**Lecture9 I/O管理**

**大纲**

1. I/O管理概述
2. I/O硬件组成
3. I/O软件组成
4. I/O管理相关技术
5. I/O性能问题

**问题**

1. 内核如何感知设备的状态并管理设备？

通过设备驱动程序，设备厂商进行设备对os的适配

1. 管理设备时，如何解决各类设备差异性大的问题？

建立抽象

1. 为什么要对设备建立抽象，统一对设备的访问接口？
2. 如何提高CPU 与设备的访问性能？
3. 为什么引入缓冲技术？

解决设备间以及设备与CPU间的速度差异

1. 一个I/O请求处理流程是怎样的？

**I/O管理概述**

1. I/O的特点
2. I/O的性能经常是系统性能的瓶颈
3. 是操作系统庞大复杂的原因之一
4. 资源多、杂，并发，均来自I/O

* 速度差异很大（技术迭代快）
* 应用
* 控制接口的复杂性（设备指令多样，现在常常由os定好框架，让设备厂商按照框架实现驱动程序）
* 传送单位
* 数据表示（如\r, \n, \r\n）
* 错误条件

1. 与其他功能联系密切，特别是文件系统
2. 设备的发展
3. 简单设备

CPU可通过I/O接口直接控制I/O 设备

1. 多设备

CPU与I/O设备之间增加了一层I/O 控制器和总线BUS

1. 支持中断的设备

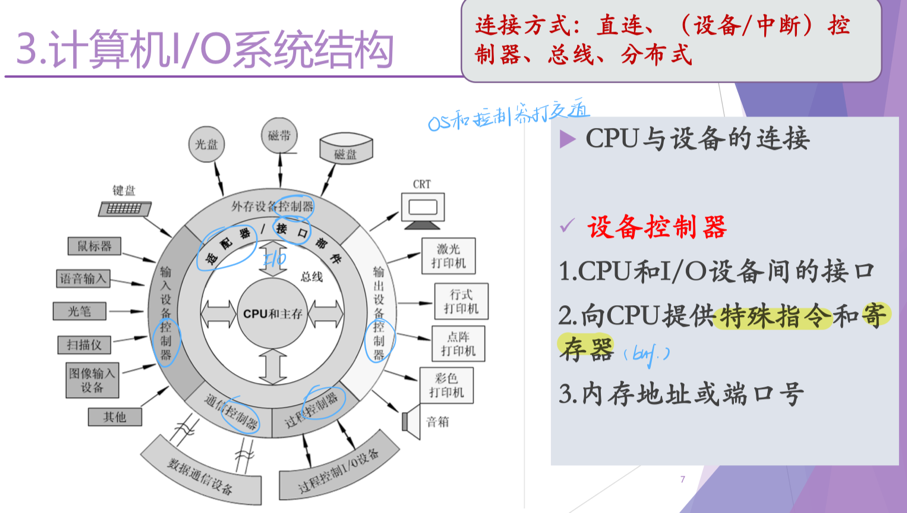
轮询变为中断，提高CPU 利用率

1. 高吞吐量设备

支持DMA

1. 各种其他设备

GPU、声卡、智能网卡、RDMA

1. 计算机I/O系统结构
2. 设备的分类
3. 按数据特征分
4. 字符设备

* 以字节为单位存储、传输信息（GPIO、键盘、鼠标、串口）
* 传输速率低、不可寻址
* I/O命令：get()；put()；通常使用文件访问接口和语义

1. 块设备（与文件系统相关）

* 以数据块为单位存储、传输信息（磁盘、磁带、光驱）
* 传输速率较高、可寻址（随机读写）
* I/O命令：原始I/O或文件系统接口；内存映射文件访问

1. 网络设备

* 格式化报文交换（以太网、无线、蓝牙）
* I/O命令：send/receive网络报文
* 通过网络接口支持多种网络协议

1. 从资源分配角度
2. 独占设备

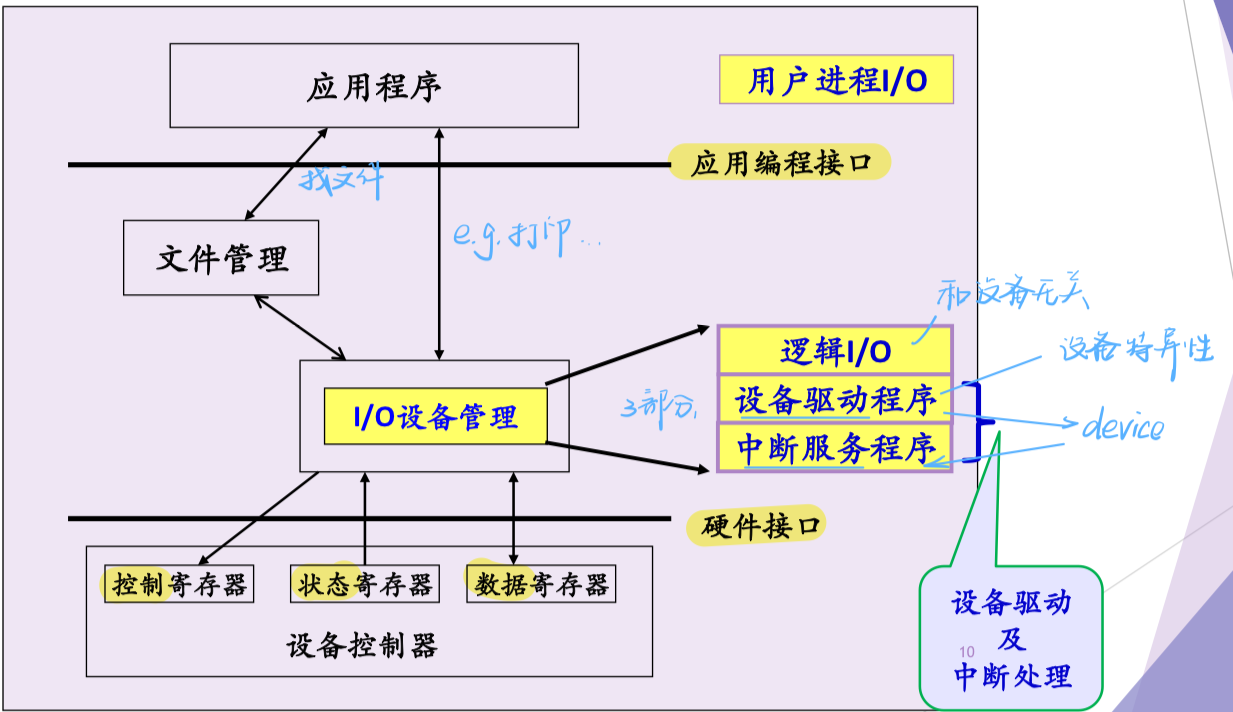
* 在一段时间内只能有一个进程使用的设备，一般为低速I/O设备（如打印机，磁带等）

