**1.为什么要配置层次式存储器？**

在计算机执行时，几乎每一条指令都涉及对存储器的访问，因此要求对存储器的访问速度能跟得上处理机的运行速度。或者说，存储器的速度必须非常快，能与处理机的速度相匹配，否则会明显的影响到处理机的运行。此外还要求存储器具有非常大的容量，而且存储器的价格还应很便宜。对于这样十分严格的三个条件，目前是无法同时满足的。于是正在现代计算机系统中都无一例外的采用了多层结构的存储器系统。

**2.可采用哪几种方式将程序装入内存？它们分别适用于何种场合？**

（1）绝对装入方式

当计算机系统很小，且仅能运行单道程序时，完全有可能知道程序将驻留在内存的什么位置。此时可以采用绝对装入方式。用户程序编译后，将产生绝对地址的目标代码。

（2）可重定位装入方式

绝对装入方式只能将目标模块装入到内存中事先指定的位置，这只适用于单道程序环境。而在多道程序环境下，编译程序不可能预知经编译后所得到的目标模块应放在内存的何处。因此，对于用户程序编译所形成的若干个目标模块，它们的起始地址通常都是从0开始的，程序中的其他地址也都是想对于起始地址计算的。此时，不可能再采用绝对装入方式，而应采用可重定位装入方式，他可以根据内存的具体情况将装入模块装入到内存的适当位置。

（3）动态运行时的装入方式

可重定位装入方式可将装入模块装入到内存中任何允许的位置，故可用于多道程序环境，但该方式并不允许程序运行时在内存中移动位置。因为，程序在内存中的移动，意味着它的物理位置发生了变化，这时必须对程序和数据的地址（绝对地址）进行修改后方能运行。然而，实际情况是，在运行过程中他在内存中的位置肯能经常要改变。动态运行时的装入程序在把装入模块装入内存后，并不立即把装入模块中的逻辑地址转换为物理地址，而是把这种地址转换推迟到程序真正要执行时才进行。因此，装入内存后的所有地址都仍然是逻辑地址。

**3.何谓静态链接？静态链接时需要解决两个什么问题？**

在程序运行之前，先将各目标模块及他们所需的库函数链接成一个完整的装配模块，以后不再拆开，我们把这种事先进行链接的方式称为静态链接方式。

需要解决的两个问题是：①对相对地址进行修改。②变换外部调用符号。

**4.何谓装入时动态链接？装入时动态链接方式有何优点？**

这是指将用户源程序编译后所得到的一组目标模块，再装入内存时，采用边装入边链接的链接方式。即在装入一个目标模块时，若发生一个外部模块调用事件，将引起装入程序去找出对应的外部目标模块，并将它装入内存。

装入时动态链接方式有以下优点：

①便于修改和更新。采用动态链接方式，由于各目标模块是分开存放的，所以要修改或更新个目标模块是件非常容易的事。

②便于实现对目标模块的共享。再采用静态链接方式时，每个应用模块都必须含有其目标模块的拷贝，无法实现对目标模块的共享。但采用装入时动态链接方式时，OS就很容易将一个目标模块链接到几个应用模块上，实现多个应用程序对该模块的共享。

**5.何谓运行时动态链接？运行时动态链接有何优点？**

在许多情况下，应用程序在运行时，每次要运行的模块可能都是不相同的。但由于事先无法知道本次要运行哪些模块，故只能是将所有可能要运行到的模块全部都装入内存，并在装入时全部链接在一起，显然这是低效的，因为往往会有部分目标模块根本就不运行。近几年流行起来的运行时动态链接方式，是对上述装入时链接方式的一种改进。这种链接方式是，将对某些模块的链接推迟到程序执行时才进行。亦即，在执行过程中，当发现一个被调用模块尚未装入内存时，立刻由OS去找到该模块，将之装入内存，将其链接到调用者模块上，这样不仅能加快程序的装入过程，而且可节省大量的内存空间。

