Duke University

Assignment #2: Thread-Safe Malloc
Performance Report
ECE 650 – Spring 2019

Ruihan Xu
rx29
Date of submission: 2019/2/5

1. Objective

The objective of this assignment is to implement the thread safe malloc() and free() using both lock and no_lock. The performance will be tested to check whether the implementation is good or not. Moreover, the memory allocation will be based on only one algorithm, best fit.

2. Design

In the previous assignment, although I passed all of the test cases provided, I failed two TA's test cases. In this Assignment, I encountered a lot of difficulties when I was trying to directly implement the multi-threaded solution based on my previous work. Therefore, I simplified my free list by changing it to a single linked list with a header only and no tail as there are no time requirement and the test cases are less. The partition and the merge of free blocks makes the implantation extremely hard in multi-threaded versions, so I deleted those two functions.

3. Implementation

For the lock version, I used Mutex. The malloc part and the free part both need to access the free list. One is to remove fitted block from it and the other one is to add freed block to it. Threads will share the same list so there might have race condition. As a matter of fact, almost all the code within malloc and free are working with the list consistently. So I have decided to use mutex to lock the whole part of Malloc and Free. Notice that rwlock seems good to be used to improve speed. However, chances are that, although very small and are not likely to be reflected on tests, two threads reading the free list and find the same block to return.So the safest way is actually using mutex to prevent any threads from working on the list at the same time.

For the unlock version, I need to utilize the thread local storage. This means that each thread will have its own free list. Again, coalesce and partitioning can be extremely hard, and I was not able to make things work. Therefore, I simply deleted them due to the deadline of this assignment. There will be a __thread block_t* head_nolock for each thread to work on. Also, notice that a thread may free a block before it actually has a header. So we need to check that before add the block to free list. Also, since sbrk is not thread safe, we need to use mutex before and after it.

4. Performance

With the intention of making sure the implementation is correct and get an average performance measurement, I have run ./thread_test_measurements 10 times each for both version. The results are shown below:

```
rx29@vcm-8327:~/ECE650/Assignment2/tmp/thread_tests$ ./thread_test_measurement
No overlapping allocated regions found!
Test passed
Execution Time = 0.206548 seconds
Data Segment Size = 45036608 bytes
rx29@vcm-8327:~/ECE650/Assignment2/tmp/thread_tests$ ./thread_test_measurement
No overlapping allocated regions found!
Test passed
Execution Time = 0.145203 seconds
Data Segment Size = 44947808 bytes
rx29@vcm-8327:~/ECE650/Assignment2/tmp/thread_tests$ ./thread_test_measurement
No overlapping allocated regions found!
Test passed
Execution Time = 0.232619 seconds
Data Segment Size = 44865456 bytes
rx29@vcm-8327:~/ECE650/Assignment2/tmp/thread_tests$ ./thread_test_measurement
No overlapping allocated regions found!
Test passed
Execution Time = 0.176886 seconds
Data Segment Size = 45945632 bytes
rx29@vcm-8327:~/ECE650/Assignment2/tmp/thread_tests$ ./thread_test_measurement
No overlapping allocated regions found!
Test passed
Execution Time = 0.165690 seconds
Data Segment Size = 45066848 bytes
rx29@vcm-8327:~/ECE650/Assignment2/tmp/thread_tests$ ./thread_test_measurement
No overlapping allocated regions found!
Test passed
Execution Time = 0.147193 seconds
Data Segment Size = 44571304 bytes
rx29@vcm-8327:~/ECE650/Assignment2/tmp/thread_tests$ ./thread_test_measurement
No overlapping allocated regions found!
Test passed
Execution Time = 0.130849 seconds
Data Segment Size = 44390416 bytes
rx29@vcm-8327:~/ECE650/Assignment2/tmp/thread_tests$ ./thread_test_measurement
No overlapping allocated regions found!
Test passed
Execution Time = 0.142507 seconds
Data Segment Size = 44785488 bytes
rx29@vcm-8327:~/ECE650/Assignment2/tmp/thread_tests$ ./thread_test_measurement
No overlapping allocated regions found!
Test passed
Execution Time = 0.208201 seconds
Data Segment Size = 44349904 bytes
rx29@vcm-8327:~/ECE650/Assignment2/tmp/thread_tests$ ./thread_test_measurement
No overlapping allocated regions found!
Test passed
Execution Time = 0.160438 seconds
Data Segment Size = 44673304 bytes
```

