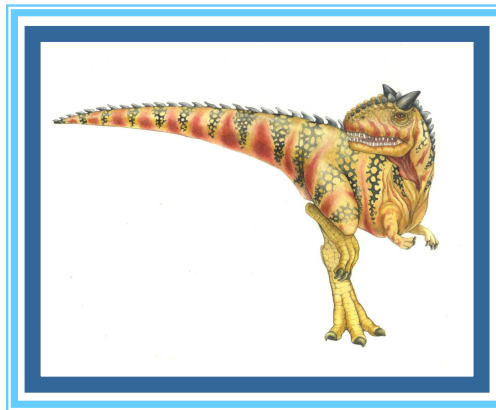


# Chapter 13: I/O Systems

---





# Chapter 13: I/O Systems

---

- Overview
- I/O Hardware
- Application I/O Interface
- Kernel I/O Subsystem
- Performance of I/O and OS efforts to improve 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





# Overview

---

- I/O management is a **major** component of operating system design and operation (I/O subsystem)
  - Important aspect of computer operation
  - I/O devices vary greatly
  - Various methods to control them
  - Performance management
  - New types of devices frequent
- Basic I/O hardware elements:
  - Ports, buses, device controllers connect to various devices
- **Device drivers** encapsulate device details
  - Present uniform device-access interface to I/O subsystem





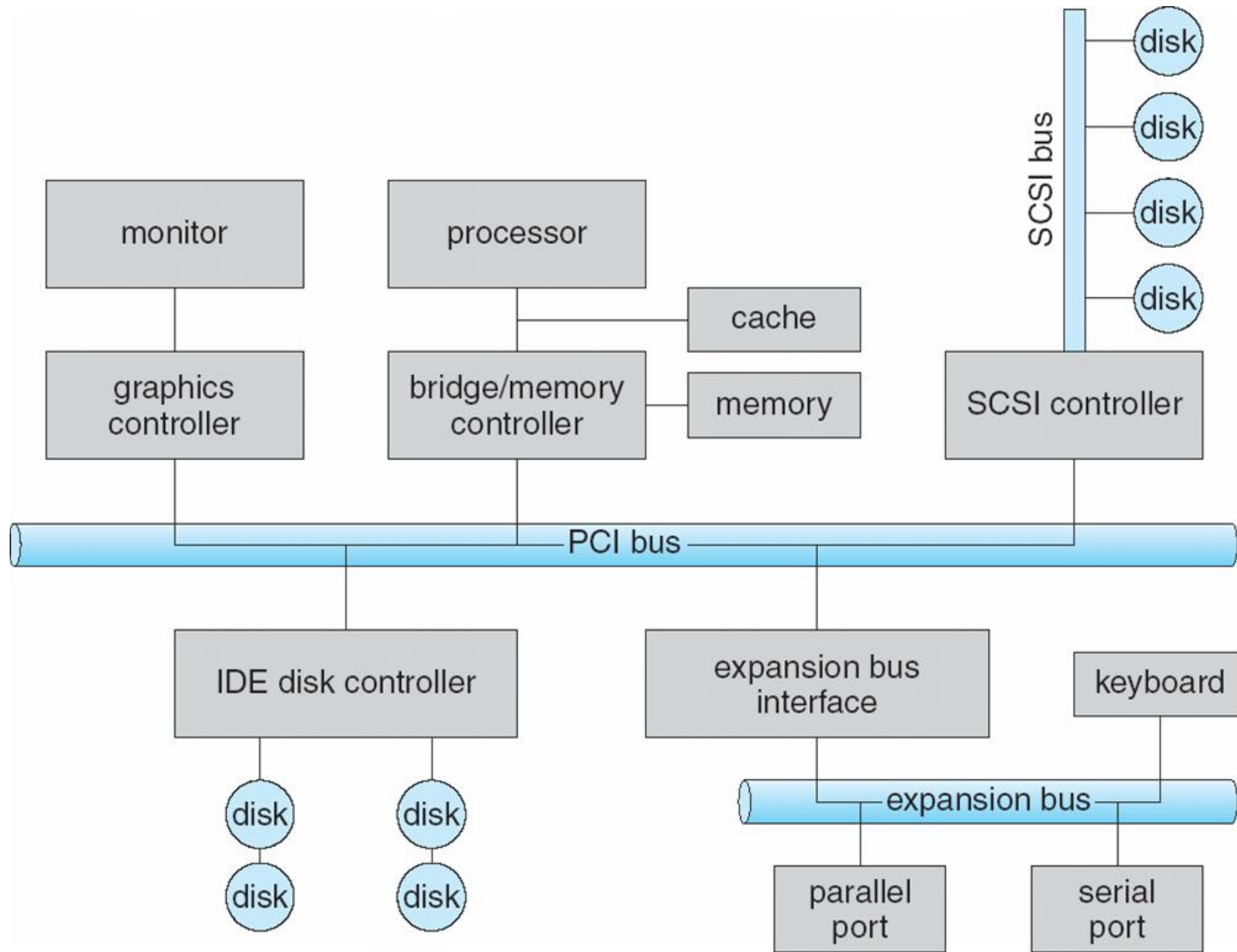
# I/O Hardware

- Incredible variety of I/O devices
  - Storage (disk, tape)
  - Transmission (network card, modem)
  - Human-interface (screen, keyboard, mouse)
- Common concepts – signals from I/O devices interface with computer
  - **Port** – connection point for device
  - **Bus** (set of wires with protocol for message transfer)
    - ▶ **PCI bus** common in PCs and servers
    - ▶ **expansion bus** connects relatively slow devices
  - **Controller** – electronics that operate port, bus, device
    - ▶ Sometimes integrated (e.g., serial port - simple)
    - ▶ Sometimes separate circuit board (host adapter, e.g., SCSI)
      - Contains processor, microcode, private memory, bus controller, etc.





