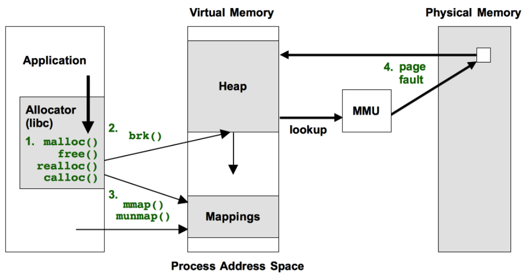
BLG312E Computer Operating Systems Homework 2

Mohamad Chahadeh 150220901
Department of Computer Engineering
Istanbul Technical University
Istanbul, Turkiye
Email: chahadehm22@itu.edu.tr

***Abstract*—In This homework, custom Memory API functions like the ones available in standard library in C were implemented and tested. moreover, different allocating strategies were used and block list for free and allocated memory regions were defined and tested.**

I. Introduction

In the standard library for the C language, functions and macros for memory allocation and de-allocation are predefined or handled by the compiler to make the job easier on the developer and to provide a safe environment in the code. however in this Homework, We defined our own functions to first create a new heap in memory, allocate memory on that heap using four different memory allocating strategies *(First Fit, Next Fit, Worst Fit, Best Fit)*.



II. Code Explanation

A. Headers

1) *Includes*:
The following are the libraries included in this project:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/mman.h> // for mmap function
#include <stdbool.h>  // for boolean data type
```

2) *Definitions*:
First, I need to define a `typedef struct` for the header of each memory block, this struct should contain three variables that tell us everything we need about the memory block, these variables are the size of the heap, the next block's pointer, and if it's free or not. I also define a pointer for the block list and a pointer to always point to the start of the heap, and another variable for the heap size. then I define the prototypes the four functions I have.

```c
// Structure for the memory block header
typedef struct block_header {
    size_t size;
    struct block_header *next;
    bool is_free;
} BlockHeader;

// Global variables
BlockHeader *block_list = NULL;
void *heap_start = NULL;
size_t heap_size = 0;

// Function prototypes
int InitMyMalloc(int HeapSize);
void *MyMalloc(int size, int strategy);
int MyFree(void *ptr);
void DumpFreeList();
```

3) *Functions*: the following four functions are defined:

a) `InitMyMalloc` : This function is used to allocate a new heap in the memory, this function uses the `mmap()` system call. once the heap is allocated, its pointer is put in both the `heapstart` and the `blocklist` functions to point to, at the beginning the list will consist of one empty block holding the entire memory region as shown below.

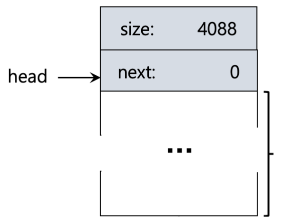


Figure 1. a heap region initialized using mmap()

b) `MyMalloc`: a function to allocate memory regions on the newly create heap, this function uses simple pointer arithmetics to choose the block to allocate from based on one of the following strategies:

- **Best Fit:** Allocates memory to the smallest block that is large enough to accommodate the requested size, minimizing wastage.
- **First Fit:** Searches for the first available block of memory that is large enough to accommodate the requested size and allocates it.
- **Next Fit:** Similar to First Fit but starts the search for the next available block from the last allocated position, reducing fragmentation.
- **Worst Fit:** Allocates memory to the largest available block, which may lead to more fragmentation but can accommodate larger requests.

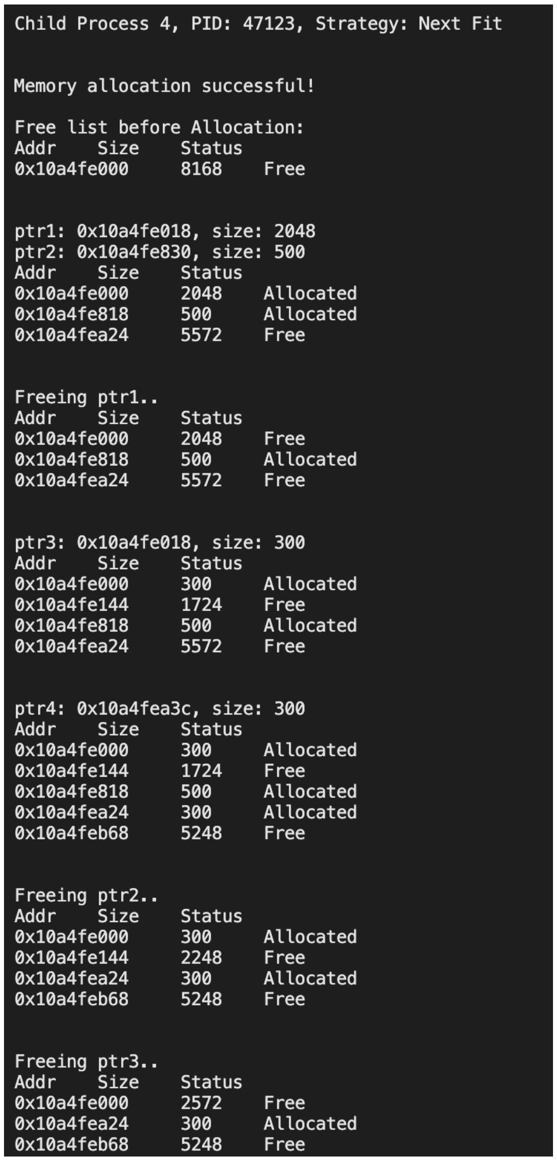
after finding the suitable block, the function splits the block by changing the size of the block to the requested size, setting the free variable to false, and then creating a new header for the rest empty memory region.

c) `MyFree`: this function frees takes a pointer in the heap and frees its region, then runs a loop that merges any adjacent block together. a free a region in memory is simple, all you have to do is mark the blockheader at `ptr - sizeof(BlockHeader)` as free, this simple implementation is made possible by the fact that we keep the allocated block in the list as well rather than being solely for free blocks. for the merging part, we start at the head of the list, and then check every block and it's next neighbor, if they are completely adjacent, we add the next block's size (plus the size of its header) to the previous block's size, and assign the next's next as the first block's next.

d) `DumpFreeList`: for debugging purposes, prints out each memory block's address, size, and status (if free or not).

III. Demo and Results

For demonstrating the functionality of my implementation, I implemented three processes that would initalize a heap and allocate to it each with a different strategy, the figure shows a sample of the output for one process using Worst-Fit as a strategy.



