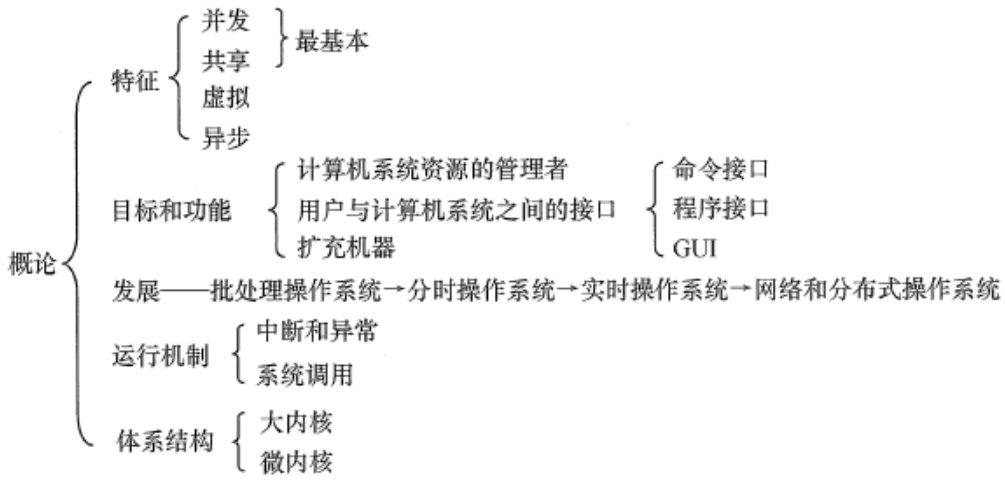
计算机操作系统



1操作系统 概论

1．1 什么是操作系统

操作系统是指控制和管理整个计算机的**硬件与软件资源**，合理地组织、调度计算机的**工作**与**资源的分配**，进而为**用户和其他软件**提供**方便接口**的**程序集合。**

1．2 操作系统的作用

操作系统作为计算机资源的管理者

> **处理机管理**（进程**控制**、进程**同步**、进程**通信**、**死锁**处理、处理机**调度**）
>
> **存储器管理**（提高内存**利用率**，内存的**分配与回收**、**地址映射**、内存**保护**与**共享**、内存**扩充**）
>
> **文件管理**（计算机中的信息都是以**文件的形式**存在的）
>
> **设备管理**（完成用户的I/O**请求**，方便用户**使用设备**、并提高设备的**利用率**）

作为用户与计算机硬件系统之间的接口

> **命令接口**（用户通过**控制台或终端**输入**操作命令**，向系统**提供**各种**服务要求**）
>
> **程序接口**（由`系统调用`组成，用户在**程序**中使用这些**系统调用**来请求**操作系统**为其**提供服务**）
>
> **图形接口** 最常见的`图形用户界面GUI`（最终还是通过**调用程序接口**实现的）

用作扩充机器

> 没有任何**软件支持**的计算机称为**裸机。**操作系统将裸机改造成功能更强、使用更方便的机器。我们将**覆盖了软件**的机器称为**扩充机器或虚拟机。**

1．3 操作系统的特征

并发：指两个或多个事件在**同一时间间隔内**发生，微观上还是程序在分时地交替执行

共享：指系统中的资源可供**内存**中**多个并发执行的进程共同使用**，主要包括两个方式，一种为**互斥共享方式**，在一段时间内只允许一个进程访问该资源；另一种为**同时访问方式。**

虚拟：指把一个**物理上的实体**变为若干个**逻辑上的对应物。**

异步：在**多道程序环境**下，允许多个程序并发执行，但由于**资源有限**，进程的执行**不是一贯到底**，而是以**不可预知的速度**向前推进

操作系统最基本的特征是并发和共享

1.4 操作系统的运行机制

1.内核程序和应用程序(内核态和用户态)

> 在计算机系统中，通常CPU执行两种不同性质的程序：一种是**操作系统内核程序**；另一种是**用户自编程序或系统外层的应用程序。**内核程序是应用程序的"管理者"。"管理程序"可以执行一些特权指令，而"被管理程序"出于安全考虑不能执行这些指令。所谓特权指令，是指计算机中不允许用户直接使用的指令。
>
> 操作系统在具体实现上划分了用户态（目态）和核心态（管态），以严格区分两类程序。

2.层次式结构

> **一些与硬件关联较紧密的模块**，诸如时钟管理、中断管理、设备驱动等处于最底层。其次是**运行频率较高的程序**，诸如进程管理、存储管理和设备管理等。 上面的这两部分内容**构成了操作系统的内核**，这部分内容的指令操作工作在**核心态。**

3.内核

> 内核是计算机上配置的底层软件，是计算机功能的延伸，包括以下4个方面的内容：时钟管理，中断机制，原语，系统控制的数据结构及处理。

1.5 中断和异常

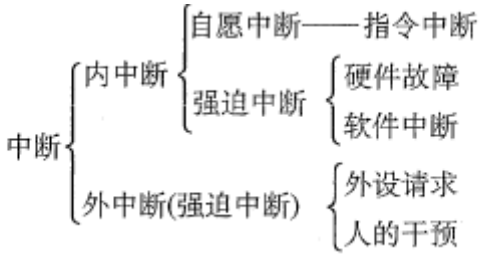
1.中断的引入——为了支持CPU和设备之间的并行操作

中断也称外中断，指来自CPU执行指令**以外的事件**的发生。这一类中断通常是与当前执行的指令无关的事件。

2.异常的引入——表示CPU执行指令本身时出现的问题

异常也称内中断、例外或陷入，指源自**CPU执行指令内部的事件。**对异常的处理一般要**依赖与当前程序的运行现场。**

3.中断和异常的联系与区别



4.中断执行的流程

不同计算机的中断（指外中断）处理过程各具特色，就其多数而论，中断处理流程如图 1.3 所示。各阶段处理流程的描述如下：

1）关中断。CPU 响应中断后，首先要保护程序的现场状态，在保护现场的过程中，CPU 不应响应更高级中断源的中断请求。否则，若现场保存不完整，在中断服务程序结束后，也就不能正确地恢复并继续执行现行程序。

