Linux Device Drivers

- Registering your device
- File operations Table
- Synchronization
- Interrupt handling

Hacking the Kernel

- The kernel is BIG
  - It is so big that grep just isn't good enough anymore
  - a souce grep tool is needed
  - I HIGHLY recommend cscope
    - it integrates with both vim and emacs
    - I can help you get set up with the vim
- Find a way not to always reboot your machine
  - use kdb or gdb with a serial connection
  - or use user mode linux
  - or use the ioclt trick i'll show later

Linux Device Scheme

- you can create a device “pointer” by using mknod
  - man mknod for more information
  - devfs can do this for you
- Each node has a major and minor number
- Each driver registers a table to functions to a major number
  - this table is a struct file_operations instance
- minor numbers can be used by the driver to differenciate between mulitiple devices
- /proc/devices has all registered major numbers

Linux Module scheme

- insmod/modprobe
  - modprobe seaches on dependancies
  - all it does it check the version of the kernel the module was compiled against and tries to resolve symbols
  - there are very few symbols the kernel actually exports
    - look for EXPORT_SYMBOL in the source
  - errors can occur with version numbers, so watch out
  - /proc/ksyms has all the symbols you can link against
- if your program will not insmod due to symbols and you aren't using any undefined symbols, then you need to tell gcc to optimize a little (-O1)

loading and unloading a driver

- insmod some_object_file.o
  - init_module() is called
- rmmod some_object_file
    - notice there is no .o
  - cleanup_module is then called
  - remember to clean everything up
    - or else your kernel will panic when it tries to access your stuff that has been removed

Registering your device

- int register_chrdev(int major, char *name, fops)
  - This is the old way to do it, but it still works
  - it retuns negative on failue
  - if major is zero it allocates you a major and returns it
- devfs_register(dir, char *name, flags, major, minor, more_flags, fops, void *private_data)
  - parameters more complicated, but it creates device nodes in /dev with associated major and minor numbers for you with minimal hassle
  - the same major and minor numbers remain associated with your device even if you remove and reinsert it

common struct file_operations fields

- read
- write
- poll
- ioctl
- open
- release

Code examples

Code and Slides Available at:
www.acm.uiuc.edu/sigops/lkm_tut

Synchronization

- wait queues (wait_queue_head_t)
- wait_on_interrupatble(wait_queue_head *)
  - This may lead to race conditions if you check the condition, then go to sleep and in the meantime someone wakes up the queue
- wait_event_interruptible(wait_queue_head, condition)
  - this is a macro, so you dont need to pass a pointer
  - returns 0 once done waiting
  - returns -ERESTARTSYS if an interrupt happened

Synchronization

- semaphores
  - think of them as the number of things on a shelf
  - up puts something back up on the shelf
  - down taks an item down, or waits if there are 0
  - down_interruptable is the one we want

Synchronization

- spin_locks
  - basically a tight while loop of waiting
  - only used for very small sections of code
  - Used to sync with other processors
  - usually used with interrupts off
    - if interrupted with a spin_lock and that interrupt handler tries to aquire the lock, then you have deadlocked that processor
  - spin_lock_init(spinlock_t *);
  - use spin_lock_irqsave(&my_lock, flags)
  - and spin_lock_irqrestore(&my_lock, flags);
    - these are macros, flags cannot be passed around
  - #include <asm/spinlock.h>

Synchronization question

- Since it makes sence to disable interrupts in the interrupt handler we use spin_lock_irqsave
- What happens if we use regular spin_lock somewhere else like in a read method where interrupts are still on???

Synchronization

- Read write spinlocks
  - same as normal spinlocks, but many readers can have the lock at the same time
  - rwlock_t ny_lock = RW_LOCK_UNLOCKED;
    - this is how we initilize it
  - read_lock_irqsave(&my_lock, flags)
  - write_lock_irqsave
  - read_unlock_irqrestore
  - write_lock_irqrestore
  - I point these out because you may try to reinvent the wheel because these come in handy
  - don't worry about starving the writer
    - Spin locks are for SHORT LOCKS ONLY

Synchronization

- Cool atomic operations
  - int test_and_set_bit(int bit_no, void *addr)
  - int test_and_clear_bit(int bit_no, void *addr)
  - int test_and_change_bit(...)
  - #include <asm/bitops.h>

ioctl's

- ioctls are commands you can preform on file descriptors that do not fit into the read/write/poll model
  - we can use ioctl's for turning something on/off for the whole device or just one instace of it
  - we can also use ioctl's to remove modules that are stuck

stuck modules

- there is a usage count on each module
  - the module can only be removed when the count is zero
  - instead of not incremetning and decrementing the count we can reset it if it wrong

ioctl to unstick a stuck module

```
int basic_ioctl (struct inode *inode, struct file *filp,
        unsigned int cmd, unsigned long arg)
{
    int ret = 0;
    switch(cmd) {
        case TEMP_IOCHARDRESET:
            while(MOD_IN_USE)
                MOD_DEC_USE_COUNT;
            MOD_INC_USE_COUNT; // this is because this is open
            break;
        default:
            return -ENOTTY;
    }
    return ret;
}
```