http://blog.csdn.net/sugar\_z\_/article/details/51526696

操作系统的基本特征

1. 并发：同一段时间内多个程序执行(注意区别并发和并行，前者是同一时刻的多个事件，后者是统一时间段内的多个事件)
2. 共享：系统中的资源可以被内存中多个并发执行的进线程共同使用
3. 虚拟：通过时分复用（如分时系统）以及空分复用（如虚拟内存）技术实现把一个物理实体虚拟为多个
4. 异步：系统中的进程是以走走停停的方式执行的，且以一种不可预知的速度推进

操作系统的主要功能

1. 处理机管理：处理机分配都是以进程为单位，所以处理机管理也被看做是进程管理。包括进程控制，进程同步，进程通信和进程调度
2. 存储器管理（或者内存管理）：内存分配，内存保护，地址映射，内存扩充
3. 设备管理：管理所有外围设备，包括完成用户的IO请求；为用户进程分配IO设备；提高IO设备利用率；提高IO速度；方便IO的使用
4. 文件管理：管理用户文件和系统文件，方便使用同时保证安全性。包括：磁盘存储空间管理，目录管理，文件读写管理以及文件共享和保护
5. 提供用户接口：程序接口（如API）和用户接口（如GUI）

进程和线程的区别

进程：进程是进程实体的运行过程，是系统进行资源分配和调度的一个独立单位（具有动态、并发、独立、异步的特性，以及就绪、执行、阻塞3种状态；也有5状态或者7状态；资源拥有单位的属性）；引入进程是为了使多个程序可以并发的执行，以提高系统的资源利用率和吞吐量。

线程：是比进程更小的可独立运行的基本单位，可以看做是轻量级的进程（具有轻型实体，独立调度分派单位，可并发执行，共享进程资源等属性）；引入目的是为了减少程序在并发执行过程中的开销，使OS的并发效率更高。

两者的对比：   
1. 调度方面：在引入线程的OS中，线程是独立的调度和分派单位，而进程作为资源的拥有单位(相当于把未引入线程的传统OS中的进程的两个属性分开了)。由于线程不拥有资源，因此可以显著的提高并发度以及减少切换开销。   
2. 并发性：引入了线程的OS中，进程间可以并发，而且一个进程内部的多个线程之间也是可以并发的，这就使OS具有更好的并发性，有效的提高了系统资源利用率和吞吐量。   
3. 拥有资源：无论OS是否支持线程，进程都是基本的资源拥有单位，线程只拥有很少的基本的资源，但是线程可以访问所隶属的进程的资源（进程的代码段，数据段和所拥有的系统资源如fd）   
4. 系统开销：创建或者撤销进程的时候，系统要为之创建或回收PCB，系统资源等，切换时也需要保存和恢复CPU环境。而线程的切换只需要保存和恢复少量的寄存器，不涉及存储器管理方面的工作，所以开销较小。此外，统一进程中的多个线程由于共享地址空间，所以通信同步等都比较方便。

进程的几种状态

主要是3中基本状态，5状态和7状态可以直接看书   
1. 就绪状态：进程获得了除了CPU之外的所有的必要资源，只要获得CPU就可以立即执行，此时的进程处于就绪态   
2. 执行状态：进程已经获得CPU，正在运行，在多处理其系统中，会有多个进程同时处于运行状态   
3. 阻塞状态：处于执行状态的进程由于发生某些事件而暂时无法继续执行，放弃处理机而处于暂停状态，此时进程就处于阻塞（执行受到阻塞）状态

就绪->执行：调度进程为其分配了处理机   
执行->就绪：时间片用完   
执行->阻塞：申请临界资源而未被满足，如IO请求或者申请缓存   
阻塞->就绪：请求得到满足，如IO完成

进程同步

多进程虽然提高了系统资源利用率和吞吐量，但是由于进程的异步性可能造成系统的混乱。进程同步的任务就是对多个相关进程在执行顺序上进行协调，使并发执行的多个进程之间可以有效的共享资源和相互合作，保证程序执行的可再现性

同步机制需要遵循的原则：   
1. 空闲让进：当没有进程处于临界区的时候，应该许可其他进程进入临界区的申请   
2. 忙则等待：当前如果有进程处于临界区，如果有其他进程申请进入，则必须等待，保证对临界区的互斥访问   
3. 有限等待：对要求访问临界资源的进程，需要在有限时间呃逆进入临界区，防止出现死等   
4. 让权等待：当进程无法进入临界区的时候，需要释放处理机，边陷入忙等