1. 共享设备

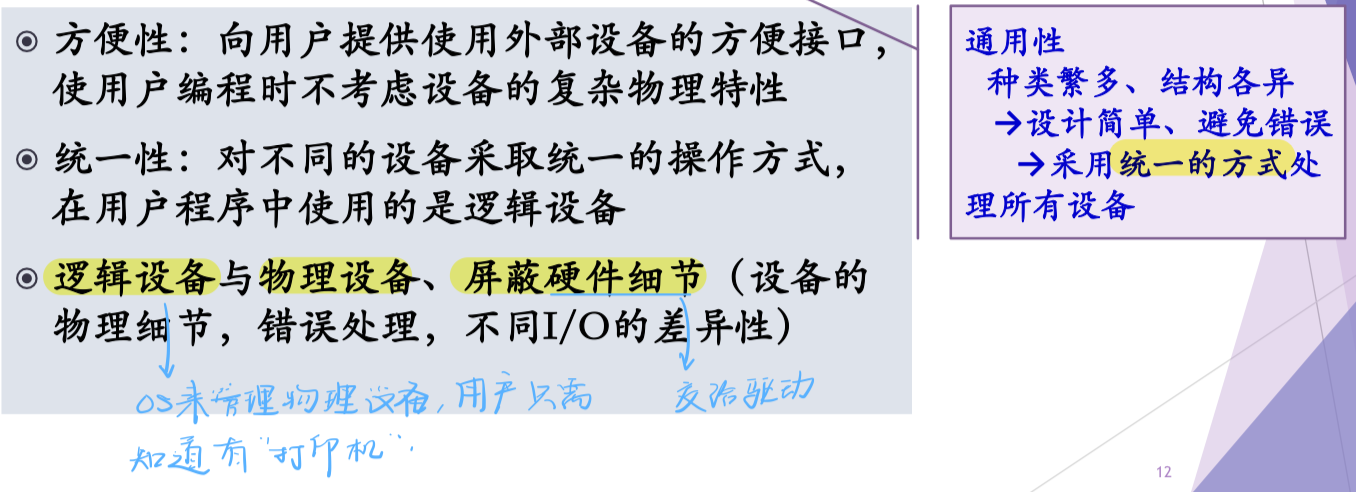
* 在一段时间内可有多个进程共同使用的设备，多个进程以交叉的方式来使用设备，其资源利用率高（如硬盘）

1. 虚设备

* 在一类设备上模拟另一类设备，常用共享设备模拟独占设备，用高速设备模拟低速设备，被模拟的设备称为虚设备
* 目的：将慢速的独占设备改造成多个用户可共享的设备，提高设备的利用率
* 实例：SPOOLing技术，利用虚设备技术——用硬盘模拟输入输出设备（硬盘作为缓冲区）

1. I/O管理
2. I/O管理的目标和任务
3. 按照用户的请求，控制设备的各种操作，完成I/O设备与内存之间的数据交换，最终完成用户的I/O请求

* 设备分配（给进程）与回收
* 记录设备的状态
* 根据用户的请求和设备的类型，采用一定的分配算法，选择一条数据通路
* 执行设备驱动程序，实现真正的I/O操作
* CPU将数据送入设备的数据寄存器，通过设备的控制寄存器启动设备
* 设备中断处理：处理外部设备的中断
* 缓冲区管理：管理I/O缓冲区

1. 建立方便、统一的独立于设备的接口（例如将设备抽象为文件）
2. 提高性能：充分利用各种技术（通道，中断，缓冲，异步I/O等）提高CPU与设备、设备与设备之间的并行工作能力，充分利用资源，提高资源利用率

* 并行性
* 均衡性（使设备充分忙碌）
* CPU与I/O的速度差别大→减少由于速度差异造成的整体性能开销→尽量使两者交叠运行

1. 保护

* 设备传送或管理的数据应该是安全的、不被破坏的、保密的

**I/O硬件组成**

1. I/O设备组成

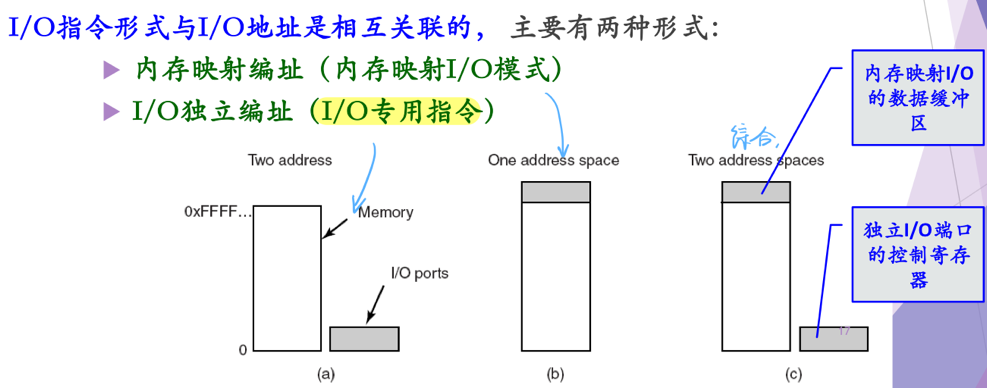
I/O设备一般由机械和电子两部分组成

1. 机械部分：设备本身（物理装置）
2. 电子部分：设备控制器(或适配器)
3. 作用

* （端口）地址译码（用于交换数据和状态等信息）
* 按照主机与设备之间约定的格式和过程接收计算机发来的数据和控制信号或向主机发送数据和状态信号
* 将计算机的数字信号转换成机械部分能识别的模拟信号，或反之（数模转换）
* 实现设备内部硬件缓冲、数据加工等提高性能或增强功能

1. 实现

* 操作系统将命令写入控制器的接口寄存器（或接口缓冲区）中，以实现输入／输出，并从接口寄存器读取状态信息或结果信息
* 当控制器接受一条命令后，可独立于CPU完成指定操作，CPU可以另外执行其他计算；命令完成时，控制器产生一个中断，CPU响应中断，控制转给操作系统；通过读控制器寄存器中的信息，获得操作结果和设备状态
* 控制器与设备之间的接口常常是一个低级接口
* 控制器的任务：把串行的位流转换为字节块，并进行必要的错误修正
* 首先，控制器按位进行组装位流中的信息，然后存入控制器内部的缓冲区中形成以字节为单位的块
* 在对块验证检查和并证明无错误时，再将它复制到内存中