2）保存断点。为保证中断服务程序执行完毕后能正确地返回到原来的程序，必须将原来的程序的断点（即程序计数器 PC）保存起来。

3）引出中断服务程序。其实质是取出中断服务程序的入口地址送入程序计数器 PC。

4）保存现场和屏蔽字。进入中断服务程序后，首先要保存现场，现场信息一般是指程序状态字寄存器 PSWR 和某些通用寄存器的内容。

5）开中断。允许更高级中断请求得到响应。

6）执行中断服务程序。这是中断请求的目的。

7）关中断。保证在恢复现场和屏蔽字时不被中断。

8）恢复现场和屏蔽字。将现场和屏蔽字恢复到原来的状态。

9）开中断、中断返回。中断服务程序的最后一条指令通常是一条中断返回指令，使其返回到原程序的断点处，以便继续执行原程序。

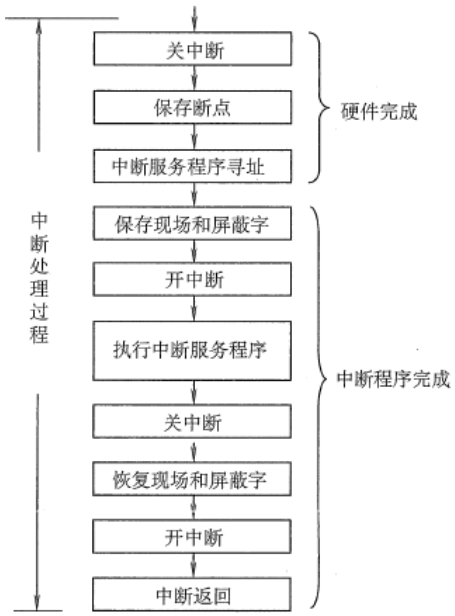
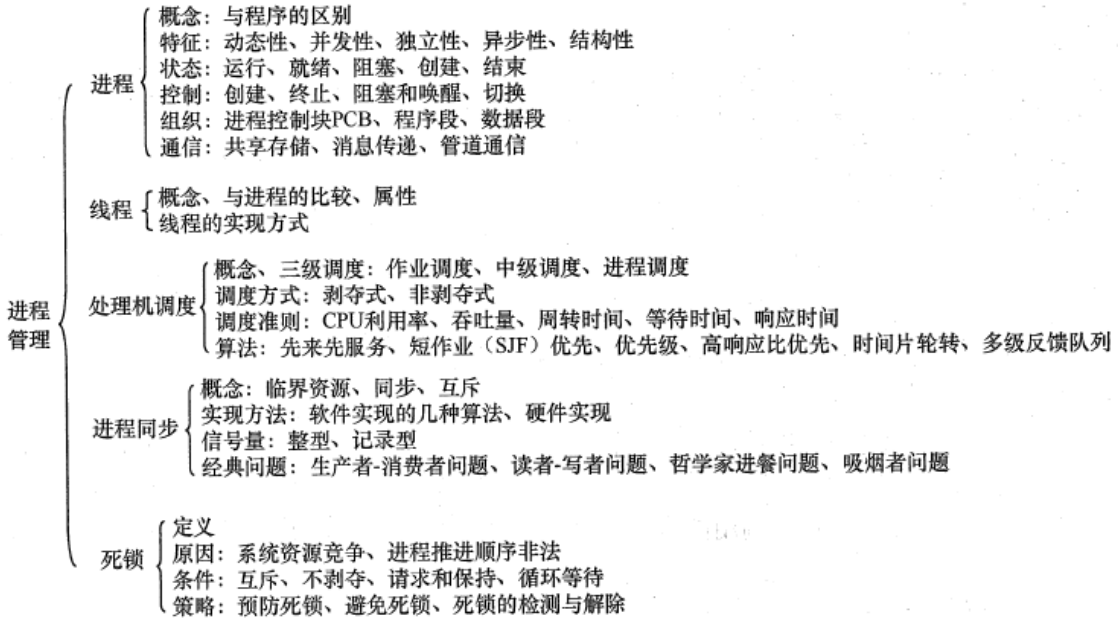


图 1.3　中断处理的流程

4 进程



4.1 进程的定义

进程是指一个具有**独立功能的程序**对某个**数据集**在处理机上的**执行过程和分配资源**的基本单位。每个进程包含**独立的地址空间**，进程各自的地址空间是**私有的**，只执行自己**地址空间中的程序**，且只能**访问**自己地址空间中的**数据**，相互访问会导致**指针的越界错误。**

进程是**操作系统进行资源分配**的基本单位。

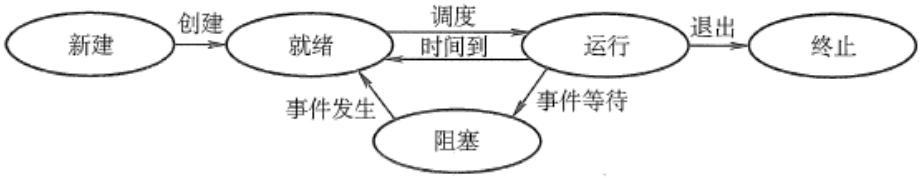
4.2 进程与程序的区别

（1） 程序是**永存**的；进程是**暂时**的，是**程序**在**数据集**上的一次执行，有创建有撤销，存在是暂时的；
（2）程序是**静态**的观念，进程是**动态**的观念；
（3）进程具有**并发性**，而程序没有；
（4）进程是**竞争计算机资源**的**基本单位**，程序不是。
（5）进程和程序**不是一一对应**的： **一个程序**可对应**多个进程**即多个进程可执行同一程序； **一个进程**可以执行**一个或几个程序**

4.3 进程的状态

进程的状态包括以下五种，分别为运行态，就绪态，阻塞态，新建态，终止态。

- 就绪态的进程通过调度算法从而获得 CPU 时间，转为运行态；
- 而运行态的进程，在分配给它的 CPU 时间片用完之后就会转为就绪态，等待下一次调度；
- 阻塞态是缺少需要的资源从而由运行态转换而来，但是该资源不包括 CPU，缺少 CPU 会让进程从运行态转换为就绪态。



5 线程

5.1 线程的定义

- 线程是**独立调度**的基本单位，**一个进程**中可以有**多个线程。**
- 线程是一个"**轻量级进程**"，是一个**基本的 CPU 执行单元**，也是**程序执行流**的**最小单元。**它可以**减小**程序在**并发执行**时所付出的**时空开销**，提高**操作系统**的**并发性能。**
- 线程**共享进程**拥有的**全部资源**，它**不拥有系统资源**，但是它可以**访问进程**所拥有的**系统资源。**线程没有自己独立的地址空间，它**共享它所属的进程的空间。**

