第二、三章 处理器管理

中断技术

- 中断概念、作用（为什么需要中断？）
  - 概念：中断是指程序执行过程中，遇到急需处理的事件时，暂时中止CPU上现行程序的运行，转去执行相应的事件处理程序，待处理完成后再返回原程序被中断处或调度其他程序执行的过程
  - 作用：在中断事件发生后，中断装置能**改变CPU操作的执行顺序**

- 中断分类（强迫中断/自愿中断，硬中断/软中断）
  - 外中断：又称中断，指**来自处理器之外的中断信号**，外中断又分为**可屏蔽中断**和**不可屏蔽中断**；时钟中断，键盘中断，外部设备中断等；高级中断屏蔽低级中断
  - 内中断：又称异常，指**来自处理器内部**，**内中断不可屏蔽**；大部分异常发生在用户态，只有缺页异常发生在内核态；主存奇偶校验错、非法操作码、地址越界、缺页、调试、访管、算术溢出……异常分为故障、陷阱和终止
  - 硬中断vs软中断：中断与异常要通过硬件设施来产生中断请求，称为硬中断；不必由硬件产生中断源而引发的中断称为软中断

- 中断响应基本流程
  - 发现中断源 -> 保护现场（现场信息PSW保存至核心栈）-> 处理中断（用户态 -> 内核态） -> 恢复现场（恢复原运行程序PSW继续执行）

- 中断描述符表(IDT)
  - 中断描述符表（Interrupt Descriptor Table，IDT）
    - 包含256个表项，每个中断/异常都对应一个
    - 每个表项称为一个门描述符(gate descriptor)，作用是把程序控制权转交给中断/异常处理程序
  - IDT的位置由CPU的中断描述符表寄存器idtr确定
    - idtr是一个48位寄存器
    - 高32位是IDT基址，低16位为IDT界限(通常为2k=256*8)

- 关中断vs屏蔽中断
  - 中断屏蔽：指在中断请求产生后，系统用软件方式有选择地封锁部分中断而允许其余部分中断仍能得到响应

    作用：
    - 延迟或禁止某些中断的响应
    - 协调中断响应与中断处理的关系：保证高优先级中断能打断低优先级中断
    - 防止同级中断相互干扰
  - 关中断：CPU执行关中断指令后，不再例行检查中断信号，直到执行开中断指令后才会恢复检查

- 中断上半部分、下半部分

  - 中断处理流程
    - 上半部：中断处理程序
      - 简单快速，执行时禁止部分或全部中断
    - 下半部
      - 执行与中断处理密切相关但中断处理程序本身不执行的工作
      - 执行期间可以响应中断
  - 没有严格的划分规则
    - 如果任务对时间非常敏感，放在中断处理程序中执行
    - 如果任务与硬件相关，放在中断处理程序中执行
    - 如果任务需确保不被其他中断（同级中断）打断，放在中断处理程序中执行
    - 其他任务，考虑放置在下半部分执行

- Linux中断下半部分实现方法（各种方法的对比）

  - 软中断
  - tasklet
  - 工作队列
    - tasklet实现在软中断之上
    - tasklet和软中断也称为可延迟函数

下半部机制	状　态
BH	在2.5中去除
任务队列（task queue）	在2.5中去除
软中断（softirq）	从2.3开始引入
tasklet	从2.3开始引入
工作队列（work queue）	从2.5开始引入

- Windows DPC、APC （作用，区别）

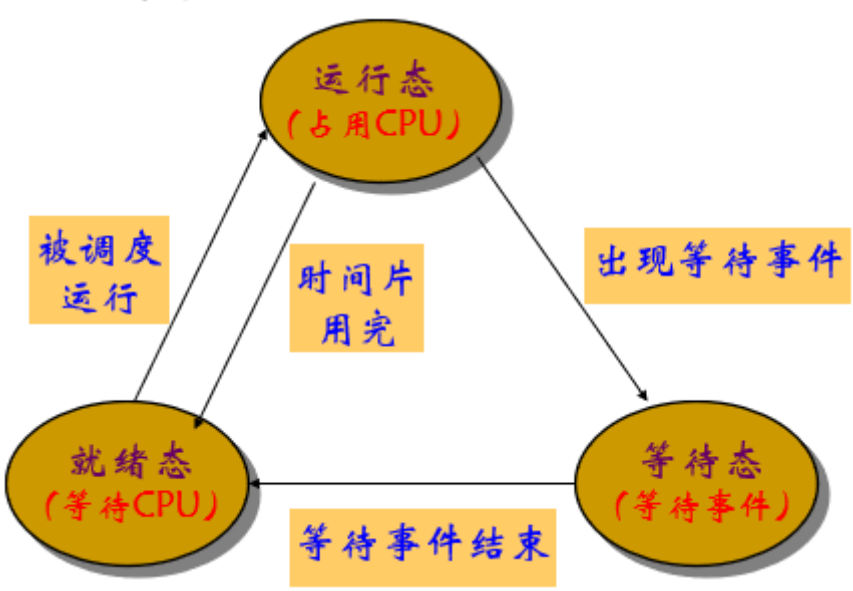
  ？？？什么玩意儿

-进程（线程）

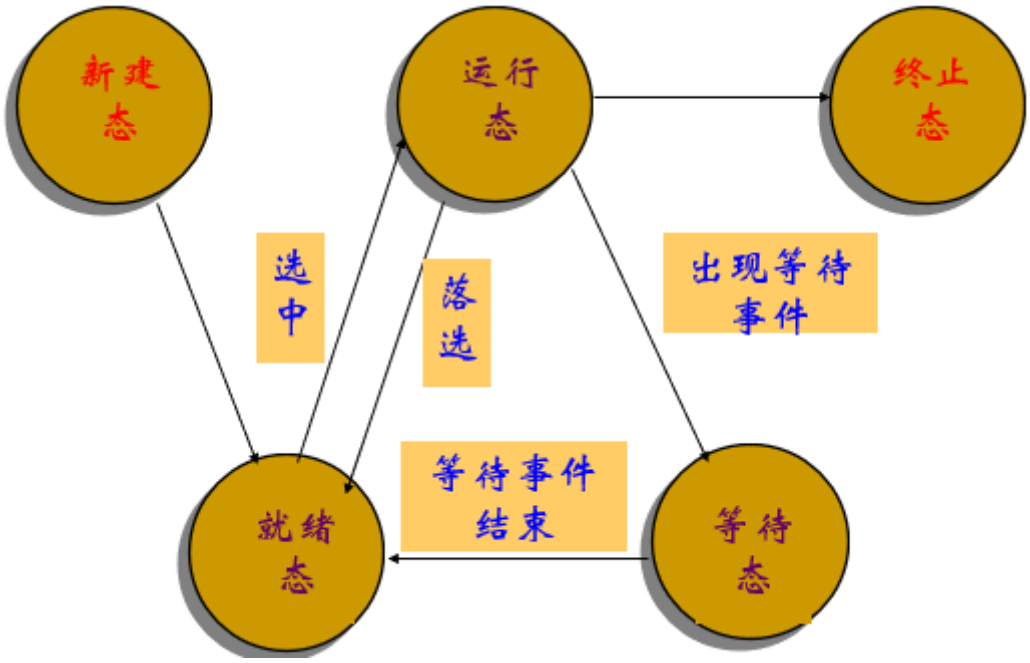
- 进程定义、作用（为什么引入进程？）
  - 定义：程序的一次执行过程，是可并发执行的程序在某个数据集合上的一次计算活动，是系统进行**资源分配**的基本单位
  - 作用：

        - 程序的并发性
        - 资源的共享性
- 进程属性，进程和程序的区别
    - 进程属性：
        - 动态性：是程序的一次执行过程
        - 共享性：同一程序同时运行于不同数据集合上时，构成不同进程
        - 独立性：独立实体，有自己的虚存空间、程序计数器和内部状态
        - 制约性：共享资源或协同工作，产生相互制约关系，必须对进程的执行次序，或相对执行速度加以协调
        - 并发性：单处理器环境并发执行，多处理器环境中并行执行
    - 进程是程序的一次执行过程

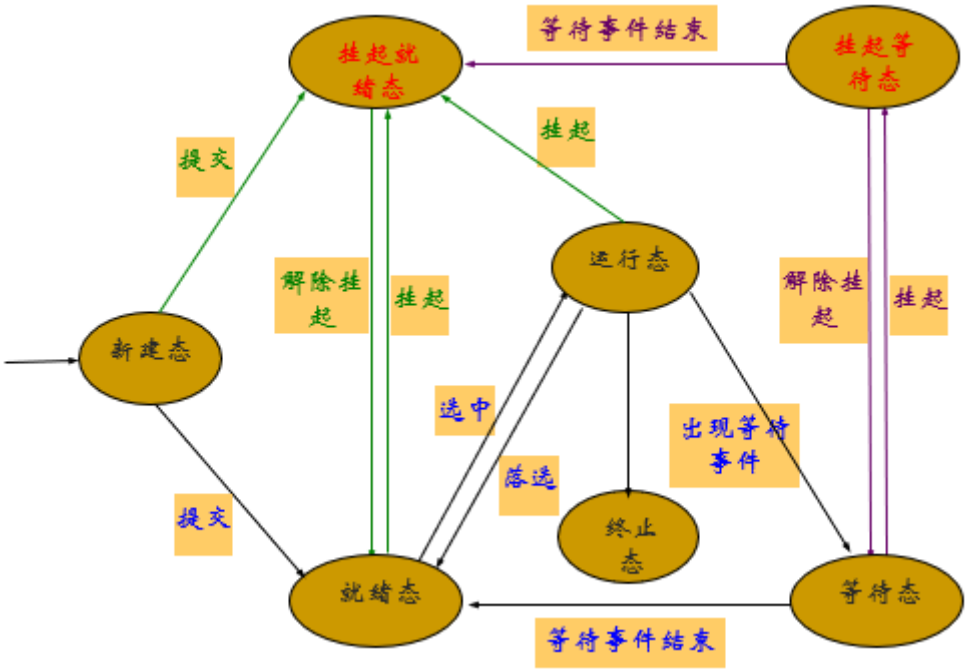
- 进程三态模型、五态模型、七态模型
    - 三态模型



    - 五态模型



    - 七态模型



- 进程的挂起
  - 为了让某些进程暂时不参与低级调度，释放它占有的资源，以平滑系统负荷（即从内存移出到外存？）

    - 进程挂起的原因多种多样，如
      - 内存资源已经不能满足进程运行的要求，此时必须把某些进程挂起(suspend)，对换到磁盘对换区中
      - 可能系统出现故障，需要暂时挂起一些进程，以便故障消除后，再解除挂起并恢复进程运行
      - 用户调试程序过程中，可以请求挂起他的进程，以便进行某种检查和修改

