Linux Device Driver
(ch1~ch3 & case study)

Rock Kuo

rock@nchc.org.tw

Grid Technology Division,
NCHC, Taiwan

Agenda

- **Introduction**
- **Linux Device Driver**
  - Ch1. An Introduction to Device Drivers
  - Ch2. Building & Running Modules
  - Ch3. Char Drivers
- **Jollen's Experience**
- **Reference**

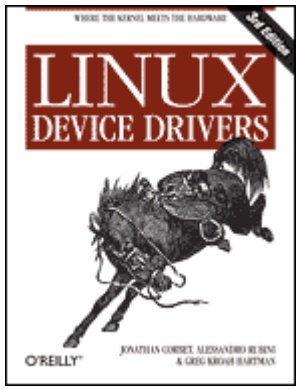
Introduction

- Kernel-correlation experiences
  - Kerrighed, GPFS, Xen-> porting kernel
  - Kernel modules
  - /dev/gpfs0, /dev/xen
- Support or enhance Kernel development
  - Fix Kerrighed DSM Module

Background

- C language
  - Pointer, Structure
- Data Structure
  - List, Queue, Table...
- Basic UNIX operations
- UNIX term
  - System call, Pipelines...
- Hardware
  - Concept DMA, IRQ, I/O port...

Linux Device Driver

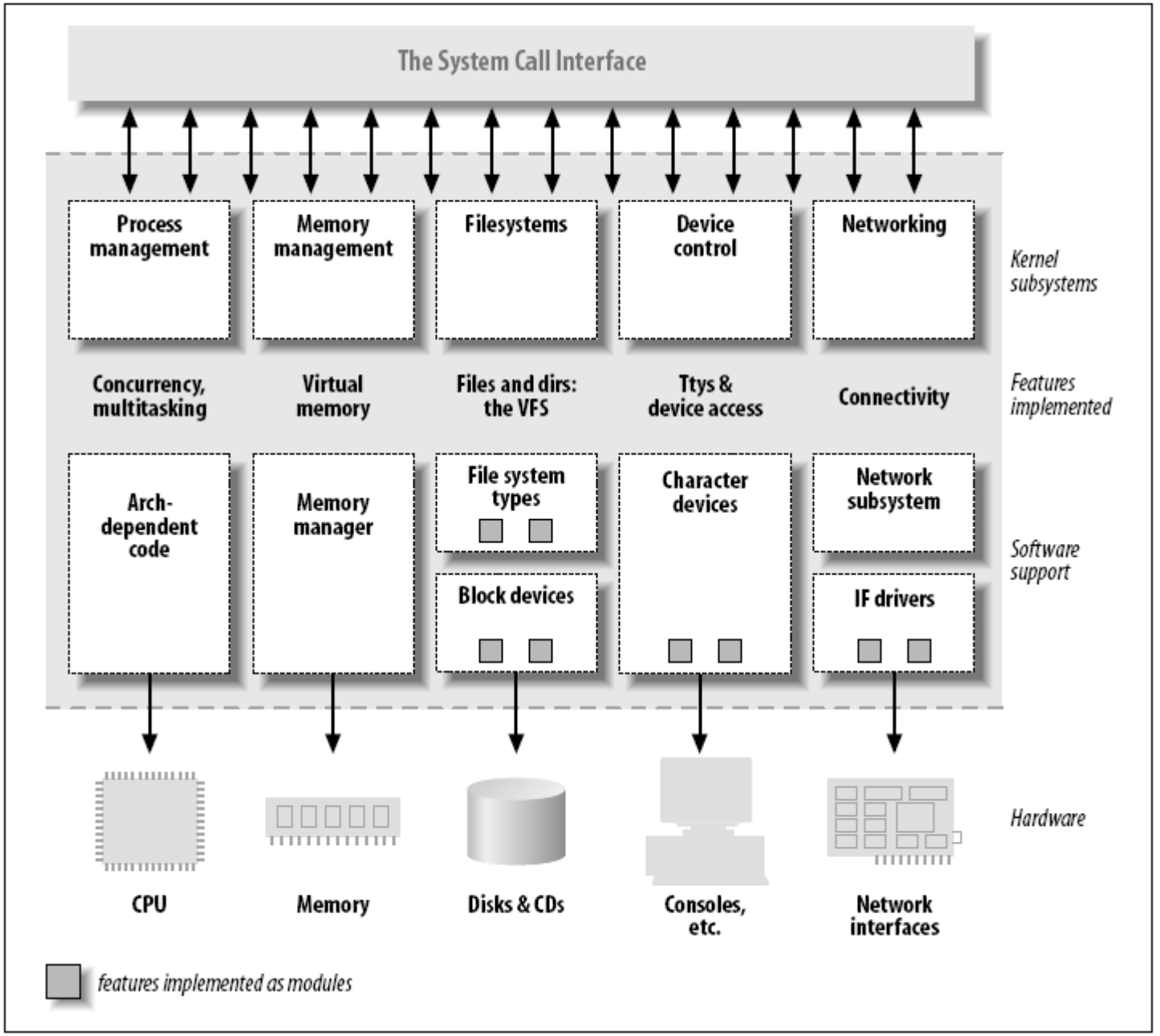


Linux Device Drivers,
by Jonathan Corbet, Alessandro Rubini & Greg Kroah-Hartman.

Grid Technology Division,
NCHC, Taiwan

Ch1. An Introduction to Device Drivers

- Role of the device driver
  - Flexible (mechanism vs. policy)
  - ex. X server vs. window/session manager
- Policy-free
  - Write kernel code to access the hardware, but don't force particular policies on the user, since different users have different needs.
- Loadable Modules
  - Dynamic linked to running kernel



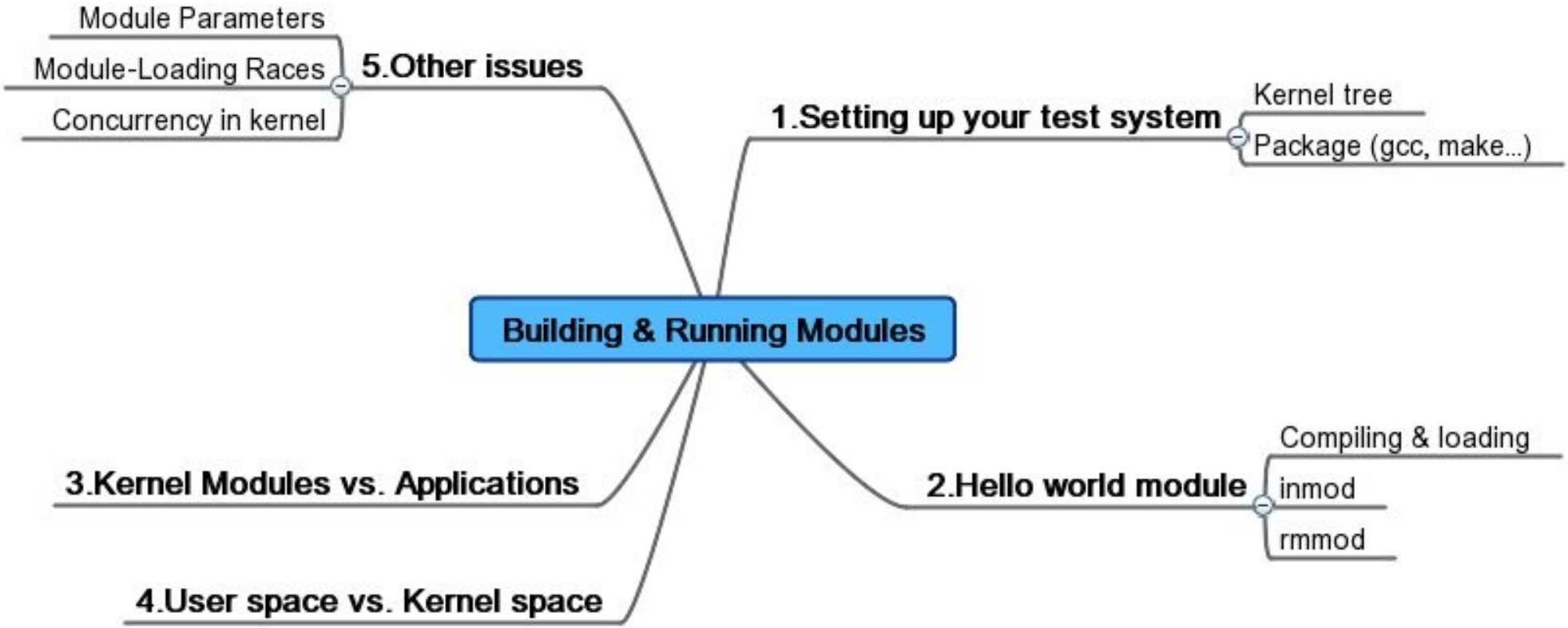
A Split View of Kernel

Source: Jonathan C, et al.

