

# IO Systems — Chapter 13

2022年12月

薛哲

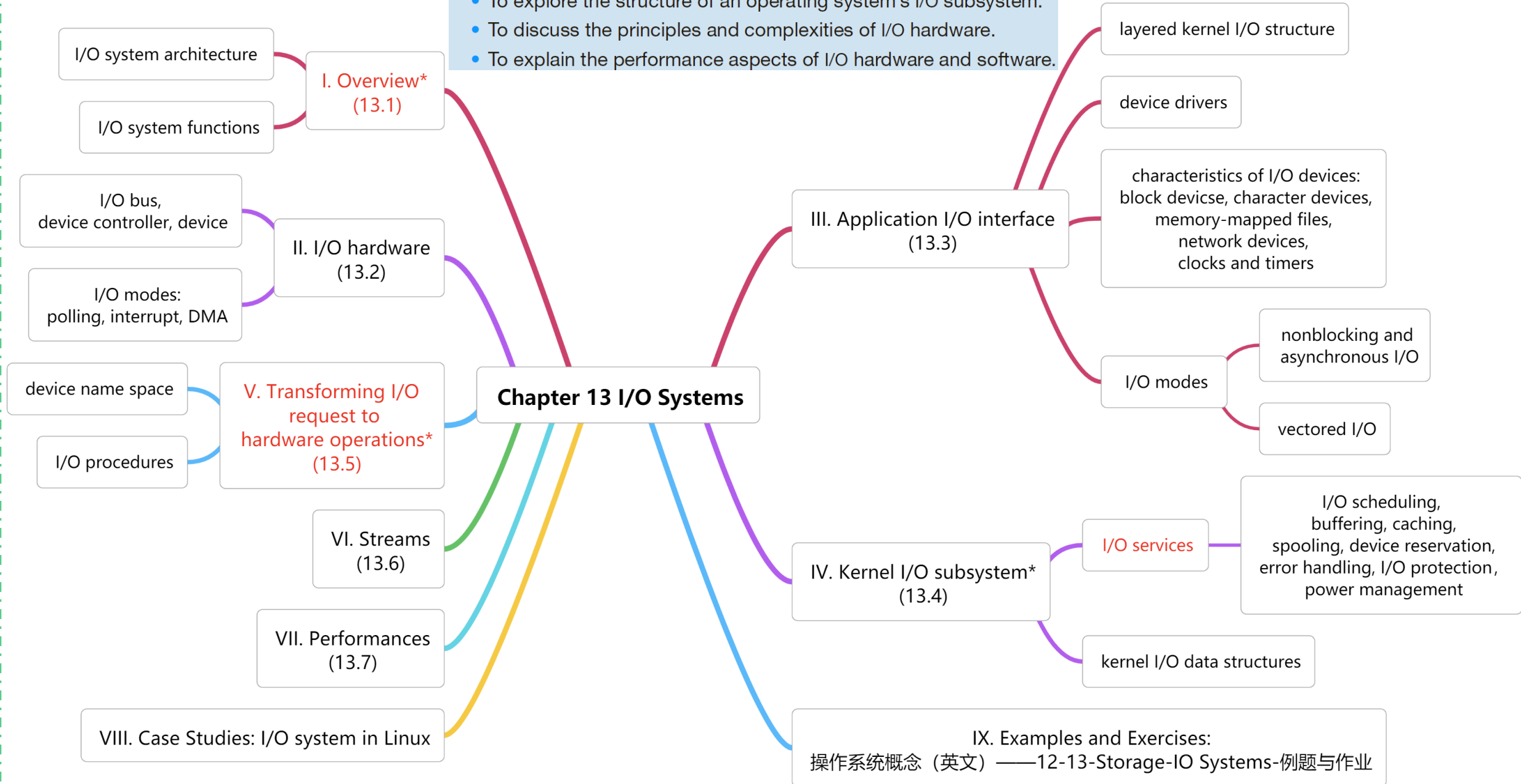
School of Computer Science (National Pilot Software Engineering School)



北京邮电大学

# CHAPTER OBJECTIVES

- To explore the structure of an operating system's I/O subsystem.
- To discuss the principles and complexities of I/O hardware.
- To explain the performance aspects of I/O hardware and software.



## 13.1 Overview

---

- I/O device (or devices)
  - ▣ hardware for input and output operations in computer systems, including
    - devices themselves and their controllers, e.g. 27' LCD + Nvidia, disk drive=disk + disk controller
- I/O system
  - ▣ hardware and software components for input and output
  - ▣ including
    - devices, device controllers, and **I/O subsystem**
- I/O subsystem
  - ▣ OS component to manage and control I/O operations and I/O devices in computer systems, including
    - kernel I/O subsystem, and drivers
- **Kernel** I/O subsystem
  - ▣ a part of I/O subsystem, residing in kernel space

# Overview

---

- I/O subsystem consists of
  - ▣ Kernel I/O subsystem
    - a component that conducts scheduling, buffering, caching, and spooling, error handling
    - a general device-driver interface, e.g. device switch table DSW in Unix
  - ▣ drivers for specific hardware devices
    - 位于内核空间, 如Linux; or
    - 位于用户空间, 如在采用微内核架构的OS中

# Linux驱动程序

---

- 在linux中，一切皆文件，以文件方式实现外设I/O操作
- 设备驱动程序被装载在内核空间中运行，不允许用户程序直接访问内核空间，只能通过文件接口访问，方法
  - ▣ 内核将设备驱动程序操作接口以文件接口的形式导出到用户空间
  - ▣ 为相应的设备在/dev目录下建立相应的操作接口文件
  - ▣ 自linux2.6及其之后的内核版本，内核在系统启动时还会创建sysfs文件系统，并将设备操作接口导出到/sys目录下

# Overview

---

- I/O system calls provided for users
  - ▣ request and release devices
  - ▣ read, write, reposition
  - ▣ get or set device attributes
  - ▣ logically attach (附着、挂载) or detach devices

# Overview

---

## ■ Objects of I/O subsystem

- ▣ providing methods for users to manage and control a variety of devices, by means of standardized services or application I/O interface (§13.3) /\* 通用性
  - user-oriented level
  - or: logical I/O subsystem, in some other textbooks ▶
- ▣ adapting the system to various devices, encapsulating/hiding the details and oddities of different devices, by means of **device drivers** (§13.3) /\* 适应性
  - hardware-oriented level
  - or: physical I/O subsystem

