# 操作系统概论

## 操作系统的定义

计算机操作系统是指控制和管理计算机的软、硬件资源，合理组织计算机的工作流程，方便用户使用的程序集合。

## 操作系统的功能

进程管理、存储管理、设备管理、文件管理、用户接口



## 操作系统的特征（必考）

（1）并发(Concurrence)：在操作系统中同时存在许多活动。多个事件会在同一时间段内发生。

* + **Concurrence**，**并发**，两个或多个事件在同一时间间隔内发生。
  + **Parallel**，**并行**，两个或多个事件在同一时刻发生。
* 在多道程序处理时，宏观上并发，微观上交替执行（在单处理器情况下）。
* 并行是并发的一种特殊情况
* 并行必须有多个处理器
* 并发执行的条件：达到封闭性和可再现性

（2）共享(Sharing)：系统中的资源可供内存中多个并发执行的进程共同使用。

* 并发和共享是操作系统的两个最基本的特征，它们互为存在的条件。

（3）虚拟(Virtual)：通过某种技术把一个物理实体变为若干个逻辑上的对应物。

（4）异步(Asynchronism)：也称不确定性（ Nondeterminacy ），指进程的执行顺序和执行时间的不确定性。

## 操作系统的分类

### （1）批处理操作系统

* 优点：
  + 资源利用率高
  + 吞吐量大
* 缺点：
  + 用户交互性差，调试程序困难
  + 作业平均周转时间长

### （2）分时操作系统

分时：把计算机的系统资源（尤其是CPU时间）进行时间上的分割，每个时间段称为一个时间片（time slice），每个用户依次轮流使用时间片。

分时操作系统的特征

* 多路性：多个用户同时工作。也称为同时性。
* 独立性：各用户独立操作，互不干扰，感觉不到计算机为其它用户服务。
* 及时性：系统能及时对用户的操作进行响应。
* 交互性：分时系统的基本属性。

响应时间：从终端发出命令到系统处理完这条命令并给予回答所需的时间。

### （3）实时操作系统

在实时计算中，系统的正确性不仅仅依赖于计算的逻辑结果，而且依赖于结果产生的时间。

实时系统：能够在指定或者确定的时间内完成系统功能和对外部或内部、同步或异步时间做出响应的系统。

### （4）网络操作系统

网络操作系统是在通常操作系统功能的基础上提供网络通信和网络服务功能的操作系统。

### （5）分布式操作系统

分布式系统：以计算机网络为基础，处理和控制的分散（相对于集中式系统），基本特征是处理上的分布，即功能和任务的分布。

优点：处理能力增强、速度更快、可靠性增强

### （6）多处理机操作系统

### （7）嵌入式操作系统

以应用为中心、以计算机技术为基础、软件硬件可裁剪、适应应用系统，对功能、可靠性、成本、体积、功耗严格要求的专用计算机系统。

# 作业管理和用户接口

## 1. 用户与操作系统的接口

（1）作业级接口：操作系统为用户对作业运行全过程控制提供的功能。

（2）程序级接口：系统为用户在程序一级提供有关服务而设置，由一组系统调用命令组成。

## 2. 作业管理

### （1）作业的概念

用户在一次计算过程中或一次事务处理过程中，要求计算机系统所做工作的总称。

### （2）作业的组成

由程序、数据和作业说明书三部分组成。

### （3）作业的类型

#### 脱机作业------脱机用户接口------批处理方式

把对作业进行控制的意图连同程序和数据及发生故障后的处理措施一起输入系统中，由系统根据该意图来控制作业的执行。

作业控制块：

用来对作业进行描述的数据结构称为作业控制块（JCB）

作业控制块是批处理作业存在的标志

保存系统对于作业进行管理所需要的全部信息

#### 联机作业------联机用户接口------交互式方式

由一组操作系统命令实现，采用人机对话的方式来控制作业的运行。

## 3. 系统调用（必考）

系统调用是操作系统提供给软件开发人员的唯一接口，开发人员可利用它使用系统功能。

### （1）程序状态类型

管态与算态

* + 用户工作的状态——算态，用户态
  + 系统程序工作的状态——管态，系统态

特权指令与访管指令

* + 特权指令：只允许管态下使用的指令。
  + 访管指令：引起访管中断，实现从算态到管态的转变（设定程序状态字为管态），在管态下由操作系统代替用户完成操作，再由管态返回到算态。
* 用户程序在算态下运行——只能使用算态指令
* 操作系统是系统程序在管态下运行——可以使用算态指令，也可使用特权指令

### （2）系统调用及其功能

系统调用：至少包括一条访管指令在内的、由操作系统代替用户完成的某些特定功能的程序段。

* + 将系统转入管态，只能由汇编语言直接访问
  + 是操作系统提供给编程人员的唯一接口

功能：完成与硬件相关的工作以及控制程序的执行。

设备管理、文件管理、进程控制、进程通信、存储管理、系统管理、通过系统调用接口也可使用系统命令

### （3）系统调用的实现过程



### （4）系统调用与普通过程调用的相同点和不同点

系统调用是动态调用，而CALL调用方式是静态调用。

# 一、进程管理(最重要的部分)

## （一）进程的概念，线程的概念，PCB的概念

### 1. 进程

#### （1）进程的概念

* + 程序在处理机上执行时所发生的活动（Dijkstra）
  + 是一个容器，该容器用以聚集相关资源（A. S. Tanenbaum）
  + 是具有独立功能的程序关于某个数据集合上的一次运行活动，是系统进行资源分配和调度的独立单位

#### （2）进程的特征

动态性、并发性、独立性、异步性、交互性

#### （3）进程与程序的区别

* 程序是静态的，进程是动态的
* 进程与程序的组成不同，进程＝程序＋数据＋PCB
* 进程的存在是暂时的，程序的存在是永久的
* 一个程序可以对应多个进程，一个进程可以包含多个程序

### 2. 线程

#### （1）线程的引入

* 进程：资源分配单位（存储器、文件）和CPU调度（分派）单位。
  + 进程是拥有自己资源的单元体。
  + 进程是被调度分派在处理器上运行的单元体。
* 缺点：时间空间开销大，限制并发度的提高
* 线程：作为CPU调度单位，而进程只作为其他资源分配单位。
  + 只拥有必不可少的资源，如：线程状态、寄存器上下文和栈
  + 同样具有就绪、阻塞和执行三种基本状态

#### （2）线程的概念

##### 线程的概念

* + 线程是进程内一个相对独立的、可调度的执行单元。
  + 进程中的一个运行实体，是一个CPU调度单位
  + 资源的拥有者还是进程

##### 多线程机制

* + 一个进程可以有多个线程，这些线程共享进程资源，驻留在相同的地址空间（内存地址空间），共享数据和文件。
  + 一个线程修改了一个数据项，其他线程可以读取和使用此结果数据。一个线程打开并读一个文件时同一进程中的其他线程也可以同时读此文件。
  + 这些线程运行在同一进程的相同的地址空间内。

#### （3）引入线程的好处

* 创建一个新线程花费时间少（结束亦如此）
* 两个线程的切换花费时间少（如果机器设有**“**存储[恢复]所有寄存器**”**指令，则整个切换过程用几条指令即可完成）
* 因为同一进程内的线程共享内存和文件，因此它们之间相互通信无须调用内核
* 适合多处理机系统

#### （4）线程运行状态

线程运行状态：各线程之间也存在着共享资源和相互合作的制约关系，致使线程在运行时也具有间断性。

① 执行状态② 就绪状态③ 阻塞状态

### 3. 进程和线程

#### （1）进程和线程的比较

调度：

线程上下文切换比进程上下文切换要快得多。

线程的创建时间比进程短；终止时间比进程短。

同进程内的线程切换时间比进程短。

拥有资源：

进程间相互独立，同一进程的各线程间资源共享——某进程内的线程在其他进程不可见。

由于同进程内线程间共享内存和文件资源，可直接进行不通过内核的通信。

系统开销：

线程减小并发执行的时间和空间开销。

并发性：

在系统中建立更多的线程来提高并发程度。

#### （2）多线程OS中的进程

多线程OS中的进程有以下属性：

* + 作为系统资源分配的单位。
  + 可包括多个线程。
  + 进程不是一个可执行的实体。

通信：进程间通信IPC，线程间可以直接读写进程数据段（如全局变量）来进行通信—— 需要同步和互斥手段的辅助以保证数据的一致性。

#### （3）进程和线程

资源所有权（Process）：组织资源的实体单位

调度的实体（Thread）：调度、使用CPU的实体单位

#### （4）OS对线程的实现方式

* 核心级线程（内核线程，kernel-level thread）：由操作系统内核进行管理。操作系统内核给应用程序提供相应的系统调用和应用程序接口API，以使用户程序可以创建、执行、撤消线程。
* 用户级线程（用户线程，User-level thread）：管理过程全部由用户程序完成，操作系统内核心只对进程进行管理。

##### 内核线程KLT

* 依赖于OS核心，由内核的内部需求进行创建和撤销，用来执行一个指定的函数。
  + 内核维护进程和线程的上下文信息；
  + 线程切换由内核完成；
  + 一个线程发起系统调用而阻塞，不会影响其他线程的运行。
  + 时间片分配给线程，所以多线程的进程获得更多CPU时间。
  + Windows NT和OS/2支持内核线程。
* 优点：
  + 对多处理器，核心可以同时调度同一进程的多个线程
  + 阻塞是在线程一级完成
  + 核心例程是多线程的
* 缺点：
  + 在同一进程内的线程切换调用内核，导致速度下降

##### 用户线程ULT

* 用户级线程仅存在于用户空间中，线程的创建、撤消、线程之间的同步与通信等功能，都无须内核来实现。
  + 用户线程的维护由应用进程完成（通过线程库，用户空间，一组管理线程的过程）；
  + 内核不了解用户线程的存在；
  + 用户线程切换不需要内核特权；
  + 用户线程调度算法可针对应用优化。
* 线程库
  + 创建、撤销线程；在线程之间传递消息和数据；调度线程执行；保护和恢复线程上下文
* 优缺点

优点：

* + 线程切换不调用核心
  + 调度是应用程序特定的：可以选择适合的算法
  + ULT可运行在任何操作系统上（只需线程库）

缺点：

* + 大多数系统调用是阻塞的，因此核心阻塞进程，故进程中所有线程将被阻塞
  + 核心只将处理器分配给进程，同一进程中的两个线程不能同时运行于两个处理器上
* 用户级线程的实现
  + 运行时系统(Runtime System)：用于管理和控制线程的函数(过程)的集合，其中包括用于创建和撤消线程的函数、 线程同步和通信的函数以及实现线程调度的函数等。
  + 内核控制线程，又称为轻量级进程LWP(Light Weight Process)：每一个进程都可拥有多个LWP，可通过系统调用来获得内核提供的服务，这样，当一个用户级线程运行时，只要将它连接到一个LWP上，此时它便具有了内核支持线程的所有属性。

### 4. 作业、程序、进程和线程的比较

* 作业——用户向计算机提交任务的任务实体
* 程序——一组有序的指令集合
* 进程——系统分配资源的基本单位
* 线程——处理机调度的基本单位

### 5. PCB的概念

系统为了管理进程设置的一个专门的数据结构，用来记录进程的外部特征，描述进程的变化过程。

## （二）进程的状态管理&进程的调度

### 1. 进程控制块PCB（Process Control Block）

#### （1）基本概念

* + 系统为了管理进程设置的一个专门的数据结构，用来记录进程的外部特征，描述进程的变化过程
  + 进程的组成： program+data+PCB
  + PCB是系统感知进程存在的唯一标志，进程与PCB是一一对应的

#### （2）PCB的内容

* 进程描述信息：
  + 进程标识符**(process ID)**，唯一，通常是一个整数
  + 进程名，通常基于可执行文件名（不唯一）
  + 用户标识符**(user ID)**；进程组关系
* 进程控制信息：
* 所拥有的资源和使用情况：
* CPU现场保护结构：寄存器值

#### （3）PCB是进程存在的唯一标志？

* 1）包含了进程的描述信息和控制信息
* 2）是进程的动态特征的集中反映
* 3）系统根据PCB而感知某一进程的存在

#### （4）PCB表组织

* PCB表：
  + 系统把所有PCB组织在一起，并把它们放在内存的固定区域，就构成了PCB表
  + PCB表的大小决定了系统中最多可同时存在的进程个数，称为系统的并发度

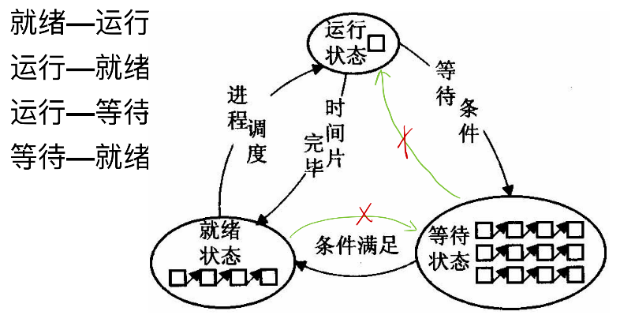
组织方式：链表、索引表

### 2. 进程的状态

#### （1）三种最基本的进程状态

* 运行状态（Running），进程占有CPU，并在CPU上运行
* 就绪状态（Ready），一个进程已经具备运行条件，但由于无CPU暂时不能运行的状态（当调度给其CPU时，立即可以运行）
* 等待状态（阻塞状态，Blocked），阻塞态、封锁态、睡眠态，指进程因等待某种事件的发生而暂时不能运行的状态（即使CPU空闲，该进程也不可运行）

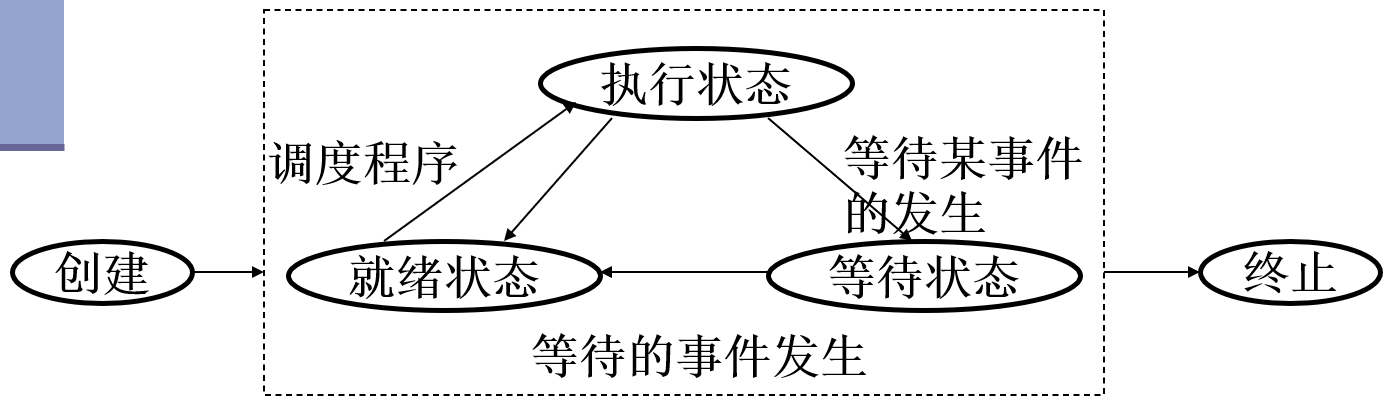
#### （2）进程状态转换（图会画，记住状态）



#### （3）五状态进程

增加：

* + 创建状态(New)：进程刚创建，但还不能运行如：分配和建立PCB表项、建立资源表格并分配资源，加载程序并建立地址空间表。
  + 结束状态(Exit)：进程已结束运行，回收除PCB之外的其他资源，并让其他进程从PCB中收集有关信息。



#### （4）挂起进程模型

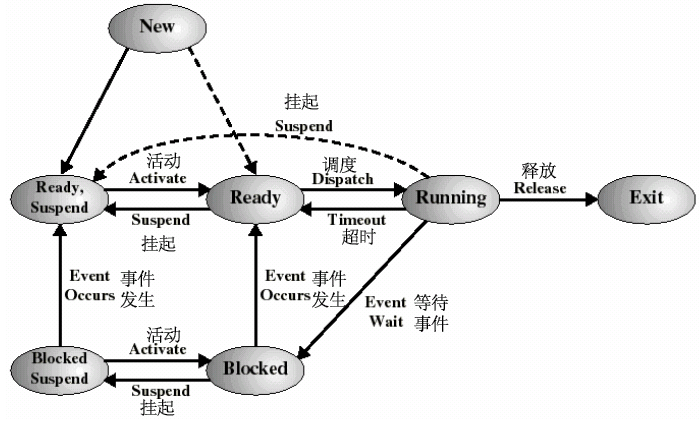
挂起（Suspend）：一些低优先级进程可能等待较长时间而被对换至外存，为运行进程提供足够内存。

* + 阻塞挂起（Blocked, suspend）：进程在外存并等待某事件的出现。
  + 就绪挂起（Ready, suspend）：进程在外存，但只要进入内存，即可运行。

#### （5）七状态进程

挂起（Suspend）：把一个进程从内存转到外存

激活（Activate）：把一个进程从外存转到内存



### 3. 进程的控制

#### （1）基本概念

* 创建、撤消进程以及完成进程各状态之间的转换，由具有特定功能的原语完成
* “原语” 是由若干条机器指令构成、完成一种特定功能的程序段；这段程序在执行期间不允许被分割，必须一次执行完。
* 进程控制原语：
  + 进程创建原语
  + 进程撤消原语
  + 进程状态转换原语

#### （2）进程的建立

* 何时创建：
  + 用户登录、 作业调度、提供服务、应用请求
* 进程创建的基本过程
  + 首先从空闲的PCB集合中申请一个新的PCB，同时获得该进程的内部标识；
  + 然后向该PCB中填写各种参数；
  + 把该进程的状态设置成就绪状态，并将该PCB插入到就绪队列中。

