COP 4225 Advanced Unix Programming

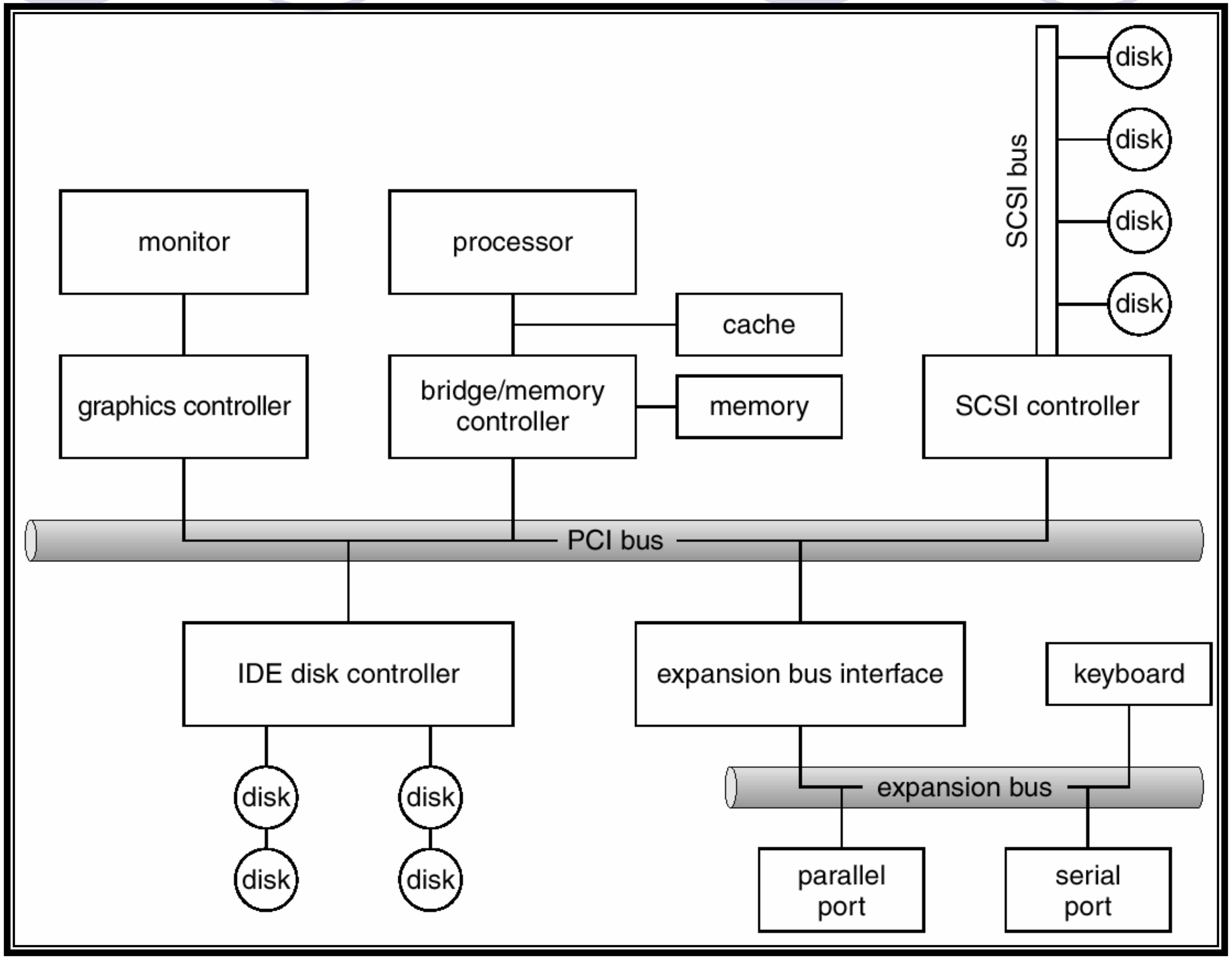
I/O Systems

Chi Zhang
czhang@cs.fiu.edu

I/O Hardware

- The kernel is structured to use device-driver modules.
- Common concepts
  - Port (for one device)
  - Bus (shared direct access)
  - Controller (host adapter) accepts commands from the processor through buses
    - The controller has one or more registers for data and control signals.

