

Lecture 11: 知识 - IO管理概述

考试大纲

- (一) I/O 管理基础
 - 1. 设备
 - 设备的基本概念, 设备的分类, I/O 接口, I/O 端口
 - 2. I/O 控制方式
 - 轮询方式, 中断方式, DMA 方式
 - 3. I/O 软件层次结构
 - 中断处理程序, 驱动程序, 设备独立软件, 用户层 I/O 软件
 - 4. 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, 阻塞/非阻塞 I/O
- (二) 设备独立软件
 - 1. 缓冲区管理
 - 2. 设备分配与回收
 - 3. 假脱机技术 (SPooling)
 - 4. 设备驱动程序接口
- (三) 外存管理
 - 1. 磁盘
 - 磁盘结构, 格式化, 分区, 磁盘调度方法
 - 2. 固态硬盘
 - 读写性能特性, 磨损均衡

IO 设备

IO 管理基础概念

概念: 什么是 IO 设备

- 概念:
 - **IO** 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;
- **UNIX** 中的 **IO**:
 - **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;

IO 分类与部件类型

概念: 分类 - 按信息交换的单位

- 块设备:

- 概念：
 - 如磁盘等数据传输的基本单位是**块**；
- 特点：
 - 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：
 - 概念：
 - 鼠标、键盘等数据传输的基本单位是**字符**；
 - 特点：
 - 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；

概念：机械部件

- 概念：
 - I/O 设备的机械部件主要用来执行具体 IO 操作；
- 举例：
 - 我们看得见摸得着的鼠标/键盘的按钮；显示器的 LED 屏；移动硬盘的磁臂、磁盘盘面；

概念：电子设备

- 概念：
 - I/O 设备的电子部件通常是一块插入主板扩充槽的印刷电路板。

IO 控制器

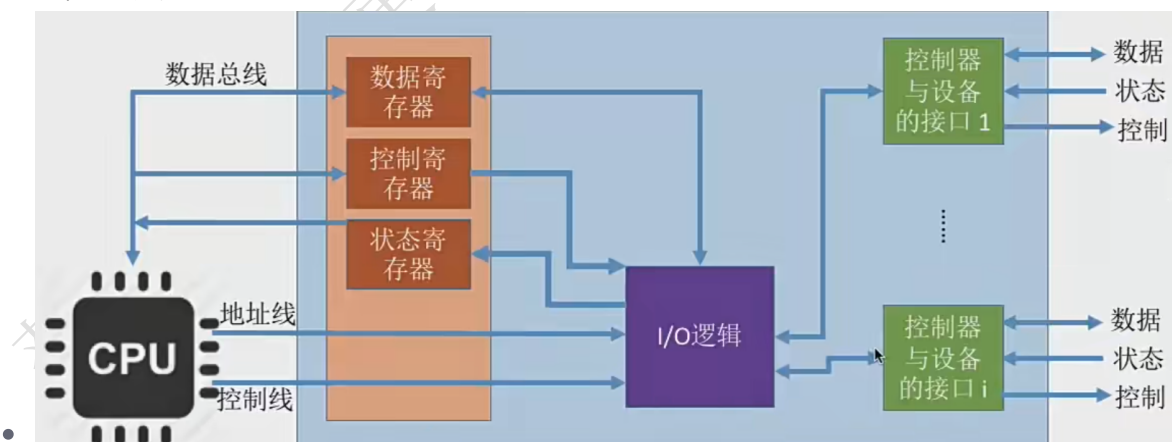
概念：IO 控制器的作用

- 概念：
 - CPU 无法直接控制 IO 设备的机械部件，因此 IO 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 IO 控制器，又称设备控制器；
 - CPU 可控制 IO 控制器，又由 IO 控制器来控制设备的机械部件；
- 功能：
 - 1. 接受和识别 CPU 发出的命令；
 - 如 CPU 发来的 read/write 命令，IO 控制器中会有相应的控制寄存器来存放命令和参数；
 - 2. 向 CPU 报告设备的状态；
 - 当前设备是忙碌还是空闲；
 - IO 控制器中会有相应的状态寄存器用于记录 I/O 设备的当前状态。如：1 表示空闲，0 表示忙碌
 - 3. 数据交换；
 - IO 控制器中会设置相应的数据寄存器；
 - 输出时，数据寄存器用于暂存 CPU 发来的数据，之后再由控制器传送设备；
 - 输入时，数据寄存器用于暂存设备发来的数据，之后 CPU 从数据寄存器中取走数据；

- 4. 地址识别；
 - 类似于内存的地址，为了区分设备控制器中的各个寄存器，也需要给各个寄存器设置一个特定的“地址”；
 - IO 控制器通过 CPU 提供的“地址”来判断 CPU 要读/写的是哪个寄存器；
- 组成：
 - CPU 与控制器接口：
 - 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；
 - IO 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；
 - 控制器与设备接口：
 - 用于实现控制器和设备之间的通信；
- 补充：
 - 1. 一个 IO 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，且这些寄存器都要有相应的地址，才能方便 CPU 操作；
 - 如：每个控制/状态寄存器对应一个具体的设备；
 - 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 IO；
 - 另一些计算机则采用 IO 专用地址，即寄存器独立编址；
 - 3. 注意数据总线是双向的，而地址线、控制线只能从 CPU 发往 IO 控制器；

