1. 段式存储管理
2. 概述

一个作业是由若干个段组成的，因而用户希望将作业按照逻辑顺序划分为若干个段，要访问的逻辑地址是由段号+段内偏移量决定的，每个段都从0开始编址。段是信息的逻辑单位，因而段式存储可以实现共享，因而可以实现动态链接，源程序经过编译后形成的目标程序还需要进行链接后形成可执行程序才能够执行。在装入时进行的链接为静态链接，动态链接是指在作业运行过程中，需要调用某段时，才将该段调入内存中进行链接，可以看到，动态链接是以段为单位的。

1. 段的划分

根据程序的模块结构，将作业地址空间划分为大小不同的段，每个程序段都有段名和段号，段号从0开始，每一段也从0开始编址，段内地址是连续的。每个程序的段大致分为主程序段，子程序段，库函数段，数据段等，同时在物理内存中，也会分成与这些段大小相同的块。作业在装入时是一次性装入的，如果不是则为请求分段式的管理。

1. 分段式管理的优势

可以对程序中不同的段分别进行编写和编译，可以针对不同类型的段采取不同的保护，可以以段为单位进行共享，包括动态链接进行代码共享。

1. 分段式管理的数据结构
2. 进程段表

描述组成进程地址空间的各段，可以是指向系统段表中表项的索引，每段都有段基址。

1. 系统段表

描述系统所有占用的段。

1. 空闲段表

描述了内存中所有空闲段，可以结合到系统段表中。

1. 地址变换

系统中设置了段表基址寄存器和段表长度寄存器，在进行地址变换时，系统将逻辑地址中的段号S与段表长度STL进行比较，如果没有越界，则根据段表的起始地址和该段的段号，计算出该段对应段表项的位置，读出该段在内存中的起始地址，然后再检查逻辑地址中该段的段内位移是否超过该段的段长，如果没有越界，则将该段的基址和段内地址相加，得到要访问的物理内存地址。

同样的，段表在内存中时，每访问内存中的一个数据都至少访问内存两次，因此再增设一个联想寄存器，保存最近常用的段表项，由于段比页大，因此段表项的数目比页表少，所需要的联想寄存器也会小一些。

1. 硬件支持

段表基址寄存器保存正在运行进程的段表的基址，段表长度寄存器用于保存正在运行进程的段表的长度，以及联想寄存器。

1. 优缺点

优点：

没有内碎片，外碎片可以通过内存紧缩来消除；

便于改变进程占用空间的大小；

便于实现共享和保护，允许多个进程共享多个段；

缺点：

作业需要全部装入内存；

不能实现存储扩容；

1. 分页和分段的比较
2. 分页是由于系统管理需要，分段是由于用户应用的需要，因此一条指令可能会跨越两个页的交界处，而不会跨越两个段的交界处；
3. 页的大小是系统固定的，而段的大小则不固定；
4. 分页的逻辑地址是一维的，各个模块在链接时必须在同一个地址空间中，分段是二维的，各个模块在链接时可以把每个段组织为一个地址空间；
5. 段比页大，段表比页表小，缩短查找时间；
6. 分段可以实现内存共享，分页不行；
7. 都不能实现存储扩容；
8. 段页式存储
9. 概述

结合分页和分段，既有分段便于实现共享，易于保护，动态链接等优点，又有分页系统能够解决内存的外碎片问题，这就是段页式存储管理。将用户程序分为多个段，每个段再划分为多个页，每个段赋予一个段名，也就是将用户程序按照段来划分，将物理内存按照页来划分，以页为单位进行分配，它的地址结构是由段号，段内页号，页内地址三者组成的。

1. 地址管理

同时配置段表和页表，由于允许将一个段中的页进行不连续分配，因此段表的内容有所变化，它不再是段内起始地址和段长，而是页表起始地址和页表长度。通过段表来查找页表，然后通过页表来得到真正内存物理地址。

1. 地址变换

段表：记录每一段的页表起始地址和页表长度；

页表：记录每一个段所对应的逻辑页号和内存页号的对应关系，每个段有一个页表，一个程序可能有多个页表；

空闲内存页表：同分页式存储；

物理内存分配：同分页式存储；

1. 硬件支持

段表基址寄存器，段表长度寄存器，在进行地址变换时，首先将段号与段长进行比较，如果没有越界，则利用段表起始地址和段号求出该段对应的段表项，得到该段的页表起始地址，并利用逻辑地址中的段内页号来获得对应的页表项位置，读出该页所在的物理页号，再通过物理页号和逻辑地址中的页内地址构成最终的物理页地址。

段页式存储管理中，为了获得一条指令或数据，至少需要三次访问内存，第一次访问段表，第二次访问页表，第三次才是访问数据。同样的，设置一个连向存储器，存储段号和页号，对应着页的物理页号。