A Better Malloc and Free

Team: Derek Schatel (dschatel) and Monica Ho (mwh66)

Implementation:

For our version of malloc and free, we decided to go with an implementation that allowed for the most amount of allocations possible. To accomplish this, malloc takes the size and stores the value in a short int, then uses bit shifting and masking to attach an allocation flag to the data (0 for unallocated, 1 for allocated). The max size of the array is 5000 bytes, and this number plus an allocation bit can easily fit within a short int, thus limiting metadata to 2 bytes total.

After creating the metadata, malloc begins at the first index in the memblock array and checks the metadata there to see if it is sufficient to hold the requested allocation. If it is, malloc will calculate the remaining block size and store the remaining size in a second piece of metadata. Malloc then overwrites the old metadata with the data from the requested allocation, steps forward in the array past the allocated size, and creates a second two-byte metadata block in that space using the remaining size data.

The free implementation simply retrieves the location in the memblock of the pointer, sets the allocation flag to 0, then systematically runs through the whole array and merges any blocks that it finds next to one another with 0 allocation flags.

This implementation sacrifices speed for size. In total, in a 5000 byte array, malloc can allocate 1,666 1-byte requests before running out of memory.

Findings:

Workload times undoubtedly differ based on the system running the workload. For purposes of generating results, we ran our workload on the iLab machines at cd.cs.rutgers.edu. Prior to each iteration of the workload (i.e. when running each test 100 times, between each one of these 100 times) we re-initialized the array to be 'clean' so that each test could run on a fresh memblock. Using the six workload tests (workloads E and F are detailed in testcases.txt), we received the following data:

Average Time for Test One: 0.053647 seconds
Average Time for Test Two: 0.060480 seconds
Average Time for Test Three: 0.009105 seconds
Average Time for Test Four: 0.109920 seconds
Average Time for Test Five: 0.126659 seconds
Average Time for Test Six: 0.120201 seconds

Overall, it appears that despite this implementation being worse for time, the average execution times are relatively low. For example, test five taking 0.12 seconds to execute 5000 mallocs and 5000 frees seems relatively quick in the long run.