**互斥锁 和 自旋锁的区别**

Mutex属于sleep-waiting类型的锁。Spin lock则不然，它属于busy-waiting类型的锁

自旋锁与互斥锁有点类似，只是自旋锁不会引起调用者睡眠，如果自旋锁已经被别的执行单元保持，调用者就一直循环在那里看是 否该自旋锁的保持者已经释放了锁，"自旋"一词就是因此而得名。其作用是为了解决某项资源的互斥使用。因为自旋锁不会引起调用者睡眠，所以自旋锁的效率远 高于互斥锁。虽然它的效率比互斥锁高，但是它也有些不足之处：

    1、自旋锁一直占用CPU，他在未获得锁的情况下，一直运行－－自旋，所以占用着CPU，如果不能在很短的时 间内获得锁，这无疑会使CPU效率降低。

    2、在用自旋锁时有可能造成死锁，当递归调用时有可能造成死锁，调用有些其他函数也可能造成死锁，如 copy\_to\_user()、copy\_from\_user()、kmalloc()等。

因此我们要慎重使用自旋锁，自旋锁只有在内核可抢占式或SMP的情况下才真正需要，在单CPU且不可抢占式的内核下，自旋锁的操作为空操作。自旋锁适用于锁使用者保持锁时间比较短的情况下。

互斥锁：线程会从sleep（加锁）——>running（解锁），过程中有上下文的切换，cpu的抢占，信号的发送等开销。

自旋锁：线程一直是running(加锁——>解锁)，死循环检测锁的标志位，机制不复杂。

**两种锁的区别**

互斥锁的起始原始开销要高于自旋锁，但是基本是一劳永逸，临界区持锁时间的大小并不会对互斥锁的开销造成影响，而自旋锁是死循环检测，加锁全程消耗cpu，起始开销虽然低于互斥锁，但是随着持锁时间，加锁的开销是线性增长

**两种锁的应用**

互斥锁用于临界区持锁时间比较长的操作，比如下面这些情况都可以考虑

1 临界区有IO操作

2 临界区代码复杂或者循环量大

3 临界区竞争非常激烈

4 单核处理器

至于自旋锁就主要用在临界区持锁时间非常短且CPU资源不紧张的情况下，自旋锁一般用于多核的服务器。

守护线程

 死锁避免

死锁避免的基本思想：系统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，如果分配后系统可能发生死锁，则不予分配，否则予以分配，这是一种保证系统不进入死锁状态的动态策略。   
如果操作系统能保证所有进程在有限时间内得到需要的全部资源，则系统处于安全状态否则系统是不安全的

内碎片和外碎片

**内部碎片**是处于**（操作系统分配的用于装载某一进程的内存）区域内部的存储块**。占有这些区域或页面的进程并不使用这个[存储](http://baike.baidu.com/view/87682.htm)块。而在进程占有这块存储块时，系统无法利用它。直到进程释放它，或进程结束时，系统才有可能利用这个存储块。

**外部碎片**指的是还**没有被分配出去（不属于任何进程），但由于太小了无法分配**给申请内存空间的新进程的内存空闲区域。

外部碎片是处于任何两个已分配区域或页面之间的空闲存储块。这些存储块的总和可以满足当前申请的长度要求，但是由于它们的地址不连续或其他原因，使得系统无法满足当前申请。

**紧凑技术**(Memory Compaction)解决，通过移动进程在内存中的地址，使得这些外部碎片空间被填满

分段和分页

**段式存储**管理是一种符合用户视角的逻辑单位内存分配管理方案。在段式存储管理中，将程序的地址空间划分为若干段（segment），如代码段，数据段，堆栈段；这样**每个进程有一个二维地址空间**，相互独立，互不干扰。段式管理的优点是：**没有内碎片**（因为段大小可变，改变段大小来消除内碎片）。但段**换入换出时，会产生外碎片**（比如4k的段换5k的段，会产生1k的外碎片）

在**页式存储**管理中，将程序的逻辑地址划分为固定大小的页（page），而物理内存划分为同样大小的帧，程序加载时，可以将任意一页放入内存中任意一个帧，这些帧不必连续，从而实现了离散分离。页式存储管理的优点是：没有外碎片（因为页的大小固定），但会产生内碎片（一个页可能填充不满）。

**两者的不同点：**

**目的不同**：分页是由于系统管理的需要而不是用户的需要，它是信息的物理单位；是一个一维的地址空间。分段的目的是为了能更好地满足用户的需要，它是信息的逻辑单位，是一个二维的地址空间，它含有一组其意义相对完整的信息；

**大小不同**：页的大小固定且由系统决定，而段的长度却不固定，由其所完成的功能决定；

**地址空间不同**： 段向用户提供二维地址空间；页向用户提供的是一维地址空间；

**信息共享**：段是信息的逻辑单位，便于存储保护和信息的共享，页的保护和共享受到限制；

**内存碎片**：页式存储管理的优点是没有外碎片（因为页的大小固定），但会产生内碎片（一个页可能填充不满）；而段式管理的优点是没有内碎片（因为段大小可变，改变段大小来消除内碎片）。但段换入换出时，会产生外碎片（比如4k的段换5k的段，会产生1k的外碎片）。