- 进程的组成（代码段、数据段、**进程控制块PCB**）

进程控制块

- 也称为**进程描述符**（process descriptor）
  - 进程存在的唯一标识
  - 操作系统用来记录和刻画进程状态及环境信息的数据结构，是进程动态特征的汇集
  - 操作系统掌握进程的唯一资料结构和管理进程的主要依据
- 进程控制块包含三类信息
  - 标识信息
    - 包括用户外部标识符（用户/用户组标识ID）和系统内部标识号（进程/进程组标识ID）
  - 现场信息
  - 控制信息
    - 用于管理和调度进程，包括进程调度相关信息、进程组成信息、进程族系信息、进程间通信信息、进程段/页表、进程映像在外存中的地址、CPU的占用和使用信息、进程特权信息、资源清单

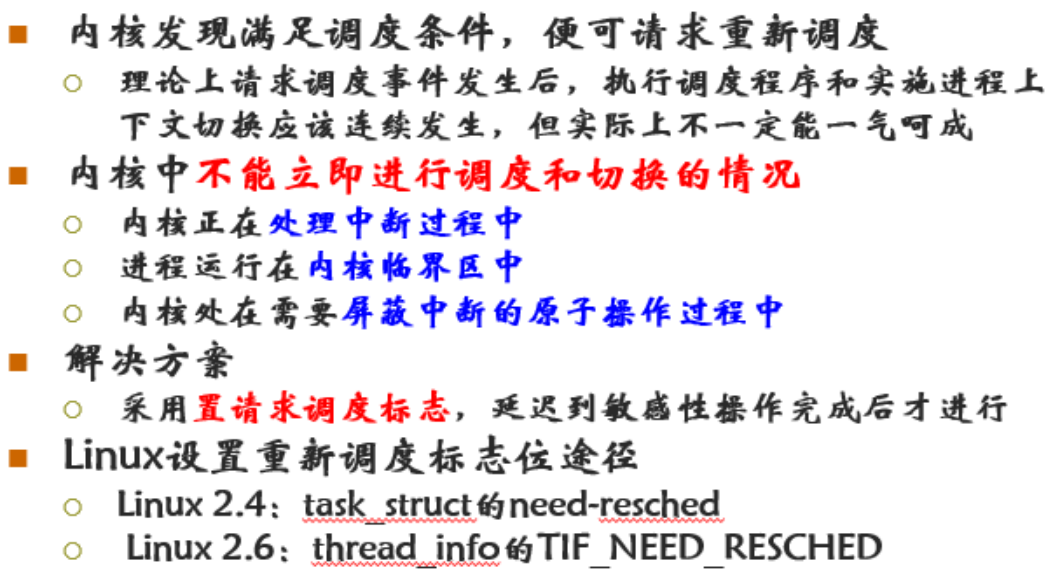
89

- 进程上下文

- 进程上下文是进程**物理实体**和支持进程**运行的环境**
  - 进程在其上下文中执行，当系统调度新进程占有处理器时，新老进程随之发生上下文切换
  - 用户级（用户堆栈、共享存储区）、寄存器、系统级（PCB、主存管理信息、核心栈）
    - 用户级上下文
      - 由程序块（可执行的机器指令序列）、数据块（进程可访问的信息）、共享内存区（进程通信使用的内存区）、用户栈（存放函数调用过程中的信息）组成，**占用进程的虚存空间**
    - 系统级上下文
      - 由进程控制块（进程的状态）、内存管理信息（进程页表或段表）、核心栈（进程内核态运行时的工作区）等操作系统管理进程所需要的信息组成
    - 寄存器上下文
      - 由处理器状态寄存器（进程当前状态）、指令计数器（下一条该执行的指令地址）、栈指针（指向用户栈或核心栈当前地址）、通用寄存器等组成
      - 当进程不处于运行态时，处理器状态信息保存在寄存器上下文中

- 进程切换过程
  - 进程切换是**让处于运行态的进程中断运行，让出处理器**，这时要做一次进程上下文切换
    - 保存老进程的上下文而装入被保护了的新进程的上下文，以便新进程运行
  - 进程切换发生位置：**内核态**

- 进程上下文切换时机（立即切换？）



- 进程切换 vs 模式切换
  - 进程切换≠模式切换，模式切换是用户态和核心态切换，进程切换在核心态完成。进程切换之前一定要有模式切换，模式切换不一定导致进程切换

- Linux进程的虚存映像

  ？？？ 应该不考

- 线程概念、作用（为什么引入线程？）
  - 概念：是进程中能够独立执行的实体；引入线程后，**进程是资源分配的基本单位，线程是调度的基本单位**
  - 作用：引入线程则是为了**减少程序并发执行时所付出的时空开销**，使得并发粒度更细、并发性更好；把进程的两项功能：“独立分配资源”与“被调度分派执行”分离开来

- 处理器调度的层次：低级调度、中级调度、高级调度（分别指什么？）
  - **高级调度**：又称**作业调度**，按一定的原则从外存的作业后备队列中挑选一个作业调入内存，并创建进程。**每个作业只调入一次，调出一次。**作业调入创建PCB，调出撤销PCB；

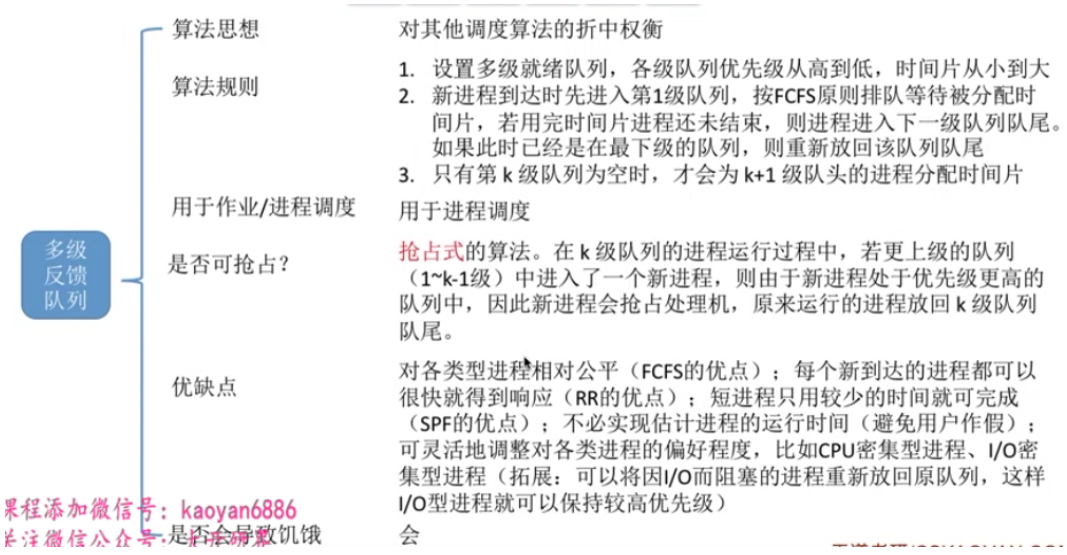
    作业：一个具体任务。作业是任务实体，进程是完成任务的执行实体
  - **中级调度**：按照某种策略决定哪个处于挂起状态的进程重新调入内存；内存不够时，可将某些进程的数据调出至外存（进程状态为**挂起状态**，被挂起的进程PCB被组织成**挂起队列**），等内存空闲或进程需要运行时再重新调入内存。一个进程可能会多次调入、调出内存，**中级调度的频率比高级调度更高**
  - **低级调度**：又称**进程/线程调度**，按某种策略从就绪队列中选取一个进程，将处理机分配给它。是操作系统中**最基本的一种调度**，进程调度的**频率很高**

    进程调度的时机：
    - 当前运行进程**主动放弃**处理机：进程正常终止、发生异常而终止、主动请求阻塞（如：等待I/O）
    - 当前运行进程**被动放弃**处理机：分配时间片用完、有更紧急的事需要处理（如：I/O中断）、优先级更高的进程进入就绪队列

不能进行进程调度与切换的情况：

处理中断的过程中、进程在**操作系统内核程序临界区**中、在**原子操作**过程中（原语）

- 处理器调度算法 （计算题）
  - **CPU利用率**：**CPU有效工作时间/CPU总的运行时间**
  - **周转时间**：**作业完成时间-作业提交时间**
  - 平均周转时间：周转时间之和/作业数
  - 带权周转时间：周转时间/运行时间
  - 平均带权周转时间：带权周转时间之和/作业数
  - **响应时间**：从**交互式进程**提交请求到获得响应的时间间隔
  - 调度算法：
    - FCFS先来先服务：长作业有利
    - SJF最短作业优先：短作业有利，饥饿
    - SRTF最短剩余作业优先：把SJF算法改为抢占式的；如果新作业需要的CPU时间比当前作业剩余所需CPU时间短，SRTF强行赶走当前正在执行作业；饥饿
    - HRRF最高响应比优先：响应比=（等待时间+要求服务时间）/要求服务时间；不会导致饥饿
    - RR时间片轮转：公平轮流的为各个进程服务。按照各进程到达就绪队列的顺序，轮流地让各进程执行一个**时间片**，若进程未在一个时间片内执行完，则剥夺处理机，将进程重新放到就绪队列队尾排队；公平；高频率的进程切换，有一定开销；不区分任务紧急程度
    - 优先级调度：每个作业/进程有各自的优先级，调度时选择优先级最高的作业/进程；抢占式，非抢占式都有；分为静态优先级和动态优先级；静态优先级可能导致饥饿
    - 多级反馈队列：饥饿



- Linux 2.6 O(1)调度算法 (Bitmap, 动态优先级如何更新等)

  ？？？ 应该不考

-同步、通信

- 串行、并发（与并行的区别）
  - 串行：一次运行一个进程

- 并行：多个事件在同一时刻同时发生
- 并发：多个事件在**同一时间间隔内**发生，**宏观上同时发生，微观上交替发生**

- 进程调度中死锁、饥饿的概念，产生原因
  - 死锁：因**争夺资源陷入永远等待**的状态
  - 饥饿：由于其他进程总是优先于它，而**被调度程序无限期地拖延**而不能被执行

- 并发进程之间的关系（独立、竞争、 共享合作、通信合作）
  - 进程互斥：进程互斥指若干个进程因相互争夺独占型资源时所产生的竞争制约关系
  - 进程同步：指两个以上进程基于某个条件来协调它们的活动

