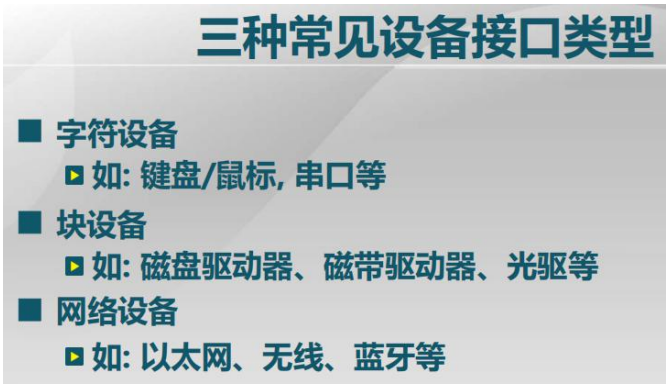
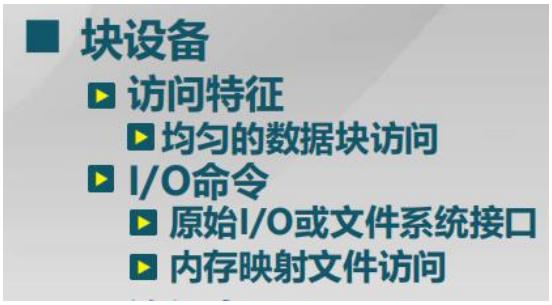
第一节 I/O 特点

I/O 子系统是计算机操作系统当中负责与外设打交道的部分。



字符设备通常速度很慢；块设备通常是存储设备；网络设备是计算机系统与外界打交道的最重要手段。

比如键盘：是通过敲击键盘，一个字节一个字节按顺序进行访问的。通常将字符设备封装成文件，使用文件访问接口和语义。

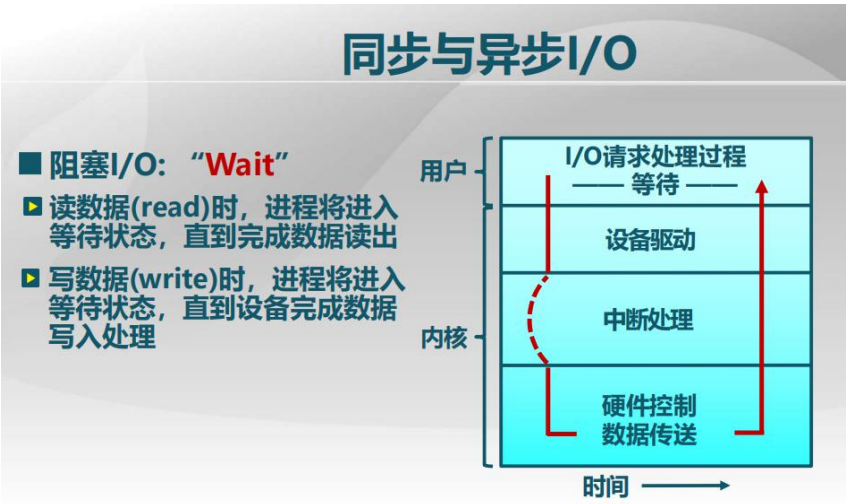


原始 I/O：直接对磁盘上的扇区进行读写控制，可提高性能

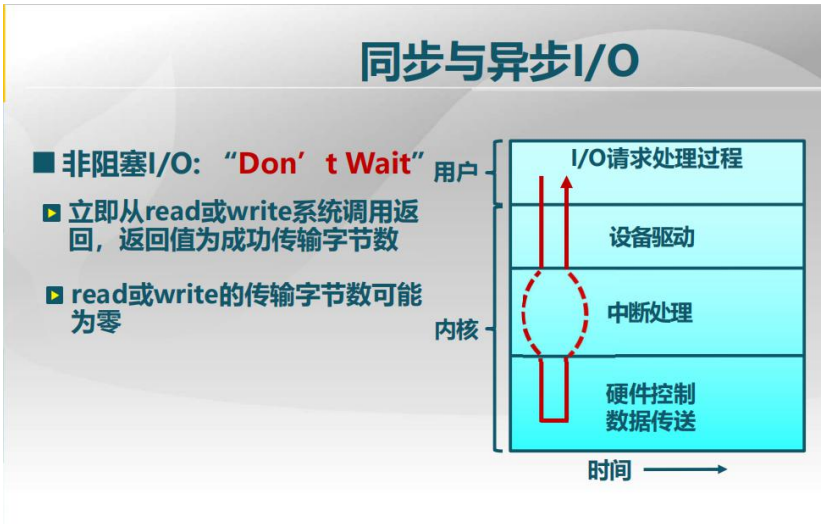
内存映射文件访问：把磁盘映射到内存当中，用内存映射文件对磁盘上的数据进行访问。

网络设备的交互复杂。网络设备的 I/O 命令是专门的网络报文收发接口，不同的网络协议封装在网络接口下。

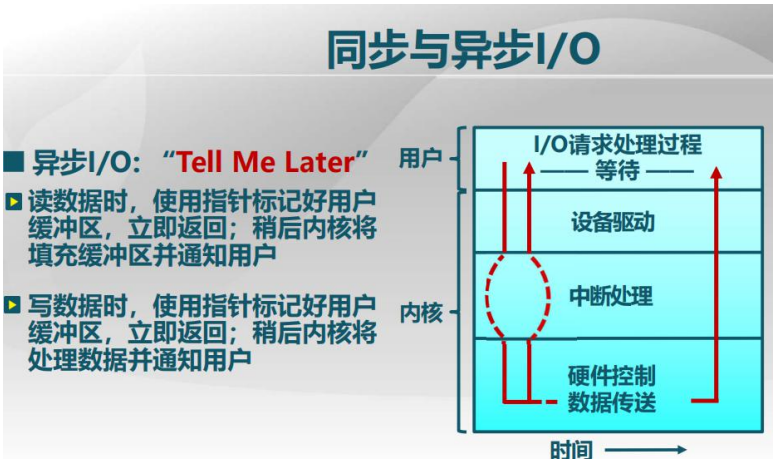
下面看 CPU 与设备之间的交互：



用户发出 I/O 请求，该请求会送到操作系统内核当中的设备驱动，设备驱动会将其转换为硬件的控制，控制相应的硬件进行相应的操作，硬件操作完成之后，会产生中断，由内核中的中断处理例程进行响应，送到设备驱动，然后回到用户态。以上是阻塞 I/O 过程的描述，也就是从发出请求到得到数据，中间的时间，进程要处于等待状态，一直到有数据返回。



