OPERATING SYSTEMS

I/O Management, System Protection and Security

Kakoli Bora
Department of Computer Science

OPERATING SYSTEMS

I/O Hardware, polling and interrupts

Kakoli Bora
Department of Computer Science

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

Overview

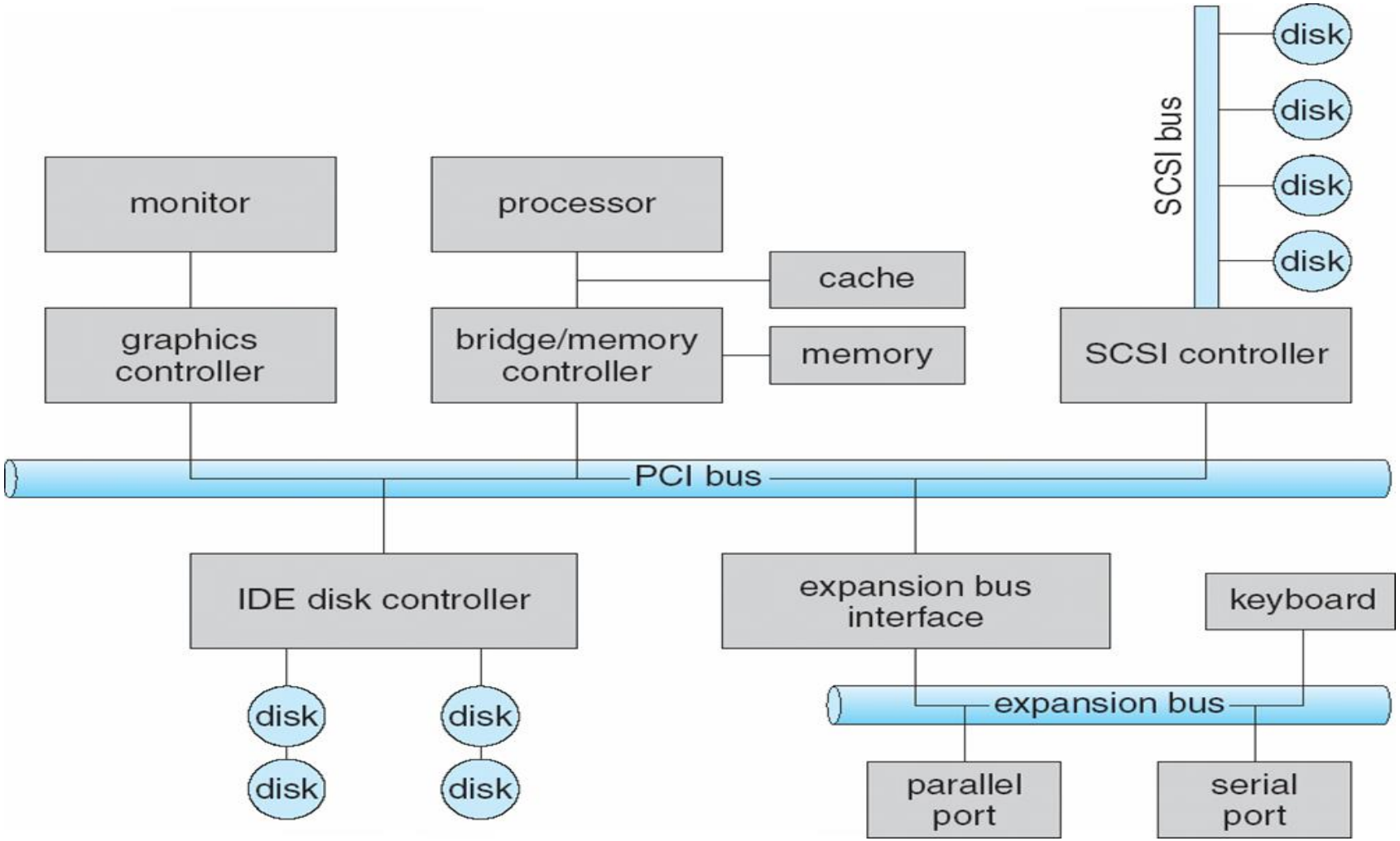
- I/O management is a major component of operating system design and operation
  - Important aspect of computer operation
  - I/O devices vary greatly
  - Various methods to control them
  - Performance management
  - New types of devices frequent
- Ports, buses, device controllers connect to various devices
- **Device drivers** encapsulate device details
  - Present uniform device-access interface to I/O subsystem