A Typical PC Bus Structure



I/O Hardware

- Expansion bus connects relatively slow devices
- Devices have addresses, used by
  - Direct I/O instructions (in, out)
    - Slower
    - Space limited
  - Memory-mapped I/O (mov, add, or, …)
    - Faster
    - Prone to software faults
- An I/O port typically consists of four registers
  - *status, control, data-in, data-out*

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

- Producer-consumer handshake
  - `command-ready` bit in the command register
  - `busy` bit in the status register
- Busy-wait cycle to wait for I/O from device
  - The processor polls the `busy` bit until it becomes clear
  - The processor sets the `write` bit in the command register and writes a byte into the `data-out` register before setting the `command-ready` bit
- Wastes CPU time

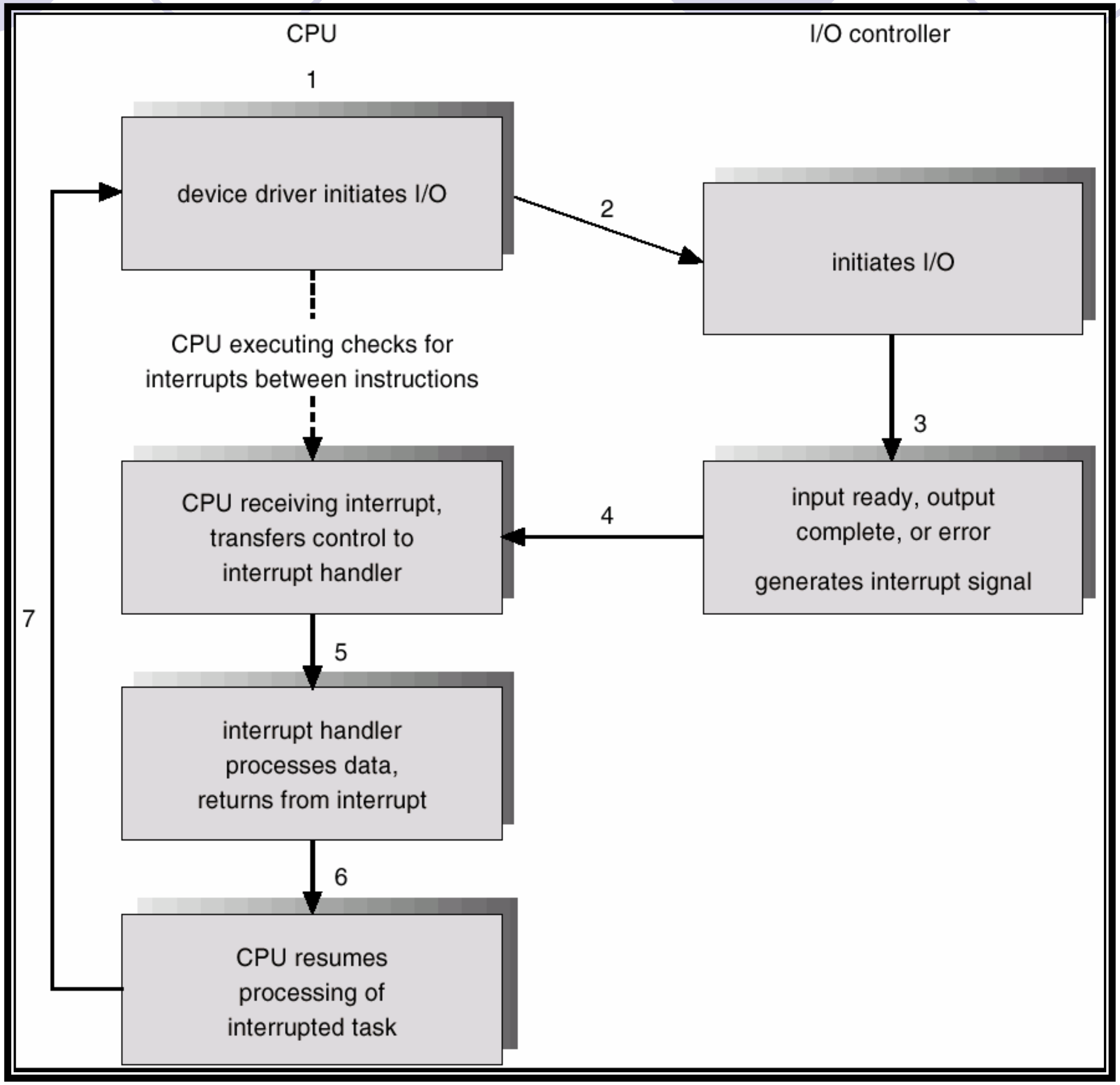
Interrupts

- CPU Interrupt request line triggered by I/O device
- Interrupt vector to dispatch interrupt to correct handler
  - Registered at boot time.
  - Based on priority (Some unmaskable)
- CPU saves a small amount of state, and jumps to the interrupt handler
- Interrupt handler processes interrupts
- Interrupt handler then return to the execution state prior to the interrupt.

