

---

CS 422/522 Design & Implementation  
of Operating Systems

## Lecture 14: I/O Devices & Drivers

Zhong Shao  
Dept. of Computer Science  
Yale University

1

---

### The big picture

- ◆ Previous lectures
  - Management of CPU & concurrency
  - Management of main memory & virtual memory
- ◆ Future lectures --- "**Management of I/O devices**"

2

## Concurrency: a summary

---

- ◆ Thread vs. process
- ◆ How to implement threads/processes ?
  - \* thread/process state transition diagram
  - \* thread/process scheduler
  - \* context switch
  - \* thread/process creation / finish
- ◆ How to write concurrent programs ?
  - \* how to eliminate race condition ? how to synchronize?
  - \* locks, condition variables, monitors, semaphore, message passing
- ◆ Multithreading model (kernel vs. user threads)
- ◆ How to deal with deadlocks
- ◆ Effective CPU scheduling (local + global)

3

## Virtual memory: a summary

---

- ◆ Goal: multiprogramming with protection + illusion of infinite memory
- ◆ Approaches
  - HW-based solution for protection
    - \* dual mode operation + address space
  - address translation: virtual address -> physical address
    - \* segmentation + paging + multilevel paging
  - making address translation faster? use TLB
  - demand paged virtual memory
  - techniques for dealing with thrashing
- ◆ Other topics
  - kernel memory allocator (similar to malloc-free packages)
  - virtual memory-based hack (exploiting page fault)

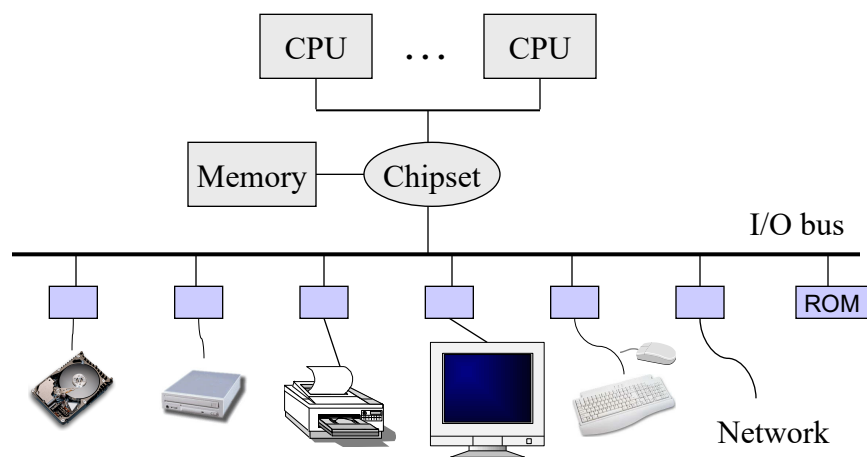
4

## The big picture

- ◆ Previous lectures
  - Management of CPU & concurrency
  - Management of main memory & virtual memory
- ◆ Future lectures --- "Management of I/O devices"
  - This week: **I/O devices & device drivers**
  - This week: storage devices
  - Next week: file systems
    - \* File system structure
    - \* Naming and directories
    - \* Efficiency and performance
    - \* Reliability and protection

5

## Raw hardware revisited



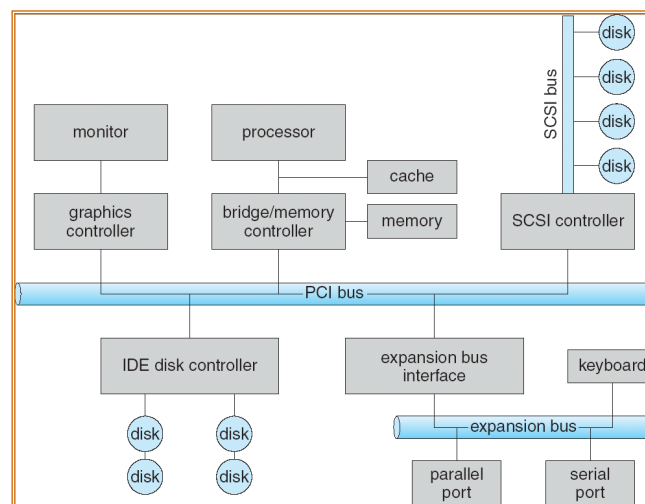
6

## I/O hardware

- ◆ A computer = CPU(s) + Memory + I/O devices
- ◆ Common concepts
  - **Port** (a connection point between a machine and a device)
  - **Bus** (one or more devices share a common set of wires)
  - **Controller** (has private processor, microcode, memory)
- ◆ The processor gives commands and data to a controller to accomplish an I/O transfer
  - The controller has a few registers for data & control signals
    - \* typical registers: status, control, data-in, data-out
  - Special I/O instructions (w. port addr) or memory mapped I/O

