ECE 650 – Spring 2020
Project #2: Thread-Safe Malloc

Name: Weihan Zhang NetID: wz125

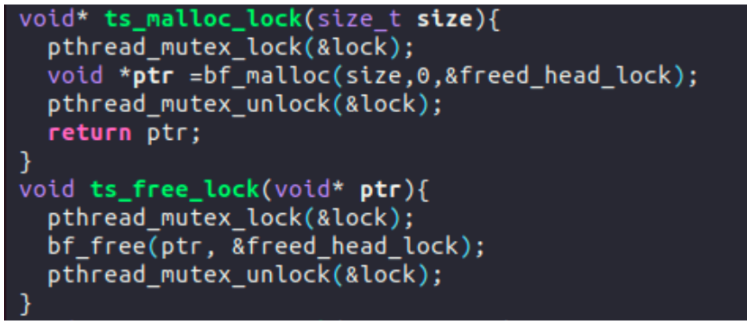
1 Implementations

1.1 Critical Sections

In malloc functions, there are two parts we can think of them as critical sections. One is the situation when we called the not thread-safe `sbrk` function to allocate space on the heap, and the other is when we manipulate the linked list which is used to store the current free list. So, we have to prevent the race conditions occurred in these parts when we implement multi-threading.

1.2 Lock-based Synchronization

To make sure the malloc/free functions are thread-safe, we introduce a synchronization construct which is called mutex lock (mutual exclusion lock) from pthread library to guard a critical section. Since in the malloc and free functions, the focus of implementing is the free list, we could realize it by locking the entire malloc/free functions.

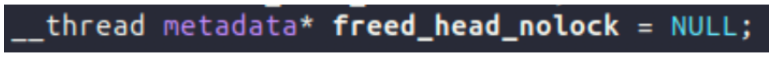


```
void* ts_malloc_lock(size_t size){
  pthread_mutex_lock(&lock);
  void *ptr =bf_malloc(size,0,&freed_head_lock);
  pthread_mutex_unlock(&lock);
  return ptr;
}
void ts_free_lock(void* ptr){
  pthread_mutex_lock(&lock);
  bf_free(ptr, &freed_head_lock);
  pthread_mutex_unlock(&lock);
}
```

Figure 1.1 The implementation of mutex

1.3 No-Lock (Thread Local Storage) Synchronization

In this version, we cannot use mutex, but we can use the Thread Local Storage. Because we are using multi-threading and the free list is a global variable, all threads will access the same variable, and if one thread modifies it, it affects all other threads. The operating system helps us to deal with this problem by using Thread Local Storage (TLS). The purpose of TLS is to associate data with a particular thread of execution. So, by declaring the head of the free list as a TLS variable, each thread can have its free list associated with it. Then, the other part we need to prevent race conditions is just sbrk function. We can acquire a lock immediately before calling the `sbrk` and release a lock immediately after calling the `sbrk`.



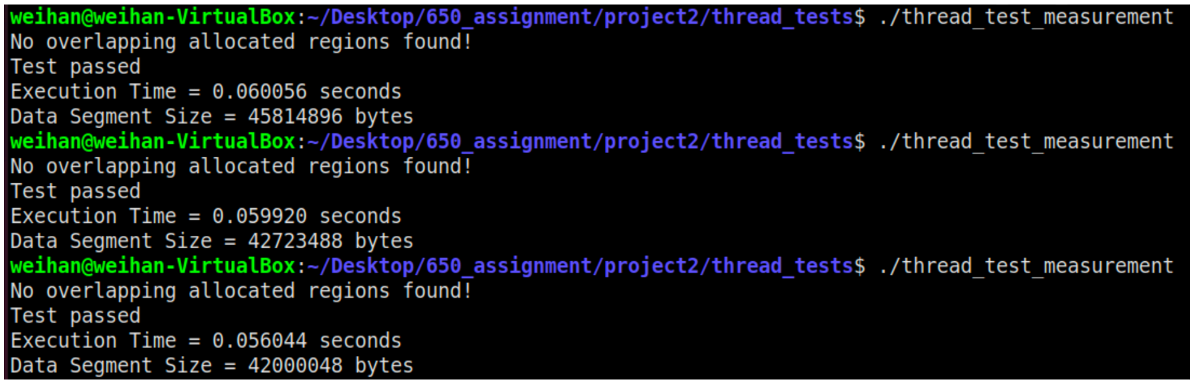
```
__thread metadata* freed_head_nolock = NULL;
```

