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.

MemoryData

MemoryData represents the metadata for a given block of memory. Whenever malloc is called, it not only allocates the given amount of bytes, but also creates a MemoryData struct to represent its metadata. MemoryData holds the memory block's size and whether it's free or not. It also holds next and prev pointers for iteration and merging adjacent free memory blocks.

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: 39961 microseconds
memgrind2: 25 microseconds
memgrind3: 144 microseconds
memgrind4: 171 microseconds
memgrind5: 49 microseconds
memgrind6: 81 microseconds
memgrind1 through memgrind 4 all malloc'ed 1000 times and freed 1000 times. However, memgrind1 was drastically less efficient than the other memgrinds. We believe this is because memgrind1 had to print the most error messages. While memgrind2 never printed error messages and memgrind3 and memgrind4 would only occasionally print errors for not having enough space in memory, memgrind1 would always print hundreds of error messages for not having enough space or for having no such variable be allocated. memgrind5 took a similar amount of time to execute as 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. Similarly, memgrind6 had a low run time due to not needing to print many error messages.

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.