# A Typical PC Bus Structure





# I/O Hardware (Cont.)

---

- I/O instructions control devices (processor -> controller)
- Devices usually have **registers** where device driver places commands, addresses, and data to write, or read data from registers after command execution
  - Data-in register (read by host),
  - Data-out register (write by host),
  - Status register (read by host, indicate states),
  - Control register (write by host, start command, etc.)
    - ▶ Typically above registers are 1-4 bytes, or FIFO buffer
- Devices have addresses, used by
  - Direct I/O instructions
  - The unique address in the device controller (I/O address)





# 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 (轮询)

## How host communicates with devices?

### ■ For each byte of I/O, repeatedly

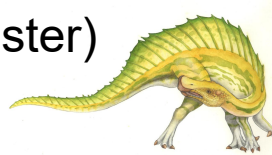
ready to accept next commands

1. Read busy bit from status register until 0
2. Host sets read or write bit, and if write, copies data into data-out register
3. Host sets command-ready bit
4. Controller sets busy bit, executes transfer
5. Controller clears busy bit, error bit, command-ready bit when transfer done

command available to execute

### ■ Step 1, host is **busy-wait** cycle (polling) to wait for I/O from device

- Reasonable if device is fast
- But inefficient if device slow
- CPU switches to other tasks?
  - ▶ But if miss a cycle data overwritten / lost (e.g., small register)





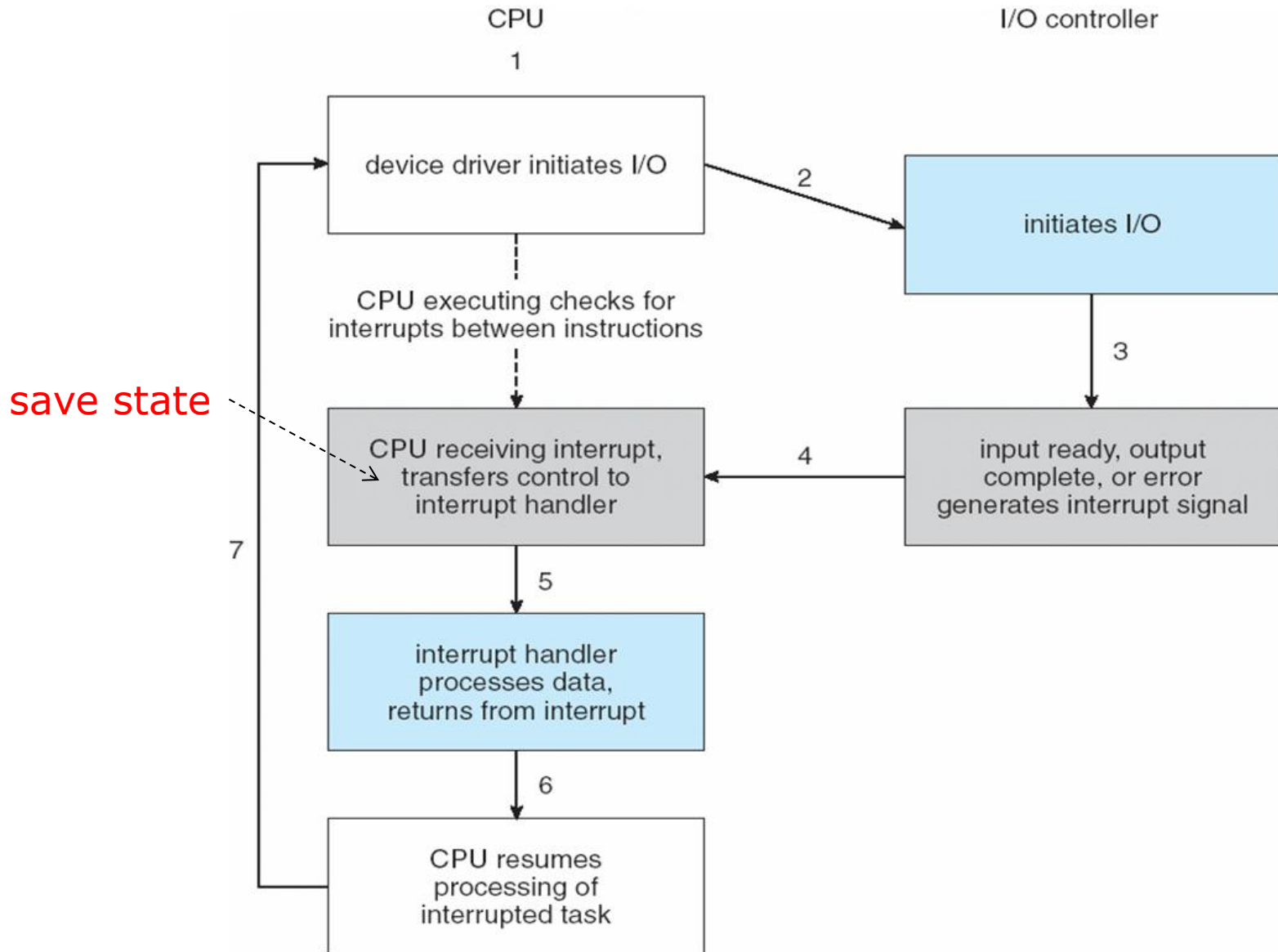
# Interrupts

- Polling – host repeatedly checks devices
  - How to be more efficient? – devices notify CPU when ready
- CPU (hardware wire) **Interrupt-request line** triggered by I/O device
  - Checked by processor after each instruction
- **Interrupt handler** receives interrupts
  - **Maskable** to ignore or delay some interrupts
    - ▶ when CPU executes critical tasks that can't be interrupted
- **Interrupt vector** (remembering handler addresses) to dispatch interrupt to correct handler
  - Based on priority (can defer low priority interrupts without masking)
  - Some **nonmaskable** (e.g., unrecoverable memory errors)
  - Interrupt chaining if more than one device at same interrupt number





