Operating Systems
[12. I/O Systems]

Chung-Wei Lin
cwlin@csie.ntu.edu.tw
CSIE Department
National Taiwan University

From Operating System Concepts (10th Edition)

Objectives

- Explore the structure of an operating system's I/O subsystem
- Discuss the principles and complexities of I/O hardware
- Explain the performance aspects of I/O hardware and software

Outline

- **Overview**
- I/O Hardware
- Application I/O Interface
- Kernel I/O Subsystem
- Transforming I/O Requests to Hardware Operations
- STREAMS
- Performance

Overview

- I/O subsystem of the kernel separates the rest of the kernel from the complexities of managing I/O devices
  - The control of devices is a major concern of OS designers
- Two conflicting trends
  - Increasing standardization of software and hardware interfaces
  - Increasing variety of I/O devices
- **Device drivers** encapsulate the details and oddities of devices
  - Device drivers provide a uniform device-access interface to the I/O subsystem
    - As system calls provide a standard interface between the application and the operating system

Outline

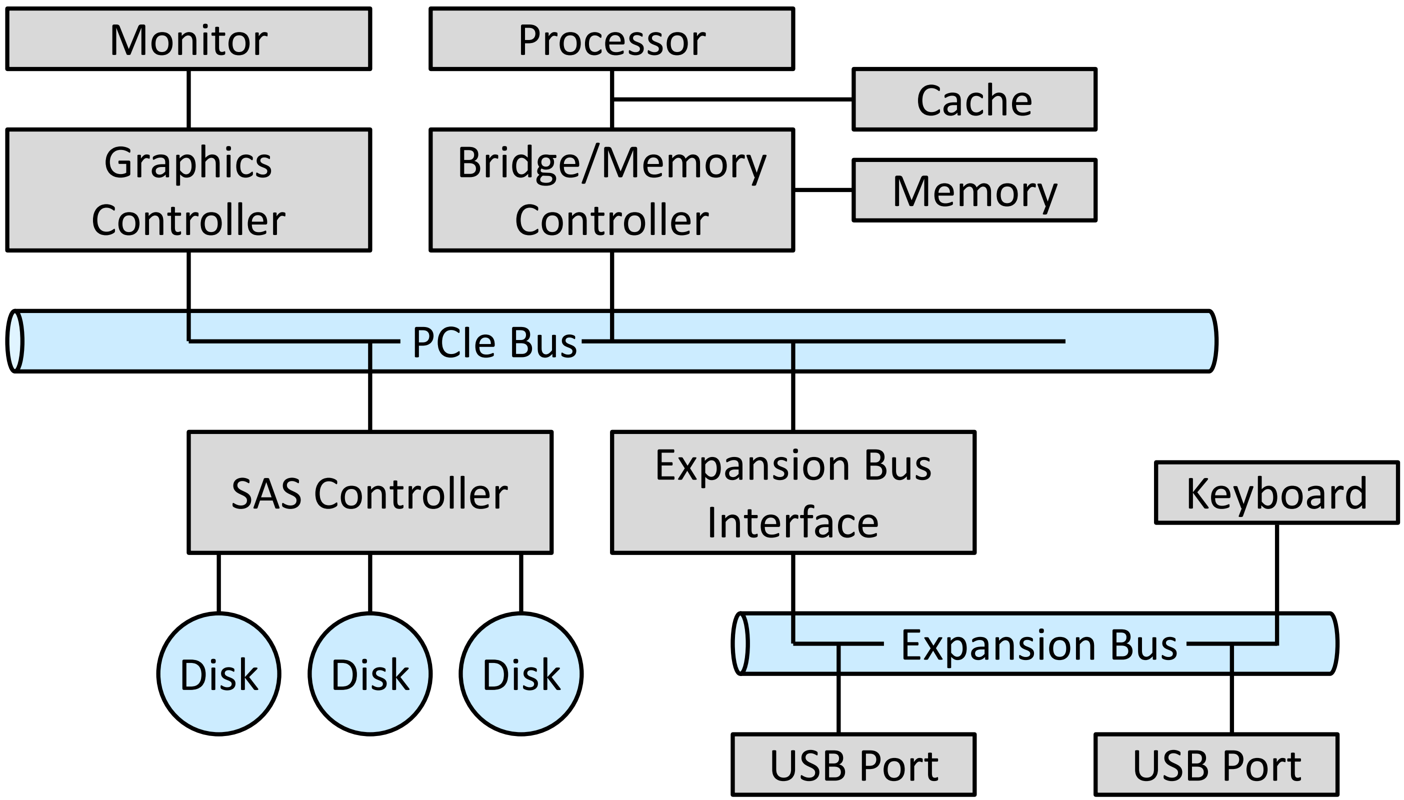
- Overview
- **<u>I/O Hardware</u>**
  - Memory-Mapped I/O
  - Polling
  - Interrupts
  - Direct Memory Access
- Application I/O Interface
- Kernel I/O Subsystem
- Transforming I/O Requests to Hardware Operations
- STREAMS
- Performance

I/O Hardware (1/2)

- Variety of I/O devices
  - Storage devices (e.g., disks, tapes)
  - Transmission devices (e.g., network connections, Bluetooth)
  - Human-interface devices (e.g., screen, keyboard, mouse, audio)
  - More specialized devices (e.g., steering of a jet)
- A device communicates with a computer system by signals
  - **Port**
    - The connection point for a device
  - **Bus** (like the Peripheral Component Interconnect (PCI) bus)
    - Devices share a common set of wires
  - A **daisy chain** usually operates as a bus
    - Device A has a cable that plugs into device B
    - Device B has a cable that plugs into device C
    - Device C plugs into a port on the computer

Typical PC Bus Structure

- A **PCIe bus** connects the processor-memory to fast devices
- An **expansion bus** connects relatively slow devices
- A **serial-attached SCSI** (SAS) bus plugged into an SAS controller connects disks



I/O Hardware (2/2)

Controller

- A collection of electronics that can operate a port, a bus, or a device
- **Serial-port controller**
  - A simple device controller
  - A single chip (or portion of a chip) in the computer that controls the signals on the wires of a serial port
- **Fibre channel** (FC) bus controller
  - More complex
  - Often implemented as a separate circuit board or a **host bus adapter** (HBA)
- Disk controller
  - Implement the disk side of the protocol for some kinds of connection (e.g., SAS, SATA)

