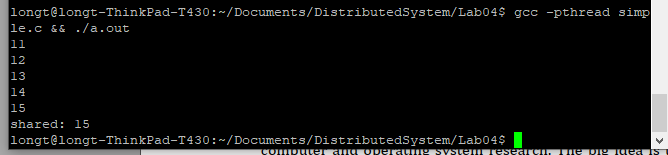
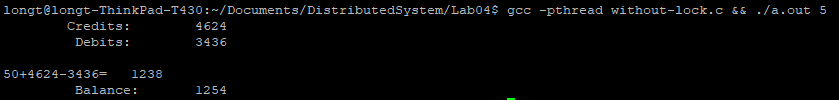
Question 4:

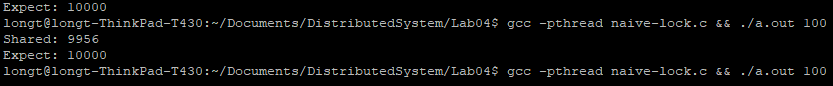


Question 5:

Without increase the number of threads and value of NUM\_TRANS, I can already see the different between balance and the equation INIT\_BALANCE + credits – debits. This occurrence happened because some of the loop function in our 5 threads access the same resource balance or debits, credits concurrently without waiting for other thread to finish its execution on the so-called resources.

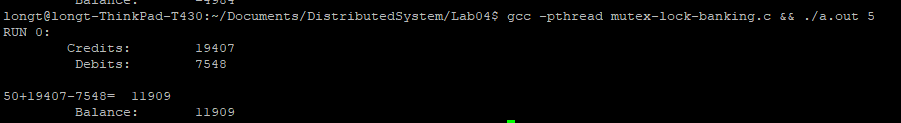


Question 6:



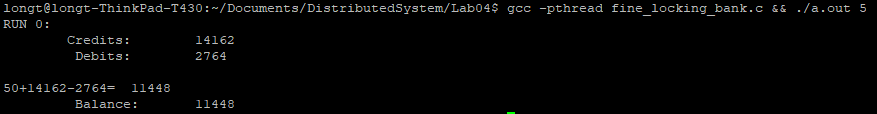
We use lock variable with the hope to identify that only 1 specific thread can access to resources, in this case, *shared* variable, but the difference still occurred since there will be a time that 2 or more thread that accessed to *lock* variable when its value equals to 0 and increase shared at the same time, continue the loop, so that the shared cannot exceed number of loop multiply with 100, the difference happened at line shared++ which is shared = shared + 1, the shared on the right hand side will be, for example 2 concurrence equation shared = shared + 1 and shared = shared + 1, if it happened at the same time, shared on the right hand side of the 2 line of code will have the same value thus assign the same value to the new shared, then 2 equations this time will only have effect as 1 equation 🡺 shared variable is 9956 when it is expected to be 10000.

Question 7:



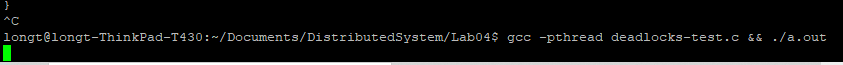
After using mutex for locking critical sections, the experiments show no difference between balance and its expected value 🡺 The improvement can be seen.

Question 8:



Without experiment on large resources of code, we can predict that fine locking strategy give a faster runtime since its decrease waiting time of other thread when access to a specific amount of resource. But in the example code, since there are only 2 type of resources which is balance and credits versus balance and debits, the runtime of fine locking is slightly slower than coarse due to the lock and unlock operations.

Question 9:



In the experiment we cannot see the output since the two thread are blocking each other by the mutex lock\_a and mutex lock\_b, thread 1 lock mutex a and thread 2 lock mutex b cause the other thread cannot keep its operation going since thread 1 need mutex b of thread 2 is locking and thread 2 need mutex a of thread 1 is locking 🡺 deadlock.