• 图示：

- IO 控制器组成



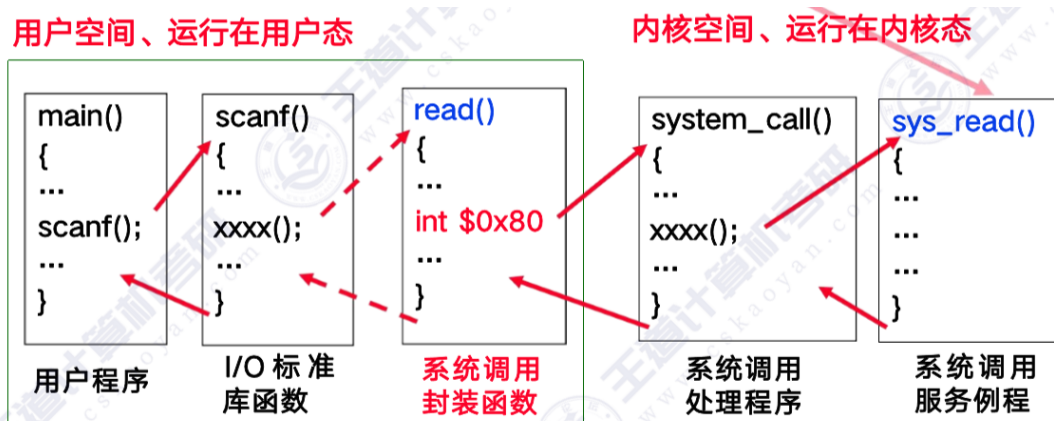
• 真题：2024年

- 题目：
 - 键盘中断服务例程执行结束时，数据所输入的存放位置是：
 - A. 用户缓冲区
 - B. CPU 的通用寄存器
 - C. 内核缓冲区
 - D. 键盘控制器的数据缓冲区

• 答案：

- C

- 键盘中断服务例程执行结束时，所输入的数据存放位置是内核缓冲区，当键盘中断发生时，操作系统会暂停当前正在执行的任务，转而执行键盘中断服务例程；
- 这个例程会从键盘控制器读取输入的数据，并将其存放在内核缓冲区中，然后当用户程序准备好读取数据时，它可以从内核缓冲区中获取。
- 分析：从 IO 设备中读取数据的过程
 - (1) 用户进程 A 执行，假设执行了 `scanf` 函数，期望从键盘获得一个字符的数据；
 - (2) 进入 `scanf` 调用的库函数，然后进入系统调用，然后到标准接口 `read` 函数；
 - 在 `read` 函数中有一条 `int $0x80` 的陷入指令，使得当前进程进行上下文切换、进入系统调用的执行；



- (3) 然后再到驱动程序 `foo_read` 函数，这些都是普通的函数调用，都是在进程 A 的资源平台上运行的；
- (4) 接下来在 `foo_read` (`sys_read`) 函数中，在使用 `outb` 指令启动了这次 IO 操作后，进程 A 就会被阻塞起来；
 - 进程 A 阻塞时，在其 `sys_read` 中执行以下语句，等待内核缓冲区数据；


```
Sleep_if_empty(kernalbuf)           // 如果内核缓冲区为空，则进程一直阻塞等待
do {
    GETCH(kernalbuf)                 // 从内核缓冲区中取出字符
    put_fs_byte(c)                   // 将取到的字符复制到用户缓冲区当中
} while (nr>0 && !EMPTY(kernalbuf)) // 还有待输入的字符并且内核缓冲区不空则继续取字符
.....
}
```
- (5) 若当前内核缓冲区无数据，当前进行进程切换，调度另一个进程 B 去运行；
- (6) 当 B 在运行时，如果 IO 操作完成，就会发生一次中断，把进程 B 打断，并跳转到中断处理程序 `foo_interruptible` 去执行，然后在这里再去唤醒进程 A；
 - A 的键盘中断服务例程：


```
foo->data =inb (DEV_FOO_DATA_PORT) // 将设备控制器中读出数据

PUTCH (kernalbuf)                 //将取出的字符送入内核缓冲区
... ..

Wake_up_interruptible(); // 唤醒相应的阻塞进程
```
- (7) 进程 A 被唤醒后，从内核缓冲区取出数据；

- 总结：用户程序发出 IO 请求后，系统的处理流程
 - 用户程序 -> 系统调用程序 -> 设备驱动程序 -> 中断处理程序

概念：内存映像 IO 与寄存器独立编址

- 概念：
 - 内存映射 IO -> 控制器中的寄存器与内存地址统一编址；
 - 寄存器独立编址 -> 控制器中的寄存器使用单独的地址；
- 寄存器独立编址的缺点：
 - 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；
- 图示：
 - 内存映像 IO：



