Report of Thread-Saft Malloc

Name: Junfeng Zhi | NetID: jz399

1. Overview of Implementation

In this assignment, I implement two different thread-safe versions of malloc() and free() functions. Both version of malloc use the best fit allocation policy. In my code. I use linked-list to record the free block and this list will cause race condition. Therefore, I need to ensure that different thread has its own linked-list or only one thread can edit the linked-list at one moment.

1.1 Lock version

For lock version, I use mutex lock to avoid race condition. I add lock and unlock operation before and after the original malloc/free code. Therefore, only one thread can operate the linked-list at one moment.

1.2 Unlock version

For unlock version, I use thread-local storage to create a linked-list for every thread, so each thread can only access to its own linked-list. Therefore, race condition will not exist. However, since sbrk() function is not thread-saft, we still need to add a mutex lock/unlock operation before/after the sbrk() function.

2 Results and Analysis of performance test

2.1 Experiment results

Version	Average Execution Time /s	Average Data Segment Size /bytes
Lock	0.2138	42721168
Non-lock	0.1896	42892517

2.2 Analysis

Based on the experiment result, we can see that non-lock version runs faster than lock version. This is because non-lock version only use lock to protect sbrk function, while lock version use lock to protect the whole malloc and free function. Therefore, non-lock version allows more code to run simultaneously outside the lock operation.

On the other hand, both versions of malloc/free use similar data segment size. This shows that two version of code use the same mechanism to reuse and allocate blocks. But non-lock version has lower utilization of old-blocks than lock-version. Since each thread has its own linked-list, when one thread call free function and then exit, other thread can not reuse the blocks recorded in that thread's linked-list.