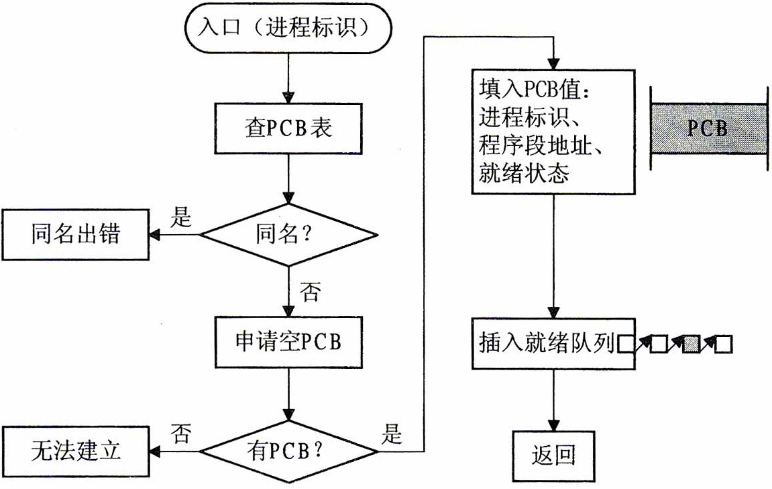
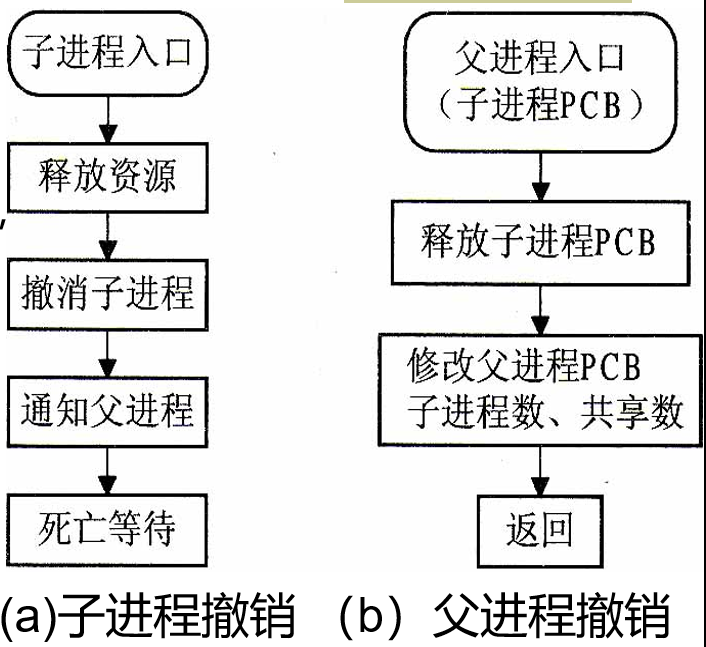


图 1 进程创建原语

#### （3）进程的撤销

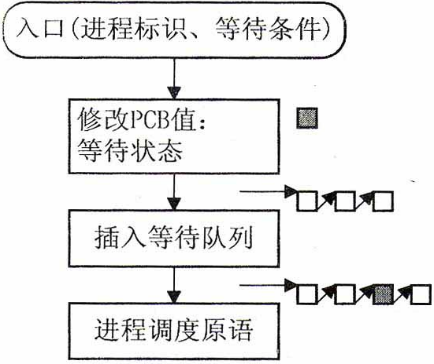
* 引起进程终止(Termination of Process)的事件
  + 正常结束
  + 异常结束
  + 外界干预
* 撤消进程的两种策略：
  + 撤消指定进程
  + 撤消该进程及其所有子孙进程
* 进程终止的基本过程
* 找到相应进程的PCB；
* 若进程正处于执行状态，则立即停止，设置重新调度标志；
* 撤消属于该进程的所有“子孙”进程；
* 释放被撤消进程的所有资源；
* 释放进程的PCB；
* 若调度标志为真，则进行重新调度



#### （4）进程的状态转换原语

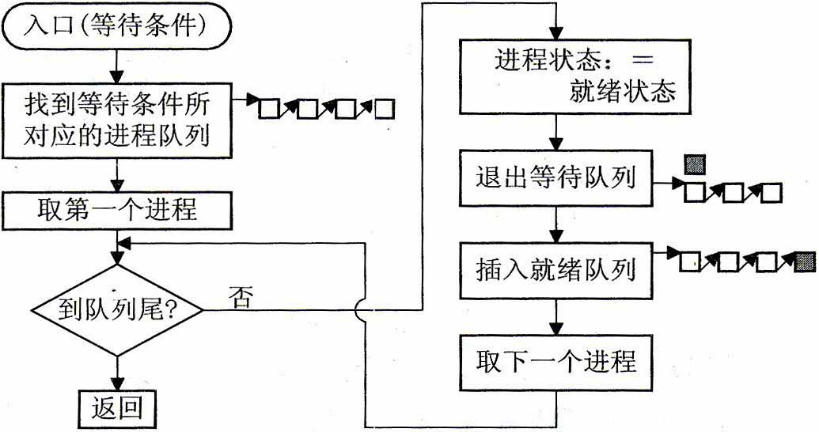
##### 进程的状态转换原语——等待

* 进程的等待
  + 在进程的运行过程中, 如果申请某一种条件而没有被满足, 进程不得不中止当前的运行, 进程等待原语就会被激活
* 进程等待原语
  + 使调用该原语的进程变为等待状态；
  + 将指定的进程变为等待状态；
  + 将某进程及其所有子孙进程变为等待状态。
* 进程等待的基本过程
* 找到相应进程的PCB；
* 如果该进程为执行状态，则保护其现场，将其状态改变为等待状态，停止运行，并把该PCB插入到相应的等待队列中去；
* 若为就绪状态，则将其状态修改为等待状态，把它移出就绪队列，并插入到等待队列中去



##### 进程的状态转换原语——唤醒

* 进程因等待某事件的发生而处于等待状态，当等待事件发生后，就要用唤醒原语将其唤醒。
* 唤醒原语的基本操作：
  + 在等待队列中找到相应进程的PCB，将其从等待队列中移出；
  + 置其状态为就绪状态，然后把该PCB插入就绪队列中；
  + 等待调度程序调度。

****

##### 进程调度原语

找到就绪队列的首指针, 按照调度算法所规定的选择原则（比如优先级法）选中一个进程, 将该进程的PCB中的状态由就绪状态改变为运行状态, 然后使其退出就绪队列, 恢复该进程的现场参数, 该进程便进入运行状态。

##### 进程挂起原语

检查被挂起进程的状态，若处于活动就绪状态，便将其改为静止就绪；对于活动阻塞状态的进程，则将之改为静止阻塞。 为了方便用户或父进程考查该进程的运行情况而把该进程的PCB复制到某指定的内存区域。

##### 进程激活原语

先将进程从外存调入内存，检查该进程的现行状态，若是静止就绪，便将之改为活动就绪；若为静止阻塞便将之改为活动阻塞。假如采用的是抢占调度策略，则每当有新进程进入就绪队列时，应检查是否要进行重新调度，即由调度程序将被激活进程与当前进程进行优先级的比较，如果被激活进程的优先级更低，就不必重新调度；否则，立即剥夺当前进程的运行，把处理机分配给刚被激活的进程。

##### Linux中的进程控制原语

* 在Linux系统中, 进程控制的原语有:
  + 进程建立fork、
  + 进程监控ps、
  + 进程优先级的确定nice、
  + 进程等待lock、
  + 进程唤醒wakeup、
  + 进程终止kill等。
* 在Linux中, 系统引导时会自动建立一个进程, 称为进程0, 这个进程是所有进程的祖先, 负责完成进程的调度。然后进程0建立自己的子进程: 进程1。除进程1外, 进程0将建立其他许多与系统管理有关的进程。

### 4. 进程的调度

#### （1）基本概念

##### 目标

高CPU的利用率、大吞吐量、快响应时间

##### 调度的类型

处理机调度可以分成三级：

* + 高级调度（调度对象：作业）
  + 中级调度（调度对象：进程）
  + 低级调度（微观调度，低级调度，Ready <－>Running）

##### 进程调度的任务

控制协调进程对CPU的竞争，即按一定的调度算法从就绪队列中选中一个进程，把CPU的使用权交给被选中的进程。

##### 调度的性能准则

* + 面向用户的调度性能准则：周转时间、响应时间、截止时间、公平性、优先级
  + 面向系统的调度性能准则：吞吐量、处理机利用率、各种设备的均衡利用
  + 调度算法本身的调度性能准则：易于实现、执行开销比

##### 进程调度的时机

* 现运行进程完成任务正常结束或因出现错误异常结束；
* 时间片到 (按时间片运行)；
* 进程提出I/O请求 ––– 阻塞，调新进程；
* 执行原语操作，进入阻塞状态；
* 具有更高优先级的进程进入就绪队列，要求使用处理机（可剥夺调度）。

##### 进程调度的方式

###### 非剥夺调度(nonpreemptive scheduling)

某一进程被调度运行后，除非由于它自身的原因不能运行，否则一直运行下去。

###### 剥夺调度(preemptive scheduling)

当有比正在运行的进程优先级更高的进程就绪时，系统可强行剥夺正在运行进程的CPU，提供给具有更高优先级的进程使用。

#### （2）调度算法

##### 先来先服务（FCFS, First Come First Service）

* 按照作业提交或进程变为就绪状态的先后次序分派CPU；
* 当前作业或进程占用CPU，直到执行完或阻塞，才出让CPU（非抢占方式）；在作业或进程唤醒后（如I/O完成），并不立即恢复执行，通常等到当前作业或进程出让CPU。
* 特点：
  + 比较有利于长作业，而不利于短作业。
  + 有利于CPU繁忙的作业，而不利于I/O繁忙的作业。
  + 最简单的算法。

##### 最短作业优先（SJF, Shortest Job First）

* 对预计执行时间短的作业（进程）优先分派处理机。通常后来的短作业不抢先正在执行的作业。
* 又称为“最短进程优先”SPN（ Shortest Process Next ）
* SJF对于给定的进程集合，平均周转时间最小

###### SJF的特点

优点：

* + 比FCFS改善平均周转时间和平均带权周转时间
  + 提高系统的吞吐量；

缺点：

* + 对长作业非常不利，可能长时间得不到执行；
  + 未能依据作业的紧迫程度来划分执行的优先级；
  + 难以准确估计作业（进程）的执行时间，影响调度性能。

###### SJF的变形

最短剩余时间优先，SRT，Shortest Remaining Time

* + 允许比当前进程剩余时间更短的进程来抢占

最高响应比优先HRRN（Highest Response Ratio Next）

* + 响应比R = (等待时间 + 要求执行时间) / 要求执行时间 = 1 + 等待时间/要求执行时间
  + 是FCFS和SJF的折衷

##### 时间片轮转算法（RR, Round Robin）

###### 基本思路

* + 通过时间片轮转，提高进程并发性和响应时间特性，从而提高资源利用率；

###### 执行过程

* + 将系统中所有的就绪进程按照FCFS原则，排成一个队列。
  + 每次调度时将CPU分派给队首进程，让其执行一个时间片。时间片的长度从几个ms到几百ms。
  + 在一个时间片结束时，发生时钟中断。调度程序暂停当前进程的执行，将其送到就绪队列的末尾，并通过上下文切换执行当前的队首进程。
  + 进程可以未使用完一个时间片，就出让CPU（如阻塞）。

###### 时间片长度的确定

* 时间片长度变化的影响
  + 过长－>退化为FCFS算法，进程在一个时间片内都执行完，响应时间长。
  + 过短－>用户的一次请求需要多个时间片才能处理完，上下文切换次数增加，响应时间长。
* 对响应时间的要求：
  + T(响应时间)=N(进程数目)\*q(时间片)
* 时间片长度的影响因素：
  + 就绪进程的数目：数目越多，时间片越小（当响应时间一定时）
  + 系统的处理能力：应当使用户输入通常在一个时间片内能处理完，否则使响应时间，平均周转时间和平均带权周转时间延长。

##### 基于优先级的调度算法（Priority Scheduling）

###### i. 基本思想

* + 系统为每个进程设置一个优先数（对应一个优先级），把所有的就绪进程按优先级从大到小排序，调度时从就绪队列中选择优先级最高的进程投入运行，仅当占用CPU的进程运行结束或因某种原因不能继续运行时，系统才进行重新调度 。

ii. 剥夺方式

* + 非剥夺（抢占）的优先级调度法(一般的)
  + 可剥夺（抢占）的优先级调度法

###### iii. 优先级的类型

* 静态优先级：创建进程时就确定，直到进程终止前都不改变。通常是一个整数。依据：
  + 进程类型（系统进程优先级较高）
  + 对资源的需求（对CPU和内存需求较少的进程优先级较高）
  + 用户要求（紧迫程度和付费多少）
* 动态优先级：在创建进程时赋予的优先级，在进程运行过程中可以自动改变，以便获得更好的调度性能。如：
  + 在就绪队列中等待时间延长则优先级提高，使优先级较低的进程在等待足够的时间后，其优先级提高到可被调度执行；
  + 进程每执行一个时间片，就降低其优先级，从而一个进程持续执行时，其优先级降低到出让CPU。

##### 多级队列算法（Multiple-level Queue）

* 本算法引入多个就绪队列，通过各队列的区别对待，达到一个综合的调度目标。
* 基本思想：
  + 根据作业或进程的性质或类型的不同，将就绪队列再分为若干个子队列。
  + 每个作业固定归入一个队列。
  + 各队列不同处理：不同队列可有不同的优先级、时间片长度、调度策略等。如：系统进程、用户交互进程、批处理进程等。

##### 多级反馈队列算法（Multiple-level Queue）

###### i. 多级反馈队列算法是时间片轮转算法和优先级算法的综合和发展

* 优点：
  + 为提高系统吞吐量和缩短平均周转时间而照顾短进程
  + 为获得较好的I/O设备利用率和缩短响应时间而照顾I/O型进程
  + 不必估计进程的执行时间，动态调节
* 优先级分组法
  + 保留非剥夺式优先级和剥夺式优先级各自的优点，克服其缺点。
  + 方法：组间可剥夺，组内不可剥夺（组内相同优先级则按FCFS处理）

###### ii. 基本思想

* 设置多个就绪队列，分别赋予不同的优先级，如逐级降低，队列1的优先级最高。每个队列执行时间片的长度也不同，规定优先级越低则时间片越长，如逐级加倍
* 新进程进入内存后，先投入队列1的末尾，按FCFS算法调度；若按队列1一个时间片未能执行完，则降低投入到队列2的末尾，同样按FCFS算法调度；如此下去，降低到最后的队列，则按"时间片轮转"算法调度直到完成。
* 仅当较高优先级的队列为空，才调度较低优先级的队列中的进程执行。如果进程执行时有新进程进入较高优先级的队列，则抢先执行新进程，并把被抢先的进程投入原队列的末尾。

###### iii. 几点说明

* 时间片的变化，进入更低级队列。最终采用最大时间片来执行，减少调度次数。
* I/O次数不多，而主要是CPU处理的进程： I/O型进程：让其进入最高优先级队列，以及时响应I/O交互。通常执行一个小时间片，要求可处理完一次I/O请求的数据，然后转入到阻塞队列。
* 计算型进程：每次都执行完在I/O完成后，放回优先I/O请求时离开的队列，以免每次都回到最高优先级队列后再逐次下降。
* 为适应一个进程在不同时间段的运行特点，I/O完成时，提高优先级；时间片用完时，降低优先级；

###### iv. 特点

* + 短作业优先。
  + 输入/输出进程优先。
  + 运算型进程有较长的时间片。
  + 采用了动态优先级, 使用珍贵资源CPU的进程优先级不断降低。 采用了可变时间片以适应不同进程对时间的要求, 运算型进程将获得较长的时间片。

#### （3）实时调度

##### 实时调度算法的分类

###### i. 非抢占式调度算法

* + 非抢占式轮转调度算法（具有一定的不确定性 最长等待时间：n\*T）
  + 非抢占式优先调度算法

###### ii. 抢占式调度算法

* + 基于时钟中断的抢占式优先权调度算法
  + 立即抢占(Immediate Preemption)的优先权调度算法（等待时间最短）



* (a)(b)适合软实时任务

##### 优先级反转

i. 优先级反转：髙优先级的进程必须等待低优先级的进程完成

ii. 优先级继承：所有使用到髙优先级进程所需资源的进程，继承髙优先级直到用完竞争资源，再回到原来的优先级。

##### 常用实时调度算法

###### i. 静态表调度算法（Static table-driven scheduling）

适用于周期性的实时应用。通过对所有周期性任务的分析预测（到达时间、运行时间、结束时间、任务间的优先关系），事先确定一个固定的调度方案。这种方法的特点是有效但不灵活。

###### ii. 静态优先级调度算法（Static priority-driven scheduling）

把通用的优先级调度算法用于实时系统，但优先级的确定是通过静态分析（运行时间、到达频率）完成的。

###### iii. 动态分析调度算法（Dynamic planning-based scheduling）

在任务下达后执行前进行调度分析，要求满足实时性要求。

iv. 无保障动态调度算法（Dynamic best effort scheduling）

在任务下达时分配优先级，开始执行，在时限到达时未完成的任务被取消。用于非周期性任务的实时系统。

##### 常用的几种实时调度算法

###### i. 最早截止期优先EDF（ Earliest Deadline First ）算法



EDF算法用于非抢占调度方式

###### ii. 最低松弛度优先即LLF（ Least Laxity First ）算法

* + 根据任务紧急（或松弛）的程度，来确定任务的优先级；
  + 实现该算法时要求系统中有一个按松弛度排序的实时任务就绪队列，松弛度最低的任务排在队列最前面，调度程序总是选择就绪队列中的队首任务执行；
  + 主要用于可抢占调度方式中。

#### （4）多处理机调度（有多个处理器）

##### 与单处理机调度的区别

* 注重整体运行效率（而不是个别处理机的利用率）
* 更多样的调度算法
* 多处理机访问OS数据结构时的互斥（对于共享内存系统）
* 调度单位广泛采用线程

##### 对称式多处理系统(SMP)

按控制方式，SMP调度算法可分为： 集中控制 & 分散控制

###### i. 集中控制

* + 静态分配(static assignment)：每个CPU设立一个就绪队列，进程从开始执行到完成，都在同一个CPU上。
    - 优点：调度算法开销小。
    - 缺点：容易出现忙闲不均。
  + 动态分配(dynamic assignment)：各个CPU采用一个公共就绪队列，队首进程每次分派到当前空闲的CPU上执行。

###### ii. 分散控制

* + 自调度(self-scheduling)：各个CPU采用一个公共就绪队列，每个处理机都可以从队列中选择适当进程来执行。需要对就绪队列的数据结构进行互斥访问控制。
    - 最简单的一种调度方式，直接由单处理机环境下的调度方式演变而来
    - 系统中的公共就绪队列可按照单处理机系统中所采用的各种方式加以组织； 其调度算法也可沿用单处理机系统所用的算法
    - 只要系统中有任务，只要公共就绪队列不空，就不会出现处理机空闲的情况，也不会发生处理器忙闲不均的现象，有利于提高处理器的利用率。
  + 优点：简单，处理机利用率高

##### 非对称式多处理系统(ASMP)

* 主－从处理机系统，由主处理机管理一个公共就绪队列，并分派进程给从处理机执行。
* 从机空闲时， 便向主机发送一索求进程的信号， 然后， 便等待主机为它分配进程。
* 各个处理机有固定分工，如执行OS的系统功能，I/O处理，应用程序。

##### 成组调度(gang scheduling)

将一个进程中的一组线程，每次分派时同时到一组处理机上执行，在剥夺处理机时也同时对这一组线程进行。

* 优点
  + 通常这样的一组线程在应用逻辑上相互合作，成组调度提高了这些线程的执行并行度，有利于减少阻塞和加快推进速度，最终提高系统吞吐量。
  + 每次调度可以完成多个线程的分派，在系统内线程总数相同时能够减少调度次数，从而减少调度算法的开销。

