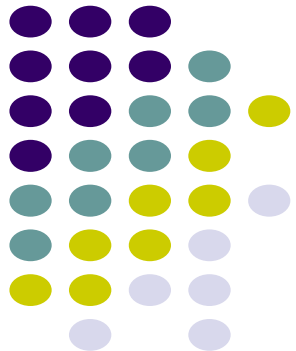
操作系统

第6章 设备管理



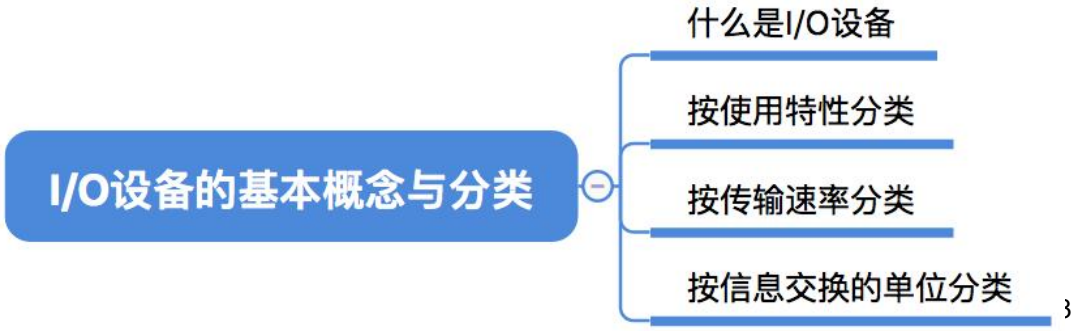
内容

- 5.1 I/O系统
- 5.2 I/O控制方式
- 5.3 缓冲管理
- 5.4 设备分配
- 5.5 设备处理
- 5.6 磁盘存储器管理

知识回顾


用户
应用程序（软件）
操作系统
裸机（纯硬件）


作为系统资源的管理者
提供的功能
处理机管理
存储器管理
文件管理
设备管理
目标
安全、高效


I/O设备的基本概念与分类
什么是I/O设备
按使用特性分类
按传输速率分类
按信息交换的单位分类

什么是I/O设备

“I/O” 就是 “输入/输出”（Input/Output）

I/O 设备就是可以将数据输入到计算机，或者可以接收计算机输出数据的外部设备，属于计算机中的硬件部件。

鼠标、键盘——典型的输入型设备

显示器——输出型设备

移动硬盘——即可输入、又可输出的设备

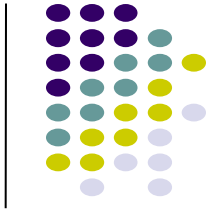
什么是I/O设备

UNIX系统将外部设备抽象为一种特殊的文件，用户可以使用与文件操作相同的方式对外部设备进行操作

Write操作：向外部设备写出数据

Read操作：从外部设备读入数据

I/O系统的基本功能



- 隐藏物理设备的细节
  - I/O设备类型多，差异大。利用设备控制器（硬件）进行控制。隐藏物理设备的使用细节
- 与设备的无关性
  - 用户仅提供逻辑设备名来使用设备；即插即用功能
- 提高处理机和I/O设备的利用率
  - 设备之间、设备与处理机之间均可并行操作。要求CPU快速响应I/O请求，减少对设备运行的干预时间

I/O系统的基本功能

- 对I/O设备进行控制
  - 即驱动程序的功能。具体控制方式与设备的传输速率和传输数据单位有关。
- 确保对设备的正确共享
  - 以共享属性来分类，分为独占设备、共享设备
- 错误处理
  - 设备包括机械和电气部分，易出错和故障。临时性错误（重试）、持久性错误（向上层报告）

I/O设备分类

- I/O设备的类型繁多，从OS观点看，其重要的性能指标有：数据传输速率、数据的传输单位、设备共享属性等。因而从以下不同角度进行分类
  - 按设备的使用特性分类
  - 按传输速率分类
  - 按信息交换的单位分类
  - 按设备的共享属性分类

I/O设备的分类——按使用特性

I/O设备按使用特性分类

- 人机交互类外部设备 —— 数据传输速度慢
- 存储设备 —— 数据传输速度快
- 网络通信设备 —— 数据传输速度介于上述二者之间



人机交互类外设：鼠标、键盘、打印机等——用于人机交互

存储设备：移动硬盘、光盘等——用于数据存储

网络通信设备：调制解调器等——用于网络通信

I/O设备的分类——按传输速率分类

低速设备：鼠标、键盘等——传输速率为每秒几个到几百字节

中速设备：如激光打印机等——传输速率为每秒数千至上万个字节

高速设备：如磁盘等——传输速率为每秒数千字节至千兆字节的设备

I/O设备的分类——按信息交换的单位分类

I/O设备按信息交换的单位分类

- 块设备：传输速率较高，可寻址，即对它可随机地读/写任一块
- 字符设备：传输速率较慢，不可寻址，在输入/输出时常采用中断驱动方式

块设备：如磁盘等——数据传输的基本单位是“块”

字符设备：鼠标、键盘等——数据传输的基本单位是字符。

I/O设备的分类——按共享属性分类

- I/O控制器的功能
  - 接受和识别CPU发出的命令
  - 向CPU报告设备的状态
  - 数据交换
  - 地址识别
  - 数据缓冲区
  - 差错控制

独占设备：指在一段时间内只允许一个用户（进程）访问的设备

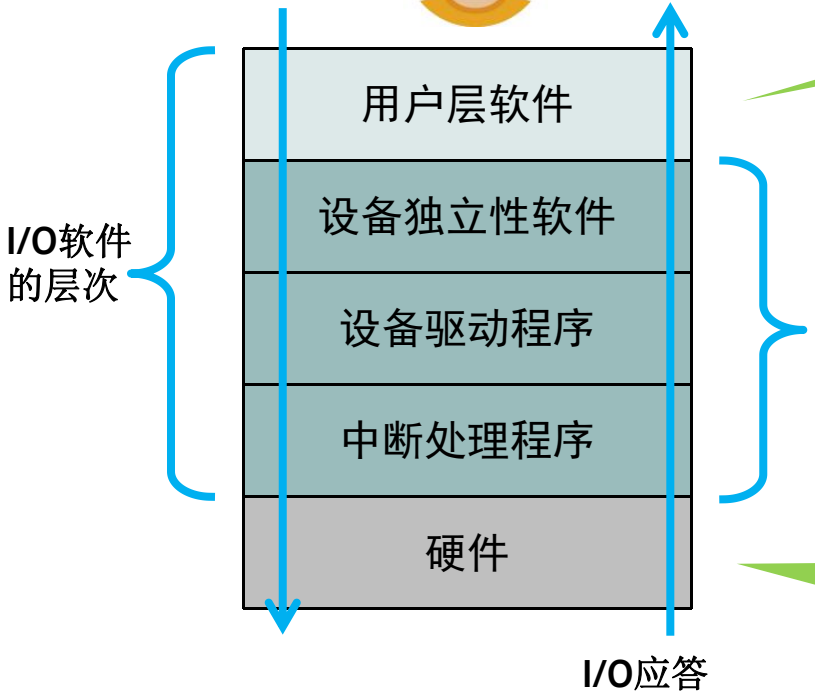
共享设备：一段时间内允许多个进程同时访问的设备。而某一时刻仍然是一个进程访问

虚拟设备：通过虚拟技术将一台独占设备变换为若干台逻辑设备，供若干个用户（进程）同时使用

I/O系统的层次和模型

- I/O软件向下与硬件相关，向上与文件系统、虚拟存储、用户直接交换，都需要 I/O系统来实现 I/O操作。目前均为层次式I/O系统设计，单向调用
- I/O软件的层次——四层
  - 1）用户层软件：实现与用户交互的接口
  - 2）设备独立性软件：用户程序与驱动程序的统一接口
  - 3）设备驱动程序：实现系统对设备发出指令
  - 4）中断处理程序：保存CPU环境，转入中断处理，恢复现场

I/O软件层次结构



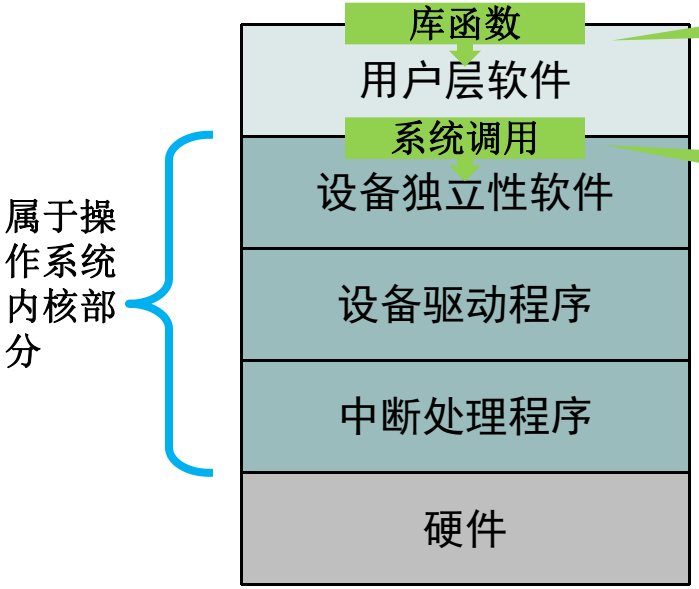
越上面的层次越接近用户

每一层会利用其下层提供的服务，实现某些功能，并屏蔽实现的具体细节，向高层提供服务（“封装思想”）

属于操作系统的内核部分；
即“I/O系统”；
或称“I/O核心子系统”

越下面的层次越接近硬件

用户层软件



用户层软件实现了与用户交互的接口，用户可直接使用该层提供的、与I/O操作相关的库函数对设备进行操作

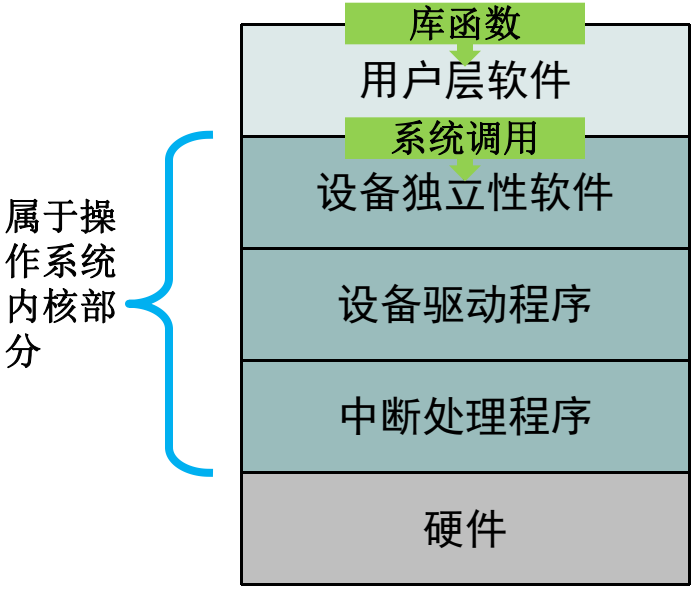
Eg：printf(“hello, world!");

用户层软件将用户请求翻译成格式化的I/O请求，并通过“系统调用”请求操作系统内核的服务

Eg：printf(“hello, world!”); 会被翻译成等价的 write 系统调用，当然，用户层软件也会在系统调用时填入相应参数。

Windows 操作系统向外提供的一系列系统调用，但是由于系统调用的格式严格，使用麻烦，因此在用户层上封装了一系列更方便的库函数接口供用户使用（Windows API）

设备独立性软件



设备独立性软件，又称设备无关性软件。与设备的硬件特性无关的功能几乎都在这一层实现

主要实现的功能：

①向上层提供统一的调用接口（如read/write 系统调用）

②设备的保护：原理类似与文件保护。设备被看做是一种特殊的文件，不同用户对各个文件的访问权限是不一样的，同理，对设备的访问权限也不一样。

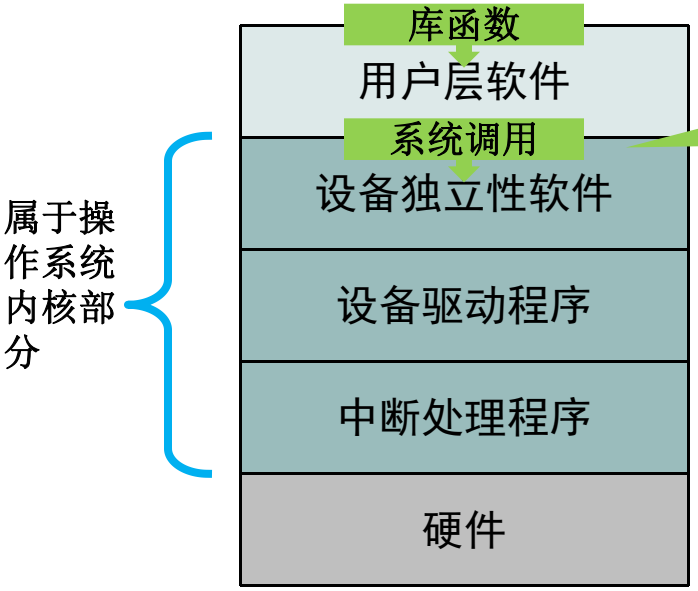
③差错处理：设备独立性软件需要对一些设备的错误进行处理

④设备的分配与回收：系统对独占设备进行统一分配

⑤数据缓冲区管理：可以通过缓冲技术屏蔽设备之间数据交换单位大小和传输速度的差异

⑥建立逻辑设备名到物理设备名的映射关系；根据设备类型选择调用相应的驱动程序

设备独立性软件

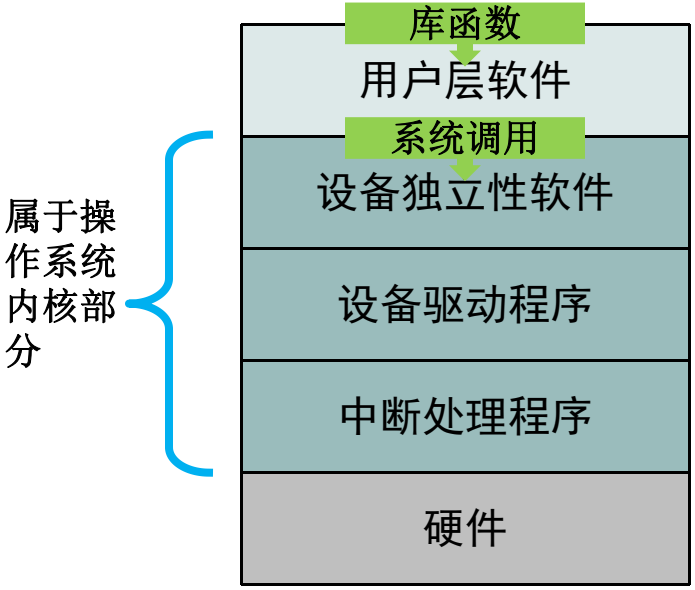


设备独立性软件，又称设备无关性软件。与设备的硬件特性无关的功能几乎都在这一层实现

主要实现的功能：
⑥建立逻辑设备名到物理设备名的映射关系；根据设备类型选择调用相应的驱动程序

用户或用户层软件发出I/O操作相关系统调用的系统调用时，需要指明此次要操作的I/O设备的逻辑设备名（eg：去学校打印店打印时，需要选择打印机1/打印机2/打印机3，其实这些都是逻辑设备名）设备独立性软件需要通过“逻辑设备表（LUT，Logical UnitTable）”来确定逻辑设备对应的物理设备，并找到该设备对应的设备驱动程序

设备独立性软件



主要实现的功能：
⑥建立逻辑设备名到物理设备名的映射关系；根据设备类型选择调用相应的驱动程序

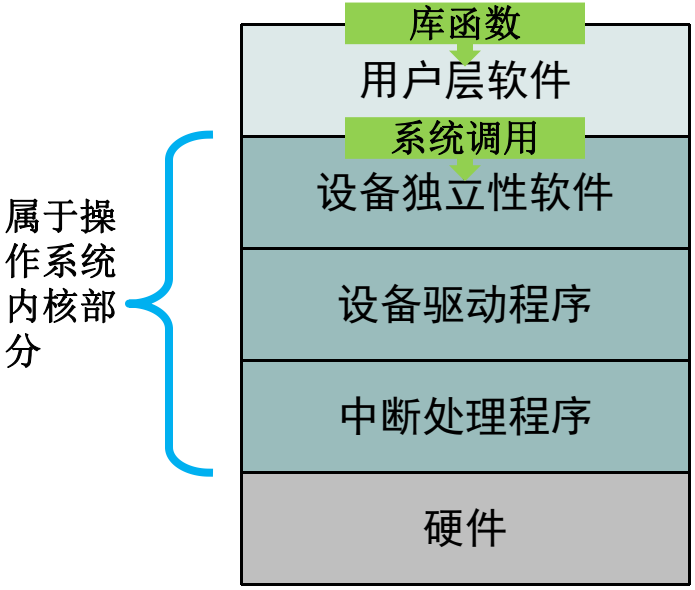
逻辑设备名	物理设备名	驱动程序入口地址
/dev/打印机1	3	1024
/dev/打印机2	5	2046

I/O设备被当做一种特殊的文件

不同类型的I/O设备需要有不同的驱动程序处理

操作系统系统可以采用两种方式管理逻辑设备表（LUT）： 第一种方式，整个系统只设置一张LUT，这就意味着所有用户不能使用相同的逻辑设备名，因此这种方式只适用于单用户操作系统。
第二种方式，为每个用户设置一张LUT，各个用户使用的逻辑设备名可以重复，适用于多用户操作系统。系统会在用户登录时为其建立一个用户管理进程，而LUT就存放在用户管理进程的PCB中。

设备独立性软件



主要实现的功能：
⑥建立逻辑设备名到物理设备名的映射关系；根据设备类型选择调用相应的驱动程序

逻辑设备名	物理设备名	驱动程序入口地址
/dev/打印机1	3	1024
/dev/打印机2	5	2046

I/O设备被当做一种特殊的文件

不同类型的I/O设备需要有不同的驱动程序处理

思考：为什么不同类型的I/O设备需要有不同的驱动程序处理？

为何不同的设备需要不同的设备驱动程序？

各式各样的设备，外形不同，其内部的电子部件（I/O控制器）也有可能不同

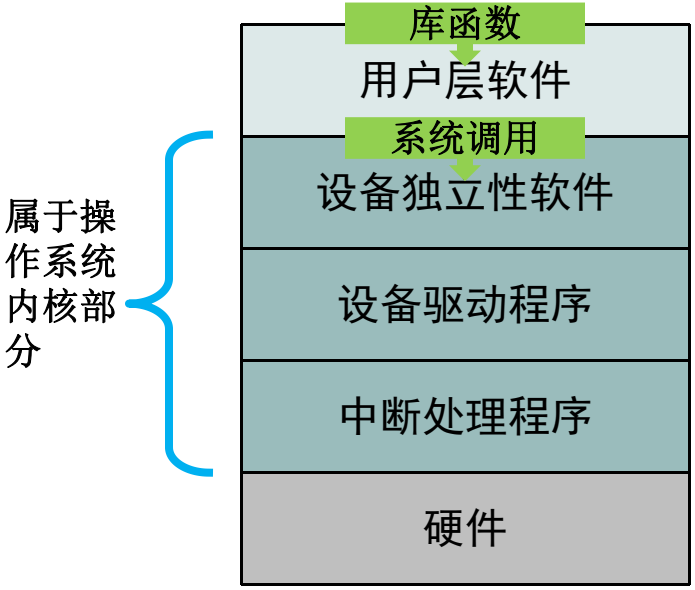
为何不同的设备需要不同的设备驱动程序？

佳能打印机的厂家规定状态寄存器为 0 代表空闲，1代表忙碌。有两个数据寄存器

惠普打印机的厂家规定状态寄存器为 1代表空闲，0代表忙碌。 有一个数据寄存器

不同设备的内部硬件特性也不同，这些特性只有厂家才知道，因此厂家须提供与设备相对应的驱动程序，CPU执行驱动程序的指令序列，来完成设置设 备寄存器，检查设备状态等工作

设备驱动程序



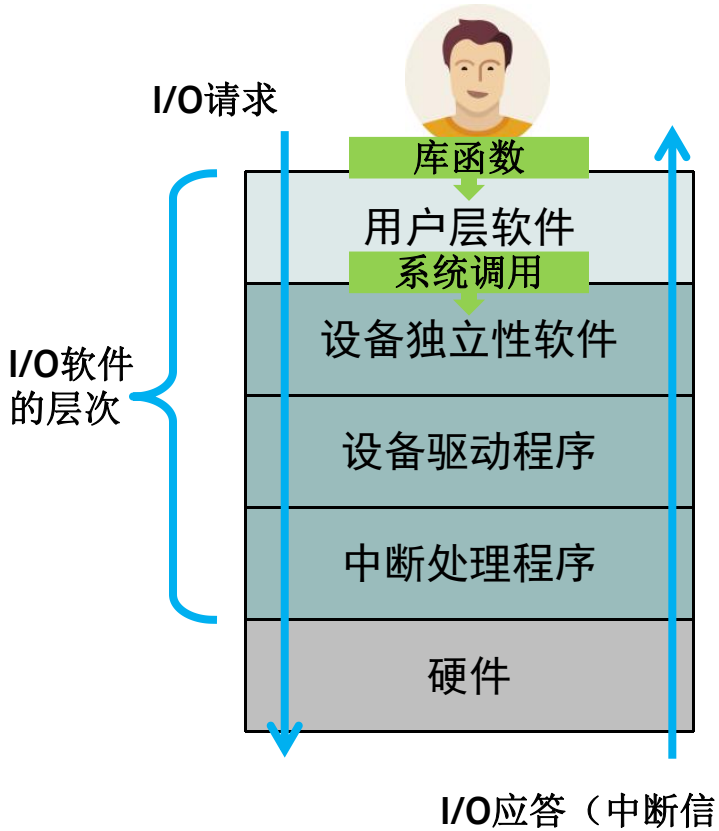
逻辑设备名	物理设备名	驱动程序入口地址
/dev/打印机1	3	1024
/dev/打印机2	5	2046

主要负责对硬件设备的具体控制，将上层发出的一系列命令（如read/write）转化成特定设备“能听得懂”的一系列操作。包括设置设备寄存器；检查设备状态等

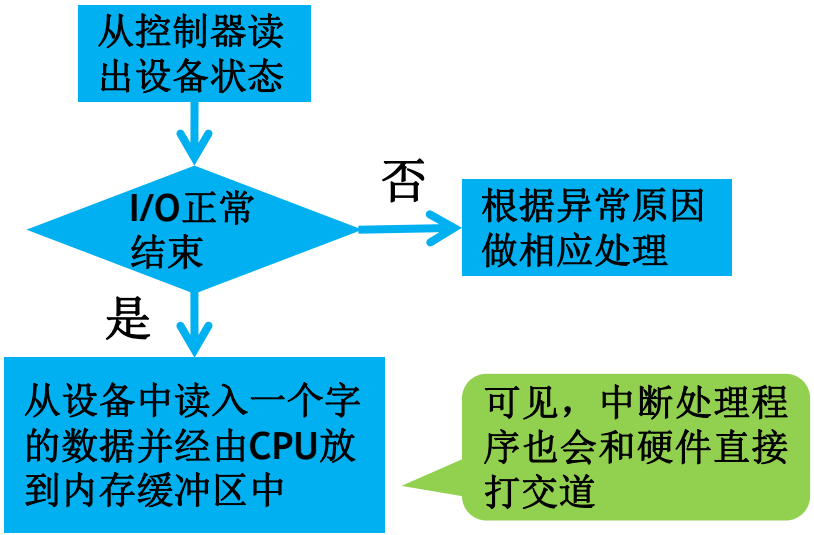
不同的I/O设备有不同的硬件特性，具体细节只有设备的厂家才知道。因此厂家需要根据设备的硬件特性设计并提供相应的驱动程序

驱动程序一般会以一个独立进程的方式存在

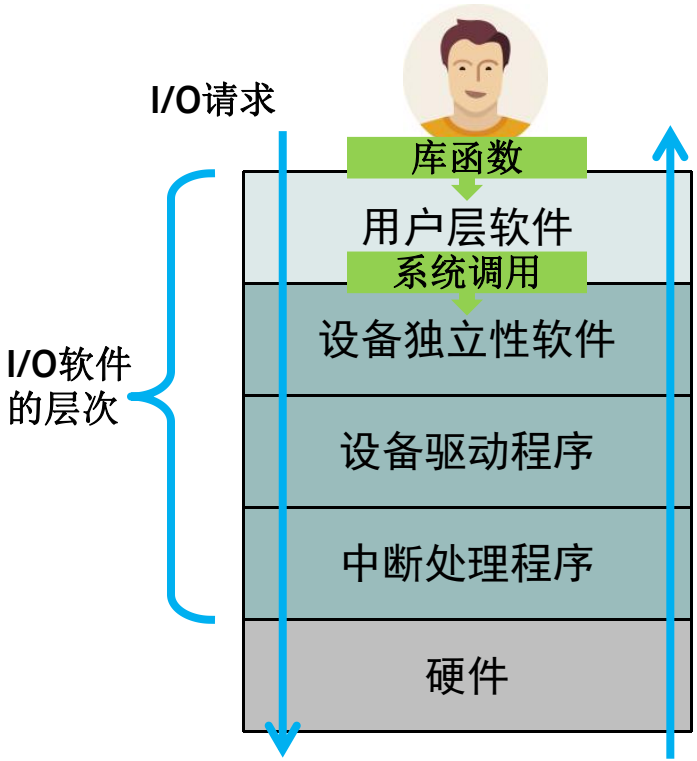
中断处理程序



当I/O任务完成时，I/O控制器会发送一个中断信号，系统会根据中断信号类型找到相应的中断处理程序并执行。中断处理程序的处理流程如下：



中断处理程序

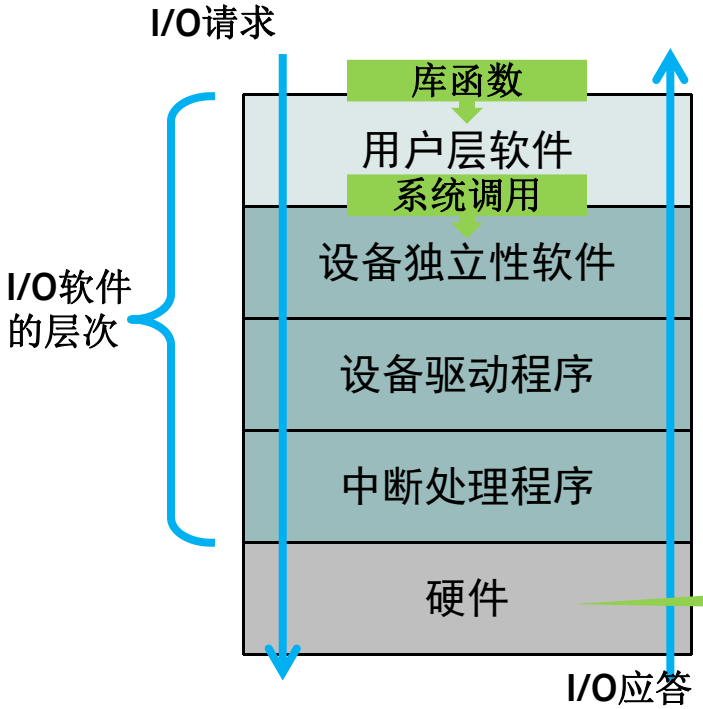


当I/O任务完成时，I/O控制器会发送一个中断信号，系统会根据中断信号类型找到相应的中断处理程序并执行。中断处理程序的处理流程如下：

用户通过调用用户层软件提供的库函数发出的I/O请求

- 用户层软件通过“系统调用”请求设备独立性软件层的服务
- 设备独立性软件层根据LUT调用设备对应的驱动程序
- 驱动程序向I/O控制器发出具体命令
- 等待I/O完成的进程应该被阻塞，因此需要进程切换，而进程切换必然需要中断处理

知识回顾



实现与用户交互的接口，向上提供方便易用的库函数

①向上层提供统一的调用接口（如 read/write 系统调用）；②设备的保护；③差错处理；④设备的分配与回收；⑤数据缓冲区管理；⑥建立逻辑设备名到物理设备名的映射关系；根据设备类型选择调用相应的驱动程序...

