Joseph Pennington (2912079)

EECS 678 Lab07 Report

1. Due to the fact the Linux uses a preemptive scheduler with threads, this leads to the inconsistency of the final value of count. The preemptive scheduling causes synchronization issues if mutual exclusion is not used. Without mutual exclusion, each thread may enter a race condition and attempt to read, modify, and write the same value to memory at the same time. Overall, those issues then lead to the inconsistency in the count variable.

2. The probability of the final value of count being inconsistent as the loop bound increases occurs because the more times the loop is ran, the more opportunities there are for the threads to enter the race condition. On the other hand, if the loop bound is small, there is less of a chance for the threads to enter the race condition because the loop is executed fewer times.

3. Local variables that are printed out are always consistent because print is called within the context of the executed thread. This thread uses its own stack instead of the overhead process' stack.

4. My solution ensures that the final value of count will always be consistent because it uses the lock function to enable mutual exclusion. This means that the count variable is only being modified in the critical section of each thread and that the other threads are not able to access their critical sections early.

5. I think that the times for each version is different because mutual exclusion does not allow the threads to preempt each other. Preemption allows the program to be executed as fast as possible given how long each task will take. Without preemption, each thread must finish before it starts the next thread. This then increases the total amount of time it takes the process to finish and leads to the different times for each version.