IKO31204
Pemrograman Sistem
Jilid 6: Linux Kernel Module (II)

Fakultas Ilmu Komputer - Universitas Indonesia
Sep 2012

topik

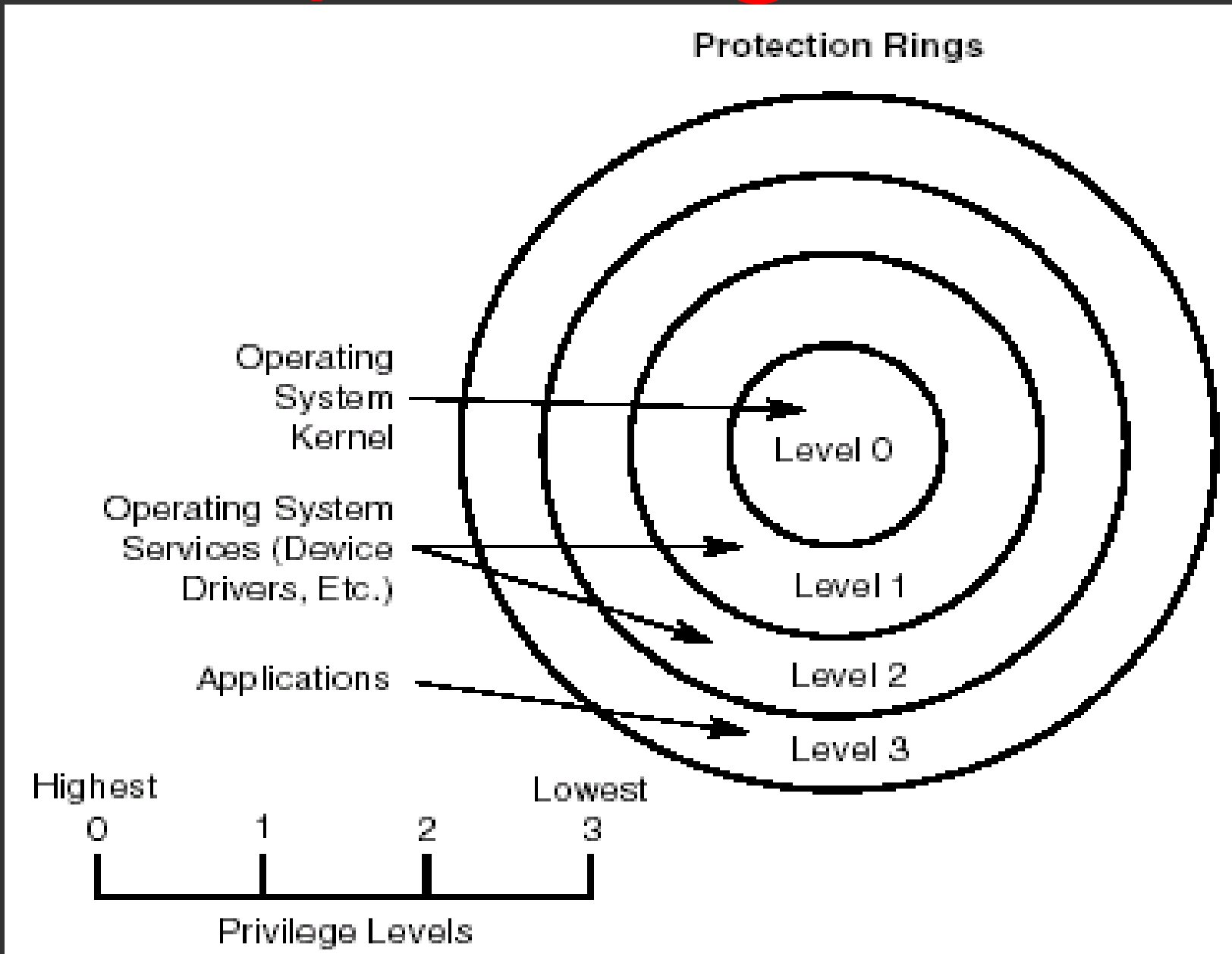
> Simple hello world module
> __init and __exit Macros
> Compiling kernel modules
> Licensing & Module Docs
> Modules Spanning Multiple Files
> Passing Arguments to Module
> Cross Version Kernel Module

