Brief Overview of Device Drivers

Glenn Elliott

LITMUS^{RT} Lunch

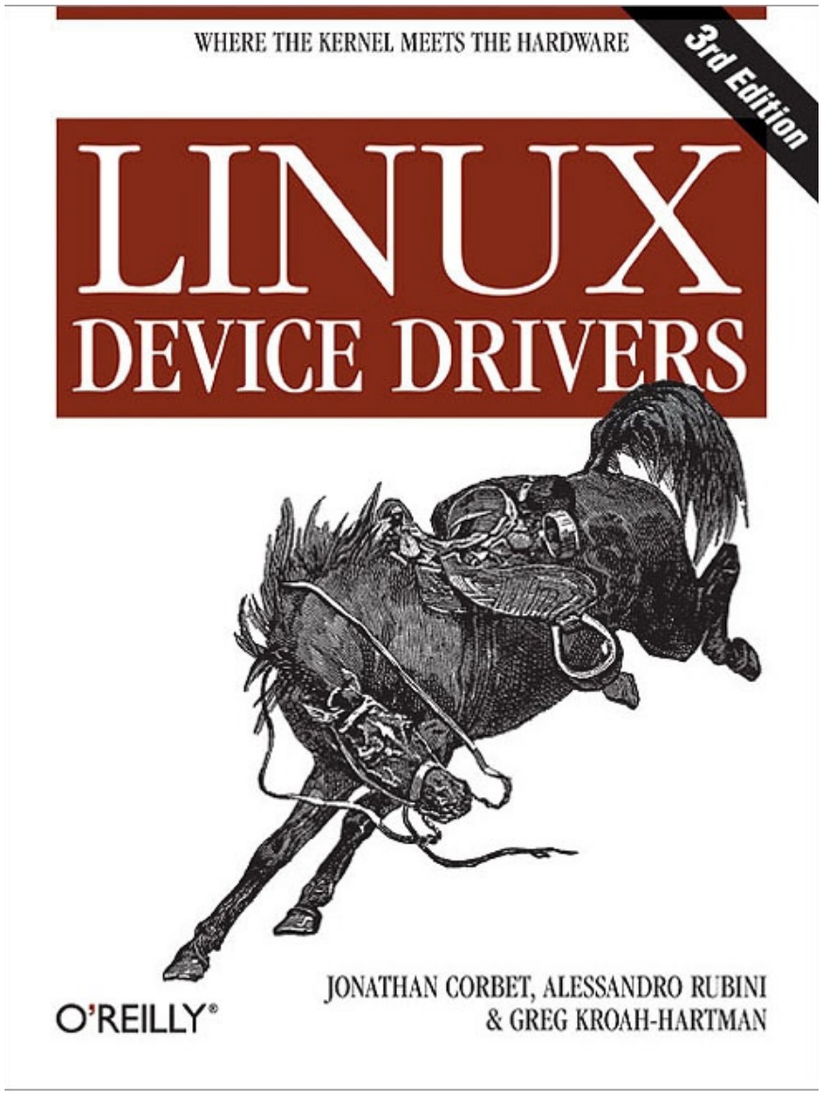
October 25, 2010

Information

- Free online copy (w/ kernel API updates):

  http://lwn.net/Kernel/LDD3/

- 600+ pages



What is a device driver?

- Usually a module in the kernel
- Conforms to a standard API set by the kernel to provide access to users
- Translates API-defined operations into device-specific operations

Types of Device Drivers

- Char Device
  - Stream of bytes
  - Sequential access (though back and forth may also be occasionally used)
- Block Device
  - Can host a filesystem
  - High performance (I/O scheduling)
- Network Device
  - Packet transmission
  - Interrupt optimization