Outline

- Overview
- **I/O Hardware**
  - **Memory-Mapped I/O**
  - Polling
  - Interrupts
  - Direct Memory Access
- Application I/O Interface
- Kernel I/O Subsystem
- Transforming I/O Requests to Hardware Operations
- STREAMS
- Performance

Memory-Mapped I/O (1/3)

- How does a processor give commands & data to a controller?
  - The controller has one or more registers for data and control signals
  - The processor reads and writes bit patterns in these registers
- Special (direct) I/O instructions
  - Specify the transfer of a byte or a word to an I/O port address
  - Trigger bus lines to select the proper device and move bits into or out of a device register
- **Memory-mapped I/O**
  - Device registers are mapped into the address space of the processor
  - The CPU uses the standard data transfer instructions to read and write the device registers at their mapped locations in physical memory

Memory-Mapped I/O (2/3)

- Today, most I/O is using memory-mapped I/O
  - It is simple to use
  - Writing millions of bytes to memory is faster than issuing millions of I/O instructions
- I/O device control typically consists of four registers
  - Status register: indicate states, such as
    - Whether the current command has completed
    - Whether a byte is available to be read from the data-in register
    - Whether a device error has occurred
  - Control register: written by the host to
    - Start a command, or
    - Change the mode of a device
  - Data-in register: read by the host to get input
  - Data-out register: written by the host to send output

Memory-Mapped I/O (3/3)

- Usual I/O port addresses for PCs

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)

Outline

- Overview
- **I/O Hardware**
  - Memory-Mapped I/O
  - **Polling**
  - Interrupts
  - Direct Memory Access
- Application I/O Interface
- Kernel I/O Subsystem
- Transforming I/O Requests to Hardware Operations
- STREAMS
- Performance

Handshaking

- Example of handshaking between a host and a controller
  - The host reads the **`busy`** bit in the **`status`** register until it becomes **`clear`**
  - The host sets the **`write`** bit in the **`command`** register and writes a byte into the **`data-out`** register
  - The host sets the **`command-ready`** bit in the **`command`** register
  - The controller sets the **`busy`** bit when it notices that the **`command-ready`** bit is set
  - The controller reads the **`command`** register, sees the **`write`** command, reads the **`data-out`** register, and does the I/O to the device
  - The controller clears
    - The **`command-ready`** bit
    - The **`error`** bit in the **`status`** register to indicate that the I/O succeeded
    - The **`busy`** bit to indicate that it is finished

Polling

- In step 1, the host is **busy-waiting** or **polling**
  - It is in a loop, reading the `status` register over and over until the `busy` bit becomes clear
  - If the controller and device are fast, this method is a reasonable one
  - Otherwise, the host should probably switch to another task
- Polling becomes inefficient when it is attempted repeatedly yet rarely finds a device ready for service
  - In such instances, it is more efficient for the hardware controller to **interrupt**
    - Notify the CPU when the device becomes ready for service
    - Not require the CPU to poll repeatedly for an I/O completion

