Report of Thread-Safe Malloc

Runya Liu (NetID: rl235)

1 Description of thread-safe model

I implemented two different thread-safe versions of the malloc() and free() functions. Both of my thread-safe malloc and free functions uses the best fit allocation policy based on project 1.

In version 1 of the thread-safe malloc/free functions, I used lock-based synchronization to prevent race conditions that would lead to incorrect results. These functions can be seen as attached below:

```
void * ts_malloc_lock(size_t size) {
  pthread_mutex_lock(&lock);
  void * p = bf_malloc(size, 0, head_lock);
  pthread_mutex_unlock(&lock);
  return p;
}
void ts_free_lock(void * ptr) {
  pthread_mutex_lock(&lock);
  bf_free(ptr, head_lock);
  pthread_mutex_unlock(&lock);
}
```

Specifically, since it's allowed to use lock, I applied lock on both malloc and free. I passed in 0 to differentiate between "1" passed in bf_malloc in version 2. If it's 0, then sbrk function doesn't need to be locked again, since the whole malloc is locked. However, if version 2, then sbrk needs to be locked because it's not thread safe.

```
void * allocate_new_block(size_t size, int sbrk_lock) {
//void * allocate_new_block(size_t size) {
  data_segment += size + sizeof(Metadata);
  Metadata * new_block = NULL;
  if (sbrk_lock == 0){
    new_block = sbrk(size + sizeof(Metadata));
  }
  else{
    pthread_mutex_lock(&lock);
    new_block = sbrk(size + sizeof(Metadata));
    pthread_mutex_unlock(&lock);
  }
  new_block->size = size;
  new_block->prev = NULL;
  new_block->next = NULL;
  return new_block;
}
```

In version 2 of the thread-safe malloc/free functions, I used lock for the `sbrk` function, which is explained in version 1 above, and applied thread-local storage to prevent race conditions. These functions can be seen as attached:

```
void * ts_malloc_nolock(size_t size) {

  void * p = bf_malloc(size, 1, head_unlock);

  return p;
}

void ts_free_nolock(void * ptr) {

  bf_free(ptr, head_unlock);

}
```

Specifically, in version 2 I used __thread to make each thread has its' own head to totally avoid race conditions, while in version 1 both threads share the same head in the heap.

```
__thread Metadata * head_unlock = NULL;

Metadata * head_lock = NULL;
```

2 Performance Result Presentation

	Lock	Unlock
Execution Time (s)	0.150371	0.136546
Data Segment Size (bytes)	48089536	48089536

3. Comparison of locking vs. non-locking version

From the perspective of execution time, unlock is slightly more quickly than the lock version. It may be because in version 1 lock is applied to the whole malloc function while in version 2 it's only applies to sbrk function, making it slightly more quickly. Also, in the unlock version, there are two heads, and thus each linked list is half the size of that in version 1. This could potentially lead to a more efficient search of the best block to allocate data.