ECE650 Project 2 (thread-safe malloc and free) Report

Name: Kefan Lin   NetID: kl352

Design and implementation

In, project 1, to keep a fast speed, I keep updating 2 doubly linked lists. One is a main linked list which contains all blocks in the heap. The other is a free list which just contains all free blocks in the heap. However, this implementation cannot pass the all the test cases. Consequently, for Project2, I write another version, in which I just keep track of a singly free list. Although it has a slower speed, it is much easier to debug and get the correct result than the previous design. And both the lock version and nolock version use best-fit approach.
To implement these 2 versions, I use a flag variable called 'is_nolock' in all the functions to indicate which version it is running.

1. Lock Version

   In the lock version, the implementation is very straightforward. Just need to make a litte modification in the re-written code of project 1(the singly free list version). Specifically, I lock the mutex before calling malloc and free, and unlock the mutex after calling them. The section between locking and unlocking is the critical section. And the lock version gives us a malloc/free level concurrency.

```
//lock version malloc()
void * ts_malloc_lock(size_t size){
  pthread_mutex_lock(&lock);
  void * ans = bf_malloc(size,0);
  pthread_mutex_unlock(&lock);
  return ans;
}

//lock version free()
void ts_free_lock(void * ptr){
  pthread_mutex_lock(&lock);
  bf_free(ptr,0);
  pthread_mutex_unlock(&lock);
}
```

2. Nolock version

   In the nolock version, I just lock the mutex before calling sbrk() and unlock the mutex after calling sbrk(). Then, to achieve thread-safety, I use Thread-local storage (TLS). Specifically, for each thread, it keeps track of its own free list, so there will be no overlapping between each of them. The nolock version gives us a sbrk() level concurrency.

```
//nolock version malloc()
void * ts_malloc_nolock(size_t size){
   return bf_malloc(size,1);
}

//no lock version free()
void ts_free_nolock(void * ptr){
   bf_free(ptr,1);
}

// code section of calling sbrk()
if(is_nolock==0) {
      ans = sbrk(size);
   }
   else{
      pthread_mutex_lock(&lock);
      ans = sbrk(size);
      pthread_mutex_unlock(&lock);
   }
```

- ## Experiments results and Analysis

I wrote a bash script to run 50 times of the measurement test, and get the following result:

	Lock version	Nolock version
Min time	0.117	0.096s
Max time	0.2s	0.137s
Average time	0.148s	0.114s
Min size	41970160 Bytes	41765248 Bytes
Max size	44343488 Bytes	43382960 Bytes
Average size	42757520 Bytes	42617438 Bytes

In this table, we can notice that the nolock version runs faster than lock version according to the time measurement. The nolock version is 23% faster than the lock version. From my perspective, this result derives from the different implementation of the 2 versions. For lock version, all the threads use a public freelist head pointer and ,of course, use the same freelist. As a consequence, the freelist is gonna be extremely long, and thus make it hard to find the best fit free block. For nolock version, as I use thread-local storage, each thread has a private freelist. Because of this, the traversal speed of each freelist in each thread will be faster than the lock version. Hence, it has a faster speed than the lock version.

As for the data segment size result, the difference between the 2 version is quite small. I think it is because that both the 2 versions has almost the same hit rate in the freelists, so that the total data segment size will be almost the same. Hence, the nolock version has no improvement than the lock version.