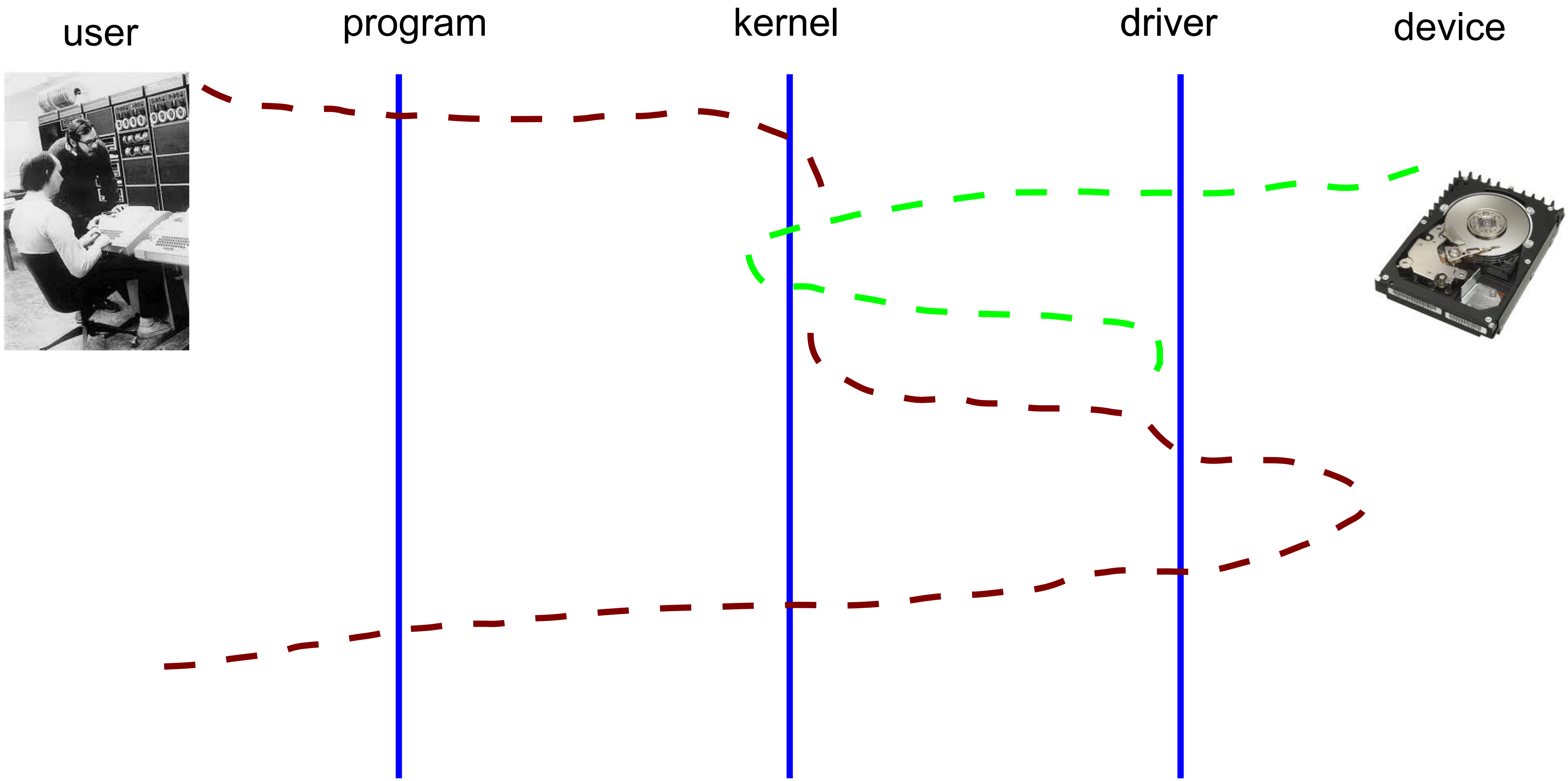
Interrupts and Handlers

ECE 373

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
    - no msleep(), no mutex(), careful with kzalloc()...
- Interrupts interrupting interrupts?

