

CISC 7310X

C02c: Application I/O and  
OS I/O Subsystem

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

# Acknowledgement

- This slides are a revision of the slides by the authors of the textbook

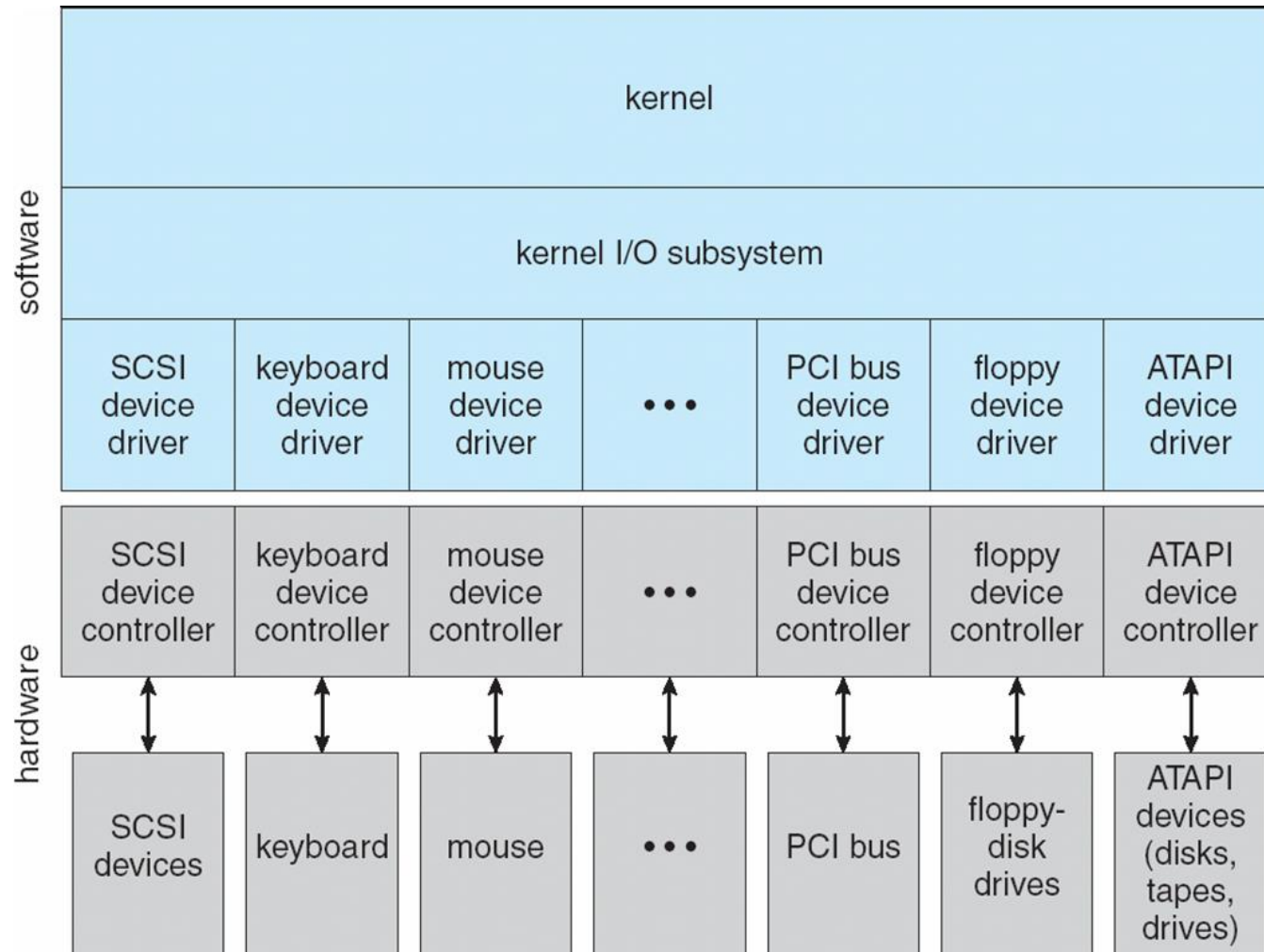
# Outline

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

# Application I/O Interface

- I/O system calls encapsulate device behaviors in generic classes
- Device-driver layer hides differences among I/O controllers from kernel
- New devices talking already-implemented protocols need no extra work
- Each OS has its own I/O subsystem structures and device driver frameworks
- Devices vary in many dimensions
  - Character-stream or block
  - Sequential or random-access
  - Synchronous or asynchronous (or both)
  - Sharable or dedicated
  - Speed of operation
  - read-write, read only, or write only

# A Kernel I/O Structure



# Characteristics of I/O Devices

- Need to understand general characteristics to achieve device independent
- A couple of dimensions
  - Size of transfer: Character-stream or block
  - Access order: sequential or random access
  - Predictability and responsiveness: Synchronous and asynchronous
  - Shared or dedicated
  - Speed of operation, e.g., latency, seek time, transfer rate
  - Read-write, read only, or write only

