Report (ECE 650 HW1)

Name: Yichen Huang

NetID: yh348

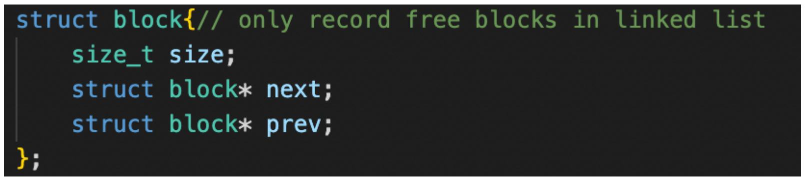
Date: 01/28/2022

Overview: Implementation Description

The Implementation can be generally divided into 2 parts (Malloc and Free), they are described below respectively.

Malloc:

The implementation is done in a straightforward way. The information (address, size) of each free block is stored in a doubly linked list (free list). The free list consists of nodes with structure shown in figure 1.



```
struct block{// only record free blocks in linked list
    size_t size;
    struct block* next;
    struct block* prev;
};
```

Figure 1. The structure of nodes in free list.

The steps are written below.

1. Check whether the size is less or equal to 0, if so, return NULL directly.
2. Implement functions named “ff_findBlock” and “bf_findBlock”, obtaining the finding result.
   - ff_findBlock (First Fit Method): traversing the free list and find the first free block which has enough space
   - bf_findBlock (Best Fit Method): traversing the free list and find the free block which has the smallest extra space than requested.

If there is a valid block, then remove it from free list and see if the block needs to be split (whether it has enough space to split or not). The removing method considers various conditions including

- The block found is tail
- The block found is head
- The block found is head and tail at the same time (there is only one block in the free list)
- The block is not head or tail (in the middle of free list)

If there is no valid block, then use sbrk() to assign a new space.

Free:

The implementation of free is to add free block to free list, and merge the free blocks if needed.

1. Check whether the pointer is NULL, if so, return NULL directly.
2. Insert the freed block to the free list in an ascending order, various conditions are also considered. The methods include
   - The free list is empty
   - There is only one block in the linked list (inserting before head and after tail are considered)
   - Inserting before head
   - Inserting after tail
   - Inserting in the middle of the linked list
3. See if the block needs to be merged. Merging should happen in the following three conditions:
   - In free list, the address of block before the current block is adjacent to the current block: merge with the previous block
   - In free list, the address of block after the current block is adjacent to the current block: merge with the next block
   - In free list, both the address of blocks before the current block and after the current block are adjacent to the current block: merge with both previous and next block

Performance Result Presentation

With two different implementation methods, different results related to

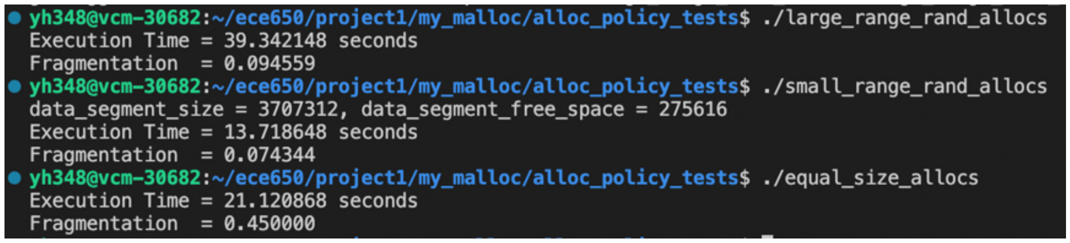
- Execution time: the run-time of the program
- Fragmentation: the amount of unallocated data segment space divided by total data segment space

are shown in Table 1.

