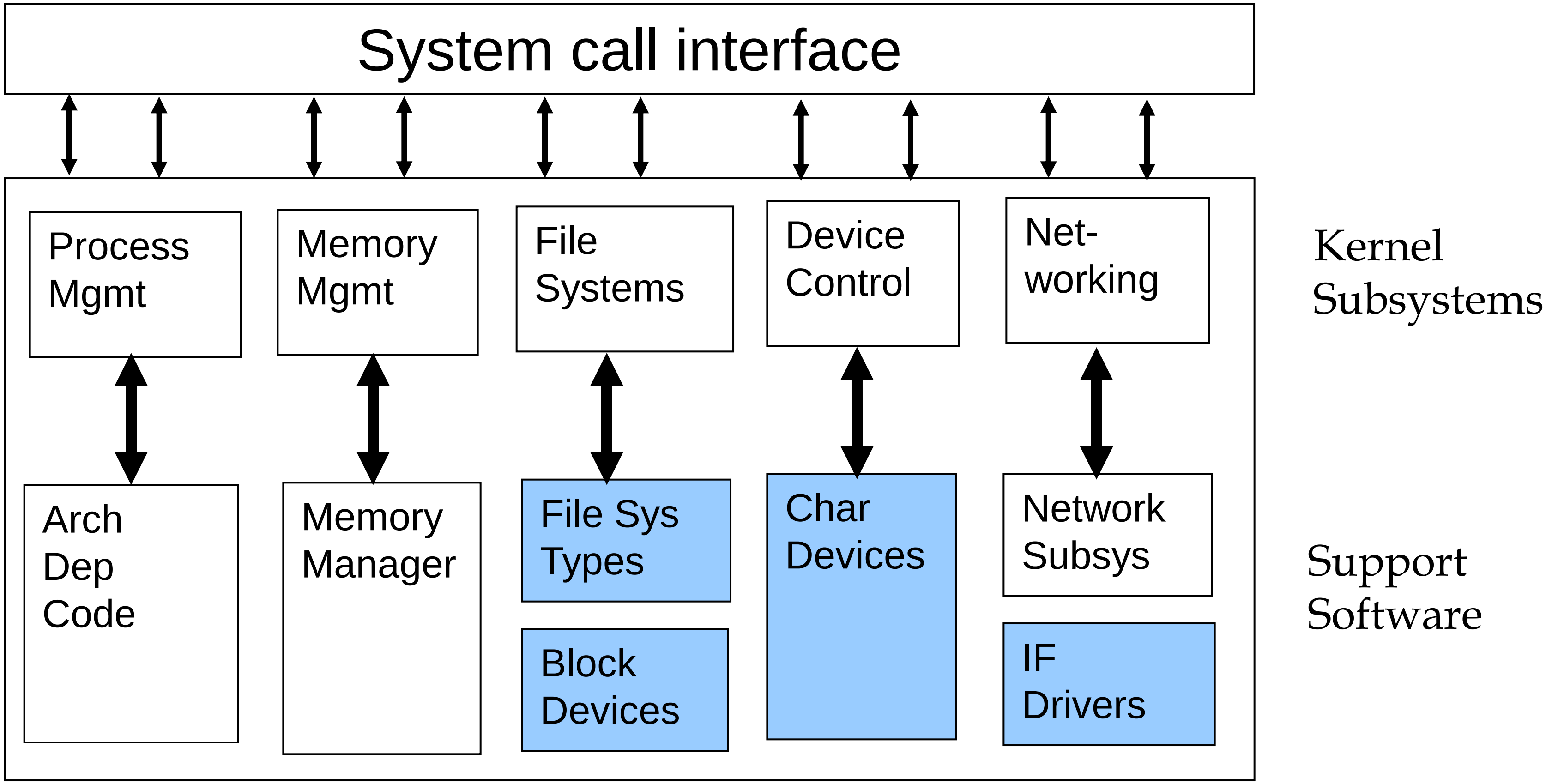
Session #8: Device Drivers

Programming at the Kernel Level

What's a "Device Driver" Anyway?

