An array of threads and an integer array are created, both with size NUM_THREADS. The init function initializes the data structures and creates the server thread. All threads in the created array are started in the thread_function and their respective IDs are passed as parameter to the function. Thread_function generates a random integer between 1 and (total bytes)/6 which is to be the size of the memory to be allocated to the thread. After generating a random memory size, my_malloc function is called by the thread which passes its id and the random memory size to my_malloc. My_malloc gains access to the shared queue- myqueue-and ads a new struct (containing thread id and requested memory size) to the queue. Since myqueue is shared, mutex is used to protect access to it by my_malloc. The thread is blocked by semaphore until the server threads processes its request. Server thread reads the memory requests from the queue, and pops them. If memory is available according to the thread's requirement, server thread appends the thread_message array with the starting address of the memory array on the index of the requesting thread's id. Otherwise, server thread appends the thread-message array with -1. The starting address increases with each subsequent request granted by server thread until new requests cannot be accommodated within given MEMORY_SIZE. Mutex lock is used when server thread is reading from the queue and writing to thread_message array as both are shared resources. After processing the request, it releases the semaphore of the respective thread which had the request and unlocks the mutex. The thread after being unblocked, checks the thread_message array for its request. If request is granted, it sets all bytes allocated to it to the char value of its own id. All threads are joined and after that server thread is joined. Dump_memory function is called to display the memory allocations by printing the memory array. Each byte allocated to a thread is depicted by id number of that thread.