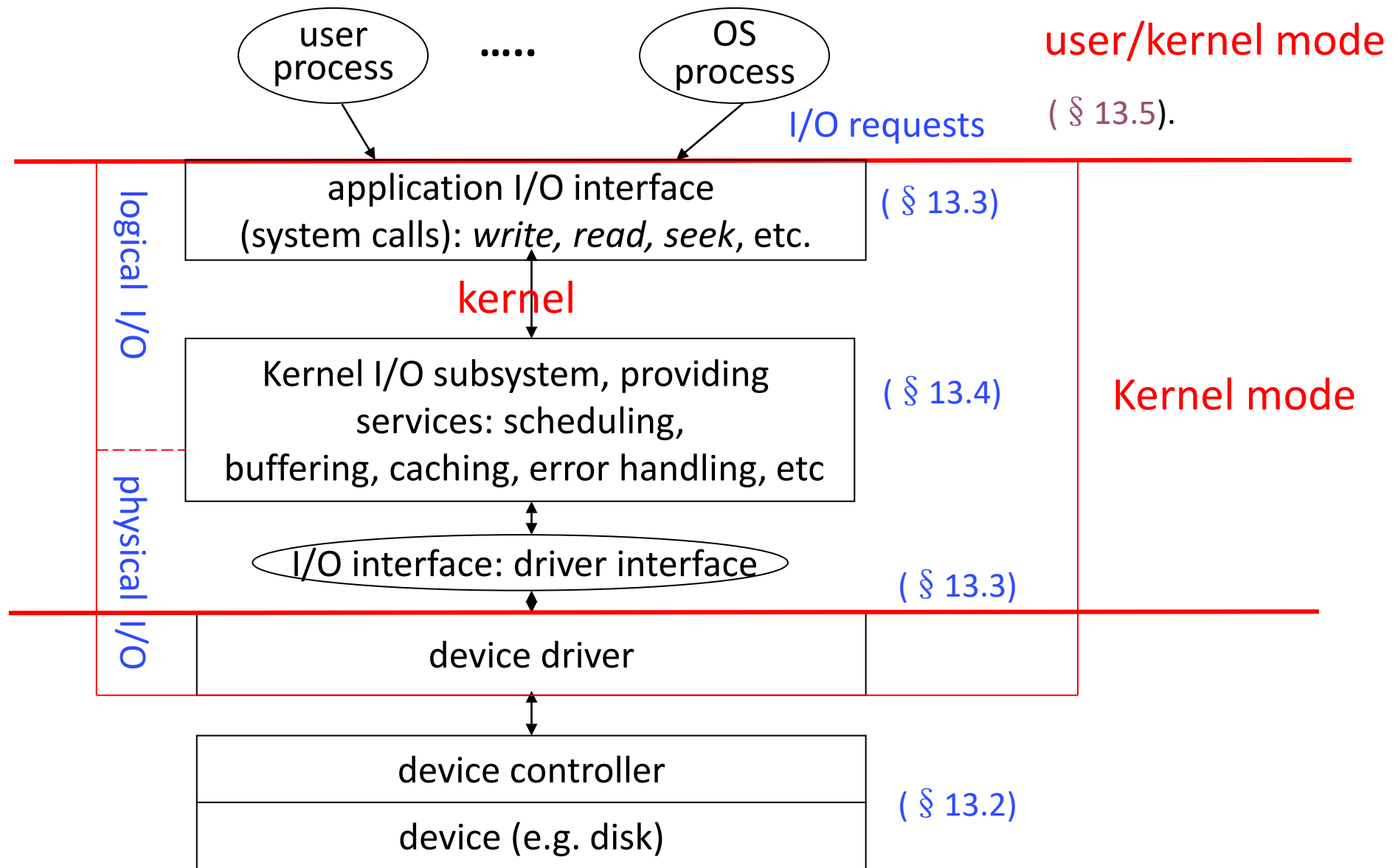


Fig. 13.0.1 I/O system architecture



- I/O系统（设备管理）功能

对计算机系统I/O操作进行管理，包括

- ▣ 选择和分配输入输出设备，以便进行数据传输操作
- ▣ 控制CPU与输入输出设备间数据交换
- ▣ 提高设备与设备间、设备与CPU间、进程与进程间并行操作程度
  - e.g. I/O调度
- ▣ 为用户提供友好编程接口，使用户应用程序同具体设备硬件特性分离，用户使用逻辑I/O设备

## 13.2 I/O Hardware

---

- Incredible variety of I/O devices
  - ▣ storage, e.g. disk
  - ▣ transmission, e.g. network
  - ▣ human-interface, e.g. keyboard
- Devices communicate with computer systems via port or bus, controlled by device controllers
- Common concepts – signals from I/O devices interface with computer
  - ▣ **Port** (端口)– connection point for device, e.g. a serial port
  - ▣ **Bus** (总线)- **daisy chain** or shared direct access
    - **PCI** bus common in PCs and servers, connecting the processor– memory subsystem to fast devices
    - **expansion bus** connects relatively slow devices, e.g. keyboard and serial and USB ports
    - PCI Express (**PCIe**), with throughput of up to 16 GB per second
    - **Small Computer System Interface (SCSI)** bus, connecting disks

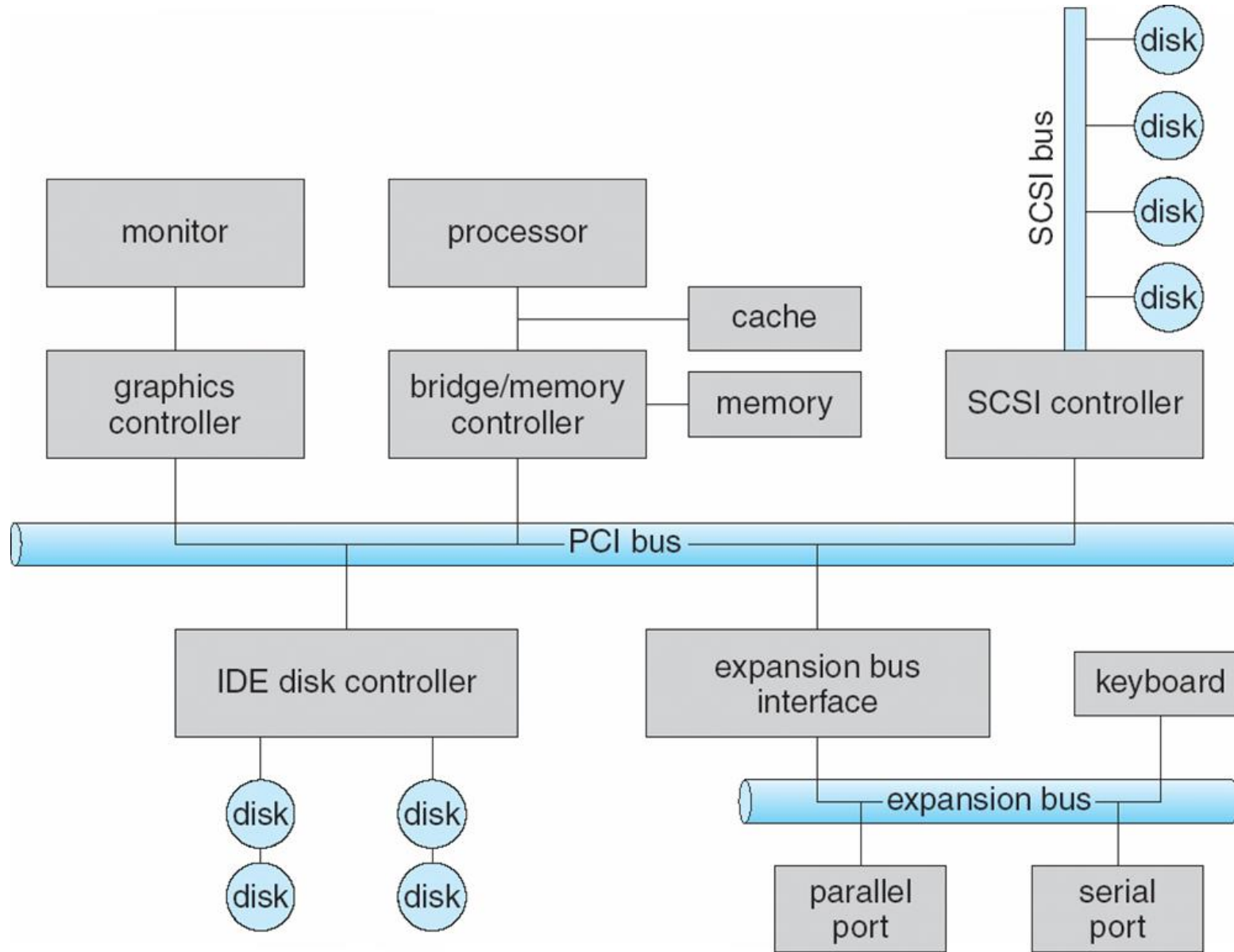


Fig. 13.1 A Typical PC Bus Structure

# Device Controller

---

- **Controller** (device controller, 设备控制器, **host adapter**)
  - ▣ electronics that operate port, bus, device, to
    - controlling basic I/O operations, and
    - transferring data between I/O devices and device controller registers
  - ▣ sometimes integrated, e.g. HDD; sometimes separate circuit board (host adapter), e.g. graphic card
  - ▣ contains processor, microcode, private memory, bus controller, etc
    - some talk to per-device controller with bus controller, microcode, memory, etc

# Device Controller

---

- The controller has one or more **registers** for **data and control signals**. Typically, a I/O device/port typically consists of four device controller registers
  - ▣ status register
  - ▣ control register
  - ▣ data-in register
  - ▣ data-out register
    - typically 1-4 bytes, or FIFO buffer
- CPU provides I/O instructions to control devices
- To give command and data to a controller to accomplish an I/O transfer, CPU communicates with the controller by reading and writing bit patterns in these registers
  - ▣ CPU runs I/O instructions in device drivers programs to read and write device registers