##### 专用处理机调度(dedicated processor assignment)

* 为进程中的每个线程都固定分配一个CPU，直到该线程执行完成。
  + 缺点：线程阻塞时，造成CPU的闲置。
  + 优点：线程执行时不需切换，相应的开销可以大大减小，推进速度更快。
* 适用场合：CPU数量众多的高度并行系统，单个CPU利用率已不太重要。

#### （5）必考

1. 有五个待运行的作业，估计它们的运行时间分别是：9，6，3，5和X。采用哪种次序运行这些作业将得到最短的平均响应时间？（答案将依赖于X）

2. 有五个批处理作业A~E，他们几乎同时到达一个计算中心。估计它们的运行时间分别为10，6，2，4和8分钟，其优先级由外部设定，分别为3，5，2，1和4，其中5为最高优先级。对于下列每种调度算法，计算其平均进程周转时间。忽略进程切换开销。

* + 轮转法（时间片粒度为2分钟）
  + 优先级调度
  + 先来先服务（按照10，6，2，4，8次序）
  + 最短作业优先

## （三）进程同步和互斥

### 1. 进程间的制约关系

* 直接制约关系：进行协作－－等待来自其他进程的信息，“同步”
* 间接制约关系：进行竞争－－独占分配到的部分或全部共享资源，“互斥”
* 进程间这种相互依赖又相互制约，相互合作又相互竞争的关系表现为同步和互斥两个方面。
* 同步和互斥机制的主要任务：**维持进程并发性**，**以提高系统效率**

### 2. 进程间的同步

#### （1）同步是一种什么关系？

* 进程同步: 指进程之间的一种协调配合关系, 它表现在进程的执行顺序的规定上。
* 概念：相互协调的几个进程在某些确定点上协调它们的工作，一个进程到达了这些点后，除非另一进程已完成了某些操作，否则就需要停下来等待这些操作的完成。

#### （2）进程同步的传送消息实现

* 如果对一个事件或消息赋以唯一的消息名，则过程wait (消息名）表示进程等待合作进程发来消息，功能是等待到消息名为true的进程继续执行；
* 过程signal (消息名）表示向合作进程发送消息，功能则是向合作进程发送所需要的消息名，并将其值置为true。

### 3. 进程间的互斥

#### 同步里边的互斥是一种什么关系？

* 进程互斥：两个或两个以上的进程由于不能同时使用同一资源，只能一个进程使用完了另一个进程才能使用的现象。
* 互斥关系也是一种协调关系, 从广义上讲它也属于同步关系的范畴。

### 4. 临界资源及其访问过程

#### 临界资源与临界区（必考）

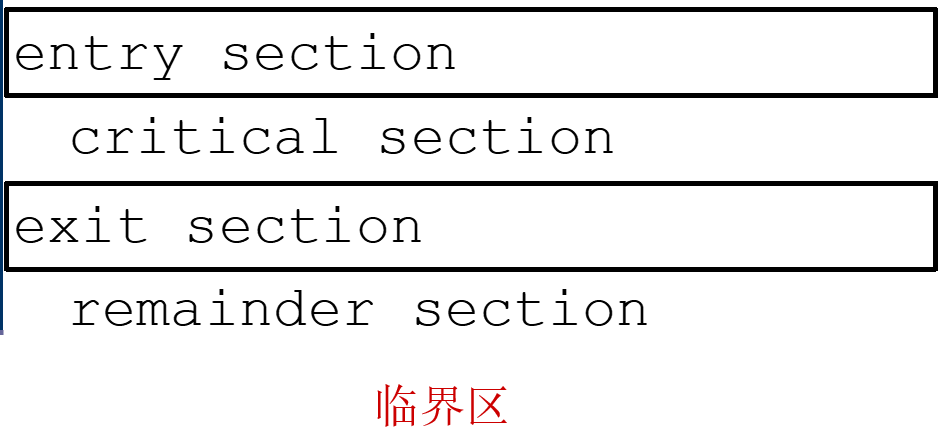
##### 什么叫临界资源？

* 临界资源(critical resource)：一次仅允许一个进程访问的资源。（打印机）
  + 一次只允许一个进程访问的资源
  + 资源状态为临界：0 或 1

##### 什么叫临界区？

* 临界区(critical section)：临界段，在每个程序中，访问临界资源的那段程序。
* 对临界段的设计有如下原则：
  + 每次至多只允许一个进程处于临界段中。
  + 对于请求进入临界段的多个进程, 在有限时间内只让一个进入。 
  + 进程只应在临界段中停留有限时间。

##### 临界区的访问过程



* 临界区(critical section)：进程中访问临界资源的一段代码。
* 进入区(entry section)：在进入临界区之前，检查可否进入临界区的一段代码。如果可以进入临界区，通常设置相应"正在访问临界区"标志
* 退出区(exit section)：用于将"正在访问临界区"标志清除。
* 剩余区(remainder section)：代码中的其余部分。

### 5. 同步机制应遵循的准则（必考）

#### 空闲让进

空闲让进：当无进程在互斥区时，任何有权使用互斥区的进程可进入。

#### 忙则等待

忙则等待：不允许两个以上的进程同时进入互斥区。

#### 有限等待

有限等待：任何进入互斥区的要求应在有限的时间内得到满足。

#### 让权等待

让权等待：处于等待状态的进程应放弃占用 CPU，以使其他进程有机会得到 CPU的使用权。

### 6. 进程互斥的软件方法

Dekker算法

##### 尝试1：单标志

问题：该算法可确保每次只允许一个进程进入临界区。但它强制两个进程轮流进入。如当Pi退出时将turn置为1，以便 Pj能进入，但Pj暂不需要进入，而这时Pi又需要进入时，它无法进入。这不能保证准则1。

##### 尝试2：双标志、先检查

问题：该算法可确保准则1。但又出现新问题。当pi和pj都未进入时，它们各自的访问标志都为false。如果pi和pj几乎同时要求进入，它们都发现对方的标志为false，于是都进入了。这不能保证准则2。

##### 尝试3：双标志、后检查

问题：该算法可确保准则2。但又出现新问题。它可能造成谁也不能进入。如，当pi和pj几乎同时都要进入，分别把自己的标志置为true，都立即检查对方的标志，发现对方为true. 最终谁也不能进入。 这不能保证准则1。

##### 算法4（ Peterson算法）：先修改、后检查、后修改者等待

##### 软件解法的缺点

忙等待

实现过于复杂，需要较高的编程技巧

### 7. 进程互斥的硬件方法

##### 中断禁用 (关中断, Interrupt Disabling)

##### 专门的机器指令

##### 优缺点

* 硬件方法的优点
  + 适用于任意数目的进程，在单处理器或多处理器上
  + 简单，容易验证其正确性
  + 可以支持进程内存在多个临界区，只需为每个临界区设立一个布尔变量
* 硬件方法的缺点
  + 等待要耗费CPU时间，不能实现"让权等待"
  + 可能"饥饿"：从等待进程中随机选择一个进入临界区，有的进程可能一直选不上
  + 可能死锁

### 8. 用锁实现进程互斥

* 锁即操作系统中的一标志位，０表示资源可用，１表示资源已被占用。用户程序不能对锁直接操作，必须通过操作系统提供的上锁和开锁原语来操作。
* 通常锁用w表示，上锁开锁原语分别用lock(w)、unlock(w)来表示。

#### 进程的交互关系：可以按照相互感知的程度来分类



互斥，指多个进程不能同时使用同一个资源；

死锁，指多个进程互不相让，都得不到足够的资源；

饥饿，指一个进程一直得不到资源（其他进程可能轮流占用资源）

### 9. 信号量和PV原语操作（必考，深入地掌握，彻底地理解）

#### （1）信号量的定义

信号量是一个数据结构, 它由两个变量构成: 整型变量V、指针变量S。

* + 初始化指定一个非负整数值，表示空闲资源总数（又称为“资源信号量”）
  + 若为非负值表示当前的空闲资源数，若为负值其绝对值表示当前等待临界区的进程数
  + 信号量的值只能被P、V操作原语进行改变

信号量定义：

struct semaphore

{

int value;

pointer\_PCB queue;

}

信号量声明：

semaphore s;

* 必须置一次且只能置一次初值
* 初值应该大于等于零，不能为负数
* 只能执行P、V操作

#### （2）信号量的物理意义

S>0表示有S个资源可用

S=0表示无资源可用

S<0则| S |表示S等待队列中的进程个数

进程等待队列 s.queue是阻塞在该信号量的各个进程标识

P(S)：表示申请一个资源

V(S)：表示释放一个资源

* 操作系统对信号量只能通过初始化和两个标准的原语来访问，对信号量的操作只有三种原子操作
  + 初始化：通常将信号量的值初始化为非负整数
  + P操作(wait操作）
    - 使信号量的值减1(申请一个单位的资源 (s.value--))
    - 如果使信号量的值变成负数, 则执行P操作的进程被阻塞(当s.value < 0时, 资源已分配完毕, 进程自己阻塞在S的队列上----让权等待)
  + V操作(signal操作）
    - 使信号量的值加1(释放一个单位资源 (s.value++))
    - 如果信号量的值不是正数, 则使一个因执行v操作被阻塞的进程解除阻塞(若s.value <= 0, 则唤醒一个等待进程)

#### （3）P原语wait(s)

P(s)

{

s.value = s.value -1 ; //表示申请一个资源;

if (s.value < 0) //表示没有空闲资源;

{

该进程状态置为等待状态

将该进程的PCB插入相应的等待队列末尾s.queue;

}

}

#### （4）V原语signal(s)

V(s)

{

s.value = s.value +1; //表示释放一个资源；

if (s.value < = 0) //表示有进程处于阻塞状态；

{

唤醒相应等待队列s.queue中等待的一个进程

改变其状态为就绪态，并将其插入就绪队列

}

}

#### （5）信号量及P、V操作讨论

* P.V操作必须成对出现，有一个P操作就一定有一个V操作。
* 当为互斥操作时，它们同处于同一进程；当为同步操作时，则不在同一进程中出现。
* 如果P(S1)和P(S2)两个操作在一起，那么P操作的顺序至关重要：一个同步P操作与一个互斥P操作在一起时同步P操作在互斥P操作前，而两个V操作无关紧要。

#### （6）P、V操作的优缺点

优点：

简单，而且表达能力强*（用P、V操作可解决任何同步互斥问题）*

缺点：

不够安全，P、V操作使用不当会出现死锁；

遇到复杂同步互斥问题时实现复杂

#### （7）利用信号量实现互斥



* 在互斥问题中，对信号量mutex必须设置一次初值，初值必须为1
* 在每个进程中将临界区代码置于P和V原语之间,P、V原语操作应该分别紧靠临界区的头部和尾部，从而提高进程的并发度
* Mutex的取值为：1,0,-1,-2,…,-(n-1)
* P、V操作必须成对出现，而且它们同处于同一个进程中

应用场景1：对临界区的访问控制

应用场景2：对独占资源的访问控制

#### （8）利用信号量实现同步

一般情况下，同步信号量的初始值都为0.

一般情况下，有几次同步过程，就定义几个同步信号量。

#### （9）经典进程同步问题

##### 生产者－消费者问题(the Producer-Consumer Problem) 同步+互斥

i. 问题描述

若干进程通过有限的共享缓冲区交换数据。其中，

* + “生产者”进程不断写入;
  + “消费者”进程不断读出；
  + 共享缓冲区共有N个；
  + 任何时刻只能有一个进程可对共享缓冲区进行操作。

ii. 问题分析

* 为解决生产者消费者问题，应该设两个同步信号量，一个说明空缓冲区的数目，用S1表示，初值为有界缓冲区的大小N，另一个说明已用缓冲区的数目，用S2表示，初值为０。
* 由于在此问题中有M个生产者和N个消费者，它们在执行生产活动和消费活动中要对有界缓冲区进行操作。由于有界缓冲区是一个临界资源，必须互斥使用，所以，另外还需要设置一个互斥信号量mutex，其初值为1。

**分析P操作的顺序很重要（V操作不会产生死锁）**

##### 读者/写者问题(Readers and Writers Problem) 复杂、特殊的互斥问题

i. 问题描述：对共享资源的读写操作，有两组并发进程——读者和写者，共享一组数据区

* 要求：
  + 允许多个读者同时执行读操作
  + 不允许读者、写者同时操作
  + 不允许多个写者同时操作

**“读－写”互斥，“写－写”互斥，“读－读"允许**

ii. 问题分析

* 第一类，读者优先

如果读者来：

1）无读者、写者，新读者可以读

2）有写者等，但有其它读者正在读，新读者也可以读

3）有写者写，新读者等

如果写者来：

1）无读者，新写者可以写

2）有读者，新写者等待

3）有其它写者，新写者等待

* 第一类读者写者问题的解法
* 设有两个信号量w=1，mutex=1
* 另设一个全局变量readcount =0（不是信号量）
* w用于读者和写者、写者和写者之间的互斥
* readcount表示正在读的读者数目
* mutex用于对readcount 这个临界资源的互斥访问
* 写优先
* 修改以上读者写者问题的算法，使之对写者优先，即一旦有写者到达，后续的读者必须等待，无论是否有读者在读。
* 提示：增加一个信号量，用于在写者到达后封锁后续的读者

##### 哲学家进餐问题(the Dining Philosophers Problem)

i. 问题描述（由Dijkstra首先提出并解决）

* + 5个哲学家围绕一张圆桌而坐，
  + 桌子上放着5支筷子，每两个哲学家之间放一支；
  + 哲学家的动作包括思考和进餐，
  + 进餐时需要同时拿起他左边和右边的两支筷子，
  + 思考时则同时将两支筷子放回原处。
* 问题：如何保证哲学家们的动作有序进行？如：
  + 不出现相邻者同时要求进餐；
  + 不出现有人永远拿不到筷子；

ii. 问题分析

为防止死锁发生可采取的措施：

* 最多允许4个哲学家同时坐在桌子周围
* 仅当一个哲学家左右两边的筷子都可用时，才允许他拿筷子（√）
* 给所有哲学家编号，奇数号的哲学家必须首先拿左边的筷子，偶数号的哲学家则反之

## （四）进程死锁（必考）

#### （1）死锁的规范定义

一组进程中，每个进程都无限等待被该组进程中另一进程所占有的资源，因而永远无法得到资源，这种现象称为进程死锁，这一组进程就称为死锁进程。

* **如果死锁发生，会浪费大量系统资源，甚至导致系统崩溃。**

#### （2）产生死锁的原因和必要条件

死锁指进程处于等待状态，且等待事件永远不会发生。

* + 参与死锁的进程最少是两个；
  + 参与死锁的进程至少有两个已经占有资源；
  + 参与死锁的所有进程都在等待资源；
  + 参与死锁的进程是当前系统中所有进程的子集。

##### 产生死锁的原因

* + 资源不足导致的资源竞争：多个进程所共享的资源不足，引起它们对资源的竞争而产生死锁。
  + 并发执行的顺序不当。进程运行过程中，请求和释放资源的顺序不当，而导致进程死锁. 如P,V操作的顺序不当

##### 产生死锁的四个必要条件（必考）

* 互斥(mutual exclusion)
* 占有等待 (hold and wait)
* 不可剥夺 (no preemption)
* 环路等待(circular wait)
* 互斥条件
  + 指进程对所分配到的资源进行排它性使用, 即在一段时间内某资源只能由一个进程占有。如果此时还有其它进程申请该资源，则它只能阻塞, 直至占有该资源的进程释放。
* 占有且等待（请求和保持条件）
  + 进程已经保持了至少一个资源, 但又提出了新的资源要求, 而该资源又已被其它进程占有, 此时请求进程阻塞, 但又对已经获得的其它资源保持不放。
* 非抢占（非剥夺）条件
  + 进程已获得的资源, 在未使用完之前, 不能被剥夺, 只能在使用完时由自己释放。
* 循环等待条件
  + 在发生死锁时, 必然存在一个进程-资源的封闭的环形链. 即进程集合{P0, P1, P2, …, Pn}中的P0正在等待一个P1占用的资源; P1正在等待P2占用的资源, ……, Pn正在等待已被P0占用的资源.
  + 循环等待只是死锁的必要条件。

#### （3）处理死锁的基本方法



预防死锁：通过限制如何申请资源的方法来确保至少有一个条件不成立。

避免死锁：根据有关进程申请资源和使用资源的额外信息，确定对于一个申请，进程是否应该等待。

检测死锁和恢复：通过算法来检测并恢复

忽视此问题：认为死锁不可能在系统内发生。如Unix采用这种方法。

##### 死锁的预防

* 定义：在系统设计时确定资源分配算法，保证不发生死锁
* 死锁的四个必要条件记做C1， C2， C3， C4，死锁记做D，则有逻辑公式：D→ C1∧C2∧ C3∧ C4。

推导得： ¬ C1∨¬ C2∨¬ C3∨¬ C4→¬D

* 具体的做法：破坏产生死锁的四个必要条件之一
  + 互斥条件 **×**
  + 请求和保持条件（占有和等待）
  + 不可剥夺条件
  + 环路等待条件

###### i. 破坏请求和保持条件

* 方法一：要求每个进程在运行前必须一次性申请它所要求的所有资源，
* 方法二：进程提出申请资源前必须释放已占有的一切资源。
* 优点: 简单、易于实现、安全
* 缺点:
  + 一个进程可能被阻塞很长时间，等待资源，发生饥饿
  + 资源严重浪费, 进程延迟运行;

###### ii. 破坏不可剥夺条件

* 方法一：OS可以剥夺一个进程占有的资源, 分配给其他进程（只有当两个进程优先级相同时）。
* 方法二：一个已经保持了某些资源的进程, 当它再提出新的资源请求而不能立即得到满足时, 必须释放它已经保持的所有资源, 待以后需要时再重新申请
* 适用条件
  + 资源的状态可以很容易地保存和恢复，如CPU寄存器、内存空间，不能适用于打印机、磁带机
* 缺点
  + 实现复杂、代价大, 反复申请/释放资源, 系统开销大, 降低系统吞吐量

###### iii. 破坏环路等待条件

* 采用资源有序分配法：
  + 把系统中所有资源编号，进程在申请资源时必须严格按资源编号的递增次序进行，否则操作系统不予分配
  + 例：哲学家就餐问题
* 问题:
  + 此方法要求资源类型序号相对稳定, 不便于添加新类型的设备.
  + 易造成资源浪费, 类型序号的安排只能考虑一般作业的情况, 限制用户简单、自主地编程
  + 限制进程对资源的请求；资源的排序占用系统开销；

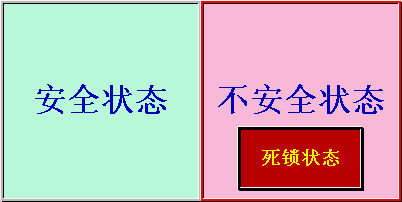
##### 死锁的避免

* 在系统运行过程中，对进程发出的每一个系统能够满足的资源申请进行动态检查，并根据检查结果决定是否分配资源，若分配后系统可能发生死锁，则不予分配，否则予以分配。
* 安全状态：指系统能按某种进程顺序(P1, P2, …，Pn) 来为每个进程Pi分配其所需资源，直至满足每个进程对资源的最大需求，使每个进程都可顺利地完成。如果系统无法找到这样一个安全序列，则称系统处于不安全状态(称〈P1, P2, …, Pn〉序列为安全序列)。

###### i. 安全状态与不安全状态

安全状态：如果存在一个由系统中所有进程构成的安全序列P1，…Pn，则系统处于安全状态

不安全状态：不存在一个安全序列。不安全状态不一定导致死锁，只是很可能死锁。



###### ii. 安全序列

一个进程序列{P1，…，Pn}是安全的，如果对于每一个进程Pi(1≤i≤n），它以后尚需要的资源量不超过系统当前剩余资源量与所有进程Pj (j < i )当前占有资源量之和，系统处于安全状态。

* 安全序列可以不唯一!