- 临界区概念、竞争条件
  - 临界区：并发进程中**与共享变量有关的程序段**
  - 竞争条件：当多个并发进程访问临界资源时，**结果依赖于它们执行的相对速度**
  - 临界区互斥三原则：**互斥使用，有空让进，忙则等待，有限等待，让权等待**
    - 一次至多一个进程能够进入临界区内执行
    - 如果已有进程在临界区，其他试图进入的进程应等待
    - 进入临界区内的进程应在有限时间内退出，以便让等待进程中的一个进入
  - 临界区互斥的软硬件实现方法、与信号量方法实现互斥的差别（各自优缺点、适用场合）
    - 软件方法：
      - Peterson算法：为每个进程设置标志，当标志值为true时表示此进程要求进入临界区，再设置一个指示器turn以指示可以由哪个进程进入临界区；不满足让权等待；实现复杂，效率低下

```
bool inside[2];
inside[0]=false;
inside[1]=false;
enum {0,1} turn;
cobegin
process P0( ) {
 inside[0]=true;
 turn=1;
 while(inside[1]&&turn==1);
 /*临界区*/;
 inside[0]=false;
}
coend
```

本质：
inside声明自己需要使用
turn谦让其他进程使用

```
process P1( ) {
   inside[1]=true;
   turn=0;
   while(inside[0]&&turn==0);
   {临界区};
   inside[1]=false;
}
```

    - 硬件方法：
      - 关中断：进入临界区时关中断，进程退出临界区时开中断；关中断的时间过长会影响性能和系统效率；不适合多处理机系统；这个权力给用户也危险，若用户未开中断...
      - 测试并设置指令TS：不满足让权等待

- TS指令实现进程互斥
  - 把一个临界区与一个布尔型变量s相关联
    - 变量s代表临界资源的状态，把它看成一把锁，s的初值置为true，表示没有进程在临界区内，资源可用
  - 系统利用TS指令实现临界区的上锁和开锁原语操作

```
bool TS(bool &x) {
  if(x) {
      x=false;
      return true;
   }
   else
     return false;
}
```

```
bool s=true;
cobegin
process Pi( ) {//i=1,2,...,n
   while(!TS(s));      /*上锁*/
  /*临界区*/;
   s=true;            /*开锁*/
}
coend
```

本质：TS基于原语单向修改成false，其他进程单向修改为true

- 对换指令XCHG：不满足让权等待
  - 对换指令实现进程互斥

```
bool lock=false;
cobegin
Process Pi( ){ //i=1,2,...,n
   bool keyi=true;
   do {
     SWAP(keyi, lock);
   }while(keyi);          /*上锁*/
   /*临界区*/;
   SWAP(keyi, lock);     /*开锁*/
}
coend
```

本质：
任意时刻只让一个进程拥有false

- 信号量：
  - 一般信号量：解决进程同步问题（前V后P）；P申请资源，V释放资源
    - PV操作是不可中断过程
      - 单CPU系统中通常在屏蔽中断的条件下执行

```
typedef struct semaphore {
          int value;          /*信号量值*/
          struct pcb *list;   /*信号量队列指针*/
};
void P(semaphore &s) {
          s.value--;
          if(s.value<0)
      sleep(s.list);   // 内核函数，阻塞自己，被置成等待信号量s状态
}
void V(semaphore &s) {
          s.value++;
   if(s.value<=0)
      wakeup(s.list);  //内核函数，释放一个等待信号量s的进程
}
```

  - 二值信号量：解决进程互斥问题

- 设s为一个记录型数据结构
  - 一个分量为value，它仅能取值0和1
  - 另一个分量为信号量队列list
- 把二元信号量上的P、V操作记为BP和BV
- BP和BV操作原语的定义如下

```
typedef struct binary_semaphore {
 int value;                  /*value取值0 or 1*/
 struct pcb *list; };
void BP(binary_semaphore &s) {
        if(s.value==1)
            s.value=0;
        else
            sleep(s.list);
}
```

```
void BV(binary_semaphore &s) {
        if(s.list is empty( ))
            s.value=1;
        else
            wakeup(s.list);
}
```

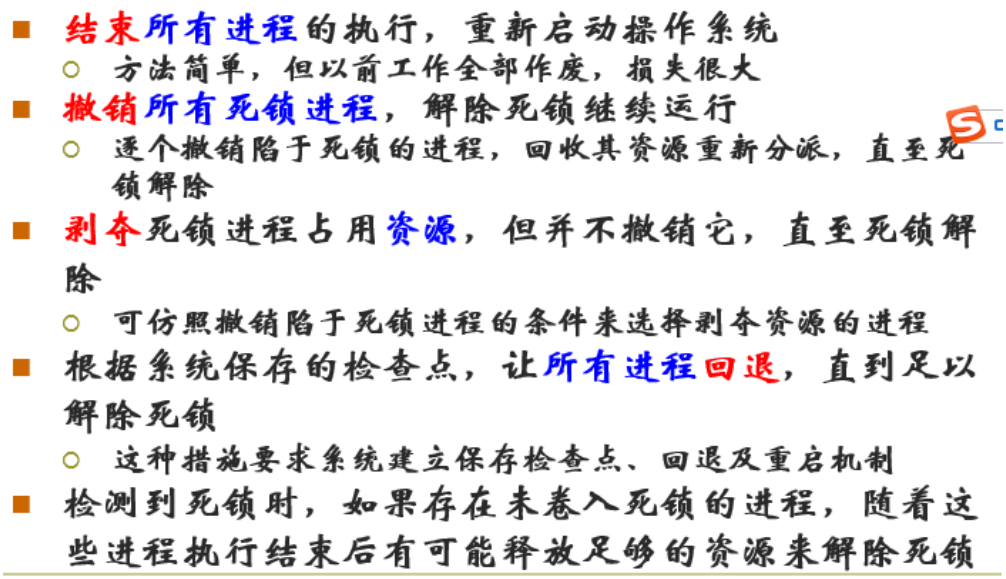
- 管程
  - 把分散在各进程中的临界区集中起来管理，并把共享资源用数据结构抽象地表示，由于临界区是访问共享资源的代码段，建立一个"秘书"程序（管程）管理到来的访问

- 进程通信（IPC）方式及各自原理，适用场合
  - 并发进程之间的交互必须满足两个基本要求：同步和通信；进程同步是一种进程通信
  - 进程通信IPC（InterProcess Communication）:进程之间互相交换信息的工作
  - 信号量
  - 信号通信机制（软中断）
    - 通过发送一个指定信号通知进程某个异步事件发生，迫使进程执行信号处理程序；信号处理完毕后，被中断进程将恢复执行
  - 管道：半双工；发送进程以字符流形式把大量数据送入管道，接收进程从管道中接收数据，所以叫管道通信
  - 共享内存：指两个或多个进程共同拥有一块内存区，该区中的内容可被进程访问
  - 消息传递：在不同进程之间传递数据的机制；基本原语（阻塞/同步问题）：发送消息，接收消息

- 产生死锁的四个必要条件
  - 互斥条件（mutual exclusion）：进程互斥使用资源
  - 占有和等待条件（hold and wait）：进程在请求资源得到满足而等待时，不释放已占有资源
  - 不剥夺条件（no preemption）：又称不可抢占，一个进程不能抢夺其他进程占有的资源
  - 循环等待条件（circular wait）：又称环路条件，存在循环等待链，其中每个进程都在等待链中下一个进程所持资源

- 死锁防止、死锁避免、死锁检测和恢复方法的区别
  - 死锁防止：破坏四个必要条件之一；按序分配
  - 死锁避免：
    - 银行家算法：

      缺点：一方面很难知道每个进程需要的资源最大数，另一方面系统中进程的数量是动态变化的
  - 死锁检测和解除
    - 进程-资源分配图
    - 解除：



第四章 存储管理

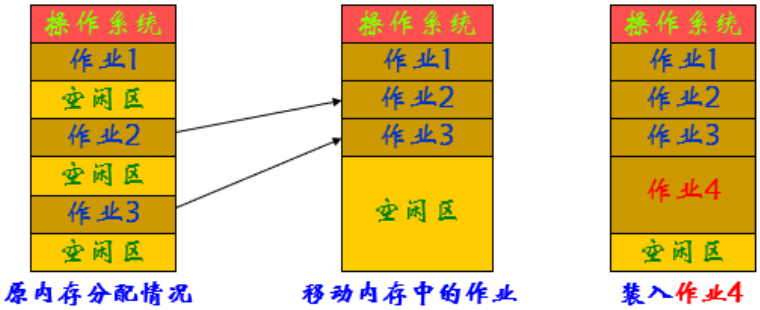
- 地址转换：逻辑地址和物理地址

- 静态重定位vs动态重定位
  - 静态重定位：装载程序实现装载代码模块的加载和地址转换，逻辑地址修改成内存物理地址
  - 动态重定位：由装载程序实现装载代码模块的加载，逻辑地址不做任何修改；内存起始地址放入重定位寄存器，程序执行过程中，每当CPU引用内存地址时，由硬件截取此逻辑地址，并在它被发送到内存之前加上重定位寄存器的值，以便实现地址转换

实存管理：

- 固定分区、可变分区
  - 固定分区：内存空间被划分成数目固定不变的分区，各分区大小不等每个分区只装入一个作业，若多个分区中都装有作业，则它们可以并发执行；缺点：大作业无法装入（覆盖）；内存空间利用率不高；分区数目确定，限制多道运行程序的道数
  - 可变分区：按照作业大小分区，划分的时间、大小、位置都是动态的；优点：克服固定分区内存资源的浪费问题，有利于多道程序设计，提高内存资源利用率
    - 可变分区匹配算法：最先适应first fit、下次适应next fit、最优best fit、最坏worst fit、快速适应quick fit（为经常使用到的长度的空闲区设立单独的空闲区链表）

- 主存不足的解决方法：移动、对换、覆盖
  - 移动：
    - 解决方案：移动技术（内存紧凑）
      - 把内存中的进程分区连接到一起，使分散的空闲区汇集成片
        - 方法一：是把所有当前占用的分区内容移动到内存的一端
        - 方法二：把占用分区内容移动到内存的一端，产生足够大小的空闲区时停止移动



  - 对换：
    - 对换技术定义
      - 如果当前一个或多个驻留进程都处于阻塞态，此时选择其中一个进程，将其暂时移出内存，腾出空间给其他进程使用，同时把磁盘中的某个进程换入内存，让其投入运行
    - 对换的作用
      - 用于分时系统中，解决内存容量不足问题，使分时用户获得快速响应时间
      - 也可用于批处理系统，平衡系统负载

