06 输入输出系统

6.1 I/O系统的功能、模型和接口

> *I/O系统管理的主要对象是「I/O设备」和相应的「设备控制器」。*

6.1.1 基本功能

- 方便使用I/O设备
  - 隐藏物理设备的细节
  - 与设备的无关性
- 提高CPU和I/O设备利用率
  - 提高处理机和I/O设备的利用率
  - 对I/O设备进行控制
- 共享设备时提供方便
  - 确保对设备的正确共享
  - 错误处理

1.隐藏物理设备的细节

> *配置硬件设备：设备控制器。其中包含若干个存放控制命令、参数的寄存器。*

2.与设备的无关性

> *与设备的无关性是在较晚时才实现的，这是在隐藏物理设备细节的基础上实现的。*

1. 抽象的I/O命令
2. 抽象的逻辑设备名。
3. 提高可移植性和易适应性。

3.提高处理机和I/O设备的利用率

> *要尽可能地让* `处理机和I/O设备并行操作` *，以提高它们的利用率。*

- 处理机能快速响应用户的I/O请求，使I/O设备尽快地运行起来。
- 尽量减少在每个I/O设备运行时处理机的干预时间。

4.对I/O设备进行控制

> *对I/O设备进行控制是驱动程序的功能。*

- 采用轮询的可编程I/O方式
- 采用中断的可编程I/O方式
- 直接存储器访问方式
- I/O通道方式

采用何种控制方式，与设备的传输速率、传输的数据单位等因素有关。

5.确保对设备的正确共享

从设备的共享属性上，可将系统中的设备分为如下两类：

1. 独占设备
   进程互斥地访问这类设备。即系统一旦把这类设备分配给了某进程后，便由该进程独占，直至用完释放。

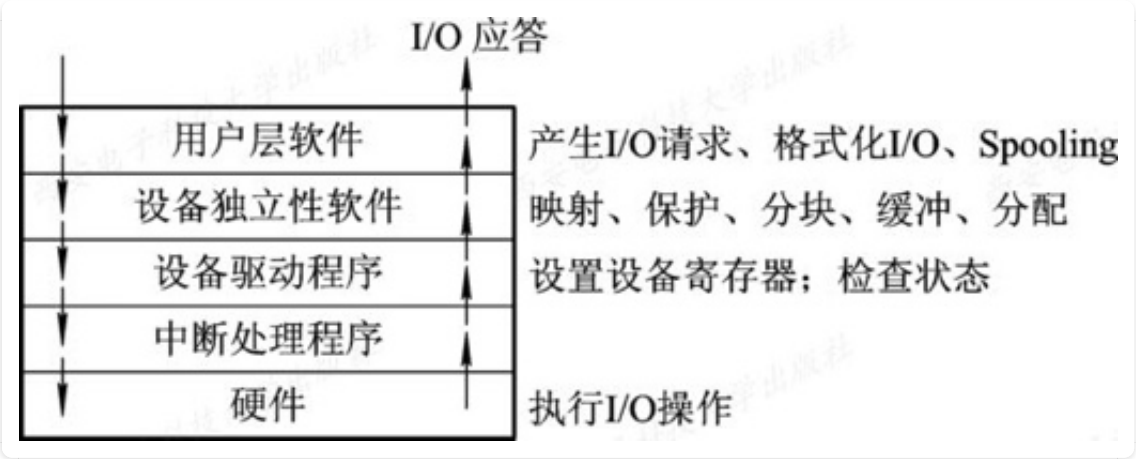
2. 共享设备
   一段时间内允许多个进程同时访问的设备。

6.错误处理

> *大多数的设备都包括了较多的机械和电气部分，运行时容易出现错误和故障。*
> *从处理的角度，可将错误分为临时性错误和持久性错误。对于临时性错误，可通过重试操作来纠正，只有在发生了持久性错误时，才需要向上层报告。*