安全状态一定是没有死锁发生的

不安全状态：不存在一个安全序列。

不安全状态不一定导致死锁

###### iii. 死锁避免

* 死锁预防是严格破坏4个必要条件之一，一定不出现死锁；而死锁的避免是不那么严格地限制死锁必要条件的存在，其目的是提高系统的资源利用率。万一当死锁有可能出现时，就小心避免这种情况的发生。
* 不需像死锁预防那样，事先采取限制措施破坏产生死锁的必要条件; 在资源的动态分配过程中, 采用某种策略防止系统进入不安全状态, 从而避免发生死锁
  + 如果一个新的进程的资源请求会导致死锁, 则拒绝启动这个进程
  + 如果满足一个进程新提出的一项资源请求会导致死锁, 则拒绝分配资源给这个进程

iv. 银行家算法（深入的理解掌握并能够进行分析）

* **银行家算法**
  + 银行家拥有一笔周转资金
  + 客户要求分期贷款，如果客户能够得到各期贷款，就一定能够归还贷款，否则就一定不能归还贷款
  + 银行家应谨慎的贷款，防止出现坏帐
* **用银行家算法避免死锁**
  + 操作系统（银行家）
  + 操作系统管理的资源(周转资金)
  + 进程（要求贷款的客户）
* n：系统中进程的总数；m：资源类总数
* 银行家算法中的数据结构
  + (1) 可利用资源向量Available。含有m个元素的数组，其中的每一个元素代表一类可利用的资源数目，其初始值是系统中所配置的该类全部可用资源的数目，其数值随该类资源的分配和回收而动态地改变。
  + (2) 最大需求矩阵Max。这是一个n×m的矩阵，它定义了系统中n个进程中的每一个进程对m类资源的最大需求。如果Max［i,j］=K，则表示进程i需要Rj类资源的最大数目为K。
* 银行家算法总结
* 优点
  + 比死锁预防限制少
  + 无死锁检测方法中的资源剥夺, 进程重启
* 缺点
  + 必须事先声明每个进程请求的最大资源
  + 考虑的进程必须是无关的, 也就是说, 它们执行的顺序没有任何同步要求的限制
  + 进程的数量保持固定不变，且分配的资源数目必须是固定的
  + 在占有资源时, 进程不能退出

##### 死锁的检测

* 允许死锁发生，操作系统不断监视系统进展情况，判断死锁是否发生
* 一旦死锁发生则采取专门的措施，解除死锁并以最小的代价恢复操作系统运行
* 检测时机：
  + 当进程等待时检测死锁（缺点是系统的开销大）

检测时效性不高，代价大

* + 定时检测

检测频率比较低，开销较小，时效性差

* + 系统资源利用率下降时检测死锁

检测频率和时效性介于两者之间，比较折中、权衡的检测时机

###### i. 资源分配图

###### ii. 死锁定理

* 如果资源分配图中没有环路，则系统中没有死锁，如果图中存在环路则系统中可能存在死锁。
* 如果每个资源类中只包含一个资源实例，则环路是死锁存在的充分必要条件。

###### iii. 资源分配图

* 1）找一个非孤立点进程结点且只有分配边，去掉分配边，将其变为孤立结点
* 2）再把相应的资源分配给一个等待该资源的进程，即将某进程的申请边变为分配边
* 3）重复以上步骤，若所有进程成为孤立结点，称该图是可完全简化的，否则称该图是不可完全简化的。

一个给定的进程-资源图的全部化简序列导致同一不可化简图。

死锁状态的充分条件：当且仅当资源分配图是不可完全简化的。

##### 死锁的解除

* 以最小的代价恢复系统的运行。方法如下：
  + 1）重新启动（代价最大，成本最高）
  + 2）撤消进程
  + 3）剥夺资源
  + 4）进程回退

i. 方法2：进程终止

* + 终止所有的死锁进程－OS中常用方法
  + 一次只终止一个进程直到取消死锁循环为止－基于某种最小代价原则。
* 选择原则
  + 已消耗CPU时间最少
  + 到目前为止产生的输出量最少
  + 预计剩余的时间最长
  + 目前为止分配的资源总量最少
  + 优先级最低
* 终止进程需要做很多工作

ii. 方法3：资源抢占：逐步从进程中强占资源给其它进程使用，直到死锁环被打破为止 。考虑如下问题：

* + 选择一个牺牲品：抢占哪些资源和哪个进程，确定抢占顺序以使代价最小。
  + 饥饿：确保资源不会总是从同一个进程中被抢占

iii. 方法4：回滚：把每个死锁进程备份到前面定义的某些检查点, 并且重新启动所有进程－需要系统构造重新运行和重新启动机制

## （五）进程的通信问题

### 1. 基本概念

* 进程通信是指进程之间可直接以较高的效率传递较多数据的信息交换方式。
* 低级通信：只能传递状态和控制信息等，进行简单的信号交换。
  + 优点：速度快
  + 缺点：
    - 传送信息量小：效率低，每次通信传递的信息量固定，若传递较多信息则需要进行多次通信。
    - 编程复杂：用户直接实现通信的细节，易出错。
* 高级通信：高效、大量的数据传递。

### 2. 进程间通信类型

#### （1）类型概述

* 共享存储器系统
  + 通过数据、数据区的共享，写入与读出达到通信的目的
* 消息传递系统
  + 直接通信方式：消息缓冲
    - 采用进程的消息缓冲队列
    - 消息发送者将消息直接放在接收者的消息缓冲队列
  + 间接通信方式：邮箱通信
    - 利用中间者——信箱、邮局来传递信件。
    - 发送进程将消息发送到信箱中，接收进程从信箱中取出消息
* 管道通信 (共享文件方式)
  + 用以连接读、写进程的共享文件

#### （2）共享存储器系统（Shared-Memory System）

##### 基于共享数据结构的通信方式

* + 诸进程公用某些数据结构，进程通过它们交换信息。如生产者-消费者问题中的有界缓冲区。

##### 基于共享存储区的通信方式

* + 高级通信，在存储器中划出一块共享存储区，进程在通信前，向系统申请共享存储区中的一个分区，并指定该分区的关键字，若系统已经给其它进程分配了这样的分区，则将该分区的描述符返回给申请者。接着，申请者把获得的共享存储分区连接到本进程上，此后可读写该分区。
* **以上两种方式的同步互斥都要由进程自己负责。**

##### UNIX的共享存储区

* 创建或打开共享存储区(shmget)：依据用户给出的整数值key，创建新区或打开现有区，返回一个共享存储区ID。
* 连接共享存储区(shmat)：连接共享存储区到本进程的地址空间，可以指定虚拟地址或由系统分配，返回共享存储区首地址。父进程已连接的共享存储区可被fork创建的子进程继承。
* 拆除共享存储区连接(shmdt)：拆除共享存储区与本进程地址空间的连接。
* 共享存储区控制(shmctl)：对共享存储区进行控制。如：共享存储区的删除需要显式调用shmctl(shmid, IPC\_RMID, 0)。

### 3. 消息传递系统（Message passing system）

* 进程间的数据交换以消息为单位，程序员利用系统的通信原语实现通信。
* 消息传递系统可分为：
  + 直接通信：发送进程直接把消息发送给接收者，并将它挂在接收进程的消息缓冲队列上。接收进程从消息缓冲队列中取得消息。也称为消息缓冲通信。
  + 间接通信：发送进程将消息发送到某种中间实体中（信箱），接收进程从中取得消息。也称信箱通信。在网络中称为电子邮件系统。
* 两种方式的主要区别？
  + 前者需要两进程都存在，后者不需要。

### 4. 管道通信Pipe

* 所谓管道，是指用于连接一个读进程和一个写进程的文件，称pipe文件。向管道提供输入的进程（称写进程），以字符流的形式将大量数据送入管道，而接受管道输出的进程（读进程）可从管道中接收数据。该方式首创于UNIX，它能传送大量数据，被广泛采用。

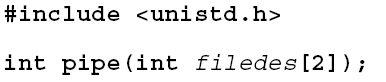


#### 管道通信的实现

* 管道：是所有Unix都提供的一种IPC机制
  + 一个进程将数据写入管道，另一个进程从管道中读取数据
  + 是连接接收进程和发送进程的共享文件
    - **（1）互斥访问**
    - **（2）写后读，读后写的同步**
    - **（3）只有在管道双方都存在时才能通信**
  + 无名管道，有名管道

#### （1）UNIX无名管道

* 通过pipe系统调用创建无名管道，得到两个文件描述符
  + 文件描述符fildes[0]只能从管道读，fildes[1]只能向管道写；
  + 通过系统调用write和read进行管道的写和读；



* 使用管道的两种限制：
  + 管道只能在具有亲缘关系的进程之间进行通信（具有公共祖先的进程之间）
    - 通过fork传递管道的描述符
  + 管道通信方式是单向的，进程间双向通信通常需要两个管道。

#### （2）管道破裂

* 如果一个管道的读端已经关闭，进程还继续向写端写数据，如：

pipe(fdes);

close(fdes[0]);

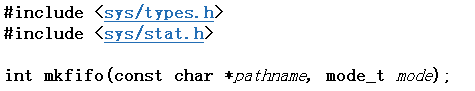
write(fdes[1], “Let me die”, 10);

则进程会收到一个SIGPIPE信号，表示管道破裂。默认动作为结束进程。

* 读一个写端已经关闭的管道则read返回0。

#### （3）UNIX有名管道

* FIFO，有名管道
  + 特殊的文件类型：  
    1，先入先出  
    2，类似管道，在文件系统中不存在数据块，而是与一块内核缓冲区相关联  
    3，有名字，FIFO的名字包含在系统的目录树结构中，可以按名访问
  + open，close，read，write等普通文件操作



# 二、存储管理

## （一）存储管理的基本概念

* **内存：**由存储单元（字节或字）组成的一维连续的地址空间，简称内存空间。用来存放当前正在运行程序的代码及数据，是程序中指令本身地址所指的、亦即程序计数器所指的存储器。
* **分为：**
  + 系统区：用于存放操作系统
  + 用户区：用于装入并存放用户程序和数据
* **存储管理的目的：**
  + 充分利用内存，为多道程序并发执行提供存储基础
  + 尽可能方便用户使用，如：自动装入用户程序，用户程序中不必考虑硬件细节
  + 解决程序空间比实际内存空间大的问题

### 1. 存储系统的组织与四大功能

#### （1）存储是一个层次化的结构

* 存储组织是指在存储技术和CPU寻址技术许可的范围内组织合理的存储结构。
  + 其依据是访问速度匹配关系、容量要求和价格。
  + “寄存器-内存-外存”结构
  + “寄存器-缓存-内存-外存”结构

#### （2）存储管理的四大功能

1. 存储空间的管理、分配和回收
   * 记录内存的使用情况——设置相应的内存分配表（内存分配回收的依据）
   * 静态存储分配；动态存储分配
   * 分配和回收算法及相应的数据结构。
2. 地址再定位（地址变换、地址映射）：
   * 可执行文件生成中的链接技术
   * 程序加载(装入)时的重定位技术
   * 进程运行时硬件和软件的地址变换技术和机构
3. 存储共享和保护：两个或多个进程共用内存中相同区域
   * 代码和数据共享
   * 地址空间访问权限、基址—限长存储保护
     + 上、下界存储保护
   * 保护过程----防止地址越界、防止操作越权
4. 存储器扩充：存储器的逻辑组织和物理组织，虚拟存储；
   * 由应用程序控制：覆盖；
   * 由OS控制
     + 交换（整个进程空间）
     + 虚拟存储的请求调入和预调入（部分进程空间）

### 2. 地址再定位

#### （1）基本概念

* 名空间：程序中由符号名组成的空间。
* 物理地址（绝对地址，实地址）：内存中存储单元的地址。物理地址可直接寻址。
* 逻辑地址（相对地址，虚地址）：用户的程序经过汇编或编译后形成目标代码，目标代码通常采用相对地址的形式。是指相对于某个基准量(通常用0)编址时所使用的地址。（不能够直接用来寻址）
  + 其首地址为0，其余指令中的地址都相对于首地址来编址。
  + 不能用逻辑地址在内存中读取信息。
* 逻辑地址空间通过地址再定位机构转换到绝对地址空间。

#### （2）地址的再定位方法

* 将逻辑地址空间的程序装入到物理地址空间时，由于两个空间不一致，需要进行地址变换，所引起的对有关地址部分的调整过程称为地址再定位。
* 程序在成为进程前的准备工作
  + 编辑：形成源文件（符号地址）
  + 编译：形成目标模块（模块内符号地址解析）
  + 链接：由多个目标模块或程序库生成可执行文件（模块间符号地址解析）
  + 装入：构造PCB，形成进程（使用物理地址）
* 再定位方法
  + 静态再定位
  + 动态再定位

##### 静态地址再定位

* 在程序执行之前进行地址再定位，由装配程序完成。
  + 在可执行文件中，列出各个需要重定位的地址单元和相对地址值。当用户程序被装入内存时，一次性实现逻辑地址到物理地址的转换，以后不再转换（一般在装入内存时由软件完成）。即：装入时根据所定位的内存地址去修改每个重定位地址项，添加相应偏移量。
* **优点：**不需硬件支持，可以装入有限多道程序。
* **缺点：**
  + 程序装入内存后不能移动
  + 一个程序通常需要占用连续的内存空间
  + 不易实现共享

##### 动态地址再定位

* 在执行寻址时重定位——在程序运行过程中要访问数据时再进行地址变换，即在逐条指令执行时完成地址映射。
* 一般为了提高效率，此工作由硬件地址映射机制来完成。硬件支持，软硬件结合完成
* 硬件上需要一对寄存器的支持：基址寄存器、变址寄存器。
* 优缺点：
* 优点：程序占用的内存空间是动态可变的，当程序从某个存储区移到另一个区域时，只需要修改相应的寄存器BR的内容即可。
  + 一个程序不一定要求占用一个连续的内存空间。
  + 可以部分地装入程序运行。
  + 便于多个进程共享同一个程序的代码。
* 动态地址重定位的代价：
  + 需要硬件的支持。
  + 实现存储管理的软件算法较为复杂。

## （二）分区存储管理

### 1. 概述

* **原理：**把内存分为一些大小相等或不等的分区，每个应用进程占用一个或几个分区。操作系统占用其中一个分区。
* **特点：**适用于多道程序系统和分时系统
* **问题：**可能存在内碎片和外碎片。
  + 内碎片：占用分区之内未被利用的空间
  + 外碎片：占用分区之间难以利用的空闲分区（通常是小空闲分区）。
* **分区方式：**
  + 固定分区(fixed partitioning)
  + 动态分区(dynamic partitioning)
  + 分区分配算法
* 分区的数据结构：分区表，或分区链表
  + 只记录空闲分区，或同时记录空闲和占用分区
* 内存紧缩(compaction)：将各个占用分区向内存一端移动，使各个空闲分区聚集在另一端，然后将各个空闲分区合并成为一个空闲分区。
  + 对占用分区进行内存数据搬移占用CPU时间
  + 如果对占用分区中的程序进行"浮动"，则其重定位需要硬件支持。
  + 紧缩时机：每个分区释放后，或内存分配找不到满足条件的空闲分区时

### 2. 固定分区(fixed partitioning)

* 把内存划分为若干个固定大小的连续分区。
  + **分区大小相等：**只适合于多个相同程序的并发执行（处理多个类型相同的对象）。
  + **分区大小不等：**多个小分区、适量的中等分区、少量的大分区。根据程序的大小，分配当前空闲的、适当大小的分区。
* 优点：易于实现，开销小。
* 缺点：
  + 内碎片造成浪费
  + 分区总数固定，限制了并发执行的程序数目。
* 采用的数据结构：分区表－－记录分区的大小和使用情况

### 3. 动态分区(dynamic partitioning)

* 动态创建分区：在装入程序时按其初始要求分配，或在执行过程中通过系统调用进行分配或改变分区大小。
* 优点：没有内碎片。
* 缺点：有外碎片；如果大小不是任意的，也可能出现内碎片。

### 4. 分区分配算法

* 分区分配算法：寻找某个空闲分区，其大小需大于或等于程序的要求。若是大于要求，则将该分区分割成两个分区，其中一个分区为要求的大小并标记为“占用”，而另一个分区为余下部分并标记为“空闲”。分区的先后次序通常是从内存低端到高端。
* 分区释放算法：需要将相邻的空闲分区合并成一个空闲分区。(这时要解决的问题是：合并条件的判断和合并时机的选择)

#### （1）最先匹配法(first-fit)：按分区的先后次序，从头查找，找到符合要求的第一个分区

* + 该算法的分配和释放的时间性能较好，较大的空闲分区可以被保留在内存高端。
  + 但随着低端分区不断划分而产生较多小分区，每次分配时查找时间开销会增大。

#### （2）下次匹配法(next-fit)：按分区的先后次序，从上次分配的分区起查找（到最后分区时再回到开头），找到符合要求的第一个分区

* + 该算法的分配和释放的时间性能较好，使空闲分区分布得更均匀，但较大的空闲分区不易保留。

#### （3）最佳匹配法(best-fit)：找到其大小与要求相差最小的空闲分区

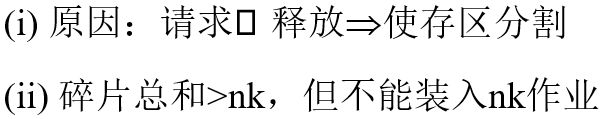
* + 从个别来看，外碎片较小，但从整体来看，会形成较多外碎片。较大的空闲分区可以被保留。