Thread, interrupted



user

program

kernel

driver

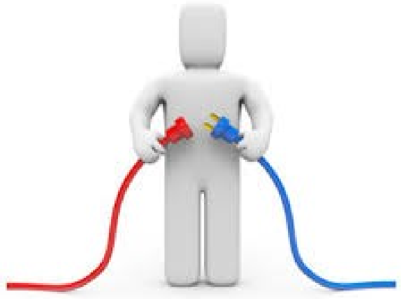
device

Basic interrupt flow

- Allocate interrupt and set up handler
  - specific to what your driver needs to process
  - USB device would clean URBs, possibly refill if needed
  - Network device will process received packets
  - Disk device will process data blocks
- Tell device what interrupt line/number to use
- Run...
- Disable interrupt in device then OS when done

Interrupt Types

- Legacy
  - Hardware based, individual wires, limited availablity
  - Chaining of handlers on same interrupt – coordination of handlers
- MSI – PCI message signaled interrupt
  - No more chaining,
  - Each PCI device gets its own interrupt
- MSI-X – MSI eXtended
  - Many interrupts per PCI device
- SW interrupts
  - SW triggers, Timers



Basic Wiring

- Request an IRQ using device info

```
err = request_irq(pdev->irq, ece_irq_handler, 0, "ece_int",
data);
```

http://lxr.free-electrons.com/source/include/linux/interrupt.h

  - irq: interrupt index in OS tables
  - ece_irq_handler: ptr to interrupt handler function
  - 0: flags
    - IRQF_SHARED, IRQF_SAMPLE_RANDOM, IRQF_TRIGGER_*
  - "ece_int": name seen in last column of /proc/interrupts
  - data: "magic cookie" data passed into intr handler
    - Same as in the timer handler

Legacy and MSI Setup

- Legacy
  - Request an IRQ using PCI device info

```
err = request_irq(pdev->irq, ece_irq_handler, 0, "ece_int",
data);
```

- MSI
  - Enable MSI for the device
  - Request an IRQ using PCI device info

```
pci_enable_msi(pdev);

err = request_irq(pdev->irq, ece_irq_handler, 0, "ece_msi",
data);
```

MSI-X Setup

MSI-X needs an array for many interrupt vectors

- Prep an array of msix data structures

  ```
  entries = kcalloc(num_vectors, sizeof(struct
  msix_entry), GFP_KERNEL);
  ```

- Enable MSI-X for the device

  … and get block of interrupt vectors

  ```
  pci_enable_msix(pdev, entries, num_vectors)
  ```

- Request an IRQ for each msix entry

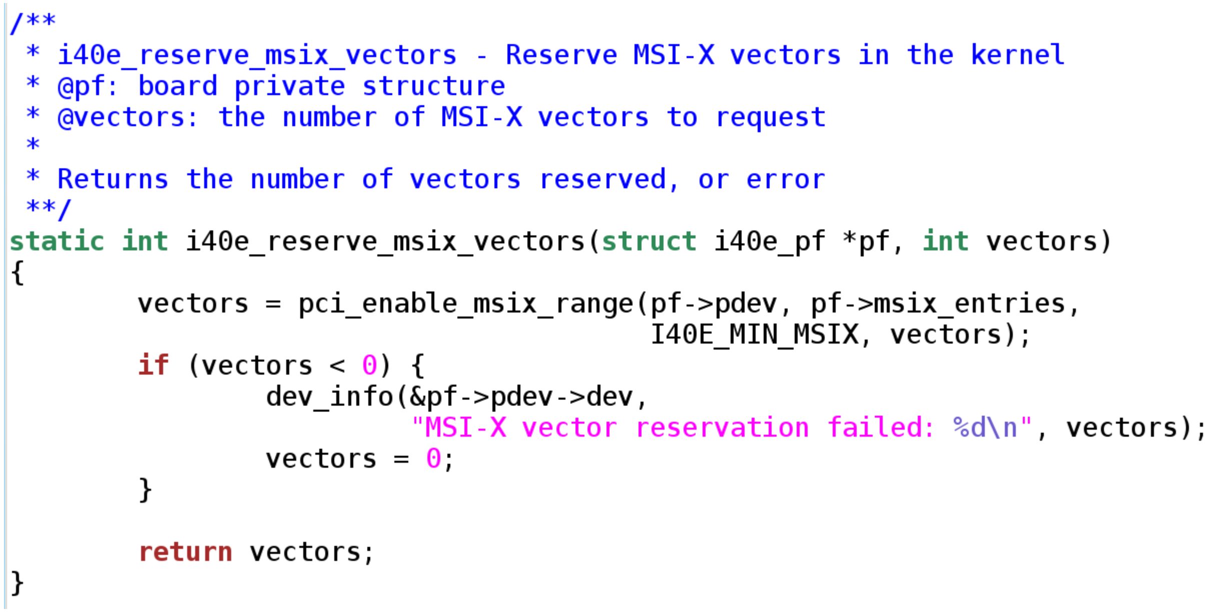
```
for (i = 0; i < num_vectors; i++)

    err= request_irq(entries[i].vector,
    ece_irq_handler, 0, "ece_msix", i);
```

