第五章 I/O管理

2022年8月22日 星期一 10:24

设备概述

设备的分类

按信息交换的单位分类

块设备
- 传输速度较高+可寻址+可随机读/写任一块
- 属于有结构设备；如磁盘，硬盘，USB

字符设备
- 传输速率低+不可寻址+时常采用中断I/O方式
- 属于无结构类型；如交互式终端机，打印机，鼠标

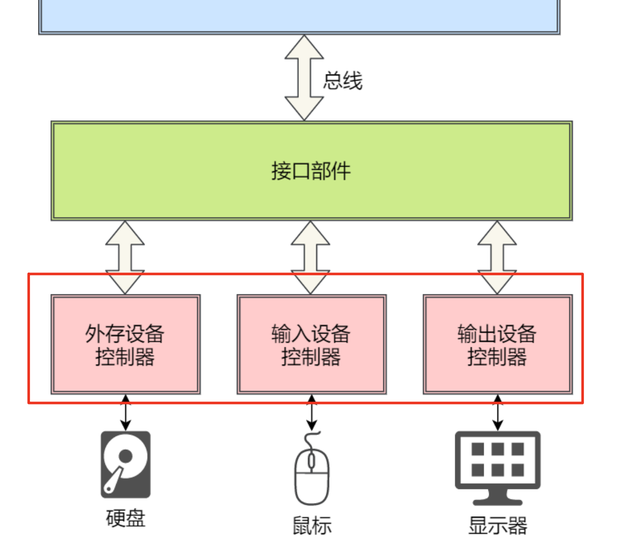
按传输速度分类

低速设备
- 键盘，鼠标

中速设备
- 激光打印机

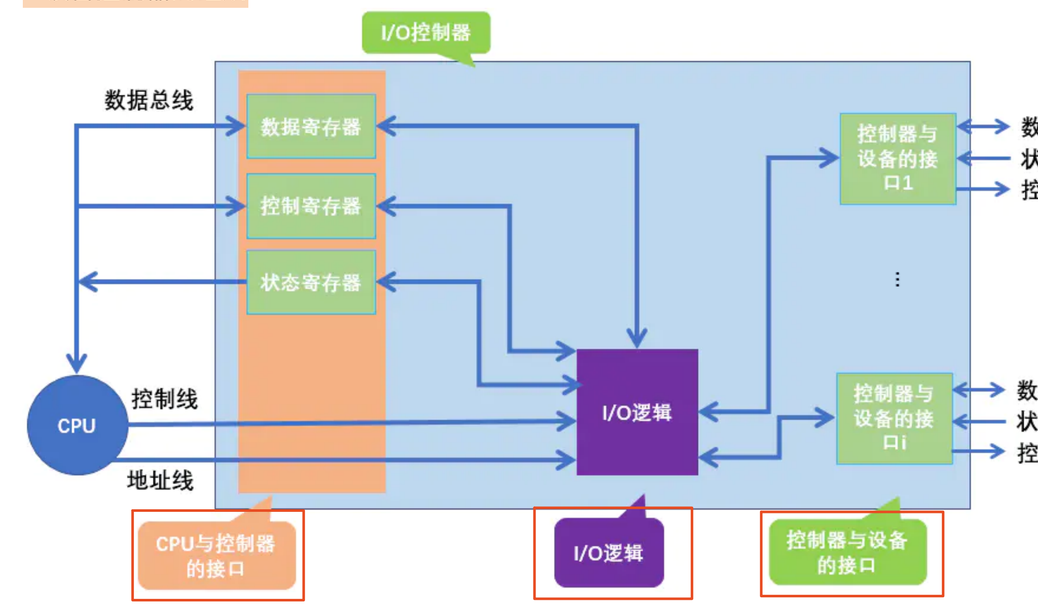
高速设备
- 磁盘机，光盘机

设备控制器(I/O接口)



1.设备控制器主要功能
- 接受和识别CPU发来的命令
- 数据交换（设备和控制器的数据交换+控制器和主存数据交换）
- 标识和报告设备的状态，以供CPU处理
- 地址识别；数据缓冲；差错控制
- 设备控制器不属于操作系统范畴，它是属于硬件

2.设备控制器的组成



左边的部分

是CPU与控制器的接口，用于实现CPU与控制器之间的通信。CPU通过控制线发出命令，通过地地址线指明要操作的设备；通过数据线来取出（输入）数据，或放入（输出）数据。

右边的部分

是控制器与设备的接口，用于实现控制器和设备之间的通信。

中间的部分

I/O逻辑是用于负责接收和识别CPU的各种命名（如地址译码），并根据CPU的命令对相应的设备发送命令。

3.设备控制器有三种寄存器(设备控制器中可被CPU直接访问的寄存器，也叫I/O端口)

