**一.操作系统概述**

**1. 操作系统的四个特性**

并发：同一段时间内多个程序执行(注意区别并行和并发，前者是同一时刻的多个事件，后者是同一时间段内的多个事件)

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

虚拟：通过时分复用（如多道程序技术,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

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

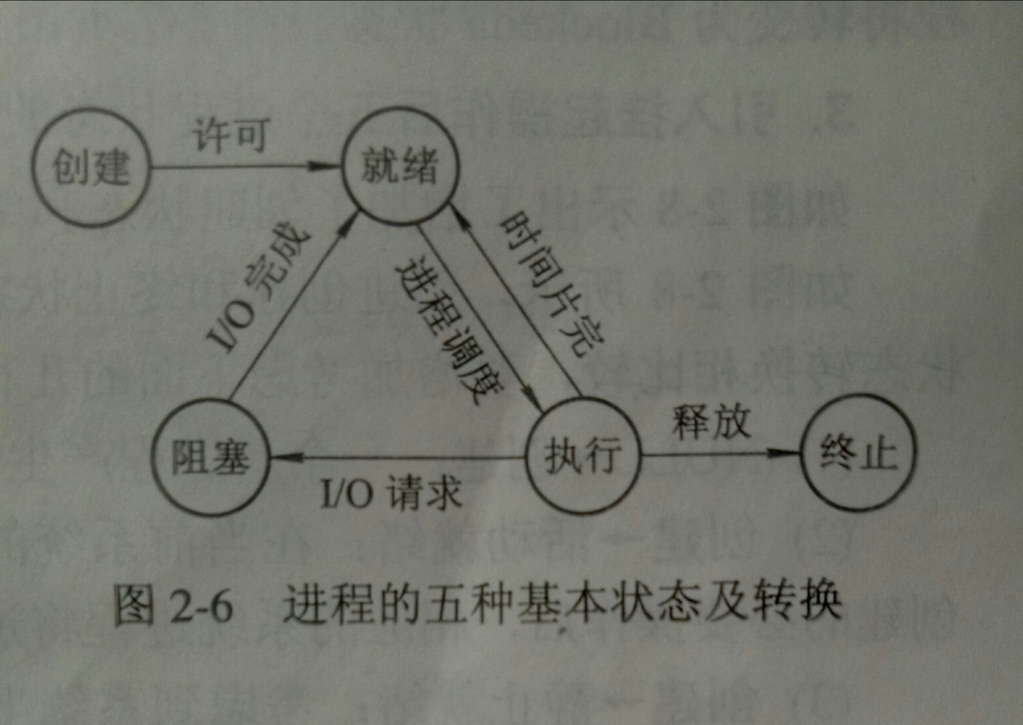
**2.操作系统的主要功能**

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，进程通信和调度(作业调度和进程调度)  
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  
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  
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：文件存储空间管理，目录管理，文件读写管理以及文件共享和保护  
提供用户接口：程序接口（如API）和用户接口（如GUI）

二.进程和线程

**3.进程的状态与转换**

　　进程的三种基本状态：就绪(Ready)状态，执行(Running)状态，阻塞(Block)状态  (引入的两种常见的状态：创建状态，终止状态)



运行状态：进程正在处理机上运行。在单处理机环境下，每一时刻最多只有一个进程处于运行状态。

就绪状态：进程已处于准备运行的状态，即进程获得了除处理机之外的一切所需资源，一旦得到处理机即可运行。

阻塞状态，又称等待状态：进程正在等待某一事件而暂停运行，如等待某资源为可用（不包括处理机）或等待输入/输出完成。即使处理机空闲，该进程也不能运行。

**4.进程同步的几种机制**

同步机制需要遵循的原则：

　　1.空闲让进，2.忙则等待，3.有限等待，4.让权等待

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

进程同步的机制：硬件同步机制、信号量机制、管程机制

　　硬件同步机制：关中断

　　　　　　　　　利用Test-and-Set指令实现互斥(Test-and-Set TS指令：”测试并建立“)

