Systems Project 1.0: A better malloc() and free()
Stephen Petrides & Andrew Macneille
6 October 2016

All memory to be allocated is kept in a static array of size 8192 bytes.

Buddy Allocator:

Divide memory into partitions to satisfy the requests for memory.
Every block in the memory system has an order from 1 to 13, making the total memory size 8192 bytes. The first byte of every block is dedicated to meta data which stores the level of the block, contained in the 6 higher order bits, whether or not the block is a left or right buddy, and whether or not the block is allocated.

Splitting algorithm for the Buddy Allocator:

A request for memory is received for k number of bytes. The ceiling of the log base 2 of k is calculated; this is the order of the requested number of bytes. The array is searched for any block of that order, the first block found is allocated and the meta data of that block is updated (allocated bit is set). If no block of that order is present a block of higher order is split into two, decreasing their capacity to hold bytes, in order to satisfy the request.

Our implementation consists of two major methods, mynewmalloc() and free(), which are implemented using various helper
methods such as, jump_next(), jump_back(), merge() and isLeft().

The three methods, jump_next, jump_back and isLeft() are implemented using bit shifting. The purpose of jump_next() is to return the position in the memoryblock of the next position that meta data is held at in the array in consecutive order moving from left to right. The purpose of jump_back() is to return the position in memory block of the position of meta data that is held exactly one block behind the current block, moving from right to left.

You can view the order of the memory as a heap structure. The jumping methods allow us to traverse the heap in log(n) time by jumping to the exact location of the buddy block, or the neighbor block in the array, which could be one level higher or lower. The jump methods also allow us to merge() up two unallocated blocks in log n time.

merge() is implemented by checking whether or not one block and his buddy is allocated and if not, deleting the meta data of the right buddy and updating the level of the left buddy to one more than the current level that it is on.

Output:

```
Time Elapsed test0: 0.051200 secs
Time Elapsed test1: 0.001800 secs
Error: memgrind.c.77
```

```
Time Elapsed test2: 0.000100 secs
Error: memgrind.c.107
Time Elapsed test3: 0.000300 secs
Time Elapsed test4: 0.003900 secs
Time Elapsed test5: 0.023000 secs
```

Our fastest tests are C and D while out slowest cases are for A and F. We believe our fastest cases are working because our heap search algorithm can easily find the next block to be allocated. However, when blocks need to be broken down and built back back up this takes more time and therefore tests A and F require more time.

There are errors in test C and D because these fill up the heap before memory is freed so that more memory can be allocated.

It is very surprising the difference between the tests because they are allocating the same amount but at different times.