进程执行过程中，发出 I/O 命令后不等待，请求和返回过程都不经过中断处理。这种方式可能读写不成功，或者读写的数据量与预期不一致。于是有了第三种方式：

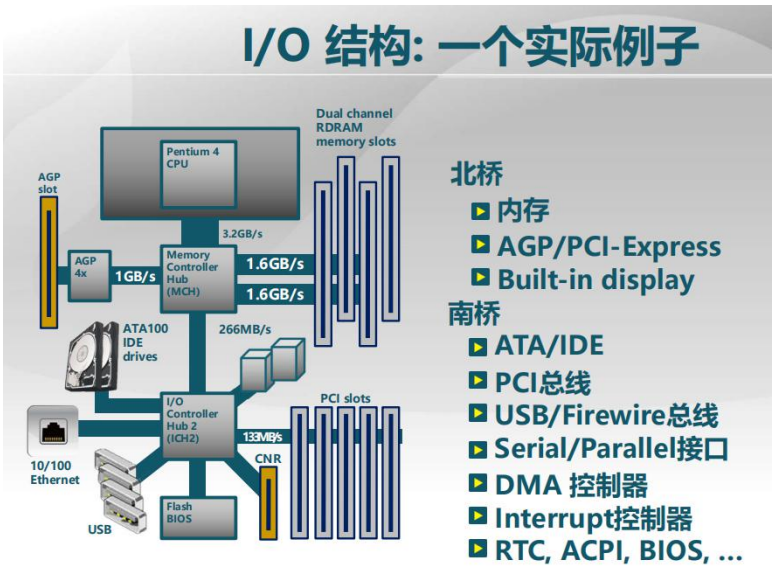


异步 I/O 是把上面两者结合。

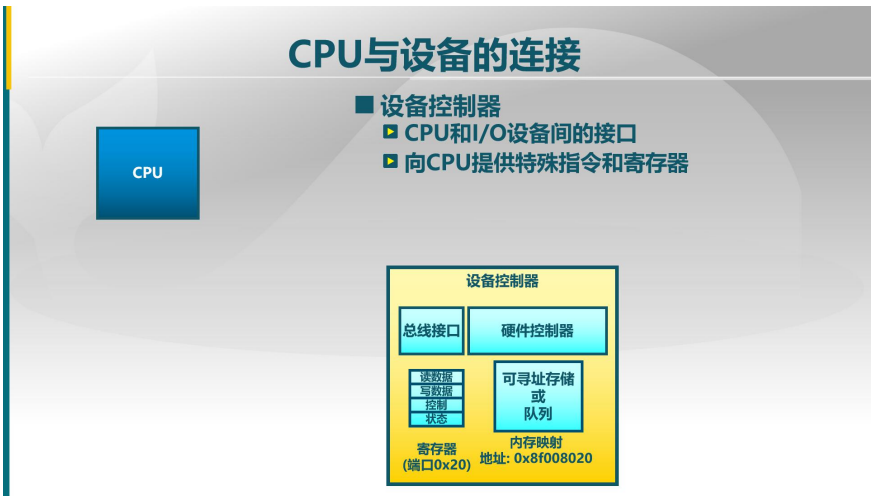
1.字符设备包括 ABCD
A.键盘
B.鼠标
C.并口
D.串口
2.块设备包括 ABCD
A.硬盘
B.软盘
C.光盘
D.U 盘
3.网络设备包括 ABC
A.以太网卡
B.wifi 网卡
C.蓝牙设备
D.网盘设备
网盘在模拟实现上应该算块设备(我的理解：其功能是云端存储而不是通讯)

第二节 I/O 结构

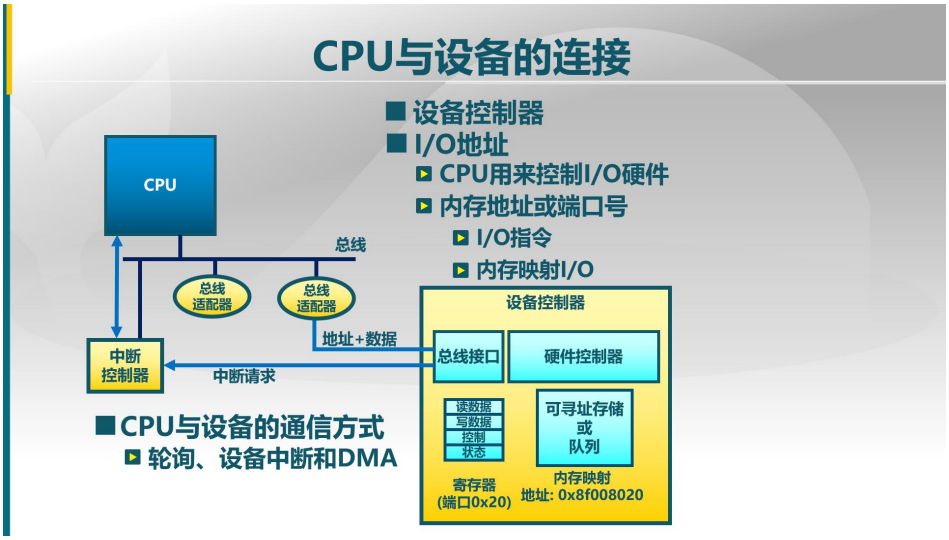
先从硬件结构说起：



在计算机系统中，CPU 为了与外界相连，在主板上分成了两段，一段是北桥：和高速的内存、显卡相连，这时它的速度通常是若干个 G；还有一部分是南桥：它负责与各种各样的设备相连，如 PCI 总线：磁盘、网络都是通过这下面连的，基本原理中说，CPU 通过总线来连接内存和 I/O 设备，这里我们就细化成北桥连高速设备，南桥连 I/O 设备。



在这种结构下我们还需要进一步细化 CPU 到底如何来识别每一个设备，它们的连接关系是什么样的。首先，上图是 CPU 与 I/O 设备：设备上有设备控制器，设备控制器提供 CPU 和 I/O 设备之间的接口，上图中就是总线接口。设备控制器中有相应的一组寄存器(标读数据、写数据、控制、状态那个)，可以进行数据的交互和状态、控制的交互。也可以把(寄存器中内容?I guess)映射到内存当中，给一段内存区域，对这段内存区域的访问就相当于对 I/O 设备的访问,这也就是 I/O 地址(的一种可能形式)。



