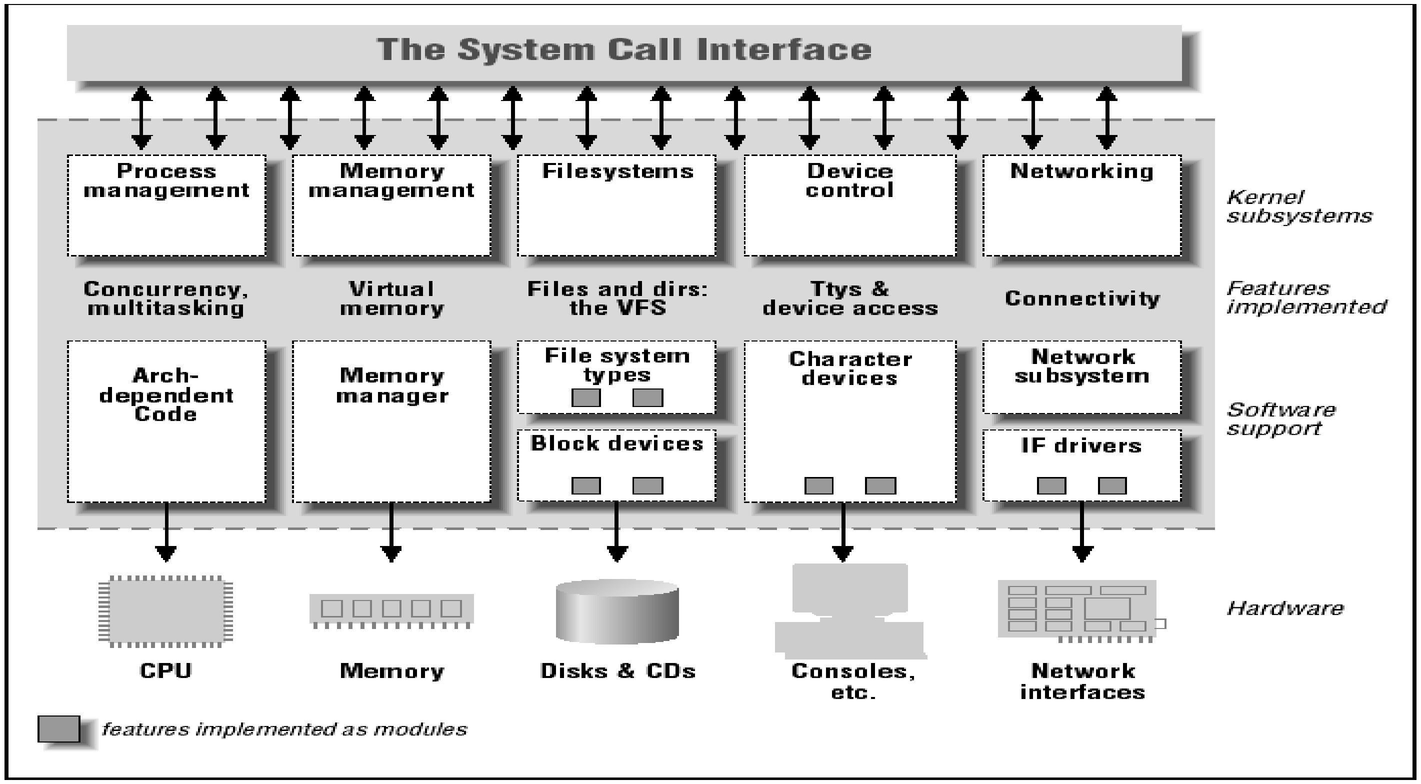
Device Driver Architecture

Brian Chang, 22 July 2002

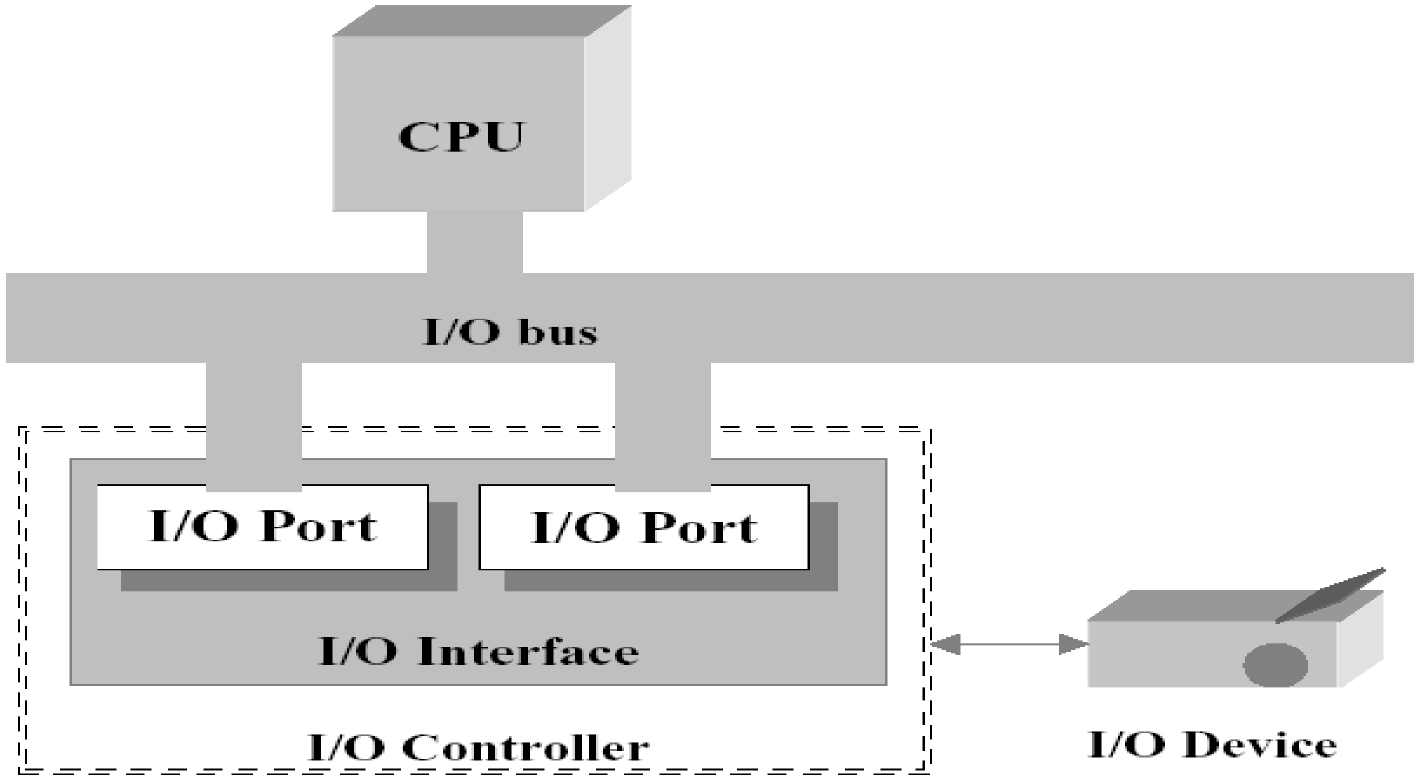
Brian_c@castlenet.com.tw

2002 Linux kernel trace seminar

A split view of the kernel



PC’s I/O Device Architecture



Device Files

Each device file has three attributes

- Type – either block or character
- Major number – from 1 to 255
- Minor number – from 0 to 255

ex.:

dev/had	block	3	0	first IDE disk
dev/tty0	char	4	0	current virtual console

Reference: Documentation/devices.txt (Major number)

Device Driver Load

- **Statically** ： the corresponding device file class is registered during system initialization
- **Dynamically** ： the corresponding file class is registered/unregistered when a module is loaded/unloaded

Device file descriptors (1)

- **chrdevs table:** all device_struct descriptors for character device files are included in the chrdevs table

- **blkdevs table:** all device_struct descriptors for block device files are included in the blkdevs table

Device file descriptors (2)

/fs/devices.c

```
struct device_struct {
    const char * name;
    struct file_operations * fops;
};
static struct device_struct chrdevs[MAX_CHRDEV];
```

/fs/block_dev.c

```
static struct {
    const char *name;
    struct block_device_operations *bdops;
} blkdevs[MAX_BLKDEV];
```

/include/linux/major.h

```
#define MAX_CHRDEV 255
#define MAX_BLKDEV 255
```

Character device file operations

/include/linux/fs.h

```
struct file_operations {
      struct module *owner;
      loff_t (*llseek) (struct file *, loff_t, int);
      ssize_t (*read) (struct file *, char *, size_t, loff_t *);
      ssize_t (*write) (struct file *, const char *, size_t, 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);
      int (*mmap) (struct file *, struct vm_area_struct *);
      int (*open) (struct inode *, struct file *);
     .  .  .  .  .  .
```

