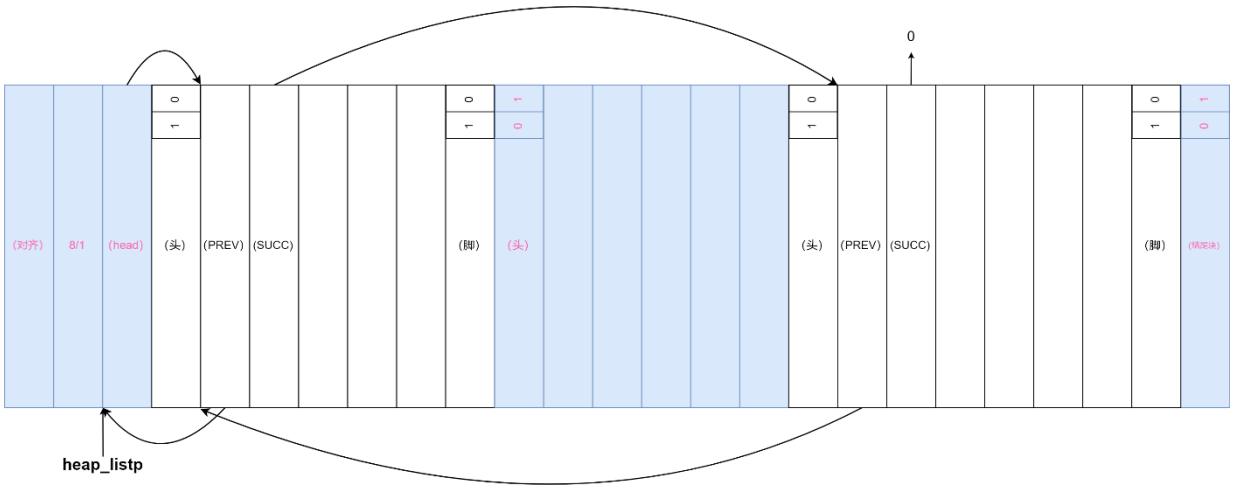
操作系统实验 3

题目：动态内存分配器的实现

PB17111623 范睿

一、 主要步骤

1. 确定链表结构



链表结构如上图所示。

- 分配块：蓝色

  最左的三个字和最右的三个字为初始化时分配的。

  中间的分配块为用户请求的，它的最左边为分配块的头。

- 分配块的头：

  第 0 位：指示本块分配情况（0 未分配，1 已分配）

  第 1 位：指示前块分配情况（0 未分配，1 已分配）

  第 3~31 位：空闲块大小（从头到脚，单位 byte，最小 4 个字）

- 空闲块：白色

  每个空闲块都有一个头、脚、PREV（指前驱），SUCC（指后继），各占一个字。

- 空闲块的头：

  第 0 位：指示本块分配情况（0 未分配，1 已分配）

  第 1 位：指示前块分配情况（0 未分配，1 已分配）

  第 3~31 位：空闲块大小（从头到脚，单位 byte，即有多少个 8 位）

- 空闲块的脚：

  第 0 位：指示本块分配情况（0 未分配，1 已分配）

  第 1 位：指示后块分配情况（0 未分配，1 已分配）

  第 3~31 位：空闲块大小（从头到脚，单位 byte，即有多少个 8 位）

2. 函数功能及核心代码：

一共使用如下几个函数：

```
int mm_init(void);
void *mm_malloc(size_t size);
static void place(void* ptr, size_t asize);
static void *extend_heap(size_t words);
static void *coalesce(void *ptr);
```

```
static void *find_fit(size_t asize);
void Change_hnf(void* bp);
void LINK(void* bp);
void Delete(void* bp);
```

1）`int mm_init(void);`

初始化函数

- 调用 mem_sbrk 请求四个字的空间，分别存放[0, 8/1, 0, 3]
  第一个字：对齐
  第二个字：8 表示第二个字和第三个字的大小，1 表示此块为分配块
  第三个字：为链表头指针，指向链表中第一个 node
  第四个字：结尾块，放 000…00011，以为此此块已分配，此块的前块已分配，此块大小为 0
- 调用 extend_heap()，分配一个大小为 4096 byte 的空闲块

2）`void *mm_malloc(size_t size);`

用户请求 size（单位：byte）大小的空间

- 计算比 size 大的最小的可对其空间大小 asize
- 调用 find_fit()找到一个满足大小的空闲块
- 若找到了，调用 place()并返回指针
- 若没找到，调用 extend_heap()增加堆大小，增加的堆大小取 size 和一次性分配的大小中大的那个

3）`static void place(void* ptr, size_t asize);`

调用 place 时，ptr 指向一个空闲块，且空闲块大小一定大于 asize

- 将当前块（ptr 指向的块）从空闲链表中删除
- 将 asize 大小的分配块放入 ptr 指向的这个空闲块
  若放后剩下的空间不足 4 个字，将这整个空闲块都分配
  若剩下的空间足够大，将剩下的变成一个空闲块加入链表

4）`static void *extend_heap(size_t words);`

- 计算可对其字长 asize
- 请求 asize 的空间（mem_sbrk），返回指针 bp 指向结尾块的后一位
- 写新请求的空间的头、脚、前驱、后继
- 恢复结尾块
- 将此空闲块合并

5） `static void *coalesce(void *ptr);`

- 若前后快都已分配，不需要合并
- 前块已分配，后块未分配，将本块和后块合并
- 前块未分配，后块已分配，将本块和前块合并
- 前后块都未分配，将此三块合并
- 将合并块头插进链表

