Operating Systems

Lecture 9

I/O System

Summary

- This lecture describes the Input/Output subsystem:
  - Introduction
  - Pooling
  - Interrupts
  - Direct Memory Access (DMA)
  - Programming interface
  - Kernel I/O subsystem
  - Efficiency

Summary

- This lecture describes the Input/Output subsystem:
  - Introduction
  - Interrupts
  - Direct Memory Access (DMA)
  - Programming interface
  - Kernel I/O subsystem
  - Efficiency

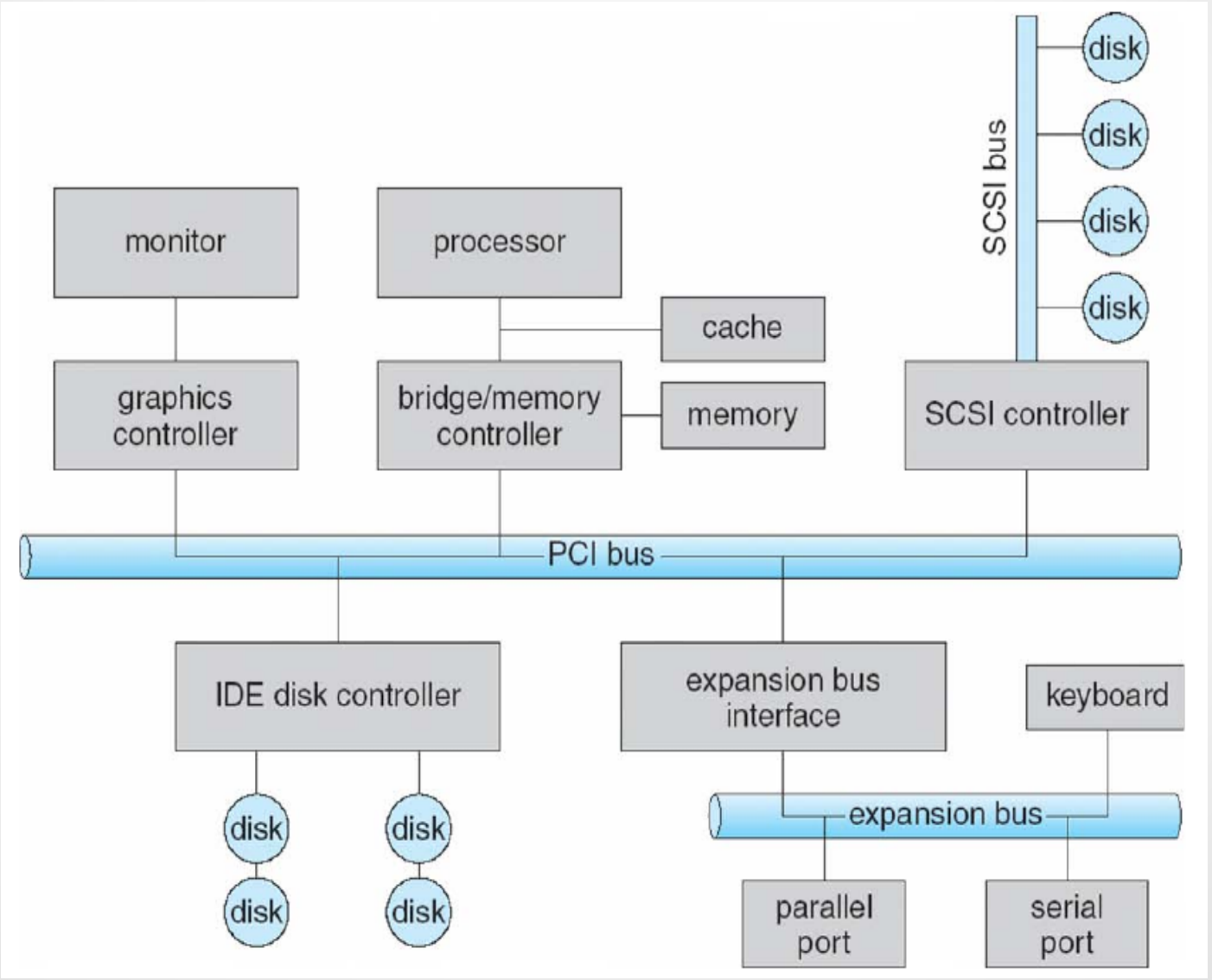
Introduction

- I/O management is a major component of operating system design and operation
  - Important aspect of computer operation
  - I/O devices vary greatly
    - Storage
    - Transmission
    - Human-interface
  - Various methods to control them
  - Performance management
  - New types of devices appear frequently