5.2 线程的实现方式

线程的实现方式有两种，一种是**用户级**线程，另一种是**内核级**线程。

5.3 进程与线程的区别

1. 进程（Process）是**系统进行资源分配和调度**的基本单位，线程（Thread）是**CPU调度和分派**的基本单位；
2. 线程**依赖于进程**而存在，一个进程至少有一个线程；
3. 进程有自己的**独立地址空间**，线程共享所属进程的地址空间；
4. 进程是拥有**系统资源的一个独立单位**，而线程自己基本上不拥有系统资源，只拥有一点在运行中必不可少的资源和其他线程共享本进程的相关资源
5. 在**进程切换**时，涉及到整个**当前进程CPU环境**的**保存环境**的设置以及**新被调度运行的CPU环境**的设置，而**线程切换**只需**保存和设置少量的寄存器**的内容，并**不涉及存储器管理方面**的操作，可见，进程切换的开销远大于线程切换的开销；
6. 线程之间的**通信**更方便，同一进程下的**线程共享全局变量等数据**，而进程之间的通信需要以**进程间通信(IPC)的方式**进行；
7. **多线程程序**只要有**一个线程**崩溃，**整个程序**就崩溃了，但**多进程程序**中**一个进程崩溃**并**不会对其它进程造成影响**，因为进程有自己的独立地址空间，因此多进程更加健壮

5.4进程间的通信方式有哪些

1.共享内存（高级）

共享内存就是两个进程**同时共享一块内存**，然后在这块**内存上的数据**可以**共同修改和读取**，达到**通信**的目的。

2.无名管道（高级）

无名管道是**半双工**的通信方式；并且只能在具有**亲缘关系**的进程之间使用（亲缘关系是指进程间的父子关系，兄弟关系等），具有亲缘关系的进程在**创建时**同时拥有一个**无名管道的句柄**，可以进行**读写**；无名管道**不存在磁盘节点**，**只存在于内存**中**用完即销毁。**

3.命名管道（高级）

命名管道也是**半双工**的通信方式；可以在**不具有亲缘关系**的进程间通信；命名管道**存在磁盘节点**，**有对应的FIFO文件**，凡是可以**访问该路径的文件**的进程均可以进行通信。

4.消息队列（高级）

消息队列是**消息的链表**，存放在**内核**中并由**消息队列标识符**标识。消息队列克服了**信号传递信息少**、管道只能承载**无格式字节流**以及**缓冲区大小受限**等缺点。

5.套接字

套接字是**网络编程的api**，通过套接字可以**不同的机器间的进程**进行通信，常用于**客户端进程和服务器进程**的通信。

6.信号

操作系统通过信号来**通知进程系统**中发生了**某种预先规定好的事件**，它也是用户进程之间**通信和同步**的一种原始机制。

6 调度

6.1 调度的层次

作业调度，又称**高级**调度。就是**内存与辅存**之间的调度
中级调度，又称**内存**调度。引入中级调度是为了提高内存利用率和系统吞吐量，使那些**暂时不能运行的进程**，**调至外存等待**，把此时的进程状态称为挂起状态。当他们**具备运行条件**且**内存又稍有空闲**时，由中级调度来决定，把**外存**上的那些已具备运行条件的就绪进程再**重新调入内存**
进程调度，又称为**低级调度**。按照某种方法和策略从就绪队列中选取一个进程分配 CPU

6.2 调度算法

- **先来先服务调度算法**（FCFS）：调度**最先进入就绪队列**的作业 / 进程。有利于长作业 / 进程，但不利于短作业 / 进程。有利于ＣＰＵ繁忙型，不利于ＩＯ繁忙型
- **短作业优先调度算法**（SJF）：调度**估计运行时间最短**的作业。长作业有可能会饿死，处于一直等待短作业执行完毕的状态。
- **优先级调度算法**：为每个进程分配一个优先级，按**优先级**进行调度。为了防止低优先级的进程永远等不到调度，可以随着时间的推移增加等待进程的优先级。
- **高响应比优先调度算法**：把**响应比**作为优先权，其中响应比 = (等待时间 + 要求服务时间) / 要求服务时间 = 响应时间 / 要求服务时间，同时考虑了等待时间的长短和估计需要的执行时间长短，很好的平衡了长短进程。非抢占，无饥饿问题。
- **时间片轮转调度算法**：将所有**就绪进程按 FCFS 的原则**排成一个队列，**用完时间片**的进程排到**队列最后。**无饥饿问题，为短进程提供好的响应时间；若时间片小，进程切换频繁，吞吐量低；若时间片太长，实时性得不到保证。
- **多级反馈队列调度算法**：设置**多个就绪队列**1、2、3...，**优先级递减**，**时间片递增。**只有等到**优先级更高**的队列为空时才会**调度当前队列**中的进程。如果进程**用完了当前队列的时间片**还未执行完，则会被**移到下一队列。**

7 进程同步

7.1 临界资源与临界区

临界资源即**一次仅允许一个进程使用**的资源。

临界区即对**临界资源进行访问的代码。**

7.2 同步与互斥

- **同步**

  多个进程因为**合作**而使得进程的**执行**有**一定的先后顺序。**

- **互斥**

  多个进程在**同一时刻**只有**一个进程能进入临界区。**同步是在对临界区互斥访问的基础上，通过其它机制来实现有序访问的。

7 . 3 同步机制的 4 个原则

- **1.空闲让进**
  当无进程处于临界区，可允许一个请求进入临界区的进程立即进入自己的临界区
- **2.忙则等待**
  当已有进程进入自己的临界区，所有企图进入临界区的进程必须等待
- **3.有限等待**
  对要求访问临界资源的进程，应保证该进程能在有限时间内进入自己的临界区
- **4.让权等待**
  当进程不能进入自己的临界区，应释放处理机

7.3 信号量

它**允许多个线程**同一时刻访问同一资源，但是需要**限制**同一时刻访问此资源的**最大线程数目。**

信号量（semaphore）的数据结构为**一个值和一个指针**，指针指向等待该信号量的下一个进程。信号量的**值 S 与相应资源的使用情况**有关。当 S 大于 0 时，表示**当前可用资源的数量**；当 S 小于 0 时，其**绝对值表示等待使用该资源的进程个数。**注意，信号量的**值仅能由 PV 操作**来改变。

执行一次 **P 操作**意味着**请求分配一个单位资源**，因此S的值减1；当 S < 0 时，表示已经没有可用资源，请求者必须等待别的进程释放该类资源，它才能运行下去。

而执行一个 **V 操作**意味着**释放一个单位资源**，因此 S 的值加 1；若 S < =0，表示有某些进程正在等待该资源，因此要唤醒一个等待状态的进程，使之运行下去。

7.4 管程

管程是由一组**数据**以及定义在这组数据之上的对这组数据的**操作**组成的**软件模块**，这组**操作**能**初始化并改变管程中的数据**和**同步进程。**

