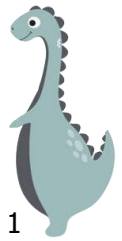
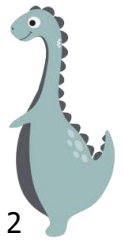
第12章 I/O系统

- 本章内容
- 1. 概述
- 2. I/O硬件
- 3. 应用程序I/O接口
- 4. 内核I/O子系统
- 5. 转换I/O请求为硬件操作



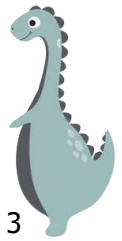
设备多样性及其分类

- 计算机设备种类繁多，从不同的角度出发，I/O设备可分成不同的类型
- 按设备的使用特性分
  - 存储设备：用来保存各种信息的设备。如磁盘，磁带等。
  - I/O设备：向CPU传输信息或输出经过CPU加工处理信息的设备。如键盘、显示器和打印机等。



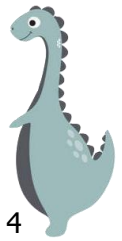
设备多样性及其分类

- 按信息交换单位分
  - 块设备：处理信息的基本单位是字符块。一般块的大小为512B～4KB，如磁盘、磁带等是块设备。
  - 字符设备：处理信息的基本单位是字符。如键盘、打印机和显示器是字符设备。



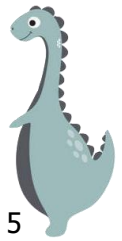
设备多样性及其分类

- 按设备的从属关系分
  - 系统设备：指在操作系统生成时已经登记在系统中的标准设备。如键盘、显示器、打印机等。
  - 用户设备：指操作系统生成时未登记入系统的非标准设备。如鼠标、绘图仪，扫描仪等。



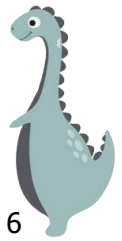
设备管理的任务和功能

- I/O设备在功能与速度方面存在很大差异，需要采用多种方法来控制设备。
- 这些方法形成了I/O子系统的核心，该子系统使内核其他部分不必涉及I/O设备管理。



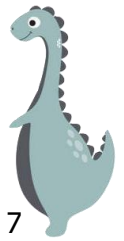
设备管理的任务和功能

- 设备管理的主要任务：
  - 完成用户提出的I/O请求
  - 分配I/O设备
  - 提高I/O设备的利用率
  - 方便用户使用I/O设备



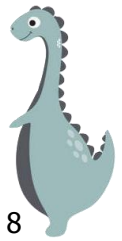
设备管理的任务和功能

- 设备管理应具备以下功能：
  - 设备分配：根据用户的I/O请求，为之分配设备。
  - 设备处理：负责启动设备及I/O操作完成时的中断处理。
  - 缓冲管理：为缓和CPU与I/O速度不匹配的矛盾常设置缓冲区。缓冲管理负责缓冲区的分配和释放及有关管理工作。
  - 设备独立性：又称设备无关性，是指用户编制程序时所使用的设备与物理设备无关。



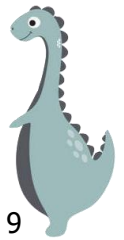
设备管理的任务和功能

- 设备独立性有两种类型：
  - 独立于同类设备的具体设备号
  - 独立于设备类型



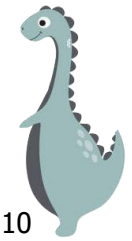
第12章 I/O系统

- 本章内容
- 1. 概述
- 2. I/O硬件
- 3. 应用程序I/O接口
- 4. 内核I/O子系统
- 5. 转换I/O请求为硬件操作



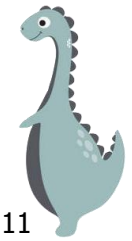
基本架构

- 控制器
  - 用于操作总线或设备的一组电子器件。
  - 处于CPU与I/O设备之间，它接收从CPU发来的命令，并去控制I/O设备工作。
  - 在微型计算机中，又称为接口卡。



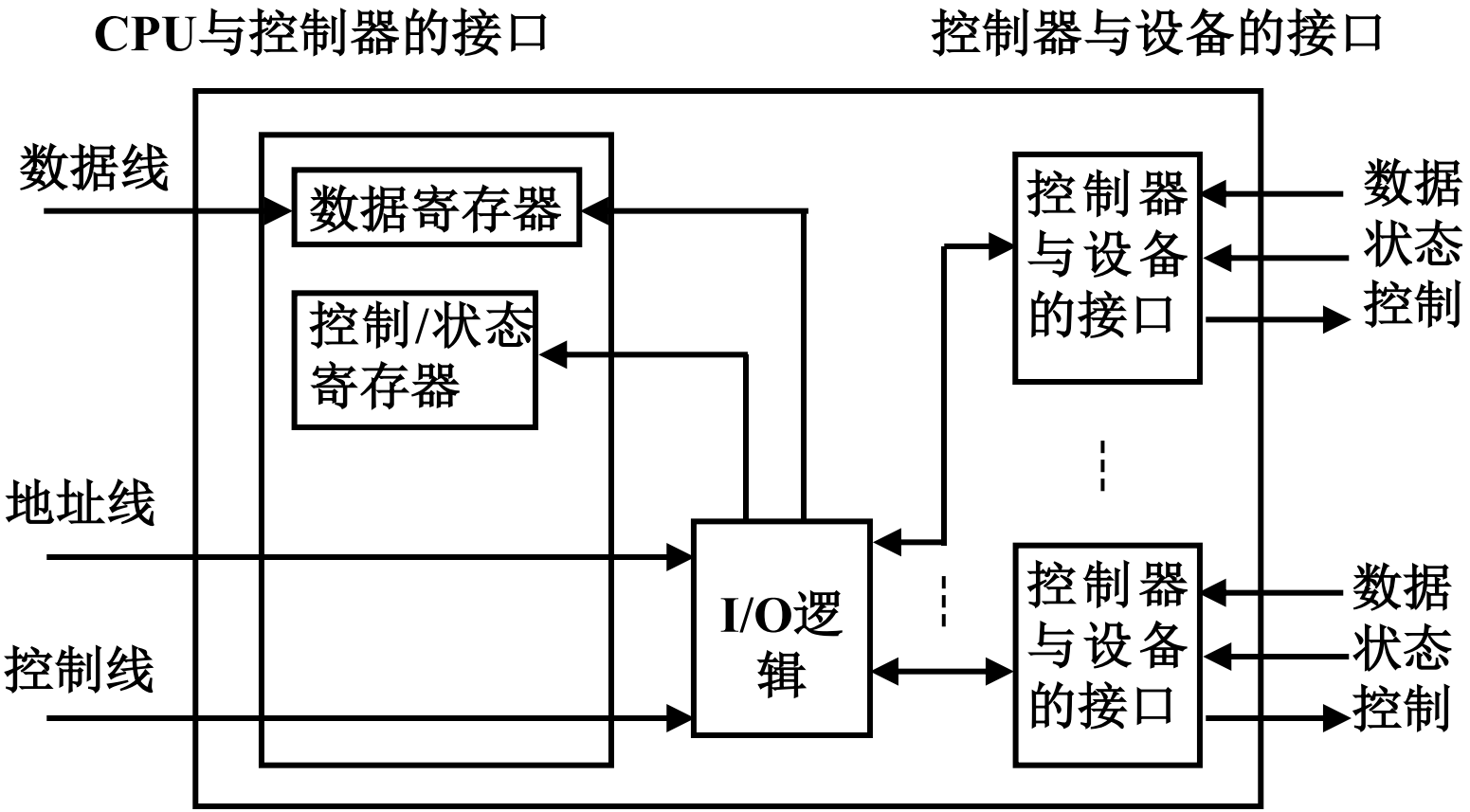
基本架构

- 控制器由三部分组成
  - 设备控制器与处理机的接口：实现CPU与设备控制器之间的通信。
  - 设备控制器与设备的接口：实现设备与设备控制器之间的通信。
  - I/O逻辑：实现对设备的控制，它负责接收命令、对命令进行译码、再根据译出的命令控制设备。



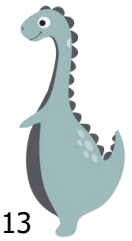
基本架构

- 控制器由三部分组成



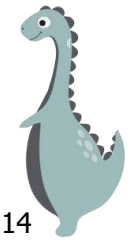
基本架构

- 控制器通常有四种寄存器：
  - 状态寄存器 Status registers：包含一些主机可读取的位信息，指示设备的各种状态
  - 控制寄存器 Control registers：可以被主机用来向设备发送命令或改变设备状态
  - 数据输入寄存器 Data-in registers：可以被主机读取数据
  - 数据输出寄存器 Data-out registers：被主机写入数据以发送数据



基本架构

- 控制器的功能
  - 接收和识别来自CPU的命令：用控制寄存器。
  - 实现CPU与控制器、控制器与设备之间的数据交换：用数据寄存器。
  - 记录设备的状态供CPU查询：用状态寄存器。
  - 识别控制的每个设备地址：用地址译码器。



基本架构

- I/O端口
  - 控制器电路中可被CPU直接访问的寄存器。
  - I/O端口有地址，以接受CPU的访问。
  - 编址方式1：独立编址，配合直接I/O指令。
    - PC机早期采用，现兼容保留
  - 编址方式2：存储器映射I/O，I/O端口和内存分享CPU的地址空间，因此CPU能够像访问普通内存一样访问I/O端口。
    - 被绝大多数CPU架构采用，包括PC机

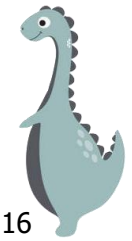
思考：I/O指令是特权指令还是普通指令？

思考：此处的“地址空间”是物理地址还是逻辑地址？

基本架构

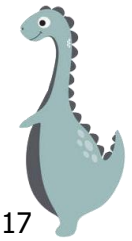
- I/O端口
  - 例：早期PC机独立编址I/O端口

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)



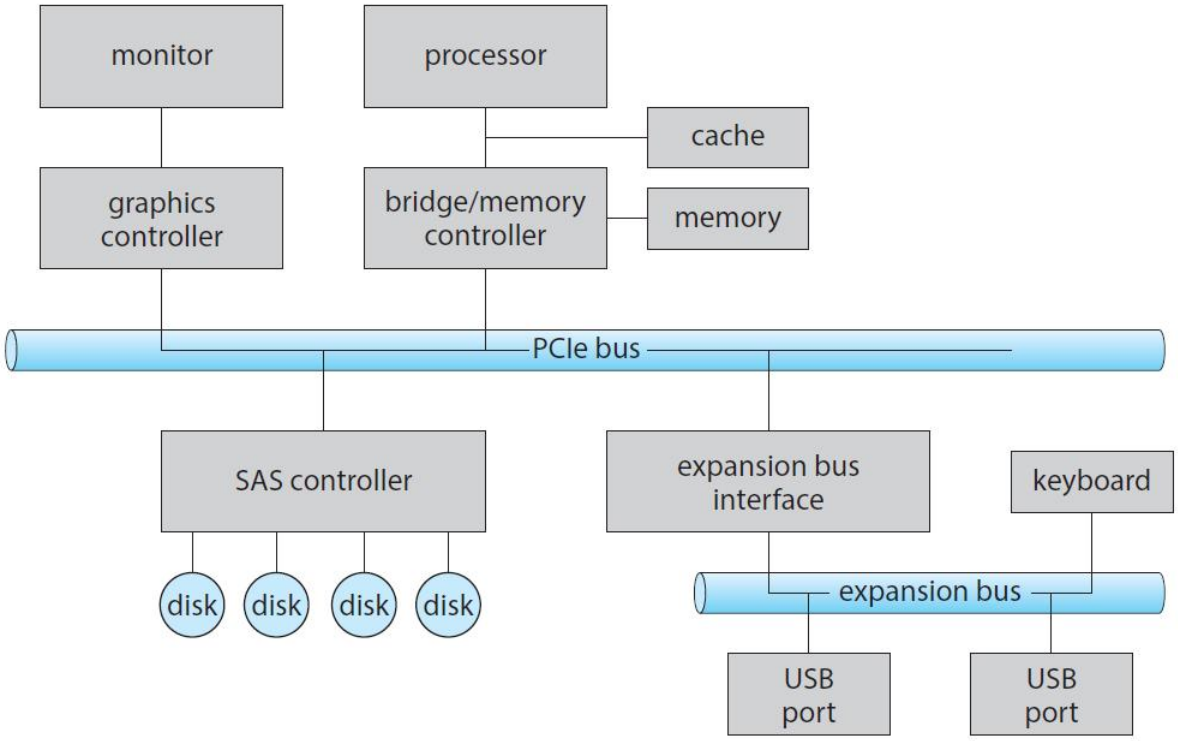
基本架构

- I/O端口
  - CPU不能直接与I/O设备通信，必须借助控制器做“中间人”
  - CPU通过I/O端口与“中间人”通信
  - I/O端口是CPU能触及的最远端，软件的势力范围到此为止



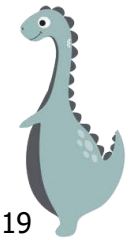
基本架构

- 总线
  - 一组线，和一组严格定义的可以描述在线上传输信息的协议



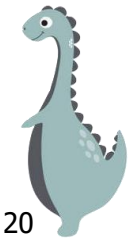
基本架构

- 系统总线
  - 可简单粗暴的理解为CPU引脚的延申
  - 例：ISA总线、PCI总线、PCI-E总线
  - CPU能直接触及的势力范围
  - 一切要接受CPU直接访问的元件（如内存、设备控制器等）都要挂在系统总线上



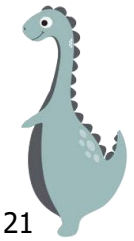
基本架构

- 扩展总线
  - 例：SAS总线、SATA总线、USB总线
  - 总线控制器挂在系统总线上，接受CPU控制
  - 总线接受总线控制器的指挥，CPU无法直接控制



