Final Review

ECE 373

Prelims

- Questions?

- The end is near!!!

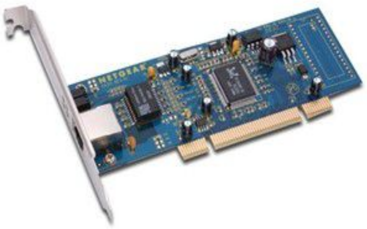
What's fair game

- The entire term
- Focus more on material after mid-term
- Be prepared for anything though
- Open book, open notes
- Touching on high points today

Bits and Pieces

- Minimal callback hooks and compile headers
  - Basic #includes
  - MODULE_LICENSE(lic) – legal strings
  - __init, __exit
  - module_init(func), module_exit(func)
- Time values
  - Jiffies, HZ

PCI



- Communication/connection method for devices
- Devices use both port mapped and memory mapped I/O (focus on MMIO only for final)
- BAR – base address register
  - Starting address for device memory map
  - Driver uses `writel()` and `readl()` to access device registers with MMIO
- Callbacks
  - Probe, remove

Order matters

Instance 1	Instance 2	Value
read very_important_count		5
add 5 + 1 = 6		6
write very_important_count		6
	read very_important_count	6
	Add 6 + 1 = 7	7
	write very_important_count	7

Instance 1	Instance 2	Value
read very_important_count		5
	read very_important_count	5
add 5 + 1 = 6		6
	Add 5 + 1 = 6	6
write very_important_count		6
	write very_important_count	6

Atomic action

- CPU instructions for atomic increment, decrement, test_and_set
- All cores must coordinate CPU cache – expensive
  - `atomic_inc(x)`
  - `atomic_inc_and_test(x)`
  - `set_bit(n, *s)`
  - `clear_bit(n, *s)`
  - `test_bit(n, *s)`

Locks and synchronization

- Mutex's
- Spin locks
- Semaphores
- Completions
- RCU
- When is it safe to lock?

Delayers

- `mdelay(), udelay(), ndelay()`
- While loop on a counter, no scheduler action
  - http://lxr.linux.no/#linux+v2.6.38/include/linux/delay.h#L46
  - http://lxr.linux.no/#linux+v2.6.38/arch/x86/lib/delay.c#L116
- Only way to get short period delays
- Not very friendly for long periods
- Could block jiffies update if interrupts disabled

Sleepers

- `msleep(), usleep()`
- Loop that can be scheduled out
- Friendly for long periods
- Not good to use in interrupt context (both SOFTIRQ and HARDIRQ contexts)

WARN, BUG

- Code warnings
  - `BUG(), BUG_ON(expr)`
  - `WARN(), WARN_ON(expr), WARN_ONCE()`
- BUG stops kernel thread
- Both produce stack dump output
- WARN does not stop kernel thread

DMA in Linux

- Function APIs exist for each driver type to control DMA
- DMA consists of mapping memory for DMA, unmapping when finished
- Mapping memory means pinning it down, not allowing it to be swapped out by memory manager
- DMA deals with physical address (or bus address)

What is a userspace driver?

- A way to drive hardware from outside kernel
- Accesses various resources
  - I/O ports
  - Memory regions
  - Control interfaces
- Resources presented by kernel scanning hardware in standardized ways
  - e.g. PCI bus scan
- Not a .ko file!

Mapping a file is as simple as...

- `open()` the file
- `mmap()` the file
- Inspect the memory region returned from `mmap()`
- `munmap()` the file
- `close()` the file

Why use a userspace driver?

- Quick prototyping
- Free from kernel ABI changes
  - Application Binary Interface
- Typically doesn't blow up the machine
  - Constrained to mapped space
- Ease of use
  - Can be written in most compiled and scripted languages
- May not have ability to change the kernel

Types of OS's in the wild

- Single-user (Phones, PC's)
- Multi-user (Servers, mainframes, "cloud")
- Real-time (Stop lights, shuttle navigation)
- Embedded (Watch, routers, car engine, mp3)

HW Interrupts

- Hardware wants attention
  - Data waiting, might be time-sensitive
- Interrupt handlers
  - Temporarily take over current thread, whether kernel or user
  - Can't be scheduled, can't sleep

Interrupts: Be quick!

- Blocking other interrupt handling and user job
- Grab HW info, stash away for later
- Wake up driver code with worker thread or waiting on a lock
- Don't call code that might sleep
  - *sleep(), kmalloc(), other I/O functions
- Locks?
  - Be careful...

Character drivers

- Typical types of char devices
  - Mice
  - Keyboards
  - Printers
- Stream data to and from device, no set size
- Links file_operations through struct cdev

Beginning to hook it all up

- Structure "`file_operations`" provides function pointers into system call interface
- Main linkage into /dev filesystem for char drivers
- Driver does not need to implement all of them
- Behaves similarly to object-oriented code

Snippet of file_operations

- `struct file_operations {`
  - `struct module *owner;`
  - `int (*open) (struct inode *, struct file *);`
  - `int (*release) (struct inode *, struct file *);`

  -
    ```
    ssize_t (*read) (struct file *,
                char __user *,
                size_t, loff_t *);
    ```

  -
    ```
    ssize_t (*write) (struct file *,
                 const char __user *,
                  size_t, loff_t *);
    ```
- `}`

Block drivers

- Drivers that transfer fixed-block sizes
- Primarily for disk and storage devices
- Block I/O is mostly because of how disks are laid out
- Spindle drives would write in small clusters
- Clusters create blocks
- Not as necessary on modern drives

The Kconfig framework

- Complex infrastructure to enable/disable kernel features
- Used to manipulate makefiles
- Layered, like an onion (and stinky too!)
- Can implement multiple dependencies

Building and booting a kernel

- make, make modules_install, make install
- GRUB
- vmlinuz and vmlinux images
- Modules installation
- Initial RAM disks (initrd)

Add your own code!

- How to add new pieces to the kernel
- Editing/adding Kconfig
- Creating your makefile
- Enjoying your time with maintainers...

Descriptors

- What is a descriptor?
  - Hardware field describing what work to do
  - Hardware field describing what work was done
- Carries bits and fields
- Carries pointers to buffers needing to be DMA'd into hardware
- Carries pointers to buffers DMA'd out of hardware

Filling it out

```
do {
        buffer_info = &tx_ring->buffer_info[i];
        tx_desc = E1000_TX_DESC(*tx_ring, i);
        tx_desc->buffer_addr =  cpu_to_le64(buffer_info->dma);
        tx_desc->lower.data = cpu_to_le32(txd_lower |
                                          buffer_info->length);
        tx_desc->upper.data = cpu_to_le32(txd_upper);

        i++;
        if (i == tx_ring->count)
                i = 0;
} while (--count > 0);

tx_desc->lower.data |= cpu_to_le32(adapter->txd_cmd);
```

That should do it...