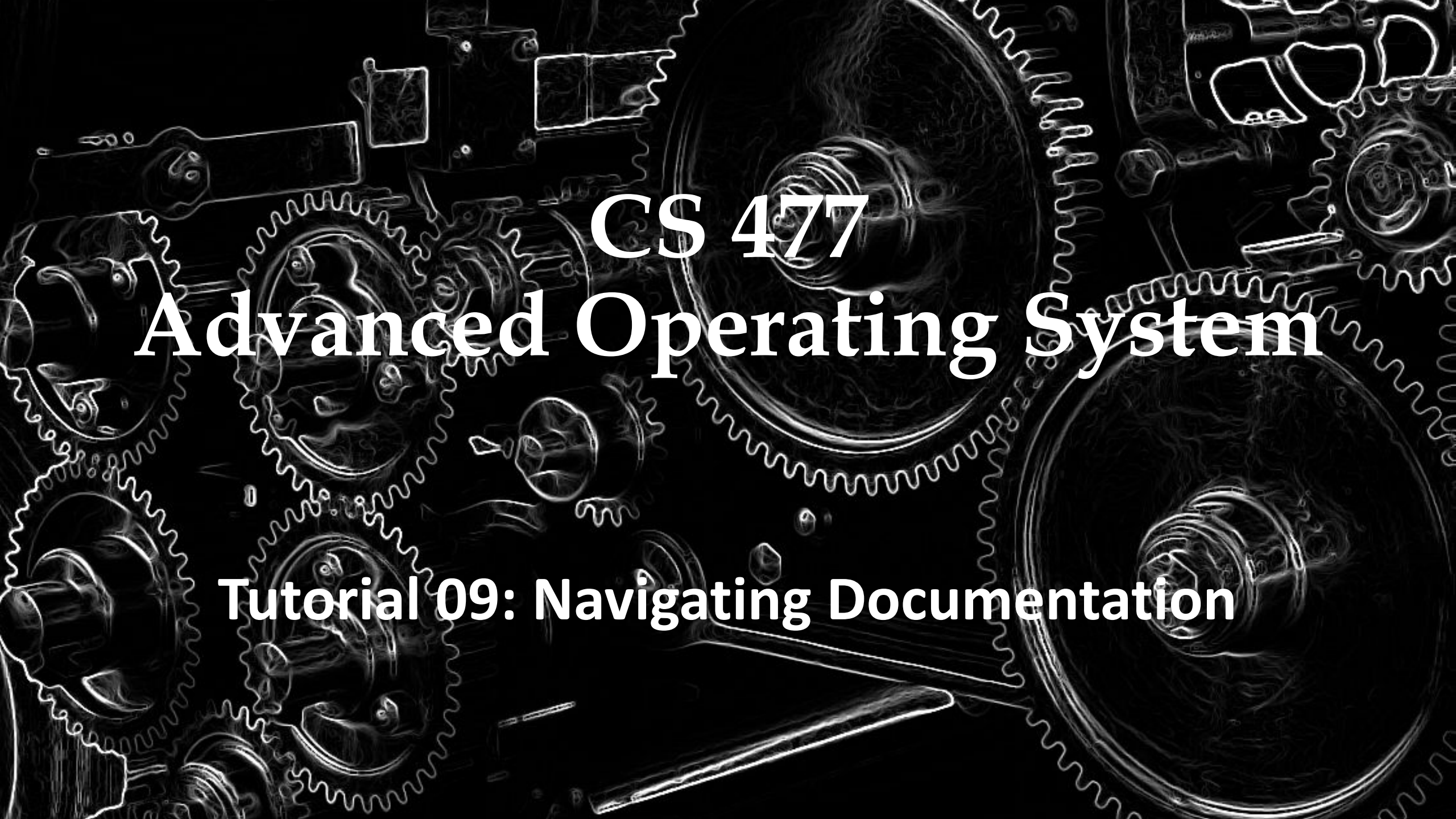

CS 477
Advanced Operating System

Tutorial 09: Navigating Documentation

Today's Tutorial

- Goal: help you systematically navigate kernel documentation

- Relevant source code available here: https://github.com/sidharth-sundar/advos-25-demo/tree/main/Week-9-Demo

Your Task

- **Design a kernel module that registers an interrupt handler on the ACPI interrupt, and**
  - **In the top half, increments a counter every time the interrupt is invoked**
  - **In the bottom half, implemented using work queues, prints the time at which the work function is run**
  - **Prints the counter on module exit**

- We'll get through the first part, and if we have time we'll try the 2nd

Where do we start?

- How do we register an interrupt handler?
- How do we trigger an interrupt?