# Interrupt-Driven I/O Cycle





# 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

nonmaskable





# 中断(Interrupt)和陷阱(trap)的区别

- 陷阱通常由处理机正在执行的现行指令引起，而中断则是由与现行指令无关的中断源引起的。
- 陷阱处理程序提供的服务为当前进程所用，而中断处理程序提供的服务则不是为了当前进程的。
- CPU在执行完一条指令之后，下一条指令开始之前响应中断，而在一条指令执行中也可以响应陷阱。
  - 例如执行指令非法时，尽管被执行的非法指令不能执行结束，但CPU仍可对其进行处理。
- 在有的系统中，陷阱处理程序被规定在各自的进程上下文中执行，而中断处理程序则在系统上下文中执行。





# 中断控制方式

## 优点

- 能够支持多道程序和设备的并行操作，CPU的利用率大大提高
- 具有实时响应能力，可应用于实时控制场合。

## 缺点

- 控制器数据缓冲寄存器较小，完成一次I/O可能要多次中断驱动，发生中断次数较多。

- 中断驱动方式仅适合于中、慢速设备。对于大批量成组数据交换，可以利用DMA方式。

响

- ▶ 一方面高速外设由于中断方式可能来不及响应而丢失数据
- ▶ 另一方面成组数据交换多次地通过中断进行，也显得速度太慢。





# Direct Memory Access

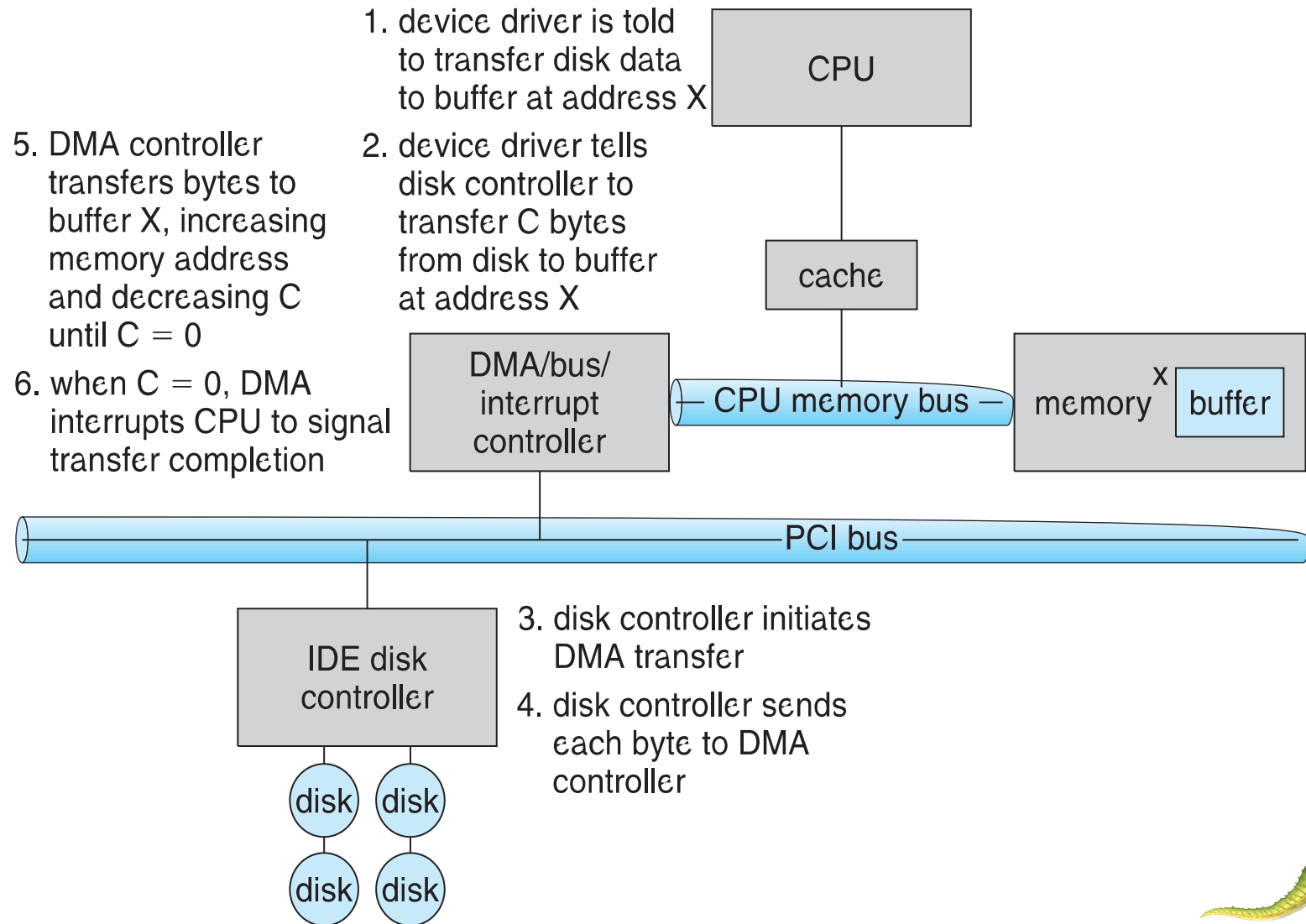
---

- Used to avoid **programmed I/O** (watch status bits, feed data into register one byte at a time) for large data movement
- Requires **DMA** controller
- Bypasses CPU to transfer data directly between I/O device and memory
- In detail, OS writes DMA command block into memory
  - Source and destination addresses
  - Read or write mode
  - Count of bytes
  - Writes location of command block to DMA controller (CPU leaves, DMA controller proceeds transfer)
  - When done, interrupts to signal completion





# Six Step Process to Perform DMA Transfer







# DMA方式的数据传送处理过程

区别①：中断方式时是在数据缓冲寄存器满之后发出中断要求CPU进行中断处理，而DMA方式则是在所要求传送的数据块全部结束时要求CPU中断处理，大大减少CPU进行中断处理的次数。

区别②：中断方式的数据传送是在中断处理时由CPU控制完成的，DMA方式是在DMA控制器的控制下不经过CPU控制完成的，这就排除了并行操作设备过多时CPU来不及处理或速度不匹配而造成数据丢失等现象。





# DMA控制方式

## ■ 优点

- 数据在内存和设备之间直接传送，CPU不干预。
- 数据的传输控制完全由DMA控制器完成，速度快，适合高速成组数据传输。
- 数据块在传输过程中，CPU与外设并行工作，比中断控制方式的并行性高。