Character device file operations

/include/linux/fs.h

```
struct file_operations {
   . . . . . .
    int (*flush) (struct file *);
    int (*release) (struct inode *, struct file *);
    int (*fsync) (struct file *, struct dentry *, int datasync);
    int (*fasync) (int, struct file *, int);
    int (*lock) (struct file *, int, struct file_lock *);
    ssize_t (*readv) (struct file *, const struct iovec *, unsigned long, loff_t *);
    ssize_t (*writev) (struct file *, const struct iovec *, unsigned long, loff_t *);
   . . . . . .
```

Character device file operations

/include/linux/fs.h

```
struct file_operations {
    . . . . . .
      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);
};
```

Block device file operations

/include/linux/fs.h

```
struct block_device_operations {
      int (*open) (struct inode *, struct file *);
      int (*release) (struct inode *, struct file *);
      int (*ioctl) (struct inode *, struct file *, unsigned, unsigned long);
      int (*check_media_change) (kdev_t);
      int (*revalidate) (kdev_t);
      struct module *owner;
};
```

The file structure (1)

/include/linux/fs.h

```
struct file {
      struct list_head          f_list;
      struct dentry            *f_dentry;
      struct vfsmount           *f_vfsmnt;
      struct file_operations  *f_op;
      atomic_t                  f_count;
      unsigned int              f_flags;
      mode_t                    f_mode;
      loff_t                    f_pos;
      unsigned long             f_reada, f_ramax, f_raend, f_ralen, f_rawin;
      struct fown_struct        f_owner;
      unsigned int              f_uid, f_gid;
      •   •   •   •   •   •
```

The file structure (2)

/include/linux/fs.h

```
struct file {
    . . . . . .
    int                     f_error;
    unsigned long           f_version;

    /* needed for tty driver, and maybe others */
    void                    *private_data;

    /* preallocated helper kiobuf to speedup O_DIRECT */
    struct kiobuf           *f_iobuf;
    long                    f_iobuf_lock;
};
```

Device Register/Unregister (1)

- Register

  init_module()

  → *devfs_register_chrdev()/ devfs_register_blkdev()*

  → *register_chrdev() / register_blkdev ()*

- Unregister

  cleanup_module ()

  → *devfs_unregister_chrdev() / devfs_unregister_blkdev()*

  → *unregister_chrdev()/unregister_blkdev()*

Device Register/Unregister (2)

/fs/devices.c

int register_chrdev(unsigned int major, const char * name, struct file_operations *fops)

int unregister_chrdev(unsigned int major, const char * name)

/fs/block_dev.c

int register_blkdev(unsigned int major, const char * name, struct block_device_operations *bdops)

int unregister_blkdev(unsigned int major, const char * name)

Monitoring I/O Operations -- Polling Mode

- Polling mode

Ex. lp.c

```
static ssize_t lp_write( ・ ・ ・ ){
   ・ ・ ・ ・ ・
  do{
     ・ ・ ・ ・ ・
     if (copy_size > 0) {
     ・ ・ ・ ・ ・
     } else if (current->need_resched)
          schedule ();
     ・ ・ ・ ・ ・
  } while (count > 0); ・ ・ ・ }
```

Monitoring I/O Operations -- Interrupt Mode (1)

- Interrupt Mode

Ex. serial.c

```
static int startup(struct async_struct * info)
{
    .  .  .  .  .
  retval = request_irq(state->irq, handler, SA_SHIRQ, "serial", &IRQ_ports[state->irq]);
  .  .  .  .  .
}
static int rs_ioctl(struct tty_struct *tty, struct file * file,
                    unsigned int cmd, unsigned long arg)
{
  .  .  .  .  .
  case TIOCMIWAIT:
    .  .  .  .  .
    interruptible_sleep_on(&info->delta_msr_wait);
    .  .  .  .  .
}
```

Monitoring I/O Operations -- Interrupt Mode (2)

```
static void rs_interrupt(int irq, void *dev_id, struct pt_regs * regs)
{
    • • • • •
    check_modem_status(info);
    • • • • •
}
static _INLINE_ void check_modem_status(struct async_struct *info)
{
     • • • • •
    if (status & UART_MSR_ANY_DELTA) {
      • • • • •
      wake_up_interruptible(&info->delta_msr_wait); }
     • • • • •
}
```

Generic parallel printer driver (1)

