LUT University

LAND OF THE CURIOUS

JANUARY 17, 2023

OPERATING SYSTEMS AND SYSTEMS PROGRAMMING (CT30A3370) 6 CREDITS

Chapter 13: I/O Systems

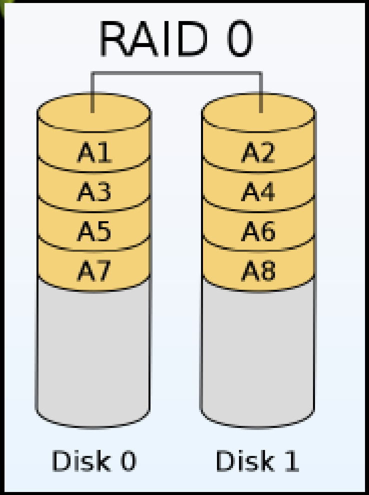
Venkata Marella

REVIEW OF RAID

- Raid 0

$$\text{Capacity} = n * \min(\text{disk sizes})$$
$$\text{MTTF}_{\text{group}} = \text{MTTF}_{\text{disk}} / n$$

MTTF is the average time that an item will function before it fails



Raid Level	Pros	Cons	Storage Efficiency	Minimum Number of disks
RAID-0	• Performance (great read and write performance) • Great capacity utilization (the best of any standard RAID configurations)	• No data redundancy • Poor MTTF	100% assuming the drives are the same size	2

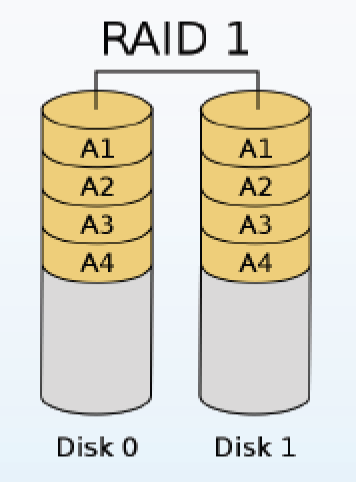
Silberschatz, Galvin Operating System Concepts – 7th Edition and Gagne ©2005

REVIEW OF RAID (CONT.)

- Raid 1

Capacity = min(disk sizes)

P(dual failure) = P(single drive)^2

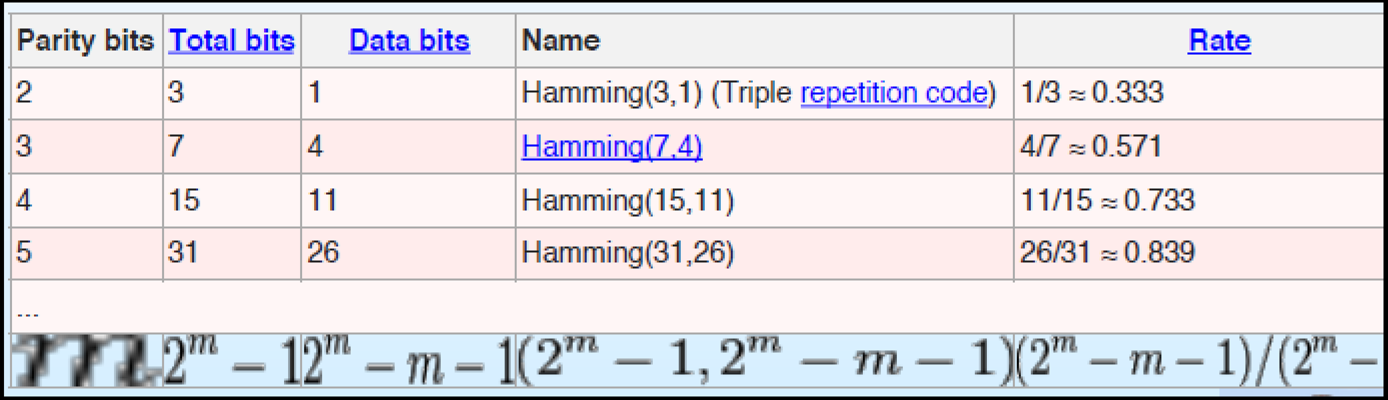


Raid Level	Pros	Cons	Storage Efficiency	Minimum Number of disks
RAID-1	• Great data redundancy/availability • Great MTTF	• Worst capacity utilization of single RAID levels • Good read performance, limited write performance	50% assuming two drives of the same size	2

Silberschatz, Galvin Operating System Concepts – 7th Edition and Gagne ©2005

REVIEW OF RAID (CONT.)

- Raid 2
  - Level 2 is the only RAID level of the ones defined by the original Berkeley document that is not used today, for a variety of reasons.
  - It is expensive and often requires many drives.
  - The controller required was complex, specialized and expensive.
  - The performance of RAID 2 is also rather substandard in transactional environments due to the bit-level striping.



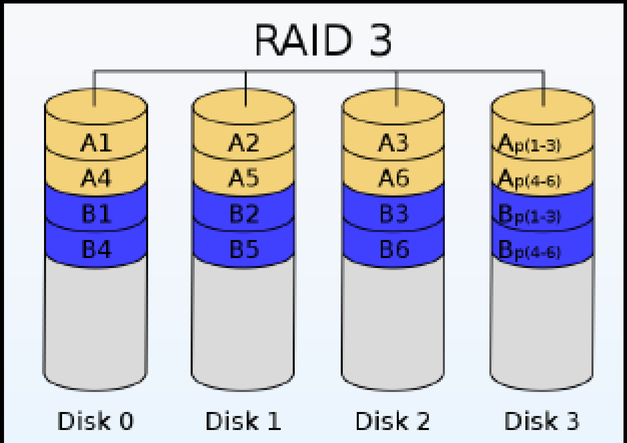
Parity bits	Total bits	Data bits	Name	Rate
2	3	1	Hamming(3,1) (Triple repetition code)	1/3 ≈ 0.333
3	7	4	Hamming(7,4)	4/7 ≈ 0.571
4	15	11	Hamming(15,11)	11/15 ≈ 0.733
5	31	26	Hamming(31,26)	26/31 ≈ 0.839
...				
[illegible]	$2^m - 1$	$2^m - m - 1$	$(2^m - 1, 2^m - m - 1)$	$(2^m - m - 1)/(2^m -$

Review of RAID (Cont.)

REVIEW OF RAID (CONT.)

- Raid 3

```
Capacity = min(disk sizes) * (n-1)
```



Raid Level	Pros	Cons	Storage Efficiency	Minimum Number of disks
RAID-3	• Good data redundancy/availability (can tolerate the lose of 1 drive) • Good read performance since all of the drives are read at the same time • Reasonable write performance but parity computations cause some reduction in performance • Can lose one drive without losing data	• Spindles have to be synchronized • Data access can be blocked because all drives are accessed at the same time for read or write	(n - 1) / n where n is the number of drives	3 (have to be identical)

HOW PARITY WORKS

- Using XOR

  1 xor 1 = 0
  1 xor 0 = 1
  0 xor 1 = 1
  0 xor 0 = 0

- Parity of bit streams

  10010011101110110001101...
  10000001000000010000000...
  00010010101110100001101...

- Using Parity to recover

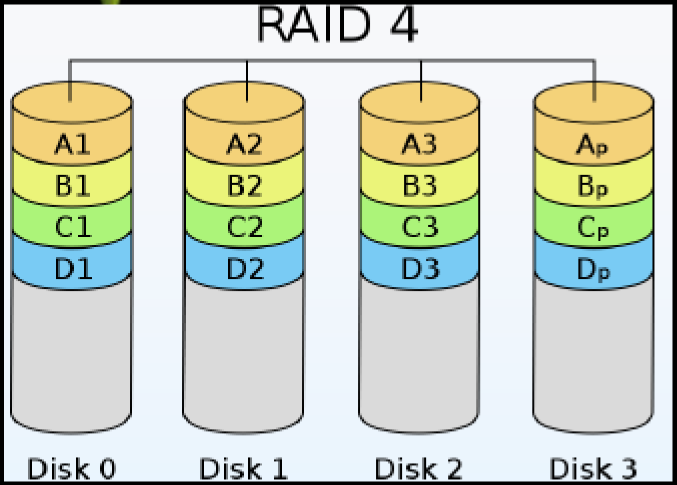
  10010011101110110001101...
  1000___________________...
  00010010101110100001101...

Silberschatz, Galvin Operating System Concepts – 7th Edition and Gagne ©2005

REVIEW OF RAID (CONT.)

- Raid 4

```
Capacity = min(disk sizes) * (n-1)
```



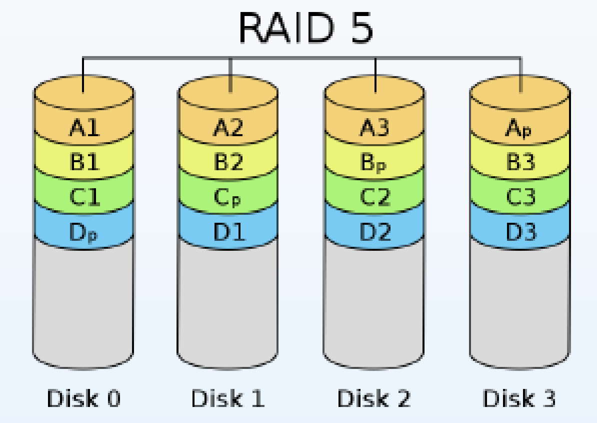
Raid Level	Pros	Cons	Storage Efficiency	Minimum Number of disks
RAID-4	• Good data redundancy/availability (can tolerate the lose of 1 drive) • Good read performance since all of the drives are read at the same time • Can lose one drive without losing data	• Single parity disk (causes bottleneck) • Write performance is not that good because of the bottleneck of the parity drive	`(n - 1) / n` where `n` is the number of drives	3 (have to be identical)

Silberschatz, Galvin Operating System Concepts – 7th Edition and Gagne ©2005

REVIEW OF RAID (CONT.)

- Raid 5

```
Capacity = min(disk sizes) * (n-1)
```

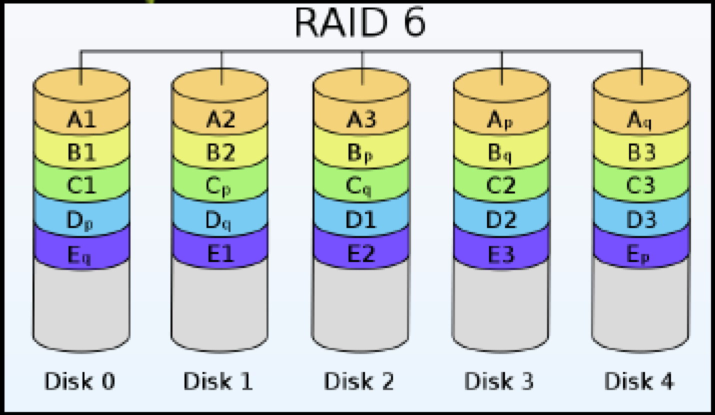


Raid Level	Pros	Cons	Storage Efficiency	Minimum Number of disks
RAID-5	• Good data redundancy/availability (can tolerate the lose of 1 drive) • Very good read performance since all of the drives can be read at the same time • Write performance is adequate (better than RAID-4) • Can lose one drive without losing data	• Write performance is adequate (better than RAID-4) • Write performance for small I/O is not good at all	(n - 1) / n where n is the number of drives	3 (have to be identical)

REVIEW OF RAID (CONT.)