设置设备寄存器、检查设备状态

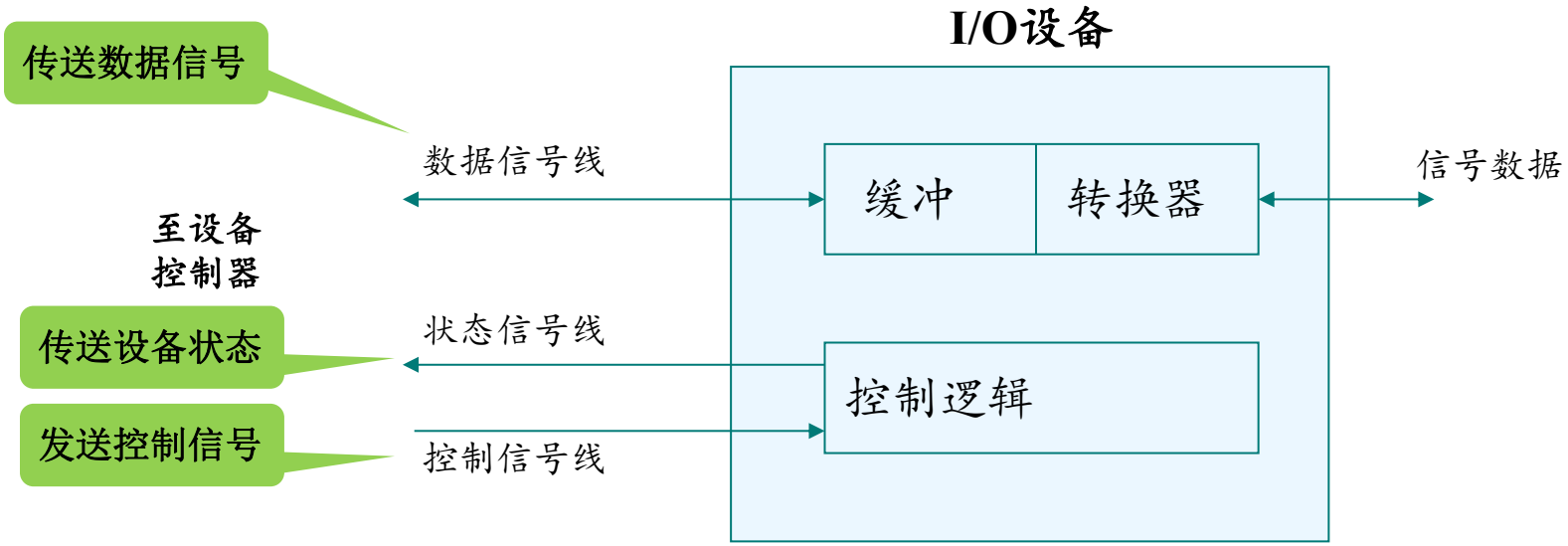
进行中断处理

执行I/O操作，有机械部件、电子部件组成

只需理解一个特点即可：直接涉及到硬件具体细节、且与中断无关的操作肯定是在设备驱动程序层完成的；没有涉及硬件的、对各种设备都需要进行的管理工作都是在设备独立性软件层完成的）

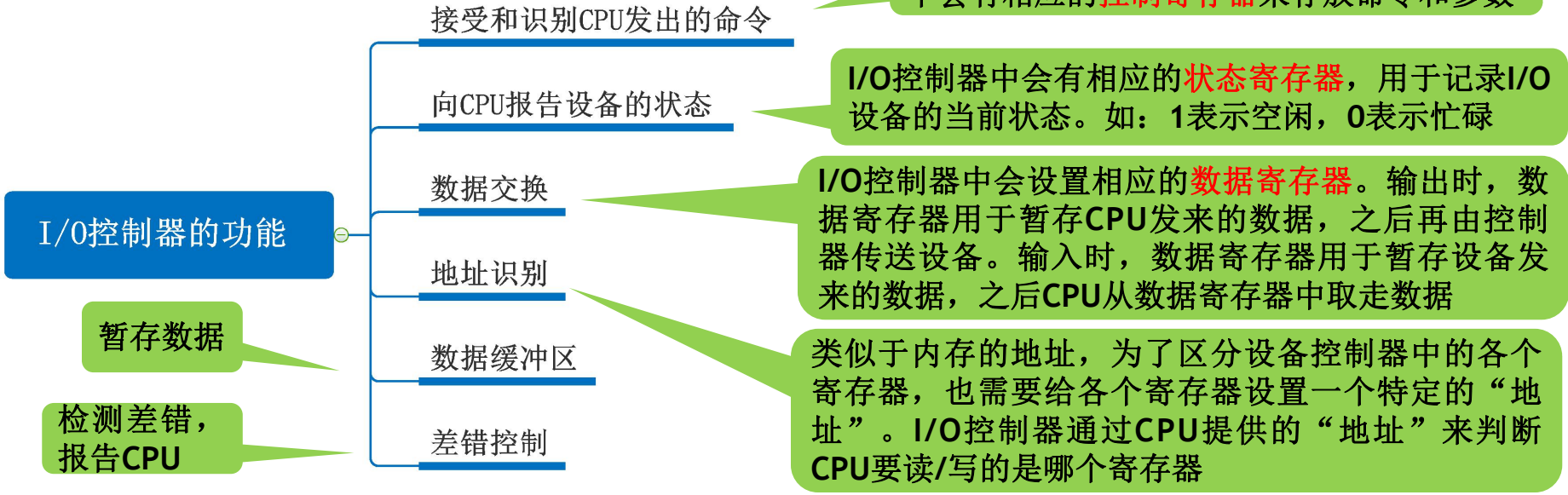
设备与控制器之间的接口

- 通常设备并不是直接与CPU进行通信，而是与设备控制器通信，因此，在设备与设备控制器之间有一接口，在该接口中有三种类型的信号，各对应一条信号线。（如下页图）

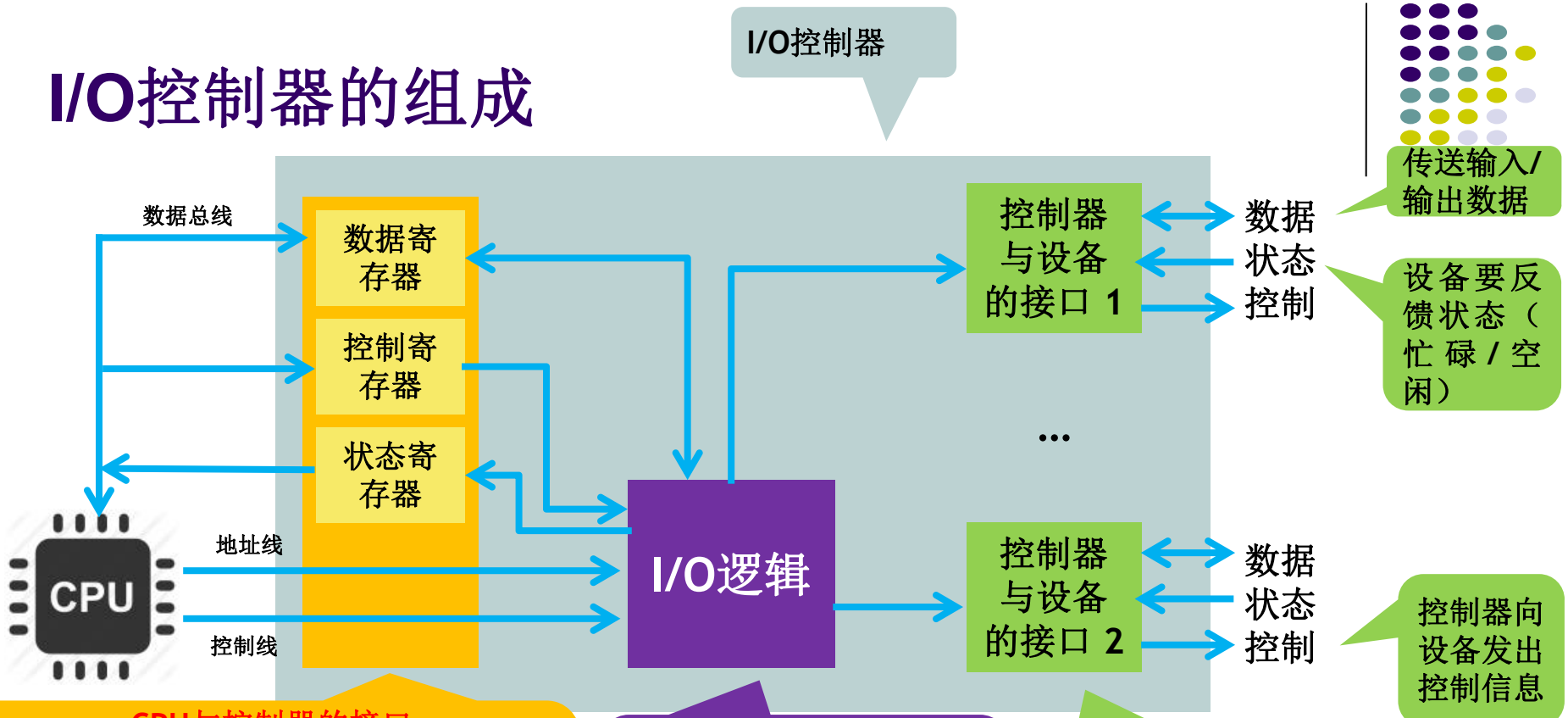


设备控制器

CPU无法直接控制I/O设备，因此I/O设备还要有一个电子部件作为CPU和I/O设备机械部件之间的“中介”，用于实现CPU对设备的控制。
这个电子部件就是I/O控制器，又称设备控制器。CPU可控制I/O控制器，又由I/O控制器来控制设备。



I/O控制器的组成



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

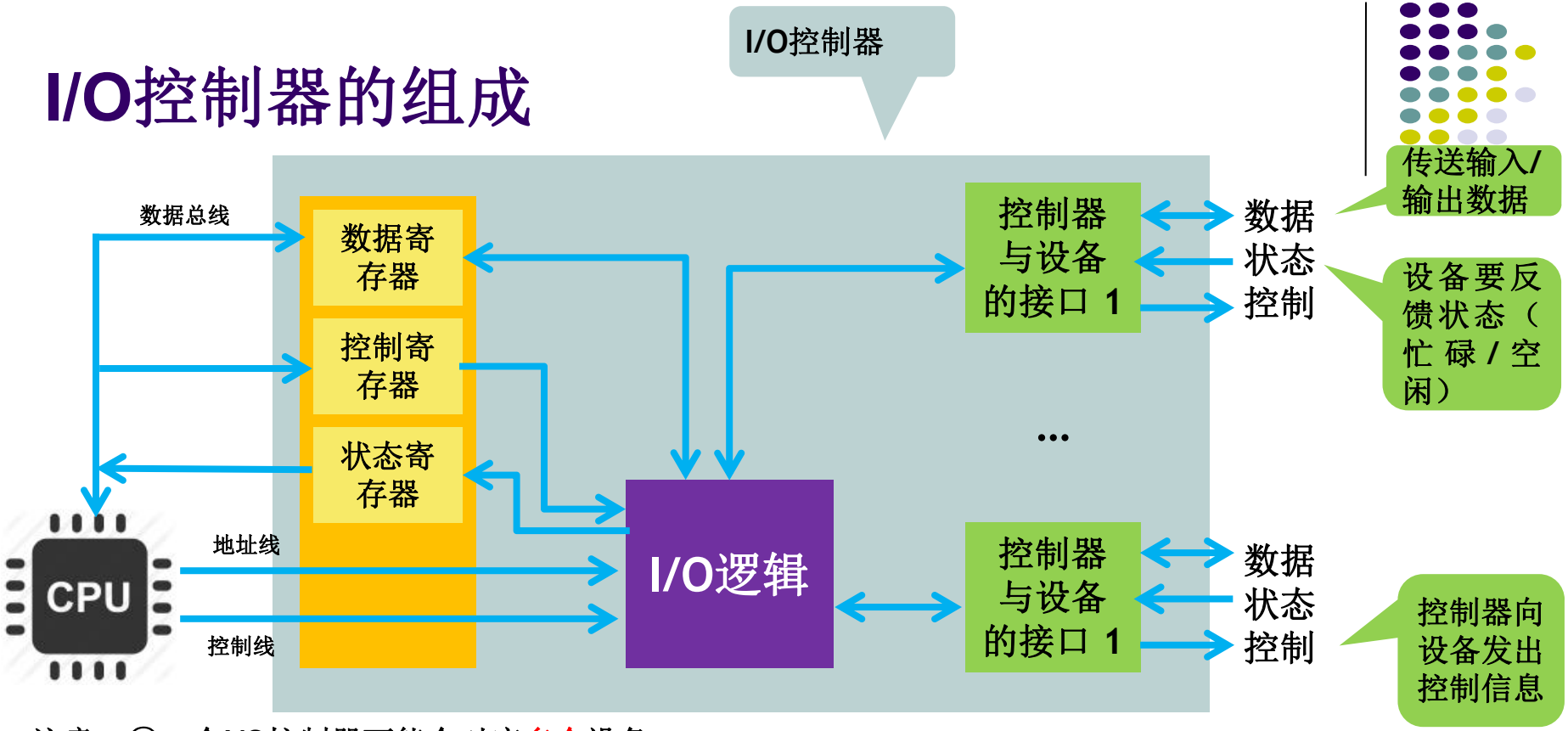
I/O逻辑

负责接收和识别CPU的各种命令（如地址译码），并负责对设备发出命令

控制器与设备的接口

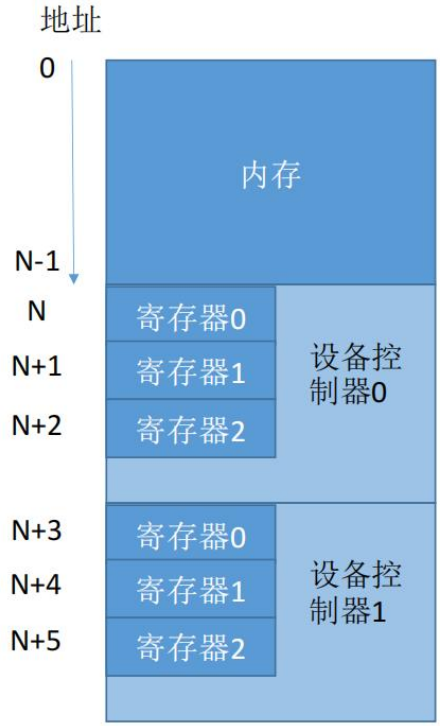
用于实现控制器与设备之间的通信

I/O控制器的组成



注意：①一个I/O控制器可能会对应多个设备；
②数据寄存器、控制寄存器、状态寄存器可能有多个（如：每个控制/状态寄存器对应一个具体的设备），且这些寄存器都要有相应的地址，才能方便CPU操作。有的计算机会让这些寄存器占用内存地址的一部分，称为内存映像I/O；另一些计算机则采用I/O专用地址，即寄存器独立编址。

内存映像I/O v.s. 寄存器独立编址

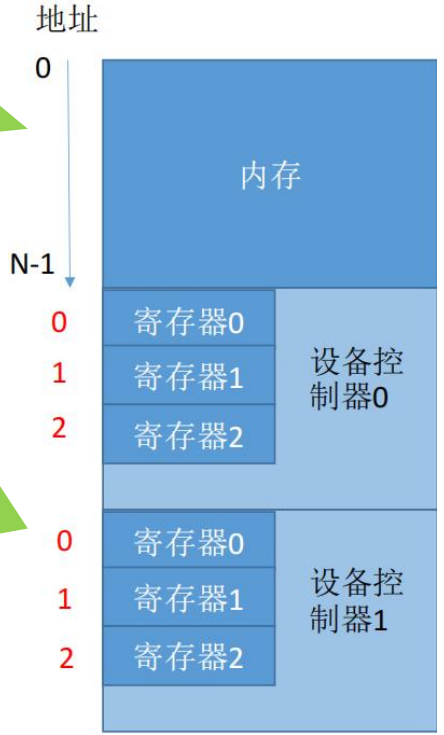


内存映射I/O。控制器中的寄存器与内存地址统一编址

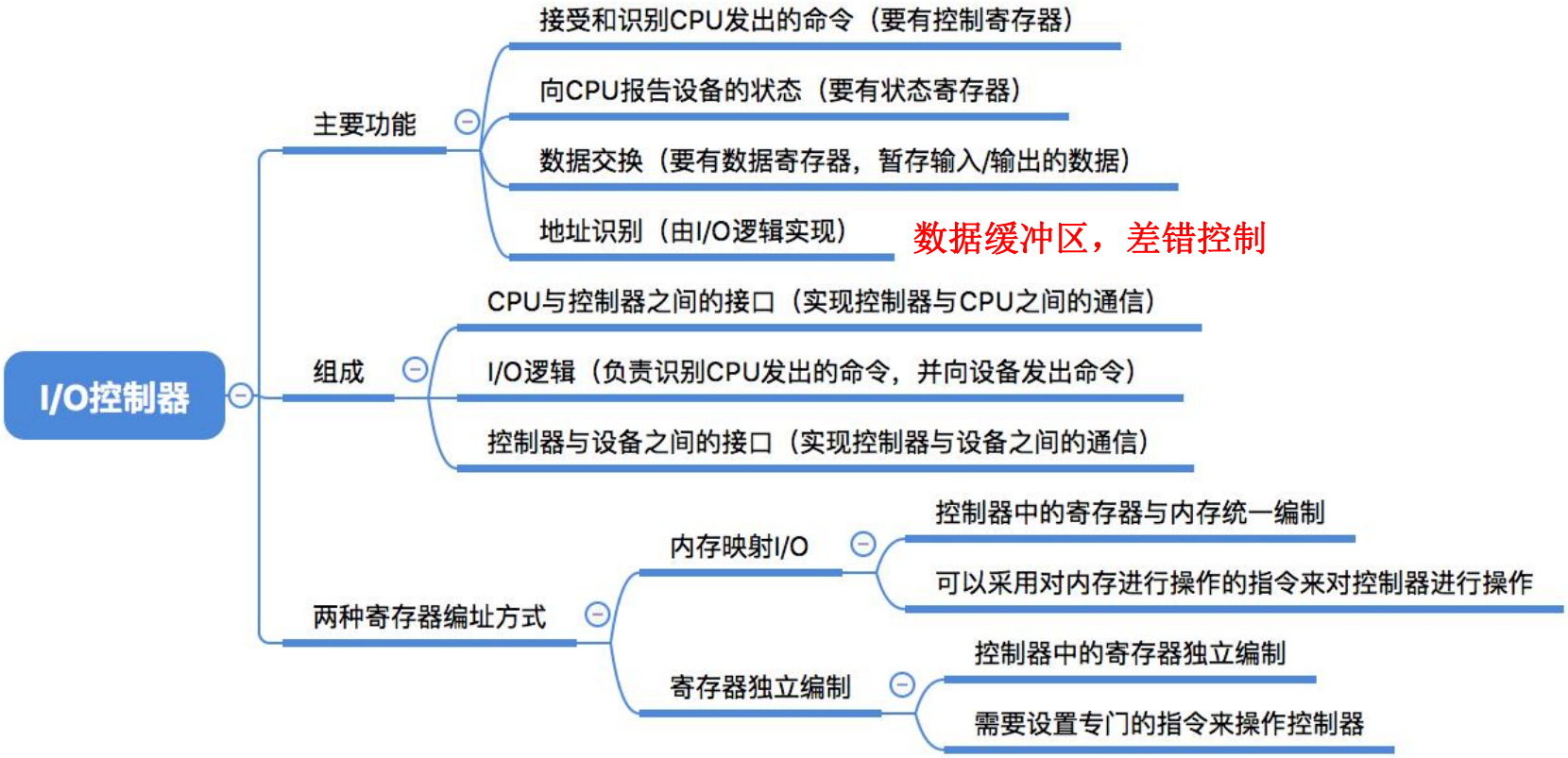
优点：简化了指令。可以采用对内存进行操作的指令来对控制器进行操作

寄存器独立编址。控制器中的寄存器使用单独的地址

缺点：需要设置专门的指令来实现对控制器的操作，不仅要指明寄存器的地址，还要指明控制器的编号



知识总览


I/O控制器
主要功能
接受和识别CPU发出的命令（要有控制寄存器）
向CPU报告设备的状态（要有状态寄存器）
数据交换（要有数据寄存器，暂存输入/输出的数据）
地址识别（由I/O逻辑实现）
数据缓冲区，差错控制
组成
CPU与控制器之间的接口（实现控制器与CPU之间的通信）
I/O逻辑（负责识别CPU发出的命令，并向设备发出命令）
控制器与设备之间的接口（实现控制器与设备之间的通信）
两种寄存器编址方式
内存映射I/O
控制器中的寄存器与内存统一编制
可以采用对内存进行操作的指令来对控制器进行操作
寄存器独立编制
控制器中的寄存器独立编制
需要设置专门的指令来操作控制器

I/O通道

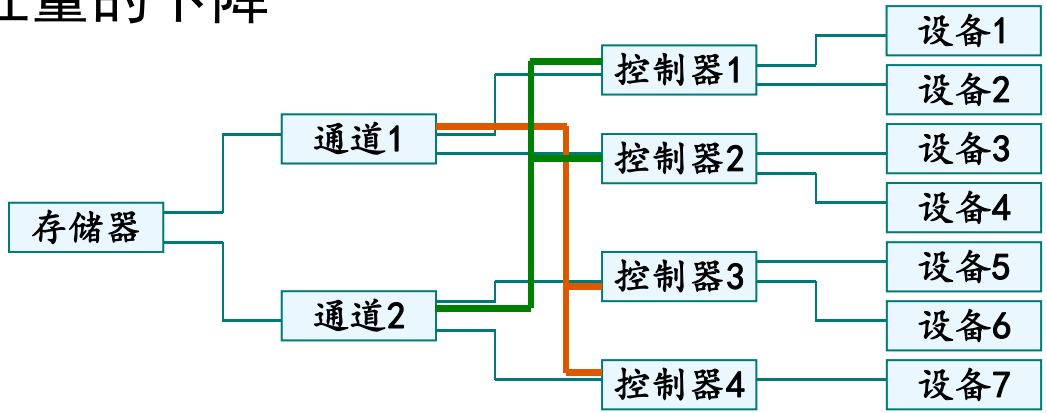
- 通道
  - 通道是专门用于处理I/O的处理机，它控制内存和外设直接进行数据交换
  - 目的：减轻CPU的负担
  - 工作方式：CPU向通道发送I/O命令，通道执行通道程序，完成I/O后向CPU发送中断信号
- 通道与一般处理机的不同
  - 指令类型单一，仅能执行I/O指令。
  - 通道没有自己的内存，通道程序放在主存中

通道的类型

- 字节多路通道：
  - 是一种按字节交叉方式工作的通道，采用多路分时复用——按时间片轮转方式共享主通道。
- 数组选择通道：独占使用，成组（块）传送；
- 数组多路通道：<上述两种技术的结合>

