Project 2: Thread-Safe Malloc

NetID: ys270
Student Name: Yue Shao

Overview of Implementation:

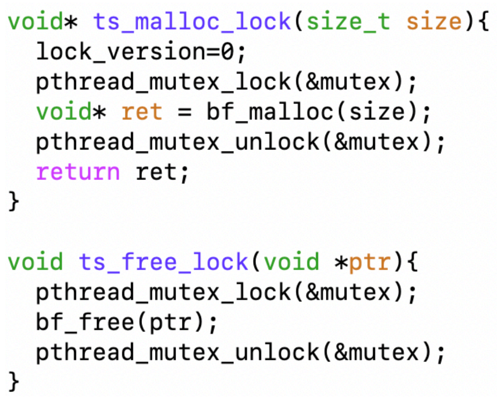
Because I did not pass all the testcases of project1, so I rewrite it with simple free list which just keep track of all the free space. This time I just use a head pointer to trace the begin of the free list, and all blocks in the list are sorted in ascending order.
For both versions in project 2, we need to initialize a lock before we try to lock and unlock (the no lock version also needs to lock and unlock for calling *sbrk*):

```
#include <pthread.h>
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

As to determine which implementation version is running, I use a global variable called *lock_version*. If it is equal to 0, then the lock version is running, the head of the freelist is called *fhead_lock*. If it is equal to 1, the no_lock version is running, the head of the freelist is called *fhead_nolock*. Every time when we need to use or update the head of the freelist, we could tell which head pointer we need to use by the value of *lock_version*.

- Lock Version:

The section between pthread_mutex_lock and pthread_mutex_unlock is called critical section, this part would not run simultaneously. Based on this idea, we could easily implement this thread-safe version by adding lock and unlock before and after bf_malloc and bf_free (as the picture below):



```
void* ts_malloc_lock(size_t size){
  lock_version=0;
  pthread_mutex_lock(&mutex);
  void* ret = bf_malloc(size);
  pthread_mutex_unlock(&mutex);
  return ret;
}

void ts_free_lock(void *ptr){
  pthread_mutex_lock(&mutex);
  bf_free(ptr);
  pthread_mutex_unlock(&mutex);
}
```

This LOCK_VERSION provides a malloc and free level concurrency.

- No lock Version:

For this version, we should only add lock and unlock right before and after sbrk call, with no lock added at other places. The main idea of this version is to use the __thread (as the picture below) to create a free list for each thread (every free list has a different head pointer). On every thread, codes are executed sequentially, so every free list is independent on each other, no overlapping memory region should appear.

```
__thread block * fhead_nolock = NULL;
```

This NOLOCK_VERSION prodes a sbrk level concurrency.

Result & Analysis:

	LOCK_VERSION	NOLOCK_VERSION
AVG TIME	0.207702s	0.100775s
MAX TIME	0.285541s	0.141163s
MIN TIME	0.133537s	0.086205s
AVG SIZE	42747141	42436315
MAX SIZE	43683216	43910352
MIN SIZE	41895696	41658176

Analysis of execution time:
As can be seen from the result, the NOLOCK version is faster than the Lock version. I think this is because NOLOCK version would only lock during the *sbrk* call, so other operations, like free block adding or removal can happen simultaneously. Besides, for LOCK version, all threads share one singly free list, so the free list could be really long, making it hard to find the best fit one.

Analysis of data segment size:
Based on the results, there is not evident difference between two versions on size. This means if a free block cannot hit on single free list (LOCK version), then it might not hit on multiple free lists (NOLOCK version) either.

Conclusion:

In summary, these two versions mainly differ in the execution time. There is no evident improvement of NOLOCK version on data segment size.