Introduction

- Device drivers encapsulate device details
  - Present uniform device-access interface to I/O subsystem
- 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.

A typical PC bus structure



Source: Operating System Concepts – 9th Edition
Silberschatz, Galvin and Gagne ©2013

Summary

- This lecture describes the Input/Output subsystem:
  - Introduction
  - Pooling
  - Interrupts
  - Direct Memory Access (DMA)
  - Programming interface
  - Kernel I/O subsystem
  - Efficiency

Polling

- Polling consists in active querying of the device, if the requested operation has finished.
- Works in a loop with – potentially – a lot of idle cycles.

Polling

- For each byte of I/O
  1. Read busy bit from status register until 0
  2. Host sets read or write bit and in casse of write copies data into data-out register
  3. Host sets command-ready bit
  4. Controller sets busy bit, executes transfer
  5. Controller clears busy bit, error bit and command-ready bit when transfer is done
- Step 1 uses a busy-wait cycle to wait for I/O from device
  - Reasonable if device is fast
  - But inefficient if device is slow
  - Can CPU switch to other tasks?
    - But if cycle is missed, the data may be overwritten / lost

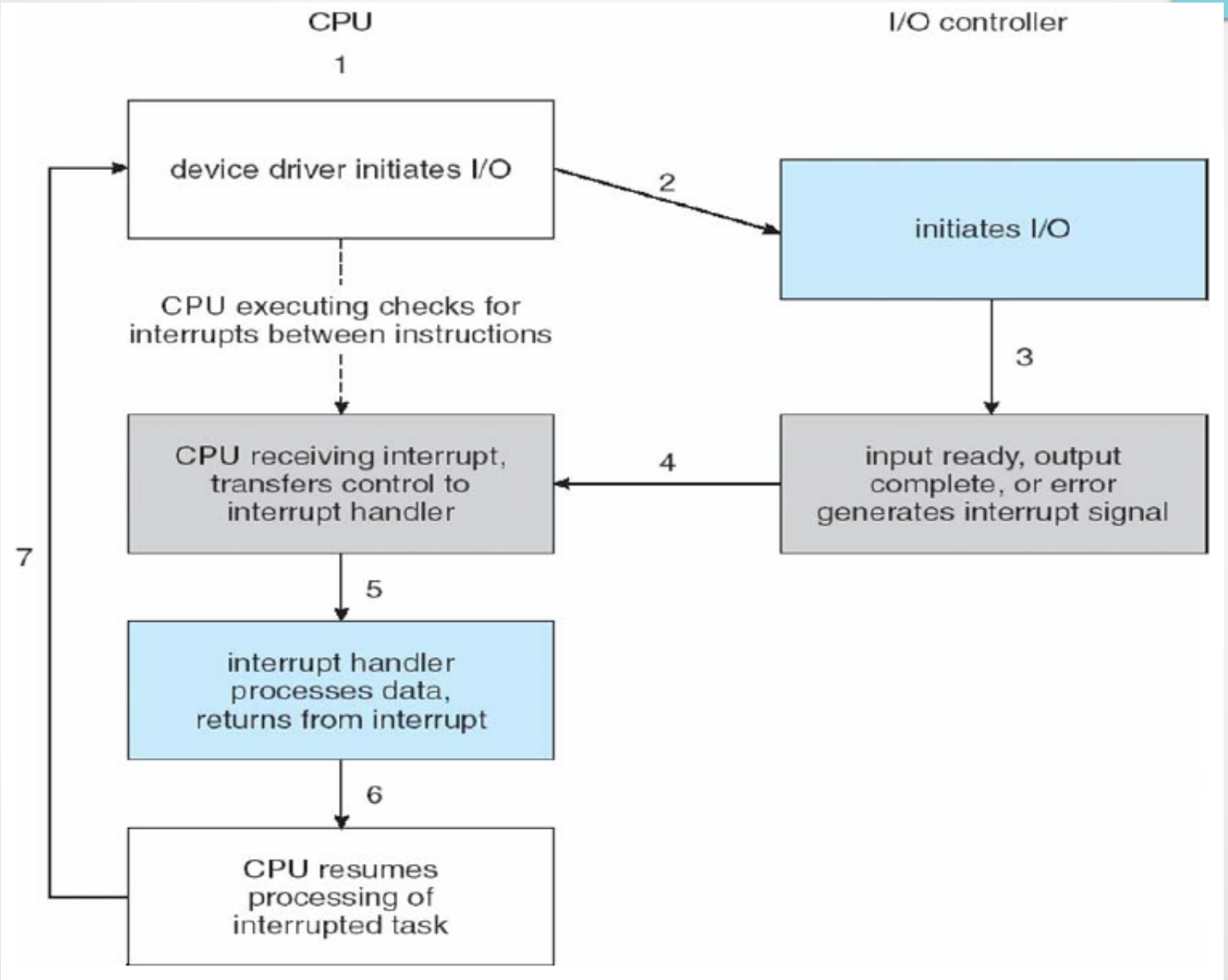
Summary

- This lecture describes the Input/Output subsystem:
  - Introduction
  - Pooling
  - Interrupts
  - Direct Memory Access (DMA)
  - Programming interface
  - Kernel I/O subsystem
  - Efficiency

Interrupts

- Polling can happen in 3 instruction cycles:
  - Read status, 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 if more than one device at same interrupt number

Interrupt-driven I/O cycle



Source: Operating System Concepts – 9th Edition
Silberschatz, Galvin and Gagne ©2013

Interrupts

- 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
- Multi-CPU systems can process interrupts concurrently
  - If operating system designed to handle it
- Used for time-sensitive processing, frequent, must be fast

Summary

- This lecture describes the Input/Output subsystem:
  - Introduction
  - Pooling
  - Interrupts
  - Direct Memory Access (DMA)
  - Programming interface
  - Kernel I/O subsystem
  - Efficiency

