进程：资源拥有者，负责资源分配

线程：指令执行者

多线程优点：

1. 响应快
2. 共享资源：进程通信快
3. 经济：硬件需求低。进程创建需要的内存和资源分配比较昂贵
4. 利用了多处理器的价值

多线程模式：

1. 多对一：多个用户线程映射到内核级线程。从操作系统角度属于单线程；用户角度是多线程。优点：快速创建用户级线程。缺点：可能会阻塞用户级线程，不能并行。
2. 一对一：开销大
3. 多对多：

进程通信：

1. 共享存储器系统：在共享存储系统中，相互通信的进程共享某些数据结构或共享存储区，进程通过这些空间进行通信。

* 基于共享数据结构的通信方式
* 基于共享存储区的通信方式：在内存中划分一块共享存储区，数据的形式和位置和访问控制由进程负责，而不是OS。

1. 管道通信系统：管道是用于连接一个读进程和一个写进程以实现它们之间的通信的一个共享文件。必须实现三个方面的协调：

* 互斥
* 同步，当写进程把一定数量的数据写入pipe时，便去睡眠等待，直到读进程取走数据后再把它唤醒。当读进程读一空pipe时，也应睡眠等待，直到读进程取走数据后再把它唤醒。
* 确定对方是否存在。

1. 消息传递系统：以格式化的消息为单位，将通信数据封装在消息中，并利用操作系统提供的一组通信命令，在进程间进行消息传递。
2. 客户机-服务器系统

消息传递的方式：

1. 直接通信：是指发送进程利用OS所提供的发送原语，直接把消息发送给目标进程。
2. 间接通信：指发送和接收进程，都通过共享中间实体的方式进行消息的发送和接收，完成进程间的通信。

线程的运行状态：

1. 执行状态
2. 就绪状态
3. 阻塞状态

进程在多线程中不在是可执行的实体。

进程控制块（PCB）：

1. 堆栈：临时数据
2. 堆：在进程运行期间动态分配内存
3. 数据段：全局变量
4. 文本段：程序代码

长期调度：调度对象是作业。其主要功能是根据某种算法，决定将外存上处于后备队列中的哪几个作业调入内存，为它们创建进程、分配必要的资源，并将它们放入就绪队列。

短期调度：调度对象是进程（或内核级线程）。根据某种算法，决定就绪队列的哪个进程应获得处理机，并由分派程序将处理机分配给被选中的进程。

中期调度：内存调度。主要目的是提高内存利用率和系统吞吐量。为此，应把那些暂时不能运行的进程调制外存等待，此时进程状态称为就绪驻外存状态。当它们已具备运行条件且内存又稍有空闲时，由中级调度来决定，把外存上那些已具备运行条件的就绪程序再重新调入内存，并修改其状态为就绪状态，挂在就绪队列上等待。

调度准则：

1. CPU利用率：
2. 吞吐量：单位时间内系统所完成的作业数
3. 周转时间：进程创建到终止时间
4. 等待时间：进程在内存等待时间
5. 响应时间：创建进程到获得CPU时间

强占式调度：如果一个进程有执行能力，调度算法剥夺其CPU周期为强占式调度。

强占式：等待时间>响应时间

非强占式：等待时间=响应时间。

调度算法：

1. 先到先服务：大量短CPU运行时间在长CPU运行时间之后，形成护航效果。
2. 最短作业优先：

* 强占式：最短剩余时间优先。缺点：CPU切换大。且不知道作业CPU运行时间
* 非强占式

可以用指数平均预测。

1. 轮转法调度:响应时间最短，时间片比较小。缺点：上下文切换大。
2. 多级队列：将进程就绪队列分成若干个，将不同类型或性质的进程固定分配在不同的就绪队列，不同的就绪队列采用不同的调度算法。（适合多处理机模式）
3. 多级反馈队列：
4. 优先级调度算法
5. 高响应比优先调度算法。

进程调度的任务：

1. 保存现场
2. 选取进程
3. 把处理器分配给进程（由分派器完成）。

PCB中的信息：

1. 进程标识符
2. 处理机状态，即上下文
3. 进程调度信息：进程状态、进程优先级、进程调度需要的其他信息、事件（进程阻塞的原因）。
4. 进程控制信息

系统态：能执行一切指令，访问所有寄存器和存储区

用户态：仅能执行规定的指令，访问规定的寄存器和存储区。

进程阻塞与挂起的区别：阻塞的进程还在内存中，挂起的进程在外存。

进程同步的原则：

1. 互斥：多个线程不可同时使用临界区
2. 前进：当无进程处于临界区时，应允许一个请求进入临界区的进程立即进入自己的临界区，以有效的利用资源。当进程使用完临界区之后，要退出临界区。
3. 有限等待。不能一直给一个进程临界区，进程滞留临界区的时间有限
4. 让权等待：当进程不能进入自己的临界区时，应立即释放处理机，以免进程陷入忙等待。