**6.在动态分区分配方式中，应如何将各空闲分区链接成空闲分区链？**

为了实现对空闲分区的分配和链接，在每个分区的起始部分设置一些用于控制分区分配的信息，，以及用于链接各分区所用的前向指针，在分区尾部则设置一后向指针。通过前、后向链接指针，可将所有的空闲分区链接成一个双向链。为了检索方便，在分区尾部重复设置状态位和分区大小表目。当分区被分配出去以后，把状态位由“0”改为“1”，此时，前、后向指针已无意义。

**7.为什么要引入动态重定位？如何实现？**

在动态运行时装入的方式中，作业装入内存后的所有地址仍然都是相对（逻辑）地址。而将相对地址转换为绝对（物理）地址的工作被推迟到程序指令要真正执行时进行。为使地址的转换不会影响到指令的执行速度，必须有硬件地址变换机构的支持，即须在系统中增设一个重定位寄存器，用它来存放数据在内存中的起始地址。程序在执行时，真正访问的内存地址是相对地址与重定位寄存器中的地址相加而形成的。地址变换过程是在程序执行期间，随着对每条指令或数据的访问自动进行的，故称为动态重定位。当系统对内存进行了“紧凑”，而使若干程序从内存的某处移至另一处时，不需对程序做任何修改，只要用该程序在内存中的新起始地址去置换原来的起始地址即可。

**8.什么是基于顺序搜索的动态分区分配算法？他可能分为哪几种？**

为了实现动态分区分配，通常是将系统中的空闲分区链接成一个链。所谓顺序搜索，是指依次搜索空闲分区链上的空闲分区，去寻找一个其大小能满足要求的分区。基于顺序搜索的动态分区分配算法分为四种：①首次适应算法②循环首次适应算法③最佳适应算法④最坏适应算法。

**9.在采用首次适应算法回收内存时，可能出现哪几种情况？应怎样处理这些情况？**

可能出现的情况有四种：

（1）回收区与插入点的前一个空闲分区相邻接，此时应将回收区与插入点的前一分区合并，不必再为回收分区分配新表项，而只需修改其前一分区的大小。

（2）回收分区与插入点的后一空闲分区相邻接，此时也可将两分区合并，形成新的空闲分区，但用回收区的首址作为新空闲区的首址，大小为两者之和。

（3）回收区同时与插入点的前、后两个分区邻接，此时将三个分区合并，使用第一个分区的表项和首址，取消其后一分区的表项，大小修改为三者之和。

（4）回收区没有与之相邻接的空闲分区，这时应为回收区单独建立一个新表项，填写回收区的首址和大小，并根据其首址插入到空闲链中的适当位置。

**10.什么是基于索引搜索的动态分区分派算法？它可能分为哪几种？**

基于顺序搜索的动态分区分配算法，比较适用于不太大的系统。当系统很大时，系统中的内存分区可能会很多，相应的空闲分区链就可能很长，这时采用顺序搜索分区方法可能会很慢。为了提高搜索空闲分区的速度，在大、中型系统中往往会采用基于索引搜索的动态分区分配算法。它分为三种：①快速适应算法②伙伴系统③哈希算法。

**11.令Buddyk(x)为大小为2k、地址为x的块的伙伴系统地址，试写出Buddyk(x)的通用表达式。**



**12.分区存储管理中常用哪些分配策略？比较他们的优缺点。**

（1）固定分区分配将整个用户空间划分为若干个固定大小的区域，在每个分区中只装入一道作业，这样就形成了最早的、也是最简单的一种可运行多道程序的分区式存储管理方式。

①分区大小相等。其缺点是缺乏灵活性，即当程序太小时，会造成内存空间的浪费。当程序太大时，一个分区又不足以装入该程序，致使该程序无法运行。尽管如此，对于利用一台计算机同时控制多个相同对象的场合，因为这些对象所需的内存空间大小往往相同，这种划分方式比较方便和实用，所以被广泛采用。②分区大小不等。增加了存储器分配的灵活性，可以根据用户的需要来划分。固定分区分配是最早出现的、可用于多道程序系统的存储管理方式，由于每一个分区的大小固定，必然会造成存储空间的浪费。

