Report for ECE650 homework2

Yilin Yang | yy379

Section 1: Thread-safe malloc implementation

The objective for this project is to implement thread-safe malloc library in two versions. Thread safety in this project refers to the ability of multiple threads to safely allocate and free memory blocks simultaneously without causing race conditions or other synchronization problems.

For the first version, I used locks in pthread library to prevent race conditions between different threads. Prior to allocating memory, the malloc function would lock a certain area and release the lock once the memory has been allocated. Similar to the lock function, the free function will take possession of the lock before dealing with memory and release it once the deallocation is finished. In this context, simply add a mutex lock can ensure the thread safety.

```
//Create thread-safe malloc for the best-fit allocator by using mutex lock
void *ts_malloc_lock(size_t size) {
    int lock_flag = 1;
    pthread_mutex_lock(&lock);
    void * ptr = bf_malloc(size, head_free_lock, tail_free_lock, lock_flag);
    pthread_mutex_unlock(&lock);
    return ptr;
}

//Create thread-safe free for the best-fit allocator by using mutex lock
void ts_free_lock(void *ptr) {
    pthread_mutex_lock(&lock);
    bf_free(ptr, head_free_lock, tail_free_lock);
    pthread_mutex_unlock(&lock);
}
```

For the second version, lock is only used when sbrk() is called because sbrk function is not thread-safe. In addition, a thread-local storage is used to make the explicit free list for each thread is independent.

```
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
__thread block_fl * head_free_nolock = NULL;
__thread block_fl * tail_free_nolock = NULL;
```

```
//Thread Safe malloc/free: non-locking version
void *ts_malloc_nolock(size_t size){
    int lock_flag = 0;
    void* ptr = bf_malloc(size, head_free_nolock, tail_free_nolock, lock_flag);
    return ptr;
}
```

Section 2: Performance Results & Analysis

From the provided benchmark, I got the performance result below:

Version	Execution Time (s)	Data Segment Size (byte)	Iteration times
Lock	0.167815	48089536	20000
No Lock	0.152537	48089678	20000

The performance of the two thread-safe malloc implementations (version 1 and version 2) is related to several factors in the test program, such as the number of threads, the iteration times, the allocation size, and the frequency of allocations and deallocations. In my result, the lock version is a little bit slower. The reason for the speed difference is the lock usage. In version 2, the lock is only applied when using sbrk() function so the code will run faster because the code can be execute without the lock in the most of time. For the data segment size, they are very similar as they all used best-fit algorithm when allocate memory blocks.