## 实验名称：maclloclab

这次实验是实现一个C语言的malloc函数分配内存的算法，包括实现malloc, free, realloc等函数。衡量函数性能的标准为内存空间利用率（Memory utilization）及分配算法的时间复杂度。

首先要了解存储器实现的必要技术，针对空闲块的组织方法有以下三种：

a.隐式空闲链表(implicit free list)

b.显式空闲链表(explicit free list)

c.分离空闲链表(segregated free list)

查找空闲块的四个方法：

1.首次适配：较大块在后面，搜索时间变长。

2.下一次适配：速度最快。

3.最佳适配：存储器利用率最高。

4.分离适配，特殊情况包括伙伴系统，优点在于快速搜索和快速合并（大小相同，地址连续）。

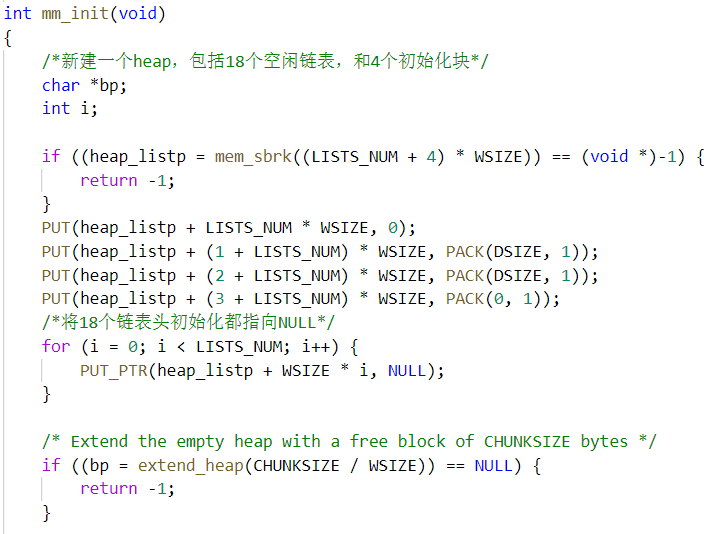
然后要考虑分割和合并操作，分割块(放置块)：在申请一个块的时候，如果这个块的大小远大于我们的需求，我们就要分割块(这个分割块的界线在于最小块的大小是多少)，分割出来的块标为空闲的，放入空闲链表中(但是在针对realloc优化的时候发现，很多时候在realloc不分割的时候空间利用率更好，可能是针对特定的case才有这种情况)。

合并块：我们想要降低空间利用率，就要降低假碎片的出现，包括内部碎片和外部碎片。

具体实现如下

1. 创建初始空闲链表

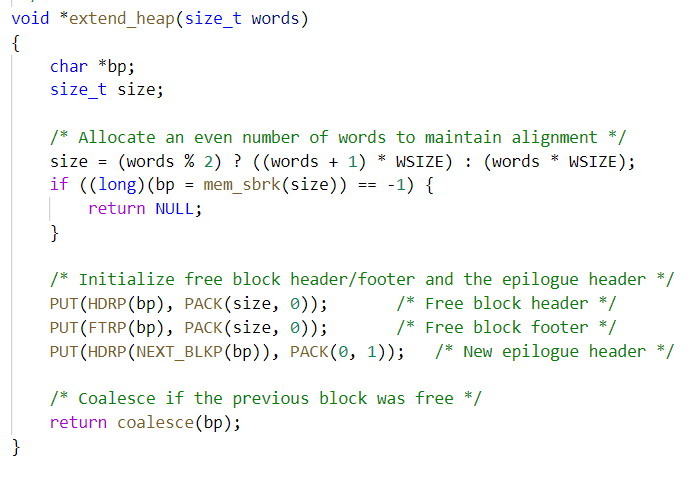
在调用mm\_malloc或者mm\_free之前，应用必须通过调用mm\_init函数来初始化堆。mm\_init函数从内存系统得到4个字，并将它们初始化，创建一个空的空闲链表。



然后调用extend\_heap函数，这个函数将堆扩展CHUNKSIZE字节，并且创建初始的空闲块。此刻，分配器已经初始化了，并且准备好接受来自应用的分配和释放请求。

