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

Jackson Bainbridge - Netid: jdb343
Acharya Krishnateja Vedantam - Netid: av709

Instructions

In this assignment, you will implement malloc() and free() library calls for dynamic memory allocation that detect common programming and usage errors. Malloc(size_t size) is a system library call that (hopefully) returns a pointer to a block of memory of at least the requested size. This memory comes from a main memory resource managed by the operating system. The free(void *) function informs the operating system that you are done with a given block of dynamically-allocated memory, and that it can reclaim it for other uses. You will use a large array to simulate main memory (static char myblock[4096]). Your malloc() function will return pointers to this large array and your free() function will let your code know that a previously-allocated region can be reclaimed and used for other purposes. Programmers can easily make some very debilitating errors when using dynamic memory. Your versions of malloc() and free() will detect these errors and will react nicely by not allowing a user to do Bad Things. Your malloc() function should use a “first free” algorithm to select blocks of memory to allocate.

Code Design and Explanation

Our approach to this code was to initialize a linked list and set its point in memory to the point in memory of the array of size 4096. By doing that we could set blocks of metadata right before the data that would have been stored in the memory (array of 4096) if we were asked to store data.

The metadata elements contained a short field to hold the size of data, a char field that acted as a flag for if the data was free or not, and a next field pointing to the next block of metadata.

myMalloc

To malloc memory we call onto a few helper methods the first being size_left. What size_left does is calculate the amount of memory the user can still allocate. This

ensures the user does not over allocate and instead of crashing the code we output an error message.

The second helper method in myMalloc is find_free. What find_free does is find the first free block since I used the first fit algorithm. I was hoping by making a whole other method for finding a free block the code would be easier to read and understand.

The last helper method for myMalloc is split. What split does is when the user asks to allocate a smaller size of memory than the size of the currently free blocks the code splits the larger free block into two smaller free blocks. One block the perfect size for the memory that's being asked to be allocated and another block after this now non-free block with the remaining free memory of the original free block. This process will be done many times since the linked list is initialized to have one large block of memory that is the size of the 4096 array.

The idea behind myMalloc was to first check for the following errors: if the input is 0, if the input is larger than 4096 minus the size of a metadata block(16 bytes), or if the user is asking to allocate more space than what is available. The previously mentioned method size_left is what checks for the last error. The error would occur when the user asks for say 4080 bytes (4096 - size of a metadata block) then 1 more byte, there is no more room in memory so an error is pushed out. MyMalloc also initializes the linked list being it will be the first function called every time. MyMalloc then checks the size of the memory being asked to allocate, using find_free we find the first available free block, if the first free block is the perfect fit we set the metadata's size to the size of the input, the isfree condition to 'n' indicating it is not free, and output the pointer to the actual point in memory, which comes after the metadata block. If the first free block is larger than the asked size, then we perform split to make the perfect size and now that we have the perfect size we do what we would have done if we had first found the perfect size. If neither of these conditions can be met then there is no room in memory so an error is pushed out.

myFree

myFree also has a helper method named merge. What merge does is after a block is freed and it has another free block next to it, merge combines these two free blocks for the sake of versatility. Having three free blocks, first of size 5 bytes, second of size 10 bytes, and third with the rest of the free memory is potential wasted space. With merge we would merge all these blocks back together so in the case that a block of size 5 or 10 is never needed again, those smaller blocks would not be wasted and instead all

merged back together again to obtain the original block of size 4096 so split can be called to partition off the necessary amount of memory being asked to be allocated.

The idea behind myFree is very simple. We find the pointer given to it and set its isfree flag in the metadata block to ‘f’ meaning it is free. We loop through the memory and find the pointer. Since the linked list will go through the metadata and not the actual memory we need to increment to find the actual pointer to the memory. So we loop through and compare a pointer to the linked list that has been incremented to the pointer given to us. When we find a match we decrement the pointer to get back to the metadata and set its isfree flag to ‘f’ indicating it is free. We also check in this method if we are attempting to free a block thats isfree flag is set to ‘f’ meaning the user is attempting to free an already freed pointer, if this happens an error is pushed out. If we get to the end of the memory and a match has not been found, this is the result of the user entering a pointer that was not allocated so an error is pushed out.

Here I have included a photo of my whiteboard that has evolved slowly over the course of these two weeks.