Program Design:-

- I have declared an abstract class Lock in both of the program. Lock class have two pure virtual function i.e lock() and unlock(). This means any derived class inheriting from abstract class, must implement these two functions.
- As according to problem statement, the input to the our program is a file named inp-param.txt
- I have used two global variable named totalEntryWaitingTime and totalExitWaitingTime for calculating average waiting time for threads to enter and exit from Critical Section.
- The main function and testCS function used in the program is as per the problem statement.
- Filter_Lock class has been created for implementing filter lock and ptl class has been created for implementing Peterson tree lock.
- Every class which is used for locking and unlocking have a constructor and destructor function, for run time allocation and freeing of memory.

Filter Lock Algorithm:-

I have used the same algorithm that is described in the book. In the constructor I have created two dynamic array level and victim as given in the pseudo code. Rest of the program also follows the pseudo code given in the book.

Peterson Tree Algorithm:-

There are n number of threads given, so consider a binary tree consisting of 2n-1 nodes, where node-0 to node-n-1 are normal two thread Peterson locks and from node-n to node-2n-1 are the threads given.

We have taken thread id as 0,1,2,..n-1 for n threads, so each thread will be mapped to n+ thread_id nodes of binary tree. So by this way we can map each thread to leaf of the tree.

Algorithm:-

Acquiring the locks:-

1. Suppose that there are n threads that want to access to a critical region.
2. The first step uses n/2 two-threads Peterson locks. Then two threads are assigned for each two-threads Peterson lock, i.e (n/2)th lock is applied on nth and (n+1)th thread, (n/2+1)th lock is applied on (n+2)th thread and (n+3)th thread and so on. At the end of this step only n/2 threads acquired the lock.

3. Similar to the first step, In second step, n/2 winner threads of leaf level uses n/4 two-threads Peterson locks and two threads are assigned for each Peterson lock. From here, we will get n/4 winner threads for upper level.
4. This procedure continues up to it reaches the root, where it's necessary only one Peterson lock. The thread that acquires the last Peterson lock can enter to the critical region.

Releasing the locks:-

1. For releasing the lock, we will simply release all the locks which had been acquired in the traversal from leaf node to root.

Program Description:-

We have created a 2d array, whose dimension is (n-1) * 3, where n is the number of threads.

- n-1 corrosponds to the number of two-thread Peterson locks, which are in the internal nodes.
- For every locks(row in the array) we have three column, 0^{th} column corresponds to current thread of two-thread Peterson lock, 1th column corresponds to other competing threads of two-thread Peterson lock, and 2^{nd} column corresponds to victim among two competing threads.
- Consider an example containing 4 threads , where threads have id 0,1,2 & 3 respectively.
- Initially our 2d array is like this:-

	0	**1**	**2**
0	0	0	0
1	0	0	0
2	0	0	0

- Suppose each thread will manage to run first two line of for loop of lock function, before getting context switched by other thread. It means thread 0 and 1, perform manipulation on, 1th row of the array and also suppose 0^{th} comes first. Similarly during the first iteration, thread 2 and 3 will do array manipulation on 2th row, and also suppose thread 2 execute first lines of loop before thread 3. So now array will something like this :-

	0	**1**	**2**
0	0	0	0

1	1	1	1(th1 victim)
2	1	1	1(th3 victim)

So it means, th0 and th2 is the winner and they will compete for the 0^{th} row lock, by same logic.

Suppose th0 wins again, so now the table looks like this:-

	0	**1**	**2**
0	1	1	1(th2 victim)
1	1	1	1(th1 victim)
2	1	1	1(th3 victim)

So now th0 can enters into critical section.

Now, when th0 exit critical section, it will call unlock function, which will make arr[0][0] = 0, which enables th2 to enter into critical section, similarly in second iteration arr[0][2] = 0 will make th1 to move a level higher, and this process goes on, till each thread enters into critical section.

Note:- Sometimes when the number of threads are higher, in some system program will get dumped. The reason is unavailability of memory in that system at that point of time. I have run my program on the workstation provided in the lab,having system specification of 16GB RAM and 20 core CPU. So if it is getting dumped in your system, I request you to kindly run 3-4 times on same input.