- A Set of Linkable Functions
- An Independently Loadable Program
- A Seperate Task or Set of Tasks
- A Well-defined driver API
- All of the Above??

Linux Architecture



System call interface

Process Mgmt

Memory Mgmt

File Systems

Device Control

Net-working

Kernel Subsystems

Arch Dep Code

Memory Manager

File Sys Types

Block Devices

Char Devices

Network Subsys

IF Drivers

Support Software

Features that might be implemented as modules 🟦

Linux Device Drivers

- Devices treated like files
  - Everything in Linux is a file
  - Device files in /dev/...
- Device Classes
  - Character
  - Block
  - Pipe
  - Network

Low-level System Calls

```
#include <fcntl.h>
#include <unistd.h>
#include <sys/ioctl.h>

int open (const char *path, int oflags);
size_t read (int file_des, void *data, size_t len);
size_t write (int file_des, const void *data, size_t len);
int close (int file_des);
int ioctl (int file_des, int cmd, …);
```

Problems

- Kernel calls are inefficient
- Data isn't buffered

Standard I/O Library stdio

```
#include <stdio.h>

FILE *fopen (const char *filename, const char *mode);
size_t fread (void *buff, size_t size, size_t n, FILE *stream);
size_t fwrite (const void *buff, size_t size, size_t n, FILE *stream);
int fclose (FILE *stream);
```

getc, putc, and gets families
Formatted I/O: printf, etc.

Problems

- Not good for device control (no ioctl function)
- Not deterministic

Installable Kernel Modules are...

- A way to “extend” the kernel
- Dynamically loaded and unloaded
- Executed at Privilege Level 0
- Useful for:
  - Device Drivers
  - Real-time tasks

Installable Kernel Modules

- Shell Commands
  - insmod <module name> [<param>…]
  - rmmod <module name>
- Required Functions
  - module initialization
  - module cleanup

