ECE 651 Assignment #2 Thread-Safe Malloc

netID: dl261 Student: De Lan

OVERVIEW

For every block of memory that I allocate and free, I implement a struct as a meta block to record this block's information: size, nextFree and prevFree. I keep one doubly linked lists to record all free memory blocks.

1. Compare with HW1 and analysis

In the former HW1, I kept two doubly linked lists, one of which stored the physically adjacent allocated blocks (denoted as physicList) and the other is a free list (denoted as freeList). The advantages are that I free a block and merge adjacent free blocks in O(1) by:
1) Add this block to the end of freeList;
2) Checking the physicList whether its adjacent blocks are also free;
3) If so, merge and delete one block from the freeList.
Now the end of freeList is refreshed, so next time if I malloc a space and I check for available block from the end of freeList, it would find the appropriate block fast.

Now that we have multiple threads, there is no need to keep the physicList, as:
1) Adjacent blocks cannot be physically adjacent. They are separated by allocated blocks by other threads;
2) We cannot merge free blocks by checking their physical positions in physicList;
3) If thread #1 and thread#2 are freeing the adjacent blocks concurrently in thread #3, there is inevitable race condition in thread #3's physicList.
We only need a doubly linked freeList. If we add the newly freed block to the end of freeList, it's not convenient to merge physically adjacent free blocks. So we need to make a physically sorted freeList and sacrifice O(n) time to insert the newly freed block into freeList by the information saved in meta block. I compared the new design with former design in hw1, both using Best Fit strategy.

HW1 former design:

```
dl261@vcm-2352:~/ece650/hw1/my_malloc/alloc_policy_tests$ ./equal_size_allocs
Execution Time = 3.283878 seconds
Fragmentation  = 0.327273
dl261@vcm-2352:~/ece650/hw1/my_malloc/alloc_policy_tests$ ./small_range_rand_allocs
data_segment_size = 3795408, data_segment_free_space = 59680
Execution Time = 4.251630 seconds
Fragmentation  = 0.015724
dl261@vcm-2352:~/ece650/hw1/my_malloc/alloc_policy_tests$ emacs Makefile
dl261@vcm-2352:~/ece650/hw1/my_malloc/alloc_policy_tests$ ./large_range_rand_allocs
Execution Time = 114.754515 seconds
Fragmentation  = 0.040172
```

HW1 new design:

```
dl261@vcm-2352:~/ece650/hw1/my_malloc/alloc_policy_tests$ ./equal_size_allocs
Execution Time = 28.803655 seconds
Fragmentation  = 0.378947
dl261@vcm-2352:~/ece650/hw1/my_malloc/alloc_policy_tests$ ./small_range_rand_allocs
data_segment_size = 3529960, data_segment_free_space = 71352
Execution Time = 6.852078 seconds
Fragmentation  = 0.020213
dl261@vcm-2352:~/ece650/hw1/my_malloc/alloc_policy_tests$ ./large_range_rand_allocs
Execution Time = 147.596626 seconds
Fragmentation  = 0.041030
```

2. New design in HW2

(1) No-lock version

I keep a global startFree as the head of the freeList for Thread Local Storage.

MALLOC

I traverse the freeList for best fit free block.

1) Found
   A. If I need to split it. Split it into two blocks and replace current block with new block in the freeList.
   B. No need to split, I just delete it from the freeList.
2) Not found, sbrk() new space and no need to update freeList.

FREE

1) Insert current block into freeList.
2) Check if the nextFree or prevFree blocks are physically adjacent. If so, merge them.

This design is thread safe because I keep only an independent freeList and its head in each thread. MALLOC in thread #1 will either change its own freeList or globally sbrk() and leave its freeList unchanged. If thread #1 FREEs its own block, that's fine, just add new blocks to its sorted freeList. If thread #1 FREEs blocks MALLOCed by thread #2, just add the block to freeList in thread #1 and nothing in thread #2 will be touched. Then both MALLOC and FREE can happen concurrently in multiple threads.