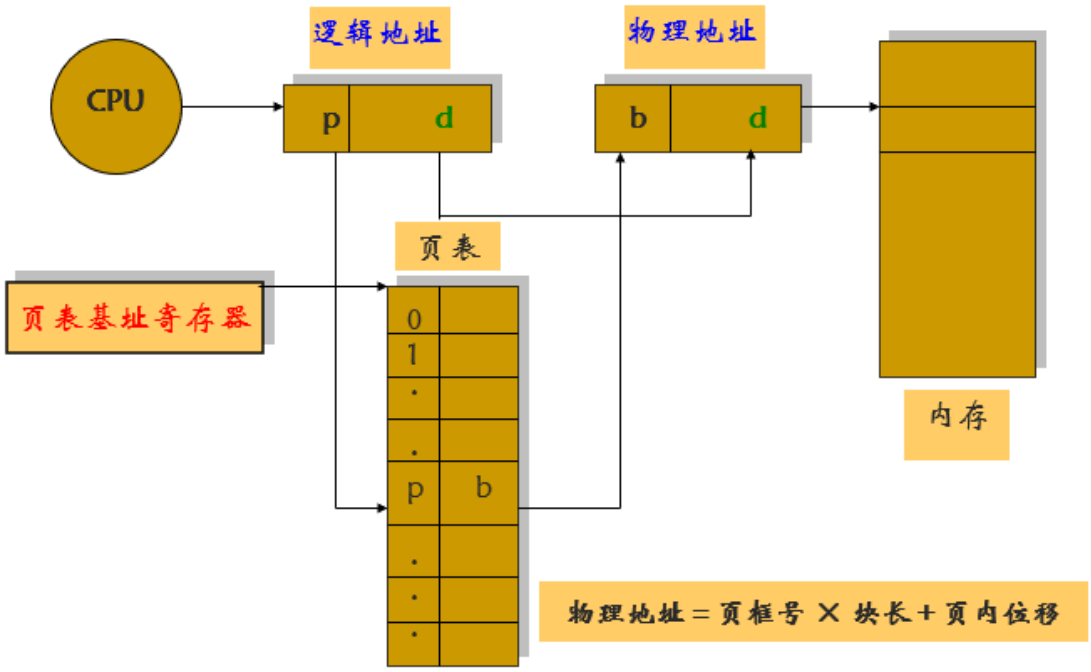
  - 覆盖：
    - 背景
      - 移动和对换技术解决因多个程序存在而导致内存区不足问题，这种内存短缺只是暂时的
      - 如果程序长度超出物理内存总和，或超出固定分区大小，则出现内存永久性短缺，大程序无法运行
    - 覆盖技术思路
      - 指程序执行过程中程序的不同模块在内存中相互替代，以达到小内存执行大程序的目的
    - 覆盖的实现技术
      - 把用户空间分成固定区和一个或多个覆盖区
        - 把控制或不可覆盖部分放在固定区
        - 其余按调用结构及先后关系分段并存放在磁盘上，运行时依次调入覆盖区

- 分页机制：动态重定位；为什么引入分页？解决内存碎片；允许程序存放到若干不相邻的空闲块中

- 基本概念
  - 页面：进程逻辑地址空间分成大小相等的区
  - 页框：内存地址空间分成大小相等的区，大小与页面相等
  - 逻辑地址形式

页号	页内位移

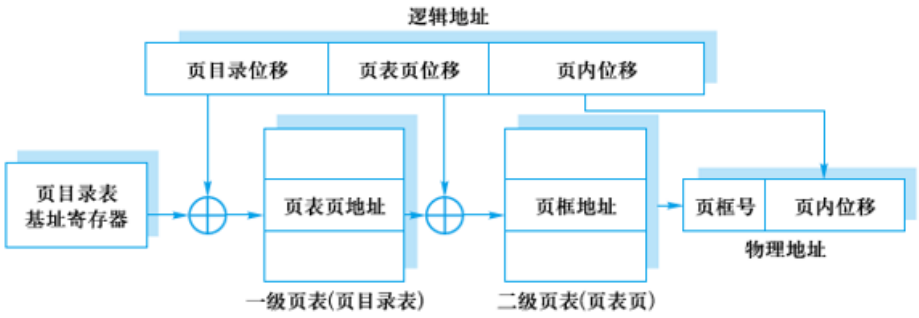
    - 页号：地址所在页面的编号
    - 页内偏移：32位系统中，常规页面尺寸为12位（4KB），页号共20位，最多可保护2^{20}个页面
  - 内存页框表
    - 该表长度取决于内存划分的物理块数，编号可与物理块号一致
    - 页框表的表项给出物理块使用情况
      - 0为空闲，1为占用



- 快表：在硬件中设置相联存储器，用来存放进程最近访问的部分页表项；包含页号及对应页框号
- 保护：

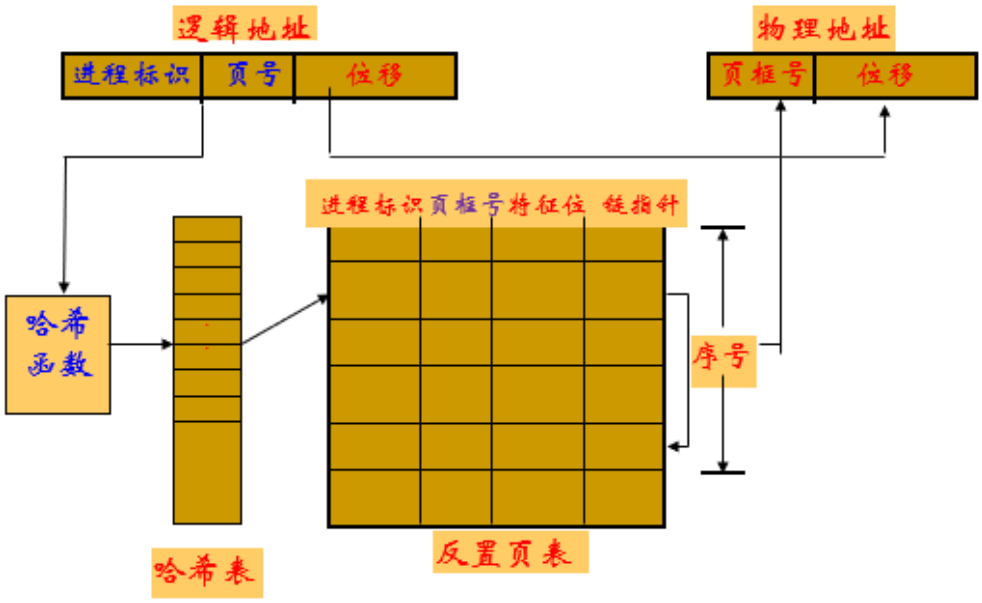
  标志位保护方法：在页表中增加标志位，指出此页的信息只读／读写／只可执行／不可访问等，进程访问此页时核对访问模式
- 多级页表：
  - 背景：解决页表保存所带来的存储开销问题
    - 32位系统中的常规分页（4 KB(2^{12})）
    - 4 GB(2^{32})的逻辑地址空间由1兆(2^{20})个页组成
    - 若每个页表项占用4个字节，则需要占用4 MB(2^{22})连续内存空间存放页表
  - 解决思路
    - 为每个进程建一张页目录表
      - 页目录表的每个表项对应一个页表页
      - 页表页的每个表项给出页面和页框的对应关系
      - 页目录表是一级页表，页表页是二级页表
    - 逻辑地址结构由三部分组成：页目录、页表页和页内位移

- 二级页表地址转换需3次访问内存
  - 访问页目录
  - 访问页表页
  - 访问指令

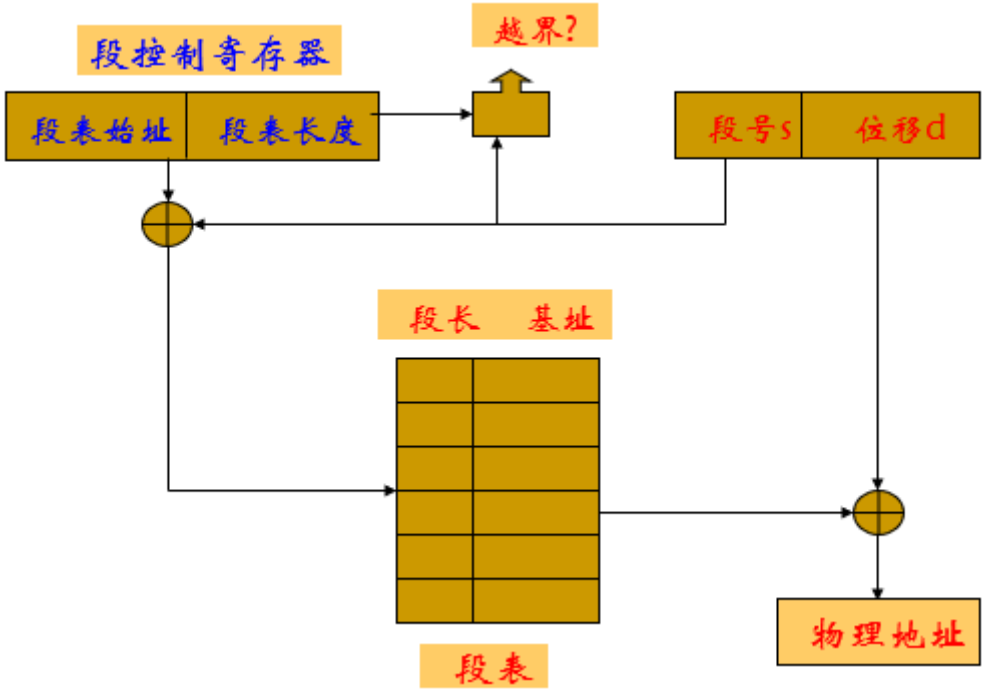


可解决分散存放页表页问题，但未解决页表页如何占用内存空间的问题

- 反置页表：为所有进程维护一张表，用于实现从逻辑地址到物理地址的转换



- 分段机制：为什么引入分段？满足用户（程序员）编程和使用要求



- 分段分页的比较：分页：提高内存空间利用率；分段：满足用户（程序员）编程和使用要求

  分段是信息的**逻辑单位**，由源程序的逻辑结构所决定，用户可见
  - **段长**可根据用户需要来规定，**段起始地址**可从任何内存地址开始
  - 分段方式中，源程序(段号，段内位移)经**链接装配后地址仍保持二维结构**

  分页是信息的**物理单位**，与源程序的逻辑结构无关，用户不可见
  - **页长**由系统确定，**页面**只能以页大小的整倍数地址开始
  - 分页方式中，源程序(页号，页内位移)经**链接装配后地址变成了一维结构**

