Vamsi Kurkal and Armin Grossreider

Our program uses the process heap to allocate memory and catch common programming errors.

First, the program checks to see if the heap is in use. If not then it sets the program break.

Then, if heap is in use, the program checks to see if there is already a free memory block that’s size is greater than or equal to the user’s needs. If so, it returns that mem block and prints “mem reuse”. This search occurs linearly with O(N) time, N being the number of iterations the program must take until it finds a big enough block, or if no suitable blocks found, the number of total blocks in memory.

If not, than the program grows the heap by the necessary amount and returns the pointer to the allocated space. Than it prints “malloc success”. The actual act of allocating space is O(1);

The free method goes to the pointed data, goes traverses along the bytes up by the distance of one memstruct, and checks if the resultant pointer is pointing to a valid block of data. If it is, the data is erased, the memstruct’s flag is set to 1 and “free success” is printed out. All of this takes O(1) time.

The program organizes memory as blocks of memstructs.

Each memstruct contains a free flag which designates free status, a size, a pointer to the next memstruct, a special code field, and a data pointer.

The special code is there to make sure that whenever I deal with a memstruct, I am actually working with one. If I get a pointer to an invalid memstruct, the special code will be a random set of numbers, not the code I designated, meaning that I am not looking at a memstruct. This is how the free method looks for valid blocks. The next field allows the malloc method to traverse memory easily and look for a block that meets the user’s needs.

By increasing the heap size with the brk() and sbrk(), you treat memory as contiguous segments. So, the saturation limit of this malloc method is dependent on the resource limit set by the OS. Fragmentation is mitigated somewhat by re-using old freed blocks.

The biggest bottleneck is finding a previously freed memstruct that can be sent to the user. That search degrades the efficiency of this program to O(N)