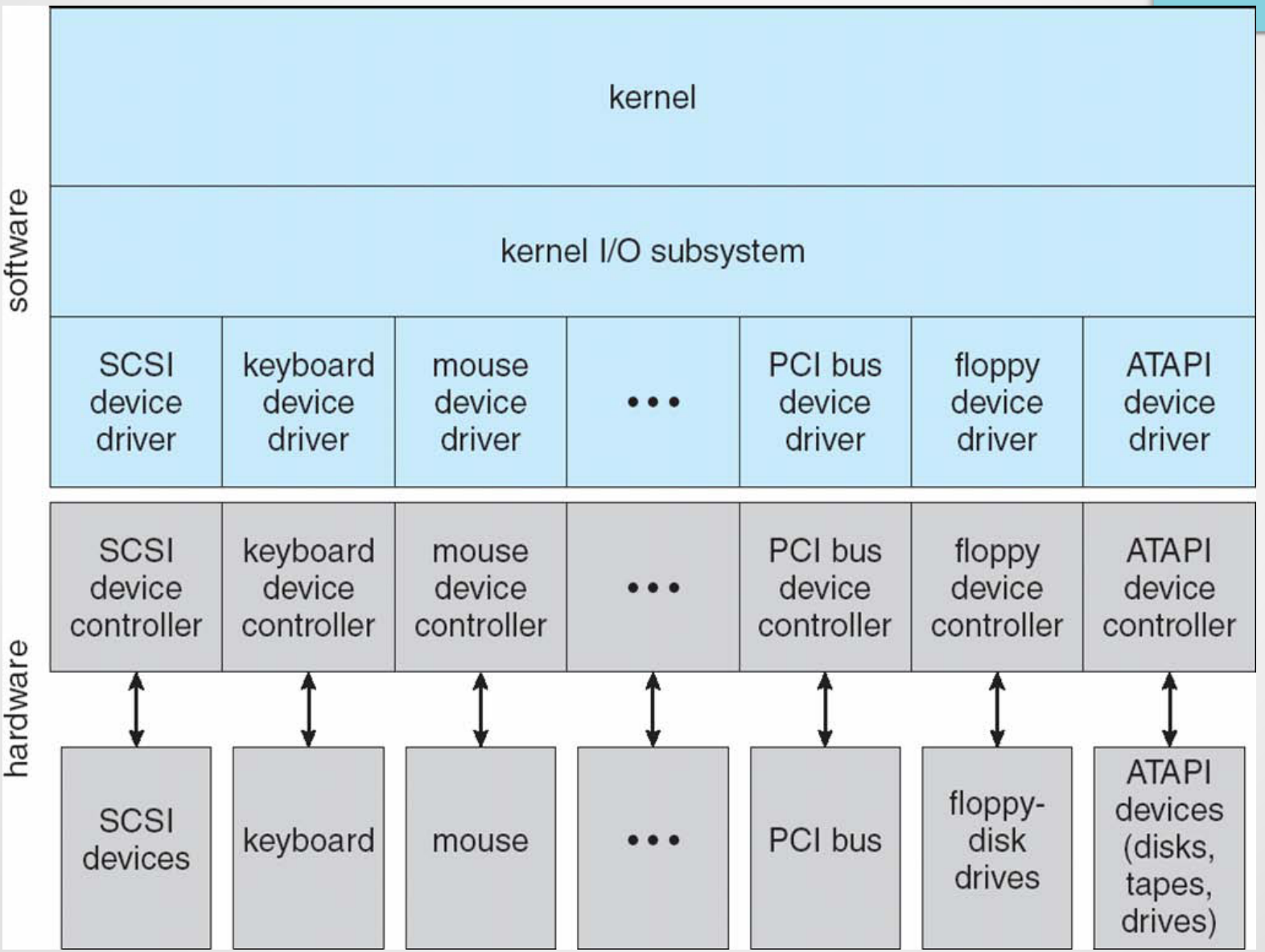
Direct Memory Access (DMA)

- Used to avoid programmed I/O (one byte at a time) for large data movement
- Requires DMA controller
- Bypasses CPU to transfer data directly between I/O device and memory
- OS writes DMA command block into memory
  - Source and destination addresses
  - Read or write mode
  - Count of bytes
  - Writes location of command block to DMA controller
  - Bus mastering of DMA controller – grabs bus from CPU
    - Cycle stealing from CPU but still much more efficient
  - When done, interrupts to signal completion
- Version that is aware of virtual addresses can be even more efficient - DVMA

Summary

- This lecture describes the Input/Output subsystem:
  - Introduction
  - Pooling
  - Interrupts
  - Direct Memory Access (DMA)
  - Programming interface
  - Kernel I/O subsystem
  - Efficiency

Programming interface



Source: Operating System Concepts – 9th Edition
Silberschatz, Galvin and Gagne ©2013

I/O device characteristics

- Due to data transfer mode:
  - Character devices (keyboard),
  - Block devices (disk).
- Due to access method:
  - Random access (disk),
  - Sequential access (modem, magnetic tape).
- Due to transfer schedule:
  - synchronous (tape),
  - asynchronous (keyboard).
- Due to access policy:
  - Shared (keyboard),
  - Dedicated (printer).

I/O device characteristics (cont.)

