Kernel Module Design Basics

Advanced Embedded Linux Development

with **Dan Walkes**

University of Colorado **Boulder**

Learning objectives:

Kernel Module Dependencies

Kernel Symbol Table

Module Init, Exit and Error Handling

Module Parameters

Version Dependency

- Your driver is only allowed to load against the kernel for which it was compiled
  - vermagic
  - target kernel, compiler version, processor, configuration variables must match
  - insmod: “Error inserting <driver.ko>: -1 Invalid module format”
  - Rebuild with the appropriate kernel source

Version Dependency

- APIs break between kernel revisions
  - KERNEL_VERSION macro can be used to handle compatibilities.
  - See an example use at https://github.com/cu-ecen-5013/ldd3-samples/commit/d6dc003dd4526549abe2718a825e1ef64861ca3b

Platform Dependency

- How do I keep up with all possible variations?
- Option 1:
  - Release under a GPL compatible license
  - Ideally add to the mainline kernel

Linux Device Drivers 3rd Edition Chapter 2
https://github.com/dell/dkms
https://www.linuxjournal.com/article/6896

Platform Dependency

- How do I keep up with all possible variations?
- Option 2:
  - Distribute in source form with scripts to compile
  - Not mentioned in the book: Dynamic Kernel Module Support (DKMS). see https://www.linuxjournal.com/article/6896

Linux Device Drivers 3rd Edition Chapter 2
https://github.com/dell/dkms
https://www.linuxjournal.com/article/6896

Platform Dependency

- How do I keep up with all possible variations?
- Option 3 (for embedded systems - not in book)
  - Build for a single target kernel configuration

Linux Device Drivers 3rd Edition Chapter 2
https://github.com/dell/dkms
https://www.linuxjournal.com/article/6896

Kernel Symbol Table

- Your module can export symbols for use by other modules
  - Recall modprobe discussion earlier
- “Module stacking”

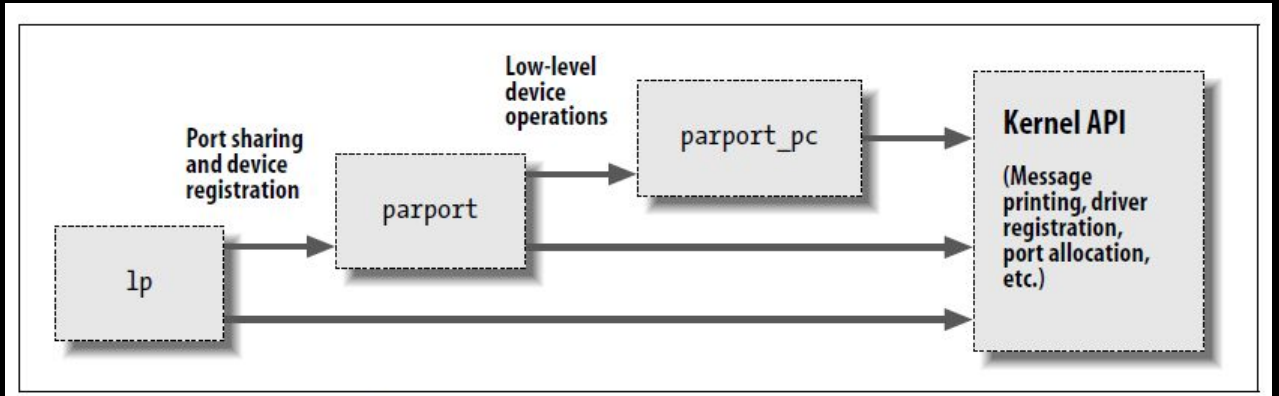

Figure 2-2. Stacking of parallel port driver modules

Linux Device Drivers 3rd Edition Chapter 2

Kernel Symbol Table

```
EXPORT_SYMBOL(name);
EXPORT_SYMBOL_GPL(name);
```

- What does EXPORT_SYMBOL_GPL do?
  - Makes the symbol available to GPL licensed modules only.
  - See MODULE_LICENSE discussed earlier

```
/*
 * $Id: hello.c,v 1.5 2004/10/26 03:32:21 corbet Exp $
 */
#include <linux/init.h>
#include <linux/module.h>
MODULE_LICENSE("Dual BSD/GPL");
```