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

# Types of I/O Devices

- 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 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

# Block Devices: Examples

- Naming
  - Examples on Linux
    - by label, by uuid, by id, and by path
    - Running examples
      - `lsblk -f`
      - `ls /dev/disk/`
- Read and write a block a time
- Essential behavior
  - `read()`, `write()`
  - For random-access block devices
    - `seek()`

# Character Devices

- Character devices include keyboards, mice, serial ports
  - Read and write a character a time
  - Essential behavior
    - `get()`, `put()`
  - Libraries layered on top allow line editing

# Network Devices

- Varying enough from block and character to have own interface
- Linux, Unix, Windows and many others include **socket** interface
  - Separates network protocol from network operation
  - Includes `select()` functionality
- Approaches vary widely (pipes, FIFOs, streams, queues, mailboxes)

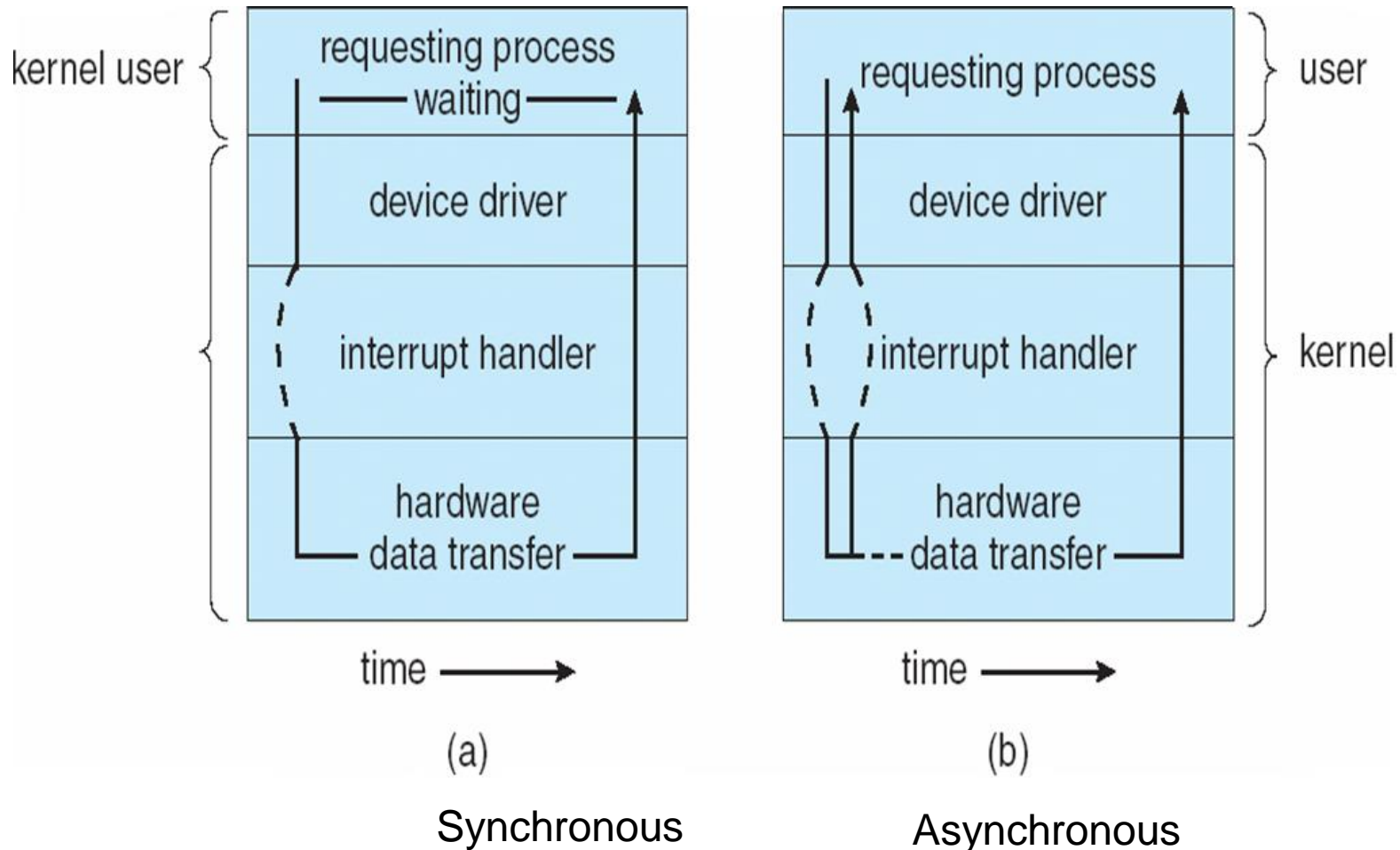
# Clocks and Timers

- Provide current time, elapsed time, timer
- Normal resolution about 1/60 second
- Some systems provide higher-resolution timers
- **Programmable interval timer** used for timings, periodic interrupts
- `ioctl()` (on UNIX) covers odd aspects of I/O such as clocks and timers