“瓶颈”问题

- 由于通道价格昂贵，致使数量较少，使它成为I/O系统的瓶颈，进而造成系统吞吐量的下降



解决“瓶颈”问题最有效的办法是增加设备到主机间的通路而不增加通道

设备管理

- 5.1 I/O系统
- 5.2 I/O控制方式
- 5.3 缓冲管理
- 5.4 设备分配
- 5.5 设备处理
- 5.6 磁盘存储器管理

设备驱动程序

- 设备处理程序又称设备驱动程序，是I/O进程与设备控制器之间的通信程序
- 主要任务：
  - 接收上层软件发来的抽象要求，比如：Read、Write等命令，再把它们转化为具体要求发送给设备控制器，启动设备执行；
  - 同时，还负责把设备控制器发来的信号传送给上层软件

设备驱动程序的功能

- 接收由设备独立性软件发来的命令和参数，转换为具体要求。（盘块号->盘面、磁道号、扇区号）
- 检查用户I/O请求的合法性，了解设备状态，传递或设置参数
- 发出I/O命令，启动设备，或挂在相应的设备队列上
- 及时响应由控制器或通道发来的中断请求，根据中断类型调用相应的中断处理程序进行处理。

设备驱动程序的特点

- 驱动程序主要是指在请求I/O的进程与设备控制器之间的一个通信和转换程序。
- 驱动程序与设备控制器和I/O设备的硬件特性紧密相关，因而对不同类型的设备应配置不同的驱动程序。
- 驱动程序与I/O设备所采用的I/O控制方式紧密相关。（中断驱动、DMA方式）
- 其中一部分必须用汇编语言书写，固化在ROM中。
- 驱动程序应允许可重入。

设备处理方式

- 为每一类设备设置一个进程，专门用于执行该类设备的I/O操作。
- 在整个系统中设置一个I/O进程，专门用于执行系统中所有各类设备的I/O操作。
- 不设置专门的设备处理进程，只为各类设备设置相应的设备驱动程序，供用户进程或系统进程调用。（较多采用）

设备驱动程序的处理过程

- 设备驱动程序的主要任务是启动指定设备。具体的处理过程如下：
  - 将抽象要求转换为具体要求
  - 检查I/O请求的合法性
  - 读出和检查设备的状态
  - 传送必要的参数（磁盘在读写前，要传递参数至控制器的寄存器中）
  - 启动I/O设备

中断机构和中断处理程序

- 中断是多道程序实现的基础（进程切换），也是设备管理的基础（处理机和外设并行执行）。中断是IO系统最低一层，是整个IO系统的基础
- 中断和陷入
  - 中断——CPU对IO设备发来的中断信号的一种响应。中断是由外部设备引起，也称为外中断。
  - 陷入——由CPU内部事件引起的中断。如溢出、非法指令、地址越界、电源故障等。也称为内中断。

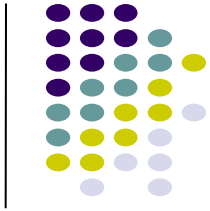
中断向量表和中断优先级

- 中断向量表
  - 为每种设备配以相应的中断处理程序，把该程序的入口地址放在中断向量表的一个表项中，并规定一个中断号用于设备的中断请求。
- 中断优先级
  - 系统中有多个中断信号源，系统为他们规定不同的优先级。如：键盘<打印机<磁盘
- 对多中断源的处理方式
  - 屏蔽（禁止）中断
    - 对任何新到的中断请求都暂时不处理，让其等待。
  - 嵌套中断
    - 多中断请求时，优先响应优先级最高的；高优先级中断请求可抢占低优先级中断的处理机。

中断处理程序

- 当一个进程请求I/O操作时，该进程将被挂起。直到设备完成I/O操作后，设备控制器向CPU发送一个中断请求。CPU响应后，转中断处理程序。中断处理程序执行相应的处理，处理完后解除进程的阻塞状态

中断处理程序的步骤

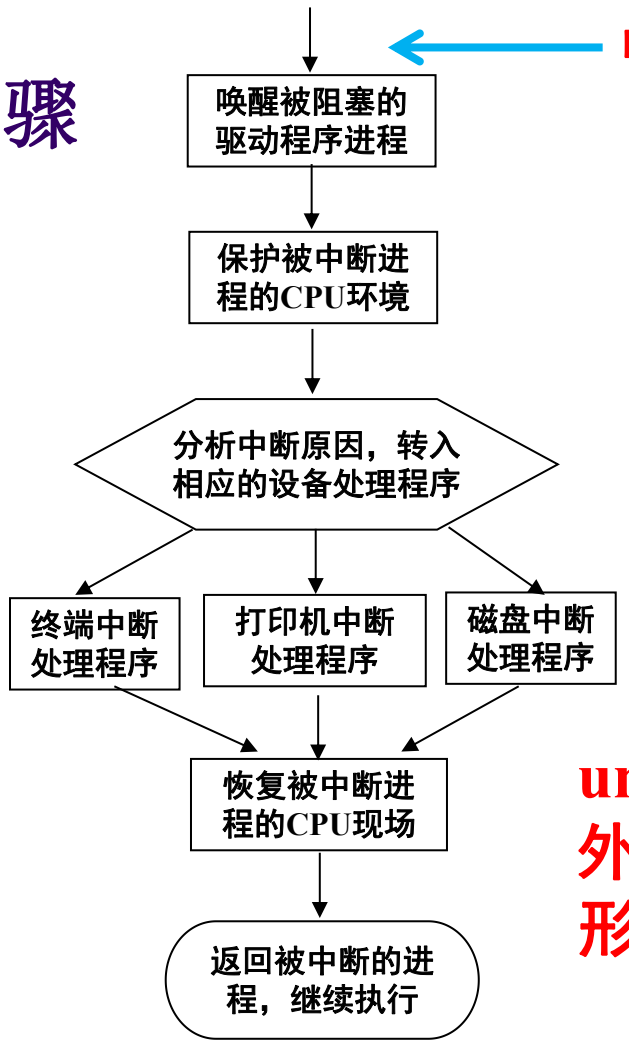


- （1）测定是否有未响应的中断信号
  - 当设备完成一个字符的读入（如字符设备），设备控制器向处理机发送一个中断请求信号，请求处理机将字符读入内存。处理机执行完当前指令后都要检测是否有未响应的中断信号。
- （2）保护被中断进程的CPU环境
  - 保存程序状态字PSW和程序计数器PC中下一条指令的地址，入中断保留区（栈）中。所有CPU寄存器的内容入栈。
- （3）转入相应的设备处理程序
  - 处理机测试各个中断源，确定IO设备，并发送确认信号。设备收到信号后，取消中断请求信号。将设备中断处理程序的入口地址装入到PC中

中断处理程序的步骤

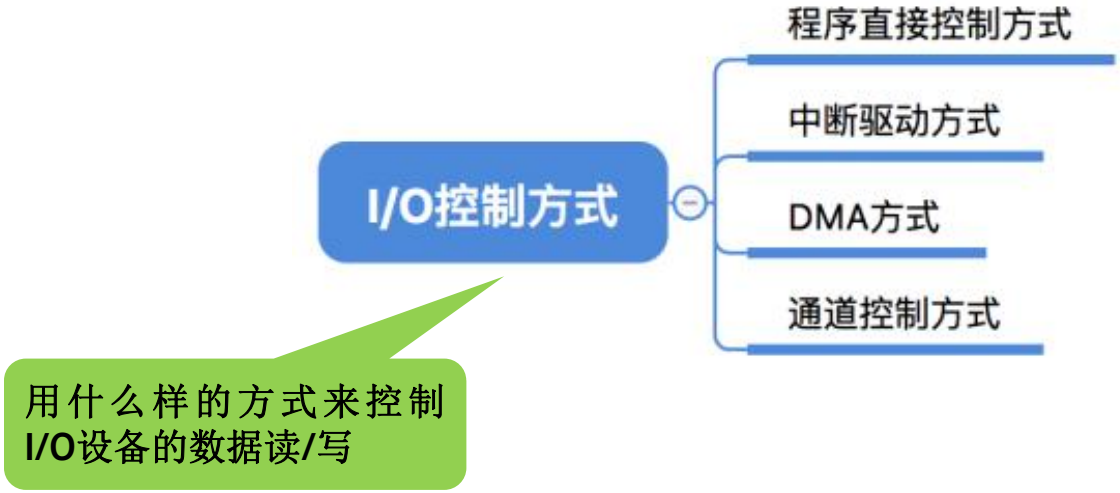
- （4）进行中断处理
  - 不同的设备有不同的中断处理程序
- （5）恢复被中断进程的现场
  - 中断处理完成后，恢复CPU现场，退出中断。是否返回被中断的进程，取决于：
  - a）采用屏蔽中断方式。返回至被中断的进程
  - b）采用中断嵌套方式，考虑有无优先级更高的中断请求

中断处理程序的步骤



unix把除第四步之外的分集中起来，形成总控程序。

I/O控制方式

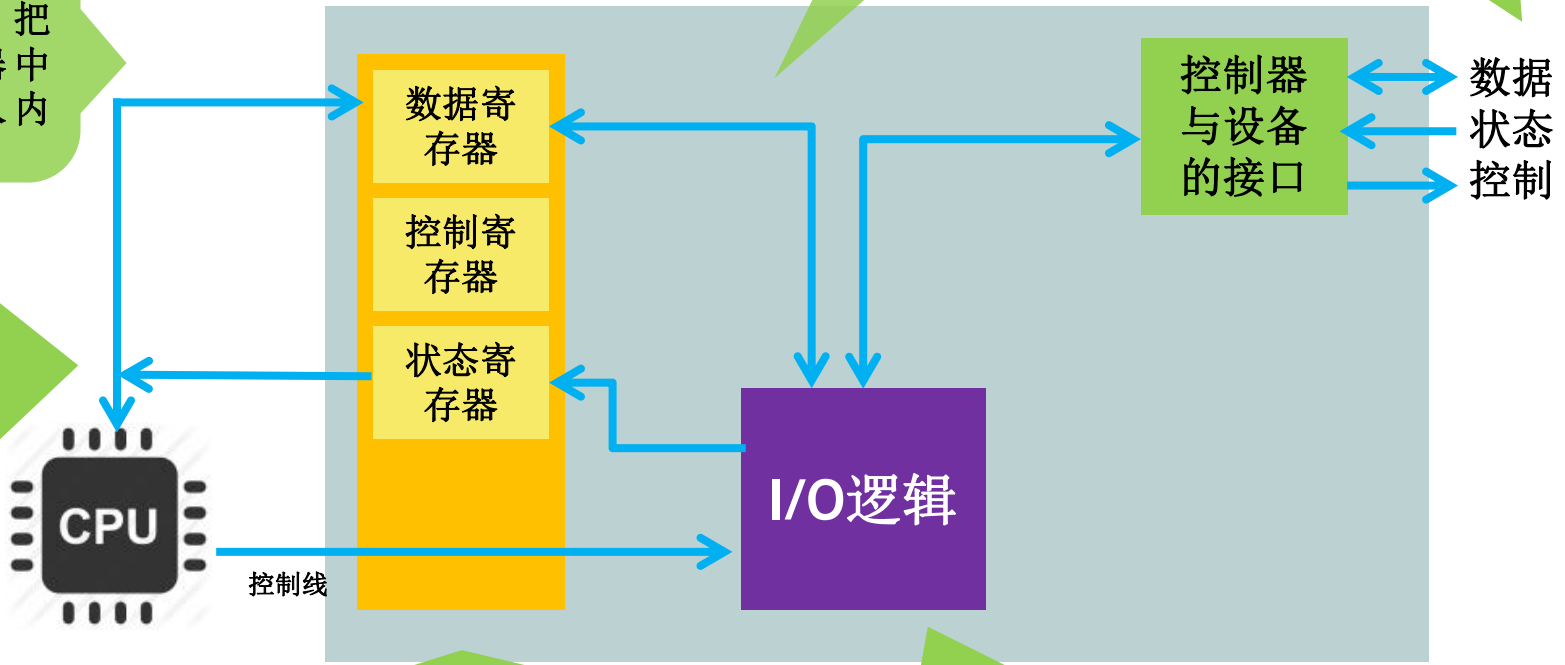


需要注意的问题：

1. 完成一次读/写操作的流程；
2. CPU干预的频率；
3. 数据传送的单位；
4. 数据的流向；
5. 主要缺点和主要优点。

I/O控制器的组成

完成一次读/写操作的流程（以读操作为例）



⑤CPU发现设备已就绪，即可将数据寄存器中的内容读入CPU的寄存器中，再把CPU寄存器中的内容放入内存

④控制器将输入的数据放到数据寄存器中，并将状态改为0（已就绪）

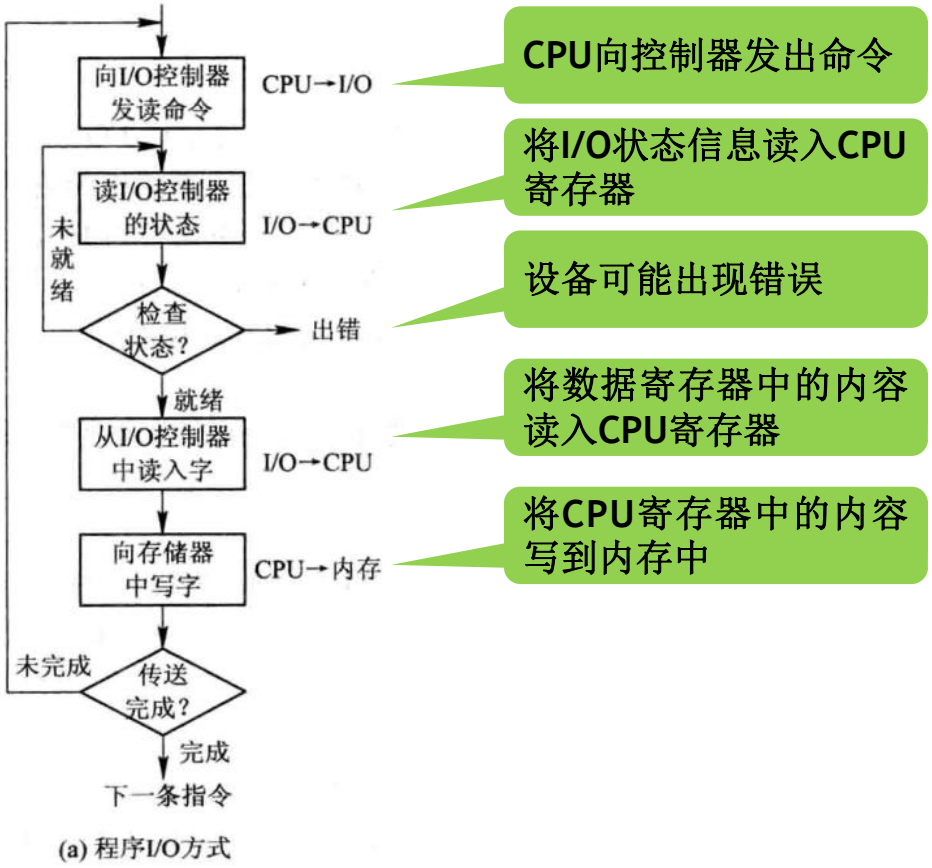
③输入设备准备好数据后将数据传给控制器，并报告自身状态

②轮询检查控制器的状态（其实就是在不断地执行程序的循环，若状态位一直是1，说明设备还没准备好要输入的数据，于是CPU会不断地轮询）

①CPU向控制器发出读指令。于是设备启动，并且状态寄存器设为1（未就绪）

⑥若还要继续读入数据，则CPU继续发出读指令

程序直接控制方式



(a) 程序I/O方式

CPU向控制器发出命令

将I/O状态信息读入CPU寄存器

设备可能出现错误

将数据寄存器中的内容读入CPU寄存器

将CPU寄存器中的内容写到内存中

程序直接控制方式

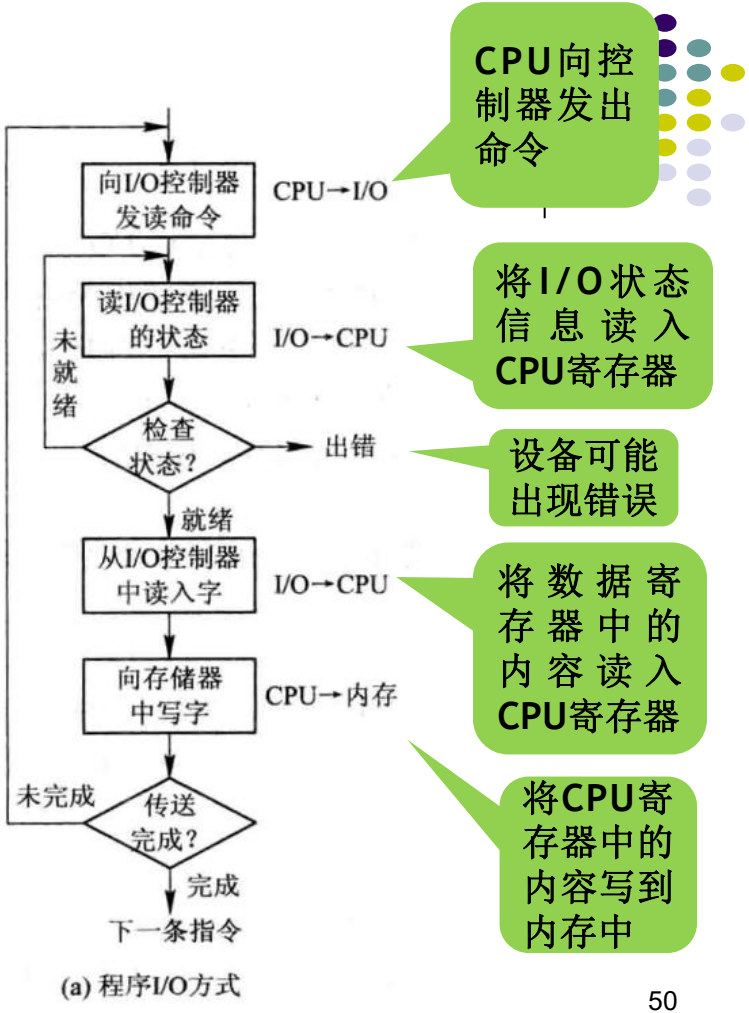
1. 完成一次读/写操作的流程（见右图，Key word：轮询）

2. CPU干预的频率很频繁，I/O操作开始之前、完成之后需要CPU介入，并且在等待I/O完成的过程中CPU需要不断地轮询检查。

3. 数据传送的单位每次读/写一个字

4. 数据的流向
读操作（数据输入）：I/O设备→CPU→内存
写操作（数据输出）：内存→CPU→I/O设备
每个字的读/写都需要CPU的帮助

5. 主要缺点和主要优点
优点：实现简单。在读/写指令之后，加上实现循环检查的一系列指令即可（因此才称为“程序直接控制方式”）
缺点：CPU和I/O设备只能串行工作，CPU需要一直轮询检查，长期处于“忙等”状态，CPU利用率低。

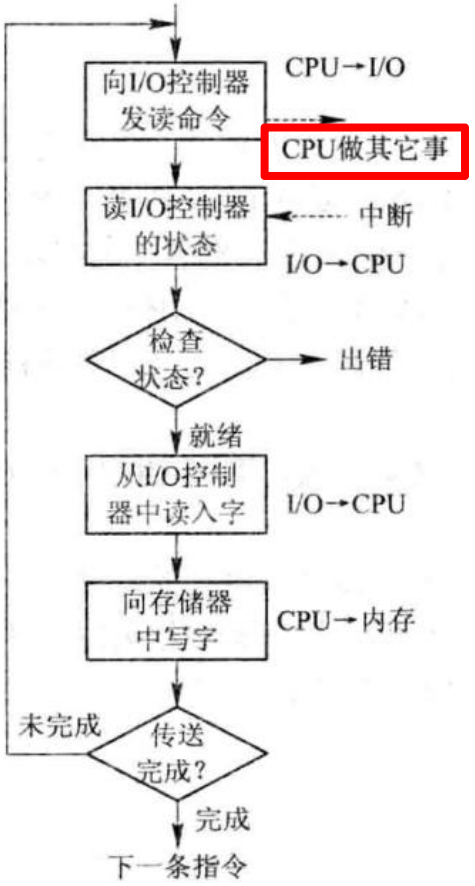


(a) 程序I/O方式

中断驱动方式

引入中断机制。由于I/O设备速度很慢，因此在CPU发出读/写命令后，可将等待I/O的进程阻塞，先切换到别的进程执行。当I/O完成后，控制器会向CPU发出一个中断信号，CPU检测到中断信号后，会保存当前进程的运行环境信息，转去执行中断处理程序处理该中断。处理中断的过程中，CPU从I/O控制器读一个字的数据传送到CPU寄存器，再写入主存。接着，CPU恢复等待I/O的进程（或其他进程）的运行环境，然后继续执行。

注意：
①CPU会在每个指令周期的末尾检查中断；
②中断处理过程中需要保存、恢复进程的运行环境，这个过程是需要一定时间开销的。可见，如果中断发生的频率太高，也会降低系统性能。



(b) 中断驱动I/O方式

中断驱动方式

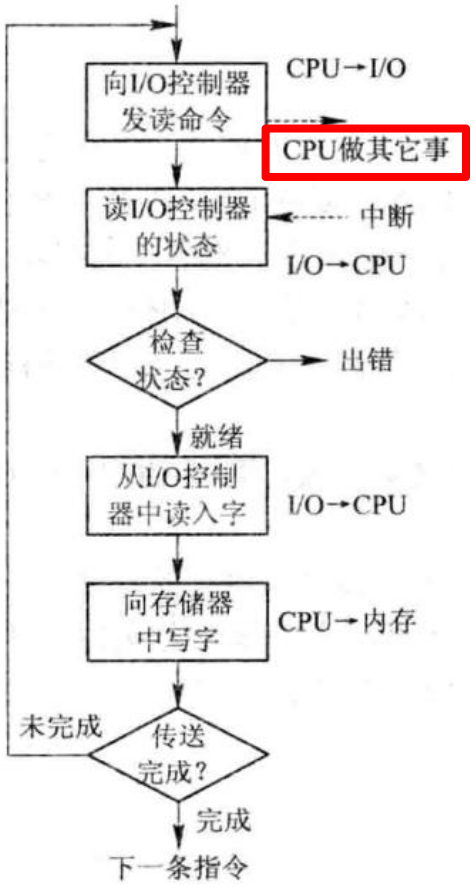
1. 完成一次读/写操作的流程（见右图，Key word：中断）

2. CPU干预的频率
每次I/O操作开始之前、完成之后需要CPU介入。
等待I/O完成的过程中CPU可以切换到别的进程执行。

3. 数据传送的单位
每次读/写一个字

4. 数据的流向
读操作（数据输入）：I/O设备→CPU→内存
写操作（数据输出）：内存→CPU→I/O设备

5. 主要缺点和主要优点
优点：与“程序直接控制方式”相比，在“中断驱动方式”中，I/O控制器会通过中断信号主动报告I/O已完成，CPU不再需要不停地轮询。CPU和I/O设备可并行工作，CPU利用率得到明显提升。
缺点：每个字在I/O设备与内存之间的传输，都需要经过CPU。而频繁的中断处理会消耗较多的CPU时间



(b) 中断驱动I/O方式

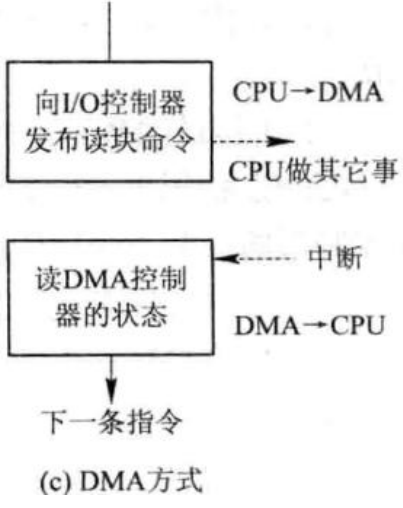
DMA方式

与“中断驱动方式”相比，DMA方式（Direct Memory Access，直接存储器存取。主要用于块设备的I/O控制）有这样几个改进：

①数据的传送单位是“块”。不再是一个字、一个字的传送；

②数据的流向是从设备直接放入内存，或者从内存直接到设备。不再需要CPU作为“快递小哥”。

③仅在传送一个或多个数据块的开始和结束时，才需要CPU干预。

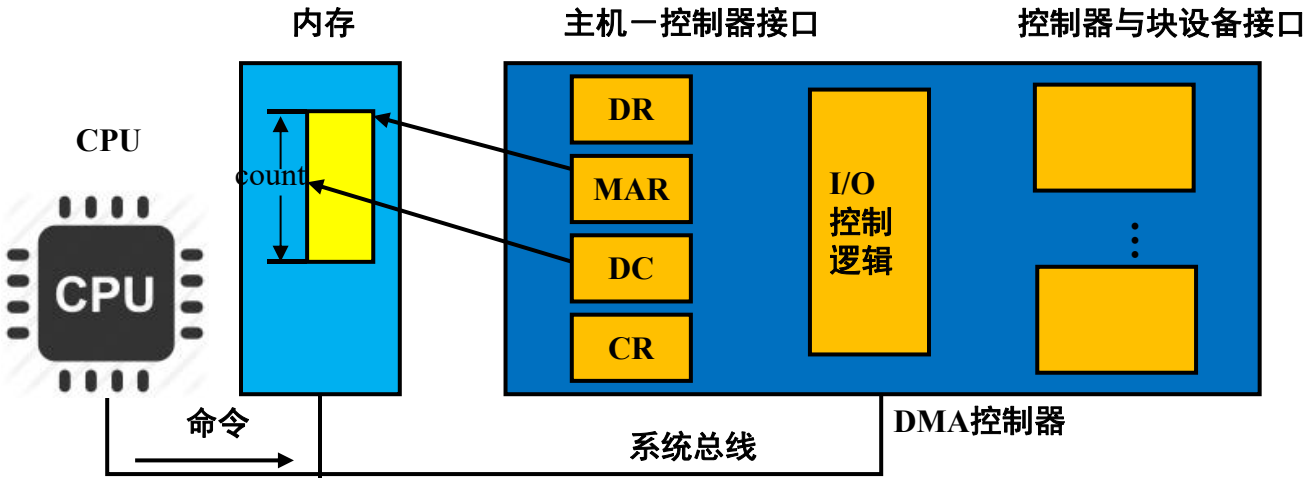


(c) DMA方式

CPU指明此次要进行的操作（如：读操作），并说明要读入多少数据、数据要存放在内存的什么位置、数据在外部设备上的地址（如：在磁盘上的地址）

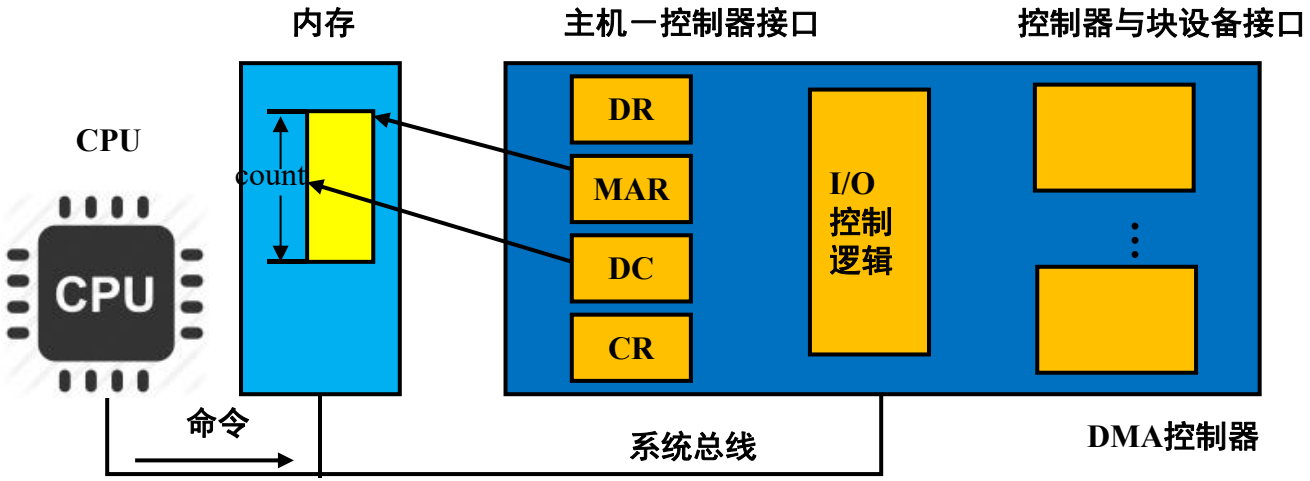
控制器会根据CPU提出的要求完成数据的读/写工作，整块数据的传输完成后，才向CPU发出中断信号