(2) Lock version

Everything is pretty much the same as the no-lock version. Except that I keep a global startFree as the head of the freeListLock and it is not declared as __thread. Each thread shares the same freeList. And I add mutex when I need to adjust the freeList in each thread. And because of the locks I add, the time efficiency will be worse than no-lock version but the data segment size should be smaller.

3. Results from performance experiments

(1) Using unchanged given variables: 4 threads, 20000 items, item-size: 32 * (4 ~ 32) bytes

1) No-lock version

2) Lock version

(2) Using changed variables: 8 threads, 20000 items, item-size: 32 * (4 ~ 32) bytes

1) No-lock version

```
dl261@vcm-2352:~/ece650/ECE650_hw2/thread_tests$ ./thread_test_measurement
No overlapping allocated regions found!
Test passed
Execution Time = 0.347490 seconds
Data Segment Size = 85301200 bytes
```

2) Lock version

```
dl261@vcm-2352:~/ece650/ECE650_hw2/thread_tests$ ./thread_test_measurement
No overlapping allocated regions found!
Test passed
Execution Time = 0.465593 seconds
Data Segment Size = 88686160 bytes
```

(3) Using changed variables: 4 threads, 20000 items, item-size: 32 * (4 ~ 16384) bytes

1) No-lock version

```
dl261@vcm-2352:~/ece650/ECE650_hw2/thread_tests$ ./thread_test_measurement
No overlapping allocated regions found!
Test passed
Execution Time = 0.476183 seconds
Data Segment Size = 18607358392 bytes
```

2) Lock version

```
dl261@vcm-2352:~/ece650/ECE650_hw2/thread_tests$ ./thread_test_measurement
No overlapping allocated regions found!
Test passed
Execution Time = 0.546660 seconds
Data Segment Size = 18488608872 bytes
```

(4) Using changed variables: 8 threads, 20000 items, item-size: 32 * (4 ~ 16384) bytes

1) No-lock version

```
dl261@vcm-2352:~/ece650/ECE650_hw2/thread_tests$ ./thread_test_measurement
No overlapping allocated regions found!
Test passed
Execution Time = 0.992363 seconds
Data Segment Size = 37077134056 bytes
```

2) Lock version

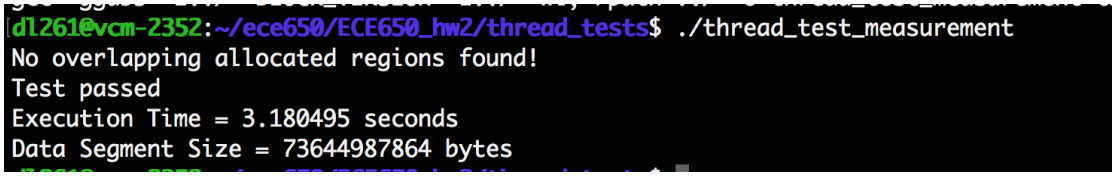
```
dl261@vcm-2352:~/ece650/ECE650_hw2/thread_tests$ ./thread_test_measurement
No overlapping allocated regions found!
Test passed
Execution Time = 1.257770 seconds
Data Segment Size = 37066857464 bytes
```

(5) Using changed variables: 8 threads, 40000 items, item-size: 32 * (4 ~ 16384) bytes

1) No-lock version

```
dl261@vcm-2352:~/ece650/ECE650_hw2/thread_tests$ ./thread_test_measurement
No overlapping allocated regions found!
Test passed
Execution Time = 2.079996 seconds
Data Segment Size = 73657649072 bytes
```

2) Lock version



4. Analysis of the results

As we can tell from the results, the no-lock version is more or less faster than the lock version, because mutex will block the other threads when a thread is operating on its freeList. And because the two versions share similar allocation and free strategies, the data segment sizes are pretty close. I run the two version for several times and the average Data Segment Size of the no-lock version is larger than the lock version. This is because the threads in the lock version share the same freeList, hence the segment efficiency is better.