Installing a Module

Install command

```
insmod my_module.ko my_int = 1 my_string = “Hello”
```

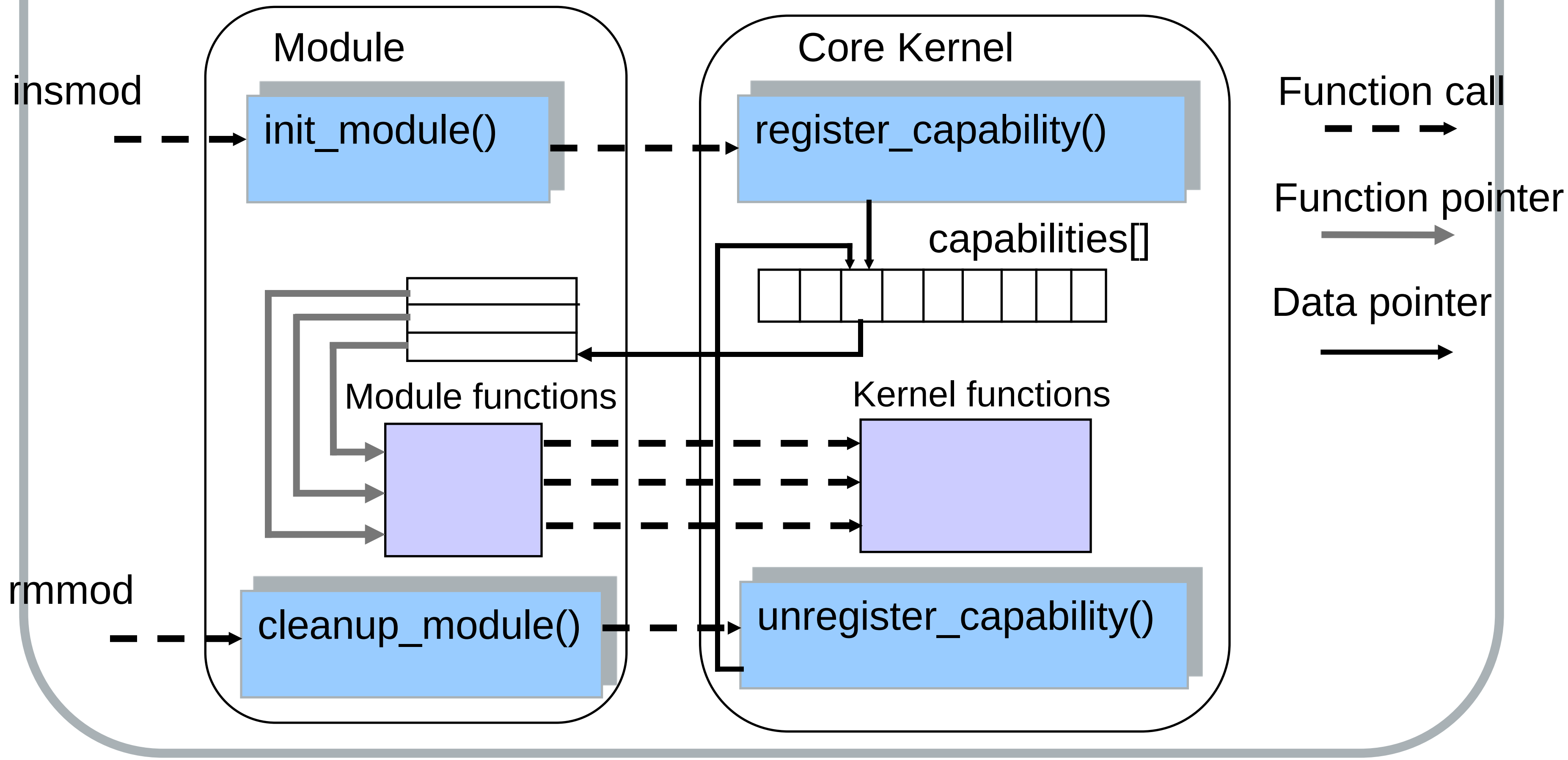
In my_module.c

```
int my_int;
char *my_string;

module_param (my_string, charp, S_IRUGO);
module_param (my_int, int, S_IRUGO);

static int __init my_init (void)
{
        register_capability();
        return 0;
}
module_init (my_init);
```

Linking a Module to the Kernel



A Module Example

- cd ~/drivers/hello
- Look at hello.c and Makefile
- make
- /sbin/lsmod
- In another terminal window, as root user
  - tail -f /var/log/messages
- /sbin/insmod hello.ko my_string="your name" my_int=47
  - What's wrong?
    - MODULE_LICENSE() is missing
- /sbin/rmmod

Kernel modules and the GPL

- GPL = GNU General Public License
- Applications in User Space need not be GPL
- Kernel is GPL
  - Any code built into the kernel is GPL
- Kernel modules need not be GPL
  - Some developers disagree

Allocating Device Numbers

dev_t - a 32-bit number in kernel 2.6
MKDEV(int major, int minor) - creates a dev_t
MAJOR(dev_t dev)
MINOR(dev_t dev)

```
int register_chrdev_region (dev_t first, unsigned int count, char *name);
```

Allocates count device numbers starting at first, if available

or

```
int alloc_chrdev_region (dev_t *dev, unsigned int firstminor,
            unsigned int count, char *name,);
```

Dynamically assigns a major device number for count devices starting at firstminor, usually zero.

Registering a Device

In init_module()

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

struct cdev *my_cdev = cdev_alloc ( );
my_cdev->ops = &my_fops;
```

or

```
void cdev_init (struct cdev *my_cdev, struct file_operations *my_fops);
```

followed by

```
my_cdev.owner = THIS_MODULE;
int cdev_add (struct cdev *my_cdev, dev_t num, unsigned int count);
```

File Operations Table

```
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 (*aio_read)(struct kiocb *, char __user *, size_t, loff_t *);
        ssize_t (*write)(struct file *, const char __user *, size_t, loff _t *);
        ssize_t (*aio_write)(struct kiocb *, const char __user *, size_t, loff _t *);
        int (*readdir)(struct file *, void *, filedir_t);
        unsigned int (*poll)(struct file *, struct poll_table_struct *);
        int (*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 *);
        int (*release)(struct inode *, struct file *);
```

File Operations Table 2