referensi

The Linux Kernel Module Programming Guide (Google it, now !), Chapter 2 & 5

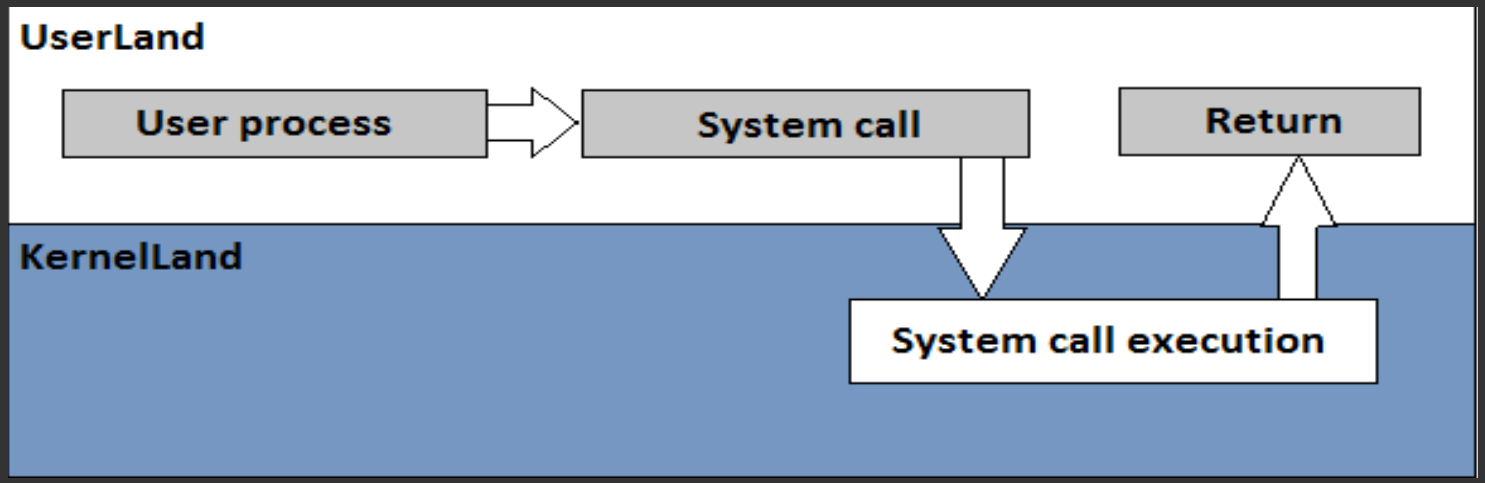
Implementing a System Call on Linux 2.6 for i386 (*Google it, now !*)

privilege level



module vs program

Item	module	program
Entry code	method init_module dan cleanup_module	method main()
functions / library	system calls, cth: /proc/kallsyms	standard C library, cth: printf (HINT: gunakan strace)
resource	ring 0	ring 3
code space	kernel space / kernel land memory (default 1/4)	user space / user land memory (default 3/4)



hello module

Linux Kernel Module Programming Guide, Ch 2

komponen module

fungsi esensial:
1. start function
2. end function

fungsi tambahan:
1. module parameter
2. module description

(HINT: pelajari dari cth pd buku referensi)

hello-1 module

```
/*
* hello–1.c – The simplest kernel module.
*/

#include <linux/module.h>           /* Needed by all modules */
#include <linux/kernel.h>        /* Needed for KERN_INFO */

int init_module(void)   {
      printk(KERN_INFO "Hello world 1.\n");
      /*    A non 0 return means init_module failed;
            module can't be loaded. */
      return 0;
}
void cleanup_module(void)   {
      printk(KERN_INFO "Goodbye world 1.\n");
}
```

Hello Summary

> module **init** is called when module is loaded

> module **cleanup** is called when module is unloaded

printk

> The server can't use stdlib due to userspace/kernel space issues
> Most of C library is implemented in the kernel
> **printk** is **printf** for kernel programs.
> Ada 8 (delapan) kemungkinan untuk mencetak info dengan printk

KERN_EMERG, KERN_ALERT, KERN_CRIT, KERN_ERR, KERN_WARNING, KERN_NOTICE, KERN_INFO, KERN_DEBUG

printk

Pelajari Debugging by Printing pada

http://www.makelinux.net/ldd3/chp-4-sect-2

Apa perbedaannya antara ke-delapan log-level pada printk ?