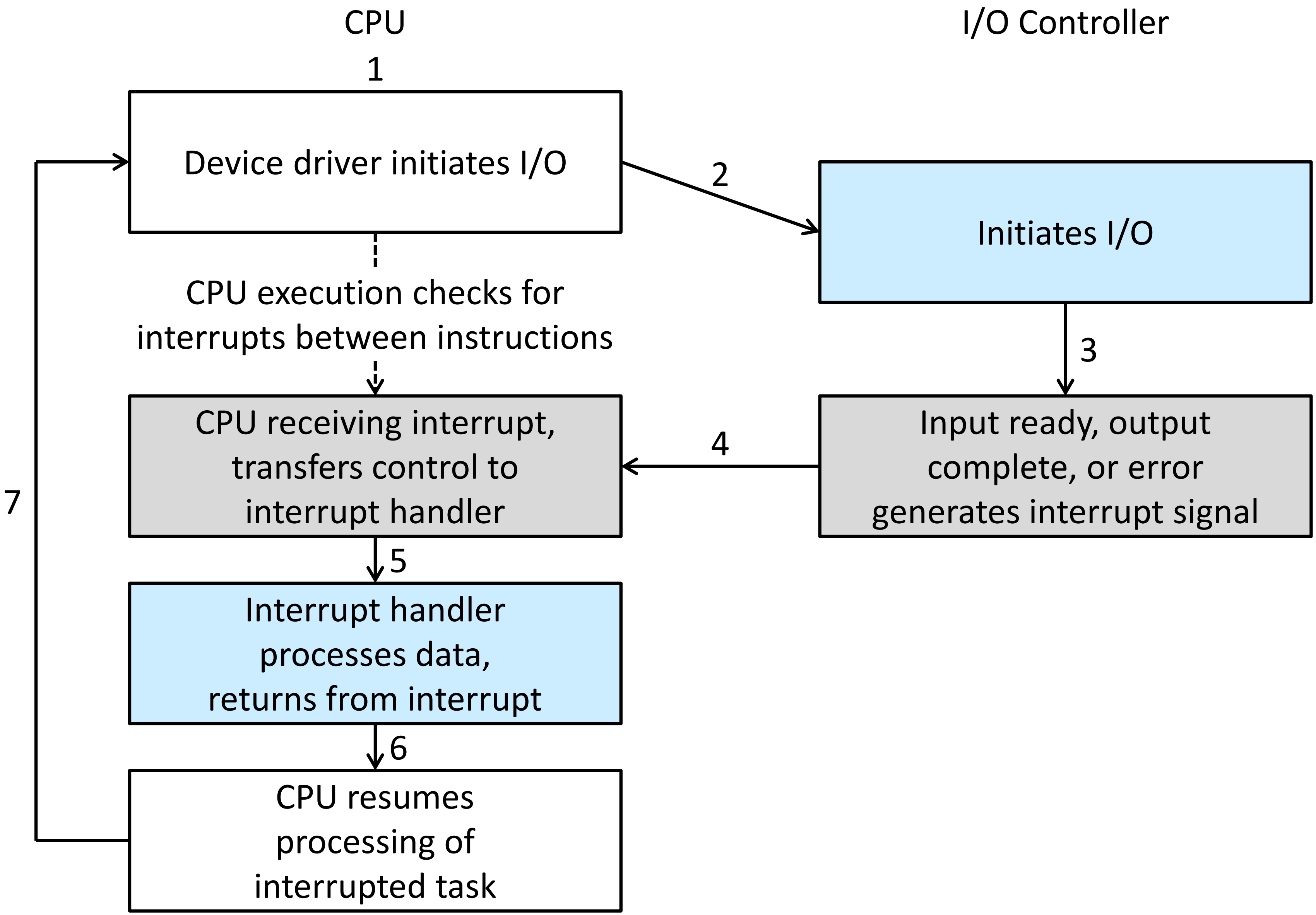
Outline

- Overview
- **I/O Hardware**
  - Memory-Mapped I/O
  - Polling
  - **Interrupts**
  - Direct Memory Access
- Application I/O Interface
- Kernel I/O Subsystem
- Transforming I/O Requests to Hardware Operations
- STREAMS
- Performance

Interrupts (1/2)

- **Interrupt-request line**
  - A wire that the CPU senses after executing every instruction
- **Interrupt-handler routine**
  - When the CPU detects a signal on the interrupt-request line, it performs a state save and jumps to the routine at a fixed address in memory
- **Interrupt vector**
  - An array (table) of pointers to interrupt routines
- There are frequent interrupts
  - Example: a quiet desktop computer performs almost 23,000 interrupts over 10 seconds

Interrupt-Driven I/O Cycle



Intel 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

Interrupts (2/2)

- Desired features
  - The ability to defer interrupt handling during critical processing
    - **Nonmaskable** and **maskable** interrupt request lines
  - An efficient way to dispatch to the proper interrupt handler for a device
    - Interrupt vector and **interrupt chaining**
  - Multilevel interrupts to distinguish between high- and low-priority interrupts and respond with the appropriate degree of urgency
    - **Interrupt priority levels**
  - A way for an instruction to get the operating system's attention directly
    - **Traps**
    - This item is not listed in Chapter 1

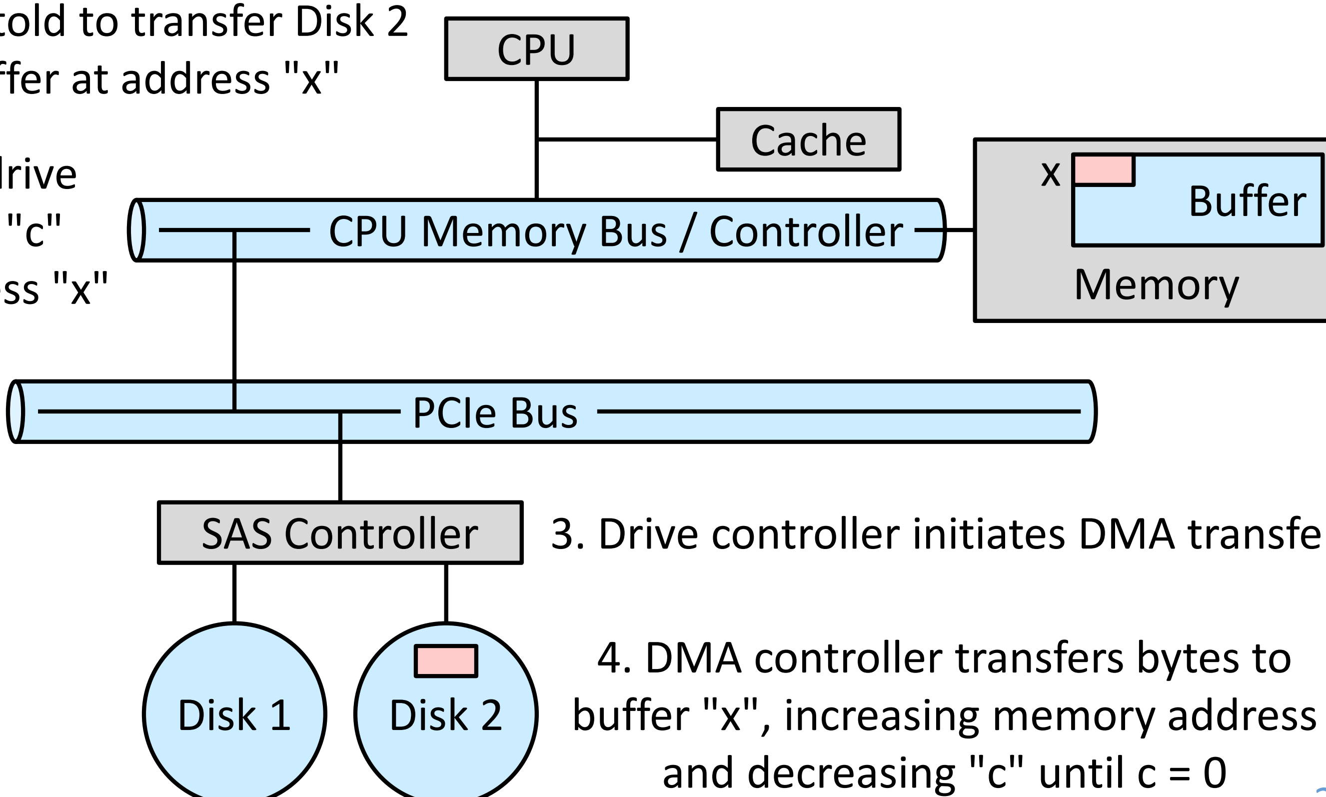
Outline

- Overview
- **I/O Hardware**
  - Memory-Mapped I/O
  - Polling
  - Interrupts
  - **Direct Memory Access**
- Application I/O Interface
- Kernel I/O Subsystem
- Transforming I/O Requests to Hardware Operations
- STREAMS
- Performance

Direct Memory Access (1/2)

- Avoid **programmed I/O** (one byte at a time) for large transfers
  - Direct memory access (DMA) vs. memory-mapped I/O
    - https://stackoverflow.com/questions/3851677/what-is-the-difference-between-dma-and-memory-mapped-io