  分页的用户进程地址空间是一维的，程序员只需给出一个记忆符即可表示一个地址。
  分段的用户进程地址空间是二维的，程序员在标识一个地址时，既要给出段名，也要给出段内地址。

  分段比分页更容易实现信息的共享和保护。
  不能被修改的代码称为纯代码或可重入代码（不属于临界资源），这样的代码是可以共享的。可修改的代码是不能共享的（比如，有一个代码段中有很多变量，各进程并发地同时访问可能造成数据不一致）

- 虚存管理：解决主存不足
  - 虚存：自动实现部分装入和部分对换功能，为用户提供一个比物理内存容量大得多的内存
  - 程序局部性原理（空间、时间）

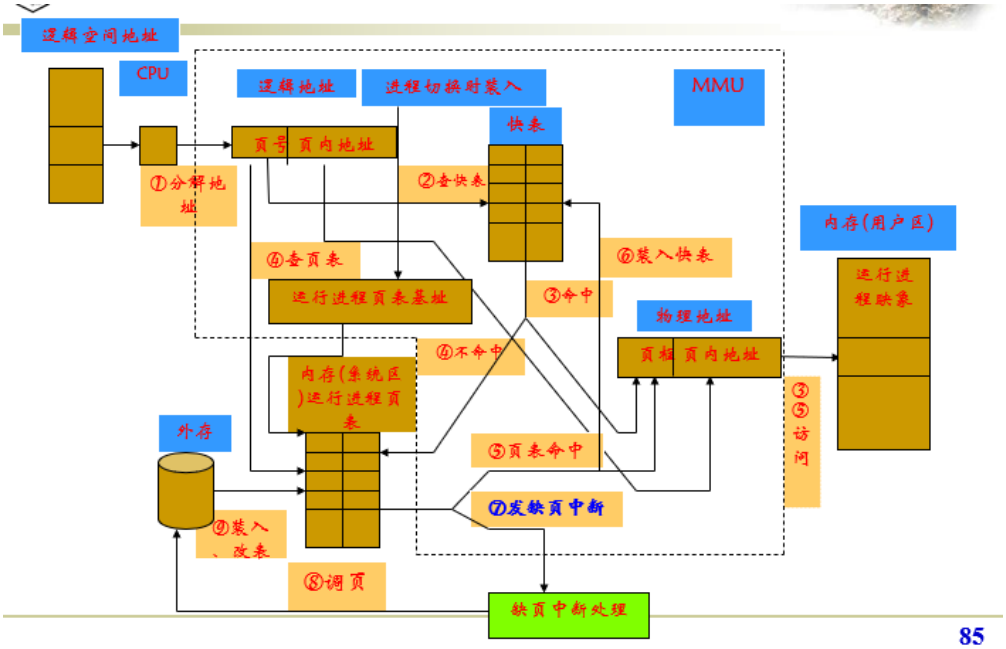
程序局部性原理

指程序在执行过程中的一个较短时间内，所执行的指令地址或操作数地址分别局限于一定的存储区域中

又可细分时间局部性和空间局部性

- 顺序局部性：程序中只有少量分支和过程调用，大都是顺序执行的指令
- 时间局部性：程序中存在迭代和循环结构，处理器最近访问的单元在不久将来再次被访问或多次循环访问
- 空间局部性：程序对一定范围的数组、或数据堆栈进行集中访问，对它们的连续引用是对位置相邻的数据项的操作

- 请求分页式
  - 分页式虚存不把作业信息(程序和数据)全部装入内存，仅装入立即使用的页面，在执行过程中访问到不在内存的页面时，产生**缺页中断**，再从磁盘动态地装入



  - 缺页中断：
    - 因素：

影响缺页中断率的因素

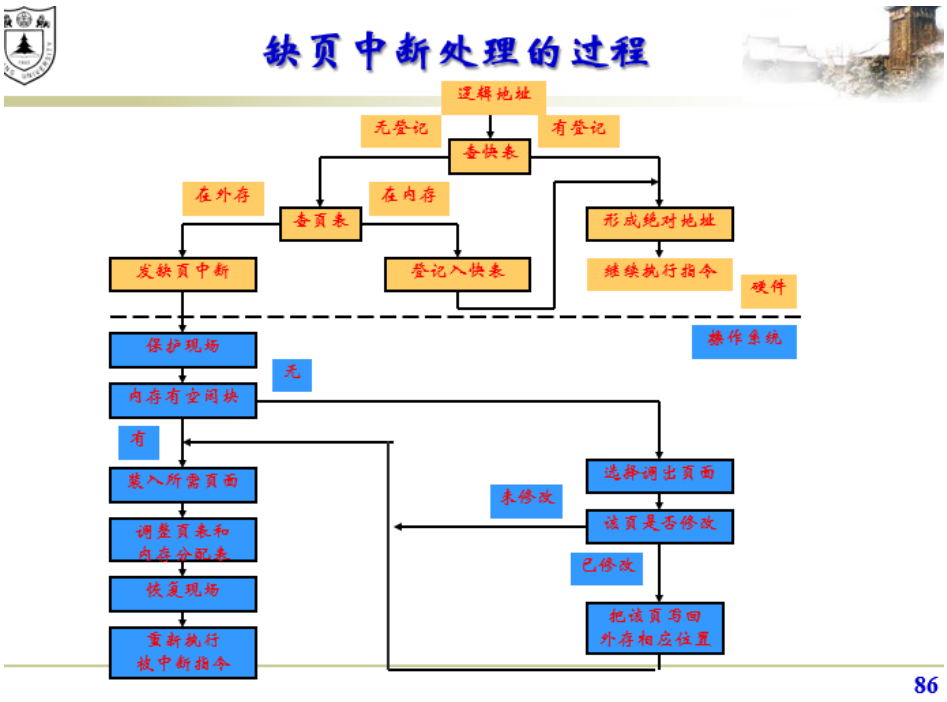
假定作业p共计n页，系统分配给它的内存块只有m块（1≤m≤n）

- 如果作业p在运行中成功的访问次数为s，不成功的访问次数为F，则总的访问次数A为
  - A＝S＋F

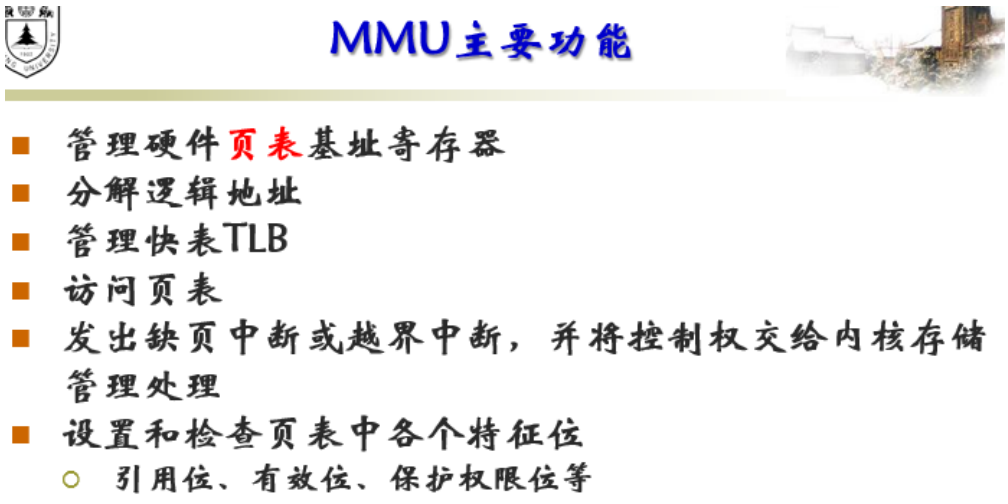
定义

- f＝F/A
- 称f为缺页中断率，影响缺页中断率f的因素有
  - 内存页框数
  - 页面大小
  - 页面替换算法
  - 程序特性

- 处理过程：



- 页面装入/消除策略：
  - 页面装入策略
    - 请页式调度：产生缺页中断时加载、系统开销大
    - 预调式调度：预先调入若干页面到内存，需要有好的预测算法支持
  - 页面清除策略
    - 请页式清除
      - 仅当页面被选中进行替换，且其内容被修改过才写回磁盘，可成批处理
    - 预清除
      - 写出的页面仍然在内存，直到页替换算法选中此页从内存中移出
      - 如果写回页面在替换前又被修改，此时将无意义
- MMU作用：内存管理单元MMU完成逻辑地址到物理地址的转换功能，它接受逻辑地址作为输入，物理地址作为输出，直接送到总线上，对内存单元进行寻址



- 全局页面替换算法：随机法, FIFO, 最佳：以后不再访问的页或距现在最长时间后再访问的页，LRU最近最少使用，Clock，二次

        - OPT：最佳页面替换算法：调入一页而必须淘汰一个旧页时，所淘汰的页应该是以后不再访问的页或距现在最长时间后再访问的页。
        - 随机页面替换算法：要淘汰的页面是由一个随机数产生程序所产生的随机数来确定
        - FIFO（FCFS）：先进先出页面替换算法：总是淘汰最先调入主存（在主存中驻留时间最长）的那一页。
        - SCR：第二次机会页面替换算法：初始引用位为 1，访问时引用位置 1，淘汰时遇 0 淘汰零，遇 1 跳过，遇全 1 全清零并淘汰第一个
        - Clock：同第二次机会页面替换算法，区别是不再是队列而是循环队列
        - 改进的 Clock 算法：淘汰 r=0,m=0 页面，若失败则淘汰 r=0,m=1 页面，扫描过程中将指针所经过的页面 r 置 0，若又失败则指针回起始位，淘汰此时 r=0,m=0 的页面即可
        - LRU：最近最少用页面替换算法：总是淘汰在最近一段时间里较久未被访问的页面
    - Belady 异常：FIFO，增加物理页框数量可能导致更多的缺页异常
    - 局部页面替换算法
  - 请求分段式 地址转换，缺页、段中断处理
  - 请求段页式