DMA控制器



DR （Data Register，数据寄存器）：暂存从设备到内存，或从内存到设备的数据。
MAR （Memory Address Register，内存地址寄存器）：在输入时，MAR 表示数据应放到内存中的什么位置；输出时 MAR 表示要输出的数据放在内存中的什么位置。
DC （Data Counter，数据计数器）：表示剩余要读/写的字节数。
CR（Command Register，命令/状态寄存器）：用于存放CPU发来的I/O命令，或设备的状态镭息。

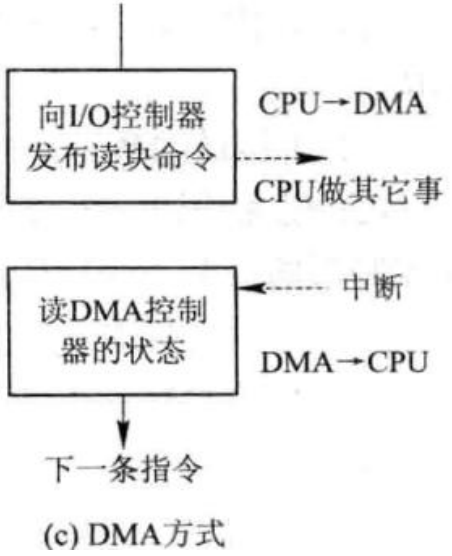
DMA工作过程



- CPU向磁盘控制器发送:读命令到CR(命令/状态寄存器)中、内存的目标起址到MAR(内存地址寄存器)中、数据字节数到DC(数据计数器)中、磁盘中的源地址到控制器的控制逻辑上;
- 启动DMA控制器进行数据传送。此后，CPU可以执行其它任务;
- DMA控制器按照命令传送数据：先从磁盘读入一个字节的数据送入DR(数据寄存器)后，再传送到内存中。
- 修改并检查DC(数据计数器)中的数值：若DC(数据计数器)中的值不为0，则继续传送下一个字节；为0，则发出一个中断请求。

DMA方式

1. 完成一次读/写操作的流程（见右图）



(c) DMA方式

2. CPU干预的频率
仅在传送一个或多个数据块的开始和结束时，才需要CPU干预。

3. 数据传送的单位
每次读/写一个或多个块（注意：每次读写的只能是连续的多个块，且这些块读入内存后在内存中也必须是连续的）

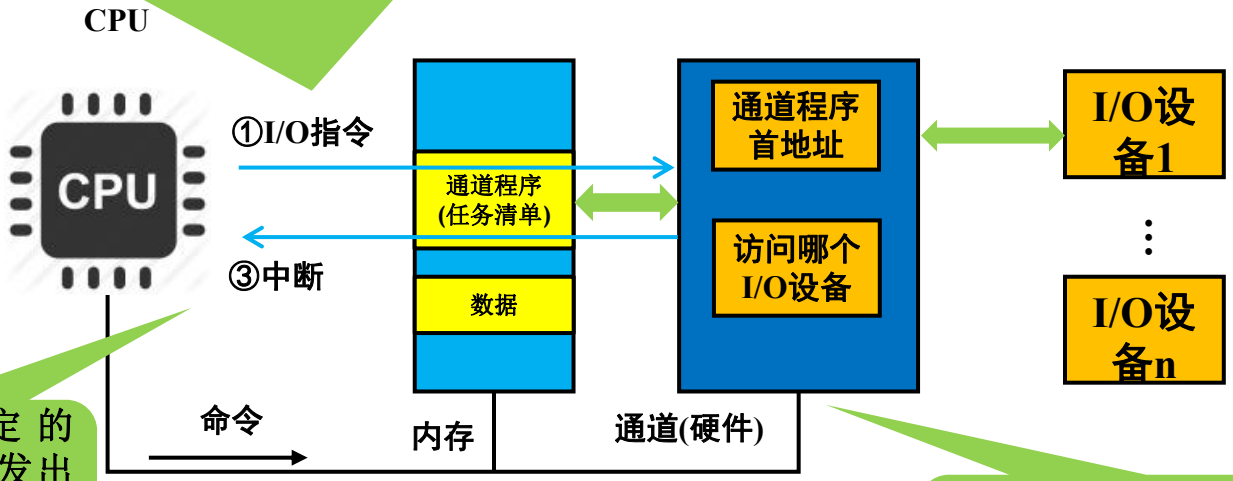
4. 数据的流向（不再需要经过CPU）
读操作（数据输入）：I/O设备→内存
写操作（数据输出）：内存→I/O设备

5. 主要缺点和主要优点
优点：数据传输以“块”为单位，CPU介入频率进一步降低。数据的传输不再需要先经过CPU再写入内存，数据传输效率进一步增加。CPU和I/O设备的并行性得到提升。
缺点：CPU每发出一条I/O指令，只能读/写一个或多个连续的数据块。
如果要读/写多个离散存储的数据块，或者要将数据分别写到不同的内存区域时，CPU要分别发出多条I/O指令，进行多次中断处理才能完成。

通道控制方式

通道：一种硬件，可以理解为是“弱鸡版的CPU”。通道可以识别并执行一系列通道指令

①CPU向通道发出I/O指令。指明通道程序在内存中的位置，并指明要操作的是哪个I/O设备。之后CPU就切换到其他进程执行了



③通道执行完规定 的任务后，向CPU发出中断信号，之后CPU对中断进行处理

②通道执行内存中的通道程序（其中指明了要读入/写出多少数据，读/写的数据应放在内存的什么位置等信息）

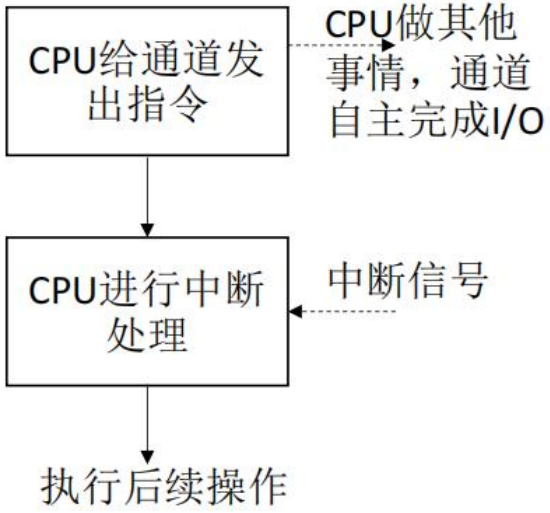
通道控制方式

通道：一种硬件，可以理解为是“弱鸡版的CPU”。通道可以识别并执行一系列通道指令

1. 完成一次读/写操作的流程（见右图）

与CPU相比，通道可以执行的指令很单一，并且通道程序是放在主机内存中的，也就是说通道与CPU共享内存

2. CPU干预的频率
极低，通道会根据CPU的指示执行相应的通道程序，只有完成一组数据块的读/写后才需要发出中断信号，请求CPU干预。

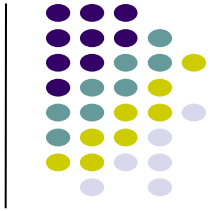


3. 数据传送的单位
每次读/写一组数据块

4. 数据的流向（在通道的控制下进行）
读操作（数据输入）：I/O设备→内存
写操作（数据输出）：内存→I/O设备

5. 主要缺点和主要优点
缺点：实现复杂，需要专门的通道硬件支持
优点：CPU、通道、I/O设备可并行工作，资源利用率很高。

通道程序



- 通道是通过执行通道程序，并与设备控制器共同实现对I/O设备的控制的。
- 通道程序由一系列通道指令（通道命令）构成。
- 每条指令都包含以下信息
  - 操作码——指令执行的操作：读/写
  - 内存地址——字符送入/取出内存的首址
  - 计数——表示本条指令所要读/写数据的字节数
  - 通道程序结束位P　P=1表示本条指令是最后一条
  - 记录结束标志位R　R=1表示这是处理某记录的最后一条指令

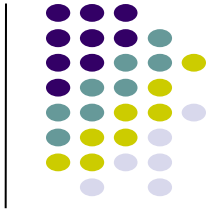
下列通道程序的功能：

- 将内存中不同地址的数据，写成多个记录。

操作	P	R	计数	内存地址
write	0	0	80	813
write	0	0	140	1034
write	0	1	60	5830
write	0	1	300	2000
write	0	0	50	1850
write	1	1	250	720

前三条指令分别是：
将813~892单元中的80个字符和1034~1173单元中的140个字符及5830-5889单元中的60个字符写成一个记录；
第4条指令是单独写一个具有300个字符的记录;第5、6条指令共写含300个字符的记录

知识回顾



	完成一次读/写的过程	CPU干预频率	每次I/O的数据传输单位	数据流向	优缺点
程序直接控制方式	CPU发出I/O命令后需要不断轮询	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了上一阶段的最大缺点。总体来说，整个发展过程就是要尽量减少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，以便更多地去完成数据处理任务。
中断驱动方式	CPU发出I/O命令后可以做其他事，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	高	块	设备→内存 内存→设备	
通道控制方式	CPU发出I/O命令后可以做其他事。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	

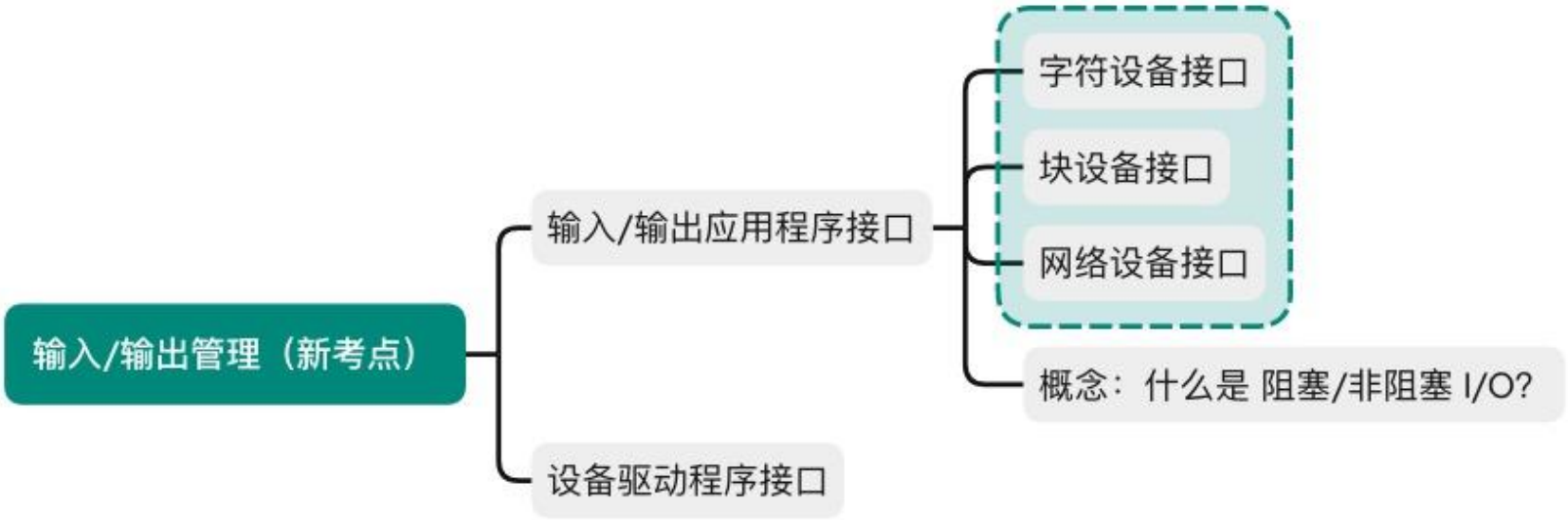
例题

一个典型的文本打印页面有 50 行，每行 80 个字符，假定一台标准的打印机每分钟能打印 6 页，向打印机的输出寄存器中写一个字符的时间很短，可忽略不计。若每打印一个字符都需要花费 50μs 的中断处理时间（包括所有服务），使用中断驱动 I/O 方式运行这台打印机，中断的系统开销占 CPU 的百分比为（　）。

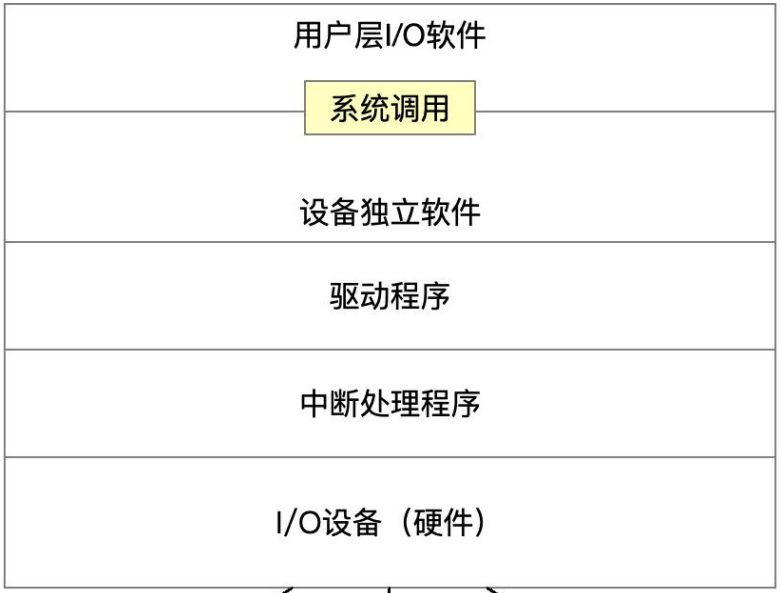
A. 2%　　B. 5%

C. 20%　　D. 50%

新考点


输入/输出管理（新考点）
输入/输出应用程序接口
字符设备接口
块设备接口
网络设备接口
概念：什么是 阻塞/非阻塞 I/O?
设备驱动程序接口

输入/输出应用程序接口



显然，用户层的应用程序无法用一个统一的系统调用接口来完成所有类型设备的I/O

如：键盘、打印机，不可“寻址”，每次读1个字符

字符设备

块设备

如：磁盘，可“寻址”，每次读/写1个块

网络设备

如：网络控制器（网卡），数据该给谁?

输入/输出应用程序接口

字符设备接口
get/put 系统调用：向字符设备读/写一个字符

块设备接口
read/write 系统调用：向块设备的**读写指针位置**读/写多个字符；
seek系统调用：**修改读写指针位置**

网络设备接口，又称“**网络套接字(socket)接口**”
socket 系统调用：**创建一个网络套接字**，需指明网络协议（TCP? UDP?）
bind：将套接字绑定到某个本地“**端口**”
connect：将套接字连接到远程地址
read/write：从套接字读/写数据

如：键盘、打印机，**不可“寻址”，每次读1个字符**

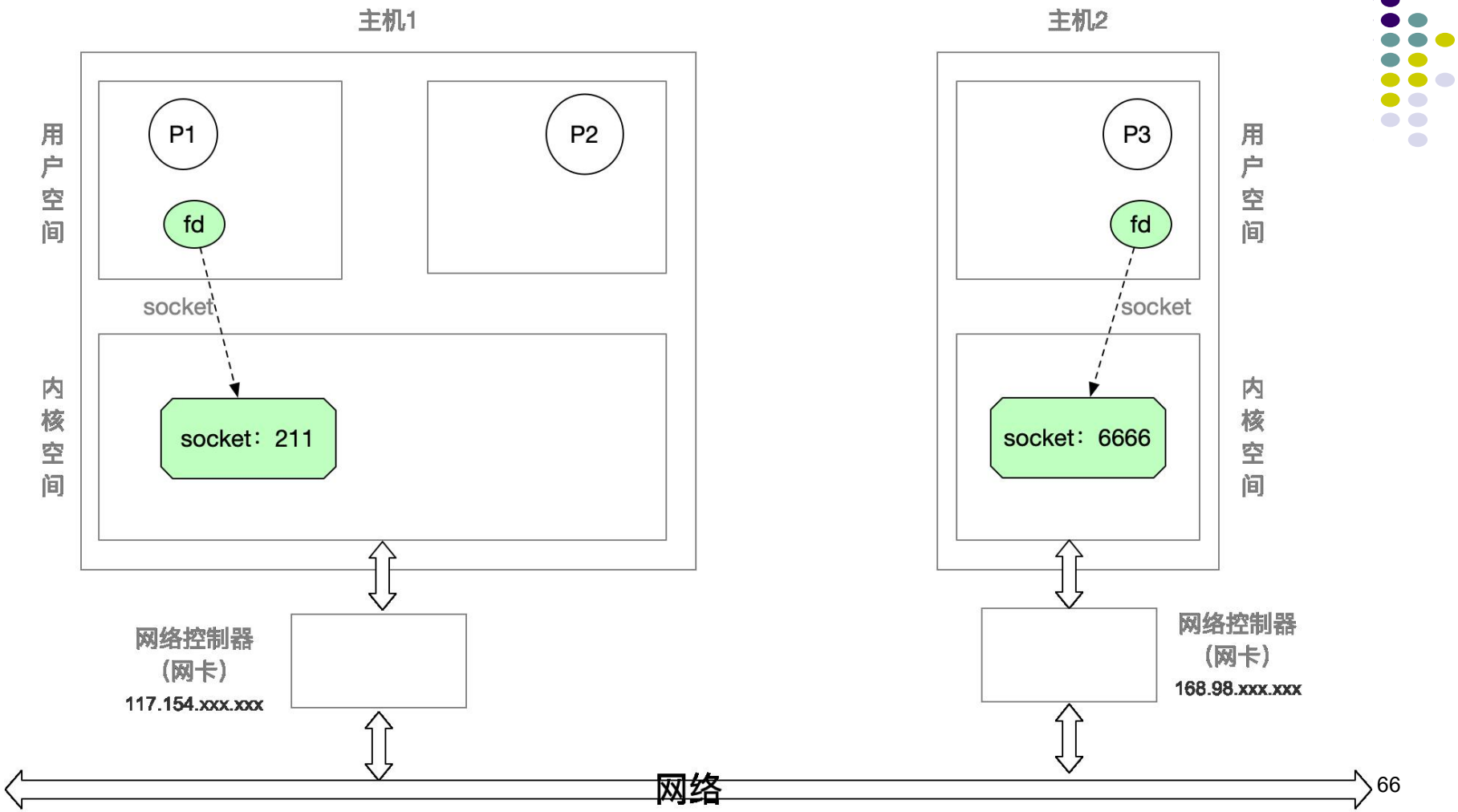
字符设备

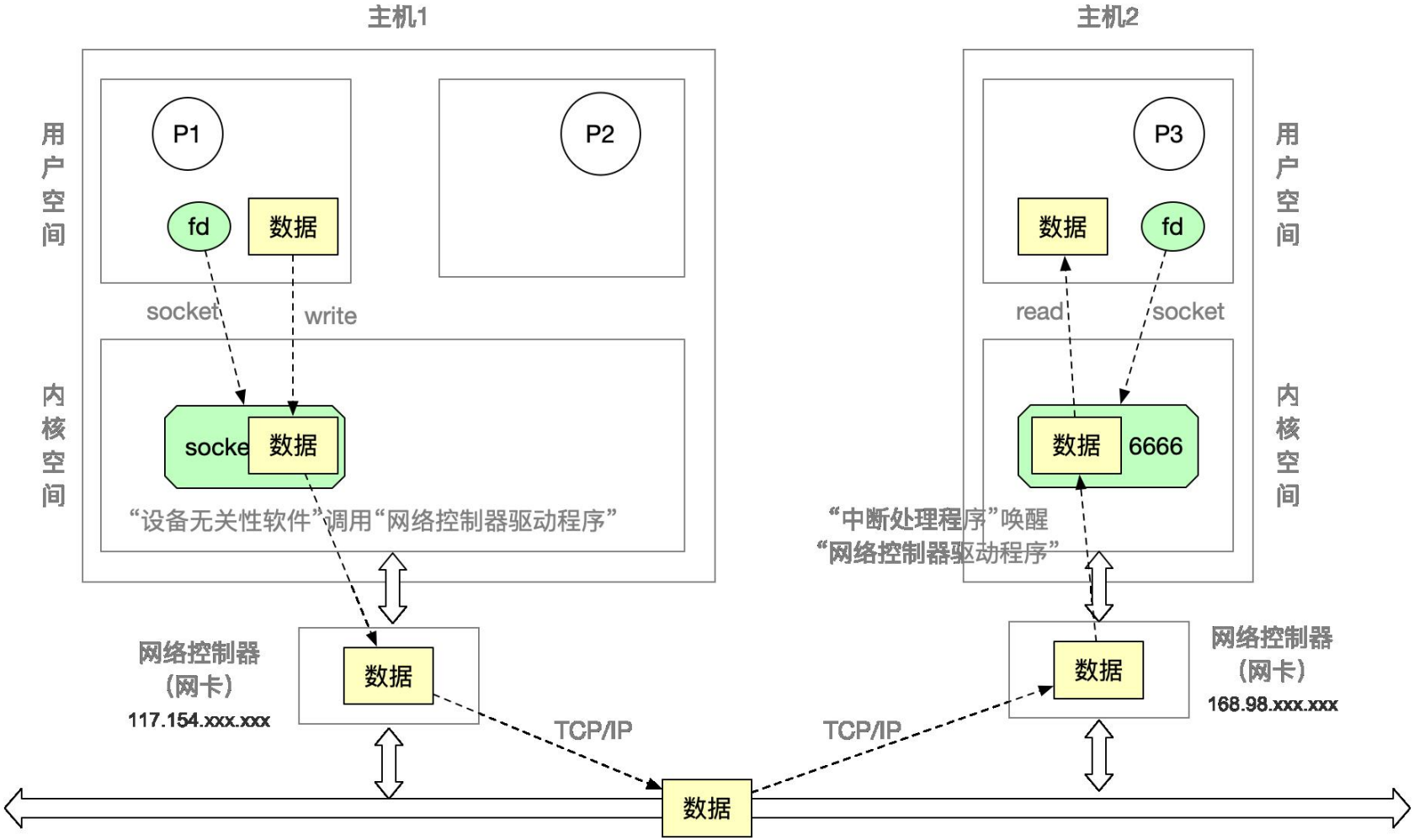
块设备

如：磁盘，**可“寻址”，每次读/写1个块**

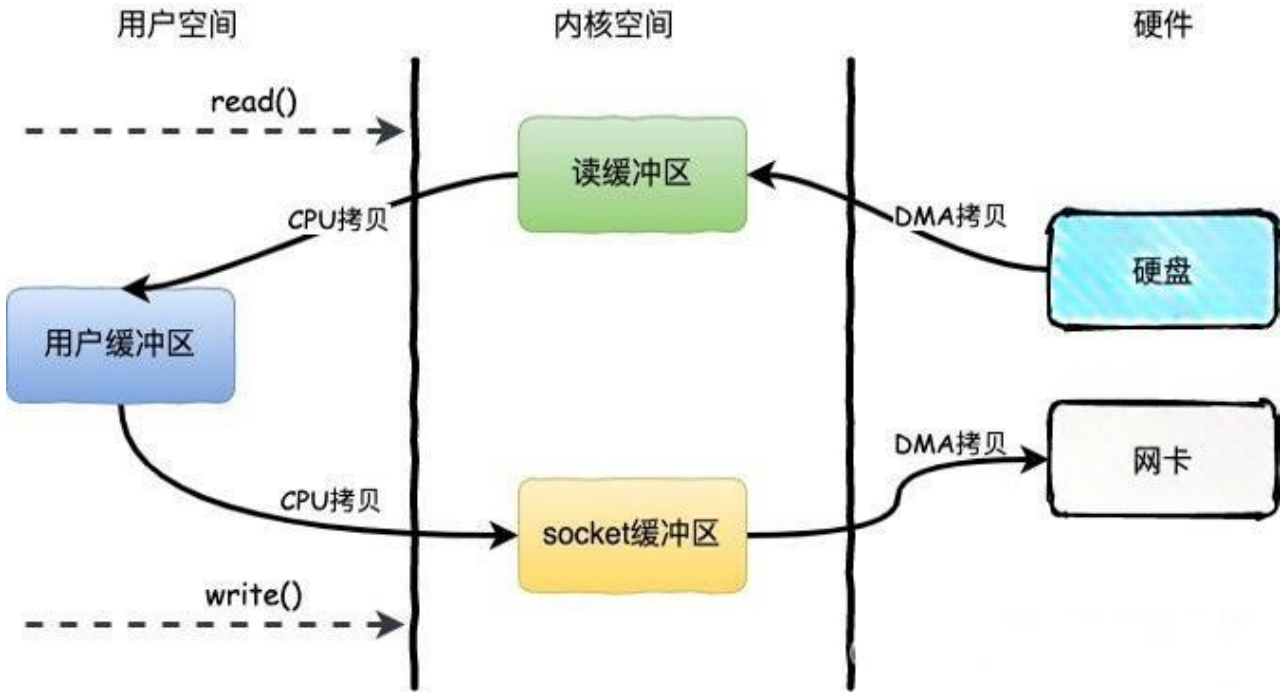
网络设备

如：网络控制器（网卡），**数据该给谁?**


主机1
主机2
用户空间
内核空间
P1
P2
P3
fd
fd
socket
socket
socket：211
socket：6666
网络控制器
（网卡）
117.154.xxx.xxx
网络控制器
（网卡）
168.98.xxx.xxx
网络