Table 1. Results of First Fit and Best Fit

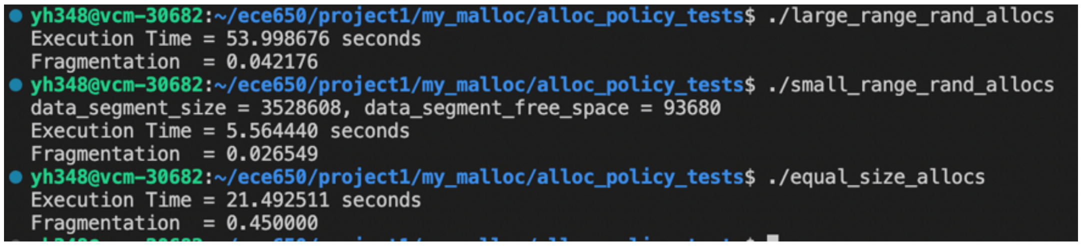
Implementation	Execution Time (s)		Fragmentation	
	First-Fit	Best-Fit	First-Fit	Best-Fit
Large	39.342148	53.998676	0.094559	0.042176
Small	13.718648	5.564440	0.074344	0.026549
Equal	21.120868	21.492511	0.450000	0.450000

The data printed in terminal has been shown in figure 2 and figure 3.



```
● yh348@vcm-30682:~/ece650/project1/my_malloc/alloc_policy_tests$ ./large_range_rand_allocs
  Execution Time = 39.342148 seconds
  Fragmentation  = 0.094559
● yh348@vcm-30682:~/ece650/project1/my_malloc/alloc_policy_tests$ ./small_range_rand_allocs
  data_segment_size = 3707312, data_segment_free_space = 275616
  Execution Time = 13.718648 seconds
  Fragmentation  = 0.074344
● yh348@vcm-30682:~/ece650/project1/my_malloc/alloc_policy_tests$ ./equal_size_allocs
  Execution Time = 21.120868 seconds
  Fragmentation  = 0.450000
```

Figure 2. Results of First Fit Implementation



```
● yh348@vcm-30682:~/ece650/project1/my_malloc/alloc_policy_tests$ ./large_range_rand_allocs
  Execution Time = 53.998676 seconds
  Fragmentation  = 0.042176
● yh348@vcm-30682:~/ece650/project1/my_malloc/alloc_policy_tests$ ./small_range_rand_allocs
  data_segment_size = 3528608, data_segment_free_space = 93680
  Execution Time = 5.564440 seconds
  Fragmentation  = 0.026549
● yh348@vcm-30682:~/ece650/project1/my_malloc/alloc_policy_tests$ ./equal_size_allocs
  Execution Time = 21.492511 seconds
  Fragmentation  = 0.450000
```

Figure 3. Results of Best Fit Implementation

Results Analysis

There are three malloc & free patterns used, each of them have different characteristics. The execution time and fragmentation are different for these three patterns. The comparison and corresponding analysis are provided below.

large_range_rand_allocs: Execution time of First Fit method is much smaller than Best Fit method. In this condition, length of free list does not change, the time is influenced by traversing time. Since the whole free list should be traversed every time if we use Best Fit method, run-time will be larger. The fragmentation of First Fit Method is twice the fragmentation of Best Fit Method, which means the space utilization of Best Fit Method is better than with First Fit Method because Best Fit Method always examine the whole free list and find the most suitable block.

small_range_rand_allocs: With “small range rand allocs”, execution time with First-Fit method is much larger than with Best-Fit Method. In this condition, length of free list decreases so that the traversing time does not influence a lot. However, sbrk() will always be used for First Fit Method, because the free list cannot be fit perfectly. As a system call, sbrk() takes a lot of time to compute, so the execution time becomes longer for First-Fit Method. The fragmentation of Best Fit Method is much smaller than First-Fit Method, the reason for that is similar to the one with large range rand allocs.

equal_size_allocs: With “equal size allocs” pattern, the execution time and fragmentation of First Fit and Best Fit are quite similar. The reason should be in this condition, both First Fit and Best Fit choose the first block in free list, so the performance does not have evident difference.