数据寄存器

CPU 向 I/O 设备写入需要传输的数据

比如要打印的内容是「Hello」，CPU 就要先发送一个 H 字符给到对应的 I/O 设备。

命令寄存器

CPU 发送一个命令，告诉 I/O 设备，要进行输入/输出操作，于是就会交给 I/O 设备去工作，任务完成后，会把状态寄存器里面的状态标记为完成。

状态寄存器

目的是告诉 CPU ，现在已经在工作或工作已经完成

如果已经在工作状态，CPU 再发送数据或者命令过来，都是没有用的，直到前面的工作已经完成，状态寄存标记成已完成，CPU 才能发送下一个字符和命令。

4.CPU与I/O端口的通信方式

端口I/O（独立编制）

每个控制寄存器被分配一个 I/O 端口，可以通过特殊的汇编指令操作这些寄存器，比如 in/out 类似的指令。

内存映射I/O（统一编制）

将所有控制寄存器映射到内存空间中，这样就可以像读写内存一样读写数据缓冲区。

设备控制方式（I/O控制方式）

- 外围设备和内存之间的控制方式
- 用于控制设备和内存或CPU之间的数据传送

1.程序直接控制方式

一种轮待询等的方法，让 CPU 一直查寄存器的状态，直到状态标记为完成

2.中断驱动程序

有一个硬件的中断控制器，当设备完成任务后触发中断到中断控制器，中断控制器就通知 CPU

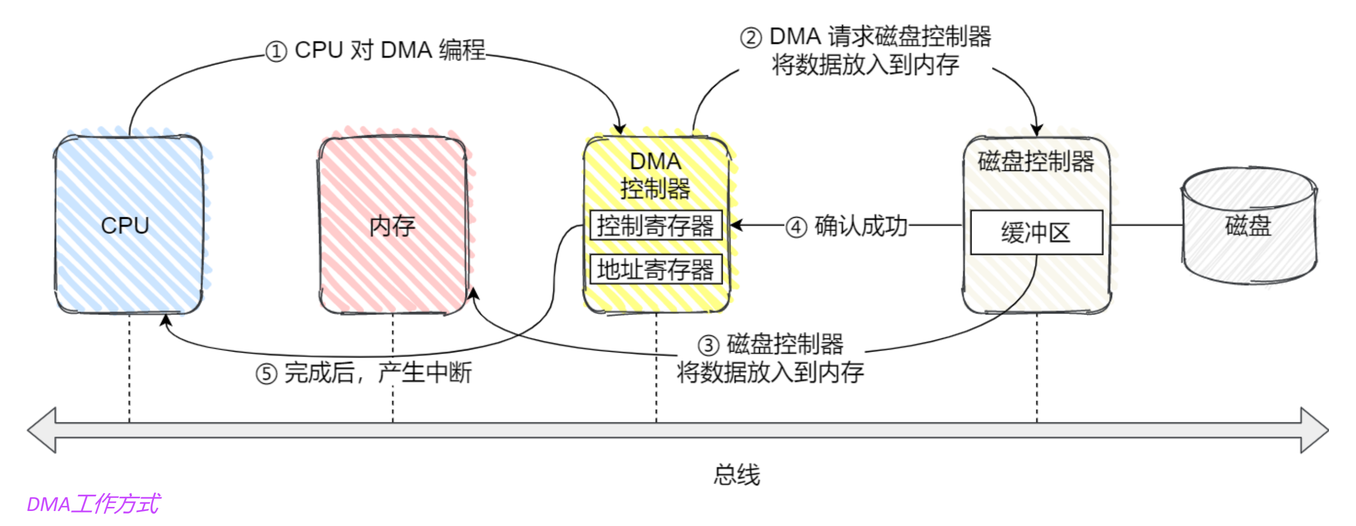
一个中断产生了，CPU 需要停下当前手里的事情来处理中断。
- 软中断，例如代码调用 INT 指令触发
- 硬件中断，就是硬件通过中断控制器触发的

中断的方式对于频繁读写数据的磁盘，并不友好，这样 CPU 容易经常被打断，会占用 CPU 大量的时间，使用DMA解决

3.DMA方式（Direct Memory Access）

DMA概念

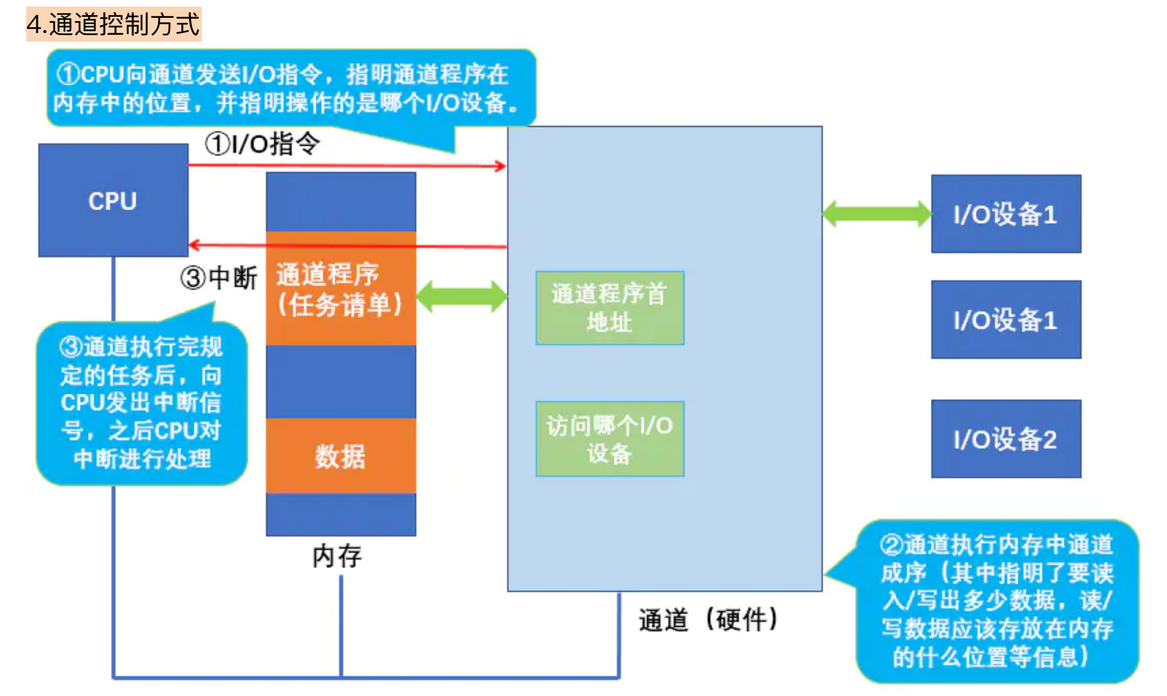
可以使得设备在 CPU 不参与的情况下，能够自行完成把设备 I/O 数据放入到内存。要实现 DMA 功能要有 DMA 控制器硬件的支持。