Magic cookie,
irq context

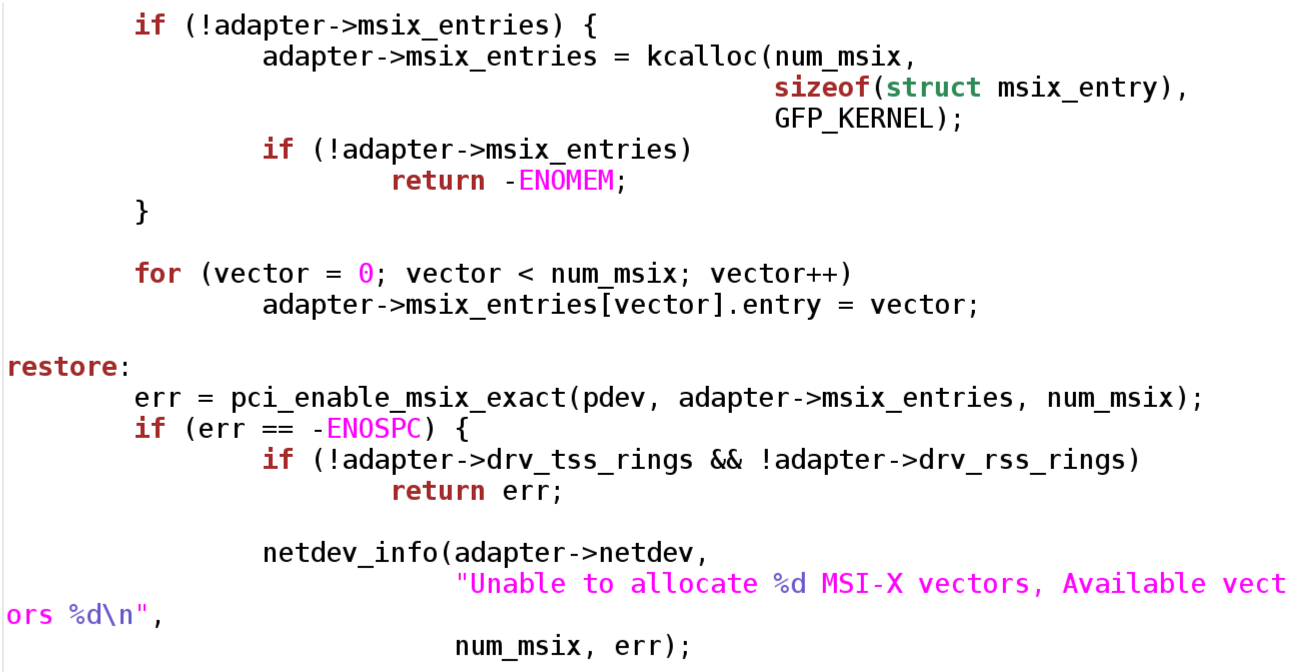
Example: MSI-X

```
struct msix_entry *msix_list;
char v_name[16];

msix_list = kcalloc(v_num, sizeof(struct msix_entry), GFP_KERNEL);
if (NULL == msix_list)
        return NULL;

/* prep the vector array */
for (v = 0; v < v_num; v++)
        msix_list[v].entry = v;

while (v_num >= least_vectors_needed) {

        /* try to get a block of vectors */
        err = pci_enable_msix(pdev, msix_list, v_num);

        if (0 == err)           /* success */
                break;
        else if (err < 0)       /* nasty failure, quit now */
                v_num = 0;
        else                    /* err == num vectors we should try */
                v_num = err;
}

/* failed, so clean up and return */
if (v_num < least_vectors_needed) {
        kfree(msix_list);
        msix_list = NULL;
        return NULL;
}

/* init all the vectors */
for (v = 0; v < v_num; v++) {
        snprintf(v_name, sizeof(v_name), "ece373_v_%02d", v);
        err = request_irq(msix_list[v].vector, ece_irq_handler,
                          0, v_name, ece_data);
}
ece_data->msix_list = msix_list;
```

