Project1 Malloc Library Part 1

Shuyan Sun (ss1316)

January 20, 2022

1. Implementation

First of all, I implemented a struct **block_t** to store the size of the current block, the next block it points at, and the previous block it points at.

All the struct block_ts are linked together as a double linked list, each block_t represents a consecutive space, and the actual space one block occupied is its variable size plus sizeof (block_t).

A pointer called head points to the start of the linked list and a pointer called tail points to the end of the linked list.

struct block_t contains:

size_t size

block_t * next

block_t * prev

malloc

When the malloc functions are called, it would search the linked list from the block that pointer head points at to find a suitable block. There are two situations that the program regard them as suitable. One situation is that the size of the block is greater than the input size plus sizeof(block_t), which means this block could be split into two parts (implementing by **split_block** function); the other situation is that the size of block is exactly equal to the input size(implementing by **update_block** function).

Especially, **ff_malloc** function would use the the first suitable block, while **bf_malloc** would use the smallest block among all the suitable blocks. If there is no suitable block, both of them would create new space by calling **create_space** function. In this function, the program would use sbrk function to create new free space.

free

When the free functions are called, the program would find the situation where the latest freed block in the linked list. At first, the free function would link the latest freed block into the linked list. Then it would call **merge_blocks** function to merge blocks which are adjacent. The implementation would be introduced in the following. **ff_free** and **bf_free** functions share the same logic, thus I implemented bf_free by calling ff_free function.

split_block

The current block is separated into two parts. The first part is allocated, and the size of it is updated to the input size, while the size of the second part is the original size of the current block menus size and sizeof(block_t). After that, updating the linked list.

merge_blocks

In this function, the program checks the previous block of the input block and the next one of it. If the blocks are adjacent, they would be merged into one greater block.

get_data_segment_size

The size of data segment would increase only when the sbrk function is called in the create_space function. Therefore, I used a global variable data_segment_size to record the size of data segment, and when the sbrk function is called, the program make it plus the new space.

When the program call the function, the function return the variable data_segment_size.

get_data_segment_free_space_size

The size of free space in the data segment is equal to the sum of the size each block in the linked list. Thus, when the function is called, the program starts from the head of the list, and then traverses the list until it reaches the end of it. Using a variable to record the increasing size. After traversing the whole list, the program return the variable.

2. Results

	First Fit (Time/Fragmentation)	Best Fit (Time/Fragmentation)
Equal Size	14.927 sec/ 0.45	15.112 sec/ 0.450
Small Size	14.242 sec/ 0.087	9.829 sec/ 0.027
Large Size	34.478 sec/ 0.094	55.872 sec/ 0.042

3. Result Analysis

Equal Size

In this situation, the performance of first fit and best fit functions are almost the same. Since the size of each bunk is equal, when first fit and best fit operate, they would take the same decision.

Small Size

The result shows that BF costs less time than FF, at the same time, it has less fragmentation than FF too. The reason is that allocations of random size are relatively small. BF gives a better solution to allocate data. Compared with FF, BF needs less times of splitting blocks and merging blocks, thus it needs less time overall. For BF is a better solution to allocate data, it would leave less free space, and the fragmentation is smaller.

Large Size

The result shows that BF takes more time to search for the suitable block, while it has a better performance. The reason is that most of time, BF needs to traverse the whole linked list to find out the most suitable block, while FF only needs to find out the first suitable block. In this situation, BF costs more time, while its fragmentation is smaller.