DMA工作方式
- CPU 需对 DMA 控制器下发指令，告诉它想读取多少数据，读完的数据放在内存的某个地方就可以了；
- 接下来，DMA 控制器会向磁盘控制器发出指令，通知它从磁盘读数据到其内部的缓冲区中，接着磁盘控制器将缓冲区的数据传输到内存；
- 当磁盘控制器把数据传输到内存的操作完成后，磁盘控制器在总线上发出一个确认成功的信号到 DMA 控制器；
- DMA 控制器收到信号后，DMA 控制器发中断通知 CPU 指令完成，CPU 就可以直接用内存里面现成的数据了；

DMA方法的特性
- 整个过程仅仅在开始和结束时需要CPU干预
- 每个DMA控制器对应一台设备与内存传递数据

4.通道控制方式



通道方式过程

当CPU要完成一组相关的读（或写）操作及相关控制时，只需向I/O通道发送一条I/O指令

以给出其所要指向的通过程序的首地址和要访问的I/O设备，之后CPU就可以去执行其他操作了

特性
- CPU、通道、I/O设备可并行工作，资源利用率很高。
- 实现复杂，需要专门的硬件支持。
- 一个通道可以控制多台设备与内存的数据交换

四种方式的比较

I/O控制方式	过程	CPU干预频率	每次传输数据的单位	数据流向	优缺点
程序直接控制方式	CPU发出指令后需要不断轮询。	极高	字	设备-CPU-内存 内存-CPU-设备	每一个阶段的优点都是解决上一个阶段的最大缺点。总体来说就是尽量减少CPU对I/O过程的干预，把CPU从繁杂的I/O控制事务中解脱出来，以便更多地去完成其他任务。
中断驱动方式	CPU发出I/O指令后可以做其他事，本次I/O完成后设备控制器发出中断信号。	高	字	设备-CPU-内存 内存-CPU-设备	
DMA方式	CPU发出I/O名命令后可以做其他事，本次I/O完成后DMA控制器发出中断信号。	中	块	设备-内存 内存-设备	
通道控制方式	CPU发出I/O指令后可以做其他事。通道会执行通道程序以完成I/O，完成后通道向CPU发出中断信号。	低	一组块	设备-内存 内存-设备	

I/O管理要完成哪些功能?

状态跟踪

要能实时掌握外部设备的状态

设备存取

要实现对设备的存取操作

设备分配

在多用户环境下，负责设备的分配与回收

设备控制

包括设备的驱动，完成和故障的中断处理

I/O软件层次结构

四种结构

用户层I/O软件

用户通过统一的接口发送命令，如发送read命令

设备独立性软件

对用户的命令进行解析，如解析read命令

设备驱动程序

针对不同的硬件将命令解析为指令，如将解析好的read命令转换为指令

中断处理程序

中断正在运行的进程，转而执行用户命令，如中断当前进程，执行read命令

层次结构的特点
- 每层都是利用其下层提供的服务，完成输入/输出功能中的某些子功能
- 屏蔽子功能实现的细节，向高层提供服务
- 仅最底层才涉及硬件的具体特性

应用程序I/O接口

字符设备接口
- 字符设备是指数据的存取和传输都是以字节为单位的设备
- 字符设备都属于独占设备

块设备接口
- 块设备是指数据的存取和传输都是以数据块为单位的设备，如磁盘
- 磁盘设备的I/O常使用DMA方式

网络设备接口
- 许多OS提供的网络I/O接口为网络套接字接口

阻塞/非阻塞I/O
- 大多数OS提供的 I/O接口都是采用阻塞 I/O

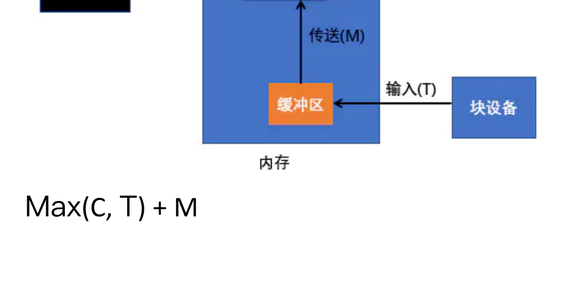
缓冲区管理

引入缓冲区的目的?
- 缓和CPU与I/O设备之间速度不匹配的矛盾
- 减少对CPU的中断频率，放宽了CPU对中断响应时间的限制
- 解决了数据粒度不匹配的问题
- 提高CPU与I/O设备之间的并行性

缓冲区实现方法?
- 采用硬件缓冲器
- 采用缓冲区(位于内存区域)

