Stacey Mui and Andrew Lee
Systems Programming
Programming Assignment 3 Readme
October 19, 2015

The goal of this assignment is to implement a malloc and free function in C language that that not only imitates dynamic memory allocations, but also checks the errors common to it. The functions are used to manage the space of a char array of 5000 bytes. In order to manage the space, a MemEntry struct was created containing pointers to neighboring structs, a one bit integer indicating if a memory space was free and a 13 bit unsigned integer indicating the size of allocated memory. Memory allocation begins with two MemEntry structs on either side of the array. When an int variable greater than 0 but less than 100 is passed into the mymalloc function, memory space from the left is allocated and returned. However, if an int variable greater than or equal to 100 but less than 5000 is passed into the function, memory space from the right side is allocated and returned. Any other case would be considered a saturation fault and an error message is printed.

So when a malloc is called for, let's say 50 bytes of space, a MemEntry struct pointer will first go to the MemEntry struct on the left, called smallRoot. There the pointer will traverse the array by moving to the MemEntry structs next to its current one until it encounters a chunk of memory that is not only marked free by its MemEntry struct, but can also hold 50 bytes of space. If the pointer finds a chunk of memory that exceeds the needed amount, it will create a new chunk of free space to the right of the allocated chunk by splitting off the unnecessary amount of memory. The same can be said for a request that exceeds 100 bytes, let's say 500. However, the pointer would then go to the MemEntry struct on the right, called BigRoot and traverses the array backwards by accessing the MemEntry struct previous to where it is. If space is found that can hold 500 bytes and then some, the extra space will be split off from the left and will be considered a chunk of free space. This format of creating free memory chunks is used in order to prevent fragmentation by having as much free space as possible gathered at the middle of the array. If by any chance, there does not exist a chunk of free memory large enough for the request, a saturation error message is also printed.

In the free function, the argument pointer is first checked if it is null. If it is, a null error is printed. A pointer will then traverse the array to find the memory space that is requested to be free. If the space is not found, an error message is printed. Otherwise, the space is labeled as free and it is then fused with neighboring free memory spaces and the pointer that was used on it will become null. Due to the pointer traversing the array at the beginning, the worst case time of the function is linear. Although we know that there are means of making the worst case time constant, we kept the array traversal method for the sake of more understandable coding on our part.

An extra function that we also implemented is the helper function inspect heap. This method is called at the end of the malloc function and it checks if the array has memory leaks.