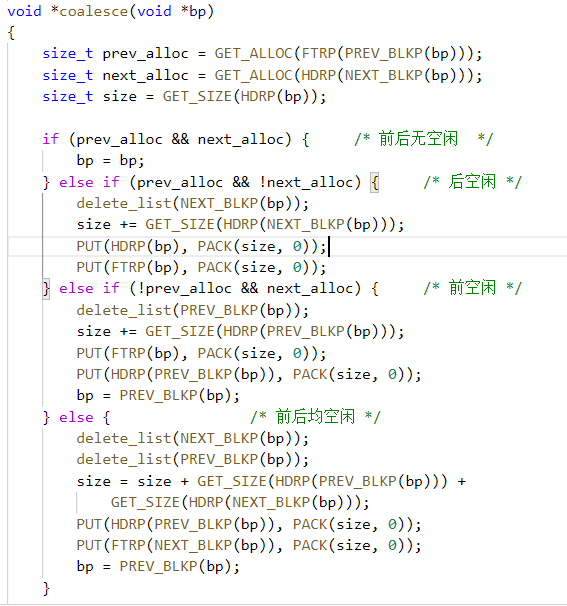
extend\_heap函数会在两种不同的环境中被调用：1)当堆被初始化时；2)当mm\_malloc不能找到一个合适的匹配块时。为了保持对齐，extend\_heap将请求大小向上舍入为最接近的2字(8字节)的倍数，然后向内存系统请求额外的堆空间。

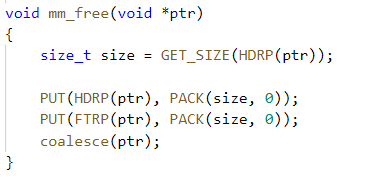
堆开始于一个双字对齐的边界，并且每次对extend\_heap的调用都返回一个块，该块的大小时双字的整数倍。因此，对mem\_sbrk的每次调用都返回一个双字对齐的内存片，紧跟在结尾块的头部后面。这个头部变成了新的空闲块的头部，并且这个片的最后一个字变成了新的结尾块的头部。最后，在很可能出现的前一个堆以一个空闲块结束的情况中，我们调用coalesce函数来合并两个空闲块，并返回指向合并后的块的块指针。



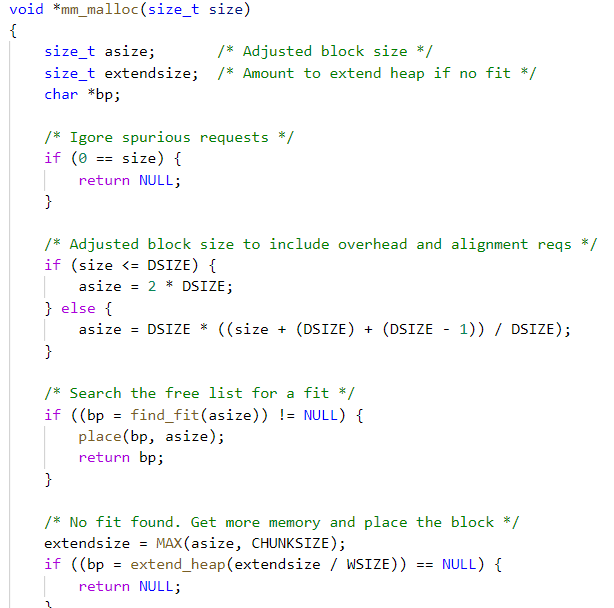
1. 释放和合并块  
   应用通过调用mm\_free函数，来释放一个以前分配的块，这个函数释放所请求的块(bp)，然后使用边界标记合并技术将之于邻接的空闲块合并起来。

普通块结构如图



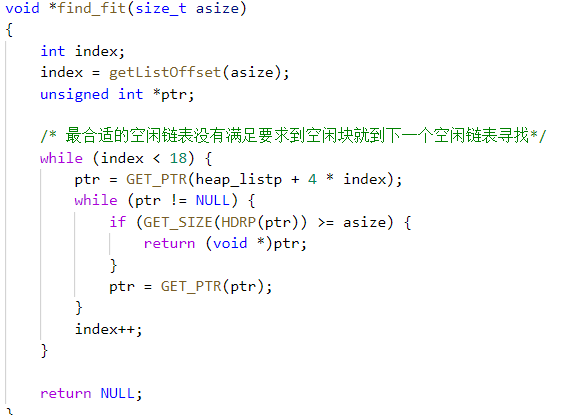


1. 分配块  
   一个应用通过调用mm\_malloc函数来向内存请求大小为size字节的块。在检查完请求的真假之后，分配器必须调整请求块的大小，从而为头部和脚部留有空间，并满足双字对齐的要求。



1. 放置策略

采用分离适配，用分离式空闲链表组织，它接近于最佳适配策略，不需要进行彻底的对搜索。它的搜索时间减少了，因为搜索被限制在堆的某个地方，而不是整个堆；内存利用率得到改善，因为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对的最佳适配搜索。



1. 最终结果

