COP 4610 Project 1 Report

The output of our program without synchronization gave multiple lines of output that claimed that the pthreads were all seeing different values for our shared variable. This occurred because all the pthread were trying to modify the pthreads at the same time and as a result race conditions occurred. It must be noted that when you run the program with the command line argument of 1, everything seems to work fine but as you keep increasing the number of threads that you want to run the race condition is more noticeable. For instance, when you have two threads running at the same time you get two different final outputs. The same thing occurs for when you try it with 3, 4 and 5 as command line arguments.

The output of our program with synchronization gave multiple lines of output that claimed that the pthreads all incremented the value of the shared variable by one and then in the final output it showed that all the pthreads had the same shared variable value. The reason this occurred is because we implemented a mutex into our code, which locks whenever a pthread is performing an operation and then unlocks once it is finished. The locking operation of the mutex eliminates the race condition since only one pthread has access to the shared memory at a time. In this code we also added a barrier which makes all the pthreads wait until the others are done in order to print out their final value at the same time. In all the test runs for our program, all the pthreads saw the same final value, showing that with the addition of the mutex and the barrier the race condition problem was fixed from our unsynchronized version.

The reason why the unsychronized version had race conditions and the synchronized version did not is because all the pthreads were modifying the shared variable in the unsynchronized version without waiting for the other pthread to finish modifying it. In other words, one pthread may be adding 1 to the shared variable and another can also be doing the same thing at the same time causing the shared variable to skip one incrementing cycle. The mutex added in the synchronized version avoids the race condition since it creates a key and lock analogy in which one and only one pthread can modify the shared variable once at a time until it is unlocked for another pthread to use.