Another example: MSI-X



```
/**
 * i40e_reserve_msix_vectors - Reserve MSI-X vectors in the kernel
 * @pf: board private structure
 * @vectors: the number of MSI-X vectors to request
 *
 * Returns the number of vectors reserved, or error
 **/
static int i40e_reserve_msix_vectors(struct i40e_pf *pf, int vectors)
{
        vectors = pci_enable_msix_range(pf->pdev, pf->msix_entries,
                                        I40E_MIN_MSIX, vectors);
        if (vectors < 0) {
                dev_info(&pf->pdev->dev,
                         "MSI-X vector reservation failed: %d\n", vectors);
                vectors = 0;
        }

        return vectors;
}
```



```
        if (!adapter->msix_entries) {
                adapter->msix_entries = kcalloc(num_msix,
                                                sizeof(struct msix_entry),
                                                GFP_KERNEL);
                if (!adapter->msix_entries)
                        return -ENOMEM;
        }

        for (vector = 0; vector < num_msix; vector++)
                adapter->msix_entries[vector].entry = vector;

restore:
        err = pci_enable_msix_exact(pdev, adapter->msix_entries, num_msix);
        if (err == -ENOSPC) {
                if (!adapter->drv_tss_rings && !adapter->drv_rss_rings)
                        return err;

                netdev_info(adapter->netdev,
                            "Unable to allocate %d MSI-X vectors, Available vect
ors %d\n",
                            num_msix, err);
```

Handler

- Interrupt handler is called with cookie argument for context
- Handler often needs to check interrupt cause register in device

```
int_bits = readl(hw_addr + INT_CAUSE)
```

- Returns
  - IRQ_NONE – not mine, shared by some other handler
  - IRQ_HANDLED – done and handled
  - http://lxr.free-electrons.com/source/drivers/net/ethernet/intel/i40e/i40e_main.c#L3070

Handler be quick!

- Blocking other interrupt handling and user jobs
- Grab HW info, stash away for later
- Don't call code that might sleep
  - sleep(), malloc(), other I/O functions
  - kmalloc GFP_ATOMIC maybe
  - Scheduler can't put interrupt handlers in wait queue
- Locks?
  - no mutex or semaphore
  - atomics and completions okay
- Use Top half / Bottom half concept
  - Wake up driver code with worker thread or waiting on a completion