I/O 地址通过总线连到 CPU 上，总线和实际设备之间有总线适配器，I/O 地址映射过来可能是内存地址，也可能是 I/O 空间的端口，对 I/O 端口有相应的 I/O 指令，如果是内存，可直接访问存储，就对应着对 I/O 设备的访问。以上是从 CPU 到设备(CPU 告知设备该如何做)。

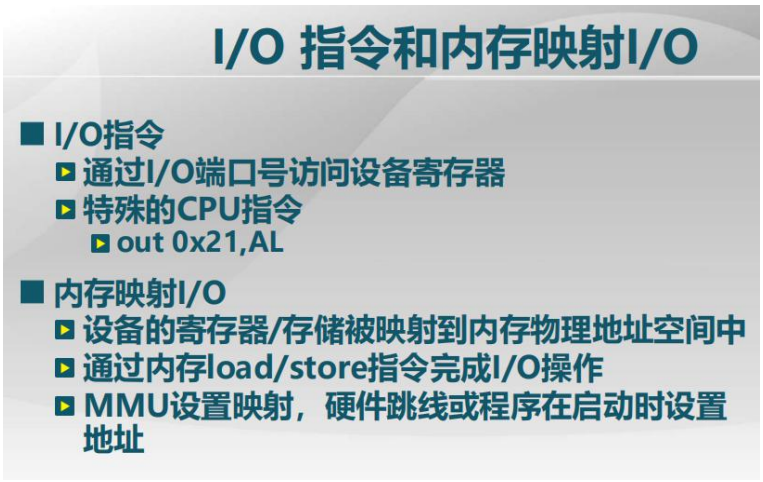
还有从设备到 CPU 的通道，即中断控制器。设备产生中断之后，在中断控制器汇总，然后送给 CPU，CPU 就能对外部设备的事件做出响应。

CPU 与设备的通信方式：

轮询：不用中断控制器，CPU 直接访问 I/O 端口，或访问设备所对应的内存地址空间。

设备中断：外部设备有事件要通知 CPU，要通过中断到 CPU。

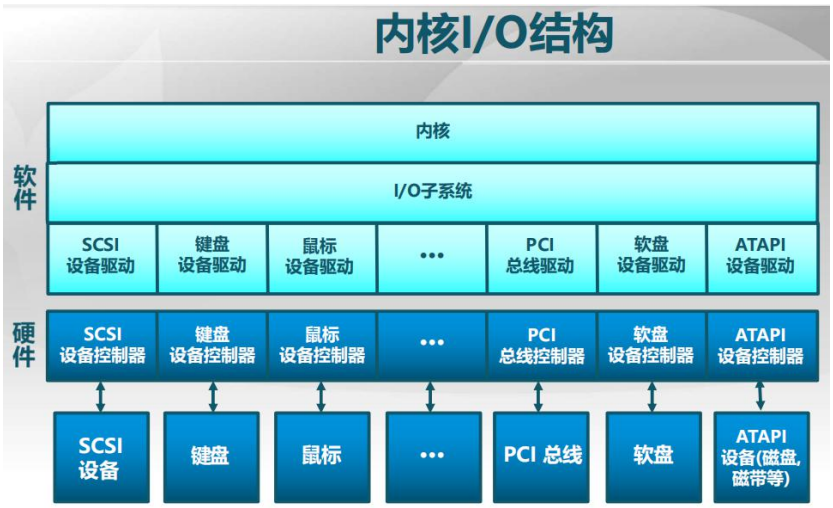
DMA：外部设备要把数据直接放到内存当中，在 DMA 控制器的控制下，把数据从 I/O 设备直接送到内存单元。



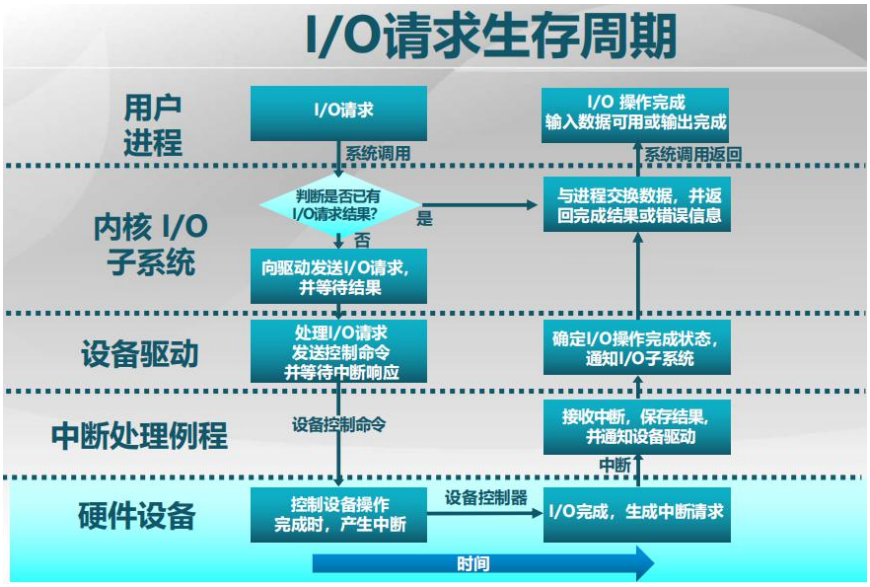
I/O 指令：即通过端口号来区别访问的是哪个设备。其通过特殊的 CPU 指令来完成读写(out、in)