- 寄存器独立编址：



补充：I/O 请求转成硬件操作

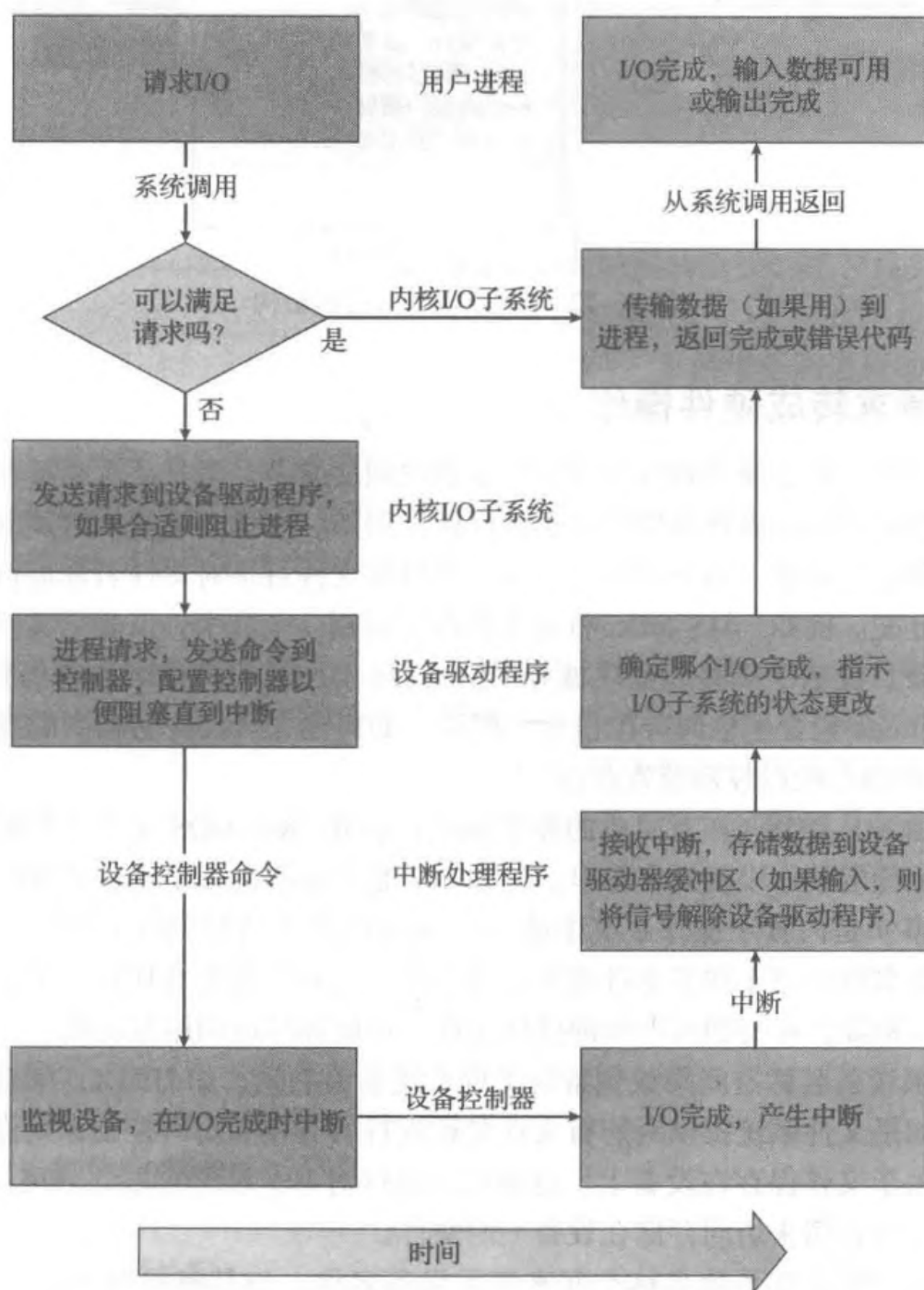
补充：read 与阻塞

- 概念：

- 1. 针对以前已经打开文件的文件描述符，进程调用阻塞系统调用 `read`
- 2. 内核中的系统调用代码检查参数是否正确。对于输入，如果数据已在缓冲缓存，则数据返回到进程，并完成 I/O 请求；
- 3. 否则，必须执行物理 I/O 请求；该进程从运行队列移到设备的**等待队列**（阻塞队列），并调度 I/O 请求；最后，I/O 子系统发送请求到**设备驱动程序**；
 - 根据操作系统的不同，这个请求可以通过子程序调用或内核消息来传递；
- 4. 设备驱动程序分配**内核缓冲区**空间，来接收数据并调度 IO；
 - 最终，设备驱动程序通过写入**设备控制器寄存器**，对**设备控制器**发送命令；
- 5. **设备控制器**控制设备硬件，以便执行数据传输；
- 6. 驱动程序可以**轮询**检测状态和数据，或者它可以通过 `DMA` 来传输到内核内存；
 - 假设 `DMA` 控制器管理传输，当传输完成时它会产生中断；

- 7. 正确的中断处理程序通过中断向量表收到中断，保存任何必要的数
据，并向**内核设备驱动程序**发送信号通知，并从中断返回；
- 8. **设备驱动程序**接收信号，确定 **I/O** 请求是否完成，确定请求状态，并对内核 **I/O** 子系统发送信号来通知请求已经完成；
- 9. 内核传输数据或返回代码到请求**进程的地址空间**，并且将进程从等待队列移到就绪队列；
- 10. 移动进程到**就绪队列**使得这个进程不再阻塞；
 - 当调度程序为进程分配 **CPU** 时，该进程就在系统调用完成后继续执行；

• 图示：



IO 控制方式

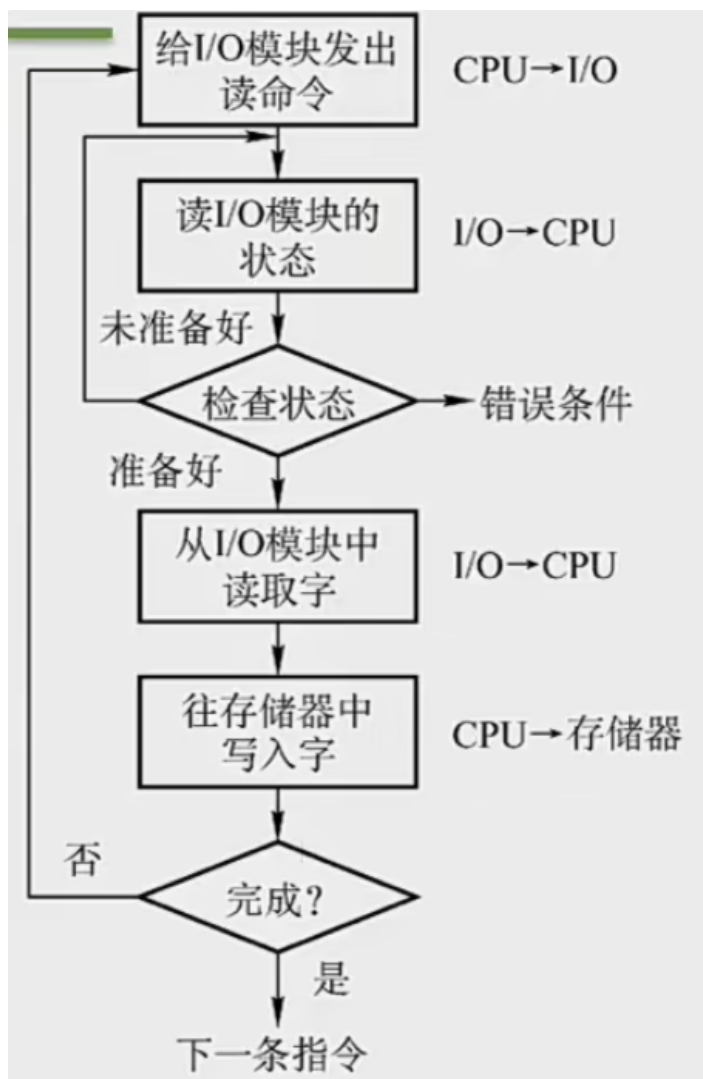
分类：四种控制方式

- 程序直接控制方式
- 中断驱动方式
- DMA 方式
- 通道控制方式

程序直接控制方式

概念：程序直接控制方式

- 方式：
 - 轮询；
- 实现：
 - 1. CPU 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)；
 - 2. 轮询检查**控制器的状态** (其实就是在不断地执行程序循环，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)；
 - 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；
 - 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)；
 - 5. CPU 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；
- 分析：
 - CPU 干预频率：
 - **很频繁**，IO 操作开始之前、完成之后需要 CPU 介入，并且在等待 IO 完成的过程中 CPU 需要不断地轮询检查；
 - 数据传送的单位：
 - 每次读写一个**字**；
 - 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 -> IO 设备每个字的读/写都需要 CPU 的帮助；
- 特点：
 - 优点：
 - 实现简单；
 - 在读/写指令之后，加上实现循环检查的系列指令即可 (因此才称为“程序直接控制方式”)；
 - 缺点：
 - CPU 和 IO 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；
- 举例：完成一次读写操作 (读操作举例)

