ECE650 Project2

Yue(Joey) Yu
yy373

In this project, I write two versions of thread-safe malloc based on best fit strategy. For both of these two versions, I need define a lock:

    pthread_mutex_t lock;

1.

Based on HW1, first, I use lock-based synchronization to prevent race conditions. If without the lock, and we are running several threads, it will be easy that the memory allocated by mallocs will overlap, for example, the start address of memory1 will be bigger than the start address of memory2 but smaller than the end address of memory2, which may cause serious problems. So, it is easy to have an idea that we can put the malloc function just between pthread_mutex_lock(&lock) and pthread_mutex_ublock(&lock), which is as follows:

```
void *ts_malloc_lock(size_t size){
    pthread_mutex_lock(&lock);
    void * result = lock_malloc(size);
    pthread_mutex_unlock(&lock);
    return result;
}
void ts_free_lock(void *ptr){
    pthread_mutex_lock(&lock);
    ff_free(ptr);
    pthread_mutex_unlock(&lock);
}
```

2.

For the second version, I plan not to use lock. But for the function sbrk(), which is not thread-safe, you have to acquire a lock immediately before calling sbrk and release a lock immediately after calling sbrk, which is like this:

```
pthread_mutex_lock(&lock);
Node* new=(Node *)sbrk(size+sizeof(Node));
pthread_mutex_unlock(&lock);
```

Compared to the first version, I plan to use:

```
__thread Node* nolockhead=NULL;
__thread Node* nolocktail=NULL;
```

As the head and the tail of the freelist. For every thread, the nolockhead and nolocktail are independent, which means they not do share data with other thread’s head and tail. Thus, each thread has their own freelist, which mean there would be the race conditions.

3.

	Lock		Nolock	
	Time(second)	Data Segment Size(bytes)	Time(second)	Data Segment Size(bytes)
1	0.216053	43342192	0.135492	48089536
2	0.264984	42743672	0.147573	48089536
3	0.14978	42555792	0.11983	48089536
4	0.164907	42470960	0.171359	48089536
5	0.219547	43236080	0.144618	48089536
6	0.182721	43287976	0.149988	48089536
7	0.159134	43116384	0.13169	48089536
8	0.14906	44615536	0.148017	48089536
9	0.200527	42287352	0.129091	48089536
10	0.165823	43735376	0.128544	48089536
Avg Time	0.1872536		0.1406202	

I did the experiments for about 10 times. However, If I want to draw a more accurate conclusion, I should do the experiments for more than 100 times, since the results of multi-threaded programming are full of uncertainty.

The runtime of Nolock version is faster than the lock version, the lock of the nolock version is only for the sbrk(), while the other functions, like split, remove can run simultaneously, which save lots of time. And for the lock version, the freelist would be quite long. So for the best fit search, it takes more time to search for a suitable size, while the nolock version, the thread of each freelist is independent, so each one of them would not be very long, which saves time to search. However, for data segment size, it is nearly the same between lock and nolock version.

All in all, the nolock version is better than lock version.