在**一个时刻**只能有**一个进程使用管程。**进程在无法继续执行的时候不能一直占用管程，必须将进程阻塞，否者其它进程永远不能使用管程。

管程引入了**条件变量**以及相关的操作：**wait() 和 signal()** 来实现同步操作。对条件变量执行 wait() 操作会导致调用进程阻塞，把管程让出来让另一个进程持有。signal() 操作用于唤醒被阻塞的进程。

9 死锁

9.1 死锁的定义

多个进程因**竞争资源**而造成的一种僵局（互相等待），若无外力作用，这些进程都将无法向前推进。

9.2 死锁的原因

资源竞争；进程推进顺序不当

9.3 死锁产生的必要条件

互斥： 一个资源一次只能被一个进程所使用。

不剥夺： 一个资源仅能被占有它的进程所释放，而不能被别的进程强制剥夺。

请求与保持： 指进程占有自身本来拥有的资源并要求其他资源。

循环等待： 存在进程资源的循环等待链，链中每一个进程已获得的资源同时被下一个进程所请求。

9.4 怎么处理死锁?

即死锁的处理策略：可分为预防死锁、避免死锁和死锁的检测及解除。

9.4.1、预防死锁：破坏四个必要条件之一

这是一种较简单和直观的**事先预防**的方法。

- 1. **破坏互斥条件**

  即允许进程同时访问某些资源。但是，有的资源是不允许被同时访问的，像打印机等等。所以，这种办法并无实用价值。

- 2. **破坏不可剥夺条件**

  当一个进程已占有了某些资源，它又申请新的资源，但不能立即被满足时，它必须释放所占有的全部资源，以后再重新申请。

- 3. **破坏请求与保持条件**

  可以实行资源预先分配策略。即进程在运行前，一次性地向系统申请它所需要的全部资源。

- 4.**破坏循环等待条件**：实行顺序资源分配法

  首先给系统中的资源编号，规定每个进程，必须按编号递增的顺序请求资源，同类资源一次申请完。

9.4.2、避免死锁

该方法同样是属于**事先预防**的策略，在资源的动态分配过程中，用 某种方法去**防止系统进入不安全状态**，从而避免发生死锁。

银行家算法：

主要思想是**避免系统进入不安全状态**，在每次进行**资源分配**时，它首先**检查系统是否有足够的资源满足要求**，如果有，则**先试行分配**，并对分配后的**新状态进行安全性检查**。如果新状态安全，则正式分配上述资源，否则拒绝分配上述资源。这样就保证系统始终处于安全状态，从而避免死锁现象的发生。

安全序列：

是指系统能按**某种进程推进顺序**（P1, P2, P3, ..., Pn），为每个进程 Pi 分配其所需要的**资源**，直至**满足每个进程**对资源的最大需求，使每个进程都可以**顺序地完成。**这种推进顺序就叫**安全序列**【银行家算法的核心就是找到一个安全序列】。