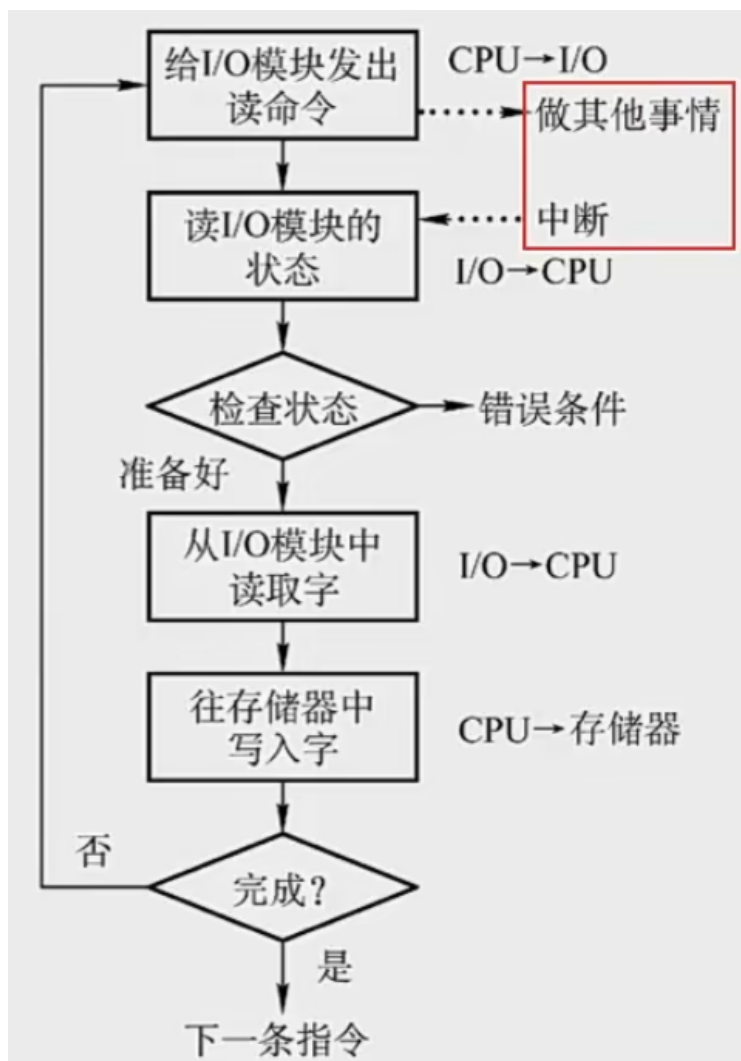


中断驱动方式

概念：中断驱动方式

- 方式：
 - 引入中断机制；
- 实现：
 - 由于 IO 设备速度很慢，因此在 CPU 发出读/写命令后，可将等待 IO 的进程阻塞，此时 CPU 先切换到别的进程执行；
 - 当 IO 完成后，IO 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 IO 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 IO 的进程（或其他进程）的运行环境，然后继续执行；
- 分析：
 - CPU 干预频率：
 - 低，每次 IO 操作开始之前、完成之后需要 CPU 介入；
 - 等待 IO 完成的过程中 CPU 可以切换到别的进程执行；
 - 数据传送的单位：
 - 每次读写一个字；

- 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 → IO 设备每个字的读/写都需要 CPU 的帮助;
- 特点:
 - 优点:
 - IO 控制器会通过中断信号主动报告 IO 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 IO 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;
- 注意:
 1. CPU 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;
- 举例:



真题: 2023年

- 题目:

【2023 统考真题】进程 P 通过执行系统调用从键盘接收一个字符的输入，已知此过程中与进程 P 相关的操作包括：①将进程 P 插入就绪队列；②将进程 P 插入阻塞队列；③将字符从键盘控制器读入系统缓冲区；④启动键盘中断处理程序；⑤进程 P 从系统调用返回；⑥用户在键盘上输入字符。以上编号①~⑥仅用于标记操作，与操作的先后顺序无关。请回答下列问题。

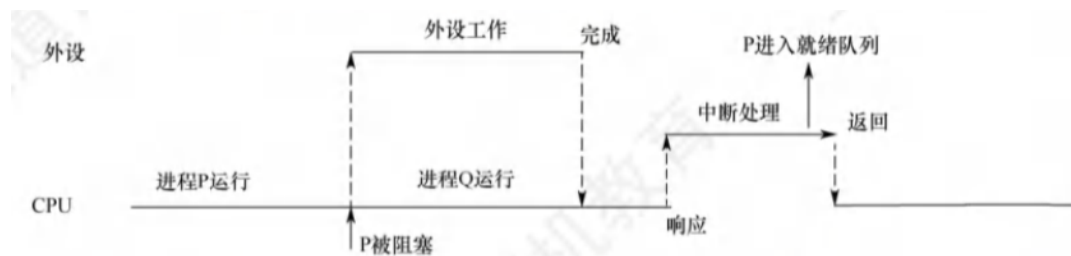
- 1) 按照正确的操作顺序，操作①的前一个和后一个操作分别是上述操作中的哪一个？操作⑥的后一个操作是上述操作中的哪一个？
- 2) 在上述哪个操作之后 CPU 一定从进程 P 切换到其他进程？在上述哪个操作之后 CPU 调度程序才能选中进程 P 执行？
- 3) 完成上述哪个操作的代码属于键盘驱动程序？
- 4) 键盘中断处理程序执行时，进程 P 处于什么状态？CPU 是处于内核态还是处于用户态？