/drivers/char/lp.c

If this driver is built into the kernel, you can configure it using the kernel command-line. For example:

lp=parport1,none,parport2	(bind lp0 to parport1, disable lp1 and bind lp2 to parport2)
lp=auto	(assign lp devices to all ports that have printers attached, as determined by the IEEE-1284 autoprobe)
lp=reset	(reset the printer during initialisation)
lp=off	(disable the printer driver entirely)

Generic parallel printer driver (2)

<u>/drivers/char/lp.c</u>

If the driver is loaded as a module, similar functionality is available using module parameters. The equivalent of the above commands would be:

```
# insmod lp.o parport=1,none,2
# insmod lp.o parport=auto
# insmod lp.o reset=1
```

Generic parallel printer driver (3)

/drivers/char/lp.c

```
. . . . . .
__setup("lp=", lp_setup);
module_init(lp_init_module);
module_exit(lp_cleanup_module);
```

Generic parallel printer driver (4)

/drivers/char/lp.c

```
static int __init lp_init_module (void)
{
    if (parport[0]) {
            if (!strncmp(parport[0], "auto", 4))
                    parport_nr[0] = LP_PARPORT_AUTO;
            else {
                    . . . . . .
            }
    }
    return lp_init();
}
```

Generic parallel printer driver (5)

/drivers/char/lp.c

```
int __init lp_init (void)
{
   . . . . . .
   if (devfs_register_chrdev (LP_MAJOR, "lp", &lp_fops)) {
              printk ("lp: unable to get major %d\n", LP_MAJOR);
              return -EIO;
   }
   devfs_handle = devfs_mk_dir (NULL, "printers", NULL);
   if (parport_register_driver (&lp_driver)) {
              printk ("lp: unable to register with parport\n");
              return -EIO;
   }
```

/include/asm-i386/errno.h

```
#define EIO 5 /* I/O error */
```

Generic parallel printer driver (5a)

/drivers/char/lp.c

```
static struct file_operations
lp_fops = {
        owner:   THIS_MODULE,
        write:   lp_write,
        ioctl:   lp_ioctl,
        open:    lp_open,
        release: lp_release,
#ifdef CONFIG_PARPORT_1284
        read:    lp_read,
#endif};
```

```
R, "lp", &lp_fops)) {
 %d\n", LP_MAJOR);

 "printers", NULL);
er)) {
        printk ("lp: unable to register with parport\n");
        return -EIO;
}
```

Generic parallel printer driver (6)

/drivers/char/lp.c

```
int __init lp_init (void)
{
   . . . . . .
   if (!lp_count) {
              printk (KERN_INFO "lp: driver loaded but no devices found\n");
#ifndef CONFIG_PARPORT_1284
              if (parport_nr[0] == LP_PARPORT_AUTO)
                            printk (KERN_INFO "lp: (is IEEE 1284 support enabled?)\n");
#endif
      }

      return 0;
}
```

Generic parallel printer driver (7)

/fs/devfs/base.c

```
int devfs_register_chrdev (unsigned int major, const char *name,
                                   struct file_operations *fops)
{
    if (boot_options & OPTION_ONLY) return 0;
    return register_chrdev (major, name, fops);
}   /*  End Function devfs_register_chrdev  */
```

Generic parallel printer driver (8)

/fs/devfs/devices.c

```
int register_chrdev(unsigned int major, const char * name, struct file_operations *fops)
{
      if (major == 0) {
            write_lock(&chrdevs_lock);
            for (major = MAX_CHRDEV-1; major > 0; major--) {
                  if (chrdevs[major].fops == NULL) {
                        chrdevs[major].name = name;
                        chrdevs[major].fops = fops;
                        write_unlock(&chrdevs_lock);
                        return major;
                  }
            }
            write_unlock(&chrdevs_lock);
            return -EBUSY;
      }
```

Generic parallel printer driver (9)

/fs/devfs/devices.c

```
int register_chrdev(unsigned int major, const char * name, struct file_operations *fops)
{
. . . . . .
if (major >= MAX_CHRDEV)
            return -EINVAL;
     write_lock(&chrdevs_lock);
     if (chrdevs[major].fops && chrdevs[major].fops != fops) {
            write_unlock(&chrdevs_lock);
            return -EBUSY;
     }
     chrdevs[major].name = name;
     chrdevs[major].fops = fops;
     write_unlock(&chrdevs_lock);
     return 0;
}
```

/include/linux/major.h

```
#define MAX_CHRDEV 255
```

Generic parallel printer driver (10)

