ECE 650 – Spring 2020
Assignment #1: Malloc Library Part 1

By Zhu

- Overview:

In this assignment, we are required to implement memory library functions (malloc & free). The core system call API I used is 'sbrk'.

There are some allocation policies I obeyed in the assignment:

a) Keep a data structure that represents list of free memory regions.

First, I used a double linked list struct to keep the information about not only free memory regions, but non-allocatable memory regions. I distinguished them by setting variable's property: is_free (1 represents free region, 0 represents not free). The node is telling the meta information about the memory next to it, or to say managed by it.
The creation of metanode is as followed:

```
node_ptr = sbrk(0);
void * malloced = sbrk(size + NODE_SIZE);
```

The relationship can be clearly found that every memory chunk is controlled by a single metanode which may connect with other nodes.

b) Merging the adjacent free regions into a single free region of memory.

Merging is applicable for both BF and FF strategy. When others invoke free API to release memory, they would pass a pointer of a certain item to free function. I use this provided pointer to find the meta node.
By pointer math operation:

```
struct link_node* p = (struct link_node*)ptr - 1;
```

The leading memory body's address – a struct length = the first byte's address of the metanode.

c) A policy to minimize the size of the process's data segment.

The policy I designed to minimize (free) size is to split the space into two manageable part if the content size is lower than the free region. Then we logically created the free region (not expanding the whole memory, but using nodes to empower memory manageable space) and avoid to wastefully "sbrk" new space. Just to be noted, my splitting strategy is more proper for FF(fast fit) and I compared the result of BF/FF with and without splitting later. To some extent, Best Fit is already a matching algorithm when it searches spaces.

This policy may influence the time cost. Because the total number of nodes can be increased and the time to traverse it (as a common operation in my programs) would be extended.

d) Implement two strategies (First Fit & Best Fit) to allocate the memory.

They shared two same functions in my program: free and set new malloc. The biggest distinction of them is their search function when node_list is not empty(My program search function would return the node when it found the space to insert, otherwise we have to “sbrk” new space in heap).

- Performance Analysis

Scenarios	indications	FF(Fast Fit) without spliting	FF(Fast Fit) with spliting	BF(Best Fit) without spliting	BF(Best Fit) with spliting
Equal	Time(s)	396	396	396	397
	Free_size	-	-	-	-
	Largest_free	-	-	-	-
	Fragmentation	0.9998	0.9998	0.9998	0.9998
Large	Time(s)	76	114	198	280
	Free_size	9237888976	33699472	9692109232	34459416
	Largest_free	169584	5039832	0	9250808
	Fragmentation	0.9999	0.85	1	0.7315
Small	Time(s)	30	61	57	73
	Free_size	221844824	8186648	154349976	4625288
	Largest_free	0	69480	0	1288
	Fragmentation	1	0.9915	1	0.9997

a) run-time

The average costed time in FF is smaller than BF strategy. Because if we want to find the best node to insert new memory, BF needs a whole traverse in the linked_list structure. Therefore, FF is always faster.

When I implemented splitting strategy, the time would be extended. Because the nodes are fragmented so that the total number of nodes increased.

b) memory fragmentation

The fragmentation looks pretty high overall. The main reason is that I only use one linked_list to manage the data, which means I cannot dynamically arrange the received memory into the proper data structure for managing. In reality, the operation system maintains at least three linked_lists to do that, including fast bin, small bin and large bin.

An interesting thing I found is that when I do not implement splitting method in BF, the largest free node is always 0(it represents we cannot use this space anymore, it’s wasted). Because if the number of free and malloc are the same, it cannot get free_node, and it would waste massive memory in this way (waiting garbage collection operation to rearrange the space).

Generally, I would recommend my FF with splitting strategy because it has more balanced performance. However, splited BF in second scenario can save more space. Therefore, if the effectiveness of space usage is important for certain problems, then we should consider using BF.