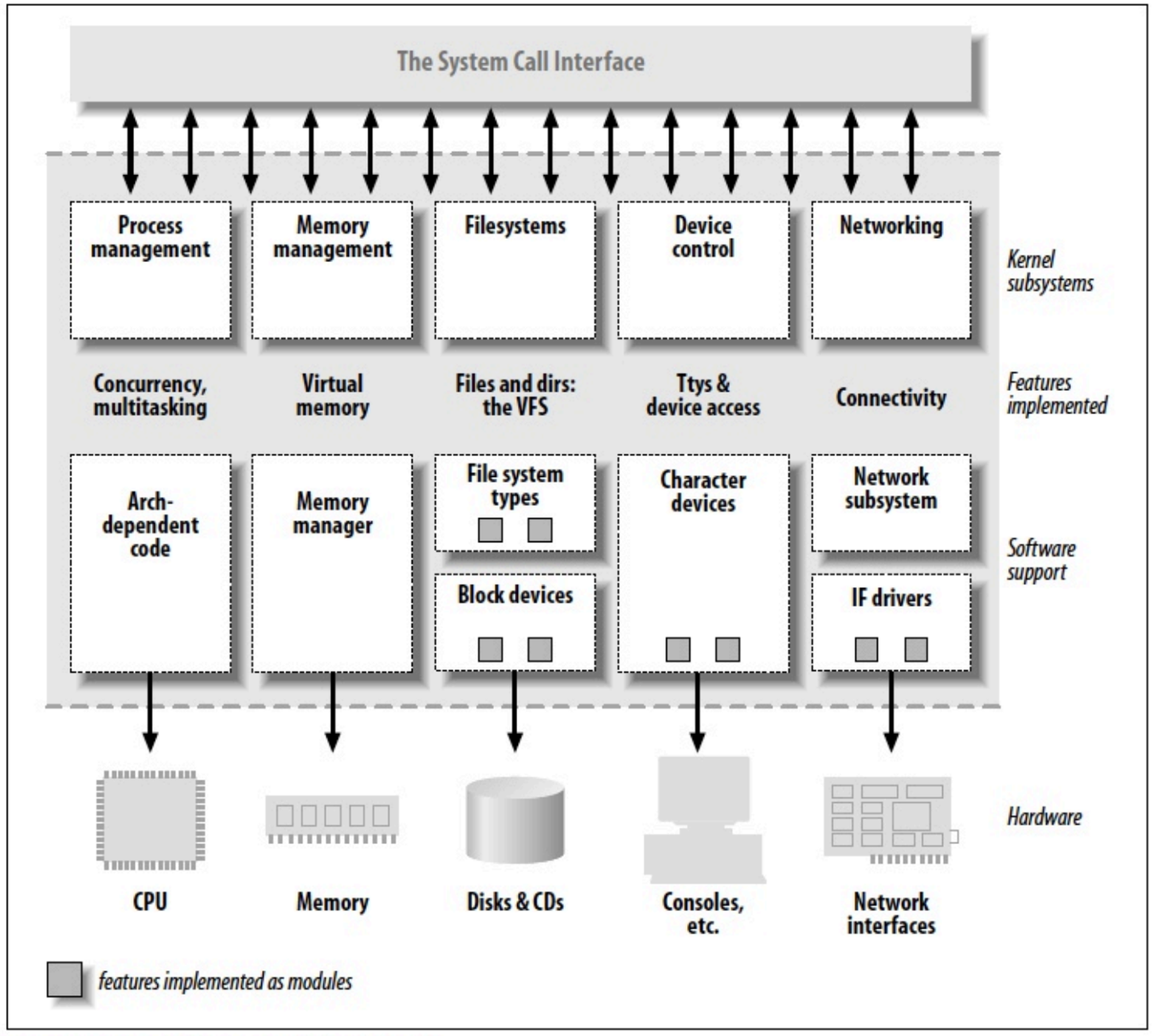


Figure 1-1. A split view of the kernel

System Structure

Modules

- Constraints on kernel-space drivers (as modules):
  - module_init()/module_exit()
    - must be reentrant (though driver define concurrent operation)
- May be stacked to implement complex systems

```c
#include <linux/init.h>
#include <linux/module.h>
MODULE_LICENSE("Dual BSD/GPL");

static int hello_init(void)
{
    printk(KERN_ALERT "Hello, world\n");
    return 0;
}

static void hello_exit(void)
{
    printk(KERN_ALERT "Goodbye, cruel world\n");
}

module_init(hello_init);
module_exit(hello_exit);
```

Module init()/exit()