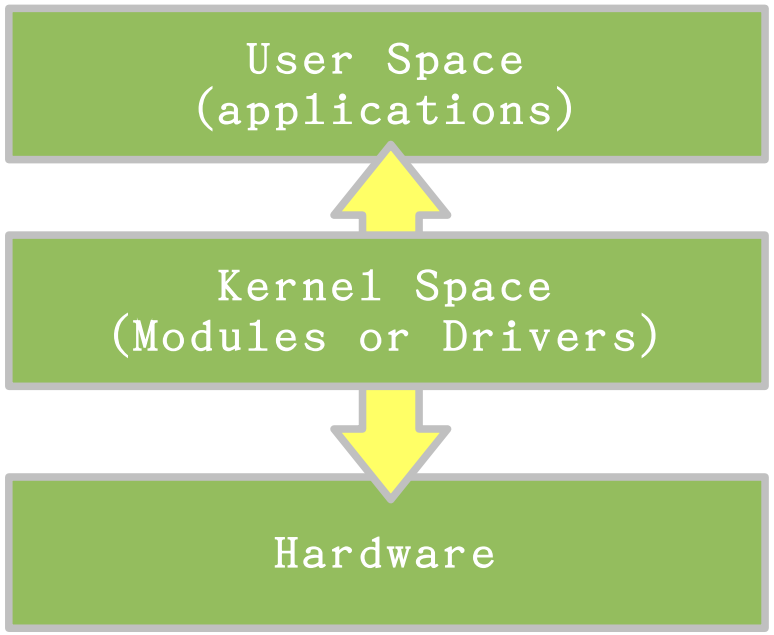
Class of Devices

- Character Device
  - A stream of bytes (ex. file)
  - open, close, read, write system call
  - ex. /dev/console, /dev/ttyS0
  - Access sequentially
- Block Device
  - Block size: 512 bytes
  - Random access
- Network Device
  - Stream-oriented (TCP)
  - Completely different char & block
  - Unique qeue name (ex. eth0)