# Device Controller

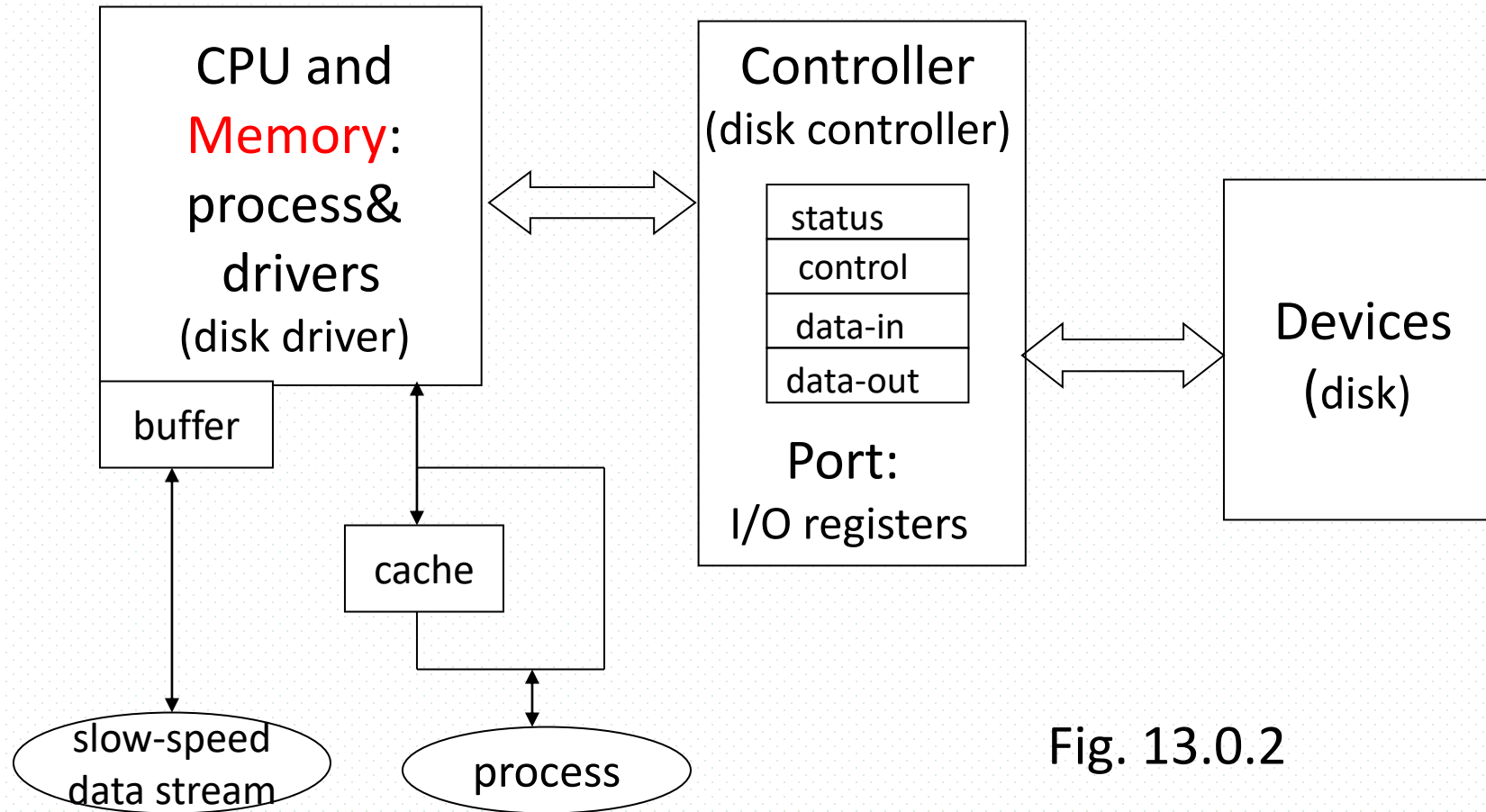


Fig. 13.0.2

# Device Controller

- Devices/ports have addresses, the processor control the device in two ways

- direct I/O instructions
  - CPU and drivers access the devices/ports by I/O instructions
- memory-mapped I/O
  - especially for large address spaces (graphics)

- Memory-mapped I/O

- data and control registers in controller are mapped into processor address space
  - e.g. mmap() in Linux
- CPU access controller registers by executing memory access instructions, e.g. LOAD A, X
- avoiding multiple buffering // 避免IO过程中多次缓存I/O数据, I/O效率高

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)

Fig. 13.2 Device I/O Port Locations on PCs

# I/O Hardware

---

- The main **control modes for data transfer** between memory/CPU and devices are:
  - ▣ polling (轮询/程序直接控制)
  - ▣ interrupt
  - ▣ direct memory access DMA



## 13.3 Application I/O Interface

---

- Layered kernel I/O structure
- Drivers
- Characteristics of I/O devices
- I/O modes
  - ▣ blocking and Nonblocking I/O
  - ▣ vectored I/O

# Layered Kernel I/O Structure

- I/O **system calls** encapsulate device behaviors in generic classes
  - ▣ e.g. `open()/release()`,  
`read()`, `write()`
- Device-**driver** layer hides differences among I/O controllers from kernel
  - ▣ e.g. device switch table DST
- New devices talking already-implemented protocols need no extra work
- Each OS has its own I/O subsystem structures and device driver frameworks