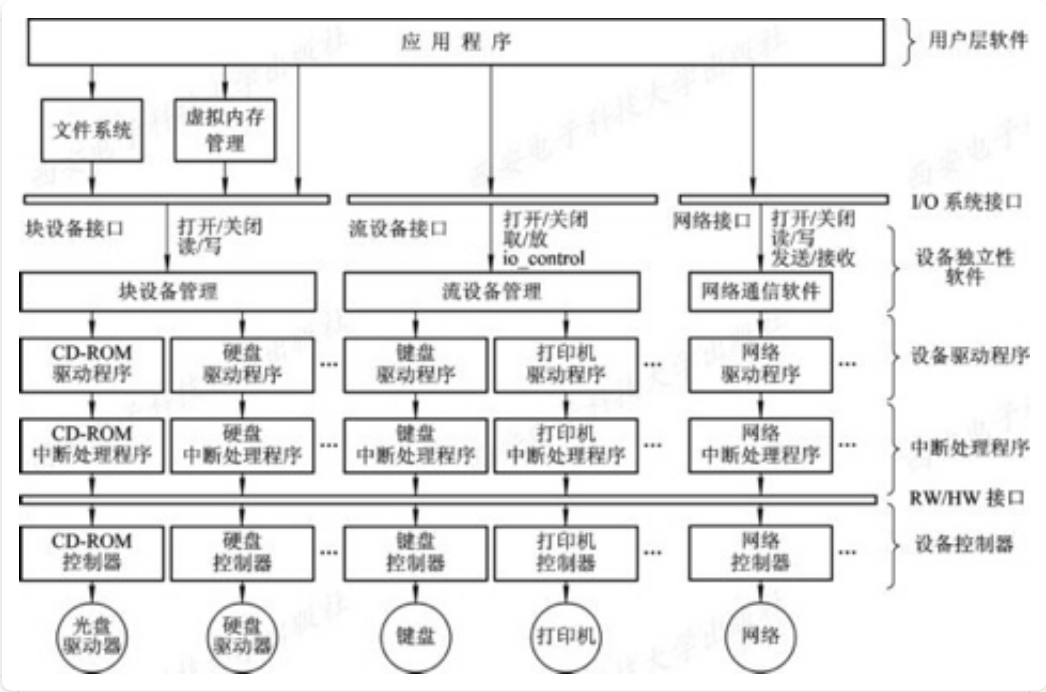
6.1.2 I/O系统的层次结构和模型

1.I/O软件的层次结构



1. 用户层软件：实现用户交互的接口。
2. 设备独立性软件：实现用户程序与设备驱动器的统一接口、设备命名、设备的保护、分配及释放。同时为设备管理和数据传送提供必要存储空间。
3. 设备驱动程序：与硬件直接相关。
4. 中断处理程序：保存被中断进程的CPU环境。

2.I/O系统中各种模块之间的层次视图



（1）I/O系统的上、下接口

- I/O系统接口
- 软件/硬件(RW/HW)接口

（2）I/O系统的分层

I/O系统本身分为三个层次：

- 中断处理程序：处于底层，直接与硬件进行交互
- 设备驱动程序
- 设备独立性软件

6.1.3 I/O系统接口

1.块设备接口

- 块设备
- 隐藏了磁盘的二维结构
- 将抽象命令映射为低层操作

2.流设备接口（字符设备接口）

> *字符设备指数据存取和传输以字符为单位的设备。*
> *基本特征：传输速率较低（每秒几个字节到数千字节），不可寻址。常用中断驱动方式*

- get和put操作
- in–control指令

3.网络通信接口

6.2 I/O设备和设备控制器

> *I/O设备一般是由执行I/O操作的* `机械部分` *和执行控制I/O的* `电子部件` *组成。*
> *通常将这两部分分开，执行I/O操作的机械部分就是一般的I/O设备，而执行控制I/O的电子部件则称为设备控制器或适配器(adapter)。*
> *在微型机和小型机中的控制器常做成印刷电路卡形式，因而也常称为控制卡、接口卡或网卡，可将它插入计算机的扩展槽中。在有的大、中型计算机系统中，还配置了I/O通道或I/O处理机。*

6.2.1 I/O设备

I/O设备的类型

按使用特性分类

- 存储设备：外存（辅存）
- I/O设备
    - 输入设备
    - 输出设备
    - 交互式设备

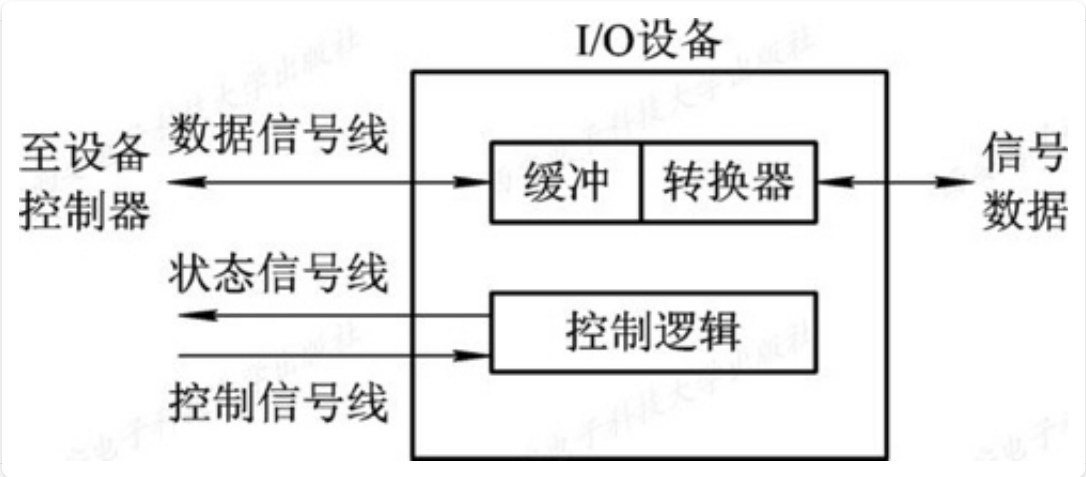
按传输速率分类

- 低速
- 中速
- 高速

6.2.2 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I/O设备中应含有与设备控制器间的接口，在该接口中有三种类型的信号，各对应一条信号线。*