- Raid 6

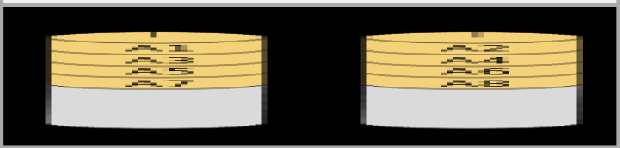
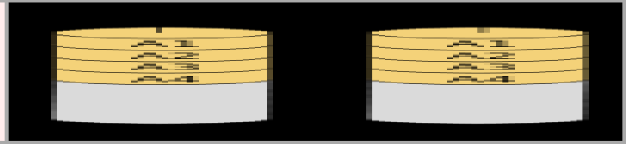
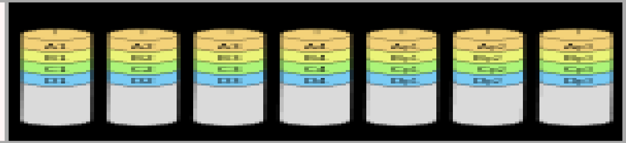
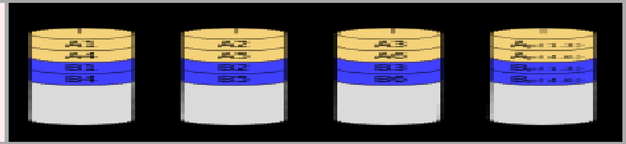
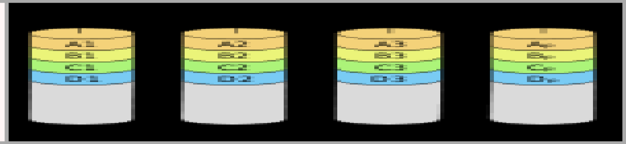
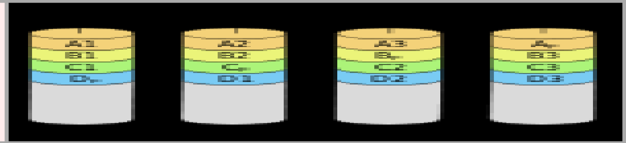
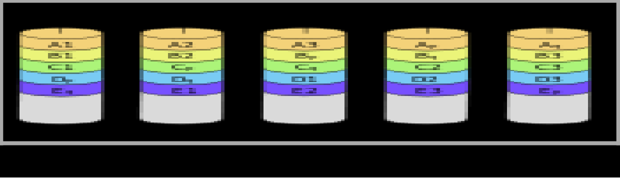
```
Capacity = min(disk sizes) * (n-2)
```



Raid Level	Pros	Cons	Storage Efficiency	Minimum Number of disks
RAID-6	• Excellent data redundancy/availability (can tolerate the lose of 2 drives) • Very good read performance since all of the drives can be read at the same time • Can lose two drives without losing data	• Write performance is not that good – worse than RAID-5 • Write performance for small I/O is not good at all • more computational horsepower is required for parity computations	`(n - 2) / n` where `n` is the number of drives	4 (have to be identical)

Silberschatz, Galvin Operating System Concepts – 7th Edition and Gagne ©2005

REVIEW OF RAID (CONT.)

Level	Description	Figure
RAID 0	Block-level striping without parity or mirroring	
RAID 1	Mirroring without parity or striping	
RAID 2	Bit-level striping with dedicated Hamming-code parity	
RAID 3	Byte-level striping with dedicated parity	
RAID 4	Block-level striping with dedicated parity	
RAID 5	Block-level striping with distributed parity	
RAID 6	Block-level striping with double distributed parity	

CHAPTER 13: I/O SYSTEMS

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

OBJECTIVES

- Explore the structure of an operating system's I/O subsystem
- Discuss the principles of I/O hardware and its complexity
- Provide details of the performance aspects of I/O hardware and software

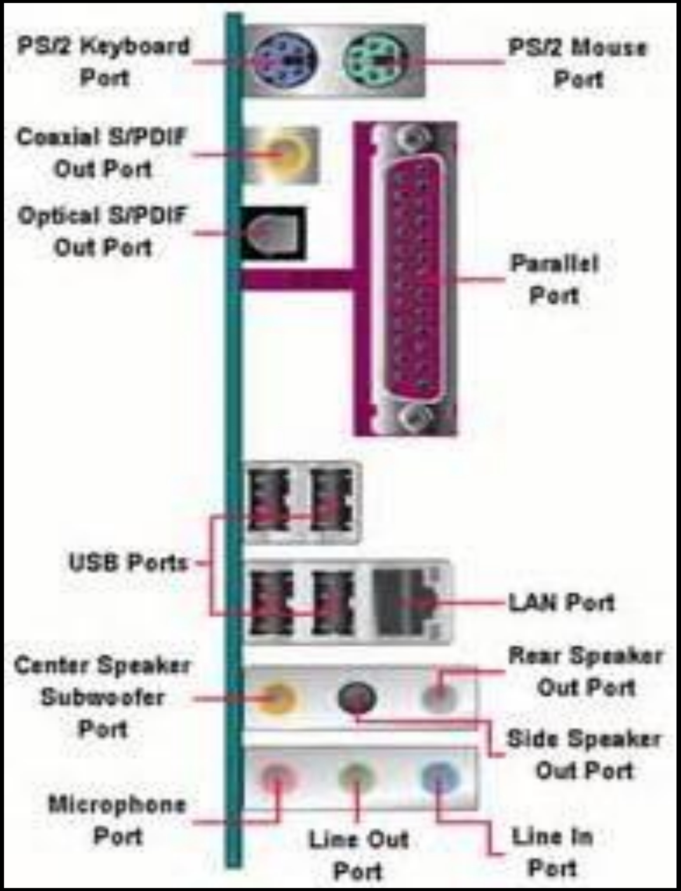
I/O HARDWARE

- Incredible variety of I/O devices
- Common concepts
  - **Port**
  - **Bus** (**daisy chain** or shared direct access)
  - **Controller** (**host adapter**)
- I/O instructions control devices
- Devices have (port) addresses, used by
  - Special I/O instructions
  - **Memory-mapped** I/O

Some systems use both.

Device control registers mapped into processor address space.

I/O PORT

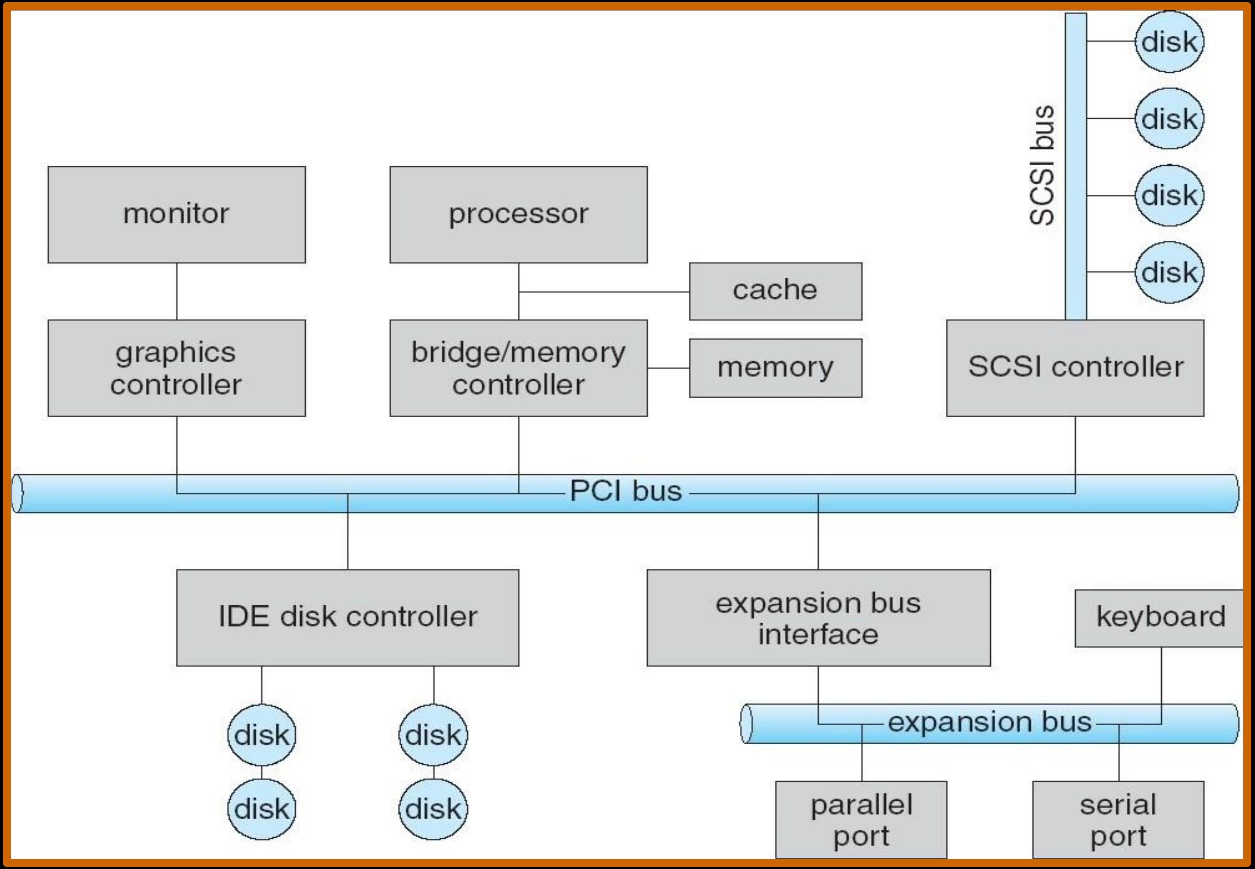


Silberschatz, Galvin Operating System Concepts – 7th Edition and Gagne ©2005

NEW I/O DEVICES

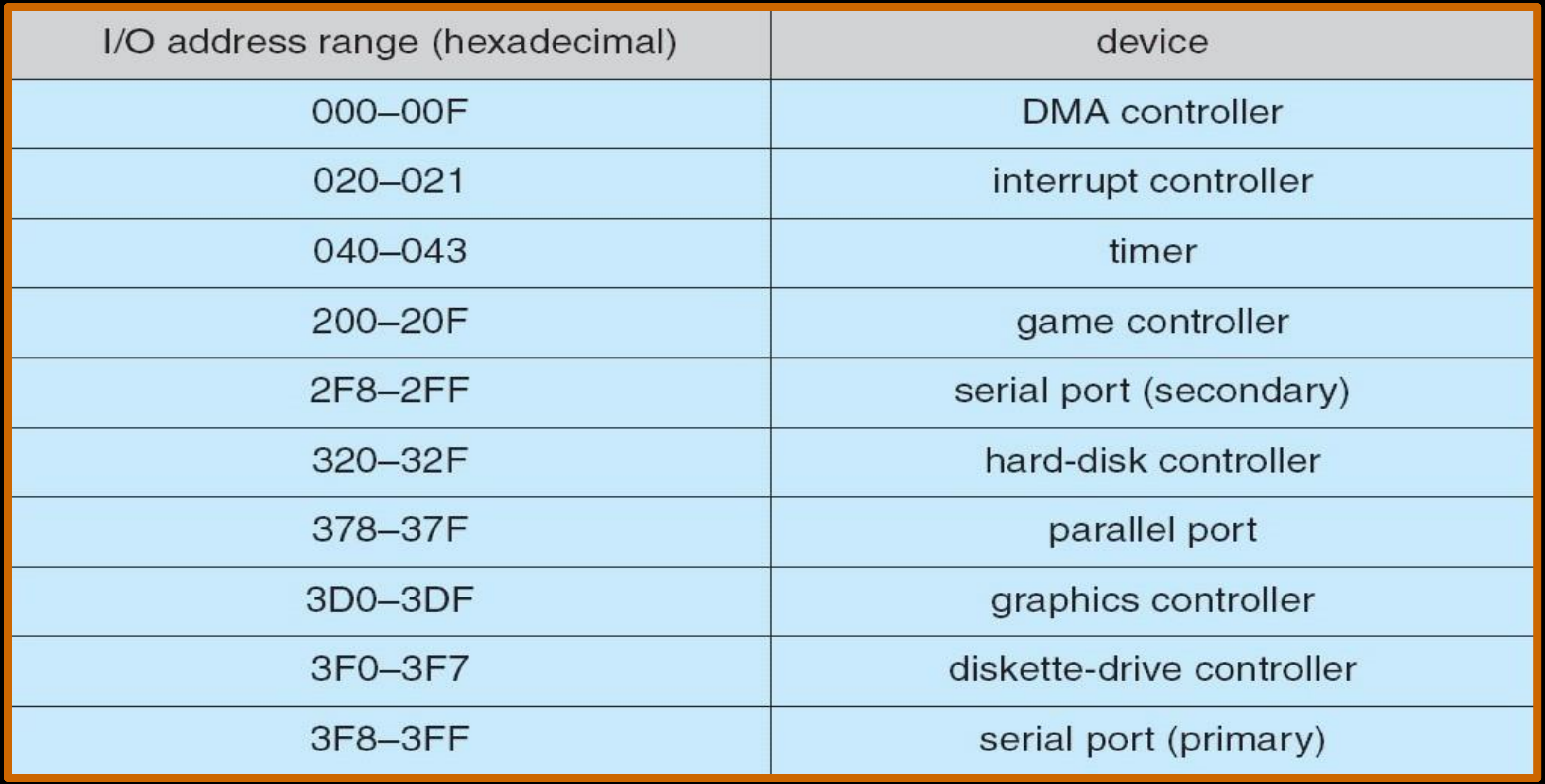
Silberschatz, Galvin Operating System Concepts – 7th Edition and Gagne ©2005

A TYPICAL PC BUS STRUCTURE



Silberschatz, Galvin Operating System Concepts – 7th Edition and Gagne ©2005

DEVICE I/O PORT ADDRESSES 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)

Silberschatz, Galvin Operating System Concepts – 7th Edition and Gagne ©2005

I/O PORT REGISTERS

CPU = host

- **Data-in**: read by the host to get input
- **Data-out**: written by the host to send output
- **Status**: device status read by the host
- **Control**: written by the host to start a command or change the mode of a device

POLLING

- (Refer to textbook p.499)Determines state of device
  - command-ready
  - busy
  - Error
- **Busy-wait** cycle to wait for I/O from device