**操作系统中进程调度策略有哪几种？**

FCFS(先来先服务，队列实现，非抢占的)：先请求CPU的进程先分配到CPU

SJF(最短作业优先调度算法)：平均等待时间最短，但难以知道下一个CPU区间长度

优先级调度算法(可以是抢占的，也可以是非抢占的)：优先级越高越先分配到CPU，相同优先级先到先服务，存在的主要问题是：低优先级进程无穷等待CPU，会导致无穷阻塞或饥饿；解决方案：最高响应比算法。

时间片轮转调度算法(可抢占的)：队列中没有进程被分配超过一个时间片的CPU时间，除非它是唯一可运行的进程。如果进程的CPU区间超过了一个时间片，那么该进程就被抢占并放回就绪队列。

多级反馈队列调度算法：将就绪队列分成多个独立的队列，每个队列都有自己的调度算法，队列之间采用固定优先级抢占调度。允许进程在队列之间移动：若进程使用过多CPU时间，那么它会被转移到更低的优先级队列；在较低优先级队列等待时间过长的进程会被转移到更高优先级队列，以防止饥饿发生。

**局部性原理**

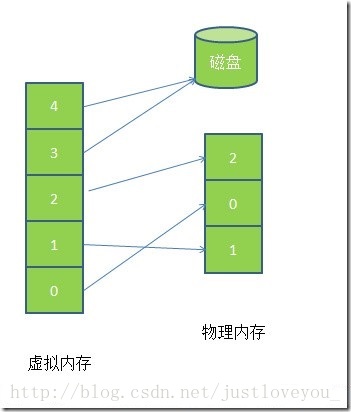
(1). 时间上的局部性：最近被访问的页在不久的将来还会被访问；

(2). 空间上的局部性：内存中被访问的页周围的页也很可能被访问。

**虚拟内存：**

虚拟内存允许执行进程不必完全在内存中。虚拟内存的基本思想是：每个进程拥有独立的地址空间，这个空间被分为大小相等的多个块，称为页(Page)，每个页都是一段连续的地址。这些页被映射到物理内存，但并不是所有的页都必须在内存中才能运行程序。当程序引用到一部分在物理内存中的地址空间时，由硬件立刻进行必要的映射；当程序引用到一部分不在物理内存中的地址空间时，由操作系统负责将缺失的部分装入物理内存并重新执行失败的命令。这样，对于进程而言，逻辑上似乎有很大的内存空间，实际上其中一部分对应物理内存上的一块(称为帧，通常页和帧大小相等)，还有一些没加载在内存中的对应在硬盘上，如图所示。

注意，**请求分页系统、请求分段系统和请求段页式系统都是针对虚拟内存的**，通过请求实现内存与外存的信息置换。



由图5可以看出，虚拟内存实际上可以比物理内存大。当访问虚拟内存时，**会访问MMU（内存管理单元）去匹配对应的物理地址**（比如图5的0，1，2）。如果虚拟内存的页并不存在于物理内存中（如图5的3,4），会产生缺页中断，从磁盘中取得缺的页放入内存，如果内存已满，还会根据某种算法将磁盘中的页换出。

**页面置换算法**

OPT（Optimal replacement）最优置换算法：理论的最优，理论；就是要保证置换出去的是不再被使用的页，或者是在实际内存中最晚使用的算法。

FIFO先进先出算法：在操作系统中经常被用到，比如作业调度（主要实现简单，很容易想到）；可能

LRU（Least recently use）最近最少使用算法：根据使用时间到现在的长短来判断；

LFU（Least frequently use）最少使用次数算法：根据使用次数来判断；

NRU （Not Recently Used）这种算法给每个页一个标志位，R表示最近被访问过，M表示被修改过。定期对R进行清零。这个算法的思路是首先淘汰那些未被访问过R=0的页，其次是被访问过R=1,未被修改过M=0的页，最后是R=1,M=1的页。

**颠簸，抖动**

　颠簸本质上是指频繁的页调度行为，具体来讲，进程发生缺页中断，这时，必须置换某一页。然而，其他所有的页都在使用，它置换一个页，但又立刻再次需要这个页。因此，会不断产生缺页中断，导致整个系统的效率急剧下降，这种现象称为颠簸（抖动）。

　　内存颠簸的解决策略包括：

如果是因为页面替换策略失误，可以修改替换算法来解决这个问题；

如果是因为运行的程序太多，造成程序无法同时将所有频繁访问的页面调入内存，则要降低多道程序的数量；

否则，还剩下两个办法：终止该进程或增加物理内存容量。