7

## A typical PC bus structure



8

## 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)

9

## CPU - I/O interaction: polling

- ◆ the status register indicates the state of the device
  - a `command-ready` bit and a `busy` bit
- ◆ Procedure for writing out a byte:
  - the host reads the "busy" bit until it becomes clear
  - the host issues "write" command, puts the byte in "data-out"
  - The host sets the "command-ready" bit
  - The controller sees "command-ready", sets the "busy" bit
  - The controller executes the "write", does I/O
  - The controller clears the "command-ready" and "busy" bits
- ◆ Inefficient: busy-wait cycle to wait for device I/O

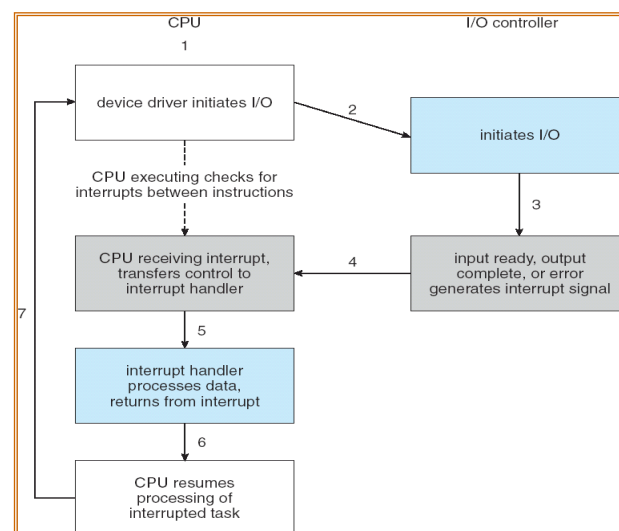
10

## CPU - I/O interaction: interrupts

- ◆ CPU interrupt request line triggered by I/O device
- ◆ Interrupt handler receives interrupts
- ◆ Maskable to ignore or delay some interrupts
- ◆ Interrupt vector to dispatch interrupt to correct handler
  - Based on priority
  - Some unmaskable
- ◆ Interrupt mechanism also used for exceptions

11

## Interrupt-driven I/O cycle



12

## 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

13

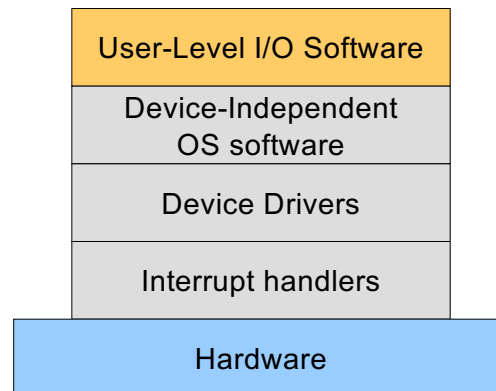
## Interrupt handling revisited/refined

- ◆ Save more context
- ◆ Mask interrupts if needed
- ◆ Set up a context for interrupt service
- ◆ Set up a stack for interrupt service
- ◆ Acknowledge the interrupt controller, enable it if needed
- ◆ Save entire context to PCB
- ◆ Run the interrupt service
- ◆ Unmask interrupts if needed
- ◆ Possibly change the priority of the process
- ◆ Run the scheduler