Repeatedly reading the **status** register until the busy bit becomes clear. **Inefficient!!**

Silberschatz, Galvin Operating System Concepts – 7th Edition and Gagne ©2005

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 **nonmaskable**
- Interrupt mechanism also used for exceptions

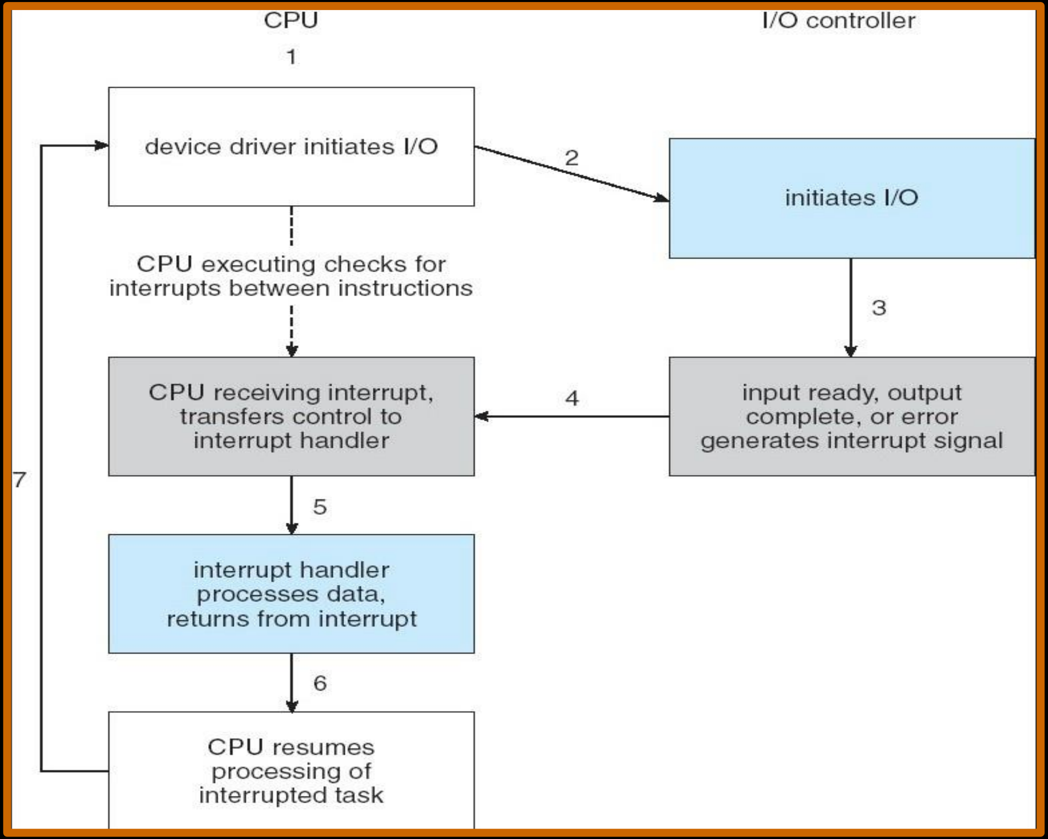
INTERRUPT TYPES

- *Maskable interrupt* (IRQ): a hardware interrupt that may be ignored by setting a bit in an I nterrupt mask register's (IMR) bit-mask.
- *Non-maskable interrupt* (NMI): a hardware interrupt that lacks an associated bit-mask, so that it can never be ignored. NMIs are used for the highest priority tasks such as timers, especially watchdog timers.
- *Inter-processor interrupt* (IPI): a special case of interrupt that is generated by one processor to interrupt another processor in a multiprocessor system.
- *Software interrupt*: an interrupt generated within a processor by executing an instruction.

Software interrupts are often used to implement system calls because they result in a subroutine call with a CPU ring level change.

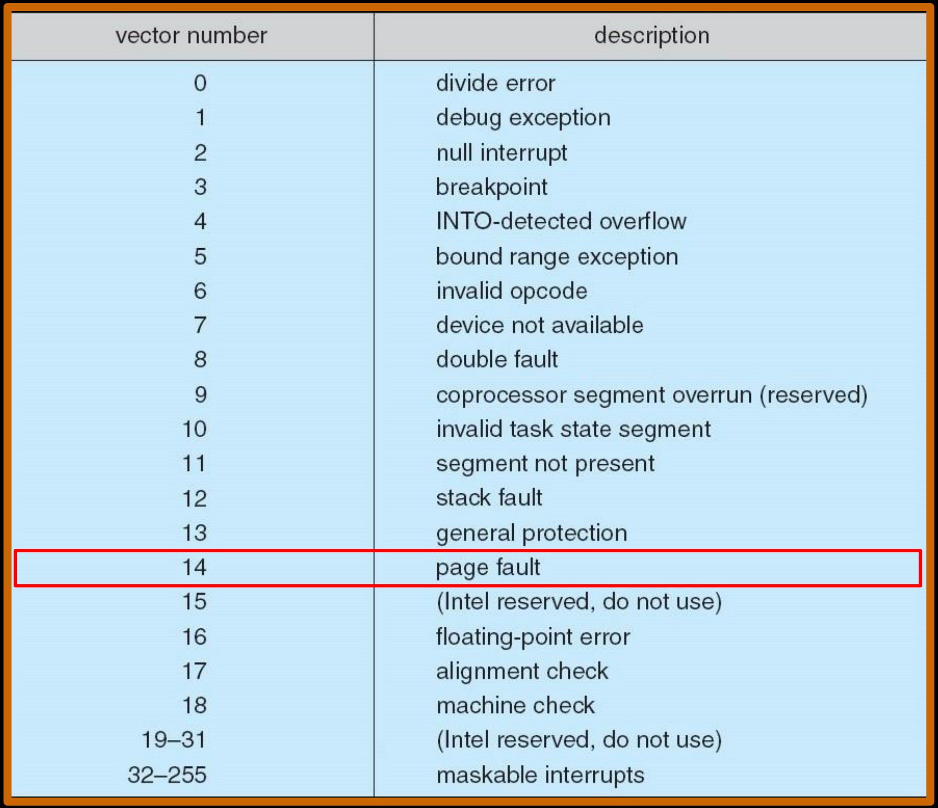
- *Spurious interrupt*: a hardware interrupt that is unwanted. They are typically generated by system conditions such as electrical interference on an interrupt line or through incorrectly designed hardware.

INTERRUPT-DRIVEN I/O CYCLE


CPU
I/O controller
1
device driver initiates I/O
2
initiates I/O
CPU executing checks for interrupts between instructions
3
CPU receiving interrupt, transfers control to interrupt handler
4
input ready, output complete, or error generates interrupt signal
7
5
interrupt handler processes data, returns from interrupt
6
CPU resumes processing of interrupted task

Silberschatz, Galvin Operating System Concepts – 7th Edition and Gagne ©2005

INTEL PENTIUM PROCESSOR EVENT-VECTOR TABLE

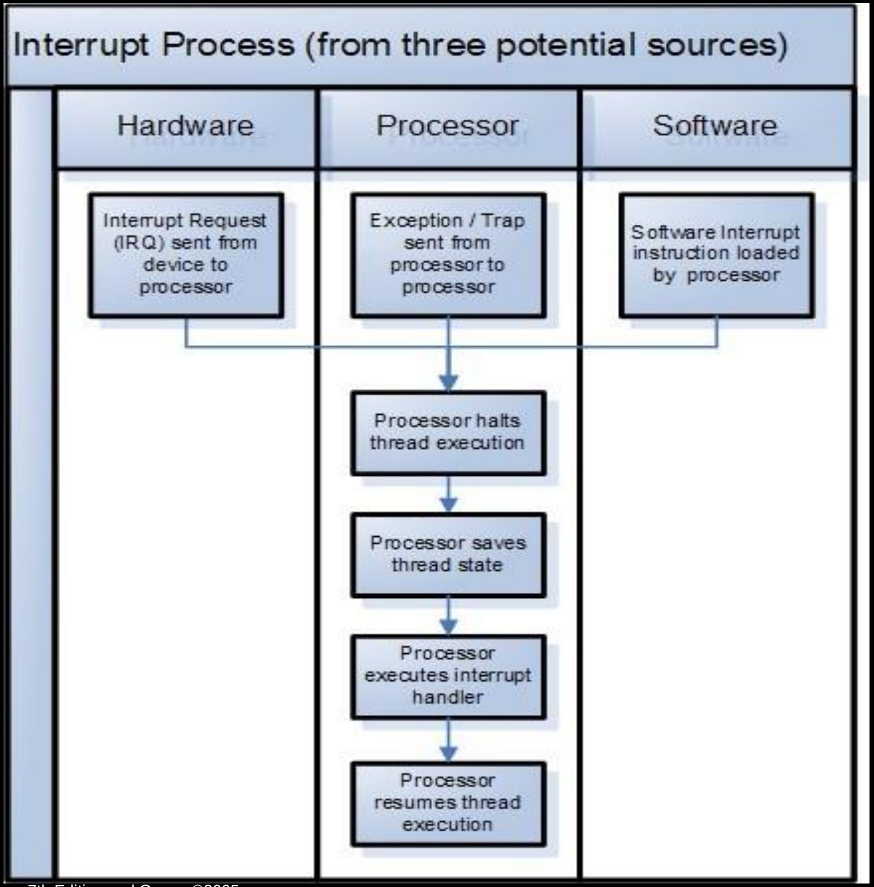


vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
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

VARIOUS INTERRUPT PROCESSING

- Page fault: saves the state of the process, moves it to the waiting queue, schedules another process to resume execution, then returns.
- Trap (s/w interrupt): saves the state of user code, switches to supervisor mode. Low priority
- Low priority interrupt can be preempted by high priority ones.

SUMMARY OF INTERRUPT

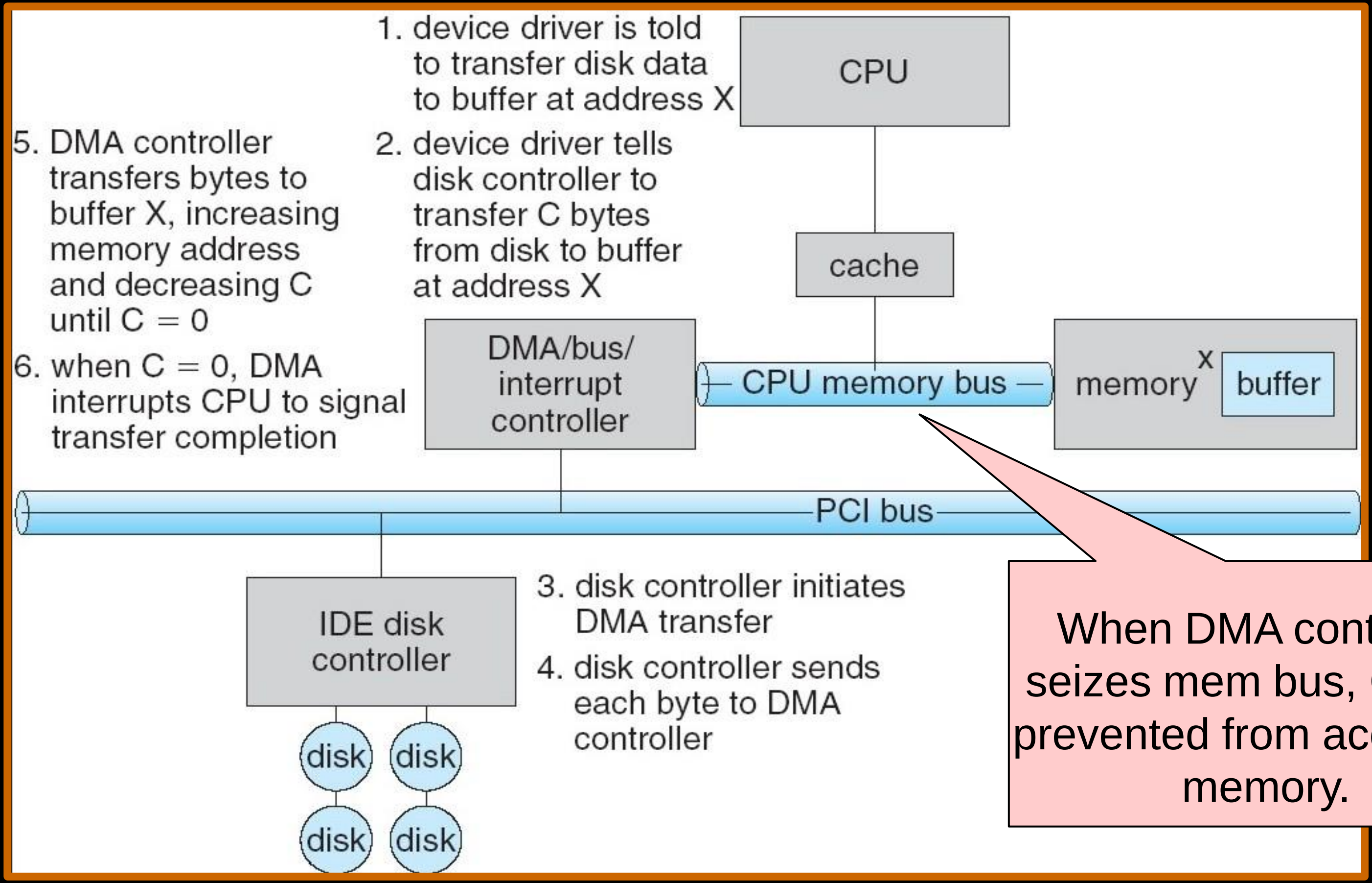

Interrupt Process (from three potential sources)
Hardware
Processor
Software
Interrupt Request (IRQ) sent from device to processor
Exception / Trap sent from processor to processor
Software Interrupt instruction loaded by processor
Processor halts thread execution
Processor saves thread state
Processor executes interrupt handler
Processor resumes thread execution

Silberschatz, Galvin Operating System Concepts – 7th Edition and Gagne ©2005

DIRECT MEMORY ACCESS

- 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

SIX STEP PROCESS TO PERFORM DMA TRANSFER



1. device driver is told to transfer disk data to buffer at address X
2. device driver tells disk controller to transfer C bytes from disk to buffer at address X
3. disk controller initiates DMA transfer
4. disk controller sends each byte to DMA controller
5. DMA controller transfers bytes to buffer X, increasing memory address and decreasing C until C = 0
6. when C = 0, DMA interrupts CPU to signal transfer completion

When DMA controller seizes mem bus, CPU is prevented from accessing memory.

