Heap Memory Management Algorithms: Performance Report

Marwan Abu Lebdeh

December 1st, 2024

1. Executive Summary

The purpose of this project is to demonstrate deep understanding of memory allocation algorithms in addition to an implementation of the system call malloc(), calloc(), realloc() and free() in C. sys malloc upon more rigorous testing seemed to be a 100 times faster than my implementation. BF algorithm was the least efficient in picking free blocks but it was the least fragmented.

2. Description of the Algorithms Implemented

2.1 malloc (Baseline)

The first thing malloc does is align the memory requested to a multiple of 4, That is to avoid alignment issues and padding issues which could slow our functions down, it then searches the heap for a free memory block and pick one based on the 4 algorithms defined below, if found the memory block will be split if it's larger than the size requested + 4 bytes, then a pointer will be returned to that allocated block on the heap.
If no free block was found, malloc will grow the heap and return the new block.

2.2 First Fit Algorithm

Linearly searches form the start of the heap until a free block with an appropriate size is found. This method is easy to implement compared to the others but could be costly since we might be searching the whole heap over and over if the free blocks are located at the end of our heap giving it a worse case of $O(N)$ with N being the size of the heap.

2.3 Best Fit Algorithm

Linearly searches the heap while choosing the smallest block that can fit the requested memory size. Could result in high external fragmentation since a lot of smaller slivers will be left that would not fit a usual requested size.

2.4 Worst Fit Algorithm

Linearly searches the heap while choosing the biggest block that can fit the requested

memory size. This will leave more memory to allocate after the block is split and should result in less external fragmentation compared to best fit. This is just a prediction as I did not find a proper study comparing both against each other extensively

2.5 Next Fit Algorithm

Linearly search after the last allocated block, if no block is found. It will linearly search the heap again starting from the start until a block is found. This assumes that the average case will be $O(m)$ with m being a subset of the heap N. the coadjacent memory block to the last allocated block would be free since the block was either split or newly allocated making it faster than first fit. At its worse this algorithm takes $O(N + m)$ making it more costly than first fit.

3. Test Implementation

4 metrics were measured

1. Time of execution

   A simple test case where the program requests a random sized memory block for 10,000 allocations with an initial heap size of 100,000 bytes, then freeing them after. Choosing a random block demonstrates how fast the different algorithms will choose a free block. An initial not so rigorous test case did not use random blocks but the same sized block 10,000 times.

2. Number of heap splits and heap growths

   The heap is fragmented by freeing every other block of 1KB in the program heap grows unnecessarily when allocations fail to reuse fragmented spaces.

3. Heap fragmentation

   The heap is fragmented by freeing every other block of 1KB in the program, observing whether the space is filled or not will test fragmentation.
   Fragmentation is calculated using the function find_frag_fraction(). This function finds the biggest free block and divides it by the free space available. The result is how much space is not fragmented. to find fragmentation the equation would be:

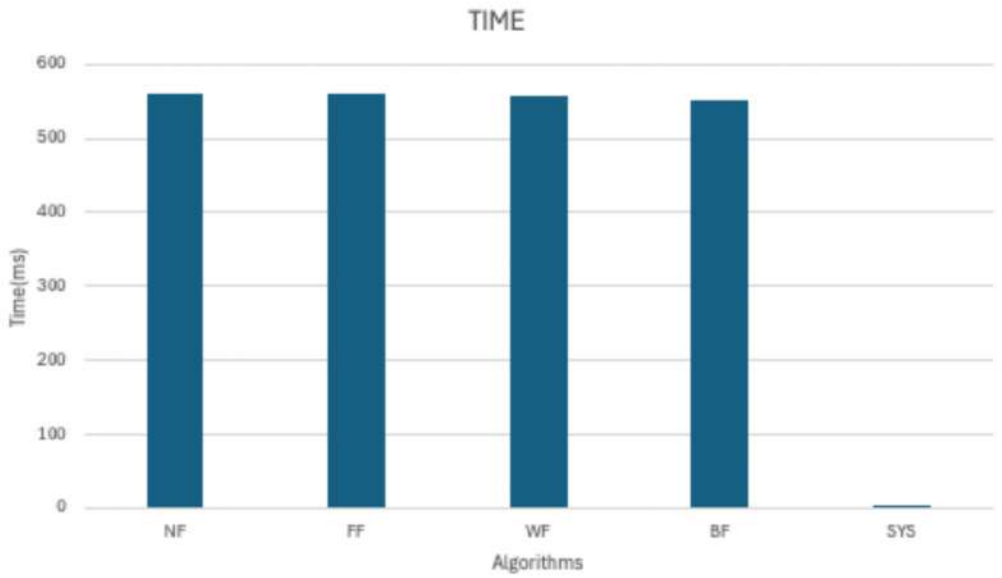
   100(1 - (max_free_block / (num_requested - max_heap))).*

4. Max heap size.

The bigger the heap at the end the smaller less efficient the algorithm is. Number of grows can indicated which algorithm had the biggest heap at the end. Tests 2,3,4 are all conducted in the same test file.

