**3.2．地址空间（储存器的抽象）**

保证多个应用程序同时处于内存中并且互不影响

**问题**

（1）用户程序可以寻址内存的每个字节，很容易破坏操作系统

（2）一个CPU想要运行多个程序是很困难的

概念：是一个进程可用于寻址内存的一套集合

保证多个应用程序同时处于内存并且不互相影响

待解决的问题：保护和重定位

基址寄存器和界限寄存器

动态重定位：把每个进程的地址空间映射到物理内存的不同部分；

方法：给每个CPU配置两个特殊的硬件寄存器，通常叫基址寄存器和界限寄存器

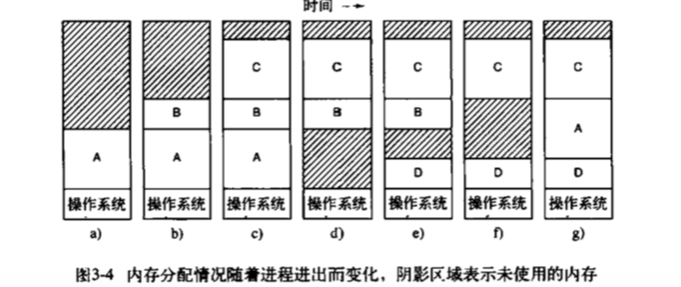
所有进程一直保存在内存想要巨大的内存引发内存超载的问题

解决方法：

（1）**交换技术**：进程完整调入内存，使该进程运行一段时间，然后把其存回磁盘。空闲进程主要存储在磁盘上，因此当其不允许时不会占用内存

（2）**虚拟内存：**是程序在只有一部分调入内存的情况下运行

（\*\*） 交换技术



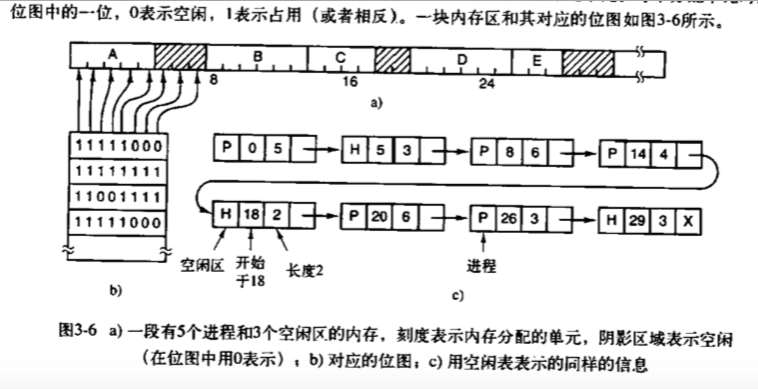
交换在内存中产生了多个空闲区（hole， 也称空洞）

**内存紧缩：**通过把所有进程尽可能的向下移动，有可能将这些小的空闲区合并成一大块

进程的数据段增长：（1）寻找空闲区 （2）空闲区不足以支持进程运行的内存空间时，将进程挂起或者退出该进程

**空闲内存管理：**

（1）使用位图的存储管理



问题与缺陷：在决定把一个占K个分配单元的进程掉入内存时，存储管理器必须搜索位图，在位图找出有K个连续0的串，查找位图连续串为0时耗时的操作（因为图中该串可能跨越字的边界）

（2）链表的存储管理

：维护一个记录以分配内存段和空闲段hole的链表。其中链表中的节点包涵一个进程或者两个进程之间的空闲区

链表结点

{

空闲区（H）标志位，

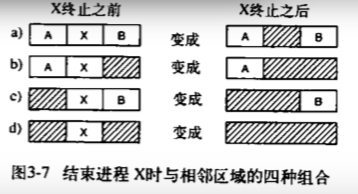
进程（P）标志位，

起始地址，

长度，

指向下一结点的指针

}；



因为进程表中表示终止进程的结点中通常含有指向对应于其段链表结点的指针，因此段链表是使用双链表；

**首次适配算法**：储存管理器沿着链表搜索，直到找到一块足够大的hole，除非空闲区和要适配的空间一样大，否则将空闲区分为两部分：一部分供进程使用，一部分形成空闲区。

**最佳适配算法**：搜索整个链表（开始到结束），找出能够容纳进程的最小空闲区。最佳适配算法试图找出最接近实际需要的空闲区，以最好的地区请求和可用空闲区，而不是拆分一个以后可能会用到的大的空闲区

**快速适配算法**：为常用大小的空闲区维护一个单独的链表，寻找一个指定大小的空闲区是十分快速的

缺陷：和所有的将大小排序的方案一样，在一个进程终止或者被唤出时，寻找它的相临快，查看是否可以合并的过程时非常费时的。如果不合并，内存将会裂出大量进程无法使用的内存块

