Assignment 5

Aryaman Srivastava 2210110206

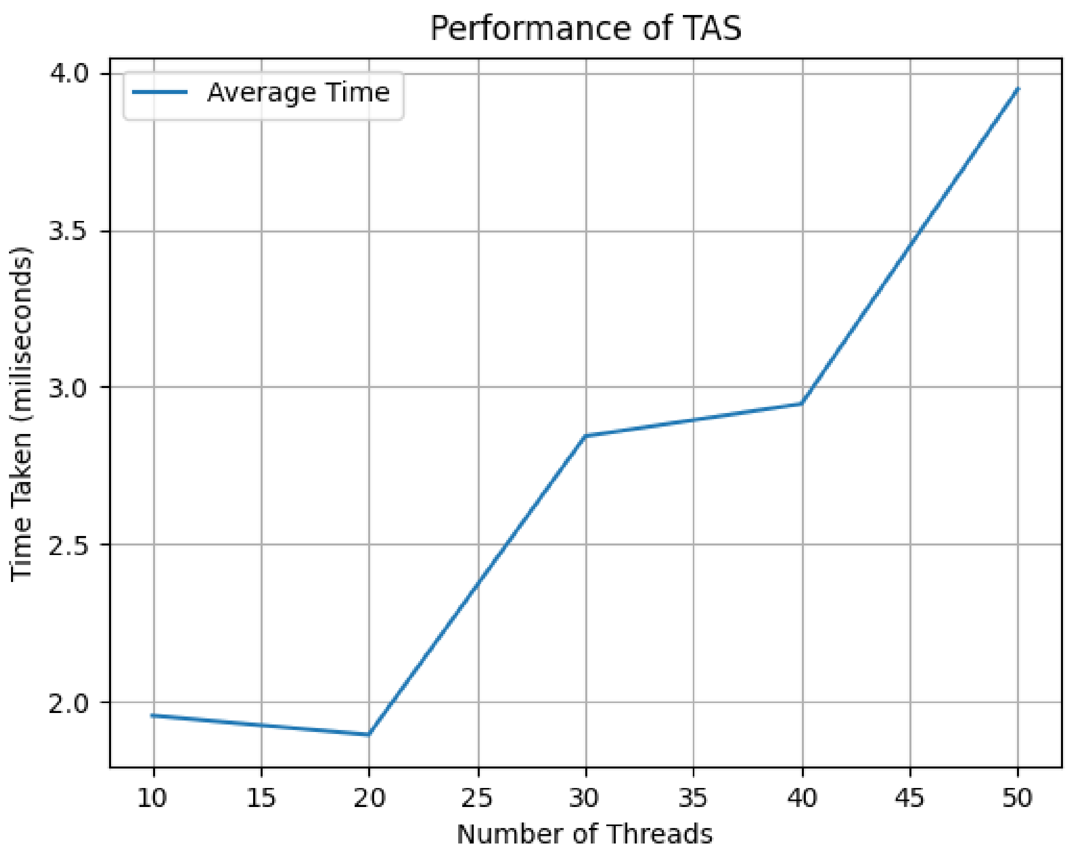
1. TAS

TAS stands for Test-and-Set, which is a synchronization primitive used in concurrent programming and operating systems. It is typically implemented as an atomic hardware instruction that allows a thread to atomically test a memory location and set it to a new value. In other words, TAS checks the current state of a variable and changes it if a certain condition is met, all in one indivisible operation.

When a thread wants to enter a critical section, it performs a TAS operation on a lock variable. If the lock is free (i.e., its value is 0), the thread sets the lock to 1 and enters the critical section. If the lock is already taken (i.e., its value is 1), the thread knows that another thread is inside the critical section and must wait or retry.

TAS is a fundamental building block for synchronization mechanisms like spinlocks and is essential for ensuring thread safety and preventing race conditions in concurrent programs.

