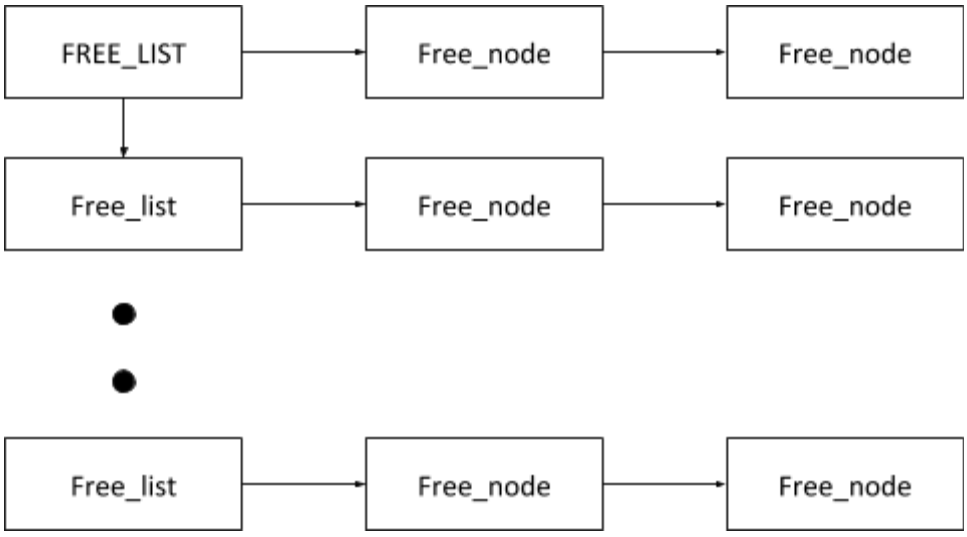
Beta Writeup

Arjun Gupta and Alhamzah Alnufaili

Current Strategy Summary



- Header

We augment each block with an 8 byte header that should contain the block's size and a "clean" bit that represents the block's state: used or free.

- Binned Free List

Our key strategy for the beta was to use a free list as effectively as possible. We did this by effectively implementing the binned free list seen in lecture. The key idea is that our free list is a linked list of linked lists. The top level list contains nodes that are effectively the "bins" in the free list which have sizes of 2^n and are organized in sorted order. Each of these nodes then has a child list that contains other memory locations that have the same size. All of these objects are contained in blocks of free memory, and no external memory is allocated to hold them. We only ever allocate memory of size exactly 2^n to simplify the process and find that it gives us better performance on average.

- Node search and Node Splitting

A lot of the time, we do not have the exact size node in our free list to hold the value of the requested size to the nearest power of two. In this case, we search up the list until we find a node that has a size greater than the necessary size and we then split the node into pieces of as large a size as possible (but still powers of 2) and place the extra space back into the free list. This optimization saves us a lot of space.

In Progress

- Garbage Collection/Coalescing

We thought that the next optimization step would be to implement garbage collection. Since the pointers given to the user can not be changed, we figured that using a mark and sweep procedure, where we mark the blocks to be freed in free and coalesce contiguous free blocks once a condition is met and rebuild the binned free list. The condition in our implementation was the number of frees reaching a threshold, and we stop merging once the block's size exceeds and size parameter.
Since we did not manage to implement variable size allocation in time for the beta, we were restricted to only coalesce blocks of the same size (so they add up to a power of 2).

We did not see *util* improvement with powers of 2 coalescing and our performance was still 100, so we expect garbage collection to improve our score once we get it running with variable sized allocation. We also plan on experimenting with different condition before garbage collecting, e.g. the ratio between free and used blocks.

Abandoned

- Variable size allocation (not just powers of 2)

We thought that allocating exactly enough space per block for requested mallocs would save us a lot of space. We would then store freed blocks into the free list where $2^n < size < 2^{n+1}$ (the lower bound on the size). In practice this did not work very well since there were issues with lower bounding the power and then not having the correct size bin in the free list to accommodate future requests. We may come back to this in the final when we get garbage collection working since the garbage collection algorithm may allow for far superior space utilization in this case.

Expected Performance

(throughput)	filename	libc	base	my	(util)
	trace_c0_v0	6373	7010	46903	100%	97%
	trace_c1_v0	29147	32061	65600	100%	72%
	trace_c2_v0	2403	2644	24157	100%	74%
	trace_c3_v0	13097	14407	56549	100%	35%
	trace_c4_v0	16634	18298	59438	100%	62%
	trace_c5_v0	40003	44003	63244	100%	79%
	trace_c6_v0	23070	25377	31642	100%	65%
	trace_c7_v0	21893	24082	27746	100%	31%
	trace_c8_v0	13609	14970	14385	96%	46%
	trace_c9_v0	32515	35767	49268	100%	29%

GeometricMean(54.731477 (util), 99.602141 (tput)) = 73.833409