经典的进程同步问题：生产者-消费者问题；哲学家进餐问题；读者-写者问题

进程间通信

进程通信就是指进程间的信息交换，交换信息可以使一个状态，也可以是很多的byte。进程间同步互斥也存在信息的交换，因此也属于是一种IPC，属于是低级通信。该低级通信存在的问题：1）通信的数据量太少；2）通信对用户不透明(数据的传递或者同步互斥都需要程序员实现)

高级通信机制（高级通信的通信细节被OS隐藏，因此使用起来增加方便而且可以传送大量的数据，尤其是管道通信）：   
1. 共享存储器系统：相互通信的进程共享某些数据结构或者是存储区，进程之间可以通过这些共享空间进行通信。分为：1）基于共享数据结构的通信，如生产者消费者系统中的有界缓冲区；2）基于共享存储区的通信，可以传输大量数据，通信的进程之间可以像读写普通存储器一样读写共享存储区   
2. 消息传递系统：进程间通信采用的是格式化的消息，可以直接使用OS提供的消息发送或者接受原语进行通信。由于隐藏了通信细节，所以简化了通信程序的复杂性   
3. 管道通信：管道是连接两个一个读进程和一个写进程之间用于实现数据交换的一个共享文件。为了协调管道通信双方，需要管道机制实现如下功能：1）互斥：统一时刻只能有一个进程对管道进行读写；2）同步：当读端发现管道为空的时候需要睡眠等待，直到有数据时候被唤醒，相应的写端也是在管道已满的时候等待直到被唤醒；3）确定对方的存在性：只有同时有读端和写端，管道才有存在意义

进程/任务调度算法

基本调度算法：   
1. 先来先服务调度算法FCFS：既可以作为作业调度算法也可以作为进程调度算法；按作业或者进程到达的先后顺序依次调度；因此对于长作业比较有利；   
2. 短作业优先调度算法SJ(P)F：作业调度算法，算法从就绪队列中选择估计时间最短的作业进行处理，直到得出结果或者无法继续执行；缺点：不利于长作业；未考虑作业的重要性；运行时间是预估的，并不靠谱   
3. 高优先权优先调度算法HPF：既可以作为作业调度也可以作为进程调度算法；调度作业时，从就绪队列中选择优先级最高的作业进行处理；由于涉及到了优先级，因此可以分为抢占式和非抢占式；而且优先级的确定也可以分为静态优先级（事先根据进程类型，进程对资源的需求，用户要求等方面确定一个固定值）；动态优先级（随进程的推进或者等待时间而增加或者减少）   
4. 高相应比算法HRN：响应比=(等待时间+要求服务时间)/要求服务时间；   
5. 时间片轮转调度RR：按到达的先后对进程放入队列中，然后给队首进程分配CPU时间片，时间片用完之后计时器发出中断，暂停当前进程并将其放到队列尾部，循环   
6. 多级反馈队列调度算法：目前公认较好的调度算法；设置多个就绪队列并为每个队列设置不同的优先级，第一个队列优先级最高，其余依次递减。优先级越高的队列分配的时间片越短，进程到达之后按FCFS放入第一个队列，如果调度执行后没有完成，那么放到第二个队列尾部等待调度，如果第二次调度仍然没有完成，放入第三队列尾部…。只有当前一个队列为空的时候才会去调度下一个队列的进程。

实时调度算法：   
1. 最早截止时间优先调度算法EDF：算法根据任务的开始截止时间确定优先级，截止时间越早，优先级越高。算法维护一个实时就绪队列，最早截止时间的任务排在最前面；可以用于抢占式调度也可以用于非抢占式调度；   
2. 最低松弛度优先调度算法LLF：松弛度=(必须完成时间-本身运行时间-当前时间);算法根据任务的松弛度确定任务的优先级，松弛度代表了任务的紧急程度，任务的紧急程度越高，被赋予的优先级越高