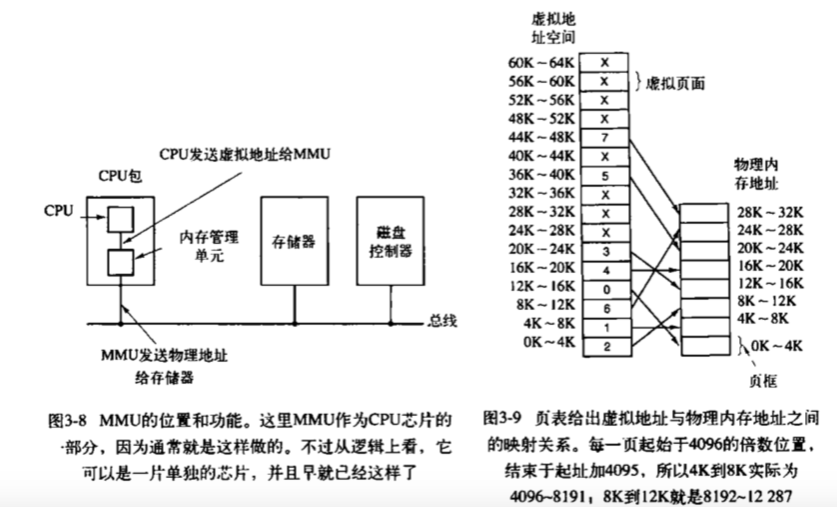
**3-3 虚拟内存（重点）**

：解决内存无法容纳足够大的运行程序（SATA磁盘的峰值传输率最高达到100MB/S）

早期解决方案：覆盖：将程序分割成许多片段

虚拟内存基本思想：每个进程都拥有自己的地址空间，这个空间分割多个快，每一个块称为**页面**。这些页面被映射到物理内存中。由硬件立刻执行必要的映射。

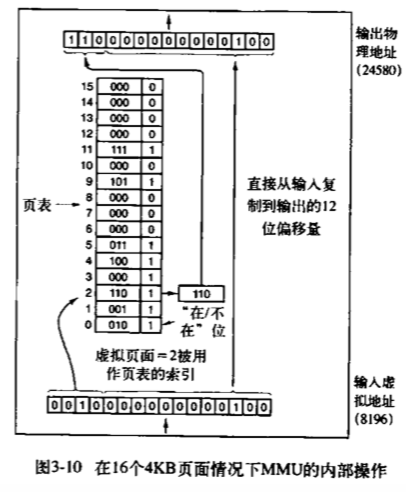
**分页技术**



MOV REG ， 32780

**缺页中断**：虚拟页表对应的物理地址为空，此时CPU陷入操作系统中，引发的中断

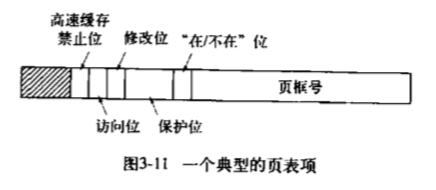
操作系统找到一个很少使用的页框且把它的内容写在磁盘，随后把需要访问的页面读到刚才回收的页框中，改变映射关系，然后重新引起陷阱的指令；



**页表：**

作为一种简单的实现，虚拟地址到物理地址的映射可以概括如下：

虚拟地址被分成虚拟页号（高位部分4位）和偏移量（低12位）

页表项结构：

虚拟内存本质上是用来创造一个新的抽象概念----地址空间，这个概念是对物理内存的抽象，类似于进程是对物理机器的抽象。虚拟内存的实现，是将虚拟地址空间分解成页，并将每一页映射到物理地址聂村的某个页框或者（暂时）解除映射。

**因此本章内容是关于操作系统的创建的抽象，以及管理这个抽象**

**3.3.3加速内存过程**

1）虚拟地址映射到物理地址必须非常快

2）如果虚拟地址空间很大，页表页会很大

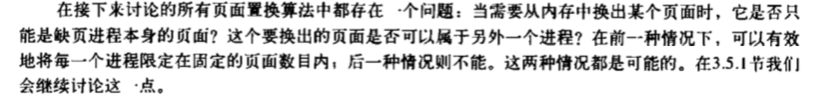
当启动一个进程是，操作系统保存在内存中的进程页表的副本载入到寄存器，进程运行过程中不需要位页表而访问内存。