#### （4）最坏匹配法(worst-fit)：找到最大的空闲分区

* + 基本不留下小空闲分区，但较大的空闲分区不被保留。

### 5. 碎片(零头)问题

* 存在于已分配的分区之间的一些不能充分利用的空白区

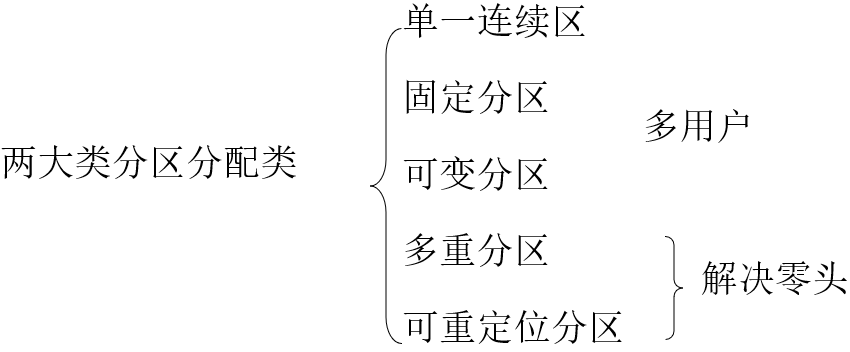


* 解决的方法

(I) 将程序装入分散存区中 ––– **多重分区**

(II) 将碎片集中(紧凑或拼接) ––– **可重定位分配**

* 移动内存已分配区的信息，使得所有分配区靠在一起使空白区连成一片，采用浮动方法。（开销非常大）



## （三）覆盖与交换技术

### 1. 引入

在多道环境下扩充内存的方法，用以解决在较小的存储空间中运行较大程序时遇到的矛盾。

* **覆盖技术：**一个作业的若干程序段，或几个作业的某些部分共享某一个存储空间。
* **交换技术：**系统将内存中某些进程暂时移到外存，把外存中某些进程换进内存，占据前者所占用的区域。

### 2. 覆盖技术(overlay)

#### （1）原理

一个程序的几个代码段或数据段，按照时间先后来占用公共的内存空间。（并不是作业的每一部分都是时时要用的）

* + 把程序划分为若干个功能上相对独立的程序段，按照其自身的逻辑结构将那些不会同时执行的程序段共享同一块内存区域
  + 程序段先保存在磁盘上，当有关程序段的前一部分执行结束，把后续程序段调入内存，覆盖前面的程序段（内存“扩大”了）
  + 一般要求作业各模块之间有明确的调用结构，程序员要向系统指明覆盖结构，然后由操作系统完成自动覆盖

#### （2）分析

* **缺点：**
  + 编程时必须划分程序模块和确定程序模块之间的覆盖关系，增加编程复杂度。
  + 从外存装入覆盖文件，以时间延长来换取空间节省。
* **实现：**函数库（操作系统对覆盖不得知），或操作系统支持

* 例子：目前这一技术用于小型系统中的系统程序的内存管理上，MS-DOS的启动过程中，多次使用覆盖技术；启动之后，用户程序区TPA的高端部分与COMMAND.COM暂驻模块也是一种覆盖结构。

### 3. 交换技术(swapping)

#### （1）引入

多个程序并发执行，可以将暂时不能执行的程序送到外存中，从而获得空闲内存空间来装入新程序。

#### （2）原理

暂停执行内存中的进程，将整个进程的地址空间保存到外存的交换区中（换出swap out），而将外存中由阻塞变为就绪的进程的地址空间读入到内存中，并将该进程送到就绪队列（换入swap in）。

* + 交换单位为整个进程的地址空间。
  + 与分区存储管理配合使用
  + 又称作"对换"或"滚进/滚出(roll-in/roll-out)"

#### （3）交换技术实现中的几个问题

**选择原则：**将哪个进程换出内存？

* + 例子：分时系统，时间片轮转法或基于优先数的调度算法，在选择换出进程时，换出要长时间等待的进程。

**交换时机的确定：**何时需发生交换？

* + 只要不用就换出（或很少再用）
  + 只在内存空间不够或有不够的危险时换出

**交换时需要做哪些工作？**

* + 需要一个盘交换区：必须足够大以存放用户程序的内存映像的拷贝；必须对这些内存映像直接存取。

**换入回内存时位置的确定**

* + 换出后再换入的内存位置一定要在换出前的原来位置上吗？
  + 受地址映射技术的影响，即绝对地址产生时机的限制

#### （4）交换技术的特点

* 优点：
  + 增加并发运行的程序数目，并且给用户提供适当的响应时间；
  + 编写程序时不影响程序结构。
* 缺点：
  + 对换入和换出的控制增加处理机开销；
  + 程序整个地址空间都进行传送，没有考虑执行过程中地址访问的统计特性。

### 4. 覆盖与交换的比较

* **共同点：**
  + 进程的程序和数据主要放在外存，当前需要执行的部分放在内存，内外存之间进行信息交换
* **不同点：**
  + 交换发生在进程或作业之间，而覆盖发生在同一进程或作业内。
  + 与覆盖技术相比，交换技术不要求用户给出程序段之间的逻辑覆盖结构。

## （四）分页存储管理

为什么要引进分页技术?

分区存储管理会产生碎片，需移动分区

### 1. 分页存储管理的基本思想

* 主存分成多个固定大小的块（物理块）
  + 主存划分为大小相等的区域，称为块或内存块（物理页面，页框）
* 作业按照主存块大小分页（逻辑页面）
  + 把用户程序按逻辑页划分成大小相等的部分，称为页（page）。从0开始编制页号，页内地址是相对于0编址
* 连续的页存放在离散的块中
  + 以页为单位进行分配，并按作业的页数多少来分配。逻辑上相邻的页，物理上不一定相邻

### 2. 逻辑地址

* + 一般一页的大小为2的整数次幂，通常为512 B~8 KB。
  + 地址的高位部分为页号，低位部分为页内地址（偏移量）

### 3. 页表

* 将页号和页内地址转换成内存地址，必须要有一个数据结构，用来登记页号和块的对应关系和有关信息。这样的数据结构称为**页表**。
* 系统为每个进程建立一个页表，页表的长度和首地址存放在该进程的进程控制块（PCB）中。
* 页表包含以下几个表项：
  + 页号：登记程序地址空间的页号
  + 块号：相应的页所对应的内存块号
  + 其它：与存储信息保护有关的信息。
* 页大小的选择
* 太大：浪费；太小：页表过长。
* 页的大小是2K ， k： 9-12。
  + IBM AS/400 VAX NS32032 ：512字节
  + Intel 80386 Motorola 68030 ：4096字节

### 4. 地址变换机构



图 2 **分页系统的地址变换机构**

页地址变换过程中有一个严重的问题：访问主存=访页表+访主存（两次访问主存）

解决方法：把页表放在一组快速存储器中（Cache），从而加快访问速度。

### 5. 计算页号和页内地址

（1）若给出的地址字为16进制，则将其转换为二进制，然后，根据页长及程序地址字的长度，分别取出程序地址字的高几位和低几位就得到页号及页内地址。如页长为2K，程序地址字为16位，则高5位为页号，低11位为页内地址。

（2）若给出的地址字为10进制，则用公式：

**程序地址字/页长**

商为页号，余数为页内地址。

如程序地址为8457， 页长为4KB，则8457/4096可得：商为2，余数为256。

### 6. 两级页表(Two-Level Page Table)

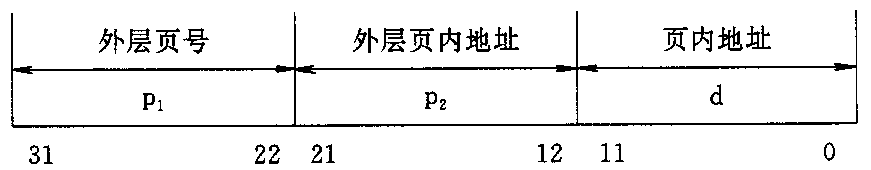


图 3 逻辑地址结构

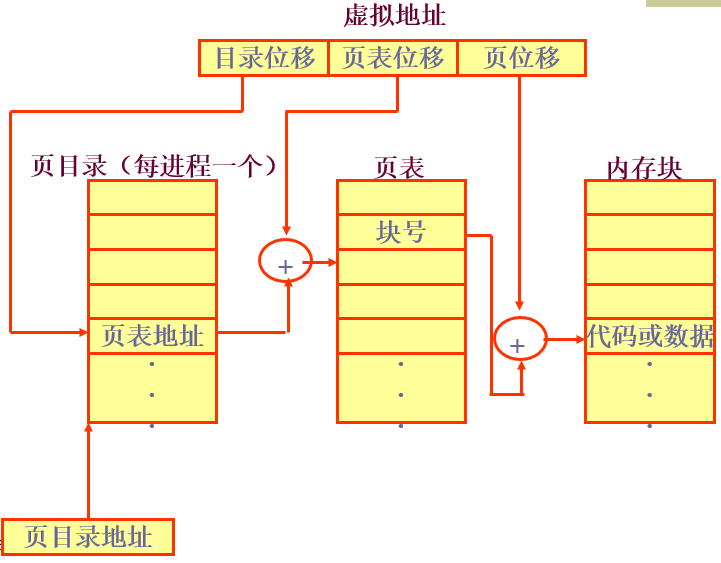


图 4 **二级页表结构及地址映射**



图 5 具有两级页表的地址变换机构

### 7. 快表

* 页地址变换过程中有一个严重的问题：
  + **访问主存＝访页表＋访主存**
* 解决方法：把页表放在一组快速存储器中（Cache），从而加快访问内存的速度。
* 相联存储器（associative memory）：为加快地址转换速度, 系统专门设置的一个高速存储器, 用于存放页表的一部分。又叫TLB（Translation lookaside buffers）
* 把这种存放在快速存储器中的页表称为快表，把存放在内存中的页表称为**慢表**。
* 快表具有并行查询能力：一次进行
  + 查联想表－物理地址(访问一次主存)
  + 查页表－物理地址(访问二次主存)
* 快表表项：
  + 页号；内存块号；标识位；淘汰位

### 8. 小结（优缺点）

* 优点：消除了靠拢操作，部分解决了碎片问题（消除了外碎片，极大地解决了内碎片）

便于管理

* 缺点：增加成本，多占用主存空间

多花费处理机时间（用于相关表格的管理）

## （五）虚拟存储器(Virtual storage)

### 1. 概述

* 解决的问题：内存小，作业大、作业多的矛盾
* 实现的原理：程序执行时的**局部性原理：**1968年， Denning.P
  + 时间局限性
  + 空间局限性
* “扩充”主存 ––– **虚拟存贮技术**
* 实现方式：操作系统统一管理各级存储器；内存中只存放当前要执行的程序部分，其余的保存在外存上，OS根据需要随机地将需要的部分对换到内存执行
* 虚拟存储技术的物质基础：二级存储器结构和动态地址转换机构
  + 二级存储结构：内存、外存
  + 动态地址转换：在操作系统的改造下，内、外存地址统一编址，扩大寻址空间
* 虚空间独立于实空间
  + 虚空间>>实空间(也可以虚空间<实空间)
  + 指令中表示地址的位数越长，可寻址空间越大，但不是无限的，一般为32位、64位。
  + 主存＋辅存≠虚存(仅与地址结构有关)

### 2. 虚空间大小

* 虚空间大小
  + 虚空间的逻辑大小 ＝ 可寻址范围
  + 虚空间的实际大小 ＝ 内存+外存对换区

### 3. 引入虚拟存储技术的好处

* **大程序：**可在较小的可用内存中执行较大的用户程序；
* **大的用户空间：**提供给用户可用的虚拟内存空间通常大于物理内存(real memory)
* **并发：**可在内存中容纳更多程序并发执行；
* **易于开发：**与覆盖技术比较，不必影响编程时的程序结构

#### 4. 虚拟存储技术的特征

* 不连续性：物理内存分配的不连续，虚拟地址空间使用的不连续（数据段和栈段之间的空闲空间，共享段和动态链接库占用的空间）
* 部分交换：与交换技术相比较，虚拟存储的调入和调出是对部分虚拟地址空间进行的；
* 虚拟扩充：通过物理内存和快速外存相结合，提供大范围的虚拟地址空间
  + 总容量不超过物理内存和外存交换区容量之和
* 多次对换：程序运行期间，分别在内、外存中的程序多次对换

### 5. 虚存管理三大策略

* 调入策略(把哪部分装入主存)
* 放置策略(放在主存什么地方)
* 淘汰策略(主存不足时，把哪部分淘汰出主存。)

## （六）请求分页存储管理——虚拟页式存储管理

分页存储管理没有办法解决作业地址空间比物理地址空间长的问题

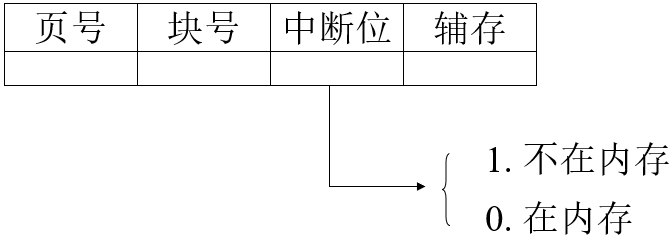
### 1. 基本原理

* **目标：**实现小内存大作业
* **实现方法：**作业运行时，只将当前的一部分装入内存，其余的放在辅存，一旦发现访问的页不在主存中，则发出缺页中断，由o.s将其从辅存调入主存，如果内存无空块，则选择一个页淘汰。
* 分页存储管理系统根据请求装入所需页面的方法称为请求分页存储管理。
* 必须解决以下两个问题：
  + （1）当程序要访问的某页不在内存时，如何发现这种缺页情况？
  + （2）当需要把外存上的某个页面调入内存时，此时内存中没有空闲块应怎么办？

### 2. 基本问题

#### （1）怎样发现页不在内存？

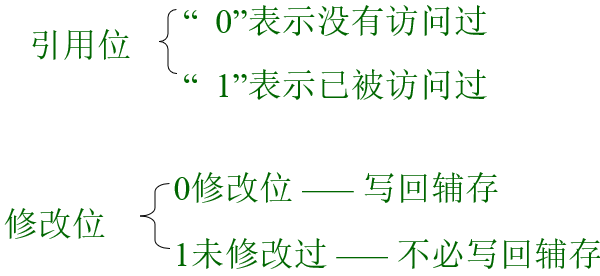
* + 方法：扩充页表，增加两个数据项：中断位、辅存的位置



* + 1：发生缺页中断，调用OS的调页程序。

#### （2）如何处理缺页一缺页中断管理？

* + 发现中断、调入所缺页
  + 如果内存不足，选一页淘汰，选择哪一页呢？
  + 淘汰时，需要注意什么？



页表表项

* 页号、驻留位、内存块号、外存地址、访问位、修改位
  + 驻留位（中断位）：表示该页是在内存还是在外存
  + 访问位：根据访问位来决定淘汰哪页（由不同的算法决定）
  + 修改位：查看此页是否在内存中被修改过



### 3. 缺页中断（Page Fault）处理

* 在地址映射过程中，在页表中发现所要访问的页不在内存，则产生缺页中断。
* 操作系统接到此中断信号后，就调出缺页中断处理程序，根据页表中给出的外存地址，准备将该页调入内存
* 此时应将缺页的进程挂起（调页完成唤醒）
* 如果内存中有空闲块，则分配一个块，将要调入的页装入该块，并修改页表中相应页表项目的驻留位及相应的内存块号
* 若此时内存中没有空闲块，则要淘汰某页（若被淘汰页在内存期间被修改过，则要将其写回外存）

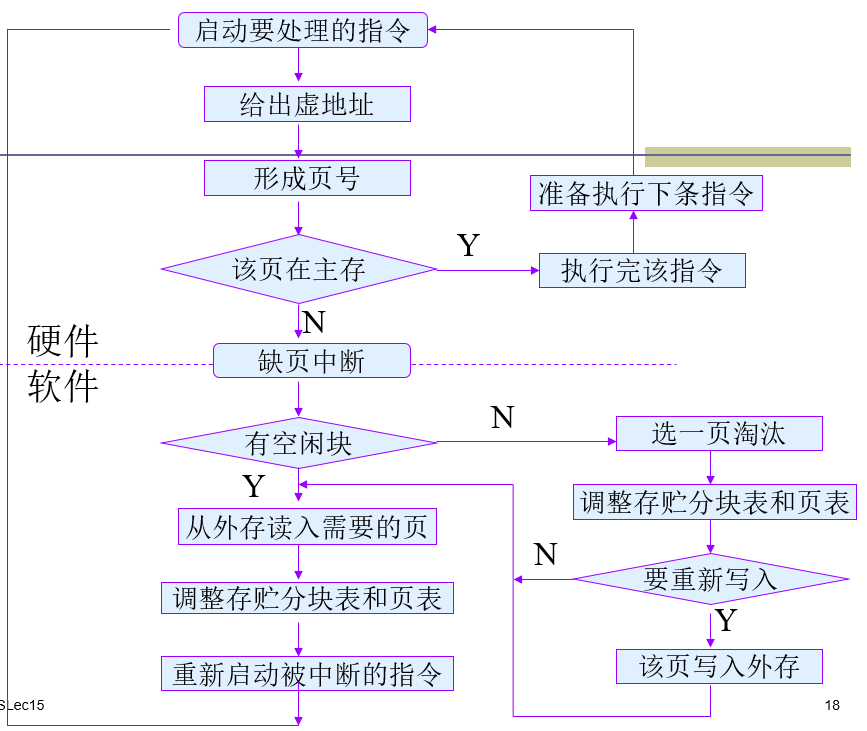


图 6 **指令执行和缺页中断处理**

思考

* 缺页中断同一般中断都是中断。区别？
  + 相同点：
    - 保护现场 中断处理 恢复现场
  + 不同点：
    - 一般中断是一条指令完成后中断，缺页中断是一条指令执行时中断
    - 一条指令执行时可能产生多个缺页中断。如指令可能访问多个内存地址，这些地址在不同的页中。

### 4. 页面置换（淘汰）算法

#### （1）概述

* 功能：需要调入页面时，选择内存中哪个物理页面被置换。称为replacement policy。
* 目标：把未来不再使用的或短期内较少使用的页面调出，通常只能在局部性原理指导下依据过去的统计数据进行预测；
* 页面锁定(frame locking)：用于描述必须常驻内存的操作系统的关键部分或时间关键(time-critical)的应用进程。实现方法为在页表中加上锁定标志位(lock bit)。

#### （2）常用页面置换算法（经常会考，大题）

* 先进先出页面算法（FIFO）：选择在内存中驻留时间最长的页并淘汰之
* 最近最久未使用置换算法（LRU，Least Recently Used）：淘汰没有使用的时间最长的页
* 最佳页面算法（OPT， Optimal）：淘汰以后不再需要的或最远的将来才会用到的页面
* 最不经常使用（LFU， Least Frequently Used）：选择访问次数最少的页面淘汰之

#### （3）性能分析

##### 影响缺页次数的因素

* + (1) 分配给进程的物理块数
  + (2) 页本身的大小
  + (3) 程序的编制方法
  + (4) 页面淘汰算法

##### 常驻集(resident set)

* 常驻集指虚拟页式管理中给进程分配的物理页面数目
* 常驻集与缺页率的关系：
  + 每个进程的常驻集越小，则同时驻留内存的进程就越多，可以提高并行度和处理器利用率；另一方面，进程的缺页率上升，使调页的开销增大。
  + 进程的常驻集达到某个数目之后，再给它分配更多页面，缺页率不再明显下降。该数目是“缺页率－常驻集大小”曲线上的拐点(curve)。
* 试验分析表明：对于所有程序来说，要使其有效地工作，它在主存中的页面数应不低于它的总页面数的一半。