- 数据信号线
- 控制信号线
- 状态信号线



6.2.3 设备控制器

1.基本功能

（1）接收和识别命令

控制器中含有控制寄存器，用来存放接收的命令和参数再译码。

（2）数据交换

通过数据总线实现CPU与控制器之间数据交换；
通过数据寄存器实现控制器和设备之间的数据交换。

（3）标识和报告设备的状态

设置一状态寄存器，为了让CPU了解。

（4）地址识别

配置地址译码器，每一个设备都有一个唯一地址。

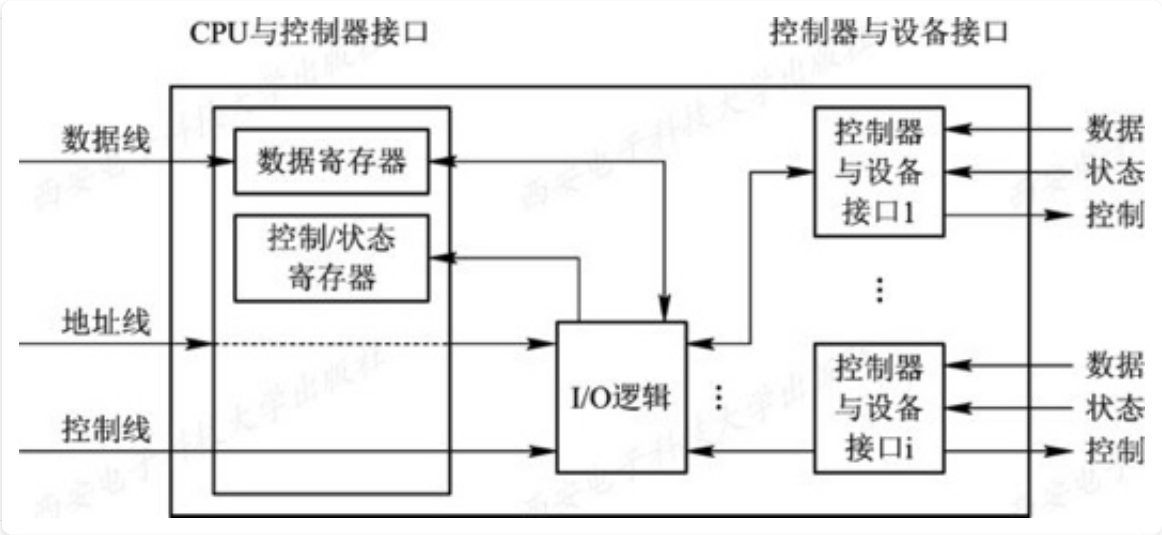
（5）数据缓冲区

解决速度不匹配的问题。

（6）差错控制

2.设备控制器的组成

> *设备控制器位于CPU与设备之间，它既要与CPU通信，又要与设备通信。*



（1）设备控制器与处理机的接口

> *实现CPU与设备控制器之间的通信*

- 数据线
  - 数据寄存器
  - 控制/状态寄存器
- 地址线
- 控制线

（2）设备控制器与设备的接口

控制器有多个设备接口，每个接口存在数据、控制、状态三种信号。

（3）I/O逻辑

> *实现对设备的控制。*

6.2.4 内存映像I/O

1.利用特定的I/O指令

在早期的计算机中，包括大型计算机，为实现CPU和设备控制器之间的通信，为每个控制寄存器分配一个I/O端口，这是一个8位或16位的整数，如图6–5(a)所示。另外还设置了一些特定的I/O指令。

2.内存映像I/O

在这种方式中，在编址上不再区分内存单元地址和设备控制器中的寄存器地址，都采用k。当k值处于0～n–1范围时，被认为是内存地址，若k大于等于n时，被认为是某个控制器的寄存器地址。

图6-5 设备寻址形式

6.2.5 I/O通道(I/O Channel)

> *在CPU和设备控制器之间增设了I/O通道主要目的是建立独立的I/O操作。*

I/O通道和一般处理机不同，表现在：

- 指令类型单一
- 没有自己的内存

1.通道类型

（1）字节多路通道(Byte Multiplexor Channel)

这是一种按字节交叉方式工作的通道。它通常都含有许多非分配型子通道，其数量可从几十到数百个，每一个子通道连接一台I/O设备，并控制该设备的I/O操作。这些子通道按时间片轮转方式共享主通道。

图6-6 字节多路通道的工作原理

（2）数组选择通道(Block Selector Channel)