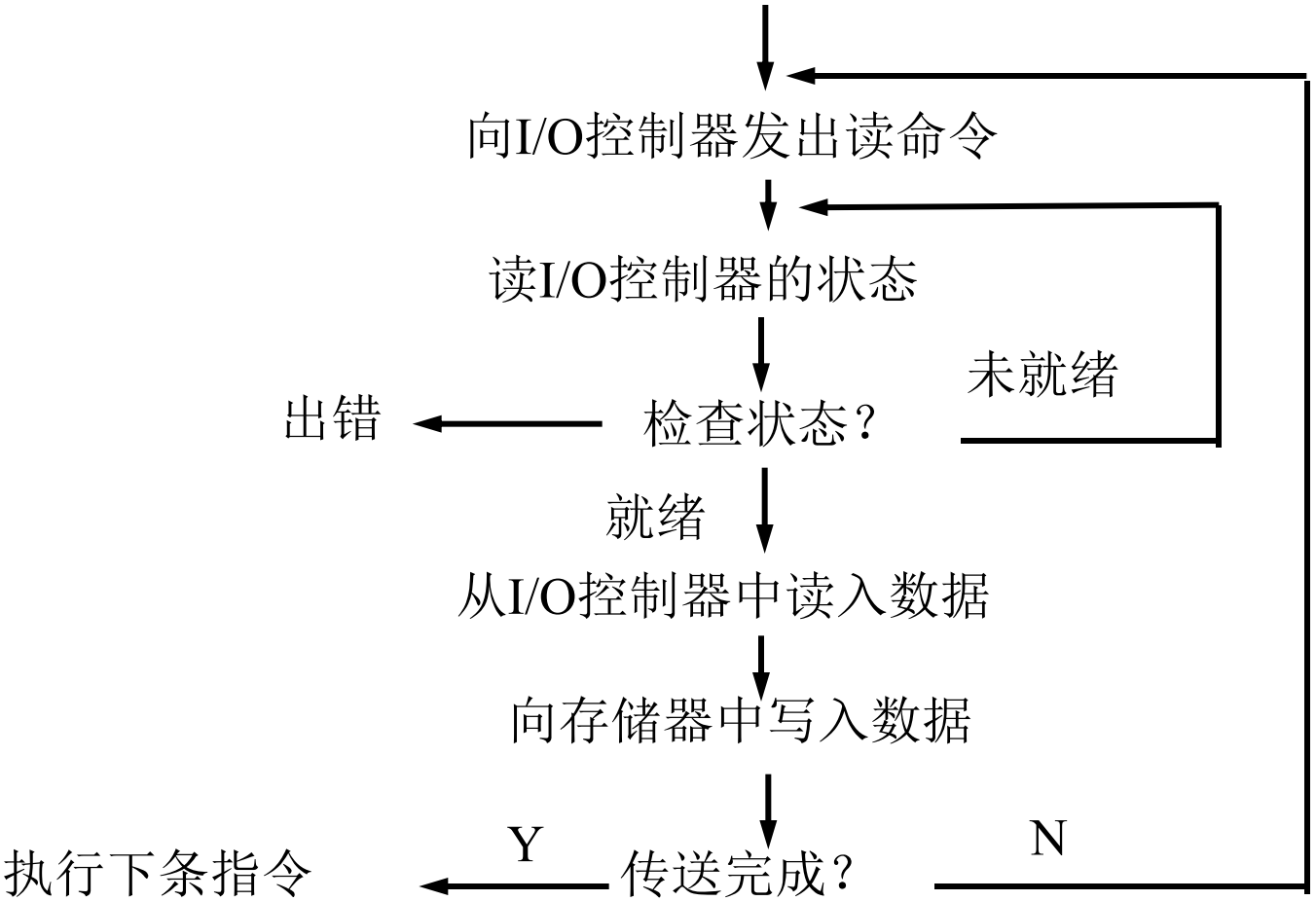
轮询

- 早期计算机系统中无中断机构，设备控制采用程序直接控制方式。
- 程序直接控制方式也称为轮询方式。
- 工作方式简单，但CPU的利用率低。



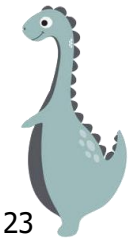
轮询

- 案例1：输入


向I/O控制器发出读命令
读I/O控制器的状态
检查状态？
出错
未就绪
就绪
从I/O控制器中读入数据
向存储器中写入数据
传送完成？
Y
N
执行下条指令

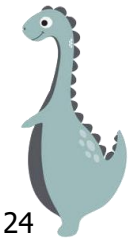
轮询

- 案例2：输出
  - 关键信息位
    - busy位指示控制器是忙还是空闲
    - command-ready位指示命令是否可供控制器执行
  - 通信过程
    - 主机反复读取busy位，直到它变为0（称为忙等或轮询）
    - 主机在命令寄存器中设置write位为1，并将一个字节写入数据输出寄存器
    - 主机将command-ready位置1



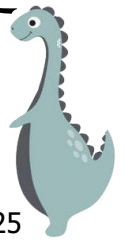
轮询

- 案例2：输出
  - 通信过程
    - 当控制器注意到command-ready位已置1时，它会将busy位置1
    - 控制器读取命令寄存器并看到write命令。它读取数据输出寄存器以获取字节，并对外部设备进行I/O操作
    - 控制器清除command-ready位，清除状态寄存器中的error位以指示设备I/O成功，并清除busy位以指示完成



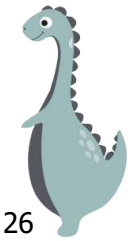
中断

- 基本概念
  - 中断：指计算机系统内发生了某个急需处理的事件，使CPU暂停当前正在执行的程序，转去处理相应的事件处理程序，待处理完毕后又返回原来被中断处继续执行。
  - 中断源：引起中断发生的事件。
  - 中断请求：中断源向CPU发出的请求中断处理的信号。
  - 中断响应：CPU收到中断请求后转向相应事件处理程序的过程。



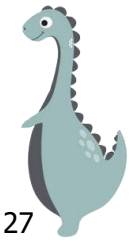
中断

- 基本概念
  - 断点：发生中断时，刚执行完的那条指令所在的单元号。
  - 恢复点：断点的逻辑后继指令的单元号。
  - 现场：是指中断的那一时刻能确保程序继续运行的有关信息。
  - 程序状态字：反映程序运行状态的一组信息。主要包括：指令地址、指令执行情况、CPU状态（管/目态）、中断屏蔽字、寻址方式等。



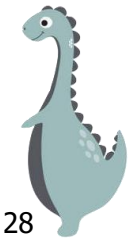
中断

- 基本概念
  - 禁止中断：不允许CPU响应中断，也称为关中断。
  - 允许中断：允许CPU响应中断，也称为开中断。
  - 开中断和关中断是为了保证某些程序执行的原子性。



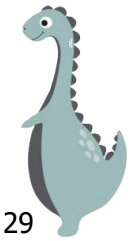
中断

- 基本概念
  - 中断向量：用来存放中断处理程序的入口地址，一般每个中断信号占用两个单元：一个单元用来存放中断处理程序的入口地址，另一个单元用来保存在处理中断时CPU应具有的状态。
  - 中断屏蔽：中断屏蔽表示暂时封锁对中断的响应，待屏蔽消除后再响应。



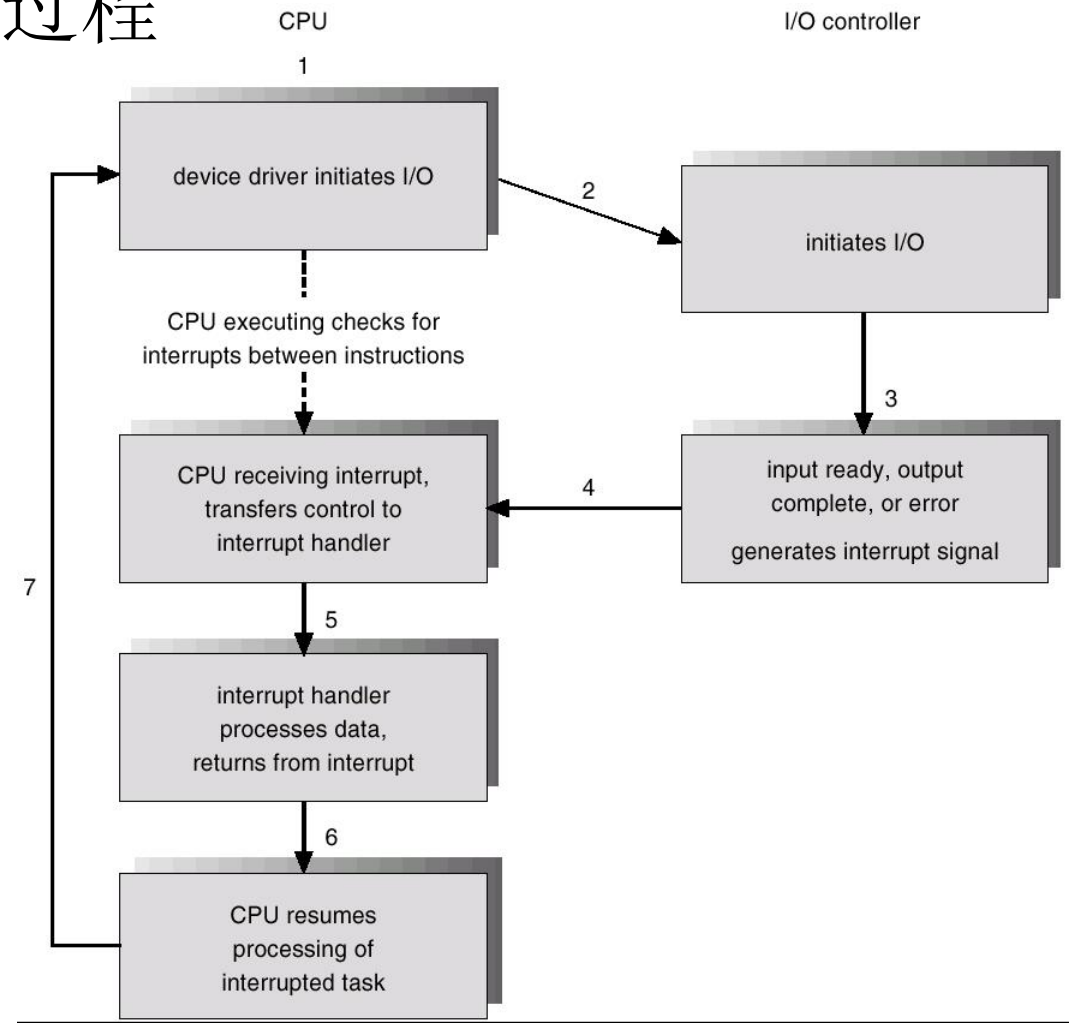
中断

- 中断处理过程
  - 一旦CPU响应中断，系统就开始进行中断处理。中断处理过程如下：
    - 保护被中断进程现场。
    - 分析中断原因，转去执行相应的中断处理程序。
    - 恢复被中断进程的现场，CPU继续执行原来被中断的进程。



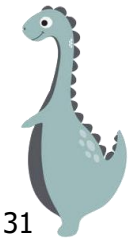
中断

- 工作过程


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

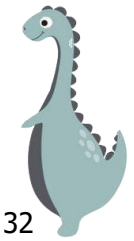
中断

- 分类及优先级
  - 根据中断信号的含义和功能可以将中断分为5类：
    - 硬件故障中断
    - 输入/输出中断
    - 外中断
    - 程序性中断
    - 访管中断



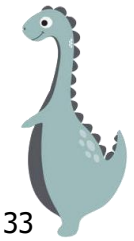
中断

- 分类及优先级
  - 硬件故障中断：机器发生故障时产生的中断。如电源故障、奇偶校验错等。
  - 输入/输出中断：外设或通道操作正常完成或发生某种错误时产生的中断。如传输结束、设备错误等。
  - 外中断：中央处理机外部的非通道式装置引起的中断。如时钟中断、控制台中断等。



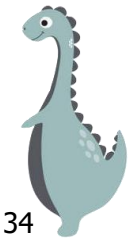
中断

- 分类及优先级
  - 程序性中断：程序执行时发生了程序性质的错误或出现了某些特定状态而产生的中断。如溢出、地址错、指令跟踪等。
  - 访管中断：对OS提出某种服务要求时发生的中断，又称软中断。



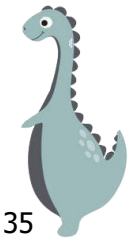
中断

- 分类及优先级
  - 强迫性中断和自愿性中断
  - 强迫性中断由随机事件引起而非程序员事先安排，硬件故障中断、程序性中断、外部中断及输入/输出中断是强迫性中断。
  - 自愿性中断：是正在运行程序所期待的事件，是由执行一条访管指令所引起的。



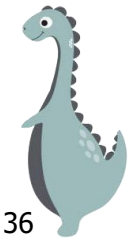
中断

- 分类及优先级
  - 根据中断信号的来源可以分为两类：
  - 外中断。指来自处理机和内存外部的中断，包括I/O设备发出的I/O中断、各种定时器引起的时钟中断等。
  - 内中断。指在处理机和内存内部产生的中断，内中断一般称为陷入或异常，包括程序运算引起的各种错误，如地址非法、校验错、存取访问控制错、算术操作溢出等。



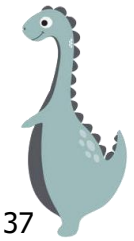
中断

- 分类及优先级
  - 中断优先级是中断响应的优先级别。
  - 当多个中断发生时，系统根据优先级决定响应中断的次序，优先响应高优先级的中断，同级中断则按硬件规定的次序响应。
  - 中断优先级由高到低的顺序为：硬件故障中断、访管中断、程序性中断、外部中断、输入/输出中断。



