CS 214: Systems Programming, Fall 2020
Assignment 1: ++Malloc

By Jonathan Wang and Justin Chen

1. Overview

This program simulates the C functions malloc() and free() with a large character array representing the memory heap. The program partitions a character array of size 4096 into separate blocks of specified sizes with each call of mymalloc(). mymalloc() allocates a specified amount of memory in the character array given a size value and myfree() frees a specified block of memory given a pointer to memory.

2. Features

Each block of memory allocated is assigned its own metadata block that holds the information about the block. The metadata blocks store the size of each memory block and whether the block is being used or not. Metadata is formatted in 2 bytes with the following structure:

x000 yyyy yyyy yyyy

where x is the InUse bool and y bits are the size. This allows for a max memory size of 4096 but if it needs to be scaled it could be expanded to 3 bytes of the form:

x000 0000 yyyy yyyy yyyy yyyy

In addition to assigning and unassigning blocks of memory, the program is also able to combine adjacent blocks of unused memory into larger blocks of memory that the user can allocate. For example, if there is an unused block of size x next to an unused block of size y, then they will be combined into a block of size:

x + y + [Size of metadata]

Additionally, when the program runs into errors that the library versions of malloc() and free() would run into, it is able to throw detailed errors. Errors follow the form:

[Function name] Error: [Error Type] (file: [File name] line: [Line number])

3. Usage

In the command terminal, use the following commands when running the program in Linux:

(1) **make**
(2) **./memgrind**

The program will then run all of the workloads 50 times and print their average times following the form:

Mean runtime for workload 1: "[Time in sec] seconds | [Time in micro sec] microseconds"
Mean runtime for workload 2: "[Time in sec] seconds | [Time in micro sec] microseconds"
Mean runtime for workload 3: "[Time in sec] seconds | [Time in micro sec] microseconds"
Mean runtime for workload 4: "[Time in sec] seconds | [Time in micro sec] microseconds"
Mean runtime for workload 5: "[Time in sec] seconds | [Time in micro sec] microseconds"

4. Efficiency

The time complexity of both the functions, mymalloc() and myfree(), is O(n) where n is the number of bytes in the memory array. Since the functions go through every block in memory, the worst case occurs when the memory is filled with blocks of the minimum size:

> For mymalloc() if the memory is filled with blocks of the minimum size then the program will check every block's metadata up until the end of memory. This means that the program will go through n comparisons (technically n/3 since the minimum size of each block is 1 byte plus 2 bytes of metadata, but the 3 is a constant so it doesn't affect big O)
>
> For myfree() if the memory is filled with blocks of the minimum size then the program will check every block's pointer value up until the end of memory. This means that the program will go through n/3 comparisons. Then the program will iterate again, checking for adjacent unused blocks to merge. This adds another n/3 comparisons so myfree() runs in O(2n/3) time or just O(n).

The space complexity of the functions mymalloc() and myfree() is O(1). The functions only loop through the existing data array and assign or unassign blocks of data. The only space these functions take is in the creation of the metadata, which has a constant size. So the time complexity is just the constant size of metadata or O(1).