Report for Thread Safe Malloc Library

- Student name: Xueyang Liu
- Student Netid: xl350

1. Design & Implementation

In this project, the previously implemented My Malloc Library was improved by adding thread-safe feature. To allow concurrency, the potential race conditions needed to be handled carefully. This feature was implemented in two different ways, a lock version and a non-lock version.

1.1 Lock Version

In the lock version, I simply added locks to previous code and make the linked list can only be accessed by one thread at once.

```
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

void *ts_malloc_lock(size_t size){
    pthread_mutex_lock(&lock);
    int allocate_lock = 0;
    void * ptr = bf_malloc(size, &head_lock, &tail_lock, allocate_lock);
    pthread_mutex_unlock(&lock);
    return ptr;
}

void ts_free_lock(void *ptr){
    pthread_mutex_lock(&lock);
    my_free(ptr, &head_lock, &tail_lock);
    pthread_mutex_unlock(&lock);
}
```

The parts between `pthread_mutex_lock(&lock)` and `pthread_mutex_unlock(&lock)` are the critical sections. Concurrency is allowed at `malloc` and `free` level.

1.2 Non-lock Version

In the non-lock version, locks were required to be only applied to `sbrk()` . In this case, it could happen that two threads were accessing the same linked list at the same time and mess it up. To avoid this, `Thread Local Storage` was used by defining the head and tail of the linked list for each thread as following.

```
__thread block_t * head_nolock = NULL;
__thread block_t * tail_nolock = NULL;
```

Also, a lock for `sbrk()` was added and since I abstracted the `generateNew()` part out, only one plcae in the code needed to be changed, as below.

```
    ......
    if(allocate_lock==0){
        newOne = sbrk(required);
    }else{
        pthread_mutex_lock(&lock);
        newOne = sbrk(required);
        pthread_mutex_unlock(&lock);
    }
    .......
```

2. Experiment Analysis

2.1 Test Results

Version	AVG Execution Time	AVG Data Segment Size	PASS RATE
Lock Version	0.255 s	43032007 bytes	200/200
Non-lock Vresion	0.253 s	42994790 bytes	200/200

2.2 Analysis

All tests were passed and the results are shown in the table above.

For average execution time, they are similar but have slightly difference. This can be caused by the difference of concurrency level. Lock version allows a `weaker` concurrency since all malloc and free operations are handled as critical sections and non-lock version allows a `stronger` one since it only prohibits `sbrk()` to run concurrently.

For average data segment size, they are also similar and they should be so. Because the malloc policy I used was the same one -- `Best-fit` malloc.