#### 典型回答

分段就是支持用户视图的内存管理方案。逻辑地址空间是由一组段构成。每个段都有名称和长度。地址指定了段名称和段内偏移。因此用户通过两个量来指定地址：段名称和段偏移。

实现分页需要操作系统和计算机硬件的协作。

实现分页的基本方法涉及将物理内存分为固定大小的块，称为页帧；而将逻辑内存也分为同样大小的块，称为页面。逻辑地址空间现在完全独立于物理地址空间。

分段允许进程的物理地址空间是非连续的。分页是提供这种优势的另一种内存管理方案。然鹅，分页避免了外部碎片和紧缩，而分段不可以。分页也避免了将不同大小的内存块匹配到交换空间的麻烦问题。

#### 知识延伸

首先先来熟悉一下逻辑地址空间和物理地址空间。

CPU 生成的地址通常称为逻辑地址，而内存单元看到的地址（即加载到内存地址寄存器）通常称为物理地址。

编译时和加载时的地址绑定方法生成相同的逻辑地址和物理地址。然而，执行时的地址绑定方案生成不同的逻辑地址和物理地址。在这种情况下，我们通常称逻辑地址为虚拟地址。通常，我们并不区分逻辑地址和虚拟地址。由程序所生成的所有逻辑地址的集合称为逻辑地址空间，这些逻辑地址对应的所有的物理地址的集合称为物理地址空间。因此，对于执行时地址绑定方案，逻辑地址空间和物理地址空间是不同的。

从虚拟地址到物理地址的运行时映射是由内存管理单元（MMU）的硬件设备来完成的。用户程序不会看到真实的物理地址，用户程序处理逻辑地址；内存映射硬件将逻辑地址转变为物理地址。

**连续内存分配**

内存通常分为两个区域：一个用于驻留操作系统，一个用户用户进程。最简单的内存分配方法之一，就是将内存分为多个固定大小的分区。每个分区可以只包含一个进程。

对于可变分区方案，操作系统有一个表，用于记录哪些内存可用和哪些内存已用。开始，所有内存都可用于用户进程，因此可以作为一大块的可用内存，称为孔。最后，正如将会看到的，内存有一个集合，以包含各种大小的孔。

综上所述，可用的内存块为分散在内存里的不同大小的孔的集合。当新进程需要内存时，系统为该进程查找足够大的孔。如果孔太大，那么就分为两块：一块分配给新进程，另一块还回到孔集合。

从一组可用孔中选择一个空闲孔的最为常用方法包括：首次适应、最优适应及最差适应。

首次适应和最优适应都有外部碎片的问题，外部碎片的一个可能解决方案是紧缩。它的目的是移动内存内容，以便将所有空闲空间合并成一整块。然而，紧缩并非总是可能的。如果重定位是静态的，并且在汇编时或加载时进行的，那么就不能紧缩。只有重定位是动态的，并且在运行时进行的，才可用紧缩。

外部碎片化问题的另一个可能的解决方案是：允许进程的逻辑地址空间是不连续的；这样，只要有物理内存可用，就允许为进程分配内存。有两种互补的技术可以实现这个解决方案：分段和分页。

##### 分段

分段的每个段都有名称和长度，地址制定了段名称和段内偏移。

虽然用户现在能够通过二维地址来引用程序内的对象，但是实际物理内存仍然是一维的字节序列。因此，我们应该定义一个实现方式，以便映射用户定义的二维地址到一维物理地址。这个地址是通过段表来实现的。段表的每个条目都有段基地址和段界限。段基地址包含该段在内存中的开始物理地址，而段界限指定该段的长度。

段表的使用如下图所示：



每个逻辑地址由两部分组成：段号 s 和段偏移 d。段号用作段表的索引，逻辑地址的偏移 d 应位于 0 和段界限之间。如果不是这样，那么会陷入操作系统中（逻辑地址试图访问段的外面）。如果偏移 d 合法，那么就与基地址相加而得到所需字节的物理内存地址。因此，段表实际上基址寄存器值和界限寄存器的值的对的数组。