解决进程同步算法：

1. 皮特森算法
2. 中断屏蔽
3. Swap指令
4. 信号量

产生死锁的必要条件：

1. 互斥条件
2. 请求和保持条件
3. 不可抢占
4. 循环等待。在发生死锁时，必然存在一个进程---资源的循环链。

处理死锁的方法：

1. 预防死锁：破坏死锁条件
2. 避免死锁：在资源动态分配过程中，用某种方法防止系统进入不安全的状态，从而可以避免发生死锁。

* 银行家算法

1. 检测死锁

* 资源分配图：不可完全简化

1. 解除死锁

* 终止所有死锁进程
* 逐个终止进程

外部碎片：连续分配法导致的外部碎片。单个内存空洞空间不够，不足以连续分配，虽然多个空洞加起来满足条件。

内部碎片：一个进程快存入一个空洞，如果一个空洞大小大于进程，就会产生内部碎片。

进程划分为固定大小块——页

内存划分为固定大小块——帧

进程不连续分配到内存——不存在外部碎片。代价：每个进程需要维护页表。页表记录了每一页对应的物理块号。

快表：频繁使用的页号抽取，形成快表。

逻辑地址：CPU产生的虚拟地址

物理地址：内存单元所看到的地址（即加载到内存的地址寄存器）

编译时绑定地址：

加载时绑定：

执行时绑定：

MMU：逻辑地址与物理地址的映射。

模式切换：用户模式——内核模：开销大

进程切换：状态改变；开销大；保存PCB和恢复PCB

动态分区分配算法：

1. 首次适应：最快
2. 最佳适应：空间利用率最高
3. 最差适应

分页存储管理方式：进程和内存地址空间都划分为相同的固定大小的快。

页面过小的话每个进程的页表过长，占用大量的内存，同时降低换出换进的效率。页面过大导致页内碎片增大。

分页存储管理的地址结构：页号+偏移量（页内地址）

页表寄存器：记录页表始址和页表长度。

页表大都驻留在内存中。进程未执行时，页表的始址和页表长度保存在PCB中。

基本地址变换机构处理逻辑：

当进程要访问某个逻辑地址中的数据时，分页地址变换结构会自动地将有效地址分为页号和页内地址两部分，在以页号索引去检索页表。查找操作由硬件执行。在执行检索之前，先将页号与页表长度进行比较，如果页号大于或等于页表长度，则表示本次访问的地址已经超越进程地址空间。若为出现越界错误，则将页表始址与页号和页表项长度的乘积相加，得到该表项在页表中的位置，于是可从中得到该页的物理块号，将之装入物理地址寄存器中。与此同时，再将有效地址寄存器中的页内地址送入物理地址寄存器的块内存中。两次访问内存，依次是访问内存中的页表，从中得到指定页的物理块号。第二次访问是从第一次访问中获得的所需数据。

具有快表的地址变换结构。

页表占用内存空间比较大的情况下的方法，需要比较大而且连续的内存空间不现实。解决方法：

1. 采用离散分配的方式解决连续分配的问题，将一个大的页表分成多个小的页表，两级或多级页表。
2. 只将当前需要部分页表向调入内存，其余驻留在磁盘上。

反置页表：以物理空间作为索引号。减少内存空间的占用。

虚拟存储器：解决一次作业内存太大无法一次调入内存的问题。

局部性规律：即在一较短时间内，程序的执行仅局限与某个部分，相应的，它所访问的存储空间也局限于某个区域。

虚拟存储器的基本工作情况：

程序运行时，如果它所要访问的页已经调入内存，便可继续执行下去；但如果程序所要访问的页尚未调入内存，便发出缺页请求，此时OS将利用请求调页功能将它们调入内存。如果内存已满，则需要置换功能。

虚拟存储器定义：指具有请求调入功能和置换功能，能从逻辑上对内存容量加以扩充的一种存储器系统。

页面置换算法：

1. 最佳置换算法
2. 先进先出算法
3. 最近最久未使用算法：局部性规律。
4. 最近最少使用算法
5. 简单Clock置换算法：当某页被访问时，访问位设置为1.置换算法在选择一页淘汰时，之需检查页的访问位。如果是0，则换出；否则，置为0，给予第二次机会。
6. 改进CLock置换算法：页面修改过的换出并重新写回磁盘开销较大。因此，改进的clock算算法，除了考虑页的使用情况之外，增加一个置换代价。页面换出是，即首要的是未使用过的和未被修改过的。其次考虑最近未被访问，但已修改的。

页面缓冲算法：将要写回磁盘的页面暂时不写入，先缓存起来。当要访问这个页面的时候直接从缓存中获取。

影响页面换进换出效率的因素：置换算法、将已修改页面写回磁盘的频率、以及将磁盘内容读入内存的频率。