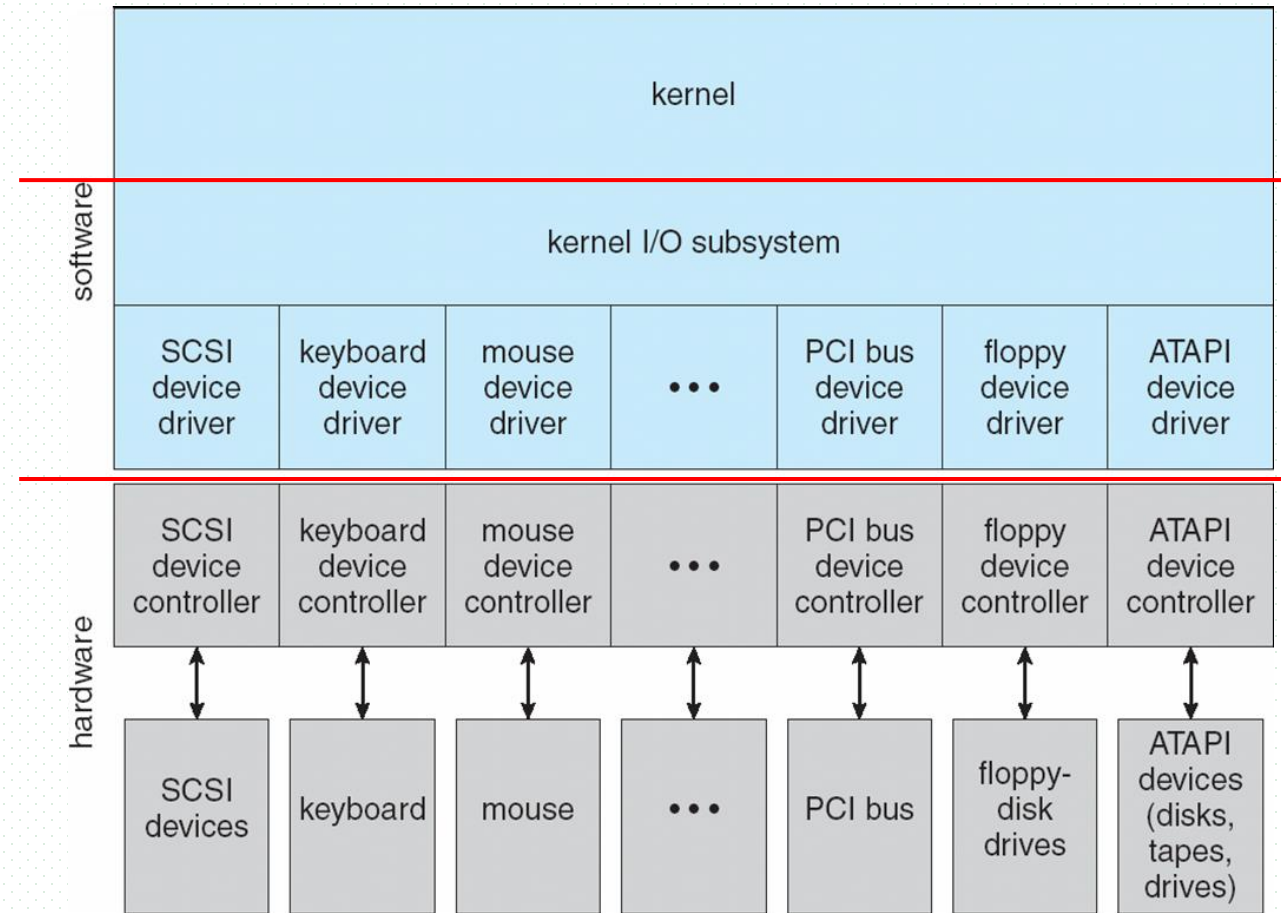


Fig. 13.6 A Kernel I/O Structure

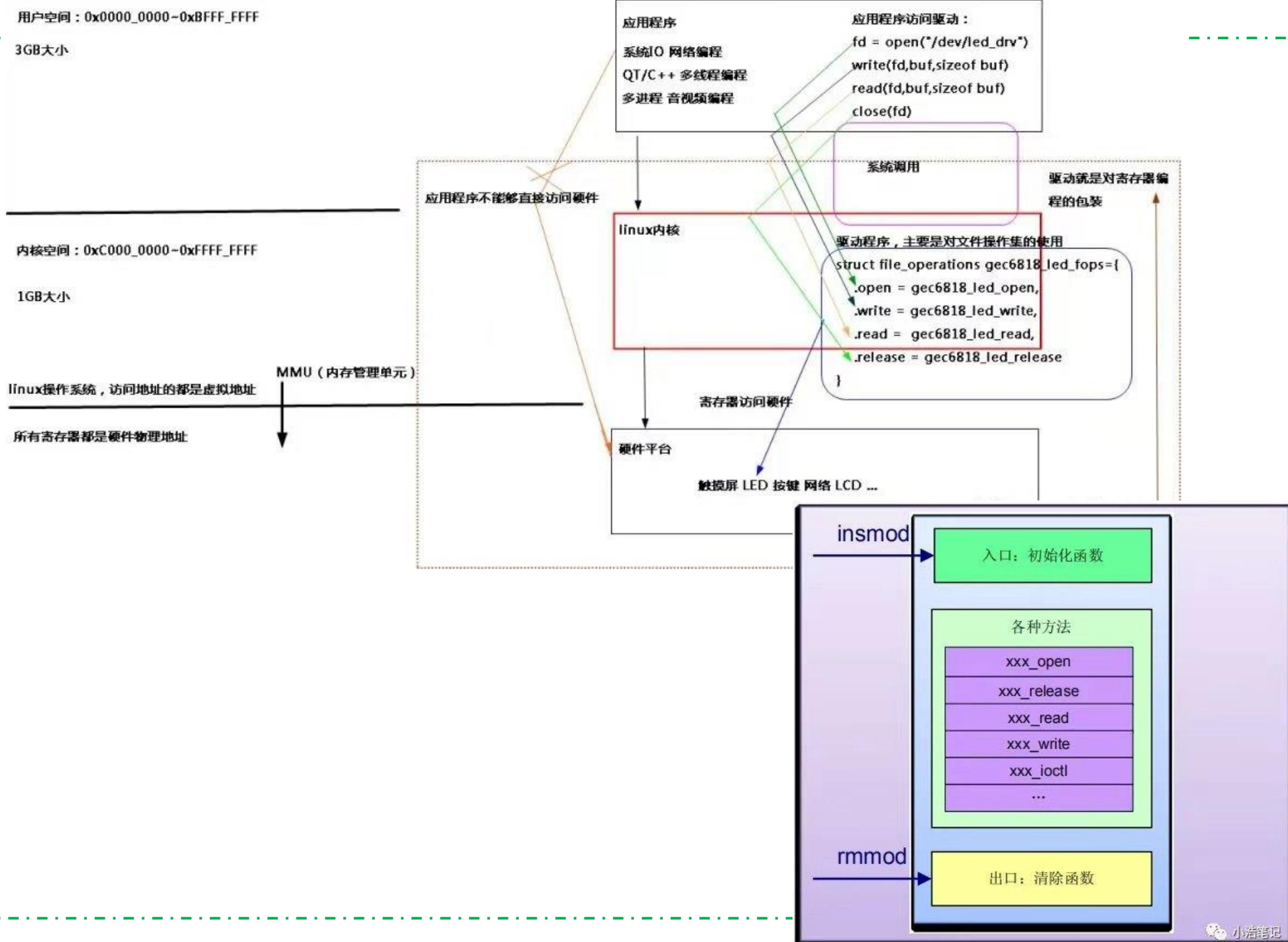
## ■ Driver

- ❑ 驱动物理设备、DMA控制器、设备控制器等直接进行I/O操作的机器指令程序集合，由CPU执行，完成对具体I/O操作的管理和控制功能
- ❑ 负责
- ❑ 设备初始化，如设置相应设备/设备控制器中有关寄存器的值、指定操作的类型（读、写）和数据流向
- ❑ 启动设备进行I/O操作、管理I/O请求队列
- ❑ 监测和处理设备出现的错误

## ■ CPU执行drivers程序,

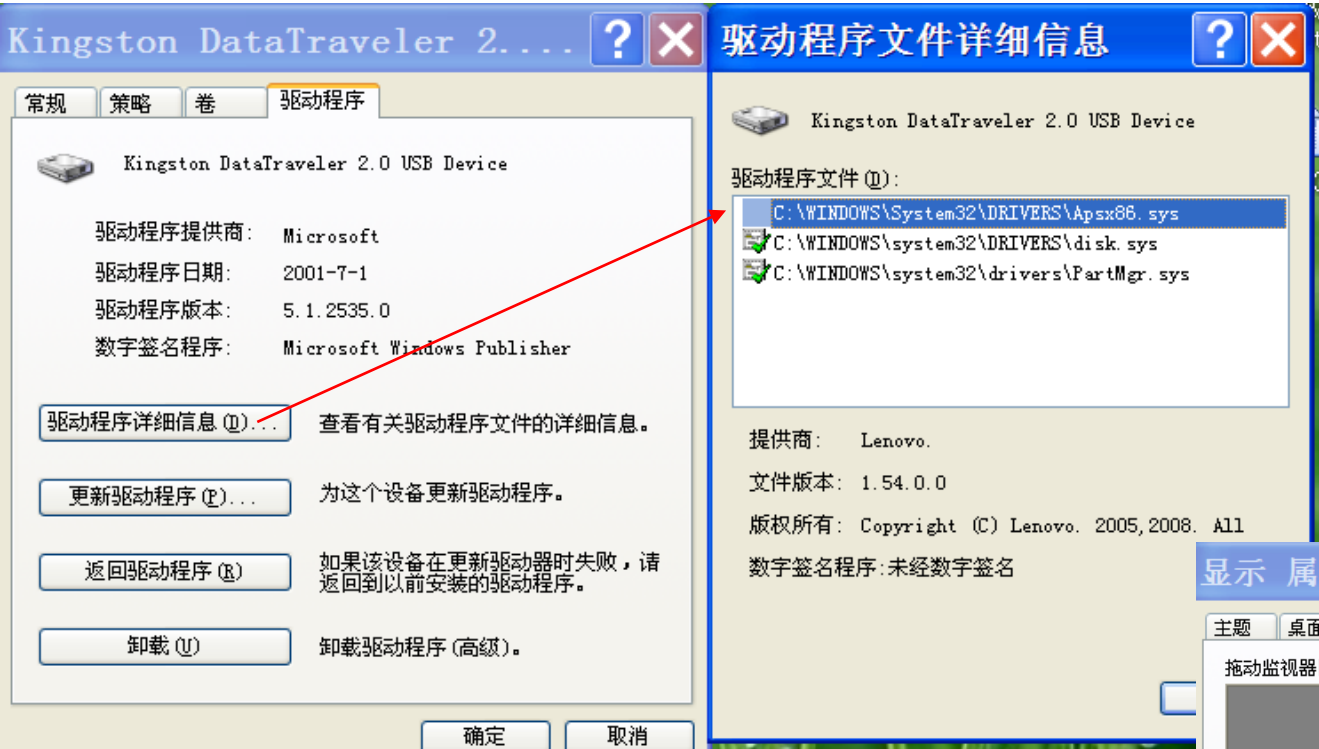
- ❑ 向device controller的control registers中写入读写命令和参数，控制IO过程
- ❑ 读取status registers，监测IO过程
- ❑ 通过从data-in registers读入数据、向data-out registers写入数据，完成具体的输入、输出操作

## ■ Example



■ Driver in Windows

U盘



显卡、显示器



ThinkPad Display 1440x900...

