ECE 650 Thread-Safe Malloc

Performance Study Report

Zhe Fan || zf70@duke.edu

February 2, 2023

1 Implementation Description

For this assignment, I implemented my own version of two different thread-safe versions (i.e. safe for concurrent access by different threads of a process) of the malloc() and free() functions. This is to solve problems like race conditions, which means that incorrect parallelcode may (sometimes often) result in correct execution due to the absence of certain timingconditions in which the bugs can manifest.

1.1 Lock-Based

For Lock-Based version, we should put lock before and after malloc() and free(), function pthread_mutex_lock() will prevent running simultaneously, which allows concurrency for malloc() and free().

```
1     pthread_mutex_t fastmutex = PTHREAD_MUTEX_INITIALIZER;
2     int lock = 0;
3
4     void * ts_malloc_lock(size_t size) {
5       pthread_mutex_lock(&fastmutex);
6       lock = 0;
7       void * p = bf_malloc(size, &free_region_Start, &
    free_region_End);
8       pthread_mutex_unlock(&fastmutex);
9       return p;
10    }
11    void ts_free_lock(void * ptr) {
12      pthread_mutex_lock(&fastmutex);
13      bf_free(ptr, &free_region_Start, &free_region_End);
14      pthread_mutex_unlock(&fastmutex);
15  }
```

1.2 Non-Lock-Based

For Non-Lock-Based version, we only put lock before and after sbrk(). And each time we use a new list to represent the free chunk, which means every free chunk is independent on each other, without overlapping memory region.

```
1    __thread chunk * free_region_Start_nolk = NULL;  //Start
2    __thread chunk * free_region_End_nolk = NULL;    //End
3
4    void * ts_malloc_nolock(size_t size) {
5      lock = 1;
6      void * p = bf_malloc(size, &free_region_Start_nolk, &
     free_region_End_nolk);
7      return p;
8    }
9    void ts_free_nolock(void * ptr) {
10     bf_free(ptr, &free_region_Start_nolk, &
     free_region_End_nolk);
11   }
12
13   ...
14   if (lock) {
15     pthread_mutex_lock(&fastmutex);
16     new = sbrk(0);
17     request = sbrk(size + META_SIZE);
18     pthread_mutex_unlock(&fastmutex);
19   }
20   ...
```

2 Performance Result Presentation and Analysis

The performance results of my Thread-Safe malloc functions are as below:

Table 2.1: Performance of Non-Lock and Lock.

	Lock	Non-Lock
Execution Time(s)	0.109	0.178
*Data segment size(bytes)	43251424	44433824

*Average for 50 times tests.

For Execution Time, Non-Lock is faster than Lock. According to Section 1, Non-Lock only lock sbrk(), but all other operations will happen simultaneously.

However, for Lock-version, it locks the malloc() and free() which makes less code run simultaneously. Therefore, Lock-Based is faster.

For Data segment size, Lock-Based and Non-Lock-Based have close size, which means they behavior similarly in find free chunk.