Project Report #1: Malloc Library

Name: Zilin Xu NetID: zx112

Section 1: Requirements and summary of development.

In this project, i implement malloc() and free() functions in C using a system call called sbrk(), and for each function I implement first_fit and best_fit two ways. In addition, in order to study the time of the project, I implement two functions called get_largest_free_data_segment_size() and get_total_free_size().I design my code and test my work on VM from ECE551.

Section 2: Design, implementation and test

2.1 The datastructure and some global variables

To implement the project, I need a datastructure to allocate memory blocks and also for each block, it need certain fields to store some basic information. My datastructure is linked list and here is the datastructure looks like.

```
//the datastructure: doubly LinkedList to represent the memory blocks
typedef struct datastructure{
  size_t size;//size of block
  int isFREE;//the free mode of one block
  struct datastructure* next;//next pointer
  struct datastructure* prev;//previous pointer
}block_t;
```

I named the datastructure “block_t” and it has four variables. Size is its size, isFREE records the free status of a block, prev and next are two pointers pointing to previous and next block.

In addition, I have some global variables for convenience. They are head and tail.

2.2 Function implementation

I will first introduce some helper functions and then introduce four main functions.

find_ff: It is the function we use to find the first free block in the memory. It first deal with the corner case head == NULL, then initialize a pointer to traverse the list to get the answer.

find_bf: It is the function we use to find the best fit block in the memory(i.e. the list). It loop

the list and jump the invalid blocks. Then it returns the best fit block.

allocate_block: It mainly servers for two malloc functions. When we don't find the suitable blocks, we use sbrk() from system call to create a new block to use. Then, we initialize the each variable of block.

split_block: When we allocate blocks that too large to use, we split it. The function takes the parameter p as the target block we want to split. Then initialize a variable new to split from p. Then reassign p and new's variables.

merge_block: It merges two adjacent free blocks in two conditions, merge itself and its next; merge itself and its previous. The algorithms are pretty similar

free_blocks: It first us a variable to get the block without its head, then set the isFree variable equals to 1. Then call merge_block to merge all adjacent blocks.

ff_malloc: First of all, we deal with corner case (i.e. size == 0). In addition, we have to check the head and tail and do corresponding operations. Then we use findBlock_ff to find a first fit block. Then, if the block is able to split, we use split function and remember maintain free_size variable. If the find_ff don't give us expect result, we call allocate_block to generate a new block.

ff_free: call the free_blocks function.

bf_malloc: Similar with ff_malloc,only difference is call find_bf function.

bf_free: same as ff_free.

2.3 The difference between first fit and best fit strategy

For the first fit, we simply traverse the list and whenever we meet a block that is free to use and size is greater than the given size, we return it.

For the best fit, we need assign a variable diff to record the difference between the given size and the block size. By comparing the difference we can figure out which block is the best fit.

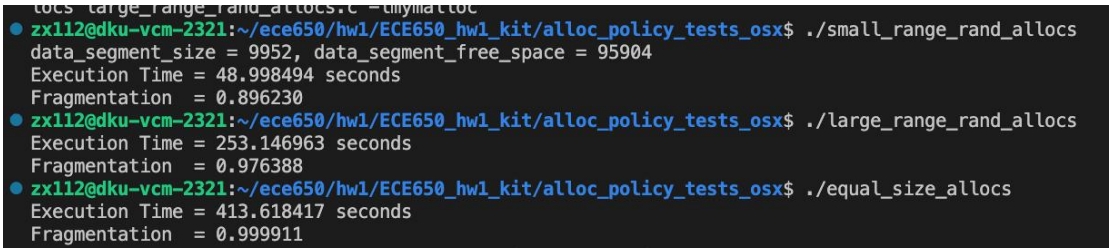
2.4 Test & Debug

I wrote a printlist() function in .c file and call it in general test to check the list and debug.

Section 3: Performance Results & Analysis

Here is the screen shoot for my result.

FF



BF

Execution time

	First fit	Best fit
Small range	48.998494 seconds	50.106909 seconds
Large range	253.146963 seconds	258.409945 seconds
Equal size	413.618417 seconds	413.642128 seconds

For equal size, the execution time for both strategies are same. That is because the program uses the same number of bytes in all of its malloc() calls. For small range and large range. They are pretty much similar and the best fit always beyond the first fir for several seconds.

The reason that they are similar is the algorithm I implemented them are same. I firstly call find_ff and find_bf functions to traverse the list and try to find a suitable block for each strategy. If we fail to find we will use allocate to generate new blocks. Meanwhile we have to check whether the block can be able to split or not.So their running time are similar. The reason that best fit is larger is, in find_bf function, we need maintain diff variable to record and figure the best fit block. So the running time for find_bf is more than find_ff. Overall the running time for best fit is larger.

Fragmentation

	First fit	Best fit
Small range	0.896230	0.896230
Large range	0.976388	0.976388
Equal size	0.999911	0.999911

The result shows they shall the same fragmentation. That may happen since they are using same space. Also the data_segment_size = 9952, data_segment_free_space = 95904 are the same.

In conclusion, I recommend the best fit strategy. Since the running time for best fit is just few seconds larger, but the best fit can ensure us to find the best block.