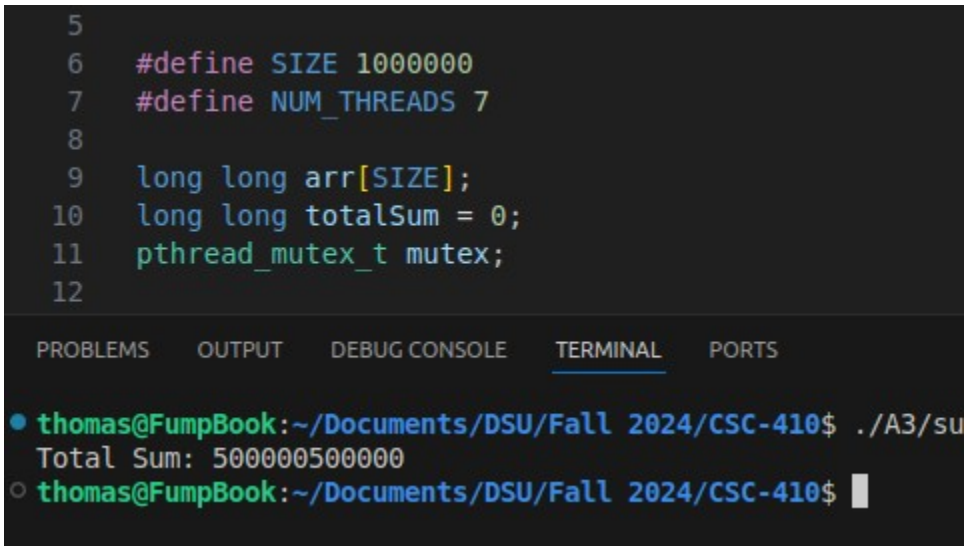
I used a mutex lock to handle safe access of the critical section, which is the accessing of the totalSum. By ensuring that only one thread accesses the totalSum at a time, race conditions are avoided, so incorrect sums are prevented. The chunk size was updated to have the final thread handle any remaining elements, so if there is an uneven number of threads, the last thread will handle slightly more than others. Below is a screenshot demonstrating a correct summation of the array with 7 defined as the number of total threads.



In the case of the matrix multiply program, synchronization techniques are not necessary. Multiple threads may be reading from arrays A and B simultaneously, but they are never writing to them after the thread creation. Initialization of A and B are done before thread creation, and each thread is writing to a separate portion of the C array, so no race conditions are possible. In order to solve the issue of rows not being divisible by the number of threads, I modified the data structure to include start and end, where the last thread will handle any remaining rows. Below is a screenshot of the multiplication working as expected, with a total of 7 threads.