Brock Sauvage
EECS 678
3/05/17
Lab 5 Report

1. Since Linux utilizes preemptive scheduling when dealing with threads, this creates problems with synchronization if no measures are taken to prevent them. Without using mutual exclusion, all of the threads are subject to race conditions, where they are all trying to read, modify, and write the same value to memory at the same time. This creates the inconsistency in value that we see printed out to the terminal, even though all of the threads are able to compute and print the value perfectly fine within their own contexts.
2. Essentially, the more times you execute the loop, the greater chance you have for the values to be inconsistent and subject to race conditions. This is a sort of "Law of Large Numbers" type problem, where your probability for inconsistency rise with the number of loops executed.
3. The local variables printed out are always consistent because the print occurs within the context of the executed thread, which utilizes the stack of this thread, rather than the stack of the overhead process.
4. By using a lock for mutual exclusion, we can guarantee that when the "count" value is being modified in the critical section of a single thread, no other threads are able to access their critical sections and preempt the executing thread. This creates consistency among the threads and solves our synchronization problem.
5. I think that the times are so different because when using mutual exclusion, threads are not allowed to preempt each other. In this context, the point of preemption is so that the program can be executed in the quickest time possible based upon the length of the tasks on hand. When preemption is no longer allowed, each thread must execute completely before moving on, increasing the time is takes the process to finish.