Bryan Corona Rangel

CSE 3320

4/13/23

Executive Summary:

In this assignment, I was tasked with implementing my interpretation of malloc as well as first fit, best fit, worst fit, and next fit. With malloc, we had to implement coalescing, which is the joining of free blocks next to each other, and splitting, which separates a block into 2 smaller blocks, in order to test them against the standard system malloc. Comparisons will be made with the given 8 tests as well as the 3 tests I have created and will be shown through the statistics information output for each.

Algorithms Description

The first algorithm was first fit and this was already completed. This algorithm works by first seeing if the current block is not null, free, and that the current size of greater than the required size. If it is, then that is the block that is returned. If not, it will keep iterating through the linked list to find one or return NULL if nothing was found.

Best fit was the next algorithm and I did have to implement this myself. This algorithm works by keeping track of 2 aspects which is a block pointer (set to NULL) and a integer which is called winning remainder(set to int max). Best fit is like first fit in which we go through the entire linked list and see if the block is not null and free but then differs in which the difference between the current block's size and the requested size has to be less than the winning remainder. It will keep going through the list and return the block that has the smallest difference between the two compared sizes. If none were found, NULL would be returned.

Next algorithm was worst fit and I also implemented this. This is exactly the opposite of best fit in which now there is an integer called losing remainder in which the difference between the current block's size and the requested size has to be greater than it. It will also go through the linked list and now return the block that has the largest difference between the two sizes.

Lastly, next fit was supposed to be implemented by myself but due to some health complications, I was not able to complete it. However, I will still discuss how it should have worked. This algorithm should've have used a global variable in order to track the current node. This is because the algorithm will need to start iterating through the linked list where the last insertion was done. Once it starts, it will look for the next block that is greater than the requested size and return that block if found.

Test Implementation

For test implementation, I will first test my allocator against the programs that were provided and show the results of the statistics. I will then test them against the 4 programs I have created myself and also show the results for the standard system malloc and show the time it takes for each to complete the programs. Timing will use the normal "time" feature in terminal and use the real time.

Test Results

First Fit Next Fit Test:

First Fit:

```
First fit should pick this one: 0x5592e988a018
Next fit should pick this one: 0x5592e988bc58
Chosen address: 0x5592e988a018

heap management statistics
mallocs:        12
frees:          3
reuses:         2
grows:          10
splits:         0
coalesces:      0
blocks:         10
requested:      16048
max heap:       9064
```

Best Fit Worst Fit Test:

Best Fit:

```
Worst fit should pick this one: 0x55a235663018
Best fit should pick this one: 0x55a2356730c4
Chosen address: 0x55a2356730c4

heap management statistics
mallocs:        7
frees:          2
reuses:         1
grows:          6
splits:         1
coalesces:      0
blocks:         7
requested:      73626
max heap:       72636
```

Worst Fit:

```
Worst fit should pick this one: 0x556c3423a018
Best fit should pick this one: 0x556c3424a0c4
Chosen address: 0x556c3423a018

heap management statistics
mallocs:        7
frees:          2
reuses:         1
grows:          6
splits:         1
coalesces:      0
blocks:         7
requested:      73626
max heap:       72636
```

Calloc Test:

All algorithms had the same results.

```
calloc test PASSED

heap management statistics
mallocs:        2
frees:          1
reuses:         0
grows:          2
splits:         0
coalesces:      0
blocks:         2
requested:      1028
max heap:       1028
```

Realloc:

All had the same results except for worst fit. There was a seg fault that I was not able to figure out (had something to do with the way increased my num_blocks function).

```
calloc test PASSED

heap management statistics
mallocs:        2
frees:          1
reuses:         0
grows:          2
splits:         0
coalesces:      0
blocks:         2
requested:      1028
max heap:       1028
```

Test 1:

All algorithms had the same results.

```
Running test 1 to test a simple malloc and free

heap management statistics
mallocs:        2
frees:          1
reuses:         0
grows:          2
splits:         0
coalesces:      0
blocks:         2
requested:      66559
max heap:       66560
```

Test 2:

