Embedded Systems Programming

Linux Device Driver Basics (Module 5)

Yann-Hang Lee

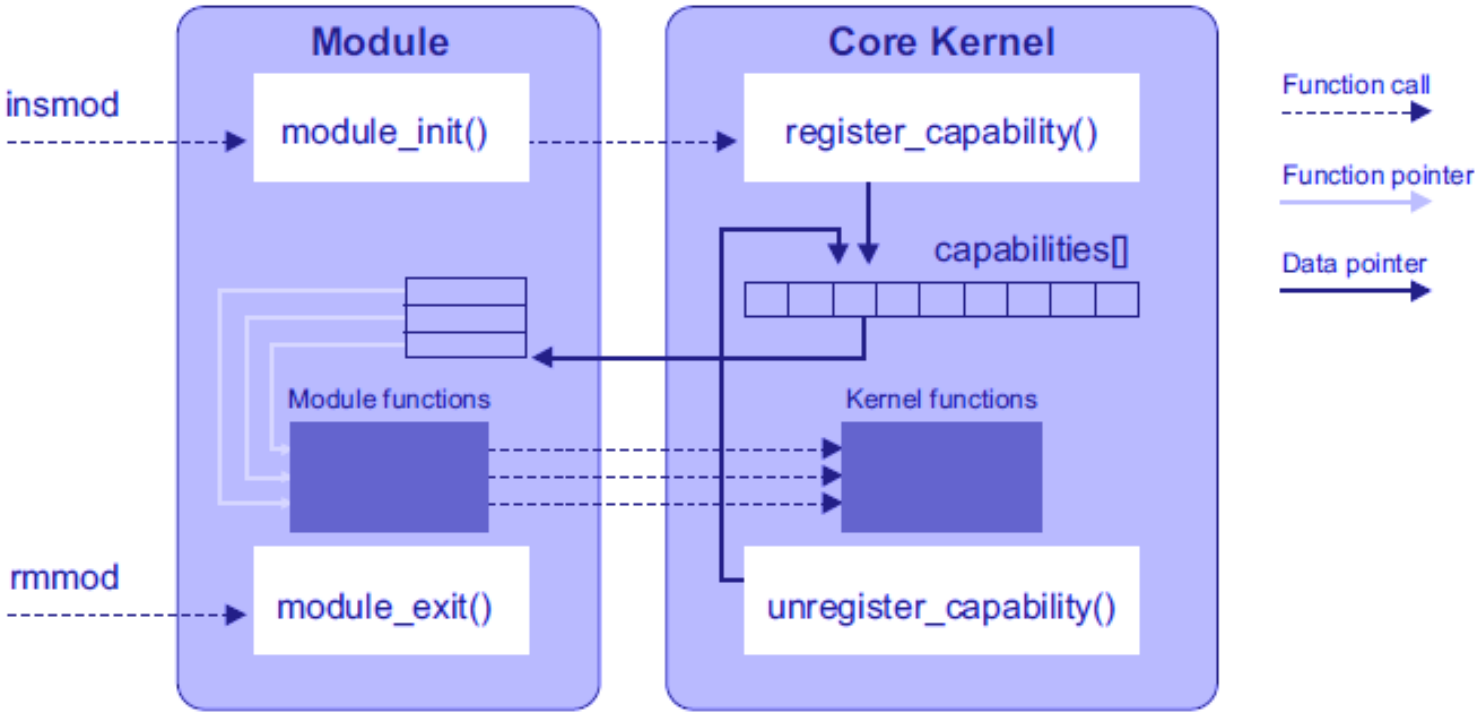
Arizona State University

yhlee@asu.edu

(480) 727-7507

Summer 2014

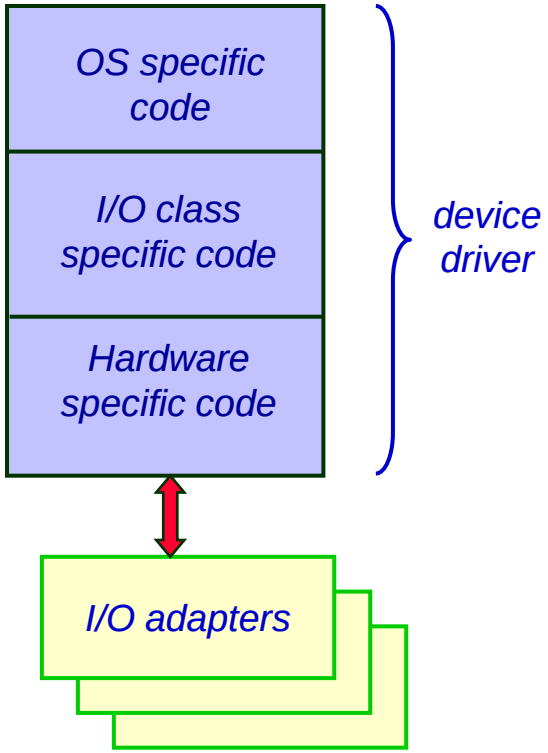
Module Registration in Kernal



- register/unregister (a module object which includes name, properties, and methods)
- invoke methods (function pointers) to perform operations

Device Driver Basics

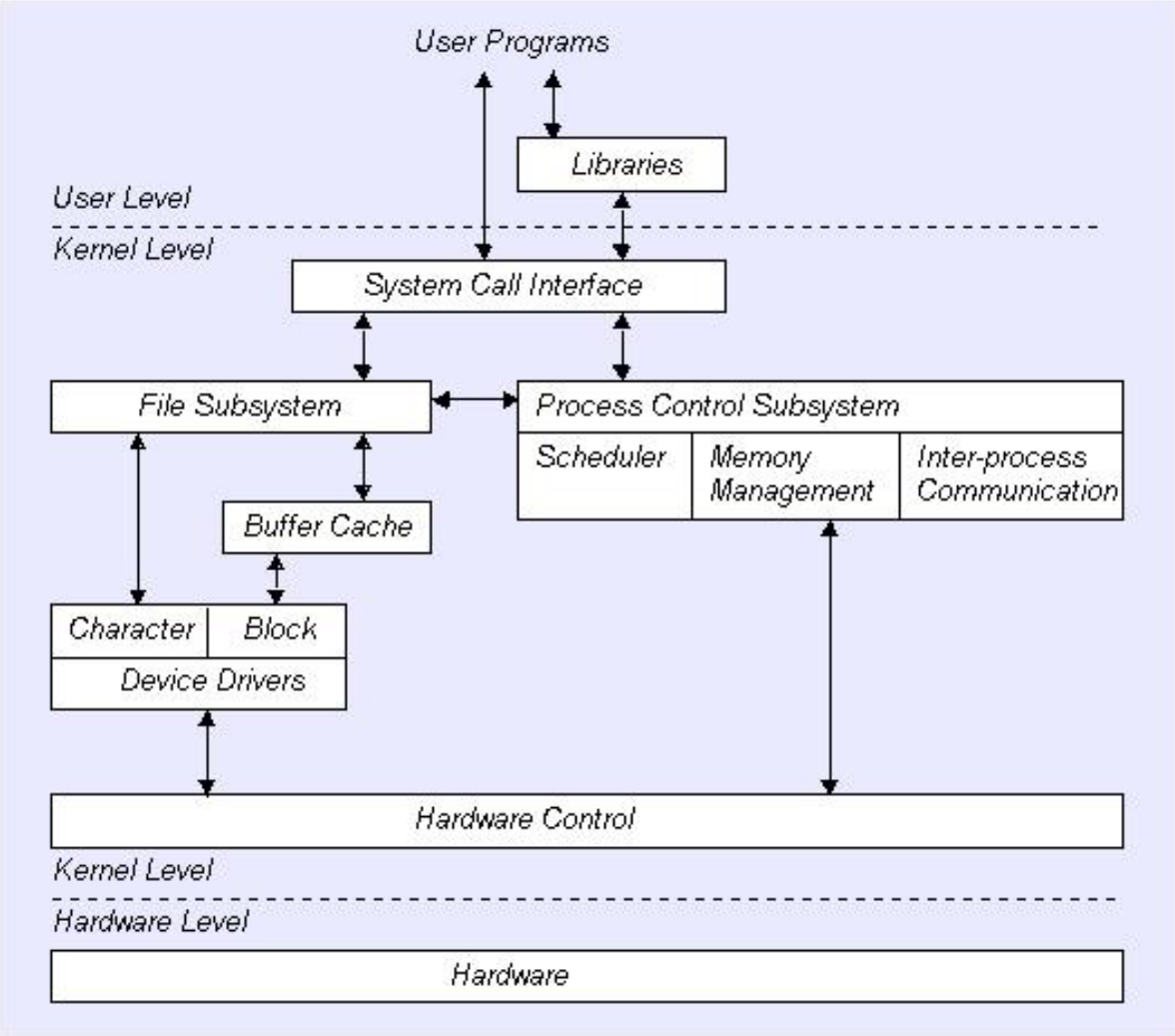
- **Purpose:**
  - a well defined and consistent interface to handle requests for device operations
  - isolate device-specific code in the drivers
- **A software interface to hardware devices**
  - resides in kernel or user spaces
  - implemented as kernel modules
- **Classification**
  - character device (terminal)
  - block (disk) -- with buffer cache
  - network
  - pseudodevice
- **When to call a device driver**
  - configuration, I/O operations, and interrupt