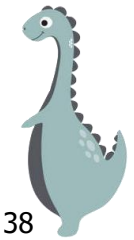
中断

- CPU与设备并行工作，仅当I/O结束时才需CPU花费极短时间做中断处理。CPU利用率大大提高。



直接内存访问

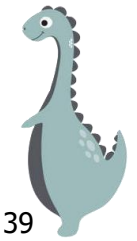
- 中断方式以字节为单位中断CPU，对块设备其效率极低，为此引入了DMA。
- DMA控制方式的思想是在外设与内存之间开辟直接的数据交换通路，在DMA控制器的控制下，设备和内存之间可以成批地进行数据交换。



直接内存访问

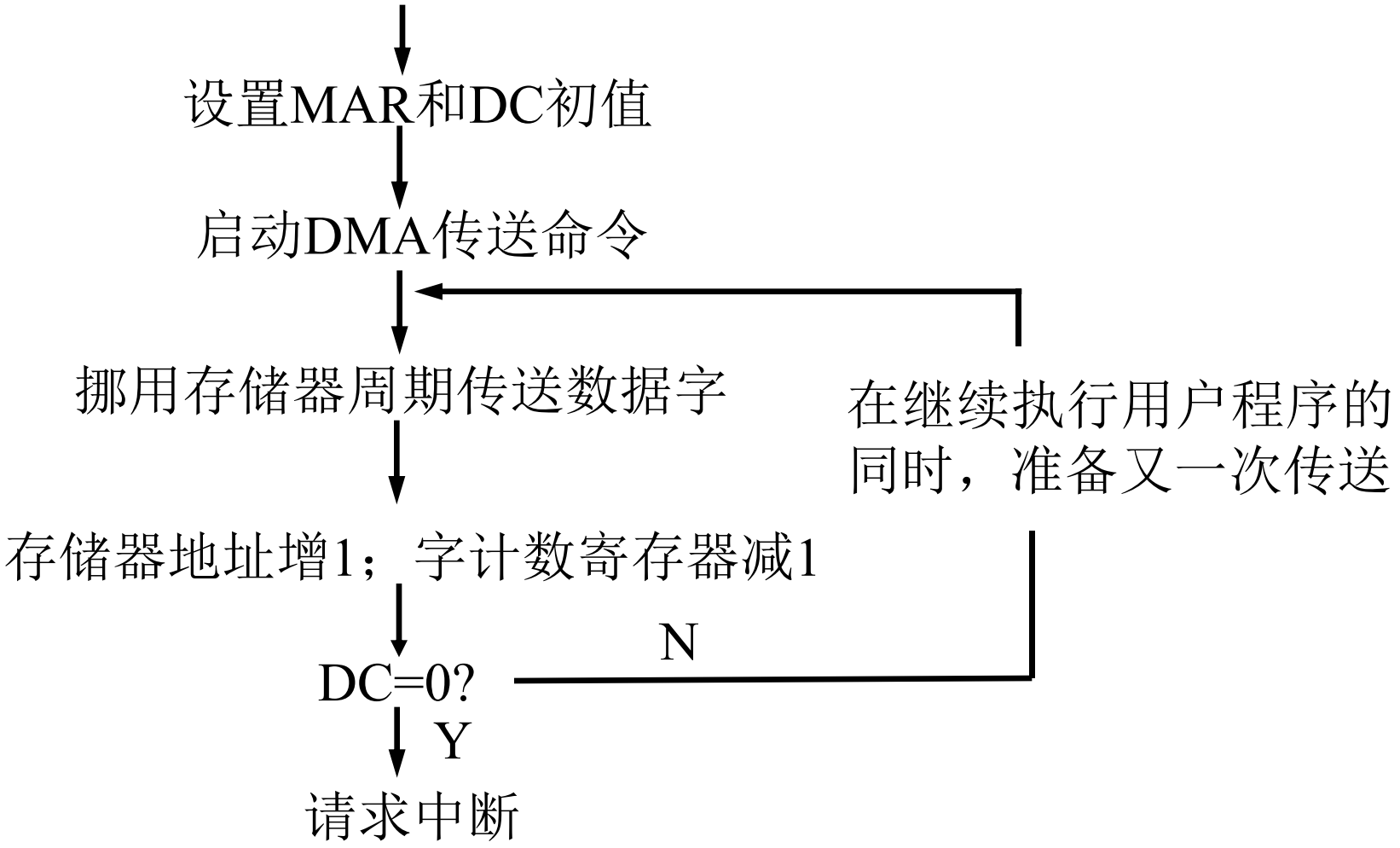
- 为实现主机与控制器之间成块数据的直接交换，必须在DMA控制器中设置如下寄存器：
  - 命令/状态寄存器CR：存放命令及状态
  - 内存地址寄存器MAR：存放内存起始地址
  - 数据寄存器DR：存放传输的数据
  - 数据计数器DC：存放要读写的字（节）数

思考：此处的地址是物理地址还是逻辑地址？



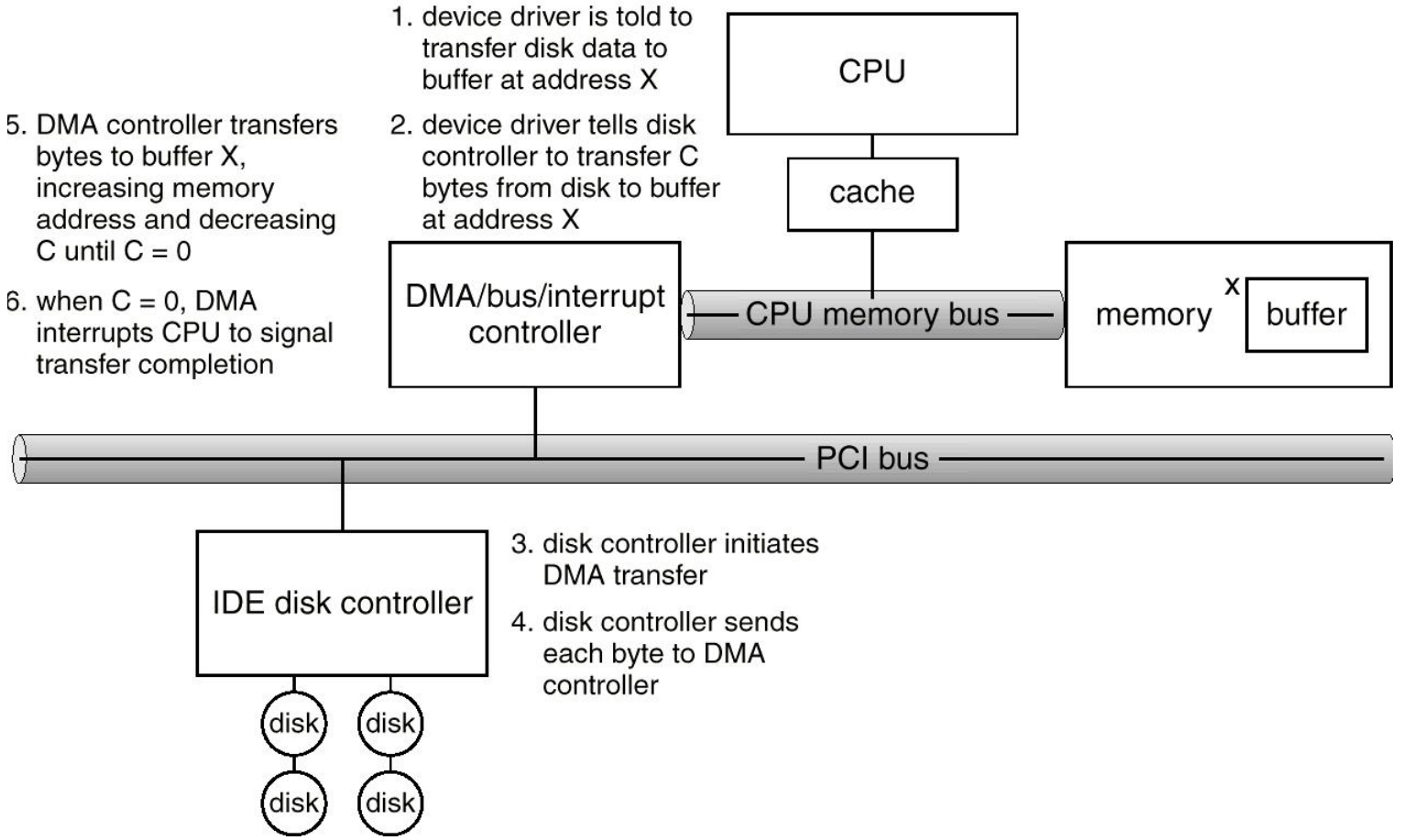
直接内存访问

- 工作过程


设置MAR和DC初值
启动DMA传送命令
挪用存储器周期传送数据字
存储器地址增1；字计数寄存器减1
DC=0?
N
Y
请求中断
在继续执行用户程序的同时，准备又一次传送

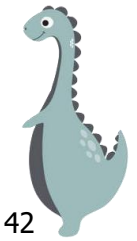
直接内存访问

- 工作过程


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
CPU
cache
DMA/bus/interrupt controller
CPU memory bus
memory
x
buffer
PCI bus
IDE disk controller
disk
disk
disk
disk

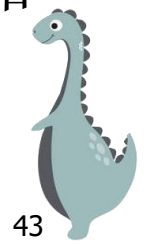
直接内存访问

- 与中断对比
  - 中断控制方式在每个数据传送完成后中断CPU，而DMA控制方式则是在所要求传送的一批数据全部传送结束时中断CPU；
  - 中断控制方式的数据传送是在中断处理时由CPU控制完成，而DMA控制方式则是在DMA控制器的控制下完成。



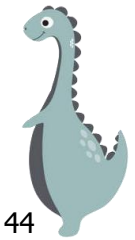
直接内存访问

- 补充说明
  - 所有系统总线都携带“DMA控制器”，DMA控制器是系统总线的一部分
  - 设备控制器针对系统总线设计，懂得配合DMA的引脚和时序
  - DMA控制器有访问内存的能力，但并不了解千奇百怪的设备控制器的工作细节，只能周期性的向其发送读/写信号
  - 为了实现DMA传输，CPU需要对DMA控制器和设备控制器两方面均做好设定



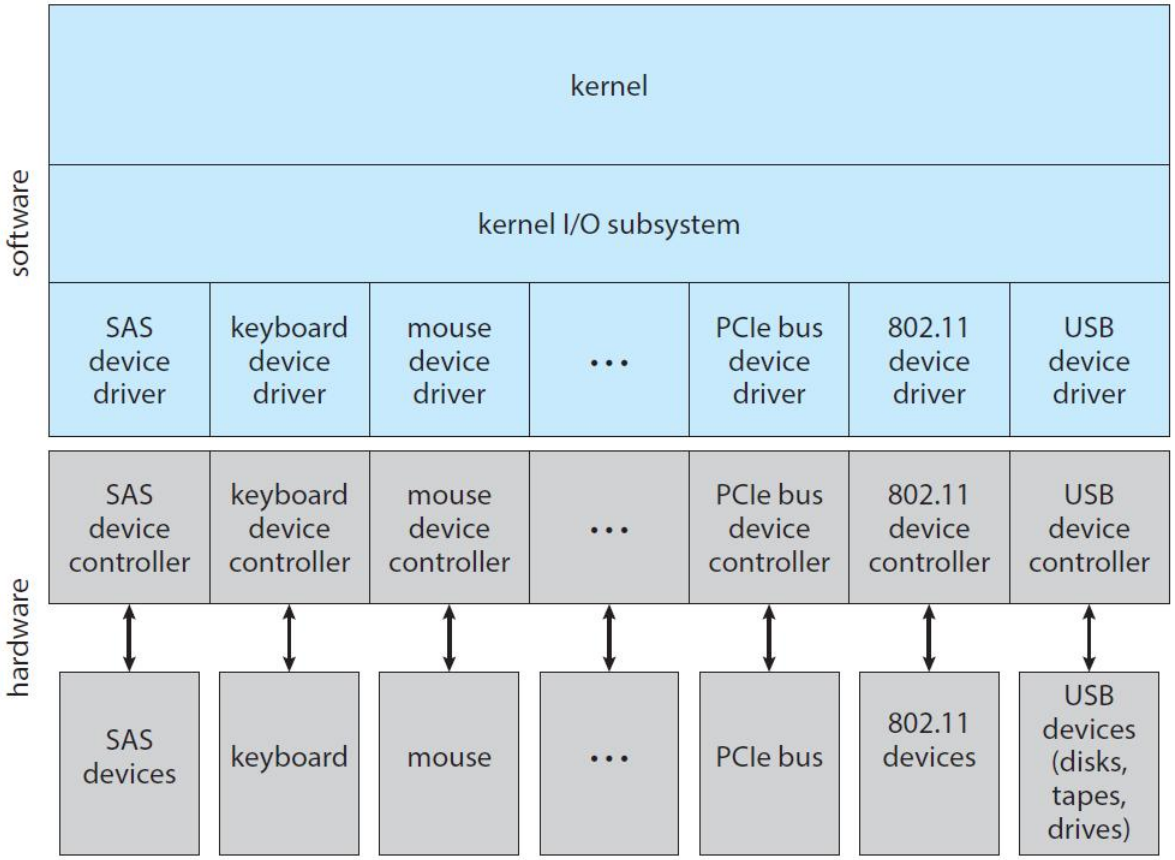
第12章 I/O系统

- 本章内容
- 1. 概述
- 2. I/O硬件
- 3. 应用程序I/O接口
- 4. 内核I/O子系统
- 5. 转换I/O请求为硬件操作



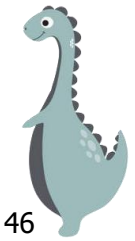
软件层次结构

- 软件层次结构图



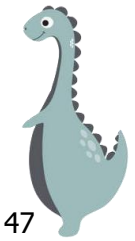
设备驱动程序

- 基本概念
  - 所有与设备相关的代码放在设备驱动程序中。
  - 设备驱动程序与设备密切相关，故应为每一类设备配置一个驱动程序。
  - 设备驱动程序是I/O进程与设备控制器间的通信程序。