Silberschatz, Galvin Operating System Concepts – 7th Edition and Gagne ©2005

DMA MODE

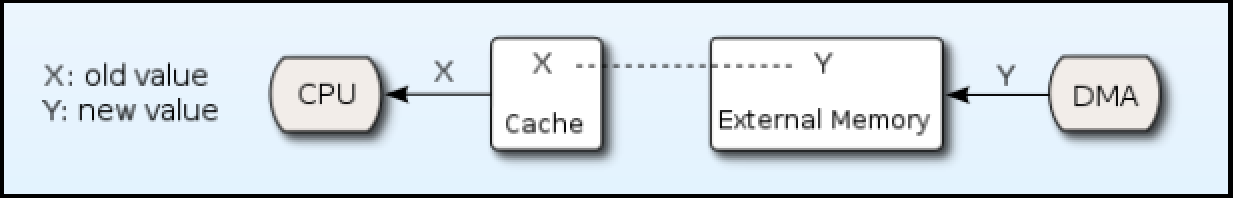
Burst mode
An entire block of data is transferred in one contiguous sequence. Once the DMA controller is granted access to the system bus by the CPU, it transfers all bytes of data in the data block before releasing control of the system buses back to the CPU, but renders the CPU inactive for relatively long periods of time.

Cycle stealing mode
In cycle stealing mode, after one byte of data transfer, the control of the system bus is deasserted to the CPU via BG. It is then continually requested again via BR, transferring one byte of data per request, until the entire block of data has been transferred. By continually obtaining and releasing the control of the system bus, the DMA controller essentially interleaves instruction and data transfers. The CPU processes an instruction, then the DMA controller transfers one data value, and so on.

Transparent mode
The *transparent mode* takes the most time to transfer a block of data, yet it is also the most efficient mode in terms of overall system performance. The DMA controller only transfers data when the CPU is performing operations that do not use the system buses.

DMA CACHE PROBLEM

- DMA updates a value in memory which has been modified by CPU in cache

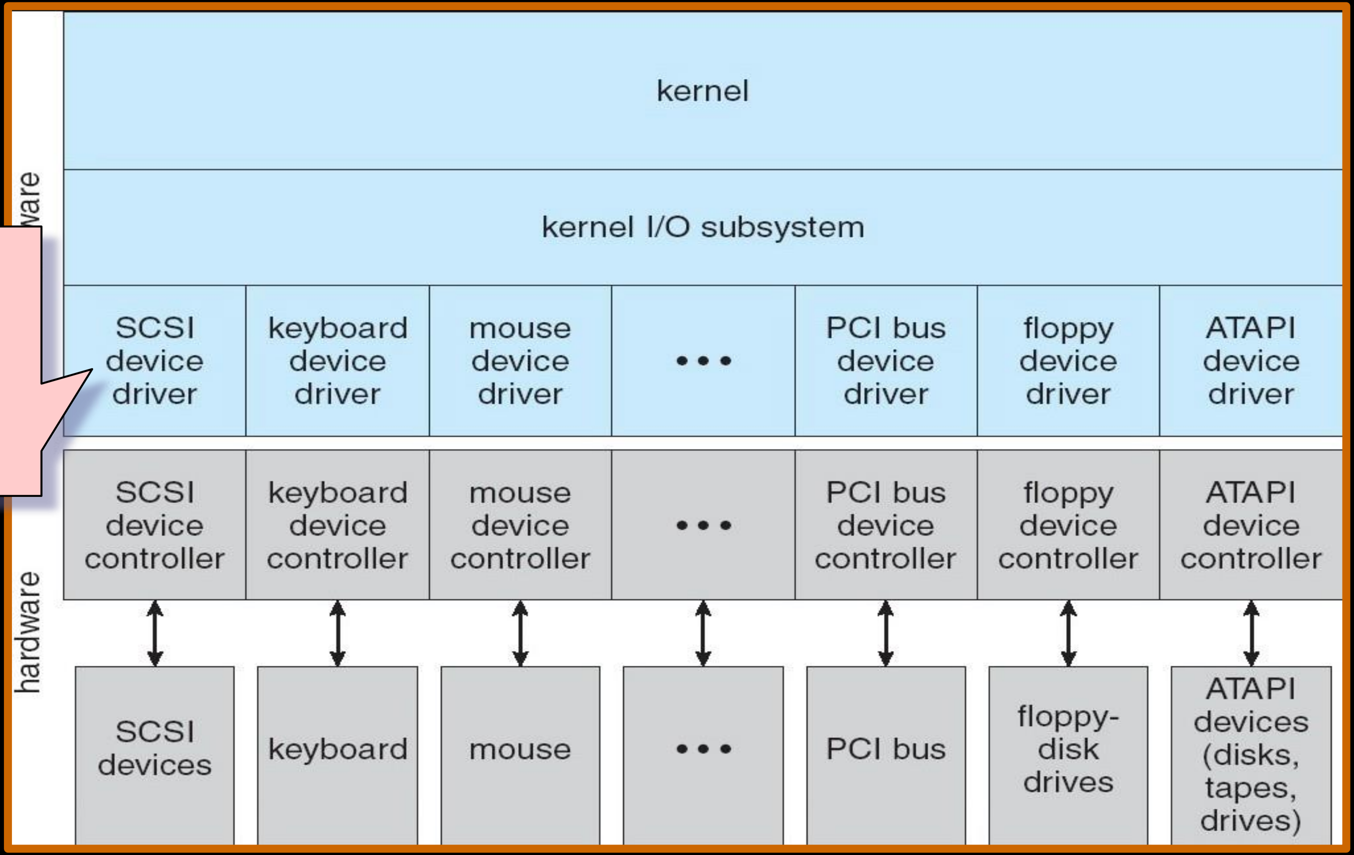


Silberschatz, Galvin Operating System Concepts – 7th Edition and Gagne ©2005

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**

A KERNEL I/O STRUCTURE



ware

kernel

kernel I/O subsystem

SCSI device driver | keyboard device driver | mouse device driver | • • • | PCI bus device driver | floppy device driver | ATAPI device driver

SCSI device controller | keyboard device controller | mouse device controller | • • • | PCI bus device controller | floppy device controller | ATAPI device controller

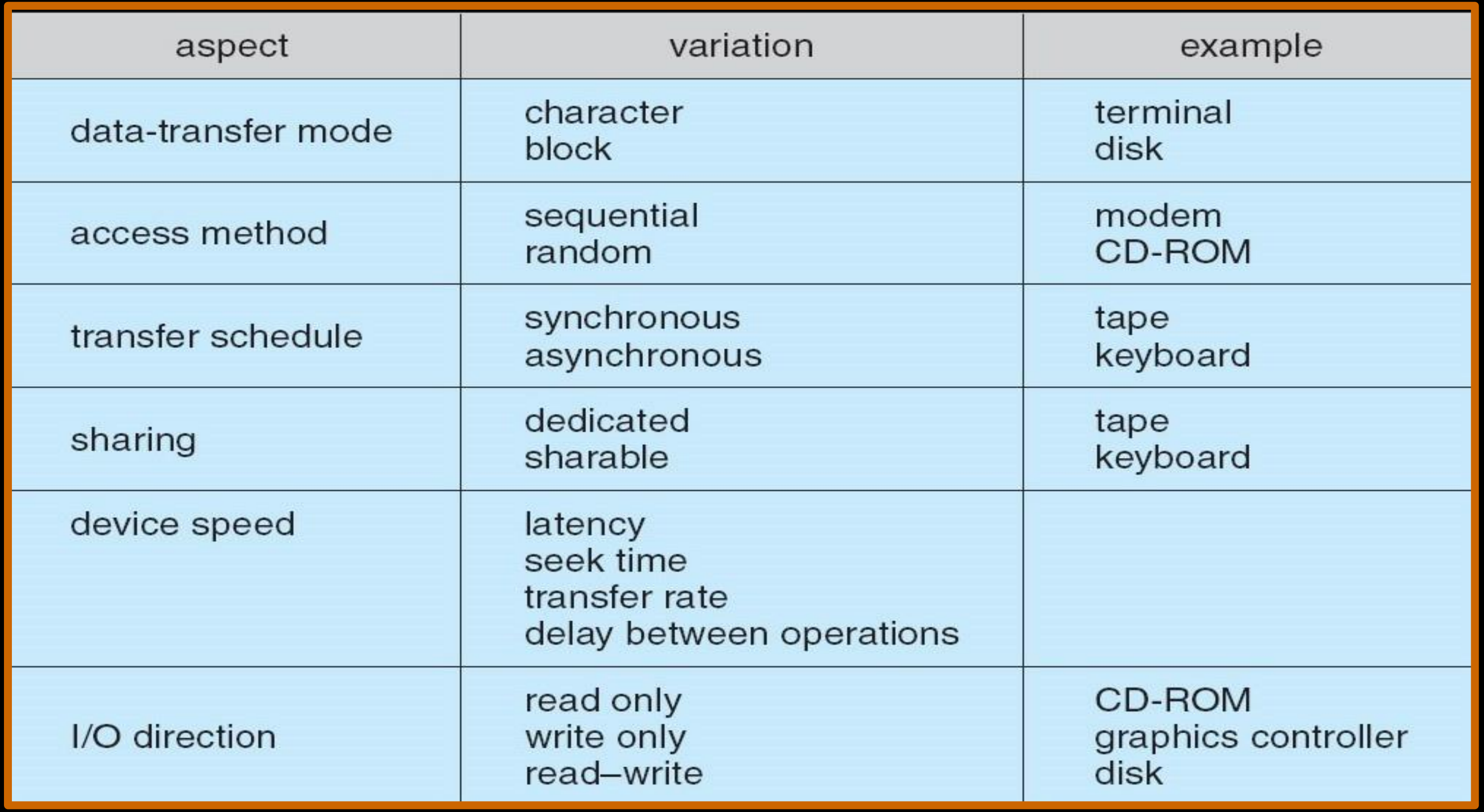
hardware

SCSI devices | keyboard | mouse | • • • | PCI bus | floppy-disk drives | ATAPI devices (disks, tapes, drives)