Ch2. Building & Running Modules



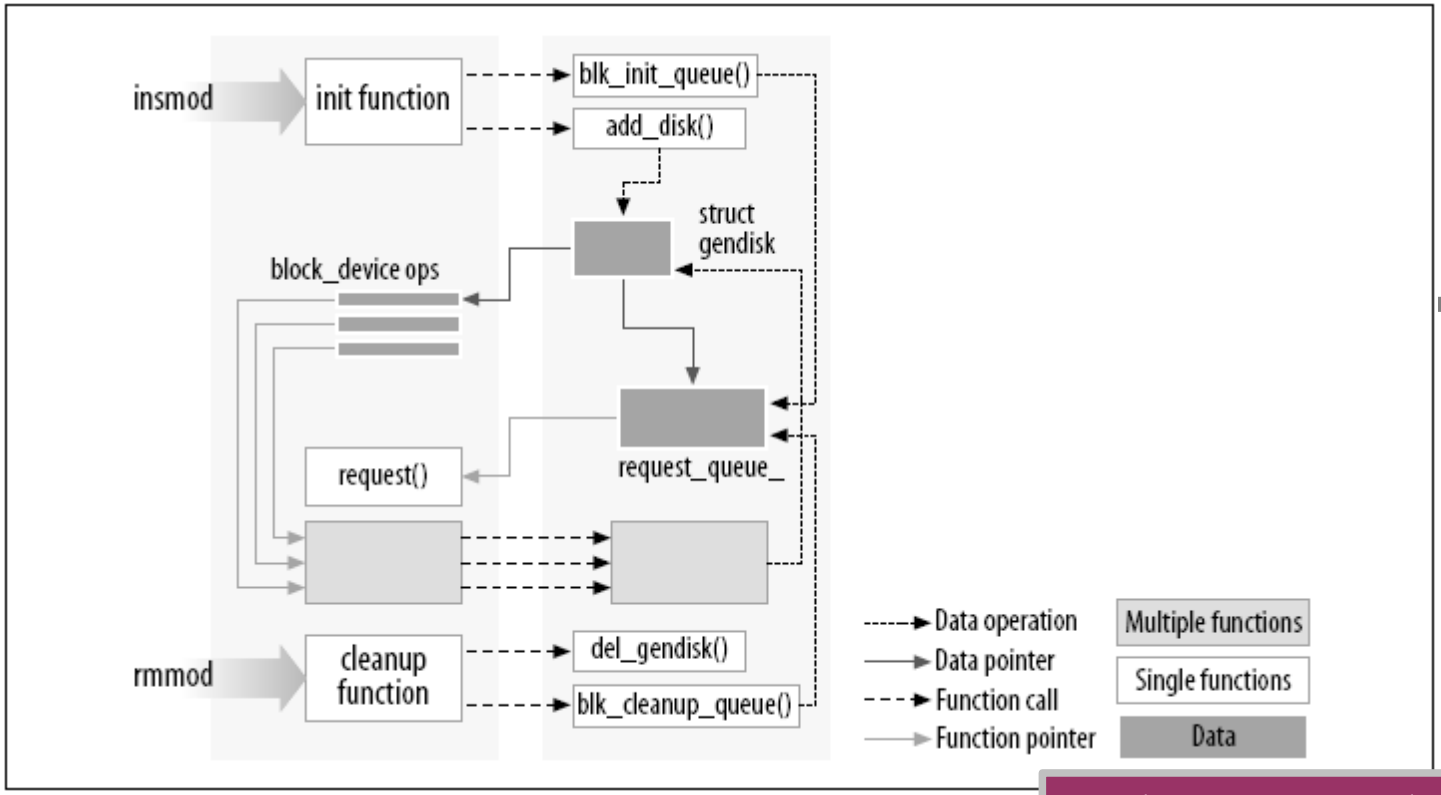
Kernel Space



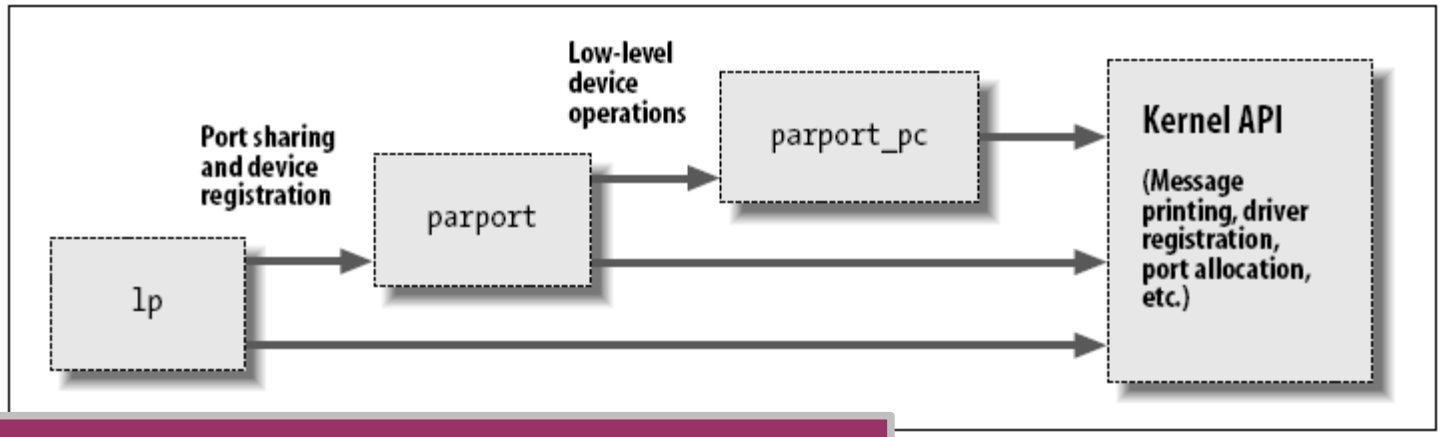
- Kernel modules vs. Application
  - Register vs. single task
  - Kernel lib vs. libc
  - 1 page vs. VM
  - Undo
- Kernel space vs. User space
  - Layer
  - Supervisior mode vs. User mode
  - switch

Example: Hello Orz Module

```
#include <linux/init.h>

#include <linux/module.h>

MODULE_LICENSE("DUAL BSD/GPL");

static int Orz_init(void) {
printk("<1> Hello Orz !!!\n");
return 0;
}
static void Orz_exit(void) {
printk("<1> Bye OGC !!!\n");
}

module_init(Orz_init);
module_exit(Orz_exit);
```