Figure 1.2 TLS variable declaration

2 Results & Analysis

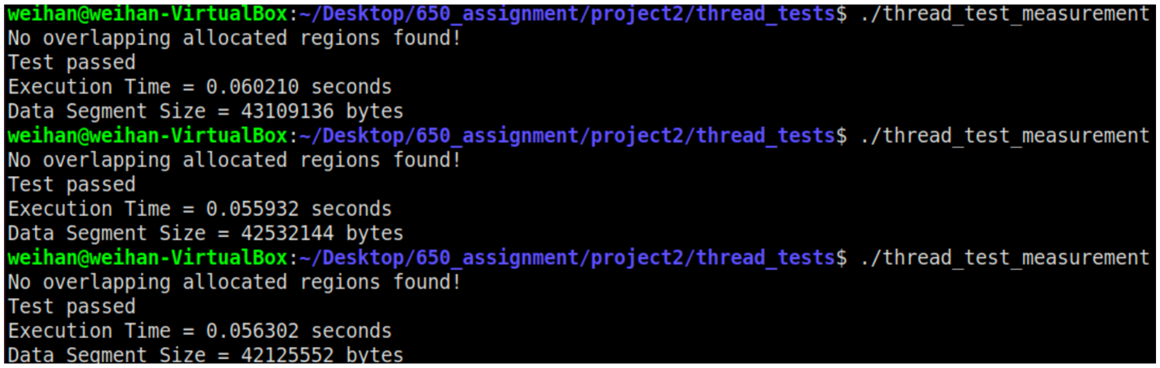
2.1 Results

Both implementations are tested multiple times and the outputs are shown in below:



```
weihan@weihan-VirtualBox:~/Desktop/650_assignment/project2/thread_tests$ ./thread_test_measurement
No overlapping allocated regions found!
Test passed
Execution Time = 0.060056 seconds
Data Segment Size = 45814896 bytes
weihan@weihan-VirtualBox:~/Desktop/650_assignment/project2/thread_tests$ ./thread_test_measurement
No overlapping allocated regions found!
Test passed
Execution Time = 0.059920 seconds
Data Segment Size = 42723488 bytes
weihan@weihan-VirtualBox:~/Desktop/650_assignment/project2/thread_tests$ ./thread_test_measurement
No overlapping allocated regions found!
Test passed
Execution Time = 0.056044 seconds
Data Segment Size = 42000048 bytes
```

Figure 2.1 Lock Version



```
weihan@weihan-VirtualBox:~/Desktop/650_assignment/project2/thread_tests$ ./thread_test_measurement
No overlapping allocated regions found!
Test passed
Execution Time = 0.060210 seconds
Data Segment Size = 43109136 bytes
weihan@weihan-VirtualBox:~/Desktop/650_assignment/project2/thread_tests$ ./thread_test_measurement
No overlapping allocated regions found!
Test passed
Execution Time = 0.055932 seconds
Data Segment Size = 42532144 bytes
weihan@weihan-VirtualBox:~/Desktop/650_assignment/project2/thread_tests$ ./thread_test_measurement
No overlapping allocated regions found!
Test passed
Execution Time = 0.056302 seconds
Data Segment Size = 42125552 bytes
```

Figure 2.2 No-Lock Version

2.2 Tradeoffs

From a theoretical perspective，lock version should have a longer execution time than no lock version in multi-threading. Since the mechanism of mutex lock is that if a thread attempts to lock a mutex that is already locked, then it blocks until the mutex is unlocked by another thread, each thread has to wait for another thread and our program will be run more sequentially. Therefore, the lock version will have a relatively longer execution time. In terms of data segment size, the no lock version will create a new free list for each thread. This could lead to some situations where the current thread could have been malloced the requested space in the free list, but the current thread cannot see other threads’ free list. It will then use the `sbrk` system call to apply a new space in the heap. Additionally, the situations can also occur where two adjacent free blocks cannot be merged because they are invisible in different threads. Therefore, these two situations could lead to a larger data segment size in no lock version implementation. Since the testcase are random and not enough to reach a conclusion, they may show a little different from the theoretical analysis.