Project 2 Report (Thread-safe Malloc)

1. Implementation of thread-safe malloc

In order to prevent race conditions in the implementation of malloc and free, which come from the execution of multi-threaded code due to different timing conditions, I use two strategies to solve this problem, including locking version and non-locking version.

1.1 Locking version

I created a LinkedList to implement malloc and free, and I introduced two global variables to track the head and the tail of the free LinkedList, which are head_free_metadata_lock and tail_free_metadata_lock. In multiple threads, the global variables should be locked in a thread in order to keep them from being affected by another thread, so I add locks to these variables. In particular, I initialize the mutex called lock in

```
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
```

Then I add

```
pthread_mutex_lock(&lock);
pthread_mutex_unlock(&lock);
```

for bf_malloc(size, &head_free_metadata_lock, &tail_free_metadata_lock, 1) and bf_free(ptr, &head_free_metadata_lock, &tail_free_metadata_lock) since they call these two variables. Between pthread_mutex_lock and pthread_mutex_unlock functions is the place called the critical section, which allows access to a shared resource by no more than one thread at the same time. I put bf_malloc and bf_free in the critical section, which can allow concurrency and also prevent race conditions. The last parameter in bf_malloc is another variable called isLock to help identify whether I should lock the function sbrk or not. For the lock version, isLock is equal to 1, the variables of the entire function bf_malloc are locked when they are called, so they don't need another lock for sbrk to ensure the thread-safe feature. When we call the function increaseHeapMemory(size_t size, int isLock), we will call sbrk function, but we don't need to add a lock to it.

1.2 Non-locking version

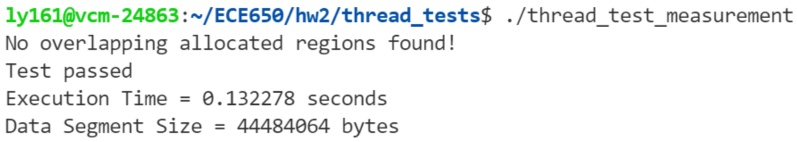
In the non-locking version of thread-safe model, I use Thread-Local Storage(TLS) to make sure the resources in one thread will not be affected by another thread, since TLS allocates separate memory for variables in different threads. So I add the keyword __thread in front of the global variables head_free_metadata_nolock and tail_free_metadata_nolock to implement TLS, which allows concurrency and also prevent race conditions because they don't have the shared global variables. So we don't add pthread_mutex_lock and pthread_mutex_unlock to bf_malloc(size,

&head_free_metadata_nolock, &tail_free_metadata_nolock, 0) and bf_free(ptr, &head_free_metadata_nolock, &tail_free_metadata_nolock). For the bf_malloc function, the variable isLock is equal to 0, which means we should use a lock before calling sbrk and release a lock after calling sbrk because it's not thread-safe. So when we call the function increaseHeapMemory(size_t size, int isLock), we will call sbrk function, it's where we should apply a lock to the sbrk function.

2. Performance result and comparison of locking vs. non-locking version

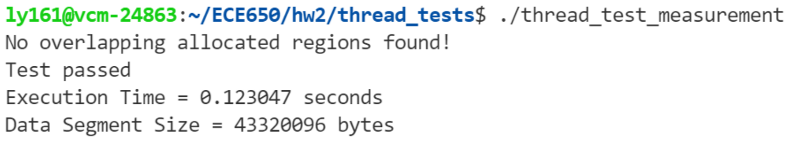
The results below show the execution time and data segment size for the locking version and non-locking version. As we can see, the execution time and data segment size of the locking version are slightly larger than that of the non-locking version. Locking version applies mutex for the variables in the bf_malloc function, where non-locking version only applies mutex for the function sbrk. The locking version has more resources locked in multi-threaded code, which will take longer time for another thread to execute. The non-locking version is more efficient for memory allocation, since more resources are allocated and freed at the same time, which can utilize the space where is just freed or allocated before. The non-locking version is slightly better compared with the locking version given these two measurements.

Locking version:



```
ly161@vcm-24863:~/ECE650/hw2/thread_tests$ ./thread_test_measurement
No overlapping allocated regions found!
Test passed
Execution Time = 0.132278 seconds
Data Segment Size = 44484064 bytes
```

Non-locking version:



```
ly161@vcm-24863:~/ECE650/hw2/thread_tests$ ./thread_test_measurement
No overlapping allocated regions found!
Test passed
Execution Time = 0.123047 seconds
Data Segment Size = 43320096 bytes
```