EC527 Assignment 4 Chen-Yu Chang

**Task 1:**

Code Session

**Task 2:**

The output changes every time I compile.

**Task 3:**

Code Session

**Task 4:**

When sleep function is added to work function, there is nothing printed in the work function since it causes the thread of sleeping for 3 seconds, which makes the main function to finish before the threads are completed.

**Task 5:**

After the sleep function is moved to the main function, there will be a 3 second delay for printing the statement.

**Task 6:**

The “After creating the thread” is printed first followed with the work statements. It is different from task 4. The statement is printed since pthread\_join makes sure that each thread is complete before going to the next one.

**Task 7:**

When we change the type to signed char, it does not print the threaded, instead of a NULL value.

**Task 8:**

When you change the value of f and \*g, I found that both change the output while the only difference of changing f and \*g is that the last row indicating the i value. When f changes, the i value does not change; however, when \*g is changed, the i value also changes since the location where the variable is stored changed.

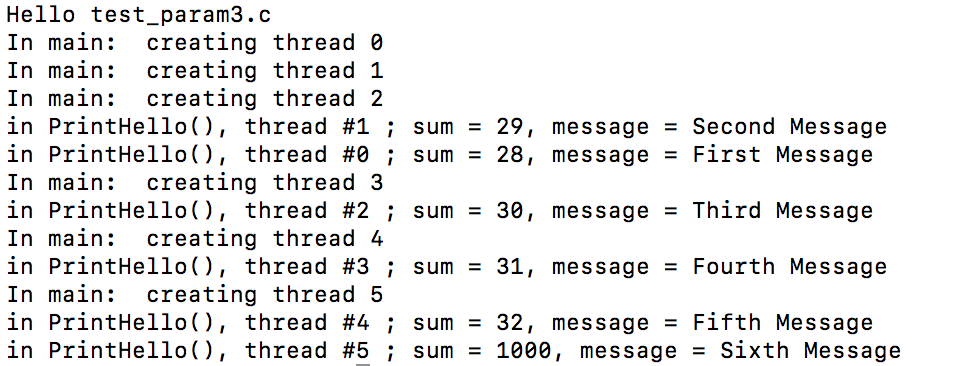
**Task 9:**

Code Session

**Task 10:**

I add a global variable set to 0, and it is increased by 1 every thread to make the thread get the unique value of the array. However, in the printed threads, some of them displayed the repetitive value in the array since they receive the same global variable’s value at the same time. Yet, I printed the final array with all elements multiplied by 5 in the thread, it printed out well.

**Task 11:**



**Task 12:**

Code Session

**Task 13:**

It does not have an output. Syncing multiple threads with only one mutex lock has a problem.

**Task 14:**

After uncommenting the sleep function, all the threads were created before printing statements. I tried to change the sleep function with variables, and it turns out still working since it prints after barrier statements.

**Task 15:**

The reason why the loop being executed after threads is execute works is that there is a mutex lock before the join loop, which is unlocked after all the others do.

**Task 16:**

Code Session

**Task 17:**

The balance is correct, but when I add the sleep function to it, the balance ends up with 999. When I increase the number of threads to 10000, the balance still stays accurate without sleep function.

**Task 18:**

Code Session

**Task 19:**

No, the answer is incorrect. It needs up to 1000000 iterations to be close to steady state values, while it takes 5.81 seconds.

**Task 20:**

When barrier is changed to 1, it only needs about 10000 iterations, which is much less than the iteration needed for barrier of 0. The time it takes for barrier of 1 is also smaller, so it is faster. Having barriers also make the result more consistent. With barriers, the reduced time and iteration attribute to making sure that all the threads are done for each iteration. Redundant iterations are not needed so that the elements will depend on the neighbor values. Also, the elements are updated consistently the same, so it gives us a more consistent result.