系统安全状态：

如果系统能找到一个安全序列，就称系统处于安全状态，否则，就称系统处于不安全状态

9.4.3、死锁的检测和解除

即在死锁产生前不采取任何措施，只检测当前系统有没有发生死锁，若有，则采取一些措施解除死锁。

死锁的检测：

根据死锁定理：S 为死锁的条件是当且仅当 **S 状态的资源分配图**是**不可完全简化的**，该条件称为死锁定理。

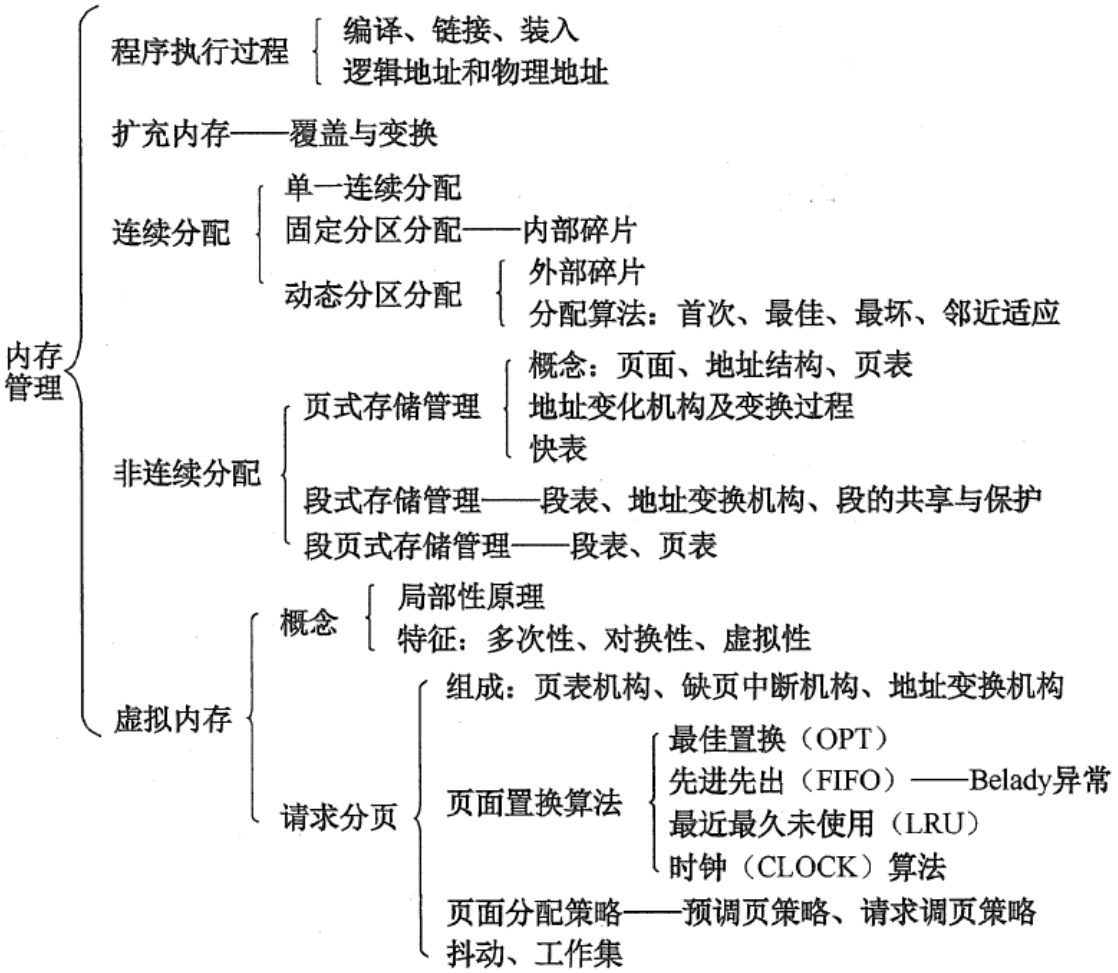
死锁的解除：

1、**资源剥夺**：挂起某些死锁进程，并抢占它的资源，将这些资源分配给其他死锁进程。（但应该防止被挂起的进程长时间得不到资源）；

2、**撤销进程**：强制撤销部分、甚至全部死锁进程并剥夺这些进程的资源；

3、**进程回退**：让一个或多个进程回退到足以避免死锁的地步。

10 存储器管理

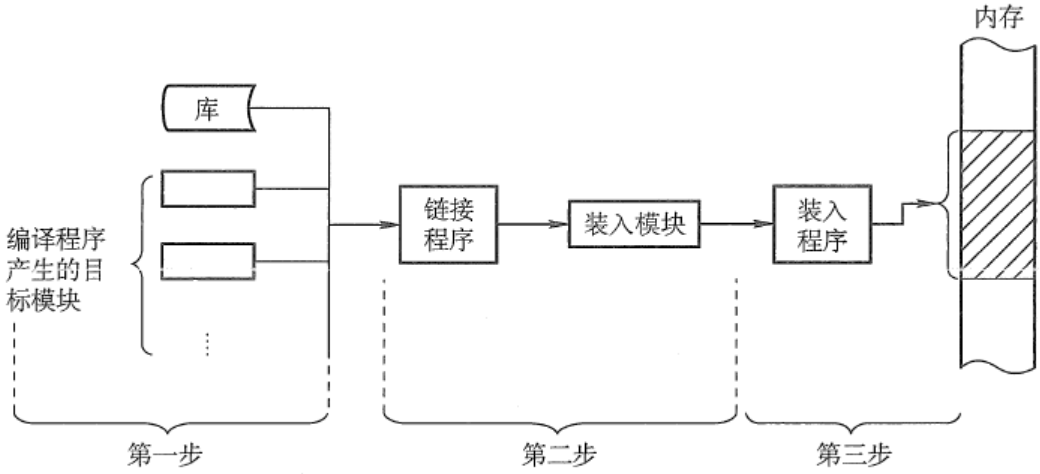


10.1存储器管理应具有的功能

- ① 内存的分配和回收：实施内存的分配，回收系统或用户释放的内存空间。
- ② 地址变换：提供地址变换功能，将逻辑地址转换成物理地址。
- ③ 扩充内存：借助于虚拟存储技术或其他自动覆盖技术，为用户提供比内存空间大的地址空间，从逻辑 上扩充内存。
- ④ 存储保护：保证进入内存的各道作业都在自己的存储空间内运行，互不干扰。

10.2 程序执行过程

- **编译**：由**编译程序**将**用户源代码**编译成若干个**目标模块**
- **链接**：由**链接程序**将编译后形成的**一组目标模块**，以及**所需库函数**链接在一起，形成一个完整的**装入模块**
- **装入**：由**装入程序**将**装入模块**装入**内存**运行



10.3 程序的链接方式

- ① **静态链接**：在**程序运行**之前，先把各个目标模块及所需库链接为一个完整的可执行程序，以后不再拆开。
- ② **装入时动态链接**：将应用程序编译后所得到的一组目标模块在**装入内存**时采**用边装入边链接**的链接方式。
- ③ **运行时动态链接**：直到**程序运行过程**中需要一些模块时，才对这些模块进行链接。

10.4 程序的装入方式

① **绝对装入**：在**编译时**就知道程序将要驻留在内存的**物理地址**，编译程序产生含有物理地址的目标代码，不适合多道程序设计。

② **可重定位装入**：根据内存当前情况，将装入模块装入到内存的适当位置，地址变换通常在**装入时**一次完成，之后不再改变，也称**静态重定位**。

③ **动态运行装入**：允许**程序运行时在内存中移动位置**，把**装入模块装入到内存后的所有地址都是相对地址**，在**程序执行过程**中每当访问到相应指令或数据时，才将要 访问的程序或数据的**相对地址转换为物理地址。动态重定位**的实现要依靠**硬件地址变换机构。**

10.5 逻辑地址与物理地址

逻辑地址：是指从**应用程序**角度看到的内存地址，又叫**相对地址。编译**后，每个目标模块都是从 0 号单元开始编址，称为该**目标模块的相对地址或逻辑地址。**不同进程可以有相同的逻辑地址，因为这些相同的逻辑地址可以映射到主存的不同位置。**用户和程序员**只需要知道逻辑地址。

物理地址：它是地**址转换的最终地址**，进程在**运行时**执行指令和访问数据最后都要通过**物理地址**从主存中**存取**，是**内存单元真正的地址。**

10.6 覆盖技术和交换技术

1.覆盖技术：

把一个**大的程序**划分为**一系列覆盖**，**每个覆盖**是一个**相对独立**的**程序单位**，把程序**执行时并不要求同时装入内存**的覆盖组成一组，成为**覆盖段**，这个覆盖段分配到同一个存储区域，这个存储区域成为**覆盖区**，它与覆盖段一一对应。覆盖段的大小由覆盖段中最大的覆盖来确定。

2.交换技术：

把**暂时不用**的某个**程序及数据部分**从**内存移到外存**中去，以便腾出必要的内存空间；或者把指定的程序或数据**从外存读到相应的内存**中，并将控制权交给他，让其在系统上运行的一种**内存扩充技术。**处理器的中级调度就是采用交换技术。

3.区别：

① 与覆盖技术相比，交换技术不要求程序员给出的 **程序段之间的覆盖结构**；

② **交换技术主**要在**进程和作业之间**进行，**覆盖技术**主要在**同一个进程或作业**中进行；

③ **覆盖技术**只能覆盖**与覆盖程序段无关的程序段**，**交换进程**由**换出和换入**两个过程组成。

１０.７ 虚拟内存

虚拟内存技术：允许将一个作业**分多次**调入内存。可以用分页式、分段式、段页式存储管理来实现。

基于局部性原理，在**程序装入**时，可以将程序的**一部分装入内存**，而其余部分留在**外存**，就可以启动程序执行。在程序执行过程中，当所访问的信息不在内存时，由**操作系统**将所需要的部分**调入内存**，然后继续执行程序。另一方面，操作系统将内存中**暂时不使用的内容**换出到外存上，从而腾出空间放入将要调入内存的信息。这样，系统就好像为用户提供了一个比实际内存大得多的存储器，称为**虚拟存储器。**

