对齐要求	已分配的块	空闲块	最小块大小（字节）
单字	头部和脚部	头部和脚部	
单字	头部，但是无脚部	头部和脚部	
双字	头部和脚部	头部和脚部	
双字	头部，但是没有脚部	头部和脚部	

9.9.12 综合：实现一个简单的分配器

构造一个分配器是一件富有挑战性的任务。设计空间很大，有多种块格式、空闲链表格式，以及放置、分割和合并策略可供选择。另一个挑战就是你经常被迫在类型系统的安全和熟悉的限定之外编程，依赖于容易出错的指针强制类型转换和指针运算，这些操作都属于典型的低层系统编程。

虽然分配器不需要大量的代码，但是它们也还是细微而不可忽视的。熟悉诸如C++ 或者 Java 之类高级语言的学生通常在他们第一次遇到这种类型的编程时，会遭遇一个概念上的障碍。为了帮助你清除这个障碍，我们将基于隐式空闲链表，使用立即边界标记合并方式，从头至尾地讲述一个简单分配器的实现。最大的块大小为 $2^{32}=4$GB。代码是 64 位干净的，即代码能不加修改地运行在 32 位(gcc -m32)或 64 位(gcc -m64)的进程中。

1. 通用分配器设计

我们的分配器使用如图 9-41 所示的 memlib.c 包所提供的一个内存系统模型。模型的目的在于允许我们在不干涉已存在的系统层 malloc 包的情况下，运行分配器。

mem_init 函数将对于堆来说可用的虚拟内存模型化为一个大的、双字对齐的字节数组。在 mem_heap 和 mem_brk 之间的字节表示已分配的虚拟内存。mem_brk 之后的字节表示未分配的虚拟内存。分配器通过调用 mem_sbrk 函数来请求额外的堆内存，这个函数和系统的 sbrk 函数的接口相同，而且语义也相同，除了它会拒绝收缩堆的请求。

·分配器包含在一个源文件中(mm.c)，用户可以编译和链接这个源文件到他们的应用之中。分配器输出三个函数到应用程序：

```
1  extern int mm_init(void);
2  extern void *mm_malloc (size_t size);
3  extern void mm_free (void *ptr);
```

mm_init 函数初始化分配器，如果成功就返回 0，否则就返回 −1。mm_malloc 和 mm_free 函数与它们对应的系统函数有相同的接口和语义。分配器使用如图 9-39 所示的块格式。最小块的大小为 16 字节。空闲链表组织成为一个隐式空闲链表，具有如图 9-42 所示的恒定形式。

第一个字是一个双字边界对齐的不使用的填充字。填充后面紧跟着一个特殊的序言块(prologue block)，这是一个 8 字节的已分配块，只由一个头部和一个脚部组成。序言块是在初始化时创建的，并且永不释放。在序言块后紧跟的是零个或者多个由 malloc 或者 free 调用创建的普通块。堆总是以一个特殊的结尾块(epilogue block)来结束，这个块是一个大小为零的已分配块，只由一个头部组成。序言块和结尾块是一种消除合并时边界条件的技巧。分配器使用一个单独的私有(static)全局变量(heap_listp)，它总是指向序言块。(作为一个小优化，我们可以让它指向下一个块，而不是这个序言块。)