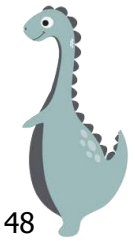
设备驱动程序

- 接收、执行来自上层的请求
  - 实现上层（内核I/O子系统）定义的标准化接口
  - 接收来自上层的与设备无关的抽象请求
  - 把请求转换成设备控制器可以接收的具体命令
  - 将命令发送给设备控制器
  - 监督命令的执行
  - 设备驱动层对内核隐藏了I/O控制器的不同细节
  - 执行命令期间，根据执行周期的长短，进程可能阻塞，可能不阻塞



设备驱动程序

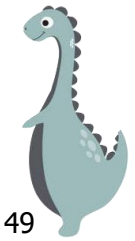
- 处理设备中断
  - 不同的驱动程序加载时，注册登记不同中断号的处理程序
  - 所有中断由内核的中断处理程序统一接收处理，然后分发至相应的驱动程序登记的回调函数
  - 必要时将硬件事件通知上层（内核I/O子系统）
  - 恢复被中断程序的现场信息



I/O系统调用的标准化封装

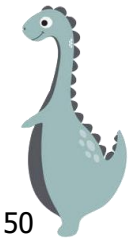
- 不同维度的设备分类

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



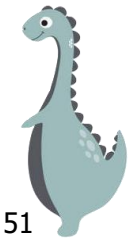
I/O系统调用的标准化封装

- I/O系统调用以若干通用类型的方式封装了设备行为
- 块设备和字符设备
  - 字符流接口
    - 例：键盘
    - 须实现get()或put()，执行单个字符的读写
    - 在此接口之上，某些库函数可提供逐行访问的功能，并提供缓冲和编辑服务



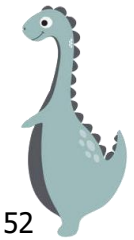
I/O系统调用的标准化封装

- 块设备和字符设备
  - 块设备接口
    - 负责访问磁盘驱动器和其他面向块的设备
    - 须实现针对随机存取设备的read()、write()及seek()等命令
    - 特殊模式：raw I/O，完全绕开文件系统，应用程序直面盘块
    - 特殊模式：direct I/O，在文件的基础上，应用程序自行管理缓冲、上锁，如数据库管理系统
    - 特殊模式：内存映射，如可执行文件



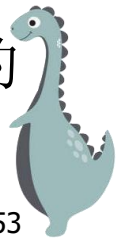
I/O系统调用的标准化封装

- 网络设备
  - 使用socket接口实现网络操作
  - 创建socket
  - connect & listen
  - send & receive
  - select()，管理一组套接字并消除轮询和忙等
  - ……



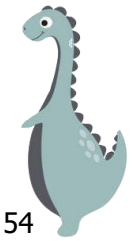
I/O系统调用的标准化封装

- 时钟和定时器
  - 基本功能
    - 给出当前时间
    - 给出流逝的时间
    - 设置定时器，在T时间触发X操作
  - 相关硬件：可编程间隔定时器
    - 可以设置为等待一定时间，然后产生中断
    - 可以设置为重复该过程，周期性产生中断
  - 操作系统接口
    - 通过模拟虚拟时钟，系统可以支持比硬件通道数更多的定时器请求



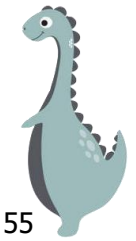
I/O系统调用的标准化封装

- 阻塞和非阻塞I/O
  - 阻塞I/O
    - 在系统调用期间，进程阻塞
    - 系统调用完成后，进程可以恢复执行并接收系统调用返回的值
    - 比非阻塞代码更易于理解，被大多数系统调用采用
  - 非阻塞I/O
    - 非阻塞调用不会停止进程的执行
    - 立刻返回，返回值指示传输了多少字节



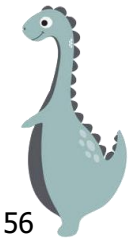
I/O系统调用的标准化封装

- 阻塞和非阻塞I/O
  - 异步系统调用
    - 立即返回，无需等待I/O完成
    - I/O在将来某个时间完成，然后通过一些变量、信号、软中断或回调函数传达给应用程序
  - 非阻塞调用和异步调用之间的区别
    - 非阻塞read()会立即返回缓冲区中现成可用的数据，系统调用随之结束，不触发磁盘操作
    - 异步read()返回意味着请求已受理，调用将完整执行，会在将来的某个时间完成



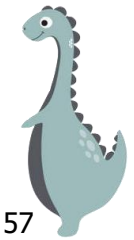
I/O系统调用的标准化封装

- 特殊
  - Vectored I/O，也称为scatter–gather
    - 加强版的DMA传输
    - 内存中多片不连续的缓冲区，一次性批量传输
  - 驱动程序接口和系统调用接口并非都是标准化的
    - 一些特殊设备，提供非标准化的驱动程序接口和系统调用接口，仅供特殊应用程序使用
    - 例：智能手环/手表，仅供同品牌的特定App访问
    - 非标准系统调用相关API：ioctl()、deviceIoControl()



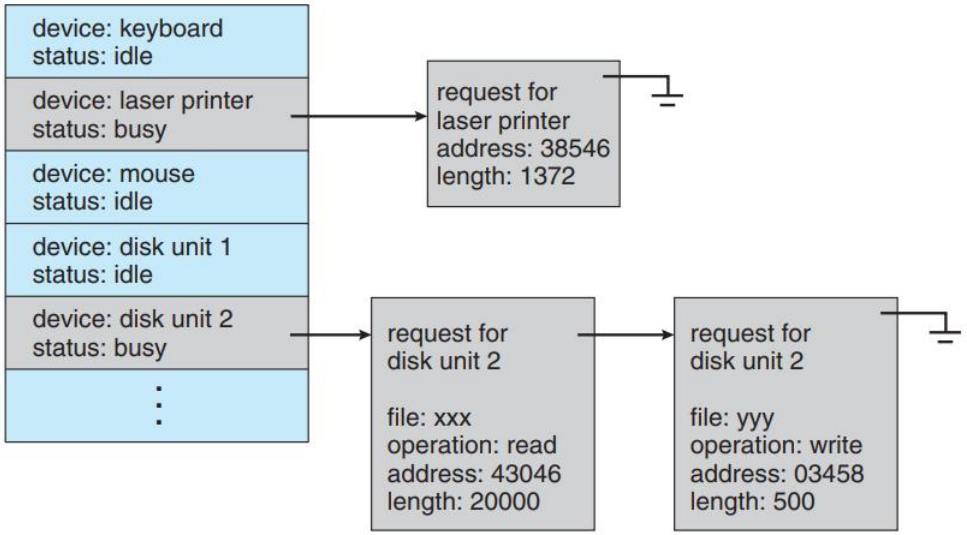
第12章 I/O系统

- 本章内容
- 1. 概述
- 2. I/O硬件
- 3. 应用程序I/O接口
- 4. 内核I/O子系统
- 5. 转换I/O请求为硬件操作



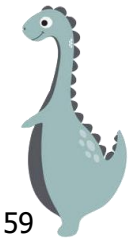
I/O调度

- 调度一组I/O请求就是确定一个好的顺序来执行这些请求。
  - 调度目标：吞吐量、公平、响应速度
  - 对阻塞I/O请求，为每个设备维护一个等待队列
  - 对异步I/O请求，为每个设备维护一个请求队列



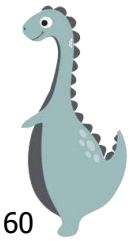
缓冲

- 缓冲：当设备间传输数据的时候，暂时存放在内存中。
  - 解决设备速度不匹配
  - 解决设备传输块的大小不匹配
  - 维持“拷贝语义”
    - 数据首先从用户缓冲提交给内核缓冲
    - I/O期间，用户程序修改用户缓冲，不影响I/O



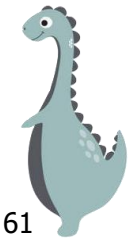
缓冲

- 缓冲的引入
  - 提高处理机与外设并行度
  - 引入缓冲的主要原因：
    - 缓和CPU与I/O设备间速度不匹配的矛盾，
    - 提高CPU与I/O设备并行操作的程度，
    - 减少设备对CPU的中断频率，放宽CPU对中断响应时间的限制。



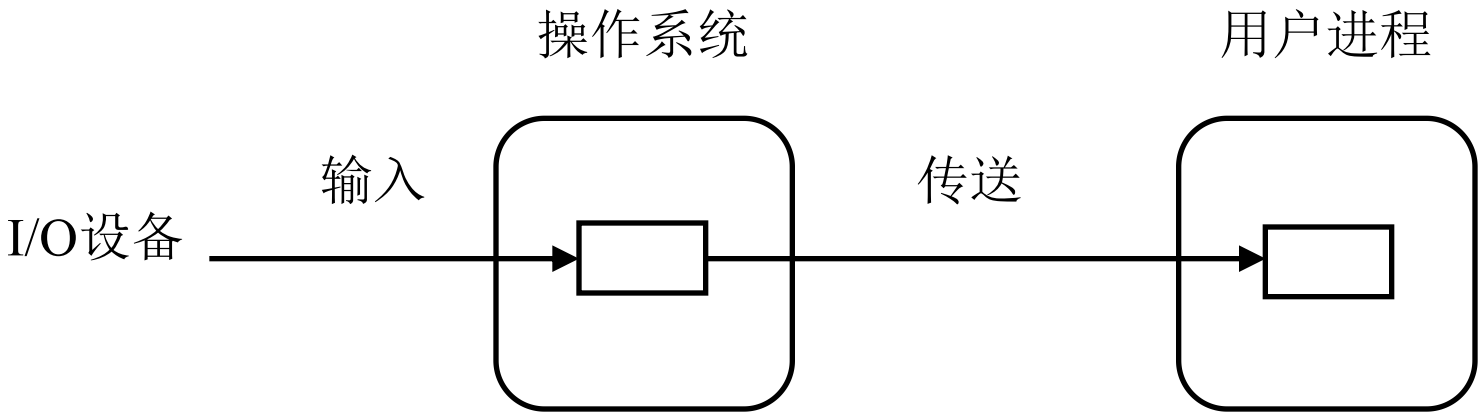
缓冲

- 缓冲的实现方法
  - 硬件缓冲器：如I/O控制器中的数据缓冲寄存器，但成本太高。
  - 软件缓冲：一片内存区域，用来临时存放输入输出数据。
  - 缓冲技术分为：
    - 单缓冲
    - 双缓冲
    - 循环缓冲
    - 缓冲池



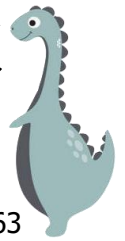
缓冲

- 单缓冲
  - 单缓冲是在设备和处理机之间设置一个缓冲区。



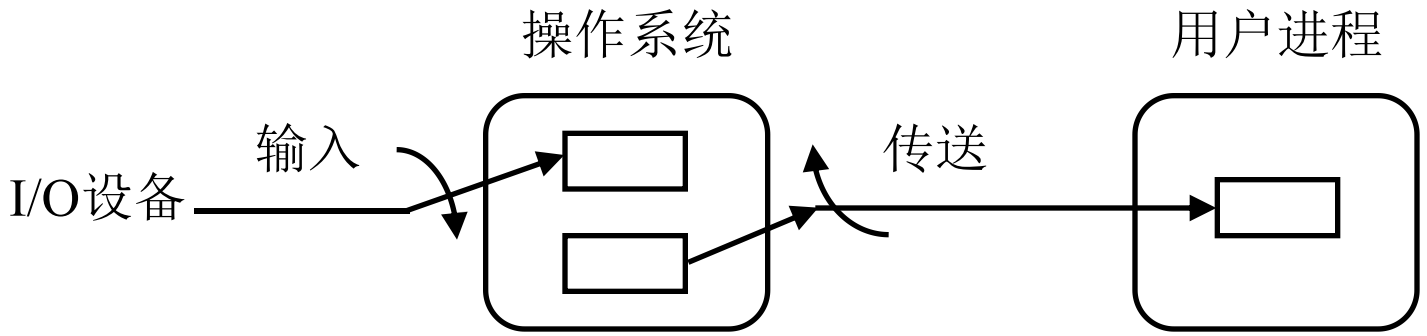
缓冲

- 单缓冲
  - 在块设备输入时，先从磁盘把一块数据输入至缓冲区，然后OS将缓冲区中的数据送到用户区。在块设备输出时，先将要输出的数据从用户区复制到缓冲区，然后再将缓冲区中的数据写到设备。
  - 在字符设备输入时，缓冲区用于暂存用户输入的一行数据。在输入期间，用户进程阻塞以等待一行数据输入完毕；在输出时，用户进程将一行数据送入缓冲区后继续执行计算。当用户进程已有第二行数据要输出时，若第一行数据尚未输出完毕，则用户进程阻塞。



