Report of Project #2: Malloc Library Part 2

Name: Kaiyuan Li | Netid : kl406

In this project, I implemented two different thread-safe versions of the malloc() and free() functions in C.

1.Implementation

Use malloc() and free() functions which applys best fit strategy in Project #1.

1.1 lock version

A race condition occurs when two threads access a shared variable at the same time. For lock version, put pthread_mutex_lock(&lock) before malloc function and put pthread_mutex_lock(&lock) after malloc function. So does free function. Using the lock, two threads cannot access the codes between lock and unlock simultaneously.

```
/* lock version */
void *ts_malloc_lock(size_t size){
    pthread_mutex_lock(&lock);
    void * res = bf_malloc(size);
    pthread_mutex_unlock(&lock);
    return res;
}
void ts_free_lock(void*ptr){
    pthread_mutex_lock(&lock);
    bf_free(ptr);
    pthread_mutex_unlock(&lock);
}
```

1.2 no lock version

For no lock version, use lock before sbrk() and unlock after sbrk(). So two threads cannot do the sbrk() simultaneously.(Thus they don't ask for the same space allocation.)

```
...
pthread_mutex_lock(&lock);
void * res = sbrk(0);
sbrk(size);
pthread_mutex_unlock(&lock);
...
```

2.Results & Analysis

Version	Execution Time	Data Segment Size
lock	0.212657	42937152
no lock	0.142521	44162464

As can be seen from the execution time of the table, no lock version runs faster than lock version, because lock version locks the entire code of malloc/free, but no lcok version only locks the sbrk() part. So there can be more simultaneous executions in the no lock version, and thus run faster. From the data segment size, the results obtained by the two strategies are similar, and it can be inferred that the process of calling sbrk and reusing the free block of the two strategies are similar.