- Due to speed (latency, seek time, transfer rate, delay between operations).
- Due to I/O direction:
  - Read-only (CD-ROM disk),
  - Write-only (graphics card),
  - Read-write (HD disk).
- Subtleties of devices handled by device drivers
- Broadly I/O devices can be grouped by the OS into
  - Block I/O
  - Character I/O (Stream)
  - Memory-mapped file access
  - Network sockets
- For direct manipulation of I/O device specific characteristics, usually an escape / back door
  - Unix `ioctl()` call to send arbitrary bits to a device control register and data to device data register

Block and character devices

- Block devices include disk drives
  - Commands include read, write, seek
  - Raw I/O, direct I/O, or file-system access
  - Memory-mapped file access possible
    - File mapped to virtual memory and clusters brought via demand paging
  - DMA
- Character devices include keyboards, mice, serial ports
  - Commands include `get()`, `put()`
  - Libraries layered on top allow line editing

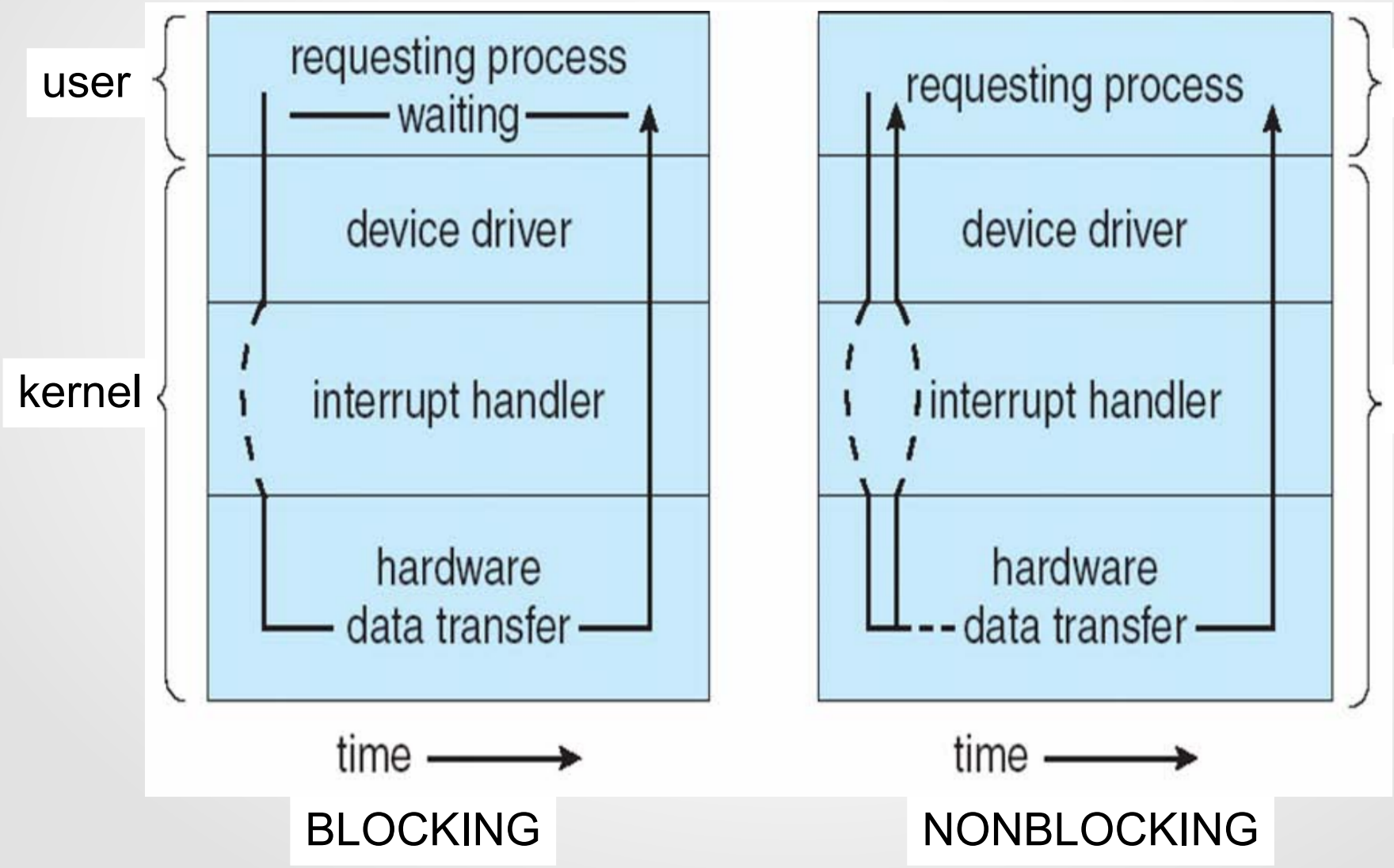
Network devices, clocks and timers

- Network devices.
  - Special due to their efficiency and addressing.
  - Usually have a different interface to „read-write-search” like block devices.
- Clocks and timers.
  - They are not designed to send/receive data, but are used for synchronization.
  - Interrupt as the only expected clock activity.

Blocking and non-blocking I/O

- Blocking I/O – a process waits until its input/output request is finished.
- Non-blocking I/O – a process is not suspended, but gets as many results as possible at a time. I/O function can generate an error, when it cannot be immediately completed.
  - Usually includes buffering.
  - For instance, non-blocking read from keyboard can consist in getting data from the keyboard buffer.
- Asynchronous I/O – a process is not suspended and can continue its operations, but also gets no results immediately. When the operation is finished, the process is notified about this fact with a kind of signal.

Two I/O methods



Summary

- This lecture describes the Input/Output subsystem:
  - Introduction
  - Pooling
  - Interrupts
  - Direct Memory Access (DMA)
  - Programming interface
  - Kernel I/O subsystem
  - Efficiency