• 解析：

• 第一问：

- 正确的执行顺序为 2、6、4、3、1、5；
 - 因此操作①的前一个操作是③，后一个操作是⑤；
 - 操作⑥的后一个操作是④；
- 下面以 `scanf()` 函数的执行过程为例分析相关操作顺序；
 - 1. 用户进程 P 调用 IO 标准库的 `scanf()` 函数，`scanf()` 调用标准库中的函数 `read()`；
 - 2. `read()` 在这个函数中有一条陷阱指令，通过它陷入内核，函数中使用的 `sys_read()` 系统调用服务例程进行执行，进入内核；
 - 内核 IO 子系统层；
 - 3. 进入内核后，在设备无关层进行若干调用，最终到设备驱动层进行处理；
 - 4. 设备驱动程序对本次 IO 进行初始化操作，然后执行操作系统的进程调度程序 `scheduler()`，由调度程序 `scheduler()` 来阻塞用户进程 P，并调度其他进程执行；
 - 操作 2
 - 5. 在用户输入字符之前，CPU 运行其他进程，键盘等待用户输入数据；
 - 操作 6
 - 6. 当用户在键盘上输入字符后，外设向 CPU 发出相应的中断请求；
 - 操作 4
 - 7. CPU 响应中断后，启动键盘中断处理程序，该处理程序对 IO 中断进行简单通用的处理后，唤醒具体的处理键盘输入的驱动程序；
 - 8. 驱动程序将字符从键盘控制器读入系统缓冲区；
 - 操作 3
 - 9. 数据传输完成后，唤醒 P，并插入就绪队列；
 - 操作 1
 - 10. 进行中断返回，进程 P 也从系统调用返回；
 - 操作 5

• 图示以上过程：



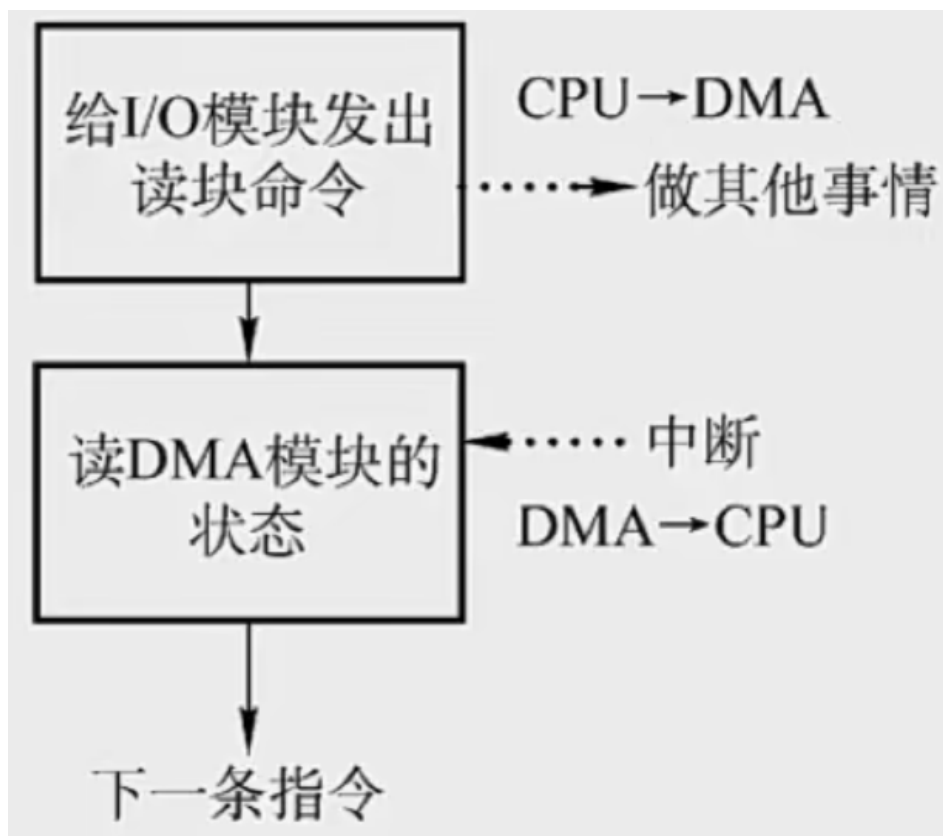
- 第二问：
 - 在操作 2 之后，即在进程进入阻塞态之后，CPU 一定从进程 P 切换到其他进程；
 - 在操作 1 之后，即进程进入就绪队列后，CPU 调度程序才能选中进程 P 执行；
- 第三问：
 - 设备驱动程序负责驱动 I/O 设备工作，比如 IO 操作初始化，执行具体的 I/O 指令；
 - 将字符从键盘控制器读入系统缓冲区是和键盘直接相关的具体操作，完成操作的代码属于键盘驱动程序，因此选 3
- 第四问：
 - 键盘中断处理程序执行时，进程 P 还在阻塞队列，处于阻塞态；
 - 中断处理程序、设备驱动程序、设备独立性软件 都属于内核 IO 软件层，执行相关代码时，CPU 处于内核态；

DMA 方式

概念： DMA 方式

- 概念：
 - DMA 方式 (Direct Memory Access)，直接存储器存取；
 - 主要用于块设备的；
 - DMA 的传送过程主要分为预处理、数据传送和后处理三个阶段；
- 特征：
 - 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；
- 实现：
 - 流程：
 - 真题：2017年
 - (1) 预处理阶段：
 - 首先，初始化 DMA 控制器中的有关寄存器、设置传送方向、测试并启动设备等；
 - 然后，CPU 继续执行原程序，直到 IO 设备准备好发送的数据 (输入情况) 或接收的数据 (输出情况) 时，IO 设备向 DMA 控制器发送 DMA 请求，再由 DMA 控制器向 CPU 发出总线请求 (这两个过程也可统称 DMA 请求)，用以传输数据；
 - 注意：预处理阶段只是向 CPU 发起了总线请求，并没有涉及中断；

