Project 2: Thread-Safe Malloc

NetID: qx37

Name: Qingyang Xu

1. Abstract

This project is to realize thread-safe malloc() and free() functions of locking and non-locking version based on the best fit allocation. For locking version, mutex is used to prevent race condition. A lock is acquired immediately before BF malloc or free operations and released immediately after malloc or free. For non-locking version, lock is not used except for sbrk() function because sbrk() is not thread-safe. To prove synchronization, Thread-Local Storage is applied to variable declarations when initialized.

2. Implementation

Based on the strategy of singly linked list realized in the previous project, this project utilizes the best fit allocation. For both locking and non-locking version, a lock needs to be initialized as a global variable for mutex in the following steps.

For locking version, the basic idea is to add a lock before and release the lock after each operation. Therefore, the steps for thread-safe malloc with lock are: add a lock, call BF malloc, release the lock. Similarly, thread-safe free with lock is realized by the steps: add a lock, call BF free with pointer to pointer to the head of the free list, release the lock.

For non-locking version, there is a little change on the free list. The head of the free list in non-locking version is declared as Thread-local storage variable. Therefore, each thread can have its own free list and will not have overlapping allocated memory regions on the same free list. An exception here is to add a lock for sbrk() function because sbrk() is not thread-safe.

3. Results

Measurement Test	Locking Version	Non-locking Version
Execution Time (Avg for 10 times)	0.140743s	0.126347s
Data Segment Size (Avg for 10 times)	43731704 bytes	43859072 bytes

4. Analysis

The average test execution time for locking version is longer than non-locking version. The result is valid because the mutex mechanism makes locking version look like running sequentially but not simultaneously. Non-locking version enters critical region only when calling sbrk(), while other operations such as searching free list or removing block from free list can happen at the same time, and these operations that happen in one thread would not disturb other threads according to TLS strategy. The data segment size of the two versions of thread-safe malloc are very close. The reason might be the similar design of these two versions and sbrk() will extend the exact size of data segment as required.act size of data segment as required.