１０.８内存分配

10.８.1内存连续分配

1、**单一连续分配**：分为系统区和用户区，系统区供给操作系统使用，用户区供给用户使用，内存中永远只有一道程序。

2、**固定分区分配**：最简单的一种多道程序管理方式，它将**用户内存空间**划分为若干个**固定大小**的**区域**，**每个分区只装入一道作业。**【方法一：分区大小相等；方法二：分区大小不等】

3、**动态分区分配**：又称为**可变分区分配**，是一种**动态划分内存**的方法。这种分区方法不预先将内存划分，而是在进程**装入内存**时，根据进程的大小动态的建立分区，并使分区的大小正好适合进程的需要。因此系统中分区的大小和数目是可变的

4.**动态分区分配算法**

(1) **首次适应(First Fit)算法**：空闲分区以**地址递增**的次序链接。分配内存时顺序查找，找到大小能满足要求的第一个空闲分区。

(2) **最佳适应(Best Fit)算法**：空闲分区按**容量递增**形成分区链，找到第一个能满足要求的空闲分区。

(3) **最坏适应(Worst Fit)算法**：又称**最大适应(Largest Fit)算法**，空闲分区以**容量递减**的次序链接。找到第一个能满足要求的空闲分区，也就是挑选出最大的分区。

(4) **邻近适应(Next Fit)算法**：又称**循环首次适应算法**，由首次适应算法演变而成。不同之处是分配内存时从上次查找结束的位置开始继续查找。

10.８.2非连续分配

1、分页

用户程序的地址空间被划分为若干固定大小的区域，称为“页”。相应地，**内存空间**分成若干个物理块，页和块的大小相等。可将用户程序的任一页放在内存的任一块中，实现了离散分配，由一个**页表**来维护它们之间的映射关系。

2、分段

分段的做法是把**作业的地址空间**被划分为**若干个段**，每个段是**一组完整的逻辑信息。**每个段的长度可以不同，可以动态改变。

3、段页式

在段页式系统中，**作业的地址空间**首先被分成若**干个逻辑段**，每段都有自己的**段号**，然后将**每段**分成若干**固定大小的页。**对内存空间的管理仍然和分页存储管理一样，将其分成若干和页面大小相同的存储块，对内存的分配以**存储块**为单位。

4、分页与分段的区别

对程序员的透明性：分页透明，但是分段需要程序员显示划分每个段
地址空间的维度：分页是一维地址空间，分段是二维的
大小是否可以改变：页的大小不可变，段的大小可以动态改变
出现的原因：分页主要用于实现**虚拟内存**，从而获得更大的地址空间；分段主要是为了使**程序和数据**可以被划分为**逻辑上独立的地址空间**并且有助于**共享和保护**

１０.９页面置换算法

1、**最佳置换算法**（Opt）：即选择那些永不使用的，或者是在最长时间内不再被访问的页面置换出去。（它是一种理想化的算法，性能最好，但在实际上难于实现）；

2、**先进先出置换算法** FIFO：总是淘汰最先进入内存的页面；

3、**最近最久未使用置换算法** LRU（Least Recently Used）：即选择最近最久未使用的页面予以淘汰。

4、**时钟（Clock）置换算法**：该算法为每个页面设置一位**访问位**，将内存中的所有页面都通过链接指针链成一个**循环队列。**当某页被访问时，其访问位置“1”。在选择一页淘汰时，就检查其访问位，如果是“0”，就选择该页换出；若为“1”，则重新置为“0”，暂不换出该页，在循环队列中检查下一个页面，直到访问位为“0”的页面为止。

１０.１０局部性原理

时间局部性：如果程序中某条指令一旦执行，**不久后**该指令可能再次执行；如果某数据被访问过，不久后该数据可能再次被访问。【原因：因为在程序中存在着大量的循环操作】

空间局部性：一旦程序访问了某个存储单元，在不久之后，其**附近的存储单元**也将被访问。【原因：因为指令通常是顺序存放、顺序执行的；数据也一般是以向量、数组、表等形式簇聚存储的】

１０.１１页表和快表

页表指出逻辑地址中的**页号**与所占主存**块号**的对应关系。

作用：页式存储管理在用动态重定位方式装入作业时，要利用页表做地址转换工作。

快表就是存放在**高速缓冲存储器**的**部分页表。**

它起页表相同的作用。由于采用页表做地址转换，读写内存数据时CPU要访问两次主存。有了快表，有时只要访问一次高速缓冲存储器，一次主存，这样可加速查找并提高指令执行速度。

１０.１２地址翻译的过程

TLB->页表（TLB不命中）->Cache->主存（Cache不命中）->外存

10.7中断、系统调用、库函数

1、中断

就是指在计算机执行程序的过程中，由于出现了某些特殊事情，使得 CPU 暂停对程序的执行，转而去执行处理这一事件的程序。等这些特殊事情处理完之后再回去执行之前的程序。中断一般分为三类：