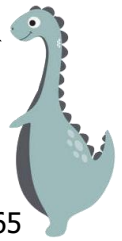
缓冲

- 双缓冲
  - 引入双缓冲，可以进一步提高处理机与设备的并行操作程度。



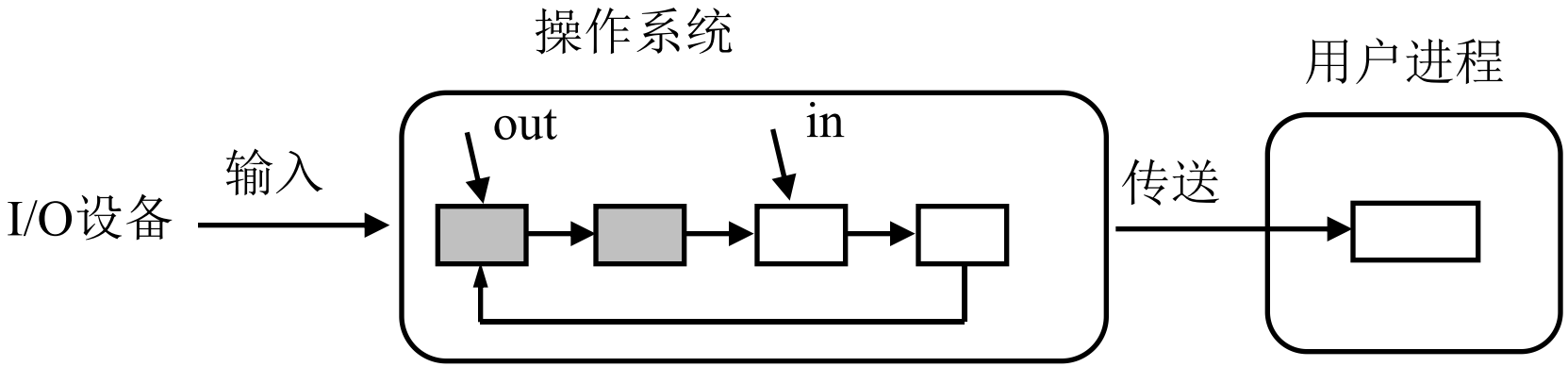
缓冲

- 双缓冲
  - 在块设备输入时，可先将第一个缓冲区装满，之后便装填第二个缓冲区，与此同时OS可将第一个缓冲区中的数据传到用户区；当第一个缓冲区中的数据处理完后，若第二个缓冲区已装满，则处理机又可处理第二个缓冲区中的数据，而设备又可装填第一个缓冲区。输出与此类似。
  - 在字符设备输入时，若采用行输入方式和双缓冲，则用户在输入完第一行后，CPU执行第一行中的命令，而用户可以继续向第二个缓冲区中输入一行数据。



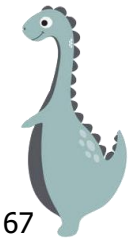
缓冲

- 循环缓冲
  - 若输入输出速度与数据处理速度相当，则双缓冲能获得较好的效果；若速度相差较大，则可以通过增加缓冲区的数量来改善性能。



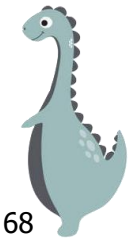
缓冲

- 循环缓冲
  - 循环缓冲中包含多个大小相等的缓冲区，每个缓冲区中有一个指针指向下一个缓冲区，最后一个缓冲区的指针指向第一个缓冲区，由此构成一个环形。
  - 循环缓冲用于输入/输出时，还需要两个指针in和out。
  - 对于输入而言，in指向下一个可用的空缓冲区，out指向下一个可以提取数据的满缓冲区。显然，对输出而言正好相反。



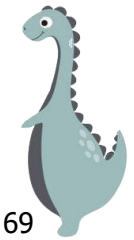
缓冲

- 缓冲池
  - 循环缓冲适用于合作进程，当系统较大且共享缓冲区的进程较多时，这要消耗大量内存。目前广泛使用的是公用缓冲池。
  - 缓冲池由多个缓冲区组成，其中的缓冲区可供多个进程共享，既能用于输入又能用于输出。



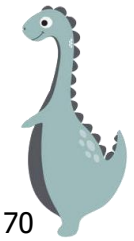
缓冲

- 缓冲池
  - 缓冲池中有三种类型的缓冲区队列：
    - 空缓冲队列：由空闲缓冲区构成的队列
    - 输入队列：装满输入数据的缓冲区队列
    - 输出队列：装满输出数据的缓冲区队列
  - 此外，还有四块工作缓冲区：
    - 用于收容输入数据的工作缓冲区
    - 用于提取输入数据的工作缓冲区
    - 用于收容输出数据的工作缓冲区
    - 用于提取输出数据的工作缓冲区



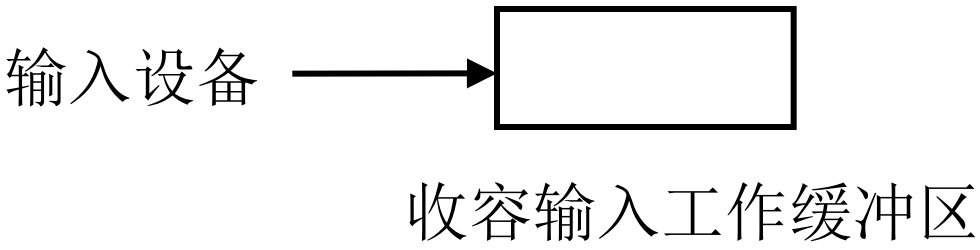
缓冲

- 缓冲池
  - 缓冲池有四种工作方式：
    - 收容输入
    - 提取输入
    - 收容输出
    - 提取输出



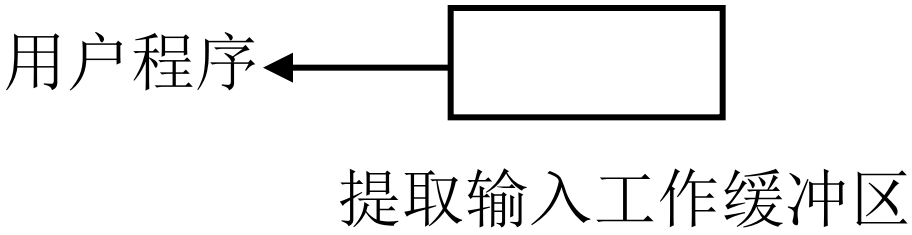
缓冲

- 缓冲池
  - 收容输入
    - 当输入进程需要输入数据时，便从空缓冲队列的队首摘下一个空缓冲区，把它作为收容输入工作缓冲区，然后把数据输入其中，装满后再将它挂到输入队列队尾。



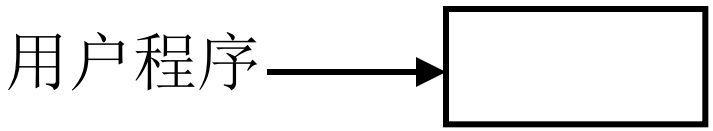
缓冲

- 缓冲池
  - 提取输入
    - 当计算进程需要输入数据时，便从输入队列取得一个缓冲区作为提取输入工作缓冲区，计算进程从中提取数据，数据用完后再将它挂到空缓冲队列尾。



缓冲

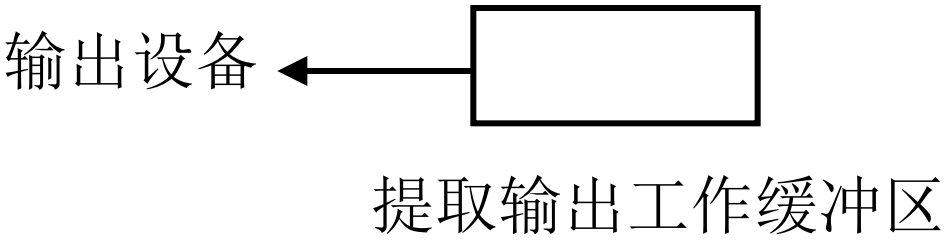
- 缓冲池
  - 收容输出
    - 当计算进程需要输出数据时，便从空缓冲队列的队首取得一个空缓冲，作为收容输出工作缓冲区，当其中装满输出数据后，再将它挂到输出队列尾。



收容输出工作缓冲区

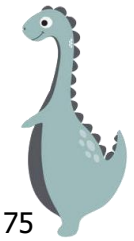
缓冲

- 缓冲池
  - 提取输出
    - 当要输出时，由输出进程从输出队列中取得一装满输出数据的缓冲区，作为提取输出工作缓冲区，当数据提取完后，再将它挂到空缓冲队列的末尾。



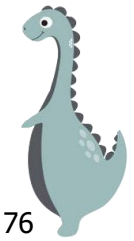
高速缓存

- 高速缓存：存放数据副本的高速存储器
- 缓冲与高速缓存的差别
  - 缓冲是为了应对数据传输而设置的中转站，用完作废
  - 缓存是为了应对未来重复的读取，而保留的一个副本，长时间驻留，但须处理与原始数据的一致性问题



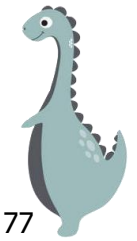
假脱机与设备预留

- 假脱机：保存设备输出的缓冲区
- Spooling技术是将独占设备改造为共享设备的技术



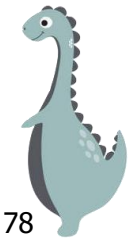
假脱机与设备预留

- Spooling技术
  - Spooling是Simultaneous Peripheral Operating On-Line的缩写，意思是外部设备同时联机操作，又称假脱机操作。
  - 在Spooling系统中，用一道程序模拟脱机输入时的外围控制机功能，把低速输入设备上的数据传送到高速磁盘上；再用另一道程序来模拟脱机输出时的外围控制机功能，把数据从磁盘传送到低速输出设备上。



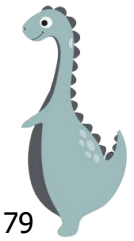
假脱机与设备预留

- Spooling技术
  - Spooling系统由三部分组成：
    - 输入井和输出井
    - 输入缓冲区和输出缓冲区
    - 输入进程和输出进程



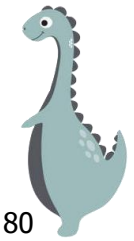
假脱机与设备预留

- Spooling技术
  - 输入井和输出井
    - 输入井和输出井是磁盘上的两个存储区域。
    - 输入井收容I/O设备输入的数据。
    - 输出井收容用户程序的输出数据。



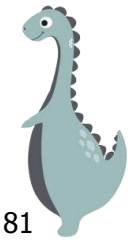
假脱机与设备预留

- Spooling技术
  - 输入缓冲区和输出缓冲区
    - 输入缓冲区和输出缓冲区是内存中的两个缓冲区。
    - 输入缓冲区用于暂存由输入设备送来的数据，以后再传送到输入井；
    - 输出缓冲区用于暂存从输出井送来的数据，以后再传送到输出设备。



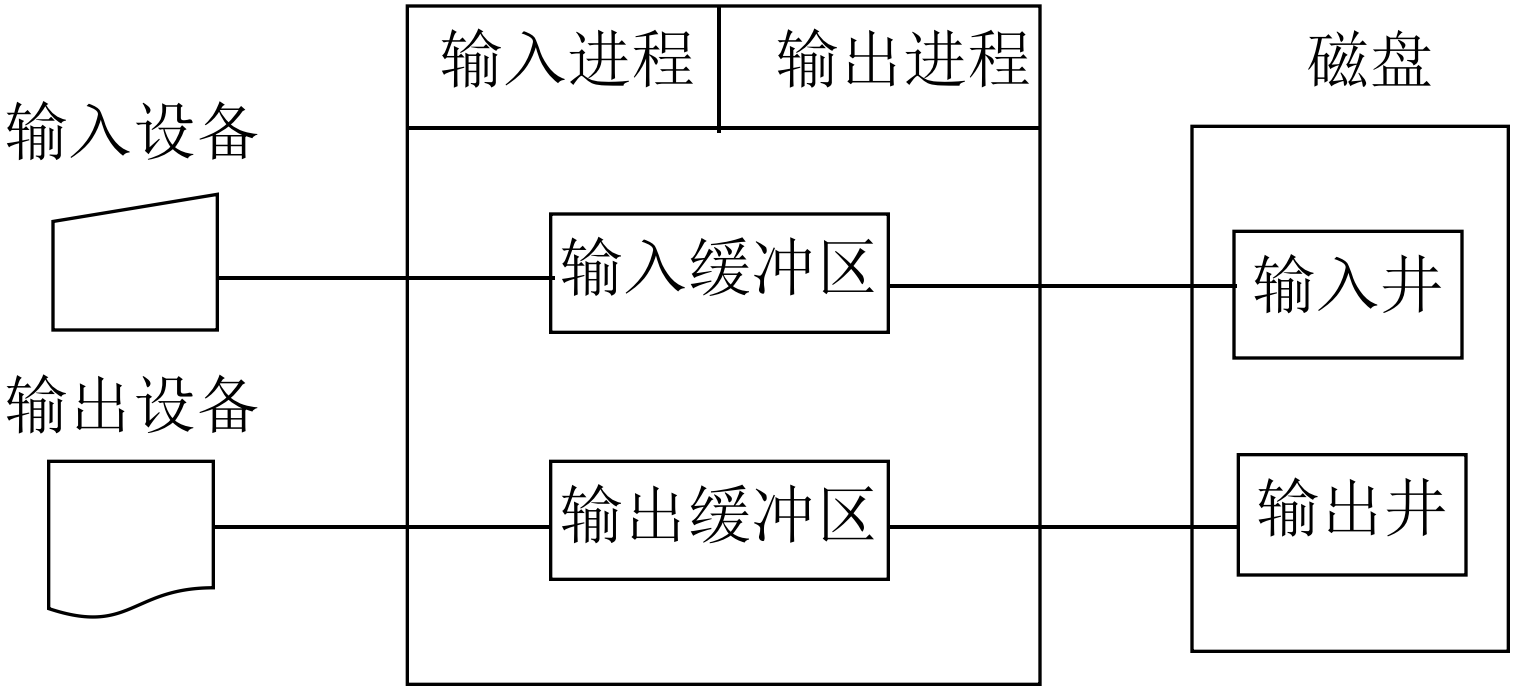
假脱机与设备预留

- Spooling技术
  - 输入进程和输出进程
    - 输入进程模拟脱机输入时的外围控制机，将用户要求的数据从输入设备通过输入缓冲区再送到输入井，当CPU需要输入数据时直接从输入井读入内存；
    - 输出进程模拟脱机输出时的外围控制机，把用户要求输出的数据先从内存送到输出井，待输出设备空闲时，再将输出井中的数据经过输出缓冲区送到输出设备上。



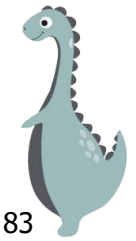
假脱机与设备预留

- Spooling技术



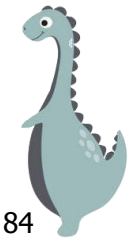
假脱机与设备预留

- 共享打印机
  - 打印机是常用的独享设备，但利用Spooling技术可以将它改造成供多个用户共享的设备。
  - 当用户进程请求打印时，Spooling系统不将打印机真正分配给它，而只为它做两件事：
    - 由输出进程在输出井中为之申请一空闲磁盘区，并将要打印的数据送入其中；
    - 输出进程再为用户进程申请一张空白的用户请求打印表，并将用户的打印要求填入其中，再将该表挂到请求打印队列上。