6）`static void *find_fit(size_t asize);`

- 从链表中第一个 node 开始找，直到找到链表尾或者找到空间比 asize 大的第一个 node

7）`void Change_hnf(void* bp);`

- 改脚函数。当某块从空闲变为分配或从分配变为空闲时调用，用来修改

此块的前块的脚和后块的头

- 只有当此块不是第一个块且此块的前块有脚（即使空闲块）时，才改前块的脚
- 后块的头一旦调用都会改

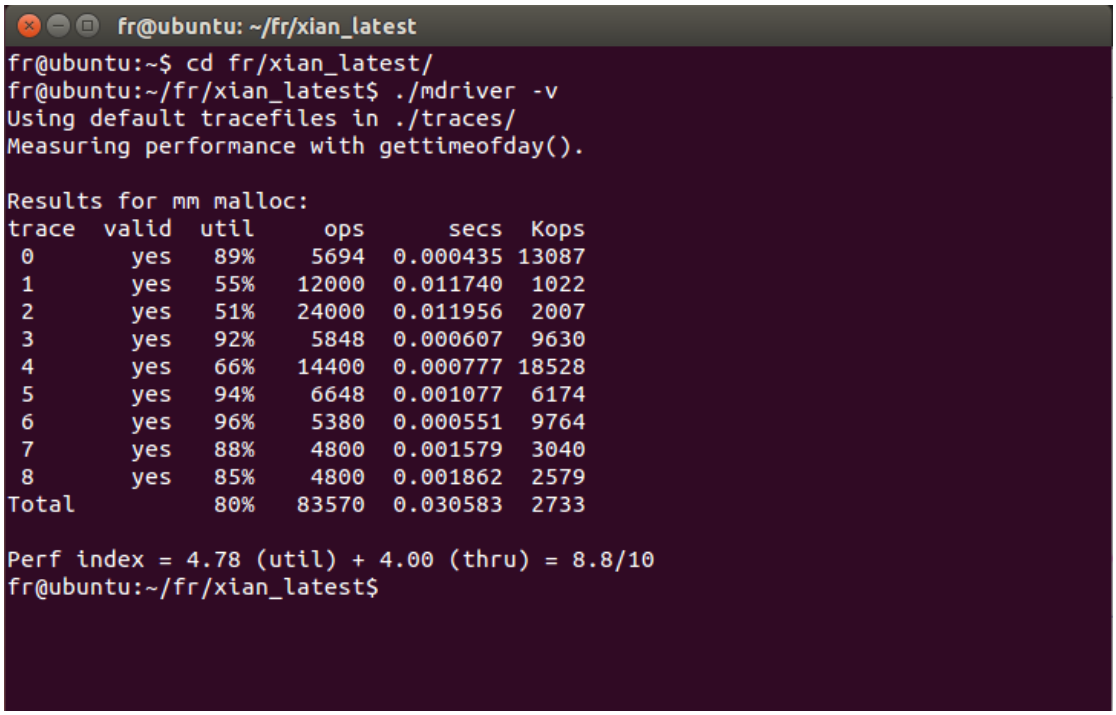
```
8） void LINK(void* bp);
```

头插函数

```
9） void Delete(void* bp);
```

删除函数：将 bp 指向的 node 从链表中删除

二、 运行结果截图



其中 trace1,2,4 利用率较低，应该是首次适配到时没有选择到最佳空闲块，导致一些空间浪费。

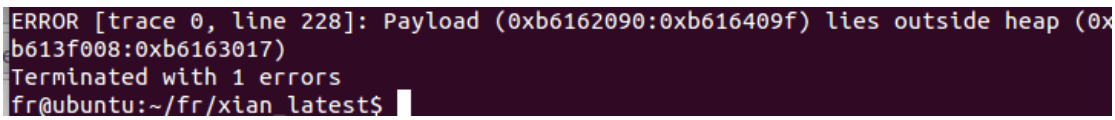
三、 技术问题

1. 链表的插入和删除

   链表的插入和删除时，需要将前驱是头指针的情况分开考虑，否则会出现 segmentation fault

2. 用户请求过小

   当用户请求的 size 大小小于 DSIZE，即 2 个字时，需要将其扩充成两个字，再加上头和与头对齐的字，一共需要四个字的空间，而不是两个字的空间。如果分配成两个字段空间，将会报错：

3. 需要调整好合并块与加入链表的顺序

   本来，我是先将空闲块加入链表，再调用 coalesce()将其合并，coalesce 函数内部逻辑不太清晰，总是报 segmentation fault 的错。然后我将代码思路改成：调用 coalesce 时，给它传递的指针 bp 指向的空闲块都不是在链表中

的，要么是新申请的空间，要么是 free 调的空间，总之这一段空闲块的前驱后继都是 NULL。然后调用 coalesce，如果可以合并，就把被合并的那一块从链表中删除，改好头脚后再将大块插入链表；如果不能合并就直接将这一块插入链表，逻辑比较清晰，结果也是正确的。

四、 实验总结

在本次试验中，我实现了在内存中利用链表结构实现 malloc 和 free。

从结果得分来看我的实现还算可以。不过还是有一些可以优化的点:

1. 空闲块的脚部的第 1 位标志位可以也存放前块的分配情况，而不是后块的。因为若想知道后块的分配情况，找到脚部地址的后一个字就是后块的信息，可以直接获取。
2. 可以设置多个头指针引导多个链表，这样可以加快查找的时间。
3. 可以设置一个指向最后一个结点的指针，可以从前往后找的同时，也从后往前找。