（2）动态分区分配。又称为可变分区分配，它是根据进程的实际需要，动态地为之分配内存空间。这种方式具有较大的灵活性，也提高了内存利用率，但是随着对内存的不断分配、释放操作会引起存储碎片的产生。

**13.为什么要引入对换？对换可分为哪几种类型？**

对换技术也称为交换技术，由于当时计算机的内存都非常小，为了使该系统能分时运行读个用户程序而引入了对换技术。根据每次对换时所兑换的数量，可将对换分为如下两类：

（1）整体对换。处理机中级调度实际上就是存储器的对换功能，其目的是用来解决内存紧张问题，并进一步提高内存的利用率和系统的吞吐量。由于在中级调度中对换是以整个进程为单位的，故称之为“进程对换”或“整体对换”。这种对换被广泛地应用于多道程序系统中，并作为处理机的中级调度。

（2）页面（分段）对换，如果对换是以进程的一个“页面”或“分段”为单位进行的，则分别称之为“页面对换”或“分段对换”，又统称为“部分对换”。这种对换方法的目的是为了支持虚拟存储系统。

**14.对文件区管理的目标和对对换空间管理的目标有何不同？**

（1）对文件区管理的主要目标。

文件区占用磁盘的大部分空间，用于存放各类文件。由于通常的文件都是较长时间地驻留在外存上，对它的访问频率是较低的。因此对文件区管理的主要目标是提高文件存储空间的利用率，然后才是提高对文件的访问速度。因此，对文件区的管理采取离散分配方式。

（2）对对换空间管理的主要目标。

对换空间只占用磁盘的小部分，用于存放从内存换出的进程。由于这些进程在对换区驻留的时间是短暂的，而对换操作的频率却极高，故对对换空间管理的主要目标是提高进程换入换出的速度，然后才是提高文件存储空间的利用率。为此，对对换区空间的管理采取连续分配方式，较少考虑外存中的碎片问题。

**15.为实现对换，系统应具备哪几个方面的内容？**

（1）对对换空间的管理。为了实现对对换区中的空闲盘块的管理，在系统中应配置相应的数据结构，用于记录外存对换区中空间盘块的使用情况。对换分区的分配和回收，分配算法有首次适应算法，循环首次适应算法，最佳适应算法和最坏适应算法。

（2）进程的换出。①选择被换出的进程。②进程换出过程。在进程换出时，应先申请对换空间，若申请成功，就启动磁盘，将该进程和数据传送到磁盘的对换区上，若传送过程未出现错误，便可回收该进程所占用的内存空间，并对该进程的进程控制块和内存分配表等数据结构做出相应的修改。若此时内存中还有可换出的进程，则继续执行换出过程，直到内存中再无阻塞进程为止。

（2）进程的换入。对对换进程将定期执行换入操作，它首先查看PCB集合中所有进程的状态，从中找出“就绪”状态但已换出的进程。当有许多这样的进程时，它将选择其中已换出到磁盘上时间最久的进程作为换入进程，为他申请内存，如果申请成功，可直接将进程从外存调入内存；如果失败，则需先将内存中的某些进程换出，腾出足够的内存空间后，再将进程调入。

**16.在以进程为单位进行对换时，每次是否都将整个进程换出？为什么？**

在以进程为单位进行对换时，并非每次都将整个进程换出。①从结构上讲，进程由程序段、数据段和PCB组成，其中进程控制块总有部分或全部常驻内存，不被换出。②程序段和数据段可能正在被若干进程共享，此时它们也不能换出。因为在进程的换出中，只能选择换出非共享的数据段和程序段，而对于哪些共享的程序和数据段，只要还有进程需要他，就不能被换出。

**17.基于离散分配时所用的基本单位不同，可将离散分配分为哪几种？**

基于离散分配时所用的基本单位不同，可将离散分配分为：

（1）分页存储管理方式。在该方式中，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”。典型的页面大小为1KB。相应的。也将内存空间分为若干个物理块或页框，页和块的大小相同。这样可将用户程序的任一页放入任一物理块中，实现了离散分配。