Char and Block Devices

- **Character Devices**
  - Accessed as a stream of bytes (like a file)
  - Typically just data channels or data areas, which allow only sequential access
  - Char devices are accessed by means of filesystem nodes
  - Example: */dev/tty1 and /dev/lp0*
  - Driver needs to implement at least the *open, close, read, and write system calls*
- **Block Devices**
  - provides access to devices that transfer randomly accessible data in fixed-size blocks
  - Examples?
- **Design Challenges**
  - Concurrency, performance, and portability

User Program & Kernel Interface



Char Device

- **cdev -- the kernel internal structure to represents a character device file**
  - defined in <linux/cdev.h>

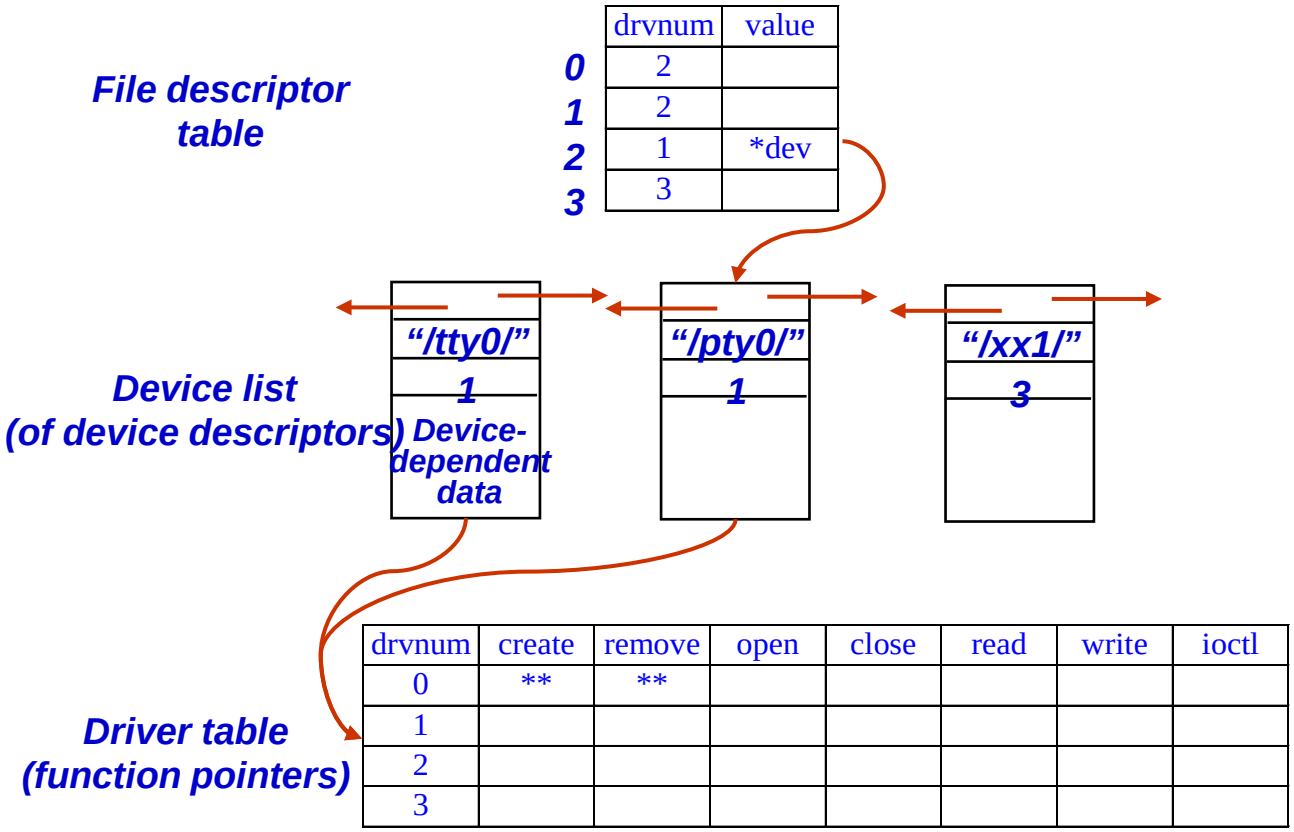
```
struct cdev {
    struct kobject kobj;
    struct module *owner;
    struct file_operations *ops;
    struct list_head list;
    dev_t dev;
    unsigned int count;
};
```

```
struct file_operations ldd_fops = {
    .owner   = THIS_MODULE,
    .llseek  = ldd_llseek,
    .read    = ldd_read,
    .write   = ldd_write,
    .ioctl   = ldd_ioctl,
    .open    = ldd_open,
    .release = ldd_release
};
```

Driver Interface for Character Devices

```
struct file_operations {                                    //   defined in inlcude/linux/fs.h
    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);