优点：简单并且映射过程中不需要访问内存。

缺点：页表很大时代价高

（1）转换检测缓冲区（TLB）

：将虚拟地址直接映射到物理地址，不是再访问页表；通常存在于MMU，包涵少量的表项

页面置换算法：

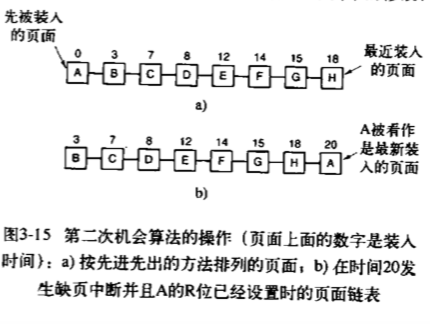
最优页面置换算法：是缺页中断发生的时间越短越好（无法实现，因为操作系统无法知晓下一个被访问的页面）

最近未被使用页面置换算法

大部分具有虚拟系统的计算机中，系统位每一个页面设置了两个状态位，当R（被访问），M（被写入）

先进先出页面置换算法（FIFO）

二次机会页面置换算法

FIFO + 检查状态位（R为0则被置换出去，R为1则清0放入链表尾部）

LRU（Least Recently Used）

LRU算法的设计原则是：**如果一个数据在最近一段时间没有被访问到，那么在将来它被访问的可能性也很小**。也就是说，当限定的空间已存满数据时，应当把最久没有被访问到的数据淘汰。

1.用一个数组来存储数据，给每一个数据项标记一个访问时间戳，每次插入新数据项的时候，先把数组中存在的数据项的时间戳自增，并将新数据项的时间戳置为0并插入到数组中。每次访问数组中的数据项的时候，将被访问的数据项的时间戳置为0。当数组空间已满时，将时间戳最大的数据项淘汰。

2.利用一个链表来实现，每次新插入数据的时候将新数据插到链表的头部；每次缓存命中（即数据被访问），则将数据移到链表头部；那么当链表满的时候，就将链表尾部的数据丢弃。

3.利用链表和hashmap。当需要插入新的数据项的时候，如果新数据项在链表中存在（一般称为命中），则把该节点移到链表头部，如果不存在，则新建一个节点，放到链表头部，若缓存满了，则把链表最后一个节点删除即可。在访问数据的时候，如果数据项在链表中存在，则把该节点移到链表头部，否则返回-1。这样一来在链表尾部的节点就是最近最久未访问的数据项。

使用LinkedHashMap实现

     LinkedHashMap底层就是用的HashMap加双链表实现的，而且本身已经实现了按照访问顺序的存储。此外，LinkedHashMap中本身就实现了一个方法removeEldestEntry用于判断是否需要移除最不常读取的数，方法默认是直接返回false，不会移除元素，所以需要重写该方法。即当缓存满后就移除最不常用的数。