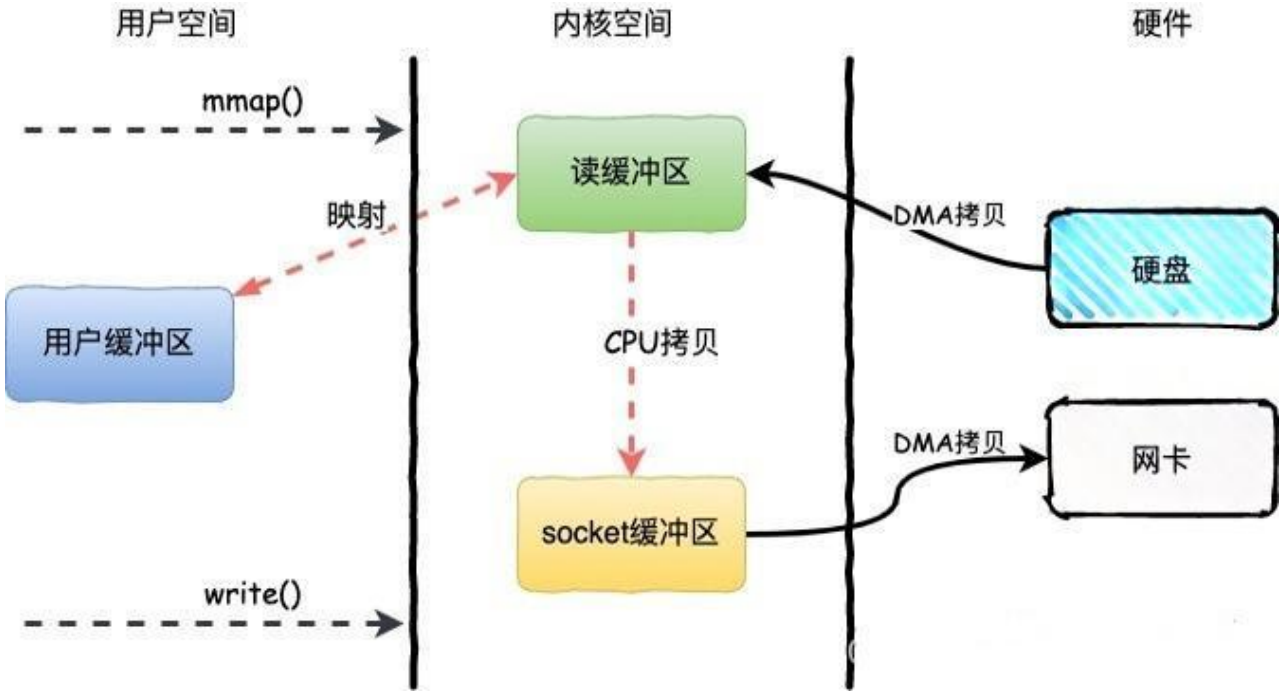

主机1
主机2
用户空间
内核空间
P1
P2
P3
fd
数据
socket
write
read
socke
6666
“设备无关性软件”调用“网络控制器驱动程序”
“中断处理程序”唤醒
“网络控制器驱动程序”
网络控制器
（网卡）
117.154.xxx.xxx
168.98.xxx.xxx
TCP/IP

地址映射



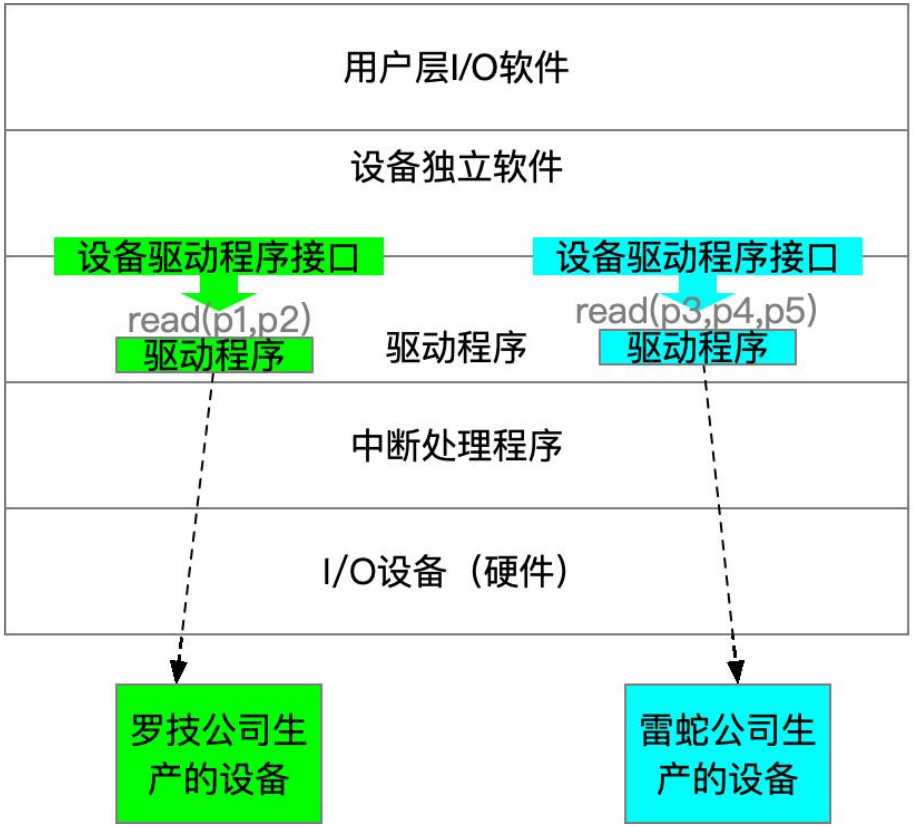
由于read和write是系统调用所以我们需要先从用户态进入到内核态，然后将磁盘当中的数据拷贝到操作系统的缓冲区当中，然后再将缓冲区当中的数据拷贝到用户态当中。在这个过程当中我们进行了两次拷贝

地址映射



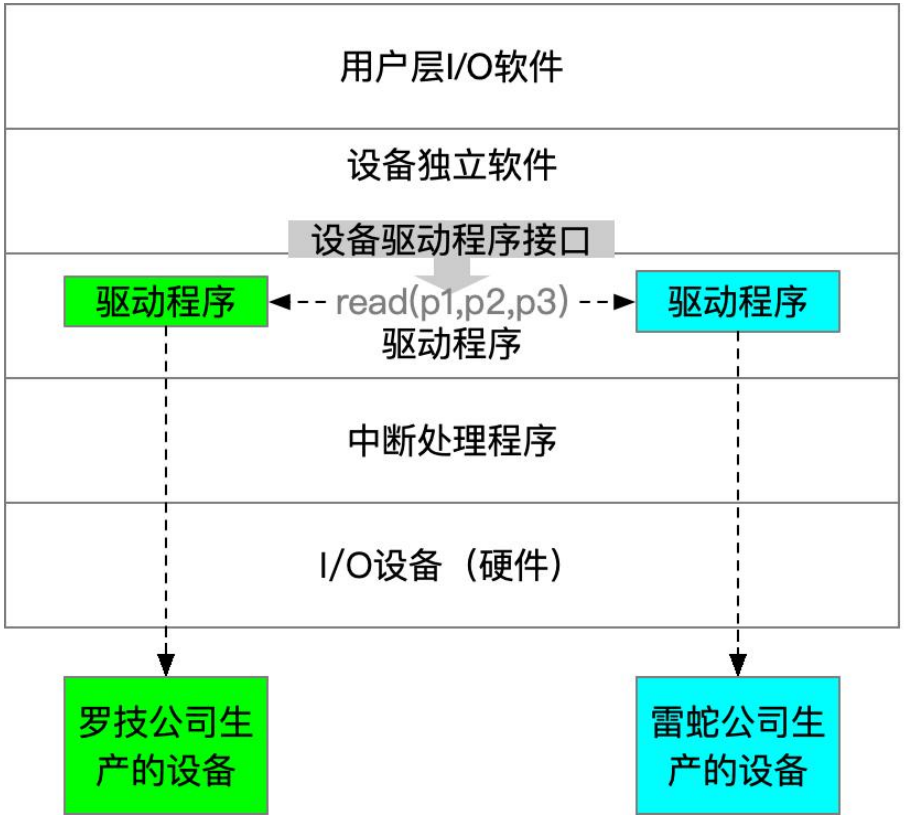
使用mmap（memory map）就可以减少一次拷贝这样带来性能上的提升是巨大的。并且我们采用内存操作比read和write要简单一些，我们不需要在用户层定义缓冲区用来保存从内核缓冲区读上来的数据，从而节约了内存的消耗

设备驱动程序接口



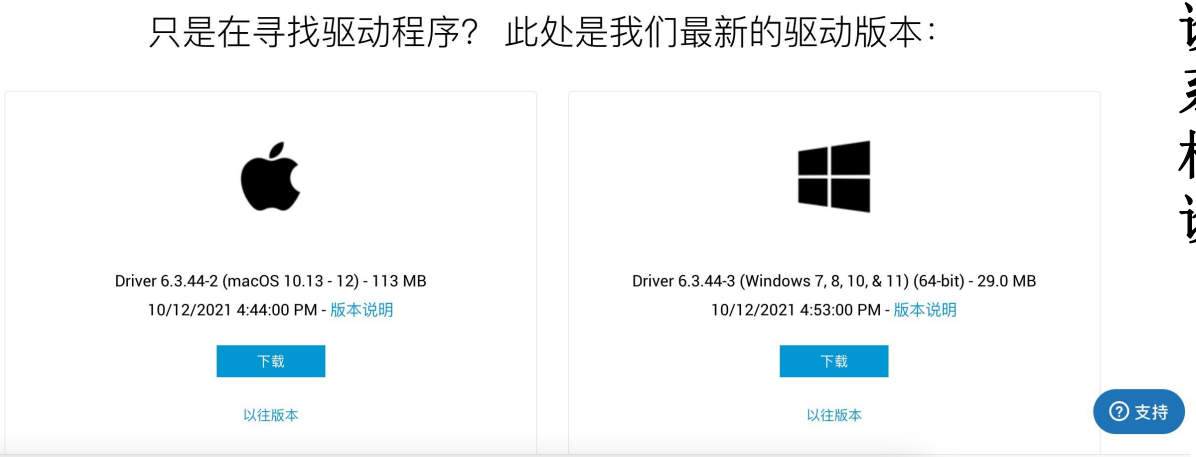
若各公司开发的设备驱动程序接口不统一，则操作系统很难调用设备驱动程序

统一标准的设备驱动程序接口



操作系统规定好设备驱动程序的接口标准，各厂商必须按要求开发设备驱动程序

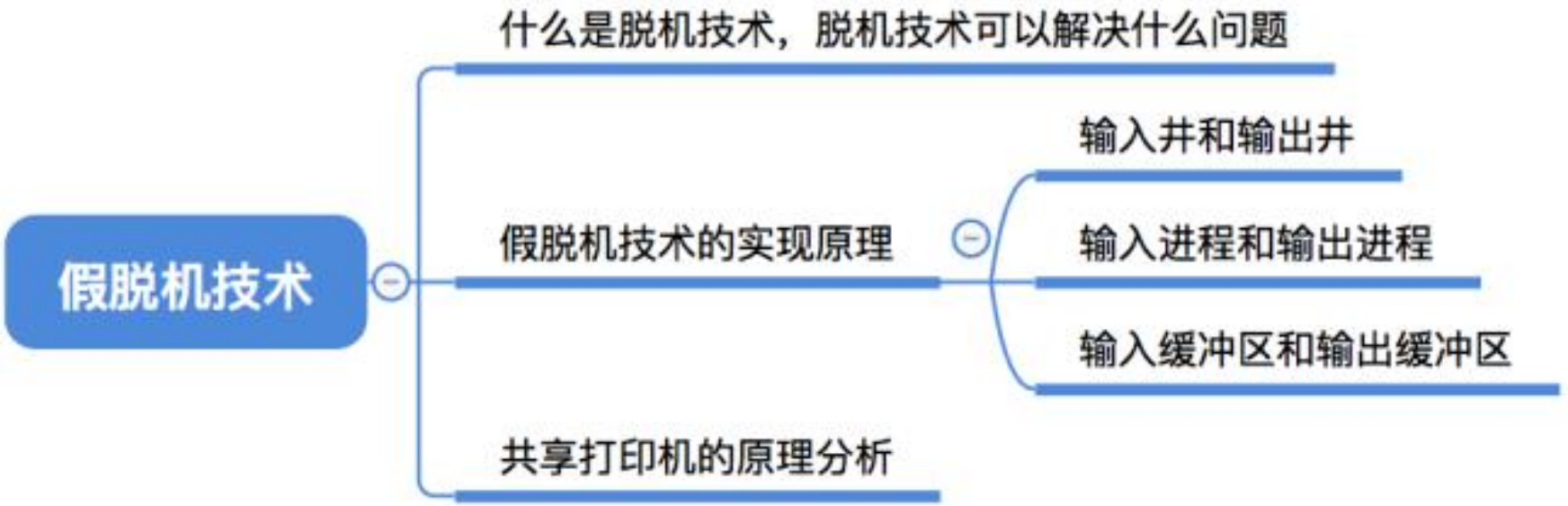
设备驱动程序接口



不同的操作系统，对设备驱动程序接口的标准各不相同。

设备厂商必须根据操作系统的接口要求，开发相应的设备驱动程序，设备才能被使用

假脱机技术（SPOOLing技术）

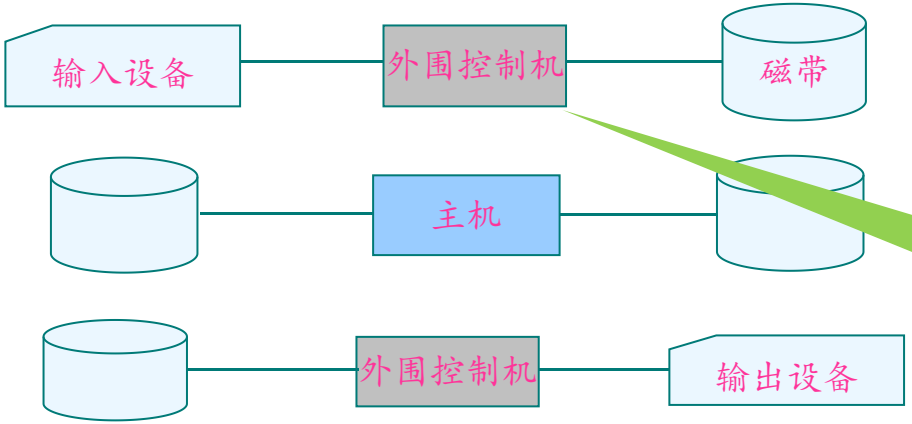


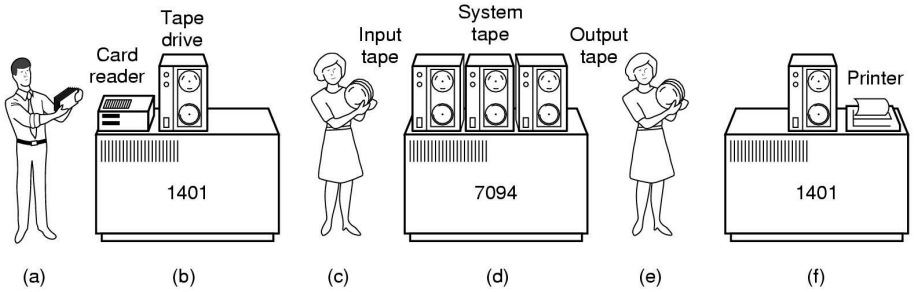
什么是脱机技术

手工操作阶段：主机直接从 I/O设备获得数据，由于设备速度慢，主机速度很快。人机速度矛盾明显，主机要浪费很多时间来等待设备

什么是脱机技术

批处理阶段引入了脱机输入/输出技术（用磁带完成）





引入脱机技术后，缓解了CPU与慢速I/O设备的速度矛盾。另一方面，即使CPU在忙碌，也可以提前将数据输入到磁带；即使慢速的输出设备正在忙碌，也可以提前将数据输出到磁带。

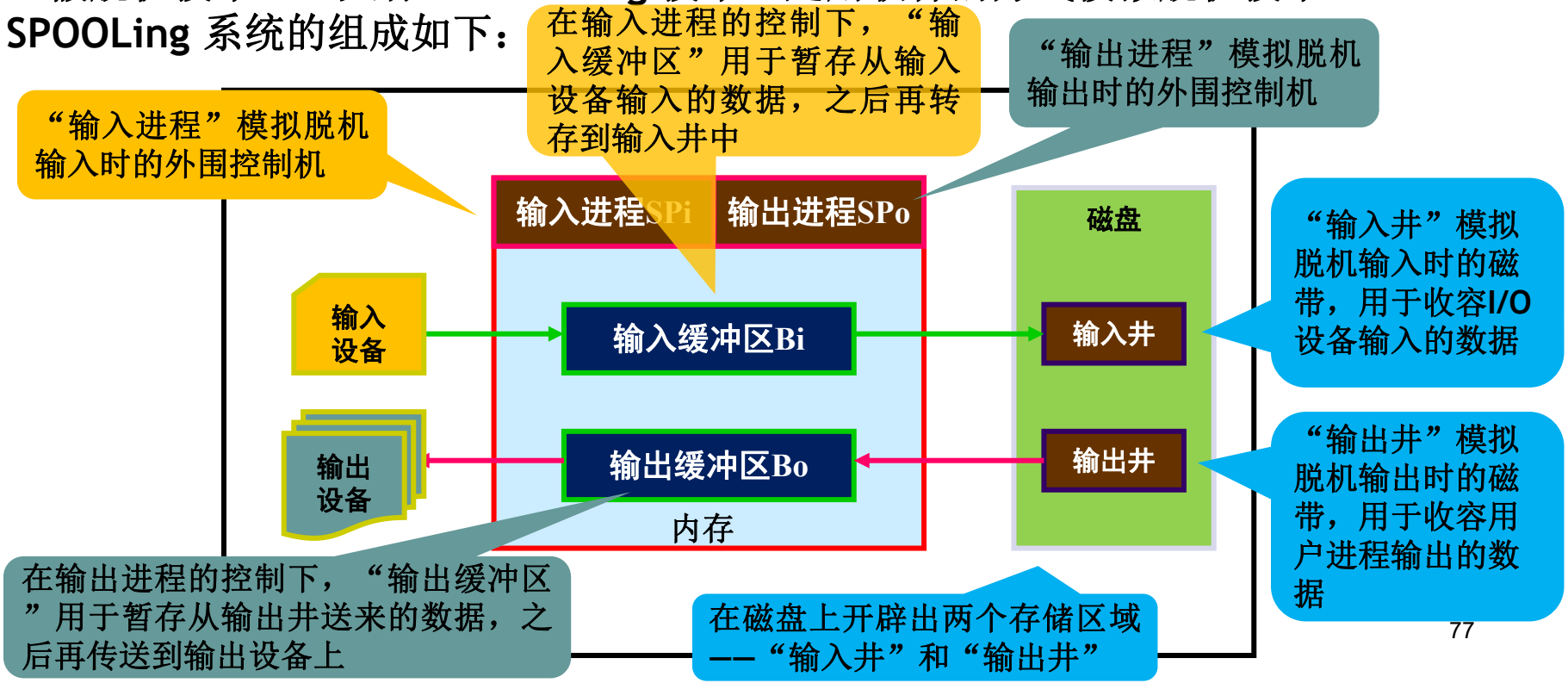
在外围控制机的控制下，慢速输入设备的数据先被输入到更快速的磁带上。之后主机可以从快速的磁带上读入数据，从而缓解了速度矛盾

假脱机技术

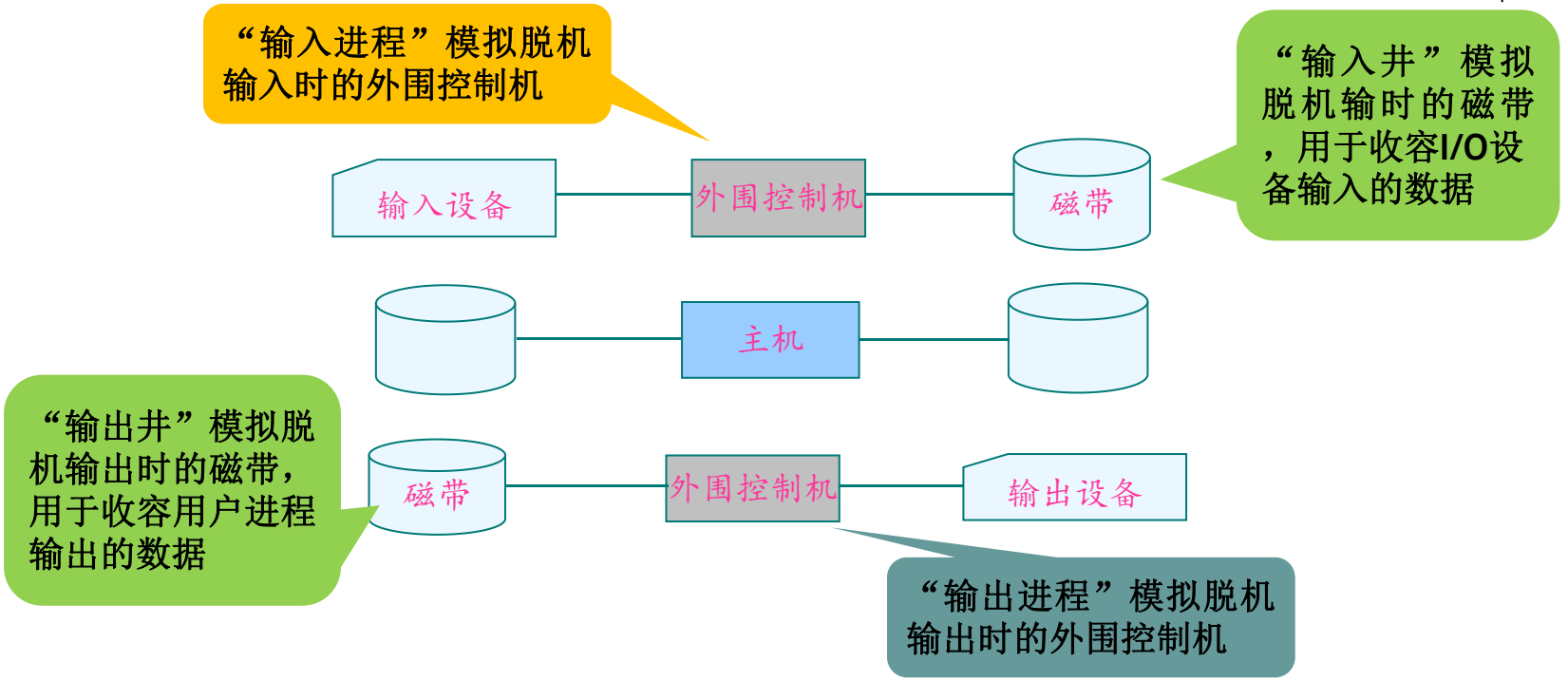
- 定义：在联机的情况下实现的同时外围操作
- 特点：是对脱机输入输出系统的模拟。因此， 必须建立在具有多道程序功能的操作系统上，而且需要高速外存的支持。
- 方式：将数据从输入设备传送到磁盘或反之。通过它可以将一台独占的物理设备虚拟为多台逻辑设备，从而允许多个用户（进程）共享。

假脱机技术——组成

“假脱机技术”，又称“SPOOLing 技术”是用软件的方式模拟脱机技术。SPOOLing 系统的组成如下：



假脱机技术——组成


“输入进程”模拟脱机输入时的外围控制机
“输入井”模拟脱机输时的磁带，用于收容I/O设备输入的数据
输入设备
外围控制机
磁带
主机
“输出井”模拟脱机输出时的磁带，用于收容用户进程输出的数据
磁带
外围控制机
输出设备
“输出进程”模拟脱机输出时的外围控制机

