# ece650-project2


- Author: Yang Xu
- netID: yx248

- Data: 02/02/2023
- Instructor: Rabih Younes
- Course: ECE650

# The Thread-Safe Malloc Implementation:

For this project, we should implement the Thread-Safe Malloc. We should implement `4 methods` :

Thread Safe malloc/free: locking version:

```
void *ts_malloc_lock(size_t size);

void ts_free_lock(void *ptr);
```

Thread Safe malloc/free: non-locking version:

```
void *ts_malloc_nolock(size_t size);

void ts_free_nolock(void *ptr);
```

For this project, we should let our best-fit version allocation policy become to thread-save. For multithreading, our policy can work correctly.

Because the code based on the allocation policy code I have completed, I only need to change a little variable name to seprate lock and non-lock version and add two locks into the original code without changing logical and the other codes.

The locking version methods:

For the locking version, I add two locks. One is for `sbrk` function, the other one is for free linked

list operation. I think this two locks can let the `sbrk` function and list operation do not interact at the same time, when an thread is doing `sbrk` , the other thread cannot do `sbrk` but it can do other free list operations, so I think it can be faster than only one lock.

**ts_malloc_lock**

I add a lock `lock_fl` for all free list operations to let the free list operations at the same time. And add the `sbrk` lock for `sbrk` .

**ts_free_lock**

I add a lock `lock_fl` for free list operation, this lock is for all free list operations, find the correct position and write in the free list.

The non-locking version methods:

I use the local thread key words to implement this version. I add a new free list, `__thread md_t *freeList_lt = NULL; // for local thread` , for each thread.

**ts_malloc_nolock**

Because there are separate free list for each thead, so I only changed the original free list to `freelist_lt` , and add the `sbrk` lock for `sbrk` . All freelist operations work in each independent thread, so do not need to care about the other logical and codes.

**ts_free_nolock**

I didn't change anything for original best-fit free, because all free list operations are work in each independent thread.

# Results from My Performance Experiments

Because my results are so close and multithreading causes the results to fluctuate, I ran them each ten times and got the average of the results.

| | no lock | | lock | |
|---|---|---|---|---|
| | time | seg size | time | seg size |
| 1 | 0.059415 | 42826240 | 0.057615 | 42844512 |
| 2 | 0.035327 | 44927104 | 0.056931 | 42774400 |
| 3 | 0.056396 | 42737216 | 0.062094 | 42783744 |
| 4 | 0.047416 | 43785344 | 0.049529 | 42781984 |
| 5 | 0.040283 | 42712448 | 0.066527 | 42778720 |
| 6 | 0.033976 | 44459904 | 0.052248 | 42714976 |
| 7 | 0.043785 | 42707648 | 0.087195 | 42824608 |
| 8 | 0.043032 | 42790400 | 0.039567 | 42729408 |
| 9 | 0.036006 | 43230592 | 0.059311 | 42780352 |
| 10 | 0.045484 | 42744640 | 0.067069 | 42888288 |
| | | | | |
| Avg | 0.044112 | 43292154 | 0.059809 | 42790099 |

# Analysis of The Results

### Execution Time

For the execution time, the lock version is slower than non-lock version. For the lock version, there are two locks, these locks make the other threads will pause, which will have a longer time. The non-lock version only has one lock for `sbrk`, so all threads run without any stop except running `sbkr` at the same time. Therefore, the non-locking version will be faster than locking version.

### Data Segment Size

For the data segment size, the lock version is less than non-lock version. For the lock version, there is only one main free list, so the free list has all free region in the heap, so when do `malloc`, it can read all free region. However, the non-locking version will separate the free list for each independent thread, so when do the `malloc` operation, the thread only can read its own free list

whose size is much smaller than the total free list, so it will do more times `sbrk` operation for segment. Therefore, the locking version's data segment size will smaller than non-locking version.

The locking verion sacrifices execution time and reduces segment size, and the non-locking version sacrifices less segment size and reduces execution time.