Fangting Ma (fm128)
650 HW2

Thread-Safe Performance Report

Implementation Description

The basic structure is quite similar to the code I wrote in homework 1. I used a doubly linked list to store all the freed blocks in an ascending order of their memory addresses, however, I changed the interfaces of those functions, imported the pthread C library to use mutex and used Thread Local Storage related functions and structures.

We are supposed to change our malloc and free functions to thread-safe versions. For lock version, I can simply lock the mutex directly before and unlock it after both the original malloc and free functions, so that at most one thread can enter the region between lock and unlock functions. Those regions are critical sections. While for no lock version, I used Thread Local Storage to create separate linked lists for each thread, so that I only need to lock the region where calls sbrk function. And that's the critical sections for no lock version of malloc and free.

Basically, the implementations of lock version and no lock version are similar, but their linked lists of free blocks are different, so I took the heads and tails of linked lists as input variables for my_malloc and my_free functions and also added a flag variable to indicate whether it is a lock version or no lock version so that I did not need to replicate the code and can use different lock strategies according to the value of this flag variable passed in.

Results & Comparison

Since multithreading has some nondeterministic factors, I tested 20 times for each version of thread-safe models of malloc and free functions on a 4 core processers virtual machine to get relatively reliable results of their performance in terms of execution time and data segment size. And I drew them into the table below so we can easily compare their performance.

	Execution Time	Data Segment Size
Lock Version	0.52597825	43177000
No Lock Version	0.23214715	43099067

We can see from the table above that no lock version takes about half less runtime to finish the execution. The main reason is that the lock version locks much larger regions than no lock version. It locks from the start and the end of both malloc and free functions, which actually makes the lock version more sequentially than no lock version. While the no lock version takes advantage of multithreading, since it only locks the regions that calls the sbrk function.

For memory allocation, we can see that both versions have quite similar allocation efficiency, since they have the same best fit allocation policy, and the only difference that may influence memory is that lock version uses one linked list, but no lock version uses four linked lists to store free blocks, which may have little impact on the memory allocation efficiency.