Free-Space Management

Annalise Tarhan

February 10, 2021

1 First run malloc.py with the flags -n 10 -H 0 -p BEST -s 0 to generate a few random allocations and frees. Can you predict what alloc()/free() will return? What do you notice about the free list over time?

```
ptr[0] = Alloc(3)   Return: 1000   List: [1003, 97]
Free(ptr[0])        Return: 0      List: [1000, 3], [1003, 97]
ptr[1] = Alloc(5)   Return: 1003   List: [1000, 3], [1008, 92]
Free(ptr[1])        Return: 0      List: [1000, 3], [1003, 5], [1008, 92]
ptr[2] = Alloc(8)   Return: 1008   List: [1000, 3], [1003, 5], [1016, 84]
Free(ptr[2])        Return: 0      List: [1000, 3], [1003, 5], [1008, 8], [1016, 84]
ptr[3] = Alloc(8)   Return: 1008   List: List: [1000, 3], [1003, 5], [1016, 84]
Free(ptr[3])        Return: 0      List: [1000, 3], [1003, 5], [1008, 8], [1016, 84]
ptr[4] = Alloc(2)   Return: 1000   List: [1002, 1],[1003, 5],[1008, 8],[1016, 84]
ptr[5] = Alloc(7)   Return: 1008   List: [1002, 1],[1003, 5],[1015, 1],[1016, 84]
```

The free list grows increasingly fragmented over time due to the absence of coalescing.

2 How are the results different when using a WORST fit policy to search the free list? What changes?

With worst fit, part of the last, never allocated block is always allocated instead of reusing a smaller, previously allocated block when possible. The result is a free list with larger blocks stretching further into the unallocated space.

3 What about when using FIRST fit? What speeds up when you use first fit?

For this example the results for first fit and best fit are the same. The difference is that with first fit, the allocator didn't need to check all the subsequent blocks in order to choose.

4 For the above questions, how the list is kept ordered can affect the time it takes to find a free location for some of the policies. Use the different free list orderings to see how the policies and the list orderings interact.

Best and worst fit searches return the same result for any list ordering. The only difference between list orderings is the time it takes. SIZESORT+ lists free blocks in increasing order, so the first sufficiently large block is the correct choice for best fit. SIZESORT- lists them in decreasing order, so the first block is worst fit's result, if it is large enough. The opposite is true too. SIZESORT+ is the worst option for worst fit, since it must traverse the entire list to find it's result, and SIZESORT- is the worst ordering for best fit for the same reason. First fit's result, on the other hand, is completely dependent on list ordering. When SIZESORT- was used, its result changed from being the same as best fit to being the same as worst fit.

5 Coalescing a free list can be quite important. Increase the number of random allocations. What happens to larger allocation requests over time? Run with and without coalescing. What differences in outcome do you see? How big is the free list over time in each case? Does the ordering of the list matter in this case?

Without coalescing, the free list becomes enormous over time, and larger allocations frequently fail. This causes search times to increase as well. With coalescing, the free list remains manageable, and while some allocations still fail even when the total amount of free space is sufficient, it happens far less frequently than it did without coalescing. Besides the differences observed earlier as a result of list ordering, which order the free spaces are listed in does not matter, since blocks coalesce with their actual neighbors, not the blocks they are listed next to.

6 What happens when you change the percent allocated fraction to higher than 50? What happens to allocations as it nears 100? What about as the percent nears 0?

The higher the percent allocated fraction, the quicker and more likely the memory space is to fill up, causing allocation requests to fail. As it nears 100, the space will fill up and rarely have space to allocate new blocks. As it nears 0, the space will remain sparsely populated and allocations will always be successful. (It can't actually dip below 50%, since blocks need to be allocated before they are freed.)

7 What kind of specific requests can you make to generate a highly-fragmented free space?

The worst scenario is a series of blocks being allocated, then every other one being freed. If all blocks are the same size, the choice of allocation strategy makes no difference, but if they are variable worst fit increases fragmentation. First fit with SIZESORT- has the same effect.
./malloc.py -H 0 -c -A +10,+8,+6,+2,+8,+12,+4,+6,+8,+12,+10,+8,-1,-3,-5,-7,-9