所请求块的空闲块。分配器执行这种搜索的方式是由*放置策略*(placement policy)确定的。一些常见的策略是*首次适配*(first fit)、*下一次适配*(next fit)和*最佳适配*(best fit)。

*首次适配*从头开始搜索空闲链表，选择第一个合适的空闲块。*下一次适配*和首次适配很相似，只不过不是从链表的起始处开始每次搜索，而是从上一次查询结束的地方开始。*最佳适配*检查每个空闲块，选择适合所需请求大小的最小空闲块。

首次适配的优点是它趋向于将大的空闲块保留在链表的后面。缺点是它趋向于在靠近链表起始处留下小空闲块的“碎片”，这就增加了对较大块的搜索时间。下一次适配是由Donald Knuth作为首次适配的一种代替品最早提出的，源于这样一个想法：如果我们上一次在某个空闲块里已经发现了一个匹配，那么很可能下一次我们也能在这个剩余块中发现匹配。下一次适配比首次适配运行起来明显要快一些，尤其是当链表的前面布满了许多小的碎片时。然而，一些研究表明，下一次适配的内存利用率要比首次适配低得多。研究还表明最佳适配比首次适配和下一次适配的内存利用率都要高一些。然而，在简单空闲链表组织结构中，比如隐式空闲链表中，使用最佳适配的缺点是它要求对堆进行彻底的搜索。在后面，我们将看到更加精细复杂的分离式空闲链表组织，它接近于最佳适配策略，不需要进行彻底的堆搜索。

9.9.8　分割空闲块

一旦分配器找到一个匹配的空闲块，它就必须做另一个策略决定，那就是分配这个空闲块中多少空间。一个选择是用整个空闲块。虽然这种方式简单而快捷，但是主要的缺点就是它会造成内部碎片。如果放置策略趋向于产生好的匹配，那么额外的内部碎片也是可以接受的。

然而，如果匹配不太好，那么分配器通常会选择将这个空闲块分割为两部分。第一部分变成分配块，而剩下的变成一个新的空闲块。图9-37展示了分配器如何分割图9-36中8个字的空闲块，来满足一个应用的对堆内存3个字的请求。

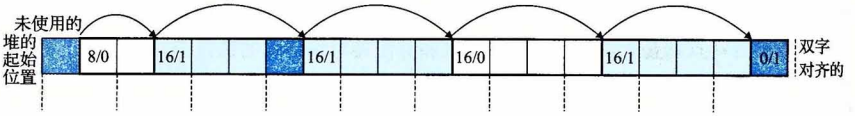

图9-37　分割一个空闲块，以满足一个3个字的分配请求。阴影部分是已分配块。没有阴影的部分是空闲块。头部标记为(大小(字节)/已分配位)

9.9.9　获取额外的堆内存

如果分配器不能为请求块找到合适的空闲块将发生什么呢？一个选择是通过合并那些在内存中物理上相邻的空闲块来创建一些更大的空闲块(在下一节中描述)。然而，如果这样还是不能生成一个足够大的块，或者如果空闲块已经最大程度地合并了，那么分配器就会通过调用sbrk函数，向内核请求额外的堆内存。分配器将额外的内存转化成一个大的空闲块，将这个块插入到空闲链表中，然后将被请求的块放置在这个新的空闲块中。

9.9.10　合并空闲块

当分配器释放一个已分配块时，可能有其他空闲块与这个新释放的空闲块相邻。这些邻接的空闲块可能引起一种现象，叫做*假碎片*(fault fragmentation)，就是有许多可用的