中间没有空隙，因此修改一个过程的大小会影响其他无关的过程的起始地址，而这又需要修改调用了这些被移动过的过程的所有过程，以使它们的访问指向这些过程的新地址。在一个有数百个过程的程序中，这个操作的开销可能是相当大的。

分段也有助于在几个进程之间共享过程和数据。这方面一个常见的例子就是共享库（shared library）。运行高级窗口系统的现代工作站经常要把非常大的图形库编译进几乎所有的程序中。在分段系统中，可以把图形库放到一个单独的段中由各个进程共享，从而不再需要在每个进程的地址空间中都保存一份。虽然在纯的分页系统中也可以有共享库，但是它要复杂得多，并且这些系统实际上是通过模拟分段来实现的。

因为每个段是一个为程序员所知道的逻辑实体，比如一个过程、一个数组或一个堆栈，故不同的段可以有不同种类的保护。一个过程段可以被指明为只允许执行，从而禁止对它的读出和写入；一个浮点数组可以被指明为允许读写但不允许执行，任何试图向这个段内的跳转都将被截获。这样的保护有助于找到编程错误。

读者应该试着理解为什么保护在分段存储中有意义，而在一维的分页存储中则没有。在分段存储中用户知道每个段中包含了什么。例如，一般来说，一个段中不会既包含一个过程又包含一个堆栈，而是只会包含其中的一个。正是因为每个段只包含了一种类型的对象，所以这个段就可以设置针对这种特定类型的合适的保护。图3-33对分段和分页进行了比较。

考查点	分页	分段
需要程序员了解正在使用这种技术吗?	否	是
存在多少线性地址空间?	1	许多
整个地址空间可以超出物理存储器的大小吗?	是	是
过程和数据可以被区分并分别被保护吗?	否	是
其大小浮动的表可以很容易提供吗?	否	是
用户间过程的共享方便吗?	否	是
为什么发明这种技术?	为了得到大的线性地址空间而不必购买更大的物理存储器	为了使程序和数据可以被划分为逻辑上独立的地址空间并且有助于共享和保护

图3-33 分页与分段的比较

页面的内容在某种程度上是随机的，程序员甚至察觉不到分页的事实。尽管在页表的每个表项中放入几位就可以说明其对应页面的访问权限，然而为了利用这一点，程序员必须跟踪他的地址空间中页面的界限。当初正是为了避免这一类管理工作，人们才发明了分页系统。在分段系统中，由于用户会认为所有的段都一直在内存中，也就是说他可以当作所有这些段都在内存中那样去访问，他可以分别保护各个段，所以不需要关心覆盖它们的管理工作。

3.7.1 纯分段的实现

分段和分页的实现本质上是不同的：页面是定长的而段不是。图3-34a所示的物理内存在初始时包含了5个段。现在让我们考虑当段1被淘汰后，比它小的段7放进它的位置时会发生什么样的情况。这时的内存配置如图3-34b所示，在段7与段2之间是一个未用区域，即一个空闲区。随后段4被段5代替，如图3-34c所示；段3被段6代替，如图3-34d所示。在系统运行一段时间后内存被划分为许多块，一些块包