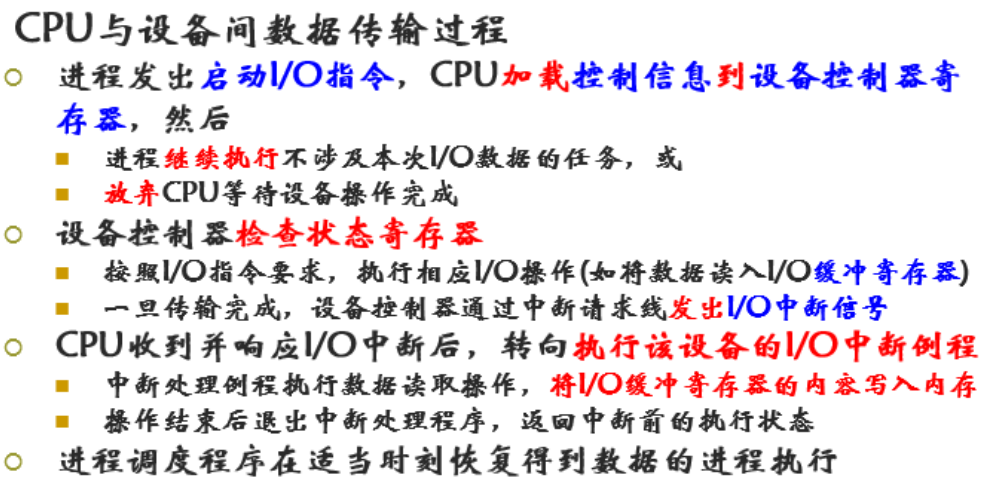
第五章 设备管理

- I/O 设备分类 （按传输单位分、按共享属性分）
  - 按I/O操作特性分：输入型、输出型、存储型
  - I/O信息交换单位分：字符设备、块设备

- I/O 控制方式：轮询、中断、DMA、通道 （互相之间的对比）
  - 轮询方式：CPU全称参与I/O操作
    - 使用**查询指令**测试设备控制器的**忙闲状态位**，决定内存和设备是否能交换数据；
    - 缺点：轮询时**终止原程序执行**，浪费CPU时间；CPU参与数据传输工作，**只能与I/O设备串行工作**

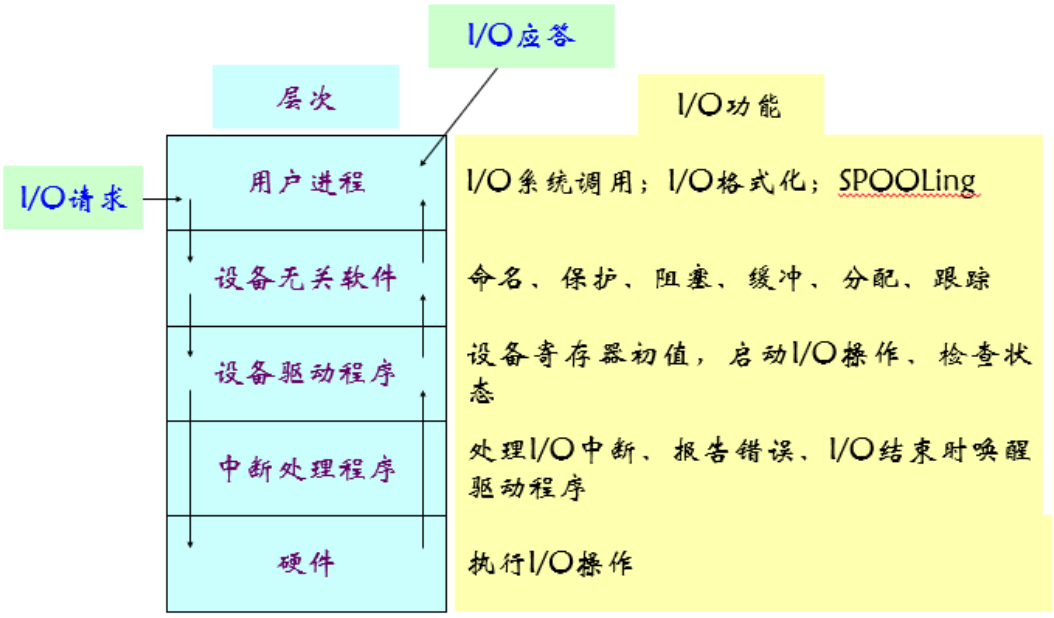
      举例：CPU从设备读入一批数据
      - CPU程序设置交换字节数和读入内存的起始地址
      - 向设备发送查询指令
      - 设备控制器把状态返回CPU
        - 如果忙或未就绪，重复测试过程，继续查询
        - 否则，开始数据传送，CPU从I/O接口读取一个字并保存到内存
      - 如果传送尚未结束，继续发查询指令，直到全部传输完成
  - 中断方式：减少CPU参与工作量
    - **中断驱动**，要求CPU与设备控制器及设备之间有**中断请求线**，控制器的状态寄存器有相应**中断允许位**
    - 缺点：**数据缓冲区大小**对中断性能的影响，当缓冲器装满后便会发生中断，导致数据传输过程中发生中断次数会较多，这样就耗用大量CPU时间；**多设备**对中断性能的影响，所有设备都通过中断处理方式实现并行工作，会使中断次数急剧增加，造成CPU**来不及响应**或丢失数据现象；响应中断请求后必须**暂停现行程序**，转入中断处理并参与数据传

输



- DMA方式：消除CPU的参与
  - I/O设备能直接与内存交换数据而**不占用CPU**；内存和设备之间有一条**数据通路，成块地传送数据，**无须 CPU 干预，实际数据传输操作由DMA直接完成
  - 优点：数据传输过程不需要占用 CPU 时间，CPU与 I/O 设备之间可以**并行**工作
  - 缺点：**每次I/O指令只能读写一个数据块**，若希望一次能够读写多个离散数据块并传送到不同的内存区域，则需要**由CPU发出多条启动I/O指令及进行多次 I/O中断处理**才能完成；若出现DMA与CPU同时经总线访问内存，CPU把总线占有权让给DMA(称"**周期窃用**")，降低CPU处理效率
- 通道方式：消除CPU的参与
  - 让**多种异构外围设备以标准的接口**连接到系统中，由**通道来管理和控制I/O操作**，实现设备和CPU、通道之间、设备之间并行操作；CPU向通道发出指令，之后并行工作，直到I/O操作完成，通道发出操作结束中断
  - 优点：再次减少CPU对I/O操作的干预，CPU和I/O设备并行工作

- I/O 软件的四个层次
  - I/O软件设计目标：
    - 编程时的设备无关性
    - 出错处理：尽可能在靠近硬件的地方处理，避免高层软件感知
    - 异步或同步传输：异步（**中断驱动**）传输：CPU在启动I/O操作后既可继续执行其他工作，直至中断到达；同步（**阻塞**）传输：启动I/O操作的进程挂起等待，直至数据传输完成
    - 缓冲技术：建立数据缓冲区，让数据**到达率与离去率**相匹配，提高系统**吞吐率**
  - 四个层次：
    - **I/O中断处理程序**：底层，进程请求I/O操作时被挂起，直到数据传输结束并产生I/O中断请求时，操作系统接管CPU后执行中断处理程序
    - **I/O设备驱动程序**：**从独立于设备的软件中接收并执行I/O请求**；包括**与设备相关的代码**，把用户提交的**逻辑I/O请求**转化为**物理I/O操作**的启动和执行
    - **独立于设备的操作系统I/O软件**：执行适用于所有设备的常用I/O功能，**并向用户层软件提供一致性接口**（设备命名与保护，提供与设备无关的块尺寸，缓冲技术，设备分配和状态跟踪，错误处理和报告）
    - **用户空间的I/O软件**：库函数实现的I/O系统调用；SPOOLing



- 缓冲（作用）
  - 目的：
    - 改善**CPU与I/O速度不匹配**
    - 协调**逻辑记录大小与物理记录大小不一致**
    - **减少I/O操作对CPU中断次数**
    - **放宽CPU中断响应时间**的要求
  - 单缓冲：

■ 每当应用进程发出I/O请求时，操作系统在内存的系统区中开设一个缓冲区

■ 块设备读取的单缓冲机制

- 先从磁盘把一块数据读至缓冲区，假设花费时间T
- 接着，把缓冲区数据送到用户区，设所消耗的时间为M
  - 此时缓冲区已空，可预读紧接着的下一块
- 然后，应用进程对这批数据进行计算，共耗时C
- 读入缓冲和计算可以并行

■ 性能分析

- 不采用缓冲技术，数据直接从磁盘传送到用户区，每批数据处理时间约为T+C
- 而采用单缓冲，每批数据处理时间为max(C, T)+M
  - 通常M远远小于C或T，故速度会快很多

  - 双缓冲

- 设计目标
  - 加快I/O操作执行速度，实现I/O并行工作和提高设备利用率
- 工作原理
  - 输入数据时，首先填满缓冲区1
    - 操作系统可从缓冲区1把数据送到用户进程区，用户进程便可对数据进行加工计算
    - 与此同时，输入设备填充缓冲区2
  - 当缓冲区1空出后，输入设备再次向缓冲区1输入
    - 操作系统又可把缓冲区2的数据传送到用户进程区，用户进程开始加工缓冲区2的数据
  - 两个缓冲区交替使用，使CPU和设备、设备和设备间的并行性进一步提高
    - 当两个缓冲区都为空且进程还要提取数据时，它才被迫等待
- 单块数据传输和处理时间分析
- 如果C<T，由于M远小于T，在将磁盘上的一块数据传送到缓冲区期间，计算机已完成将另一个缓冲区中的数据传送到用户区并对这块数据进行计算的工作
  - 单块数据的传输和处理时间为T（即max(C, T)）
  - 此时可保证块设备连续工作
- 如果C>T，当上一块数据计算完毕后，需把一个缓冲区中的数据传送到用户区，花费时间为M，再对这块数据进行计算，花费时间为C
  - 单块数据的传输和处理时间为C+M、即max(C,T)+M，此时进程不必要等待I/O

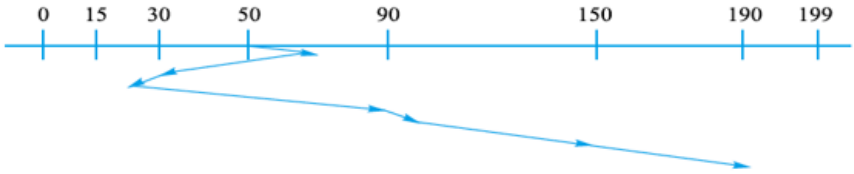
- 多缓冲
  - 双缓冲能提高设备并行工作程度，但设备和进程速度不匹配的情况仍不十分理想
    - 存在一种设备等待的可能
  - 操作系统从内存区域中分配一组缓冲区组成循环缓冲
    - 每个缓冲区的大小等于物理记录的大小
    - 多缓冲的缓冲区是系统的公共资源，可供各个进程共享，并由系统统一分配和管理
  - 缓冲区用途分类
    - 输入缓冲区、处理缓冲区、输出缓冲区
  - 缓冲区工作方式
    - 字符型设备：先进先出方式
    - 磁盘类直接存储型设备
      - 按进程要求随机访问，同一数据可能被多次访问，如何优化？

