Micole Finehart
mcf155
Sridhar Sriram
sss293

Assignment 1: Memory Allocation++

The Assignment:

For this assignment, we were asked to recreate the function calls in the C library - malloc and free - by using a static char array of size 5000 named myblock (`static char myblock[5000]`). In order to satisfy the requirements of the assignment, three files were created that interacted with one another : `mymalloc.h`, `mymalloc.c`, and `memgrind.c`. Because the bulk of the assignment was done in both `mymalloc.c` and `memgrind.c`, we will spend this readme explaining `memgrind.c` and discussing the findings from `memgrind.c`.

Explanation of `mymalloc.c`

In mymalloc.c, there were three functions that held the majority of the program's functionality:

1. `metaData createNode(size_t pointerSize, char usage)`
    - *The createNode function essentially created metadata nodes that served as indicators for the use of the space following the metadata nodes. In other words, these metadata nodes held two values: the size of the space immediately following these nodes and the "usage" of this space - 'y' indicates that the user data is currently in use, while 'n' indicates that the user data is currently NOT in use.*
2. `void myfree(void*pointer_to_be_freed, char * file, int line)`
    - *The myfree function essentially searches through the* `myblock array` *and searches to see if the pointer being freed exists*
        - *If the pointer does NOT exist, then an error is printed, saying that the "*`Pointer is not found`*"*
        - *If the pointer DOES exist, then the use of the metadata node preceding the given pointer is switched to* `'n'`*, and then there are 4 cases checked in order to properly handle the memory:*
            - ***Case 1:*** *the pointer's address exists, but the space is currently NOT being used*
                - *Then an error is printed: "*`invalid free`*"*
            - ***Case 2:*** *there is unused spaced preceding the currently freed pointer and used space following the currently freed pointer*
                - *If this is the case, then the space occupied by both the metadata preceding the currently freed pointer and the currently freed pointer are added to the metadata node PRECEDING the currently freed pointer's metadata node*
            - ***Case 3:*** *there is used space preceding and unused space following the currently freed pointer*
                - *Then, the space occupied by the metadata node and the space following this metadata node following the space of the currently freed pointer are added to the currently freed pointer's metadata node's size*

Micole Finehart
mcf155
Sridhar Sriram
sss293

Assignment 1: Memory Allocation++

- ***Case 4:*** *there is unused space preceding and following the currently freed pointer*
    - *If this is so, then preceding space's metadata node =* `sizeof(metaData)*2 + currently_node size + following_node size`

3. `void * mymalloc(size_t requested_size, char * file, int line)`
    - The mymalloc function searches through the `myblock array` in order to find an appropriate amount of space to be allocated for the requested size. There are three cases for memory allocation:
        - **Case 1:**
            - There is not enough space to allocated. Then, an error is printed saying "`ERROR: No space available @ [filename] line [line number]`"
        - **Case 2:**
            - There is enough space to allocate for the memory, but not enough space in the "found" area to create a metadata node AND store another byte of memory. Then, the entire "area" is deemed "usable" for the pointer
                - Ex: `requested_size = 11`, `sizeof(metadata node) = 12`, and the only area to allocate the requested size is 13. In this case, there is not enough space to create a metadata node AND store at least 1 byte (to do so, you would need area of 24). Then, the pointer can write in an area of 13
        - **Case 3:**
            - The other scenario in Case 2 is true. Then, a metadata node is created storing the remaining size in the area, and the address of a space for EXACTLY the requested size is returned

Findings from `memgrind.c`
One point that we did want to address was that there was a SIGNIFICANT difference between the average times of the workloads for test D when run on our personal machines versus the ilab machines. We think that this difference can only be attributed to the mass amounts of people accessing the ilab machines simultaneously.

Test	**Average Time on Personal Machines**	**Average Time on iLab Machines**
Test A	432 microseconds	293 microseconds
Test B	37 microseconds	17 microseconds

Micole Finehart
mcf155
Sridhar Sriram
sss293

Assignment 1: Memory Allocation++

Test C	1060 microseconds	831 microseconds
Test D	4300 microseconds	100331 microseconds
Test E	8 microseconds	9 microseconds
Test F	18 microseconds	13 microseconds

Division of the work:

The work on this project was fairly distributed. While there were times when tasks were delegated, the both of us ended up working on virtually every aspect of the assignment.

Problems:

The major issues that we were faced with were figuring out the functioning of free, proper allocation of memory, and the relationship between the metadata nodes and the user data.