W4118 Operating Systems

Junfeng Yang

Outline

- Interrupt
- Protection
- System call

Interrupt overview of how OS works

- First, needs to boot OS
  - CPU jumps to bootstrapping code at fixed mem location
  - Bootstrapping code loads OS from fixed disk location
  - OS initializes services
  - OS creates first user process ➔ more user processes
- Then, OS waits for events: "event driven"
  - No event ➔ OS not involved
  - Event shows up ➔ OS handles
    - Interrupts from hardware
    - System calls or exceptions caused by applications (cause interrupt)

Dual mode of operation

- Interrupt handling involves privileged operations
- Hardware protection mechanism
  - Kernel mode: can do all operations, including privileged
  - User mode: can do only non-privileged operations
- Dual mode of operation
  - Interrupt causes CPU to transit from user mode to kernel mode
  - Return from interrupt handling causes CPU to transit from kernel mode to user mode

Interrupt dispatch overview

- CPU checks for interrupts

```
while (fetch next instruction) {
        run instruction;
        if (interrupt) {
                save EIP                                                  // user mode
                find and jump to OS-provided interrupt handler // kernel mode
                restore EIP                                               // user mode
         }
}
```

- Questions
  - How does h/w find interrupt handler?
  - What does interrupt handler do?

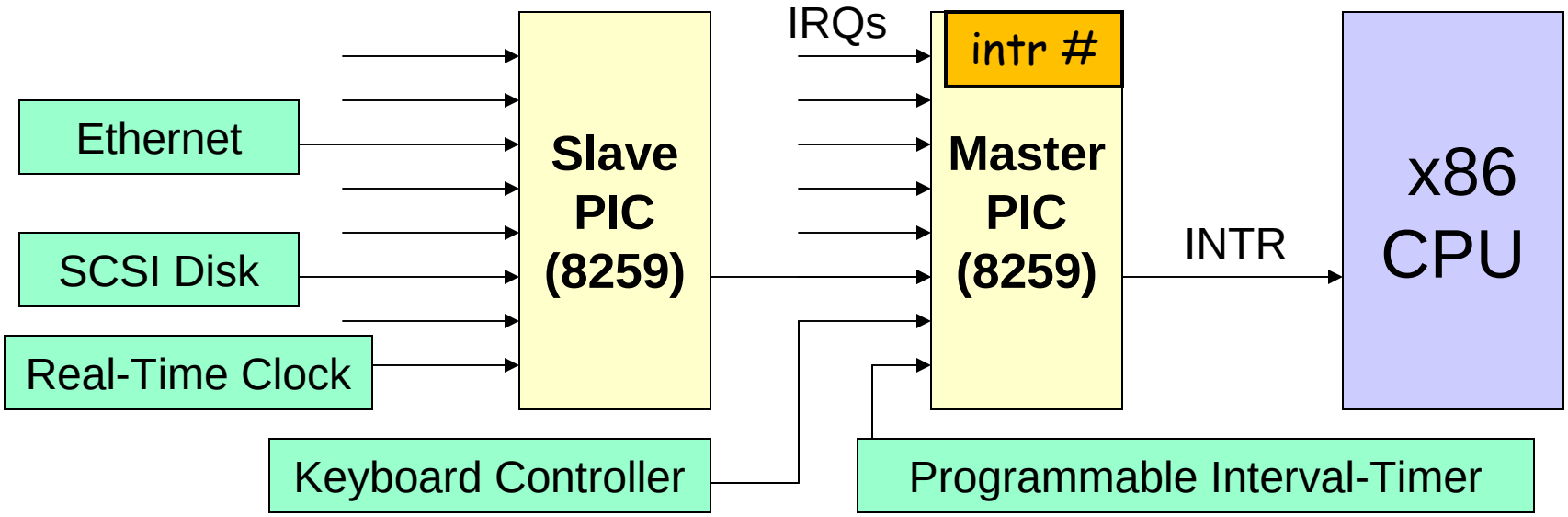
How dos h/w find interrupt handler?

- Each type of interrupt is assigned an interrupt number
- OS sets up Interrupt Descriptor Table (IDT) at boot time
  - Lives in memory, h/w knows its base
  - Each entry is an interrupt handler, also called interrupt routine or Interrupt Service Routine (ISR)
  - Defines all kernel entry points
- H/w finds interrupt handler using interrupt number as index into IDT
  - handler = IDT[intr_number]

What does interrupt handler do?