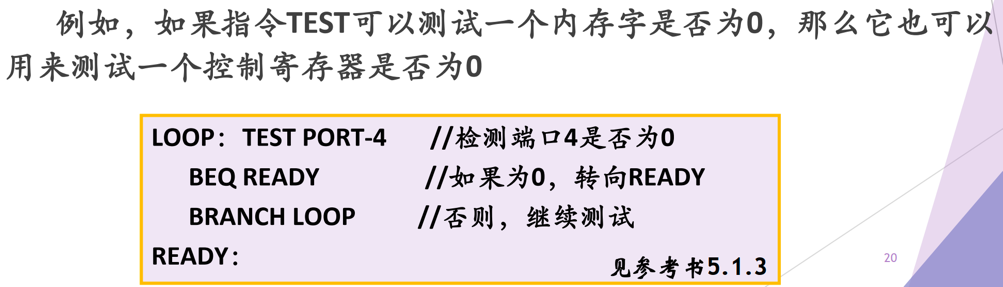
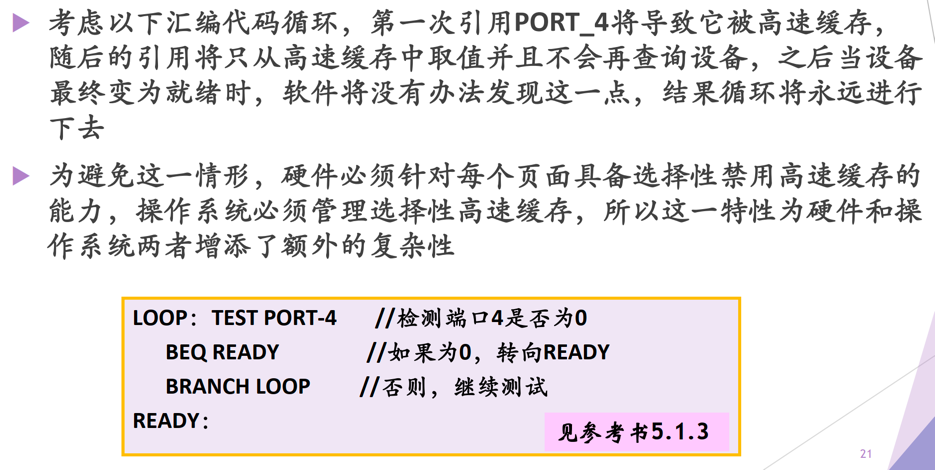
1. I/O端口地址
2. 概念
3. 指接口电路中每个寄存器具有的、唯一的地址，是个整数，亦称为I/O地址
4. 所有I/O端口地址形成I/O端口空间(受到保护)
5. I/O地址类型
6. ﻿I/O独立编址

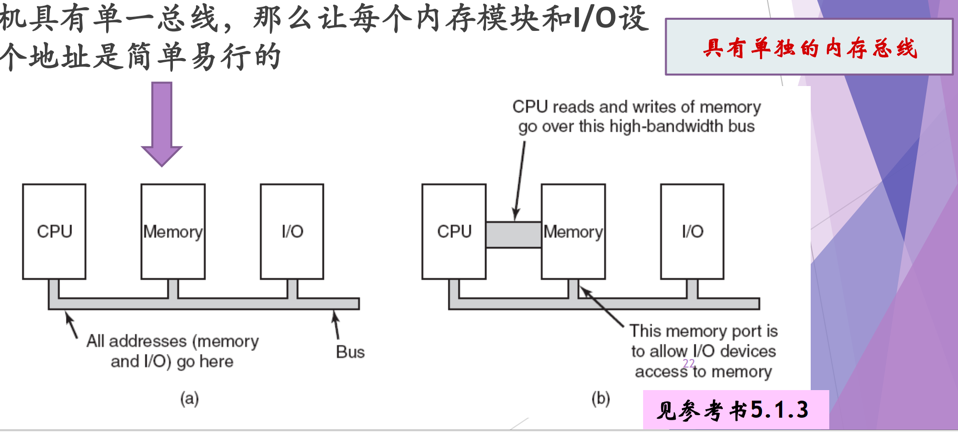
* 思想
* 分配给系统中所有端口的地址空间完全独立，与内存地址空间无关
* 使用专门的I/O指令对端口进行操作
* 优点
* 外设不占用内存的地址空间
* 编程时，易于区分是对内存操作还是对I/O端口操作
* 缺点
* I/O端口操作的指令类型少，操作不灵活
* 例子
* 8086/8088，分配给I/O端口的地址空间64K，0000H~0FFFFH，只能用in和out指令对其进行读写操作

1. ﻿内存映射编址

* 思想
* 分配给系统中所有端口的地址空间与内存的地址空间统一编址
* 把I/O端口看作一个存储单元，对I/O的读写操作等同于对内存的操作
* 优点
* 凡是可对内存操作的指令都可对I/O端口操作
* 不需要专门的I/O指令
* I/O端口可占有较大的地址空间
* 缺点
* 占用内存空间

1. 两种地址形式的对比

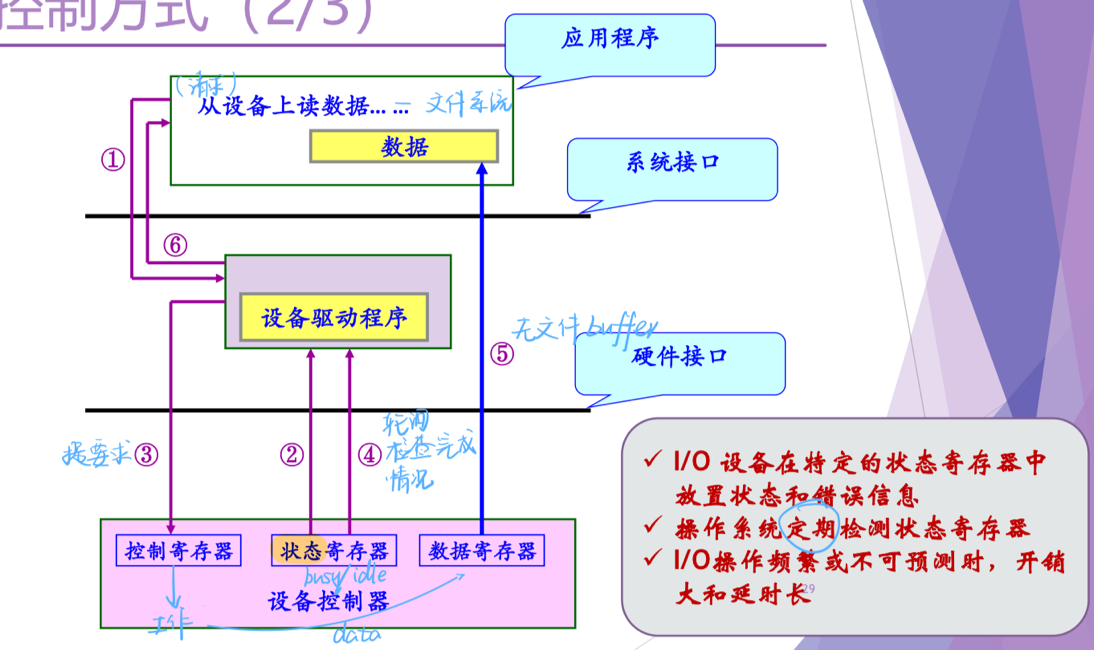
* ﻿内存映射I/O的优点
* 可以采用高级语言编写驱动程序，而不需要使用汇编语言
* ﻿不需要特殊的保护机制来阻止用户进程执行I/O操作
* 操作系统必须要做的事情：避免把包含控制寄存器的那部分地址空间放入任何用户的虚拟地址空间之中
* 可以引用内存的每一条指令也可以引用控制寄存器
* ﻿内存映射I/O的缺点
* ﻿对一个设备控制寄存器不能进行高速缓存
* ﻿如果只存在一个地址空间，那么所有的内存模块和所有的I/O设备都必须检查所有的内存引用，以便了解由谁做出响应