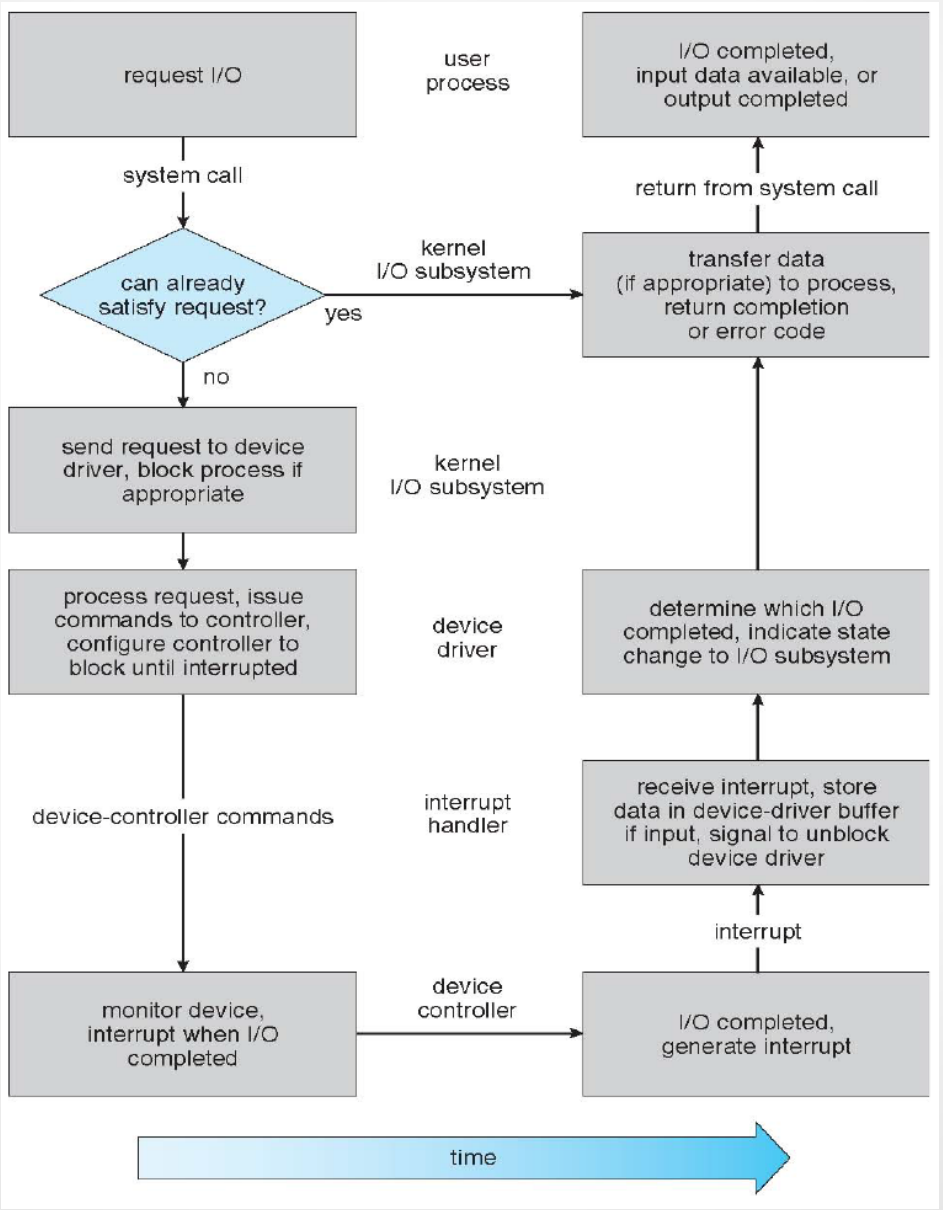
Kernel I/O subsystem

- Scheduling
  - Some I/O request ordering via per-device queue
  - Some OSs try fairness
  - Some implement Quality Of Service (i.e. IPQOS)
- Buffering - store data in memory while transferring between devices
  - To cope with device speed mismatch
  - To cope with device transfer size mismatch
  - To maintain “copy semantics”
  - Double buffering – two copies of the data
    - Kernel and user
    - Varying sizes
    - Full / being processed and not-full / being used
    - Copy-on-write can be used for efficiency in some cases

Kernel I/O subsystem (cont.)

- Caching - faster device holding copy of data
  - Always just a copy
  - Key to performance
  - Sometimes combined with buffering
- Spooling - hold output for a device
  - If device can serve only one request at a time
  - i.e., Printing
- Device reservation - provides exclusive access to a device
  - System calls for allocation and de-allocation
  - Watch out for deadlock

Life cycle of an I/O request



Source: Operating System Concepts – 9th Edition
Silberschatz, Galvin and Gagne ©2013

Streams

- STREAM – a full-duplex communication channel between a user-level process and a device in Unix System V and beyond
- A STREAM consists of:
  - STREAM head interfaces with the user process
  - driver end interfaces with the device
  - zero or more STREAM modules between them
- Each module contains a read queue and a write queue
- Message passing is used to communicate between queues
  - Flow control option to indicate available or busy
- Asynchronous internally, synchronous where user process communicates with stream head

Summary

