Error Detecting Malloc() and Free() Project Documentation

Jimmy Le[1], Carlos Dominguez[2]
Rutgers University
[1]jl1415@rutgers.edu
[2]cmd363@rutgers.edu

Error Detecting Malloc() and Free()

For this project, we have created three files, main.c, malloc-free.c, and malloc-free.h, that will take care of error checking for memory usage and memory leakage for any data set and any number of inputs from the program.

Functions

*We have identified some of the core components of our implementation for the program.

malloc-Free.c

allocate(int size, char IS_LARGE)

For this method, we first receive an input from the user, and check with a message from the user if the input is large (usually referred to as a n for no or y for yes. If n, then we go over with an array to track the array size and check for the chunk of memory that we will create for the input. Once the tracking is done, and we allocated the space, we return size. If IS_LARGE is not n, then we still track for the array location and its input with a char array, but its not at the beginning of our tracking system. The run time analysis regards to this method is O(n) depending on the initial input by the user.

my_malloc(size_t t)

For this method, we initially error check for the following cases such as checking for maximum memory usage (out of memory), and not enough memory for the allocation process. Each "not enough memory.." error checks using different methods such as too big allocation, too small memory size or too large memory to allocate. The run time analysis regards to this method is 0(n) depending on the users size_ t of t.

my_free(void* ptr)

Initially notifies the user of the address of the ptr and the address of the memory, depending on the input. We also error check for the size_ of_ chunk array and the is_ alloc array. Once all error checking is finished, this method will create an offset for memory to be used as a tracking system for the array. The run time analysis of this method is $0(n^2)$ depending on the array and pointer.

print_memory()

This is user created, and all it does is print out the memory's size, and the is_ alloc: characters in the array, and size_ of_ chunk: decimals of the array. This runs at a 0(n) time for printing, for all arrays.

Using Our Test Cases

To use our test cases, run either 'make prog1' or 'make prog2'. ./prog1 will run the test cases checking the errors outlined in the PDF–freeing unmalloc'd pointers, freeing an offset of a malloc'd pointer, and repeatedly freeing a pointer. All three cases work just fine. The second case, testing out data fragmentation, works as

expected in fragmentation handling, but for some reason is not able to free the second pointer. We suspect a one-off error that may manifest itself in further testing.