CSE 3320 Malloc Assignment

Ikechukwu Ofili

1001862556

April 14, 2023

1 Executive summary

This report documents the performance of four memory allocation algorithms (first-fit, next-fit, best-fit, and worst-fit). These algorithms are tested by writing implementations of malloc, calloc, and realloc that use the algorithms. These functions are used instead of their *stdlib* counterparts of the same name in programs that we then use to compare the performance of each allocator with each other as well as with the system malloc, calloc, etc.

Obviously, the system imple will be better than any of the algorithms used not particularly because it uses a much better allocator algorithm but mostly because of better optimization choices like growing the heap at huge increments at a time instead of every time there is no free node in the list, etc.

2 Description of the Block-finding Algorithms

There are four algorithms that this report talks about. They are:

- First fit
- Next fit
- Best fit
- Worst fit

2.1 First fit

This algorithm is probably the simplest of all the allocator algorithms. The goal is to loop through the list of memory chunks and return the first **free** chunk that is big enough for the requested size.

2.2 Next fit

Next fit is exactly like first fit but instead of searching the list of memory chunks from the beginning every time, we remember where we stopped the last time we searched for a block and we start from there. That way, we do not end up picking the same block when a supposedly "better" block could be somehere farther in the list

2.3 Best fit

This algorithm picks the block that best fits the size that was requested. In other words, it picks the smallest block in the list that has a size greater than or equal to the size requested. This reduces the number of block splits that we need to do but it also is prone to heap fragmentation since there may be various small free blocks split from the allocated blocks

2.4 Worst fit

This is the opposite of best fit. For this algorithm, we pick the biggest block in the list, as long as the size is greater than or equal to the one requested. This algorithm tries to fix the heap fragmentation problems that come with best fit with the expense of multiple block splitting

I expect best fit and worst fit to do worse time-wise since they always have to traverse the entire list each time a memory allocation takes place. However, they are better than first and worst fit in managing heap fragmentation and reducing heap size

3 Benchmarking

Test name	First fit	Next fit	Best fit	Worst fit	Libc malloc
malloc-free	0.18	0.16	0.21	0.33	0.26
ten-allocs	0.01	0.01	0.02	0.01	0.00
alternate	0.77	0.77	0.83	0.79	0.01

Table 1: Test times of libc-malloc vs my four allocator algorithms

I used GNU-time to record the time for all tests.

3.1 Discussion of Results

I was only able to write 3 tests for this benchmark but these 3 tests have results that say quite a bit about each algorithm. Especially libc's malloc

malloc-free

This is a simple program that allocates a mebibyte of memory and then

immediately frees it. It does this 10 million times. As you can see, next fit takes the cake then first fit, best fit, stdlib, and worst fit (I ran this multiple times and was getting consistent results with first and worst being the top two, libc and worst fit being the last two in that order, and best fit in the middle).

I do not really know why worst fit and best fit have different values. It might come down to actual machine instructions since for this test they **should** give the same result

ten-allocs

This program is exactly what it sounds like. It is just ten allocations in a row followed by 10 frees in a row. The purpose of this test is to see how long the actual malloc and free functions take without any sophisticated finding algorithms. To no one's surprise, my allocators have very similar time (around 0.01 to 0.02s) with libc malloc running at 0.00s everytime

alternate

This is the test with the weirdest results. All of my algortihms have similar run times just under a second (best fit and worst fit take longer because the free list is very long this time around and they have to traverse all of it. However, libc malloc only uses 0.01 seconds which is baffling to me. I don't know what magic trick this uses but it is definitely worth investigating. It also makes me think that the results from malloc-free was from the time it takes to loop 10 million times, however, most of my algorthm beats libc so that is probably not the case

4 Conclusion

In conclusion, it is very obvious that best/worst fit is the last but with the limited testing that I have conducted, libc malloc is not too far off when it comes to allocating and freeing the same spot over and over. First and next fit are very similar so there is not much to talk about. If you want to see the statistics, I have included them in the performance test directory for reference