　　　　　　　　　利用Swap指令实现进程互斥(该指令称为对换指令，用于交换两个字的内容)

　　信号量(Semaphores)机制

　　管程机制

**5.进程的通信方式有哪些？**

主要分为：管道、系统IPC（包括消息队列、信号量、共享存储器）、SOCKET、远程过程(函数)调用RPC

管道主要分为：普通管道PIPE 、流管道（s\_pipe）、命名管道（name\_pipe）

# 管道( pipe )：管道是一种半双工的通信方式，数据只能单向流动，而且只能在具有亲缘关系的进程间使用。进程的亲缘关系通常是指父子进程关系。

# 有名管道 (named pipe) ： 有名管道也是半双工的通信方式，但是它允许无亲缘关系进程间的通信。

# 信号量( semophore ) ： 信号量是一个计数器，可以用来控制多个进程对共享资源的访问。它常作为一种锁机制，防止某进程正在访问共享资源时，其他进程也访问该资源。因此，主要作为进程间以及同一进程内不同线程之间的同步手段。

# 消息队列( message queue ) ： 消息队列是由消息的链表，存放在内核中并由消息队列标识符标识。消息队列克服了信号传递信息少、管道只能承载无格式字节流以及缓冲区大小受限等缺点。

# 信号 ( sinal ) ： 信号是一种比较复杂的通信方式，用于通知接收进程某个事件已经发生。

# 共享内存( shared memory ) ：共享内存就是映射一段能被其他进程所访问的内存，这段共享内存由一个进程创建，但多个进程都可以访问。共享内存是最快的 IPC 方式，它是针对其他进程间通信方式运行效率低而专门设计的。它往往与其他通信机制，如信号两，配合使用，来实现进程间的同步和通信。

# 套接字( socket ) ： 套解口也是一种进程间通信机制，与其他通信机制不同的是，它可用于不同及其间的进程通信。

6.请分别简单说一说进程和线程以及它们的区别

* 进程是具有一定功能的程序关于某个数据集合上的一次运行活动，进程是系统进行资源调度和分派的一个独立单位。
* 线程是进程的实体，是CPU调度和分派的基本单位，它是比进程更小的能独立运行的基本单位，线程是轻量级的进程。
* 一个进程可以有多个线程，多个线程也可以并发执行

7.线程同步的方式有哪些？

线程同步的方式：

* 临界区：通过对多线程的串行化来访问公共资源或者一段代码，速度快，适合控制数据访问
* 互斥量：采用互斥对象机制，只有拥有互斥对象的线程才有访问公共资源的权限，因为互斥对象只有一个，所以可以保证公共资源不会同时被多个线程访问
* 信号量：它允许多个线程同一时刻访问同一资源，但是需要限制同一时刻访问此资源的最大线程数目。信号量对象对线程的同步方式与前面几种方法不同，信号允许多个线程同时使用共享资源，这与操作系统中PV操作相似。
* 事件（信号）：通过通知操作的方式来保持多线程的同步，还可以方便的实现多线程的优先级比较的操作

**三.处理机调度与死锁**

**8.用户态和核心态**

当程序运行在3级特权级上时，就可以称之为运行在用户态，因为这是最低特权级，是普通的用户进程运行的特权级，大部分用户直接面对的程序都是运行在用户态；

反之，当程序运行在级特权级上时，就可以称之为运行在内核态。

用户态切换到内核态的3种方式：系统调用，异常，外围设备的中断

9.处理机调度的层次

　　高级调度(长程调度或作业调度):调度对象是作业，主要用于多道批处理系统中，而在分时和实时系统中不设置高级调度

　　低级调度(进程调度或短程调度):调度对象是进程，在多道批处理，分时和实时三种类型的OS中，都必须配置这级调度

　　中级调度:内存调度，引入中级调度的主要目的是，提高内存利用率和系统吞吐量，中级调度实际上就是存储器管理中的对换功能

10.死锁

死锁的定义：所谓死锁就是一个进程集合中的多个进程因为竞争资源，而造成的互相等待现象。

死锁的原因：系统资源不足；多个进程的推进顺序不合理

死锁的必要条件：

* 互斥条件（Mutual exclusion）：资源不能被共享，只能由一个进程使用。
* 请求与保持条件（Hold and wait）：已经得到资源的进程可以再次申请新的资源。
* 不可抢占条件（No pre-emption）：进程对于已经申请到的资源在使用完成之前不可以被抢占
* 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