死锁的必要条件以及处理方式

死锁是指多个进程在运行过程中，因为争夺资源而造成的一种僵局，如果没有外力推进，处于僵局中的进程就无法继续执行。

死锁原因：   
1. 竞争资源：请求同一有限资源的进程数多于可用资源数   
2. 进程推进顺序非法：进程执行中，请求和释放资源顺序不合理，如资源等待链

死锁产生的必要条件：   
1. 互斥条件:进程对所分配的资源进行排他性的使用   
2. 请求和保持条件：进程被阻塞的时候并不释放锁申请到的资源   
3. 不可剥夺条件：进程对于已经申请到的资源在使用完成之前不可以被剥夺   
4. 环路等待条件：发生死锁的时候存在的一个 进程-资源 环形等待链

死锁处理：   
1. 预防死锁：破坏产生死锁的4个必要条件中的一个或者多个；实现起来比较简单，但是如果限制过于严格会降低系统资源利用率以及吞吐量   
2. 避免死锁：在资源的动态分配中，防止系统进入不安全状态(可能产生死锁的状态)-如银行家算法   
3. 检测死锁：允许系统运行过程中产生死锁，在死锁发生之后，采用一定的算法进行检测，并确定与死锁相关的资源和进程，采取相关方法清除检测到的死锁。实现难度大   
4. 解除死锁：与死锁检测配合，将系统从死锁中解脱出来（撤销进程或者剥夺资源）。对检测到的和死锁相关的进程以及资源，通过撤销或者挂起的方式，释放一些资源并将其分配给处于阻塞状态的进程，使其转变为就绪态。实现难度大

死锁定理：S为死锁状态的充分条件是，当且仅当S的资源分配图是不能完全简化的

内存管理方式-段式页式和段页式

由于连续内存分配方式(单一连续分配，固定分区分配，动态分区分配，动态重定位分区分配)导致的内存利用率偏低以及内存碎片的问题，进而引出离散的内存分配方式。离散内存分配可以从OS的内存管理角度引出页式(离散分配的基本单位是页)管理，也可以从程序编制角度引出段式(离散分配的基本单位是段)管理。

1. 基本分页存储管理

基本分页存储管理中不具备页面置换功能(即没有实现虚拟内存的功能)，因此需要整个程序的所有页面都装入内存之后才可以运行。因为程序数据存储在不同的页面中，而页面又离散的分布在内存中，因此需要一个**页表来记录逻辑地址和实际存储地址之间的映射关系，以实现从页号到物理块号的映射**。由于页表也是存储在内存中的，因此和不适用分页管理的存储方式相比，访问分页系统中内存数据需要两次的内存访问(一次是从内存中访问页表，从中找到指定的物理块号，加上页内偏移得到实际物理地址；第二次就是根据第一次得到的物理地址访问内存取出数据)。   
为了**减少两次访问内存导致的效率影响，分页管理中引入了快表(或者联想寄存器)机制**，包含快表机制的内存管理中，当要访问内存数据的时候，首先将页号在快表中查询，如果查找到说明要访问的页表项在快表中，那么直接从快表中读取相应的物理块号；如果没有找到，那么访问内存中的页表，从页表中得到物理地址，同时将页表中的该映射表项添加到快表中(可能存在快表换出算法)。   
在某些**计算机中如果内存的逻辑地址很大，将会导致程序的页表项会很多，而页表在内存中是连续存放的，所以相应的就需要较大的连续内存空间。为了解决这个问题，可以采用两级页表或者多级页表的方法**，其中外层页表一次性调入内存且连续存放，内层页表离散存放。相应的访问内存页表的时候需要一次地址变换，访问逻辑地址对应的物理地址的时候也需要一次地址变换，而且一共需要访问内存3次才可以读取一次数据。

1. 基本分段存储管理方式

分页是为了提高内存利用率，而分段是为了满足程序员在编写代码的时候的一些逻辑需求(比如数据共享，数据保护，动态链接等)。   
分段内存管理当中，地址是二维的，一维是段号，一维是段内地址；其中每个段的长度是不一样的，而且每个段内部都是从0开始编址的。由于分段管理中，每个段内部是连续内存分配，但是段和段之间是离散分配的，因此也存在一个逻辑地址到物理地址的映射关系，相应的就是段表机制。段表中的每一个表项记录了该段在内存中的起始地址和该段的长度。段表可以放在内存中也可以放在寄存器中。   
访问内存的时候根据段号和段表项的长度计算当前访问段在段表中的位置，然后访问段表，得到该段的物理地址，根据该物理地址以及段内偏移量就可以得到需要访问的内存。由于也是两次内存访问，所以分段管理中同样引入了联想寄存器。