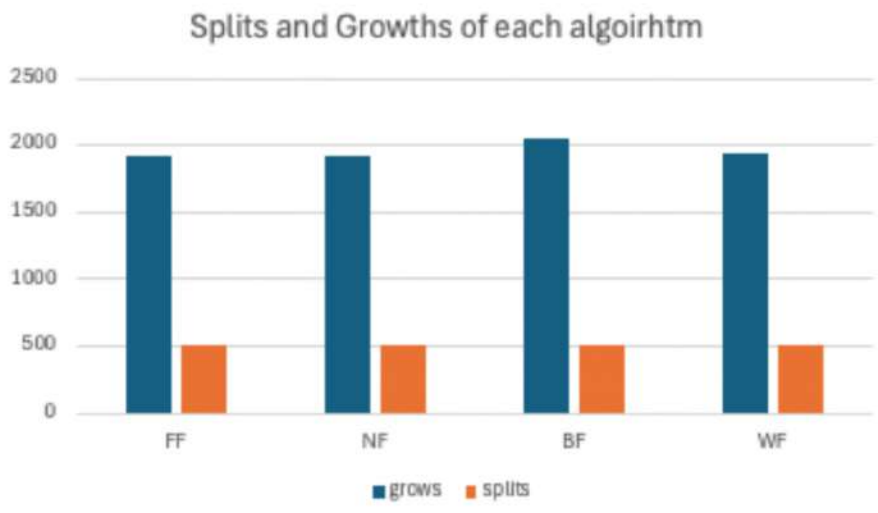
5. Test Results

1. Time performance (rigorous testing version)



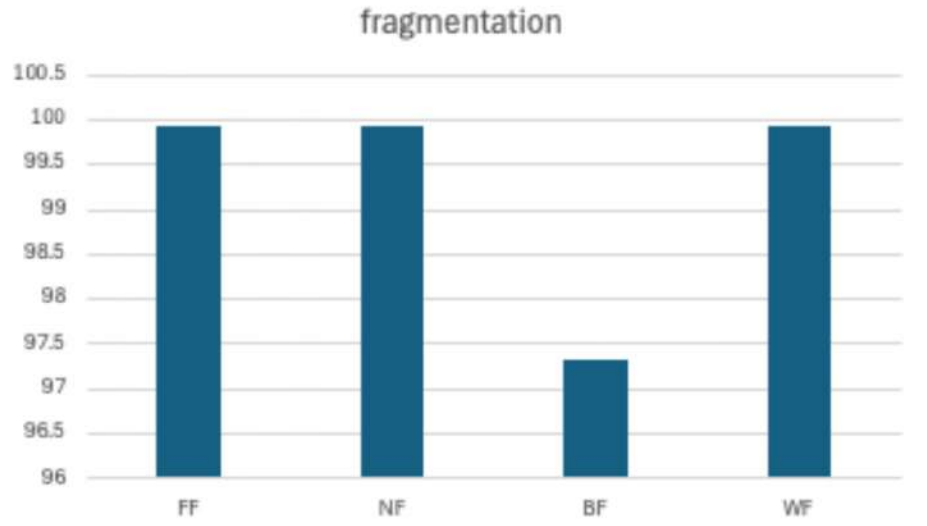
They all have small variations with best fit being the fastest at 551.8 ms. With the current performance test, sys malloc is 100 times faster. I mentioned earlier the less rigorous testing where constant blocks are allocated. This would make my malloc implementation look way better than it is where sys malloc would appear to be only 2.5 times faster.

2. Number of heap splits and heap growths, max heap size



NF,FF had the same number of growths while WF grew one extra time, BF had about 6% more growths than the rest making it the biggest heap size at the end. NF,BF had the same number of splits with FF,WF having more split than them.

3. External fragmentation



BF had the least amount of fragmentation

5. Explanation and Interpretation of Results

They all had similar results with time except system malloc being blazingly fast compared to my implementation. BF had the largest heap size at the end meaning it didn't fit blocks as efficient as the others but it had the least amount of external fragmentation.