/drivers/char/lp.c

```
static void lp_cleanup_module (void)
{
    unsigned int offset;
    parport_unregister_driver (&lp_driver);
#ifdef CONFIG_LP_CONSOLE
    unregister_console (&lpcons);
#endif
    devfs_unregister (devfs_handle);
    devfs_unregister_chrdev(LP_MAJOR, "lp");
    for (offset = 0; offset < LP_NO; offset++) {
        if (lp_table[offset].dev == NULL)
                continue;
        parport_unregister_device(lp_table[offset].dev);
    }
}
```

Generic parallel printer driver (11)

/fs/devfs/base.c

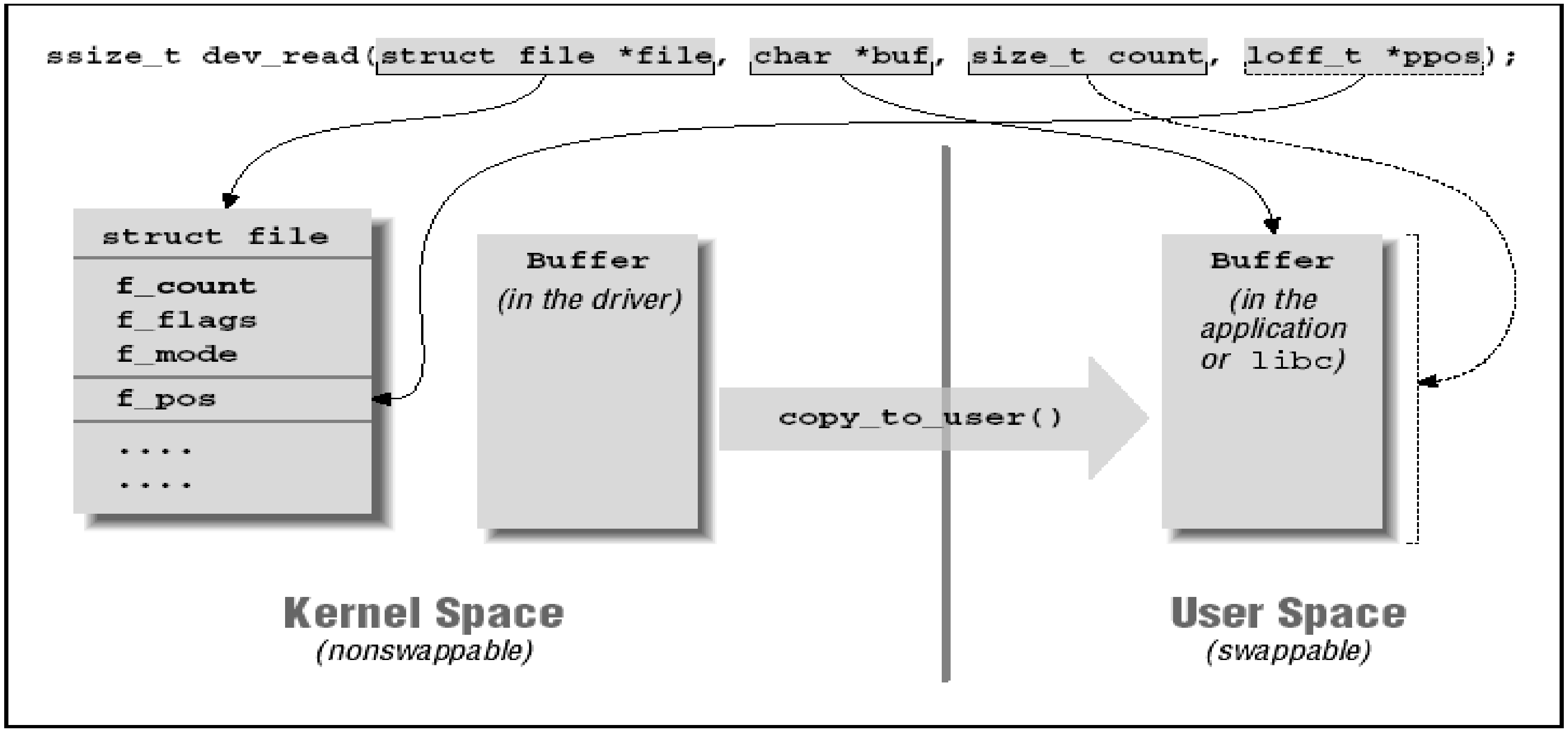
```
int devfs_unregister_chrdev (unsigned int major, const char *name)
{
    if (boot_options & OPTION_ONLY) return 0;
    return unregister_chrdev (major, name);
}   /*  End Function devfs_unregister_chrdev  */
```