解决方案：如果计算机具有单一总线，那么让每个内存模块和I/O设备查看每个地址是简单易行的

1. I/O控制方式

指﻿CPU与设备控制器的数据传输

1. ﻿可编程I/O（程序控制I/O，PIO, Programmed I/O）
2. 简介

由CPU代表进程给I/O模块发I/O命令，进程进入忙等待，直到操作完成才继续执行

1. ﻿分类

* Port-mapped 的PIO(PMIO)：通过CPU 的in/out指令
* Memory-mapped的PIO(MMIO)：通过load/store传输所有数据

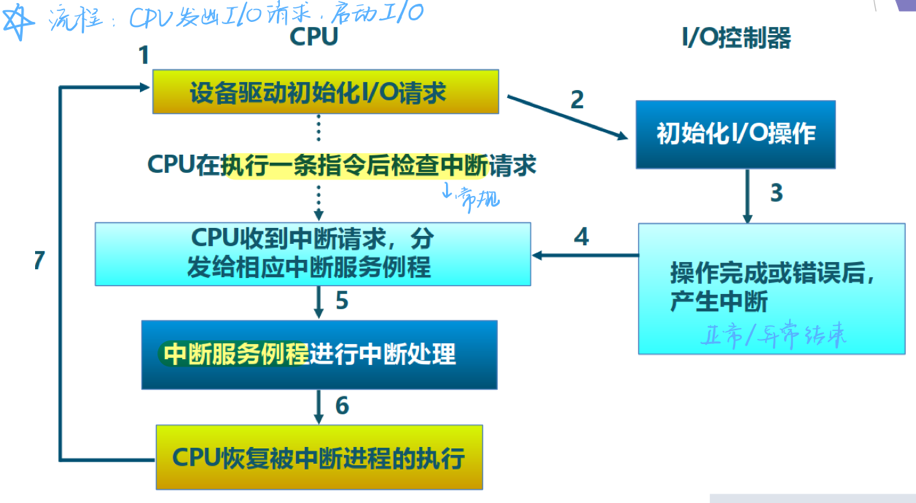
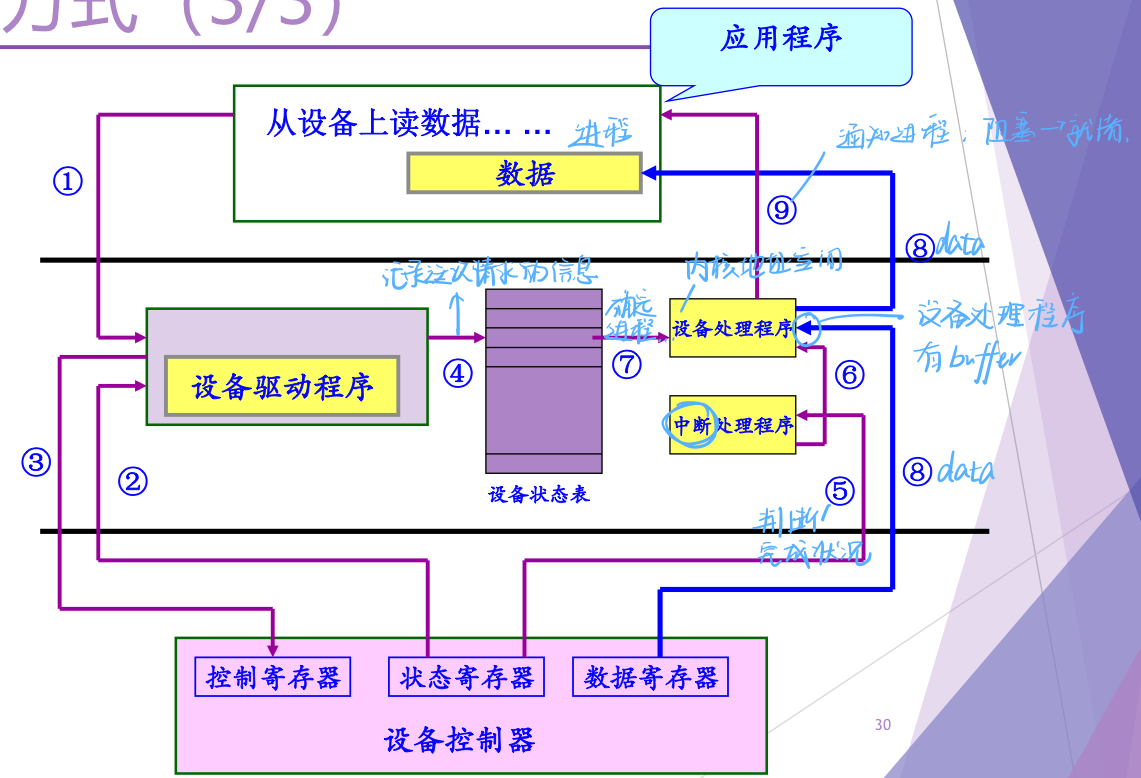
1. 优缺点

