Brandon Smith and Asad Dar
Asst1: A better malloc() and free()

README

mymalloc.c

This program is a custom implementation of malloc() and free(). malloc() takes an int and returns a chunk of memory allocated to that size. The listmem() method allows us to peek into the memory block from another source. This function lists all the allocated and unallocated blocks. free() takes a pointer and attempts to free the memory allocated at the address. For this implementation we have only allowed a certain amount of bytes to be allocated total. If the user's request for malloc exceeds this value the return value will be 0 with a message regarding the issue. Likewise if a user tries to free a pointer not associated with an allocated address the program will return an error. The behavior of malloc(0) is to return 0 or NULL. Also if there if only enough space to store the metadata for another pointer with a size of 0, the chunk before it will extend to fill this space. The structure of a block of memory is as follows.

start----------
short int size(2 bytes)
data start----------
data of length size
end----------

With our implementation of malloc you can only malloc even sizes. The reason for this is that in the metadata for the blocks we are only using a 2 byte short int. To reduce size of the metadata we are using the least significant bit to store if its allocated or not. This means that odd numbers will not be able to be malloced. To get around this if the user requests an odd size we will add 1 to it.

USAGE:
malloc(size_t size)
free(void * ptr)
listmem()

memgrind.c

This file is used to test our malloc and free implementations through various workloads. It performs the workloads 100 times and computes the average time for each workload.

Memgrind workload times

==========WORKLOAD A==========
Average time: 15700µs
SIZE: 4998 ALLOCATED: 0
SIZES MATCH

==========WORKLOAD B==========
Average time: 68µs
SIZE: 4998 ALLOCATED: 0
SIZES MATCH

==========WORKLOAD C==========
Average time: 1225µs
SIZE: 4998 ALLOCATED: 0
SIZES MATCH

==========WORKLOAD D==========
Average time: 7382µs
SIZE: 4998 ALLOCATED: 0
SIZES MATCH

==========WORKLOAD E==========
Average time: 1208µs
SIZE: 4998 ALLOCATED: 0
SIZES MATCH

==========WORKLOAD F==========
Average time: 8005µs
SIZE: 4998 ALLOCATED: 0
SIZES MATCH

Findings: It seems that workload A takes a very long time. This is because each time we allocate a new block we need to scan all the way to the end. Analyzing this it takes n(n-1)/2 operations to allocate all the bytes. Each time it allocates it needs to iterate to the end. If we were using a free only list it would be much quicker. Workload B is very quick. This is mostly because the max size of allocated is 2 bytes since my implementation makes odd sizes even. Since the max is 1 allocated byte the malloc does not need to work very hard to find the next free block. Since workload C and D seem so similar in execution but much different in average time that may raise questions. From analyzing A we know that it takes n^2 average time to scan all blocks of size 1. Now that 1/3 is taken up we need to do (2*n/3)^2 operations. Ths is a much smaller number.