ECE 650 Homework-1 Report

Prathikshaa Rangarajan - pr109

The aim of the homework is to implement *malloc()* and *free()* in C using linux BSD system call *sbrk()* which is a C wrapper on *brk()* system call.

The homework requires two different implementations of malloc - the first fit policy and best fit policy.

Design

The hardest part of this project was the design of the code. Various design decisions impact the performance and stability of the code. These were hard decisions to make.

Data Structure to track heap blocks

The code uses a doubly linked list to track only the free or available memory segments. Having a doubly linked list was helpful in making modifications to the linked list, such as removing the identified node, adding a node, etc.

In order to be able to track total data segment size, I save the initial heap point as *heap_start* by saving the *sbrk(0)* value upon the first malloc call. Tracking the first malloc call was initially incorrectly implemented by only checking if the head and tail of the linked list is NULL - this happens whenever the LL is empty, i.e. it could happen even when the entire heap memory has been allocated. This was corrected by using a global *is_first_malloc* variable as a boolean checker.

In order to track each memory location and it's size, the linked list nodes are used as block metadata at the start of each malloced and free block. A malloced block contains a valid size but invalid pointers. A free block is a part of the linked list structure with valid pointers and size.

When user gives a malloced pointer to free, the program adds the node to the linkedlist. This is then available to subsequent mallocs with its size already being valid. However, if the user frees a pointer not provided by malloc, the behaviour is undefined, similar to the functionality of *malloc()* from the C standard library.

Growing the Heap

Whenever there is insufficient memory in the free list, an *sbrk()* system call is made. In order to improve the efficiency of this method, my program requests 4x the size requested by the user, including the metadata overhead. This reduces the number of system calls made in case of large mallocs with minimal frees.

Other Functions

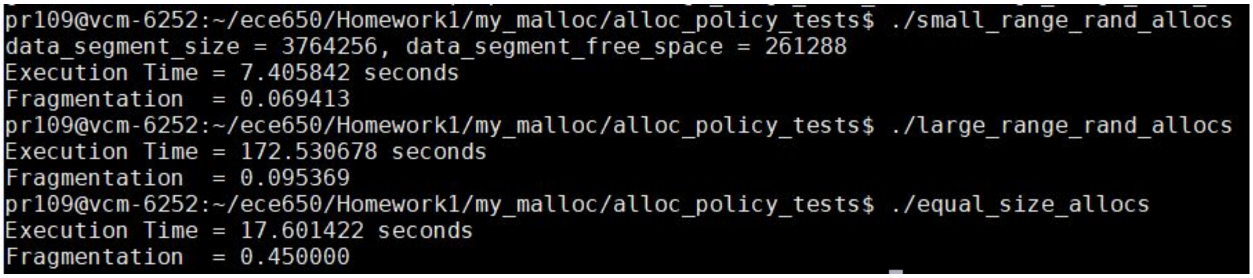
insert_free() adds the node back to the free list
remove_free() removes the node from free list when malloced
Merge() and *split()* are required functions to merge adjacent memory blocks and split a free block larger than requested size
For *ff_malloc()* and *bf_malloc()*, I use a common *my_malloc()* function which takes an additional input *ff* to choose from ff_search and bf_search as the rest is identical between the two

Performance

First Fit performs better on small range allocation pattern as opposed to the large range allocation policies. The equal fit seems to show rather absurd results which may be due to some overflow possibility. Best Fit may not have much impact on equal range allocs but may improve performance for large range allocs.

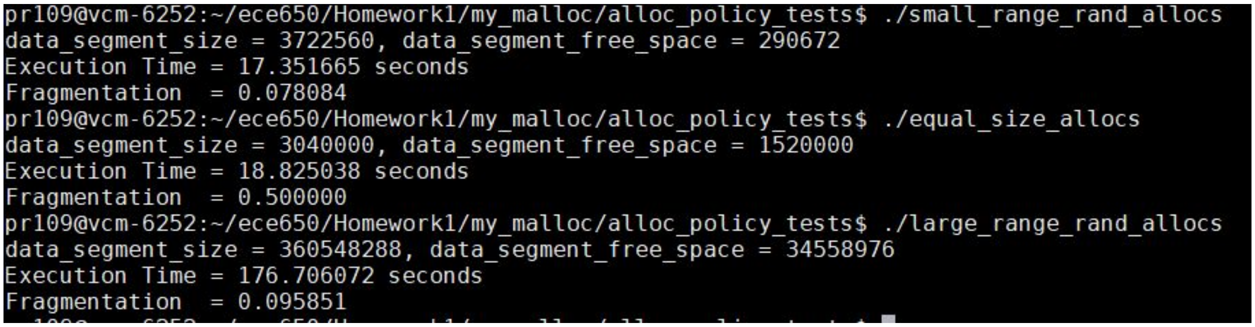
First Fit Allocation Policy

1) Split not performed if remaining memory after allocating the requested size cannot fit at least two metadata blocks



```
pr109@vcm-6252:~/ece650/Homework1/my_malloc/alloc_policy_tests$ ./small_range_rand_allocs
data_segment_size = 3764256, data_segment_free_space = 261288
Execution Time = 7.405842 seconds
Fragmentation  = 0.069413
pr109@vcm-6252:~/ece650/Homework1/my_malloc/alloc_policy_tests$ ./large_range_rand_allocs
Execution Time = 172.530678 seconds
Fragmentation  = 0.095369
pr109@vcm-6252:~/ece650/Homework1/my_malloc/alloc_policy_tests$ ./equal_size_allocs
Execution Time = 17.601422 seconds
Fragmentation  = 0.450000
```

2) Split not performed if remaining memory after allocating the requested size cannot fit at least one block metadata



```
pr109@vcm-6252:~/ece650/Homework1/my_malloc/alloc_policy_tests$ ./small_range_rand_allocs
data_segment_size = 3722560, data_segment_free_space = 290672
Execution Time = 17.351665 seconds
Fragmentation  = 0.078084
pr109@vcm-6252:~/ece650/Homework1/my_malloc/alloc_policy_tests$ ./equal_size_allocs
data_segment_size = 3040000, data_segment_free_space = 1520000
Execution Time = 18.825038 seconds
Fragmentation  = 0.500000
pr109@vcm-6252:~/ece650/Homework1/my_malloc/alloc_policy_tests$ ./large_range_rand_allocs
data_segment_size = 360548288, data_segment_free_space = 34558976
Execution Time = 176.706072 seconds
Fragmentation  = 0.095851
```

Fragmentation is higher when small remaining chunks are left to be used in the first fit allocation policy.