The Process

1. Find some initial documentation on the involved APIs

2. Check the source code to validate that the APIs are still relevant
   + Find examples of code use in the kernel to inform your implementation

3. Integrate it into your module

4. Test

- (4) is much harder if performing core kernel modifications, but is still necessary. There are ways to run a minimal install on a VM for testing.
- Can perform multiple iterations of (1-3) and defer 4
  - Tradeoff between size of code to debug and frequency of testing

Differences w/ finding documentation vs user space

- No man pages, but we do have kernel documentation:
  - https://github.com/torvalds/linux/blob/master/Documentation/index.rst
  - https://www.kernel.org/doc/html/latest/
- There are some references on stack exchange, but there are also plenty of code samples within the kernel you can check with elixir:

  https://elixir.bootlin.com/linux/v6.15/source/

1. How do we register an interrupt handler?

- 3 common sources to consult to start with:
  - Class Slides
  - Kernel Documentation
  - Textbooks
    - Linux Kernel Programming by Robert Love
    - Linux Kernel Module Programmer Guide: https://fennecj.github.io/lkmpg/
- Not all will be equally useful: find the most relevant one(s) and proceed from there
- Additional sources:
  - LWN articles, patches + commit messages, https://www.kernel.org/doc/html/latest/process/howto.html, https://www.kernel.org/doc/html/latest/process/kernel-docs.html#kernel-docs

1. Initial APIs

- Common trend is 2 APIs:
  - register_irq
  - free_irq
- LKMP has an example
- Linux Kernel Programming textbook has detailed description of APIs
- Class slides provide overview of the register_irq API as well

- So how do we use it?

2. Check the source code, make sure APIs are relevant

- We know we want to use request_irq, but how is it used?
- Let's check how request_irq is defined: https://elixir.bootlin.com/linux/v6.15/A/ident/request_irq
  - Can also try to integrate cscope in your workflow
  - Learn to locate and read kernel-doc comments!
  - Explore locations in the kernel that use the request_irq APIs
  - Check docs for request_threaded_irq as well
- We have some dangling questions
  - What values should we pass to the irq?
  - What flags should we pass?
  - What device name?
  - Does the cookie matter?

2. Check the source code, make sure APIs are relevant

- What values should we use for the *irq* argument?
  - We know there are already valid interrupts registered in /proc/interrupts
  - Let’s check what the ACPI irq number is

```
sidharth@sidharth-OMEN-Gaming-Laptop-16-ae0xxx:~$ cat /proc/interrupts | grep -i acpi
  9:          0          0          0          0          0          0          0          0          0          0        184          0          0          0          0  IR-IO-APIC    9-fasteoi   acpi
```

- What flags should we pass?
  - There’s already an interrupt handling routine (acpi) associated with this irq
  - IRQF_SHARED
- What device name? What does it even mean?
  - Just needs to be specific to the IHR
- Does the cookie matter?
  - Yes, it has to be globally unique

3. Integrate into your module

- https://github.com/sidharth-sundar/advos-25-demo/tree/main/Week-9-Demo
- If you try this exercise yourself, start from the module_skeleton code and integrate request_irq and free_irq. You should end up with something akin to the code in initial_irq_handler
  - Keep note of additional design points that you have to consider (e.g. concurrency) and how to apply (1) and (2) again
- We'll work through this in-person in the tutorial

4. Test

- Similar to user space programs
  - Set up your development environment and testbed
  - Check compilation warnings
  - Write testing code (user or kernel space) to test components of your system

- In our case,
  - Check the interrupt is registered: *cat /proc/interrupts*
  - See if we can trigger the interrupt
    - How do we do this? Let's try looking up ways to trigger the interrupt
      - More details in slide at the very end
  - See if our module detects all instances during its lifespan

Your Task

- Design a kernel module that registers an interrupt handler on the ACPI interrupt, and
  - In the top half, increments a counter every time the interrupt is invoked
  - **In the bottom half, implemented using work queues, prints the time at which the work function is run**
  - Prints the counter on module exit

- Let's try the 2nd half

1. Linked list, time retrieval, and work queues

- Let’s go through our sources:
  - Class Slides
  - Kernel Documentation
  - Textbooks
    - Linux Kernel Programming by Robert Love
    - Linux Kernel Module Programmer Guide: https://fennecj.github.io/lkmpg/
- We’re looking for APIs for work queues and getting times

2. Consult the source code

- Let’s go through the code:
  - https://elixir.bootlin.com/linux/v6.15/source/include/linux/workqueue.h#L250 <- workqueues
  - https://elixir.bootlin.com/linux/v6.15/source/include/linux/timekeeping.h#L73 <- timers

2. Consult the source code

- Let’s go through our sources:
  - https://elixir.bootlin.com/linux/v6.15/source/include/linux/workqueue.h#L250 <- workqueues
  - https://elixir.bootlin.com/linux/v6.15/source/include/linux/timekeeping.h#L73 <- timers

3 + 4

- The final product is available in https://github.com/sidharth-sundar/advos-25-demo/tree/main/Week-9-Demo/workqueue
- Try implementing the bottom half yourself. Use either your own code from the first part of the tutorial, or start from https://github.com/sidharth-sundar/advos-25-demo/tree/main/Week-9-Demo/workqueue

Triggering ACPI

- Look up “how to trigger acpi interrupt”. Some sources come up, e.g.
  - https://f.osdev.org/viewtopic.php?t=32515
  - https://www.kernel.org/doc/ols/2005/ols2005v1-pages-59-76.pdf
- “Trigger SCI linux”
  - https://stackoverflow.com/questions/45206171/how-sci-system-control-interrupt-vector-is-defined

- The ACPI-related interrupts are triggered on power-related events. On my baremetal install, I plug in/out my laptop’s charger. On your VMs, depending on if you’re using parallels, virtualbox, or QEMU, there are ways to emulate a physical power button press, and maybe your power cable interrupt is passed through to the virtual machine as well (I haven’t looked into that part)

- Can validate by checking /proc/interrupts before/after triggering the event

Added considerations for interrupt implementation

- Reason about concurrency as well:
  - Can we trigger the interrupt from multiple cores?
  - How to handle?