# Nonblocking and Asynchronous I/O

- **Blocking** - process suspended until I/O completed
  - Easy to use and understand
  - Insufficient for some needs
- **Nonblocking** - I/O call returns as much as available
  - User interface, data copy (buffered I/O)
  - Implemented via multi-threading
  - Returns quickly with count of bytes read or written
  - `select()` to find if data ready then `read()` or `write()` to transfer
- **Asynchronous** - process runs while I/O executes
  - Difficult to use
  - I/O subsystem signals process when I/O completed

# Two I/O Methods



# Vectored I/O

- **Vectored I/O** allows one system call to perform multiple I/O operations
- For example, Unix `readve()` accepts a vector of multiple buffers to read into or write from
- This scatter-gather method better than multiple individual I/O calls
  - Decreases context switching and system call overhead
  - Some versions provide atomicity
    - Avoid for example worry about multiple threads changing data as reads / writes occurring



# Questions

- Characteristics of I/O devices
- Types of I/O devices

# Kernel I/O Subsystem: Scheduling

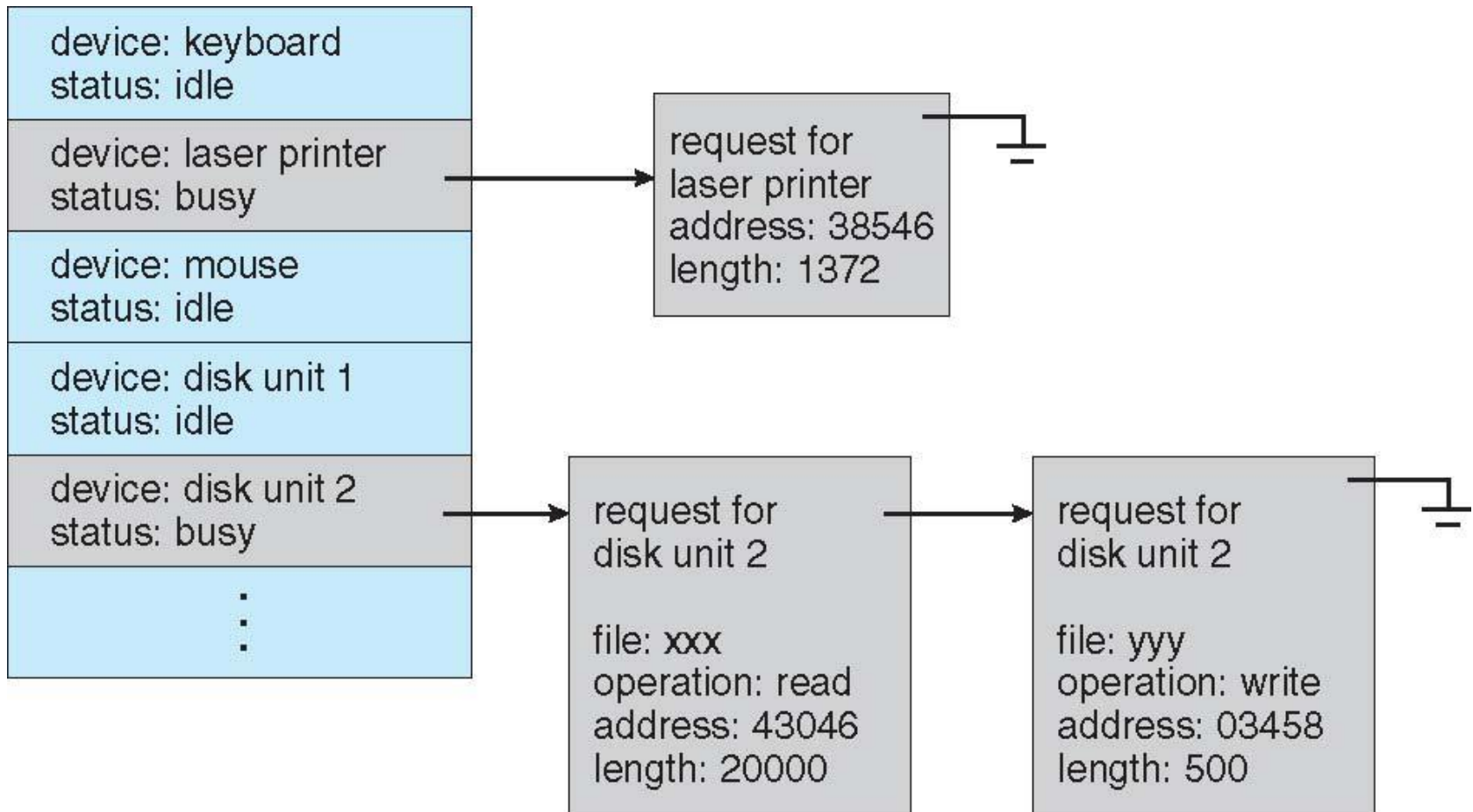
- Scheduling
  - Some I/O request ordering via per-device queue
  - Some OSs try fairness
  - Some implement Quality Of Service (i.e. IPQOS)

