Malloc - Code Report

The implementation of malloc relies on four different algorithms to decide which block of memory to choose when malloc is called:

First Fit = starts from the beginning of the list of blocks, stops when the first fit is encountered and returns that location

Next Fit = similar to first fit. It begins from the spot in the list that had the most recent allocation. From there, it performs its first fit. If it reaches the end without finding a fit, it wraps to the beginning of the list and searches again, up to the most recent allocation.

Best Fit = Finds a free block such that the difference of the free size it has and the requested size is smallest.

Worst Fit = Finds a free block such that the difference of its available size and the requested size is greatest.

In addition, splitting and coalescing of blocks were implemented

Splitting: Turns the remainder of free space in a block into a new block via pointer arithmetic, that can be used on future calls to malloc.

Coalescing: Implemented to combine consecutive free blocks. Iterate through the list and check if current->free && next->free. If so, combine them.

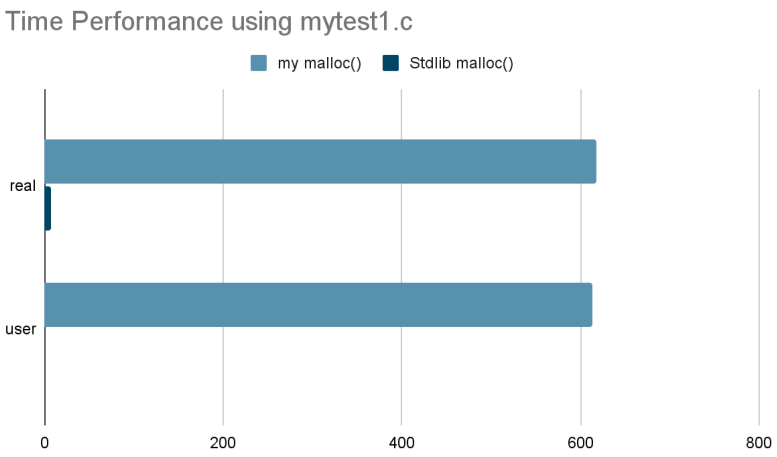
A set of tests was implemented to ensure the proper functionality and get efficiency measurement;

Test 1 = This test calls malloc and free lots of times using different sizes as well as loops. Since there are many calls in this test, I used it to measure the time performance of this malloc implementation compared to the stdlib.h malloc;

The times derived are as follows:

This malloc():
real 0m0.618s
user 0m0.614s
sys 0m0.002s

Stdlib.h malloc():
real 0m0.007s
user 0m0.000s
sys 0m0.008s



From this test it's easily derived that this 'simpler' implementation of malloc performs noticeably worse than the standard implementation, and the difference becomes more pronounced with larger amount of calls to malloc/free;

Tests 2 and 3 perform splitting of blocks and coalescing of blocks. The time discrepancy for these is not as big as mytest1 since there are only a few calls to malloc/free

There were no anomalies in the code or the results of the tests. Also, by running the suite of tests with different algorithms, we can conclude that there is no one-size-fits-all algorithm to implement dynamic memory allocation. They each have their strengths and weaknesses.

First fit is efficient, simple and fast but since it disregards the relationship between the request size and size of block currently pointed at, it can lead to fragmentation when there are many smaller allocations and frees.

Next fit follows a similar logic to first fit, except it starts the search from a globally tracked point in the list of blocks. It has the same strengths and weaknesses, and it trades some of ff's speed for less fragmentation

Best and Worst fit are slower than the previous two algorithms, in that they need to touch every single block before determining the one to use for a given request.

So, depending on which is more important, performance in terms of speed or the protection of memory (by way of minimizing internal fragmentation) a different algorithm will be a better suited tool.