Warren Scipio

Lab 5 report

1. The threads are taking a value from memory almost at the same time and incrementing it but because they can access the value while another is incrementing it can completely ignore or lose the incrementing done by the thread before. Putting in the locks/unlocks allows the threads to completely finish the job they are doing on the specific value in memory and once it is done and unlocks the next thread can begin, with no overlap.
2. The reason it is able to do the smaller numbers is because since the numbers are smaller the first thread finishes incrementing before the next thread calls/uses the value. Although it is still not perfect, even counting to the number one, if run enough times I can get a two as the final answer and thus meaning one thread started before the last updated or finished incrementing the value.
3. Within the frame of reference of the for loop only one thread actually updates the variable loc, and that is what is printed for each thread to make sure that the individual thread got to the number it was suppose to get to.
4. Its simple, it just doesn't allow any thread to update/increment a variable if another thread is acting on it, the loops are set up to use the two arguments that the user inputs, the first being the number for the loop the second being the number of times to run that loop.
5. Since there are some things that I don't need to know, I will give the simple version. The one with the locks/unlocks took longer because the threads had to wait until the locks unlocked so they could continue updating their variables whereas the one without the locks/unlocks in place didn't have to wait so the overall run time was faster.