内存映射 I/O 的地址通过 MMU 进行设置，或通过硬件的跳线来完成相应的地址映射。

操作系统中 I/O 子系统的结构：



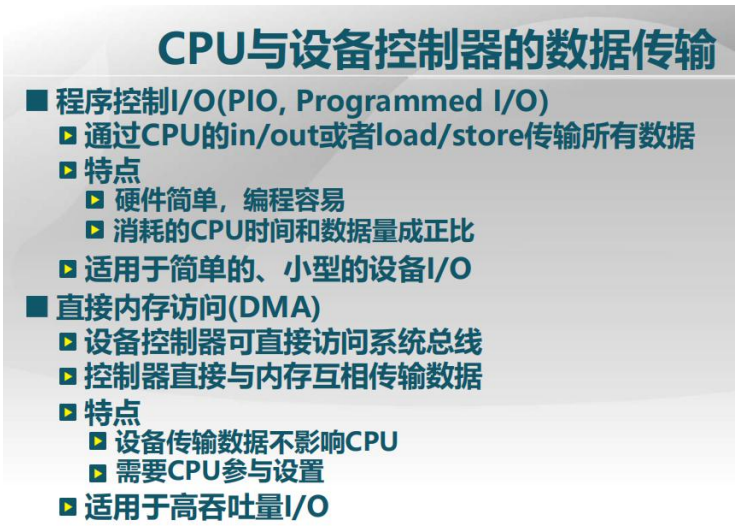
最底层是各种设备：SCSI 存储设备，鼠标，键盘等。每一个设备之上都对应一个设备控制器，不同设备的设备控制器是不同的。在这之上是软件，操作系统里的第一层是设备驱动，每一类设备对应有自己的驱动，再之上是 I/O 子系统，它用来处理各种设备共同的内容：比如 I/O 请求转换成驱动的 I/O 请求，缓存设备给出的结果，如访问某一磁盘上某一扇区的数据，若前面已经做过一次，则 I/O 子系统将这部分缓存，第二次访问时，I/O 子系统直接给结果。再之上是内核的其他内容，它们依赖于 I/O 子系统。



判断是否已有 I/O 请求结果，即看 I/O 子系统中是否有这部分内容的缓存。我想内核 I/O 子系统中等待指调入另一进程 B 执行，当前发出 I/O 请求的进程 A 进入等待状态。　在 I/O 设备操作完成后，会发出中断，中断进程 B，返回系统调用结果给进程 A。

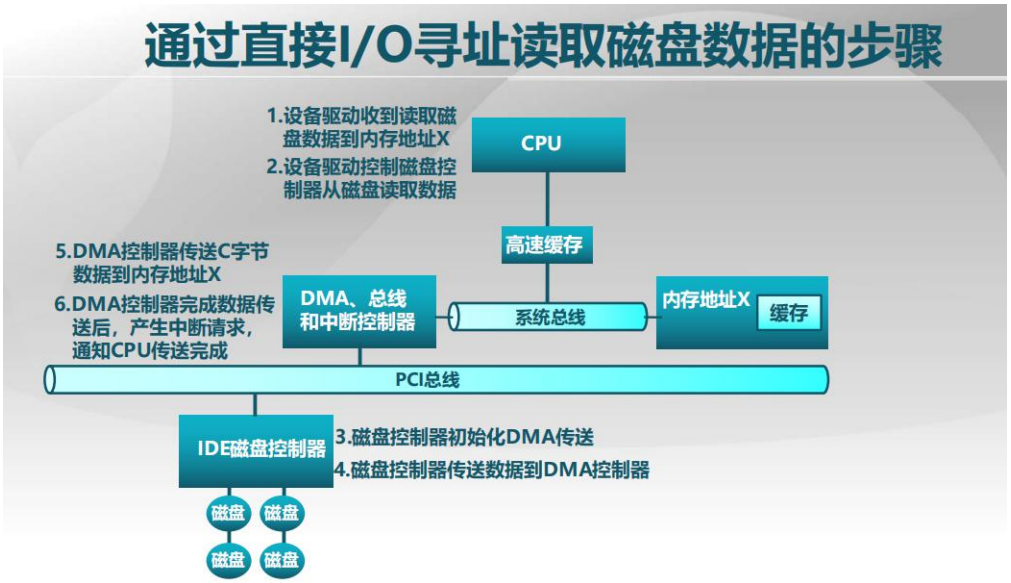
第三节 I/O 数据传输

在 I/O 子系统中，设备和 CPU 之间的数据传输性能是我们这里关注的一个重要问题。CPU 和设备之间的数据传输有两种方式：



load/store 指令是做了内存映射的情况下。PIO 方式在整个传输过程中需要 CPU 参与；DMA 方式在数据传输过程中不需要 CPU 参与，但开始和结束时需要 CPU 参与设置。

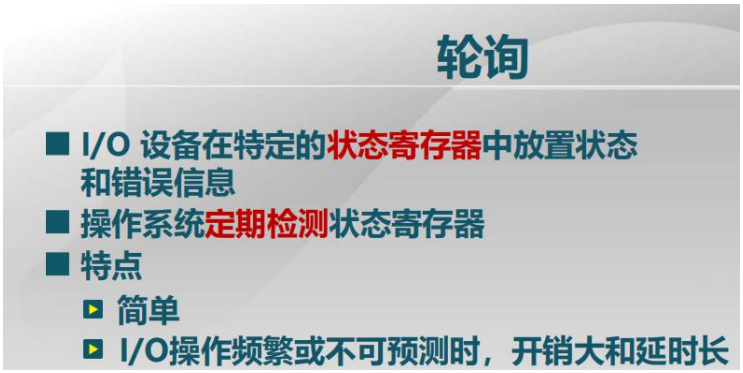
下面用磁盘数据读取为例，对 DMA 方法读取磁盘数据的过程进行细化：



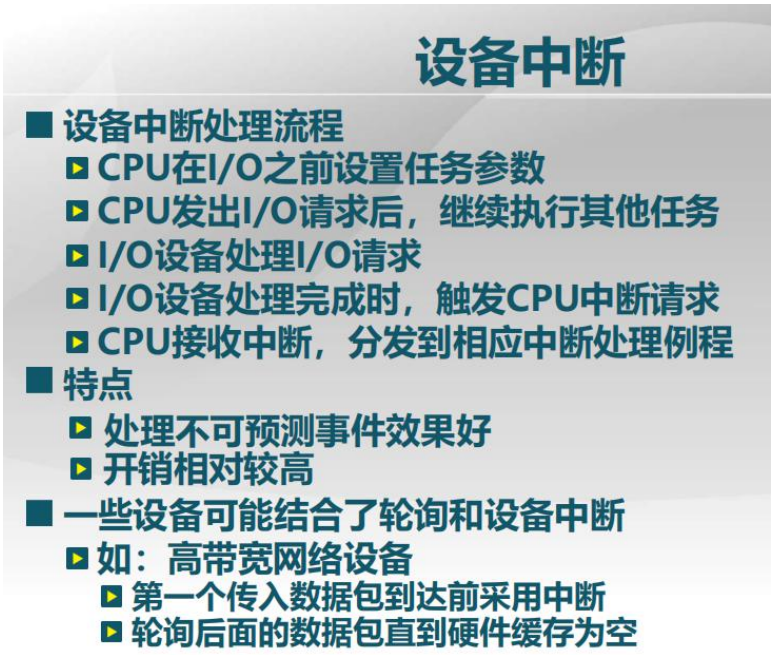
CPU 通过系统总线连到内存和 DMA 控制器，DMA 控制器通过 PCI 总线连到磁盘设备。