**11.进程调度算法 操作系统中进程调度策略有哪几种？**

FCFS(先来先服务)，优先级，时间片轮转，多级反馈

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；

短作业优先调度算法SJF：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 ；

优先级调度算法PSA:基于作业的紧迫程度，由外部赋予作业相应的优先级，调度算法是根据该优先级进行调度的

高相应比优先调度算法HRNN：响应比=(等待时间+要求服务时间)/要求服务时间=响应时间/要求服务时间

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 ;

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。

四.存储器管理

**12.内存连续分配**

连续分配方式：单一连续分配:单道程序环境下

固定分区分配：多道程序系统，每个分区的大小是固定的

动态分区分配

动态可重定位分区分配　　算法四种方式

主要是指动态分区分配时所采用的几种算法。  
动态分区分配又称为可变分区分配，它是动态划分内存的分区方法。这种分区方法不预先将内存划分，而是在进程装入内存时，根据进程的大小动态地建立分区，并使分区的大小正好适合进程的需要。因此系统中分区的大小和数目是可变的。

基于顺序搜索的动态分区分配算法

　　首次适应(First Fit)算法：空闲分区以地址递增的次序链接。分配内存时顺序查找，找到大小能满足要求的第一个空闲分区。

　　最佳适应(Best Fit)算法：空闲分区按容量递增形成分区链，找到第一个能满足要求的空闲分区。

　　最坏适应(Worst Fit)算法：又称最大适应(Largest Fit)算法，空闲分区以容量递减的次序链接。找到第一个能满足要求的空闲分区，也就是挑选出最大的分区。

基于索引搜索的动态分区分配算法

　　快速适应(quick fit)算法：又称为分类搜索法，是将空闲分区根据其容量大小进行分类，对于每一类具有相同容量的所有空闲分区，单独设立一个空闲分区链表

　　　　　　　　　　　　　这样系统中存在多个空闲分区链表

　　伙伴系统(buddy system)

　　哈希算法：构造一张以空闲分区大小为关键字的哈希表，该表的每一个表项记录了一个对应的空闲分区链表表头指针

**13.基本分页存储管理方式**

离散分配方式：将一个进程直接分散地装入到许多不相邻接的分区中，便可充分利用内存空间

离散分配分为三种：分页存储管理方式、分段存储管理方式、段页式存储管理方式

     分页存储管理方式：将用户程序的地址空间分为若干个固定大小的区域，称为”页“或”页面“。相应的，也将内存空间分为若干个物理块或页框(frame)，页和块的大小相同。这样就可以将用户程序的任一页放入任一物 理块中，实现了离散分配。

把主存空间划分为大小相等且固定的块，块相对较小，作为主存的基本单位。每个进程也以块为单位进行划分，进程在执行时，以块为单位逐个申请主存中的块空间。

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。

**14.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。

**分段分页方式的比较**

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

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

五.虚拟存储器

**15.虚拟内存**

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。

基于局部性原理，在程序装入时，可以将程序的一部分装入内存，而将其余部分留在外存，就可以启动程序执行。在程序执行过程中，当所访问的信息不在内存时，由操作系统将所需要的部分调入内存,然后继续执行程序。另一方面，操作系统将内存中暂时不使用的内容换出到外存上，从而腾出空间存放将要调入内存的信息。这样，系统好像为用户提供了一个比实际内存大得多的存储器，称为虚拟存储器。

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

虚拟存储器的实现方法：分页请求系统，请求分段系统

**16.页面置换算法(服务于分页请求系统)**

最佳(Optimal)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。

先进先出(FIFO)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。

最近最久未使用LRU(Least Recently Used)置换算法：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)。