```
Child Process 4, PID: 47123, Strategy: Next Fit


Memory allocation successful!

Free list before Allocation:
Addr    Size    Status
0x10a4fe000     8168    Free


ptr1: 0x10a4fe018, size: 2048
ptr2: 0x10a4fe830, size: 500
Addr    Size    Status
0x10a4fe000     2048    Allocated
0x10a4fe818     500     Allocated
0x10a4fea24     5572    Free


Freeing ptr1..
Addr    Size    Status
0x10a4fe000     2048    Free
0x10a4fe818     500     Allocated
0x10a4fea24     5572    Free


ptr3: 0x10a4fe018, size: 300
Addr    Size    Status
0x10a4fe000     300     Allocated
0x10a4fe144     1724    Free
0x10a4fe818     500     Allocated
0x10a4fea24     5572    Free


ptr4: 0x10a4fea3c, size: 300
Addr    Size    Status
0x10a4fe000     300     Allocated
0x10a4fe144     1724    Free
0x10a4fe818     500     Allocated
0x10a4fea24     300     Allocated
0x10a4feb68     5248    Free


Freeing ptr2..
Addr    Size    Status
0x10a4fe000     300     Allocated
0x10a4fe144     2248    Free
0x10a4fea24     300     Allocated
0x10a4feb68     5248    Free


Freeing ptr3..
Addr    Size    Status
0x10a4fe000     2572    Free
0x10a4fea24     300     Allocated
0x10a4feb68     5248    Free
```

IV. Conclusion

In Conclusion, my implementation of the memory API worked as expected and behaved normally without issues or bugs, now to answer the following questions:

1) **What are the main differences between "malloc" and "mmap"? If you had to make a decision between these two, which would you choose? Why?**
   `malloc` is generally used for dynamic memory allocation within a single process, and used to allocate variables on the heap created automatically by the compiler, while `mmap` can be used to allocate new heap separate from the original one on memory and for mapping files or devices into memory, often for inter-process communication. I would choose `malloc` for general memory allocation needs in a process, and `mmap` for specific requirements like sharing memory between processes or mapping files into memory.
2) **What was the method/call you used in MyFree() instead of free()? Do you think it is as successful as free() at correctly and safely releasing the memory back? Why?**
   The method I used in `MyFree()` involves marking the corresponding memory block as free and merging adjacent free blocks to prevent fragmentation. While similar to `free()`, its success depends on correct implementation and testing. With proper implementation, it should be as successful as `free()` at releasing memory correctly and safely, but thorough testing is essential to confirm its reliability in various scenarios.