2. 分离适配

使用这种方法，分配器维护着一个空闲链表的数组。每个空闲链表是和一个大小类相关联的，并且被组织成某种类型的显式或隐式链表。每个链表包含潜在的大小不同的块，这些块的大小是大小类的成员。有许多种不同的分离适配分配器。这里，我们描述了一种简单的版本。

为了分配一个块，必须确定请求的大小类，并且对适当的空闲链表做首次适配，查找一个合适的块。如果找到了一个，那么就(可选地)分割它，并将剩余的部分插入到适当的空闲链表中。如果找不到合适的块，那么就搜索下一个更大的大小类的空闲链表。如此重复，直到找到一个合适的块。如果空闲链表中没有合适的块，那么就向操作系统请求额外的堆内存，从这个新的堆内存中分配出一个块，将剩余部分放置在适当的大小类中。要释放一个块，我们执行合并，并将结果放置到相应的空闲链表中。

分离适配方法是一种常见的选择，C标准库中提供的GNU malloc包就是采用的这种方法，因为这种方法既快速，对内存的使用也很有效率。搜索时间减少了，因为搜索被限制在堆的某个部分，而不是整个堆。内存利用率得到了改善，因为有一个有趣的事实：对分离空闲链表的简单的首次适配搜索，其内存利用率近似于对整个堆的最佳适配搜索的内存利用率。

3. 伙伴系统

伙伴系统(buddy system)是分离适配的一种特例，其中每个大小类都是2的幂。基本的思路是假设一个堆的大小为2^m个字，我们为每个块大小2^k维护一个分离空闲链表，其中$0 \leqslant k \leqslant m$。请求块大小向上舍入到最接近的2的幂。最开始时，只有一个大小为2^m个字的空闲块。

为了分配一个大小为2^k的块，我们找到第一个可用的、大小为2^j的块，其中$k \leqslant j \leqslant m$。如果$j=k$，那么我们就完成了。否则，我们递归地二分割这个块，直到$j=k$。当我们进行这样的分割时，每个剩下的半块(也叫做伙伴)被放置在相应的空闲链表中。要释放一个大小为2^k的块，我们继续合并空闲的伙伴。当遇到一个已分配的伙伴时，我们就停止合并。

关于伙伴系统的一个关键事实是，给定地址和块的大小，很容易计算出它的伙伴的地址。例如，一个块，大小为32字节，地址为：

$$xxx\cdots x00000$$

它的伙伴的地址为

$$xxx\cdots x10000$$

换句话说，一个块的地址和它的伙伴的地址只有一位不相同。

伙伴系统分配器的主要优点是它的快速搜索和快速合并。主要缺点是要求块大小为2的幂可能导致显著的内部碎片。因此，伙伴系统分配器不适合通用目的的工作负载。然而，对于某些特定应用的工作负载，其中块大小预先知道是2的幂，伙伴系统分配器就很有吸引力了。

9.10 垃圾收集

在诸如C malloc包这样的显式分配器中，应用通过调用malloc和free来分配和释放堆块。应用要负责释放所有不再需要的已分配块。

未能释放已分配的块是一种常见的编程错误。例如，考虑下面的C函数，作为处理的一部分，它分配一块临时存储：