（2）分段存储管理方式。这是为了满足用户要求而形成的一种存储管理方式。它把用户程序的地址空间分为若干个大小不同的段，每段可定义一组相对完整的信息。在存储器分配时，以段为单位，这些段在内存中可以不相邻接，所以也同样实现了离散分配。

（3）段页式存储管理方式。这是分页和分段两种存储管理方式结合的产物。他同时具有两者的优点，是目前应用较广泛的一种存储管理方式。

**18.什么是页面？什么是物理块？页面的大小应如何确定？**

（1）页面。分页存储管理将进程的逻辑空间分成若干个页，并为各页加以编号，从0开始，如第0页、第1页等。

（2）物理块。相应的，也把内存的物理地址空间分成若干个块，同样也为它们加以编号，如0#块、1#块等等。

（1）页面大小。在分页系统中，若选择过小的页面大小，虽然一方面可以减小内存的碎片，起到减少内存碎片总空间的作用，有利于内存利用率的提高，但另一方面却会造成每个进程占用较多的页面，从而导致进程的页表过长，占用大量内存。此外，还会降低页面的换进换出的效率。然而，如果选择的页面过大，虽然可以减少页表的长度，提高页面换进换出的效率，但却又会使页内碎片增大。因此，页面的大小应选择适中，且页面大小应是2的幂，通常为1KB~8KB。

**19.什么是页表？页表的作用是什么？**

在分页系统中，允许将进程的各个页离散的存储在内存的任一物理块中，为保证进程仍然能够正确地运行，即能在内存中找到每个页面所对应的物理块，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（0~N）,依次在页表中有一页表项，其中记录了相应页表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，即了找到每页在内存中的物理块号。可见，也表的作用是实现从页号到物理块号的地址映射。

**20**.**为实现分页存储管理，需要哪些硬件支持？**

需要页表寄存器、物理地址寄存器和联想寄存器(快表)。页表寄存器和物理地址寄存器是地址变换机构所需的基本寄存器，联想寄存器是优化了地址转换过程后需要添加的一个寄存器。

**21.在分页系统中是如何实现地址变换的？**

页表功能是由一组专门的寄存器来实现的。一个页表项用一个寄存器。由于寄存器具有较高的访问速度，因而有利于提高地址变换的速度，但由于寄存器成本较高，页表又可能很大，因此，页表大多驻留在内存中，在系统中只设置一个页表寄存器，在其中存放页表在内存的始址和页表的长度。当进程要访问某个逻辑地址中的数据时，分页地址变换机构会自动将有效地址分为页号和页内地址两部分，再以页号为索引去检索页表。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，则表示本次所访问的地址已超越进程的地址空间。于是，这一错误将被系统发现，并产生已地址越界中断。若未出现越界中断，则将页表始址与页号和页表项长度的乘积相加，便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地址字段中。这样便完成了从逻辑地址到物理地址的变换。

**22.具有快表时是如何实现地址变换的？**

为了提高地址变换的速度，可 在地址变换机构中增设一个具有并行查询能力的特殊高速缓冲寄存器，又称为“联想寄存器”，或称为“快表”，在IBM系统中又取名为TLB，用以存放当前访问的那些页表项。此时的地址变换过程是：在CPU给出有效地址后，由地址变换机构自动的将页号P送入高速缓冲寄存器，并将此页号与高速缓冲中的所有页号进行比较，若其中有与此相匹配的页号，便表示索要访问的页表项在快表中。于是，可直接从快表中读出该页所对应的物理块号，并送到物理地址寄存器中。如在快表中未找到对应的页表项，则还须访问内存中的页表，找到后，把从页表中读出的物理块号送往地址寄存器；同时，再将此页表项存入快表的一个寄存器的单元中。亦即，重新修改快表。但如果联想寄存器已满，则OS必须找到一个老的且已被认为是不再需要的页表项，将它换出。

**23.较详细的说明引入分段存储管理是为了满足用户哪几方面的需求。**

（1）方便编程。通常，用户把自己的作业按照逻辑关系划分为若干段，每个段都从0开始编址，并有自己的名字和长度。因此，程序员们都迫切的需要访问的逻辑地址是由段名和段内偏移量决定的，这不仅可以方便程序员编程，也可使程序方非常直观，更具有可读性。