Interrupts

- Interrupt handler
  - Transfer data from the controller to memory (if not DMA)
  - Wake up the process waiting for the I/O completion
- Interrupt mechanism also used for exceptions
  - Page Fault in virtual memory paging
- Interrupt mechanism also used for Systems Calls
  - Trap
  - Switch to kernel mode

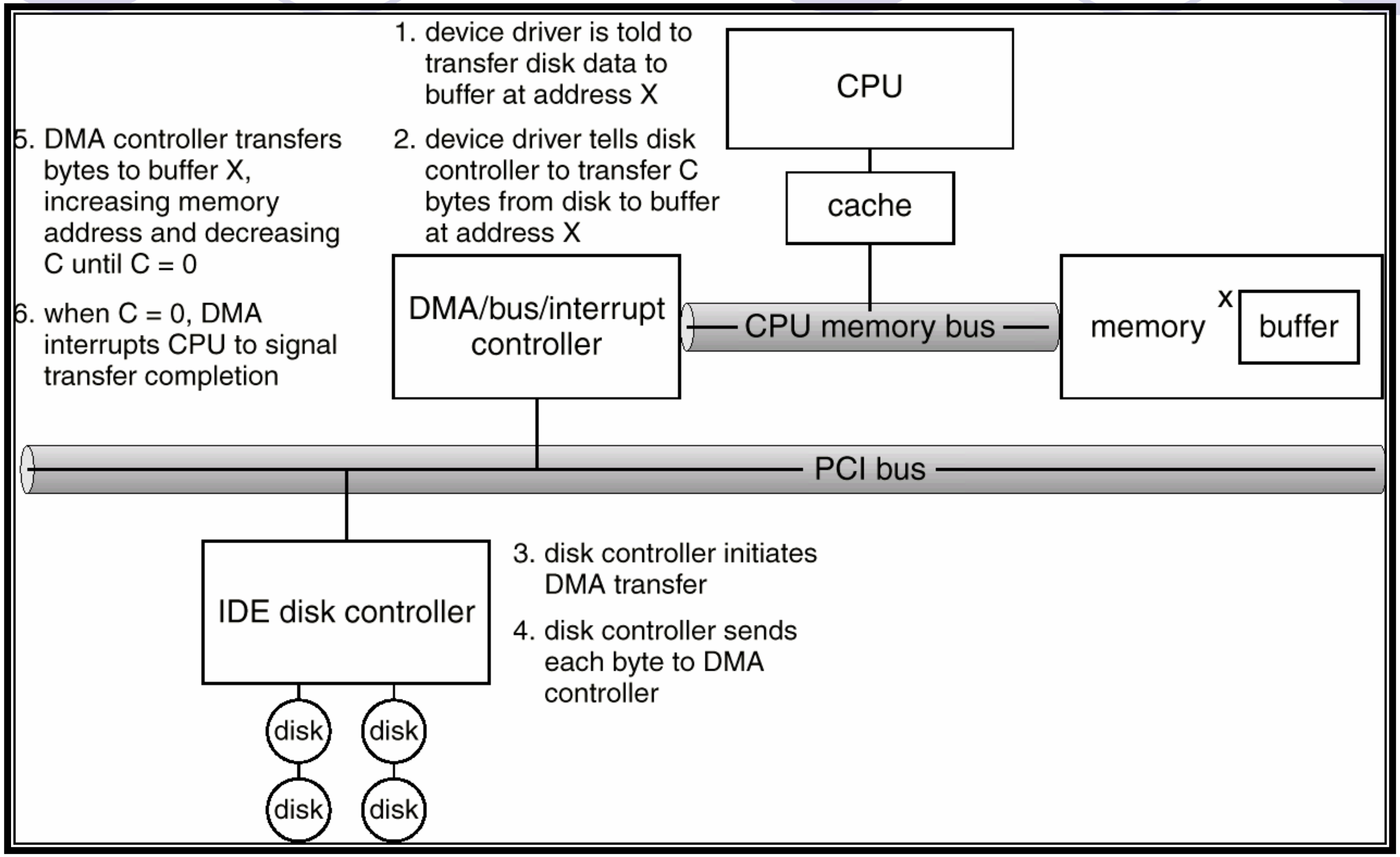
Interrupt-Driven I/O Cycle



Direct Memory Access

- Used to avoid programmed I/O (PIO) for large data movement
  - Bypasses CPU to transfer data directly between I/O device and memory
- Requires DMA controller
  - DMA-request from the device controller to the DMA controller
  - DMA-ack from the DMA controller to the device controller.