一次磁盘读取过程如下：1.CPU 执行用户代码的过程中会产生磁盘读取请求：读取磁盘 Y 处数据到内存地址 X，这个请求转到设备驱动，在内核里执行。 2.设备驱动控制磁盘控制器从磁盘读取数据。 3.见图 4.磁盘控制器通过 PCI 总线传送数据到 DMA 控制器 5.见图 6.CPU 响应这个中断请求，最后回到应用程序。

下面看设备如何通知 CPU，通知操作系统(即 CPU 如何直到有一个 I/O 请求)：



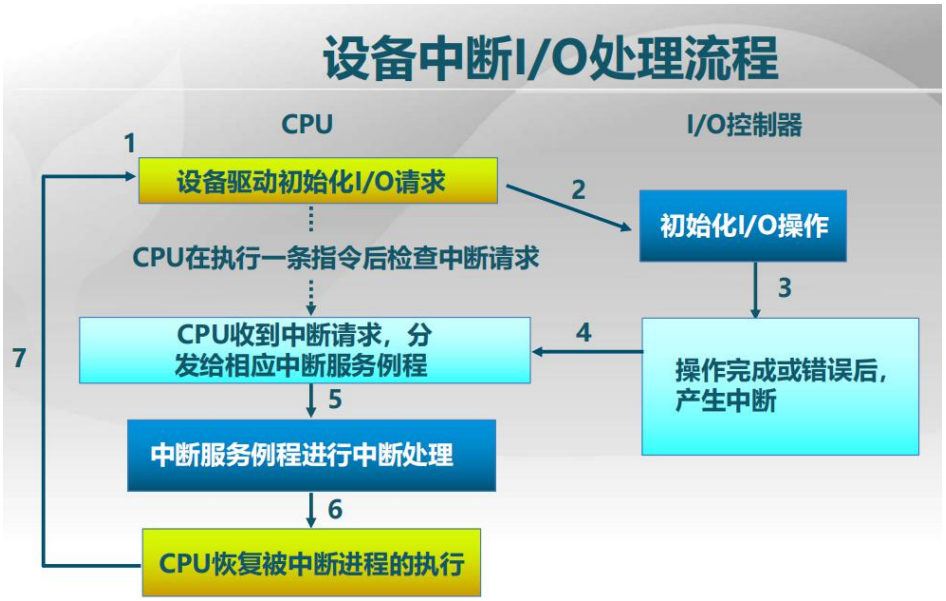
I/O 设备上定义了一组状态和控制寄存器，操作系统定期检查这些状态寄存器，从而知道设备的状态：如数据是否发送完，缓冲区是否还有空余等。



若中断较多，则 CPU 频繁中断，开销较高。

实际做法是，一些设备可能结合了轮询和设备中断。高带宽网络设备：第一个传入数据包到达前采用中断，由于输入输出数据比较多。量比较大，第一次中断处理完后，为避免频繁的中断，会进行轮询，即轮询后面的数据包直到硬件缓存为空。很很长时间轮询无果，才转回中断方式。

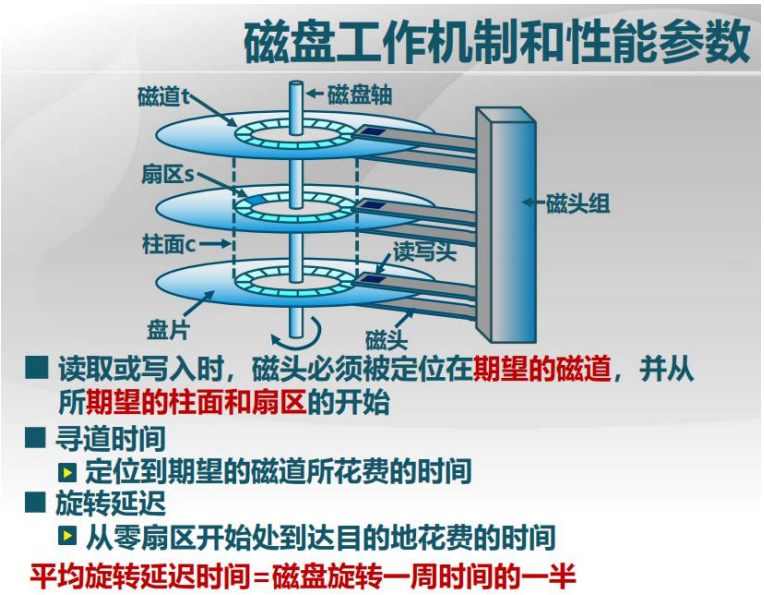
下图是设备中断方式的处理流程：



CPU 执行指令的过程当中，有指令产生 I/O 请求，这时 1，然后 I/O 控制器 2，3.由设备进行相应的操作，然后 4,CPU 在执行一条指令后检查中断请求是为了及时响应中断。

第四节 磁盘调度

在具体讨论磁盘调度前，先看磁盘的工作机理和它的性能参数。

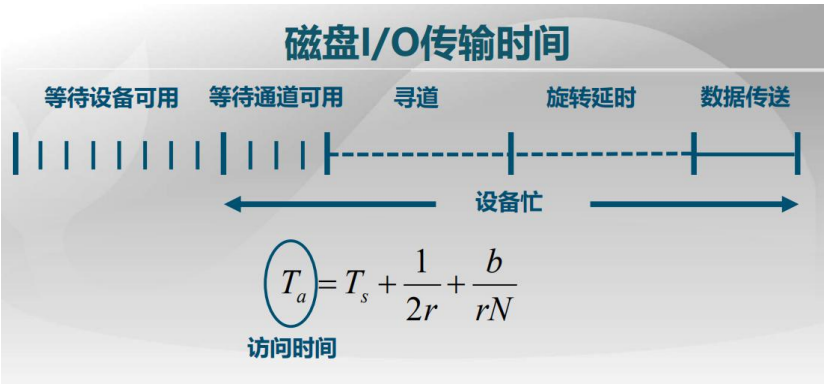


上图是一个磁盘的基本结构，它由若干个盘片和一个磁头组组成，磁头组上，每一个磁头上有一个读写头，分别对应着盘片的正反面，在这里可以读写相应的数据。盘片是围绕着磁盘轴进行旋转的，在旋转的过程中，我们要读写的数据是分布在各个盘片上的，这里，磁道、柱面和扇区可确定数据所在的位置。