# Kernel I/O Subsystem:

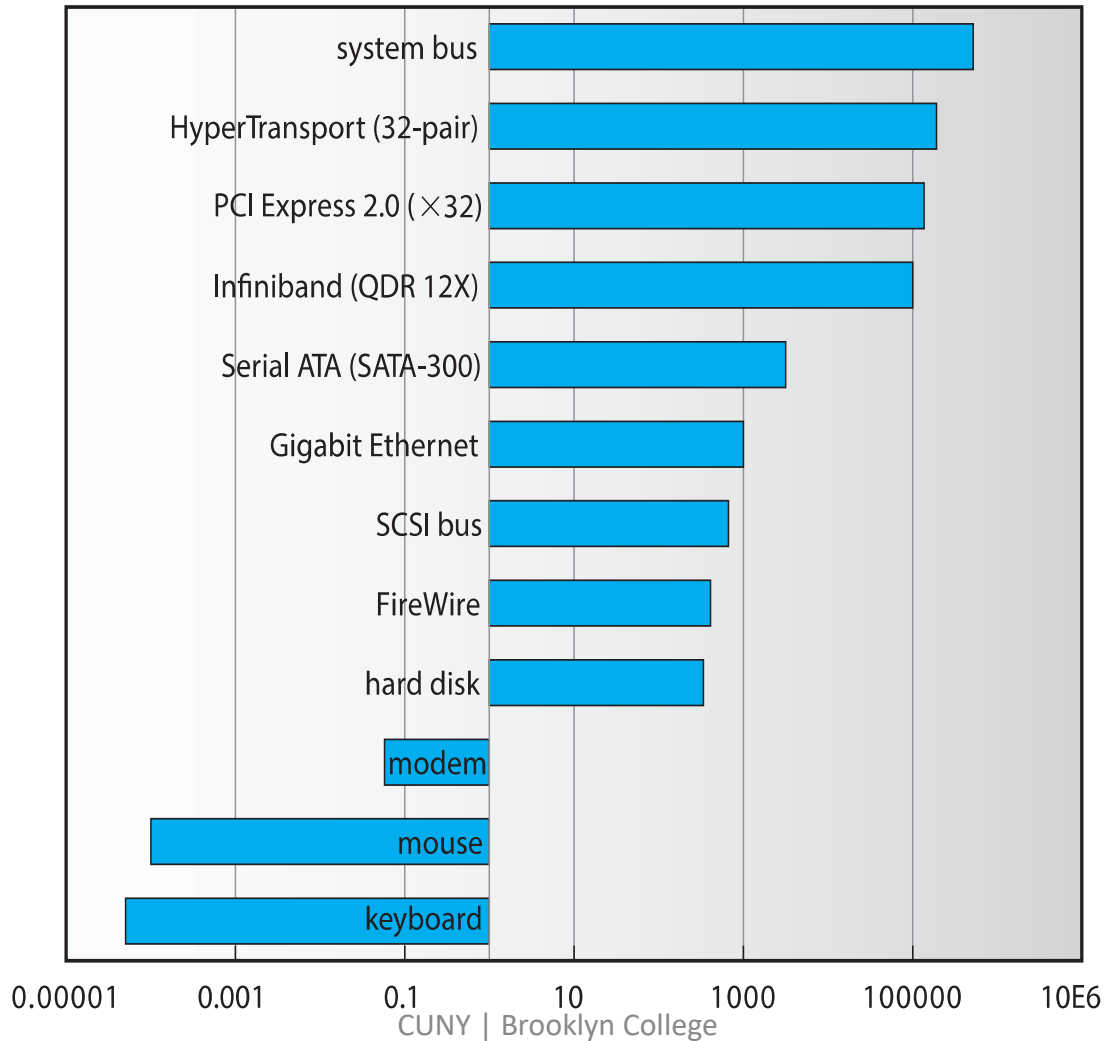
## Buffering

- **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

# Device Status Table



# Sun Enterprise 6000 Device-Transfer Rates



# Kernel I/O Subsystem

- **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

# Error Handling

- OS can recover from disk read, device unavailable, transient write failures
  - Retry a read or write, for example
  - Some systems more advanced - Solaris FMA, AIX
    - Track error frequencies, stop using device with increasing frequency of retry-able errors
- Most return an error number or code when I/O request fails
- System error logs hold problem reports