Handler Code

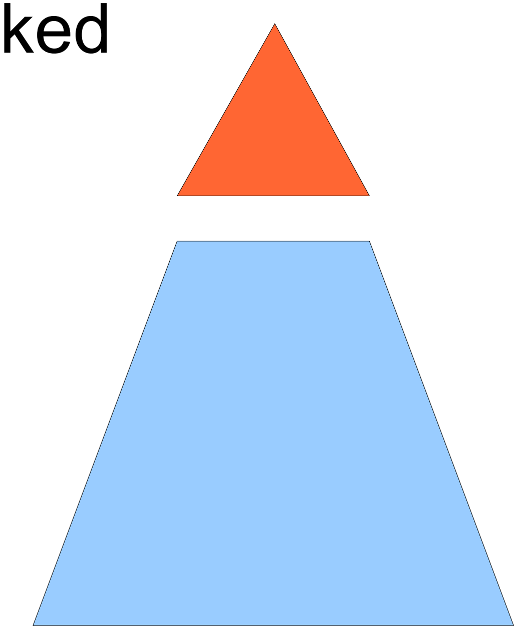
```c
#define REG_IRQ_CAUSE_READ_DONE    0x0001
#define REG_IRQ_CAUSE_WRITE_DONE   0x0002
#define REG_IRQ_CAUSE_ON_FIRE      0x0004

static irqreturn_t ece_irq_handler(int irq, void *data)
{
        struct ece_data_t *ece_data = data;
        u32 cause;

        /* no printing here - can't do anything that might sleep */
        irq_info = irq;

        cause = readl(hw_addr + REG_IRQ_CAUSE);

        switch (cause) {
        case REG_IRQ_CAUSE_READ_DONE:
        case REG_IRQ_CAUSE_WRITE_DONE:
                schedule_work(ece_data->io_task);
                break;
        case REG_IRQ_CAUSE_ON_FIRE:
                schedule_work(ece_data->shutdown_task);
                break;
        }

        return IRQ_HANDLED;
}
```

Cleanup

- Legacy
  - Turn off the device interrupts
    - free_irq(pdev, cookie)
- MSI
  - Turn off device interrupts
    - pci_disable_msi(pdev)
- MSI-X
  - Turn off device interrupts and delete array
    - pci_disable_msix(pdev)
    - kfree(entries)

Top Half / Bottom Half

- Split the activity into quick reaction part and work part
- Top Half
  - Runs in interrupt context, interrupts masked
  - Stashes away info for bottom half
  - Releases HW interrupt context
- Bottom Half
  - Processes received information
  - Sleeps, locks, waits, I/O, etc
  - Re-enables interrupt
  - Workqueue, Tasklet, I/O thread waiting for completion