寻道时间：磁头移动到指定磁道花费的时间。

旋转延迟：要想找到指定的扇区，需要等待盘片旋转到指定扇区。

下面具体分析磁盘 I/O 的时间都花在什么地方：



①要读写某个设备，要等待这个设备是可以访问的

②要想和这个设备交流，DMA 通道或 I/O 通道必须可用，即等待通道可用。

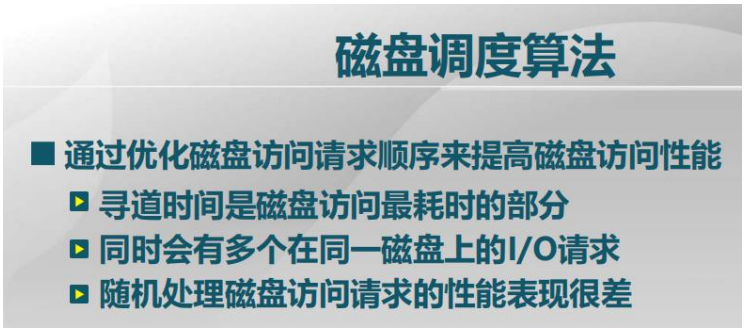
这两个时间和实际的设备操作关系不大

③寻道时间：磁头移到指定的磁道：Ts

④旋转延时：1/2r，1/r=旋转一周的时间，r 是转的速度(例如单位：圈/每秒)

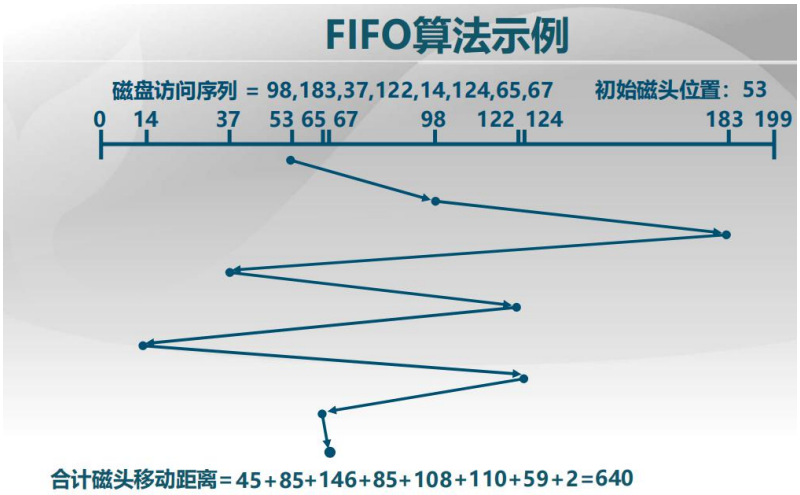
⑤数据读出时间：即传输时间，数据传送时间，b/(rN),b=传输的比特数，N=磁道上的比特数，r=磁盘转数即转的速度。