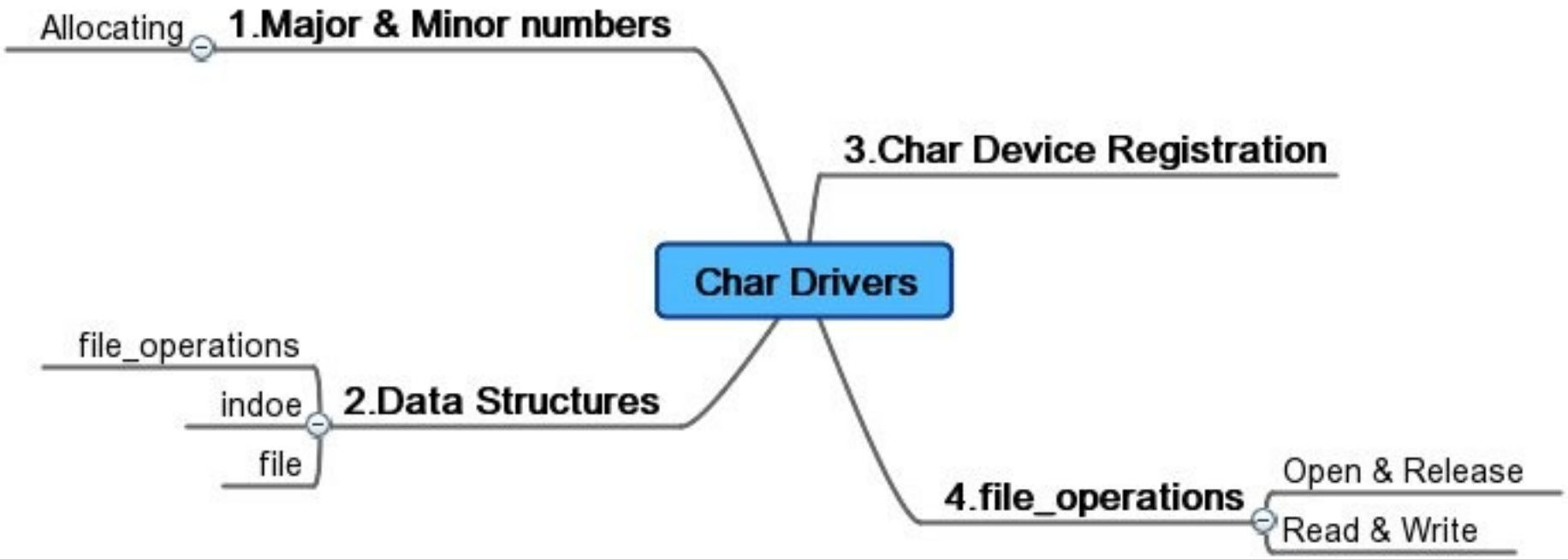
Linking a module to kernel



Stacking of parallel port driver module

Source: Jonathan C, et al.

Ch3. Char Drivers



Major & Minor Numbers

- dev_t /*<linux/types.h>*/
  - MAJOR(dev_t *dev*) /*<linux/kdev_t.h>*/
  - MINOR(dev_t *dev*)
  - MKDEV(int *major*, int *minor*)
- int register_chrdev_region(dev_t *first*, unsigned int *count*, char *name); /*<linux/fs.h>*/
- int alloc_chrdev_region(dev_t *dev, unsigned int *firstminor*, unsigned int *count*, char *name);
- void unregister_chrdev_region(dev_t *first*, unsigned int *count*);

Some Important Data Structures

- file_operations /*<linux/fs.h>*/
- File /*<linux/fs.h>*/
- Inode /*<linux/fs.h>*/

file_operations Structure

```
struct file_operations {
        struct module *owner;
        loff_t (*llseek) (struct file *, loff_t, int);
        ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
        ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
        ssize_t (*aio_read) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
        ssize_t (*aio_write) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
        int (*readdir) (struct file *, void *, filldir_t);
        unsigned int (*poll) (struct file *, struct poll_table_struct *);
        int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
        long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
        long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
        int (*mmap) (struct file *, struct vm_area_struct *);
        int (*open) (struct inode *, struct file *);
        int (*flush) (struct file *, fl_owner_t id);
        int (*release) (struct inode *, struct file *);
        int (*fsync) (struct file *, struct dentry *, int datasync);
        int (*aio_fsync) (struct kiocb *, int datasync);
        int (*fasync) (int, struct file *, int);
        int (*lock) (struct file *, int, struct file_lock *);
        ssize_t (*sendfile) (struct file *, loff_t *, size_t, read_actor_t, void *);
        ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
        unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned long, unsigned long, unsigned long);
        int (*check_flags)(int);
        int (*dir_notify)(struct file *filp, unsigned long arg);
        int (*flock) (struct file *, int, struct file_lock *);
        ssize_t (*splice_write)(struct pipe_inode_info *, struct file *, loff_t *, size_t, unsigned int);
        ssize_t (*splice_read)(struct file *, loff_t *, struct pipe_inode_info *, size_t, unsigned int);
};
```

