空闲块被切割成为小的、无法使用的空闲块。比如，图 9-38 展示了释放图 9-37 中分配的块后得到的结果。结果是两个相邻的空闲块，每一个的有效载荷都为 3 个字。因此，接下来一个对 4 字有效载荷的请求就会失败，即使两个空闲块的合计大小足够大，可以满足这个请求。

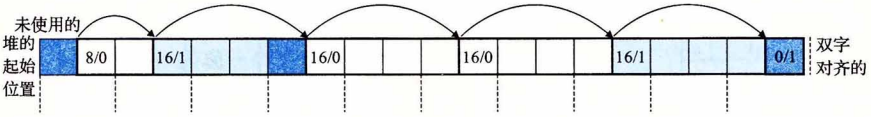

图 9-38 假碎片的示例。阴影部分是已分配块。没有阴影的部分是空闲块。头部标记为(大小(字节)/已分配位)

为了解决假碎片问题，任何实际的分配器都必须合并相邻的空闲块，这个过程称为合并(coalescing)。这就出现了一个重要的策略决定，那就是何时执行合并。分配器可以选择*立即合并*(immediate coalescing)，也就是在每次一个块被释放时，就合并所有的相邻块。或者它也可以选择*推迟合并*(deferred coalescing)，也就是等到某个稍晚的时候再合并空闲块。例如，分配器可以推迟合并，直到某个分配请求失败，然后扫描整个堆，合并所有的空闲块。

立即合并很简单明了，可以在常数时间内执行完成，但是对于某些请求模式，这种方式会产生一种形式的抖动，块会反复地合并，然后马上分割。例如，在图 9-38 中，反复地分配和释放一个 3 个字的块将产生大量不必要的分割和合并。在对分配器的讨论中，我们会假设使用立即合并，但是你应该了解，快速的分配器通常会选择某种形式的推迟合并。

9.9.11 带边界标记的合并

分配器是如何实现合并的？让我们称想要释放的块为*当前块*。那么，合并(内存中的)下一个空闲块很简单而且高效。当前块的头部指向下一个块的头部，可以检查这个指针以判断下一个块是否是空闲的。如果是，就将它的大小简单地加到当前块头部的大小上，这两个块在常数时间内被合并。

但是我们该如何合并前面的块呢？给定一个带头部的隐式空闲链表，唯一的选择将是搜索整个链表，记住前面块的位置，直到我们到达当前块。使用隐式空闲链表，这意味着每次调用 `free` 需要的时间都与堆的大小成线性关系。即使使用更复杂精细的空闲链表组织，搜索时间也不会是常数。

Knuth 提出了一种聪明而通用的技术，叫做*边界标记*(boundary tag)，允许在常数时间内进行对前面块的合并。这种思想，如图 9-39 所示，是在每个块的结尾处添加一个*脚部*(footer，边界标记)，其中脚部就是头部的一个副本。如果每个块包括这样一个脚部，那么分配器就可以通过检查它的脚部，判断前面一个块的起始位置和状态，这个脚部总是在距当前块开始位置一个字的距离。

图 9-39 使用边界标记的堆块的格式

考虑当分配器释放当前块时所有可能存在的情况：