ECE650 Proj1 Report

Yue(Joey) Yu
yy373

1.

In this project, I use two version First Fit and Best Fit to implement malloc() and free() function. I build a data structure which is similar to double linkedlist, I name it as freelist to store the free memory. Its structure is as follows:

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

The main idea is that, every time when I call my_malloc function, it will search the available memory from the freelist. If a suitable memory space based on first fit or best fit can be found in the freelist, then I remove the node from the freelist, and use this memory space again as a newly allocated space. Otherwise, I will call the sbrk function to allocate a new memory space with the total size (sizeof(Node)+size). Besides, when I call the my_free function, I would add this memory space, which is an address as the parameter of my_free function to the freelist, and do the merging operation to check whether it can be merged with its previous Node or next Node to form a new Node with bigger size.

```
void *findFirst(Node* target,size_t size);
```

First, using the while loop to search an available space on the freelist, if it is not found, return null. If there is such a space, check its size. If its size is big enough, doing the split operation, otherwise, just removing it from the freelist.

```
void splitNode(Node* temp, size_t size);
```

When I use first fit or best fit to search the freelist to get a Node with suitable size, the size of the Node may be bigger than the size I need. In order to save the memory, I build a new Node with the size SplitTarget->size-size-sizeof(Node) and replace the original Node in the freelist.

```
void AddAndCheck(Node * target);
```

When calling my_free function, I need to add the Node to the freelist, namely calling the AddAndCheck function. First, as normal, just add the Node to the freelist. Then, I need to check whether this newly added address space is continuous with its previous Node or next Node in the freelist. If it is continuous with its previous Node, than add sizeof(Node)+target->size to the size of the previous Node, and then remove the newly added Node. For another situation, doing the similar operation.

```
void *findBest(Node* target,size_t size);
```

The only difference with findFirst function is the while loop. I need to traverse all of the Nodes in the freelist, finding a Node with the smallest size larger than the target size.

2.

	First Fit		Best Fit	
	Execution Time	Fragmentation	Execution Time	Fragmentation
Small	11.265774	0.073883	4.874224	0.034287
Equal	14.865619	0.45	14.945771	0.45
Large	36.68718	0.093421	53.173882	0.093275

For both first fit and best fit, the execution time and fragmentation of equal_size_allocs are nearly the same. Allocating and freeing same bytes of space all the time, the Best fit version is actually the same as the first fit, since in the freelist, all the node are with the same size 128. And every time calling the bf_malloc function, the first Node in the freelist is always what we want, which is the same as first fit version. That's why the execution time and fragmentation are the same.

For small_size_allocs, the execution time of the Best Fit is much shorter than the first fit. In my opinion, the range of the requested bytes are very small, from 128 to 512 bytes, in 32 bytes increments, so the size of the freelist is smaller due to the BF strategy. The findBest function may traverse the freelist faster. Besides, compared to Best Fit, First Fit may often call the split function, since the size of the Node the First Fit finds are more often bigger than the Best Fit finds. It is inevitable to call the split function more often than the Best Fit. Besides, the fragmentation of Best Fit is much smaller than First Fit. It is easy to explain, since Best Fist always search a node with smallest size bigger than the request.

For large_size_allocs, the execution time of the Best fit is longer than first fit. Because the range of the requested bytes are very large from 32 to 64k bytes. In the freelist, the size of Nodes are very different. Thus, the findBest function may spend lots of time to traverse to find the most suitable node, while the findFirst function only needs to find a node with the size bigger than the requested size, which is much faster. However, it is strange that the fragmentations of both two versions are nearly the same. From my perspective, it is because the range of the requested bytes are very large, although the Best Fit try to find a Node with smallest size bigger than the requested size, the size of the Best fit Node is still far from the requested size. So, the Best fit does not help a lot under the situation of large_size_allocs.