Malloc Report

- Student Name: Huidan Tan
- NetID: ht175

Implementation

implementation overview

To implement malloc and free, I keep a list of free region. Here is my meta data structure.

```
struct meta_tag
{
    meta_t * prev;
    meta_t * next;
    size_t size;
};
```

With a double linked listed, we can easily achieve merge function when two available block is adjacent.

detailed implementation

- The logic of Malloc

  Check if there is an available block that fit the required size

  If no available block to use, allocate a new one, increase `sbrk`

  If exists available block, check if need to spilt it

- The logic of Free

  Remove this block from free-list

  check `prev` and `next` to see if merge could happen (Through check the pointer address)

Performance Result

Data set/Strategy	First-Fit Exec Time	First-Fit Fragmentation	Best-Fit Exec Time	Best-Fit Fragmentation
small_rang	6.355194	0.060242	1.460531	0.022087
equal_size	15.515026	0.450000	15.613247	0.450000
large_rang	52.195808	0.093238	61.509547	0.041318

Result Analysis

- **small_range_rand_allocs**

  This program works with allocations of random size, ranging from 128 - 512 bytes (in 32B increments).

  BF needs spend more time to search the best available block to fill in than FF. For example, we request 15 bytes. The available blocks in order are `50 byte, 20 byte, 15 bytes`. FF strategy' s search time complexity is O(1). It will pick the first one. But for BF strategy, it will iterate the whole free-list to find the best suitable one. It's search time complexity is O(2). However, the results shows that BF strategy takes less time than FF strategy. This may because `sbrk` operation is much more time consuming than iteration and `spilt` operation also takes time. As we talked before, BF will search the best block to fit in, therefore it avoids wasting space (even spilt also takes much space for meta data). Therefore, it shows that BF' s fragmentation is much better than that of FF. BF can use the space more efficiently.

- **equal_size_allocs**

  This program uses the same number of bytes (128) in all of its malloc calls.

  For each malloc call, it asks for same size. The size of each block in free-list is the same and exactly meets the requirement. Regardless of the BF or FF strategy, the first block of the free list will be selected when looking for a suitable block. Therefore, there is no significant difference in performance between BF and FF. Their fragmentations are 0.45, which is around 0.5. This is because fragmentation is recorded halfway .

- **large_range_rand_allocs**

  This program works with allocations of random size, ranging from 32 - 64K bytes (in 32B increments).

  This dataset works exactly as small_range_rand_allocs. However, the result is quite different. FF strategy takes less time than BF strategy. This because the size distribution is quite much more random and size difference is much larger. For example, the free-list may seem like this : `20 byte, 4k bytes, 16 bytes, 64 bytes, 1k bytes, 16 bytes`. It is less possible to find the best suitable one at first. BF needs to take more time to iteration in order to find the best suitable one.

- **Conclusion**

  In real life, the data sizes we request may be varied largely. It may more like `large_range_rand_allocs` compared with `small_range_rand_allocs`. Therefore, I recommend `First-Fit` strategy