工作集页面置换算法

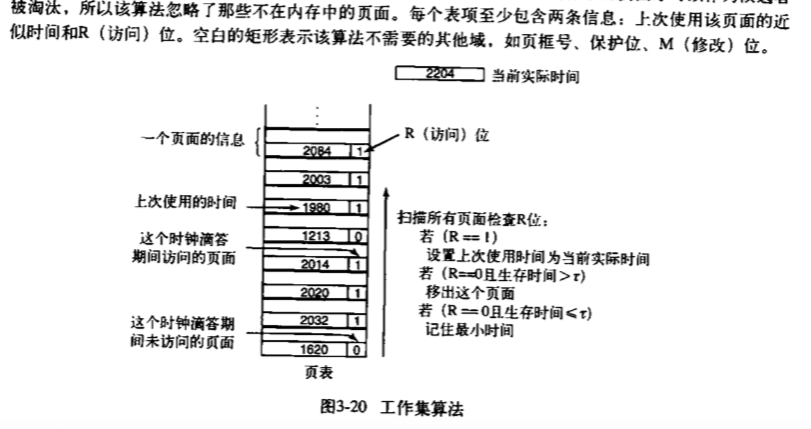
申请调页：页面是需要时被调入，而不是预先装入的

工作集：一个进程正在使用的页面的集合。由于内存较小不能一次性将进程工作集。（过去十秒中的内存访问所用到的页面的集合）

工作集模型：分页系统都会设法跟踪进程的工作集，以确保让进程运行之前它的工作集就已经在内存中，目的是在于大大的减少缺页中断率

预先调页：在进程运行前预先装入器工作集页面

目的：当发生缺页中断时，淘汰一个不在工作集的页面，选定一个K值，每次访问内存之后，最近K次内存访问所使用过的集合是唯一确定的。

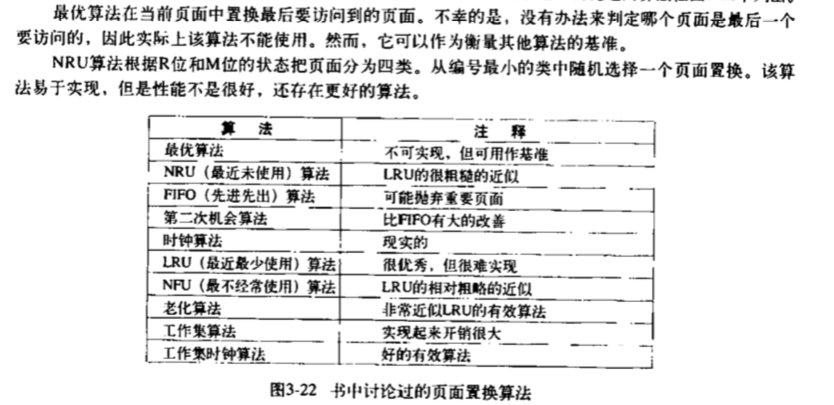
设计思想：找出一个不在工作集的页面并且淘汰它

工作集时钟页面置换算法（时钟算法 + 工作集）

工作集页面置换算法需要扫描整个页表以此比较费时

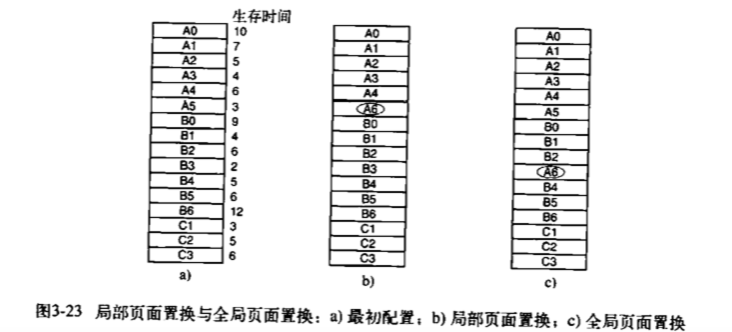
数据结构 ： 一个以页框位元素的循环表

**页面置换算法的总结**



**3.5分页系统中的设计问题**

局部页面置换和全局页面置换



局部算法：可以有效地位每个进程分配固定的内存片段

全局算法：在可运行进程之间动态的分配页框，因此分配给各个进程的页框数时随时间变化的

**页面抖动（颠簸）**

在页面置换过程中的一种最糟糕的情形是，刚刚换出的页面马上又要换入主存，刚刚换入的页面马上就要换出主存，这种频繁的页面调度行为称为抖动，或颠簸。如果一个进程在换页上用的时间多于执行时间，那么这个进程就在颠簸。  
  
频繁的发生缺页中断（抖动），其主要原因是某个进程频繁访问的页面数目高于可用的物理页帧数目。虚拟内存技术可以在内存中保留更多的进程以提髙系统效率。在稳定状态，几乎主存的所有空间都被进程块占据，处理机和操作系统可以直接访问到尽可能多的进程。但如果管理不当，处理机的大部分时间都将用于交换块，即请求调入页面的操作，而不是执行进程的指令，这就会大大降低系统效率。

**工作集（驻留集）**

工作集（或驻留集）是指在某段时间间隔内，进程要访问的页面集合。经常被使用的页面需要在工作集中，而长期不被使用的页面要从工作集中被丢弃。为了防止系统出现抖动现象，需要选择合适的工作集大小。  
  
工作集模型的原理是：让操作系统跟踪每个进程的工作集，并为进程分配大于其工作集的物理块。如果还有空闲物理块，则可以再调一个进程到内存以增加多道程序数。如果所有工作集之和增加以至于超过了可用物理块的总数，那么操作系统会暂停一个进程，将其页面调出并且将其物理块分配给其他进程，防止出现抖动现象。  
  