分段和分页的对比：   
1. 页是信息的物理单位，是出于系统内存利用率的角度提出的离散分配机制；段是信息的逻辑单位，每个段含有一组意义完整的信息，是出于用户角度提出的内存管理机制   
2. 页的大小是固定的，由系统决定；段的大小是不确定的，由用户决定   
3. 页地址空间是一维的，段地址空间是二维的

1. 段页式存储管理

先将用户程序分为若干个段，然后再把每个段分成若干个页，并且为每一个段赋予一个段名称。这样在段页式管理中，一个内存地址就由段号，段内页号以及页内地址三个部分组成。   
段页式内存访问：系统中设置了一个段表寄存器，存放段表的起始地址和段表的长度。地址变换时，根据给定的段号（还需要将段号和寄存器中的段表长度进行比较防止越界）以及寄存器中的段表起始地址，就可以得到该段对应的段表项，从段表项中得到该段对应的页表的起始地址，然后利用逻辑地址中的段内页号从页表中找到页表项，从该页表项中的物理块地址以及逻辑地址中的页内地址拼接出物理地址，最后用这个物理地址访问得到所需数据。由于访问一个数据需要三次内存访问，所以段页式管理中也引入了高速缓冲寄存器。

虚拟内存及页面置换算法

如果存在一个程序，所需内存空间超过了计算机可以提供的实际内存，那么由于该程序无法装入内存所以也就无法运行。单纯的增加物理内存只能解决一部分问题，但是仍然会出现无法装入单个或者无法同时装入多个程序的问题。但是可以从逻辑的角度扩充内存容量，即可解决上述两种问题。

**虚拟存储器就是具有请求调入功能和置换功能，可以从逻辑上对内存容量加以扩充的一种存储器系统。虚拟存储器都是建立在离散内存管理的基础上**

虚拟存储器的特征：   
1. 多次性：一个作业可以分多次被调入内存。多次性是虚拟存储特有的属性   
2. 对换性：作业运行过程中存在换进换出的过程(换出暂时不用的数据换入需要的数据)   
3. 虚拟性：虚拟性体现在其从逻辑上扩充了内存的容量(可以运行实际内存需求比物理内存大的应用程序)。虚拟性是虚拟存储器的最重要特征也是其最终目标。虚拟性建立在多次性和对换性的基础上行，多次性和对换性又建立在离散分配的基础上

1. 页面置换算法
2. 最佳置换算法：只具有理论意义的算法，用来评价其他页面置换算法。置换策略是将当前页面中在未来最长时间内不会被访问的页置换出去。
3. 先进先出置换算法：简单粗暴的一种置换算法，没有考虑页面访问频率信息。每次淘汰最早调入的页面
4. 最近最久未使用算法LRU：算法赋予每个页面一个访问字段，用来记录上次页面被访问到现在所经历的时间t，每次置换的时候把t值最大的页面置换出去(实现方面可以采用寄存器或者栈的方式实现)
5. 时钟算法clock(也被称为是最近未使用算法NRU)：页面设置一个访问为，并将页面链接为一个环形队列，页面被访问的时候访问位设置为1。页面置换的时候，如果当前指针所指页面访问为为0，那么置换，否则将其置为0，循环直到遇到一个访问为位0的页面
6. 改进型Clock算法：在Clock算法的基础上添加一个修改位，替换时根究访问位和修改位综合判断。优先替换访问为何修改位都是0的页面，其次是访问位为0修改位为1的页面。
7. 最少使用算法LFU：设置寄存器记录页面被访问次数，每次置换的时候置换当前访问次数最少的。存在问题是该访问寄存器并不能真正反映当前页面访问次数，因为访问速度比较快，所以在更新寄存器的时间间隔内访问1次和访问100次都是一样的。另外，LFU和LRU是很类似的，支持硬件也是一样的，但是区分两者的关键在于一个以时间为标准，一个以次数为标准(例如对于寄存器 pa 001111 和pb 111000，两个页面，如果采用LRU，那么被淘汰的是pa，如果采用LFU那么被淘汰的是pb)。
8. 页面缓冲算法PBA：置换的时候，页面无论是否被修改过，都不被置换到磁盘，而是先暂留在内存中的页面链表(已修改页面链表和未修改页面链表，也可以不区分)里面，当其再次被访问的时候可以直接从这些链表中取出而不必进行磁盘IO，当链表中已修改也难数目达到一定数量之后，进行依次写磁盘操作(相当于将多次IO合并为一次)