Project2 Thread-Safe Malloc

Shuyan Sun (ss1316)

February 3, 2022

1. Implementation

The code in this project was wrote based on the code in the project 1, thus the design details of previous code would not be demonstrated. For instance, the struct **block_t** and the logic of **bf_malloc**, **create_space**, **split_block**, **update_block** and **merge_blocks**.

ts_malloc_lock

In the lock version, I used functions **pthread_mutex_lock** and **pthread_mutex_unlock**. I let the program take lock operation just at the start of the malloc function, and take unlock operation just at the end of the malloc function before the function returns. The section between these two operations is called critical section, in which the lock operation only lets one thread enter each time.

ts_malloc_nolock

In the version without lock, I used **thread-local storage** by using pointers of struct with prefix **__thread**.

__thread block_t * head_no = NULL;

__thread block_t * tail_no = NULL;

It would make sure that each thread has its own linked list. In addition, because the sbrk function is not thread-safe, I used a lock immediately before calling sbrk and release a lock immediately after calling sbrk.

2. Results

	Avg Execution Time	Avg Data Segmentation Size
Lock	0.253 sec	43018504 bytes
Nolock	0.110 sec	42460864 bytes

3. Result Analysis

Execution Time

From the table shown above, we find that the execution time of the version without lock is less than the lock version. The reason of it is that in most functions of version without lock, threads can run at the same time, while in the lock version, only one thread can run in the critical section.

Data Segmentation Size

From the table shown above, we could find that their average data segmentation sizes are very similar. The reason is that no matter the lock version and the version without lock, they both used best fit malloc method. In this situation, they both used free space efficiently.