Kernel Module Preliminaries

- <linux/module.h>
  - Definitions and symbols needed by loadable modules
- <linux/init.h>
  - Initialization and cleanup functions
- MODULE_LICENSE, MODULE_AUTHOR, MODULE_DESCRIPTION, MODULE_VERSION

Module Initialization

- Should be static - Why?
  - Not meant to be used outside this file.
- What's with the __init label?
  - Module loader can drop memory associated with this function after loading

```
static int __init initialization_function(void)
{
    /* Initialization code here */
}
module_init(initialization_function);
```

Module Initialization

- What do you do in your init function?
  - Register with kernel facilities
    - devices, filesystems, crypto transforms, sysfs, proc, etc
    - Initialize data structures
- Where should your data structures be located?
  - Allocated memory, not on the stack!

```
static int __init initialization_function(void)
{
    /* Initialization code here */
}
module_init(initialization_function);
```

Linux Device Drivers 3rd Edition Chapter 2

Module Exit

- What would the exit function do?
  - Unregisters interfaces registered with init
    - Typically in reverse order used to register them
  - Frees memory allocated in init

```
static void __exit cleanup_function(void)
{
    /* Cleanup code here */
}

module_exit(cleanup_function);
```

Module Exit

- Why use __exit?
  - If your code is built directly into the kernel, or not allowed to be unloaded, can be discarded.

```
static void __exit cleanup_function(void)
{
    /* Cleanup code here */
}

module_exit(cleanup_function);
```

Error Handling

```
int __init my_init_function(void)
{
    int err;

    /* registration takes a pointer and a 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");
  fail_that: unregister_this(ptr1, "skull");
  fail_this: return err; /* propagate the error */
 }
```

- Most registrations can fail.
- Important to check for and handle failures
- goto is useful and widely used for this

Module Parameters

```
static char *whom = "world";
static int howmany = 1;
module_param(howmany, int, S_IRUGO);
module_param(whom, charp, S_IRUGO);
```

```
Mar  3 08:25:50 aesd-VirtualBox kernel: [90445.968923] (0) Hello, Mom
Mar  3 08:25:50 aesd-VirtualBox kernel: [90445.968926] (1) Hello, Mom
Mar  3 08:25:50 aesd-VirtualBox kernel: [90445.968927] (2) Hello, Mom
Mar  3 08:25:50 aesd-VirtualBox kernel: [90445.968928] (3) Hello, Mom
Mar  3 08:25:50 aesd-VirtualBox kernel: [90445.968929] (4) Hello, Mom
Mar  3 08:25:50 aesd-VirtualBox kernel: [90445.968930] (5) Hello, Mom
Mar  3 08:25:50 aesd-VirtualBox kernel: [90445.968931] (6) Hello, Mom
Mar  3 08:25:50 aesd-VirtualBox kernel: [90445.968932] (7) Hello, Mom
Mar  3 08:25:50 aesd-VirtualBox kernel: [90445.968933] (8) Hello, Mom
Mar  3 08:25:50 aesd-VirtualBox kernel: [90445.968934] (9) Hello, Mom
```

- Define with module_param(var,type,permissions)
  - S_IRUGO - readable, not changeable
  - S_IRUGO | S_IWUSR - writable by root

```
aesd@aesd-VirtualBox:~/ldd3/misc-modules$ sudo insmod ./hellop.ko howmany=10 whom="Mom"

aesd@aesd-VirtualBox:~/ldd3/misc-modules$ cat /sys/module/hellop/parameters/howmany
10
aesd@aesd-VirtualBox:~/ldd3/misc-modules$ cat /sys/module/hellop/parameters/whom
Mom
```

Linux Device Drivers 3rd Edition Chapter 2
https://github.com/cu-ecen-5013/ldd3/blob/master/misc-modules/hellop.c