14

## I/O software stack

---



15

## Direct Memory Access (DMA)

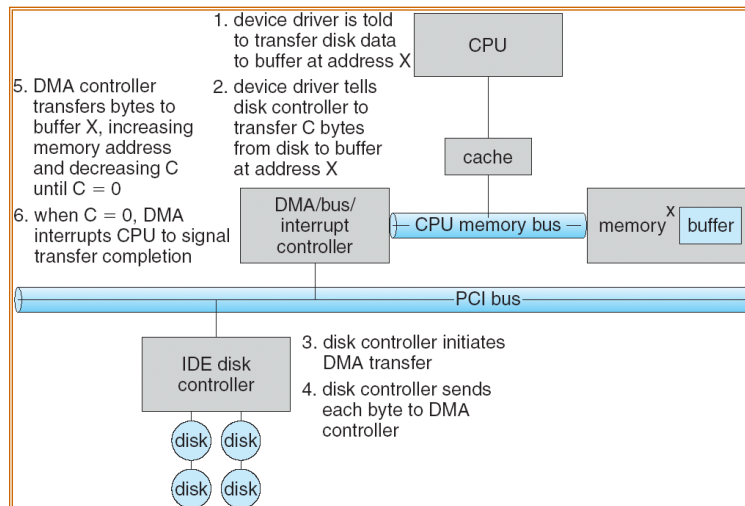
---

- ◆ Used to avoid programmed I/O for large data movement
- ◆ Requires DMA controller
- ◆ Bypasses CPU to transfer data directly between I/O device and memory

16



## Performing DMA transfer



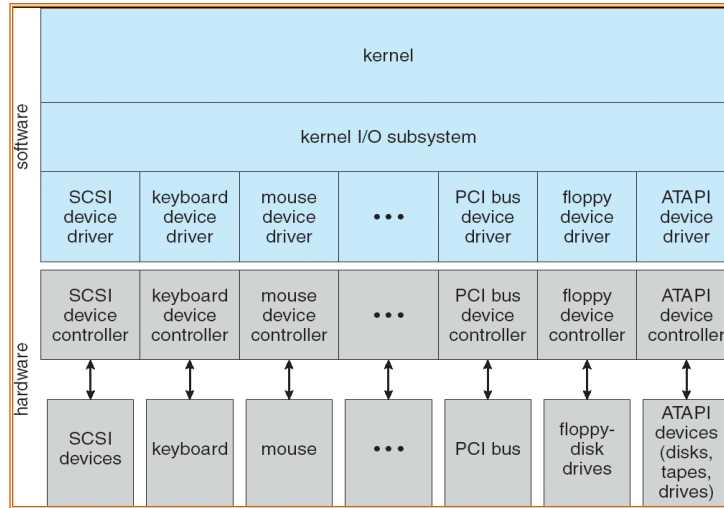
17

## 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
- ◆ Devices vary in many dimensions
  - Character-stream or block
  - Sequential or random-access
  - Sharable or dedicated
  - Speed of operation
  - read-write, read only, or write only

18

## A kernel I/O structure



19

## Characteristics of I/O devices

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

20

## Block and character devices

---

- ◆ Block devices include disk drives
  - Commands include read, write, seek
  - Raw I/O or file-system access
  - Memory-mapped file access possible
  
- ◆ Character devices include keyboards, mice, serial ports
  - Commands include `get`, `put`
  - Libraries layered on top allow line editing

21

## Network devices

---

- ◆ Different enough from the block & character devices to have own interface
  
- ◆ Unix and Windows/NT include socket interface
  - Separates network protocol from network operation
  
- ◆ Approaches vary widely (pipes, FIFOs, streams, queues, mailboxes)

22

## Clocks and timers

---

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

23

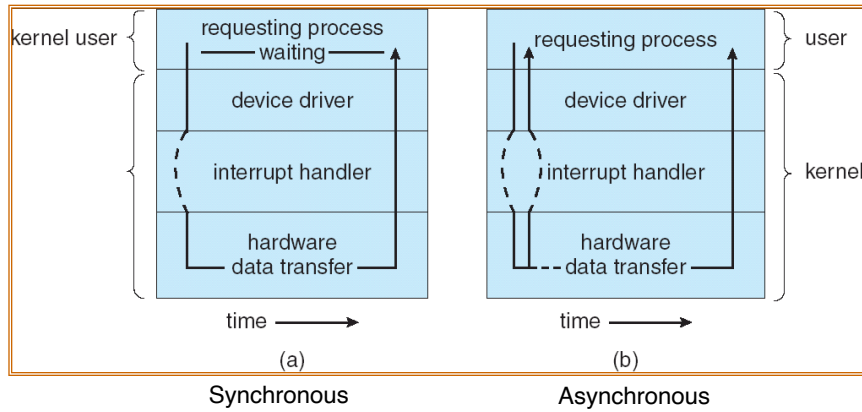
## Blocking and nonblocking I/O

---

- ◆ **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
- ◆ **Asynchronous** - process runs while I/O executes
  - Difficult to use
  - I/O subsystem signals process when I/O completed