```
          int (*fsync)(struct file *, struct dentry *, int);
          int (*aio_fsync)(struct kiocb *, int);
          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 *);
          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 *, unsigned long);
}
```

struct file

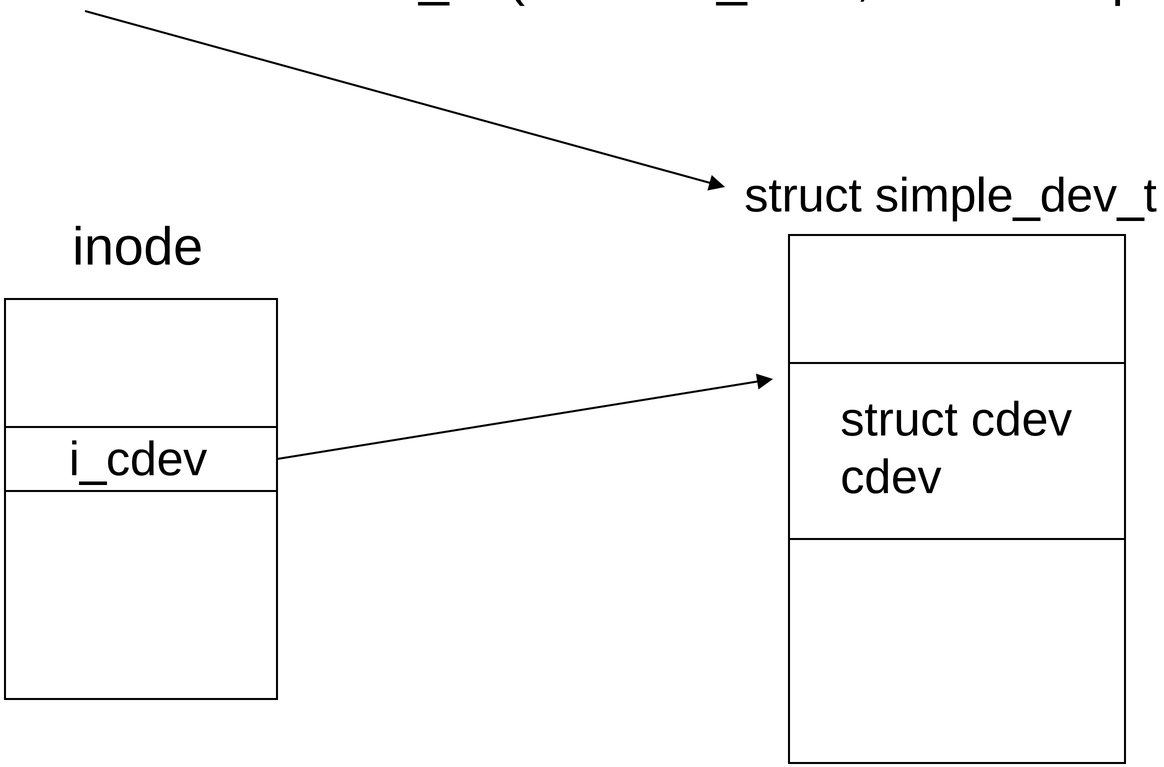
```
struct file {
        struct list_head        f_list;
        struct dentry           *f_dentry;
        struct vsfmount         *f_vsfmount;
        struct file_operations *f_op;
                    .
                    .
        void *private_data;
                    .
}
```

Our first driver

- cd ~/drivers/simple_char
- Look at simple_char.c
  - parameters, near top
  - simple_init_module, near bottom
  - file_operations
  - simple_open, back near top

container_of() macro

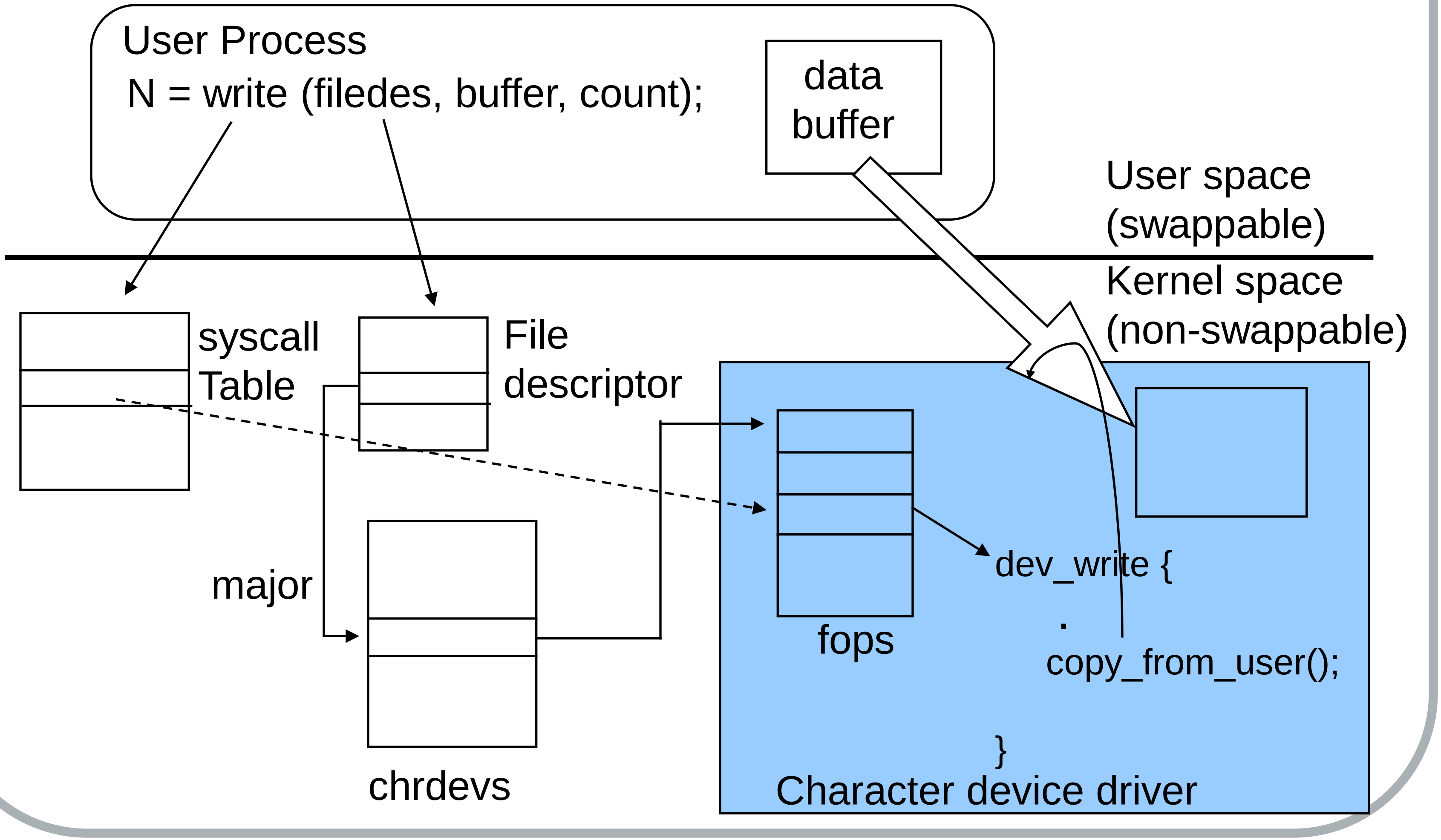
```
dev = container_of (inode->i_cdev, struct simple_dev_t, cdev);