从以上分析，主要应优化 Ts，即寻道时间，这也是磁盘调度所要优化的方面。



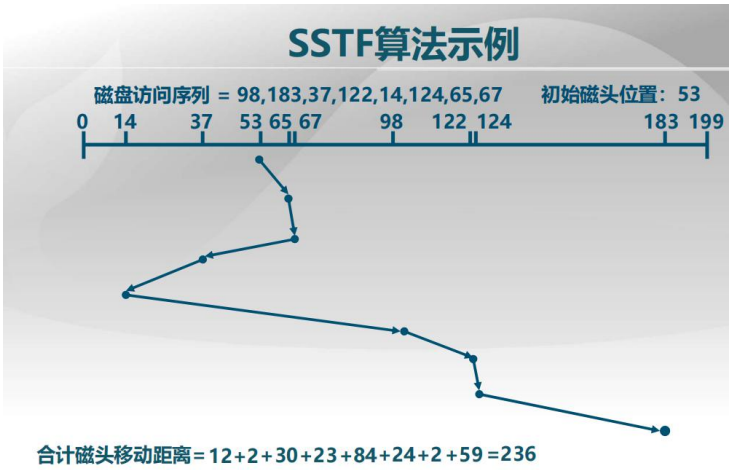
下面看具体的磁盘调度算法：

1.FIFO

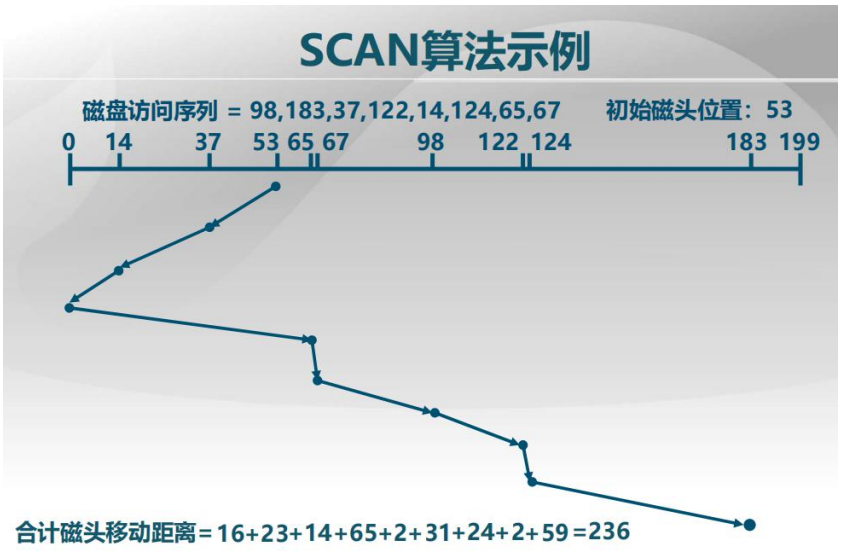


来回左移右移走了很多冤枉路，性能是很差的。

2.SSTF



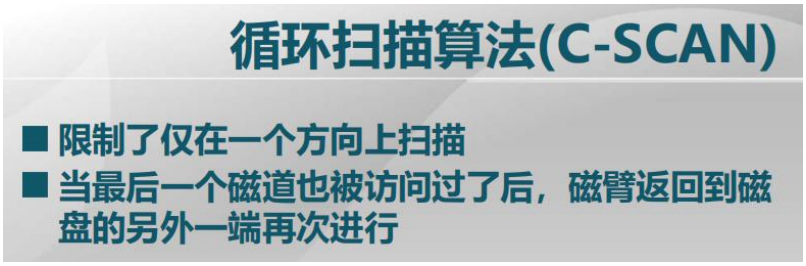
3.SCAN



假定起始时往低的方向走，如上图。

该方法特点：沿着一个方向走，顺序扫过去，判断会比较简单，不用像 SSTF 一样去找哪个是最近的。

4.C-SCAN

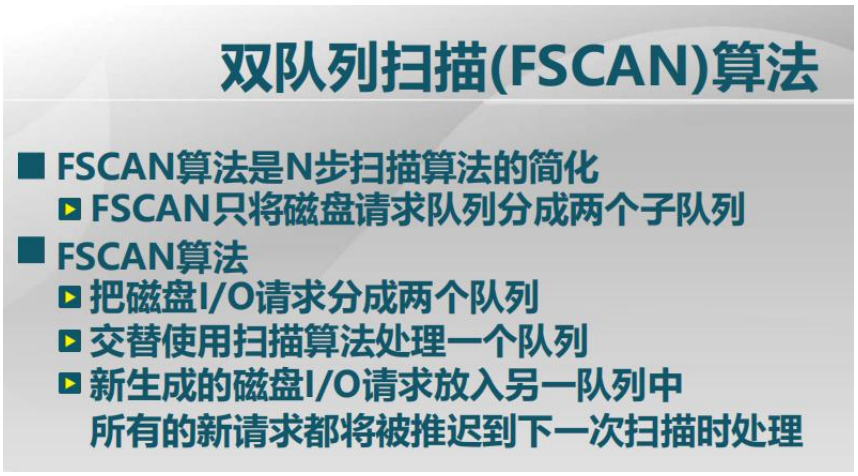


该算法相对 SCAN 提高了公平性，但即使后面没有 I/O 请求，也要扫描到最后一个磁道，这是不合适的。因此：

5.C-Look

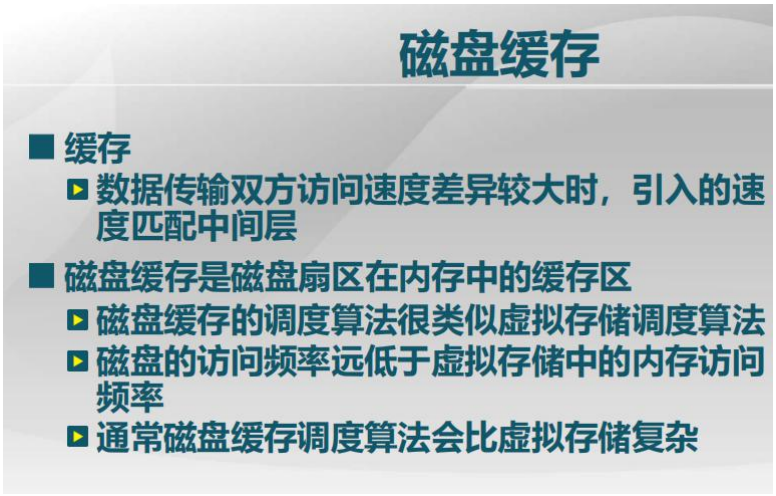
6.N 步扫描

7.FSCAN

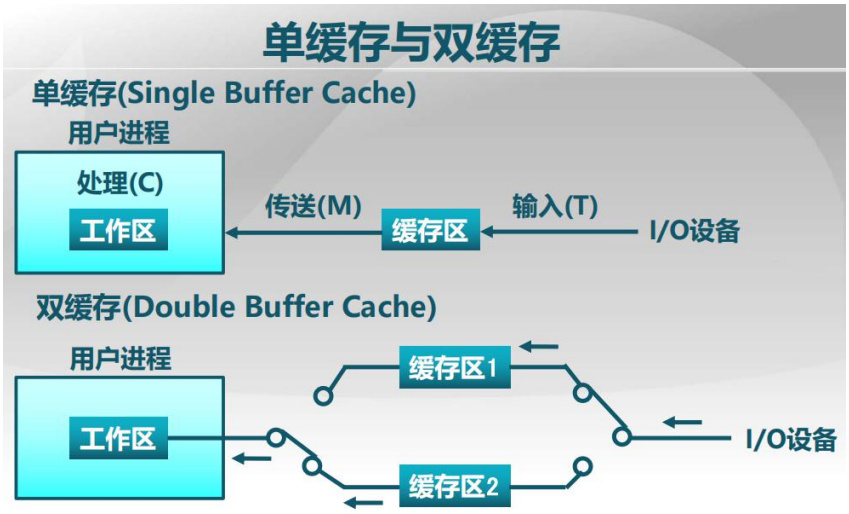


第五节 磁盘缓存

磁盘缓存是放在内存里的磁盘数据的缓存。这些缓存是为了避免对同一块磁盘扇区里的内容反复引用时有多次的磁盘访问。



磁盘缓存与虚拟存储对比：虚拟存储是利用磁盘空间来存内存中存不下的数据，而磁盘缓存是倒过来。因此两者有很多相似之处。磁盘的访问频率远低于虚拟存储中的内存访问频率,因此通常磁盘缓存调度算法会比虚拟存储复杂。



单缓存和双缓存指的是缓存区有一个还是两个。

单缓存：设置一个缓存区，I/O 设备向缓存区里写数据时，由于只有单缓存，这时 CPU(工作区)这头不能操作，等 CPU 能从缓存区里读数据时，I/O 设备是不能往缓存区写的。这与生产者消费者问题很类似。这种方法因此速度很受限制。若工作区与 I/O 设备交互频繁，用双缓存：