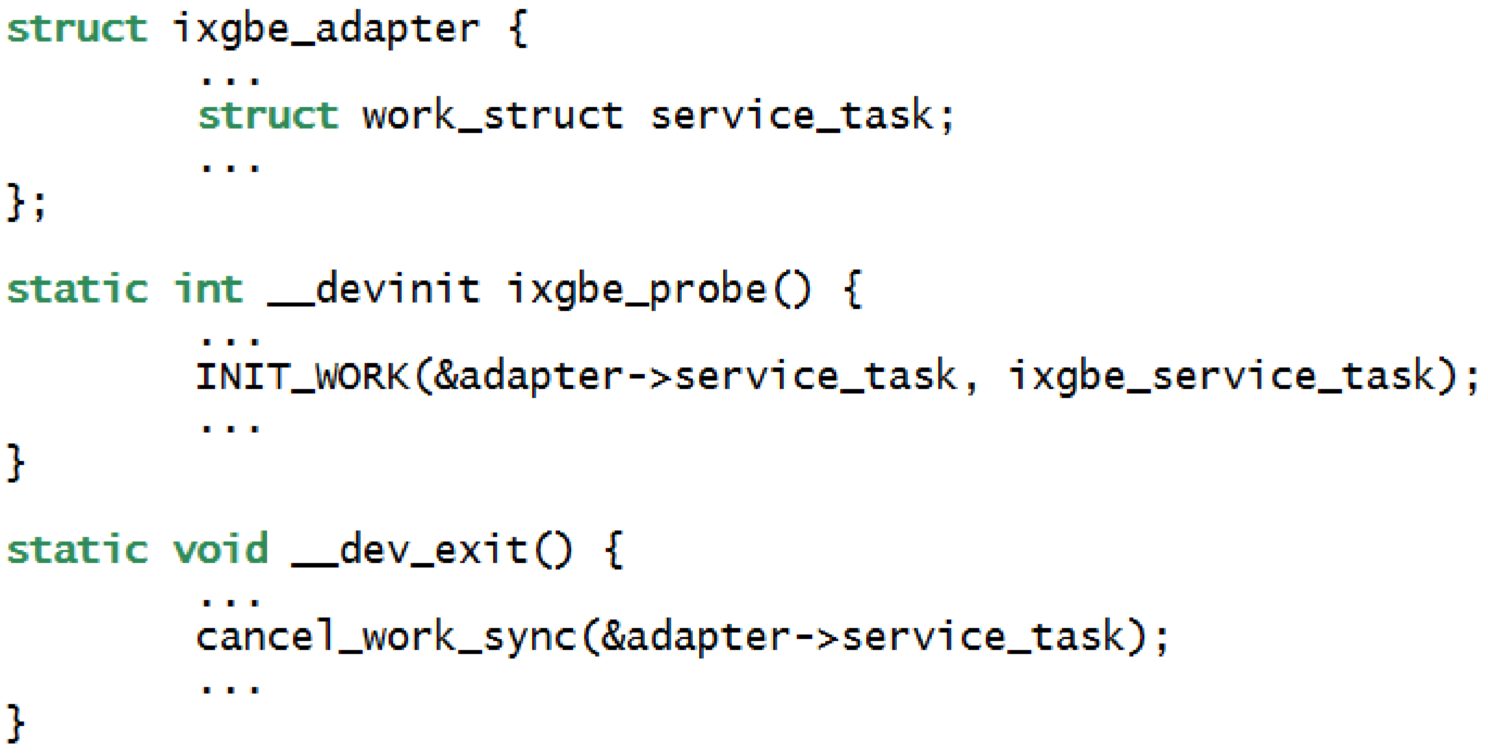
See example code

- e1000e for interrupt handler
  - Why difference between legacy and MSI handlers?
- i40e for something a bit more advanced...
  - i40e_msix_clean_rings()
  - What is NAPI?!?!
  - i40e_napi_poll()

Workqueue

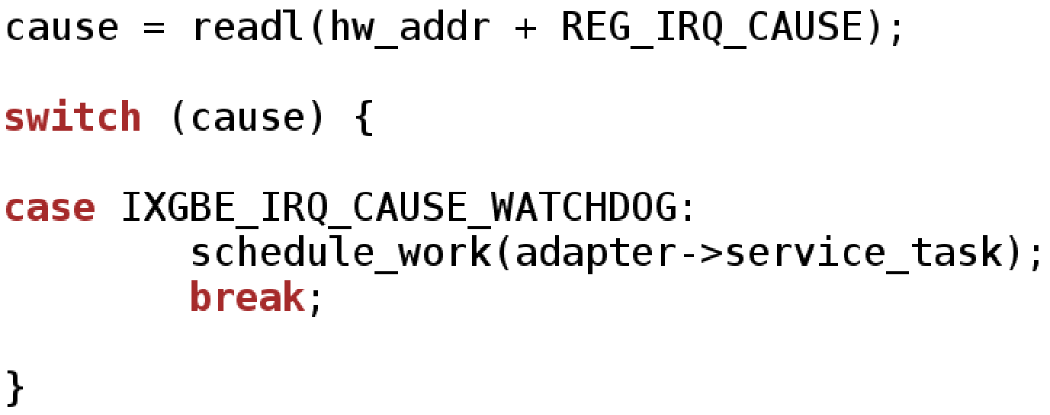
- Uses a task function, like a callback or handler
- Triggered from other threads, e.g. interrupts
- Runs in full process context – can sleep
- Runs on generic kernel work thread or your own thread
- Use it to "do stuff"

Workqueue Setup



```
struct ixgbe_adapter {
        ...
        struct work_struct service_task;
        ...
};

static int __devinit ixgbe_probe() {
        ...
        INIT_WORK(&adapter->service_task, ixgbe_service_task);
        ...
}

static void __dev_exit() {
        ...
        cancel_work_sync(&adapter->service_task);
        ...
}
```

