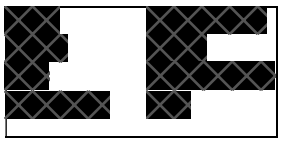
Report for Homework 1 – ECE650



Introduction

To have a comprehensive understanding of Dynamic Memory Allocation, this project required students to study the Dynamic Memory Allocation Policies and practice follow it to implement Malloc Library. There are three main requirements in this project, which are malloc() implementations, free() implementations, and Performance study report.
In this project, CLion 2021.2.3, and Ubuntu 18 LTS have been used to implements Malloc Library, and MS Words has been used to write this report.

This report mainly consists of two sections, the first section is the design, implementation, and test of the Malloc Library, and the second section is Performance Results and Analysis of the Malloc Library Implemented.

Design, implementation, and test

```
-------------------------------------------------------------------------------------------------------------------------
| LastTail   || head 1 |           | tail 1 || head 2 |           | tail 2 || head 3 |           | tail 3 || LastHead   |
|            ||        |           |        ||        |           |        ||        |           |        ||            |
|            || size 1 | content 1 | size 1 || size 2 | content 2 | size 2 || size 3 | content 3 | size 3 ||            |
-------------------------------------------------------------------------------------------------------------------------
```

Code Block 1 the structure of Boundary Tags, dynamic allocation memory data structures

At the beginning, the Boundary Tags method has been chosen as the dynamic allocation memory data structures in this implementation, which is a Bidirectional Coalescing of implicit list, raised by Prof. Donald Knuth in 1973, in which structure, two tags, a head and a tail, are used to enclose the allocated memory space. These two tags save the total size of the memory spaces used (including themselves), which can be used as an offset in finding the adjacent segments. Meanwhile, the last bit of each tag is used to represent if this segment has been freed, due to the last two bits (LSB) are always 0, because of the 4-byte aligned in the memory allocation.

The head	next_free_block_ptr	prev_free_block_ptr	…	…	The tail

Figure 1 A Free block

The head	…	…	…	…	The tail

Figure 2 An Allocated Block

At the same time, this program maintained a Free Block List which would use the free blocks' to store the prev_node_ptr and next_node_ptr (as shown in Figure 1), which would generate a doubly linked list for free blocks and this method would greatly improve the performance of malloc because the program would not go through the whole memory in the heap just need to find a free block.

Both First Fit and Best Fit implementations shared most of the codes, such as splitting the free space, increase the total allocated space by calling sbrk() when not enough free space, merging the

adjacent free regions into a single free region of memory, and also in the garbage collection, to shrink the total allocated size.

However, BF and FF only have the differences in finding the fit space. In FF, the new allocation would perform a linear search, because of it is a doubly-linked-list-like structure. The search would begin from a dummy head, also the first meta data, and go through the whole allocated memory, which have a time complexity of O(n), but has an Ω(1).

However, BF has O(n) and Ω(n) in each search because it would always go through all segments to find the min one which also meet the requirements. But there is an optimization way it can stop searching once it got the well fitted segment (the free segment has the same size of the space the allocation required).

For free() implementation, this function would set the both tags' last bit as 0 to implies that segment has been freed, and check the adjacent segments if they can be merged. For shrinking the total space allocated in memory, it would call sbrk() to deallocate the space of the segment if it is the top of the Heap.

Finally, all functions have been tested step by step and using CLion's Mem View window in the debug mode to monitor if the memory behaviors as expected.

Performance Results & Analysis

Compare the performance of the 2 implementations on the 3 different provided benchmarks. Explain the reason of the differences. Draw your conclusions on possible guidelines for choosing an implementation. Please report data in clearly understandable way.

Table 1 The Performance of BF and FF

		Equal size allocs	**Large range rand allocs**	**Small range rand allocs**
Arguments	Num_Iters	10000	50	100
	Num_Items	10000	10000	10000
FF	Time (s)	1.999194	12.300753	3.078343
	Fragmentation	0.999889	0.993432	0.993893
BF	Time (s)	2.332304	**83.587090**	1.071467
	Fragmentation	0.999889	0.953468	0.942116

As shown in the Table 1, the BFs have not much difference in Fragmentation comparing with FF. However, in large range random allocations and the small range random allocations, BF have a

even worse performance than the FF's. BF at the same time would spend a lot more time in allocating equal size memory and large range random memory. Which is due to the BF searching would go through the whole segments which will result in a worse big-theta time complexity than the FF.

In the small range random allocation, the BF has a less time consuming than FF. Which might be because the FF would produce a lot of segments in splitting a large memory space, but BF tend to choose the smallest free memory and not splitting a large memory space. Thus, BF has a larger possibility to shorten the total length of the Free Block LinkedList than FF. which would result in a shorter searching time in BF.

However, in the large range random allocation, the difference between FF and BF of possibility to shorten the total length of the Free Block LinkedList might not so significant. This is because the "large range" implies that the BF is hard to find, which would come to a splitting in most of the time. The BF wouldn't shorten the length of the LinkedList while searching for a BF would tend to go through the whole list, which result in a high consumption of time.

In conclusion, In the small range random allocation, the BF is a better choice. But in the Equal size allocation and Large range random allocation, the FF is a better choice.