650hw1report

Name: Yifan Lin

NetId: yl734

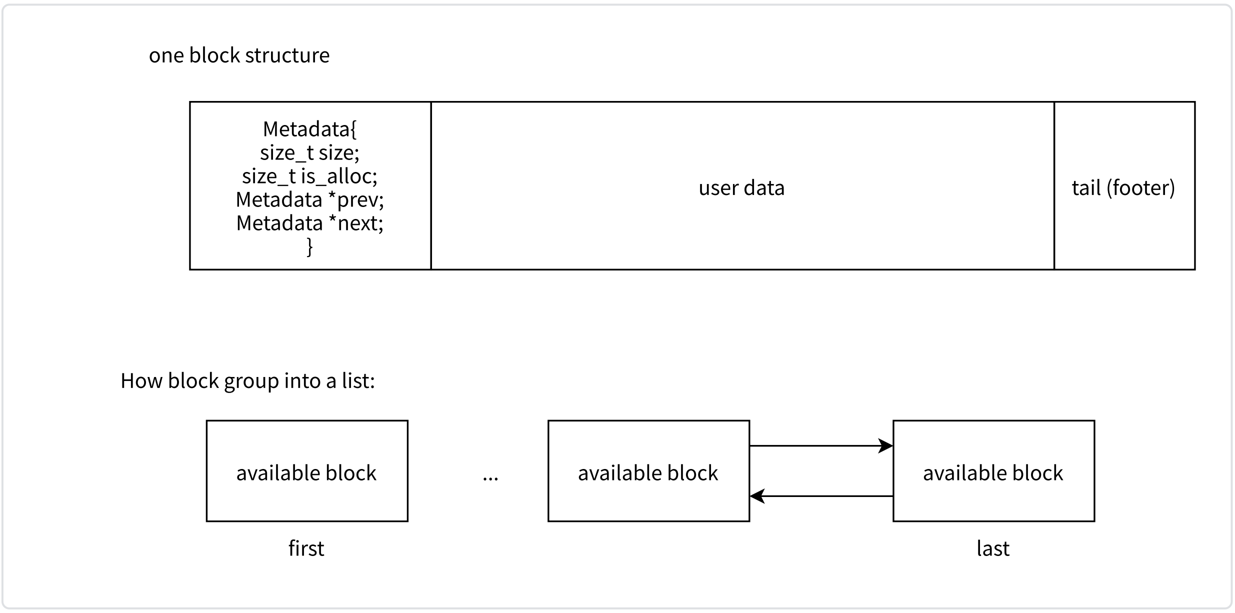
Design

Data structure

I use a Metadata structure as header of each block, and using a footer and pack the information of size and is_allocated into tail, tail is a 8 byte unsigned int.

So the memory for each block = 40 byte.

detail:



Implement Detail and Optimization:

1. **Time Optimization**
    a. All the function time complexity is O(1) (Except for ff_malloc and bf_mallloc)
    b. In order to accelerate the execution time, I define macros to do tail-related work.
    c. I maintain an out-of-order free linkedlist. When inserting the block into the free list, I always insert the block in the header to save time, therefore the linklist related function `static void remove_block(Metadata * p)` and `static void add_block_to_head(Metadata *p)` function only costs O(1) time complexity.

d. When coalesce the block with previous and next block, computing previous address base on the information in footer and next block information in Meatadata, rather than search all list to find previous or next block.
e. When using the best-fit strategy, I set a threshold(DSIZE = 8 byte) . If the block with inner fragment size is smaller than threshold, I choose this block immediately and break the searching process.
f. When reusing the old block, I set a threshold(DSIZE = 8 byte) . If the block with inner fragment size is smaller than threshold, I will not split the block into two block.

2. **Memory Optimation**
   a. In malloc data, when choosing a block much larger than the user needed size, split the block into two blocks and use the user needed size one.
   b. In free data, when freeing a block, check if the previous address block and next address block is also available . If it is, coalesce them together immediately.
   c. In order to reduce the usage of memory, I cancel the sentinels as dummy header and tail of the free list and maintain the existing header and tail block.

Result

Method	ff_malloc(first-fit)		bf_malloc(best-fit)	
	Execution time(s)	Fragmentation	Execution time(s)	Fragmentation
Small range	2.779241	0.082799	1.170692	0.023136
Equal range	1.207667	0.450000	1.445896	0.450000
Large range	12.869853	0.119004	44.355851	0.040860

From the result of the first fit malloc and best fit malloc, I find:

Execution time: the first fit is always faster than the best fit malloc, except for the situation when mallocing small range size block.

Large range size malloc always cost more time than small range and equal range, I think it is because it needs more time to search for a reusable block. And the large range size malloc in the best fit strategy costs much more execution time than the small and equal range malloc. And I think we can implement multi-level size lists to solve this problem.

Fragmentation: the best fit malloc always gets less fragmentation than first-fit, except for the situation that we malloc equal size block, it gets the same size of fragmentation. For large range malloc, because the range is large, although using best fit, it will not always choose the block with small fragmentation.