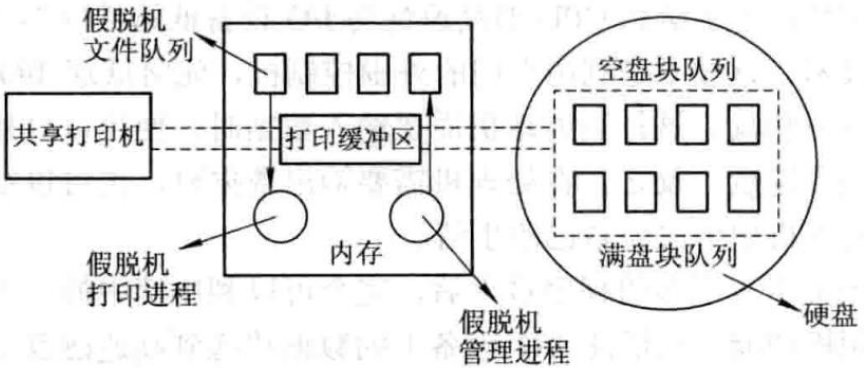
共享打印机原理分析

独占设备——只允许各个进程串行使用的设备。一段时间内只能满足一个进程的请求。
共享设备——允许多个进程“同时”使用的设备（宏观上同时使用，微观上可能是交替使用）。可以同时满足多个进程的使用请求。

打印机是种“独占式设备”，但是可以用SPOOLing 技术改造成“共享设备”

独占式设备的例子：若进程1 正在使用打印机，则进程2 请求使用打印机时必然阻塞等待

共享打印机原理分析



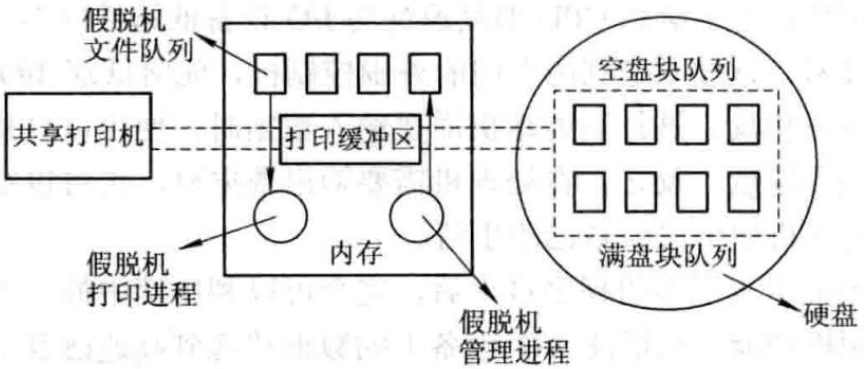
(c) 假脱机打印机系统的组成

当多个用户进程提出输出打印的请求时，系统会答应它们的请求，但是并不是真正把打印机分配给他们，而是由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缓冲区（也就是说，这个缓冲区是在磁盘上的），并将要打印的数据送入其中；

（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（其实就是用来说明用户的打印数据存放位置等信息的），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，并根据表中的要求将要打印的数据从输出井传送到输出缓冲区，再输出到打印机进行打印。用这种方式可依次处理完全部的打印任务

共享打印机原理分析

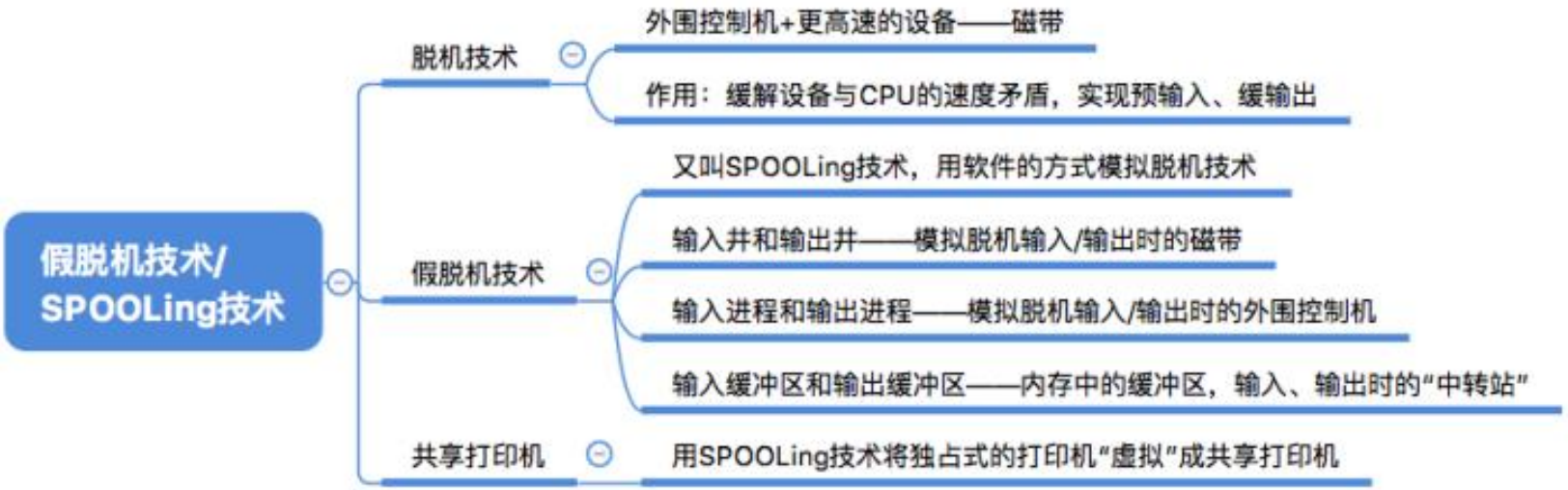


(c) 假脱机打印机系统的组成

虽然系统中只有一个台打印机，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（相当于分配了一个逻辑设备），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。

SPOOLing 技术可以把一台物理设备虚拟成逻辑上的多台设备，可将独占式设备改造成共享设备。

知识总览


假脱机技术/
SPOOLing技术
脱机技术
外围控制机+更高速的设备——磁带
作用：缓解设备与CPU的速度矛盾，实现预输入、缓输出
假脱机技术
又叫SPOOLing技术，用软件的方式模拟脱机技术
输入井和输出井——模拟脱机输入/输出时的磁带
输入进程和输出进程——模拟脱机输入/输出时的外围控制机
输入缓冲区和输出缓冲区——内存中的缓冲区，输入、输出时的“中转站”
共享打印机
用SPOOLing技术将独占式的打印机“虚拟”成共享打印机

设备管理

- 5.1 I/O系统
- 5.2 I/O控制方式
- 5.3 缓冲管理
- 5.4 设备分配
- 5.5 设备处理
- 5.6 磁盘存储器管理

与设备无关的I/O软件

- 设备独立性（设备无关性）的含义：
  - 应用程序中所用的设备，不局限于使用某个具体的物理设备。

- 与设备无关的I/O软件（设备独立性软件）含义：
  - 在设备驱动程序之上设置一层软件，以实现设备独立性。

与设备无关软件的基本概念

- 1、以物理设备名使用设备
  - 应用程序中所用的设备与系统中的物理设备直接相关。
  - 问题：不灵活，给用户带来不便。
- 2、引入了逻辑设备名
  - 逻辑设备是抽象的设备名，如/dev/printer，并不指定哪一台打印机。好处：系统设备分配灵活。
  - 可实现I/O重定向。即：用于IO操作的设备可更换，而不必改变应用程序。如修改逻辑设备表：终端->打印机

与设备无关软件的基本概念

- 3、逻辑设备名到物理设备名的映射
  - 逻辑设备表
  - 为了实现逻辑设备名到物理设备名的映射，系统必须设置一张逻辑设备表LUT（Logical Unit Table），能够将应用程序中所使用的逻辑设备名映射为物理设备名，并提供该设备驱动程序的入口地址。

逻辑设备名	物理设备名	驱动程序入口地址
/dev/tty	3	1024
/dev/printer	5	2046
…	…	…

设备的分配与回收

设备的固有属性可分为三种：独占设备、共享设备、虚拟设备。

独占设备——一个时段只能分配给一个进程（如打印机）

共享设备——可同时分配给多个进程使用（如磁盘），各进程往往是宏观上同时共享使用设备，而微观上交替使用。

虚拟设备——采用 SPOOLing 技术将独占设备改造成虚拟的共享设备，可同时分配给多个进程使用（如采用 SPOOLing 技术实现的共享打印机）

设备分配时应考虑的因素

设备的分配算法：

先来先服务
优先级高者优先
短任务优先
……

设备分配时应考虑的因素

从进程运行的安全性上考虑，设备分配有两种方式：

安全分配方式：为进程分配一个设备后就将进程阻塞，本次I/O完成后才将进程唤醒。（eg：考虑进程请求打印机打印输出的例子）

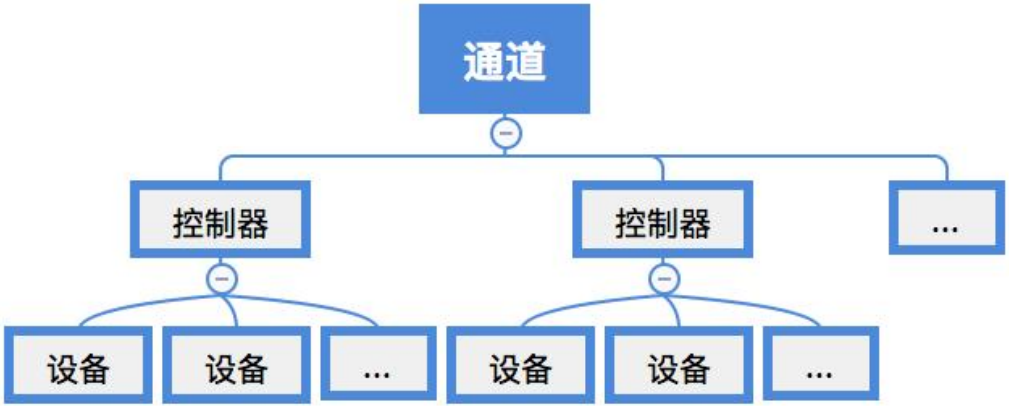
一个时段内每个进程只能使用一个设备
优点：破坏了“请求和保持”条件，不会死锁
缺点：对于一个进程来说，CPU和I/O设备只能串行工作

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。

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，使进程迅速推进
缺点：有可能发生死锁（死锁避免、死锁的检测和解除）

独占设备的分配程序

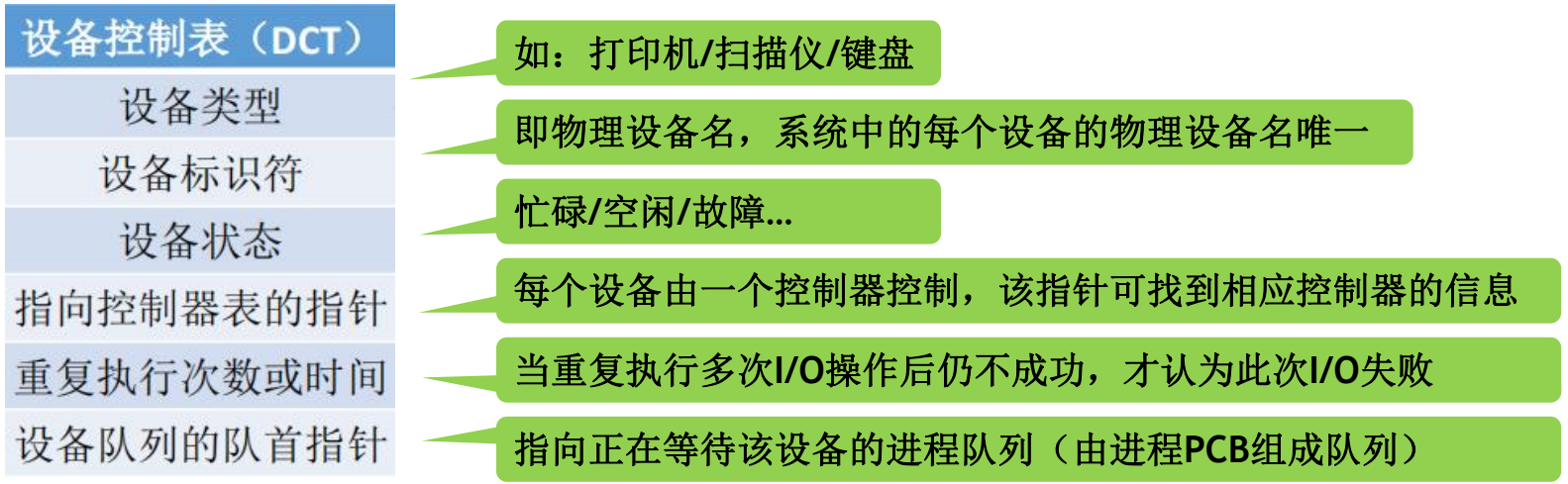
- “设备、控制器、通道”之间的关系：



一个通道可控制多个设备控制器，每个设备控制器可控制多个设备。

设备分配管理中的数据结构

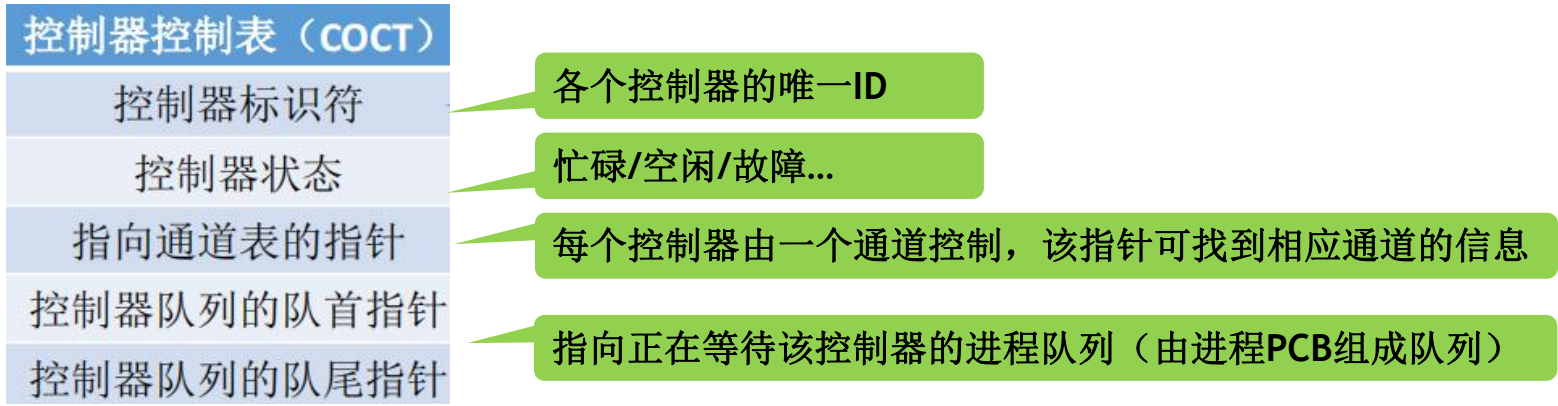
- 设备控制表（DCT）：系统为每个设备配置一张DCT，用于记录设备情况



注：“进程管理”章节中曾经提到过“系统会根据阻塞原因不同，将进程PCB挂到不同的阻塞队列中”

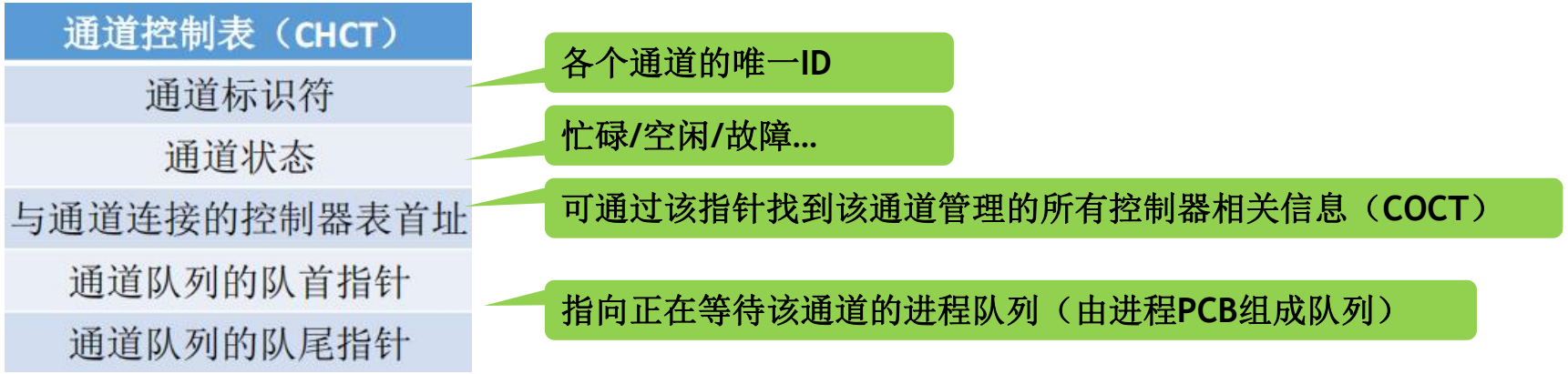
设备分配管理中的数据结构

控制器控制表（COCT）：每个设备控制器都会对应一张COCT。操作系统根据COCT的信息对控制器进行操作和管理。



设备分配管理中的数据结构

通道控制表（CHCT）：每个通道都会对应一张CHCT。操作系统根据CHCT的信息对通道进行操作和管理。



设备分配管理中的数据结构

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。

设备分配的步骤

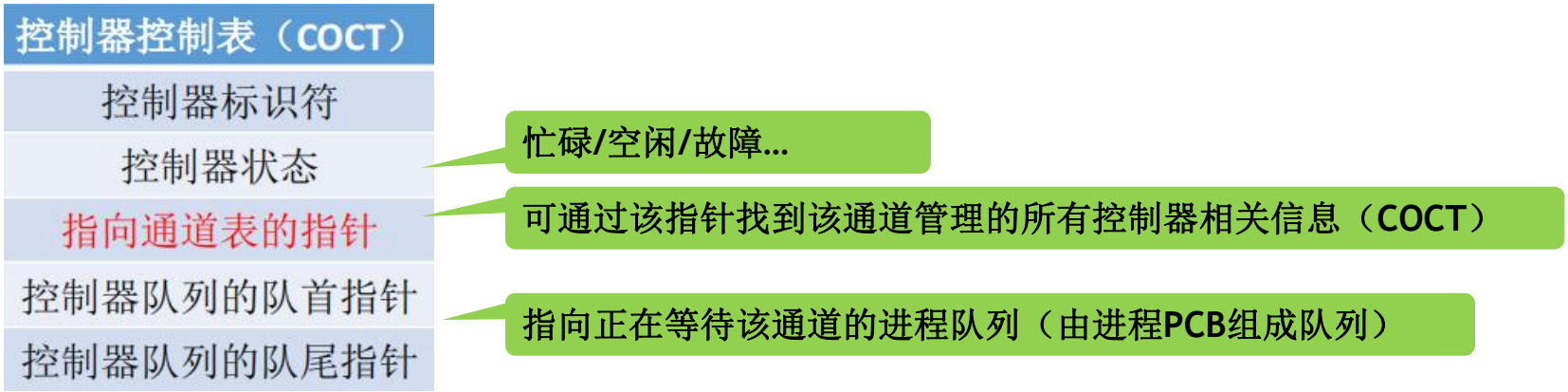
①根据进程请求的物理设备名查找SDT（注：物理设备名是进程请求分配设备时提供的参数）

设备分配的步骤

①根据进程请求的物理设备名查找SDT（注：物理设备名是进程请求分配设备时提供的参数）
②根据SDT找到DCT，若设备忙碌则将进程PCB挂到设备等待队列中，不忙碌则将设备分配给进程。

设备分配的步骤

①根据进程请求的物理设备名查找SDT（注：物理设备名是进程请求分配设备时提供的参数）
②根据SDT找到DCT，若设备忙碌则将进程PCB挂到设备等待队列中，不忙碌则将设备分配给进程。
③根据DCT找到COCT，若控制器忙碌则将进程PCB挂到控制器等待队列中，不忙碌则将控制器分配给进程。



设备分配的步骤

①根据进程请求的物理设备名查找SDT（注：物理设备名是进程请求分配设备时提供的参数）
②根据SDT找到DCT，若设备忙碌则将进程PCB挂到设备等待队列中，不忙碌则将设备分配给进程
③根据DCT找到COCT，若控制器忙碌则将进程PCB挂到控制器等待队列中，不忙碌则将控制器分配给进程。
④根据COCT找到CHCT，若通道忙碌则将进程PCB挂到通道等待队列中，不忙碌则将通道分配给进程。

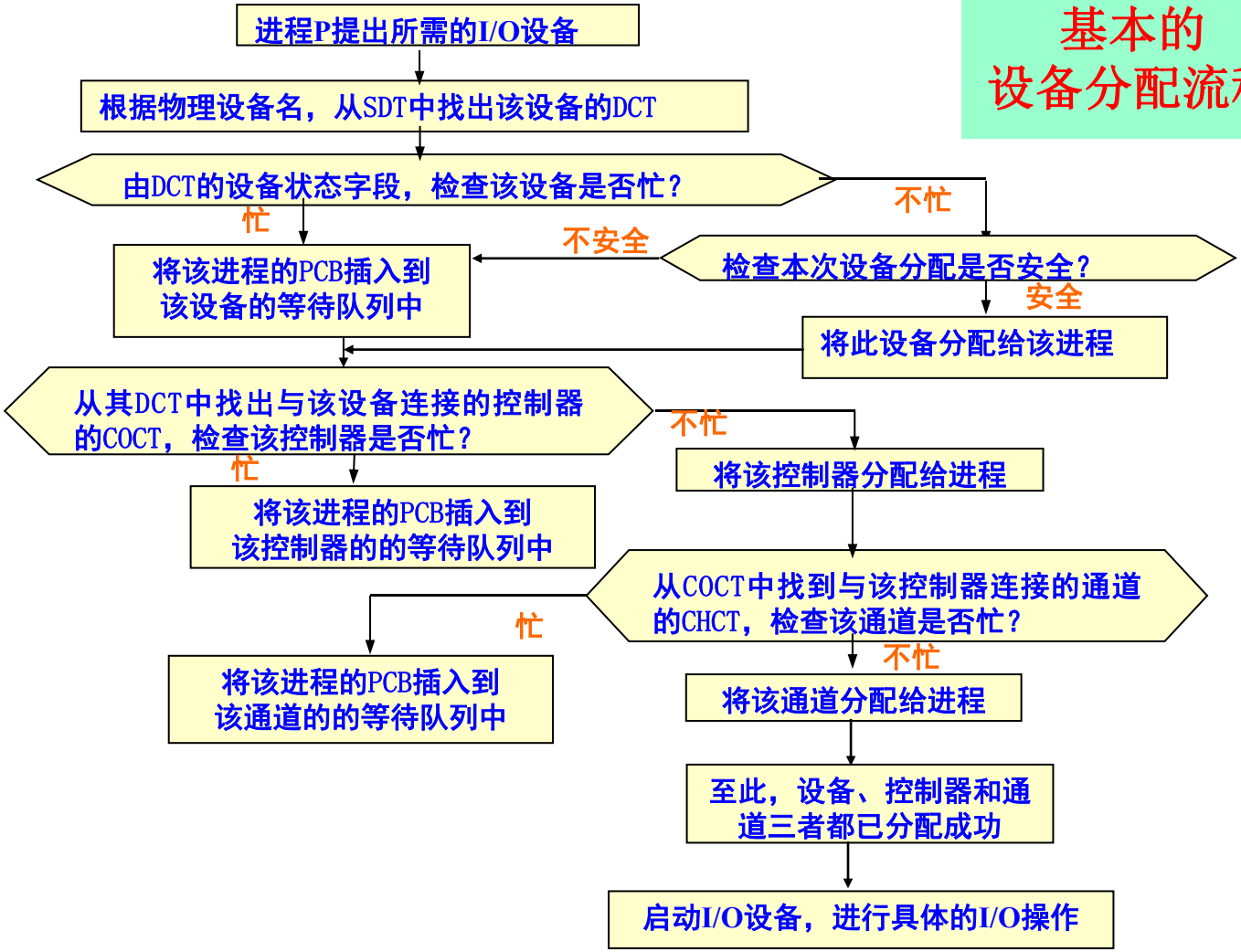
通道控制表（CHCT）
通道标识符
通道状态
与通道连接的控制器表首址
通道队列的队首指针
通道队列的队尾指针

忙碌/空闲/故障...

