  - **<clean>**
    - Your makefile must clean up all your results from ‘make’ when I command ‘make clean’.
  - **<submission>**
    - Your makefile must make one tarball file for your project files when I command ‘make submission’.

Logistics

- **This is a team-assignment of two students**
  - You can do this by yourself, but we recommend you to make a team with your colleagues.
- Submit the tarball file containing your source files and your report (in PDF format)
  - Include the ‘Makefile’ to compile the module from your source files.
  - by email to the TA’s email address. (sysprog@casp.snu.ac.kr)
  - Due Date : Tuesday, April 7, 2015, 11:59 pm
- Hint: creating a tarball in Linux:
  - *$ tar cvf {your_student_id_your_name}.tar {your_working_folder}*
  - Follow the naming convention:
    - *Team:*
      - *$ tar cvf 2015-11111_KimBob_2014-11111_CheonGook.tar ./kernellab*
    - *Alone:*
      - *$ tar cvf 2015-11111_ParkHonja.tar ./kernellab*

Reference

- **Linux Kernel Module Programming Guide**
  - http://www.tldp.org/LDP/lkmpg/2.6/html/
- **Inline Assembly Guide**
  - https://wiki.kldp.org/wiki.php/DocbookSgml/GCC_Inline_Assembly-KLDP
- **Intel 64 & IA-32 Architectures Software Developer's Manual Vol. 2 – Instruction Set Reference, A-Z**
  - http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-instruction-set-reference-manual-325383.pdf
- **Intel Architecture Software Developer's Manual Vol. 3 – System Programming**
  - https://communities.intel.com/servlet/JiveServlet/previewBody/5061-102-1-8118/Pentium_SW_Developers_Manual_Vol3_SystemProgramming.pdf
- **Implementing Reading PMU using Inline Assembly Guide**
  - http://www.mindfruit.co.uk/2012/11/intel-msr-performance-monitoring-basics.html
- **Makefile Guide**
  - https://www.cs.duke.edu/~ola/courses/programming/Makefiles/Makefiles.html

Thank you

Any Questions?