* 硬件简单，编程容易
* 消耗的CPU时间和数据量成正比
* 适用于简单的、小型的设备I/O

1. I/O设备通知CPU：PIO方式的轮询
2. 中断驱动I/O
3. 目的

减少设备驱动程序不断地询问控制器状态寄存器的开销

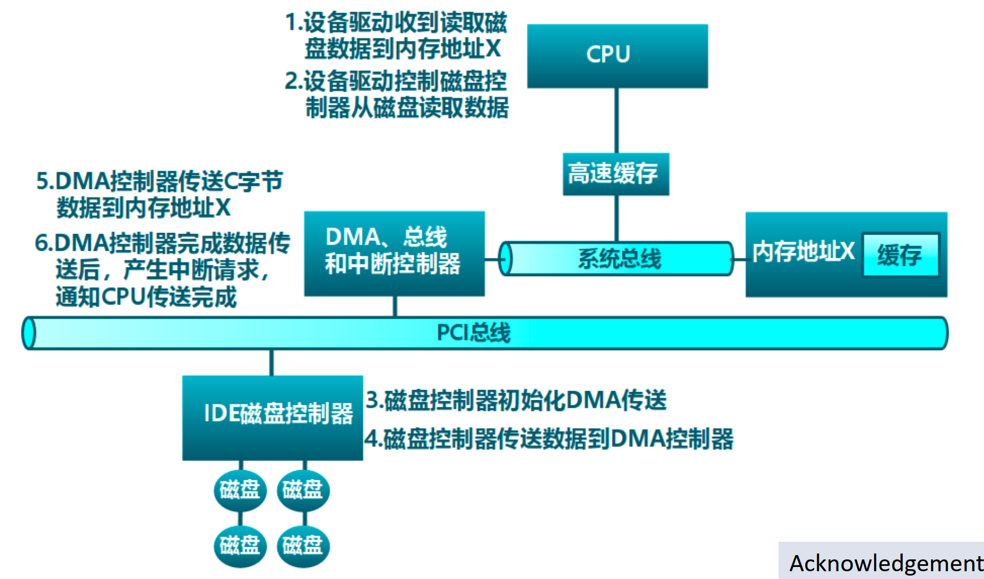
1. 实现

I/O操作结束后，由设备控制器主动通知设备驱动程序。I/O 设备向CPU发送中断请求信号：将数据准备好等状态信息通知CPU

1. 支持中断的设备类型和中断类型逐步增加
2. 优缺点

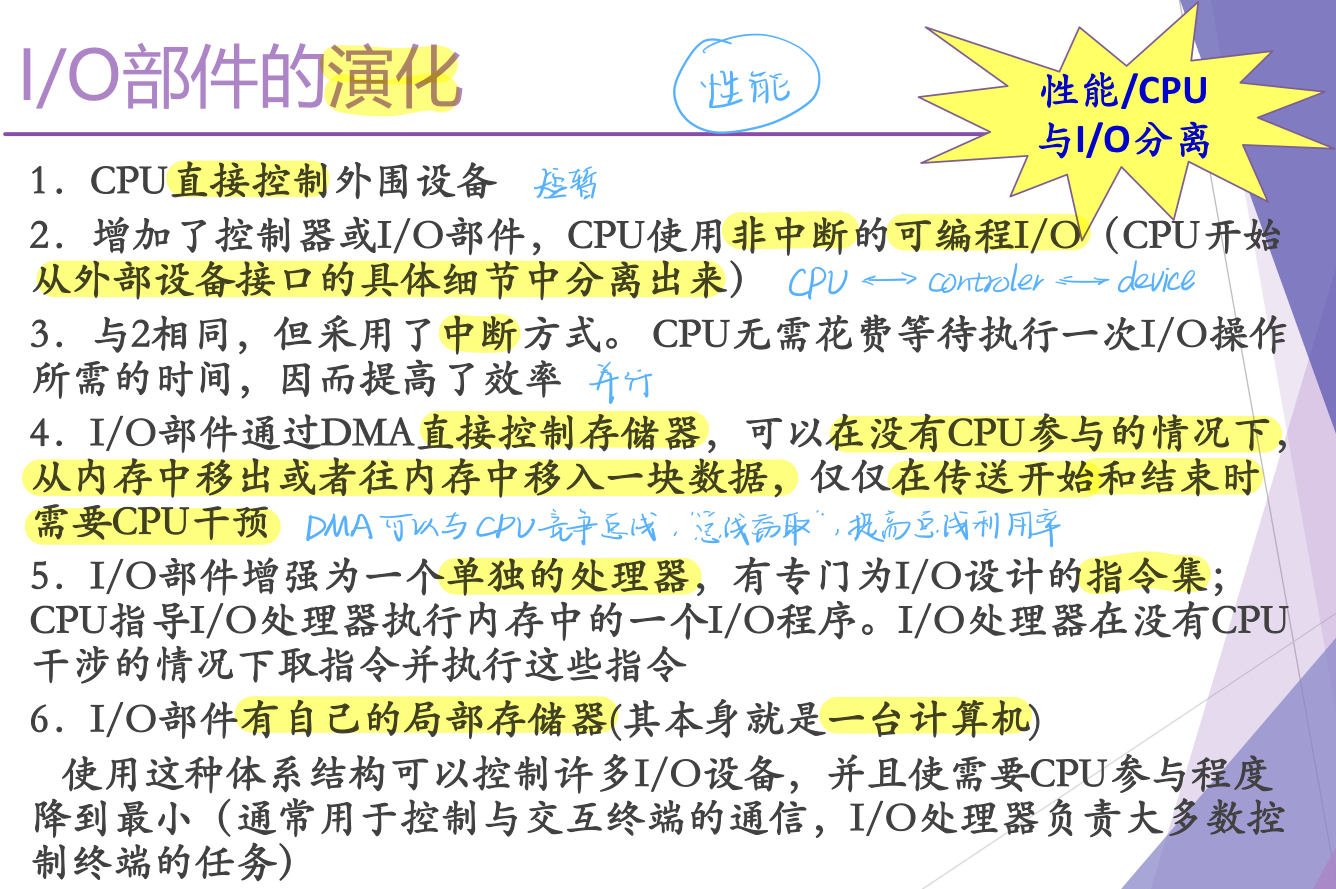
* 需要同时设置CPU 和中断控制器
* 编程比较麻烦
* CPU利用率高
* 适用于比较复杂的I/O设备