I/O Hardware

- Incredible variety of I/O devices
  - Storage (Disks, tapes)
  - Transmission (network connections, Bluetooth)
  - Human-interface (screen, keyboard, mouse, audio in and out)
- Common concepts – signals from I/O devices interface with computer
  - **Port** – connection point for device
  - **Bus** - **daisy chain** or shared direct access
    - **PCI** bus common in PCs and servers, PCI Express (**PCIe**)
    - **expansion bus** connects relatively slow devices
  - **Controller** (**host adapter**) – electronics that operate port, bus, device
    - Sometimes integrated
    - Sometimes separate circuit board (host adapter)
    - Contains processor, microcode, private memory, bus controller, etc
      - Some talk to per-device controller with bus controller, microcode, memory, etc

OPERATING SYSTEMS

A Typical PC Bus Structure



I/O Hardware

- I/O instructions control devices
- Devices usually have registers where device driver places commands, addresses, and data to write, or read data from registers after command execution
  - Data-in register, data-out register, status register, control register
  - Typically 1-4 bytes, or FIFO buffer

I/O Hardware (Cont.)

- Devices have addresses, used by
  - **Direct I/O instructions**
    - Special I/O instructions specify the transfer of a byte or word to an I/O port address.
    - The I/O instruction triggers bus lines to select the proper device and to move bits into or out of a device register.
  - **Memory-mapped I/O**
    - Device data and command registers mapped to processor address space
    - Especially for large address spaces (graphics)
- Some systems use both techniques – I/O instructions to control some devices and memory-mapped I/O to control others

Device I/O Port Locations on PCs (partial)

I/O address range (hexadecimal)	device
000–00F	DMA controller
020–021	interrupt controller
040–043	timer
200–20F	game controller
2F8–2FF	serial port (secondary)
320–32F	hard-disk controller
378–37F	parallel port
3D0–3DF	graphics controller
3F0–3F7	diskette-drive controller
3F8–3FF	serial port (primary)