作为一个例子，可以看下图：



**分页**

实现分页的基本方法涉及将物理内存分为固定大小的块，称为页帧；而将逻辑内存也分为同样大小的块，称为页面。

分页的硬件支持如下图所示：



由 CPU 生成的每个地址分为两个部分：页码（p）和页偏移（d）。页码作为页表的索引。页表包含每页所在物理内存的基地址，这个基地址与页偏移的组合就形成了物理内存地址，可发送到物理单元。

内存分页模型如下图：



页大小是由硬件来决定的。页的大小为 2 的幂；根据计算机体系结构的不同，页大小可从 512 字节到 1 GB 不等。

采用分页方案不会产生外部碎片：每个空闲帧都可以分配给需要它的进程。不过，分页有内部碎片。注意，分配是以帧为单位进行的。如果进程所要求的内存并不是页的整数倍，那么最后一个页可能用不完。

现在，页大小通常为 4～8 KB。

分页的一个重要方面是，程序员视图的内存和实际的物理内存的清楚分离。程序员将内存作为一整块来处理，而且它只包含这一个程序。事实上，一个用户程序与其他程序一起，分散在物理内存上。程序员视图的内存和实际的物理内存的不同是通过地址转换硬件来协调的。逻辑地址转变成物理地址。这种映射，程序员是不知道的，它是由操作系统控制的。注意，根据定义，用户进程不能访问不属于它的内存。

由于操作系统管理物理内存，它应知道物理内存的分配细节：哪些帧已分配，哪些帧空着，总共有多少帧，等等。这些信息通常保存在称为帧表的数据结构中。在帧表中，每个条目对应着一个帧，以表示该帧是空闲还是已占用；如果占用，是被哪个进程所占用。

每个操作系统都有自己保存页表的方法，有的为每个进程分配一个页面。页表的指针，与其他寄存器的值（如指令寄存器），一起存入进程控制块。

组织页表有分层分页、哈希页表和倒置页表。

#### 小结

多道程序操作系统的内存管理算法包括从简单的单用户系统方案，到分段和分页的方案。决定特定系统采用方案的最重要因素是所提供的硬件支持。CPU 所产生的每个内存地址都应先进行合法性检查，然后才可能映射到物理地址。检查不能通过软件来实现，因此受到可用硬件的限制。

不同的内存管理算法（连续分配、分页、分段及分段分页组合）在许多方面都不同。当比较不同内存管理策略时，需要考虑如下几点：

* 硬件支持

对单分区和多分区方案，只需要一个基地址寄存器或一个基地址 - 界限地址寄存器对就足够了；而对于分页和分段，需要映射表来定义地址映射。

* 性能

随着内存管理算法变得更为复杂，逻辑地址到物理地址的映射时间也会更长。对于简单的系统，只需对逻辑地址进行比较和加法操作。如果映射表通过快速寄存器实现，那么分页和分段操作也会很快。然而，如果映射表位于内存中，那么用户内存访问就大受影响。TLB 可使性能影响减小到可接受的水平。

* 碎片

如果多道程序的程度更高，那么多道程序系统的执行通常会更有效。给定一组进程，通过加载更多进程到内存，可以增加多道程序的程度。为了完成这个任务，应降低内存浪费或碎片。采用固定大小分配单元（如单个分区和分页）的系统会有内部碎片的问题。采用可变大小分配单元（如多个分区和分段）的系统会有外部碎片问题。

* 重定位

外部碎片问题的解决方案之一是紧缩。紧缩就是在内存中移动程序，且不会影响程序的运行。这种方案要求，执行时的逻辑地址可动态的进行重定位。如果只能进行加载时的地址重定位，那么就不能采用紧缩。

* 保护

如果提供了分页或分段，那么用户程序的不同区域可以声明为只可执行的、只读的或可读可写的。对于代码和数据的共享，这种限制是必要的；对于程序设计的常见错误，能提供运行时的简单检查。