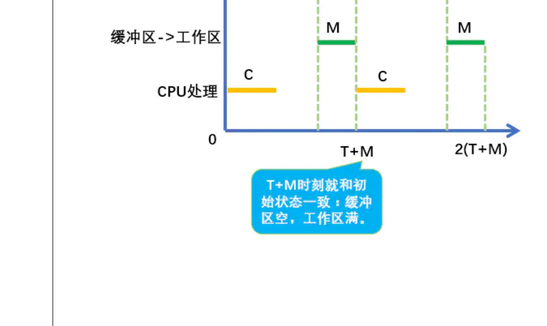
缓冲技术分类
- T时间：数据从磁盘--->缓冲区
- M时间：数据从缓冲区---->用户
- C时间：CPU对一块数据处理的时间

单缓冲

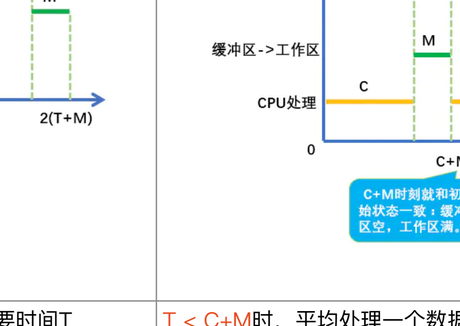


Max(C, T) + M

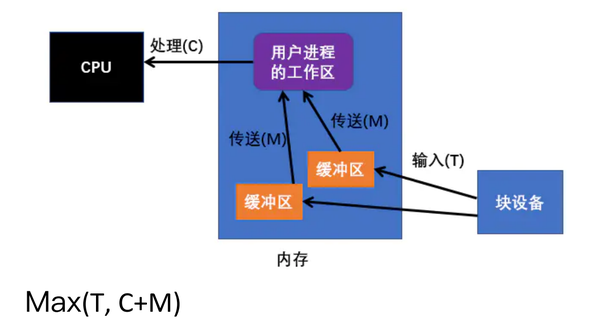
T > C时，处理一块数据平均需要（T+M）



T < C时，处理一块数据平均需要（C+M）

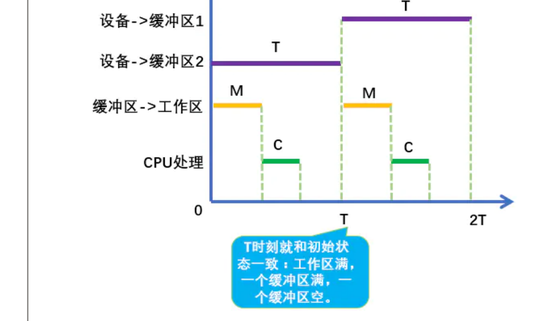


双缓冲

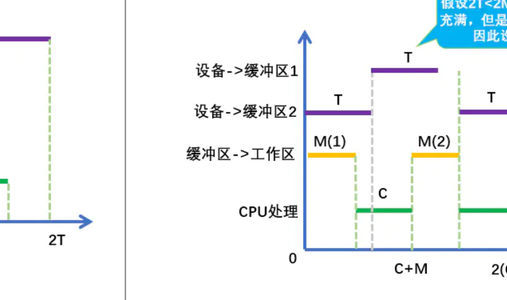


Max(T, C+M)

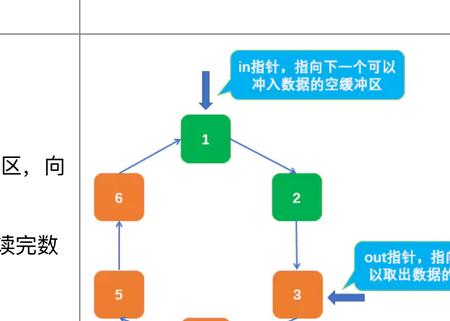
T > C+M时，平均处理一个数据块需要时间T



T < C+M时，平均处理一个数据块需要时间（C+M）



循环缓冲
- 将多个大小相等的缓冲区连接成一个循环队列。
- 下图中橙色表示已充满数据的缓冲区，绿色表示空缓冲区。
- 当需要向缓冲区中冲入数据时，只要找到in指针指向的空缓冲区，向其中冲入数据，然后再把in指针指向下一个空缓冲区。
- 当需要取出满缓冲区的内容时，找到out指针执行的满缓冲，读完数据后，将指针指向下一个满缓冲区。



缓冲池
- 缓冲池由系统中共用的缓冲区组成
- 缓冲区按使用状况可以分为
  - 空缓冲队列
  - 输入队列：存储的是从设备发送给内存的数据
  - 输出队列：存储的是从内存发送给设备的数据
- 根据一个缓冲区在实际运算中扮演的功能不同分为四种工作缓冲区
  - 用于收容输入数据的工作缓冲区（hin）
  - 用于提取输入数据的工作缓冲区（sin）
  - 用于收容输出数据的工作缓冲区（hout）
  - 用于提取输出数据的工作缓冲区（sout）

设备分配

- 设备分配指根据用户的I/O请求分配所需的设备

根据设备的特性分为三种设备
- 独占设备：一个时段只能分配给一个进程。所有字符设备都是独占设备，如输入机、打印机、磁带机等。
- 共享设备：可同时分配给多个进程使用。软硬盘、磁盘、光盘等块设备都是共享设备。
- 虚拟设备：通过软件技术将独占设备改造成共享设备。如通过 SPOOLing技术将一台打印机虚拟成多台打印机，其实质上还是独占设备。