- 驱动调度技术：I/O等待时间=查道时间+旋转延迟时间
  - 旋转延迟时间优化方法
    - 优化请求排序：按照数据的分布对输入/输出请求进行排序，提高处理的效率
    - 优化空间分布：按照数据处理的规律，合理安排其磁盘上的分布
  - 移臂调度策略

- 先来先服务FCFS：性能差
- 最短查找时间优先：存在饥饿现象

最短查找时间优先（150，30，190，20，100，55，90）

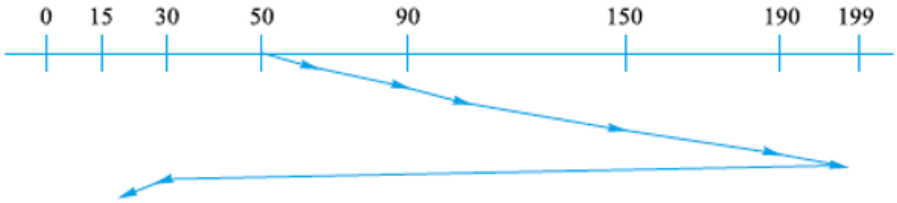
- 考虑I/O请求之间的区别，先执行**查找时间最短**的请求
- 与FIFO算法相比有较好寻道性能
- **存在“饥饿”现象**
  - 随着靠近当前磁头位置读写请求的不断到来，到来时间早但距离当前磁头位置较远的I/O请求服务**被无限期地推迟**
- 移动臂移动柱面总数
  - (55-50)+(55-30)+(30-20)+(90-20)+(100-90)+(150-100)+(190-150)=210



- 扫描SCAN：偏爱最接近里面或靠近外面的请求

扫描算法（150，30，190，20，100，55，90）

- 磁盘臂每次**沿一个方向移动**，扫过所有柱面，**遇到最近的I/O请求**便进行处理
- 直到最后一个柱面后，再向**相反方向移回**
- 扫描算法偏爱最**接近里面**或**靠近外面**的请求
- 移动臂移动柱面总数
  - (55-50)+(90-55)+(100-90)+(150-100)+(190-150)+(199-190)+(199-30)+(30-20)=328



- 电梯调度LOOK：扫描算法的改进

  - **电梯调度算法（elevator algorithm），又称LOCK算法**
    - **是扫描算法的一种改进**
      - 扫描算法**每次都移动到柱面尽头**，无论前端是否有访问请求
    - 无访问请求时，移动臂停止不动，有访问请求时，移动臂按电梯规律移动
      - 每次总是**选择沿移动臂的移动方向最近的那个柱面**
        - 若同一柱面上有多个请求，还需进行旋转优化
      - 如果当前移动方向上没有但相反方向有访问请求时，**就改变移动臂的移动方向**，然后，处理所遇到的最近的I/O请求
      - 每当要求访问磁盘时，操作系统**查看磁盘机是否空闲**
        - 如果空闲就立即移臂，将当前移动方向和本次停留位置都记录下来
        - 如果非空，就让请求者等待并把记录的访问位置按照既定的调度算法对全体等待者进行寻查定序优化

- 循环扫描算法

- 提高磁盘I/O速度的方法

- 基本思路
  - 设置磁盘数据缓冲区高速缓存，缩短等待磁盘I/O的时间
- 实现机制
  - 提前读
    - 对于采用顺序方式所访问的文件数据，提前把下一磁盘块的数据也读入磁盘缓冲区
  - 延迟写
    - 考虑到此缓冲区中的数据不久之后会再次被进程访问，将修改后的内容挂在空闲缓冲区队列末尾
    - 直至移动到空闲缓冲区队列之首，当再有进程申请缓冲区且分配到此缓冲区时，才把其中的数据写到磁盘上
  - 虚拟盘
    - 用内存空间去仿真磁盘，又叫RAM盘
    - 虚拟盘vs.数据缓冲区高速缓存：前者的内容完全由用户控制，而后者的内容由操作系统控制

- 设备分配方式
  - 独占方式：作业执行依赖独占设备（如卡片机）
    - 静态分配，作业执行前将所要使用的设备全部分配给它
  - 共享方式：执行过程中临时需要共享设备（如打印机）
    - 动态分配，作业执行过程中动态分配设备资源
  - 虚拟方式：如磁盘
    - 不需要显式分配，设备管理的主要工作是驱动调度和实施驱动

- 虚拟设备： Spooling技术原理，作用
  - 静态分配方式不利于提高系统效率
    - 设备在作业执行期间，部分时间在工作，其余时间空闲
    - 设备被分配出去以后，新的设备申请会被拒绝接受
  - 解决思路
    - 让处理器执行计算的同时，进行联机外部设备操作
      - 预输入：操作系统将一批作业从输入设备上预先输入至磁盘的输入缓冲区中暂存
        - 作业调度程序调度作业执行，作业使用数据时不必再启动输入设备，只要从磁盘的输入缓冲区中读入
        - 作业执行过程中不必直接启动输出设备，只要将作业的输出数据暂时保存到磁盘的输出缓冲区
      - 缓输出：作业执行完毕后，由操作系统组织信息成批输出
  - 特点
    - 作业执行时需要I/O数据时不再和低速设备联系，而是与高速磁盘交互，提高设备利用率，缩短作业运行时间

第六章 文件管理

- 文件，目录文件（功能，内容）
  - 概念：文件是由文件名所标识的一组信息的集合；由信息按一定结构方式组成，可持久性保存的抽象机制
  - 文件系统功能：
    - 按名存取，从逻辑文件到物理文件的转换
    - 文件目录的建立和维护
    - 文件的查找和定位
    - 文件存储空间的分配管理
    - 提供文件的存取方法和文件存储结构
    - 实现文件共享、保护和保密
    - 提供一组易用的文件操作和命令
    - 提供与设备管理交互的统一接口
  - 内容：由 FCB 和文件体两部分组成
  - 文件控制块FCB：是OS为每个文件建立的唯一数据结构，包含全部文件属性；基于FCB可以实现文件的按名存取
  - 文件目录，目录文件：
    - 文件目录：FCB的有序集合，包含许多目录项（有两种，分别描述子目录和文件）
    - 目录文件：全部由目录项构成的文件；目录文件永远不会空，至少包含两个目录项：当前目录项“.”和父目录项“..”；对于根目录，“.”和“..”都指向同一个 inode

- 文件逻辑结构：流式文件/记录式文件 （单位、结构）
  - 流式文件：是一种无结构的文件，由一系列二进制流或者字符流组成；每个字节都有一个索引
  - 记录式文件：是一种有结构的文件，包含若干逻辑记录；记录在文件中**按出现次序编号**

- 文件物理结构：顺序文件、连接文件、直接文件、索引文件
  - 顺序文件：将文件中**逻辑上连续的信息**存放到存储介质的**相邻物理块**上形成顺序结构，形成顺序文件；逻辑记录顺序和物理记录顺序完全一致；FCB中保存第一个物理块地址和文件信息块的总块数
  - 连接文件：使用**连接字（指针）**表示文件中各记录间的关系；FCB给出第一块文件信息的物理地址由，每块的连接字指出文件的下一个物理块位置；非连续存储
  - 索引文件：为每个文件建立一个**索引表**，利用索引表来搜索记录；索引表**可以**存放在FCB中或包含索引表的地址；非连续存储
  - 直接文件：利用**哈希**法将记录的关键字与其地址之间建立某种对应关系，建立hash表，实现快速存取；适用于**不能采用顺序组织方法、次序较乱、又需在极短时间内进行存取**的场合；溢出处理：顺序探查法，两次散列法等

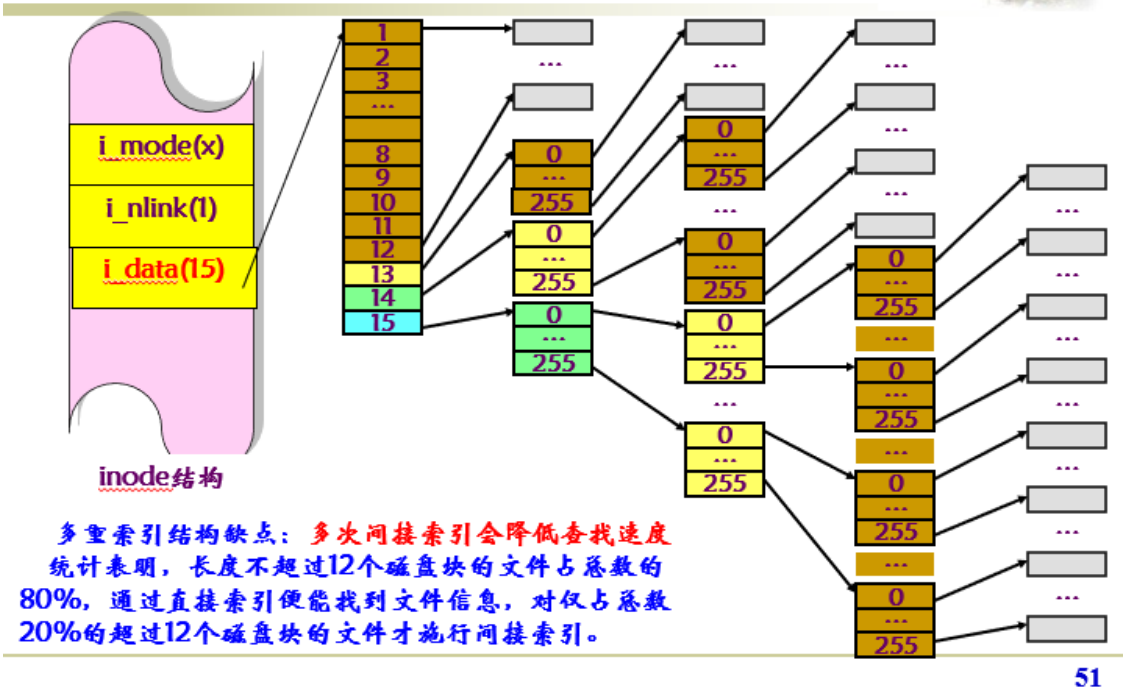
- 多重索引结构

Linux的多重索引

- 每个文件的inode中规定15个索引项，每项占用4B，登记一个存放文件信息的物理块号
- 系统仅提供流式文件而无记录概念，因此登记项中没有记录键与之对应
  - 前面12项存放文件信息的磁盘块号，称为直接索引
  - 如果文件大于12块，利用第13个索引项指向一个物理块
    - 此块中最多可放256个存放文件信息的磁盘块的块号，叫做一次间接索引
  - 大型文件还可以利用第14、第15索引项作二次和三次间接索引
    - 它们最多可放256^2个、256^3个存放文件信息的磁盘块的块号
  - 在ext2中，每个物理块存放1024B，每个文件的最大长度为$12K+256K+256^2K+256^3K$字节