# Application I/O Interface

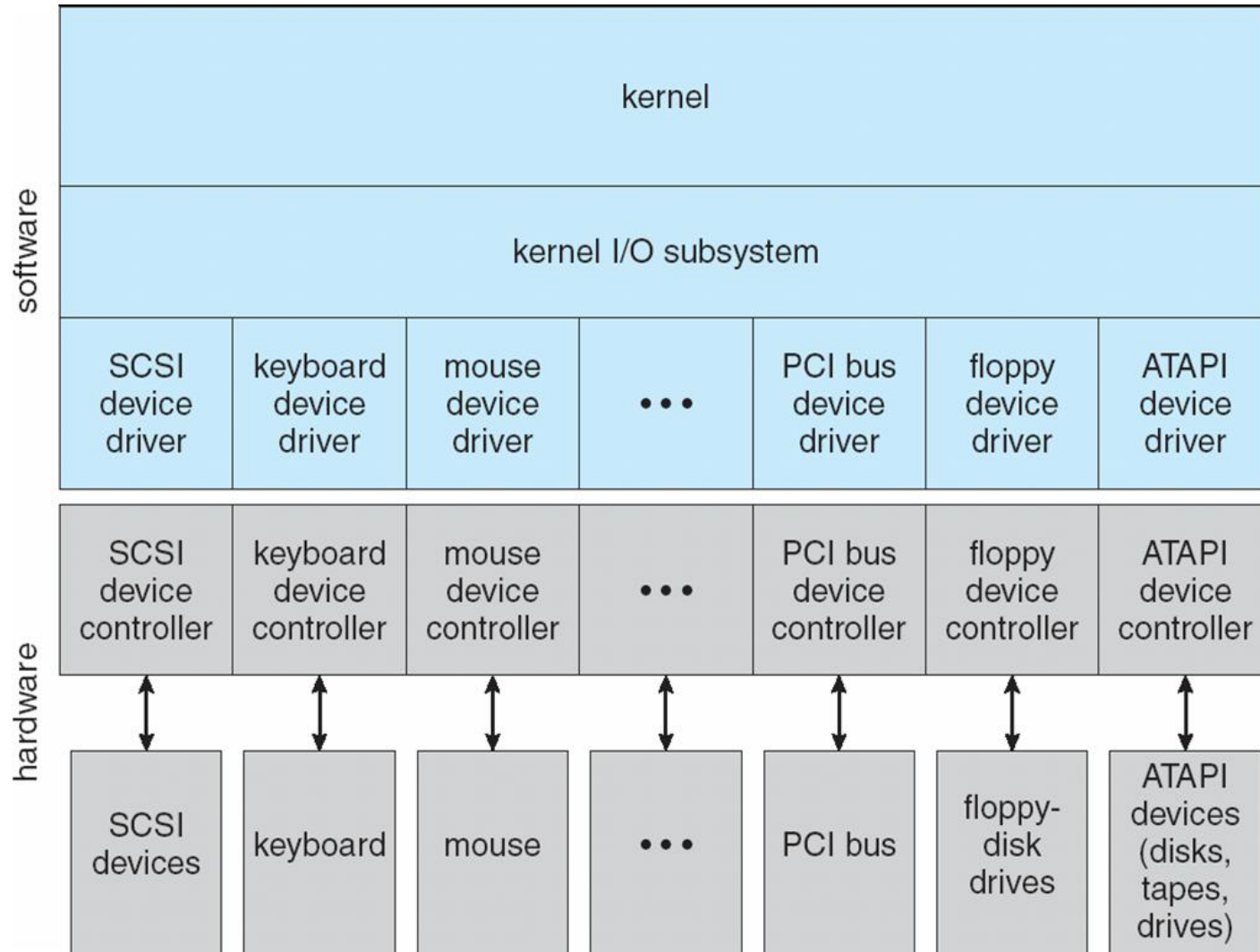
---

- 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**





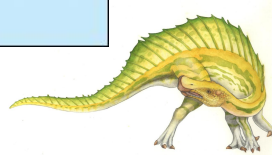
# A Kernel I/O Structure





# Characteristics of I/O Devices

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





# Block and Character Devices

---

- Block devices include disk drives
  - Commands include `read()`, `write()`, `seek()` interface
    - ▶ **Raw I/O** (application  $\leftrightarrow$  block devices, **no OS services provided**),
  - Memory-mapped file access possible
    - ▶ File mapped to virtual memory and clusters brought via demand paging (efficient)
- Character devices include keyboards, mice, serial ports
  - Commands include `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 (that is different with read/write/seek interface for disk)
  - Includes **select()** functionality
    - ▶ Manage a set of sockets, e.g., which has room to receive packets





# (Hardware) Clocks and Timers

---

- Provide current time, elapsed time, timer (to trigger operation)
- **Programmable interval timer** used for timings, periodic interrupts
  - Process scheduler uses this to generate an interrupt that preempt a process at the end of its time slice
  - Network subsystem uses this to cancel too slow operations
  - And so on
- Normal resolution about 1/60 second (interrupt rate)
- Some systems provide higher-resolution timers (more instructions)
- `ioctl()` (on UNIX) covers aspects of I/O such as clocks and timers