设备分配的策略

分配的原则
- 充分发挥设备的使用效率，尽可能让设备忙碌；要避免造成进程死锁；将用户和具体设备隔开来

设备分配的方式
- 静态分配---->对独占设备的分配---->在作业执行前分配---->不会出现死锁---->设备的使用率低
- 动态分配--->在进程执行中分配，通过系统调用发送请求，根据具体策略分配设备---->分配算法不好时会出现死锁---->设备利用率高

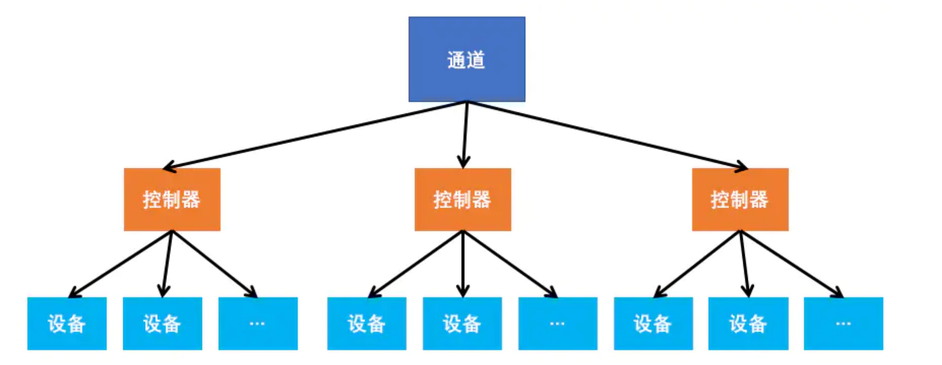
设备分配算法
- 先请求先分配，优先级高者分配。。。。(类比调度算法)

设备分配的安全性
- 设备分配安全性是指设备分配中应防止发生进程死锁

安全分配方式
- 为进程分配一个设备后就将该进程阻塞，本次I/O完成后才将进程唤醒
- 安全分配方式在一个时间段内每个进程只能使用一个设备。
- 优点：破坏了请求等待条件，不会死锁。
- 缺点：对于一个进程来说，CPU和I/O设备只能串行工作，系统资源利用率低。

不安全分配方式
- 进程发出I/O请求后，系统为其分配I/O设备，进程可继续执行，之后还可以发出新的I/O请求，只有某个I/O请求得不到满足时才将进程阻塞。
- 不安全分配方式一个进程可以同时使用多个设备。
- 优点：进程的计算任务和I/O任务可以并行处理，使进程推进。
- 缺点：有可能发生死锁。

设备分配依据的数据结构
- 设备，控制器，通道之间的关系



设备控制表（DCT）
- 系统为每个设备配置一张DCT，用于记录设备情况。

设备控制表（DCT）	
设备类型	如：打印机/扫描仪/键盘
设备标识符	即物理设备名，系统中的每个设备的物理设备名唯一。
设备状态	忙碌/空闲/故障...
指向控制器表的指针	每个设备由一个控制器控制，该指针可以找到相应的控制器信息。
重复执行次数或时间	当重复执行多次I/O操作仍不成功，才认为I/O失败。
设备队列的队首指针	指向正在等待该设备的进程队列，由进程PCB组成队列。

控制器控制表（COCT）
- 每个设备控制器都会对应一张COCT。操作系统会根据COCT的信息对控制器进行操作管理。

控制器控制表（COCT）	
控制器标识符	各个控制器的唯一ID。
控制器状态	忙碌/空闲/故障...
指向通道表的指针	每个设备由一个通道控制，该指针可以找到相应的控制器信息。
控制器队列的队首指针	指向正在等待该控制器进程的队列，由进程PCB组成队列。
控制器队列的队尾指针	

通道控制表（CHCT）
- 每个通道都会对应一张CHCT。操作系统会根据CHCT的信息对通道进行操作管理。

通道控制表（CHCT）	
通道标识符	各个通道的唯一ID
通道状态	忙碌/空闲/故障...
与通道连接的控制器表首址	可通过该指针找到该通道管理的所有控制器相关信息（COCT）
通道队列的队首指针	指向正在等待该通道进程的队列，由进程PCB组成队列
通道队列的队尾指针	