Performance Comparision:-

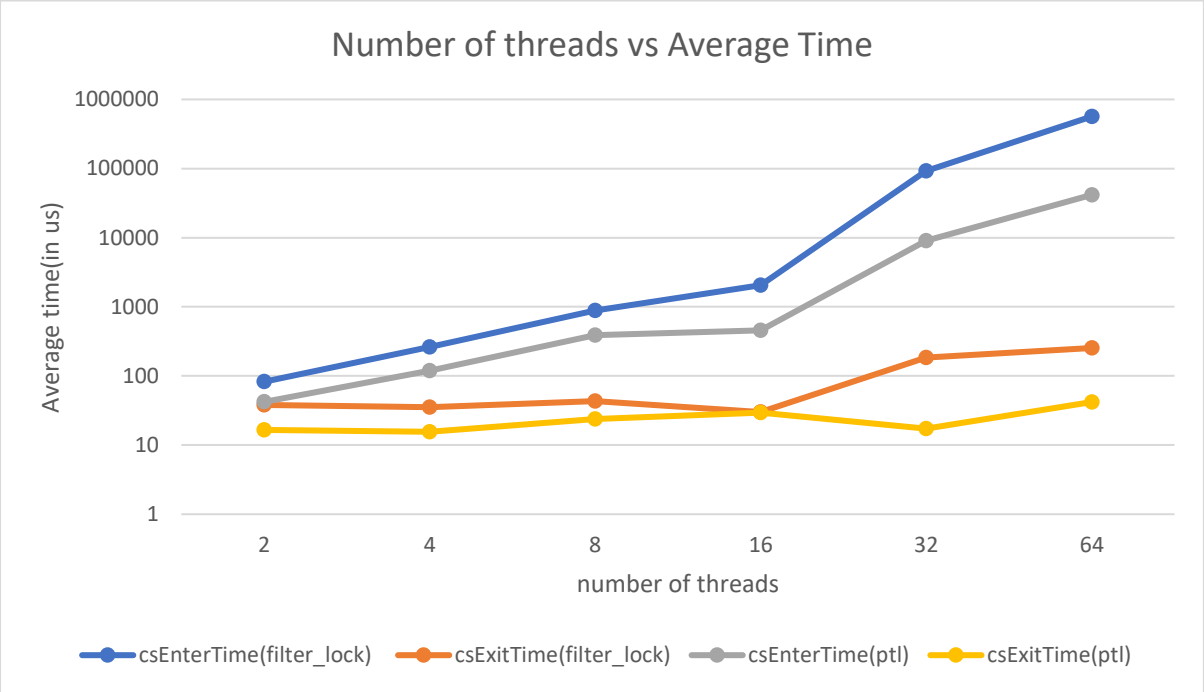
For each number of threads, I have run my program three times and taken average of the result obtained.

Number of thread is varying between 2 to 64, in the power of two.

K = 10, time1 = 10, time2 = 15

All time are in microseconds.

Number of threads	csEnterTime(filter_lock)	csExitTime(filter_lock)	csEnterTime(ptl)	csExitTime(ptl)
2	82.55	37.9	42.15	16.55
4	263.25	35.20	118.825	15.575
8	887.12	43.33	389.688	23.72
16	2060.53	30.06	457.031	29.525
32	92355.8	183.8	9038.31	17.221
64	566676	253.92	41670.1	41.96



Analysis:-

The average time taken by thread for acquiring a lock in Filter lock is more than that of Peterson tree lock because in filter lock a thread has to traverse n-1 level before obtaining a lock to enter into critical section, whereas in Peterson tree lock, thread has to traverse only log n level.

In case of unlocking, it is rather surprising that , average taken by a thread in filter lock is more than that of Peterson tree lock. In case of Peterson tree lock, a thread needs to traverse log n times, whereas in case of filter thread needs to execute only one statement i.e making its corresponding index in level array = 0. But the anomaly is that filter lock is taking more time for unlocking than ptl. The reason may would have been immediate context switching between threads in filter lock as opposed to Peterson tree lock where some more manipulation is required before getting lock.

Conclusion:-

Filter lock is taking more time than Peterson tree lock for getting into critical section.

Filter lock and Peterson lock, somehow taking similar amount of time while existing from critical section.