50

Linux的多重索引结构

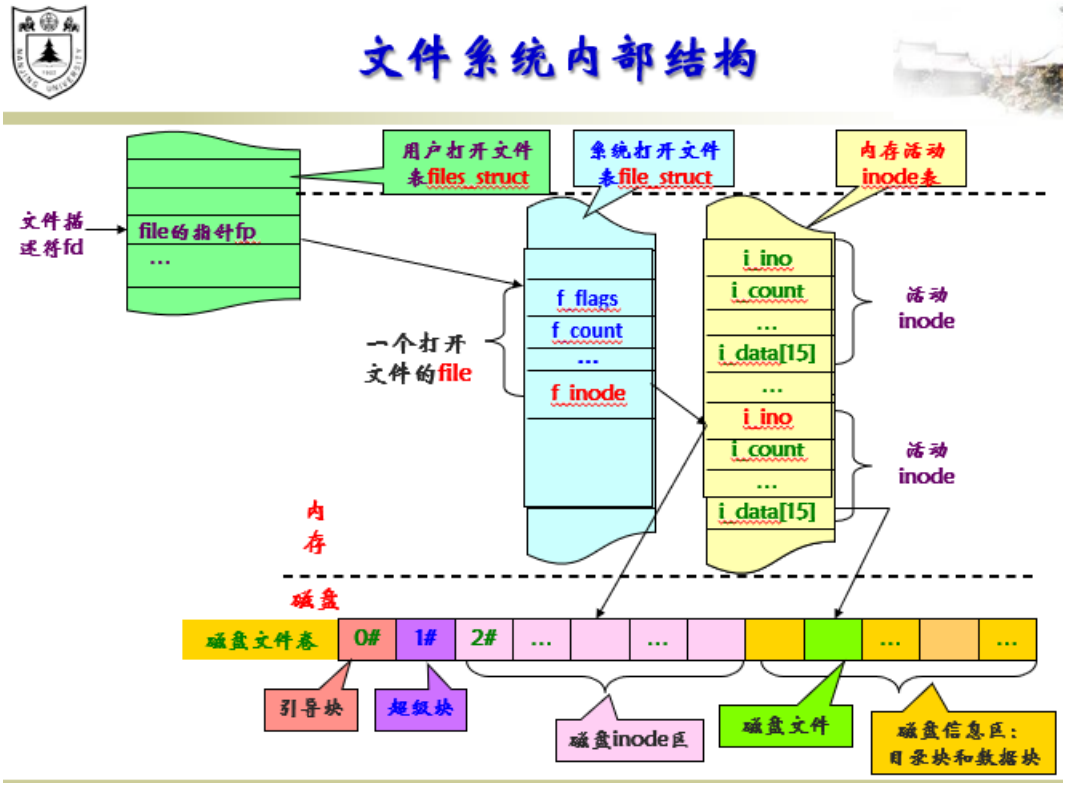


多重索引结构缺点：多次间接索引会降低查找速度
统计表明，长度不超过12个磁盘块的文件占总数的80%，通过直接索引便能找到文件信息，对仅占总数20%的超过12个磁盘块的文件才施行间接索引。

51

- 文件系统内部结构（磁盘结构、内存结构）
  - 磁盘结构：磁盘按扇区编号，扇区序列分成3部分
    - **超级块**：1#号块 ，存放文件系统结构和管理信息；**既有盘位示图的功能，又记录整个文件卷的控制数据**
    - **索引节点区**：2#～k+1#块，存放inode表；分为磁盘 inode表和内存活动inode表
    - **数据区**：k+2#～n#块，存放文件内容
  - 用户打开文件表，系统打开文件表

- 文件系统内部结构：用户打开文件表 file_struct 中 file 指针 fp->系统打开文件表 file_struct（f_flags/f_count/.../f_inode）中某个表项的 f_inode->内存活动 inode 表的一个活动 inode（这个 inode 和磁盘索引节点区的一个 inode 一一对应），活动inode 中的 i_data 指向磁盘数据块



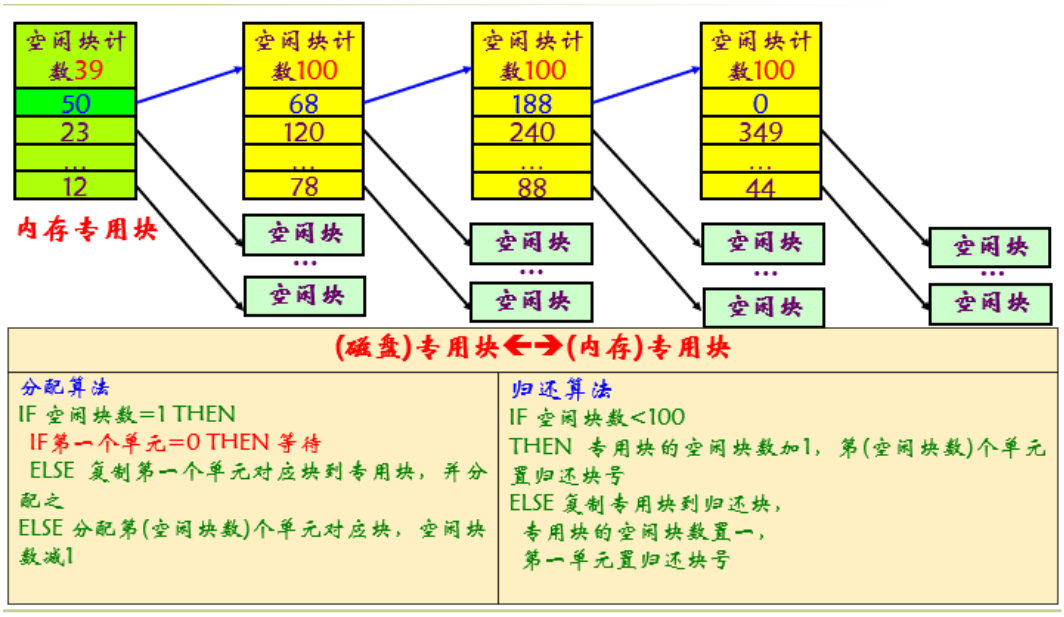
- 文件共享（静态、动态），父子进程共享，独立进程共享
  - 操作系统允许一个文件同时属于多个目录，但**实际上文件仅有一处物理存储**
  - 静态共享：
    - **无论进程是否运行，其文件的链接关系都存在**
    - 通过**文件所对应的inode**节点来实现链接，并且只允许链接到文件而非目录
    - link（oldnamep, newnamep）
  - 动态共享：
    - 指系统中不同的用户进程或同一用户的不同进程并发访问同一文件，只有当用户进程存在时才可能出现，一旦用户的进程消亡，其共享关系也就自动消失
    - 指向**活动inode表**
    - 动态共享的读写指针问题：
      - 同一用户**父、子进程**协同完成任务，使用同一读/写位移，同步地对文件进行操作；该位移指针放在相应文件的活动索引节点中，当用系统调用fork建立子进程时，父进程的pcb结构被复制到子进程的pcb结构中，使两个进程的打开文件表指向同一活动inode，达到共享同一位移指针的目的
      - 独立进程共享：位移指针不应放在相应文件的活动inode中，而需独立设置

- 文件的创建、打开、读写过程
  - 创建（create）：

        - 系统调用格式：fd = create (filenamep, mode);char *filenamep：创建的文件路径名的字符串指针；int mode：文件所具有的存取权限，文件成功创建之后，存取权限被记录在相应inode的i_mode中；int fd：创建成功后系统所返回的文件描述符，即用户打开文件表中相应文件表项的序号
        - 执行过程：
            - 为新文件分配inode和活动inode，并把inode号与文件分量名组成新目录项，记到目录中
            - 在新文件所对应的活动inode中置初值，如，置存取权限i_mode，连接计数i_nlink等
            - 分配用户打开文件表项和系统打开文件表项，置表项初值，读写位移f_offset清“0”；把各表项及文件对应的活动inode用指针连接起来，把文件描述字返回给调用者
    - 删除（unlink）：
        - 删除系统调用形式为：unlink (filenamep)，i_nlink--，如果删除前 i_nlink==1，还要把文件占用的存储空间释放
        - 在执行删除时，必须要求用户对该文件具有“写”操作权
    - 打开（open）：
        - fd = open (filenamep, mode);
        - 检索目录，把它的外存inode复制到活动inode表；若已有其他用户打开同一文件，则不执行复制inode的工作，仅把活动 inode的计数器i_count加 1
        - 根据参数mode核对权限，若非法，则这次打开失败
        - 为文件分配用户打开文件表项和系统打开文件表项，并为表项设置初值；通过指针建立这些表项与活动inode间的联系；把文件描述字返回给调用者
    - 关闭（close）：
        - close (fd);
        - 根据fd找到用户打开文件表项，再找到系统打开文件表项，释放用户打开文件表项
        - 把对应系统打开文件表项中的f_count减“1”；如果非“0”，说明还有进程共享这一表项，不用释放直接返回，否则释放表项
        - 把活动索引节点中的i_count减“1”；若不为“0”，表明还有用户进程正在使用该文件，不用释放而直接返回，否则在把该活动inode中的内容复制回文件卷上的相应inode中后，释放该活动inode
    - 读（read）
        - 系统调用形式：nr = read (fd, buf, count);buf：读出信息所应送入的用户数据区首地址，count：要求传送的字节数，nr：此系统调用执行后所返回的实际读入字节数
        - 系统根据f_flag中的信息，检查读操作合法性
        - 按活动inode中i_data[]指出的文件物理块存放地址，从文件当前位移量f_offset处开始，读出所要求的字节数到块设备缓冲区中
        - 送到buf指向的用户数据区中
    - 写（write）
        - nw = write (fd, buf, count);buf是信息传送的源地址，即把buf所指向的用户数据区中的信息写入文件中

- 文件的目录检索过程（P304,305）

- 磁盘空间管理
    - 位示图
    - 成组空闲链表法



- 虚拟文件系统（作用，基本原理）
  - 作用：为了同时支持多种文件系统，把多种具体文件系统纳入统一框架
  - 原理：提供一个通用文件系统模型，概括具体文件系统常用功能和行为，为应用程序提付标准接口

- 主存映射文件
  - 结合虚存管理和文件管理技术，解决文件读写操作效率低下问题
  - 把进程需要访问的文件映射到其虚地址空间中，以便可通过读写相应虚地址进行文件访问；内存映射文件按照文件名来访问
  - mmap()：映射文件
  - unmmap()：移去映射文件