file Structure

```
struct file {
        union {
                struct list_head        fu_list;
                struct rcu_head         fu_rcuhead;
        } f_u;
        struct path             f_path;
#define f_dentry        f_path.dentry
#define f_vfsmnt        f_path.mnt
        const struct file_operations    *f_op;
        atomic_t                f_count;
        unsigned int            f_flags;
        mode_t                  f_mode;
        loff_t                  f_pos;
        struct fown_struct      f_owner;
        unsigned int            f_uid, f_gid;
        struct file_ra_state    f_ra;
        unsigned long           f_version;
#ifdef CONFIG_SECURITY
        void                    *f_security;
#endif
        void                    *private_data;
#ifdef CONFIG_EPOLL
        struct list_head        f_ep_links;
        spinlock_t              f_ep_lock;
#endif
        struct address_space    *f_mapping;
};
```

inode Structure

```
struct inode {
        struct hlist_node       i_hash;
        struct list_head        i_list;
        struct list_head        i_sb_list;
        struct list_head        i_dentry;
        unsigned long           i_ino;
        atomic_t                i_count;
        unsigned int            i_nlink;
        uid_t                   i_uid;
        gid_t                   i_gid;
        dev_t                   i_rdev;
        unsigned long           i_version;
        loff_t                  i_size;
#ifdef __NEED_I_SIZE_ORDERED
        seqcount_t              i_size_seqcount;
#endif
        struct timespec         i_atime;
        struct timespec         i_mtime;
        struct timespec         i_ctime;
        unsigned int            i_blkbits;
        blkcnt_t                i_blocks;
        unsigned short          i_bytes;
        umode_t                 i_mode;
        spinlock_t              i_lock; /* i_blocks,
    i_bytes, maybe i_size */
        struct mutex            i_mutex;
        struct rw_semaphore     i_alloc_sem;
        struct inode_operations *i_op;
        const struct file_operations    *i_fop;
        struct super_block      *i_sb;
        struct file_lock        *i_flock;
        struct address_space    *i_mapping;
        struct address_space    i_data;
#ifdef CONFIG_QUOTA
        struct dquot            *i_dquot[MAXQUOTAS];
#endif
        struct list_head        i_devices;
        union {
                struct pipe_inode_info  *i_pipe;
                struct block_device     *i_bdev;
                struct cdev             *i_cdev;
        };
        int                     i_cindex;
        __u32                   i_generation;
#ifdef CONFIG_DNOTIFY
        unsigned long           i_dnotify_mask;
        struct dnotify_struct   *i_dnotify;
#endif

#ifdef CONFIG_INOTIFY
        struct list_head        inotify_watches;
        struct mutex            inotify_mutex;
#endif
        unsigned long           i_state;
        unsigned long           dirtied_when;
        unsigned int            i_flags;
        atomic_t                i_writecount;
#ifdef CONFIG_SECURITY
        void                    *i_security;
#endif
        void                    *i_private;
};
```

Example: scull.c

```
#include
struct file_operations scull_ops{
owner : THIS_MODULE,
read : scull_read,
write : scull_write,
open : scull_open,
release : scull_release,
};

scull_read()
scull_write()
scull_open()
scull_release()
scull_init()
scull_exit()

module_init(scull_init);
module_exit(scull_exit);
```