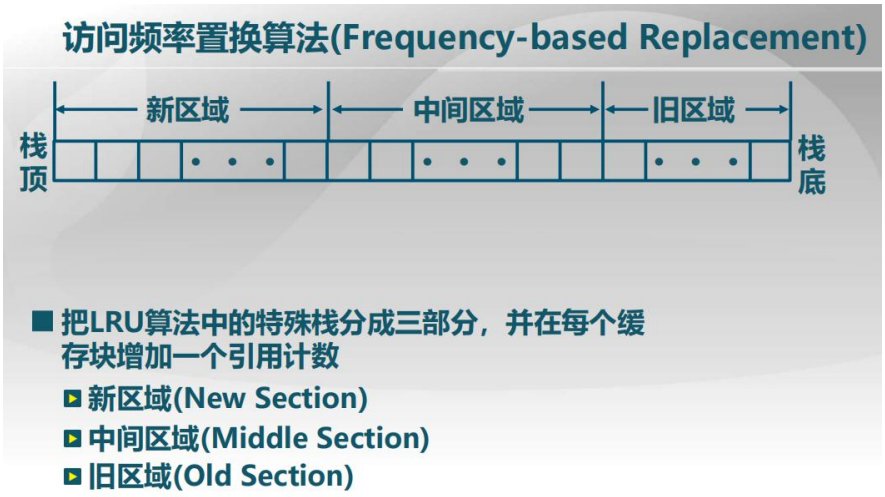
设置两个缓存区，在一个缓存区由一头在进行操作时，如 I/O 设备往缓存区 1 里写数据时，这时 CPU 就可由缓存区 2 读数据，这两者可以同时进行，因为有两个不同的缓存区。

下面看磁盘缓存的访问频率置换算法：

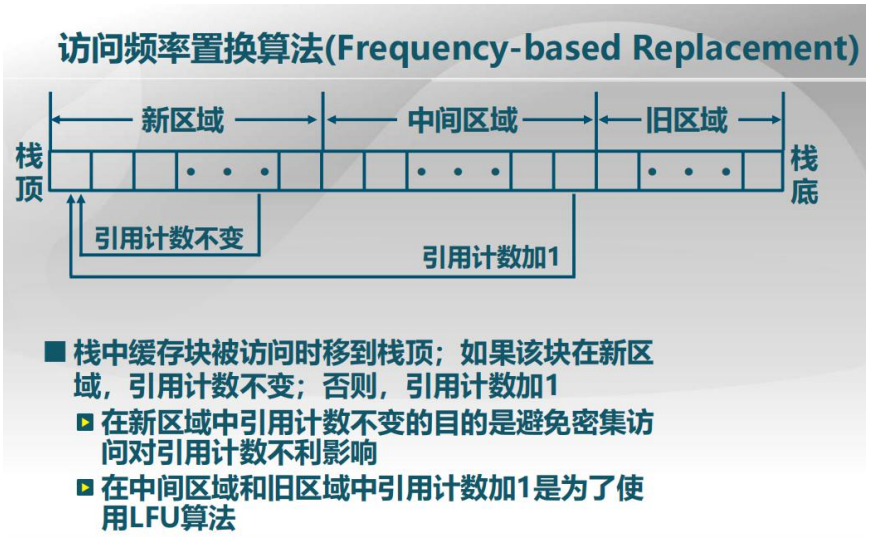
复习：LRU（The Least Recently Used，最近最久未使用算法）：缺页时，置换最长时间未引用的页面。

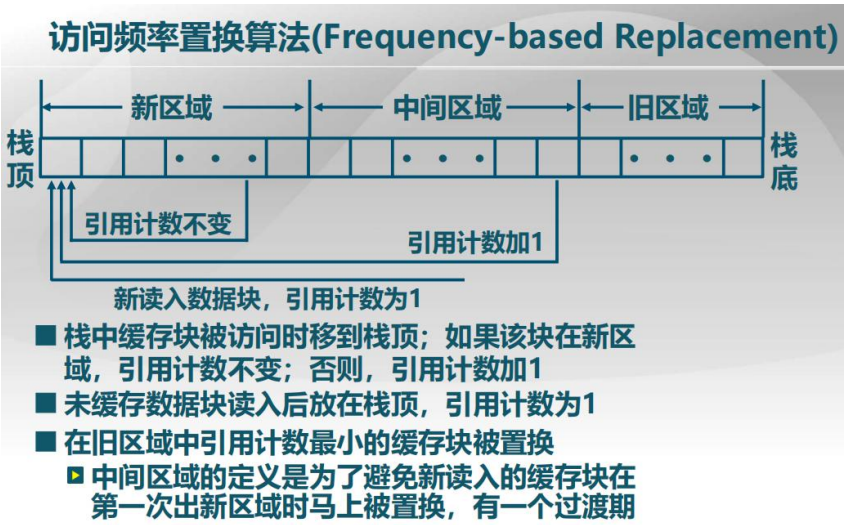
LFU（Least Frequently Used ，最近最少使用算法）：缺页时，置换访问次数最少的页面。在长周期中才使用 LFU 算法，以消除密集访问的影响。

具体地：



与以前的 LRU 算法不同，该 LRU 算法将栈分为三个部分，目的是对这三段做不同的处理。这种不同的处理体现在：





以上，就把虚拟存储里的算法，改造成一个更复杂，可以用来做磁盘缓存置换算法的访问频率置换算法。

I/O 特点：多种不同的设备，字符设备、块设备、网络设备，这些设备的访问方式各有各的特点。

I/O 数据传输：有轮询、中断、DMA 方式，这几种方式是在 I/O 子系统中常用的。

磁盘调度：对 I/O 请求，哪个先去访问。

磁盘缓存：把磁盘上哪些数据放到内存中做缓存的算法。

在设备管理子系统中，引入缓冲区的目的主要有 ABCD

A.缓和 CPU 与 I/O 设备间速度不匹配的矛盾

B.减少对 CPU 的中断频率，放宽对 CPU 中断响应时间的限制

C.解决基本数据单元大小（即数据粒度）不匹配的问题

D.提高 CPU 和 I/O 设备之间的并行性

我的理解：B.引入缓冲区后，若缓冲区中有数据，就不必再进行磁盘 I/O 操作，这样就不必频繁进行中断，这样由于中断的减少，中断响应时间就可以适当延长，不会出现多个中断挤在一起，对每个中断都要快速解决的情况。

C.数据单元大小不匹配指的就是内存和硬盘大小不匹配，引入缓冲区就有个中间缓冲。

D.若无缓冲区，因 CPU 速度比 I/O 设备快得多，这样 CPU 就必须等待 I/O 设备，引入缓冲区，CPU 可从缓冲区中不断读数据，这样就提高了并行性。