Kenneth Zhang and Kevin Pei

CS214: Systems Programming, Spring 2017
Assignment 1: Memory Allocation++

Function

This program replaces the default C malloc and free functions with custom mymalloc and myfree functions. The mymalloc and myfree functions are designed to be more designed than the default malloc and free and detect common programming errors such as freeing a pointer twice.

Structure

The mymalloc and myfree functions are contained in mymalloc.c. mymalloc.h contains various definitions, such as #define boolean char, and the definition of MemoryData. mymalloc.h also includes the definition of the size of memory, which is 5000 bytes in this instance. memgrind.c contains the various tests of mymalloc and myfree.
mymalloc linearly iterates through memory to find the first available free spot, while myfree linearly iterates through memory to find the matching address of the pointer being freed.

Memgrind

memgrind.c contains 6 different tests of the efficiency and robustness of mymalloc and myfree.
The first memgrind is 1000 malloc calls of 1 byte, followed by freeing them one at a time.
The second memgrind is 1000 separate calls of malloc of 1 byte followed by freeing it.
The third memgrind is calling either malloc or free at random, for a total of 1000 malloc calls of 1 byte and 1000 calls to free to free all pointers.
The fourth memgrind is calling either malloc or free at random, for a total of 1000 malloc calls of 1-64 bytes randomly and 1000 calls to free to free all pointers.
The fifth memgrind consists of various improper malloc and free calls, such as trying to malloc negative memory or freeing a dereferenced pointer.
The sixth memgrind first calls malloc for byte sizes that increase in powers of two, followed by freeing every other pointer and then calling malloc for blocks of 1 byte. Finally, all blocks are freed sequentially.

Efficiency

memgrind1: 5095 microseconds

memgrind2: 32 microseconds

memgrind3: 23215 microseconds

memgrind4: 1841393 microseconds

memgrind5: 51 microseconds

memgrind6: 15814 microseconds

memgrind1 through memgrind 4 all malloc'ed 1000 times and freed 1000 times. However, memgrind2 was drastically more efficient than the other memgrinds. We believe this is because the other memgrinds all had to not only linearly search through memory, but also had to print error messages. All of the other memgrinds were much more prone to not having enough memory to allocate new variables. Particularly interesting is memgrind4, which took almost 100 times longer to run than memgrind3 despite having the same concept behind them. We think this is because memgrind4 relied much more on random chance and was more prone to having no memory available when malloc was called due to the larger amounts of memory being called. Merging of the freed memory blocks may have also contributed to the longer run time. memgrind5 was longer than memgrind2 despite having much fewer calls to malloc and free, though we suspect this is because it had to print error messages, supporting the notion that printing error messages contributes to run time.

Notes

When making memgrind, you will encounter many warnings about not casting the value of malloc. Ignore these warnings, the program runs fine. We have tried casting and changing the types of the variables we store the malloced memory in, but it does not remove these errors. They do not impact the functionality of this program.