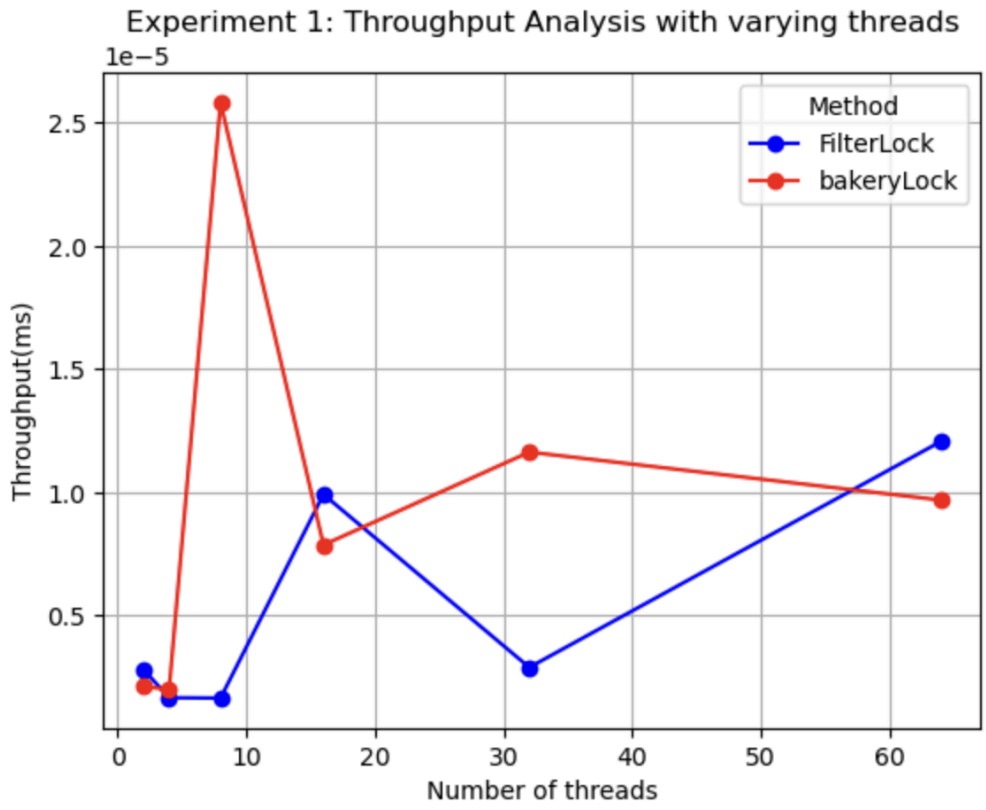
# CS5300 - Parallel & Concurrent Programming:
# Autumn 2024
# Programming Assignment 3:Comparison of Filter Lock and Bakery Locks

**Name: Dheeraj M**
**Role Number: EE21BTECH11015**

Goal: The goal of this assignment is to implement the two locking algorithms discussed in the class: Filter lock and Bakery lock. Then compare the performance of these locks by measuring two parameters: average waiting time for threads to obtain the locks and throughput. You have to implement these algorithms in C++.

## 1) Experiment 1: Throughput Analysis with varying threads:

In this experiment, you will focus on scale-
ability by increasing the threads. The y-axis will represent throughput,and the x-axis will vary the parameter n (Number of threads) ranging from 2 to 64 in powers of 2.



**Observation**: Throughput increases as the number of threads (n) increases up to a certain point, after which it either plateaus or degrades slightly.

The **Bakery Lock** typically shows a gradual decline in performance as the number of threads grows beyond a certain point, whereas the **Filter Lock** can handle higher thread counts more gracefully.

