coalesce 函数中的代码是图 9-40 中勾画的四种情况的一种简单直接的实现。这里也有一个微妙的方面。我们选择的空闲链表格式(它的序言块和结尾块总是标记为已分配)允许我们忽略潜在的麻烦边界情况，也就是，请求块 bp 在堆的起始处或者是在堆的结尾处。如果没有这些特殊块，代码将混乱得多，更加容易出错，并且更慢，因为我们将不得不在每次释放请求时，都去检查这些并不常见的边界情况。

5. 分配块

一个应用通过调用 mm_malloc 函数(见图 9-47)来向内存请求大小为 size 字节的块。在检查完请求的真假之后，分配器必须调整请求块的大小，从而为头部和脚部留有空间，并满足双字对齐的要求。第 12～13 行强制了最小块大小是 16 字节：8 字节用来满足对齐要求，而另外 8 个用来放头部和脚部。对于超过 8 字节的请求(第 15 行)，一般的规则是加上开销字节，然后向上舍入到最接近的 8 的整数倍。

code/vm/malloc/mm.c

```
1   void *mm_malloc(size_t size)
2   {
3       size_t asize;      /* Adjusted block size */
4       size_t extendsize; /* Amount to extend heap if no fit */
5       char *bp;
6
7       /* Ignore spurious requests */
8       if (size == 0)
9           return NULL;
10
11      /* Adjust block size to include overhead and alignment reqs. */
12      if (size <= DSIZE)
13          asize = 2*DSIZE;
14      else
15          asize = DSIZE * ((size + (DSIZE) + (DSIZE-1)) / DSIZE);
16
17      /* Search the free list for a fit */
18      if ((bp = find_fit(asize)) != NULL) {
19          place(bp, asize);
20          return bp;
21      }
22
23      /* No fit found. Get more memory and place the block */
24      extendsize = MAX(asize,CHUNKSIZE);
25      if ((bp = extend_heap(extendsize/WSIZE)) == NULL)
26          return NULL;
27      place(bp, asize);
28      return bp;
29  }
```

code/vm/malloc/mm.c

图 9-47　mm_malloc：从空闲链表分配一个块

一旦分配器调整了请求的大小，它就会搜索空闲链表，寻找一个合适的空闲块(第 18 行)。如果有合适的，那么分配器就放置这个请求块，并可选地分割出多余的部分(第 19 行)，然后返回新分配块的地址。