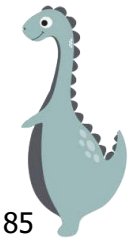
假脱机与设备预留

- 共享打印机
  - 如果打印机空闲，输出进程将从打印队列队首取出一张请求打印表，根据表中的要求将要打印的数据从输出井传送到内存缓冲区，再由打印机进行打印。
  - 打印完后，再取下一张表，直至请求队列为空。此时，输出进程阻塞，当再有打印请求时，才将输出进程唤醒。



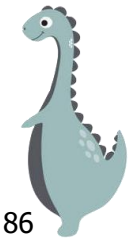
假脱机与设备预留

- 设备预留
  - Spooling以外，处理独占设备的并发请求的另一类机制
  - 设备预留：提供对设备的互斥访问
  - 预留：亦即为某进程保留该设备的使用权，在该进程获得运行之前，其他申请该设备的进程将得不到使用权。



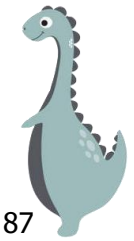
错误处理

- OS可以对短暂出错进行弥补
  - 例如：磁盘read出错后，可以重试read
- 当I/O失败时，大多会返回一个错误码
- 系统日志记录出错报告



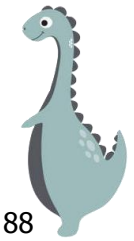
I/O保护

- 通过有意或无意地发出非法I/O指令，用户程序可以破坏系统的正常操作
- 为防止用户执行非法I/O，定义所有I/O指令为特权指令
  - 独立编址使用I/O指令
  - 统一编址使用普通的MOV指令，为此，把I/O地址映射到逻辑地址空间的高地址部分（即仅限内核态访问的区域）



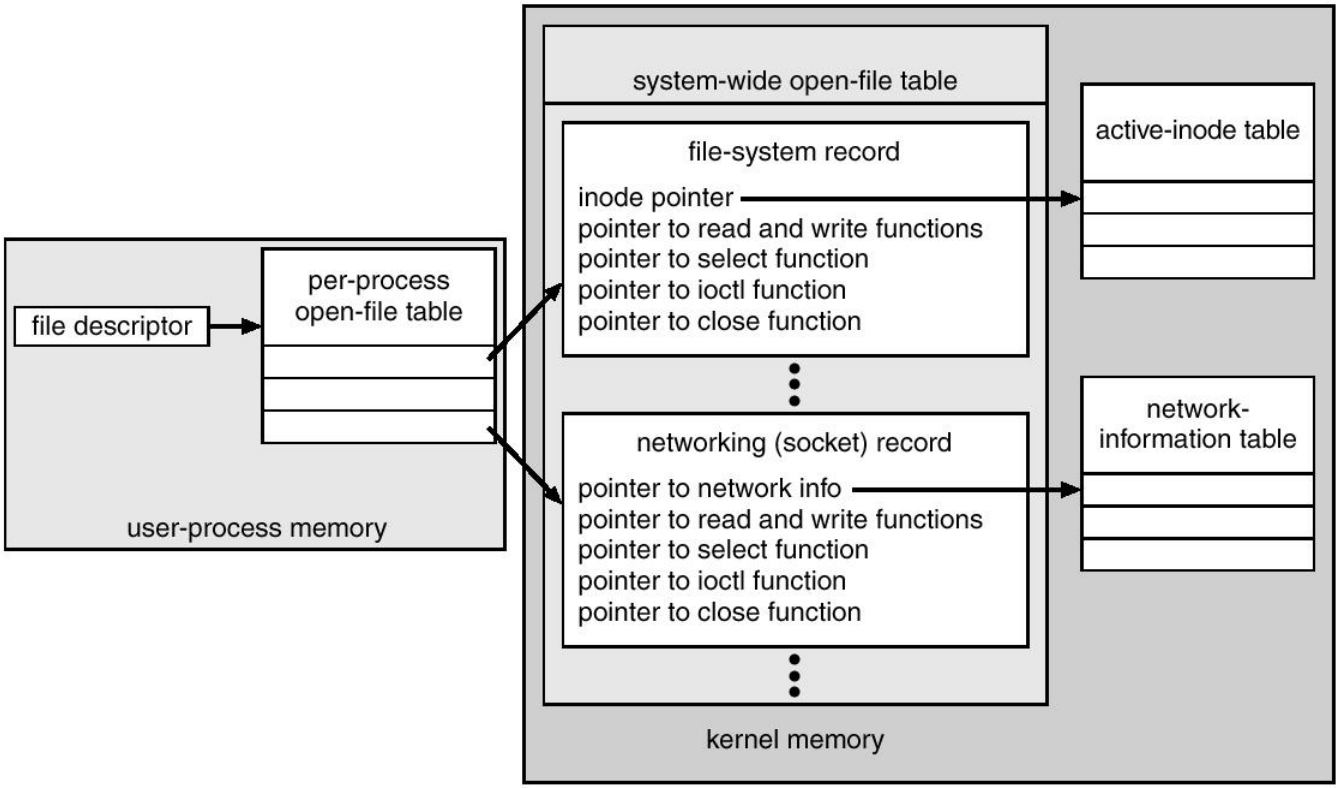
内核数据结构

- 内核需要保存I/O组件使用的状态信息，包括打开文件表，网络连接，字符设备状态等
- 许多复杂的数据结构用来跟踪缓冲，内存分配，及“脏”块
- 某些OS用面向对象和消息传递的方法来实现I/O



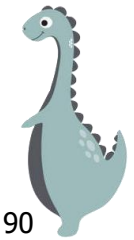
内核数据结构

- 打开文件表的结构



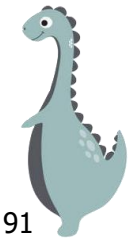
第12章 I/O系统

- 本章内容
- 1. 概述
- 2. I/O硬件
- 3. 应用程序I/O接口
- 4. 内核I/O子系统
- 5. 转换I/O请求为硬件操作



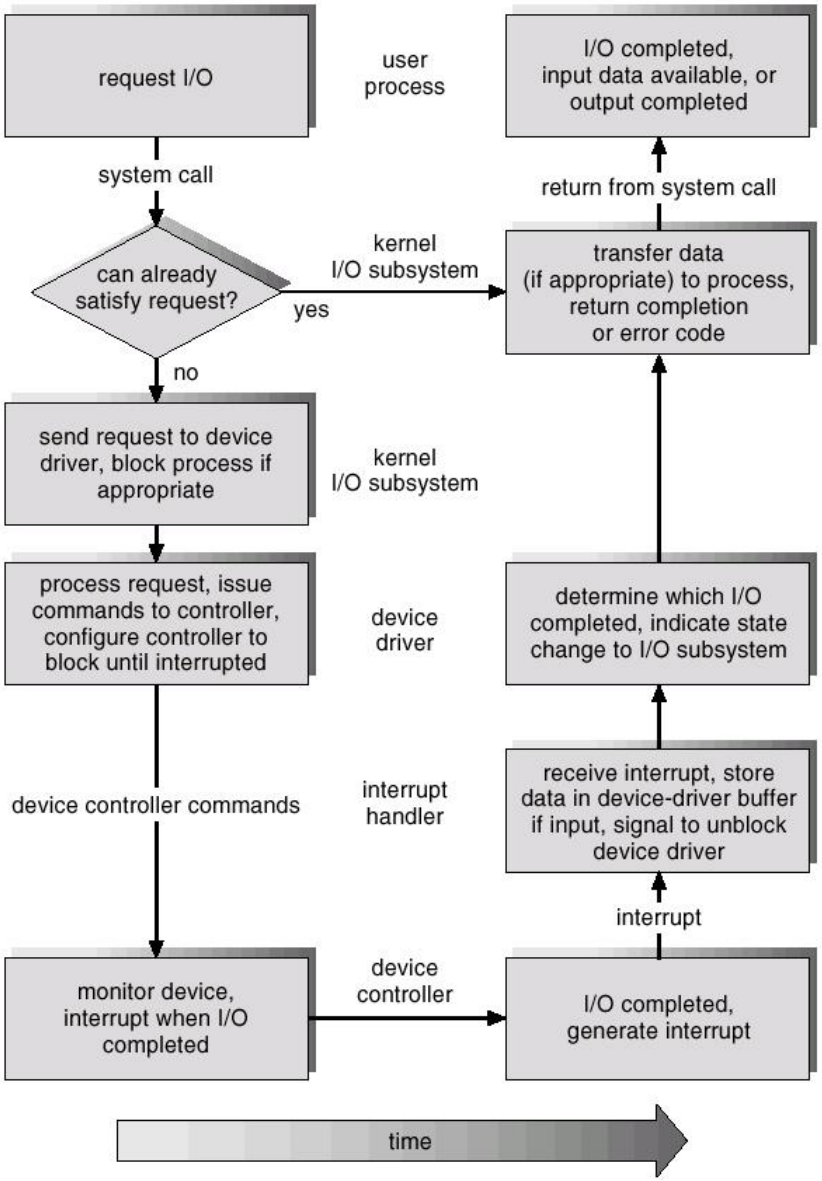
请求的响应过程

- 考虑进程从磁盘中读取一个文件
  - 确定保存文件的设备
  - 将文件名转换成设备实现表示
  - 把数据从磁盘读到缓冲区中
  - 通知进程数据可用
  - 把控制权返回给进程



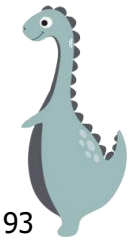
请求的响应过程

- I/O的生命周期



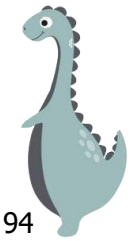
I/O软件的层次结构

- I/O软件设计的基本思想是将设备管理软件组织成一种层次结构。其中低层软件与硬件相关，而高层软件则为用户提供一个友好的、清晰而统一的接口。
- I/O设备管理软件一般分为四层：
  - 中断处理程序
  - 设备驱动程序
  - 与设备无关的软件（或设备独立性软件）
  - 用户空间的软件



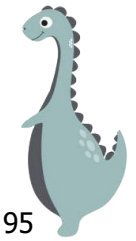
I/O软件的层次结构

- 与设备无关的软件
  - 狭义：即I/O子系统
    - 执行设备命名、设备保护、缓冲管理、独占设备的分配与释放、错误处理等
  - 广义：I/O子系统及内核中的其它子系统
    - 对设备做进一步抽象、虚拟化
    - 例：文件子系统把磁盘抽象为文件、盘块
    - 例：网络子系统把网卡抽象为套接字



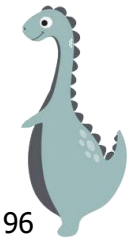
I/O软件的层次结构

- I/O与设备无关的三个层次
  - 不具备通用性，未做到设备无关
    - 提供驱动程序，但驱动程序提供的接口是非标准的，仅供特殊应用程序调用
    - 例：智能手环/手表
  - 标准化、通用化
    - 驱动程序的接口是系统定义的，统一的标准的接口，可供一般应用程序调用
    - 逻辑设备与物理设备一一对应，用户程序可见
    - 例：扫描仪、刻录机、打印机



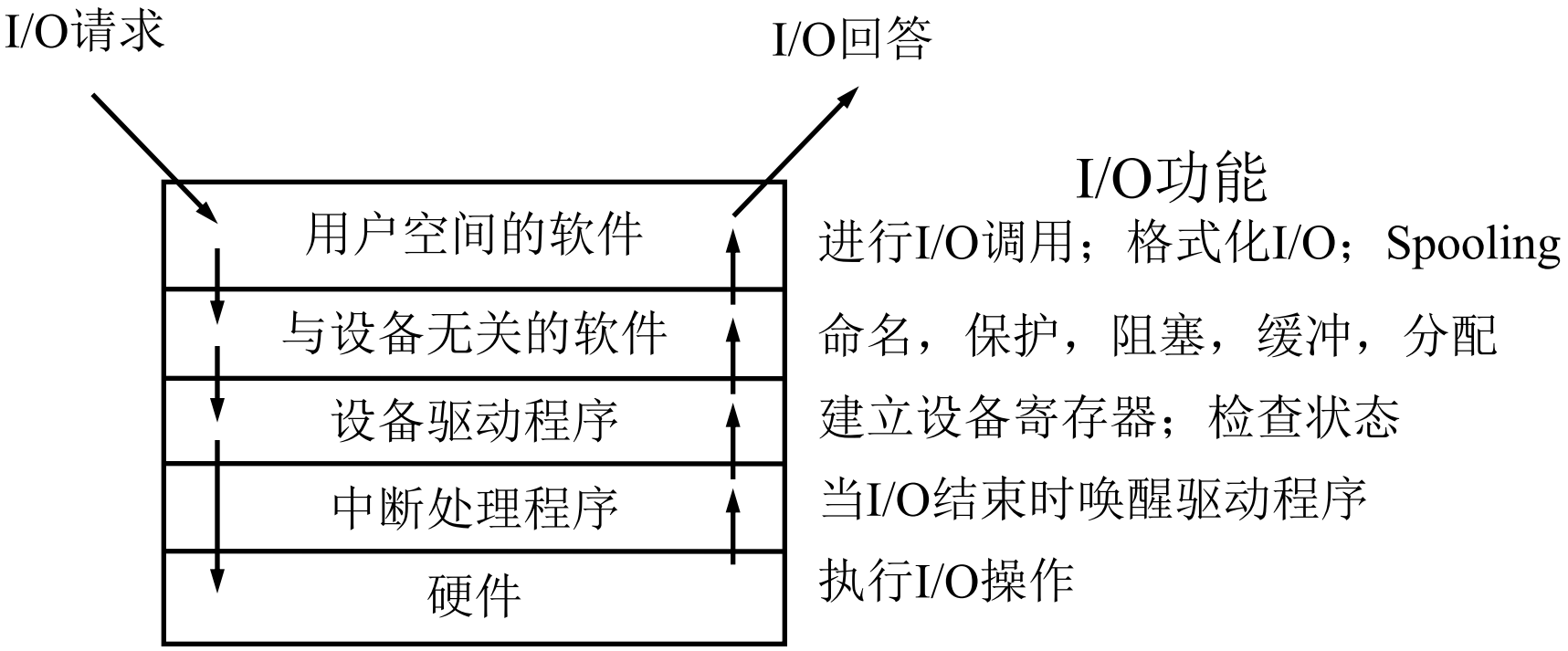
I/O软件的层次结构

- I/O与设备无关的三个层次
  - 接受内核中其它子系统的管理，虚拟化
    - 驱动程序的接口是系统定义的，统一的标准的接口
    - 经过内核某个子系统进一步虚拟化，逻辑设备与物理设备多对一，用户程序只见逻辑设备，不关心物理设备
    - 使用前对逻辑设备进行分配
    - 例：屏幕（窗口）、磁盘（文件）、网卡（套接字）、定时器（软件定时器对象）