# Nonblocking and Asynchronous I/O

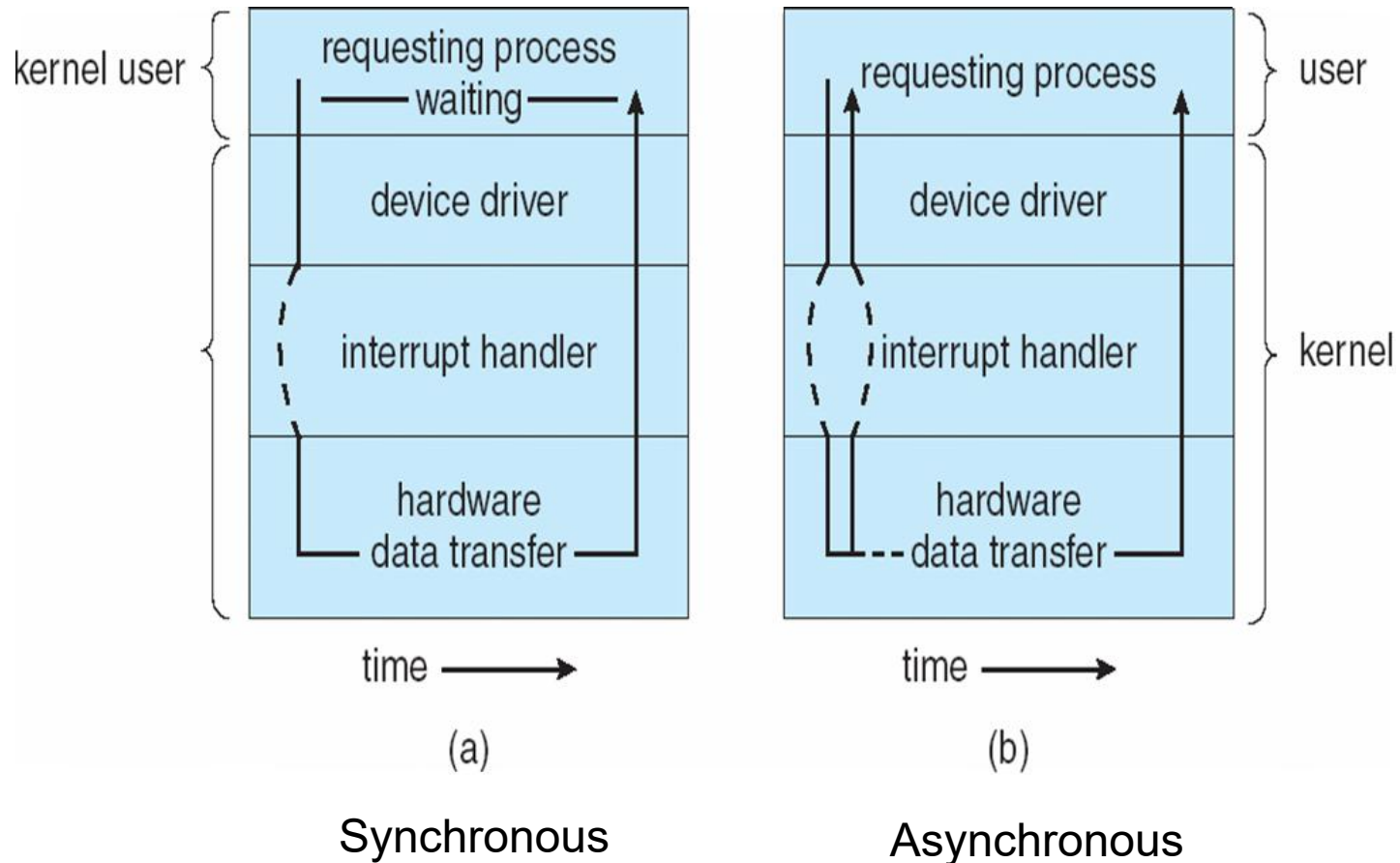
- **Blocking** - process suspended until I/O completed
  - Easy to use and understand
  - Insufficient for some needs
- **Nonblocking** - I/O call returns as much as available
  - User interface, data copy (buffered I/O)
  - Implemented via multi-threading
  - Returns quickly with count of bytes read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

**Difference between nonblocking and asynchronous:** Read()  
-- nonblocking: returns immediately as much data as possible;  
-- asynchronous: entire required data are returned