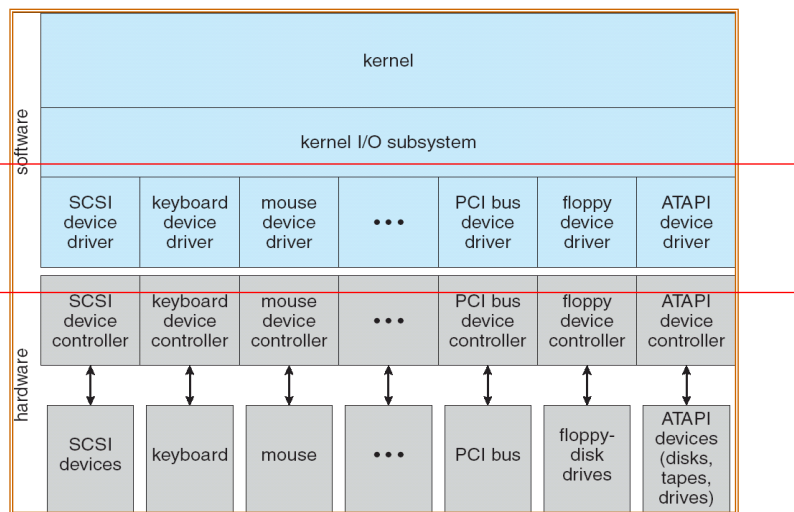
24

## Two I/O methods



25

## Next: device drivers



26

## Device driver design issues

---

- ◆ Operating system and driver communication
  - Commands and data between OS and device drivers
- ◆ Driver and hardware communication
  - Commands and data between driver and hardware
- ◆ Driver operations
  - Initialize devices
  - Interpreting commands from OS
  - Schedule multiple outstanding requests
  - Manage data transfers
  - Accept and process interrupts
  - Maintain the integrity of driver and kernel data structures

27

## Device driver interface

---

- ◆ Open( deviceNumber )
  - Initialization and allocate resources (buffers)
- ◆ Close( deviceNumber )
  - Cleanup, deallocate, and possibly turnoff
- ◆ Device driver types
  - Block: fixed sized block data transfer
  - Character: variable sized data transfer
  - Terminal: character driver with terminal control
  - Network: streams for networking

28

## Block device interface

---

- ◆ `read( deviceNumber, deviceAddr, bufferAddr )`
  - transfer a block of data from “deviceAddr” to “bufferAddr”
- ◆ `write( deviceNumber, deviceAddr, bufferAddr )`
  - transfer a block of data from “bufferAddr” to “deviceAddr”
- ◆ `seek( deviceNumber, deviceAddress )`
  - move the head to the correct position
  - usually not necessary

29

## Character device interface

---

- ◆ `read( deviceNumber, bufferAddr, size )`
  - reads “size” bytes from a byte stream device to “bufferAddr”
- ◆ `write( deviceNumber, bufferAddr, size )`
  - write “size” bytes from “bufferSize” to a byte stream device

30

## Unix device driver interface entry points

---

- ◆ `init()`: initialize hardware
- ◆ `start()`: boot time initialization (require system services)
- ◆ `open(dev, flag, id)`: initialization for read or write
- ◆ `close(dev, flag, id)`: release resources after read and write
- ◆ `halt()`: call before the system is shutdown
- ◆ `intr(vector)`: called by the kernel on a hardware interrupt
- ◆ read/write calls: data transfer
- ◆ `poll(pri)`: called by the kernel 25 to 100 times a second
- ◆ `ioctl(dev, cmd, arg, mode)`: special request processing