系统设备表（SDT）
- 记录了系统中全部设备的情况，每个设备对应一个表目。

系统控制表（SDT）
表目1
表目2
……
表目i
……

表目i	
设备类型	如打印机/扫描仪...
设备标识符	即物理设备名
DCT（设备控制表）	
驱动程序入口	

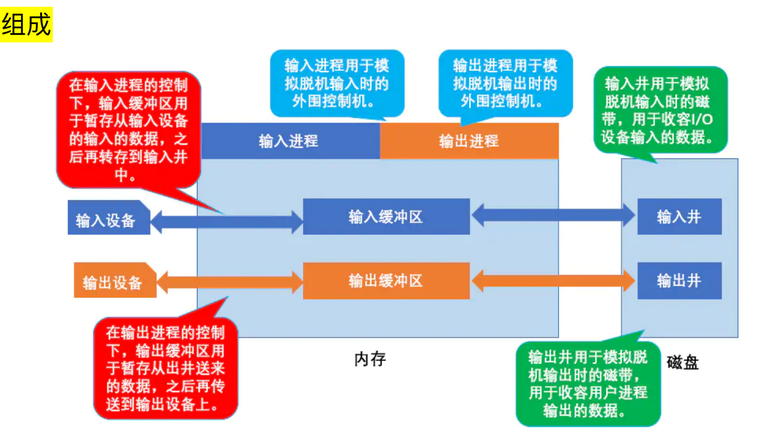
逻辑设备名到物理设备名的映射
- 为了提高设备分配的灵活性和利用率---->分别实现I/O重定向---->所有引入了设备独立性
- 设备独立性：指应用程序独立于具体使用的物理设备/用户在编程序时使用的设备与实际设备无关
- 引入设备独立性的好处
  - 方便用户编程
  - 使程序运行不受具体机器环境的限制
  - 便于程序移植
- 使用逻辑设备表LUT，将逻辑设备名映射为物理设备名
- LUT表项包括：逻辑设备名，物理设备名，设备驱动程序入口地址
- 两种方式设置逻辑设备表
  - 整个系统只设置一张LUT---->适用于单用户系统
  - 每个用户设置一张LUT---->用户登陆时，系统为用户建立一个进程，同时建立一张LUT

SPOOLing技术（假脱机技术）

- 脱机技术
  - 引入目的：缓解设备与CPUd的速度矛盾，实现预输入，缓输出
  - 组成：外围机+更高速的设备--磁带
- 假脱机技术
  - 又叫SPOOLing技术，用软件的方式模拟假脱机技术
  - os中一项将独占设备改造成共享设备的软件技术

脱机技术运行过程
- 输入时：在外围控制机的控制下，慢速的输入设备先将数据输入到更快速的磁带上，之后主机就可以快速的从磁带上读入数据，从而缓解了速度矛盾。
- 输出时：先将数据输出到快速的磁带上，之后通过外围控制机控制磁带将数据依次的输出到慢速的设备上。

SPOOLing系统的组成



输入进程和输出进程【预输入程序管理】
- 用于模拟脱机技术中的外围控制机
- 需要和用户进程并发执行才可以模拟脱机技术，所以要实现SPOOLing技术需要多道程序技术的支持。

输入井和输出井【井输入程序管理】
- 用于模拟脱机技术中的磁带

输入缓冲区和输出缓冲区【缓输出程序管理】
- 输入缓冲区用于暂存从输入设备输入的数据，之后再转存到输入井中
- 输出缓冲区用于暂存从输出井传送的数据，之后再传送到输出设备上。

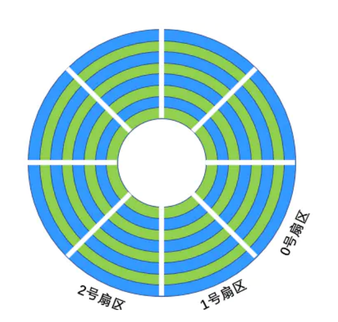
SPOOLing系统的特点
- 提高了IO速度，将对低速IO设备执行的IO操作演变为对磁盘缓冲区中数据的存取
- 缓和了CPIU和低俗IO设备之间的速度不匹配的矛盾
- 将独占设备改造为共享设备，且没有为任何进程分配设备
- 实现了虚拟设备功能，对每个进程而言，都认为自己独占了一个设备
- 以空间换时间
- 需要磁盘空间（输入输出井）和内存空间（输入输出缓冲区）

实例：共享打印机的实现
- 打印机是独占设备，只允许各个进程串行使用设备，一段时间内只能满足一个进程的请求。

实现过程
- 当多个用户提出输出打印的请求时，系统会答应它们的请求，但是并不会真正把打印机分配给它们，而是有假脱机管理进程为每个进程做两件事
- (1) 在磁盘输出井中为进程申请一个空闲磁盘块，之后假脱机管理进程会将进程要打印的数据送入刚申请的空闲磁盘块中。
- (2) 为用户进程申请一张空白的打印请求表，并将用户的打印请求填入表中（用来说明用户的打印数据存放位置等信息），再将该表挂到假脱机文件队列上
- 当打印机空闲时，输出进程会从文件队列的队头取出一张打印请求表，并根据表中的要求打印数据从输出井传送到输出缓冲区，再输出到打印机打印。
- 虽然系统中只有一台打印机，但每个进程提出打印请求时，系统都会为在输出井中为其分配一个存储区（相当于一个逻辑设备），使每个用户进程都觉得自己在独占一台打印机，从而实现对打印机的共享。

磁盘的基本概念

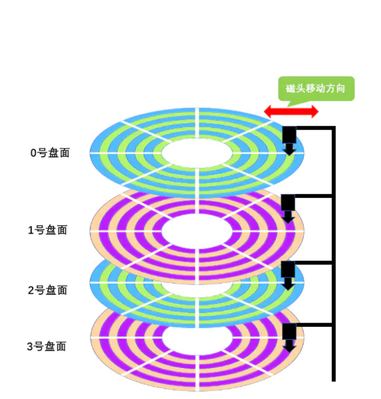
磁盘、磁道、扇区



- 磁盘的表面是由一些磁性物质组成，可以用这些磁性物理记录二进制数据
- 磁盘表面被划分长的一个个磁道。
- 一个磁道又被划分为一个个扇区，每个扇区就是一个“磁盘块”。每个扇区的数据量相同（如1KB）

如何在磁盘中读/写数据?
- 在磁盘中读写数据，需要借助磁头
- step1：将磁头移动到想要读/写的扇区所在的磁道
- step2：磁盘会转动，让目标扇区从磁头下面划过，才能完成对扇区的读/写操作

