Report – Project #2: Thread-Safe Malloc

Name: Xichun Gou NetID: xg84

1 Overview

Each region consists of two parts: metadata and data storage. Metadata, as a struct, contains region information: a pointer to the previous region, a pointer to the next region, and size (the size of the data storage part). Manage free regions through the data structure of the doubly linked list, as a freelist. When a region needs to be allocated, the region needs to be removed from the freelist. When a region is freed, the region needs to be inserted into the freelist.

```
struct metadata{
    struct metadata * prev;
    struct metadata * next;
    size_t size;
};
typedef struct metadata meta;
```

Compared with project1, in order to implement the non-locking version more conveniently, the following struct structure is added, which replaces meta * freelisthead/meta * freelisttail and points to the head and tail of each freelist.

```
struct freelist{
  meta * head;
  meta * tail;
};
typedef struct freelist flist;
```

2 Implementation

locking version	non-locking version
Use pthread_mutex_t lock, add pthread_mutex_lock(&lock); and pthread_mutex_unlock(&lock); before and after calling best fit malloc/free; when a thread acquires the lock, other threads need to wait until the thread that acquired the lock returns the lock. Subsequent threads perform subsequent malloc/free operations by acquiring locks.	A non-locking version is implemented using Thread-Local Storage. Each thread has its own freelist. By adding a parameter int lock_status to the function, in the non-locking version, the lock can only be acquired/returned before and after sbrk.

3 Results from performance experiments

locking version：

```
No overlapping allocated regions found!
Test passed
Execution Time = 0.171036 seconds
Data Segment Size = 47928976 bytes
```

non-locking version：

```
No overlapping allocated regions found!
Test passed
Execution Time = 0.160370 seconds
Data Segment Size = 48081840 bytes
```

4 Conclusion

The non-locking version takes less time to run on average. Because in the non-locking version, it is only locked before and after sbrk, and each thread needs to wait. In the locking version, as long as the malloc or free function is called, if any other thread has acquired the lock, then other threads need to wait.