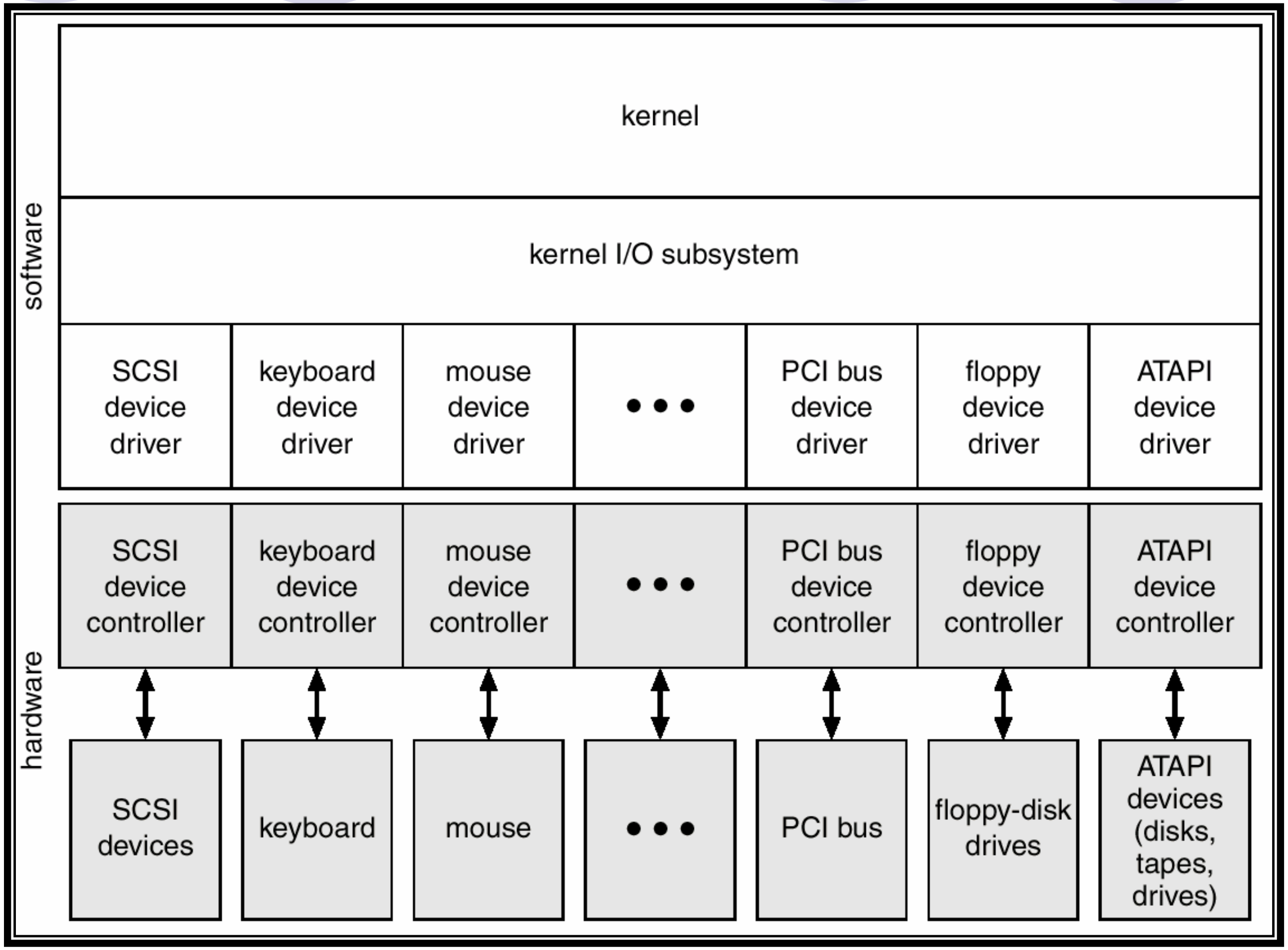
Six Step Process to Perform DMA Transfer



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
- Back-door to transparently pass arbitrary commands from an application to a device driver
  - Unix: ioctl
  - An integer argument to select one of the commands