# Two I/O Methods





# Kernel I/O Subsystem

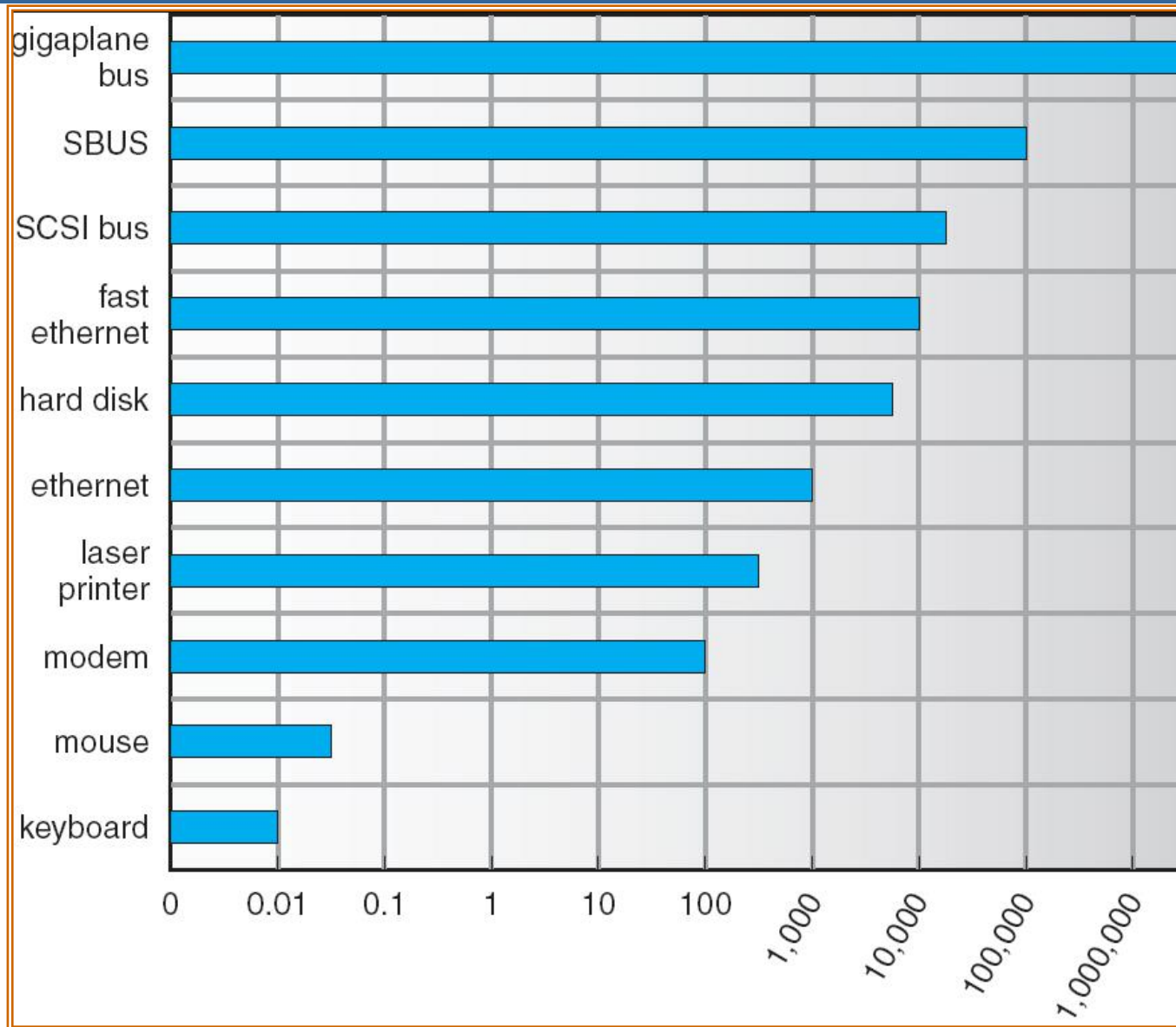
---

- Scheduling (I/O Scheduling)
  - Rearrange I/O request ordering via per-device (wait) queue
  - Some OSs try fairness (algorithms described in Sec. 12.4)
  
- **Buffering (缓冲)** - store data in **memory** while transferring between devices
  - To cope with device speed mismatch
  - To cope with device transfer size mismatch (msg -> small packets)
  - To maintain “copy semantics”
    - ▶ Guarantee buffer contents written to disk are before changes
  - **Double buffering (双缓冲)** – two copies of the data
    - ▶ Consider, e.g., modem data -> hard disk



