Başak Amasya – 26628

CS307 – HW3 – Report

In the main function, first an array to keep thread_id's is created and all thread_id's are initialized from 0 to 9. Then init() is called, which initializes semaphores, memory and starts the server. After that, all threads are created and starts to run in thread_function.

A random size is generated for each thread in the thread_function and my_malloc is called. my_malloc function creates the node struct corresponding to that thread, using its size and id. Then pushes the node to the queue. At this point, thread does a down on its semaphore (sem_wait function) since it blocks until its request is handled by the server. After it is unblocked by the server, it checks the thread_message array. If it is given the memory requested, it writes "size" many id's as char to the memory. But if corresponding entry is -1, it prompts an error saying that there is not enough memory.

server_function checks myqueue in a loop, until all threads are processed. If there is an item in the queue, it pops the thread request and checks if there is enough space in the memory. If that is the case, it writes the index of the starting memory place to the corresponding id^{th} thread_message entry and updates the index by incrementing it by size. If that' not the case, it writes -1. After server processes thread's memory request, server_thread function does an up on the corresponding semaphore (sem_post function), so that the thread is unblocked and it can proceed.

2 global variables (index and remaining_threads) are added to given variables. Index keeps the index of the next available memory space, is initialized as 0 and updated as each thread is allocated its size from the memory. remaining_threads variable is used for keeping the number of threads not processed yet by the server. When it becomes 0, meaning all thread requests are handled, server_thread function is terminated. Also, since myqueue is a shared data structure, its access should be protected by locking the mutex before using it and unlocking it after we are done using it throughout the program.

All threads are joined in the main before printing out the memory so that there is no synchronization problem. After all threads are done, content of the memory is displayed using dump_memory() function, then memory indexes are displayed and program is terminated successfully.

It can be observed that if a thread's memory index is displayed as -1, it has also prompted "Not enough memory" error. Also, the output is different in each run because memory size is generated randomly. But I observed that, since the random size generated is less than MEMORY_SIZE / 6 and there are 10 threads at total, in most of the runs all threads can be allocated to the memory and error messages are less likely to occur.