字节多路通道不适于连接高速设备，这推动了按数组方式进行数据传送的数组选择通道的形成。

（3）数组多路通道(Block Multiplexor Channel)

数组选择通道虽有很高的传输速率，但它却每次只允许一个设备传输数据。数组多路通道是将数组选择通道传输速率高和字节多路通道能使各子通道(设备)分时并行操作的优点相结合而形成的一种新通道。

2.“瓶颈”问题

由于通道价格昂贵，致使机器中所设置的通道数量势必较少，这往往又使它成了I/O的瓶颈，进而造成整个系统吞吐量的下降。

解决办法：增加通路而不增加通道。

图6-7　单通路I/O系统图6-8　多通路I/O系统

6.3 中断机构和中断处理程序

> *中断在操作系统中有着特殊重要的地位，它是多道程序得以实现的基础，因为进程之间的切换是通过中断来完成的。*
> *另一方面，中断也是设备管理的基础，为了提高处理机的利用率和实现CPU与I/O设备并行执行，也必需有中断的支持。*
> *中断处理程序是I/O系统中最低的一层，它是整个I/O系统的基础。*

6.3.1 中断简介

1.中断和陷入

（1）中断

> *指CPU对I/O设备发来的中断信号的一种响应。*
> *是外部引起的*
> *I/O设备可以是字符设备、块设备、通信设备。*

（2）陷入

> *如同运算上溢或下溢，程序出错，非法指令。*
> *是内部引起的。*

2.中断向量表和中断优先级

（1）中断向量表

（2）中断优先级

3.对多中断源的处理方式

（1）屏蔽(禁止)中断

（2）嵌套中断

图6-9 对多中断的处理方式

6.3.2 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I/O设备完成I/O操作后，设备控制器便向CPU发送一个中断请求，CPU响应后便转向中断处理程序，中断处理程序执行相应的处理，处理完后解除相应进程的阻塞状态。

图6-10 中断现场保护示意图图6-11 中断处理流程

6.4 设备驱动程序

> *设备驱动程序是I/O系统的高层与设备控制器之间的通信程序*
> *其主要任务是接收上层软件发来的抽象I/O要求，如read或write命令，再把它转换为具体要求后，发送给设备控制器，启动设备去执行。反之，它也将由设备控制器发来的信号传送给上层软件。*

6.4.1 设备驱动程序概述

1.设备驱动程序的功能

1. 接收由与设备无关的软件发来的命令和参数，并将命令中的抽象要求转换为与设备相关的低层操作序列。2. 检查用户I/O请求的合法性，了解I/O设备的工作状态，传递与I/O设备操作有关的参数，设置设备的工作方式。3. 发出I/O命令，如果设备空闲，便立即启动I/O设备，完成指定的I/O操作；如果设备忙碌，则将请求者的请求块挂在设备队列上等待。4. 及时响应由设备控制器发来的中断请求，并根据其中断类型，调用相应的中断处理程序进行处理。

2.设备驱动程序的特点

> *设备驱动程序属于低级的系统例程，它与一般的应用程序及系统程序之间有下述明显差异*
> *驱动程序是实现在与设备无关的软件和设备控制器之间通信和转换的程序，具体说，它将抽象的I/O请求转换成具体的I/O操作后传送给控制器。又把控制器中所记录的设备状态和I/O操作完成情况，及时地反映给请求I/O的进程。*
> *驱动程序与设备控制器以及I/O设备的硬件特性紧密相关，对于不同类型的设备，应配置不同的驱动程序。但可以为相同的多个终端设置一个终端驱动程序。*

驱动程序与I/O设备所采用的I/O控制方式紧密相关，常用的I/O控制方式是中断驱动和DMA方式。
由于驱动程序与硬件紧密相关，因而其中的一部分必须用汇编语言书写。目前有很多驱动程序的基本部分已经固化在ROM中。
驱动程序应允许可重入。一个正在运行的驱动程序常会在一次调用完成前被再次调用。

3.设备处理方式

1. 为每一类设备设置一个进程，专门用于执行这类设备的I/O操作。
2. 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。
3. 不设置专门的设备处理进程，而只为各类设备设置相应的设备驱动程序。

6.4.2 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，完成上层指定的I/O工作。但在启动之前，应先完成必要的准备工作，如检测设备状态是否为“忙”等。在完成所有的准备工作后，才向设备控制器发送一条启动命令。*

1. 将抽象要求转换为具体要求；
2. 对服务请求进行校验；
3. 检查设备的状态；
4. 传送必要的数据；

5. 启动I/O设备。

图6-12 状态寄存器中的格式

6.4.3 对I/O设备的控制方式

> *宗旨：尽量减少主机对I/O控制的干预，把主机从繁杂的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。*