1.内部异常中断：由计算机硬件异常或故障引起的中断；

2.软中断：由程序中执行了引起中断的指令而造成的中断；

3.外部中断：由外部设备请求引起的中断；

2、系统调用

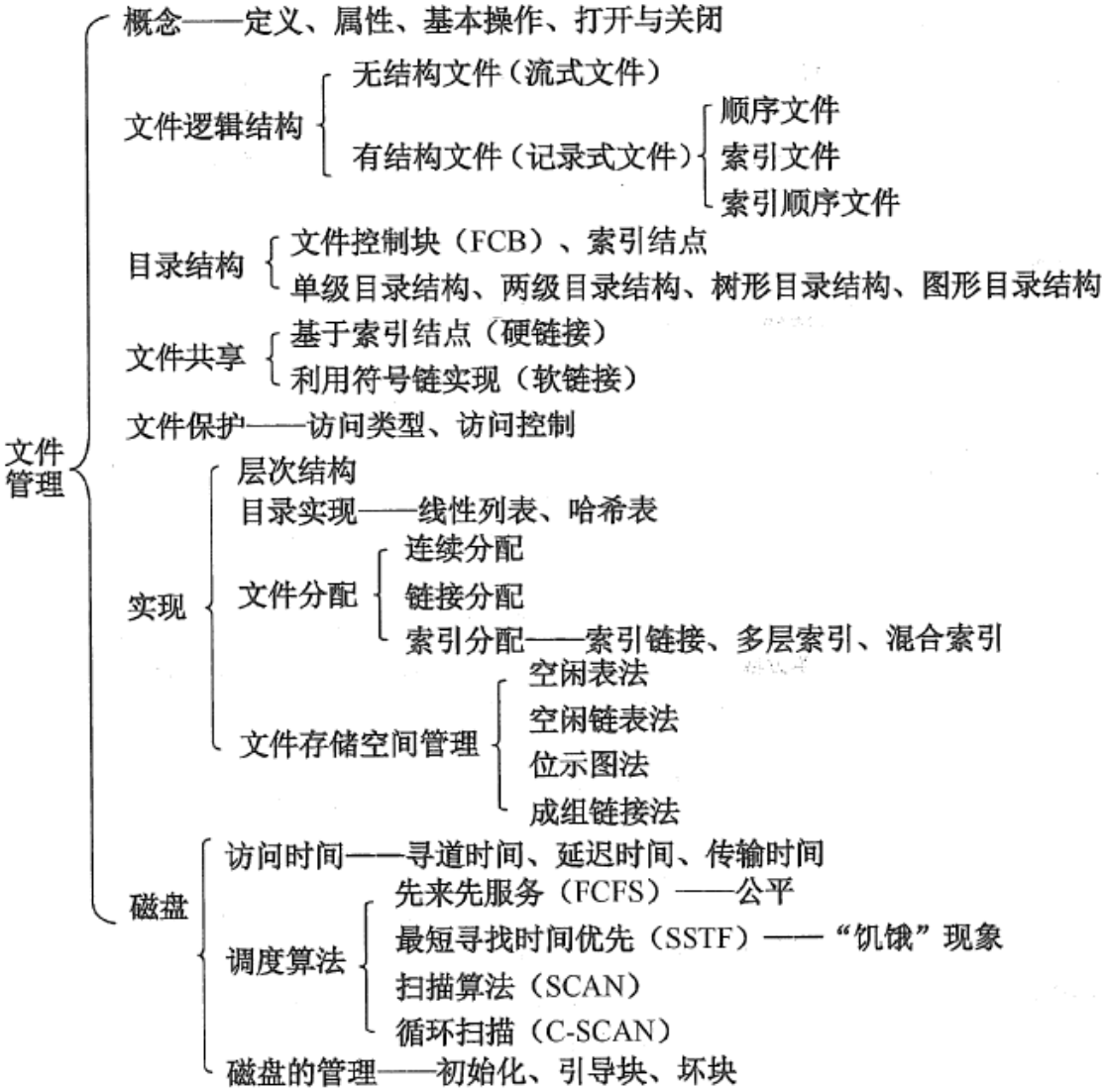
系统调用是通向操作系统本身的接口，是面向底层硬件的。用户进程需要发生系统调用时，内核将调用内核相关函数来实现用户程序不能直接调用这些函数，这些函数运行在内核态，CPU 通过软中断切换到内核态开始执行内核系统调用函数。

系统调用和中断的关系就在于，当进程发出系统调用申请的时候，会产生一个软中断。产生这个软中断以后，系统会去对这个软中断进行处理，这个时候进程就处于核心态了。

3、库函数

库函数是把函数放到库里，供别人使用的一种方式。【系统调用是为了方便使用操作系统的接口，而库函数则是为了人们编程的方便。】

11 文件管理



11.1 什么是文件

文件是以**计算机硬盘**为载体存储在计算机上的**信息集合**。

11.2 磁盘调度算法

当多个进程同时请求访问磁盘时，需要进行磁盘调度来控制对磁盘的访问。磁盘调度的主要目标是使磁盘的平均寻道时间最少。

先来先服务（FCFS）：根据进程请求访问磁盘的先后次序来进行调度。公平，简单，但由于未对寻道做任何优化，平均寻道时间可能较长
最短寻道时间优先（SSTF）：根据访问的磁道与当前磁头所在磁道距离最近的优先进行调度。该算法并不能保证平均寻道时间最短，但是比 FCFS 好很多，SSTF 会出现进行饥饿现象。
扫描（SCAN）：考虑了磁头的移动方向，要求所请求访问的磁道在磁头当前移动方向上才能够得到调度。因为考虑了移动方向，那么一个进程请求访问的磁道一定会得到调度。当一个磁头自里向外移动时，移到最外侧会改变移动方向为自外向里，这种移动的规律类似于电梯的运行，因此又常称 SCAN 算

法为电梯调度算法
循环扫描（CSCAN）：CSCAN 对 SCAN 进行了改动，要求磁头始终沿着一个方向移动。

１１.3 文件的基本操作

①创建文件：创建文件有两个必要步骤，一是在文件系统中为文件**找到空间**；二是在目录中为新文件**创建条目**，该条目记录文件**名称**、在文件系统中的**位置**及其他可能**信息。**

②写文件：为了写文件，执行一个**系统调用**，指明文件**名称**和要写入文件的**内容。**对于**给定文件名称**，系统搜索目录以**查找文件位置。**系统必须为该文件**维护**一个**写位置的指针。**每当发生写操作，便更新写指针。

③读文件：为了读文件，执行一个**系统调用**，指明文件名称和要读入文件块的内存位置。同样，需要搜索目录以找到相关目录项，系统**维护一个读位置**的指针。每当发生读操作时，更新读指针。

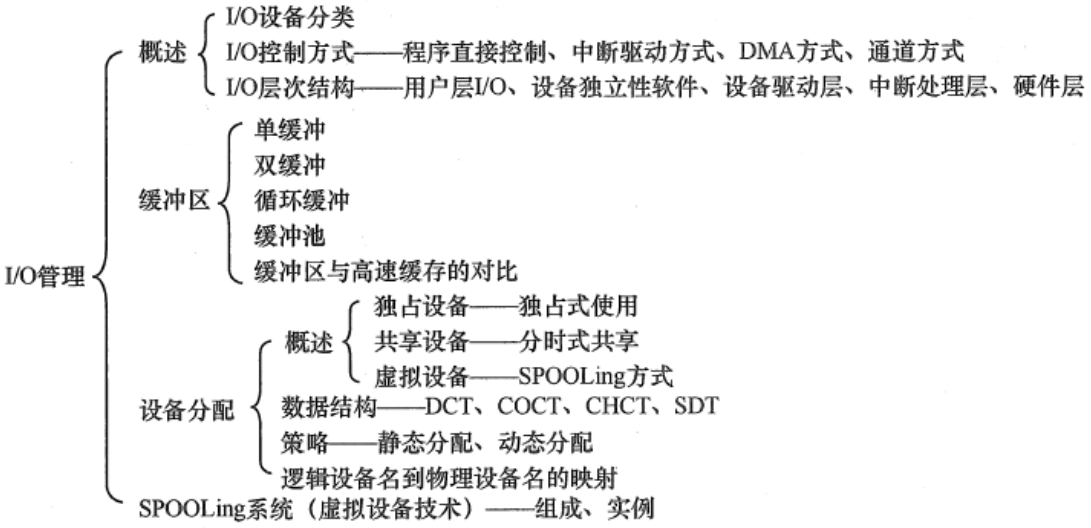
④文件重定位（文件寻址）：按某条件搜索目录，将当前文件位置设为给定值，并且不会读、写文件。