##### 颠簸（抖动）

* 在虚存中，页面在内存与外存之间频繁调度，系统效率急剧下降，甚至导致系统崩溃。这种现象称为颠簸或抖动。
* Question：常驻集越大，缺页中断率越小吗？



##### Belady现象

* Belady现象的描述：一个进程P要访问M个页，OS分配N个内存页面给进程P；对一个访问序列S，发生缺页次数为PE（S,N）。当N增大时，PE(S, N)时而增大，时而减小。
* Belady现象的原因：FIFO算法完全没有考虑进程访问内存的动态特征，即被置换的页面并不是进程不会访问的。

### 5. 请求分页管理方案的评价

* 优点：
  + 提供了虚存管理方式，作业地址空间不再受实存容量的限制；
  + 更有效的利用了主存，方便于多道程序运行，方便了用户；
* 缺点：
  + 为处理缺页中断，增加了处理机时间的开销。用时间的代价换取了空间的扩大；
  + 可能因作业地址空间过大或程序数目过多等造成系统抖动；为此采取措施会增加系统的复杂度。

## （七）分段存储管理

### 1. 分段存储管理基本思想

#### （1）划分方法

* 用户程序划分

按程序自身的逻辑关系划分为若干个程序段，每个程序段都有一个段名，且有一个段号。段号从0开始，每一段段内也从0开始编址，段内地址是连续的。

* + 逻辑地址：分段地址系统中的地址结构形式：



* 内存划分

内存空间被动态的划分为若干个长度不相同的区域，称为物理段，每个物理段由起始地址和长度确定

* 内存分配

以段为单位分配内存，每一个段在内存中占据连续空间（内存随机分割，需要多少分配多少），但各段之间可以不连续存放

#### （2）系统需要维护的数据结构

* 进程段表：描述组成进程地址空间的各段，可以是指向系统段表中表项的索引。每段有段基址(base address)和段长度
* 系统段表：系统内所有占用段
* 空闲段表：内存中所有空闲段，可以结合到系统段表中

### 2. 段地址映射

#### （1）地址映射数据结构

* + 段地址映射的数据结构有段表、段表首址指针和段表的长度。段表首址指针和段表长度存放在进程自己的PCB中。
  + 每一进程有个段表，程序的每一个段在段表中占用一个表目。



#### （2）段地址映射过程

* 程序地址字送入虚地址寄存器VR中。
* 取出段号S和段内位移W。
* 根据段表首址指针找到段表，查找段号为S的表目，得到该段的首地址。
* 把段首地址与段内位移相加，形成内存地址送入MR中，并以此地址访问内存。

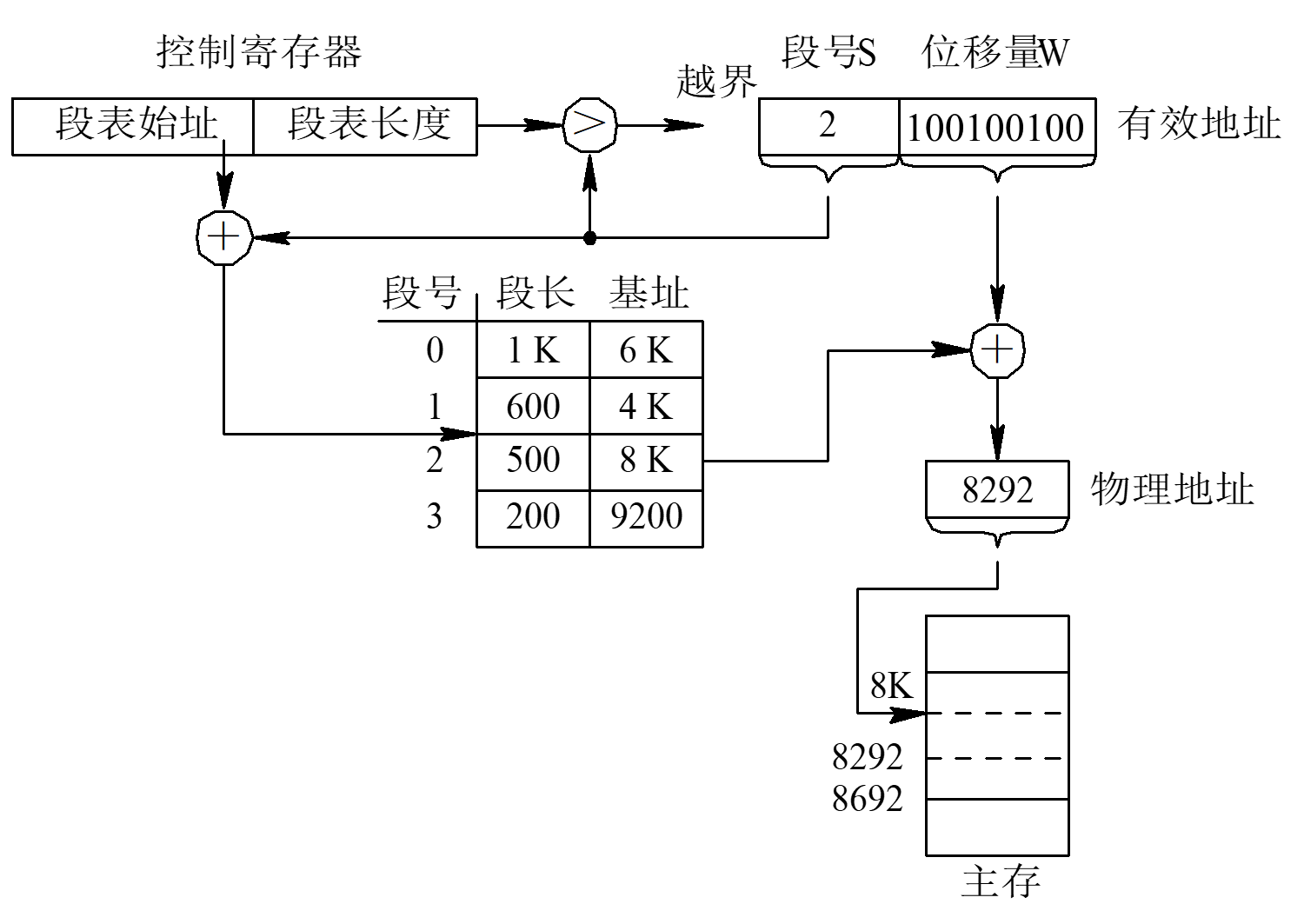


图 7 **分段系统的地址变换过程**

（3）快表

* 同页地址变换一样，在段地址变换过程中，也有两次访问内存的问题。为了加快访问内存的速度也可采用快速存储器组成快表。

#### （4）内存的分配

* 与动态分区管理相同
* 空闲块管理： 空闲块表（队列）
* 内存分配算法（三种）： 首次 最佳 最坏

#### （5）请求段式存储管理

段表SMT内容：段号（段名）、内存始址、段长，

增加：

* 特征位（在/不在内存，是否可共享）
* 存取权限位（读，写，执行）
* 标志位（是否修改过，能否移动）
* 扩充位（固定长/可扩充 ）
* 外存始址

#### （6）缺段中断处理

检查内存中是否有足够的空闲空间

①若有，则装入该段，修改有关数据结构，中断返回

②若没有，检查内存中空闲区的总和是否满足要求，是则应采用紧缩技术，转① ；否则，淘汰一（些）段，转①



图 8 **请求分段系统中的中断处理过程**

### 3. 分段的共享与保护

#### （1）共享段的分配与回收

* 共享段的分配
  + 第一个请求使用该共享段的进程，由系统为该共享段分配一物理区，再把共享段调入该区，并填写共享段表。
  + 又有其它进程需要调用该共享段时只需在调用进程的段表中，增加一表项，填写该共享段的物理地址；修正共享段的段表。
* 共享段的回收
  + 执行count∶=count-1操作。若结果为0，则须由系统回收该共享段的物理内存，以及取消在共享段表中该段所对应的表项，否则只是取消调用者进程在共享段表中的有关记录。

#### （2）分段保护

* 越界检查
* 存取控制检查
  + 只读
  + 只执行
  + 读/写
* 环保护机构
  + 一个程序可以访问驻留在相同环或较低特权环中的数据。
  + 一个程序可以调用驻留在相同环或较高特权环中的服务。

### 4. 分段与分页的主要差别

* + 段是依据程序的逻辑结构划分的，页是按内存线性空间物理划分的。
  + 段式技术中程序地址空间是二维的，分页技术中程序地址空间是一维的。
  + 段是面向用户的，页对用户而言是透明的。
  + 段长由用户决定，且各段的大小一般不相等，唯一的限制是最大长度。页长是由系统决定的，各页的长度必须相等。
  + 段的共享比页的共享更容易。

### 5. 优缺点

* 优点：
  + 消除了内碎片
  + 通过请求分段存储管理方式提供了大量虚存
  + 允许动态增加段的长度
  + 便于动态装入和链接
  + 便于程序共享
  + 便于存储保护
* 缺点：
  + 进行地址变换和实现内存紧凑（靠拢）要花费处理机时间；
  + 在辅存上管理可变长度的段比较困难。

## （八）段页式存储管理

### 1. 段页式存储管理基本思想

* 用户程序划分

按段式划分（对用户来讲，按段的逻辑关系进行划分；对系统讲，按页划分每一段）

* 内存划分

按页式存储管理方案

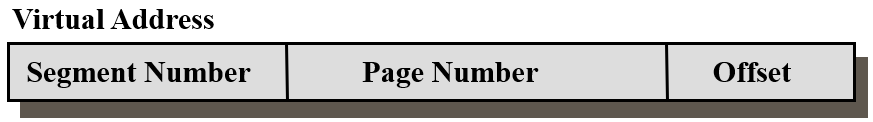
* 内存分配

以页为单位进行分配

* 逻辑地址 

### 2. 地址映射

* 段表：记录了每一段的页表始址和页表长度
* 页表：记录了逻辑页号与内存块号的对应关系（每一段有一个，一个程序可能有多个页表）
* 内存分配管理：同页式管理
* 存储管理的分配单位：段，页
* 逻辑地址的组成：段号，页号，页内偏移地址。
* 地址变换：先查段表，再查该段的页表。
* 缺段中断和缺页中断。



## （九）虚拟存储管理其它问题

### 1. 虚拟存储的调入策略（ fetch policy ）

* 请求调页(demand paging)：只调入发生缺页时所需的页面。
  + 优点：容易实现。
  + 缺点：对外存I/O次数多，开销较大
* 预调页(prepaging)：在发生缺页需要调入某页时，一次调入该页以及相邻的几个页。
  + 优点：提高调页的I/O效率。
  + 缺点：基于预测，若调入的页在以后很少被访问，则效率低。常用于程序装入时的调页。

### 2. 虚拟存储的分配策略 (assignment policy)

* 在虚拟段式管理中，如何对物理内存进行分配，可采用最佳适应、最先适应等。
* 在虚拟页式和段页式管理中，地址变换最后通过页表进行，因此不必考虑分配策略。

### 3. 清除策略(cleaning policy)

* 请求清除(demand cleaning)：该页被置换时才调出，把清除推迟到最后一刻。
  + 缺点：调入所缺页面之前还要调出已修改页面，缺页进程的等待时间较长
* 预清除(precleaning)：该页被置换之前就调出，因而可以成批调出多个页面。
  + 缺点：可能形成不必要的开销。

### 4. 常驻集和工作集策略

#### （1）常驻集

* 常驻集(resident set)
  + 常驻集指虚拟页式管理中给进程分配的物理页面数目。
* 常驻集与缺页率的关系：
  + 每个进程的常驻集越小，则同时驻留内存的进程就越多，可以提高并行度和处理器利用率；另一方面，进程的缺页率上升，使调页的开销增大。
  + 进程的常驻集达到某个数目之后，再给它分配更多页面，缺页率不再明显下降。该数目是"缺页率－常驻集大小"曲线上的拐点(curve)。
* 常驻集大小的确定方式：
  + 固定分配(fixed-allocation)：常驻集大小固定。各进程平均分配，根据程序大小按比例分配，优先权
  + 可变分配(variable-allocation)：常驻集大小可变，按照缺页率动态调整（高或低－>增大或减小常驻集），性能较好。增加算法运行的开销。
* 置换范围(replacement scope)：被置换的页面局限在本进程，或允许在其他进程。
  + 局部置换(local replacement)：容易进行性能分析
  + 全局置换(global replacement)：更为简单，容易实现，运行开销小
* 常驻集大小和置换范围的配合：三种策略
  + 固定分配+局部置换：进程开始前要依据进程类型决定分配多少页面。多了会影响并发水平，少了会使缺页率过高。
  + 可变分配+局部置换：局部转换但在大尺度上进行可变分配。
  + 可变分配+全局置换：这时OS会一直维持一定数目的空闲页面，以快速置换；主要问题是置换策略的选择，如何决定哪个进程的页面将被调出。较好的选择是页面缓冲算法。
  + 不包括“固定分配全局置换”（因为对各进程进行固定分配时不可能进行全局置换）。

#### （2）工作集策略(working set strategy)

* 常驻集大小的动态调整策略----引入工作集概念。
* 工作集是一个进程执行过程中所访问页面的集合，可用一个二元函数W(t, Δ)表示
  + t是执行时刻；
  + Δ是一个虚拟时间段，称为窗口大小(window size)，它采用"虚拟时间"单位(即阻塞时不计时)，大致可以用执行的指令数目，或处理器执行时间来计算；
  + 工作集是在[t - Δ, t]时间段内所访问的页面的集合；
  + | W(t, Δ) | 指工作集大小即页面数目；
* 工作集大小的变化：进程开始执行后，随着访问新页面逐步建立较稳定的工作集。当内存访问的局部性区域的位置大致稳定时，工作集大小也大致稳定；局部性区域的位置改变时，工作集快速扩张和收缩过渡到下一个稳定值。
* 工作集的性质：
  + 随Δ单调递增：W(t, Δ) ⊆ W(t, Δ + a)，其中a>0；
* 利用工作集进行常驻集调整的策略：
  + 记录一个进程的工作集变化；
  + 定期从常驻集中删除不在工作集中的页面；
  + 总是让常驻集包含工作集；
* 困难：
  + 工作集的过去变化未必能够预示工作集的将来大小或组成页面的变化；
  + 记录工作集变化要求开销太大；
  + 对工作集窗口大小Δ的取值难以优化，而且通常该值是不断变化的；
* 主要缺点：
  + 当局部性区域的位置改变时，工作集的变化处于过渡阶段，其快速扩张使较多新页面添加到进程的常驻集中；
  + 其中较少使用的页面至少还要经过一段虚拟时间才会被淘汰，因而带来较多不必要的调页开销。

### 5. 缺页率算法(PFF, page fault frequency)

* 页面被访问时的处理：每个页面设立使用位(use bit)，在该页被访问时设置use bit=1；
* 缺页时的处理：每次缺页时，由操作系统计算与上次缺页的"虚拟时间"间隔t，如处理器执行时间
* 缺页时对常驻集的调整：定义一个"虚拟时间"间隔的阈值(threshold)F。依据t和F来修改常驻集。
  + 如果t小于F，则所缺页添加到常驻集中；
  + 否则，将所有use bit=0的页面从物理内存清除并缩小常驻集；随后，对常驻集中的所有页面设置use bit=0；
* 可变采样间隔算法：VSWS, variable-interval sampled working set

### 6. 虚拟存储中的负载控制

* 改善时间性能的途径
  + 降低缺页率：缺页率越低，虚存平均访问时间延长得越小；
  + 提高外存的访问速度：外存和内存的访问时间比值越大，则达到同样的平均访问时间，所要求的缺页率就越低；
* 抖动问题(thrashing)
  + 随着驻留内存的进程数目增加，或者说进程并发水平的上升，处理器利用率先是上升，然后下降。
  + 处理器利用率下降的原因通常称为虚拟存储器发生“抖动”，
* 决定驻留内存的进程数目，在避免出现抖动的前提下，尽可能提高进程并发水平。

# 三、文件系统

## （一）文件系统概述

### 1. 文件和文件系统

#### （1）什么是文件？

* + 文件是赋名的信息 (数据)项的集合。
  + 文件是赋名有关联的信息单位 (记录)的集合。
* 什么是文件名？
  + 文件的标识符号，一个用来标识文件的有限长度的字符串。

#### （2）文件

* 一组带标识的在逻辑上有完整意义的信息项的序列，这个标识为文件名
  + 信息项：构成文件内容的基本单位
  + 长度：单个字节，或多个字节
  + 文件内容的意义：由文件的建立者和使用者解释
* 文件的组成︰
  + 文件体：文件本身的信息
  + 文件说明：文件存储和管理信息。如：文件名、文件内部标识、文件存储地址、访问权限、访问时间等
* 文件属性：
  + 用一组信息指定文件的类型、操作特性和存取保护等，把这组信息称为文件的属性。
  + 文件的属性一般存放在文件的目录项中。

#### （3）文件系统

* 什么是文件系统？
  + 操作系统中负责管理相关文件信息的软件机构。
* 文件系统管理的对象：
  + （1）文件
  + （2）目录
  + （3）磁盘空间
* 文件系统的组成：
  + 被管理的文件
  + 与文件管理相关的软件
  + 实施文件管理所需的数据结构

### 2. 文件的类型

* 按照性质和用途分类
  + 系统文件：只能通过系统调用为用户服务；
  + 库文件：允许用户调用但不允许用户修改；
  + 用户文件：用户委托操作系统保存的文件，又分为：
    - 1）临时文件；2） 永久文件；3） 档案文件。
* 按文件的保护方式分类
  + 只读文件、读写文件、不保护文件
* 按文件信息的流向
  + 输入文件、输出文件、输入输出文件
* 按文件的逻辑结构分类
  + 流式文件；记录式文件
* UNIX系统将文件分为三类：
* 普通文件：
  + 包含的是用户的信息，一般为ASCII或二进制文件
* 目录文件：
  + 管理文件系统的系统文件
* 特殊文件（设备文件，外部设备也看作文件）
  + 字符设备文件：和输入输出有关，用于模仿串行I/O设备，例如终端，打印机，网络等
  + 块设备文件：模仿磁盘

### 3. 文件系统的基本功能

#### （1）五个基本功能

* + 文件的结构及有关存取方法
  + 文件的目录结构和有关处理
  + 文件存储空间的管理
  + 文件的共享和存取控制
  + 文件操作和使用

#### （2）文件系统要实现的功能模块

* 文件的分块存储：与外存的存储块相配合；
* I/O缓冲和调度：性能优化；
* 文件定位：在外存上查找文件的各个存储块；
* 外存存储空间管理：如分配和释放。主要针对可改写的外存如磁盘；
* 外存设备访问和控制：包括由设备驱动程序支持的各种基本文件系统如硬盘，软盘，CD ROM等。

#### （3）文件系统的接口

为方便用户使用文件系统，文件系统通常向用户提供两种类型的接口：

* 命令接口。这是指作为用户与文件系统交互的接口。 用户可通过键盘终端键入命令，取得文件系统的服务。
* 程序接口。这是指作为用户程序与文件系统的接口。 用户程序可通过系统调用来取得文件系统的服务。

