**1. 进程和线程的区别。**

（1）进程是操作系统**分配和管理资源**的单位，线程是CPU**调度和管理**的单位，是CPU调度的最小单元。它们拥有的资源也不相同

（2）进程拥有独立的地址空间，而线程间共享地址空间

（3）进程创建的开销比较大，线程创建的开销小

（4）一个进程拥有多个线程，线程可以创建线程

**2. 死锁的必要条件，怎么处理死锁。**

答案：

死锁是指多个进程因**循环等待**资源而造成无法执行的现象。死锁会造成进程无法执行，同时会造成系统资源的极大浪费(资源无法释放)。

**死锁产生的四个必要条件**

* 互斥使用：指进程对所分配到的资源进行排它性使用，即在一段时间内某资源只由一个进程占用。如果此时还有其它进程请求资源，则请求者只能等待，直至占有资源的进程用毕释放。
* 不可抢占：指进程已获得的资源，在未使用完之前，不能被剥夺，只能在使用完时由自己释放。
* 请求和保持：指进程已经保持至少一个资源，但又提出了新的资源请求，而该资源已被其它进程占有，此时请求进程阻塞，但又对自己已获得的其它资源保持不放。
* 循环等待：指在发生死锁时，必然存在一个进程——资源的环形链，即进程集合{P0，P1，P2，···，Pn}中的P0正在等待一个P1占用的资源；P1正在等待P2占用的资源，……，Pn正在等待已被P0占用的资源。

**死锁避免**

银行家算法：判断此次请求是否造成死锁若会造成死锁，则拒绝该请求。

**3.内存管理方式：段存储，页存储，段页存储。**

**内存为程序分配空间有四种分配方式：**

1. **连续分配方式**

首先讲连续分配方式。**连续分配方式**出现的时间比较早，曾广泛应用于20世纪60~70年代的OS中，但是它至今仍然在内存管理方式中占有一席之地，原因在于它**实现起来比较方便，所需的硬件支持最少**。连续分配方式又可细分为四种：**单一连续分配、固定分区分配、动态分区分配和动态重定位分区分配**。

　　其中固定分区分配方式，因为分区固定，所以缺乏灵活性，即**当程序太小时，会造成内存空间的浪费（内部碎片）**；**程序太大时，一个分区又不足以容纳，致使程序无法运行（外部碎片）**。但尽管如此，当一台计算机去控制多个相同对象的时候，由于这些对象内存大小相同，所以完全可以采用这种内存管理方式，而且是最高效的。这里我们可以看出存储器管理机制的多面性：即没有那种存储器管理机制是完全没有用的，在适合的场合下，一种被认为最不合理的分配方案却可能称为最高效的分配方案。**一切都要从实际问题出发，进行设计。**

 　　为了解决固定分区分配方式的缺乏灵活性，出现了**动态分配方式**。动态分配方式采用一些**寻表（Eg：空闲链表）**的方式，查找能符合程序需要的空闲内存分区。但代价是增加了系统运行的开销，而且内存空闲表本身是一个文件，必然会占用一部分宝贵的内存资源，而且有些算法还会增加内存碎片。

 　　可重定位分区分配通过对程序实现成定位，从而可以将内存块进行搬移，将小块拼成大块，将小空闲“紧凑”成大空闲，腾出较大的内存以容纳新的程序进程。

1. **基本分页存储管理方式**

**连续分配方式**会形成许多“碎片”，虽然可以通过“紧凑”方式将许多碎片拼接成可用的大块空间，但须为之付出很大开销。所以提出了“**离散分配方式**”的想法。如果**离散分配的基本单位是页，则称为分页管理方式；如果离散分配的基本单位是段，则称为分段管理方式**。

 　　分页存储管理是将一个进程的逻辑地址空间分成若干个大小相等的片，称为页面或页，并为各页加以编号，从0开始，如第0页、第1页等。相应地，也把内存空间分成与页面相同大小的若干个存储块，称为(物理)块或页框(frame)，也同样为它们加以编号，如0#块、1#块等等。在为进程分配内存时，以块为单位将进程中的若干个页分别装入到多个可以不相邻接的物理块中。由于进程的最后一页经常装不满一块而形成了不可利用的碎片，称之为“**页内碎片**”。

**在分页系统中，允许将进程的各个页离散地存储在内存不同的物理块中（所以能实现离散分配方式）**，但系统应能保证进程的正确运行，即能在内存中找到每个页面所对应的物理块。为此，系统又为每个进程建立了一张页面映像表，简称**页表**。在进程地址空间内的所有页，依次在页表中有一页表项，其中记录了相应页在内存中对应的物理块号。在配置了页表后，进程执行时，通过查找该表，即可找到每页在内存中的物理块号。可见，**页表的作用是实现从页号到物理块号的地址映射**。

