单道批处理系统中，一个作业单位进入内存并独占系统资源，知道运行结束后下一个作业才能进入内存。当作业进行IO操作时，CPU只能处于等待状态。CPU利用率低下。

多道批处理系统，在内存中可以同时存在若干个作业，作业执行顺序与进入内存的次序没有严格对应的关系。一个作业在等待IO处理时，CPU调度另外一个作业运行。

因此CPU的利用率显著提高

分时系统-般进程调度算法是时间片轮转算法。多个终端用户可以通过彼此的终端相互独立地使用计算机，计算机按照时间片轮转算法，给每一个用户进程分配合适的时间片，处理用户请求。

常用排序算法的性能与待排数组的循序的关系作一个总结：  
  
1）冒泡：无关；  
2）选择：无关；  
3）插入：有关，有序程度越大，比较越少；  
4）shell：有关，它的基本思想基于插入排序；  
5）合并：有关，有序程度愈大，合并过程的比较次数越少；  
6）堆排序：有关，有序程度越大，建立堆下沉操作越少；  
7）快排序：有关，如果选择最后值作为阀值，那么有序程度越好，就越可能退化成O(n^2)；  
  无关，随机选择阀值，那么与排序程度无关。

对称矩阵压缩后存储到一维数组中，假设存储下三角的元素，那么存储的元素个数为：第一行1个、第二行2个、第三行3个。。。第10行10个，一共1+2+3+...+10=55个元素（即n(n+1)/2）

线程间共享的资源  
1.文件描述符表  
2.每种信号的处理方式  
3.当前工作目录  
4.用户ID和组ID

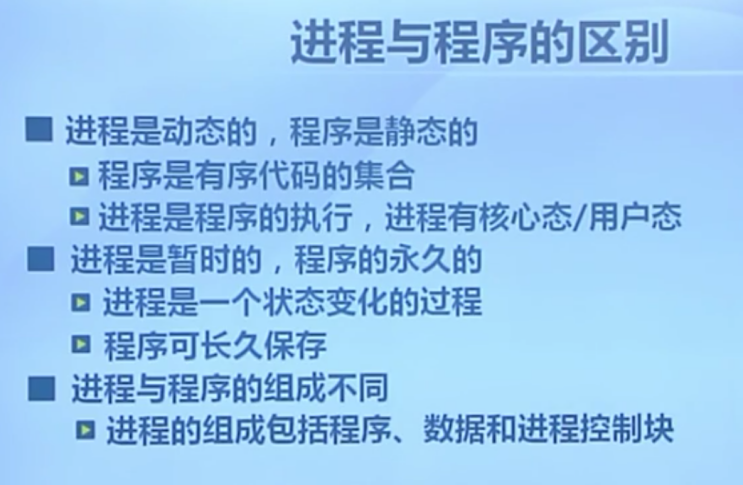
5.内存地址空间

线程间非共享资源  
1.线程id  
2.处理器现场和栈指针(内核栈)  
3.独立的栈空间(用户空间栈)  
4.errno变量  
5.信号屏蔽字  
6.调度优先级

设文件F1的当前引用计数值为1，先建立F1的符号链接（软链接）文件F2，再建立F1的硬链接文件F3，然后删除F1。此时，F2和F3的引用计数值分别是（）。

建立符号链接时，引用计数值直接复制；建立硬链接时，引用计数值加1。删除文件时，删除操作对于符号链接是不可见的，这并不影响文件系统，当以后再通过符号链接访问时，发现文件不存在，直接删除符号链接；但对于硬链接则不可以直接删除，引用计数值减1，若值不为0，则不能删除此文件，因为还有其他硬链接指向此文件。

当建立F2时，F1和F2的引用计数值都为1。当再建立F3时，F1和F3的引用计数值就都变成了2。当后来删除F1时，F3的引用计数值为2-1=1，F2的引用计数值一直不变。