Device driver layer:
Hides differences among device controllers

Silberschatz, Galvin Operating System Concepts – 7th Edition and Gagne ©2005

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

Silberschatz, Galvin Operating System Concepts – 7th Edition and Gagne ©2005

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 of line editing

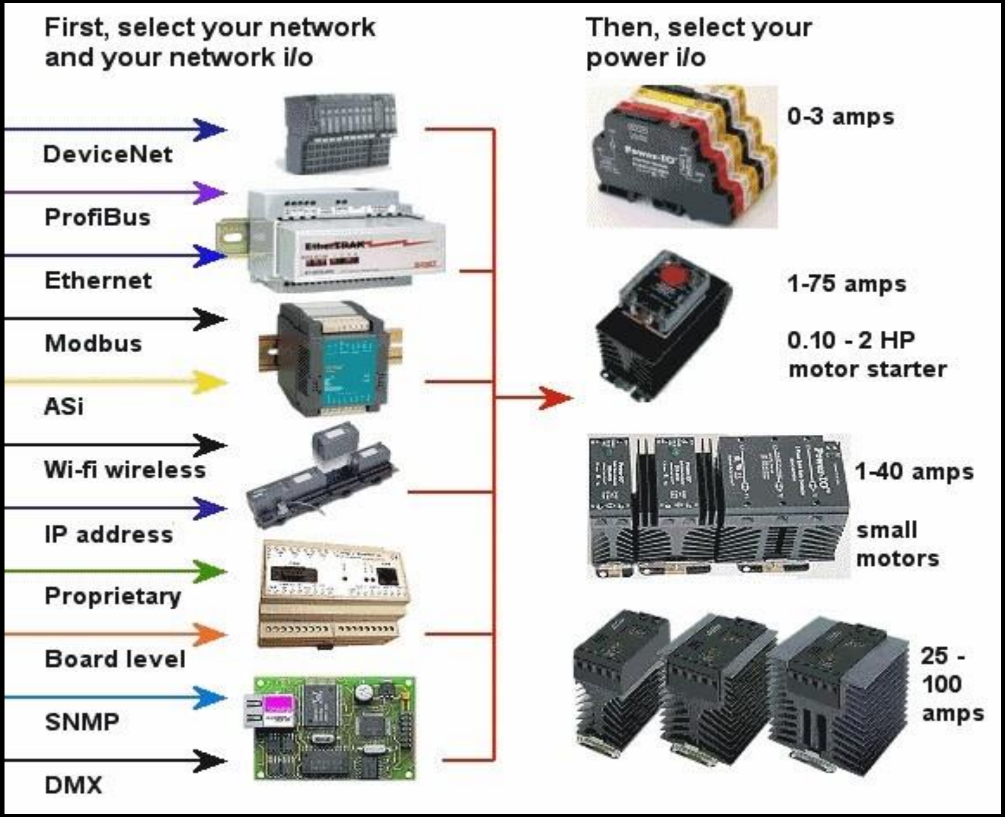
NETWORK DEVICES

- Varying enough from block and character to have own interface

- Unix and Windows NT/9*x*/2000 include socket interface
  - Separates network protocol from network operation
  - Includes `select` functionality

- Approaches vary widely (pipes, FIFOs, streams, queues, mailboxes)

NETWORK I/O DEVICES



CLOCKS AND TIMERS

- Provide current time, elapsed time, timer
- **Programmable interval timer** used for timings, to generate periodic interrupts
- `ioctl` (on UNIX) covers odd aspects of I/O such as clocks and timers
  - ioctl, which means “input-output control” is a kind of device-specific system call
  - ioctl function is useful when one is implementing a device driver to set the configuration on the device

```
int ioctl(int fd, int request, ...)
```

- `fd` is file descriptor, the one returned by open
- `request` is request code. e.g GETFONT will get current font from printer, SETFONT will set font
- third argument is `void *`. Depending on second argument, the third may or may not be present. e.g. if second argument is SETFONT, third argument may give font name as ARIAL.

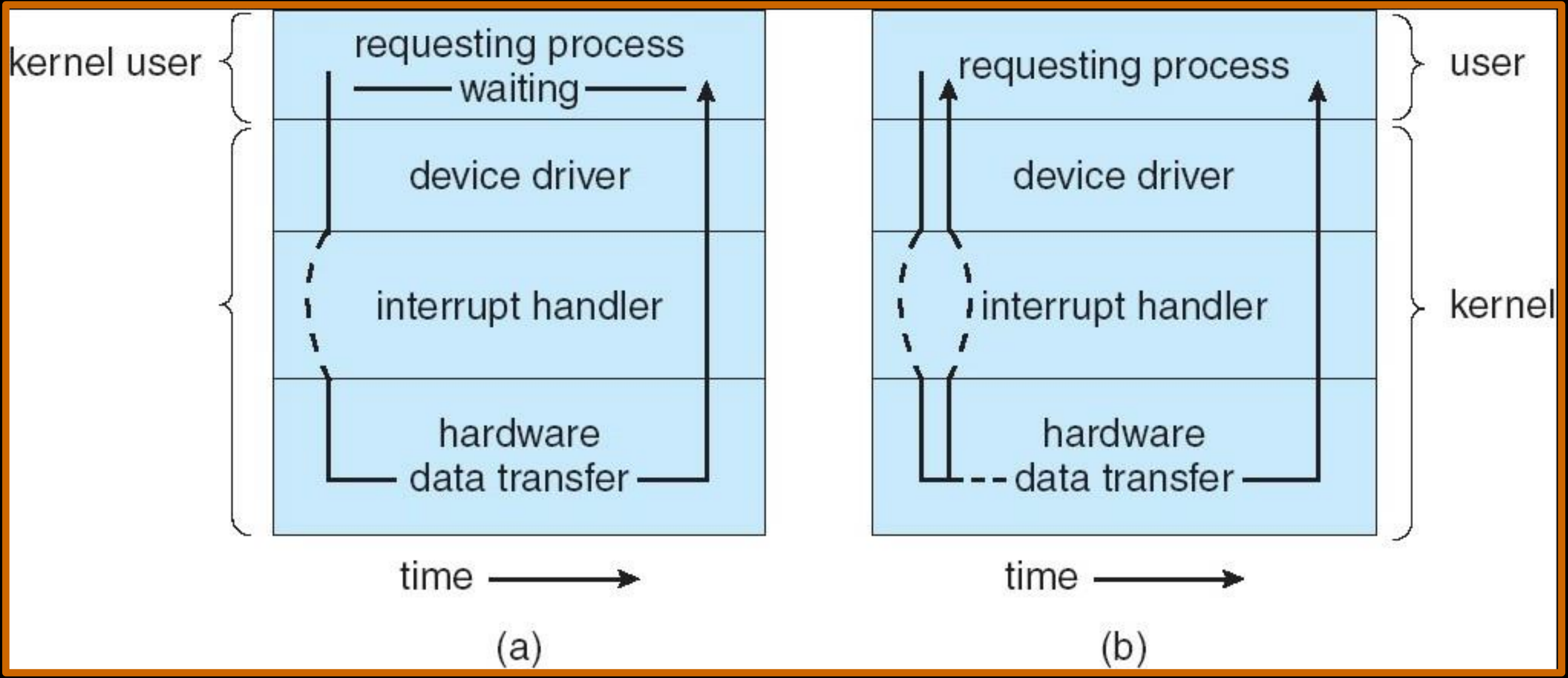
SUMMARY OF CLOCKS

S.N.	Task	Description
1	Maintaining the time of the day	The clock driver implements the time of day or the real time clock function.It requires incrementing a counter at each clock tick.
2	Preventing processes from running too long	As a process is started, the scheduler initializes the quantum counter in clock ticks for the process. The clock driver decrements the quantum counter by 1, at every clock interrupt. When the counter gets to zero , clock driver calls the scheduler to set up another process. Thus clock driver helps in preventing processes from running longer than time slice allowed.
3	Accounting for CPU usage	Another function performed by clock driver is doing CPU accounting. CPU accounting implies telling how long the process has run.
4	Providing watchdog timers for parts of the system itself	Watchdog timers are the timers set by certain parts of the system. For example, to use a floppy disk, the system must turn on the motor and then wait about 500msec for it to comes up to speed.

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

TWO I/O METHODS

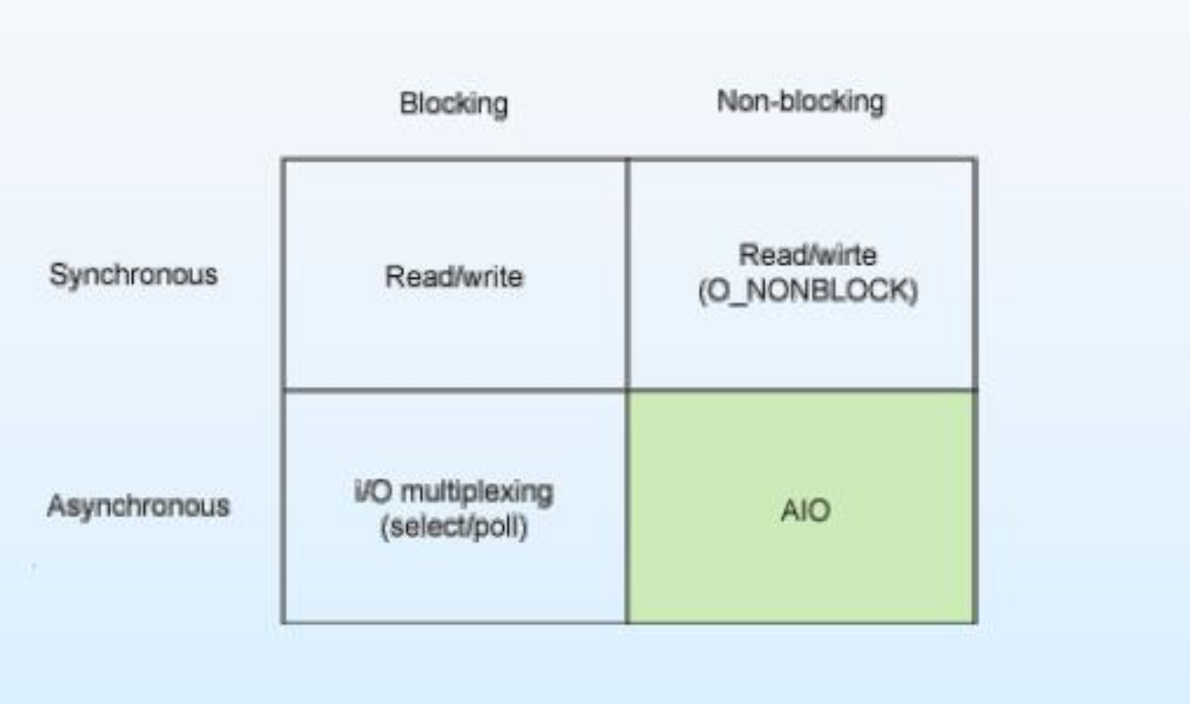


Synchronous

Asynchronous

Silberschatz, Galvin Operating System Concepts – 7th Edition and Gagne ©2005

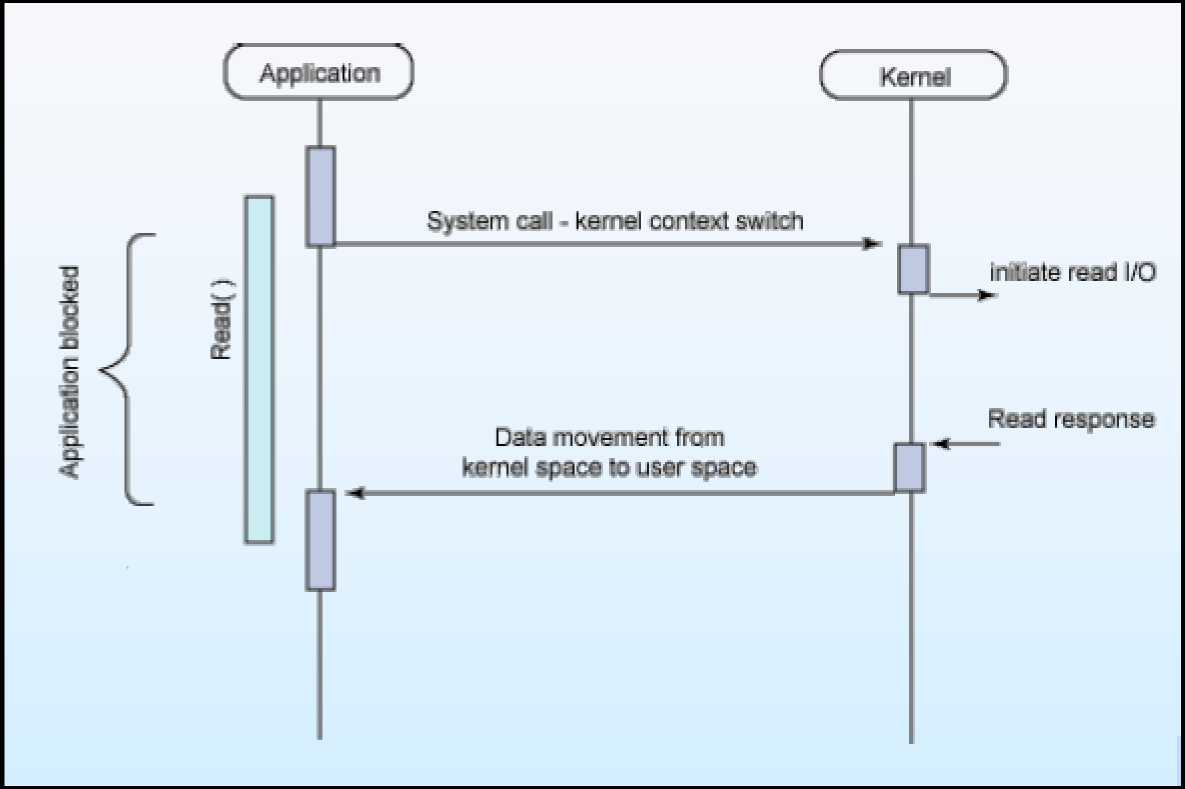
I/O MODES



	Blocking	Non-blocking
Synchronous	Read/write	Read/wirte (O_NONBLOCK)
Asynchronous	I/O multiplexing (select/poll)	AIO