⑤删除文件：先从目录中找到要**删除文件的目录项**，使之成为空项，然后回收该文件所占用的**存储空间。**

⑥截断文件：允许文件所有**属性不变**，并**删除文件内**容，即将其长度设为0并释放其空间。

这6个基本操作可以组合执行其他文件操作。

12设备管理



12.1SPOOLing技术

SPOOling技术便可将一台**物理I/O设备**虚拟为多台**逻辑I/O设备**，同样允许多个用户**共享**一台物理I/O设备。SPOOLing技术是对脱机输入、输出系统的模拟。

ＳＰＯＯＬｉｎｇ系统主要部分

（1）输入井和输出井。这是在**磁盘**上开辟的两个大**存储空间。输入井**是**模拟脱机输入**时的磁盘设备，用于暂存I/Q设备**输入的数据**；**输出井**是**模拟脱机输出**时的磁盘，用于暂存用户程序的**输出数据。**

（2）输入缓冲区和输出缓冲区。为了缓和和**CPU和磁盘**之间速度不匹配的矛盾，在**内存**中要开辟两个**缓冲区**；输入缓冲区和输出缓冲区。**输入缓冲区**用于暂存由**输入设备送来的数据**，以后**再传送到输入井。**输出缓冲区用与暂存从输出井送来的数据，以后在传送给输出设备。

（3）输入进程SPi 和输出进程SP0。这里利用两个进程来**模拟脱机I/O时的外围控制机。**其中，进程SPi模拟脱机输入时的外围控制机，将用户要求的数据从输入机通过输入缓冲区再送到输入井，当CPU需要输入数据时，直接从输入井读入内存；进程SP0模拟脱机输出时的外围控制机，把用户要求输出的数据从先内存送到输出井，待输出设备空闲时，在将输出井中的数据经过输出缓冲区送到输出设备上。

SPOOLing技术的特点：

(1)提高了I/O速度。从对低速I/O设备进行的I/O操作变为对输入井或输出井的操作，如同脱机操作一样，提高了I/O速度，缓和了CPU与低速I/O设备速度不匹配的矛盾。

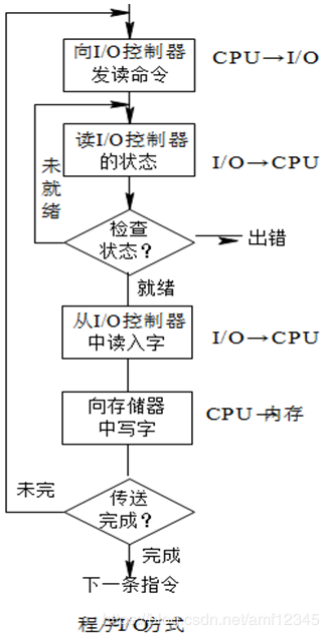
(2)将独占设备改造为共享设备。因为在SPOOLing系统的系统中，实际上并没为任何进程分配设备，而知识在输入井或输出井中为进程分配一个存储区和建立一张I/O请求表。这样，便把独占设备改造为共享设备。

(3)实现了虚拟设备功能。多个进程同时使用一独享设备，而对每一进程而言，都认为自己独占这一设备，从而实现了设备的虚拟分配。不过，该设备是逻辑上的设备。

１２.２ＩＯ控制方式

> **1.程序 I/O 方式**

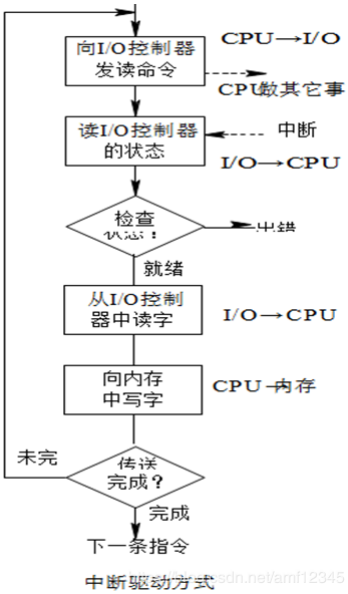
> 早期的计算机系统中， 没有中断系统，所以CPU和I/O设备进行通信，传输数据时CPU速度远快于I/O设备，于是**CPU需要不断测试I/O设备，看其是否完成了传输。**



程序I/O方式

> 2.**中断驱动方式**

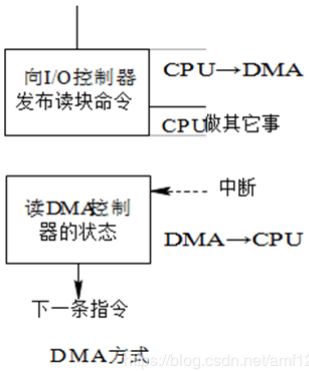
> 当某进程要**启动某个 I/O 设备工作**时，便由 CPU 向相应的**设备控制器**发出一条 **I/O 命令**，然后立即返回**继续执行原来的任务。**仅当**输完一个数据**时，才需 CPU 花费极短的时间去做些**中断处理。**



中断驱动方式

3.DMA方式（直接存储器访问）

通过在**I/O设备和内存**之间开启一个可以**直接传输数据**的**通路**，采用**DMA控制器**来控制一个**数据块的传输**，CPU只需在一个数据块传输开始阶段设置好传输所需的控制信息，并在传输结束阶段做进一步处理。



DMA方式

4.I/O通道控制方式

---- 通道控制方式与DMA控制方式类似，也是一种以**内存**为中心，实现**设备与内存直接交换数据**的控制方式。

---- 与DMA控制方式相比，**通道方式**所需要的**CPU干预更少**，而且可以做到**一个通道控制多台设备**，

从而进一步减轻了CPU负担。

---- **通道**本质上是一个**简单的处理器**，专门负责**输入、输出控制**，具有**执行I/O指令**的能力，并通过**执行通道I/O程序**来**控制I/O操作**。

---- 通道的指令系统比较简单，一般只有数据传送指令、设备控制指令等。