Due Reminders

• 6/10 11:59 pm: 100% for Lab 4
• 6/12 11:59 pm: 75% for Lab 4 and 50% for Lab 1-3
• After 6/12 11:59 pm: 0%

• Questions?
  • I will do another round of regrade for lab 1-3 this weekend
Today’s topic

• Quiz 3 Report
• Final course review
Topics Covered

• Week 1: Booting
• Week 2: Address translation
• Week 3: Virtual Memory Management
• Week 4: Quiz on Virtual Memory
• Week 5: User/Kernel Context Switch
• Week 6: System Calls and Page Fault
• Week 7: Quiz on Syscalls, Faults, and Exceptions
• Week 8: Lock and Thread Synchronization
• Week 9: Concurrency Bugs and Deadlock
• Week 10: Quiz 3 & Review
Booting
(Week 1, JOS Lab 1)

- How does x86 Processors boot with BIOS?
  - Which mode does the processor start with?
    - Real mode!
  - Addressing model in early stage of the booting
    - Seg * 16 + offset

- BIOS / Boot sector
  - Where (which address) does BIOS load the boot sector?
    - 0x7c00
  - How does boot sector load the kernel?
    - ELF header

- Processor modes: Real / Protected
  - How does CPU use memory segmentation in those modes?
    - Global descriptor Table (GDT)
Address Translation (Week 2)

- Segmentation
  - \( \text{Seg} \times 16 + \text{offset} \)
  - GDT – base + offset, offset < limit

- Paging
  - Page table / page directory

- Translation Lookaside Buffer (TLB)
  - When to invalidate TLB?
    - When updates CR3 (invalidate all entries)
    - When updates PTE (invalidate 1 entry)
Virtual Memory Management
(Week 3, JOS Lab 2)

- **Page Permission**
  - How can we set access permissions to a memory page?
    - Read/Write, Kernel/User
  - How can we set a conflicting memory permissions, e.g.,
    - Kernel RW, User R

```c
// Your code goes here:
boot_map_region(kern_pgdir, KERNBASE, -KERNBASE, 0, PTE_W | PTE_P);
```
Virtual Memory Management
(Week 3, JOS Lab 2)

• Indexing Page Directory / Tables
  • [10 bit] [10 bit] [12 bit]
  • Why 10 bits?
    • 12 bits for page offset: 4096 bytes
    • 4 byte per each page directory/table entries
    • 1024 entries, indexed by 10 bits
  • [6 bit] [6 bit] [6 bit] [6 bit] [8 bit]
    • Page size: 256 bytes, entries: 64, 4 levels → slow
  • [6 bit] [12 bit] [14 bit]
    • Page size: 16384 bytes, entries: 4096, wasting memory
User/Kernel Switch (Week 5, JOS Lab 3)

• Ring
  • How many rings are available in x86 processor?
    • 4 levels
  • Which ring level do we use for kernel? For user?
    • Ring 0 for kernel, Ring 3 for user
  • Where does CPU store the current ring level?
    • The last 2 bits of the CS register

• User/Kernel Switch
  • Difference between library call and system call
  • How can we switch an execution from
    • User -> kernel?
      • syscalls (software interrupt)
    • Kernel -> User?
      • iret

```c
if ((tf->tf_cs & 3) == 3) {

    void env_pop_tf(struct Trapframe *tf) {
        // Record the CPU we are running on for user-space debugging
        curenv->env_cpunum = cpunum();

        asm volatile(
            "\tmovl %0,%esp\n"
            "\ttpl %es\n"
            "\ttpl %ds\n"
            "\ttpl $0x8,%esp\n" /* skip tf_trapno and tf_errno */
            "\ttiret\n"
            : : "g" (tf) : "memory");

        panic("iret failed"); /* mostly to placate the compiler */
    }
}
```
Interrupt, Syscall, Exception (Week 5, JOS Lab 3)

• Interrupt & Exceptions
  • Interrupt
  • Exceptions and Fault

• Interrupt Descriptor Table (IDT) and Interrupt handlers
  • How can we set interrupt handlers?
  • How can we determine which interrupt the current one is?
    • E.g., how can we get the interrupt number?
    • Pushed by CPU? Pushed by JOS?

#define TRAPHANDLER(name, num)  
.globl name; /* define global symbol for 'name' */
.type name, @function; /* symbol type is function */
.align 2; /* align function definition */
\name: /* function starts here */  
pushl $(num);
jmp _alltraps
Interrupt, Syscall, Exceptions
(Week 6, JOS Lab 3)

• Understanding the Trapframe
  • What kind of fault it is? \( \text{pf} \)
  • What is the faulting address? \( \text{cr2} \)
  • What is the reason for the fault? \( \text{err} \)
  • What is the address of instruction that causes the fault? \( \text{eip} \)
  • Which values were generated by CPU?
  • Which values were generated by JOS?
  • Which ring level it is? \( \geq \)
Page Fault &
Copy-on-Write
(Week 6, JOS Lab3&4)

• Page fault workflow
  • When does it happen?
  • How can we know the faulting address and the cause of the fault?
  • How can we resolve the fault and get back to the normal execution?

• Page fault use cases (refer to the slide of Lecture 10)
  • Automatic stack allocation
  • Copy-on-write
  • Memory Swapping
Synchronization and Locks (Week 8)

• Data racing
  • What is this?
  • Why is this bad?
    • Inconsistent/incorrect result
  • How can we resolve this?
    • Mutual exclusion

• Lock
  • How can we implement locks?
  • What’s the difference between
    • Test-and-set (atomic)
    • Test and test-and-set (atomic)
    • *Backoff
Concurrency Bugs and Deadlock (Week 9)

- **TOCTTOU (Time of check to time of use) bug**
  - What is this and when does it happen?
    - Another thread executes between time of check and time of use
  - How can we prevent this?
    - Use lock/unlock

- **Deadlock**
  - Four necessary conditions of deadlock
    - Mutual Exclusion
      - Critical section
    - Hold-and-wait
    - No Preemption
    - Unlock if fail to acquire a lock
    - Circular wait

---

### Code Example
```c
Read
1    Thread 1::
2    if (thd->proc_info) { Time of check
3        ...
4        fputs(thd->proc_info, ...); Time of use
5        ...
6    }
7
8    Thread 2::
9    thd->proc_info = NULL; Write!
```

---

### Diagram
- **TOCTTOU**
- **Deadlock**
  - Four necessary conditions: Mutual Exclusion, Hold-and-wait, No Preemption, Circular wait

---

**Oregon State University**
You Have Learned Many Things from CS 444/544

- How to build OS internals in a nutshell
- Bootloader (JOS Lab 1)
- Setting up virtual memory (JOS Lab 2)
- Setting up interrupt handlers (JOS Lab 3a)
- Implementing system calls (JOS Lab 3b)
- Implementing locks (lock-example repository)
- Implementing page fault handler and copy-on-write fork (JOS Lab 4)
Be Confident in Computer Systems

• Now you have experience in
  • Terminal IDE tools (tmux, git, vim, ctags, make)
  • Kernel-level (Ring 0) Debugging (via remote gdb)
    • x86 Assembly
    • Paging and address translation
    • Software/hardware interrupt and exception handling
    • Enabling preemptive multitasking

• And you wrote code for multi-core OS (pedagogical)
Final Remarks

• Thank you so much for your commitment to this course
• Submit all your work by 6/10 11:59pm
  • 100% for lab4
  • 75% for lab4 and 50% for lab1-3 if submitted by 6/12 11:59 pm

• Future improvements?