## （二）文件的结构和存取法

* 文件结构分别从以下观点研究和设计文件的结构和存取方法：

1）用户观点 研究用户“思维”中的抽象文件

即逻辑文件。

2）系统观点 研究驻留在设备介质的实际文件

即物理文件。

* 文件系统的重要作用之一：用户逻辑文件和相应设备上的物理文件之间建立映射，实现二者之间的转换。
* 文件的存取方法是由文件的性质和用户使用文件的要求决定的。

### 1. 文件的逻辑结构

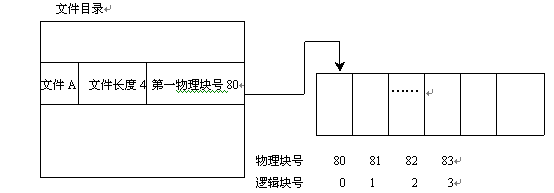
* 逻辑结构：从用户角度看文件，研究文件的组织形式
* 无结构的流式文件
  + 基本信息单位是字节或字，其长度是所含字节的数量。
  + 优点：节省存储空间，无需额外的说明和控制信息。
* 有结构的记录式文件：由若干个记录组成，文件中的记录可按顺序编号为记录1，记录2，……，记录n。
  + 定长记录文件：文件中所有记录的长度相等。文件的长度为记录个数与记录长度的积。
  + 变长记录文件：文件中的记录长度不相等。文件长度为所有记录长度之和。

### 2. 文件的物理结构

* 文件的物理结构是指文件在物理存储介质上的存储结构。
* 为了有效的分配文件存储器的空间，通常把它们分成若干块，并以块为单位进行分配和传送，称为物理块。而块中的信息称为物理记录。
* 基本的文件物理结构
  + 连续结构
  + 链接结构
  + 索引文件

#### （1）连续结构

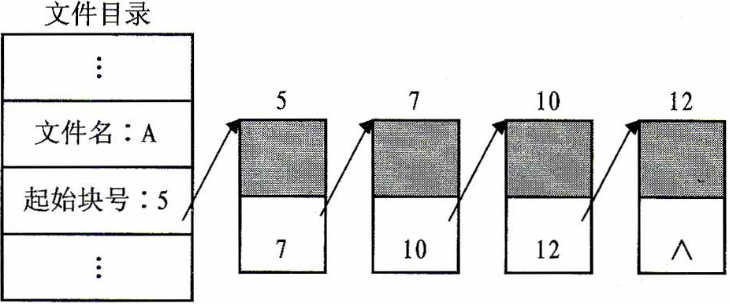
* 一个逻辑文件的信息存放在存储器上的相邻物理块中，该文件为连续文件，这样的结构称为连续结构。



* 连续结构的优缺点
* 优点：
  + 顺序存取速度快，所需的磁盘寻道次数和寻道时间最少。知道文件存储的起始块号和文件块数，就可以立即找到所需要的信息。
  + 简单，支持顺序存取和随机存取。
* 缺点：
  + 在建立连续结构文件时，要求用户给出文件的最大长度，以便系统分配足够的存储空间，但这个有时候难以办到；
  + 不便记录增删操作，一般只能在末端进行。

#### （2）串连结构〈链接结构〉

* 在每个物理块中设置一指针，指向该文件的下一个物理块号，文件的末尾块存放结束标记“NULL”。



##### 链接结构的优缺点

* 优点：
  + 文件可以动态扩充，也不必事先提出文件的最大长度。
  + 由于不连续分配，不存在外部碎片问题，所以不会造成几块连续区域的浪费。
  + 有利于文件插入和删除
* 缺点：
  + 存取速度慢，不适于随机存取，只适合顺序存取，
  + 每块设置链接字破坏物理信息的完整性
  + 链接指针占用一定的空间

##### 链接结构的变形

文件分配表(FAT)

* + 将盘块中的链接字按盘块号的顺序集中起来，构成盘文件映射表/文件分配表FAT 。

FAT实例

* 在MS-DOS和Windows系统中，文件的物理结构使用的是FAT（File Allocation Table）结构。
* 将磁盘空间划分为块，每块大小为扇区的整数倍。在FAT文件系统中块称为簇
* 一个磁盘分区能分为多少簇则FAT就有多少表项

#### （3）索引文件

* 为文件建立一张索引表，每个记录设置一个表项。索引表按记录关键字排序，本身是顺序文件。在对索引文件进行检索的时候，首先按照顺序文件检索方法查找索引表，从中找到相关表项，然后直接访问该记录。
* 当文件较大的时候，索引表也会较大。如果索引表的大小超过一个物理块，索引表的存取就成为新问题。一种较好的办法是采取间接索引，甚至多重索引。
* 索引文件既可以满足文件动态增长的要求，也可以较方便和迅速的实现随机存取。既适用顺序存取也适合随机存取。缺点是索引文件本身增加了存储开销，而且多重索引的访问时间开销也较大。

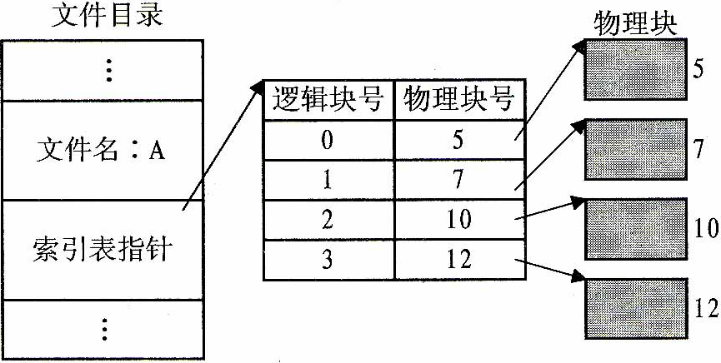


图 9 索引文件

* 索引结构的特点
* 优点：
  + 保持了链接结构的优点，又解决了其缺点：
  + 即能顺序存取，又能随机存取
  + 满足了文件动态增长、插入删除的要求
  + 能充分利用外存空间
* 缺点：
  + 索引表本身带来了系统开销，如：内外存空间，存取时间

#### （4）物理结构比较

* 从查寻时间来看
  + 连续文件最快, 索引文件和文件映照次之, 串联文件最慢
* 从空间开销来看
  + 连续文件不需要额外的空间开销; 串联文件的每个物理块中需要存放链接字; 文件映照需存放文件映照表; 索引文件为每个文件建立一张索引表。 
* 从适宜设备和存取方法来看
  + 连续文件可用于磁带和磁盘；串联文件、索引文件只适用于磁盘；串联文件只适合顺序存取；而索引文件和磁盘上的连续文件，除了能进行顺序存取外，也能实现随机存取。
* 从文件增删来看
  + 连续文件不能动态增长,其他三种都可较容易实现文件的动态改变。

### 3. 文件的存取方法

* 存取方法：读写文件存储器上的一个物理块的方法，通常有3种方法。
  + 顺序存取法：严格按文件信息单位排列的顺序依次存取。
  + 直接存取法：也称随即存取法，每次存取操作时必须先确定存取的位置。对流式文件或定长记录的文件比较容易确定存取位置；对不定长的记录式文件比较麻烦。
  + 按键存取法：文件的组织按照逻辑记录中的某个数据项的内容来存放，根据记录内容进行存取。

### 4. 存储设备、文件物理结构和存取方法的关系\*\*



## （三）文件的目录

* 文件目录组织的目标
* 基本原则：能方便而迅速地对目录进行检索，从而准确地找到所需文件。
* 对目录管理的要求如下：
  + 实现“按名存取”。
  + 提高对目录的检索速度。
  + 文件共享。
  + 允许文件重名。

### 1. 文件控制块FCB

* 文件控制块是操作系统为管理文件而设置的数据结构，存放了为管理文件所需的所有有关信息。
* 文件控制块与文件一一对应，是文件存在的标志
* 文件控制块的内容：
  + 基本信息类：① 文件名 ； ② 文件物理位置 ； ③ 文件逻辑结构 ；④ 文件的物理结构
  + 存取控制信息类
  + 使用信息类



图 10 **MS-DOS的文件控制快**

### 2. 文件目录与目录文件

* 文件目录：把所有的FCB组织在一起，就构成了文件目录，即文件控制块的有序集合
* 目录项：构成文件目录的项目（一个FCB就是目录表中的一个目录项）
* 目录文件：为了实现对文件目录的管理，通常将文件目录以文件的形式保存在外存，这个文件就叫目录文件。

### 3. 简单的文件目录

#### （1）一级目录

整个目录组织是一个线性结构，系统中的所有文件都建立在一张目录表中。它主要用于单用户操作系统。

#### （2）简单目录结构的优缺点

* 优点：
  + 结构简单、清晰，便于维护和查找。
  + 可实现按名存取。
* 缺点：
  + 查找速度慢：为查找一个文件的目录, 平均需查找目录表的一半, 若是大型目录表, 则搜索效率低下。
  + 不允许重名：文件名和文件体有一一对应关系
  + 不便于实现文件共享

### 4. 二级目录

#### （1）二级目录

在根目录下，每个用户对应一个目录（第二级目录）；在用户目录下是该用户的文件，而不再有下级目录。适用于多用户系统，各用户可有自己的专用目录。

#### （2）二级目录的优缺点

优点：

* 提高了检索目录的速度
* 在不同的用户目录中，可以使用相同的文件名。
* 不同用户还可使用不同的文件名来访问系统中的同一个共享文件

缺点：

* 缺乏灵活性，特别是不能反映现实世界中多层次的关系。

### 5. 多级目录

* 多级目录：或称为树状目录(tree-like)。在文件数目较多时，便于系统和用户将文件分散管理。
* 多级目录结构由根目录和各级目录组成，为管理上的方便，除根目录外，其它各级目录均以文件的形式组成目录文件。
* 目录文件中的每个目录项可以对应一个目录文件，也可以对应一个数据文件。如此类推，就形成多级目录结构。
* 各级目录文件称中间结点，用方框表示。数据文件称为叶结点，用圆圈表示。

### 6. 路径和工作目录

* 路径名(path name)：在多级目录结构中一个文件的唯一标识不再是文件名，而是从根结点开始，经过一个或多个中间结点，到达某个叶结点的一条路径。称这条路径为文件的路径名，它是文件的唯一标识。
* 路径名由根目录和所经过的目录名和文件名以及分隔符组成，通常使用分隔符 /。
* 当前目录(Current Directory)：把当前使用的文件所在的目录指定为工作目录(或称当前目录)。
* 绝对路径名(absolute path name)：指由根目录开始的路径名。
* 相对路径名(relative path name)：指从当前工作目录开始的路径名。

### 7. 索引结点

* 索引结点的引入：改进的多级目录
* 为了提高目录检索速度，可把目录中的文件说明（文件描述符）信息分成两个部分：
  + 符号文件目录：由文件名和文件内部标识组成的树状结构，按文件名排序；
  + 基本文件目录（索引节点目录）：由其余所有文件说明信息组成的线性结构，按文件内部标识排序；
* UNIX：i节点（索引节点）

### 8. 增加和删除目录

**（1）不删除非空目录**

当目录(文件)不空时，不能将其删除，而为了删除一个非空目录，必须先删除目录中的所有文件，使之先成为空目录， 后再予以删除。如果目录中还包含有子目录，还必须采取递归调用方式来将其删除， 在MS-DOS中就是采用这种删除方式。

**（2）可删除非空目录**

当要删除一目录时，如果在该目录中还包含有文件，则目录中的所有文件和子目录也同时被删除。

### 9. 目录查询技术

（1）线性检索法

（2）Hash方法

### 10. 文件别名的实现：文件共享

* 提供文件共享的方法有两种：
  + 各用户通过唯一的共享文件的路径名访问共享文件（该方法的访问速度慢，适用于不经常访问的文件共享），
  + 利用多个目录中的不同文件名来描述同一共享文件（即文件别名，该方法的访问速度快，但会影响文件系统的树状结构，适用于经常访问的文件共享，同时存在一定的限制）。
* 文件别名的实现方法有以下两种：
  + 基于索引结点
  + 基于符号链接

#### （1）基于索引结点(index node)的文件别名

也称为硬链接（hard link）；通过多个文件名链接(link)到同一个索引结点，可建立同一个文件的多个彼此平等的别名。别名的数目记录在索引结点的链接计数中，若其减至0，则文件被删除。

* UNIX举例："ln source target ; rm source"则该文件还存在，文件名为target；
* 限制：不能跨越不同文件卷；通常不适用于目录，否则由树状变为网状。

#### （2）基于符号链接(symbolic link, shortcut)的文件别名

它是一种特殊类型的文件，其内容是到另一个目录或文件路径的链接。建立符号链接文件，并不影响原文件，实际上它们各是一个文件。可以建立任意的别名关系，甚至原文件是在其他计算机上。

* UNIX举例："ln -s a b ; rm a"则文件a不存在，b能被控制但无法访问；
* 缺点：空间和时间开销更大。如果设置不当，上下级目录关系可能会形成环状。

#### （3）在文件间建立连接ln

* ln命令用来建立硬连接和符号连接。硬连接是一个文件的额外的名字，没有源文件，硬连接便不能存在。而对于符号连接，当原文件被删除后，符号连接仍然存在但无法访问。
* -s 建立一个符号连接而不是硬连接
* -d 建立目录的硬连接
* 硬链接：
  + 原文件名和链接文件名都指向相同的物理位址。
  + 目录不能有硬连接；硬连接不能跨越文件系统（不能跨越不同的分区）。
  + 文件在磁盘中只有一个复制，以节省硬盘空间。
  + 由于删除文件要在同一个索引节点属于唯一的连接时才能成功，因此可以防止不必要的误删除。
* 符号链接：
  + 用ln -s命令创建文件的符号连接；
  + 符号连接是Linux特殊文件的一种，作为一个文件，它的资料是它所连接的文件的路径名。类似于Windows下的快捷方式。
  + 可以删除原有的文件而保存链接文件，没有防止误删除功能。

## （四）文件系统的使用

* 在文件系统中提供对文件的各种操作，这些操作方便、灵活地使用文件及文件系统，形式分别为：系统调用或命令
* 主要操作
  + 提供设置和修改对用户文件存取权限
  + 提供建立、修改、改变、删除目录的服务
  + 提供文件共享，设置访问路径的服务
  + 提供创建、打开、读、写、关闭、撤消文件等服务
  + 文件系统维护
  + 文件系统的转储和恢复

（1）建立文件 （2）打开文件 （3）关闭文件 （4）删除文件 （5）指针定位

（6）读文件 （7）写文件 （8）文件连接(LINK) （9）复制文件 （10）目录的操作

## （五）文件存储空间的管理

### 1. 文件存储空间分配(file allocation)

#### （1）新创建文件的存储空间（文件长度）分配方法

* 预分配(preallocation)：创建时(这时已知文件长度)一次分配指定的存储空间，如文件复制时的目标文件。
* 动态分配(dynamic allocation)：需要存储空间时才分配（创建时无法确定文件长度），如写入数据到文件。

#### （2）文件存储单位：簇（cluster）（扇区的整数倍）

* 簇的大小：大到能容纳整个文件，小到一个外存存储块；
  + 簇较大：提高I/O访问性能，减小管理开销；但簇内碎片浪费问题较严重；
  + 簇较小：簇内的碎片浪费较小，特别是大量小文件时有利；但存在簇编号空间不够的问题（如FAT12、16、32）；
* 文件巻容量与簇大小的关系
  + 文件卷容量越大，若簇的总数保持不变即簇编号所需位数保持不变，则簇越大。
  + 文件卷容量越大，若簇大小不变，则簇总数越多，相应簇编号所需位数越多。

#### （3）文件存储分配数据结构

* 连续分配(contiguous)：只需记录第一个簇的位置，适用于预分配方法。可以通过紧缩(compact)将外存空闲空间合并成连续的区域。
* 链式分配(chained)：在每个簇中有指向下一个簇的指针。可以通过合并(consolidation)将一个文件的各个簇连续存放，以提高I/O访问性能。
* 索引分配(indexed)：文件的第一个簇中记录了该文件的其他簇的位置。可以每处存放一个簇或连续多个簇（只需在索引中记录连续簇的数目）。

### 2. 内存中所需的表目

* （1）系统打开文件表（整个系统一张）
  + 放在内存，用于保存已打开文件的FCB
  + 此外，文件号，共享计数，修改标志
* （2）用户打开文件表（每个进程一个）
  + 文件描述符，打开方式，读写指针，系统打开文件表入口
  + 进程的PCB中，记录了用户打开文件表的位置
* （3）用户打开文件表与系统打开文件表之间的关系
  + 用户打开文件表指向系统打开文件表
  + 如果多个进程共享同一个文件，则多个用户打开文件表目对应系统打开文件表的同一入口

### 3. 外存空闲空间的管理 (free space management)

* 存储空间管理应解决的问题：
  + 如何登记空闲区的分布情况
  + 如何按需要给一个文件分配存储空间
  + 当某一文件或某一部分不再需要保留时，如何收回它所占用的存储空间
* 常用技术：
  + 空白文件目录
  + 空白块链
  + 位示图

#### （1）空白文件目录

* 辅存上的一片连续的空闲区, 可视为一个空白文件, 系统设置一张空白文件目录来记录辅存上所有空闲块的信息。每个表目存放一个空白文件的信息, 包括该空白文件第一个空闲块号、 空闲块个数、该文件所有空闲块号等信息。
* 这种方法适合于连续文件结构。但此方法有两个明显的缺点: 
  + (1）如果文件太大, 那么在空白文件目录中将没有合适的空白文件能分配给它, 尽管这些空白文件的总和能满足需求。 
  + (2） 经过多次分配和回收，空白文件目录中的小空白文件越来越多，很难分配出去，形成碎片。

#### （2）空白块链

* 该方法把所有的空闲块链接在一起, 形成一个空闲块链表。释放和分配空白快都可以在链首处进行。

##### 空闲链表法的优缺点

* + （1） 可实现不连续分配。 
  + （2） 节省了存储开销。
  + （3） 系统开销大。
  + （4） 对于大型文件系统, 空闲链将会太长。

##### 空闲块的分配与回收

* 空闲块的分配
  + 从栈顶取出一空闲盘块号，将与之对应的盘块分配给用户，然后将栈顶指针下移一格
  + 若是最后一个盘块，将栈底盘块号所对应盘块的内容读入栈中，作为新的盘块号栈的内容，并把原栈底对应的盘块分配出去
* 空闲块的回收
  + 将回收盘块的盘块号记入空闲盘块号栈的顶部，并执行空闲盘块数加1操作
  + 当栈中空闲盘块号数目已达N时， 表示栈已满，便将现有栈中的N个盘块号， 记入新回收的盘块中，再将其盘块号作为新栈底

#### （3）位示图（Bit Map）

* 用一串二进制位反映磁盘空间分配使用情况, 每个物理块对应一位, 分配物理块为1，否则为0
* 申请物理块时，可以在位示图中查找为0的位，返回对应物理块号；归还时；将对应位转置0
* 描述能力强，适合各种物理结构