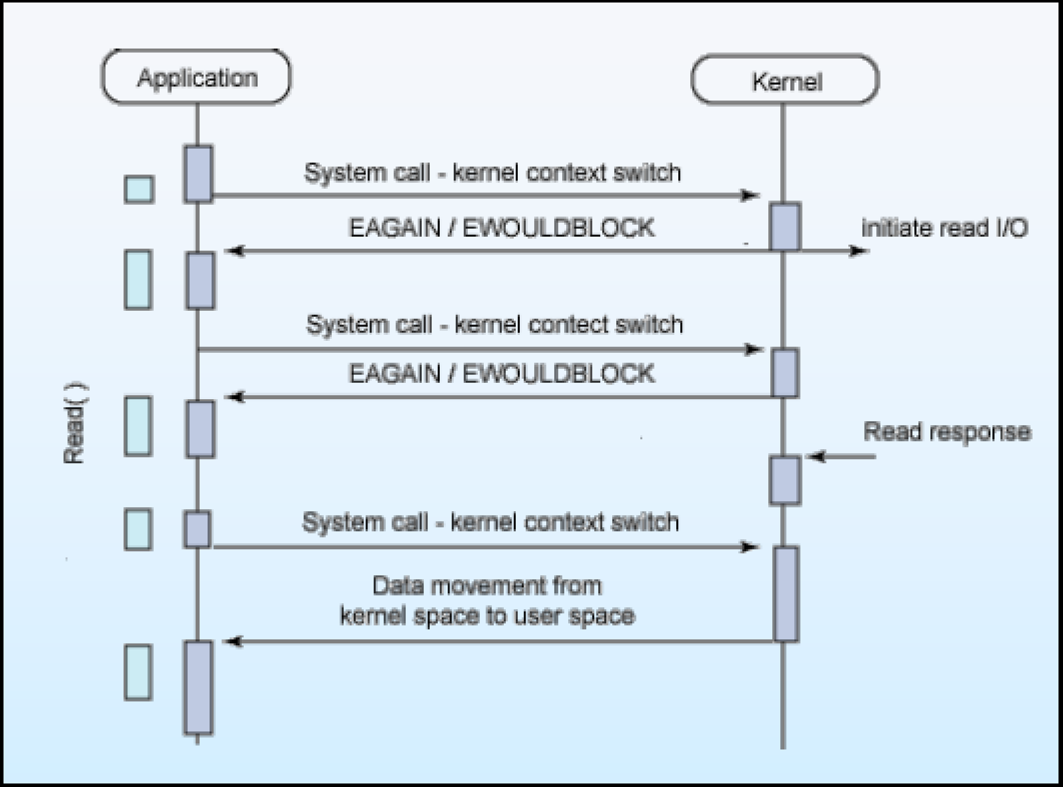
Silberschatz, Galvin Operating System Concepts – 7th Edition and Gagne ©2005

SYNCHRONOUS BLOCKING I/O



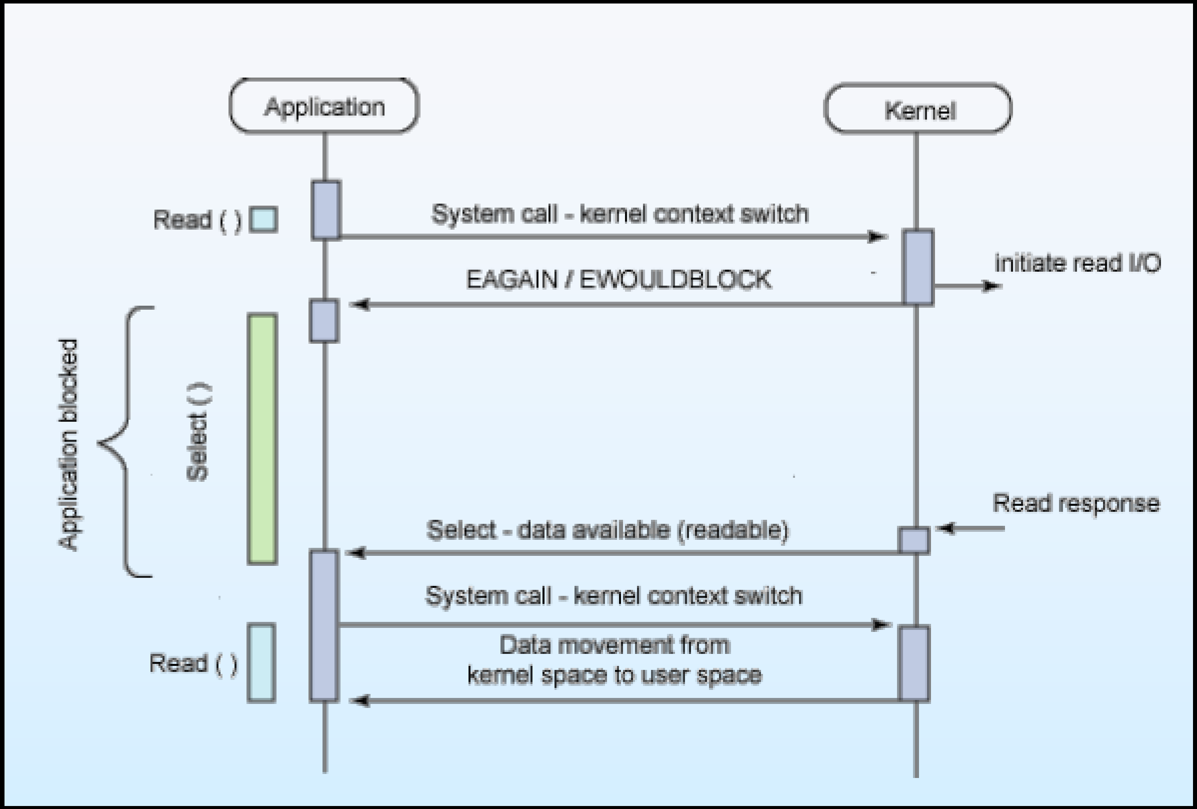
Silberschatz, Galvin Operating System Concepts – 7th Edition and Gagne ©2005

SYNCHRONOUS NON-BLOCKING I/O


Application
Kernel
System call - kernel context switch
EAGAIN / EWOULDBLOCK
initiate read I/O
System call - kernel contect switch
EAGAIN / EWOULDBLOCK
Read( )
Read response
System call - kernel context switch
Data movement from
kernel space to user space

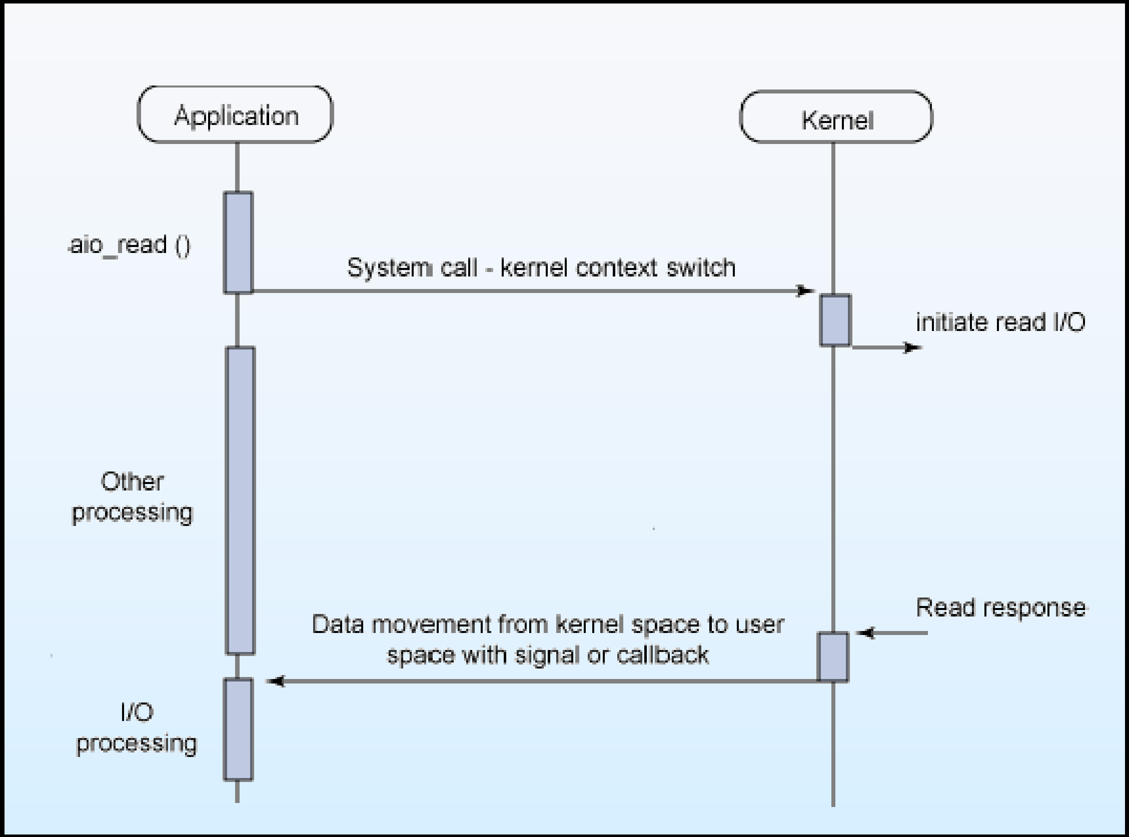
Silberschatz, Galvin Operating System Concepts – 7th Edition and Gagne ©2005

ASYNCHRONOUS BLOCKING I/O


Application
Kernel
Read ( )
System call - kernel context switch
initiate read I/O
EAGAIN / EWOULDBLOCK
Application blocked
Select ( )
Read response
Select - data available (readable)
System call - kernel context switch
Data movement from
kernel space to user space
Read ( )

Silberschatz, Galvin Operating System Concepts – 7th Edition and Gagne ©2005

ASYNCHRONOUS NON-BLOCKING I/O (AIO)


Application
Kernel
aio_read ()
System call - kernel context switch
initiate read I/O
Other processing
Read response
Data movement from kernel space to user space with signal or callback
I/O processing

Silberschatz, Galvin Operating System Concepts – 7th Edition and Gagne ©2005

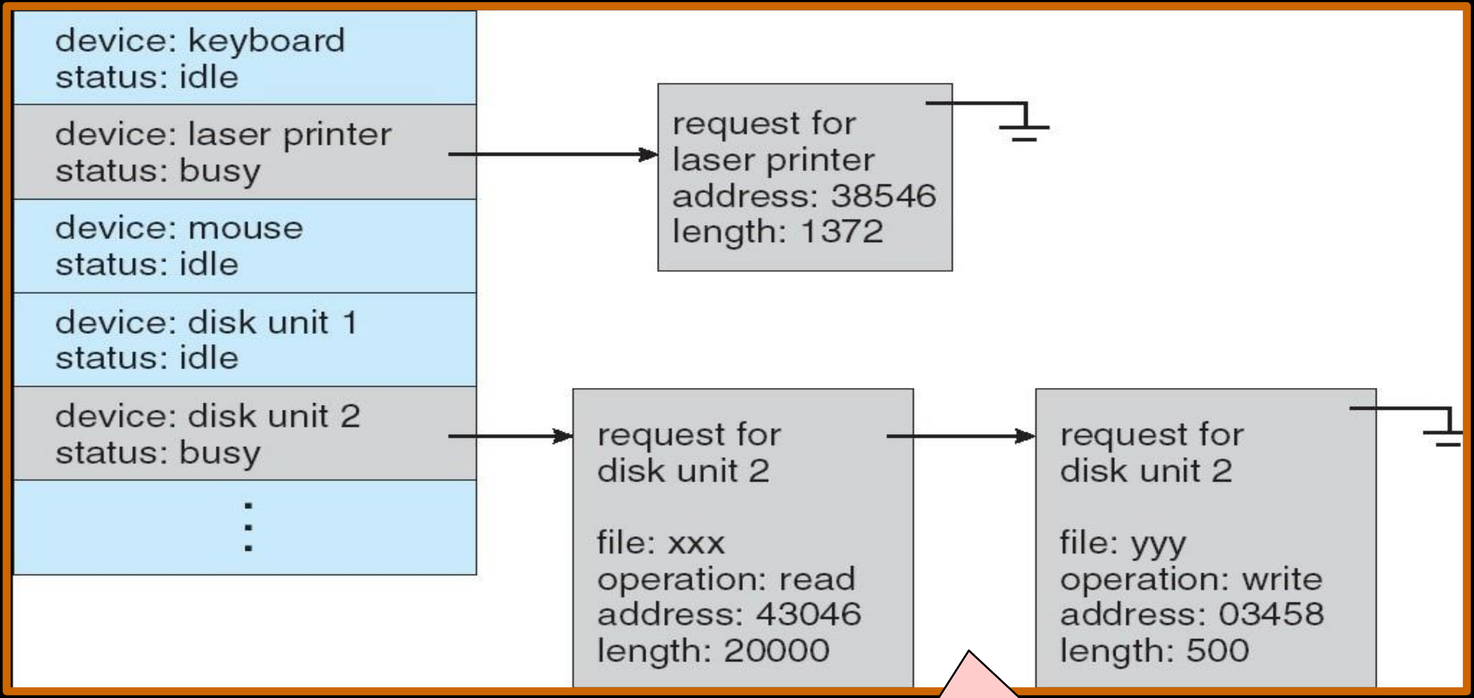
KERNEL I/O SUBSYSTEM

- **Scheduling**
  - Some I/O request ordering via per-device queue
    - E.g. disk scheduling
  - Some OSs try fairness