最少使用LFU(Least Frequently Used)置换算法：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。

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。优先替换访问位和修改位都是0的页面，其次是访问位为0修改位为1的页面。

页面缓冲算法(Page Buffering Algorithm,PBA)

进程管理

1.进程和线程以及他们的区别

进程是对运行时程序的封装，是系统进行资源调度和分配的基本单位，实现了操作系统的并发性；

线程是进程的子任务，是CPU调度和分派的基本单位，用于保证程序的实时性，实现进程内部的并发；

一个程序至少有一个进程，一个进程至少有一个线程，线程依赖于进程为存在；

进程在执行过程中拥有独立的内存单元，而多个线程共享进程的内存。从操作系统的角度来看，进程=程序+数据+PCB（进程控制块）

2.进程有哪几种状态

就绪状态：进程已获得处理机以外的所需资源，等待分配处理机资源

运行状态：占用处理机资源运行，处于此状态的进程数小于等于CPU数

阻塞状态：进程等待某种条件，在条件满足之前无法执行

孤儿进程，僵尸进程？产生的原因，这两种进程对系统的影响（如：会不会消耗系统资源，如何处理这两种进程，在编程时如何避免这两种进程）

3.线程有哪几种状态

new：在这个状态中，一个线程还没有开始

runable：在这个状态中，线程在java虚拟机上执行

blocked：在这个状态线程被阻塞

waiting：等待状态，等待被唤醒

time waiting：等待时间满后自动环形

terminated：停止

线程的状态转换如下：

4.进程间的通信

进程之间的信息交换成为进程通信。根据交换信息量的多少，进程通信分为两种类型：低级通信和高级通信。低级通信传送的信息量少，主要用于控制信息的传送；高级通信是指进程间大批量的数据交换。

进程通信的类型

（1）共享存储器通信：交互进程之间有一个可以直接访问的共享存储区，并发进程通过对这个共享存储区的读和写进行信息交换。如基于共享数据结构的通信方式（如公用数据结构，只适合传递少量数据，如信号量，低级）；基于共享存储区的通信方式（如公共内存块，高级），在这种通信方式下，操作系统只负责为通信进程提供可使用的存储空间和进程同步工具，数据交换由用户进程通过读、写命令来完成。

（2）消息传递系统（高级）：进程之间的数据交换以格式化的消息为单位，放在内核中。这种通信方式可以进一步划分为：直接通信方式（通过发送和接受原语来实现）和间接通信方式（如邮箱通信方式）。

邮箱通信方式

备注：一般地，把系统态下执行的某些具有特定功能的程序段成为原语；原语的执行必须是连续的

（3）管道（pipe)及命名管道（named pipe，FIFO)通信方式：管道：是指连接一个读进程和一个写进程之间用以实现通信的共享文件，也称pipe文件。UNIX中的管道是一个环形的缓冲区，允许两个进程以生产者/消费者模型进行通信。管道可用于具有亲缘关系的父子进程间的通信，命名管道除了具有管道所具有的功能外，他还允许无亲缘关系进程间的通信。

（4）信号(signal)：信号是一种比较复杂的通信方式，用于通知接收进程某个事件已经发生，信号产生的条件：按键、硬件异常，进程调用kill函数将信号发送给另一个进程、用户调用kill命令将信号发送给其他进程，传递的消息比较少用于通知接收进程某个时间已经发生；

（5）信号量：主要作为进程之间（系统西濠梁）及同一种进程的不同线程之间（私有信号量）的同步和互斥手段；

（6）套接字（Socket）：这是一种更为一般得进程间的通信机制，它可用于网络中不同机器之间的进程间通信，应用非常广泛。（网络上的两个程序通过一个双向的通信连接实现数据的交换，这个链接的一端称为一个socket）

