CS307 HW #3 – Memory Management API

Samuel Lee

24774

12/12/2020

For this homework, we were ask to implement a memory management API, where we had a server thread – which handled all memory requests and 10 different threads that were requesting memory. In the beginning of the program, the threads array is declared, and is initialized using the “pthread_create” method, sending it to the thread_function along with the id of the thread.

Inside the thread_function, I used the standard random library for requesting memory values between 1 and 25. Then, my_malloc function was called, giving the id and requested memory value as the parameter. In the my_malloc function, the sharedLock mutex was locked when entering critical region to create and add the node to the queue, and exited critical region afterwards. In order to wait for the server thread, a simple method “sem_wait(&semlist[threadId])” method was used to block.

In the server thread, we created a while loop, that will continue until all threads were handled. In order to check this, I created a global variable, and incremented this value whenever memory request by thread was handled. Once there was a pushed request on the queue, I first enter the critical region, initialize a node that is to be popped from the queue. Afterwards, we checked if there was memory available for the requested memory size, and if there was, I wrote the starting index of available memory in “thread_message[threadId],” otherwise, the value was set to “-1.” In order to keep track of next available memory index, I created an index variable, and updated it once memory was found and allocated to the specific thread. Once all operations are completed, the “sem_post(&semlist[threadId])” method was called to unblock the specific thread inside the thread_function, and we exited the critical region.

Once the server thread unblocked, then we check the value of “thread_message[threadId]” and if the value is greater than -1, then we set the memory from the starting point index to the requested memory size. Otherwise, we alarm the user that there is not enough memory.

In the main thread, the 10 threads are first called to join, then the server thread. The main function will also call the dump_memory() function to print out the memory array, along with the thread_message[threadId] array to check the corresponding memory allocation and starting index for each thread.

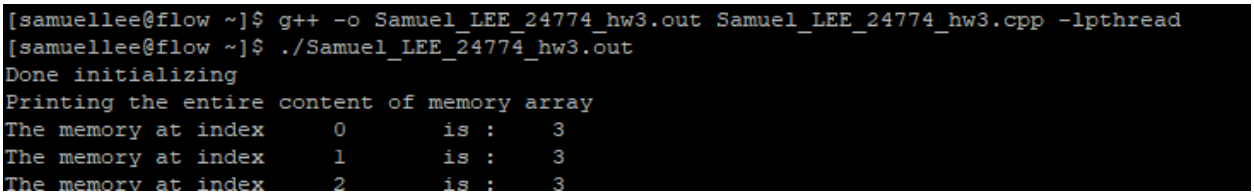

Figure 1. Working code on Flow server

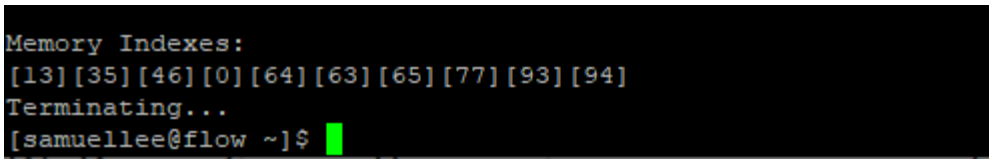

Figure 2. Starting index for each thread in memory