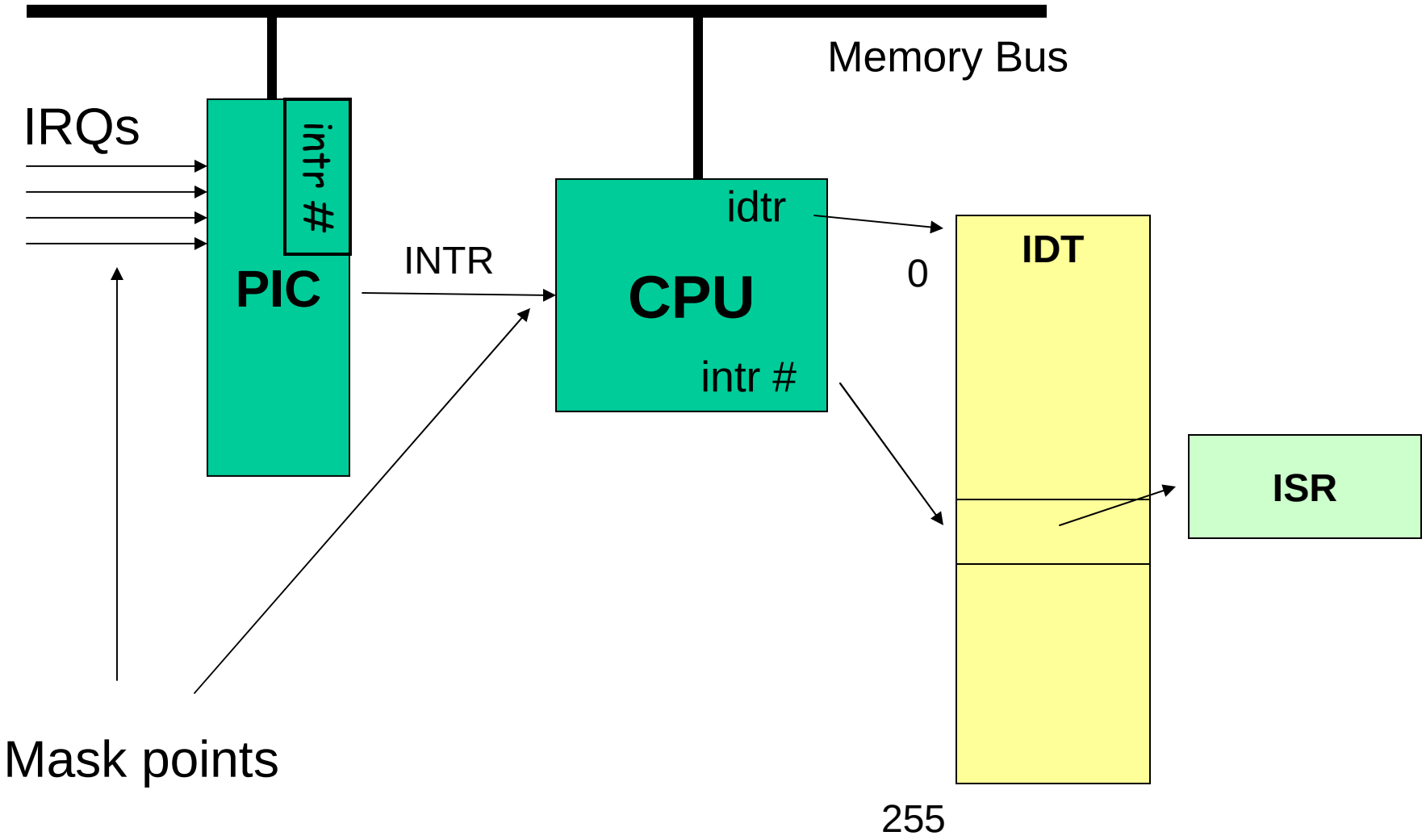
- Device-specific stuff
- Save/restore process state (registers)
- May disable interrupts to avoid further interruption
  - cli() – disable interrupts
  - sti() – enable interrupts
  - can mask specific interrupts as well
- Want to limit interrupt processing overhead: schedule for later
- Before return from interrupt, may need to do OS book keeping
  - reschedule, signals, etc.

X86 interrupt hardware (legacy)



- I/O devices raise Interrupt Request lines (IRQ)
- Programmable Interrupt controller (PIC) maps IRQ to Interrupt Numbers
- PIC raises INTR line to interrupt CPU
- Nest PIC for more devices

X86 interrupt dispatch



Interrupt v.s. Polling

- Instead for device to interrupt CPU, CPU can poll the status of device
  - Intr: "I want to see a movie."
  - Poll: for(each week) {"Do you want to see a movie?"}
- Good or bad?
  - For mostly-idle device?
  - For busy device?

Outline

- Interrupt
- Protection
- System call

Need for protection

- Kernel is privileged
- User applications are untrusted
  - Security: malicious programs read/write data
  - Reliability: buggy programs crash machine
- Must protect kernel from user applications

Hardware mechanisms for protection

- Dual model of operation
  - All operations in kernel mode
  - Non-privileged operations in user mode
  - Well defined interface to transit between modes
- Memory protection
  - E.g, base and limit registers
  - Kernel sets base and limit before creating process
- Timer interrupt
  - Kernel periodically gets back control

Example privileged operations

- I/O
- Write protected memory region
  - E.g., interrupt descriptor table
- Set base/limit registers
- Load timer interrupt handler

Outline

- Interrupt
- Protection
- System call

System call

- Applications cannot perform privileged operations themselves
- Must request OS to do so on their behalf by issuing system calls
- OS must validate system call parameters

Typical system call implementation

- Implemented as a software interrupt
- Puts arguments in certain places, e.g., regisers
  - Key argument: system call number
- Executes interrupt instruction, e.g., int 0x80
- Interrupt generated, switching to kernel mode
- Invokes interrupt handler for 0x80
- Looks up system call table to find right routine
- Jumps to appropriate system call routine
- Returns to user mode

Next lecture

- Interrupts, system calls, and protection in Linux