Anıl Şen - 29556

CS307 PA4 REPORT

Locking Mechanism

There exists two mutex locks in the program to provide mutual exclusion and avoid race condition on the nodes of the heap list. Among these locks, one is for the functions *myMalloc* and *myFree*, and another is for *print* function.

The program is following a course-grained approach on the linked list. Therefore, each thread whichever wants to call myMalloc or myFree, must obtain the mutex as class field named "heapLock". Once this is achieved, a thread does its' necessary operations and before returning, it goes into the print function to print the current situation of the list. Inside the print function, it obtains the lock named "printLock" and prints the list. The lock for print operation might be unnecessary if this print statement will only be used inside the other functions, but it is a must to do if a thread can also call this function arbitrarily inside the main without calling any other function. Therefore, as a design choice secondary lock is preferred to avoid any risk.

As a result, the program achieves atomicity in the areas where mutual exclusion is necessary, such as doing some modifications on the linked list. Since any race condition can easily disrupt the mechanism of the list, global lock must be achieved by the threads to make changes. By using this lock, we ensure that there exists at most 1 thread that can do modifications at any time.

Pseudocodes for the functions

print()

Obtain printLock

Print the list

Release printLock

myMalloc()

Obtain heapLock

Do the necessary allocating operations

Call the print function

Release heapLock

myFree()

Obtain heapLock

Do the necessary freeing operations

Call the print function

Release heapLock

initHeap()

Initialize the list with a single free node

Call the print function