- This lecture describes the Input/Output subsystem:
  - Introduction
  - Pooling
  - Interrupts
  - Direct Memory Access (DMA)
  - Programming interface
  - Kernel I/O subsystem
  - Efficiency

Performance

- I/O a major factor in system performance:
  - Demands CPU to execute device driver, kernel I/O code
  - Context switches due to interrupts
  - Data copying
  - Network traffic especially stressful

Improving I/O performance

- Reduce number of context switches
- Reduce data copying
- Reduce interrupts by using large transfers, smart controllers, polling
- Use DMA
- Use smarter hardware devices
- Balance CPU, memory, bus, and I/O performance for highest throughput
- Move user-mode processes / daemons to kernel threads

Power management

- Not strictly domain of I/O, but much is I/O related
- OSes can help manage and improve use
  - Cloud computing environments move virtual machines between servers
    - Can end up evacuating whole systems and shutting them down
- Mobile computing has power management as first class OS aspect. For example, Android implements:
  - Component-level power management
    - Understands relationship between components
    - Build device tree representing physical device topology
    - System bus -> I/O subsystem -> {flash, USB storage}
    - Device driver tracks state of device, whether in use
    - Unused component – turn it off
    - All devices in tree branch unused – turn off branch
  - Wake locks – like other locks but prevent sleep of device when lock is held
  - Power collapse – put a device into very deep sleep
    - Marginal power use
    - Only awake enough to respond to external stimuli (button press, incoming call)

Thank You