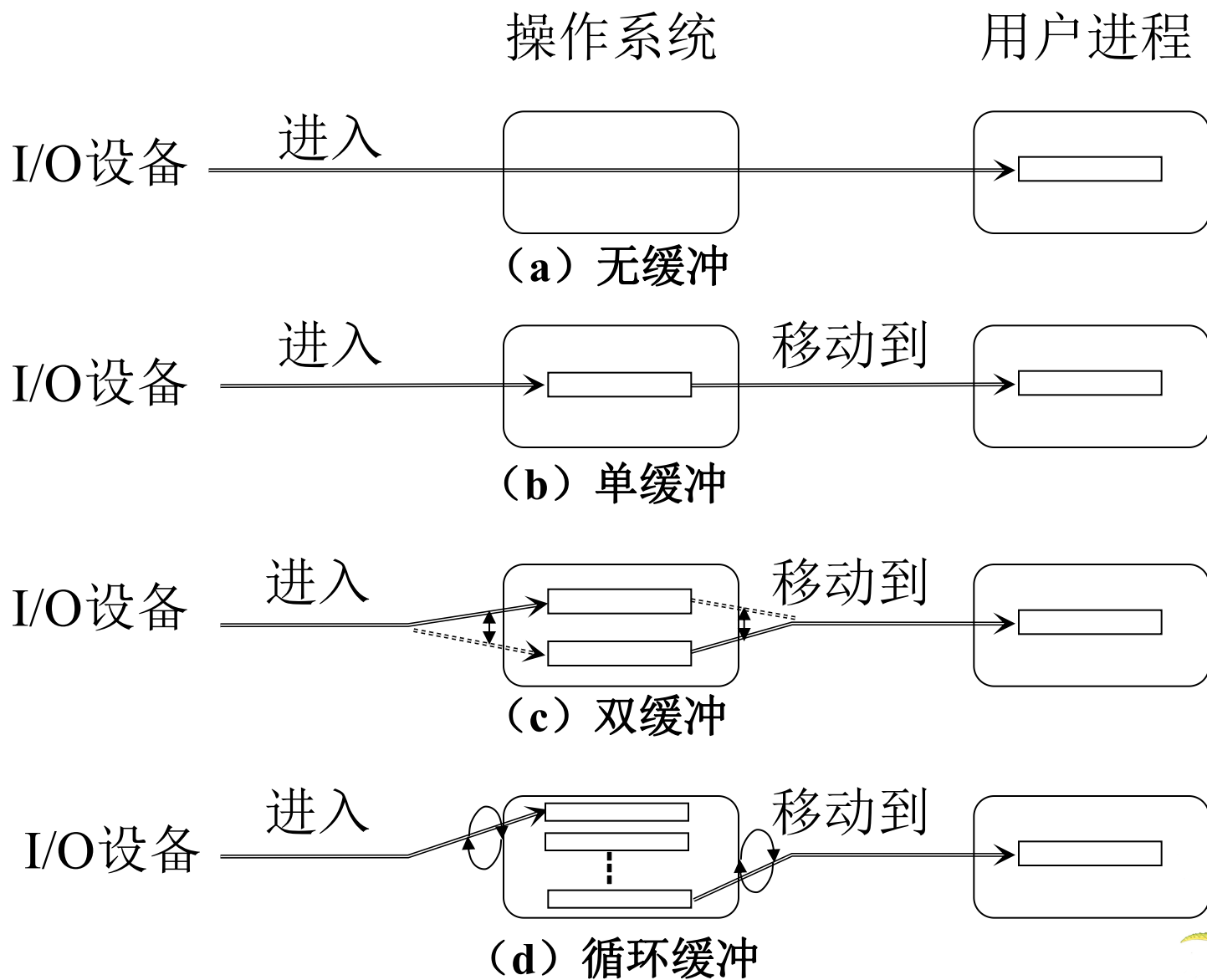


# Sun Enterprise 6000 Device-Transfer Rates





# 缓冲的种类



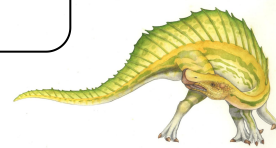
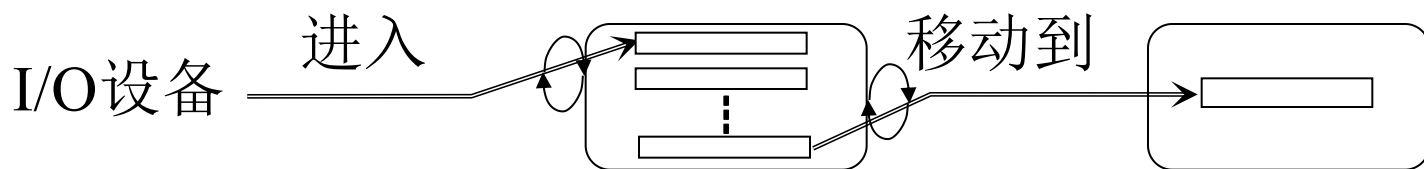


# 循环缓冲

- 双缓冲在并行时由于各自推进速度的差异使得并行受到很大限制；
- 可利用多缓冲来平滑计算与输入输出设备、输入设备与输出设备并行双方之间的数据流；
- 多缓冲通常组织成循环缓冲的形式。

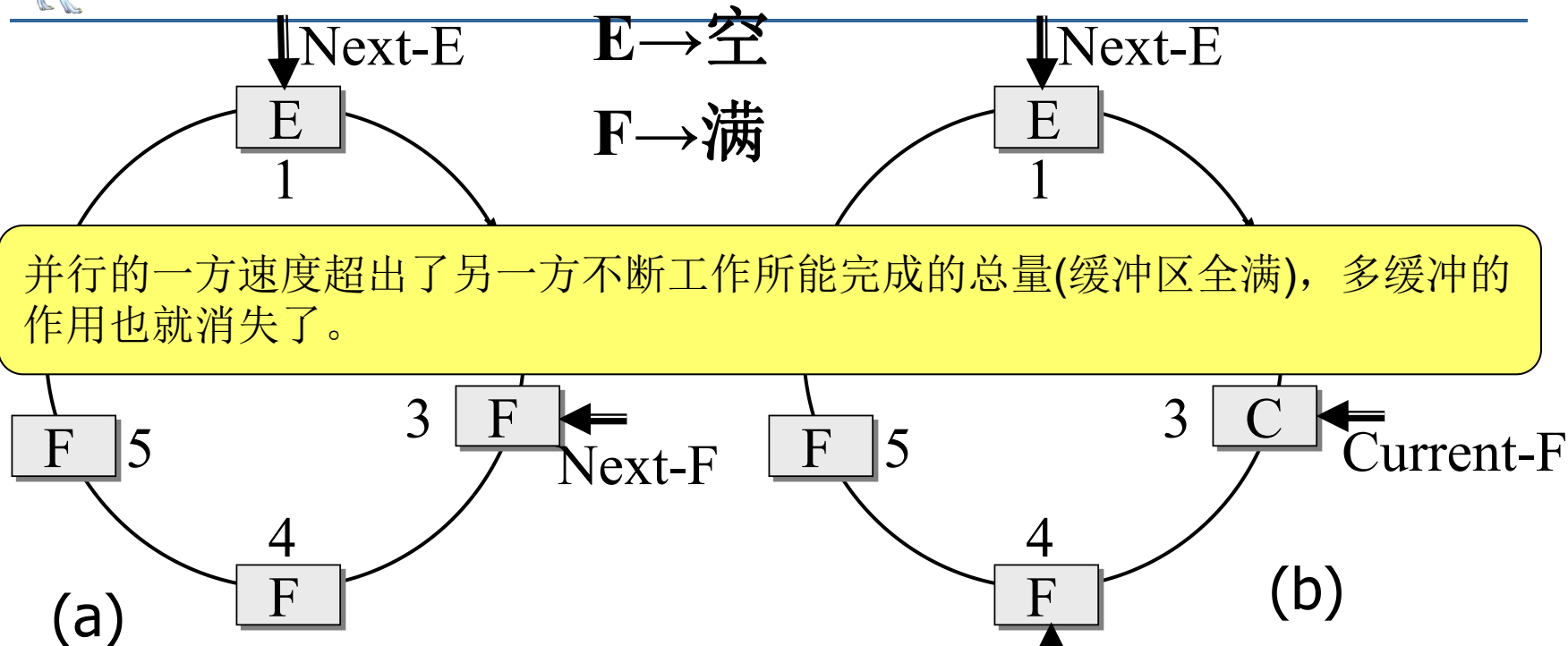
## ● 多缓冲组成

通常多个缓冲的大小相同。





# 循环缓冲

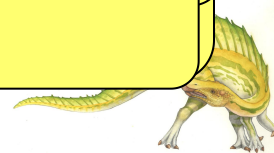


并行的一方速度超出了另一方不断工作所能完成的总量(缓冲区全满), 多缓冲的作用也就消失了。

输入与计算并行操作中, 应考虑两种同步情况:

**Next-E追上Next-F:** 意味输入进程输入数据速度大于计算进程处理数据速度, 已将全部缓冲区装满。这时阻塞输入进程, 这种情况称为系统受计算限制。

**Next-F追上Next-E:** 计算进程处理数据的速度大于输入进程的速度。这种情况称为系统受输入限制。





# Kernel I/O Subsystem

---

- **Caching (高速缓存)** - faster device (storage) holding copy of data
  - Always just a copy (while buffer contains original data)
  - Key to performance (disk => memory => cache (w/r by CPU))
  - Sometimes combined with buffering
- **Spooling (假脱机)** – buffer holding output for a device
  - If device can serve only one request at a time
  - i.e., Printing (output is firstly stored in a separate **disk** file)
  - Change exclusive devices into sharable devices
- **Device reservation (设备预留)** - 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  
(unfortunately, if an important permanent failure, unlikely to recover)
  - 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