- (2) 传输阶段:
 - DMA 以**数据块**为基本传送单位;
 - DMA 占用总线后的数据输入/输出操作都是通过循环来实现的, 这一循环也是由 DMA **控制器**实现的, 即数据传送阶段完全由 DMA (**硬件**) 控制;
- (3) 后处理阶段:
 - 1. DMA 控制器向 CPU 发出**中断请求**;
 - 2. CPU 执行相应的中断处理程序, 执行“DMA **结束**”中断服务程序;
 - 包括检查**传输过程是否正确**和**数据校验**等
- 处理:
 - CPU -> DMA :
 - 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);
 - DMA -> CPU :
 - 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;
- 分析:
 - CPU 干预频率:
 - **低**, 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;
 - 开始时不需要中断, 只是申请总线使用权;
 - 结束时需要中断;
 - 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**连续的多块**, 且这些块读入内存后**在内存中也必须是连续的**;
 - 数据的流向:
 - 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
 - 优先级:
 - CPU 和 DMA 对于总线的占用优先级:
 - 由于采用周期挪用 DMA 方式, 因此当 CPU 和 DMA 控制器都需要访问主存时, DMA 控制器可以优先获得总线使用权;
 - 因为一旦磁盘开始读/写, 就必须按时完成数据传送, 否则数据缓冲区中的数据会发生丢失;
- 特点:
 - 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**, 不能离散的存取;
- 过程:

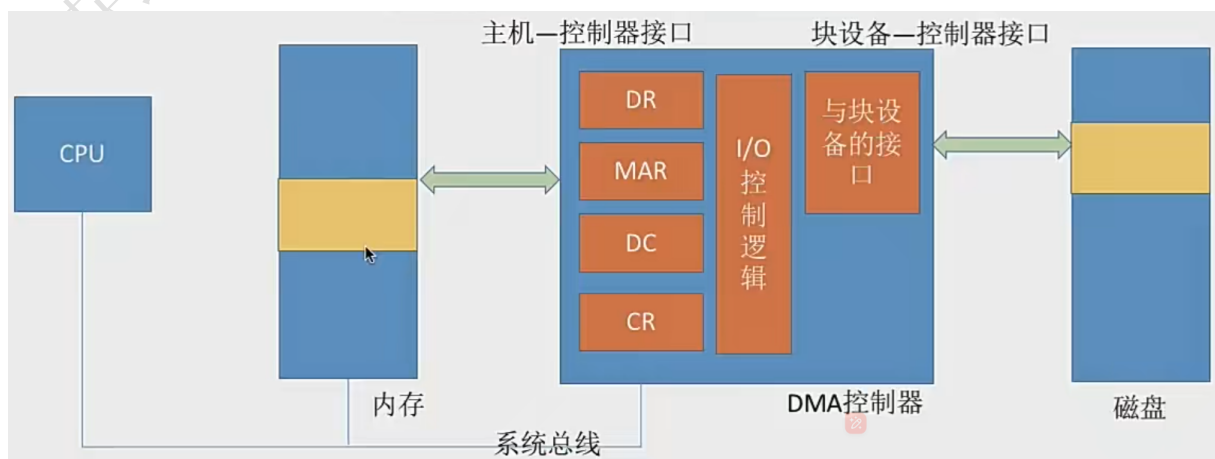


概念：DMA 构成

• 构成：

- 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 IO 命令，或设备的状态信息；

• 图示：



通道控制方式

概念：通道控制方式

- 概念：
 - 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；
- 特征：
 - 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；
- 实现：
 - 1. CPU 向通道发出 IO 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 IO 设备。之后 CPU 就切换到其他进程执行了；
 - 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）；
 - 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；
- 分析：
 - CPU 干预频率：
 - **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；
 - 因为 CPU 可以一口气丢给通道一堆事情，让通道去执行；
 - 数据传送的单位：
 - 每次读/写**一组数据块**；
 - 数据的流向：
 - 读操作（数据输入）：I/O 设备 -> 内存；
 - 写操作（数据输出）：内存 -> IO 设备；
- 特点：
 - 优点：
 - CPU、通道、IO 设备可并行工作，资源利用率很高；
 - 缺点：
 - 实现复杂，需要专门的通道硬件支持；