1. Device driver is told to transfer Disk 2 data to buffer at address "x"
2. Device driver tells drive controller to transfer "c" bytes to buffer at address "x"
3. Drive controller initiates DMA transfer
4. DMA controller transfers bytes to buffer "x", increasing memory address and decreasing "c" until c = 0
5. When c = 0, DMA interrupts CPU to signal transfer completion



Direct Memory Access (2/2)

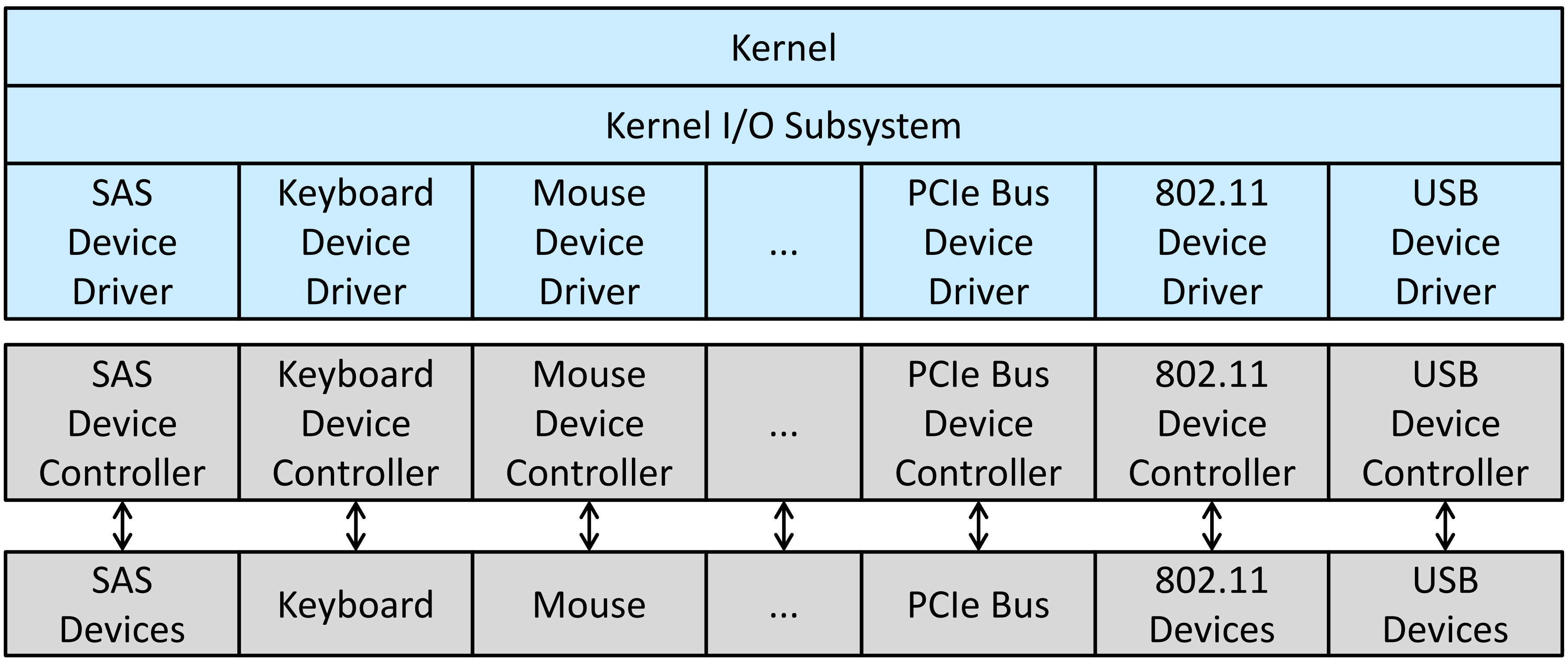
- **Scatter-gather** method
  - A command block can include a list of sources and destinations addresses that are not contiguous
- **Double buffering** (inefficient)
  - It is straightforward for the target address to be in kernel address space
  - To get the data to the user space, a second copy operation is needed
- **DMA-request** and **DMA-acknowledge** wires
  - For handshaking between the DMA controller and the device controller
- **Cycle stealing**
  - When the DMA controller seizes the memory bus, the CPU is momentarily prevented from accessing main memory
- **Direct virtual memory access** (DVMA)
  - Use virtual addresses

Outline

- Overview
- I/O Hardware
- **Application I/O Interface**
  - Block and Character Devices
  - Network Devices
  - Clocks and Timers
  - Nonblocking and Asynchronous I/O
  - Vectored I/O
- Kernel I/O Subsystem
- Transforming I/O Requests to Hardware Operations
- STREAMS
- Performance

Application I/O Interface

- Device drivers hide the differences among device controllers from the I/O subsystem of the kernel
  - I/O system calls encapsulate the behavior of devices in a few general kinds that hide hardware differences from applications
  - Each kind is accessed via a standardized set of functions, an **interface**



Characteristics of I/O Devices (1/2)

Aspect	Variation	Example
Data-Transfer Mode	Character Block	Terminal Disk
Access Method	Sequential Random	Modem CD-ROM
Transfer Schedule	Synchronous Asynchronous	Tape Keyboard
Sharing	Dedicated Sharable	Tape Keyboard
Device Speed	Latency Seek Time Transfer Rate Delay Between Operations	
I/O Direction	Read Only Write Only Read-Write	CD-ROM Graphics Controller Disk

Characteristics of I/O Devices (2/2)

- The device categories are fairly standard
  - Block I/O
  - Character-stream I/O
  - Memory-mapped file access
  - Network sockets
- Most operating systems also have an **escape** (or **back door**)
  - Transparently pass commands from an application to a device driver
  - Example:
    - UNIX system call `ioctl()` enables an application to access any functionality that can be implemented by any device driver
- The device identifier in UNIX and Linux is a tuple
  - Major number: the device type
  - Minor number: the instance of that device

Outline

- Overview
- I/O Hardware
- **Application I/O Interface**
  - Block and Character Devices
  - Network Devices
  - Clocks and Timers
  - Nonblocking and Asynchronous I/O
  - Vectored I/O
- Kernel I/O Subsystem
- Transforming I/O Requests to Hardware Operations
- STREAMS
- Performance