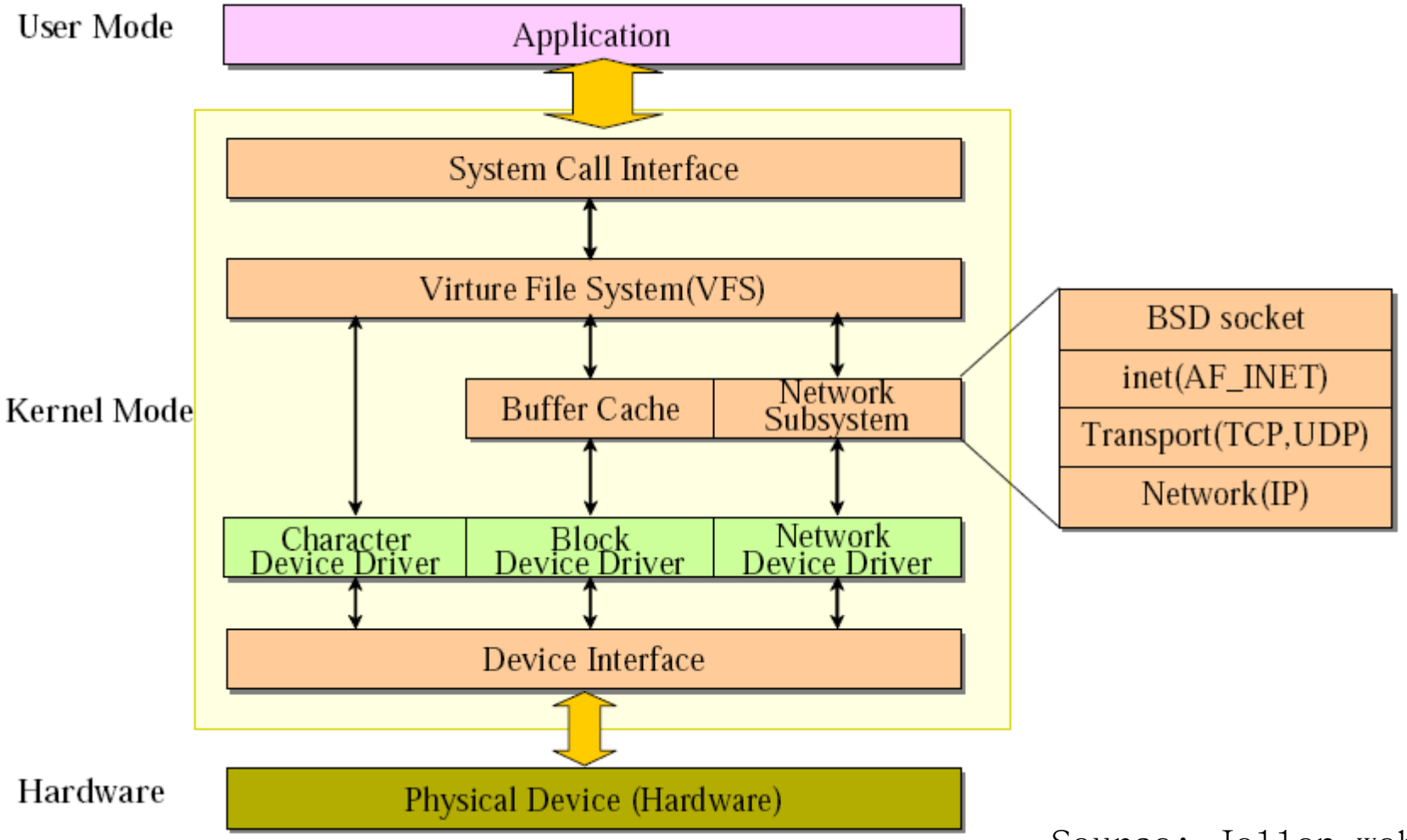
Example

```
static int scull_init(void)
{
scull_dev = MKDEV(DEV_MAJOR, DEV_MINOR);
register_chrdev_region(scull_dev, 1, "scull");
scull_cdev = cdev_alloc();
scull_cdev->owner = THIS_MODULE;
scull_cdev->ops = &scull_fops;
cdev_init(scull_cdev, &scull_fops);
cdev_add(scull_cdev, scull_dev, 1);
return(0);
}

static void scull_exit(void)
{
cdev_del(scull_cdev);
unregister_chrdev_region(scull_dev, 1);
}
```

