速度变慢，因为必须将一个回收的段从进程链表中删除并插入空闲区链表。

如果进程和空闲区使用不同的链表，则可以按照大小对空闲区链表排序，以便提高最佳适配算法的速度。在使用最佳适配算法搜索由小到大排列的空闲区链表时，只要找到一个合适的空闲区，则这个空闲区就是能容纳这个作业的最小的空闲区，因此是最佳适配。因为空闲区链表以单链表形式组织，所以不需要进一步搜索。空闲区链表按大小排序时，首次适配算法与最佳适配算法一样快，而下次适配算法在这里则毫无意义。

在与进程段分离的单独链表中保存空闲区时，可以做一个小小的优化。不必像图3-6c那样用单独的数据结构存放空闲区链表，而可以利用空闲区存储这些信息。每个空闲区的第一个字可以是空闲区大小，第二个字指向下一个空闲区。于是就不再需要图3-6c中所示的那些三个字加一位（P/H）的链表结点了。

另一种分配算法称为快速适配（quick fit）算法，它为那些常用大小的空闲区维护单独的链表。例如，有一个n项的表，该表的第一项是指向大小为4KB的空闲区链表表头的指针，第二项是指向大小为8KB的空闲区链表表头的指针，第三项是指向大小为12KB的空闲区链表表头的指针，以此类推。像21KB这样的空闲区既可以放在20KB的链表中，也可以放在一个专门存放大小比较特别的空闲区的链表中。

快速适配算法寻找一个指定大小的空闲区是十分快速的，但它和所有将空闲区按大小排序的方案一样都有一个共同的缺点，即在一个进程终止或被换出时，寻找它的相邻块，查看是否可以合并的过程是非常费时的。如果不进行合并，内存将会很快分裂出大量的进程无法利用的小空闲区。

3.3 虚拟内存

尽管基址寄存器和界限寄存器可以用于创建地址空间的抽象，还有另一个问题需要解决：管理软件的膨胀（bloatware）。虽然存储器容量增长快速，但是软件大小的增长更快。在20世纪80年代，许多大学用一台4MB的VAX计算机运行分时操作的系统，供十几个用户（已经或多或少足够满足需要了）同时运行。现在微软公司为单用户Vista系统推荐至少512MB内存，并且只能运行简单的应用程序，如果运行复杂应用程序则要1GB内存。而多媒体的潮流则进一步推动了对内存的需求。

这一发展的结果是，需要运行的程序往往大到内存无法容纳，而且必然需要系统能够支持多个程序同时运行，即使内存可以满足其中单独一个程序的需要，但总体来看，它们仍然超出了内存大小。交换技术（swapping）并不是一个有吸引力的解决方案，因为一个典型的SATA磁盘的峰值传输率最高达到100MB/s，这意味着至少需要10秒才能换出一个1GB的程序，并需要另一个10秒才能再将一个1GB的程序换入。

程序大于内存的问题早在计算时代开始就产生了，虽然只是有限的应用领域，像科学和工程计算（模拟宇宙的创建或模拟新型航空器都会花费大量内存）。在20世纪60年代所采取的解决方法是：把程序分割成许多片段，称为*覆盖*（overlay）。程序开始执行时，将覆盖管理模块装入内存，该管理模块立即装入并运行覆盖0。执行完成后，覆盖0通知管理模块装入覆盖1，或者占用覆盖0的上方位置（如果有空间），或者占用覆盖0（如果没有空间）。一些覆盖系统非常复杂，允许多个覆盖块同时在内存中。覆盖块存放在磁盘上，在需要时由操作系统动态地换入换出。

虽然由系统完成实际的覆盖块换入换出操作，但是程序员必须把程序分割成多个片段。把一个大程序分割成小的、模块化的片段是非常费时和枯燥的，并且易于出错。很少程序员擅长使用覆盖技术。因此，没过多久就有人找到一个办法，把全部工作都交给计算机去做。

采用的这个方法（Fotheringham，1961）称为*虚拟内存*（virtual memory）。虚拟内存的基本思想是：每个程序拥有自己的地址空间，这个空间被分割成多个块，每一块称作一*页*或*页面*（page）。每一页有连续的地址范围。这些页被映射到物理内存，但并不是所有的页都必须在内存中才能运行程序。当程序引用到一部分在物理内存中的地址空间时，由硬件立刻执行必要的映射。当程序引用到一部分不在物理内存中的地址空间时，由操作系统负责将缺失的部分装入物理内存并重新执行失败的指令。

从某个角度来讲，虚拟内存是对基址寄存器和界限寄存器的一种综合。8088为正文和数据分离出专门的基址寄存器（但不包括界限寄存器）。而虚拟内存使得整个地址空间可以用相对较小的单元映射到物理内存，而不是为正文段和数据段分别进行重定位。下面会介绍虚拟内存是如何实现的。

虚拟内存很适合在多道程序设计系统中使用，许多程序的片段同时保存在内存中。当一个程序等待