```



Make and install simple_char

- make
- Look at simple_load script
  - Loads the driver module
  - Creates device nodes based on devices's major number
- ./simple_load
- Try
  - cp simple_char.c /dev/simple0
  - cat /dev/simple0

Driver Processing--Write



Mutual exclusion--mutexes

- Only one process can access data at a time
- mutex guarantees exclusive access
  - void mutex_init (struct mutex *);
  - void mutex_unlock (struct mutex *);
  - void mutex_lock (struct mutex *);
  - int mutex_lock_interruptible (struct mutex *);

Registering a Device--the “old way”

In init_module()

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

Registers driver at chrdevs[major]

or

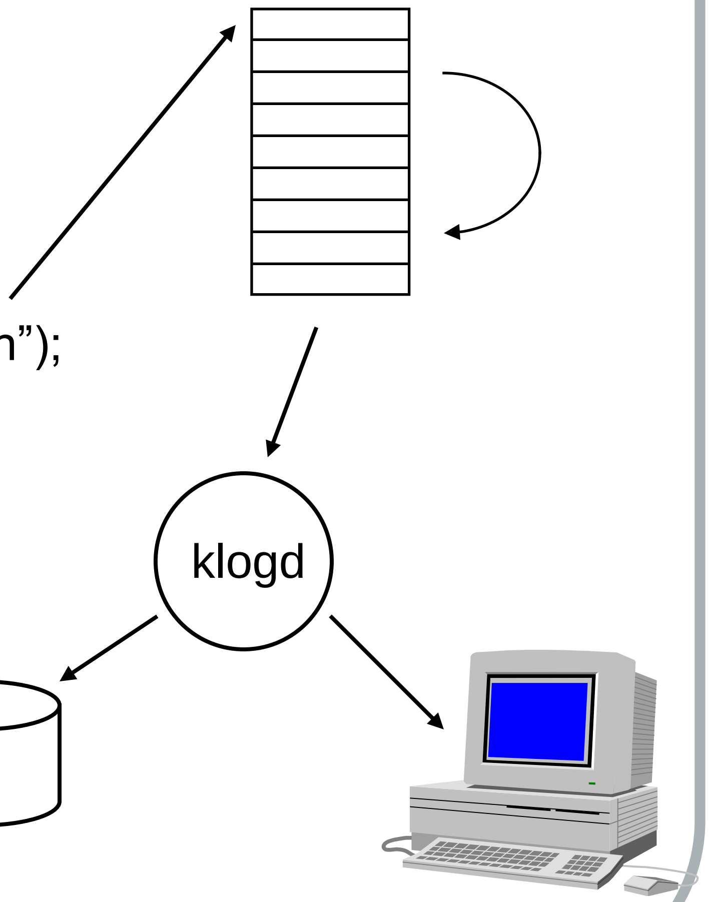
```
int register_blkdev (unsigned int major, const char *name,
            struct file_operations *fops);
```

Registers driver at blkdevs[major]

printk()

