Assignment Malloc

Bishal Giri
1001934035
April 14, 2023

Allocation Methods:

Summary

The infamous malloc() function in C, was implemented in this assignment. The function here takes in form a Linked List and allocates new memory in form of new linked nodes. The malloc function when called looks for the next free block within the list of linked nodes. The alogirthms to find the next free block include the following dynamic memory allocation techniques:

- First Fit: The first free block that is found is allocated to the user.
- Best Fit: The most optimal block with the least remaining space is allocated to the user.
- Worst Fit: The most non-optimal block with the most remaning space is allocated to the user.
- Next Fit: The next free block after the last allocated block is allocated to the user.

Following finding the free block, the block is then further split into two different blocks, the first block would fit the user's malloced size and the second block would occupy the rest and added to the linked list. In case we don't find a free block we "grow our heap" and allocated new chunk of memory to the list.

Testing

To test that my memory allocation methods like Best Fit, Worst Fit, First Fit, and Next Fit were working, I ran the bfwf.c and ffnf.c with respective methods. The following is the output of the program:

First Fit with ffnf.c (implemented from the start)

```
@bishal0922 ➜ /workspaces/malloc-bishal0922 (master) $ env
LD_PRELOAD=lib/libmalloc-ff.so tests/ffnf
First fit should pick this one: 0x555d5df22018
Next fit should pick this one: 0x555d5df23c58
Chosen address: 0x555d5df22018

heap management statistics
mallocs:        12
frees:          3
```

```
reuses:         2
grows:          10
splits:         0
coalesces:      0
blocks:         10
requested:      16048
max heap:       9064
```

Next Fit with ffnf.c

```
@bishal0922 ➜ /workspaces/malloc-bishal0922 (master) $ env
LD_PRELOAD=lib/libmalloc-nf.so tests/ffnf
First fit should pick this one: 0x55964de28018
Next fit should pick this one: 0x55964de29c58
Chosen address: 0x55964de29c58

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

Best Fit with bfwf.c

```
@bishal0922 ➜ /workspaces/malloc-bishal0922 (master) $ env
LD_PRELOAD=lib/libmalloc-bf.so tests/bfwf
Worst fit should pick this one: 0x55d813ee8018
Best fit should pick this one: 0x55d813ef80c4
Chosen address: 0x55d813ef80c4

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

Worst Fit with bfwf.c

```
@bishal0922 ➜ /workspaces/malloc-bishal0922 (master) $ env
LD_PRELOAD=lib/libmalloc-wf.so tests/bfwf
Worst fit should pick this one: 0x56456919e018
Best fit should pick this one: 0x5645691ae0c4
Chosen address: 0x56456919e018

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

Testing

To further test the dynamic memory allocation methods, I wrote four different test programs which consisted for different function calls for `malloc()` and `free()` with multiple variable pointers and sizes. I then ran these programs with different methods from my program and compared to the system `malloc()` function.

I timed the execution of each file with the `clock()` function provided in `<time.h>` and computed the total time taken to run the program with the following formula:

```
total_t = (double)(end_t - start_t) / CLOCKS_PER_SEC;
```

The following is a table with the time taken by the different dynamic memory allocation methods (FF, NF, BF, WF) and the time compared to the system implementation of `malloc()` function. The time is in seconds.

	Library w/ FF	**Library w/ NF**	**Library w/ BF**	**Library w/ WF**	**System malloc()**
t1.c	0.000008	0.0000011	0.000009	0.000007	0.0000013
t2.c	0.000021	0.000017	0.000015	0.000016	0.000039
t3.c	0.000008	0.000007	0.000007	0.000008	0.000007
t4.c	0.000008	0.000008	0.000008	0.000009	0.000007

Conclusion

From the different implementations for programs it was evident that the Best Fit method of dynamic memory allocation was the most optimal method. This is because as we iterate through the entire list in `O(n)` time, we find the best block which can best fit the needs of our malloc size. This also took the least amount of time because we only had to iterate through the entire list once. On the other hand, the worst fit method was the least optimal because it took the most amount of time to find the block with the largest remaning space. This led us to recompute our block size through multiple splittings and coalescings. In addition, the system implementation of malloc somehow took a longer time than other methods which might be an anomaly because I believe that the system would implement the most efficient methods of memory allocation.