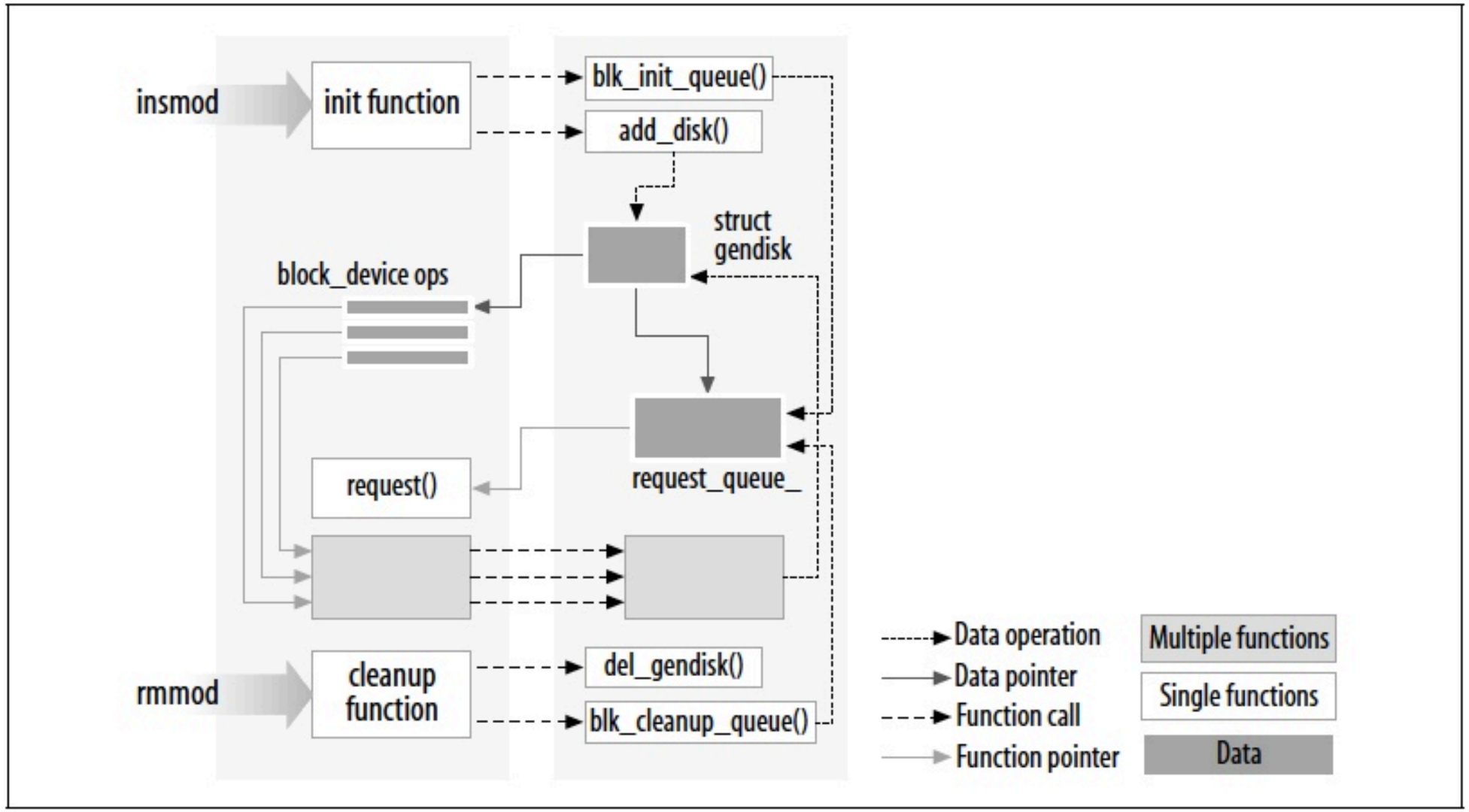


Figure 2-1. Linking a module to the kernel

Block Driver Module

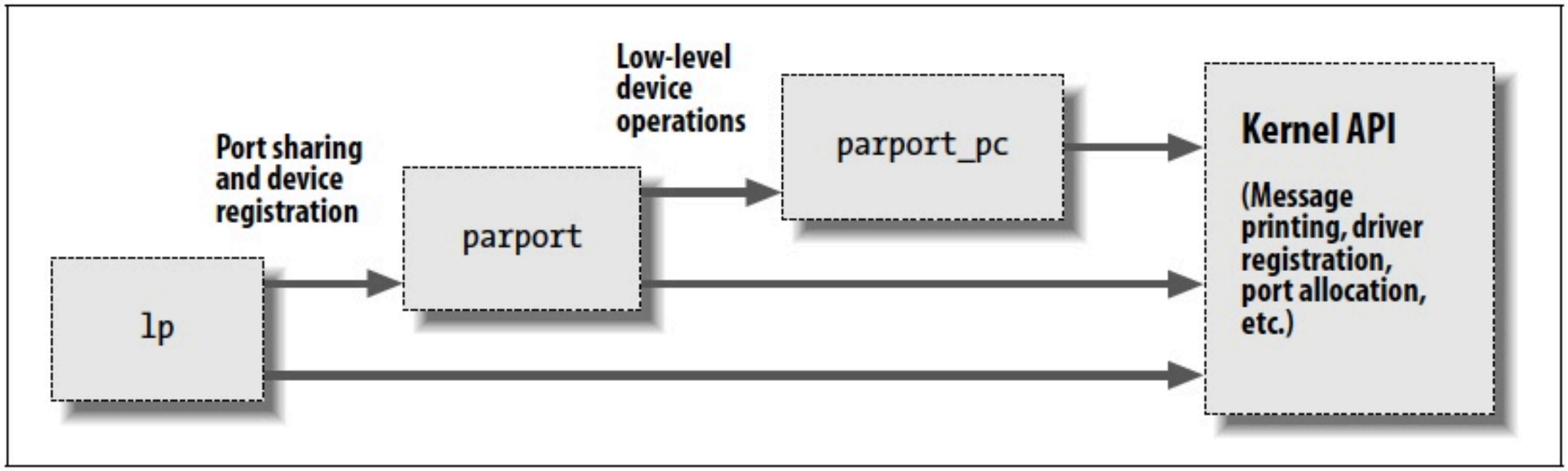


Figure 2-2. Stacking of parallel port driver modules

Stackable Drivers: printer (lp) on top of parallel port

Character Drivers

- If it's not a disk and it's not a network interface, it's probably a character device.
- Properties:
  - Does not require traditional I/O scheduling
  - Data streams
- Litmus Examples: TRACE() & FeatherTrace

Character Drivers

- Drivers register themselves with the kernel
  - “Major” number maps device to driver.
  - “Minor” number maps to device.
- Device handle appears in /dev

Character Drivers

- `int register_chrdev_region(dev_t first, unsigned int cout, char *name)`
  - Device numbers must be known apriori
- `int alloc_chrdev_region(dev_t *dev, unsigned int firstminor, unsigned int count, char *name)`
  - Let the kernel pick device numbers
- `void unregister_chrdev_region(dev_t first, unsigned int count)`
  - Free device numbers

```
crw-rw-rw-    1 root     root       1,   3 Apr 11  2002 null
crw-------    1 root     root      10,   1 Apr 11  2002 psaux
crw-------    1 root     root       4,   1 Oct 28 03:04 tty1
crw-rw-rw-    1 root     tty        4,  64 Apr 11  2002 ttyS0
crw-rw----    1 root     uucp       4,  65 Apr 11  2002 ttyS1