（2）信息共享。在实现对程序和数据的共享时，是以信息的逻辑单位为基础的。比如，为了共享某个过程、函数或文件。分页系统中的“页”只是存放信息的物理单位，并无完整的逻辑意义，这样，一个可被共享的过程往往可能需要占用数十个页面，这为实现共享增加了困难。段可以是信息的逻辑单位，因此，我们可以为该被共享过程建立一个独立的段，这就极大地简化了共享的实现。

（3）信息保护。信息保护同样是以信息的逻辑单位为基础的，而且经常是以一个过程、函数或文件为基本单位进行保护的。

（4）动态增长。在实际应用中，往往存在着一些段，尤其是数据段，在它们的使用过程中，由于数据量的不断增加，而使数据段动态增长，相应的他所需要的存储空间也会动态增加。然而，对于数据段究竟会增长到多大，事先又很难确切的知道。对此，很难采取预先多分配的方法进行解决。分段存储管理方式能很好的解决这一问题。

**24.在具有快表的段页式存储管理方式中，如何实现地址变换？**

在段页式系统中，为了便于实现地址变换，须配置一个段表寄存器，其中存放段表始址和段长TL。进行地址变换时，首先利用段号S，将它与段长TL进行比较。若S<TL，表示未越界，于是利用段表始址和段号来求出该段所对应的段表项在段表中的位置，从中得到该段的页表地址，并利用逻辑地址中的段内页号P来获得对应的页表项地址，从中读出该页所在的物理块号b，再利用块号b和页内地址来构成物理地址。为了提高执行速度，在地址变换机构中增设一个高速缓冲寄存器，每次访问它时，都须同时利用段号和页号去检索高速缓冲，若找到匹配的表项，便可从中得到相应页的物理块号，用来与页内地址一起形成物理地址；若未找到匹配表项，则仍需第三次访问内存。

**25.为什么说分段系统比分页系统更易于实现信息的共享和保护？**

信息的共享和保护都是以信息的逻辑单位为基础的，所以，经常是以一个段为基本单位进行保护和共享的。在分段系统中，由于是以段为基本单位的，不管该段有多大，我们都只需要为该段设置一个段表项，因此使实现共享变得非常容易。但是在分页系统中，可能一个信息的逻辑单位有很多页，且一页中可能含有不同程序段的数据或程序。

**26.分页和分段存储管理有何区别？**

（1）页是信息的物理单位，采用分页存储管理方式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，分页仅仅只是系统管理上的需要，完全是系统的行为，多用户是不可见的。而段是信息的逻辑单位。它通常包含的是一组意义相对完整的信息。分段的目的主要在于能更好地满足用户的需要。

（2）页的大小固定而且由系统决定，在采用分页存储管理系统中，在硬件结构上，就是把用户程序的逻辑地址划分为页号和页内地址两部分，是直接由硬件实现的，因而在每个系统中只能有一种大小的页面。而段的大小不固定，通常由编译程序根据信息的性质来划分。（3）分页用户程序地址空间是一维的，分页完全是系统的行为，故在分页系统中，用户程序的地址是属于单一的线性地址空间，程序员只需利用一个记忆符即可表示一个地址。而分段是用户的行为，故在分段系统中，用户程序的地址空间是二维的，程序员在标识一个地址时，既需给出段名，又需给出段内地址。

**27.试全面比较连续分配和离散分配方式。**

连续分配方式不需要额外的硬件支持，且实现算法相对简单。但是在很多情况下会造成内存利用率低，系统吞吐量小和CPU利用率低等情况，虽然可以通过紧凑等方式有所调节，但是紧凑也会造成很大的系统开销。离散分配方式需要额外的硬件支持，且实现的算法相对比较复杂，但是出于用户或操作系统的角度，离散分配方式在系统性能上或实现功能上明显比连续分配更灵活。比如信息的保护和共享等等方面，离散比连续更加容易实现。