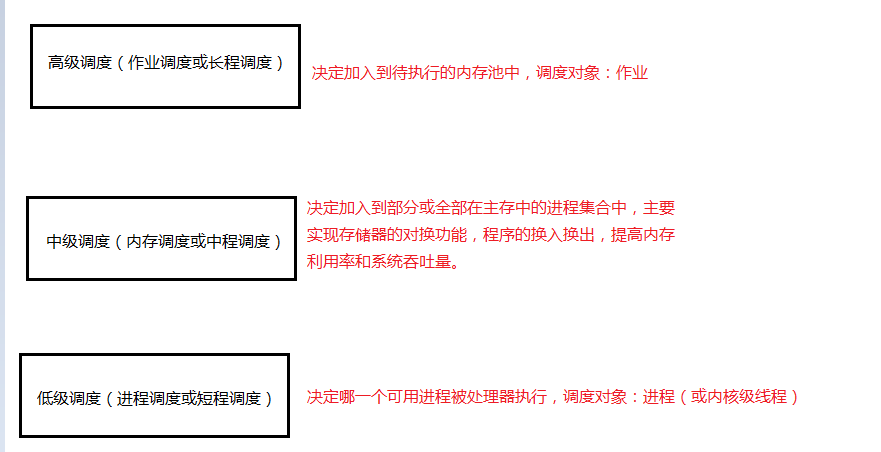


线程的同步主要是指执行的有先有后这样的顺便，比如一个线程的执行依赖另一个线程的某种消息或者条件，当他没有得到这个消息的时候应该等待，直到消息到达才被唤醒

线程的互斥主要是指执行中的线程对共享的进程系统资源的排它性，当有多个线程都要使用某一个共享资源时，任何时刻最多只允许一个线程去使用，其他使用该资源的线程必须等待，直到占用资源者释放资源

在单道批处理系统中，一个作业单独进入内存并独占系统资源，直到运行结束后下一个作业才能进入内存，当作业进行I/O操作时，CPU只能处于等待状态，因此，CPU利用率较低，尤其是对于I/O操作时间较长的作业。为了提高CPU的利用率，在单道批处理系统的基础上引入了[多道程序设计](https://baike.baidu.com/item/%E5%A4%9A%E9%81%93%E7%A8%8B%E5%BA%8F%E8%AE%BE%E8%AE%A1" \t "/home/ts/Documents\\x/_blank)（multiprogramming）技术，这就形成了多道批处理系统，即在内存中可同时存在若干道作业，作业执行的次序与进入内存的次序无严格的对应关系，因为这些作业是通过一定的[作业调度算法](https://baike.baidu.com/item/%E4%BD%9C%E4%B8%9A%E8%B0%83%E5%BA%A6%E7%AE%97%E6%B3%95" \t "/home/ts/Documents\\x/_blank)来使用CPU的，一个作业在等待I/O处理时，CPU调度另外一个作业运行，因此CPU的利用率显著地提高了。批处理系统的目的是提高系统吞吐量和资源的利用率。

多道程序能交替使用CPU，提高了CPU及其他系统资源的利用率，同时也提高了系统的效率。多道批处理系统的缺点是延长了作业的[周转时间](https://baike.baidu.com/item/%E5%91%A8%E8%BD%AC%E6%97%B6%E9%97%B4" \t "/home/ts/Documents\\x/_blank)，用户不能进行直接干预，缺少[交互性](https://baike.baidu.com/item/%E4%BA%A4%E4%BA%92%E6%80%A7" \t "/home/ts/Documents\\x/_blank)，不利于程序的开发与调试



**响应比=（等待时间+执行时间）/执行时间**

由于进行分页式存储管理时，先把相对地址变为数对：(页号，页内位移)，用页号去查页表，得到绝对地址后，才能真正去访问该地址。所以，每对内存进行一次读写，都要访问两次内存：查页表一次，读写一次

首先，页式存储是一维的，段式存储是二维的。

其次，这是两种存储管理的****设计初衷决定的****，****跟逻辑地址的结构没有关系****。

在程序员的角度，你想操作一个内存地址，

（1）页式存储，你直接给****一个逻辑地址就行****，所以是一维的。

虽然这个逻辑地址包含（页号，偏移量），但是，分页那是系统自己完成的事，我们不知道也不用关心最终地址到底在哪个页上。

（2）段式存储，你****必须给出（段号，偏移量）****，所以是二维的。

因为分段的好处就是程序模块化，不同的段权限不同，所以在哪个段上，是我们关心的。

Linux进程间通信：管道、信号、消息队列、共享内存、信号量、套接字(socket)

Linux线程间通信：互斥量（mutex），信号量，条件变量

Windows进程间通信：管道、消息队列、共享内存、信号量 （semaphore） 、套接字(socket)

Windows线程间通信：互斥量（mutex），信号量（semaphore）、临界区（critical section）、事件（event）

挂起：一般是主动的，由系统或程序发出，甚至于辅存中去。  
阻塞：一般是被动的，在抢占资源中得不到资源，被动的挂起在内存，等待某种资源或信号量。

。

按王道里面的说法，进程由运行状态转为****阻塞状态是一种主动的行为****，

进程在请求资源或等待I/O完成等情况下****主动****由运行转为****阻塞****。

而****进程调度****是将运行状态的进程转为****就绪状态****，这是被动的。

****。****

****挂起是被动的。****引起挂起的原因有：

用户请求；  
父进程请求；  
负载调节的需要；  
操作系统的需要。  
由此可见，从运行到挂起应该是被动的。

进程是操作系统资源分配的基本单位，而线程是任务调度和执行的基本单位。