# 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 instructions
  - I/O must be performed via system calls





# Kernel Data Structures

---

- Kernel keeps state info (data structure) 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?
      - Overhead compared with shared memory
      - Simplicity and flexibility





# Performance

---

- I/O a major factor in system performance:
  - Demands CPU to execute device driver
  - Context switches due to interrupts
  - Data copying
  - Network traffic (may cause high context-switch rate)





# Improving Performance

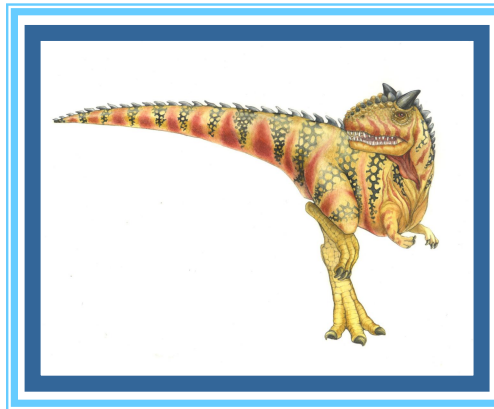
---

- Reduce number of context switches
- Reduce data copying (reduce copies in memory, while device  $\leftrightarrow$  app)
- Reduce interrupts by using large transfers, smart controllers, polling
- Use DMA
- Use smarter hardware devices (more processing inside devices)
- Balance CPU, memory, bus, and I/O performance for highest throughput



# End of Chapter 13

---





# 问题1：中断

## 简述中断处理过程

### ■ 保护现场

- 为了在中断处理完之后可以返回程序原来被中断的地方继续执行，系统需要保存中断响应时**CPU**的工作现场。

### ■ 分析原因

- 根据中断源确定中断原因，即根据中断号检索中断向量表，得到中断处理程序的入口地址。

### ■ 处理中断

- 核心调用中断处理程序，完成中断处理。

### ■ 中断返回

- 恢复中断寄存器内容和执行核心栈退栈，进程回到用户态，即返回断点。





- 缺页中断属于（ ）中断，Ctrl+C属于（ ）？
  - A、硬件故障中断      B、内中断
  - C、外部中断      D、自愿性中断
- 
- 外中断：来自处理器和内存外部的中断，包括I/O设备发出的IO中断、外部信号中断和各种定时器引起的时钟中断等；
  - 内中断：处理器和内存内部产生的中断，包括程序运行引起的各种错误，如地址非法、检验错、页面失效、存取访问控制错和算术操作溢出等。







## 问题3: spooling系统

- 为什么要引入spooling系统，可以带来什么好处？
- 对于慢速设备，一般都是独占设备，当一个进程使用该设备进行大数据量交换时，其他需要同时访问该设备的进程就需要等到较长时间，因此引入spooling系统，把独占设备改造成共享设备，即利用一台可共享的高速大容量块设备（磁盘）来模拟独占设备。
- 引入spooling技术后，提高了IO速度，将独占设备改造成共享设备，实现了虚拟设备功能，提高了系统的并行性，减少了用户进程的等待时间。





- CPU与DMA控制器可以并行执行，并通过（ ）实现彼此间的通讯和同步。
- A.I/O指令
- B.I/O中断
- C.I/O指令和I/O中断
- D.操作员
- 答案： C





- （）是CPU与I/O设备之间的接口，它接收从CPU发来的命令，并去控制I/O设备的工作，使处理机从繁杂的设备控制事务中解脱出来。
- A、中断装置
- B、通道
- C、逻辑
- D、设备控制器
- 答案： D





本地用户通过键盘登陆系统时，首先获得键盘输入信息的程序是\_\_\_\_\_。

A. 命令解释程序

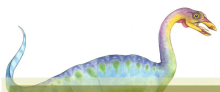
B. 中断处理程序

C. 系统调用服务程序

D. 用户登录程序

■ B



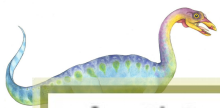


在关于 SPOOLing 的叙述中,\_\_\_\_\_描述是不正确的。(西安电子科技大学 2000 年研究生试题)

- A. SPOOLing 系统中不需要独占设备
- B. SPOOLing 系统加快了作业执行的速度
- C. SPOOLing 系统使独占设备变成共享设备
- D. SPOOLing 系统利用了处理器与通道并行工作的能力

■ A





.在采用 SPOOLing 技术的系统中,用户的打印数据首先被送到\_\_\_\_\_。

1999 年研究生试题)

A. 磁盘固定区域


B. 内存固定区域

C. 终端

D. 打印机

■ A





缓冲技术的缓冲池在\_\_\_\_\_中。

A. 主存

B. 外存

C. ROM

D. 寄存器

■ A





# Summary

---

The basic hardware elements involved in I/O are buses, device controllers, and the devices themselves. The work of moving data between devices and main memory is performed by the CPU as programmed I/O or is offloaded to a DMA controller. The kernel module that controls a device is a device driver. The system-call interface provided to applications is designed to handle several basic categories of hardware, including block devices, character devices, memory-mapped files, network sockets, and programmed interval timers. The system calls usually block the process that issues them, but nonblocking and asynchronous calls are used by the kernel itself and by applications that must not sleep while waiting for an I/O operation to complete.

The kernel's I/O subsystem provides numerous services. Among these are I/O scheduling, buffering, caching, spooling, device reservation, and error handling. Another service, name translation, makes the connection between hardware devices and the symbolic file names used by applications. It involves several levels of mapping that translate from character-string names, to specific







# Summary

---

device drivers and device addresses, and then to physical addresses of I/O ports or bus controllers. This mapping may occur within the file-system name space, as it does in UNIX, or in a separate device name space, as it does in MS-DOS.

STREAMS is an implementation and methodology for making drivers reusable and easy to use. Through them, drivers can be stacked, with data passed through them sequentially and bidirectionally for processing.

I/O system calls are costly in terms of CPU consumption, because of the many layers of software between a physical device and the application. These layers imply the overheads of context switching to cross the kernel's protection boundary, of signal and interrupt handling to service the I/O devices, and of the load on the CPU and memory system to copy data between kernel buffers and application space.