```
driver_read(…)
{
                .
                .
    printk (KERN_DEBUG “In read function\n”);
                .
                .
}
```

loglevel



printk macro

In a header file

```
#ifdef DEBUG
#define PDEBUG (fmt, args, …) printk (KERN_DEBUG fmt, ## args);
#else
#define PDEBUG (fmt, args, …)      // nothing
#endif

driver_read(…)
{
            .
            .
   PDEBUG ( “In read function\n”);
            .
            .
}
```

The /proc Filesystem

```
ls -l /proc
total 0
dr-xr-xr-x    3 root      root             0 Aug 25 15:23 1
dr-xr-xr-x    3 root      root             0 Aug 25 15:23 2
dr-xr-xr-x    3 root      root             0 Aug 25 15:23 3
dr-xr-xr-x    3 bin       root             0 Aug 25 15:23 303
dr-xr-xr-x    3 nobody    nobody           0 Aug 25 15:23 416
dr-xr-xr-x    3 daemon    daemon           0 Aug 25 15:23 434
dr-xr-xr-x    3 xfs       xfs              0 Aug 25 15:23 636
dr-xr-xr-x    4 root      root             0 Aug 25 15:23 bus
-r--r--r--    1 root      root             0 Aug 25 15:23 cmdline
-r--r--r--    1 root      root             0 Aug 25 15:23 cpuinfo
-r--r--r--    1 root      root             0 Aug 25 15:23 devices
-r--r--r--    1 root      root             0 Aug 25 15:23 filesystems
dr-xr-xr-x    2 root      root             0 Aug 25 15:23 fs
dr-xr-xr-x    4 root      root             0 Aug 25 15:23 ide
-r--r--r--    1 root      root             0 Aug 25 15:23 interrupts
-r--r--r--    1 root      root             0 Aug 25 15:23 ioports
```

Registering a /proc File

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

/proc read function

