ECE 650 Project 2 report

Can Pei cp357

Implementation

In my project, two versions of the custom concurrent malloc library are implemented, namely the locking version and the non-locking version. Both versions are based on the best-fit memory allocation policy from the previous project.

For the locking version, all threads operate on 2 shared, global metadata linked lists. Mutex lock from the C standard pthread library is used to avoid race conditions. Before a malloc/free operation is requested, the free list is locked by a specific thread and is prevented from modification from other threads. And when the operation is over, the lock is released, ready to be acquired by another thread. To sum up, the critical sections are the original bf_malloc() and bf_free() sections.

For the non-locking version, thread local storage(TLS) is leveraged, so each thread has its own 2 linked lists. In this way, no mutex lock is required when the lists are modified. However, the C standard sbrk function requires mutex lock when invoked, due to its thread-uncertain nature. Each list has two thread-local variables, demarcated by keyword __thread indicating their heads and tails.

Performance

The performance is as the table shows below:

	Speed(s)	Segment size(bytes)
Lock	0.0795	43874848
Non-lock	0.0948	50009536

Speed

Both versions are able to finish execution in a reasonable time. The locking version performs a bit better than the non-locking version. It is expected that the overhead incurred from building a new head and tail for each thread by the non-locking version contributes to the time penalty significantly. The locking version shares the same list.

Segment size

The locking version has a smaller segment size in bytes than non-locking version. That's because in the locking version only a single free list and all list exists, thus merging would happen more frequently resulting in more optimized memory space. On the contrary, the non-locking version has different lists in each thread. Although two physically adjacent spaces are mergeable, they may belong to different threads and no subsequent merge can occur. As a result, it is expected to have a larger segment size than the locking version.

Discussion and Conclusion

It is discovered that the locking version has better speed and smaller segment size than the non-locking version, thus locking version is recommended to be used.