Mengapa diperlukan perbedaan antara log-level pada printk ?

hello-2 module

```
/*
* hello–2.c – Demonstrating the module_init() and module_exit() macros.
* This is preferred over using init_module() and cleanup_module().
*/
#include <linux/module.h>          /* Needed by all modules */
#include <linux/kernel.h>          /* Needed for KERN_INFO */
#include <linux/init.h>            /* Needed for the macros */

static int __init hello_2_init(void)   {
       printk(KERN_INFO "Hello, world 2\n");
       return 0;
}

static void __exit hello_2_exit(void)     {
       printk(KERN_INFO "Goodbye, world 2\n");
}

module_init(hello_2_init);
module_exit(hello_2_exit);
```

Macros

__init macro and **__exit** macro tells the kernel that these function can be referenced by **module_init** and **module_exit**

This allows to free kernel memory that is used only at init. And ease of use for huge programming platform

Kompilasi dgn Makefile

> Hampir sama seperti berkas Makefile untuk kompilasi Aplikasi User Level
> Dalam kompilasi kernel, seluruh library yang diperlukan berada di dalam Kernel Source Code
> Makefile melakukan otomatisasi kompilasi
> Profile otomatisasi didefinisikan di dalam berkas bernama Makefile
> Module hasil ber-ekstensi .ko (untuk kernel versi 2.6 ke-atas)

Makefile

```
obj-m += hello-1.o
obj-m += hello-2.o
all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

obj-m
Adalah object-object modul yang ingin di compile
all: dan **clean:**
Menunjukkan profil otomatisasi Makefile

Referensi Makefile:
/usr/src/
{kernelsource}/linux/Documentation/kbuild/makefiles.txt

Makefile

```
obj-m += hello-1.o
obj-m += hello-2.o
all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

$(shell uname -r)
Memberikan hasil berupa versi kernel yang saat ini digunakan

all: dan **clean:**
Menunjukkan profil otomatisasi Makefile

Referensi Makefile:

Percobaan

mkdir percobaan
cd percobaan
vi hello-1.c
(copy paste seluruh hello-1.c di atas)

vi hello-2.c
(copy paste seluruh hello-2.c di atas)

vi Makefile
(copy paste seluruh isi Makefile di atas)

make
insmod hello-1.ko
insmod hello-2.ko

Licensing & Docs

Setiap Modul dapat diberikan informasi pembuat, tujuan, dependency, informasi argumen saat inisialisasi/loading

Informasi dapat dilihat dengan perintah

 # modinfo <nama modul>

hello-4 module

```
/*  hello-4.c - Demonstrates module documentation. */
#include <linux/module.h>     /* Needed by all modules */
#include <linux/kernel.h>     /* Needed for KERN_INFO */
#include <linux/init.h>       /* Needed for the macros */
#define DRIVER_AUTHOR "Peter Jay Salzman <p@dirac.org>"
#define DRIVER_DESC   "A sample driver"

/* Get rid of taint message by declaring code as GPL. */
MODULE_LICENSE("GPL");

/* Or with defines, like this: */
MODULE_AUTHOR(DRIVER_AUTHOR);  /* Who wrote this module? */
MODULE_DESCRIPTION(DRIVER_DESC);     /* What does this module do */

/* This module uses /dev/testdevice.  The MODULE_SUPPORTED_DEVICE
*   macro mightbe used in the future to help automatic configuration of
*   modules, but is currently unused other than for documentation purposes.
*/
MODULE_SUPPORTED_DEVICE("testdevice");
/* ... bersambung ke halaman selanjutnya */
```

hello-4 module

```
/* ..... sambungan dari halaman sebelumnya ..... */

static int __init init_hello_4(void)
{
	printk(KERN_INFO "Hello, world 4\n");
	return 0;
}

static void __exit cleanup_hello_4(void)
{
	printk(KERN_INFO "Goodbye, world 4\n");
}

module_init(init_hello_4);
module_exit(cleanup_hello_4);
```

Spanning Multiple Files

Kadang-kadang diperlukan pembagian *source code* pada beberapa berkas.

Alasannya, antara lain:

- berkas dibagi berdasarkan fungsi dari *source code* nya
- agar mudah u/ pembagian tugas *coding*
- seringnya perubahan pada bagian tertentu
- dll

start-stop module

```
/*
 *  start.c - Illustration of multi filed modules
 */

#include <linux/kernel.h>/* We're doing kernel work */
#include <linux/module.h>     /* Specifically, a module */

int init_module(void)
{
        printk(KERN_INFO "Hello, world - this is the kernel speaking\n");
        return 0;
}
```

start-stop module

```
/*
 *  stop.c - Illustration of multi filed modules
 */

#include <linux/kernel.h>/* We're doing kernel work */
#include <linux/module.h>     /* Specifically, a module  */

void cleanup_module()
{
       printk(KERN_INFO "Short is the life of a kernel module\n");
}
```

start-stop Makefile

```
obj-m += startstop.o
startstop-objs := start.o stop.o

all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

obj-m += startstop.o

>> hasil/nama akhir objek modul

startstop-objs := start.o stop.o

>> link antara objek start dan stop
dalam modul “startstop”

Passing command line

To allow arguments to be passed to your module, declare the variables that will take the values of the command line arguments as global and then use the module_param() macro. (defined in linux/moduleparam.h)

hello-5 module