盘面、柱面



- 磁盘是由多个盘片摞起来的，每个盘片有两个盘面。每个盘面都对应一个磁头。所有的磁头都连在同一个磁臂上。
- 磁臂可以沿着盘面作径向运动，从而带动磁头到达不同的磁道来对不同扇区的读写操作。
- 所有盘面中的相对位置相同的磁道组成了柱面。例如上图中所有盘面的最外侧磁道可以组成一个柱面。

磁盘的物理地址
- 使用（柱面号，盘面号，扇区号）来定位任意一个磁盘块
- 文件数据存放在外存中的几号块，这里的块号就可以转换为（柱面号，盘面号，扇区号）的地址形式。
- 根据物理地址读取一个“块”
  ① 根据柱面号移动磁臂，让磁头指向指定柱面。
  ② 激活指定盘面对应的磁头。
  ③ 磁盘旋转的过程，指定的扇区会从磁头下面划过。这样就完成了对指定扇区的读写。

磁盘的分类
- 根据磁头是否可以活动划分
  - 活动头磁盘
  - 固定头磁盘
- 根据盘面是否可以更换划分
  - 可换磁盘
  - 固定盘磁盘

磁盘的管理

磁盘初始化
- 低级/物理格式化：在磁盘可以存储数据之前，将它分为扇区，以便磁盘控制器能够进行读写操作

分区
- step1：将磁盘分为由一个或多个柱面组成的分区(如C区，D区)
- step2: 对物理分区进行逻辑格式化(创建文件系统)

引导块
- 计算机启动时需要运行一个初始化程序(自举程序)，该程序存放在ROM中
- 启动/系统磁盘：磁盘具有启动分区的磁盘

坏块
- 对坏块的处理实质上就是用某种机制使系统不去使用坏块

固态硬盘SSD

固态硬盘的特性
- 一个SSD由一个或多个闪存芯片和闪存翻译层组成
- SSD由半导体存储器构成，没有移动的部件，因而随机访问时间比机械磁盘要快很多
- SSD是基于闪存的存储技术
- SSD随机读写性能明显高于磁盘，优势主要体现在随机存取上
- SSD随机写比较慢，但不比常规硬盘差

磨损均衡
- 为了弥补SSD的寿命缺陷，引入了磨损均衡
- 动态磨损均衡
- 静态磨损均衡（更先进）

磁盘调度算法

一次磁盘读/写操作需要的时间

寻找时间（寻道时间）Ts【一般最长】
- 在读/写数据前，需要将磁头移动到指定磁道所花费的时间。
- 寻道时间分两步：
  (1) 启动磁头臂消耗的时间s
  (2) 移动磁头消耗的时间：假设磁头匀速移动，每跨越一个磁道消耗时间为m，共跨越n条磁道。
- Ts = s + m * n。

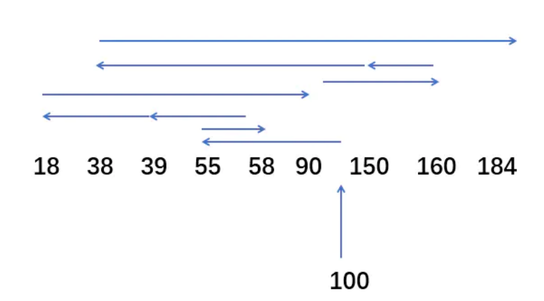
延迟时间TR
- 通过旋转磁盘，使磁头定位到目标扇区所需要的时间。
- TR = (1/2)*(1/r) = 1/2r
- 1/r就是转一圈所需的时间。找到目标扇区平均需要转半圈，因此再乘以1/2

传输时间TR
- 从磁盘读出或向磁盘中写入数据所经历的时间
- 假设磁盘转速为r，此次读/写的字节数为b，每个磁道上的字节数为N
- TR = (b/N) * (1/r) = b/(rN)
- 每个磁道可存N字节数据，因此b字节数据需要b/N个磁道才能存储。而读/写一个磁道所需的时间刚好是转一圈的时间1/r。

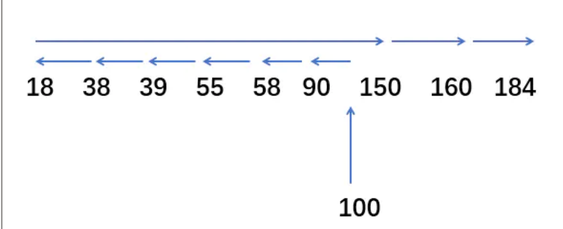
总的平均时间
- Ta = Ts + 1/2r + b/(rN)
- 无法通过操作系统优化延迟时间和传输时间。所以只能优化寻找时间。

