My_malloc function creates a node with given thread_id and size, then pushes it into waiting queue. It locks mutex before accessing queue, because it is a shared data.

Dump_memory function locks the mutex first, than access the char array and print it, then unlock the mutex.

Thread_function forms a basis for threads, with given id parameter. First it generates a random integer as size, then calls the malloc function to be enqueued. Then its semaphore is locked to make that thread wait until it is processed. When that thread is unblocked by server function, it checks the message of server function. If the message is -1, there is no memory, print an error; if it is another number, fill the array with the char of that thread starting from that number.

Server_function manages the threads and queue in a right manner, until all of the threads are processed. First it locks the mutex to check queue and char array respectively. If the queue is not empty, take the longest waiting thread (front). If there is enough memory for that thread's size, leave the starting index of that free memory as a message to that thread (in message array). Else, leave the message (-1) as an error. Unlock the semaphore of that thread and also mutex. Finally check whether all threads all processed or not. If they are, terminate the thread.

In the **main**, create an int and thread array with given number of threads. Fill the integer array with thread indexes (increasing order, 0 to num_threads). Then create threads for each number in threads array. Then make them join to the main thread, as well as server thread. Make the main thread wait, until all threads are processed, then print the lastest version of char array, also memory indexes.