1.使用轮询的可编程I/O方式

处理机对I/O设备的控制采取轮询的可编程I/O方式，即在处理机向控制器发出一条I/O指令，启动输入设备输入数据时，要同时把状态寄存器中的忙/闲标志busy置为1，然后便不断地循环测试busy(称为轮询)。当busy=1时，表示输入机尚未输完一个字(符)，处理机应继续对该标志进行测试，直至busy=0，表明输入机已将输入数据送入控制器的数据寄存器中。于是处理机将数据寄存器中的数据取出，送入内存指定单元中，这样便完成了一个字(符)的I/O。接着再去启动读下一个数据，并置busy=1。

图6-13 程序I/O和中断驱动方式的流程

2.使用中断的可编程I/O方式

当前，对I/O设备的控制，广泛采用中断的可编程I/O方式，即当某进程要启动某个I/O设备工作时，便由CPU向相应的设备控制器发出一条I/O命令，然后立即返回继续执行原来的任务。设备控制器于是按照该命令的要求去控制指定I/O设备。此时，CPU与I/O设备并行操作。

3.直接存储器访问方式

1) 接存储器访问方式的引入

虽然中断驱动I/O比程序I/O方式更有效，但它仍是以字(节)为单位进行I/O的。每当完成一个字(节)的I/O时，控制器便要向CPU请求一次中断。

该方式的特点是：(1) 数据传输的基本单位是数据块，即在CPU与I/O设备之间，每次传送至少一个数据块。(2) 所传送的数据是从设备直接送入内存的，或者相反。(3) 仅在传送一个或多个数据块的开始和结束时，才需CPU干预，整块数据的传送是在控制器的控制下完成的。可见，DMA方式较之中断驱动方式又进一步提高了CPU与I/O设备的并行操作程度。

2）DMA控制器的组成　　DMA控制器由三部分组成：主机与DMA控制器的接口；DMA控制器与块设备的接口；I/O控制逻辑。图6–14示出了DMA控制器的组成。这里主要介绍主机与控制器之间的接口。

图6–14 DMA控制器的组成

3）DMA工作过程　　当CPU要从磁盘读入一数据块时，便向磁盘控制器发送一条读命令。该命令被送入命令寄存器CR中。同时，需要将本次要读入数据在内存的起始目标地址送入内存地址寄存器MAR中。

图6–15 DMA方式的工作流程图

4.I/O通道控制方式　　1）I/O通道控制方式的引入

虽然DMA方式比起中断方式来已经显著地减少了CPU的干预，即已由以字(节)为单位的干预减少到以数据块为单位的干预，但CPU每发出一条I/O指令，也只能去读(或写)一个连续的数据块。而当我们需要一次去读多个数据块且将它们分别传送到不同的内存区域，或者相反时，则须由CPU分别发出多条I/O指令及进行多次中断处理才能完成。

2) 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。通道程序是由一系列通道指令(或称为通道命令)所构成的。

下面示出了一个由六条通道指令所构成的简单的通道程序。该程序的功能是将内存中不同地址的数据写成多个记录。

6.5 与设备无关的I/O软件

> *为了方便用户和提高OS的可适应性与可扩展性，在现代OS的I/O系统中，都无一例外地增加了与设备无关的I/O软件，以实现设备独立性，也称为设备无关性。*
> *其基本含义是：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。为每个设备所配置的设备驱动程序是与硬件紧密相关的软件。*
> *实现设备独立性，必须再在设备驱动程序之上设置一层软件，称为与设备无关的I/O软件（设备独立性软件）。*

6.5.1 与设备无关(Device Independence)软件的基本概念

1. 早期以物理设备名使用设备
2. 引入了逻辑设备名实现I/O重定向

3. 逻辑设备名称到物理设备名称的转换 ## 6.5.2 与设备无关的软件 > 与设备无关的软件是I/O系统的最高层软件。

1.设备驱动程序的统一接口

为了使所有的设备驱动程序有着统一的接口，一方面，要求每个设备驱动程序与OS之间都有着相同的接口，或者相近的接口，这样会使添加一个新的设备驱动程序变得很容易，同时在很大程度上方便了开发人员对设备驱动程序的编制。另一方面，要将抽象的设备名映射到适当的驱动程序上，或者说，将抽象的设备名转换为具体的物理设备名，并进一步可以找到相应物理设备的驱动程序入口。此外，还应对设备进行保护，禁止用户直接访问设备，以防止无权访问的用户使用。

2.缓冲管理

无论是字符设备还是块设备，它们的运行速度都远低于CPU的速度。为了缓和CPU和I/O设备之间的矛盾、提高CPU的利用率，在现代OS中都无一例外地分别为字符设备和块设备配置了相应的缓冲区。缓冲区有着多种形式，如单缓冲区、双缓冲区、循环缓冲区、公用缓冲池等，以满足不同情况的需要。