指向正在等待该通道的进程队列（由进程PCB组成队列）

注：只有设备、控制器、通道三者都分配成功时，这次设备分配才算成功，之后便可启动I/O设备进行数据传送

基本的设备分配流程


进程P提出所需的I/O设备
根据物理设备名，从SDT中找出该设备的DCT
由DCT的设备状态字段，检查该设备是否忙？
忙
不忙
将该进程的PCB插入到该设备的等待队列中
不安全
检查本次设备分配是否安全？
安全
将此设备分配给该进程
从其DCT中找出与该设备连接的控制器的COCT，检查该控制器是否忙？
忙
不忙
将该进程的PCB插入到该控制器的的等待队列中
将该控制器分配给进程
从COCT中找到与该控制器连接的通道的CHCT，检查该通道是否忙？
忙
不忙
将该进程的PCB插入到该通道的的等待队列中
将该通道分配给进程
至此，设备、控制器和通道三者都已分配成功
启动I/O设备，进行具体的I/O操作

设备分配步骤的改进

①根据进程请求的物理设备名查找SDT（注：物理设备名是进程请求分配设备时提供的参数）
②根据SDT找到DCT，若设备忙碌则将进程PCB挂到设备等待队列中，不忙碌则将设备分配给进程
③根据DCT找到COCT，若控制器忙碌则将进程PCB挂到控制器等待队列中，不忙碌则将控制器分配给进程。
④根据COCT找到CHCT，若通道忙碌则将进程PCB挂到通道等待队列中，不忙碌则将通道分配给进程。

缺点：
①用户编程时必须使用“物理设备名”，底层细节对用户不透 明，不方便编程 ②若换了一个物理设备，则程序无法运行
③若进程请求的物理设备正在忙碌，则即使系统中还有同类型的设备，进程也必须阻塞等待

改进方法：建立逻辑设备名与物理设备名的映射机制，用户编程时只需提供逻辑设备名

设备分配步骤的改进

①根据进程请求的逻辑设备名查找SDT（用户编程时提供的逻辑设备名其实就是“设备类型”）
②查找SDT，找到用户进程指定类型的、并且空闲的设备，将其分配给该进程。操作系统在逻辑设备表（LUT）中新增一个表项。
③根据DCT找到COCT，若控制器忙碌则将进程PCB挂到控制器等待队列中，不忙碌则将控制器分配给进程。
④根据COCT找到CHCT，若通道忙碌则将进程PCB挂到通道等待队列中，不忙碌则将通道分配给进程。

设备分配步骤的改进

逻辑设备表（LUT）

逻辑设备名	物理设备名	驱动程序入口地址
/dev/打印机	3	1024
/dev/扫描仪	5	2046
…	…	…

逻辑设备表（LUT）建立了逻辑设备名与物理设备名之间的映射关系。

某用户进程第一次使用设备时使用逻辑设备名向操作系统发出请求，操作系统根据用户进程指定的设备类型（逻辑设备名）查找系统设备表，找到一个空闲设备分配给进程，并在LUT中增加相应表项。

如果之后用户进程再次通过相同的逻辑设备名请求使用设备，则操作系统通过LUT表即可知道用户进程实际要使用的是哪个物理设备了，并且也能知道该设备的驱动程序入口地址。

逻辑设备表的设置问题：

整个系统只有一张LUT：各用户所用的逻辑设备名不允许重复，适用于单用户操作系统

每个用户一张LUT：不同用户的逻辑设备名可重复，适用于多用户操作系统

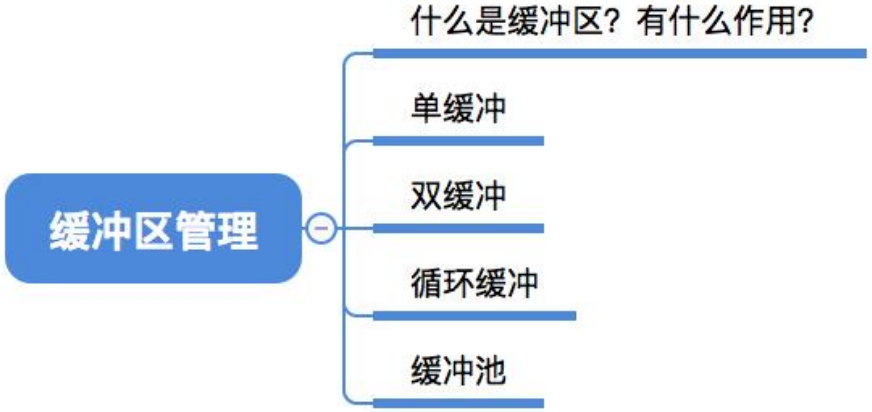
知识回顾


设备的分配与回收
应考虑的因素
固有属性
独占设备、共享设备、虚拟设备（SPOOLing）
分配算法
先来先服务、优先级高者优先、短任务优先等
安全性
安全分配方式、不安全分配方式
静态分配与动态分配
静态分配：进程运行前为其分配全部所需资源，运行结束后归还资源
动态分配：进程运行过程中动态申请设备资源
设备分配管理中的数据结构
设备控制表（DCT）
每个设备对应一张DCT，关键字段：类型/标识符/状态/指向COCT的指针/等待队列指针
控制器控制表（COCT）
每个控制器对应一张COCT，关键字段：状态/指向CHCT的指针/等待队列指针
通道控制表（CHCT）
每个控制器对应一张CHCT，关键字段：状态/等待队列指针
系统设备表（SDT）
记录整个系统中所有设备的情况，每个设备对应一个表目，关键字段：设备类型/标识符/DCT/驱动程序入口
设备分配的步骤
①根据进程请求的物理设备名查找SDT；②根据SDT找到DCT并分配设备；③根据DCT找到COCT并分配控制器；④根据COCT找到CHCT并分配通道
注：只有设备、控制器、通道三者都分配成功时，这次设备分配才算成功，之后便可启动I/O设备进行数据传送
缺点：用户编程时必须使用“物理设备名”，若换了一个物理设备，则程序无法运行。若进程请求的物理设备正在忙碌，则即使系统中还有同类型的设备，进程也必须阻塞等待
设备分配步骤的改进
用户编程时使用逻辑设备名申请设备，操作系统负责实现从逻辑设备名到物理设备名的映射（通过LUT）
逻辑设备表的设置问题
整个系统只有一张LUT：各用户所用的逻辑设备名不允许重复
每个用户一张LUT：各个用户的逻辑设备名可重复

设备管理

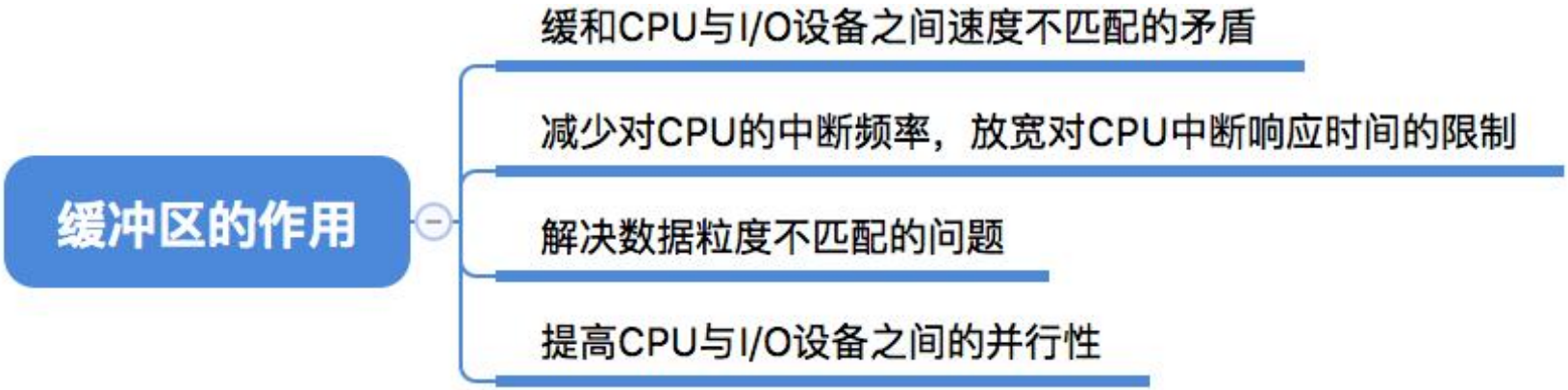
- 5.1 I/O系统
- 5.2 I/O控制方式
- 5.3 缓冲管理
- 5.4 设备分配
- 5.5 设备处理
- 5.6 磁盘存储器管理

缓冲区管理



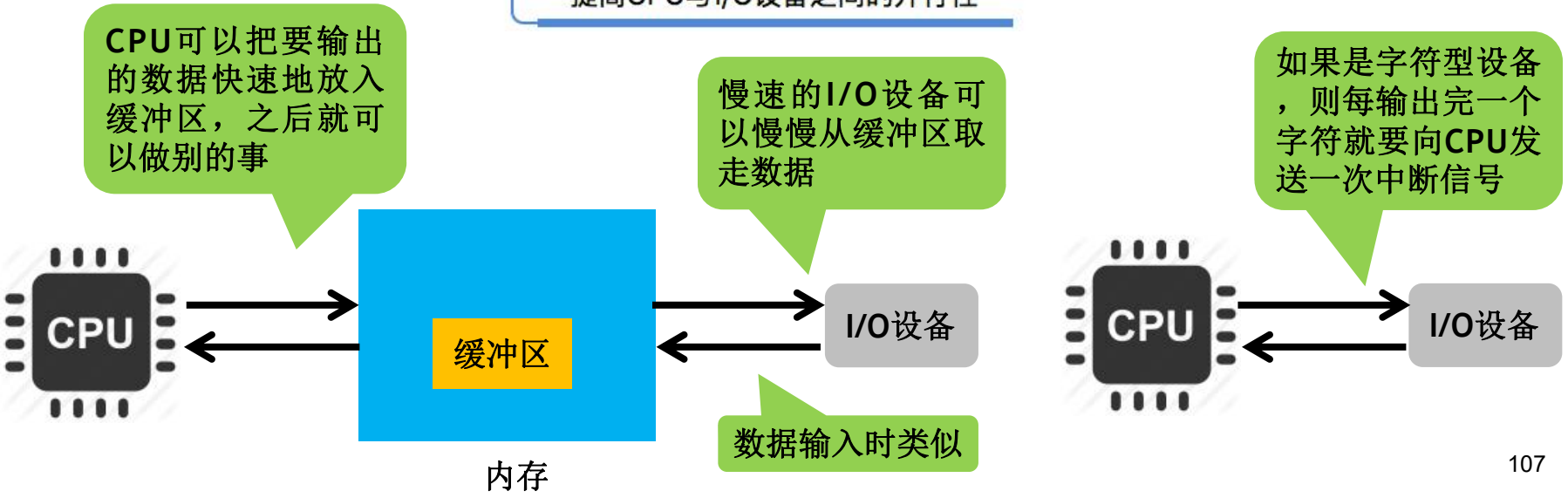
什么是缓冲区？有什么作用？

缓冲区是一个存储区域，可以由专门的硬件寄存器组成，也可利用内存作为缓冲区。
使用硬件作为缓冲区的成本较高，容量也较小，一般仅用在对速度要求非常高的场合（如存储器管理中所用的联想寄存器，由于对页表的访问频率极高，因此使用速度很快的联想寄存器来存放页表项的副本）
一般情况下，更多的是利用内存作为缓冲区，“设备独立性软件”的缓冲区管理就是要组织管理好这些缓冲区



缓冲区有什么作用？

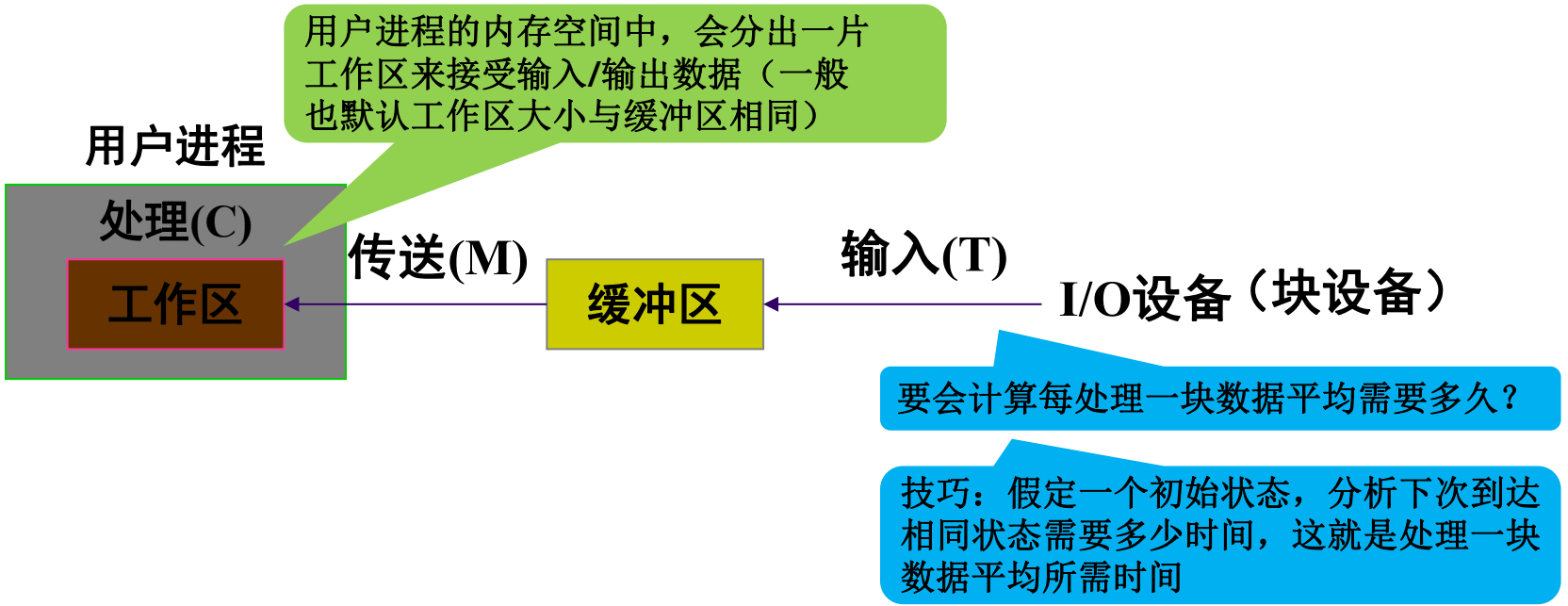
缓冲区的作用
- 缓和CPU与I/O设备之间速度不匹配的矛盾
- 减少对CPU的中断频率，放宽对CPU中断响应时间的限制
- 解决数据粒度不匹配的问题
- 提高CPU与I/O设备之间的并行性



单缓冲

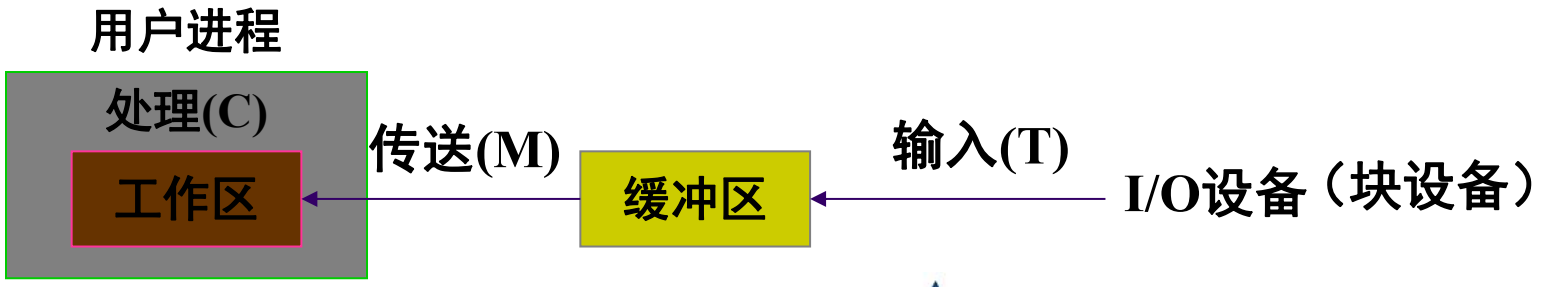
假设某用户进程请求某种块设备读入若干块的数据。若采用单缓冲的策略，操作系统会在主存中为其分配一个缓冲区（若题目中没有特别说明，一个缓冲区的大小就是一个块）。

注意：当缓冲区数据非空时，不能往缓冲区冲入数据，只能从缓冲区把数据传出；当缓冲区为空时，可以往缓冲区传入数据，但必须把缓冲区充满以后，才能从缓冲区把数据传出。

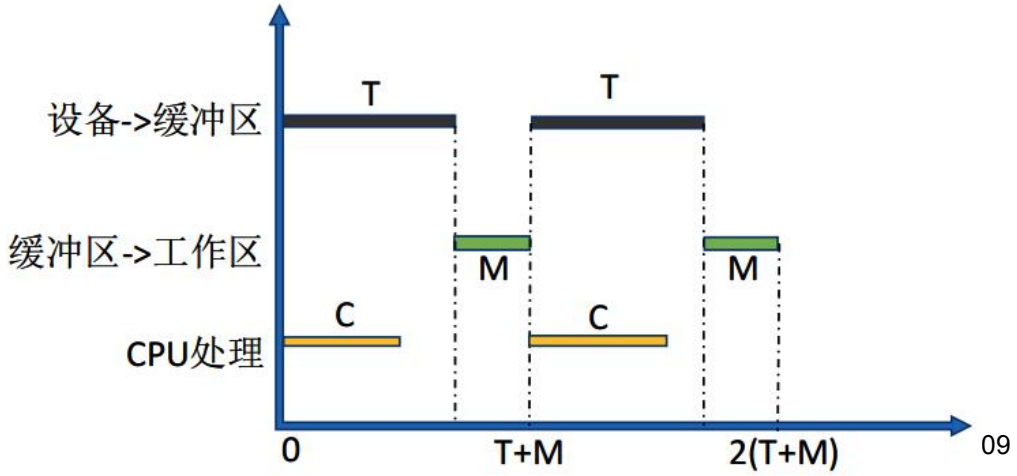


单缓冲

初始状态：工作区满，缓冲区空，假设T>C，处理一块数据的平均用时= T+M

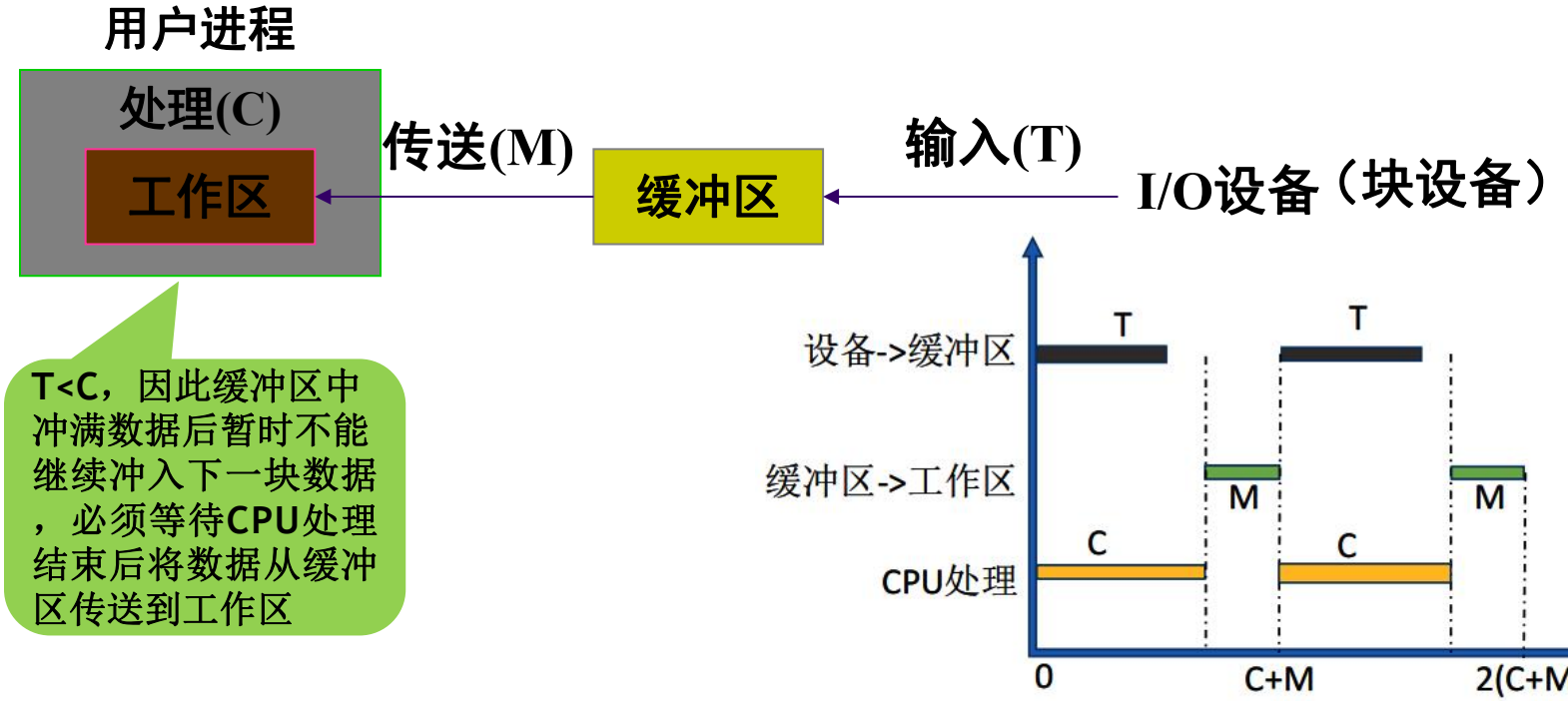


T>C，因此CPU处理完数据后暂时不能将下一块数据传送到工作区，必须等待缓冲区中冲满数据



单缓冲

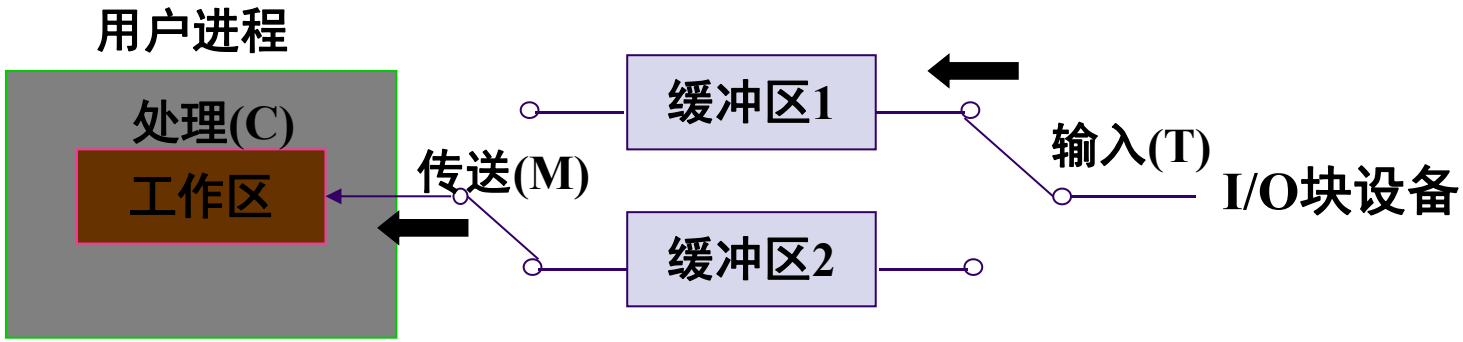
初始状态：工作区满，缓冲区空，假设T<C，处理一块数据的平均用时= C+M



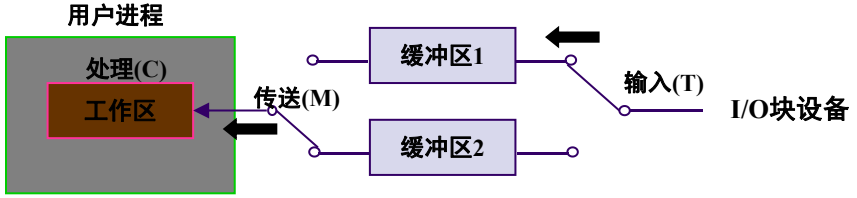
结论：采用单缓冲策略，处理一块数据平均耗时 Max(C, T)+M

双缓冲

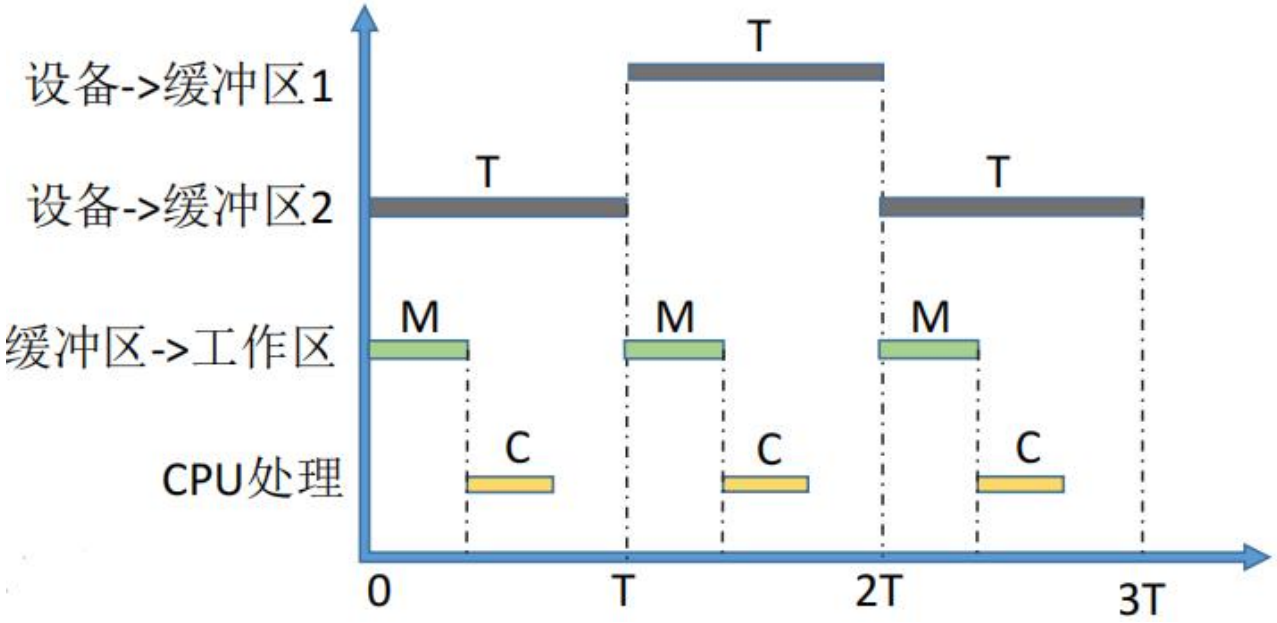
假设某用户进程请求某种块设备读入若干块的数据。若采用双缓冲的策略，操作系统会在主存中为其分配两个缓冲区（若题目中没有特别说明，一个缓冲区的大小就是一个块）