A Kernel I/O Structure



I/O system calls encapsulate device behaviors in generic classes

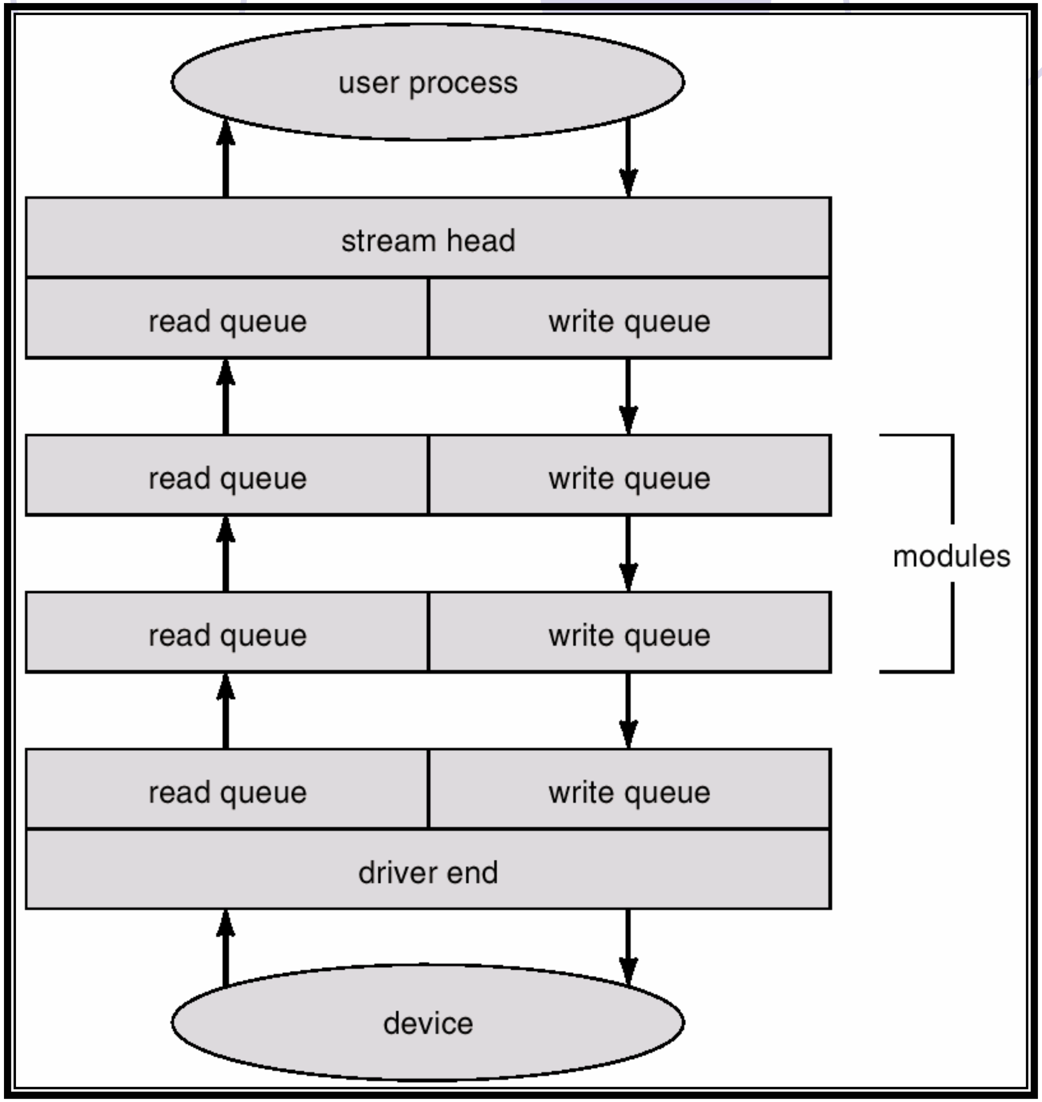
Block and Character Devices

- Block devices include disk drives
  - Commands include read, write, seek
  - Memory-mapped file access possible
- Character devices include keyboards, mice, serial ports
  - Commands include `get, put`
  - Produce data input at unpredictable time.