3.差错控制

由于设备中有着许多的机械和电气部分，因此，它们比主机更容易出现故障，这就导致I/O操作中的绝大多数错误都与设备有关。错误可分为如下两类：

(1) 暂时性错误。 (2) 持久性错误。

4.对独立设备的分配与回收

在系统中有两类设备：独占设备和共享设备。对于独占设备，为了避免诸进程对独占设备的争夺，必须由系统来统一分配，不允许进程自行使用。每当进程需要使用某(独占)设备时，必须先提出申请。OS接到对设备的请求后，先对进程所请求的独占设备进行检查，看该设备是否空闲。若空闲，才把该设备分配给请求进程。否则，进程将被阻塞，放入该设备的请求队列中等待。等到其它进程释放该设备时，再将队列中的第一个进程唤醒，该进程得到设备后继续运行。

5.独立于设备的逻辑数据块

不同类型的设备，其数据交换单位是不同的，读取和传输速率也各不相同，如字符型设备以单个字符(字)为单位，块设备是以一个数据块为单位。即使同一

类型的设备，其数据交换单位的大小也是有差异的，如不同磁盘由于扇区大小的不同，可能造成数据块大小的不一致。设备独立性软件应能够隐藏这些差异而被逻辑设备使用，并向高层软件提供大小统一的逻辑数据块。与设备无关软件的功能如图6–16所示。

图6–16 与设备无关软件的功能层次

6.5.3 设备分配

> *系统为实现对独占设备的分配，必须在系统中配置相应的数据结构。*

1.设备分配中的数据结构

（1）设备控制表DCT

系统为每一个设备都配置了一张设备控制表，用于记录设备的情况，如图6–17所示。

图6–17 设备控制表

（2）控制器控制表、通道控制表和系统设备表

(1) 控制器控制表(COCT)。系统为每一个控制器都设置了用于记录控制器情况的控制器控制表，如图6–18(a)所示。 (2) 通道控制表(CHCT)。每个通道都有一张通道控制表，如图6–18(b)所示。 (3) 系统设备表(SDT)。这是系统范围的数据结构，记录了系统中全部设备的情况，每个设备占一个表目，其中包括有设备类型、设备标识符、设备控制表及设备驱动程序的入口等项，如图6–18(c)所示。

图6–18 COCT、CHCT和SDT表

2.设备分配时应考虑的因素 #### （1）设备的固有属性

- 独占设备的分配策略。
- 共享设备的分配策略。（注意进程访问的先后次序要有合理的调度。）
- 虚拟设备的分配策略。（虚拟设备属于可共享的设备，可以将它同时分配给多个进程使用。）

（2）设备分配算法* 先来先服务。

- 优先级高者优先。

（3）设备分配中的安全性* 安全分配方式。

- 不安全分配方式。

3.独占设备的分配程序　　1) 基本的设备分配程序　　我们通过一个例子来介绍设备分配过程。当某进程提出I/O请求后，系统的设备分配程序可按下述步骤进行设备分配：　　(1) 分配设备。　　(2) 分配控制器。　　(3) 分配通道。

2) 设备分配程序的改进　　在上面的例子中，进程是以物理设备名提出I/O请求的。如果所指定的设备已分配给其它进程，则分配失败。或者说上面的设备分配程序不具有与设备无关性。为获得设备的独立性，进程应使用逻辑设备名请求I/O。

6.5.4 逻辑设备名到物理设备名映射的实现

1. 逻辑设备表LUT(Logical Unit Table)　　在逻辑设备表的每个表目中包含了三项：逻辑设备名、物理设备名和设备驱动程序的入口地址，如图6–19(a)所示。

图6–19 逻辑设备表

2. 逻辑设备表的设置问题　　在系统中可采取两种方式设置逻辑设备表：

第一种方式，是在整个系统中只设置一张LUT。　　第二种方式，是为每个用户设置一张LUT。

6.6 用户层的I/O软件## 6.6.1 系统调用与库函数### 1.系统调用　　一方面，为使诸进程能有条不紊地使用I/O设备，且能保护设备的安全性，不允许运行在用户态的应用进程去直接调用运行在核心态(系统态)的OS过程。但另一方面，应用进程在运行时，又必须取得OS所提供的服务，否则，应用程序几乎无法运行。为了解决此矛盾，OS在用户层中引入了一个中介过程——系统调用，应用程序可以通过它间接调用OS中的I/O过程，对I/O设备进行操作。

图6–20 系统调用的执行过程

2.库函数　　在C语言以及UNIX系统中，系统调用(如read)与各系统调用所使用的库函数(如read)之间几乎是一一对应的。而微软定义了一套过程，称为Win32 API的应用程序接口(Application Program Interface)，程序员利用它们取得OS服务，该接口与实际的系统调用并不一一对应。用户程序通过调用对应的库函数使用系统调用，这些库函数与调用程序连接在一起，被嵌入在运行时装入内存的二进制程序中。