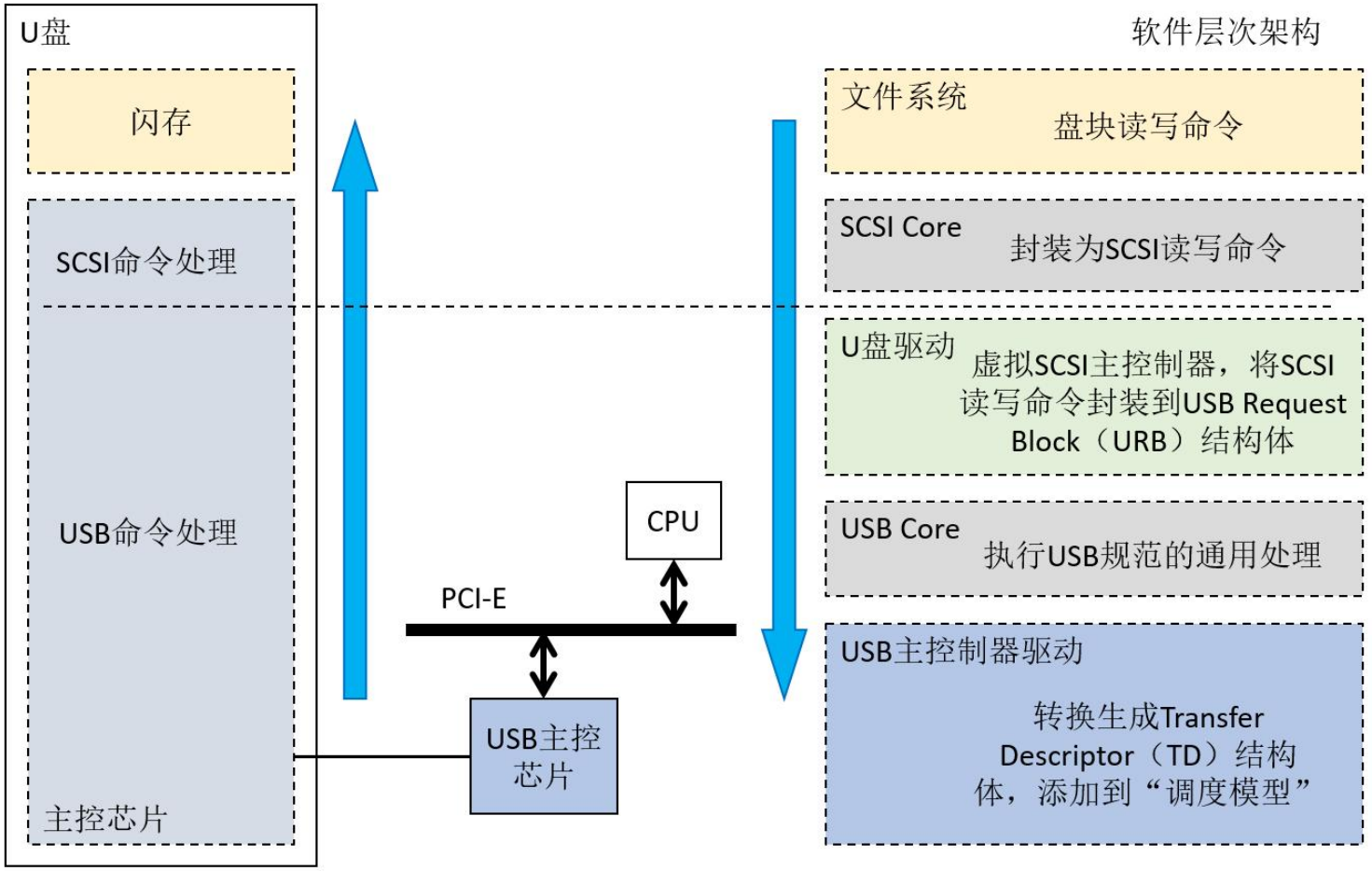


I/O软件的层次结构

- 层次结构图



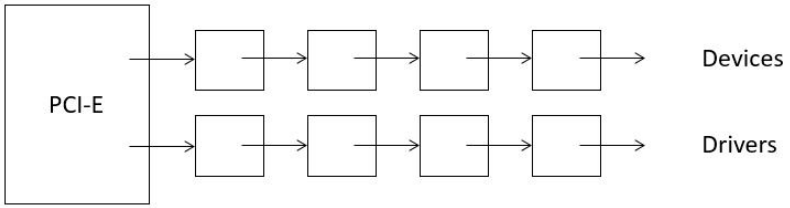
案例：Linux下U盘驱动程序架构

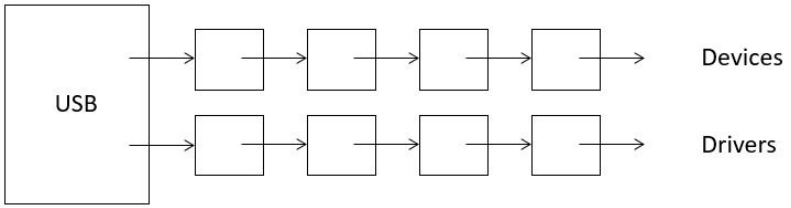


调度模型：驱动程序在内存中组织的庞大数据结构，USB主控芯片可访问并识别、处理其中的命令

案例：Linux下U盘驱动程序架构

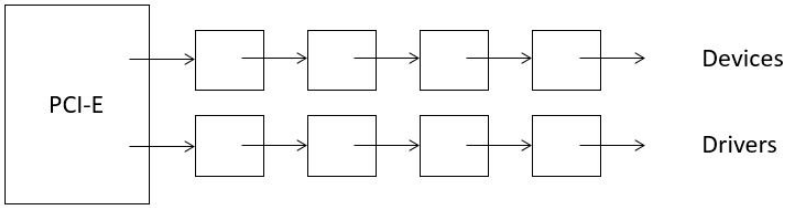
- Linux I/O子系统，设备模型
  - 每种总线结构体，包含两个链表：驱动结构体链表，设备结构体链表。
    - 对应数据类型：struct bus_type、struct device、struct device_driver
    - USB总线有USB Core模块（见前页灰色模块，属设备独立性软件，独立于下层具体的主控芯片，独立于上层的具体设备），类似，有PCI-E Core、SCSI Core

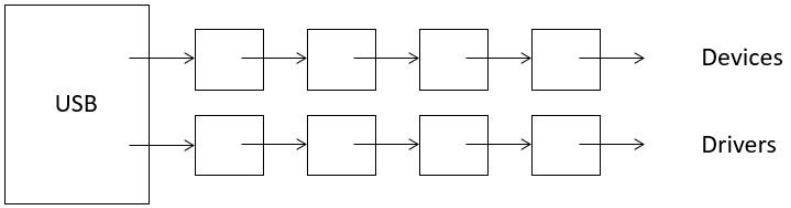




案例：Linux下U盘驱动程序架构

- Linux I/O子系统，设备模型
  - 每种总线结构体，包含两个链表：驱动结构体链表，设备结构体链表。
    - 各种总线（支持即插即用）检测到设备后，都要通过一个流程探测设备种类、参数指标等，总线的Core模块探测设备时构建device结构体挂上链表
    - 驱动程序（一个内核模块）通常随系统启动时加载，加载时构建device_driver结构体挂上链表





案例：Linux下U盘驱动程序架构

- Linux的驱动程序
  - 驱动程序的核心就是一套回调函数（函数指针）和用于匹配设备的参数指标数组，例如：

U盘驱动结构体

```
struct usb_driver usb_storage_driver = {
        .owner =        THIS_MODULE,
        .name =         "usb-storage",
        .probe =        storage_probe,
        .disconnect =   storage_disconnect,
        .id_table =     storage_usb_ids,
};
```

USB Core要求提供的函数指针 ---------->

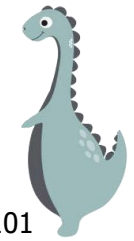
<---------- 驱动程序提供的回调函数

USB总线控制器（UHCI）驱动结构体

```
static struct pci_driver uhci_pci_driver = {
        .name =         (char *)hcd_name,
        .id_table =     uhci_pci_ids,
        .probe =        usb_hcd_pci_probe,
        .remove =       usb_hcd_pci_remove,
        .shutdown =     uhci_shutdown,
        ...
};
```

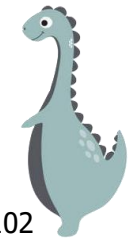
PCI-E Core要求提供的函数指针 ---------->

<---------- 驱动程序提供的回调函数



案例：Linux下U盘驱动程序架构

- Linux的驱动程序
  - 驱动匹配与设备初始化
    - 如上页所示，各种驱动结构体中都有id_table，一个数组，存放该驱动可以匹配的设备的参数指标
    - 如前述，总线的Core模块执行“即插即用”探测，已获取一些基础信息保存于device结构体
    - Core模块把基础信息与链表上的每个驱动结构体的id_table做匹配，即可找到匹配的驱动
    - 如上页所示，各种设备驱动都包括一个probe回调函数，接下来，总线Core模块调用驱动的probe回调函数，从而进一步获取设备详情，完成设备初始化



案例：Linux下U盘驱动程序架构

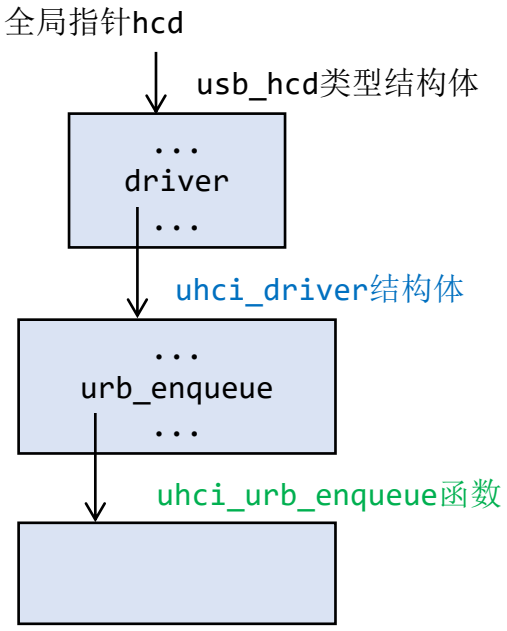
```
// 作为PCI-E总线设备的驱动结构体，处理PCI-E事务
static struct pci_driver uhci_pci_driver = {
    ...
    .id_table = uhci_pci_ids,
    .probe =    usb_hcd_pci_probe,
    ...
};

int usb_hcd_pci_probe (...) {
    ...
}

static const struct pci_device_id uhci_pci_ids[] = {
    {..., .driver_data = (unsigned long) &uhci_driver,},
    ...
};

// 作为USB主控制器的驱动结构体，处理USB事务
static const struct hc_driver uhci_driver = {
    ...
    .urb_enqueue = uhci_urb_enqueue,
    ...
};

static int uhci_urb_enqueue(...) {
    ...
}
```

例：UHCI匹配驱动及初始化过程

1）系统启动，PCI-E即插即用检测UHCI ，构建device结构体，插入设备模型链表；UHCI驱动随系统加载，插入设备模型链表。

2）PCI-E Core将device结构体与链表上各驱动的id_table比对，找到UHCI驱动（uhci_pci_driver），执行其probe回调函数，即usb_hcd_pci_probe()。

3）probe回调函数执行，造成如下局面：



案例：Linux下U盘驱动程序架构

武汉大学计算机学院 School of Computer Science, Wuhan University

各层间的接口与回调函数

SCSI Core
封装SCSI读写命令

SCSI Core要求下层提供queuecommand回调函数
SCSI Core调用该函数，向下层提交SCSI读写命令

U盘驱动
虚拟SCSI主控制器，将SCSI读写命令封装到USB Request Block（URB）结构体

U盘插入后，经过执行probe回调函数，queuecommand指向驱动中的函数：`static int queuecommand(...)`

调用：`usb_submit_urb(us->current_urb, ...)`

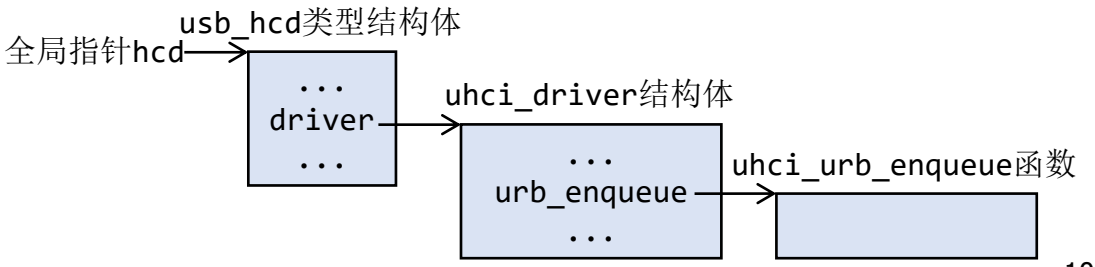
USB Core
执行USB规范的通用处理

定义：`int usb_submit_urb(struct urb *urb, ...)`

要求下层提供urb_enqueue回调函数
调用：`hcd->driver->urb_enqueue(hcd, ..., urb, ...)`

USB主控制器驱动
转换生成Transfer Descriptor（TD）结构体，添加到“调度模型”

