Lab 7

Name – Amit Malav 130050032
Lokit Kumar Paras 130050047

Part A

1)

Count -> 69125
Time taken -> 1384

Count -> 52669
Time taken -> 1747

Count -> 46292
Time taken -> 1686

Avg Count = 56028.66
Avg Time = 1605.66 micro sec

2)

No the value does not match N* K = 100000

Line responsible -> count = count+1
here the increment is not done atomically, hence one thread can read the value of count before the other thread has saved the new value of count. Hence both of them set the value of count to same value, where actually the value should have incremented twice.

Part B

1)

Count -> 65717
Time taken -> 1834

Count -> 82726
Time taken -> 1459

Count -> 96132
Time taken -> 706

Avg Count = 81525
Avg Time = 1332

2) here the reason is it is not performing the following instructions atomically

- checking the condition using locked at while(locked)
- updating the value of locked to 1, so that no other thread can enter the while loop

Single Core
The race conditions occurs when context switch happens after checking the conditon for while loop, but before the value of locked is set to 1. Hence two or more threads can enter the while loop at this point, as they all see the value of locked =0

Multi Core
When there are many processors with threads distributed among them, if two or more threads enter the while loop at the same time before the value of locked is updated by any one of them, this will lead to race conditions.

Part C

1)

Count -> 100000
Time taken -> 20041

Count -> 100000
Time taken -> 18211

Count -> 100000
Time taken -> 17070

Avg count = 100000
Avg Time = 18440.66

2)

Here we have avoided any race conditions by using the pthread_lock API, which implements mutual exclusivness using mutex locks. When pthread_mutex_lock() returns, the mutex is locked and the calling thread is the owner. If the mutex is already locked and owned by another thread, the calling thread blocks until the mutex becomes available.