Assignment 2 report

- **Qingqian Wang**
- **qw116**

1. project structure

1.1 Lock version

In the lock version, I use a mutex to lock each call of the malloc and free. To be more specific, my lock version of thread-safe malloc will run in a sequential way.

```
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

void* ts_malloc_lock(size_t size){
    pthread_mutex_lock(&mutex);
    void* temp = bf_malloc(size, FreedHeadNode);
    pthread_mutex_unlock(&mutex);
    return temp;
}
```

1.2 Nolock version

For the Nolock version, I use the thread local variable in each process. By using the individual head node, each process will have its own block list. Because each process will only search for the valid block in the local list, no concurrent problem will happen. However, this method will increase the num of data fragments.

```
__thread node* localHeadNode = NULL;

void* ts_malloc_nolock(size_t size){
    bf_malloc(size, localHeadNode);
}
```

Because the sbrk() is thread-unsafe, so I create another mutex to lock sbrk() when I use it.

```
pthread_mutex_t sbrkMutex = PTHREAD_MUTEX_INITIALIZER;
...
if(targetNode == NULL){
        pthread_mutex_lock(&sbrkMutex);
        targetNode = (node*)sbrk(newSize + sizeof(node));
        pthread_mutex_unlock(&sbrkMutex);
        targetNode->prev = NULL;
        targetNode->next = NULL;
        targetNode->size = newSize;
    }
...
```

2. Performance analysis

	Execution Time	Data Segment Size
Lock Version	0.165431	44811168
No Lock Version	0.167019	44963488

As you can see, lock version has both less execution time and the smaller data segment size. For the former, lock. For the latter, it is easy to understand because the no-lock version doesn’t share the freed block list, so even if an adjacent block is freed, you still can not merge it if they are in a different list. For the former, no lock version costs longer time because the space efficiency is lower, and it will call sbrk() more times.