Project #1: Thread-Safe Malloc

1. Thread Safe malloc/free: locking version

Firstly, there is a very simple way to implement a lock to prevent competition. However, the obvious problem is that there are performance issues with this approach, as only one process can access the critical region at the same time. Another issue to be aware of is that since I am using the global variable dummyHead, this global variable needs to create many different heads (in different threads) when no lock is applied, so I pass dummyHead as a parameter to the function to keep the code uniform and concise.

2. Thread Safe malloc/free: No locking version

I use Thread-local storage to ensure that each process has access to a different dummyHead, and because we had locked sbrk directly in the lock version, but when we use no_lock, we need to remember to lock sbrk because this function is not multi-thread safe.

3. Result

Lock Version:

```
xg73@vcm-24574:~/ece650/homework2/thread_tests$ ./thread_test_measurement
No overlapping allocated regions found!
Test passed
Execution Time = 0.518216 seconds
Data Segment Size = 43233472 bytes
```

No lock Version:

```
[xg73@vcm-24574:~/ece650/homework2/thread_tests$ ./thread_test_measurement
No overlapping allocated regions found!
Test passed
Execution Time = 0.203777 seconds
Data Segment Size = 42860256 bytes
```

As you can see from the results, the locked version runs longer because we are only locking the critical areas. The non-locked version is faster, which indicates that the non-locked version has the advantage. Comparing the data sizes, they are similar.