STREAMS

- **STREAM** – a full-duplex communication channel between a user-level process and a device
  - Character devices only
- Message passing is used to communicate between queues (e.g. putmsg vs. write)
  - Message boundaries and control information between modules
- Modules providing processing functionality can be pushed into Stream by ioctl().
  - Modular and incremental development

The STREAMS Structure



Clocks and Timers

- Provide current time, elapsed time, timer
- If programmable interval time used for timings, periodic interrupts
  - Virtual clocks
- `ioctl` (on UNIX) covers odd aspects of I/O such as clocks and timers

Blocking and Nonblocking I/O

- Blocking - process suspended until I/O completed
  - Easy to use and understand
  - Insufficient for some needs
  - Efficiencies can be improved via multi-threading
- Nonblocking - I/O call returns as much as available
  - User interface, data copy (buffered I/O)
- Asynchronous - process runs while I/O executes
  - Difficult to use
  - I/O subsystem signals process when I/O completed

Kernel I/O Subsystem

- Scheduling
  - Some I/O request ordering via per-device queue
  - Minimize disk arm seeks and improve fairness
- 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”
    - Application might change the buffer after system calls

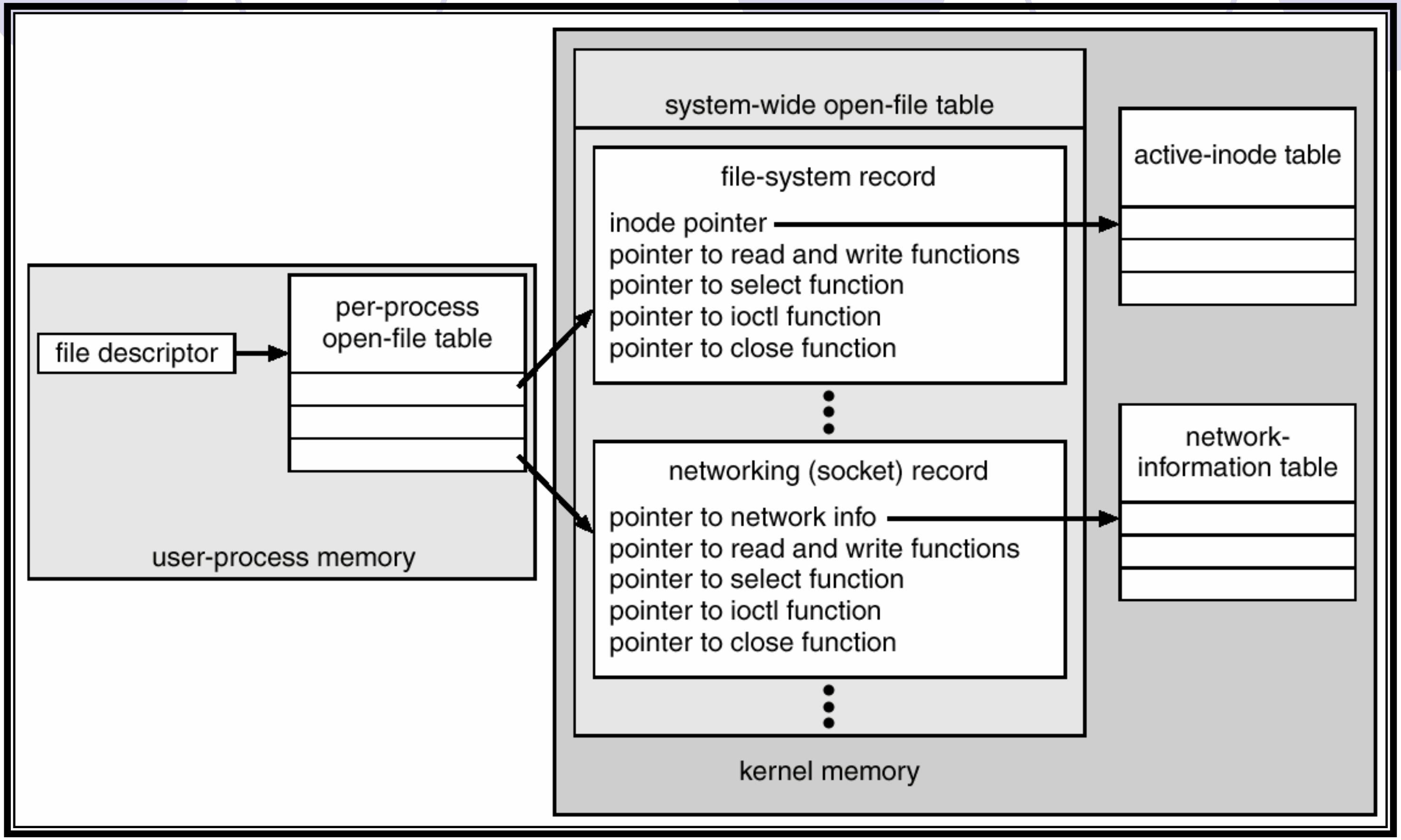
Kernel I/O Subsystem

- Spooling - hold output for a device
  - If device can serve only one request at a time
  - Each application’s output is spooled to a separate disk file
  - E.g. a daemon process for printing
- Error handling
  - Most return one bit information about the status (succes / failure)
  - an error number or code indicating the error nature (Unix: errno)