Block and Character Devices

- **<u>Block-device interface</u>** captures all the aspects necessary for accessing disk drives and other block-oriented devices
  - `read()`, `write()`, and `seek()`
  - <u>Raw I/O</u>
    - Raw-device access passes control of the device directly to the application, letting the operating system step out of the way
  - <u>Direct I/O</u>
    - The operating system allows a mode of operation on a file that disables buffering and locking
  - Memory-mapped file access can be layered on top of block-device drivers
- **<u>Character-stream interface</u>**
  - `get()` or `put()` enables an application to get or put one character
    - On top of the interface, libraries can be built to offer line-at-a-time access

Network Devices

- Most operating systems provide a network I/O interface that is different from the interface used for disks
  - The performance and addressing characteristics of network I/O differ significantly from those of disk I/O
- Network **socket** interface
  - Also provide a function called `select()` that manages a set of sockets
    - Return information about which sockets have a packet waiting to be received and which sockets have room to accept a packet to be sent
- Many other approaches
  - UNIX: half-duplex pipes, full-duplex FIFOs, full-duplex STREAMS, message queues, and sockets

Clocks and Timers

- Three basic functions
  - Give the current time
  - Give the elapsed time
  - Set a timer to trigger operation X at time T
- **Programmable interval timer** can be set to
  - Wait a certain amount of time and generate an interrupt
  - Do this once or to repeat the process to generate periodic interrupts
- **High-performance event timer** (HPET)
- **Network time protocol**
  - Use sophisticated latency calculations to keep a computer's clock accurate almost to atomic-clock levels

Nonblocking and Asynchronous I/O

- With a **blocking** system call, the calling thread is suspended
- Some user-level processes need **nonblocking** I/O
  - Example: receive keyboard input while displaying data on screen
  - Implemented via multi-threading
- An alternative to a nonblocking system call is an **asynchronous** system call
  - It returns immediately without waiting for the I/O to complete
- Comparison
  - A nonblocking `read()` returns immediately with whatever data are available
  - An asynchronous `read()` requests a transfer that will be performed in its entirety but will complete at some future time

Vectored I/O

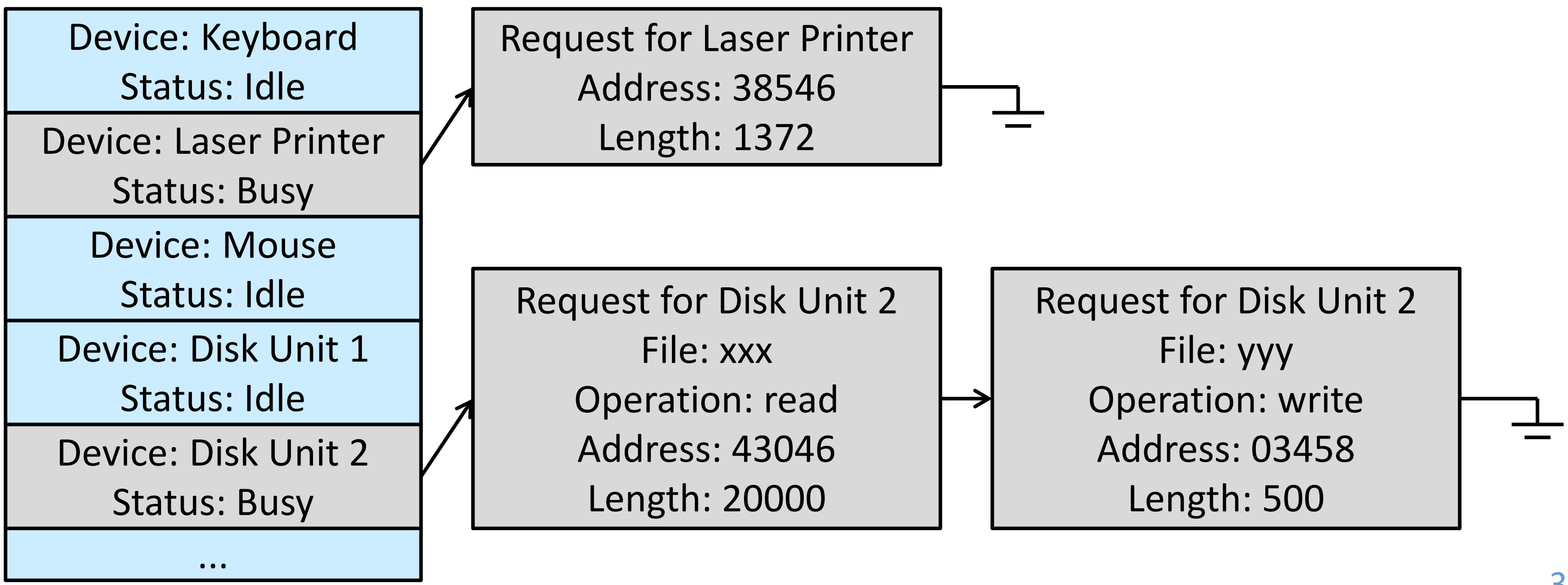
- Allow one system call to perform multiple I/O operations involving multiple locations
  - Example: the UNIX `readv` system call accepts a vector of multiple buffers to read into or write from
- This **scatter-gather** method is better than several individual system calls
  - Avoid context-switching and system-call overhead
  - Assure that all the I/O is done without interruption if atomicity is provided
    - Avoid corruption of data if other threads are also performing I/O involving those buffers

Outline

- Overview, I/O Hardware, Application I/O Interface
- **Kernel I/O Subsystem**
  - I/O Scheduling
  - Buffering
  - Caching
  - Spooling and Device Reservation
  - Error Handling
  - I/O Protection
  - Kernel Data Structures
  - Power Management
- Transforming I/O Requests to Hardware Operations, STREAMS, Performance

I/O Scheduling

- Maintain a wait queue of requests for each device
  - The I/O scheduler rearranges the order of the queue to improve the overall system efficiency and the average response time
  - The wait queue may be attached to a device-status table
    - When a kernel supports asynchronous I/O, it must be able to keep track of many I/O requests at the same time



Buffering

- Store data (in memory) being transferred between two devices or between a device and an application
  - Cope with a speed mismatch between the producer and consumer of a data stream
    - **Double buffering** decouples the producer of data from the consumer, thus relaxing timing requirements between them
  - Provide adaptations for devices that have different data-transfer sizes
    - Especially common in computer networking
  - Support **copy semantics** for application I/O
    - With copy semantics, the version of the data written to disk is guaranteed to be the version at the time of the application system call