Polling

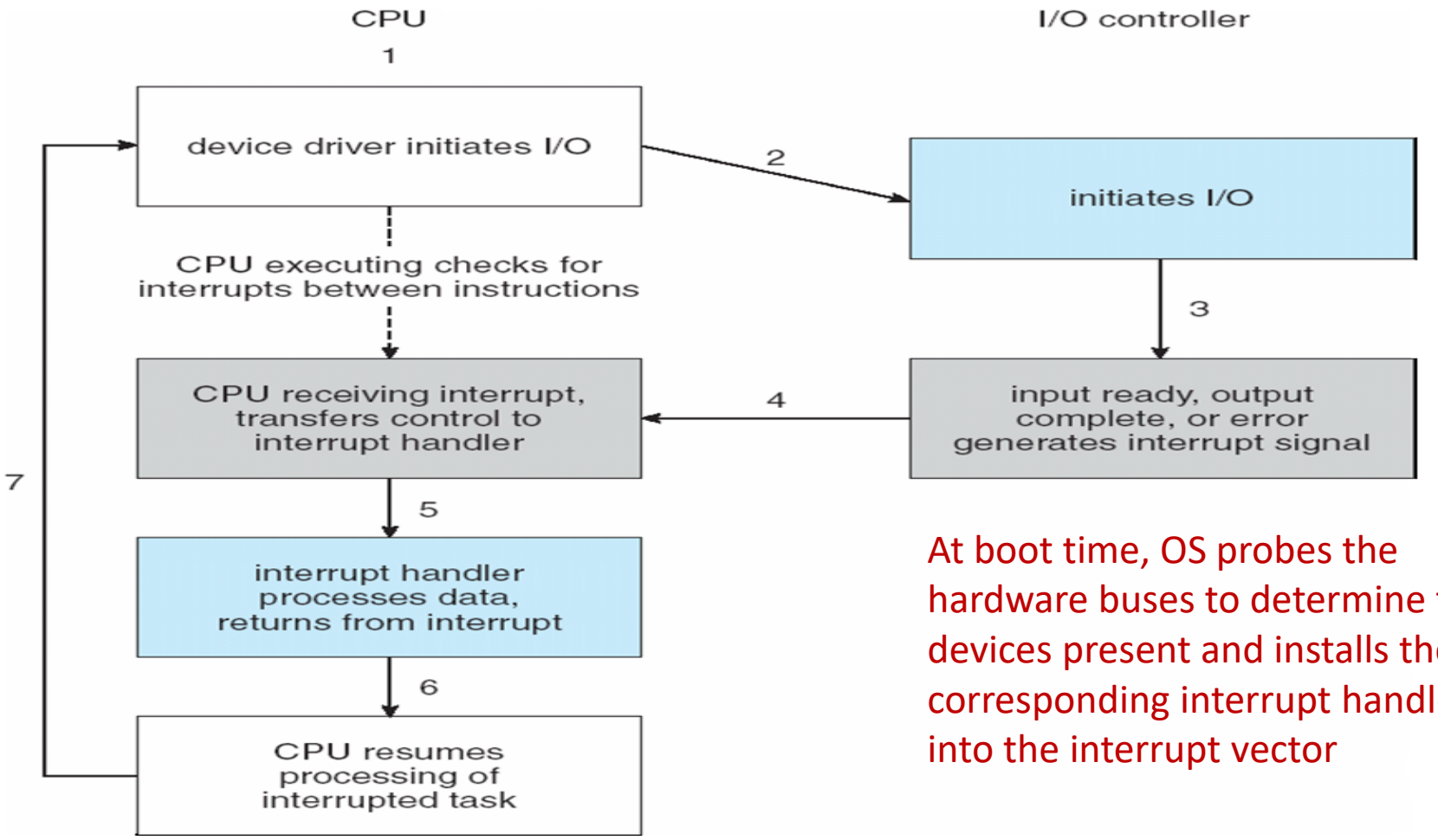
- For each byte of I/O
  1. Host repeatedly reads busy bit from status register until that bit becomes clear (**here host is busy-waiting or polling**)
  2. Host sets read or write bit and if write, copies data into data-out register
  3. Host sets command-ready bit
  4. Controller notices command-ready bt is set, sets busy bit, executes transfer
  5. Controller clears busy bit, error bit, command-ready bit when transfer done
- Step 1 above is **busy-wait** cycle to wait for I/O from device
  - Reasonable if device is fast
  - But inefficient if device slow
  - CPU switches to other tasks?
    - But if miss a cycle data overwritten / lost

Interrupts

- Polling can happen in 3 instruction cycles
  - Read a device register, logical--and to extract status bit, branch if not zero
  - How to be more efficient if non-zero infrequently?
- CPU **Interrupt-request line** triggered by I/O device
  - Checked by processor after each instruction
- **Interrupt handler** receives interrupts
  - **Maskable** to ignore or delay some interrupts
- **Interrupt vector** to dispatch interrupt to correct handler
  - Context switch at start and end
  - Based on priority
  - Some **nonmaskable**
  - Interrupt chaining (list of interrupt handlers) if more than one device at same interrupt number
- **Above features are provided by the CPU and by the Interrupt-controller hardware**
  - **Most CPUs have 2 interrupt request lines – maskable and nonmaskable**

OPERATING SYSTEMS

Interrupt-Driven I/O Cycle



At boot time, OS probes the hardware buses to determine the devices present and installs the corresponding interrupt handlers into the interrupt vector

OPERATING SYSTEMS

Intel Pentium Processor Event-Vector Table

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

Events from 0 to 31 are nonmaskable and are used to signal various error conditions

Interrupts (Cont.)

- Interrupt mechanism also used for **exceptions**
  - Terminate process, crash system due to hardware error
- Page fault executes when memory access error
- System call executes via **trap** to trigger kernel to execute request
  - Software interrupt or trap is given a lower priority compared to device interrupts
- Multi-CPU systems can process interrupts concurrently
  - If operating system designed to handle it
- Used for time-sensitive processing, frequent, must be fast
  - Note: Interrupt handlers in Solaris are executed as kernel threads

THANK YOU

Kakoli Bora
Department of Computer Science Engineering

k_bora@pes.edu