# Questions?

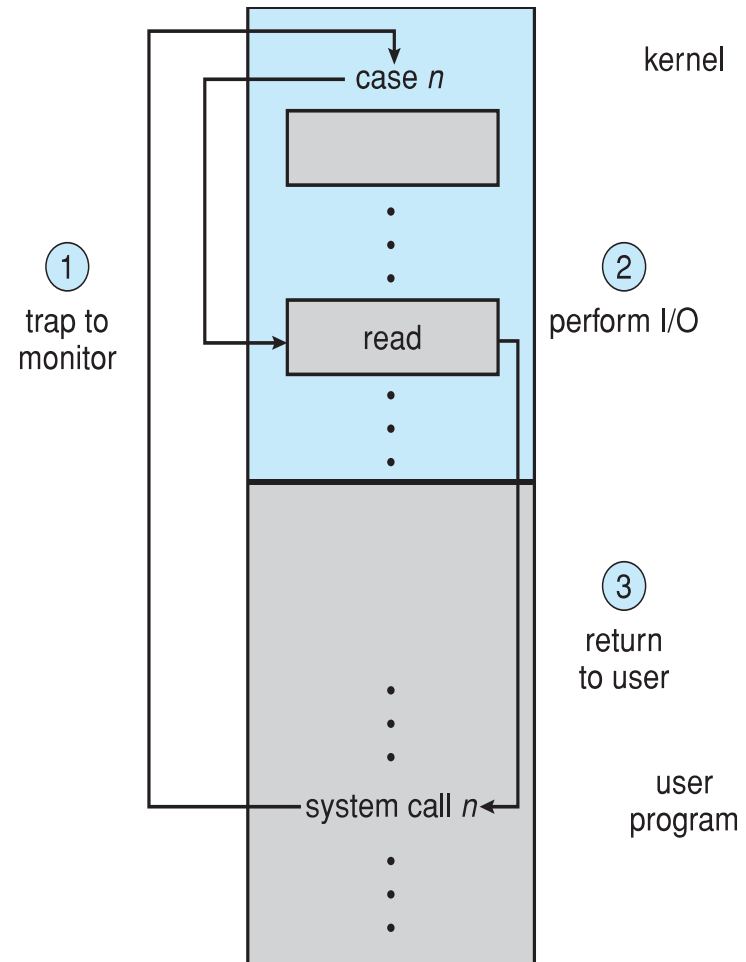
- Some considerations of kernel I/O design



# I/O Protection

- User process may accidentally or purposefully attempt to disrupt normal operation via illegal I/O instructions
  - All I/O instructions defined to be privileged
  - I/O must be performed via system calls
    - Memory-mapped and I/O port memory locations must be protected too

# Use of a System Call to Perform I/O



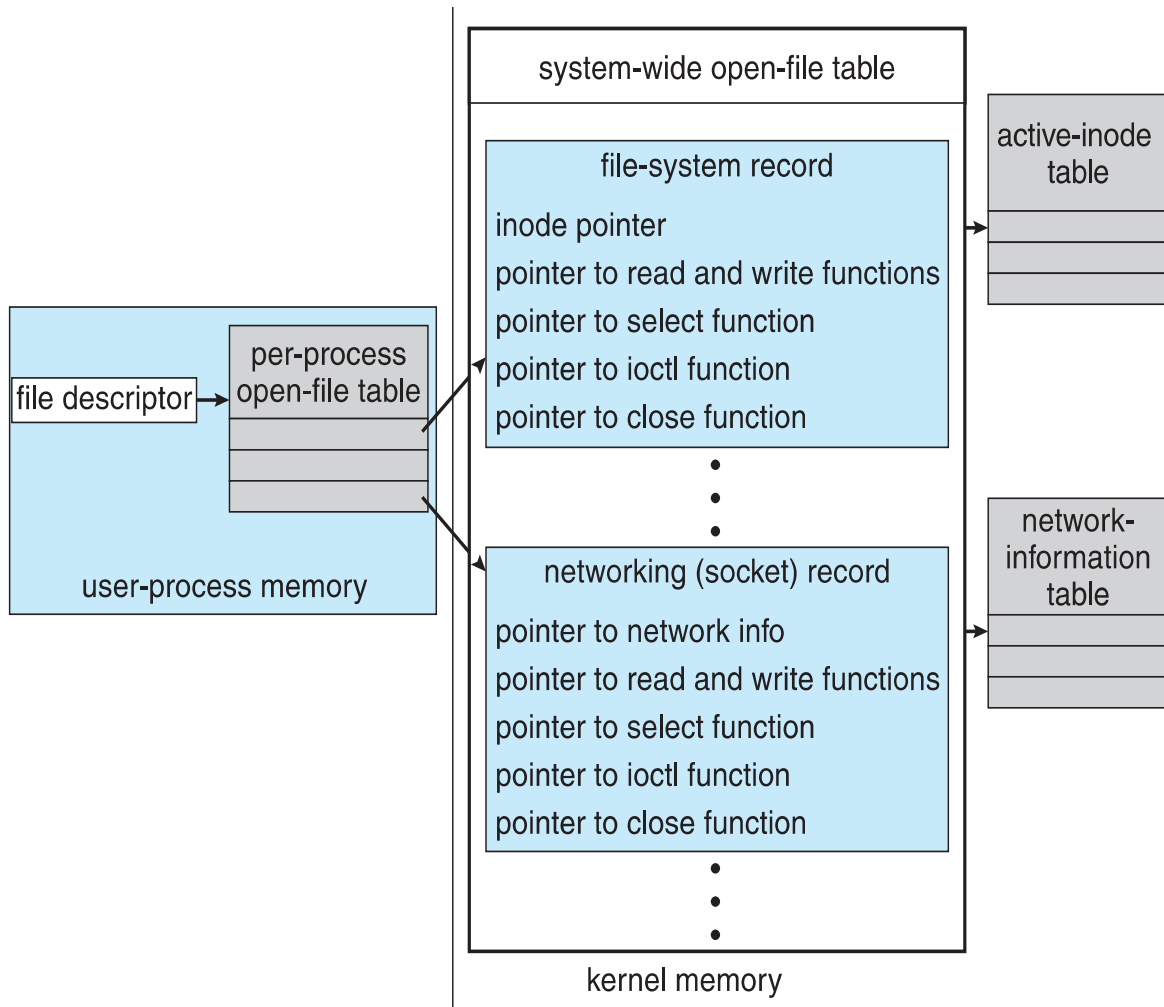
# Questions?

