Will May ([wdm0032@auburn.edu](mailto:wdm0032@auburn.edu))

COMP-3500-002

Homework 3

**Part 1:**

Please open attached file “*HW3\_pt1.cpp*” for source code.

Open “*input.*txt” for input file.

Open “*output\_part1.txt*” for output file.

(Compilation and execution instructions are included in source code file.)

**Part 2:**

Please open attached file “*HW3\_pt2.cpp*” for source code.

Open “*input.*txt” for input file.

Open “*output\_part2\_run1.txt*” for output file for first run.

Open “*output\_part2\_run2.txt*” for output file for second run.

Open “*output\_part2\_run3.txt*” for output file for third run.

(Compilation and execution instructions are included in source code file.)

**2.)**

**b.)** I did not notice any difference in any of my results in the part 2 implementation. In this implementation, I created and assigned my pthreads in main(). By doing this, each pthread was able to call the read\_data() function, where the integers were assigned groups. Then at the end of the read\_data() function, it calls the display\_data() function which is a method to display the assigned group data.

I believe that since main() created, assigned, and joined only pthreads and made it call the read\_data() method, that the pthreads were able to ensure race condition did not occur. If race condition occurred, then there would be a large discrepancy in the data and after subsequent runs, the data would most likely change. Thus, in my code I was able to correctly implement pthreads to ensure race condition did not happen.

**Part 3:**

Please open attached file “*HW3\_pt3.cpp*” for source code.

Open “*input.*txt” for input file.

Open “*output\_part3.txt*” for output file.

(Compilation and execution instructions are included in source code file.)

**2.)**

**b.)** In all the Part 1, Part 2, and Part 3 outputs I observed no changes in data. This is most likely due to the main() function only creating, assigning, and joining the pthreads, which calls read\_data(). By only allowing my main function to execute these statements, I was able to ensure race condition did not occur, nor any other negative conditions.

**c.)** The synchronization approach that I used in the Part 3 implementation included mutexes. I assigned the mutexes in the read\_data() method to limit the possibility of race condition or any mixing of data that could result in incorrect outputs. In the read\_data() method, I created and initialized eleven different mutexes (I added another mutex to allow it to gather the total integer data). This approach ensured that each group would be assigned a mutex, thus allowing the group to be unlocked then locked. This also ensured that the “critical section” could only be accessed when assigning integers to groups.