几种方式的比较：

管道：速度慢、容量有限

消息队列：容量收到系统限制，且要注意第一次读的时候，要考虑上一次没有读完数据的问题。

信号量：不能传递复杂信息，只能用来同步。

共享内存：能够很容易控制容量，速度快，但要保持同步，比如一个进程在写的时候，另一个进程要注意读写的问题，相当于线程中的线程安全。

5.进程同步的方式

原子操作、信号量机制、临界区、自旋锁管程、会合、分布式系统

各方法的大致原理，详细理解至少一种，尝试用伪代码实现一种

6.线程同步的方式

（1）互斥量（Synchronized/Lock):采用互斥对象的线程才有访问公共资源的权限。因为互斥对象只有一个，所以可以保证公共资源不会被多个线程同时访问

（2）信号量（Semphare)：它允许同一时刻多个线程访问同一资源，但需要控制同一时刻访问此资源的最大线程数量

（3）事件（信号），wait/Notify：通过通知操作的方式来保持多线程同步，还可以方便的实现多线程优先级的比较操作

（4）临界区：通过对多线程的串行化来访问公共资源或者一段代码，速度快，适合控制数据访问。

线程池的原理及作用

线程池与一般的多线程由什么区别、优点及使用场景，尝试用代码实现

进程调度

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）：平均等待时间最短，但难以指导下一个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：老化

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，除非他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，那么该进程就会被抢占并放回就绪队列

多级队列调度算法：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。其中，一个进程根据自身属性被永久地分配到一个队列中。

多级反馈队列调度算法：与多级队列调度算法相比，其允许进程在队列之间移动；若进程使用过多的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。

常见的同步模型

生产者消费者模型，尝试用代码实现

7.什么是死锁？死锁产生的条件？

（1）死锁的概念

在两个或多个并发进程中，如果每个进程持有某种资源而又等待其它进程释放它或它们现在保持着的资源，在未改变这种状态之前都不能向前推进，称这一组进程产生了死锁。通俗的讲，就是两个或多个进程无限期的阻塞、相互等待的一种状态。

（2）死锁产生的四个必要条件

互斥：至少有一个资源必须属于非共享模式，即一次只能被一个进程使用；若其他申请使用该资源，那么申请进程必须等到该资源被释放为止；

占有并等待：一个进程必须占有至少一个资源，并等待另一个资源，而该资源为其他进程所占有。

非抢占：进程不能被抢占，即资源只能被进程在完成任务后自愿释放

循环等待：若干进程之间形成一种头尾相接的环形等待资源。当每类资源数量只有1个时，有环不一定有死锁！

（3）死锁的处理基本策略和常用方法

解决死锁常用的方法主要有：预防死锁、避免死锁、检测死锁、解除死锁、鸵鸟策略 等。

1）死锁预防

死锁预防的基本思想是只要确保死锁发生的四个必要条件中至少有一个不成立，就能预防死锁的发生，具体方法包括：

打破互斥条件：允许进程同时访问某些资源。但是，有些资源是不能被多个进程所共享的，这是由资源本身属性所决定的，因此，这种办法通常无实用价值。

打破占有并等待条件：可以实行资源预先分配策略（进程在运行前一次性向系统申请它所需要的全部资源时，才一次性将所申请资源分配给该线程）或者只允许进程在没有占用资源时才可以申请资源（一个进程可申请一些资源并使用他们，但是在当前进程申请更多资源前，它必须释放当前所占有的资源）。 \*\*缺点：\*\*在很多情况下，无法预知一个进程执行前所需的全部资源，因为进程是动态执行，不可预知的；同时，会降低资源利用率，导致降低了进程的并发性。

打破非抢占条件：当一个进程占有了一部分资源，在其申请新的资源得不到满足时，它必须释放所有占有的资源。\*\*缺点：\*\*实现困难，会降低系统性能。

打破循环等待：实行资源有序分配策略。对所有资源排序编号，所有进程对资源的请求必须严格按资源序号递增的顺序提出，即只有占用了小号资源才能申请大号资源，这样就不会产生环路，预防死锁的发生。这种方法也称“有序资源分配法”。