Kernel Data Structures

- 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

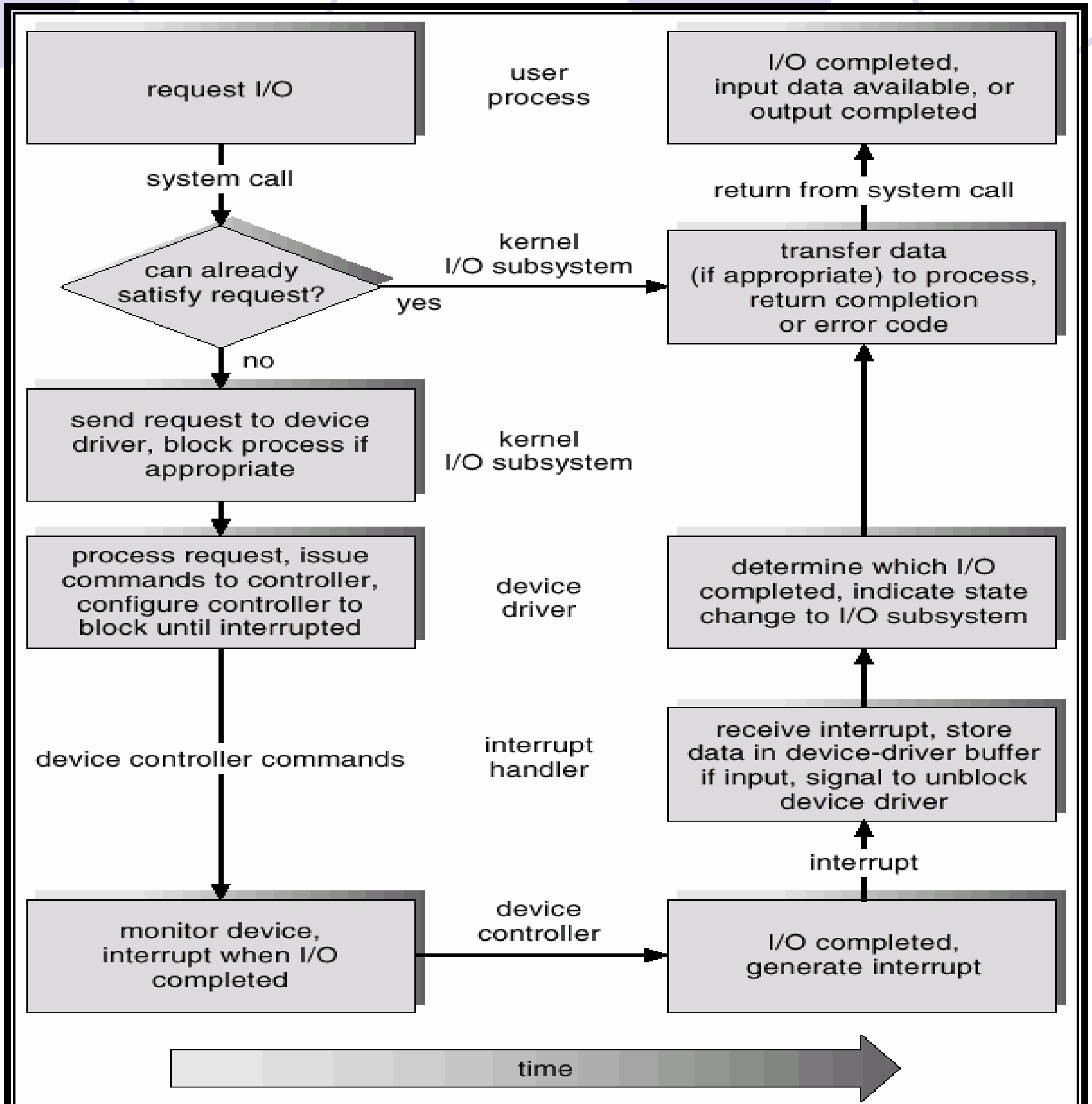
UNIX I/O Kernel Structure



I/O Requests to Hardware Operations

- Consider reading a file from disk for a process:
  - Determine device holding file
    - Longest match prefix in the mount table
    - <major, minor> device number
    - Minor passed to the driver selected by major.
  - Translate name to device representation
  - Physically read data from disk into buffer
  - Make data available to requesting process
  - Return control to process