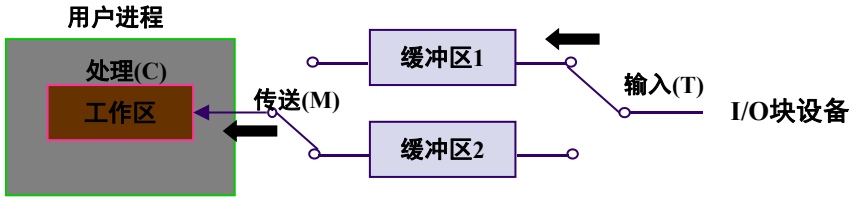
双缓冲



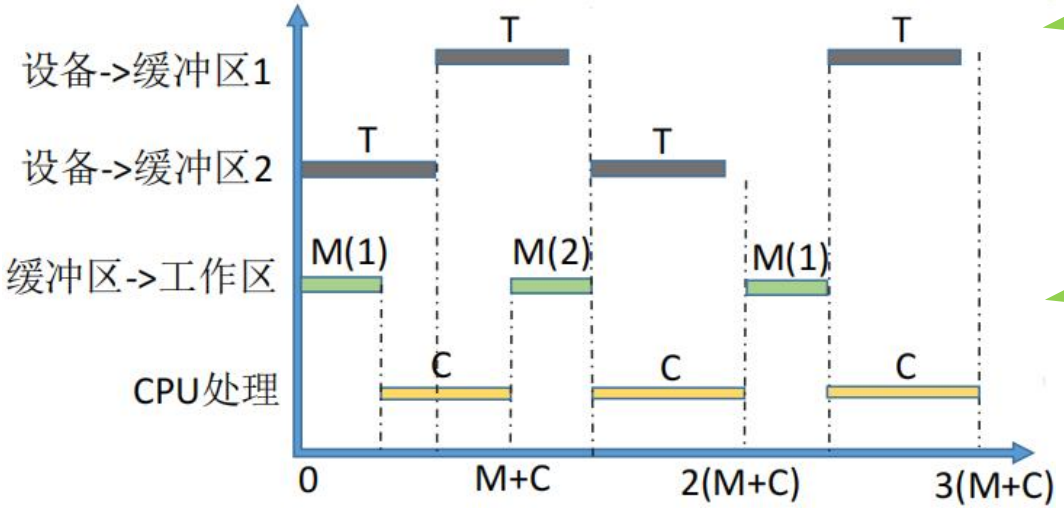
双缓冲题目中，假设初始状态为：工作区空，其中一个缓冲区满，另一个缓冲区空,假设T>C+M,处理一块数据的平均用时= T



双缓冲



双缓冲题目中，假设初始状态为：工作区空，其中一个缓冲区满，另一个缓冲区空,假设T<C+M,处理一块数据的平均用时= C+M



假设2T<2M+C，则I/O设备将缓冲区1冲满时，缓冲区2的数据尚未取空，因此I/O设备暂时不能冲入数据

总之，T<C+M 意味着设备输入数据块的速度要比处理机处理数据块的速度更快。每处理一个数据块平均耗时C+M

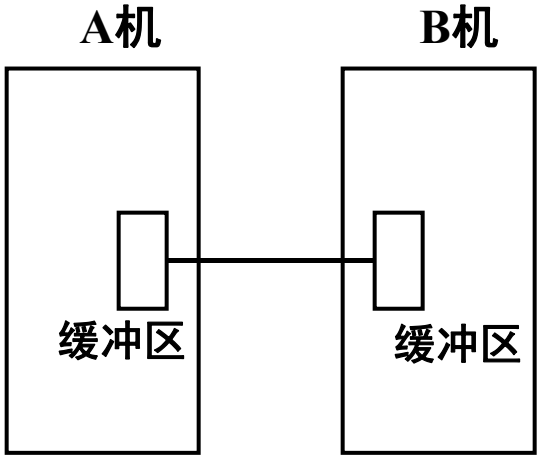
M(1) 表示“将缓冲区1中的数据传送到工作区”； M(2) 表示“将缓冲区2中的数据传送到工作区”

结论：采用双缓冲策略，处理一个数据块的平均耗时为 Max (T, C+M)

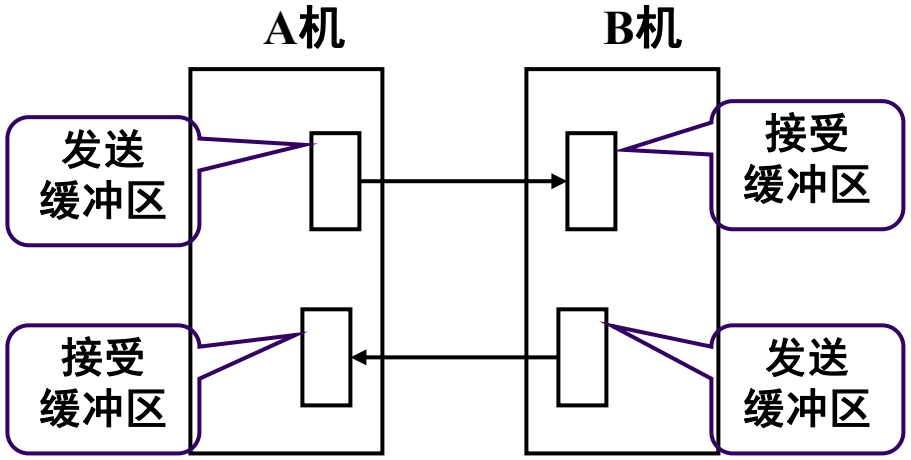
使用单/双缓冲在通信时的区别

如果在实现两台机器通信时，只配置单缓冲，那么在任一时刻只能实现单方向的数据传输。

为了实现双向数据传输，必须在两台机器中都设置两个缓冲区。分别用在发送和接收



(a)单缓冲



(b)双缓冲

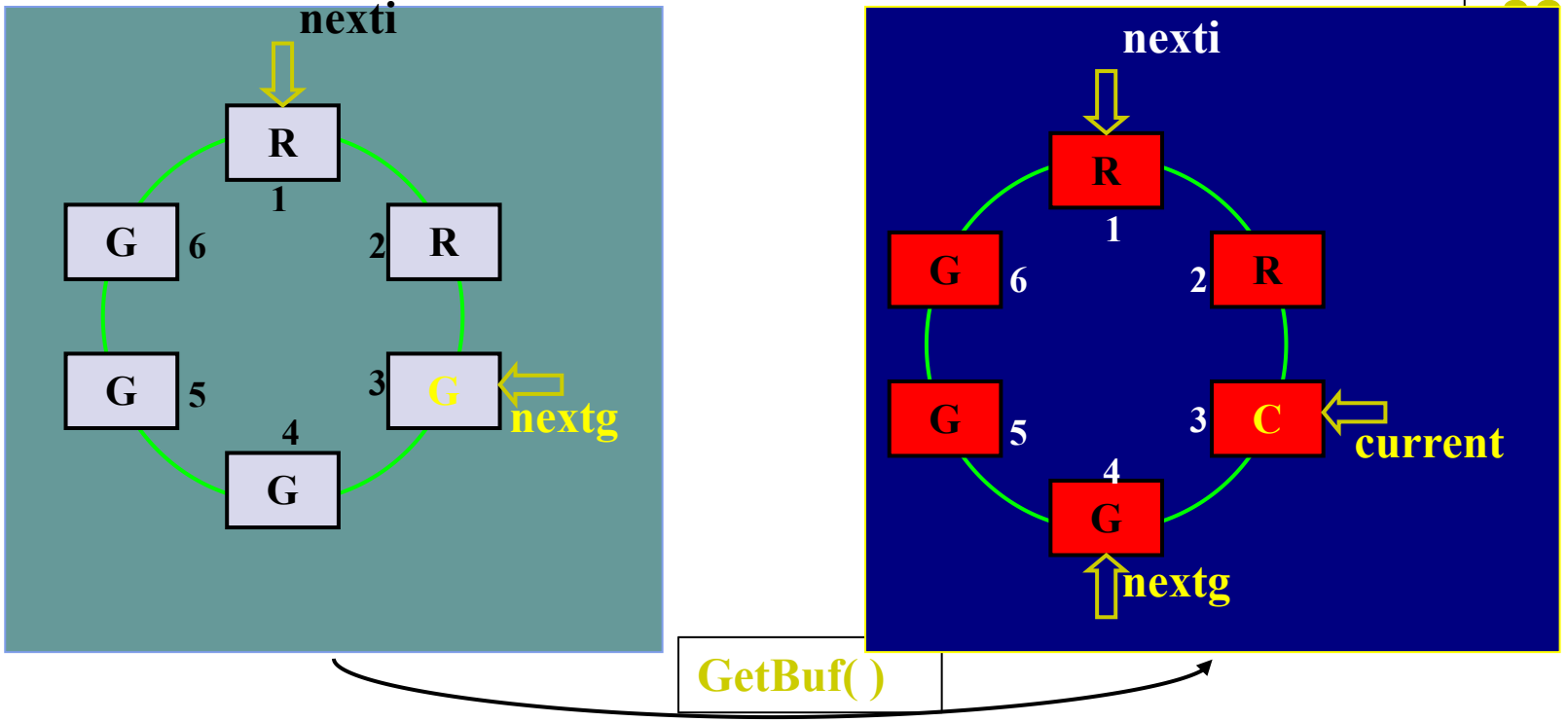
循环缓冲区

- 循环缓冲的引入
  - 当输入和输出的速度相差很大时，双缓冲效果不理想，但可增加缓冲区的数量，改善情况
- 循环缓冲是把多个缓冲区连接起来组成两部分，一部分专门用于输入，另一部分专门用于输出的缓冲结构
  - 例如对于用做输入的循环缓冲区，通常提供给输入进程和计算进程使用，输入进程不断向空缓冲区中输入数据，计算进程则从满缓冲区中提取数据用于计算

循环缓冲区

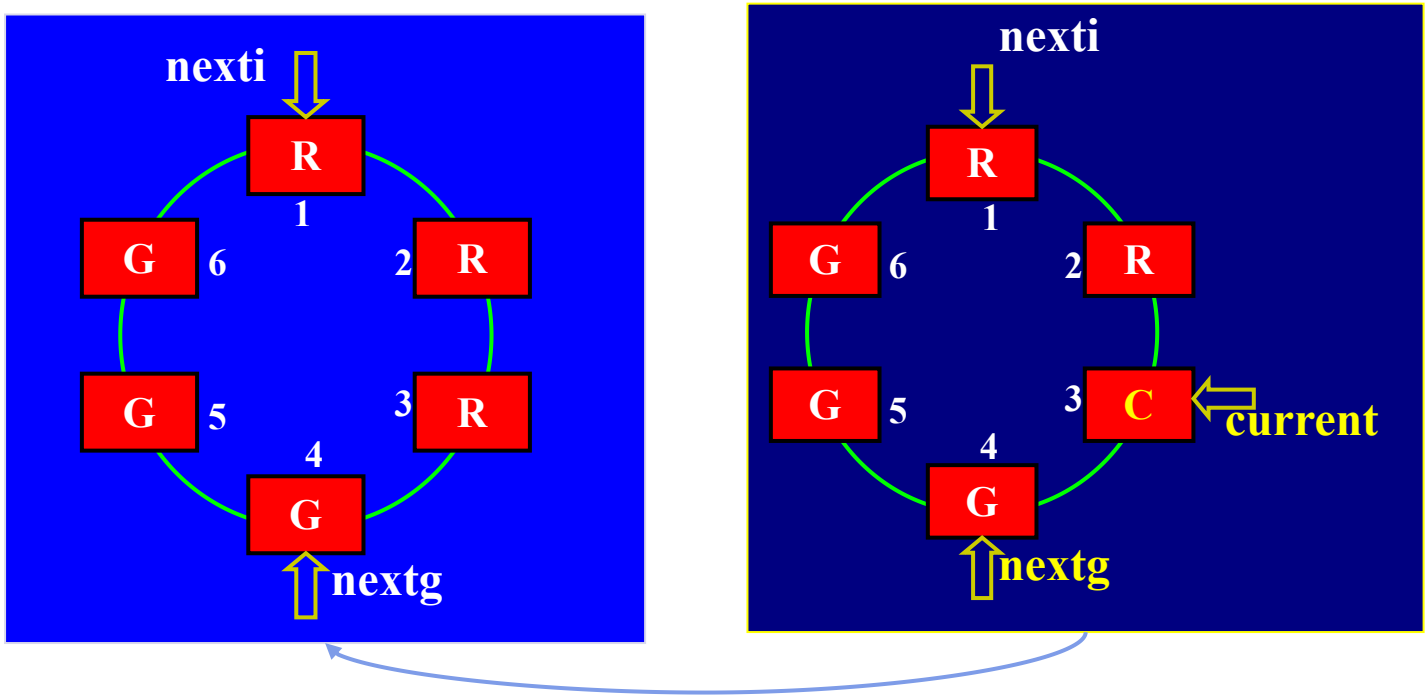
- 循环缓冲的组成
  - 多个缓冲区、多个指针
  - 每个缓冲区大小相同。
  - 用做输入的缓冲区的类型有：
  - 空缓冲区R、已装满数据的缓冲区G、工作缓冲区C
  - 指针类型：
    - 指示计算进程的下一个可用缓冲区G的指针nextg
    - 指示输入进程下次可用的空缓冲区R的指针nexti
    - 指示计算进程正在使用的缓冲区C的指针current

循环缓冲区



计算进程和输入进程可利用GetBuf( )和ReleaseBuf( )两个过程使用循环缓冲区

循环缓冲区


nexti
R
1
G
6
2
R
G
5
3
R
4
G
nextg
nexti
R
1
G
6
2
R
G
5
3
C
current
4
G
nextg
ReleaseBuf( )

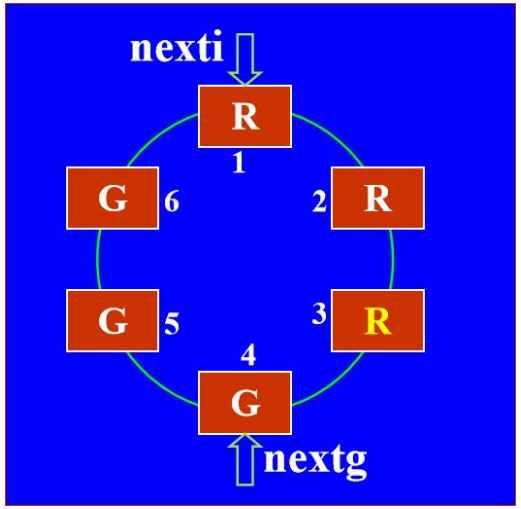
循环缓冲区

- 进程的同步
  - Nexti指针追上Nextg指针

意味着输入进程输入数据的速度大于计算进程处理数据的速度，再无空缓冲区可用。输入进程应阻塞

  - Nextg指针追上Nexti指针

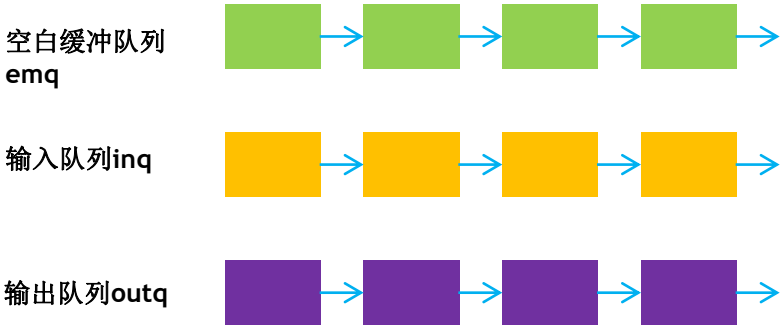
意味着输入数据的速度低于计算进程处理数据的速度，再无装有数据的缓冲区可用。计算进程应阻塞

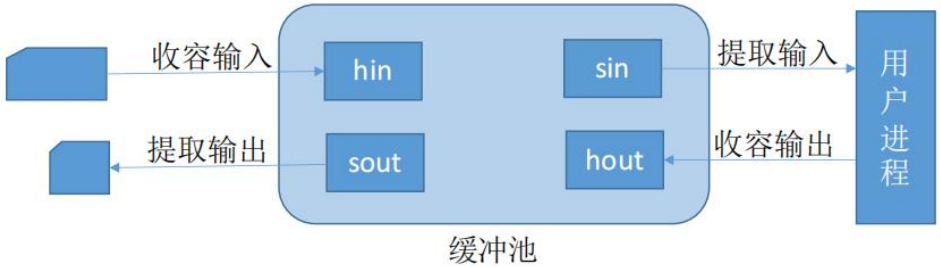


缓冲池

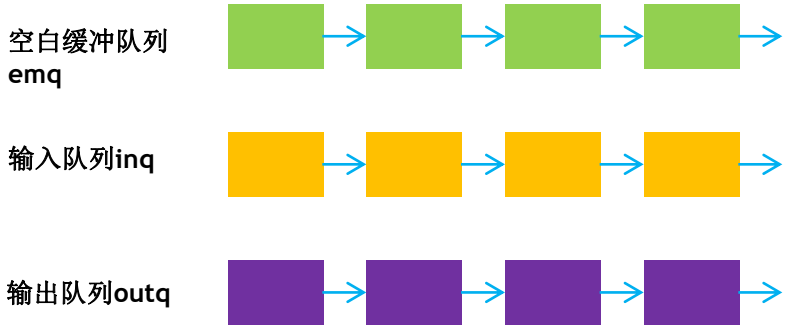
缓冲池由系统中共用的缓冲区组成。这些缓冲区按使用状况可以分为：空缓冲队列、装满输入数据的缓冲队列（输入队列）、装满输出数据的缓冲队列（输出队列）。

另外，根据一个缓冲区在实际运算中扮演的功能不同，又设置了四种工作缓冲区：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）

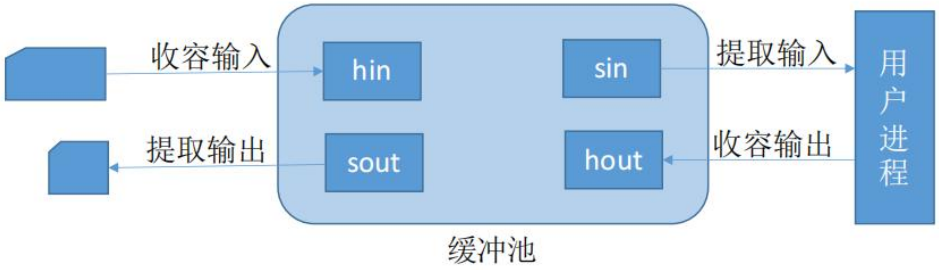




缓冲池

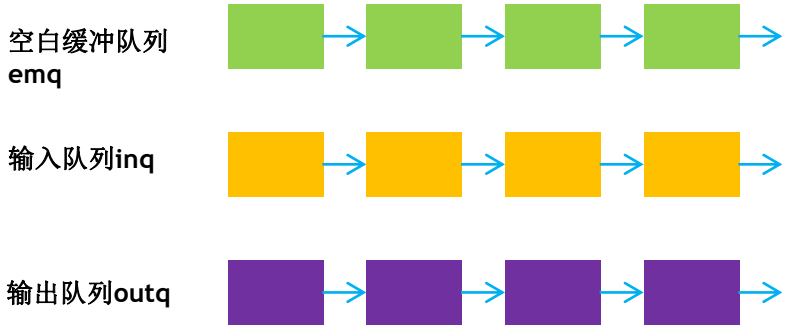


①输入进程请求输入数据

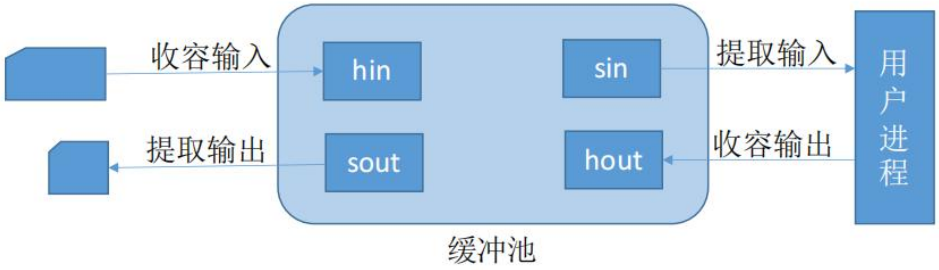


从空缓冲队列中取出一块作为收容输入数据的工作缓冲区（hin）。冲满数据后将缓冲区挂到输入队列队尾

缓冲池

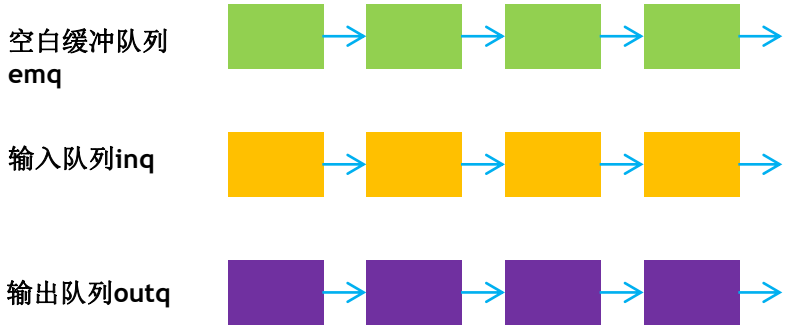


①输入进程请求输入数据
②计算进程想要取得一块输入数据

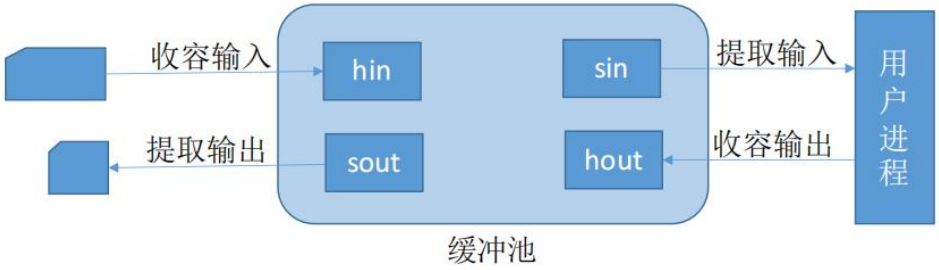


从输入队列中取得一块冲满输入数据的缓冲区作为“提取输入数据的工作缓冲区（sin）”。缓冲区读空后挂到空缓冲区队列

缓冲池

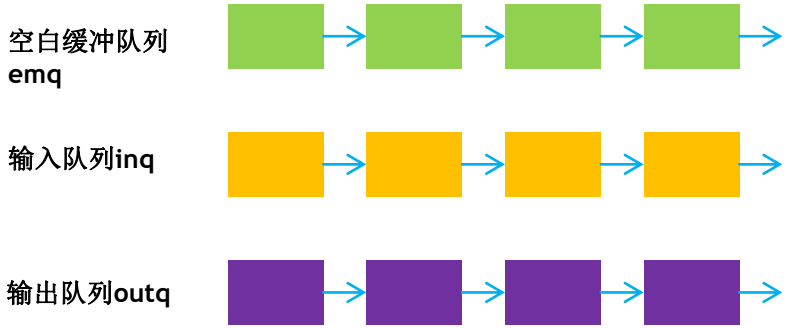


①输入进程请求输入数据
②计算进程想要取得一块输入数据
③计算进程想要将准备好的数据冲入缓冲区

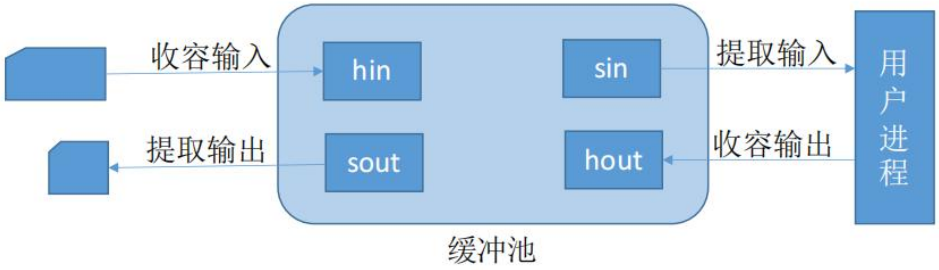


从空缓冲队列中取出一块作为“收容输出数据的工作缓冲区（hout）”。数据冲满后将缓冲区挂到输出队列队尾

缓冲池



①输入进程请求输入数据
②计算进程想要取得一块输入数据
③计算进程想要将准备好的数据冲入缓冲区
④输出进程请求输出数据



从输出队列中取得一块冲满输出数据的缓冲区作为“提取输出数据的工作缓冲区（sout）”。缓冲区读空后挂到空缓冲区队列

知识回顾


缓冲区管理
缓冲区的概念
一般利用内存作为缓冲区
缓解CPU与设备的速度矛盾、减少对CPU的中断频率、解决数据粒度不匹配的问题、提高CPU与I/O设备之间的并行性
单缓冲
设备—(T)—缓冲区—(M)—工作区—(C)—处理
处理一块数据平均耗时 Max(C, T)+M
分析问题的初始状态：工作区满，缓冲区空
双缓冲
处理一块数据平均耗时 Max(T, C+M)
分析问题的初始状态：工作区空，一个缓冲区满，另一个缓冲区空
循环缓冲
多个缓冲区链接成循环队列，in指针指向第一个空缓冲区，out指针指向第一个满缓冲区
缓冲池
三个队列：空缓冲队列、输入队列、输出队列
四种工作缓冲区
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