Report

—Project 1 : Malloc Library Part 1

Student Name: Yitong Wang
Student NetID: yw471

---

1. Overview of implementation

By reading the materials on the internet, I find there are many ways to implement it and finally I choose a more simple and efficient way: only keep free node in the linked list, which meanest is a free linked list.

1.1 Data Structure

The structure I use as below:

```
typedef struct free_link_list {
  size_t size;
  struct Node *next;
  struct Node *prev;
} Node;
```

I use a double linked list as the structure, because double linked list is more easier to do the remove and add function. The size_t size indicate the the space of each node have.

1.2 Malloc Function

When implement allocate, we search on the free linked list and find the node which size is what we need to allocate, if we find it, we just remove it from the free linked list. If we can't find the suitable size node, use sbrk() function to allocate a new space for it. Also we need to implement the split, which can split the big free node into the required size node, then leave the redundant resource on the linked list, which can be more efficiency and resource saving. ff_malloc function and the bf_malloc function has the same train of the thought except the bf_malloc should find the most suitable node first while the ff_malloc just need to allocate the first suitable node.

1.3 Free Function

When implement free, first find the address of the node which we need to free, then add it to the free linked list. Also need to implement the merge function, which merge the two adjacent free nodes into one. As for the ff_free function and the bf_function, they are the same.

2. Result & Analysis

2.1 Test Results

Patern	FF Execution Time	FF Fragmentation	BF Execution Time	BF Fragmentation
Small	11.781673 s	0.080257	2.083354 s	0.028895
Equal	15.088702 s	0.450000	15.085915 s	0.450000
Large	37.114824 s	0.093675	46.271331 s	0.083008

The provided tests were run several times in different computer , each combination of the malloc policy and the tested size resulting in different execution time, the fragmentation are the same.

2.2 Analysis

For equal size allocs, since all the block have the same size, it always will use the first block to malloc, no matter use any policy, so the bf and ff function perform the same, the execution time and the fragmentation are the same.
For small size allocs, using bf policy will take shorter time because when using bf, it just scan the whole list and easier to find the suitable one, but for the ff policy, it needs more time to finish the split or merge function.
For large range allocs, the malloc memory size is large, the free linked list is longer ,if using bf policy, it needs more time to traverse the whole list, thus the execution of the bf policy is longer than the ff policy.
When writing the code, I made some mistakes which cause me a lot of time to find and debug, one is the linked list operation, easy to make wrong arrows, one is that didn't maintain the free linked list all the time, when there is no node on the list, still need to maintain the head.