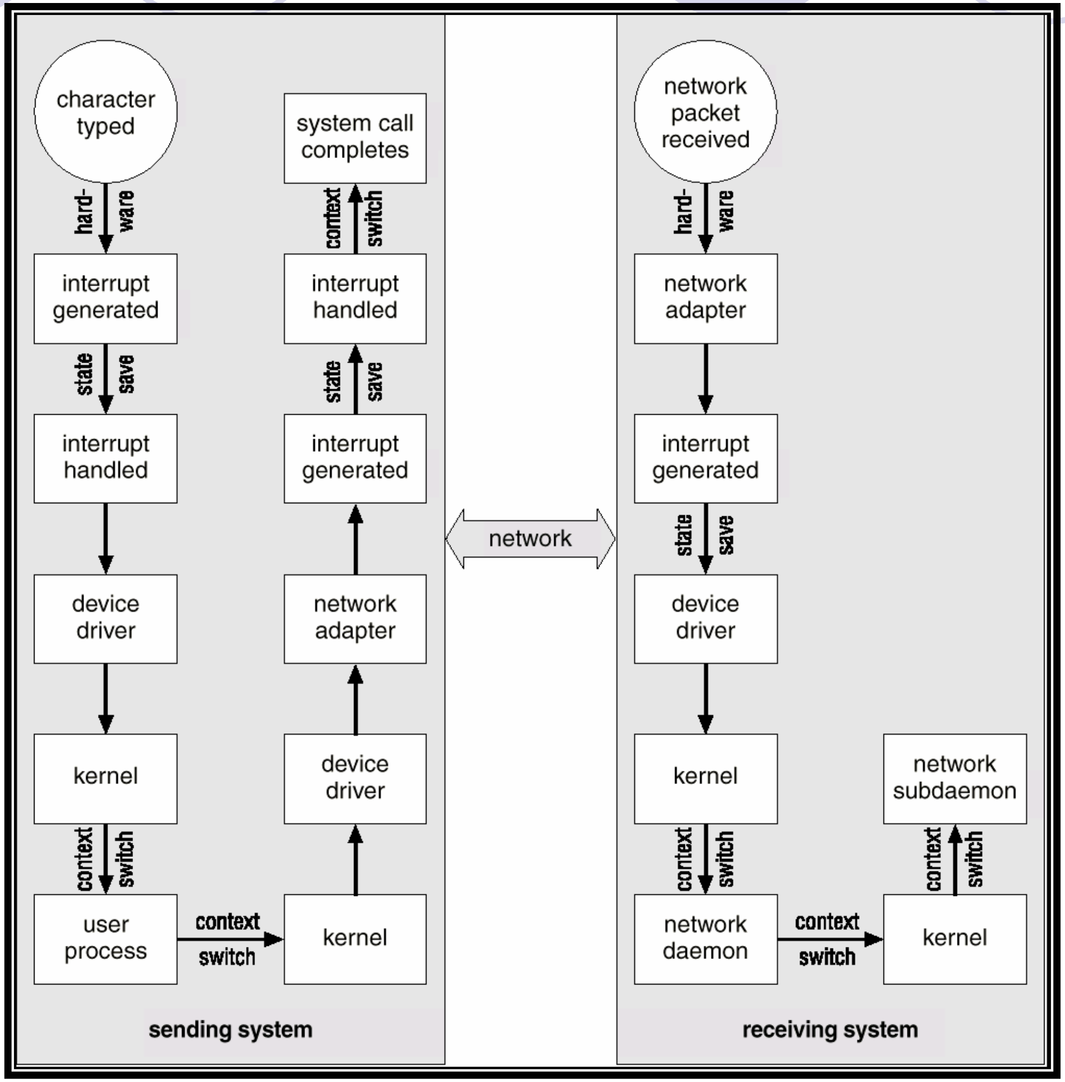
Life Cycle of An I/O Request



Performance

- I/O a major factor in system performance:
  - Demands CPU to execute device driver, kernel I/O code
  - Context switches due to interrupts
    - Sometimes Programmed I/O is more efficient, if the number of busy-waiting cycles is not excessive.
  - 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 and offload channels
- Balance CPU, memory, bus, and I/O performance for highest throughput

Device-Functionality Progression