Intel(R) GMA Driver for Mobile

常规适配器监视器疑难解答颜色管理

适配器类型

Mobile Intel(R) 965 Express Chipset Family

属性(P)

适配器信息

芯片类型: Mobile Intel(R) 965 Express Chipset

DAC 类型: Internal

内存大小: 384 MB

适配器字符串: Mobile Intel(R) GMA X3100

BIOS 信息: Intel Video BIOS

Mobile Intel(R) 965 Expr...

常规驱动程序资源

Mobile Intel(R) 965 Express Chipset Family

设备类型: 显示卡

制造商: Intel Corporation

位置: PCI 总线 0、设备 2、功能 0

设备状态

这个设备运转正常。

若此设备有问题，单击“疑难解答”来启动疑难解答。

疑难解答(T)...

设备用法(U):

显卡

Mobile Intel(R) 965 Expr...

常规驱动程序资源

Mobile Intel(R) 965 Express Chipset Family

资源设置(R):

资源类型	设置
内存范围	E0000000 - EFFFFFFF
输入/输出范围	1800 - 1807
中断请求	16

设置基于(B):

☒ 使用自动设置(U)

更改设置(C)...

冲突设备列表:

没有冲突。

Mobile Intel(R) 965 Expr... 驱动程序文件详细信息

常规驱动程序资源

Mobile Intel(R) 965 Express Chipset Family

驱动程序提供商: Intel Corporation

驱动程序日期: 2007-8-9

驱动程序版本: 6.14.10.4860

数字签名程序: Microsoft Windows Hardware Compatibility

驱动程序详细信息(I)...

更新驱动程序(E)...

返回驱动程序(R)

卸载(U)

驱动程序文件(I):

- C:\WINDOWS\system32\DRIVERS\igxmp32...
- C:\WINDOWS\system32\hccutils.dll
- C:\WINDOWS\system32\hkcnd.exe
- C:\WINDOWS\system32\ig4dev32.dll
- C:\WINDOWS\system32\ig4icd32.dll
- C:\WINDOWS\system32\igfxcfg.exe
- C:\WINDOWS\system32\igfxCoIn\_v4860.dl

提供商: Intel Corporation

文件版本: 6.14.10.4860

版权所有: Copyright (c) 1998-2006 Intel

数字签名程序: Microsoft Windows Hardware



# Characteristics of I/O Devices

- Devices can be classified in many dimensions, refer to Fig.13.7
  - ❑ character-stream device or block device
  - ❑ sequential or random-access
  - ❑ synchronous or asynchronous
  - ❑ sharable or dedicated
  - ❑ speed of operation
  - ❑ 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

Figure 13.7 Characteristics of I/O devices

# Characteristics of I/O Devices

---

- 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 and Character Devices

---

- Block devices
  - transfer a block (or a cluster/chunk) of bytes as a unit
  - applications access block devices through three schemes
  - **file-system interfaces**
    - e.g. file access commands include **read**(fd, buffer, nbytes), **write**(fd, buffer, #block) and **seek**( ), only for random access devices
  - **direct I/O**
    - memory-mapped file access possible
    - file mapped to virtual memory and clusters brought via demand paging
  - **Raw I/O**
    - in database management system, access a block device as a simple linear array of blocks
- Block devices include disk and other block-oriented devices

# Block and Character Devices

---

- Character-stream devices
  - ▣ transfer bytes one by one
  - ▣ commands include `get()`, `put()`
  - ▣ on top of this interface, libraries can be built that offer *line-at-one-time access*, with buffering and editing services
- Character-stream devices include *keyboards*, mice and *serial ports*

# 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()`
    - `poll( )`, `epoll( )`
- Approaches vary widely (pipes, FIFOs, streams, queues, mailboxes)

# Clocks and Timers

---

- Provide current time, elapsed time, timer
- Normal resolution about 1/60 second, and some systems provide higher-resolution timers
- **Programmable interval timer** used for timings, periodic interrupts
  - ▣ the hardware to measure elapsed time and generate interrupt (once or periodically) to trigger some operations
  - ▣ e.g. **ticker** in some systems
- Applications
  - ▣ OS provides interface for users to use timers
  - ▣ I/O subsystem uses programmable interval timer to invoke the flushing of dirty cache buffers to disk periodically

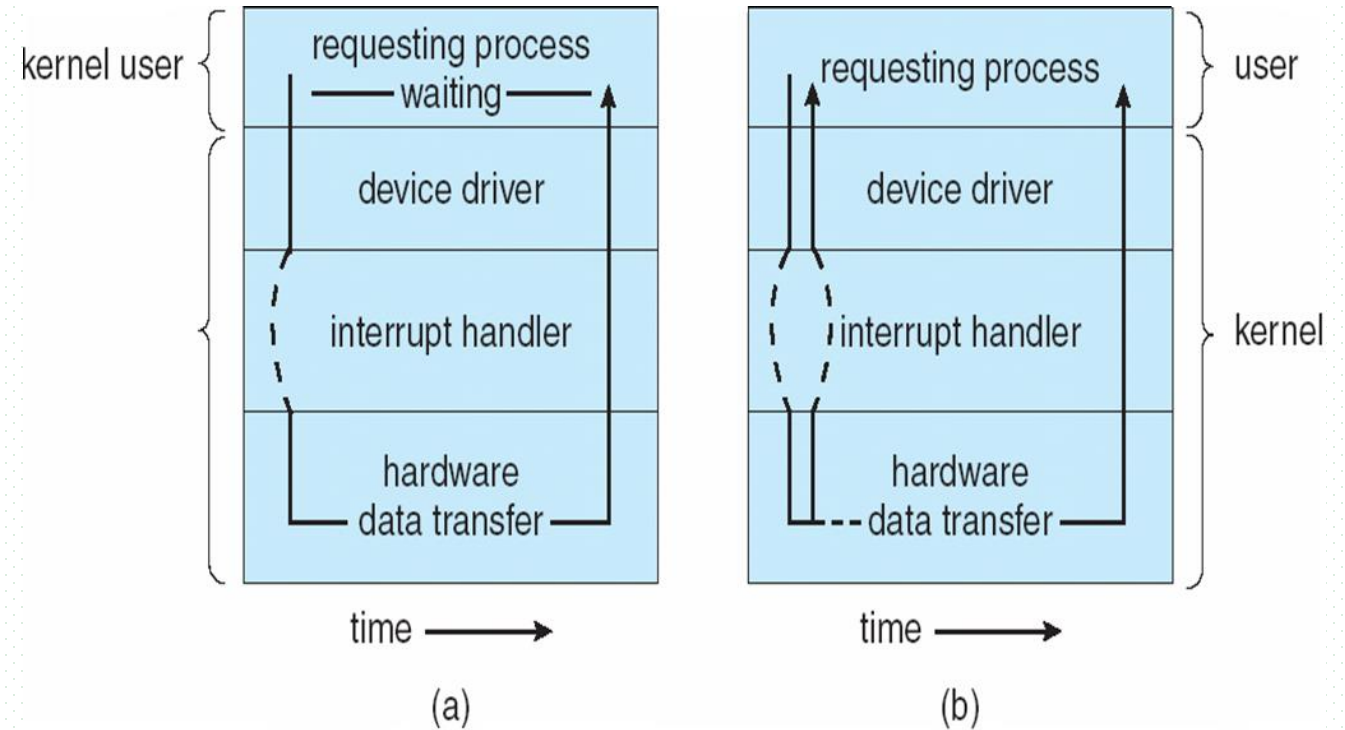
# Nonblocking and Asynchronous I/O

---

- Blocking I/O
  - ❑ the process requesting the I/O operation are suspended, until I/O completed
  - ❑ easy to use and understand for applications, but insufficient for some needs
  - ❑ used in most operating systems
- Nonblocking I/O
  - ❑ the process issues the I/O request is not suspended, and can continue its execution
  - ❑ implemented via multi-threading
  - ❑ e.g. user interface that receives keyboard and mouse input while processing and displaying data on the screen
  - ❑ `select()` to find if data ready, then `read()` or `write()` to transfer

# Nonblocking and Asynchronous I/O

- Asynchronous I/O system call as an alternative to nonblocking I/O system calls
  - ❑ a process conducts a asynchronous I/O system call, the system call returns immediately, without waiting for I/O completion
  - ❑ the process then runs while I/O executes, and I/O subsystem will signal process when all I/O completed
  - ❑ refer to Fig. 13.0.3 (b)



Synchronous

Asynchronous

# Vectored I/O

---


- **Vectored I/O** allows one system call to perform multiple I/O operations
  - ▣ //一个系统调用完成多路IO操作
- This scatter-gather method enables transferring contents of I/O multiple separate buffers via one system call
- For example, Unix **readve ()** accepts **a vector of multiple buffers** to read into or write from
- It is 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