正确选择工作集的大小，对存储器的利用率和系统吞吐量的提嵩，都将产生重要影响。

管理内存动态分配的方法

缺页中断率算法：（PFF）指出何时增加或者减少分配给一个进程的页面，但是没有说明替换哪一个页面

**负载控制**

当内存容量无法容纳所有进程的工作集时，依然会发生抖动；

将某些进程交换至磁盘，直到消抖

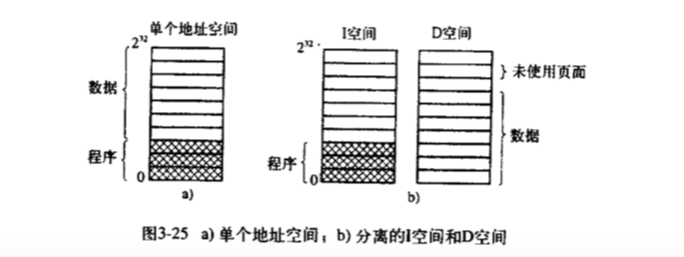
**页面大小：（操作系统的参数）**

有两个理由选择小页面的理由：（1）内部碎片，随便选择一个正文段，数据段，堆栈段可能不会恰好装好，平均一半为空

（2）相比于大页面，利用率更高

随着存储器越来越大，页面页趋向更大

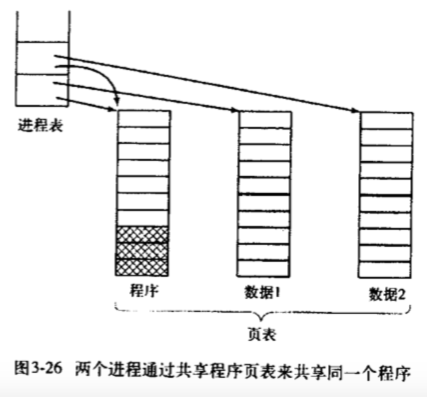
**分离的指令空间和数据空间**



**共享页面**

适合对象：只读页面

每个进程表都有两个指针：一个指向I空间页表，一个指向D空间页表

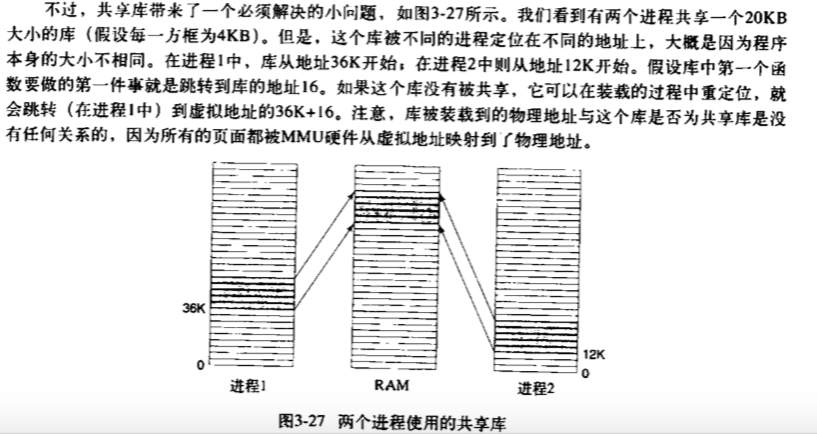


两个（A，B）或者更多进程在共享某些代码时，A被从内存上撤销时，B将会产生大量的缺页中断

共享库：（.so/DLL动态链接库)

优点：（1）使可执行文件更小，节省空间

（2）更新DLL/.so 用来的可执行文件依旧可以使用，因此miro每次更新DLL文件



问题：当进程1.2调用共享库时，需要跳转的地址分别为36 k+ 12k， 12k + 16K；

解决方法（1）；写时复制，并且为每一个共享这个库的进程创建新的页面，在创建的时候重定位，不符合共享库的目的

（2）位置无关代码：在编译共享库的时候，用一个特殊的编译选项告知编译器，不要产生使用绝对地址的指令。相反，只能产生使用相对地址的指令。因此无论共享库放在虚拟内存的什么位置，这种只使用相对偏移量的代码就叫做**位置无关代码**。

**内存映射文件**

其中的一种更为通用的机制特例**共享库**

机制思想：进程可以发起一个系统调用，将一个文件映射到其虚拟地址空间的一部分

两个以上的进程同时映射同一个文件，通过共享内存通信

页守护进程：定期唤醒检查内存状态，如果空闲页框过少，使用页面置换算法；

操作系统在下面四段时间里做的与分页相关：进程创建，进程执行，缺页中断，进程终止

分布式共享内存：该方法允许网络上的多个进程共享一个页面集合，