Figure 1: Lock version

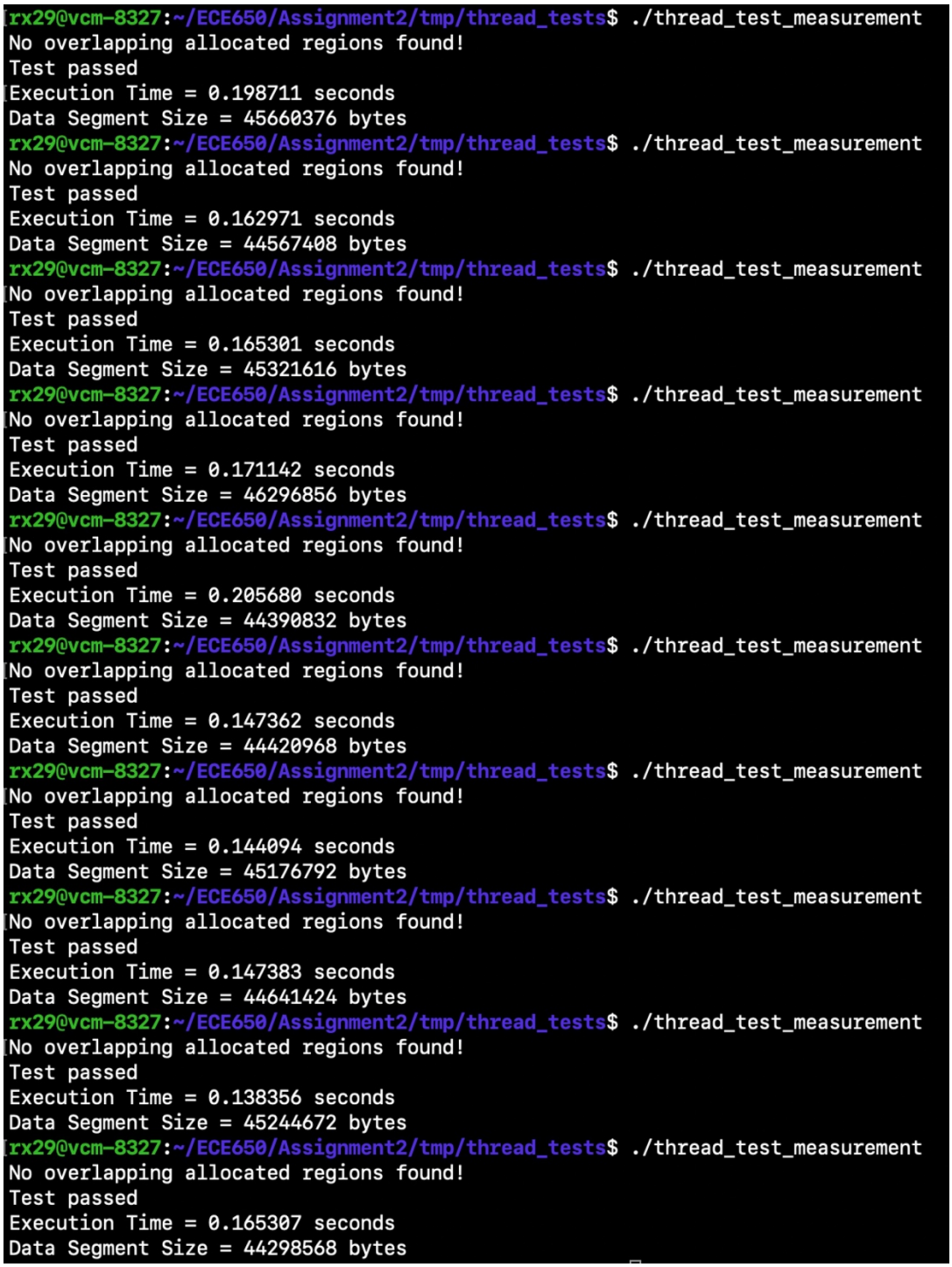

Figure 2: no_lock

The average execution time of lock version is : 0.1716s
The average data segment size of lock version is : 44863276
The average execution time of no_lock version is : 0.1646s
The average data segment size of no_lock version is : 45001951

From the result, we can see that the difference is not very significant. There could have many reasons behind it. First of all, the implementation of my lock version is very close to a single thread as there are not much of a concurrency. The no _lock version however, has lots of concurrency which should make the execution time faster. The result does showed this. An improvement of 4%. The speed improvement is not huge could due to the fact that it is limited by the traversing time of the list and the size of the test cases are not large enough.

As for the data segment size, the two version performs similar. This is because there are no merge and partition in both of them so a tied performance can be expected.