Jollen's Experience

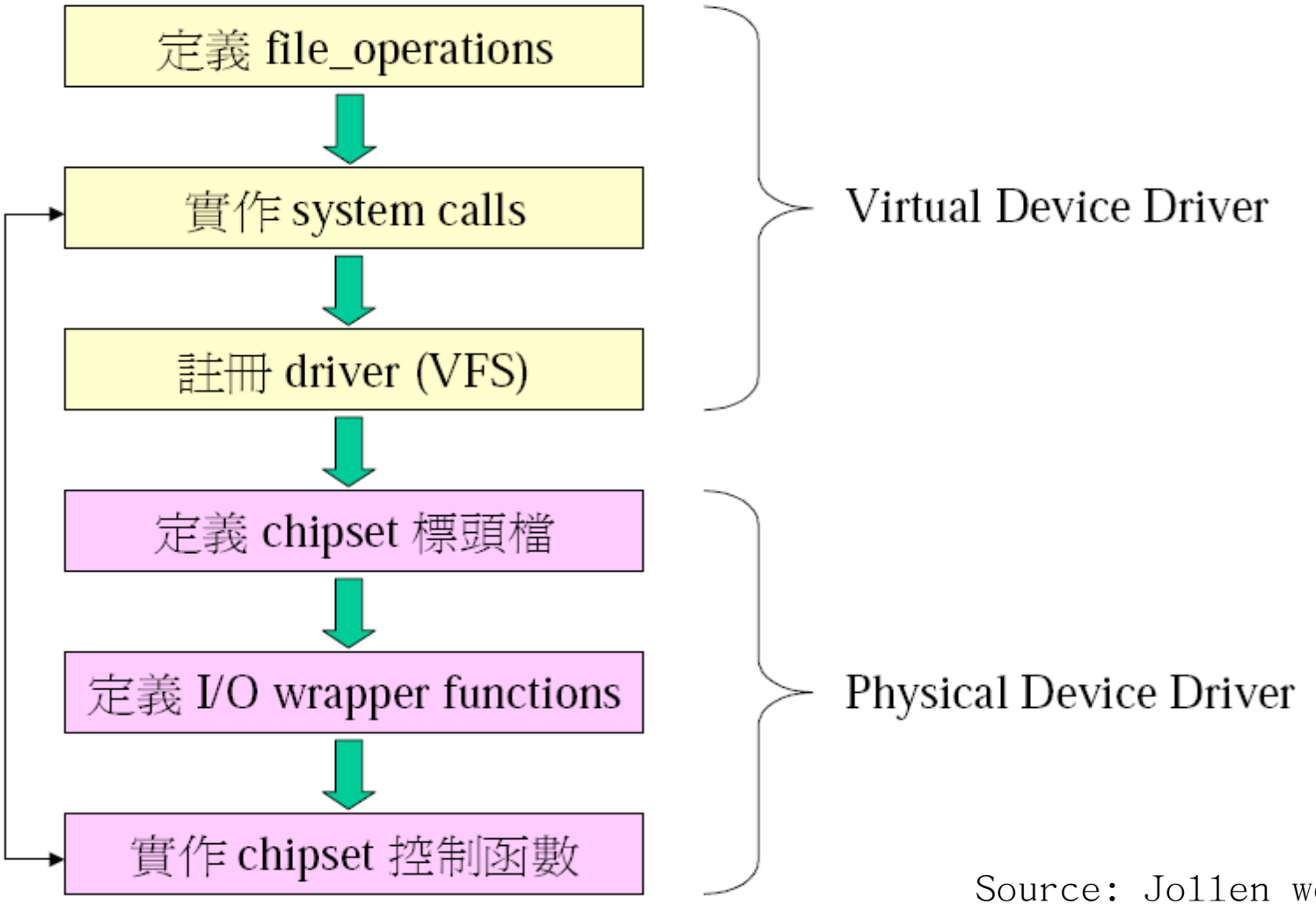
Grid Technology Division,
NCHC, Taiwan

Architecture



Source: Jollen web site

Development Procedure



Source: Jollen web site

定義 file_operations

```
struct file_operations card_ops = {
.open:      card_open,
.write:     card_write,
.release:   card_release,
.ioctl:     card_ioctl,
}
```

實做 system calls

```
#include
#include  “card.h”

int card_open(structure inode *indoe, struct file *file)
{
MOD_INC_USE_COUNT;
return 0;
}

card_write()
card_release()
card_ioctl()
```

Future

- Linux Device Driver
  - Ch4. Debugging Techniques
  - Ch5. Concurrency & Race Conditions
  - Ch6. Advanced Char Driver Operations
- Patching Kerrighed-based Kernel
  - Memory Management
  - Process management

Reference

- Jonathan Corbet, Alessandro Rubini & Greg Kroah-Hartman, Linux Device Drivers 3e, O'REILLY, 2005
- Robert L., Linux System Programming, O'REILLY, 2007
- Jollen web site, http://www.jollen.org
- NCHC Grid Architecture Research Group, http://trac.nchc.org.tw/grid/wiki
- Xavier Calbet, "Writing device driver in Linux: A brief tutorial," 2006
- Alan Cox, "Writing Linux Mouse Drviers," 1999
- Adutya Kulkarni, "Writing a Linux device driver," 2000

Discussion

rock@nchc.org.tw