Technical Report: Malloc

Devin Wingfield; 8917

12/01/2025

CSE3320-003

Table of Contents

Executive Summary:

In this report, I will compare five different methods to allocating memory through malloc and then deallocating that memory using free. First, I will introduce each of the memory allocation techniques that was implemented along with their pros and cons. Next, I will describe the tests I used to measure the performance of each of the memory allocation techniques and what they are testing. Finally, I will display the results I obtained from the tests along with the conclusions I gathered from them. This will end with the rankings of the four memory allocations methods and the significantly faster system malloc.

Algorithms Implemented:

The algorithms that were implemented to manage memory allocation for malloc were First, Next, Best, and Worst fit. These are the basic memory allocation techniques that any basic programmer can implement and therefore makes it easier to analyze.

First Fit(FF) is a simple algorithm that starts from the beginning of the heap and finds the first available block of memory that meets the size requirement for the requested size. Since FF starts at the beginning of the heap, this means it is easy for the beginning of the heap to become fragmented. However, this does mean it finds blocks faster than best fit and worst fit as I will explain and show later.

Next Fit(NF) is similar to First Fit in that it tries to find the first available block that can store the requested size. But, where it differs is where it starts. NF starts at the last visited block when trying to find a free and large enough block in the heap rather than the start every time. This does mean you need another variable to store the last visited block and more edge cases to deal with looping from the end to the start of the heap. This does solve, however, the problem of fragmentation in the beginning of the heap while keeping the same speed of First Fit.

Best Fit(BF) is different from both FF and NF because rather than finding the first available block of memory, it searches through the entire heap and finds the optimized block in terms of size. Meaning it finds the smallest block of memory that meets the requested size. While this will decrease the max heap size because of this optimization, it means it will always run in O(n) time complexity making it slower than FF and NF.

Worst Fit(WF) works similar to BF in that it searches the entire heap for a specific block, but differs by trying to find the block with the largest size rather than the smallest size that fits the requested size. The purpose of this is to minimize fragmentation of the heap. It still has the same time complexity of BF of O(n) since it still needs to search the entire heap to find the largest block within the heap.

Test Implementation

There are two tests that were implemented, the first is to test time to allocate and free memory. The reason for this is to analyze the difference between the implemented memory allocation algorithms and the system's memory allocation technique. The second test measures fragmentation of the heap and the max heap size. This will ultimately analyze the difference between the four implemented memory allocation techniques by ranking them in their ability to deal with fragmentation and heap size.

The first test, analyzing performance speed, uses the header time.h for the program to use timing variables to store start and stop times for using malloc and free. However, the problem with this experiment is that it runs on github codespaces which runs multiple processes constantly. Leading to a large variability of results. To counteract this, I average the results of the elapsed times over a hundred iterations of the test. This allows for more consistent results and a better basis for my conclusions.

The second test, analyzing fragmentation and heap size, uses the statistics implemented in src/malloc.c. Through the numbers provided, I can determine the ability of each of the algorithms to deal with fragmentation and minimize heap size. Additionally, after running each experiment a few times, it was observed that the results of the statistics did not change. Allowing me to take the results from one sample rather than having to run more than once. The test that I implemented to best analyze this problem consists of five steps. First is creating a heap with blocks of varying size and then deallocating half of the memory by freeing every other block in the heap. This causes fragmentation within the heap and presents the challenge for each algorithm to handle it by trying to allocate more memory of varying size. After which most of the memory is deallocated and another large allocation is requested to test coalescing and heap size. This process therefore tests the ability of each algorithm to deal with splitting, coalescing, fragmentation, and heap size.

Test Results:

Here I will present my findings from the experiments I performed and then give my conclusions based on the results from both tests. These conclusions will lead to my final ranking of each of the algorithms based on runtime and ability to deal with fragmentation.

Malloc test:

	Malloc Time	Free Time
System	127 ns	35 ns
First Fit	153 ns	42 ns
Next Fit	154 ns	44 ns
Best Fit	206 ns	45 ns
Worst Fit	209 ns	46 ns

From the Malloc Test, there are three distinct groups for malloc runtime and two groups for free runtime. The groups for malloc runtime consist of System in group one, First Fit and Next Fit in group two, and Best Fit and Worst Fit in the last group. As expected, group 1 outperformed all other memory allocation algorithms. Around 26ns from group two and ~80ns from group three. Group two out performed group three by about 50ns, which was to be expected as well. With group two's ability to not have to search through the entire heap every use of malloc, they shorten their search time by 75%. Lastly, group three took the longest again because of its need to search through the entire heap before it could return a chosen block. However, in the Fragmentation and Heap Size Test, we can see there are trade offs to the speed of groups two and three.

Fragmentation & Heap Size Test:

	First Fit	Next Fit	Best Fit	Worst Fit
Reuses	10	10	11	10
Grows	22	22	21	22
Splits	10	10	10	10
Coalesces	30	29	29	30
Blocks	1	2	1	1
Requested	20296	20296	20296	20296
Max Heap	17016	17016	11992	17016

While the results of the Fragmentation and Heap Size Test are similar across the board, there is one major talking point. Best Fit's max heap size is significantly smaller than all the others. This is supported by the fact the heap was grown one less time than the other algorithms. Showing that the theory of Best Fit minimizing max heap size is correct. The other talking point is Next Fit's coalesce and block final numbers. This is either because there wasn't enough time to update block's and coalesce's counters or because there is a used block separating two free blocks. It is hard to tell which scenario is happening without further experimentation.

Conclusion

Now that each memory allocation's technique was explored and quantified, we can begin ranking each method. First is the System's technique. With its obvious superiority in both malloc and free runtime, the System's ability to efficiently handle complex situations is proven. Additionally, if you run *ps ax -o rss,vsz,pid,command | grep "./tests/malloc"* you can obtain the virtual and resident size the program uses with the System's malloc. You'll notice that it is significantly smaller than even Best Fit. Tied for second is Best Fit and First Fit. If you are trying to optimize the size of your heap, the best option would be Best Fit. However, if you don't care about the size of your heap and are trying to go for speed then First Fit is the better option. Summarized, these two are overall the same but depending on the situation, one could be a better option than the other. In fourth place would be Worst Fit. The reasoning for this is because of Worst Fit's ability to limit fragmentation in the Fragmentation and Heap Size Test. In the end, it allowed for all free blocks to be combined, unlike Next Fit. Subsequently, Next Fit is in last place. While there is a possibility that it is just an inability to update the counters, this did not occur when running the experiment on the other techniques. Giving a larger probability to the fact that Next Fit did not allow for the coalescing between the last two free blocks.

In the end, this report analyzed and conveyed its findings about five different memory allocation techniques. It was found that the System's technique was superior above the rest in both the time it took to run and the max heap size the program used. Additionally, the report was able to rank each of the four subsequent techniques in order of overall efficiency of runtime and max heap size using the result obtained from experimentation.