Caching

- A region of fast memory that holds copies of data
  - Comparison
    - A buffer may hold the only existing copy of a data item
    - A cache holds a copy on faster storage of an item that resides elsewhere
  - Caching and buffering are distinct functions, but sometimes a region of memory can be used for both purposes

Spooling and Device Reservation

- **Spool**
  - A buffer that holds output for a device that cannot accept interleaved data streams
    - Example: printer
- Device reservation
  - Exclusive device access by enabling a process to allocate an idle device and to deallocate that device when it is no longer needed
    - Should avoid deadlock

Error Handling

- Devices and I/O transfers can fail in many ways
- Operating systems can often compensate effectively for transient failures
  - A disk **`read()`** failure results in a **`read()`** retry
  - A network **`send()`** error results in a **`resend()`**
- An I/O system call usually return one bit of information about the status (success or failure) of the call
  - Example: SCSI
    - A **sense key** identifies the general nature of the failure
    - An **additional sense code** states the category of failure
    - An **additional sense-code qualifier** gives even more detail
  - Error-log information

I/O Protection

- A user process may accidentally or purposely disrupt the normal operation by illegal I/O instructions
- Define all I/O instructions to be privileged instructions
  - A user cannot issue I/O instructions directly
  - A user executes a system call to request that the operating system perform I/O on its behalf
    - Trap to kernel
    - Perform I/O
    - Return to calling thread
- Any memory-mapped and I/O port memory locations must be protected from user access

Kernel Data Structures

- The kernel uses data structures to track
  - Open files
  - Network connections
  - Character-device communications
  - Other I/O activities
- Some operating systems use object-oriented methods
  - Example: Windows uses a message-passing implementation for I/O
  - Convert an I/O request into a message
    - For output, the message contains the data to be written
    - For input, the message contains a buffer to receive the data
  - Add overhead
    - By comparison with procedural techniques that use shared data structures
  - Simplify the structure and design of the I/O system and add flexibility

Power Management (1/2)

- Motivations
  - Greenhouse gas reduction
  - Cooling
    - Cooling a data center may use twice as much electricity as powering
- Operating systems play a role in power use
- Power collapse
  - The ability to put a device into a very deep sleep state
    - The device uses only marginally more power than powering off
    - The device is still able to respond to external stimuli

Power Management (2/2)

- Example: Android
  - Component-level power management in Android
    - Build a device tree to understand the relationship between components
    - If a component is unused, turn it off
    - If all components on a bus are unused, turn the bus off
    - If all components in the device tree are unused, maybe let the system enter power collapse
  - Wakelock
    - Temporarily prevent the system from entering power collapse
- **Advanced configuration and power interface** (ACPI)
  - Provide code that runs as routines callable by the kernel for
    - Device state discovery and management
    - Device error management
    - Power management

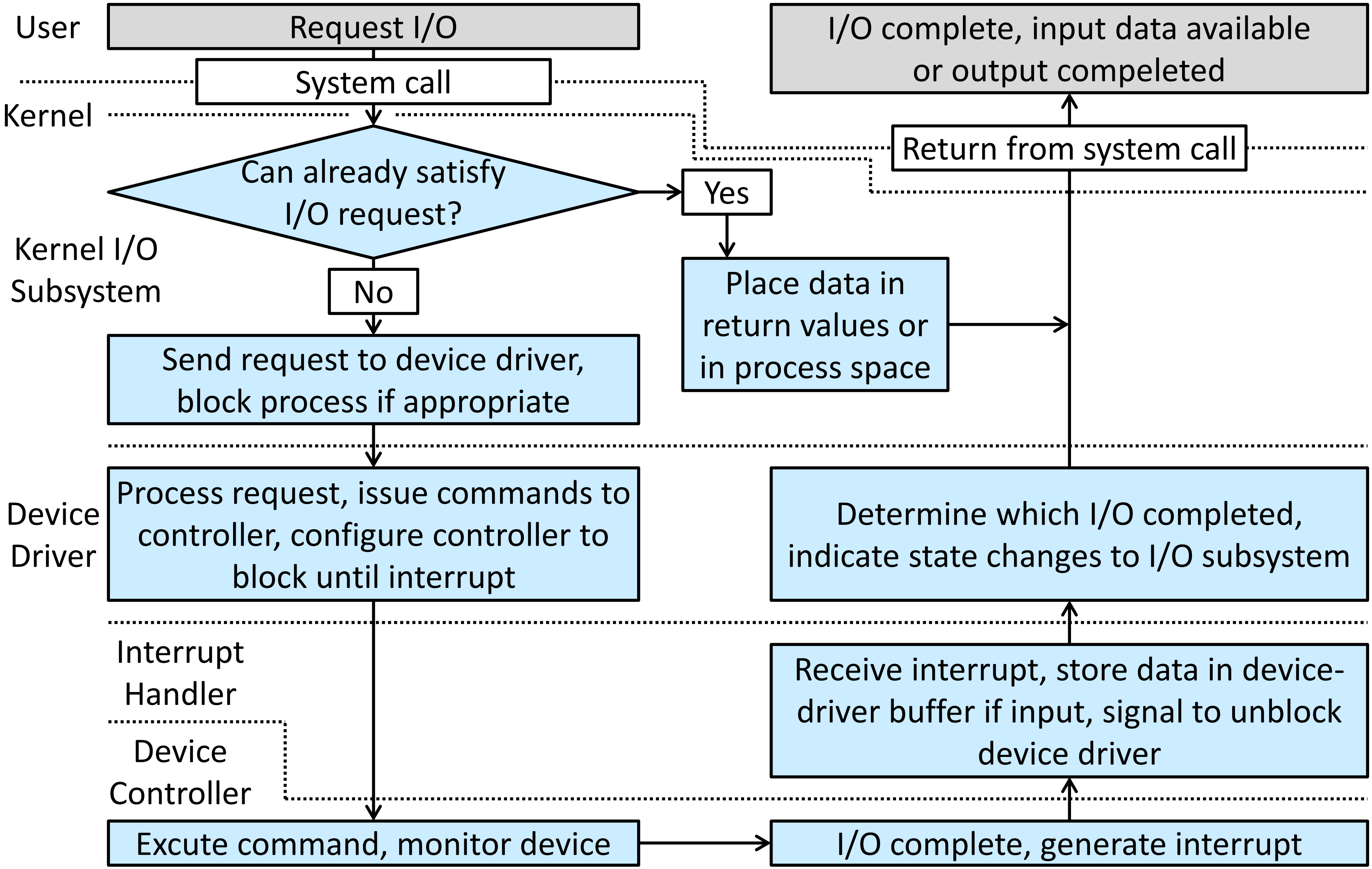
Outline

- Overview
- I/O Hardware
- Application I/O Interface
- Kernel I/O Subsystem
- **Transforming I/O Requests to Hardware Operations**
- STREAMS
- Performance