Generic parallel printer driver (12)

/fs/devfs/devices.c

```
int unregister_chrdev(unsigned int major, const char * name)
{
	if (major >= MAX_CHRDEV)
		return -EINVAL;
	write_lock(&chrdevs_lock);
	if (!chrdevs[major].fops || strcmp(chrdevs[major].name, name)) {
		write_unlock(&chrdevs_lock);
		return -EINVAL;
	}
	chrdevs[major].name = NULL;
	chrdevs[major].fops = NULL;
	write_unlock(&chrdevs_lock);
	return 0;
}
```

Generic parallel printer driver (13)



Generic parallel printer driver (14)

/drivers/char/lp.c

```
static ssize_t lp_read(struct file * file, char * buf, size_t count, loff_t *ppos) {
  .  .  .  .  .  .
   if (copy_from_user (kbuf, buf, copy_size))
                return -EFAULT;
  .  .  .  .  .  .
}
```

/drivers/char/rocket.c

```
int copy_from_user(void *to, const void *from_user, unsigned long len) {
       int        error;
       error = verify_area(VERIFY_READ, from_user, len);
       if (error)
            return len;
       memcpy_fromfs(to, from_user, len);
       return 0;
}
```

Generic parallel printer driver (15)

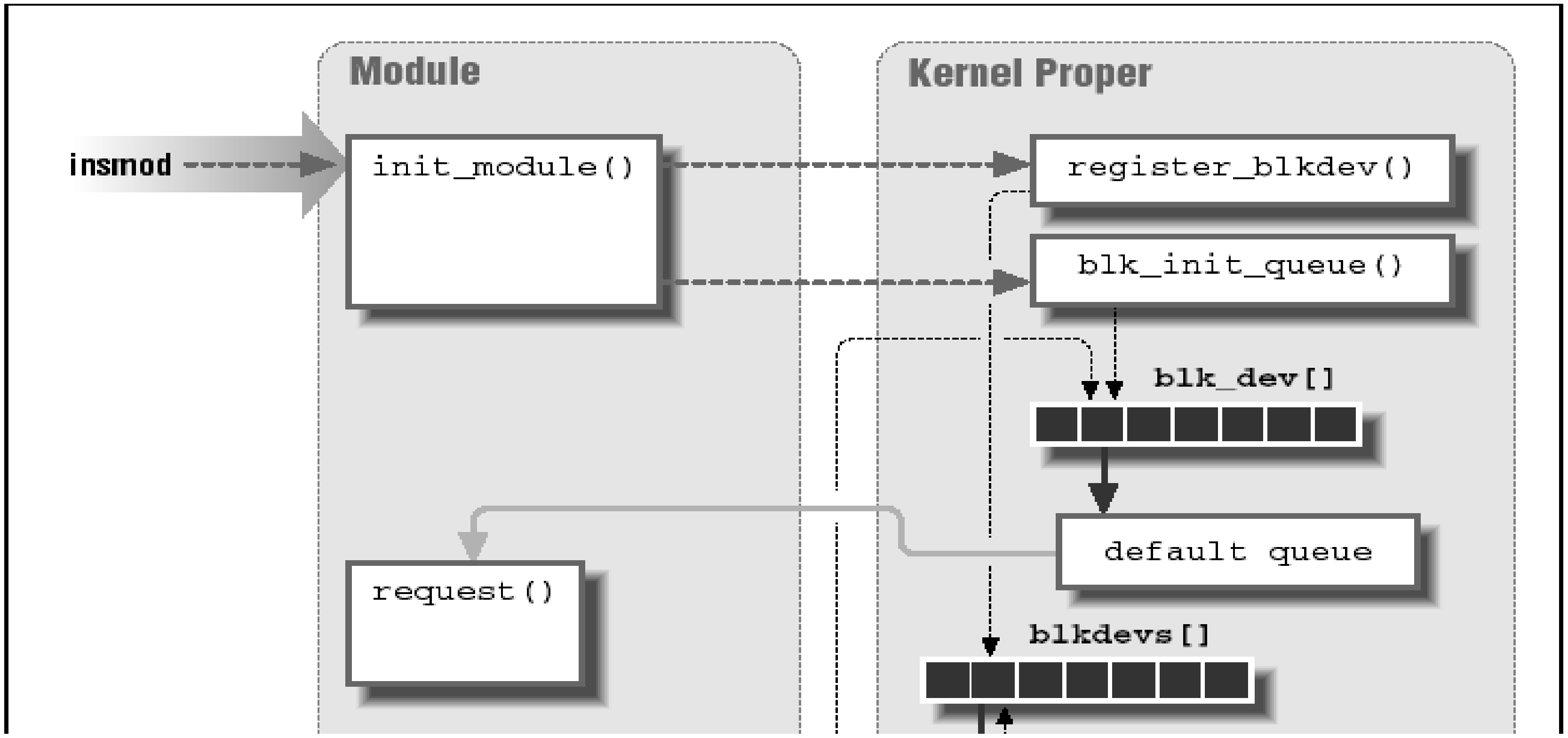
/drivers/char/lp.c

```
static ssize_t lp_write(struct file * file, const char * buf, size_t count, loff_t *ppos){
   .  .  .  .  .  .
   if (copy_from_user (kbuf, buf, copy_size))
                return -EFAULT;
  .  .  .  .  .  .
}
```

