ECE650 Project2 Report

1.0 Description of thread-safe model

There are two versions of thread-safe model. One is with lock, and another is without lock. 这 The lock pthread_mutex_t is used. Here is the description of them.

1.1 Version with lock

First of all, the whole process of malloc and free needs to be locked. This will prevent data competition between processes. The lock is applied at the beginning of malloc and free and unlocked at the end of both functions. And the same lock is used for both functions.

```
void *ts_malloc_lock(size_t size){
  //get lock
  pthread_mutex_lock(&lock);
  void* pointer = bf_malloc(size, headNode, tailNode, 1);
  //unlock
  pthread_mutex_unlock(&lock);
  return pointer;
}

//Thread Safe malloc/free: locking version
void ts_free_lock(void *ptr){
  //get lock
  pthread_mutex_lock(&lock);
  bf_free(ptr, headNode, tailNode);
  //unlock
  pthread_mutex_unlock(&lock);
}
```

1.2 Version without lock

This version only needs a lock on the sbrk() function, which is the newNode() function here. So a new function called newNode_lock() is written here.

```
//get lock for sbrk()
pthread_mutex_lock(&lock_sbrk);
void * nNode = sbrk(size + sizeof(node));
//unlock
pthread_mutex_unlock(&lock_sbrk);
```

For the case of not using locks to prevent data contention, the headnode and tailnode of each process should be independent. This means that each process needs to have its own free list and not overlap with each other, so two global variables are set here.

```
__thread node* headNode_nl = NULL;
__thread node* tailNode_nl = NULL;
```

Then modify the overall function to pass these two global variables into the corresponding function.

First, to distinguish between the lock and no lock cases, we add two extra variables to each function: headNode and tailNode. When passing parameters, to prevent shallow copies, we talk about passing in the addresses of the corresponding headNode and tailNode. Second, in order to distinguish between the locked and no-locked cases of sbrk(), we use the mode parameter here to select the corresponding newNode() function.

2.0 Performance result presentation

The error can be reduced by running the test program several times and averaging the results. Here, the two versions were tested 3 times, and the average was obtained as follows.

Table 1 Result of performance

Performance	**No.1**		**No.2**		**No.3**		**Average**	
Version	Lock	Nolock	Lock	Nolock	Lock	Nolock	Lock	Nolock
Execution Time (s)	0.335917	0.257998	0.285704	0.171028	0.198822	0.141318	0.273481	0.22834
Data Segment Size (bytes)	42202608	42681328	42905624	43011848	43279952	43005592	42796061	42899589

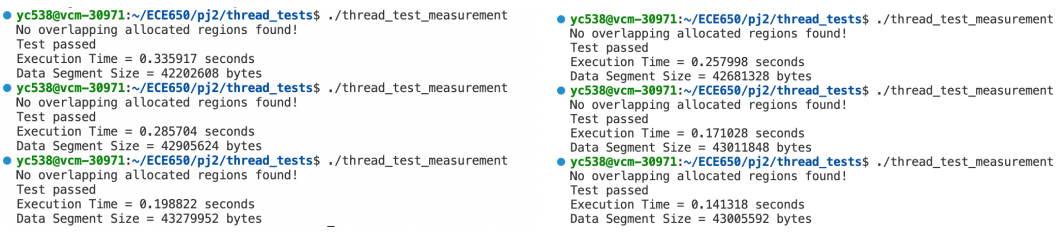

Figure 1 Result of performance

3.0 Comparison of locking vs. non-locking version

According to the above data, the lock version runs slower than the no lock version. But the Data Segment Size of the lock version is smaller than that of the no lock version. This is because lock version puts a lock on the entire process. A large portion of these processes have no data contention, so they do not need to be locked. Locking the entire process affects the overall runtime, as each thread must wait for the other threads to finish before it can continue. The nolock version, on the other hand, does not need to wait and only needs to be locked when sbrk() is called. This saves runtime. But since each thread has its own free list, this leads to space memory not being used to the maximum, e.g. it cannot be merged with other free lists. So, lock will use less memory.

So, it is recommended to run the no lock version if you need to save time and the lock version if you need to save memory.