6.6.2 假脱机(Spooling)系统### 1.假脱机技术　　在20世纪50年代，为了缓和CPU的高速性与I/O设备低速性间的矛盾，而引入了脱机输入、脱机输出技术。该技术是利用专门的外围控制机，先将低速I/O设备上的数据传送到高速磁盘上，或者相反。这样当处理机需要输入数据时，便可以直接从磁盘中读取数据，极大地提高了输入速度。反之，在处理机需要输出数据时，也可以很快的速度把数据先输出到磁盘上，处理机便可去做自己的事情。

2.SPOOLing的组成　　如前所述，SPOOLing技术是对脱机输入/输出系统的模拟，相应地，如图6-21(a)所示，SPOOLing系统建立在通道技术和多道程序技术的基础上，以高速随机外存(通常为磁盘)为后援存储器。SPOOLing的工作原理如图6-21(b)所示。

图6–21 SPOOLing系统组成及工作原理

SPOOLing系统主要由以下四部分构成：　　(1) 输入井和输出井。　　(2) 输入缓冲区和输出缓冲区。　　(3) 输入进程和输出进程。　　(4) 井管理程序。

3. SPOOLing系统的特点　　(1) 提高了I/O的速度。　　(2) 将独占设备改造为共享设备。　　(3) 实现了虚拟设备功能。

4. 假脱机打印机系统　　打印机是经常用到的输出设备，属于独占设备。利用假脱机技术可将它改造为一台可供多个用户共享的打印设备，从而提高设备的利用率，也方便了用户。共享打印机技术已被广泛地用于多用户系统和局域网络中。假脱机打印系统主要有以下三部分：　　(1) 磁盘缓冲区。　　(2) 打印缓冲区。　　(3) 假脱机管理进程和假脱机打印进程。

5. 守护进程(daemon)　　前面是利用假脱机系统来实现打印机共享的一种方案，人们对该方案进行了某些修改，如取消该方案中的假脱机管理进程，为打印机建立一个守护进程，由它执行一部分原来由假脱机管理进程实现的功能，如为用户在磁盘缓冲区中申请一个空闲盘块，并将要打印的数据送入其中，将该盘块的首址返回给请求进程。另一部分由请求进程自己完成，每个要求打印的进程首先生成一份要求打印的文件，其中包含对打印的要求和指向装有打印输出数据盘块的指针等信息，然后将用户请求打印文件放入假脱机文件队列(目录)中。

6.7 缓冲区管理> 缓冲区是一个存储区域，它可以由专门的硬件寄存器组成，但硬件成本较高，容量也较小。

> *主要功能是组织好这些缓冲区，并提供获得和释放缓冲区的手段。*

6.7.1 缓冲的引入　　引入缓冲区的原因有很多，可归结为以下几点：　　(1) 缓和CPU与I/O设备间速度不匹配的矛盾。　　(2) 减少对CPU的中断频率，放宽对CPU中断响应时间的限制。　　(3) 解决数据粒度不匹配的问题。　　(4) 提高CPU和I/O设备之间的并行性。

图6–22 利用缓冲寄存器实现缓冲

6.7.2 单缓冲区和双缓冲区　　1. 单缓冲区(Single Buffer)　　在单缓冲情况下，每当用户进程发出一I/O请求时，操作系统便在主存中为之分配一缓冲区，如图6-23所示。

图6–23 单缓冲工作示意图

2. 双缓冲区(Double Buffer)　　由于缓冲区是共享资源，生产者与消费者在使用缓冲区时必须互斥。如果消费者尚未取走缓冲区中的数据，即使生产者又生产出新的数据，也无法将它送入缓冲区，生产者等待。如果为生产者与消费者设置了两个缓冲区，便能解决这一问题。

图6–24 双缓冲工作示意图

如果在实现两台机器之间的通信时仅为它们配置了单缓冲，如图6–25(a)所示，那么，它们之间在任一时刻都只能实现单方向的数据传输。例如，只允

许把数据从A传送到B，或者从B传送到A，而绝不允许双方同时向对方发送数据。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，一个用作发送缓冲区，另一个用作接收缓冲区，如图6–25(b)所示。

图6–25 双机通信时缓冲区的设置

6.7.3 环形缓冲区　　1. 环形缓冲区的组成　　(1) 多个缓冲区。在环形缓冲中包括多个缓冲区，其每个缓冲区的大小相同。作为输入的多缓冲区可分为三种类型：用于装输入数据的空缓冲区R、已装满数据的缓冲区G以及计算进程正在使用的现行工作缓冲区C，如图6-26所示。

图6–26 环形缓冲区

2. 环形缓冲区的使用　　计算进程和输入进程可利用下述两个过程来使用形环缓冲区。　　(1) Getbuf过程。　　(2) Releasebuf过程。

3. 进程之间的同步问题　　使用输入循环缓冲，可使输入进程和计算进程并行执行。相应地，指针Nexti和指针Nextg将不断地沿着顺时针方向移动，这样就可能出现下述两种情况：　　(1) Nexti指针追赶上Nextg指针。　　(2) Nextg指针追赶上Nexti指针。

