In the main function, a thread Id array (thread_id) is created which stores and integer from 0 to NUM_THREADS-1 (number of threads is this case 10) for thread identification. These thread_ids are used passed to the thread_function which is called when each thread is created, and each thread gets a unique identification.

The thread_function function starts by generating a random memory size, srand(time(NULL)) in the main ensures a unique size is generated every time, the id is type casted to integer to be passed to the my_malloc function along with the randomly generated size(the sid1 pointer is dereference to send this id).The my_malloc function when called inside the thread function is enclosed by a mutex because it is a critical region and ensures that one thread with a unique thread_id safely allocates a space in the shared queue achieving mutual exclusion. After a thread's insertion in the queue it is blocked on its semaphore till the server_function processes it request by allocating a value to an index in the thread_message array based on the thread_id(so each thread_id= index in the thread_message which shows the value allocated to a thread by the server_function) .

The my_malloc function takes a thread id and randomly generated size from the thread_functin and creates a new node and fills the node with these details and pushes (in the queue) this as a thread requesting to be processed by the server in the queue.

Depending on the value stored at the index (which is same as the thread_id for a thread) in the thread_message array, the thread_function does one of the following things:

1. If the thread_message value stored at the index same as the thread_id is -1 then it displays an error message saying that there is no available space in the memory. printf is used to output because unlike cout it is thread safe.
2. If the thread_message value stored at the index same as the thread_id is greater than -1, then the thread function then allocates the number of memory blocks in the memory array which are equal to the random size generated by marking them with the respective thread_id(here 48 is added to integer value of the id1 to convert it to char value of the thread_id).The value of the thread_id is stored in the index of memory starting from the index stored in the thread_message for that thread_id. To ensure safe allocation in memory of the id mutex are used to achieve mutual exclusion (this might not be necessary because the server_function only unblock the semaphores of the relevant thread only).

The server_function checks if the queue is empty or not. If it is not empty that means, there are threads that are waiting with requests in the queue and are blocked, this is checked by a global variable count which is incremented every time a thread is processed and the loop terminates if the count id equal to the NUM_THREAD which means all requests have been processed. The server_fuction looks at the first element stored in the queue currently and check if the size that thread is requesting for can fit in the memory by checking the next available space and the space remaining. If size is greater than remaining size, then at thread_message index for that particular thread id -1 is stored indicating to the thread_finction that this memory size is not available and cannot be allocated. Otherwise at thread_message index for that particular thread id it stores the value of the next available index in the memory array. Then the nexy_avail variable in incremented according to the size used by the current thread (next_avail is a global variable which keeps track of the index of next available space in the memory array).After both cases the server_function only unblock the semaphores of the relevant thread and pops or removes it from the queue. The count variable is also incremented here to indicate that the server has finished processing the request of a thread.

After all threads are processed, that is the count is equal to NUM_THREADS, all threads are joined in the main to ensure all of these threads terminate before the main thread. The memory is then printed out by dump memory function which only prints out the value stored at each index stored in the memory array.