31

## Device driver: other design issues

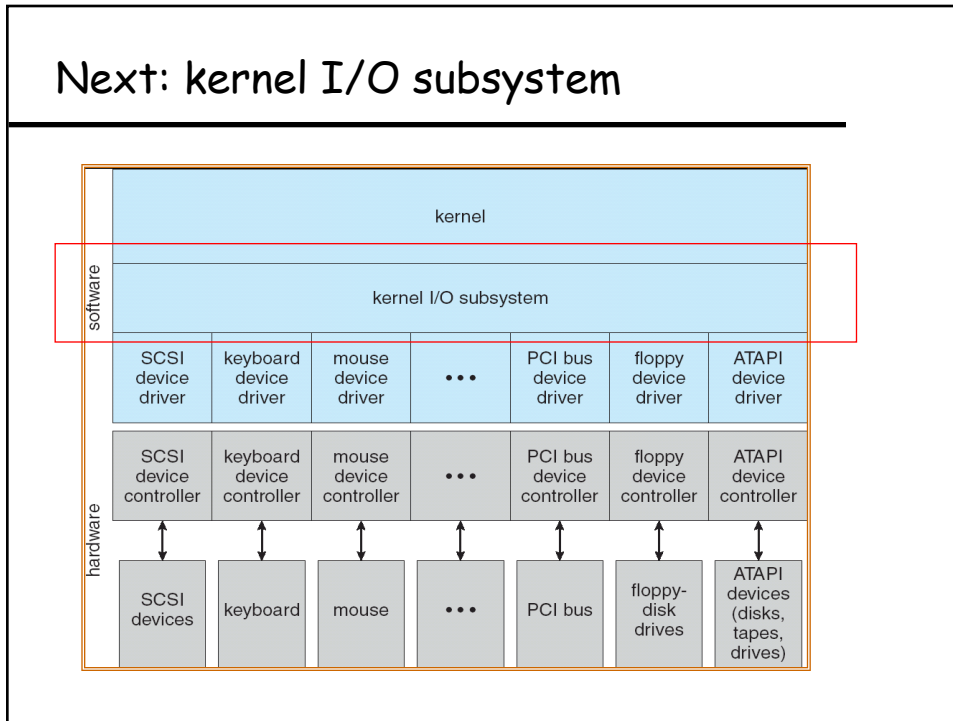
---

- ◆ Build device drivers
  - Statically
  - Dynamically
- ◆ How to down load device driver dynamically?
  - Load drivers into kernel memory
  - Install entry points and maintain related data structures
  - Initialize the device drivers