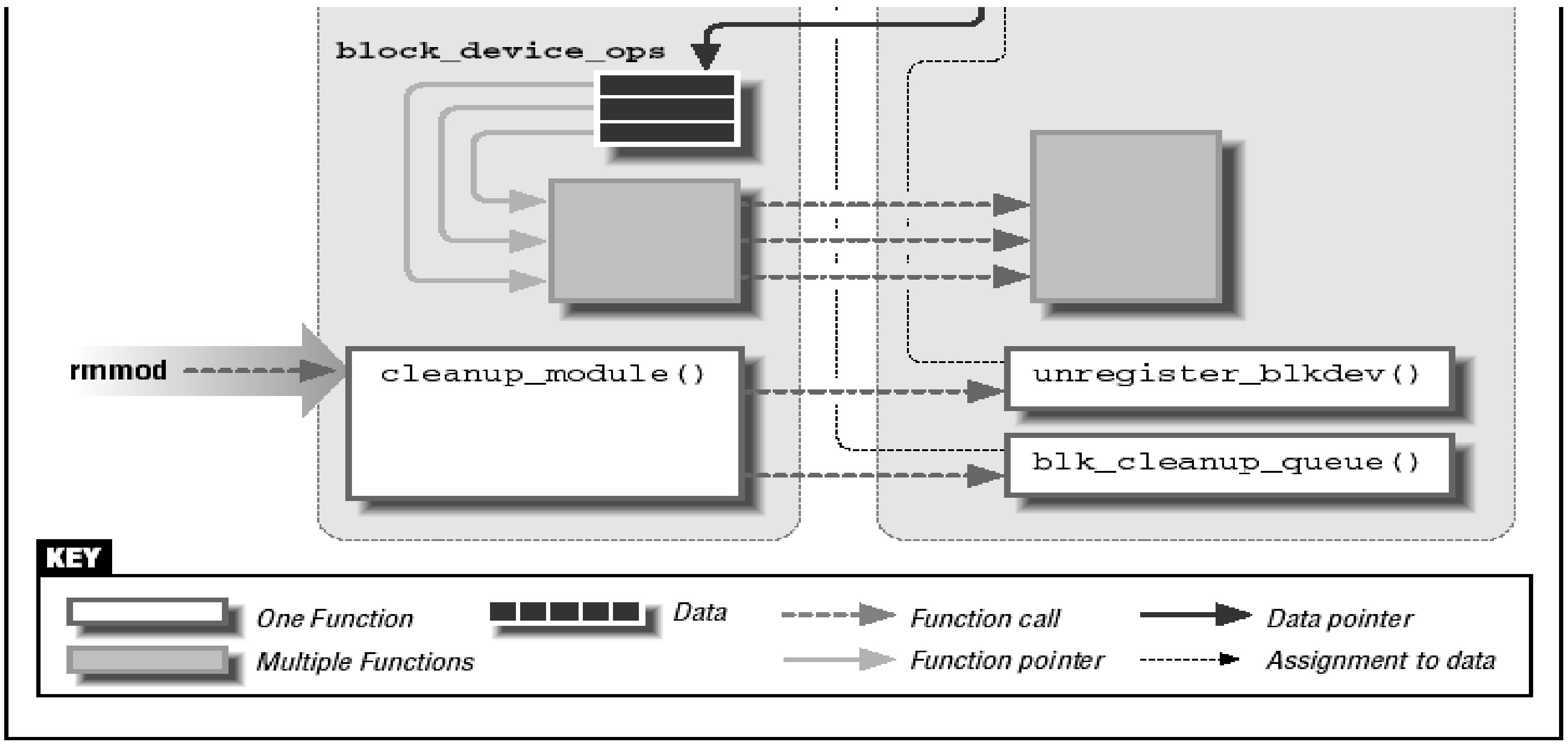
/drivers/char/rocket.c

```
int copy_from_user(void *to, const void *from_user, unsigned long len){
      int       error;
      error = verify_area(VERIFY_READ, from_user, len);
      if (error)
          return len;
      memcpy_fromfs(to, from_user, len);
      return 0;
}
```