6.7.4 缓冲池(Buffer Pool)　　1. 缓冲池的组成　　缓冲池管理着多个缓冲区，每个缓冲区由用于标识和管理的缓冲首部以及用于存放数据的缓冲体两部分组成。缓冲首部一般包括缓冲区号、设备号、设备上的数据块号、同步信号量以及队列链接指针等。为了管理上的方便，一般将缓冲池中具有相同类型的缓冲区链接成一个队列，于是可形成以下三个队列：　　(1) 空白缓冲队列emq。　　(2) 输入队列inq。　　(3) 输出队列outq。

2. Getbuf过程和Putbuf过程　　在数据结构课程中，曾介绍过队列和对队列进行操作的两个过程，第一个是 Addbuf(type，number)过程。该过程用于将由参数number所指示的缓冲区B挂在type队列上。第二个是Takebuf(type)过程。它用于从type所指示的队列的队首摘下一个缓冲区。

3. 缓冲区的工作方式　　缓冲区可以工作在如下四种工作方式，如图6–27所示。

图6–27 缓冲区的工作方式

6.8 磁盘存储器的性能和调度

6.8.1 磁盘性能简述　　磁盘设备是一种相当复杂的机电设备，在此仅对磁盘的某些性能，如数据的组织、磁盘的类型和访问时间等方面做扼要的阐述。

1. 数据的组织和格式　　磁盘设备可包括一个或多个物理盘片，每个磁盘片分一个或两个存储面(Surface)(见图6-28(a))，每个盘面上有若干个磁道(Track)，磁道之间留有必要的间隙(Gap)。为使处理简单起见，在每条磁道上可存储相同数目的二进制位。

图6-28 磁盘的结构和布局

图6-29 磁盘的格式化

2. 磁盘的类型　　对于磁盘，可以从不同的角度进行分类。最常见的有：将磁盘分成硬盘和软盘、单片盘和多片盘、固定头磁盘和活动头(移动头)磁盘等。下面仅对固定头磁盘和移动头磁盘做些介绍。　　(1) 固定头磁盘。
(2) 移动头磁盘。

3. 磁盘访问时间　　磁盘设备在工作时以恒定速率旋转。为了读或写，磁头必须能移动到所指定的磁道上，并等待所指定的扇区的开始位置旋转到磁头下，然后再开始读或写数据。

6.8.2 早期的磁盘调度算法　　1. 先来先服务(FCFS)　　这是最简单的磁盘调度算法。它根据进程请求访问磁盘的先后次序进行调度。

2. 最短寻道时间优先(SSTF)　　该算法选择这样的进程，其要求访问的磁道与当前磁头所在的磁道距离最近，以使每次的寻道时间最短，但这种算法不能保证平均寻道时间最短。

图6-30 FCFS调度算法

图6-31 SSTF调度算法

6.8.3 基于扫描的磁盘调度算法　　1. 扫描(SCAN)算法　　SSTF算法的实质是基于优先级的调度算法，因此就可能导致优先级低的进程发生“饥饿”(Starvation)现象。因为只要不断有新进程的请求到达，且其所要访问的磁

道与磁头当前所在磁道的距离较近，这种新进程的I/O请求必然优先满足。在对SSTF算法略加修改后，则可防止低优先级进程出现“饥饿”现象。

2. 循环扫描(CSCAN)算法　　SCAN算法既能获得较好的寻道性能，又能防止“饥饿”现象，故被广泛用于大、中、小型机器和网络中的磁盘调度。但也存在这样的问题：当磁头刚从里向外移动而越过了某一磁道时，恰好又有一进程请求访问此磁道，这时，该进程必须等待，待磁头继续从里向外，然后再从外向里扫描完处于外面的所有要访问的磁道后，才处理该进程的请求，致使该进程的请求被大大地推迟。

图6–32　SCAN调度算法示例

图6–33 CSCAN调度算法示例

3. NStepSCAN和FSCAN调度算法　　1）NStepSCAN算法　　在SSTF、SCAN及CSCAN几种调度算法中，都可能出现磁臂停留在某处不动的情况，例如，有一个或几个进程对某一磁道有较高的访问频率，即这个(些)进程反复请求对某一磁道的I/O操作，从而垄断了整个磁盘设备。我们把这一现象称为“磁臂粘着”(Armstickiness)。在高密度磁盘上容易出现此情况。

2）FSCAN算法　　FSCAN算法实质上是N步SCAN算法的简化，即FSCAN只将磁盘请求队列分成两个子队列。一个是由当前所有请求磁盘I/O的进程形成的队列，由磁盘调度按SCAN算法进行处理。另一个是在扫描期间，将新出现的所有请求磁盘I/O的进程放入等待处理的请求队列。这样，所有的新请求都将被推迟到下一次扫描时处理。