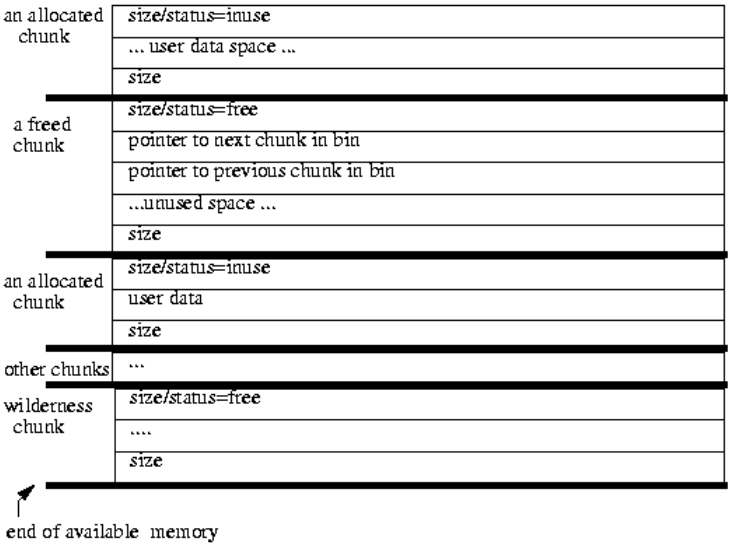
User Level Memory Manager

Serena Hogan

The goal of this project is to create an user level memory manager in C by implementing alternative versions of malloc and free, which will be called mymalloc and myfree. These functions will be responsible for managing memory allocated on the heap and have the same APIs as malloc and free defined in the C header stdlib.h.

This implementation of malloc will follow some of the design principles used in buddy memory allocation and dlmalloc. Available memory will be grouped into chunks. Each chunk will have a tag specifying whether the chunk is valid (i.e. in use) and will contain the size of the chunk (represented as a 4 byte integer) in a field at the beginning and the end of the chunk to allow for traversal of the chunks in both directions. All the free chunks will be sorted by size into bins of varying powers of 2. Each bin will be a doubly linked list of chunks of the same size. Mymalloc will have to keep track of the head and tail of each of these bins. To minimize the memory overhead of this manager, the linked list can be implemented using the memory of the free chunks themselves by storing pointers to the next chunk and previous chunk in the bin in each free chunk.



[1]

When a process requests x bytes, mymalloc will first check if there are any free chunks big enough to store x bytes. If there is not a big enough contiguous chunk, mymalloc will use the mmap system call to allocate additional memory from the kernel. To minimize the overhead of calling mmap, mymalloc will allocate four pages at a time each time it needs to allocate new memory. If the memory requested is larger than four pages, mymalloc will adjust its request to mmap to accommodate the requested memory. Once there is a big enough chunk to store the x

[1] https://gee.cs.oswego.edu/dl/html/malloc.html

bytes, mymalloc will check whether the x bytes could fit in a chunk half the size. If it can, mymalloc will split the chunk in half and update the necessary bins. Mymalloc will continue this process until it can no longer split the chunk into halves big enough to fit the x bytes.

In order to free memory allocated, myfree will go to the address pointed to by the passed in pointer minus 4 in order to find the size of the allocated chunk. Myfree will then mark the block as invalid. Myfree will also check if this chunk has a contiguous chunk that is its buddy (i.e. the other half of the bigger chunk it was originally split from). If it has a buddy, it will be merged with it, updating the bins as needed. This process will continue until the chunk has no contiguous buddy it can be merged with.