## 13.4 Kernel I/O Subsystem

### ■ I/O scheduling

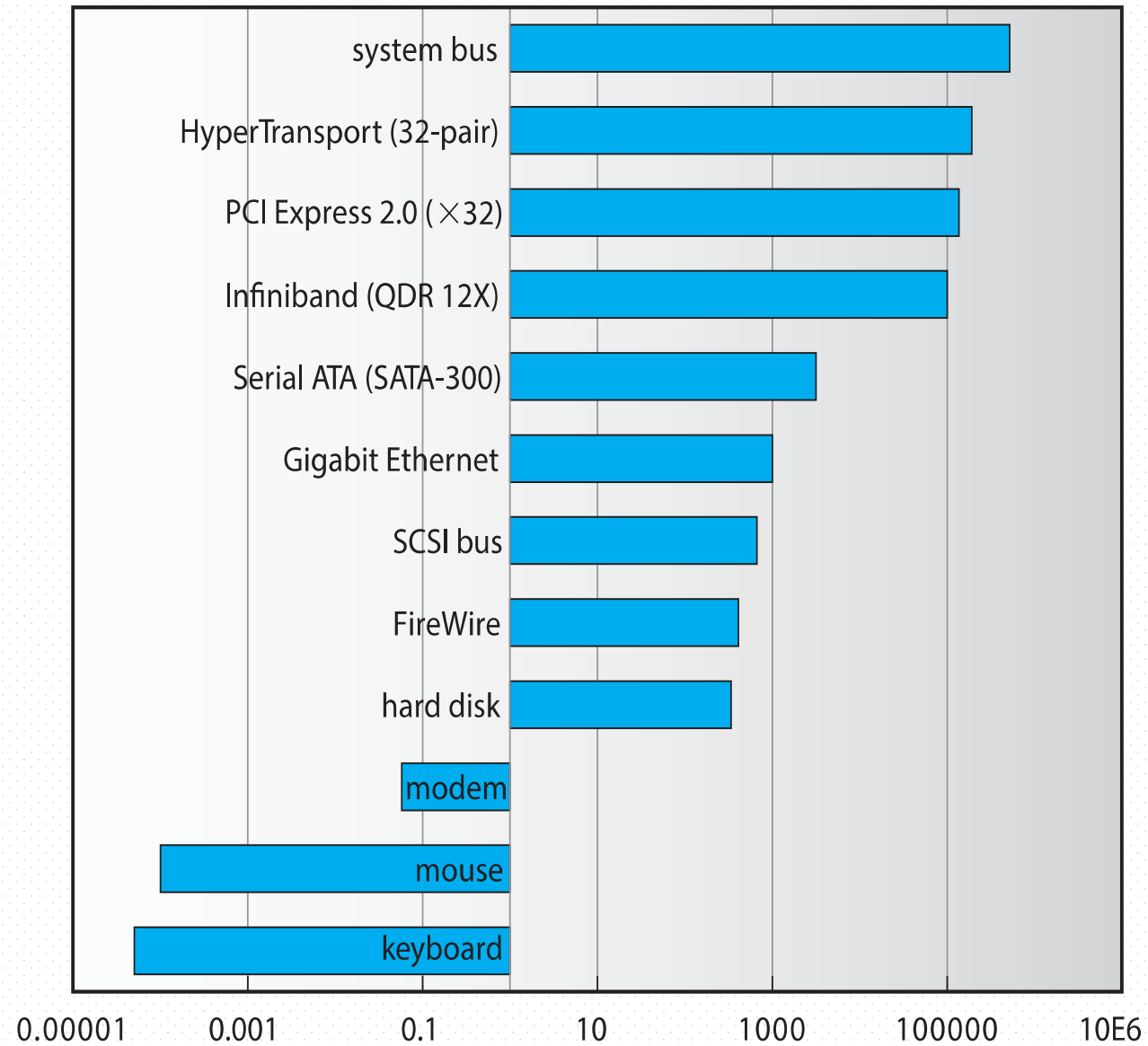
- ❑ determine the order in which more than one I/O requests are executed
- ❑ objectives
  - optimize overall system performance, e.g. fairness of device access among process, reduce average waiting time for I/O to complete
- ❑ implementation
  - each I/O request is ordered in **per-device queue**

### ■ Buffering - store data in memory while transferring between devices

- ❑ to cope with device speed mismatch with that of CPU memory access 
  - 磁盘外部传输速率, 磁盘内部传输速率
- ❑ to cope with device transfer size mismatch
  - e.g. fragmentation and reassembly in computer network
- ❑ to maintain "copy semantics" for application I/O
  - e.g. IP packet = IP head + IP payload



# Device-Transfer Rates



# Kernel I/O Subsystem

---

## ■ Caching

- ❑ holding copy of data in a faster device, i.e. cache
  - cache: a region of fast memory that holds copies of data
- ❑ access to cached data is more efficient than access to the original data on disks
- ❑ always just a copy in cache
- ❑ a region of memory may be used as both cache and buffer

## ■ Spooling - hold output for a device

- ❑ if device can serve only one request at a time
- ❑ i.e., Printing

## ■ Device reservation(预留) - provides exclusive access to a device

- ❑ system calls for allocation and de-allocation
- ❑ Watch out for deadlock

# Spooling (假脱机)

- Spool: **S**imultaneous **p**eripheral(辅助, 外围) **o**peration **o**n **l**ine
  - ▣ a spool is a buffer that holds output for a device, such as a printer, that can not accept *interleaved* data stream
  - ▣ to store a data document in a queue, while it awaits its turn to be printed, e.g. 打印机队列
- Spooling
  - ▣ OS中提供的一种设备管理机制
  - ▣ 将一台独占型设备(如打印机)模拟为多台共享设备, 使多个用户可同时并发使用独占型设备, 从而提高设备利用率和系统效率
  - ▣ 包括 input spooling, output spooling

# Spooling (假脱机)

- 采用output spooling的打印机
  - ▣ 当一个进程要求打印输出时，output spooling并不是将某台打印机分配给该进程，而是在磁盘上的**output spooling 存储区**中为其分配一块**独立**存储空间，为该进程的输出数据建立一个**输出数据文件**并存放在分配的存储空间内
  - ▣ output spooling 存储区中的所有打印输出数据文件构成**spooling 目录**
  - ▣ 此**输出数据文件**对该进程而言相当于虚拟打印机。各进程的打印输出都以输出文件形式存放在output spooling 存储区并行成打印输出队列。
  - ▣ 由output spooling 中的守护进程(daemon)控制过程，**依次**将输出队列中的各打印文件实际打印输出