First Fit and Worst Fit:

```
Running test 2 to exercise malloc and free

heap management statistics
mallocs:        1027
frees:          514
reuses:         1
grows:          1026
splits:         1
coalesces:      2
blocks:         1025
requested:      1180670
max heap:       1115136
```

Best Fit:

```
Running test 2 to exercise malloc and free

heap management statistics
mallocs:        1027
frees:          514
reuses:         1
grows:          1026
splits:         0
coalesces:      1
blocks:         1025
requested:      1180670
max heap:       1115136
```

Test 3:

All algorithms had the same results.

```
Running test 3  to test coalesce

heap management statistics
mallocs:        4
frees:          3
reuses:         1
grows:          3
splits:         1
coalesces:      2
blocks:         2
requested:      5472
max heap:       3424
```

Test 4:

All algorithms had the same results.

```
Running test 4 to test a block split and reuse

heap management statistics
mallocs:        3
frees:          2
reuses:         1
grows:          2
splits:         1
coalesces:      1
blocks:         2
requested:      4096
max heap:       3072
```

Personal Test 1:

For my first test I made an array, ptrs, used malloc on each element, and then freed only half the array. After made another array, ptrs2, used malloc, and freed the rest of the first array. I then made another array, ptrs2, used malloc, and then freed ptr2 and ptrs3. The malloc sizes were different for each array and the results came out like this:

Best fit and Worst fit:

```
heap management statistics
mallocs:        3001
frees:          3000
reuses:         2001
grows:          1000
splits:         1001
coalesces:      1998
blocks:         2
requested:      2263024
max heap:       1000000
```

First Fit:

```
heap management statistics
mallocs:        3001
frees:          3000
reuses:         1001
grows:          2000
splits:         1001
coalesces:      2998
blocks:         2
requested:      2263024
max heap:       1632000
```

Times:

Standard Malloc: .003s

Best Fit: .005s

Worst Fit: .003s

First Fit: .003s

Next Fit: N/A

It seems that the first fit was the worst memory wise since it had a bigger heap than best/worst fit and also had 1000 more coalesces.

Personal Test 2:

This test was similar to personal test 1 in which 3 arrays were used but now they are of different sizes and malloc sizes and we see again that best fit/worst fit have a lower heap and they had more reuses and less splits compared to the other algorithms.

First Fit:

```
heap management statistics
mallocs:        11101
frees:          2100
reuses:         101
grows:          11000
splits:         101
coalesces:      2097
blocks:         9003
requested:      1357024
max heap:       1256000
```

Best Fit/Worst Fit:

```
heap management statistics
mallocs:        11101
frees:          2100
reuses:         601
grows:          10500
splits:         1
coalesces:      1497
blocks:         9003
requested:      1357024
max heap:       1128000
```

Times:

Standard Malloc: .003s

Best Fit: .004s

Worst Fit: .002s

First Fit: .003s

Next Fit: N/A

Personal Test 3:

First Fit:

```
heap management statistics
mallocs:        1005
frees:          1004
reuses:         99
grows:          906
splits:         99
coalesces:      1002
blocks:         2
requested:      1231258
max heap:       1026720
```

Best Fit/Worst Fit:

```
heap management statistics
mallocs:        1005
frees:          1004
reuses:         100
grows:          905
splits:         99
coalesces:      1001
blocks:         2
requested:      1231258
max heap:       1025696
```

Times:

Standard Malloc: .003s

Best Fit: .004s

Worst Fit: .003s

First Fit: .007s

Next Fit: N/A

Test Results Explanation

As can be seen, for the most part, the algorithms had similar performance with one another with few small differences. One anomaly that I came across was with realloc and worst fit. It was only until trying to run all the tests that I found realloc started to seg fault if worst fit was used. In no other algorithm did that occur and I was not able to figure out why that happened in time. In most cases, best/worst fit did better and the timings were mixed with either my implementation being faster or the standard system malloc.

Conclusion

In conclusion, I have implemented and tested my version of malloc with 4 ways of doing fitting algorithms. For the most part, it is seen to be similar in performance to the standard system malloc. Most of the statistics stayed the same other than a few keys ones which were reuses, grows, splits/coalesces and most importantly, max heap. As for the timings, it was not clear which malloc was better or even with algorithm was better because all had different timings that were either better or worse.