crw--w----    1 vcsa     tty        7,   1 Apr 11  2002 vcs1
crw--w----    1 vcsa     tty        7, 129 Apr 11  2002 vcsa1
crw-rw-rw-    1 root     root       1,   5 Apr 11  2002 zero
```

‘ls -l’ on /dev

Character Drivers

- Char drivers must interface with users through a well defined API.
  - llseek()
  - read()
  - aio_read()
  - write()
  - aio_write()
  - readdir()
  - poll()
  - ioctl() **- for direct device control**
  - mmap()
  - open()
  - flush()
  - release()
  - fsync()
  - aio_fsync()
  - fasync()
  - lock()
  - readv()
  - writev()
  - sendfile()
  - sendpage()
  - get_unmapped_area()
  - check_flags()
  - dir_notify()

```
struct file_operations scull_fops = {
    .owner =    THIS_MODULE,
    .llseek =   scull_llseek,
    .read =     scull_read,
    .write =    scull_write,
    .ioctl =    scull_ioctl,
    .open =     scull_open,
    .release =  scull_release,
};
```

Example Char Driver Operations

Block Drivers

- Character Drivers can be used to support filesystems, though performance would be terrible
- Block Drivers:
  - Centered on performance
  - API is centered on “requests”
  - Transfer blocks of data

Block Drivers

- Every block device has a request queue
- Requests are scheduled
  - Reorder request to optimize disk performance (exploit locality)
  - Merge adjacent requests
  - Deadline scheduler (best effort)
  - Anticipatory scheduler