 　　为了能够将用户地址空间中的**逻辑地址，变换为内存空间中的物理地址**，在系统中必须设置**地址变换机构**。地址变换任务是借助于页表来完成的。

**页表**的功能可由一组专门的寄存器来实现。由于寄存器成本较高，且大多数现代计算机的页表又很大，使页表项总数可达几千甚至几十万个，显然这些页表项不可能都用寄存器来实现，因此，页表大多驻留在内存中。因为一个进程可以通过它的PCB来时时保存自己的状态，等到CPU要处理它的时候才将PCB交给寄存器，所以，系统中虽然可以运行多个进程，但也只需要一个页表寄存器就可以了。

**由于页表是存放在内存中的，这使得CPU在每存取一个数据时，都要两次访问内存。为了提高地址变换速度，在地址变化机构中增设了一个具有并行查询能力的告诉缓冲寄存器，又称为“联想寄存器”（Associative Lookaside Buffer）。**

　　在单级页表的基础上，为了适应非常大的逻辑地址空间，出现了两级和多级页表，但是，他们的原理和单级页表是一样的，只不过为了适应地址变换层次的增加，需要在地址变换机构中增设外层的页表寄存器。

1. **基本分段存储管理方式**

**分段存储管理方式**的目的，主要是为了满足用户（程序员）在编程和使用上多方面的要求，其中有些要求是其他几种存储管理方式所难以满足的。因此，这种存储管理方式已成为当今所有存储管理方式的基础。

（1）方便编程；

（2）**信息共享**：分页系统中的“页”只是存放信息的物理单位（块），并无完整的意义，不便于实现共享；然而段却是信息的逻辑单位。由此可知，为了实现段的共享，希望存储器管理能与用户程序分段的组织方式相适应。

（3）信息保护；

（4）**动态增长**；

（5）**动态链接**。

　　分段管理方式和分页管理方式在实现思路上是很相似的，只不过他们的基本单位不同。分段有**段表**，也有**地址变换机构**，为了提高检索速度，同样增设**联想寄存器（具有并行查询能力的告诉缓冲寄存器）**。所以有些具体细节在这个不再赘述。

**分页和分段的主要区别：**

1、两者相似之处：两者**都采用离散分配方式，且都要通过地址映射机构来实现地址变换**。

2、两者不同之处：

（1）页是信息的**物理单位**，分页是为实现离散分配方式，以消减内存的外零头，提高内存的利用率。或者说，分页仅仅是由于**系统管理的需要**而不是用户的需要。段则是信息的**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地满足用户的需要**。

（2）**页的大小固定**且由系统决定，而**段的长度却不固定**。

（3）分页的作业地址空间是**一维**的，即单一的线性地址空间；而分段的作业地址空间则是**二维**的。

1. **段页式存储管理方式**

　　前面所介绍的分页和分段存储管理方式都各有优缺点。**分页系统能有效地提高内存利用率，而分段系统则能很好地满足用户需求。**我们希望能够把两者的优点结合，于是出现了段页式存储管理方式。

　　段页式系统的基本原理，是分段和分页原理的结合，即**先将用户程序分成若干个段，再把每个段分成若干个页**，并为每一个段赋予一个段名。在段页式系统中，地址结构由段号、段内页号和页内地址三部分所组成。

　　和前两种存储管理方式相同，段页式存储管理方式同样需要增设联想寄存器。

**离散分配方式**基于将一个进程直接分散地分配到许多不相邻的分区中的思想，分为分页式存储管理，分段存储管理和段页式存储管理. 分页式存储管理旨在提高内存利用率，满足系统管理的需要，分段式存储管理则旨在满足用户(程序员)的需要，在实现共享和保护方面优于分页式存储管理，而段页式存储管理则是将两者结合起来，取长补短，即具有分段系统便于实现，可共享，易于保护，可动态链接等优点，又能像分页系统那样很好的解决外部碎片的问题，以及为各个分段可离散分配内存等问题，显然是一种比较有效的存储管理方式。