1. I/O 设备通知CPU：中断方式的提醒
2. DMA
3. ﻿简介

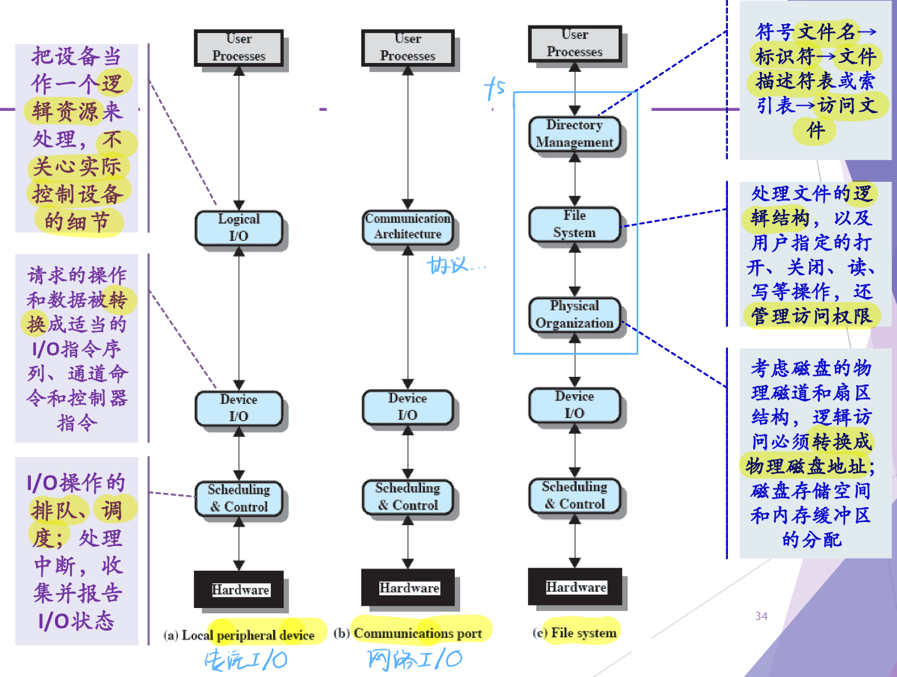
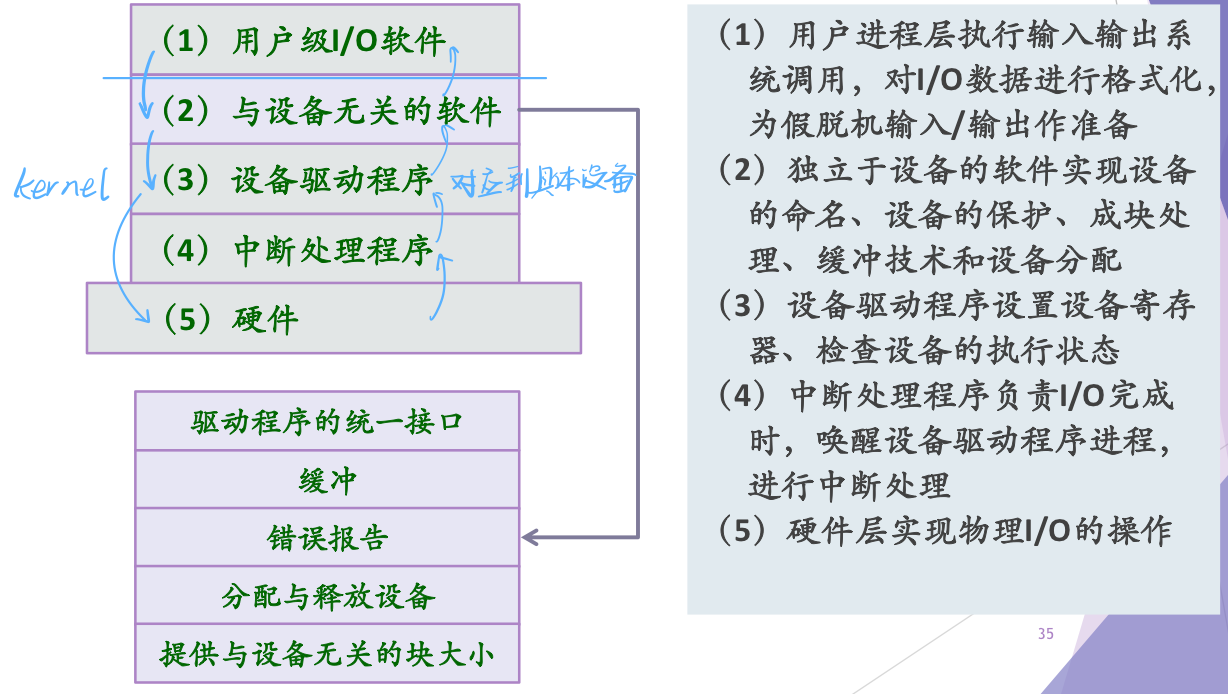
* 设备控制器可直接访问系统总线
* 控制器直接与内存互相传输数据
* 需要同时设置CPU、DMA控制器和中断控制器

1. 优缺点

* 编程比较麻烦，需要CPU参与设置
* 设备传输数据不影响CPU
* 适用于高吞吐量I/O 设备

1. I/O部件的演化

**I/O软件组成**

1. 分层的设计思想
2. 把I/O软件组织成多个层次
3. 每一层都执行操作系统所需要的功能的一个相关子集，它依赖于更低一层所执行的更原始的功能，从而可以隐藏这些功能的细节；同时，它又给高一层提供服务
4. 较低层考虑硬件的特性，并向较高层软件提供接口
5. 较高层不依赖于硬件，并向用户提供一个友好的、清晰的、简单的、功能更强的接口
6. I/O软件层次
7. ﻿设备独立性（设备无关性）
8. ﻿概念

用户编写的程序可以访问任意I/O设备，无需事先指定设备

1. ﻿用户角度

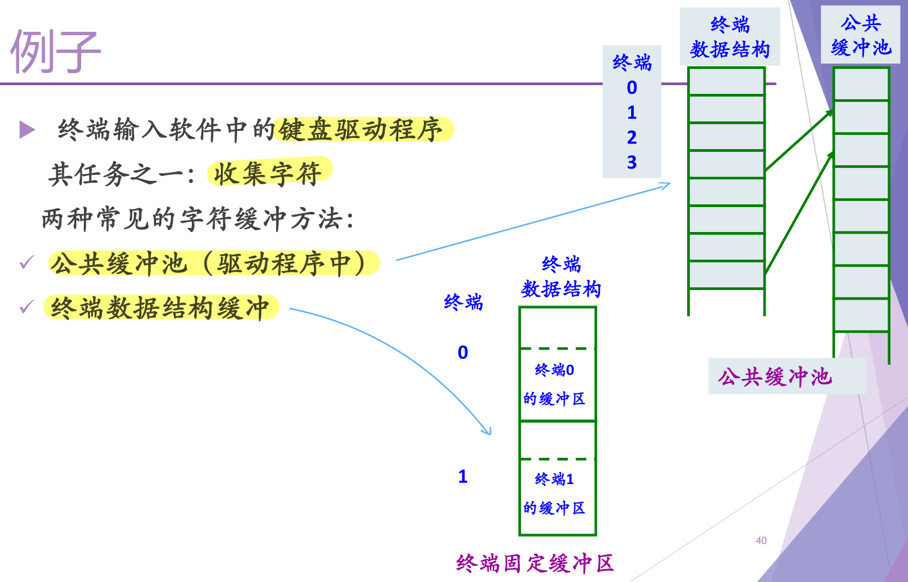
用户在编制程序时，使用逻辑设备名，由系统实现从逻辑设备到物理设备（实际设备）的转换（一个逻辑设备可以对应多个物理设备），并实施I/O操作