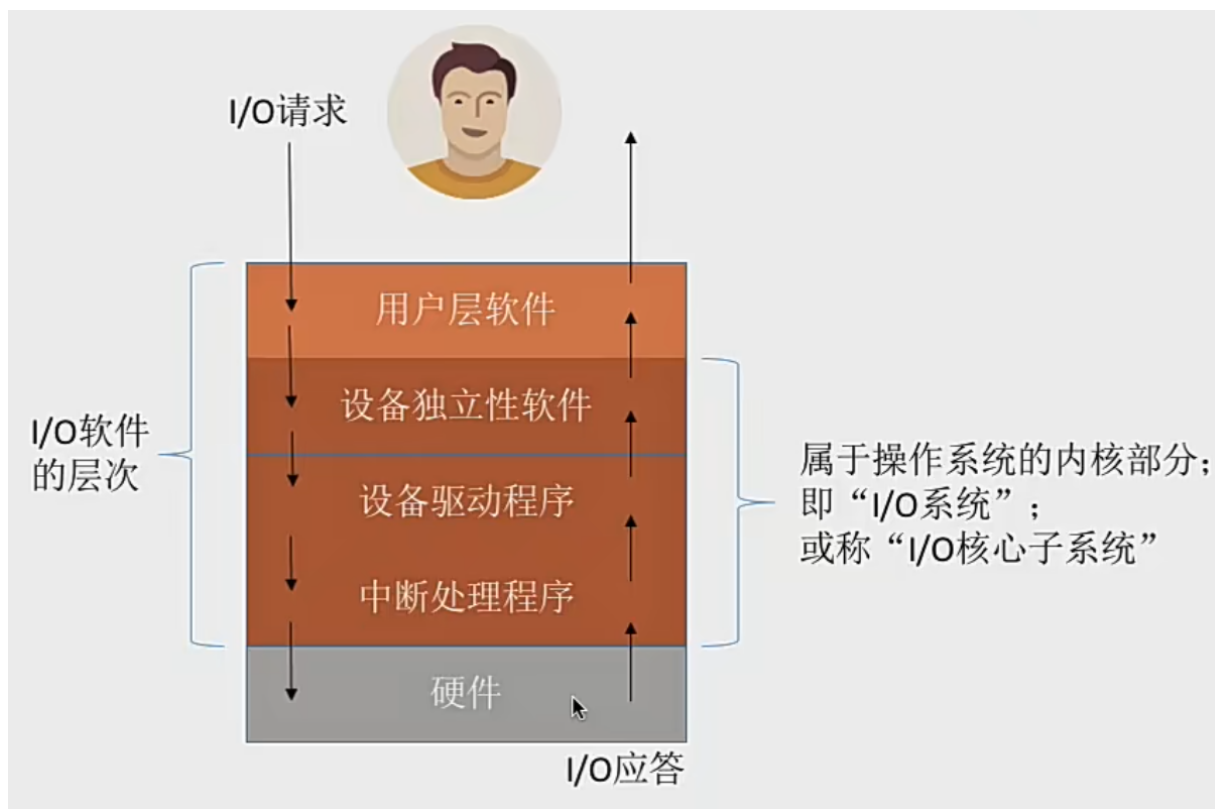
总结

	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命令后可以去做其他事，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命令后可以去做其他事，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命令后可以去做其他事。通道会执行程序以完成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	

IO 软件层次结构

概念： IO 软件层次结构

- 概念：
 - 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”）；
- 注意：
 - 理解并记住 IO 软件各个层次之间的顺序，要能够推理判断某个处理应该是在哪个层次完成的；
 - 最常考的是设备独立性软件、设备驱动程序这两层。只需理解一个特点即可：
 - 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；
 - 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；
- 图示：



总结：IO 事件层次处理顺序

- 总结：
 - 从上往下：如果是主机**主动**发出 IO 请求（比如发起读磁盘 IO），则是从上往下的顺序；
 - 从下往上：如果是主机**被动**响应硬件的 IO 应答（比如用户敲键盘），则是从下往上的顺序；
- 从上往下：
 - (1) 用户层软件
 - 又被称作 用户级IO软件
 - (2) 设备独立性软件
 - 又被称作 系统调用处理程序 或者 设备无关软件
 - (3) 设备驱动程序
 - 真题：2013年
 - 用户发出读 IO 请求时，负责计算磁盘的柱面号、磁头号和扇区号的程序是设备驱动程序；
 - (4) 中断处理程序
 - (5) 硬件
 - 如果是提问系统的处理顺序，则不包含此部分，因为这是硬件的范畴，不属于系统处理；
 - 如果是提问 XXX 软件/程序来处理，则不包含此部分，因为这是硬件的范畴，不属于软件/程序；
- 从下往上：
 - (1) 硬件

- 如果是提问系统的处理顺序，则不包含此部分，因为这是硬件的范畴，不属于系统处理；
- 如果是提问 `XXX` 软件/程序来处理，则不包含此部分，因为这是硬件的范畴，不属于软件/程序；
- (2) 中断处理程序
 - 真题：2010年
 - 问题：本地用户通过键盘登录系统时，首先获得键盘输入信息的程序是？
 - 答案：中断处理程序
- (3) 设备驱动程序
- (4) 设备独立性软件
- (5) 用户层软件

用户层软件

概念：用户层软件

- 作用：
 - 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 `IO` 操作相关的库函数对设备进行操作；
- 实现：
 - 用户层软件将用户请求翻译成格式化的 `IO` 请求，并通过“系统调用”请求操作系统内核的服务；
 - 比如 `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
 - 如 `WindowsAPI`；

设备独立性软件

概念：设备独立性软件

- 概念：
 - 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；
- 功能：
 - 1. 设备的**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；
 - 2. **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；
 - 3. 设备的**分配与回收**
 - 4. 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；
 - 5. 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；

- 举例：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)；
- 逻辑设备表：
 - 概念：
 - 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；
 - 图示：

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
...	...	...
- 实现：
 - 方法一：
 - 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；
 - 方法二：
 - 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；

设备驱动程序

引入：为什么需要设备驱动程序

- 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；

概念：设备驱动程序

- 概念：
 - 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；
 - 因此驱动程序与设备控制器和 **I/O** 设备的硬件特性紧密相关；
 - **真题：**2013年
 - 用户程序发出磁盘 **I/O** 后，由磁盘驱动程序来负责计算数据所在磁盘的柱面号、磁头号以及扇区号；
- 原因：
 - 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；
 - 设备独立性不能直接访问硬件，而是通过设备驱动程序；
- 注意：