- Design for I/O protection

# Kernel Data Structures for I/O

- Kernel keeps state info for I/O components, including open file tables, network connections, character device state
- Many, many complex data structures to track buffers, memory allocation, "dirty" blocks
- Some use object-oriented methods and message passing to implement I/O
  - Windows uses message passing
    - Message with I/O information passed from user mode into kernel
    - Message modified as it flows through to device driver and back to process
    - Pros / cons?

# UNIX I/O Kernel Structure



# Questions?

- Kernel data structures for I/O

# Power Management

- Not strictly domain of I/O, but much is I/O related
- Computers and devices use electricity, generate heat, frequently require cooling
- 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

# Power Management: Examples

- 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)

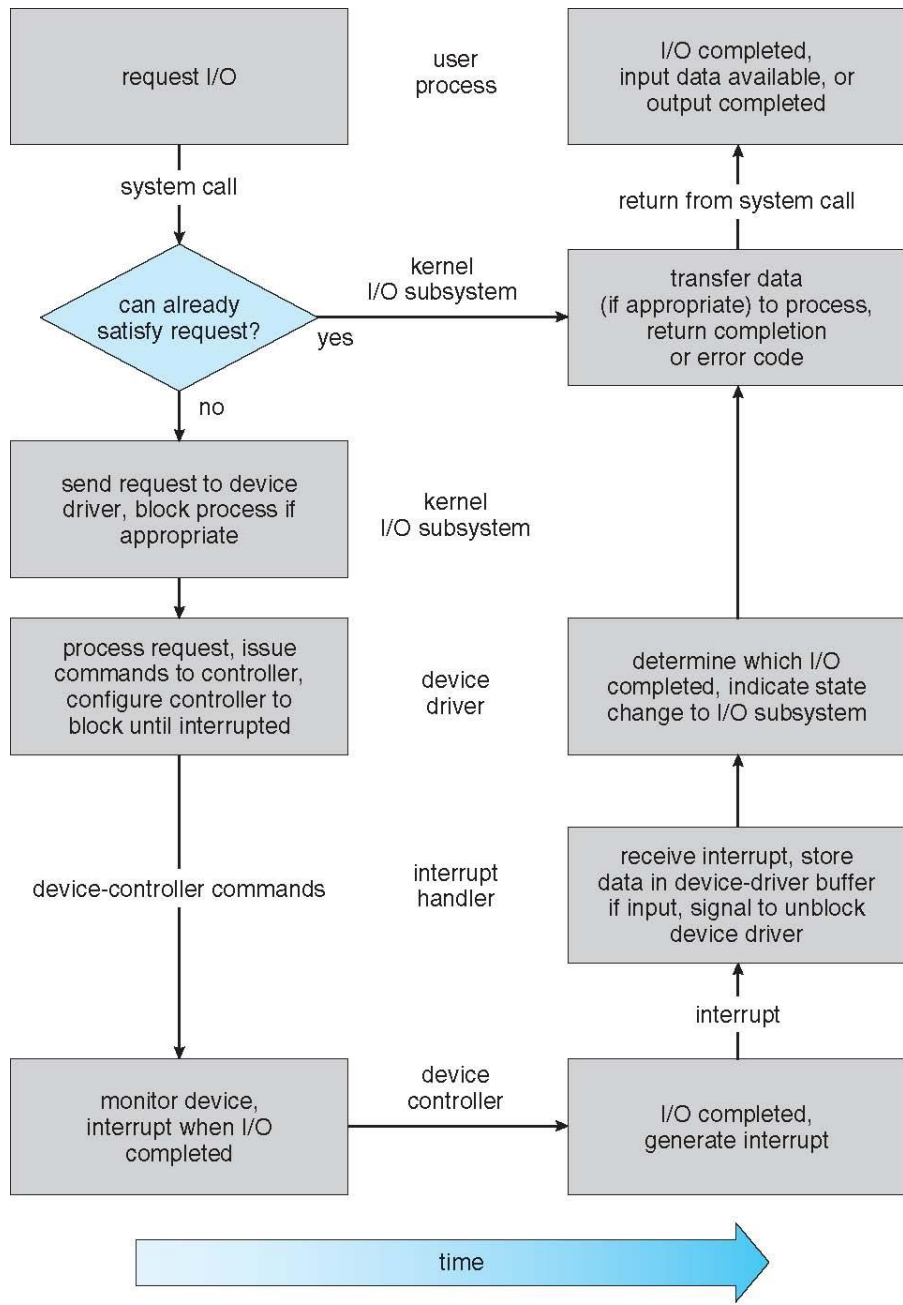


# Questions?

- I/O and power management?

# Life Cycle of An I/O Request

- Consider reading a file from disk for a process:
  - Determine device holding file
  - Translate name to device representation
  - Physically read data from disk into buffer
  - Make data available to requesting process
  - Return control to process



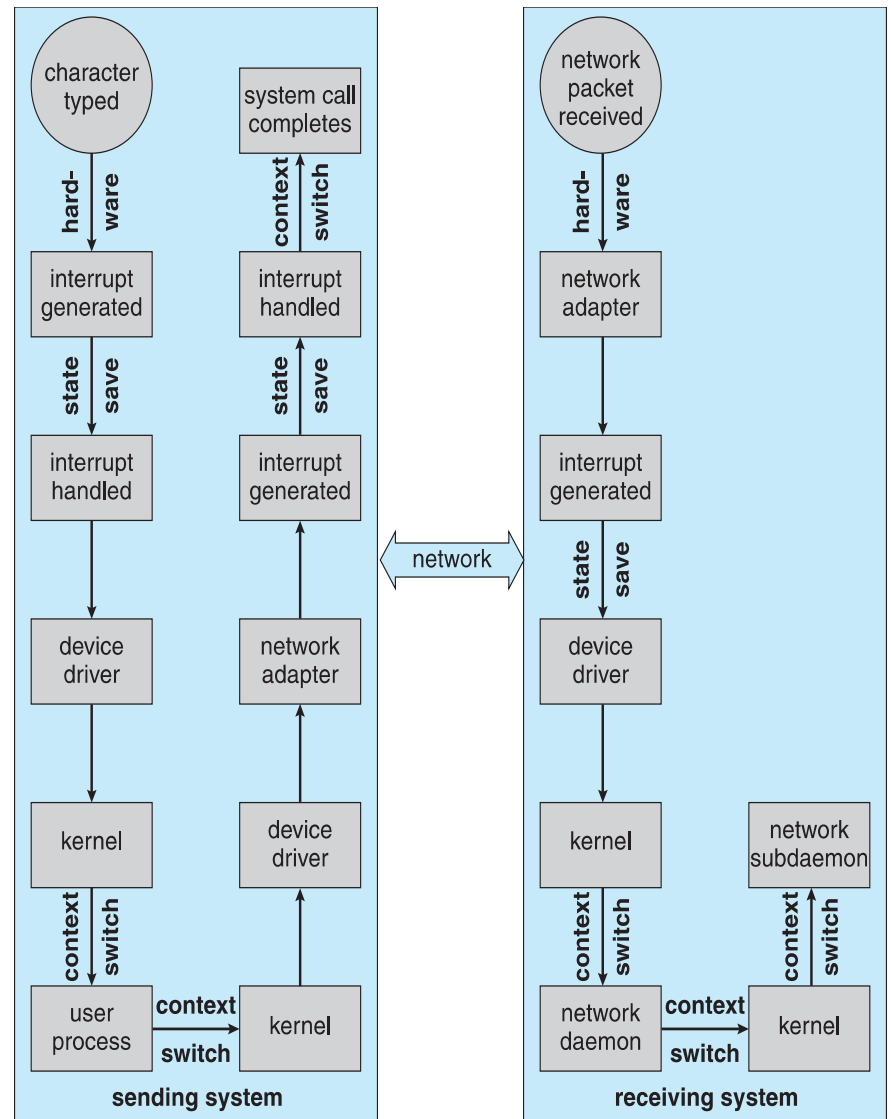
# Questions?