1. 系统角度

设计并实现I/O软件时， 除了直接与设备打交道的低层软件之外，其他部分的软件不依赖于硬件（封装硬件细节）

1. 优点
2. ﻿设备分配时的灵活性
3. 易于实现I/O重定向

**I/O相关技术**

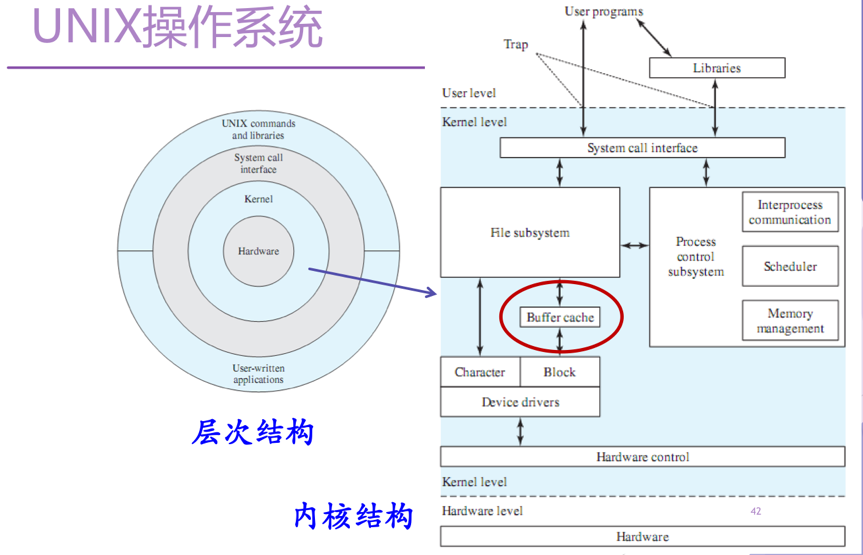
1. 缓冲技术
2. 简介
3. ﻿操作系统中最早引入的技术
4. 用于解决CPU与I/O设备之间速度的不匹配问题
5. 凡是数据到达和离去速度不匹配的地方均可采用缓冲技术
6. 作用
7. 提高CPU与I/O设备之间的并行性
8. 减少了I/O设备对CPU的中断请求次数，放宽CPU对中断响应时间的要求
9. 实现：缓冲区
10. 缓冲区的分类

* ﻿硬缓冲：由硬件寄存器实现（例如：设备中设置的缓冲区）
* 软缓冲：在内存中开辟一个空间，用作缓冲区

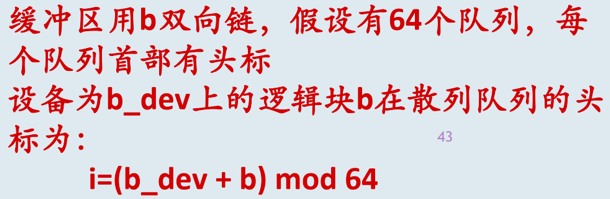
1. 缓冲区管理

* 单缓冲（I/O）
* 双缓冲（I+O）
* 缓冲池（多缓冲，循环缓冲）：统一管理多个缓冲区，采用有界缓冲区的生产者/消费者模型对缓冲池中的缓冲区进行循环使用

1. ﻿UNIX System V 缓冲技术
2. 简介

* 采用缓冲池技术，可平滑和加快信息在内存和磁盘之间的传输
* 缓冲区结合提前读（预取）和延迟写技术对具有重复性及阵发性I/O进程，提高I/O速度很有帮助（减少访问设备的次数）
* 可以充分利用之前从磁盘读入、虽已传入用户区但仍在缓冲区的数据（尽可能减少磁盘I/O的次数，提高系统运行的速度）

1. 缓冲池和缓冲区

* ﻿缓冲池
* 200个缓冲区(512字节或1024字节/个)
* ﻿系统通过缓冲控制块来实现对缓冲区的管理
* 缓冲区
* 缓冲控制块或缓冲首部
* 缓冲数据区

1. 空闲缓冲区队列（av链）

队列头部为bfreelist

1. 设备缓冲队列（b链）

链接所有分配给各类设备使用的缓冲区，按照散列方式组织

1. ﻿逻辑设备号和盘块号

分别标志出文件系统和数据所在的盘块号，是缓冲区的唯一标志

1. 状态项

指明该缓冲区当前的状态：忙/闲、上锁/开锁、是否延迟写、数据有效性等

1. ﻿两组指针（av和b）

* 用于对缓冲池的分配管理
* 每个缓冲区可以同时在av链和b链
* 开始：在空闲av链(缓冲区从未被使用过时)
* 开始IO请求：在设备IO请求队列和设备对应的b链
* IO完成：在空闲av链和设备b链（不从设备b链中脱离，因为可能还有下一次访问，局部性原理）

1. 缓冲区分配：﻿近似LRU算法

* ﻿当进程想从指定的盘块读取数据时，系统根据盘块号在设备b链(散列队列)中查找
* 如找到缓冲区，则将该缓冲区状态标记为“忙”，并从空闲av队列中取下，然后完成从缓冲区到内存用户区的数据传送
* 如果在设备b链中未找到时，则从空闲av链队首摘取一个缓冲区，插入设备I/O请求队列；并从原设备b链中取下，插入由读入信息盘块号确定的新的设备b链中
* 当数据从磁盘块读入到缓冲区后，缓冲区从设备I/O请求队列取下；当系统完成从缓冲区到内存用户区的数据传送后，要把缓冲区释放，链入空闲av链队尾
* 当数据从磁盘块读入到缓冲区，并传送到内存用户区后，该缓冲区一直保留在原设备b链中，即它的数据一直有效。如它又要被使用，则又要从空闲av链中取下，使用完后插入到空闲av链队尾。如它一直未使用，则该缓冲区从空闲av链队尾慢慢升到队首，最后被重新分配，旧的盘块数据才被置换

1. ﻿设备管理有关的数据结构
2. ﻿描述设备、控制器等部件的表格

系统中常常为每一个部件、每一台设备分别设置一张表格，常称为设备表或部件控制块。这类表格具体描述设备的类型、标识符、状态，以及当前使用者的进程标识符等

1. 建立同类资源的队列

系统为了方便对I/O设备的分配管理，通常在设备表的基础上通过指针将相同物理属性的设备连成队列（称设备队列）

1. 面向进程I/O请求的动态数据结构

每当进程发出I/O请求时，系统建立一张表格（称I/O请求包），将此次I/O请求的参数填入表中，同时也将该I/O有关的系统缓冲区地址等信息填入表中。 I/O请求包随着I/O的完成而被删除

