起来得到一个16页面的块（i）。

Linux用伙伴算法管理内存，同时有一些附加特性。它有个数组，其中的第一个元素是大小为1个单位的内存块列表的头部，第二个元素是大小为2个单位的内存块列表的头部，下一个是大小为4个单位的内存块列表的头部，以此类推。通过这种方法，任何2的幂次大小的块都可以快速找到。

这个算法导致了大量的内部碎片，因为如果想要65页面的块，必须要请求并且得到一个128页面的块。

为了缓解这个问题，Linux有另一个内存分配器，slab分配器。它使用伙伴算法获得内存块，但是之后从其中切出slab（更小的单元）并且分别进行管理。

因为内核频繁地创建和撤销一定类型的对象（如task_struct），它使用了对象缓存。这些缓存由指向一个或多个slab的指针组成，而slab可以存储大量相同类型的对象。每个slab要么是满的，要么是部分满的，要么是空的。

例如，当内核需要分配一个新的进程描述符（一个新的task_struct）的时候，它在task结构的对象缓存中寻找，首先试图找一个部分满的slab并且在那里分配一个新的task_struct对象。如果没有这样的slab可用，就在空闲slab列表中查找。最后，如果必要，它会分配一个新的slab，把新的task结构放在那里，同时把该slab连接到task结构对象缓存中。在内核地址空间分配连续的内存区域的kmalloc内核服务，实际上就是建立在slab和对象缓存接口之上的。

第三个内存分配器vmalloc也是可用的，并且用于那些仅仅需要虚拟地址空间连续的请求。实际上，这一点对于大部分内存分配是成立的。一个例外是设备，它位于内存总线和内存管理单元的另一端，因此并不理解虚拟地址。然而，vmalloc的使用导致一些性能的损失，主要用于分配大量连续虚拟地址空间，例如动态插入内核模块。所有这些内存分配器都是继承自System V中的那些分配器。

3. 虚拟地址空间表示

虚拟地址空间被分割成同构连续页面对齐的区域。也就是说，每个区域由一系列连续的具有相同保护和分页属性的页面组成。代码段和映射文件就是区（area）的例子（见图10-15）。在虚拟地址空间的区之间可以有空隙。所有对这些空隙的引用都会导致一个严重的页面故障。页大小是确定的，例如Pentium是4KB而Alpha是8KB。Pentium支持4MB的页框，Linux可以支持4MB的大页框。而且，在PAE（物理地址扩展）模式下，2MB的页大小是支持的。在一些32位机器上常用PAE来增加进程地址空间，使之超过4GB。

在内核中，每个区是用vm_area_struct项来描述的。一个进程的所有vm_area_struct用一个链表链接在一起，并且按照虚拟地址排序以便可以找到所有的页面。当这个链表太长时（多于32项），就创建一个树来加速搜索。vm_area_struct项列出了该区的属性。这些属性包括：保护模式（如，只读或者可读可写）、是否固定在内存中（不可换出）、朝向哪个方向生长（数据段向上长，栈段向下长）。

vm_area_struct也记录该区是私有的还是跟一个或多个其他进程共享的。fork之后，Linux为子进程复制一份区链表，但是让父子进程指向相同的页表。区被标记为可读可写，但是页面却被标记为只读。如果任何一个进程试图写页面，就会产生一个保护故障，此时内核发现该内存区逻辑上是可写的，但是页面却不是，因此它把该页面的一个副本给当前进程同时标记为可读可写。这个机制就说明了写时复制是如何实现的。

vm_area_struct也记录该区是否在磁盘上有备份存储，如果有，在什么地方。代码段把可执行二进制文件作为备份存储，内存映射文件把磁盘文件作为备份存储。其他区，如栈，直到它们不得不被换出，否则没有备份存储被分配。

一个顶层内存描述符mm_struct收集属于一个地址空间的所有虚拟内存区相关的信息，还有关于不同段（代码，数据，栈）和用户共享地址空间的信息等。一个地址空间的所有vm_area_struct元素可以通过内存描述符用两种方式访问。首先，它们是按照虚拟地址顺序组织在链表中的。这种方式的有用之处是：当所有的虚拟地址区需要被访问时，或者当内核查找分配一个指定大小的虚拟内存区域时。此外，vm_area_struct项目被组织成二叉“红黑”树（一种为了快速查找而优化的数据结构）。这种方法用于访问一个指定的虚拟内存地址。为了能够用这两种方法访问进程地址空间的元素，Linux为每个进程使用