Transform I/O Requests to HW Operations

- Example: reading a file from disk
  - Refer to the data by a file name
  - Map from the file name through the file-system directories
    - To obtain the space allocation of the file
  - Physically read data from the disk into a buffer
  - Make the data available to the requesting process
  - Return control to the process

Life Cycle of Blocking Read Request

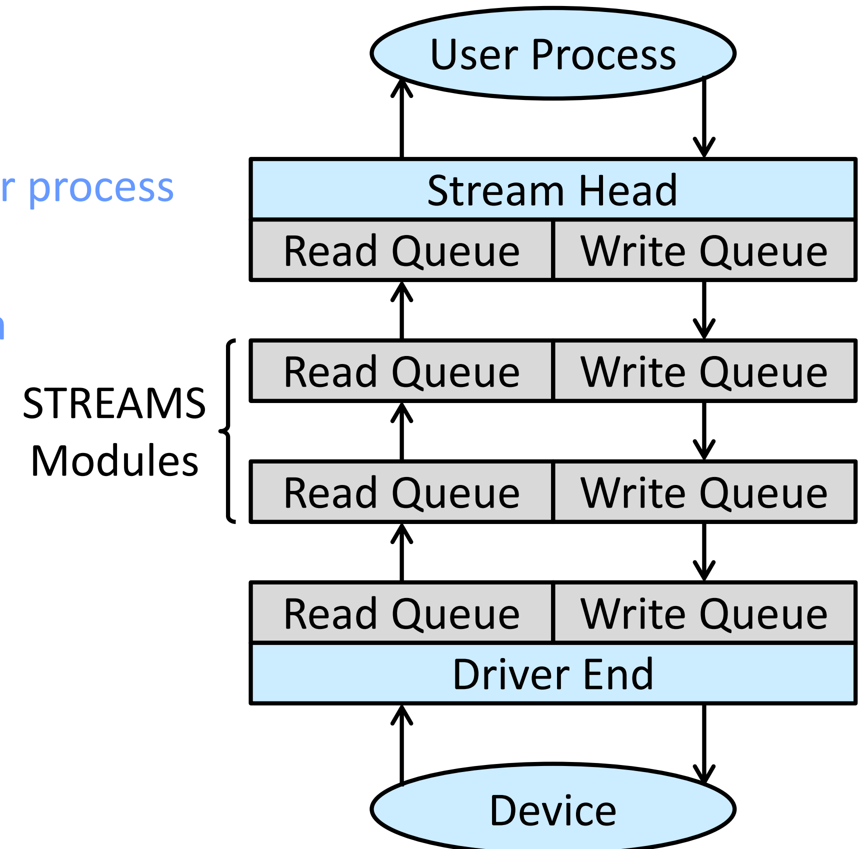


Outline

- Overview
- I/O Hardware
- Application I/O Interface
- Kernel I/O Subsystem
- Transforming I/O Requests to Hardware Operations
- **STREAMS**
- Performance

STREAMS

- An interesting approach from UNIX System V
  - Benefit: a modular and incremental approach
- A full-duplex connection between a device driver and a user-level process
  - A STREAM consists of
    - A stream head interfacing with the user process
    - A driver end controlling the device
    - Zero or more stream modules between the stream head and the driver end
  - Each of these components contains a pair of read and write queues
  - Message passing is used to transfer data between queues
    - A queue may support flow control