1. 建立I/O队列

如请求包队列

1. 设备的分配技术
2. ﻿独占设备的分配

在申请设备时，如果设备空闲，就将其独占，不再允许其他进程申请使用，一直等到该设备被释放，才允许被其他进程申请使用考虑效率问题，并避免由于不合理的分配策略造成死锁

1. ﻿静态分配

* 在进程运行前, 完成设备分配；运行结束时，收回设备
* 缺点：设备利用率低

1. 动态分配

* 在进程运行过程中，当用户提出设备要求时，进行分配，一旦停止使用立即收回
* 优点：效率好
* 缺点：分配策略不好时, 产生死锁

1. ﻿分时式共享设备的分配
2. ﻿所谓分时式共享就是以一次I/O为单位分时使用设备，不同进程的I/O操作请求以排队方式分时地占用设备进行I/O
3. 由于同时有多个进程同时访问，且访问频繁，就会影响整个设备使用效率，影响系统效率。因此要考虑多个访问请求到达时服务的顺序，使平均服务时间越短越好
4. ﻿设备驱动程序
5. 简介
6. ﻿与设备密切相关的代码放在设备驱动程序中，每个设备驱动程序处理一种设备类型
7. 设备驱动程序的任务是接收来自与设备无关的上层软件的抽象请求，并执行这个请求
8. 每一个控制器都设有一个或多个设备寄存器，用来存放向设备发送的命令和参数：设备驱动程序负责释放这些命令，并监督它们正确执行
9. 当设备驱动程序释放一条或多条命令后，系统有两种处理方式：多数情况下，执行设备驱动程序的进程必须等待命令完成，这样，在命令开始执行后，它阻塞自已，直到中断处理时将它解除阻塞为止；而在其他情况下，命令执行不必延迟就很快完成
10. ﻿设备驱动程序与外界的接口
11. 与操作系统的接口

为实现设备无关性，设备作为特殊文件处理。用户的I/O请求、对命令的合法性检查以及参数处理在文件系统中完成。在需要各种设备执行具体操作时，通过相应数据结构转入不同的设备驱动程序

1. 与系统引导的接口

初始化，包括分配数据结构，建立设备的请求队列

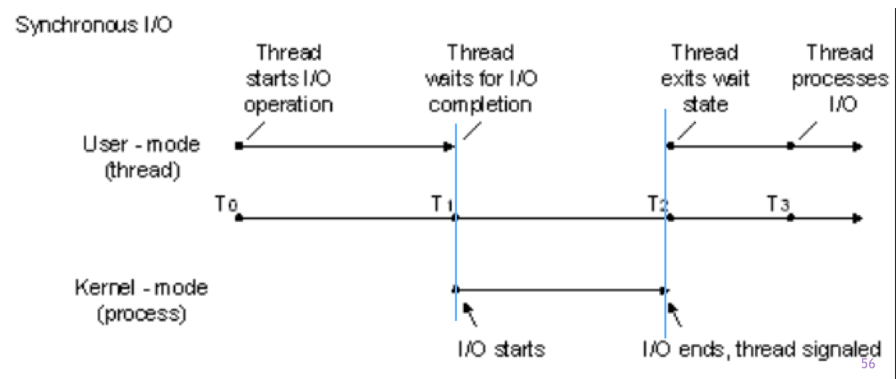
1. 与设备的接口
2. I/O进程
3. 简介
4. ﻿I/O进程：专门处理系统中的I/O请求和I/O中断工作的系统进程
5. ﻿一种典型的I/O实现方案
6. ﻿I/O请求的进入
7. 用户程序：调用send将I/O请求发送给I/O进程；调用block将自己阻塞，直到I/O任务完成后被唤醒
8. 系统：利用wakeup唤醒I/O进程，完成用户所要求的I/O处理
9. I/O中断的进入
10. 当I/O中断发生时，内核中的中断处理程序发一条消息给I/O进程，由I/O进程负责判断并处理中断
11. 进程运行过程
12. ﻿是系统进程，一般赋予最高优先级。一旦被唤醒，它可以很快抢占处理机投入运行
13. I/O进程开始运行后，首先关闭中断，然后用receive去接收消息
14. 两种情形

* 没有消息，则开中断，将自己阻塞
* 有消息，则判断消息类型（I/O请求或I/O中断）
* I/O请求：准备通道程序，发出启动I/O指令，继续判断有无消息
* I/O中断：进一步判断正常或异常结束

正常：唤醒要求进行I/O操作的进程

异常：转入相应的错误处理程序

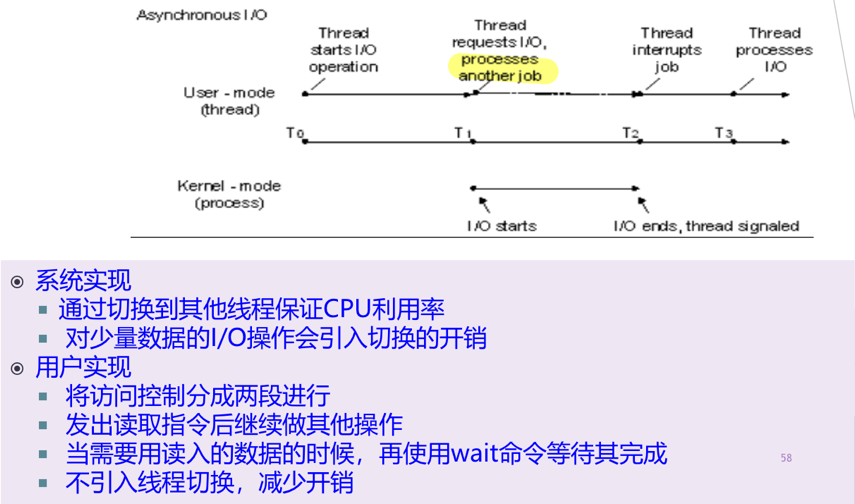
**I/O性能问题**

1. 设计思想
2. 目标
3. 使CPU利用率尽可能不被I/O降低
4. 使CPU尽可能摆脱I/O（分离）
5. ﻿采取的措施
6. ﻿减少或缓解速度差距 → 缓冲技术
7. 使CPU不等待I/O → 异步I/O（I/O多路复用）
8. 让CPU摆脱I/O操作 → DMA、通道
9. ﻿同步I/O
10. ﻿在I/O处理过程中， CPU处于空闲等待状态
11. 而在处理数据的过程中，不能同时进行I/O操作
12. ﻿异步I/O
13. 简介
14. Windows提供两种模式的I/O操作：异步和同步
15. ﻿异步I/O（非阻塞I/O）

用于优化应用程序的性能

* 通过异步I/O，应用程序可以启动一个I/O操作，然后在I/O请求执行的同时继续处理（发起I/O的进程一定会被阻塞）
* 基本思想：填充I/O操作间等待的CPU时间

1. ﻿同步I/O（阻塞I/O）

应用程序被阻塞直到I/O操作完成