如前页所述，经过系统启动，PCI-E检测、匹配驱动、执行probe回调函数，完成初始化后形成如下局面：



案例：Linux下U盘驱动程序架构

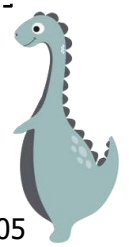
- U盘驱动的内核线程
  - U盘的probe初始化函数：创建内核线程

```
kernel_thread(usb_stor_control_thread, ...);
```

  - 内核线程无限循环：阻塞-苏醒-处理传输任务
  - 收到SCSI命令：queuecommand回调函数唤醒

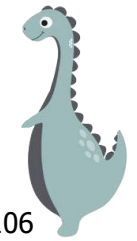
```
static int usb_stor_control_thread(...) {
    ...
    for(;;) {
        ...
        //信号量
        down_interruptible(&us->sema)
        ...
        //执行传输，最终调用usb_submit_urb()
        us->proto_handler(us->srb, us);
        ...
    }
    ...
}
```

```
static int queuecommand(...) {
    ...
    //srb即SCSI request block
    us->srb = srb;
    //信号量
    up(&(us->sema));
    ...
}
```



补充：Linux中断处理上、下半部机制

- 概述
  - 问题
    - 中断处理不能太久，期间会阻碍其它中断的响应
    - 但有些任务就是需要较长时间
  - 解决方案：中断处理的上、下半场机制
    - 所有中断由OS内核统一接收处理，不同的驱动程序注册登记不同中断号的处理程序，内核的中断处理程序分发至驱动程序的回调函数
    - 中断处理期间要关闭中断，为此，处理必须非常简短，只做必要的处理，引入下半部（bottom half）机制，登记并处理剩余的工作

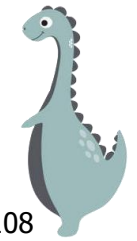


补充：Linux中断处理上、下半部机制

- 机制1：softirq（软中断）
  - 此处“软中断”与系统调用软中断指令无关
  - 登记
    - 预先定义最多32种软中断及其处理函数，并设置一32位任务向量
    - 驱动程序可将某位置为1，表示有此类软中断有待处理

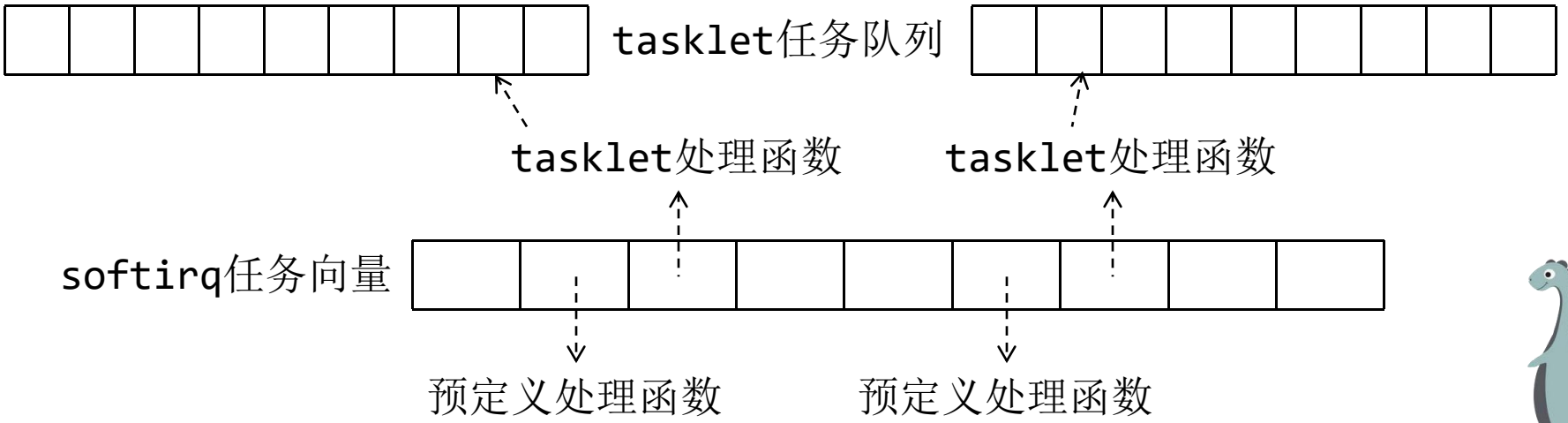
补充：Linux中断处理上、下半部机制

- 机制1：softirq（软中断）
  - 处理
    - 上半部结束后，中断处理程序开放中断，检查softirq任务向量，调用相应的处理函数，即进入下半部
    - 软中断仍在中断上下文中，只是不妨碍嵌套处理其它中断，但（在本CPU上）不会被另一个软中断抢占
    - 多CPU环境中，softirq遵循谁触发，谁执行的原则
  - 应用情况
    - softirq必须是系统内核编译前预先定义的，不利于驱动程序的灵活发挥，为此扩展出tasklet机制
    - 目前只有网络和SCSI直接使用softirq机制



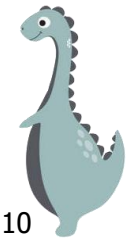
补充：Linux中断处理上、下半部机制

- 机制2：tasklet
  - 基本机制
    - 本质上是一个softirq，但是其处理程序设置了一个tasklet任务队列，允许驱动程序灵活登记
    - 其实占用了两个softirq，因而有两个tasklet任务队列，只是优先级不同



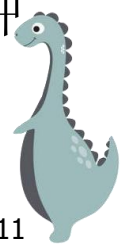
补充：Linux中断处理上、下半部机制

- 机制2：tasklet
  - 其它特征
    - 提供加锁机制，不同类的tasklet可以在多个CPU上并行，同类的tasklet不可并行
    - 处于中断上下文，可以用自旋锁，不可以阻塞、睡眠，不可以使用信号量



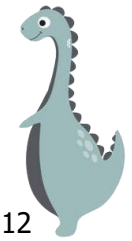
补充：Linux中断处理上、下半部机制

- 机制3：workqueue
  - 基本机制
    - 也提供了任务队列，称为工作队列，驱动的中断处理程序可以将剩余任务登记在此
    - 同时提供了后台worker线程池，随着线程调度，在后台线程的上下文处理剩余任务
  - 其它特征
    - 工作于内核线程上下文，可以阻塞、睡眠，可以使用信号量，也可以用自旋锁
    - 虽然不工作于中断上下文，也被算作下半部机制的一种



习题

- 教材习题
  - DMA如何提高系统并发性？它如何使硬件设计复杂化？
  - 为什么随着CPU速度的提高，提高系统总线和设备速度很重要？



习题

- 选择题
  - 缓冲技术中的缓冲池在 _____ 中。

    A. 主存　　B. 外存

    C. ROM　　D. 寄存器
  - CPU输出数据的速度远远高于打印机的打印速度，为了解决这一矛盾，可采用_____ 。

    A. 并行技术　　B. 通道技术

    C. 缓冲技术　　D. 虚存技术

习题

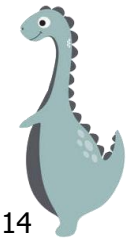
- 选择题
  - 通过硬件和软件的功能扩充，把原来独占的设备改造成能为若干用户共享的设备，这种设备称为_____。

    A. 存储设备　　B. 系统设备

    C. 用户设备　　D. 虚拟设备
  - 为了使多个进程能有效地同时处理输入/输出，最好使用 _____ 结构的缓冲技术。

    A. 循环缓冲　　B. 缓冲池

    C. 单缓冲　　D. 双缓冲



习题

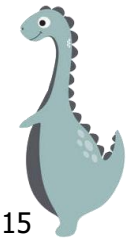
- 选择题
  - 如果I/O设备与存储设备进行数据交换不经过CPU来完成，这种数据交换方式是 _____ 。

    A. 程序查询　　B. 中断方式

    C. DMA方式　　D. 无条件存取方式
  - 在采用Spooling 技术的系统中．用户的打印结果首先被送到 _____ 。

    A. 磁盘固定区域　B. 内存固定区域

    C. 终端　　D. 打印机



习题

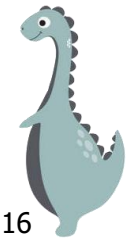
- 选择题
  - 设备管理程序对设备的管理是借助一些数据结构来进行的，下面的 _____ 不属于设备管理数据结构。

    A. DCT　　B. COCT

    C. CHCT　　D. JCB
  - 操作系统中的Spooling技术，实质是将 _____ 转化为共享设备的技术。

    A. 虚拟设备　　B. 独占设备

    C. 脱机设备　　D. 块设备



习题

- 选择题
  - 按 _____ 分类可将设备分为块设备和字符设备。

    A. 从属关系　　B. 操作特性

    C. 共享属性　　D. 信息交换单位
  - _____ 算法是设备分配常用的一种算法。

    A. 短作业优先　　B. 最佳适应

    C. 先来先服务　　D. 首次适应

习题

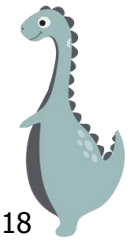
- 选择题
  - 在下面关于设备属性的论述中，正确的是_____。

A．字符设备的一个基本特征是可寻址的。

B．共享设备必须是可寻址和可随机访问的设备。

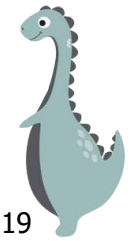
C．共享设备是指在同一时刻，允许多个进程同时访问的设备。

D．在分配共享设备和独占设备时，都可能引起进程死锁。



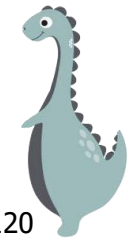
习题

- 填空题
  - 进行设备分配时所需的数据表格主要有 ①、②、③ 和 ④。
  - 引起中断发生的事件称为 _____。
  - 常用的I/O控制方式有程序直接控制方式、中断控制方式、① 和 ②。
  - 通道是一个独立于 ① 的专管 ②，它控制 ③ 与内存之间的信息交换。



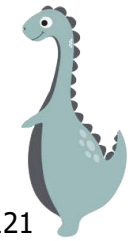
习题

- 填空题
  - SPOOLing系统是由磁盘中的 ① 和 ② ，内存中的 ③ 和 ④ 以及 ⑤ 和 ⑥ 所构成。
  - 设备分配程序分配外部设备时，先分配 ① ，再分配 ② ，最后分配 ③ 。
  - 中断方式适合于 ① ，DMA方式适合于 ② 。
  - 缓冲区的组织方式可分为 ① 、 ② 、 ③ 和缓冲池。



习题

- 填空题
  - 缓冲池中有三种类型的缓冲队列：__①__、__②__、__③__。
  - 大多数设备控制器由三部分构成：__①__、__②__、__③__。



习题

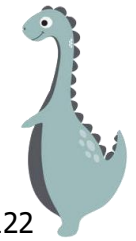
- 考研题
  - 程序员利用系统调用打开I/O设备时，通常使用的设备标识是（）。09

    A、逻辑设备名　　B、物理设备名

    C、主设备号　　D、从设备号
  - 下列选项中，能引起外部中断的事件是（）。

    A、键盘输入　　B、除数为0

    C、浮点运算下溢　　D、访存缺页

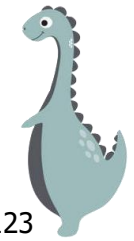


习题

- 考研题
  - 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____。10

A.命令解释程序　　B.中断处理程序

C.系统调用程序　　D.用户登陆程序



习题

- 考研题
  - 用户程序发出磁盘I/O请求后，系统的正确处理流程是_____。

    A、用户程序→系统调用处理程序→中断处理程序→设备驱动程序

    B、用户程序→系统调用处理程序→设备驱动程序→中断处理程序

    C、用户程序→设备驱动程序→系统调用处理程序→中断处理程序

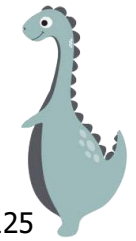
    D、用户程序→设备驱动程序→中断处理程序→系统调用处理程序

习题

- 考研题
  - 某文件占10个磁盘块，现要把该文件磁盘块逐个读入主存缓冲区，并送用户区进行分析。假设一个缓冲区与一个磁盘块大小相同，把一个磁盘块读入缓存的时间为100μs，将缓冲区的数据传送到用户区的时间是50μs，CPU对一块数据进行分析的时间是50μs。在单缓冲区及双缓冲区结构下，读入并分析完该文件的时间分别是____。11

A、1500μs，1000μs　B、1550μs，1100μs

C、1550μs，1550μs　D、2000μs，2000μs

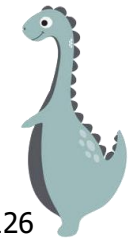


习题

- 考研题
  - 中断处理和子程序调用都需要压栈保护现场，中断处理一定会保存而子程序调用不需要保存其内容的是（）。12

    A．程序计数器　　B.程序状态寄存器

    C．通用数据寄存器　　D.通用地址寄存器



习题

- 考研题
  - 操作系统的I/O子系统通常由四个层次组成，每一层明确定义了与邻近层次的接口。其合理的层次组织排列顺序是()。12

    A、用户级I/O软件、设备无关软件、设备驱动程序、中断处理程序

    B、用户级I/O软件、设备无关软件、中断处理程序、设备驱动程序

    C、用户级I/O软件、设备驱动程序、设备无关软件、中断处理程序

    D、用户级I/O软件、中断处理程序、设备无关软件、设备驱动程序