```
/*
 *  hello-5.c - Demonstrates command line argument passing to a module.
 */
#include <linux/module.h>
#include <linux/moduleparam.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/stat.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Peter Jay Salzman");

static short int myshort = 1;
static int myint = 420;
static long int mylong = 9999;
static char *mystring = "blah";
static int myintArray(2) = { -1, -1 };
static int arr_argc = 0;

/* ... lanjut ke halaman selanjutnya .... */
```

hello-5 module

```
/* ... terusan ... */
/*
 * module_param(foo, int, 0000)
 * The first param is the parameters name
 * The second param is it's data type
 * The final argument is the permissions bits,
 * for exposing parameters in sysfs (if non-zero) at a later stage.
 */
module_param(myshort, short, S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP);
MODULE_PARM_DESC(myshort, "A short integer");
module_param(myint, int, S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH);
MODULE_PARM_DESC(myint, "An integer");
module_param(mylong, long, S_IRUSR);
MODULE_PARM_DESC(mylong, "A long integer");
module_param(mystring, charp, 0000);
MODULE_PARM_DESC(mystring, "A character string");
```

hello-5 module

```
/*
 * module_param_array(name, type, num, perm);
 * The first param is the parameter's (in this case the array's) name
 * The second param is the data type of the elements of the array
 * The third argument is a pointer to the variable that will store the number
 * of elements of the array initialized by the user at module loading time
 * The fourth argument is the permission bits
 */
module_param_array(myintArray, int, &arr_argc, 0000);
MODULE_PARM_DESC(myintArray, "An array of integers");
```

hello-5 module

```
static int __init hello_5_init(void) {
	int i;
	printk(KERN_INFO "Hello, world 5\n=============\n");
	printk(KERN_INFO "myshort is a short integer: %hd\n", myshort);
	printk(KERN_INFO "myint is an integer: %d\n", myint);
	printk(KERN_INFO "mylong is a long integer: %ld\n", mylong);
	printk(KERN_INFO "mystring is a string: %s\n", mystring);
	for (i = 0; i < (sizeof myintArray / sizeof (int)); i++) {
		printk(KERN_INFO "myintArray(%d) = %d\n", i, myintArray(i));
	}
	printk(KERN_INFO "got %d arguments for myintArray.\n", arr_argc);
	return 0;
}
static void __exit hello_5_exit(void) {
	printk(KERN_INFO "Goodbye, world 5\n");
}

module_init(hello_5_init);
module_exit(hello_5_exit);
```

Coba hello-5

```
# insmod hello-5.ko mystring="bebop" mybyte=255 myintArray=-1
mybyte is an 8 bit integer: 255
myshort is a short integer: 1
myint is an integer: 20
mylong is a long integer: 9999
mystring is a string: bebop
myintArray is -1 and 420

# rmmod hello-5
Goodbye, world 5

# insmod hello-5.ko mystring="supercalifragilisticexpialidocious"  mybyte=256
   myintArray=-1,-1
mybyte is an 8 bit integer: 0
myshort is a short integer: 1
myint is an integer: 20
mylong is a long integer: 9999
mystring is a string: supercalifragilisticexpialidocious
myintArray is -1 and -1
```

Coba hello-5

```
# rmmod hello-5
Goodbye, world 5

# insmod hello-5.ko mylong=hello
hello-5.o: invalid argument syntax for mylong: 'h'
```

Cross Version Kernel Module

Sangat disarankan untuk meng-kompile modul sesuai dengan Versi Kernelnya.
Jika Cross Version dilakukan, kita (mungkin) perlu "memaksa" saat proses insmod dan rmmod.
Alasan Cross Version:
> dapat memudahkan saat proses development (tidak sering reboot utk ganti kernel)
> kernel sudah terlanjur dicompile & diinstall
> tidak boleh reboot server, dll
Pelajari:
http://guidespratiques.traduc.org/guides/vo/lkmpg/2.6/html/x380.html

tanya jawab