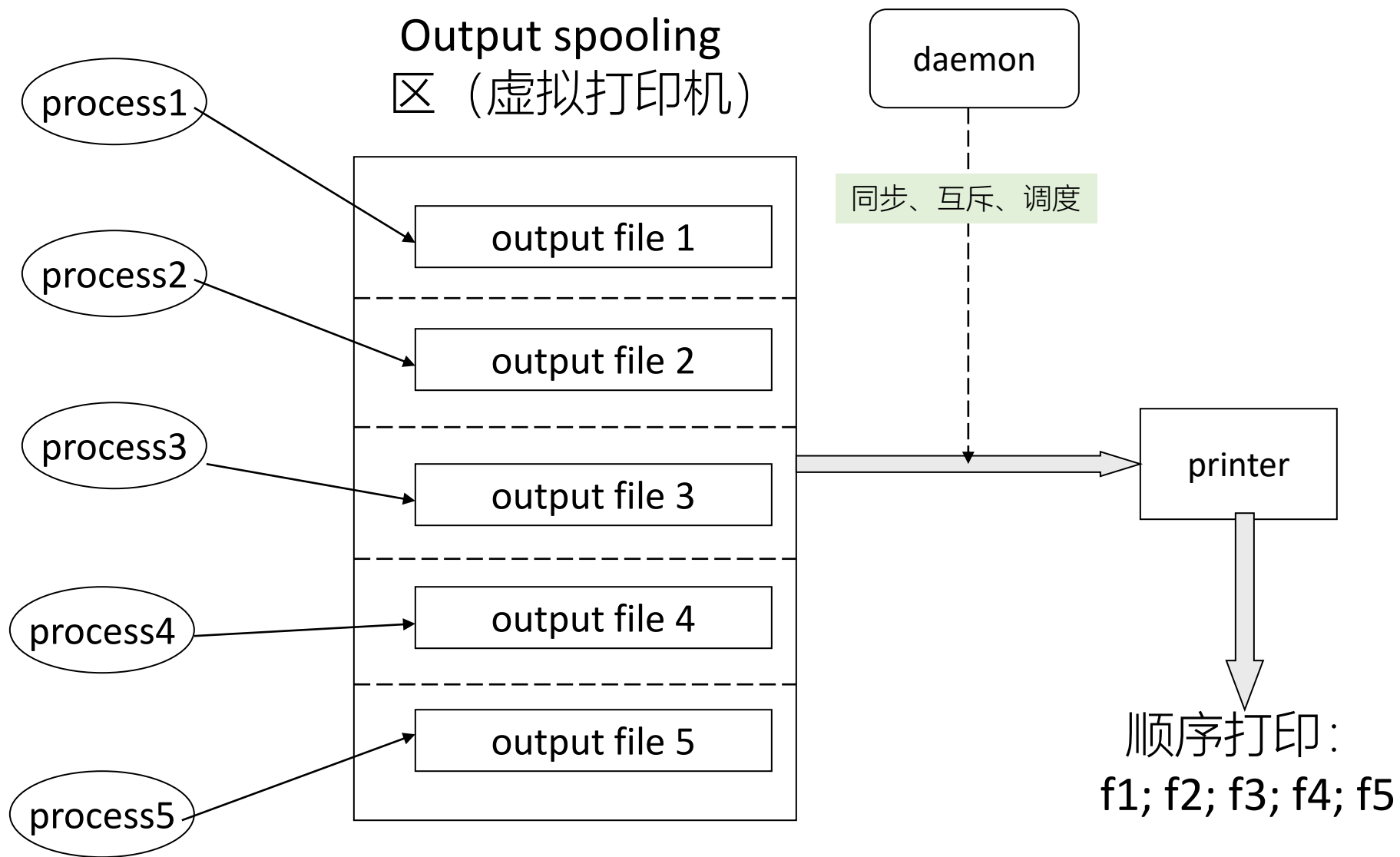


Fig. 13.0.5 Output spooling机制

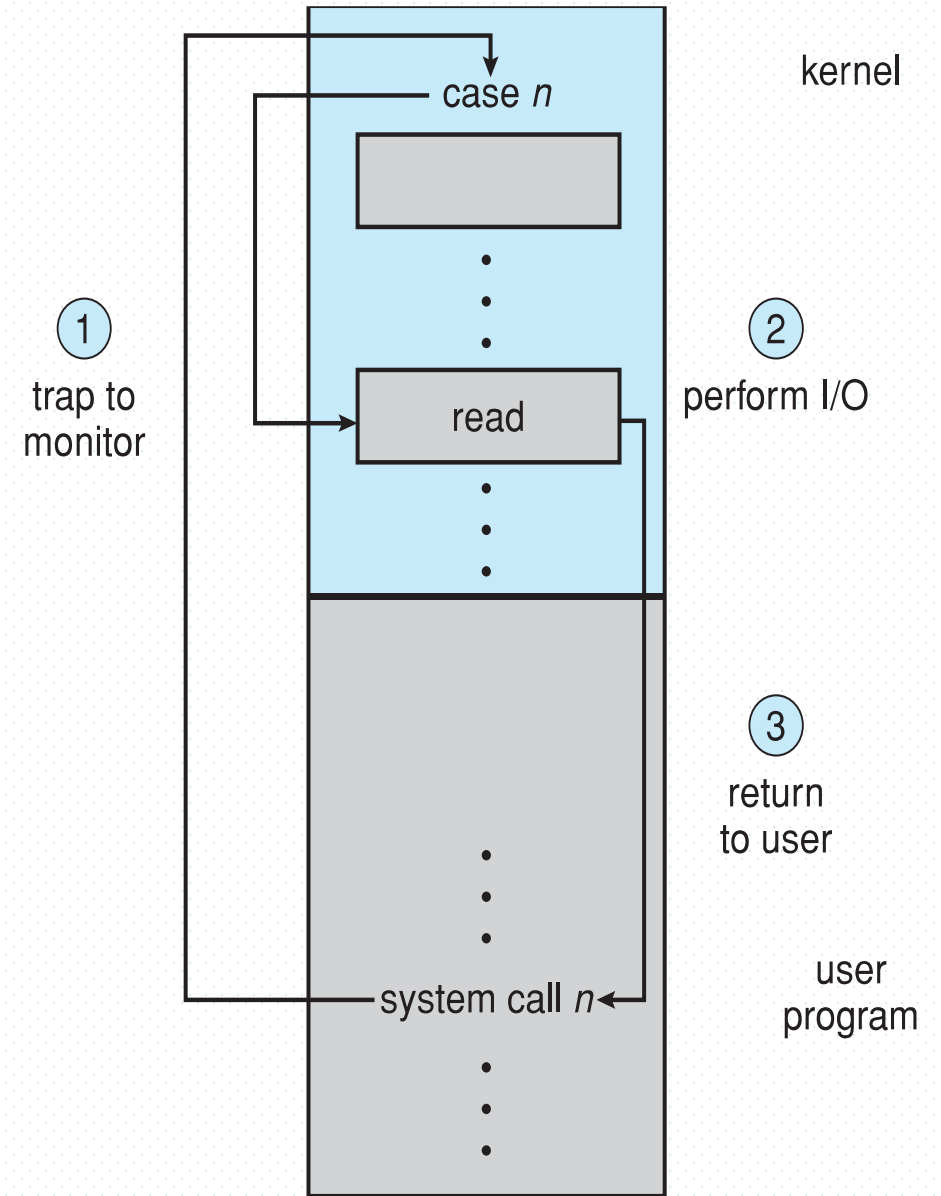
# Error Handling

---

- 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

# 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

# Power Management

---

- Not strictly domain of I/O, but much is I/O related
- Mobile computing has power management as first class OS aspect
- Computers and devices use electricity, generate heat, frequently require cooling
- OS can help manage and improve use
  - ▣ Cloud computing environments move virtual machines between servers
    - Can end up evacuating whole systems and shutting them down



# 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
  - ▣ 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?

# Functionalities of kernel I/O subsystem

---

- I/O subsystem coordinates an extensive collection of I/O services that are available to applications and to other parts of the kernel, such as
  - ▣ management of the name space for devices and files
  - ▣ access control to files and devices
  - ▣ operation control ( e.g. a modem cannot seek() )
  - ▣ file system space allocation
  - ▣ device allocation
  - ▣ I/O scheduling
  - ▣ buffering, caching, spooling
  - ▣ device status monitoring, error handling, and failure recovery
  - ▣ device driver configuration and initialization

## 13.5 Transforming I/O Requests to Hardware Operations

---

- Device Name Space
- Life Cycle of An I/O Request
  - ▣ I/O procedure

# Device Name Space

---

- 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

# Device Name Space

- When an application wants to access a file, it refers to the data by the file name, and the file system maps from the file name through the file system directories to the disk controller to obtain the disk space allocation of the file, e.g. `read(f1, buffer, nbytes)`
  - ▣ the file to be accessed is located in a device, such as disk, so the file name must be mapped to `disk controller` to conduct disk file I/O operations,  
e.g. `read(disk-controller, buffer, nbytes)`
- Devices or device controllers in the system should be named
  - ▣ device naming maps or connects `logical device names` (file names) to physical device controller

port\_address

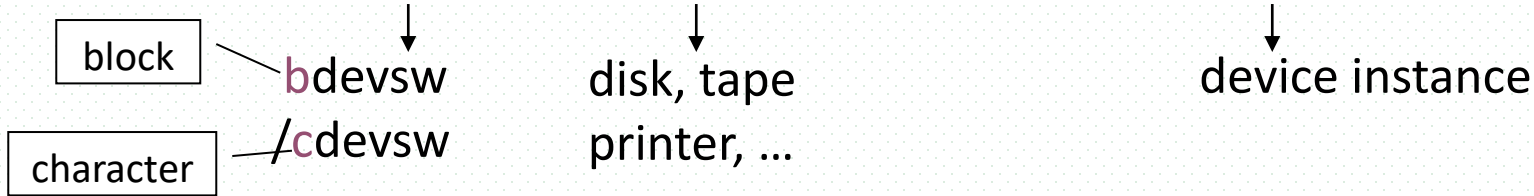
# Device Name Space

---

- Two implementations of device naming
  - ▣ device name space is separated from the file-name space
    - e.g. in MS-DOS, **c:** file
      - c: is mapped to a specific port address through a device table
  - ▣ device name space is incorporated in the regular file-system name space, names can be used to access the devices themselves or the files stored on the devices
    - 将设备作为文件，用文件命令统一管理设备
- E.g. Linux系统用主设备号和次设备号描述系统设备
  - ▣ 主设备号相同的设备具有相同设备类型，使用相同的驱动程序
  - ▣ 次设备号（minornumber）用来区分同一个驱动程序控制的不同设备实例