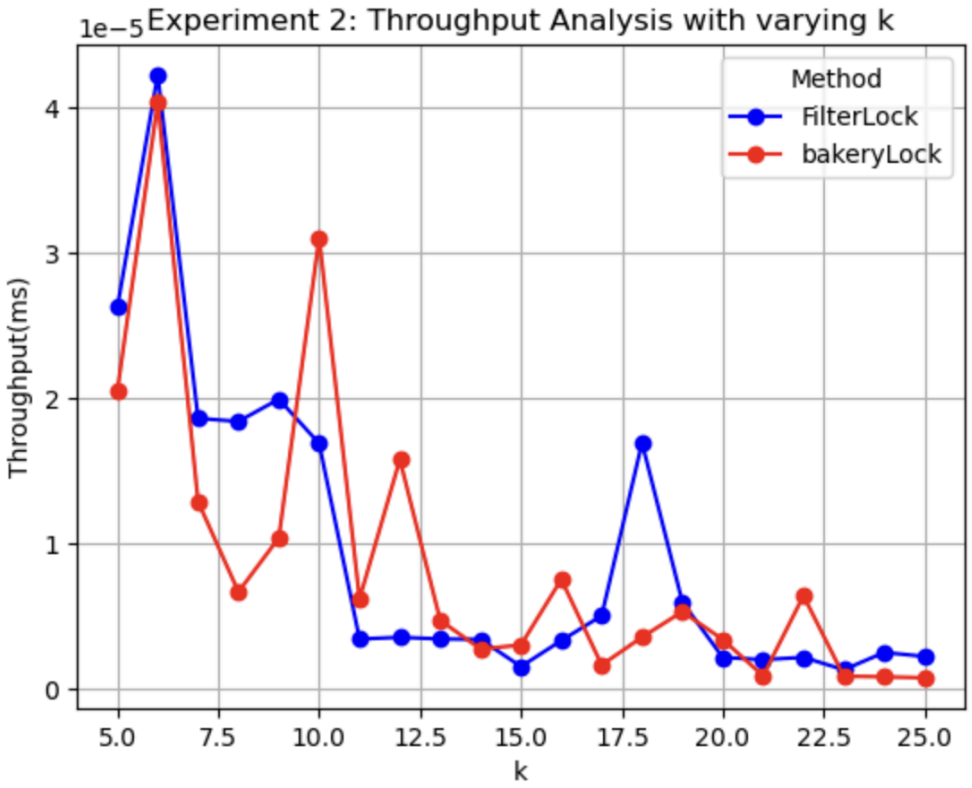
**Reasons for This Behavior**:

1. **Filter Lock**: The Filter Lock divides threads into levels, with each level filtering threads before they can enter the critical section. As the number of threads increases, the hierarchical filtering reduces contention since only a smaller subset of threads reaches the final level where contention is highest. This structure allows for better throughput scalability because the lock doesn't become a bottleneck immediately with increasing threads.
2. **Bakery Lock**: The Bakery Lock ensures that threads enter the critical section in the order they request access, which ensures fairness. However, this fairness comes at a cost—every thread must check and recheck the tickets of every other thread, leading to increasing overhead as n increases. This leads to a more noticeable performance degradation with high thread counts as the lock acquisition mechanism becomes slower and more complex with higher contention.

**Key Takeaway**: At low to moderate thread counts, both locks perform well, but as n increases, the Bakery Lock's complexity causes throughput to decline, while the Filter Lock's hierarchical filtering mechanism allows for better scalability.

## 2) Experiment 2 : Throughput Analysis with varying k

In this experiment, the primary focus is on throughput analysis. The y-axis of the graph signifies throughput, measured in tasks executed per unit of time, as described earlier. Meanwhile, the x-axis is designated for varying the parameter k within the range of 5 to 25.




**Observation**: Throughput increases with higher values of k (the number of times each thread enters the critical section) but reaches a plateau or fluctuates at very high values of k.

**Reasons for This Behavior**:

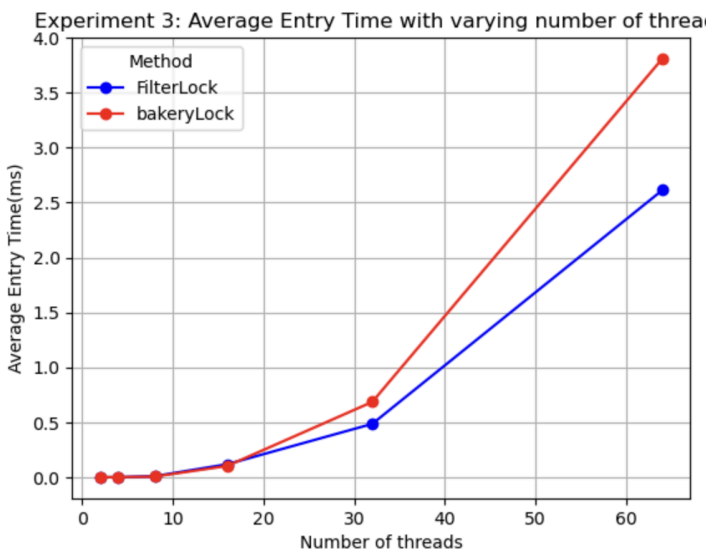
1. **Filter Lock**: When k is small, threads frequently enter and exit the critical section. This constant need to acquire the lock introduces overhead. As k increases, each thread spends more time in the critical section, leading to fewer lock acquisitions per unit of time, reducing lock contention and improving throughput. However, after a certain point, increasing k does not improve throughput further because threads begin to monopolize the critical section for long periods, and throughput becomes limited by how long threads hold the lock.
2. **Bakery Lock**: Similar to the Filter Lock, the Bakery Lock benefits from larger k values initially, as it reduces the need for frequent lock acquisition. However, the Bakery Lock's overhead in determining the next thread to enter the critical section increases with more