- **Buffering** - store data in memory while transferring between devices
  - To cope with device speed mismatch, e.g. receiving data from modem to disk.
  - To cope with device transfer size mismatch, e.g. network packet
  - To maintain “copy semantics” (when a write() system call specifies a buffer for storing the data, and modifies its contents after the system call)

DEVICE-STATUS TABLE

AKA “DEVICE-CONTROL TABLE”



device: keyboard
status: idle

device: laser printer
status: busy

device: mouse
status: idle

device: disk unit 1
status: idle

device: disk unit 2
status: busy

⋮

request for
laser printer
address: 38546
length: 1372

request for
disk unit 2

file: xxx
operation: read
address: 43046
length: 20000

request for
disk unit 2

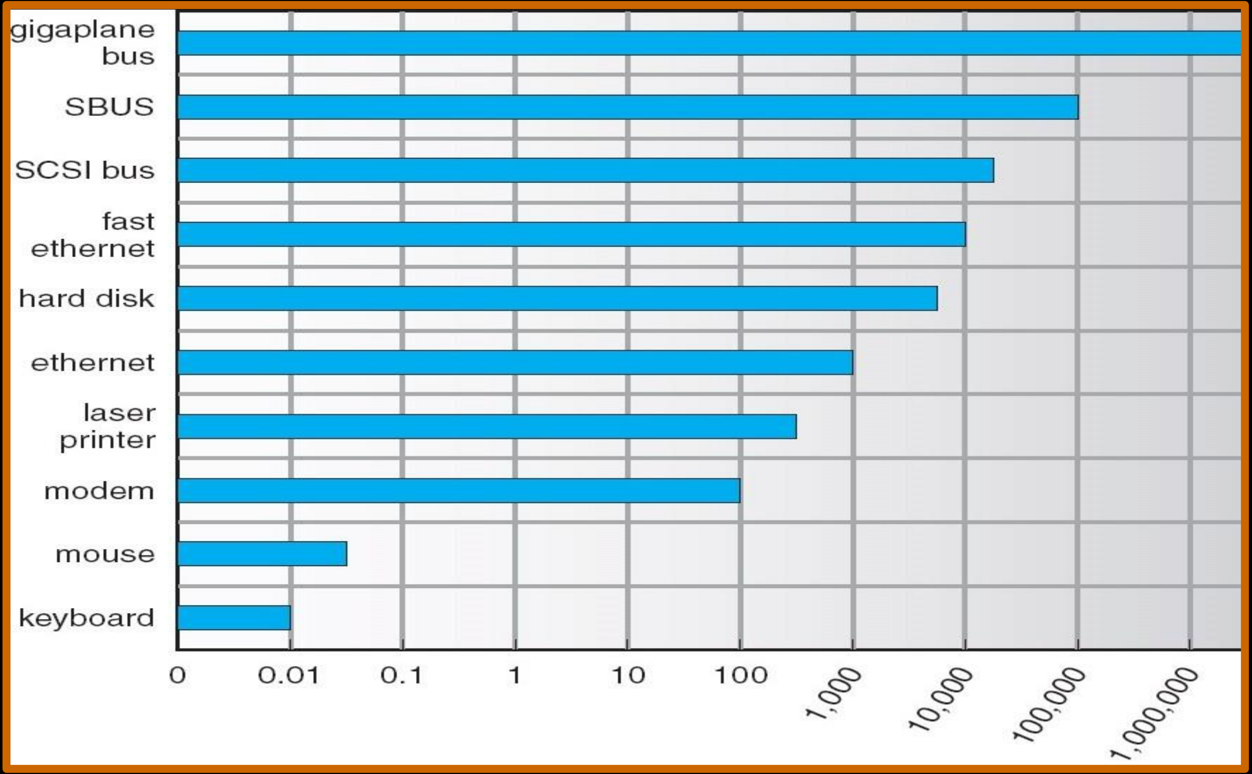
file: yyy
operation: write
address: 03458
length: 500

I/O scheduling on
each device queue

Silberschatz, Galvin Operating System Concepts – 7th Edition and Gagne ©2005

SUN ENTERPRISE 6000 DEVICE-TRANSFER RATES

To illustrate the differences in device speeds.



Silberschatz, Galvin Operating System Concepts – 7th Edition and Gagne ©2005

KERNEL I/O SUBSYSTEM

- **Caching** - fast memory holding copy of data
  - Always just a copy
  - Key to performance
- **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 deallocation
  - Watch out for deadlock

ERROR HANDLING

- OS can recover from disk read, device unavailable, transient write failures

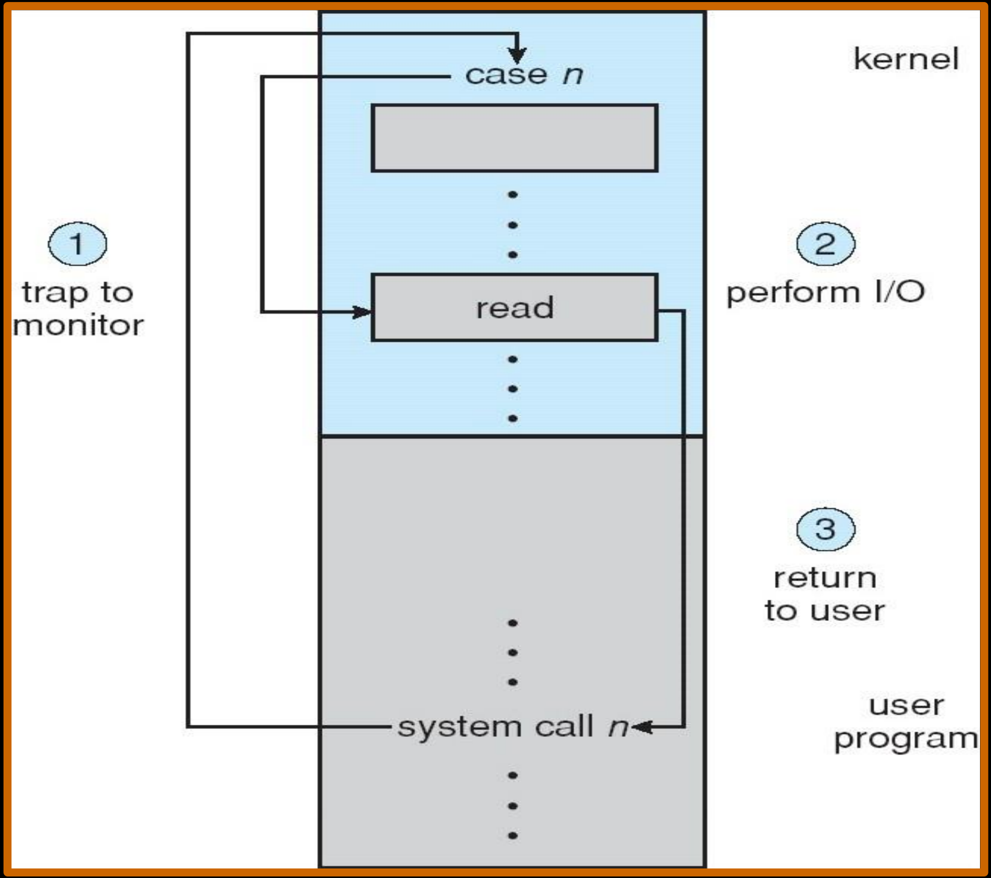
- Most return an error number or code when I/O request fails

- System error logs hold problem reports

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– cannot be issued directly
  - 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

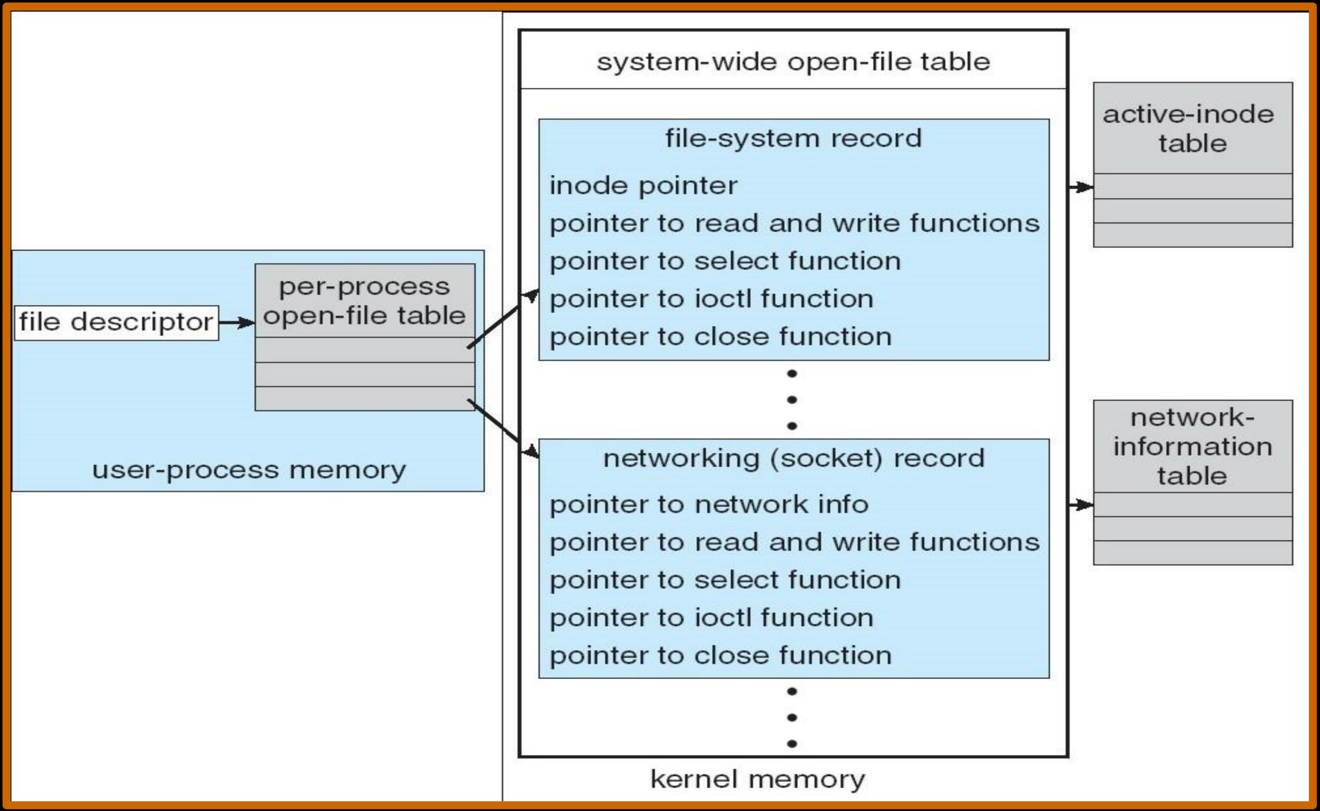


Silberschatz, Galvin Operating System Concepts – 7th Edition and Gagne ©2005

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. e.g. Unix provides file-system access to a variety of entities such as *user files, raw disk, network socket* etc.