32



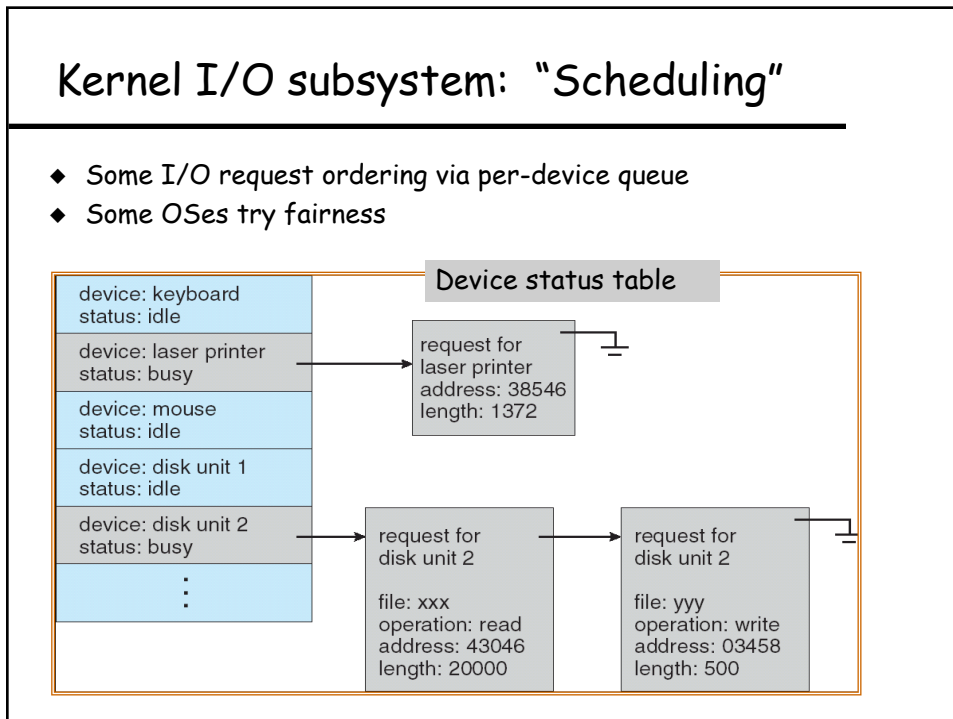
## Next: kernel I/O subsystem



33

## Kernel I/O subsystem: "Scheduling"

- ◆ Some I/O request ordering via per-device queue
- ◆ Some OSes try fairness



34

## Kernel I/O subsystem (cont'd)

---

- ◆ Buffering - store data in memory while transferring between devices
  - To cope with device speed mismatch
  - To cope with device transfer size mismatch (e.g., packets in networking)
  - To maintain "copy semantics"
    - \* Copy data from user buffer to kernel buffer
- ◆ How to deal with address translation?
  - I/O devices see physical memory, but programs use virtual memory
- ◆ Caching - fast memory holding copy of data
  - Always just a copy
  - Key to performance
- ◆ Spooling - hold output for a device
  - If a device can serve only one request at a time, i.e., printing

35

## Error handling

---

- ◆ OS can recover from disk read, device unavailable, transient write failures
- ◆ Most return an error number or code when I/O request fails
- ◆ System error logs hold problem reports

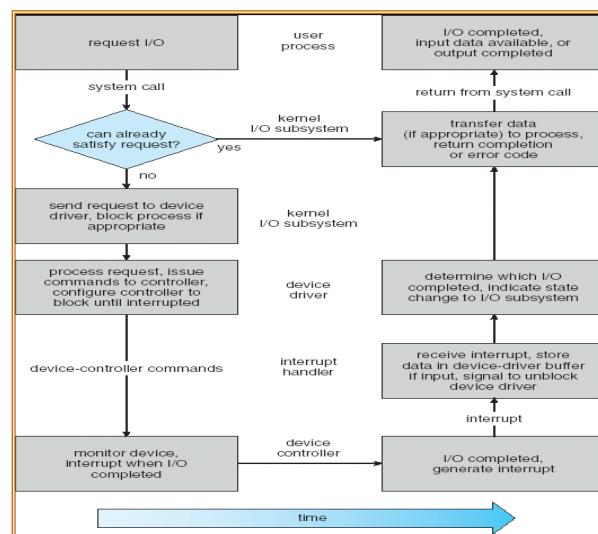
36

## 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

37

## Life cycle of an I/O request



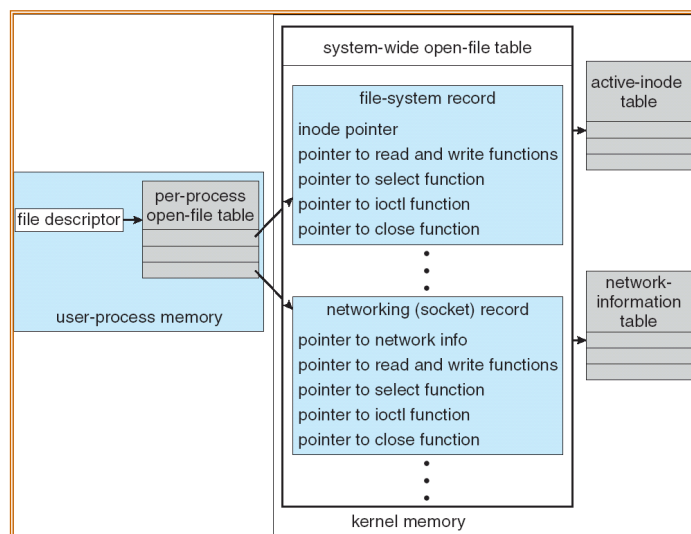
38

## 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

39

## UNIX I/O kernel structure



40

## I/O requests to hardware operations

---

- ◆ 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

41

## Another example: blocked read w. DMA

---

- ◆ A process issues a read call which executes a system call
- ◆ System call code checks for correctness and cache
- ◆ If it needs to perform I/O, it will issues a device driver call
- ◆ Device driver allocates a buffer for read and schedules I/O
- ◆ Controller performs DMA data transfer, blocks the process
- ◆ Device generates an interrupt on completion
- ◆ Interrupt handler stores any data and notifies completion
- ◆ Move data from kernel buffer to user buffer and wakeup blocked process
- ◆ User process continues

42