

Computer Operating Systems

BLG 312E

Homework 2 Report

KAAN KARATAŞ

karatask20@itu.edu.tr

Faculty of Computer and Informatics Engineering

Department of Computer Engineering

Date of submission: 03.05.2024

1. Implementation

1.1. InitMyMalloc

The 'InitMyMalloc' function is the initiator for the memory allocation system, and is responsible for setting up the memory list to manage dynamic memory allocation. Initially, it checks if the memory list is already initialized or if the provided 'HeapSize' parameter is non-positive to ensure the logic works correctly. If either condition is met, the function returns with the value of -1, as failure. Following this, the function ensures that the 'HeapSize' parameter is adjusted to align with the system's page size, to make sure of effective memory management.

Then, memory allocation for the heap is requested using the 'mmap' system call, specifying the desired size. If the memory allocation fails, most probably due to insufficient memory, the function returns -1, indicating failure. Conversely, upon successful memory allocation, the function initializes the memory list node with the allocated size, marks it as free, and sets its next pointer to NULL.

Finally, the function concludes by returning 0 as specified to signify successful initialization.

1.2. MyMalloc

The 'MyMalloc' function dynamically allocates memory based on the specified strategy. It first checks if the memory list is initialized, prompting initialization if not. Then, it iterates through memory blocks, employing BestFit, WorstFit, FirstFit, or NextFit strategies to find the most suitable block for allocation.

Upon finding a suitable block, it allocates memory accordingly, for the cases entirely or by dividing larger blocks. If no suitable block is found, NULL is returned.

1.2.1. dividing blocks

The 'divideBlock' function splits a memory block into two parts to accommodate allocation requests. It takes a block pointer and requested size as input. It calculates the address for the new block, updates its attributes, then adjusts the original block's attributes accordingly.

1.3. MyFree

The 'MyFree' function is responsible for deallocating memory previously allocated by 'MyMalloc'. It first checks if the pointer is NULL, in which case it does nothing and returns. Then, it verifies if the pointer falls within the heap memory range to make sure that it's a

valid pointer. If valid, it marks the corresponding node as free and invokes 'combine' to coalesce adjacent free memory spaces. If the pointer is outside the heap range, it prints an error message indicating an invalid pointer. This function ensures proper memory management by freeing allocated memory and to maintain memory coherency within the heap.

1.3.1. combine

The 'combine' function merges adjacent free memory blocks within the memory list. It initializes a pointer to traverse the memory list, starting from the beginning. While traversing, it checks if the current node and its next node are both free. If so, it merges them by updating the current node's next pointer to skip the next node and adjusts the size of the current node to encompass both blocks. This process continues until the end of the memory list is reached.

1.4. DumpFreeList

The 'DumpFreeList' function displays the current state of the free memory blocks within the memory list. It begins by initializing a pointer to traverse the memory list, starting from the beginning. Then, it prints a header indicating the format of the subsequent output. Next, it iterates through each node in the list, printing the address, size, and status (empty or full) of each node.

Finally, it moves to the next node in the list until reaching the end. This function provides insight into the available memory blocks and their utilization status, to aid in debugging and for monitoring memory usage within the system.

2. Extra Comments

2.1. To Malloc Or Not To Malloc

The main difference between "malloc" and "mmap" lies in their approach to memory allocation. "Malloc" allocates memory from the process's heap, managed by the C runtime library, while "mmap" creates memory mappings between the virtual address space and files, devices, or anonymous memory regions.

Of course the decision would depend on the specific requirements of the application, but for general-purpose dynamic memory allocation within a process, "malloc" is simpler and more commonly used. But, "mmap" is more flexible and provided more control over memory mappings, making it suitable for specialized use cases such as managing large memory regions efficiently.

2.2. MyFree Call

In the 'MyFree()' function, I used a custom written method of releasing memory instead of the standard 'free()' call. The method involved marking the memory block as free by updating its status within the memory list and then merging adjacent free memory blocks.

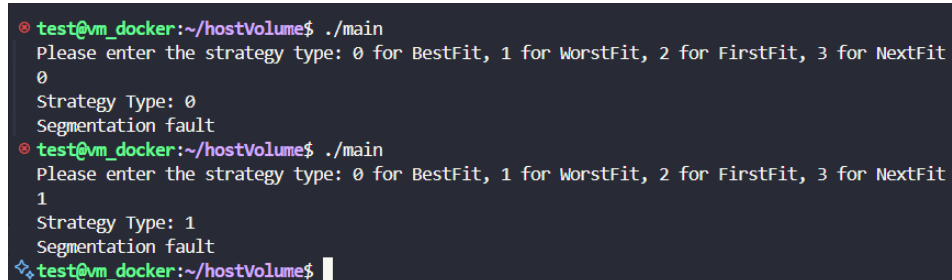
I have not extensively used a system call in this function to compare with the 'free()' call. Additionally, While this custom method I wrote may achieve the goal of releasing memory, it may not be as successful or as safe as the 'free()' function in all scenarios as the 'free()' function is extensively tested, optimized and used.

2.3. Compiling The Code

The code can be compiled with the command: 'gcc main.c mymalloc.c -o main -Wall -Werror'. The .c specifications are the c files that should be compiled, and the argument with specified with '-o' identifier is the name of the output file. '-Wall -Werror' specifiers are there to ensure we get notified with all the errors and warnings in error format. Then the compiled output can be executed with the following command: './main'

3. Outputs

Unfortunately, I could not get past segmentation faults, which resulted in a lack of screenshots to show the outputs of the processes :(



```
test@vm_docker:~/hostVolume$ ./main
Please enter the strategy type: 0 for BestFit, 1 for WorstFit, 2 for FirstFit, 3 for NextFit
0
Strategy Type: 0
Segmentation fault
test@vm_docker:~/hostVolume$ ./main
Please enter the strategy type: 0 for BestFit, 1 for WorstFit, 2 for FirstFit, 3 for NextFit
1
Strategy Type: 1
Segmentation fault
test@vm_docker:~/hostVolume$
```

Figure 3.1: Segmentation Output