UNIX I/O KERNEL STRUCTURE


system-wide open-file table
file-system record
inode pointer
pointer to read and write functions
pointer to select function
pointer to ioctl function
pointer to close function
active-inode table
file descriptor
per-process open-file table
user-process memory
networking (socket) record
pointer to network info
pointer to read and write functions
pointer to select function
pointer to ioctl function
pointer to close function
network-information table
kernel memory

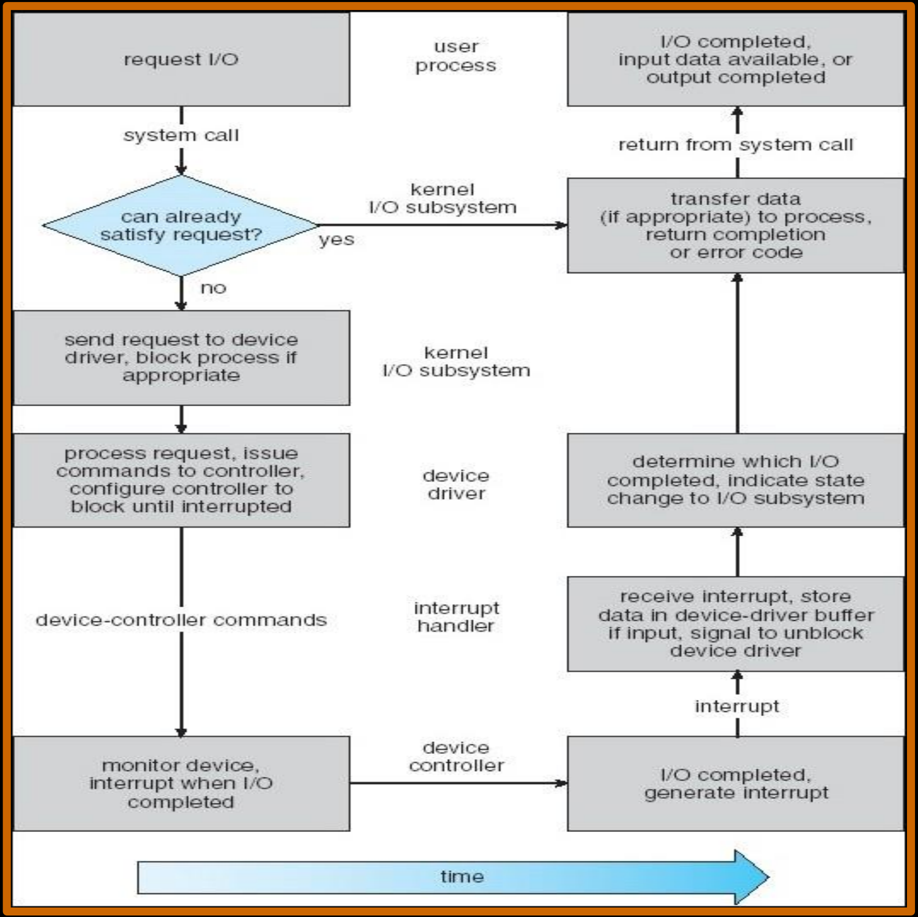
Silberschatz, Galvin Operating System Concepts – 7th Edition and Gagne ©2005

TRANSFORMING I/O REQUESTS TO HARDWARE OPERATIONS

- Consider reading a file from disk for a process:

  - Determine device holding file
    - MS-DOS uses the c: disk id; Unix uses the mount table
  - 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


request I/O
user process
I/O completed, input data available, or output completed
system call
return from system call
kernel I/O subsystem
can already satisfy request?
yes
transfer data (if appropriate) to process, return completion or error code
no
send request to device driver, block process if appropriate
kernel I/O subsystem
process request, issue commands to controller, configure controller to block until interrupted
device driver
determine which I/O completed, indicate state change to I/O subsystem
device-controller commands
interrupt handler
receive interrupt, store data in device-driver buffer if input, signal to unblock device driver
interrupt
monitor device, interrupt when I/O completed
device controller
I/O completed, generate interrupt
time

Silberschatz, Galvin Operating System Concepts – 7th Edition and Gagne ©2005

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

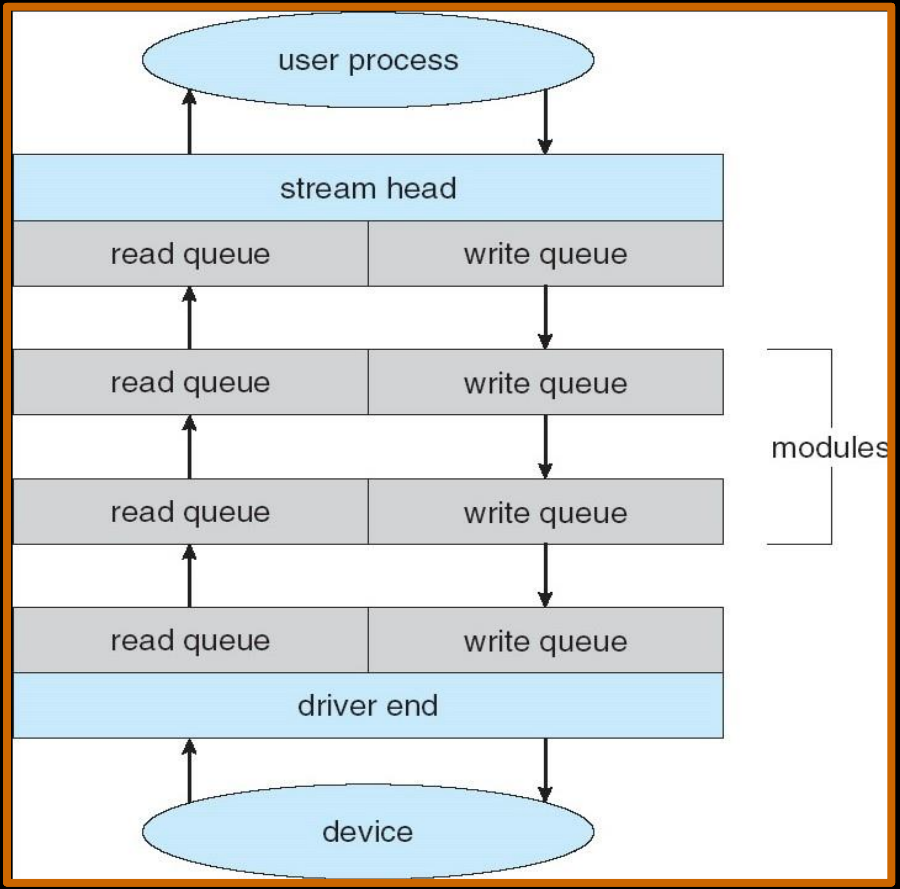
  - driver end interfaces with the device

  - 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
- **STREAM** provides a framework for a modular and incremental approach to writing device drivers and network protocols.

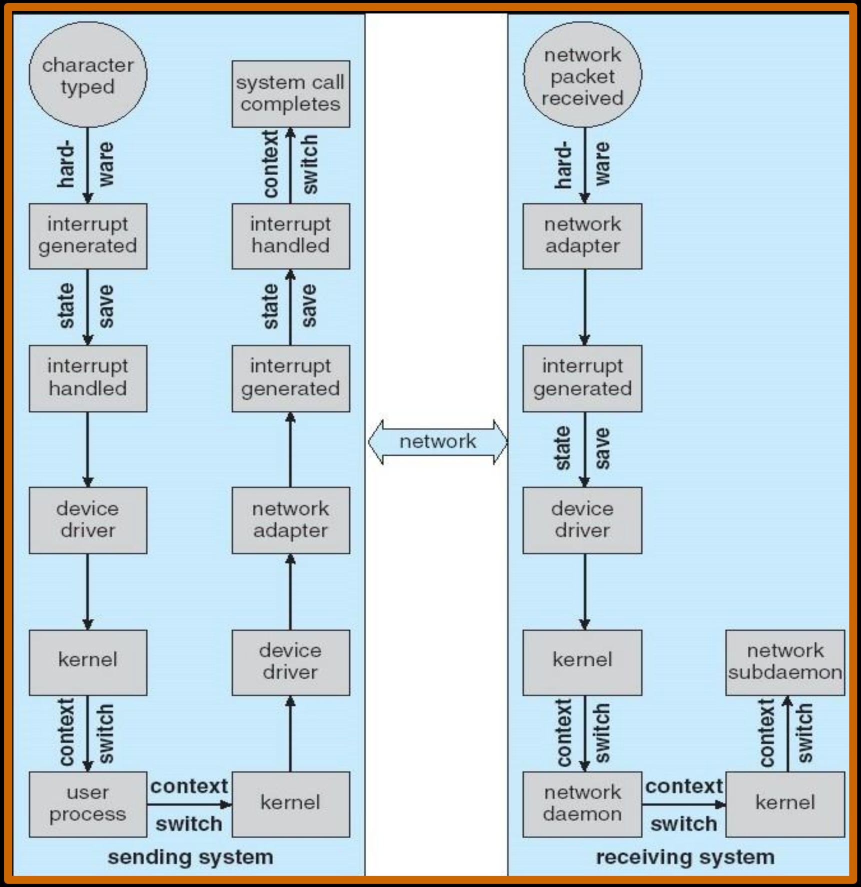
THE STREAMS STRUCTURE



PERFORMANCE

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

INTERCOMPUTER COMMUNICATIONS

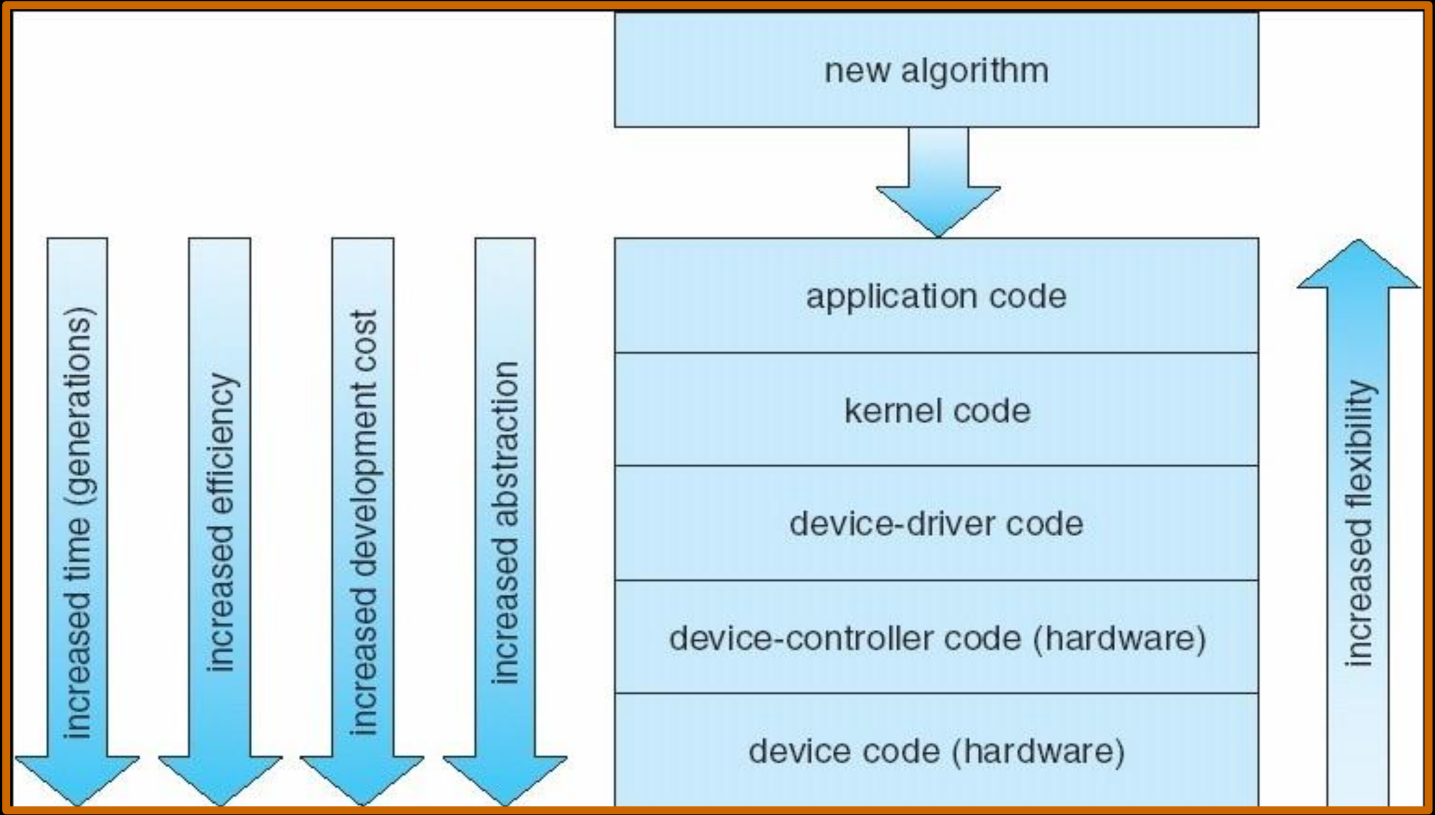


Silberschatz, Galvin Operating System Concepts – 7th Edition and Gagne ©2005

IMPROVING PERFORMANCE

- Reduce number of context switches
- Reduce data copying
- Reduce interrupts by using large transfers, smart controllers, polling
- Use DMA
- Balance CPU, memory, bus, and I/O performance for highest throughput

DEVICE-FUNCTIONALITY PROGRESSION



Silberschatz, Galvin Operating System Concepts – 7th Edition and Gagne ©2005

LUT University