### 4. 文件卷

* 磁盘分区(partition)：通常把一个物理磁盘的存储空间划分为几个相互独立的部分，称为“分区”。
* 文件卷(volume)：或称为"逻辑驱动器(logical drive)"。在同一个文件卷中使用同一份管理数据进行文件分配和外存空闲空间管理，而在不同的文件卷中使用相互独立的管理数据。
  + 一个文件不能分散存放在多个文件卷中，其最大长度不超过所在文件卷的容量。
  + 通常一个文件卷只能存放在一个物理外设上（并不绝对），如一个磁盘分区或一盘磁带。
* 格式化(format)：在一个文件卷上建立文件系统，即：
  + 建立并初始化用于进行文件分配和外存空闲空间管理的管理数据。
  + 通常，进行格式化操作使得一个文件卷上原有的文件都被删除。
* 磁盘交叉存储(disk interleaving)：将一个文件卷的存储块依次分散在多个磁盘上。如4个磁盘，则磁盘0上是文件卷块0, 4, 8, …，磁盘1上是文件卷块1, 5, 9, …。
  + 优点：提高I/O效率。
  + 需要相应硬件设备

## （六）文件的共享与保护

* 文件共享的目的：
  + 节省存储空间
  + 进程间通过文件交换信息
* 共享和保护是一个问题的两个方面
  + 共享：一个或一部分文件由事先规定的某些用户共同使用
  + 保护：文件不得被未经文件主授权的任何用户使用

### 1. 文件共享

#### （1）文件共享的方式

* 同名共享：
  + 各个用户使用同一个文件名（包括其路径）来访问某一文件。
* 异名共享：
  + 各个用户使用各自不同的文件名来访问某个文件。
  + 异名共享所采用的方法称为文件的勾链：

1）基于索引点的共享方法

容许目录项链接到目录树中任一节点上

2）基于符号链的共享方法

只容许链接到数据文件的叶子节点上

##### 基于索引点的共享（硬链接）

* 文件的目录结构由两部分构成：目录项和索引节点。其中目录项由文件名和索引节点号组成。
* 通过多个文件名链接(link)到同一个索引结点，可建立同一个文件的多个彼此平等的别名。
* 共享数目记录在索引结点的链接计数中，若其减至0，则文件被删除。

##### 基于符号链的共享（软链接）

* 文件主才拥有指向其索引结点的指针；而共享该文件的其他用户，则只有该文件的路径名，并不拥有指向其索引结点的指针。
* 文件的拥有者把一个共享文件删除后， 其他用户试图通过符号链去访问一个已被删除的共享文件时，会因系统找不到该文件而使访问失败，于是再将符号链删除。
* 优势：不同的文件系统、计算机网络环境下可用
* 问题：系统开销大

#### （2）文件的打开结构共享

* 三部分组成：
  + 进程打开文件表
  + 系统打开文件表
  + 内存inode
* 父子进程打开文件的共享
* 同名或异名打开文件的共享

### 2. 文件的保护

* 对拥有权限的用户，应该让其进行相应操作，否则应禁止。
  + 对用户进行分类
  + 对访问权限分类
  + 用访问控制矩阵实现文件保护
  + 存取控制表实现文件保护
  + 用户权限表实现文件保护
  + 用口令实现文件保护

#### （1）对用户进行分类

* 按用户对文件访问权力的差别把用户分成几类，然后对每个文件规定各类用户的存取权限。通常将用户分成三类：
  + 文件主
  + 文件主的同组用户或合作用户
  + 其它用户

#### （2）对访问权限分类

* 对文件的访问系统首先要检查访问权限，只允许合法的用户访问。文件的存取权限一般有以下几种：
  + 仅允许执行 (E)。
  + 仅允许读 (R)。
  + 仅允许写 (W)
  + 仅允许在文件尾写 (A)
  + 仅允许对文件进行修改（U）
  + 允许改变文件的存取权限（C）
  + 允许取消文件（D）
* 这几种权限可进行适当的组合。

#### （3）用访问控制矩阵实现文件保护

* 一维代表所有用户，一维代表系统中的所有文件。
* 优点：一目了然
* 缺点：矩阵往往过大。

#### （4）用存取控制表实现文件保护（一维向量的形式）

#### （5）用户权限表实现文件保护

#### （6）用口令实现文件保护

* 存取控制表、用户权限表都将占据大量存储空间, 可采用另一种较简单的方法：口令。
* 用户为自己的每一个文件设置一个口令, 存放在文件的FCB中。 任何用户要存取该文件, 都必须提供和FCB中一致的口令, 才有权存取。
* 优点：简便
* 缺点：
  + 保护级别少（可访问和不可访问）
  + 保密性差。
  + 不易改变存取控制权限。

#### （7）用密钥（Encryption）实现文件保护

* 在文件建立保存时, 加密程序根据用户提供的代码键对文件进行编码加密, 在读取文件时, 用户提供相同的代码键, 解密程序根据该代码键对加密文件进行译码解密, 恢复为源文件。
* 只有知道代码键的用户才能正确访问文件, 且代码键不存放在系统中, 故该方法保密性很强。但耗费大量编码、译码时间, 系统开销大而且降低了访问速度。
* 一般可将几种安全控制手段综合使用。

# 四、设备管理（输入/输出系统）

## （一）概述

### 1. 设备的分类

（1）按传输速率分

* + 低速设备
  + 中速设备
  + 高速设备

（2）按信息交换的单位分类

* + 字符设备：速率较低、中断驱动。
  + 块设备：速率高（几兆）、可随机访问任一块、DMA方式驱动。

（3）按资源管理方式分类

* + 独占型设备
  + 共享型设备
  + 虚拟设备

（4）按外部设备的从属关系分

* + 系统设备
  + 用户设备

### 2. 设备管理的目标

**（1）实现设备独立性**

**（2）提高设备利用率**

**（3）设备的统一管理**

* + 速度
  + 传递单位
  + 操作方法和特性
  + 出错条件

### 3. 设备管理的功能

**（1）监视系统中所有设备的状态**

**（2）设备的分配**

**（3）I/O控制**

### 4. 设备管理数据结构

* 设备控制块(DCB)
  + 设备标识符
  + 设备属性
  + 设备I/O总线地址
  + 设备状态
  + 等待队列指针

## （二）I/O系统硬件特点

* 设备组成
* （1）物理部分：物理设备机械部分，设备本身
* （2）电子部分：设备控制器
* 电子部分（设备控制器）完成的工作
  + （端口）地址译码
  + 接受主机发来的数据和控制信号，向主机发送数据和状态信号
  + 将计算机的数字信号转换成机械部分能识别的模拟信号，或反之
  + 实现设备内部硬件缓冲、数据加工等提高性能或增强功能

### 1. 设备接口

* 操作系统将命令写入控制器的接口寄存器（或接口缓冲区）中，以实现输入／输出，并从接口寄存器读取状态信息或结果信息
* 接口内的组成部分：
  + 设备识别线路
  + 命令寄存器
  + 数据缓冲寄存器（输入/输出）
  + 控制寄存器
  + 状态寄存器

### 2. 端口编址方法

* 内存映像编址（内存映像I/O模式）：分配给系统中所有端口的地址空间与内存的地址空间统一编址
  + 优点
    - 凡是可对存储器操作的指令都可对I/O端口操作
    - 不需要专门的I/O指令
    - I/O端口可占有较大的地址空间
  + 缺点：占用内存空间
* I/O独立编址（I/O专用指令）：分配给系统中所有端口的地址空间完全独立，主机使用专门的I/O指令对端口进行操作
  + 优点
    - 外部设备不占用内存的地址空间
    - 程序设计时易于区分是对内存操作还是对I/O端口操作
  + 缺点：对I/O端口操作的指令类型少，操作不灵活

## （三）I/O控制方式

### 1. 程序直接控制方式(循环查询I/O方式)

浪费大量CPU时间

### 2. I/O中断方式

* I/O控制器能发中断
* 工作过程

同前相比，CPU利用率大大提高。

缺点：数据缓冲寄存器每满一次都要中断一次，如果设备较多时，中断次数会很多，使CPU的计算时间大大减少。

### 3. 直接存储器存取方式(direct memory access，DMA)

#### （1）工作过程

* 1、当进程要求输入时，把要求传送的内存始址（M）和要传的字节数送入DMA的内存地址寄存器和传送字数寄存器
* 2、把启动位置1。设备开始工作。进程（A）挂起。调度另一进程（B）
* 3、一批数据输入完成后，DMA中断B，转向中断处理程序。
* 4、中断处理程序唤醒A，返回B的断点继续执行。
* 5、以后OS调度A运行时，A从M处取数据处理。

#### （2）DMA方式与中断的主要区别

* 中断时机
  + 中断方式是在数据缓冲寄存器满后，发中断请求，CPU进行中断处理
  + DMA方式则是在所要求传送的数据块全部传送结束时要求CPU进行中断处理
* 数据传输
  + 中断方式的数据传送由CPU控制完成
  + DMA方式是在DMA控制器的控制下不经过CPU控制完成的

### 4. I/O通道控制方式(I/O channel control)

#### （1）概述

* 通道：在CPU的控制下独立地执行通道程序，对外部设备的I/O操作进行控制，以实现内存与外设之间成批的数据交换。
* 通道程序是由通道指令组成，一个通道可以分时的方式执行几道程序。每道程序控制一台外部设备，因此每道通道程序称为子通道。

#### （2）通道的种类

* 字节多路通道
* 数据选择通道
* 数据多路通道
* 在一大型系统中可以同时存在这三种类型的通道以便控制各种不同类型的设备。

#### （3）通道的工作过程

* 某进程在运行过程中，若提出了I/O请求，则通过系统调用进入操作系统。系统首先为I/O操作分配通道和外设，然后按I/O请求生成通道程序并存入内存，把起始地址送入通道的首地址寄存器（CAW），接着CPU发出启动通道的指令。
* 中央处理机启动通道后，通道的工作过程为：
  + 根据CAW，从内存取出通道指令，送入通道控制字寄存器（CCW），并修改CAW，使其指向下一条通道指令。
  + 执行CCW中的通道指令，进行实际的I/O操作，执行完毕后如果还有下一条指令，则返回前一步，否则转下一步。
  + 发出中断信号通知CPU通道程序已执行完成。

### 5. 外围处理机输入输出方式(peripheral processor unit)

## （四）缓冲技术

### 1. 缓冲的引入

* + 缓和CPU与I/O设备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。
  + 提高CPU和I/O设备之间的并行性。
* 用缓冲技术来匹配CPU与设备速度的差异和负荷的不均匀，从而提高处理机与外设的并行程度。

### 2. 常用的缓冲技术

* 硬件缓冲器：在设备控制器中有硬件缓冲器，通常容量较小
* 软件缓冲技术：由缓冲区和对缓冲区的管理两部分组成
  + 1、单缓冲
  + 2、双缓冲
  + 3、环形缓冲
  + 4、缓冲池

#### （1）单缓冲

* 最简单的一种缓冲形式。当进程发出一I/O请求时，OS为之分配一缓冲区。
* 对于输入：设备先将数据送入缓冲区，OS再将数据传给进程。
* 对于输出：进程先将数据传入缓冲区，OS再将数据送出到设备。
* 思考：单缓冲能加快进程的执行速度吗？

#### （2）双缓冲技术

* 为了加快输入输出速度，引入双缓冲技术。
* 原理：设置两个缓冲区buf1和buf2。读入数据时，首先输入设备向buf1填入数据，然后进程从buf1提取数据，在进程从buf1提取数据的同时。输入设备向buf2中填数据。当buf1取空时，进程又从buf2中提取数据，与此同时输入设备向buf1填数。如此交替使用两个缓冲区，使CPU和设备的并行操作的程度进一步提高。

#### （3）环形缓冲技术

* 在主存中分配一组大小相等的存储区作为缓冲区，并将这些缓冲区链接起来。
* 系统必须考虑到这种方案的约束条件，即IN<>OUT( 初始状态除外)。

#### （4）缓冲池

* 缓冲池由内存中一组大小相等的缓冲区组成
* 缓冲池属于系统资源，由系统进行管理
* 缓冲池中各缓冲区可根据需要组成各种缓冲区队列。
  + ① 空(闲)缓冲区；
  + ② 装满输入数据的缓冲区；
  + ③ 装满输出数据的缓冲区。

## （五）其他技术

### 1. 总线技术

在计算机系统内各种子系统，如CPU、内存、I/O设备等之间，构建公用的信号或数据传输通道，这种可共享的传输通道称为总线。

### 2. USB技术

* USB（Universal Serial Bus）通用串行总线
* 适用于低、中速的外围设备
* USB的传输方式
  + （1）等时传输方式
  + （2）中断传输方式
  + （3）控制传输方式
  + （4）批传输方式

### 3. SCSI接口技术

* 小型计算机系统接口（Small Computer System Interface）
* **IDE与SCSI二者的区别主要在于：**
  + **IDE的工作方式需要CPU的全程参与**
  + **SCSI接口则完全通过独立的高速的SCSI卡来控制数据的读写操作**
* 优缺点

SCSI接口优点：

1. 适应面广，在一块SCSI控制卡上就可以同时挂接15个设备

2. 高性能（具有很多任务、宽带宽及少CPU占用率等特点）

3. 具有外置和内置两种

SCSI接口缺点：

价格昂贵、安装复杂

## （六）磁盘的驱动调度

### 1. 磁盘概述

### 2. 磁盘调度算法

安排访盘顺序，考虑：

* + 公平：一个I/O请求在有限时间内满足
  + 高效：减少设备机械运动所带来的时间浪费

#### （1）先来先服务FCFS ( First-Come, First Served )

* 按访问请求到达的先后次序服务
* 优点：简单，公平；
* 缺点：效率不高，相邻两次请求可能会造成最内到最外的柱面寻道，使磁头反复移动，增加了服务时间，对机械也不利

#### （2）最短寻道时间优先SSTF ( Shortest Seek Time First )

* 优先选择距当前磁头最近的访问请求进行服务，主要考虑寻道优先
* 优点：改善了磁盘平均服务时间；
* 缺点：造成某些访问请求长期等待得不到服务

#### （3）扫描算法（电梯算法）

* **克服了最短寻道优先的缺点，既考虑了距离，同时又考虑了方向**
* **具体做法：当设备无访问请求时，磁头不动；当有访问请求时，磁头按一个方向移动，在移动过程中对遇到的访问请求进行服务，然后判断该方向上是否还有访问请求，如果有则继续扫描；否则改变移动方向，并为经过的访问请求服务，如此反复**

#### （4）循环扫描调度算法CSCAN

* 电梯算法杜绝了饥饿，但当请求对磁道的分布是均匀时，磁头回头，近磁头端的请求很少（因为磁头刚经过），而远端请求较多，这些请求等待时间要长一些。
* 总是从0号柱面开始向里扫描。移动臂到达最后个一个柱面后，立即带动读写磁头快速返回到0号柱面。返回时不为任何的等待访问者服务。返回后可再次进行扫描

#### （5）调度算法的选择

* 实际系统相当普遍采用最短寻道时间优先算法，因为它简单有效，性价比好。
* 扫描算法更适于磁盘负担重的系统。
* 磁盘负担很轻的系统也可以采用先来先服务算法
* 一般要将磁盘调度算法作为操作系统的单独模块编写，利于修改和更换。

### 3. 提高磁盘I/O速度的方法

#### （1）磁盘高速缓存

* 两种方式：
  + 在内存中开辟一个单独的存储空间作为磁盘高速缓存。
  + 把所有未利用的内存空间变为一个缓冲池 ，供分页系统和磁盘I/O共享。
* 置换算法
  + 最近最久未使用LRU、最少使用LFU等。
* 周期性写回：
  + 周期性地强行将已修改盘块写回磁盘。周期一般为几十秒。

#### （2）优化数据分布

##### 优化物理块的分布

* + 物理块连续分配可以减少磁头的移动。
  + 增加物理块的大小也可减少磁头的移动。

##### 优化索引结点的分布

* + 可将索引结点放在中间位置。
  + 进一步可将磁道分组，每组都有索引结点和文件数据

#### （3）其它方法

##### 提前读

* + 在访问文件时经常是顺序访问，因此在读当前块时可以提前读出下一块。
  + 提前读已经被广泛应用：UNIX、OS/2 、 Netware等。

##### 延迟写

* + 修改缓存中的数据后一般应立即写回磁盘，但该盘块可能还会被修改，立即写回会带来很大的开销。
  + 置上延迟写标志。直到该盘块淘汰时或周期性写回时。
  + 延迟写也被广泛应用： UNIX、OS/2 等。

##### 虚拟盘

* + 利用内存仿真磁盘，又称RAM盘。
  + 虚拟盘同磁盘高速缓存的区别：虚拟盘的内容完全由用户控制，用户可见。缓存的内容完全由系统控制，用户不可见。

## （七）设备分配

### 1. 设备分配方式

#### （1）静态分配

在作业级进行的，当一个作业运行之前由系统一次分配满足需要的全部设备，这些设备一直为该作业占用，直到作业撤消。这种分配不会出现死锁，但设备的利用效率较低。

#### （2）动态分配

在进程运行的过程中进行的，当进程需要使用设备时，通过系统调用命令向系统提出设备请求，系统按一定的分配策略给进程分配所需设备，一旦使用完毕立即释放。显然这种分配方式有利于提高设备的使用效率，但会出现死锁，这是应力求避免的。

### 2. 设备分配算法

* 先请求先服务
* 2优先级高的优先服务

### 3. 设备分配技术

* 根据设备的特性把设备分成独占设备、共享设备和虚拟设备三种。
* 针对这三种设备采用三种分配技术:
  + 独享分配
  + 共享分配
  + 虚拟分配

## （八）SPOOLing系统

* Simultaneaus Periphernal Operations On-Line(外部设备同时联机操作）。
* 在联机情况下实现的同时外围操作称为SPOOLing，也称为假脱机操作。
* 虚拟设备，资源转换技术
* SPOOLing系统的组成
  + 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

### 1. SPOOLing系统工作原理

* 作业执行前预先将程序和数据输入到输入井中
* 作业运行后，使用数据时，从输入井中取出
* 作业执行不必直接启动外设输出数据，只需将这些数据写入输出井中
* 作业全部运行完毕，再由外设输出全部数据和信息

好处：

* 实现了对作业输入、组织调度和输出的统一管理
* 使外设在CPU直接控制下，与CPU并行工作（假脱机）

注意：SPOOLing只提高设备利用率，缩短用户程序执行时间，并不提高CPU利用率

### 2. SPOOLing系统的特点

1、提高了I/O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

通道：用于控制I/O设备与内存间的数据传输。启动后可独立于CPU运行，实现CPU与I/O的并行。

中断：指CPU在收到外部中断信号后，停止原来工作，转去处理该中断事件，完毕后回到原来断点继续工作。

SPOOLing技术

* Simultaneous Peripheral Operation On-Line ，同时外围设备联机操作--假脱机技术。
* 思想：利用磁盘作缓冲，将输入、计算、输出分别组织成独立的任务流，使I/O和计算真正并行
* SPOOLing系统的特点
  + 提高了I/O速度
  + 将独占设备改造为共享设备
  + 实现了虚拟设备功能