常用的磁盘调度算法

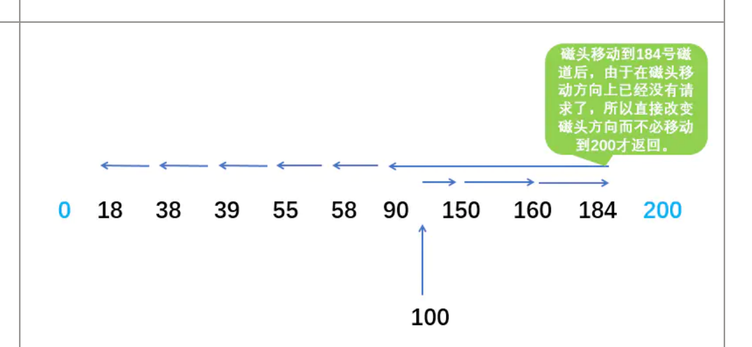
先来先服务算法（FCFS）
- 算法思想：根据进程请求访问磁盘的先后顺序进行调度
- 举例
  - 假设磁头的初始位置是100号磁道，有多个进程先后陆续地请求访问55、58、39、18、90、160、150、38、184号磁道。
  - 按照先来先服务算法规则，按照请求到达的顺序，磁头需要一次移动到55、58、39、18、90、160、150、38、184号磁道。
  - 磁头共移动了 45 + 3 + 19 + 21 + 72 + 70 + 10 + 112 + 146 = 498个磁道。
  - 响应一个请求平均需要移动498 / 9 = 55.3个磁道（平均寻找长度）
- 优点：公平；如果请求访问的磁道比较集中的话，算法性能还算可以。
- 缺点：如果大量进程竞争使用磁盘，请求访问的磁道很分散，FCFS在性能上很差，寻道时间长。



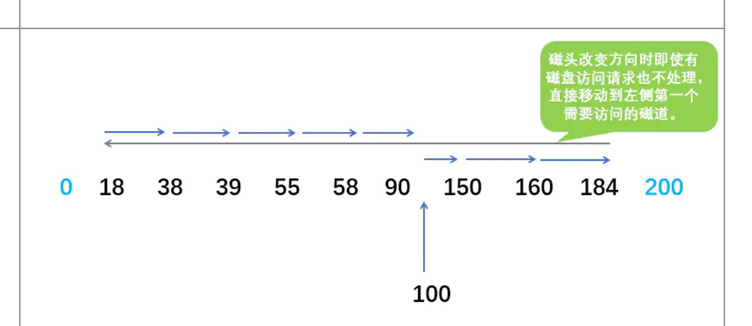
最短寻找时间优先（SSTF）
- 算法思想：优先处理的磁道是与当前磁头最近的磁道。可以保证每次寻道时间最短，不能保证总的寻道时间最短
- 是贪心算法的思想，只是选择眼前最优，但是总体未必最优
- 举例
  - 假设磁头的初始位置是100号磁道，有多个进程先后陆续地请求访问55、58、39、18、90、160、150、38、184号磁道。
  - 磁头总共移动了（100 -18）+（184 -18） = 248个磁道。响应一个请求平均需要移动248 / 9 = 27.5个磁道（平均寻找长度）
  - 如果在处理18号磁道的访问请求时又来了一个38号磁道的访问请求，处理38号磁道的访问请求又来了一个18号磁道访问请求。
  - 如果有源源不断的18号、38号磁道访问请求，那么150、160、184号磁道请求的访问就永远得不到满足，从而产生饥饿现象。
  - 这里产生饥饿的原因是磁头在一小块区域来回移动。
- 缺点：可能产生饥饿现象



扫描算法（SCAN）
- 为了防止饥饿问题，规定
  - 磁头只有移动到请求最外侧磁道或最内侧磁道才可以反向移动
  - 如果在磁头移动的方向上已经没有请求，就可以立即改变磁头移动，不必移动到最内/外侧的磁道。
  - 这就是扫描算法的思想。由于磁头移动的方式很像电梯，因此也叫电梯算法。
- 举例
  - 假设某磁盘的磁道为0~200号，磁头的初始位置是100号磁道，且此时磁头正在往磁道号增大的方向移动
  - 有多个进程先后陆续的访问55、58、39、18、90、160、150、38、184号磁道
  - 磁头共移动了（184 – 100）+（184 -18） = 250个磁道。响应一个请求平均需要移动 250 / 9 = 27.5个磁道（平均寻找长度）
- 优点：性能较好，寻道时间较短，不会产生饥饿现象。
- 缺点：SCAN算法对于各个位置磁道的响应频率不平均。
  - 假设此时磁头正在往右移动，且刚处理过90号磁道，那么下次处理90号磁道的请求就需要等待低头移动很长一段距离
  - 而响应了184号磁道的请求之后，很快又可以再次响应184号磁道请求了



C-SCAN算法
- 为了解决各个位置磁道的响应频率不平均这个问题，规定
  - 只有磁头朝某个特定方向移动时才处理磁道访问请求
  - 而返回时直接快速移动至最靠边缘的并且需要访问的磁道上而不处理任何请求。
  - 通俗理解：SCAN算在改变磁头方向时不处理磁盘访问请求而是直接移动到另一端最靠边的磁盘访问请求的磁道上。
- 举例
  - 假设某磁盘的磁道为0~200号，磁头的初始位置是100号磁道，且此时磁头正在往磁道号增大的方向移动
  - 有多个进程先后陆续的访问55、58、39、18、90、160、150、38、184号磁道
  - 磁头共移动了（184 -100）+（184 – 18）+（90 – 18）=322个磁道
  - 响应一个请求平均需要移动322 / 9 = 35.8个磁道（平均寻找长度）
- 优点：相比于SCAN算法，对于各个位置磁道响应频率很平均。
- 缺点：相比于SCAN算法，平均寻道时间更长。



磁盘调度算法比较

	优点	缺点
FCFS	公平，简单	平均寻道距离大，仅应用在磁盘I/O较少的场合
SSTF	性能比FCFS好	不能保证平均寻道时间最短，可能出现饥饿现象
SCAN	寻道性能较好，可避免饥饿现象	不利于远离磁道一端的访问请求
C-SCAN	消除了对两端磁道请求的不公平	