CS 307 Programming Assignment 4 Report

Emir Asal / 27933

I've created a struct 'Node' to store the memory fields for each thread. These nodes are used when I've created a list of 'Node' called heap. Furthermore, for my Heap Manager to function when there are multiple threads "myMalloc" and "myFree" methods are protected by a mutex lock.

1. **initHeap Function**
   Within this function only thing I did was to put the free element within my heap and initialize its values. (id=-1, index=0)

2. **myMalloc Function**
   First, I acquired the lock to make it compatible with multiple threads. Later using a 'for loop' I checked if there is enough space. If there is I first inserted the new memory space for the new thread. But to prevent the element from being added at the end of the list, I used heap.insert and heap.begin functions to insert the node just before the free memory. (Free memory must always be last) Later I've updated the size and index of the free memory. (id = -1) (0 if there is no space left). For the last part informed the user of the status of the heap.
   If there is not enough space for that thread, user is informed accordingly, and returned -1.

3. **myFree Function**
   First, again we are iterating over our heap to find the node with the correct id. If we find it, we set its id to -1 and check for its right and left nodes (if there is any). If it's left or right node is the free node (with id -1) we merge them by removing it and increasing the size of the current node with id -1. Also, indexes are updated accordingly. For the last part user is informed accordingly and the status of the heap is printed. If no thread memory with that id exists, no operation is done, and user is informed.

4. **print Function**
   Since we have "---" between thread memories we need different approach to manage them. First, I've printed the first index within my heap. (index = 0) It will not create problem since we always have at least 1 node within the heap. After that, we iterate over all the elements within our heap. Starting from 1 and ending with heap.size() -1. (If there is only one element this for loop does not run).