**5、页面置换算法：**

**FIFO算法**：先入先出，即淘汰最早调入的页面。

**OPT(MIN)算法**：选未来最远将使用的页淘汰，是一种最优的方案，可以证明缺页数最小。可惜，MIN需要知道将来发生的事，只能在理论中存在，实际不可应用。

**LRU(Least-Recently-Used)算法**：用过去的历史预测将来，选最近最长时间没有使用的页淘汰(也称最近最少使用)。性能最接近OPT。**与页面使用时间有关**。

**LFU(Least Frequently Used)算法**：即最不经常使用页置换算法，要求在页置换时置换引用计数最小的页，因为经常使用的页应该有一个较大的引用次数。**与页面使用次数有关**。

**6、内部碎片和外部碎片的区别**

（1）内部碎片：由于采用分页的方式分配出去的内存的单位是页，而程序使用的内存不一定占满一页从而产生的未使用的内存称为内部碎片；

（2）外部碎片：即传统的内存碎片，分配过的内存被回收之后由于太小不能被利用的即为外部碎片。

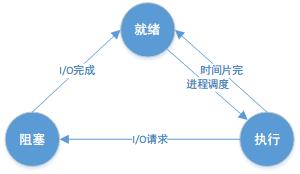
**4. 进程的几种状态。**

进程的基本状态：

(1) 阻塞态：等待某个事件的完成

(2) 就绪态：等待系统分配处理器以便运行

(3) 执行态：占有处理器正在运行



执行态 -> 阻塞态：往往是由于等待外设，等待主存等资源分配或等待人工干预而引起的。

阻塞态 -> 就绪态：则是等待的条件已满足，只需分配到处理器后就能运行。

执行态 -> 就绪态：不是由于自身原因，而是由外界原因使运行状态的进程让出处理器，这时候就变成就绪态。例如时间片用完，或有更高优先级的进程来抢占处理器等。

就绪态 -> 执行态：系统按某种策略选中就绪队列中的一个进程占用处理器，此时就变成了运行态

**5. IPC几种通信方式。**

（1）管道：半双工，一个可读一个可写；

（2）消息队列

（3）共享内存

（4）socket

（5）共享文件

**6. 什么是虚拟内存。**

虚拟内存的基本思想是每个程序都拥有自己的地址空间，这个空间被分割成多个块，每块成称为一页或者页面，每一页有连续的地址范围，这些页被映射到物理内存中，但不是所有页都必须在物理内存中才能运行。当程序引用自己地址空间的内容时，由硬件执行必要的映射，如果引用不再内存中的地址时，由操作系统负责将缺失部分装入物理内存中并重新执行失败的指令。

7. 虚拟地址、逻辑地址、线性地址、物理地址的区别。

**8. 线程间同步的方式**

（1）互斥量

（2）信号量

**9. 常见的进程调度策略**

（1）调度的分类

**非抢占式：**分派程序一旦把处理器分配给某个进程就让他一直运行下去，直到进程完成或发生调度进程调度某事件而阻塞时，才把处理机分配给另一个进程

**抢占式：**操作系统将正在运行的进程强行暂停，由调度程序将CPU分配给其他就绪进程的调度方式。

（2）常见调度策略

(1)FCFS:调度的顺序就是任务到达就绪队列的顺序，对短作业不公平

(2)SJF:短作业优先调度

(3)HRN:最高响应比优先法（响应比：R=1+W/T W:作业在等待队列中等待的时间，T:估计作业执行的时间）

(4)优先权调度:每个任务关联一个优先级，调度优先级最高的任务

(5)Round Robin(RR):设置一个时间片，按时间片来轮转调度

**10、中断的概念**

中断（英语：Interrupt）是指 处理器接收到来自硬件或软件的信号，提示发生了某个事件，应该被注意，这种情况就称为中断。通常，在接收到来自外围硬件（相对于中央处理器和内存）的异步信号，或来自软件的同步信号之后，处理器将会进行相应的 硬件／软件 处理。发出这样的信号称为进行中断请求（interrupt request，IRQ）。硬件中断导致处理器通过一个运行信息切换（context switch）来保存执行状态（以程序计数器和程序状态字等寄存器信息为主）；软件中断则通常作为CPU指令集中的一个指令，以可编程的方式直接指示这种运行信息切换，并将处理导向一段中断处理代码。中断在计算机多任务处理，尤其是即时系统中尤为有用。

**1、硬件中断**

由硬件发出或产生的中断称为硬中断，按硬中断事件的来源和实现手段可将中断划分为外中断和内中断：

（1）外中断：又称为中断或**异步中断**，是指 来自处理器以外的中断信号，包括时钟中断、键盘中断、外部设备中断等。外中断又分为可屏蔽中断和不可屏蔽中断，各个中断具有不同的优先级，表示事件的紧急程度，在处理高一级中断时，往往会部分或全部屏蔽低等级中断。

（2）内中断：又称为异常或**同步中断**（产生时必须考虑与处理器时钟同步），是指来自处理器内部的中断信号，通常是由于程序执行过程中，发现与当前指令关联的、不正常的或错误的事件。内中断可以细分为：

o 访管中断，由执行系统调用而引起的。

o 硬件故障中断，如电源失效、总线超时等。

o 程序性中断，如非法操作、地址越界、除数为0和浮点溢出等。

**2、软件中断：**是一条CPU指令，用以产生一个中断。由于软中断指令通常要运行一个切换CPU至内核态（Kernel Mode/Ring 0）的子例程，它常被用作实现系统调用（System call）。处理器通常含有一个内部中断屏蔽位，并允许通过软件来设定。一旦被设定，所有外部中断都将被系统忽略。这个屏蔽位的访问速度显然快于中断控制器上的中断屏蔽寄存器，因此可提供更快速地中断屏蔽控制。