```
        int (*read_procmem) (struct file *filp, char __user *buf, size_t
                count, loff_t *pos);
```

Registering a /proc file

```
        struct proc_dir_entry *proc_create (const char
        *name, mode_t mode, struct proc_dir_entry *parent,
        const struct file_operations *fops);
```

Removing a /proc file

```
        void remove_proc_entry (const char *name, struct
proc_dir_entry *parent);
```

ioctl--Another way to get info from driver

User Space

```
int ioctl (int fd, int cmd, …);
```

Driver Method

```
int (*ioctl) (struct file *filp, unsigned int cmd,
        unsigned long arg);
```

- Easier to code in driver than /proc file
- Requires User Space program to read or write
- Can leave in production code because nobody knows it's there

Choosing ioctl() commands

type - “Magic” number, 8 bits

_IOC_TYPEBITS

number - Ordinal number for command, 8 bits

__IOC_NRBITS

direction - From application’s viewpoint, bit mask

_IOC_NONE

_IOC_READ

_IOC_WRITE

_IOC_READ | _IOC_WRITE (both ways)

size - of item being transferred

Using pointer arguments

```
int access_ok (int type, const void *addr,
                    unsigned long size);

put_user (datum, ptr)
__put_user (datum, ptr)

get_user (local, ptr)
__ get_user (local, ptr)

int capable (int capability);
```

Blocking I/O

- What if…
  - No data available (yet) on read?
  - Buffer full on write?
- Process must "sleep" until condition is satisfied
- Use wait queue (head)

Wait queue API

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

Create a wait queue

```
        DECLARE_WAIT_QUEUE_HEAD(name);
```

or

```
        wait_queue_head_t my_queue;
        init_waitqueue_head (&my_queue);
```

Sleep on a queue

```
        wait_event (queue, condition)
        wait_event_interruptible (queue, condition)
        wait_event_timeout (queue, condition, timeout)
        wait_event_interruptible_timeout (queue, condition, timeout)
```

Wake up a process on a queue

```
        wake_up (&queue);
        wake_up_interruptible (&queue);
```

Putting a process to “sleep”

```
DECLARE_WAIT_QUEUE_HEAD(my_wait);
int flag = 0;
int buff_count;

ssize_t sleepy_write (struct file *file, char *buff, int count, loff_t *offset)
{
        copy_from_user (buffer, buff, buff_count = count);
        flag = 1;
        wake_up_interruptible (&my_wait);
        return count;
}
ssize_t sleepy_read (struct file *file, char *buff, int count, loff_t *offset)
{
        wait_event_interruptible (my_wait, flag != 0);
        copy_to_user (buffer, buff, buff_count);
        return buff_count;
}
```

Non-blocking operations

- O_NONBLOCK flag in filp->f_flags
- If set, functions return -EAGAIN if data transfer not possible
- Affects open(), read(), and write() file operations

The seq_file interface

```
void *seq_start (struct seq_file *sfile, loff_t *pos);
void *seq_next (struct seq_file *sfile, void *v, loff_t *pos);
void seq_stop (struct seq_file *sfile, void *v);
int seq_show (struct seq_file *sfile, void *v);

int seq_open (struct *file, struct seq_operations *seq_ops);
```

Review

- Device drivers
  - User space APIs
  - Device classes
- Loadable kernel modules
  - Modules and the GPL
- Basic character device driver
  - Debugging – printk(), /proc files, ioctl
- Blocking I/O