CS 214 Systems Programming
Assignment 1: ++Malloc
Rithvik Aleshetty: rra76
Steven Nguyen: shn27

Assignment Summary

Throughout class and lectures, it was been mentioned several times how malloc and free are not intelligent on their own. Unfortunately, this means that errors while using the two functions are not detected. This can cause some costly errors which can go undetected. This project involves simulating memory using a static array of 4096. The requirements included building a better malloc and free which contain the basic functionality of both functions as well as detecting user errors.

Our Solution

Metadata

Metadata was required in order to store information about the memory allocated for the user. For this, we used a struct to simulate a node. The two fields of the struct were **used** and **size** which were both unsigned short integers. The size of the metadata was **4 bytes**. The size field was the number of bytes until the next node or the end of the memory depending on if there are other nodes. The used field was the number of bytes the user has allocated using malloc. Below is the idea.

Size: 94 Used: 90	90 bytes	Size: 84 Used: 80	80 Bytes

myMalloc

The malloc method was built to detect user errors. At first it checks if the user is trying to allocate 0 bytes or if the amount of memory the user is asking for is beyond the capacity of our static array. If it catches either error, the proper message is displayed. The malloc method checks whether it is the first time it is being called by checking if the node0 (our first metadata node) has been created or not. If it has not been created, it is then created. From here this is the basics of our allocation while loop:

- Make sure there is enough space in our current block
    - Make sure it is empty
        - Allocate the space
    - If it is not empty but there is still enough space to squeeze new node before next node:
        - Allocate the space here
    - If there is not enough space between the nodes:
        - Continue to the next node if there is one
        - If no next node: show error message saying no more space

CS 214 Systems Programming
Assignment 1: ++Malloc
Rithvik Aleshetty: rra76
Steven Nguyen: shn27

myFree

The implementation of the free method was where the bulk of the error checking took place. At first the method checks whether the user tries to free something when they have not even called malloc yet. This is done by checking if our metadata has been created or not. If they have not called malloc, then the proper message is displayed. The method also checks if the user is trying to free space not in bounds of the array. This means that they are freeing something that they shouldn't be freeing. In our free algorithm we set up **3 nodes**. One of these nodes is for the metadata of the node before the one we are attempting to free (if it exists). One is the node of the memory address the user is trying to free and the third one is the node after the one we are freeing. The reason for the three nodes is so that the nodes can be **merged** if it is possible. If the node before or after is not available to be merged, the used field of our current node is set to 0 so it can be merged in the future or so we can check for the error in which the user tries to free something twice or something that has not been allocated.

Additional Workloads

Workload E:

- Malloc 1 byte 150 times
    - Free 0, 4, 8, 12, …
    - Free 1, 5, 8, 13, …
    - Free 3, 7, 11, ...
    - Free 2, 6, 10, 14, ...
- Why we chose this workload:
    - Since our implementation of free involved merging the nodes, a good test was to try all the ways in which nodes could me merged. The first free listed above tests the case in which there is no merging. The rest of the tests try left, then right, and then both respectively.

Workload F:

- Malloc 1 byte 150 times
    - Free bytes 0 to 148 inclusive starting from 148
    - Free 149
- Why we chose this workload
    - The freeing of bytes 0 to 148 tests merging left from a node that is already merged left
    - The freeing of 149 tests merging left into a node that is already merged left

Timings

Average time for Workload A: 0.000007 seconds

CS 214 Systems Programming
Assignment 1: ++Malloc
Rithvik Aleshetty: rra76
Steven Nguyen: shn27

Average time for Workload B: 0.000048 seconds

Average time for Workload C: 0.000003 seconds

Average time for Workload D: 0.000004 seconds

Average time for Workload E: 0.000076 seconds

Average time for Workload F: 0.000095 seconds

Memory Efficiency

Our simulated linked list had metadata of 4 bytes for each node. Rather than having more fields within our struct for the next or previous part of the data, doing the math using our size field was enough.