    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 *, int datasync);
    int (*fasync) (int, struct file *, int);
    int (*lock) (struct file *, int, struct file_lock *);
    ……..  };
```

Basic Device Driver Structure



- Major number to identify the driver associated with the device
- Minor number provides a way for the driver to differentiate multiple devices.
- *dev_t* type in *<linux/types.h>:* 8bis for major and 8 bits for minor in kernel 2.4, 12bis for major and 20 bits for minor in kernel 2.6.

Register Char Device

```
static dev_t myDev = 0;
static char myName[] = "TestDevice";
static int result = 0;

static int hello_init(void)
{
    if((result = alloc_chrdev_region(&myDev,0,1,myName))!= 0){
                printk("Unable to allocate device number\n");
                return -1;
        }
    printk(KERN_ALERT "My major is %d, minor is %d\n“,
                MAJOR(myDev),MINOR(myDev));
    return 0;
}
static void hello_exit(void)
{
    if(result == 0){
                unregister_chrdev_region(myDev, 1);
        }
}
```

Device Registration

- **Allocate and initialize a *cdev struct***
  - *cdev_alloc or cdev_init*
  - *cdev* gets initilized with a file_opreations structure
- ***cdev_add to register operatons of your device driver***
  - *int cdev_add(struct cdev *dev, dev_t num, unsigned int count)*
- ***cdev_del* — remove a cdev from the system**