# Device Name Space

- In Unix, OS kernel names each device as  
(device types, *major* device number, *minor* device number)



- physical device name
- In Unix, each device corresponds to a **device file**, the name of the device is the name of its device file. This device file name is visible to users, and can be viewed as a part of **logical device name**
  - all device files are located under subdirectory **/dev**
  - e.g. to print a file  $f_1$  on a printer **/dev/pt<sub>1</sub>**, we can use command:

cat  $f_1$  > /dev/pt<sub>1</sub>      or:   cp  $f_1$  /dev/pt<sub>1</sub>

# Life Cycle of An I/O Request

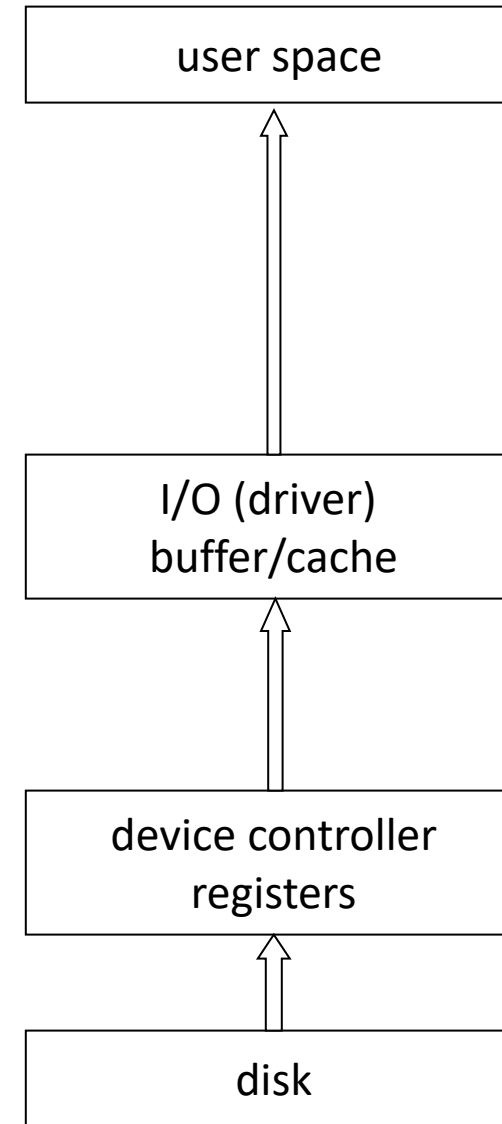
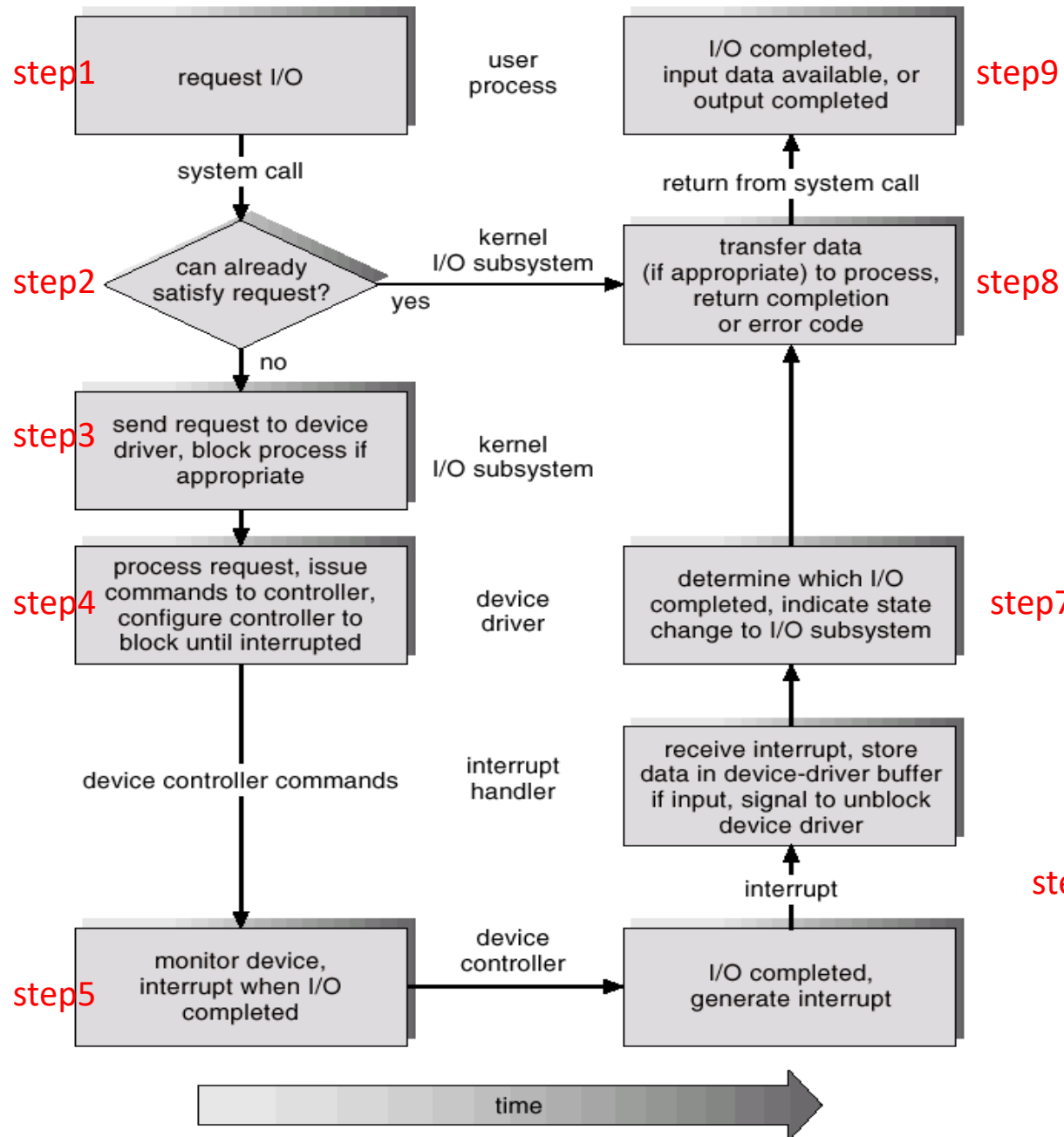
- Step1. the process issues a system call ***read(fd, buffer, nbytes)***
- step2. OS内核代码检查该系统调用的权限和参数正确性等
  - ▣ 分析访问数据所在地址，如果所访问的数据已经在系统buffer cache中，则从中返回所需访问的数据，此I/O request结束
- step3. （当采用阻塞式I/O）进程被阻塞，进入waiting；I/O subsystem 对此I/O request 请求进行排队调度； 当此I/O request被调度后，通过子程序调用或内核消息方式启动相应的设备驱动程序
- step4. driver启动后，
  - ▣ 为接收数据分配buffer 空间；
  - ▣ 调度I/O 操作， e.g. 磁臂径向移动顺序；
  - ▣ 向device controller 的相关寄存器写控制命令(device controller command)



## Life Cycle of An I/O Request

---

- step5. device controller 被启动，完成整个数据读取操作
  - 假设采用DMA方式，driver 需要设置DMA相关寄存器的内容
  - DMA方式直接将数据由disk—> controller register—> buffer
- step6. 数据传输完成后，DMA产生中断，通过interrupt vector table进入中断处理程序；中断处理程序通知driver.
- step7.driver得知传输完毕后，确定是哪一个I/O request完成；然后通知kernel I/O subsystem
- step8. kernel I/O subsystem 将数据由buffer传递到用户进程的地址空间中；将进程由阻塞态转为就绪态。
- step9.进程重新被调度执行





Thanks for your  
attention



北京邮电大学