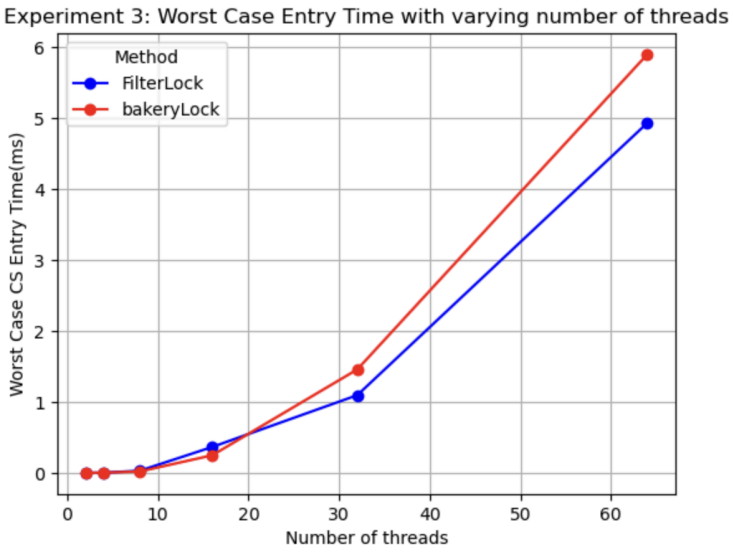
threads. As k increases, threads spend more time in the critical section, but the cost of checking all threads' tickets doesn't scale as efficiently as the Filter Lock. This results in less pronounced throughput gains with the Bakery Lock at high k values compared to the Filter Lock.

**Key Takeaway**: Both locks see a throughput increase with higher k, but the Filter Lock benefits more due to its efficient filtering, while the Bakery Lock's ticket-based mechanism limits its scalability at higher values of k.

## 3) Experiment 3: Average Entry Time and Worst-Case Entry Time Analysis with varying threads

In this experiment, you will focus on the average and worst-case entry time by increasing the threads. The y-axis will measure the average time it takes for a thread to enter the critical section from the time it requested the entry. The x-axis will vary the number of threads, ranging from 2 to 64 in powers of 2.




**Observation:** Both the average and worst-case entry times increase as the number of threads (n) increases, but the increase is more pronounced in the Bakery Lock.

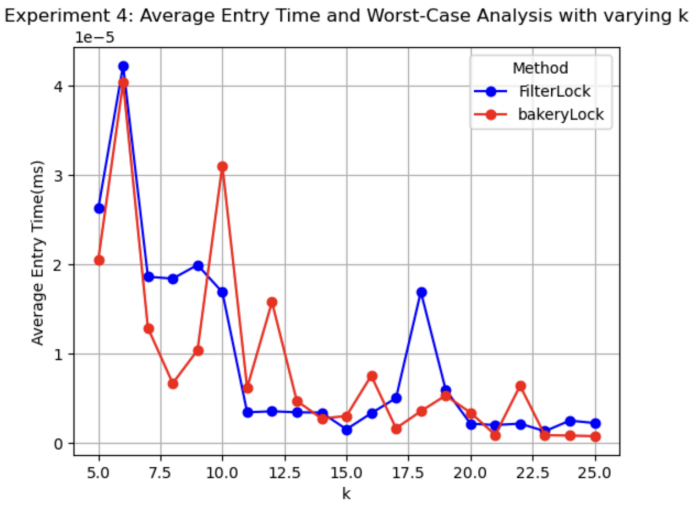
**Reasons for This Behavior**:

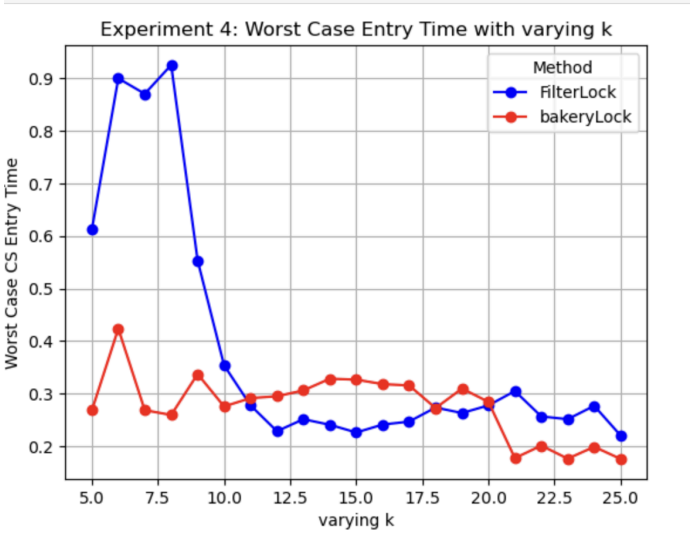
1. **Filter Lock**: The average entry time increases moderately as n increases because more threads need to be filtered through the lock's hierarchical levels. However, the hierarchical structure allows multiple threads to progress through the lower levels simultaneously, reducing overall contention. As a result, the worst-case entry time remains relatively stable even as thread counts grow, making the Filter Lock better suited for higher thread counts in terms of both average and worst-case performance.
2. **Bakery Lock**: The Bakery Lock uses a ticket system where each thread must check all other threads' tickets to determine whether it can enter the critical section. As the number of threads grows, each thread's ticket-checking process becomes more time-consuming, increasing both average and worst-case entry times. The worst-case scenario, where a thread waits for many others before it can enter, becomes significantly longer at high thread counts, resulting in poor performance under heavy contention.

**Key Takeaway:** The Filter Lock has a more scalable design in terms of entry time, as its hierarchical filtering reduces contention. The Bakery Lock, due to its ticketing system, faces a larger performance hit as the number of threads increases.

## 4) Experiment 4: Average Entry Time and Worst-Case Analysis with varying k

Similar to Step 3, in this experiment, you will focus on the average and worst-case entry time by varying k. The y-axis will measure the average time it takes for a thread to enter the critical section from the time it requested the entry. The x-axis will vary k, ranging from 5 to 25.




**Observation:** As k increases, both average and worst-case entry times remain relatively stable, but worst-case times can increase slightly.

**Reasons for This Behavior:**

1. **Filter Lock:** Since k represents the number of times a thread enters the critical section, increasing k means that threads spend more time in the critical section. However, the lock acquisition process itself (the time spent entering the critical section) doesn't change much as k varies. The worst-case entry time can increase slightly because threads spend longer in the critical section, meaning that a waiting thread may have to wait longer for others to finish.
2. **Bakery Lock:** Similar to the Filter Lock, the Bakery Lock's lock acquisition time remains stable as k increases. However, since the Bakery Lock guarantees a strict ordering of threads, threads at the end of the queue may have to wait longer if the critical section is held by a thread for a long period. Thus, while the average entry time remains relatively stable, the worst-case time can increase due to longer waits for threads with higher ticket numbers.

**Key Takeaway**: Increasing k mainly affects the time spent in the critical section, not the lock acquisition time. Both locks handle this similarly, but the Bakery Lock's worst-case performance may degrade slightly due to its strict ordering mechanism.

## Conclusion: Which Lock Algorithm is Better?

**Based on the observations from the experiments:**

1. **Filter Lock** is generally better in terms of scalability and performance, especially when dealing with a large number of threads. Its hierarchical filtering mechanism reduces contention, leading to better throughput and more stable entry times, both in average and worst-case scenarios.
2. **Bakery Lock** performs well at lower thread counts, where its fairness mechanism ensures that all threads have equal access to the critical section. However, as the number of threads increases, the overhead of its ticketing system grows, leading to higher lock acquisition times and reduced throughput.

For systems where scalability is crucial, such as those with a high number of threads, the Filter Lock is the preferred choice due to its better performance under heavy contention.

If fairness is the primary concern, and the number of threads is limited, the Bakery Lock provides a fairer solution, albeit at the cost of scalability.