Lab2 实验报告

221300034 高志轩

存储的变量：在 DMA 类中，我主要存储了 self.heap(模拟内存)，self.free_blocks(存储空闲块)，self.allocated_blocks(存储已分配的块)。

数据结构设计：
1. self.heap：使用列表进行模拟。
2. self.free_blocks：使用列表进行模拟，列表元素是二元组，形如[(start， size)]。每一个元素表示 heap 中空闲块的起始位置和大小。
3. self.allocated_blocks：使用字典进行模拟，其中每一个键对应的值也是一个字典，形如 {id : {‘start’: start, ‘size’: size, ‘value’: value}}。id 表示分配块的编号，start 表示分配块的起始位置，size 表示大小，value 是一个列表，存储该块的内容。self.allocated_blocks 的格式也就是 self.data()需要返回的格式。

分配策略：
1. 对于 malloc：

我采用了 Best-fit 和 Worst-fit 相结合的方法。

一开始的时候我只采用了 Best-fit 的策略，发现外部碎片的指标比较高。我发现当我只使用 Best-fit 的策略时，当策略匹配到一个 best-fit 块，但是这个 best-fit 块只比所需要分配的大小只多一点点时，将会产生一个非常小的碎片，这个碎片未来可能比较难以被使用，所以将会留下一个较小的碎片。

但是，注意到如果 Best-fit 策略匹配到一个大小恰好与所需大小相等的块，这是非常好的，因为随着这个块的放入，外部碎片一定会减少，所以如果遇到这样的完全匹配的块时，我们将直接退出空闲列表的遍历，并分配内存。

于是，我考虑了当出现找到的 best-fit 块大小比所需的只大一点时，将会使用 Worst-fit 策略，将分配大小最大的空闲块给所需内存。其中，切换策略的判断条件为：not equal and (best_fit_size - size) < best_fit_size * ratio，其中 ratio 为一个常数比例。

2. 对于 compact：

注意到，malloc 操作并不会增加外部碎片，于是 compact 操作只需要在 free 的时候考虑。为了使外部碎片最少，理想情况下是每一次 free 操作后都进行 compact，但是将会降低执行效率。

于是，我使用了另一个比率来进行判断，也就是如果 self.frag / self.size > self.ratio，此时则进行 compact。我通过维护 self.frag 来记录当前有的碎片数，如果碎片数与 heap 大小之比大于一个比例时，此时进行 compact。