Registering a Block Device Driver(1)



Registering a Block Device Driver(2)



The floppy driver (1)

/drivers/block/floppy.c

```
int init_module(void)
{
      if (floppy)
                parse_floppy_cfg_stri
      return floppy_init();
}
int __init floppy_init(void)
{
 .    .    .    .    .    .
      devfs_handle = devfs_mk_dir (NULL, "floppy", NULL);
      if (devfs_register_blkdev(MAJOR_NR,"fd",&floppy_fops)) {
                printk("Unable to get major %d for floppy\n",MAJOR_NR);
                return -EBUSY;
      }
 .    .    .    .    .    .
```

/drivers/char/lp.c

```
static struct
block_device_operations floppy_fops = {
                owner:     THIS_MODULE,
                open:      floppy_open,
                release:   floppy_release,
                ioctl:     fd_ioctl,
                check_media_change:
                           check_floppy_change,
                revalidate:
                           floppy_revalidate,};
```

The floppy driver (1a)

/drivers/block/floppy.c

```
int init_modul
{
    if (floppy)
        pa
    return flo
}
int __init flop
{
 .   .   .   .   .   .
    devfs_handle = d      _mk_dir (NULL, "floppy", NULL);
    if (devfs_register_blkdev(MAJOR_NR,"fd",&floppy_fops)) {
        printk("Unable to get major %d for floppy\n",MAJOR_NR);
        return -EBUSY;
    }
 .   .   .   .   .   .
```

/fs/devfs/base.c

```
int devfs_register_blkdev (unsigned int major, const
char *name, struct block_device_operations *bdops)
{
    if (boot_options & OPTION_ONLY)
            return 0;
    return register_blkdev (major, name, bdops);
}
```

Tl

/fs/devfs/base.c

```
int register_blkdev( ・ ・ ・ ){
        if (major == 0) {
                for (major = MAX_BLKDEV-1; major > 0; major--) {
                        if (blkdevs[major].bdops == NULL) {
                                blkdevs[major].name = name;
                                blkdevs[major].bdops = bdops;
                                return major;          }          }
                return -EBUSY;          }
        if (major >= MAX_BLKDEV)
                return -EINVAL;
        if (blkdevs[major].bdops && blkdevs[major].bdops != bdops)
                return -EBUSY;
        blkdevs[major].name = name;
        blkdevs[major].bdops = bdops;
        return 0;
}
```

}

・ ・ ・ ・ ・ ・

The floppy driver (2)

/drivers/block/floppy.c

```
int __init floppy_init(void)
{
   .  .  .  .  .  .
   blk_init_queue(BLK_DEFAULT_QUEUE(MAJOR_NR), DEVICE_REQUEST);
   .  .  .  .  .  .
   if (floppy_grab_irq_and_dma()){
            del_timer(&fd_timeout);
            blk_cleanup_queue(BLK_DEFAULT_QUEUE(MAJOR_NR));
            devfs_unregister_blkdev(MAJOR_NR,"fd");
            return -EBUSY;
   }
  .  .  .  .  .  .
}
```

The floppy driver (3)

/drivers/block/floppy.c

```
void cleanup_module(void)
{
      int dummy;

      devfs_unregister (devfs_handle);
      devfs_unregister_blkdev(MAJOR_NR, "fd");

      blk_cleanup_queue(BLK_DEFAULT_QUEUE(MAJOR_NR));
      /* eject disk, if any */
      dummy = fd_eject(0);
}
```

/drivers/block/floppy.c

```
#ifndef fd_eject
#define fd_eject(x) -EINVAL
#endif
```

Reference

- Understanding the LINUX KERNEL - O’ reilly
- Linux Device Drivers - O’ reilly
- Linux Kernel Internal – Addison Wesley