2）死锁避免的基本思想

死锁避免的基本思想是动态的检测资源分配状态，以确定循环等待条件不成立，从而确保系统处于安全状态。所谓的安全状态是指：如果系统能按某个顺序为每个进程分配资源（不超过其最大值），那么系统状态的状态是安全的，换句话说就是，如果存在一个安全序列，那么系统处于安全状态，资源分配图算法和银行家算法是两种经典的死锁避免的算法，其可以确保系统始终处于安全状态。其中，资源分配图算法应用场景为每种资源类型只有一个实例（申请边，分配边，需求边，不形成环才允许分配），而银行家算法应用于每种资源类型可以有多个实例的场景。

3）死锁解除

死锁解除的常用两种方法为进程终止和资源抢占。所谓的进程终止是指简单地终止一个或多个进程以打破循环等待，包括两种方式：终止所有死锁进程和一次只终止一个进程直至取消死锁循环位置；所谓资源抢占是指从一个或多个死锁进程哪里抢占一个或多个资源，此时必须考虑三个问题：

选择一个牺牲品

回滚：回滚到安全状态

饥饿（在代价因素加上回滚次数，回滚的越多则越不可能继续被作为牺牲品，避免一个进程总是被回滚）

计算：

x\*(m-1)\_1=n

n为总资源数 m为进程需要数，x为允许的进程数

存储器管理&文件系统、IO

windows下的内存是如何管理的

3种

虚拟内存 最适合用来管理大型对象或者结构数组

内存映射 最适合用来管理大型数据流（通常来自文件）以及在单个计算机上运行多个进程之间共享数据

内存堆栈 最适合用来管理大量的小对象

物理内存与虚拟内存

分页与分段

页是信息的物理单位，分页是为了实现离散分配方式，以减少内存的外零头，提高内存的利用率。分页仅仅是由于系统管理的需要，而不是用户的需要。

段是信息的逻辑单位，它含有一组其意义相对完整的信息。分段的目的是为了能更好的满足用户的需要。

页的大小固定且由系统确定，把逻辑地址分为页号和页内地址两部分，由机器硬件实现的。因此一个系统只能由一种大小的页面。段的长度却不固定，决定于用户所编写的程序，通常由编写程序对源代码进行编辑时，根据信息的性质来划分。

分页的作业地址空间是一维的，即单一的线性空间。

分段的作业地址空间是二维的，程序在标识一个地址时，既需要给出段名，又需要给出段内地址。

页面置换算法

（了解大致原理，并能详述其中一种）

最佳置换算法（Optimal Page Replacement Algorithm） 是将未来最久不使用的页替换出去，这听起来很简单，但是无法实现，但是这种算法可以作为衡量其他算法的基准

最近不常使用算法（Not Recently Used Replacement Algorithm） 算法给每个页一个标志位，R表示最近被访问国，M表示被修改果。定期对R进行清零。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。

先进先出页面置换算法（First-In，First-Out Page Replacement Algorithm） 算法的思想是淘汰在内存中最久的页，算法的性能接近与随机淘汰。并不好。

改进型FIFO算法（Second Change Page Replacement Algorithm） 算法是在FIFO的基础上，为了避免置换出经常使用的页，增加一个标志位，如果最近使用过将R置为1，当页将会淘汰时，如果R为1，则不淘汰页，将R置为0.而那些R=0的页将会被淘汰时，直接淘汰。这首那个算法避免了经常被使用的页被淘汰。

时钟替换算法（Clock Page Replacement Algorithm） 虽然改进型FIFO算法避免置换出常用的页，但由于需要经常移动页，效率并不高。因此在改进型FIFO算法的基础上，将队列首位相连形成一个环路，当缺页中断产生时，从当前位置开始找R=0的页，而所经过的R=1的页被置0，并不需要移动页。

最久未使用算法（LRU Page Replacement Algorithm） LRU算法的思路是淘汰最近最长未使用的页。这种算法性能比较好，但实现起来比较困难。