- I/O request operation from beginning to end (life cycle)?

# I/O and 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

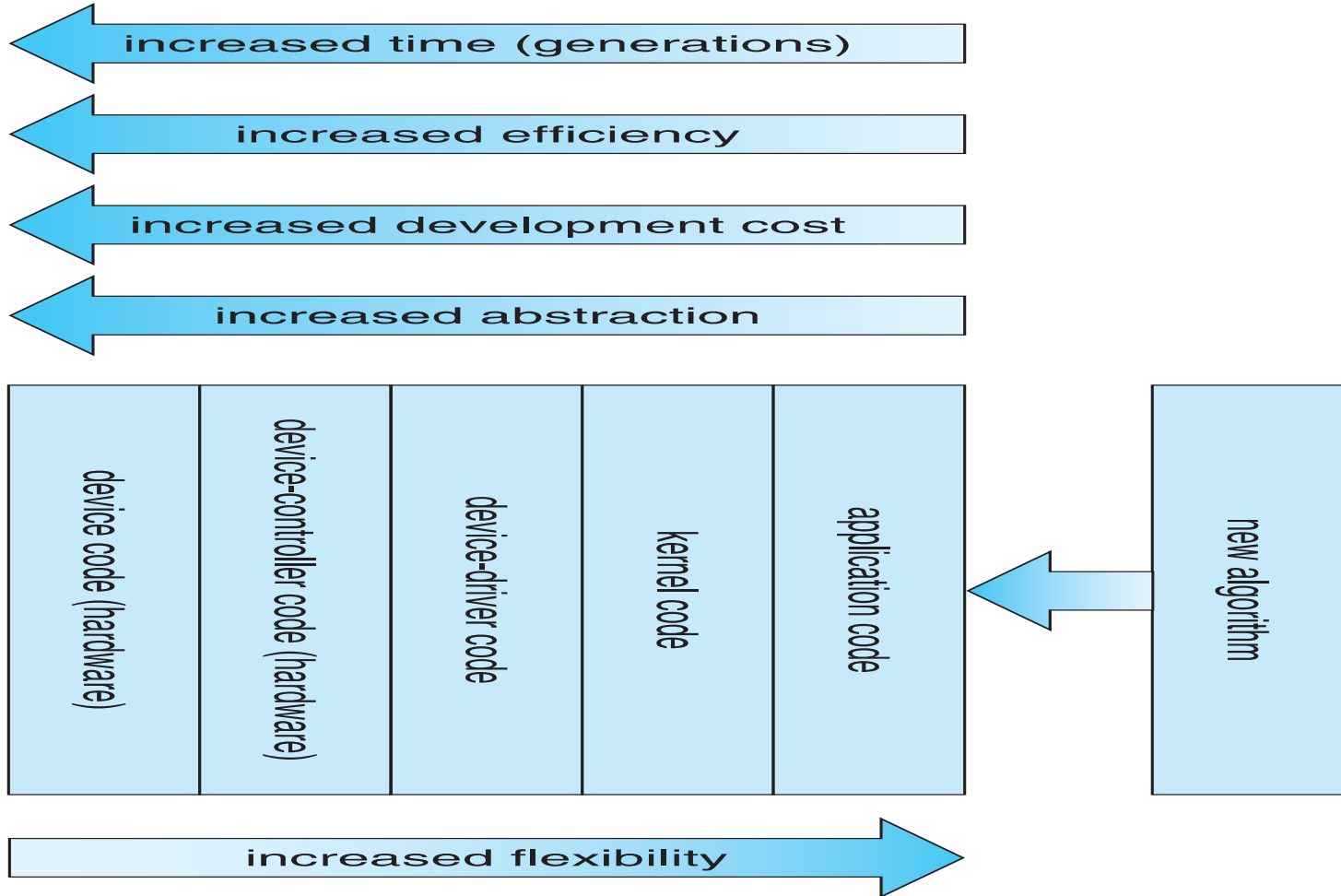
# Intercomputer Communications



# Improving 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

# Device-Functionality Progression





# Design Consideration: Access Right

- A design consideration
  - What kind of access right should we give to device drivers?
  - Unrestricted
    - Kernel mode
    - Relatively easier to design, can affect the others
  - Restricted
    - User mode
    - More difficult to design, isolated from the others

# Design Consideration: Load Device Drivers

- Relink the kernel with the new driver
  - Require reboot
- Add to the kernel an entry indicating a new driver is needed
  - Load the driver during reboot
- Install and run the device driver on the fly
  - Hot-pluggable

# Questions?

- System performance and I/O
- Device-function progression
- Access right