CS 307 HW 3- REPORT **Eylül Onar 25320**

In this project, we try to implement a memory management API where a struct with memory size and id are generated by threads. The struct pushed to the shared queue will be popped by the server thread in order to handle the request for memory access. It will check whether there exists available memory and update the thread_message array. Then, unblocked thread will check and read from the thread_message array and update the memory. my_malloc() function has two parameters; thread id and its size. First of all, in order to synchronize the access to the shared data structure, we lock the mutex while the node nd with assigned parameters is added to myqueue. In the server_function, we check all threads in while loop and lock the mutex until the memory server responses to the thread request. The server thread pops the node nd, when it is in the queue and increments the count by 1. It reads from the popped item and returns an answer to the requesting thread depending on the memory size availability. It returns either -1 for declining the request due to no available size, or the start point of the memory location when the memory is available(the node nd's size should be smaller than the memorysize-index). When all 10 requesting threads are done, the server thread terminates. thread_function() creates a random memory size generated between 1 and 25 (memory size/6), then calls the my_malloc() function and blocks the thread until the server thread handles the memory requests. If the request is declined, we display a message saying there is no available memory, if it is granted, thread_function() will access the memory and set all the bytes allocated to its id value. dump_memory() function that will be called in main() prints out the memory array. Lastly, in main(), we create all posix threads in for loop where it takes thread id array as parameter that is also created in main(). Then, we join the threads and call the dump_memory() function in order to print the memory. We then print out thread_message[i] (the memory indexes) in a loop.