- 驱动程序一般会以一个独立进程的方式存在；
- 真题：2022年
- 进程在执行驱动程序时可能会进入阻塞态；

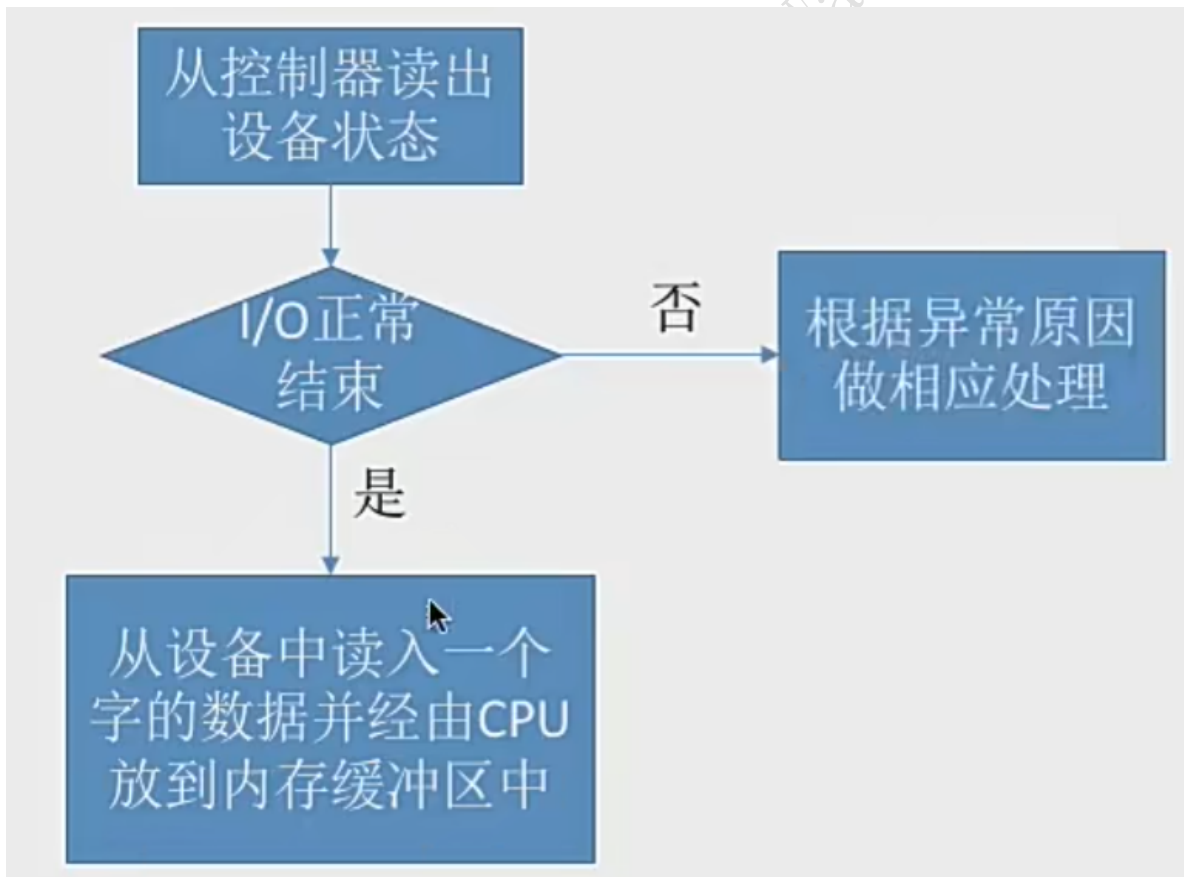
总结：驱动程序的工作内容

- (1) 初始化设备；
- (2) 驱动程序控制完成读写设备的操作；

中断处理程序

概念：中断处理程序

- 概念：
 - 当 IO 任务完成时，IO 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；
 - 因此可以知道，中断处理程序也会和硬件直接打交道；
- 实现：



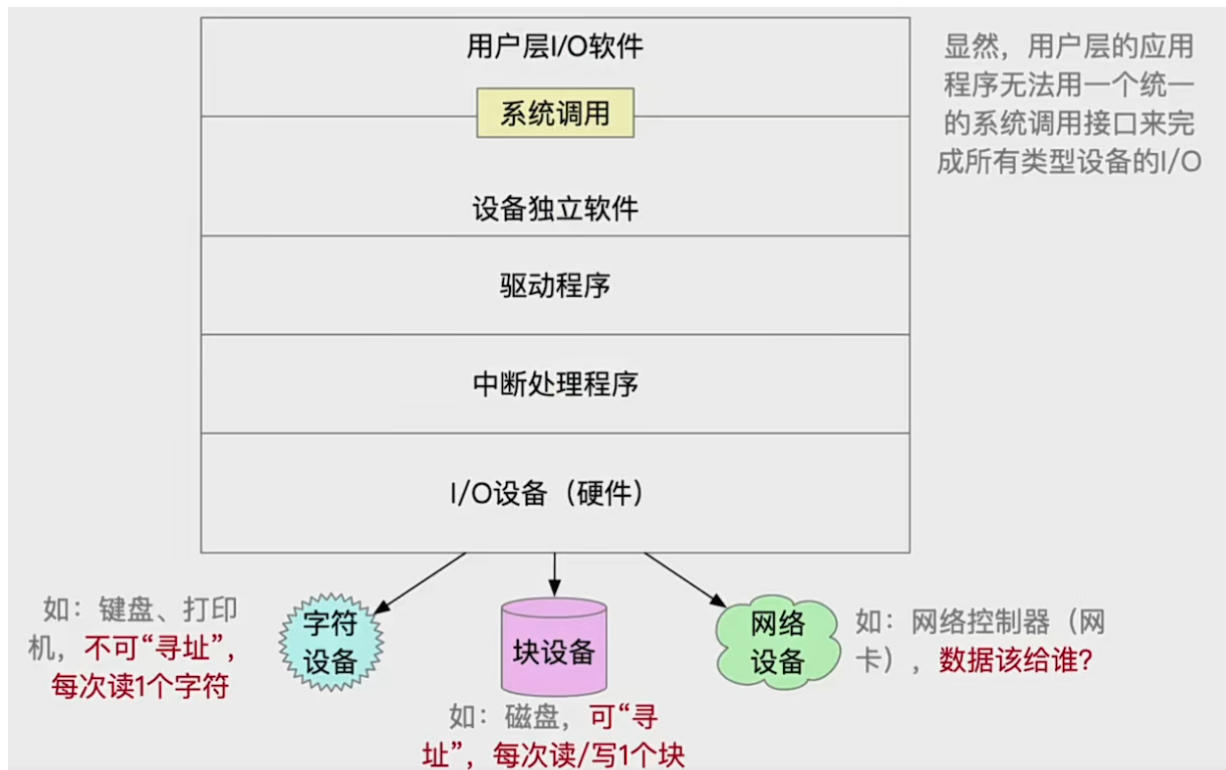
输入/输出应用程序接口

概念：输入/输出应用程序接口

- 问题：
 - 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 IO；
 - 所以设备独立软件需要为上层，提供不同的应用程序接口；
- 分类：三种类型的应用程序接口

- 1. 字符设备接口；
- 2. 块设备接口；
- 3. 网络设备接口；

• 图示：



概念： 三种类型的应用程序接口

- 字符设备接口：
 - 没有地址的概念；
 - `get/put` 系统调用：向字符设备读/写一个字符；
 - 以**字符**为单位；
- 块设备接口：
 - 有地址的概念；
 - `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：
 - 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

• 图示：



概念：阻塞 IO 和非阻塞 IO

- 阻塞 IO：
 - 概念：
 - 应用程序发出 IO 系统调用，进程需转为阻塞态等待；
 - 举例：
 - 字符设备接口--从键盘读一个字符 `get`
- 非阻塞 IO：
 - 概念：
 - 应用程序发出 IO 系统调用，系统调用可迅速返回，进程无需阻塞等待
 - 举例：
 - 块设备接口--往磁盘写数据 `write`