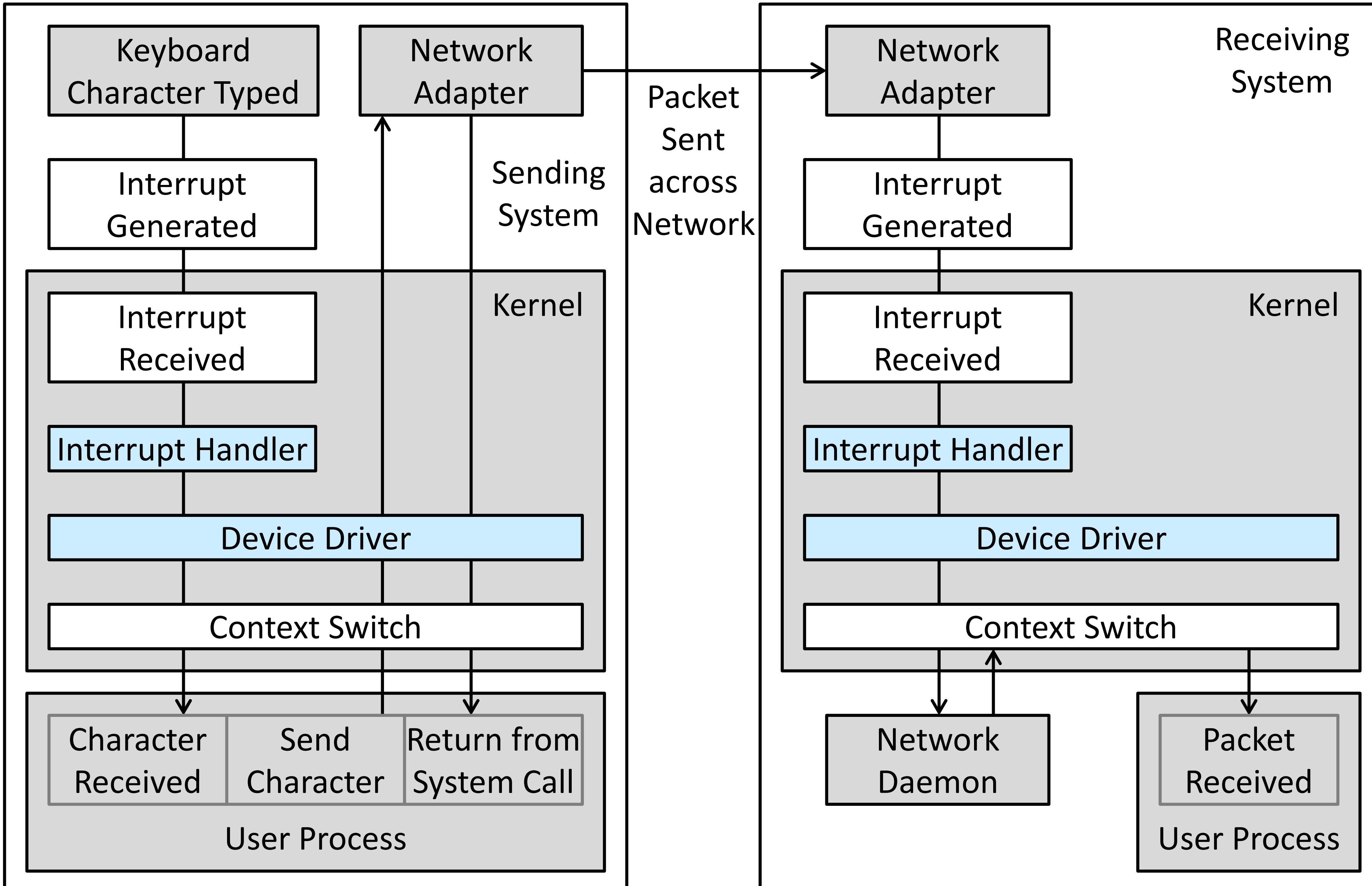
Outline

- Overview
- I/O Hardware
- Application I/O Interface
- Kernel I/O Subsystem
- Transforming I/O Requests to Hardware Operations
- STREAMS
- **Performance**

Performance

- I/O is a major factor in system performance
  - Heavy demands on the CPU to
    - Execute device-driver code
    - Schedule processes fairly and efficiently as they block and unblock
  - Resulting context switches
  - Interrupt-handling mechanisms in the kernel
  - Data copies between
    - Controllers and physical memory
    - Kernel buffers and application data space
  - Network traffic

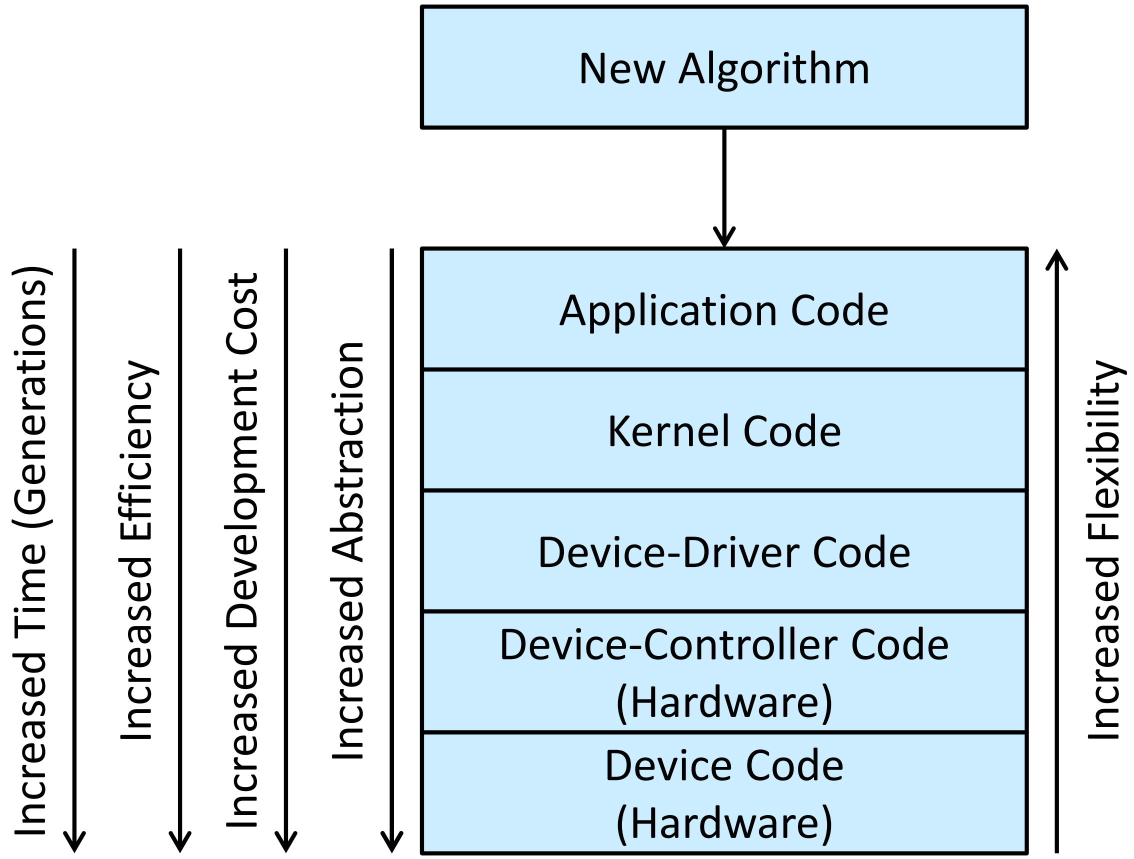
Intercomputer Communications



Improving I/O Performance

- Reduce the number of context switches
- Reduce the number of times that data must be copied
- Reduce the frequency of interrupts by large transfers, smart controllers, and polling (if busy waiting can be minimized)
- Increase concurrency by using DMA-knowledgeable controllers or channels
  - To offload simple data copying from the CPU
- Move processing primitives into hardware
  - To allow their operation in device controllers to be concurrent with CPU and bus operation
- Balance CPU, memory subsystem, bus, and I/O performance
  - Because an overload in any one area will cause idleness in others

Device Functionality Progression



Objectives

- Explore the structure of an operating system's I/O subsystem
- Discuss the principles and complexities of I/O hardware
- Explain the performance aspects of I/O hardware and software

Q&A