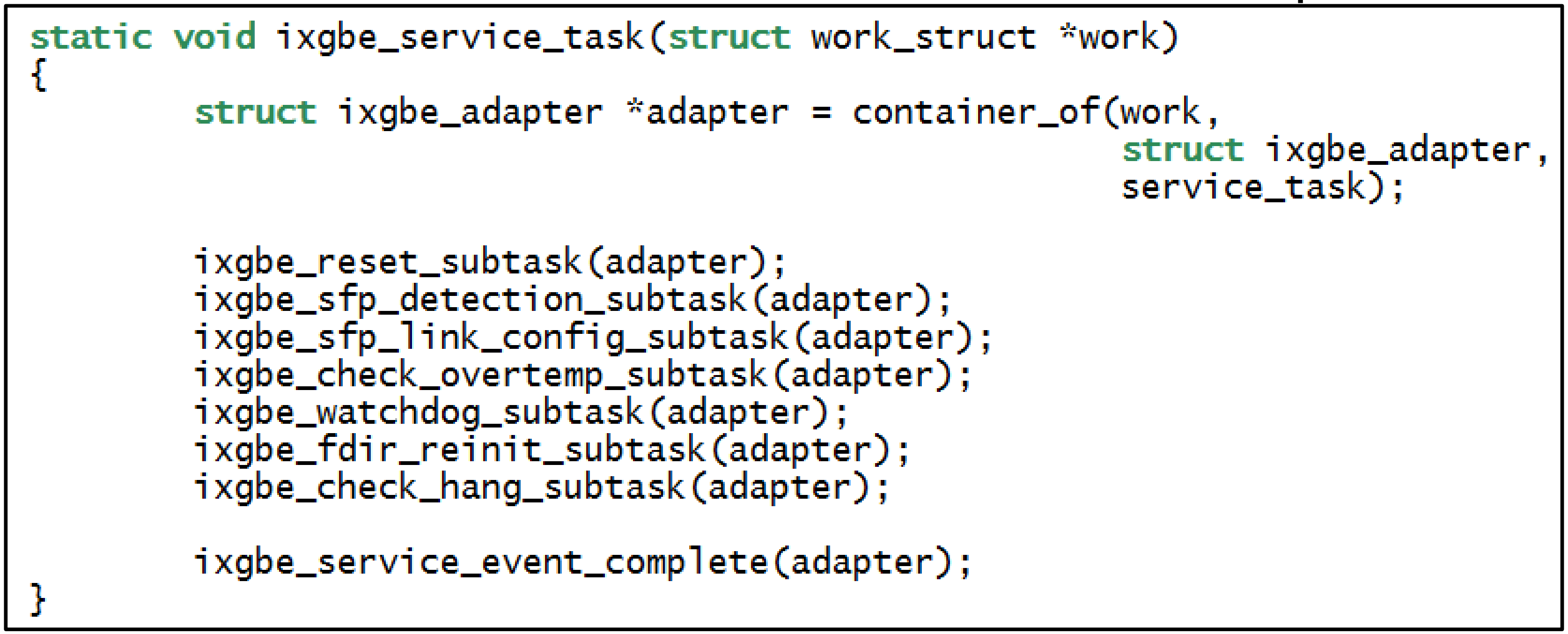
Workqueue Usage



```
cause = readl(hw_addr + REG_IRQ_CAUSE);

switch (cause) {

case IXGBE_IRQ_CAUSE_WATCHDOG:
        schedule_work(adapter->service_task);
        break;

}
```

Interrupt handler

Workqueue code



```
static void ixgbe_service_task(struct work_struct *work)
{
        struct ixgbe_adapter *adapter = container_of(work,
                                                     struct ixgbe_adapter,
                                                     service_task);

        ixgbe_reset_subtask(adapter);
        ixgbe_sfp_detection_subtask(adapter);
        ixgbe_sfp_link_config_subtask(adapter);
        ixgbe_check_overtemp_subtask(adapter);
        ixgbe_watchdog_subtask(adapter);
        ixgbe_fdir_reinit_subtask(adapter);
        ixgbe_check_hang_subtask(adapter);

        ixgbe_service_event_complete(adapter);
}
```

Readings

- LDD3: Chap 10
- ELDD: ppg 72-74, 92-103