Project 2 Report

Name: Zheyou Guo Duke ID: zg69

1. Abstract

This project is to implement malloc and free operation in a thread-safe way. When multiple threads are running simultaneously within one process, their shared heap, static data and code part will interact with each other and generate unexpected results due to race condition. This project uses two methods to solve this problem: Pthread Mutex and Thread Local Storage(TLS). After comparing their results of different number threads, it is concluded that TLS boasts an overall better performance than Pthread Mutex.

2. Design

2.1 Pthread Mutex

The main design idea of this method is to use lock to separate different threads. In other words, one thread will wait for another thread to complete before it takes any modifications to protect critical sections. In this way, multiple threads are executed independently to avoid unexpected interactions.

2.2 Thread Local Storage

This method is to use a static memory which are local to each thread to separate different threads. For static variables, they are global threads variable, shared by all the threads. By contrast, each thread has their own thread-local variables, which are literally local to the specific thread. In this project, there are multiple free lists for corresponding threads. And they are still on heap instead of stack, but with a different layout(shown in figure 1).

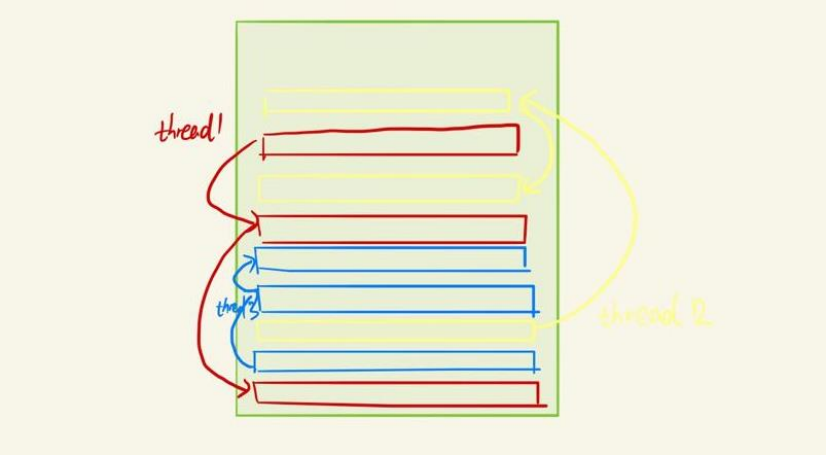

Fig 1. Multiple free lists

1. Result

No lock:

```
zg69@ece551:~/project2-kit/thread_tests$ ./thread_test
No overlapping allocated regions found!
Test passed
zg69@ece551:~/project2-kit/thread_tests$ ./thread_test_malloc_free
No overlapping allocated regions found!
Test passed
zg69@ece551:~/project2-kit/thread_tests$ ./thread_test_malloc_free_change_thread
No overlapping allocated regions found!
Test passed
```

Lock:

```
zg69@ece551:~/project2-kit/thread_tests$ ./thread_test
No overlapping allocated regions found!
Test passed
zg69@ece551:~/project2-kit/thread_tests$ ./thread_test_malloc_free
No overlapping allocated regions found!
Test passed
zg69@ece551:~/project2-kit/thread_tests$ ./thread_test_malloc_free_change_thread
No overlapping allocated regions found!
Test passed
```

	#thread = 1		#thread = 2		#thread = 4		#thread = 8	
	Time (s)	Data Size	Time (s)	Data Size	Time (s)	Data Size	Time (s)	Data Size
No lock	0.136	9357k	0.219	21532k	0.317	43052k	0.558	86686k
Lock	0.136	9357k	0.588	21608k	1.760	43035k	5.071	85988k
Growth Rate	0	0	1.68	0.004	4.55	-0.0004	8.09	-0.008

Table 1. Results of different number of threads

$$(growth = \frac{t_{lock} - t_{no_{lock}}}{t_{no_{lock}}})$$

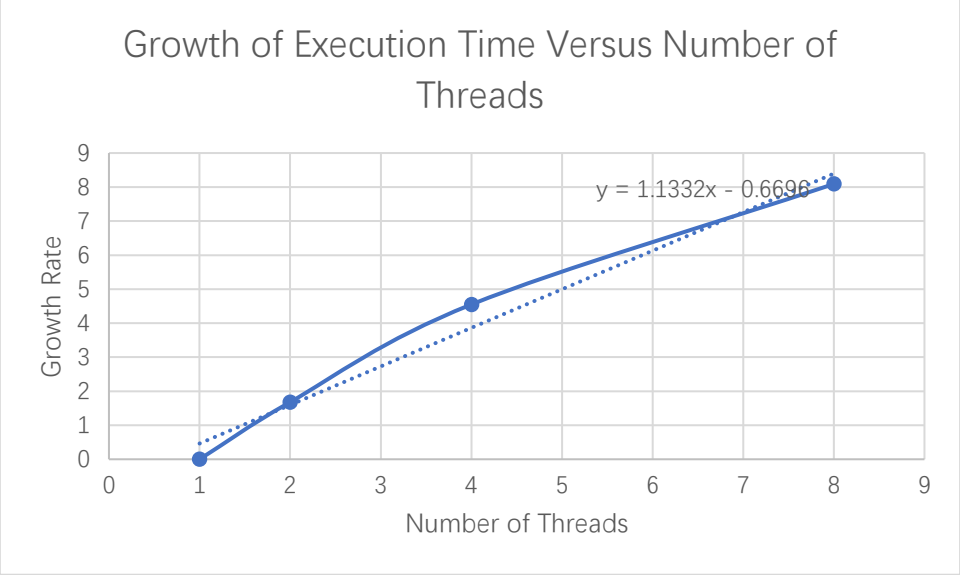

Fig 2. Growth of Execution Time Versus Number of Threads

As shown from those results and table 1, tests are all passed and there is no overlapping allocated region indicated. Execution time of no-lock method is generally shorter than that of lock method. To further explore their relationship, I amplified the range of number of threads to see how it works. According to figure 2, with the increment of number of threads, growth rate of execution time of lock method over no-lock method grows linearly. By contrast, the data segment size remains approximately the same.

2. Analysis

For phenomenon observed above, corresponding discussion and explanation are listed below.

For pthread mutex, the lock actually prevents other threads from being executed simultaneously. For this task, malloc and free, there are too many places where unexpected race conditions happen. To avoid mess, almost every part in free and malloc function is locked and they are executed sequentially. In this way, there is no point to realize multi-threads execution which just saves a tiny amount of time. However, for thread local storage method, if there are four threads, each of them has their own free list, independent from each other. So four threads will be executed simultaneously, saving lots of time.

When there is only one thread, the whole function is executed sequentially, so the two methods cost the exactly same time. When the number of threads increases, it highlights the advantages of thread local method. More threads are used to share the responsibility and thus the execution time of thread local storage method decreases linearly.

For data segmentation size, the value increases with the increasing of number of threads for both methods. That can be explained by the fact that the whole chunk of memory is divided into pieces to different threads. For each thread, the utilization is not as efficient as when the whole chunk is placed to one single thread, since the available resource becomes less for each thread. And there is no much difference between pthread mutex method and thread local storage method. Overall, storage local method is more suitable in the thread-safe malloc and free task.