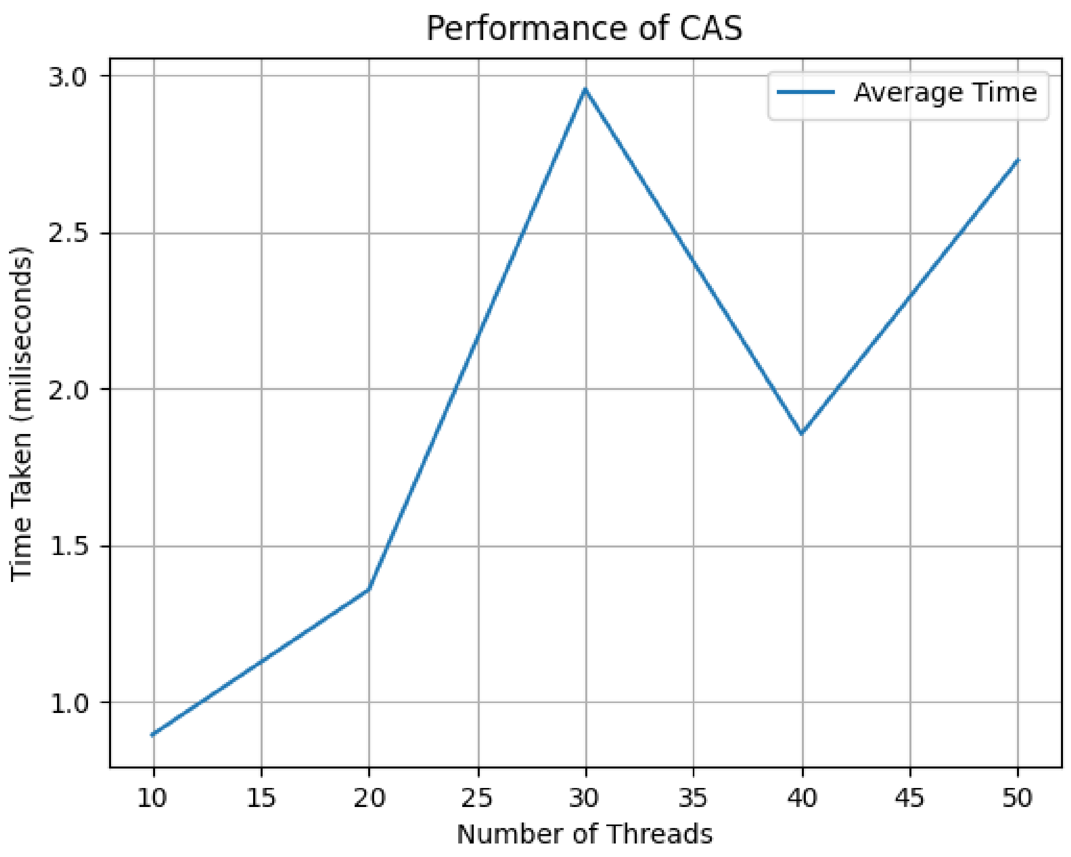
The program simulates multithreaded access to critical sections. It reads parameters from "inp-params.txt" (number of threads, critical sections, delays), creates threads to enter critical sections, logs actions (Requested, Entered, Exited) with timestamps, sorts them, and writes the sorted actions to "output-tas.txt" along with the total execution time. The program uses pthreads, file I/O, time functions, and random delays to mimic concurrent access and synchronization in a system.



2. CAS

CAS, or Content Addressable Storage, is a data storage technology that retrieves information based on its content rather than its location. Unlike traditional storage systems that use specific addresses to access data, CAS identifies data by its unique characteristics or content, such as a cryptographic hash. This enables efficient and fast data retrieval, making it ideal for applications like data deduplication, archival storage, and compliance management. CAS systems ensure data integrity, prevent duplicates, and simplify data management by allowing users to access information using its content-based identifier, enhancing data security and optimizing storage utilization in modern digital environments.

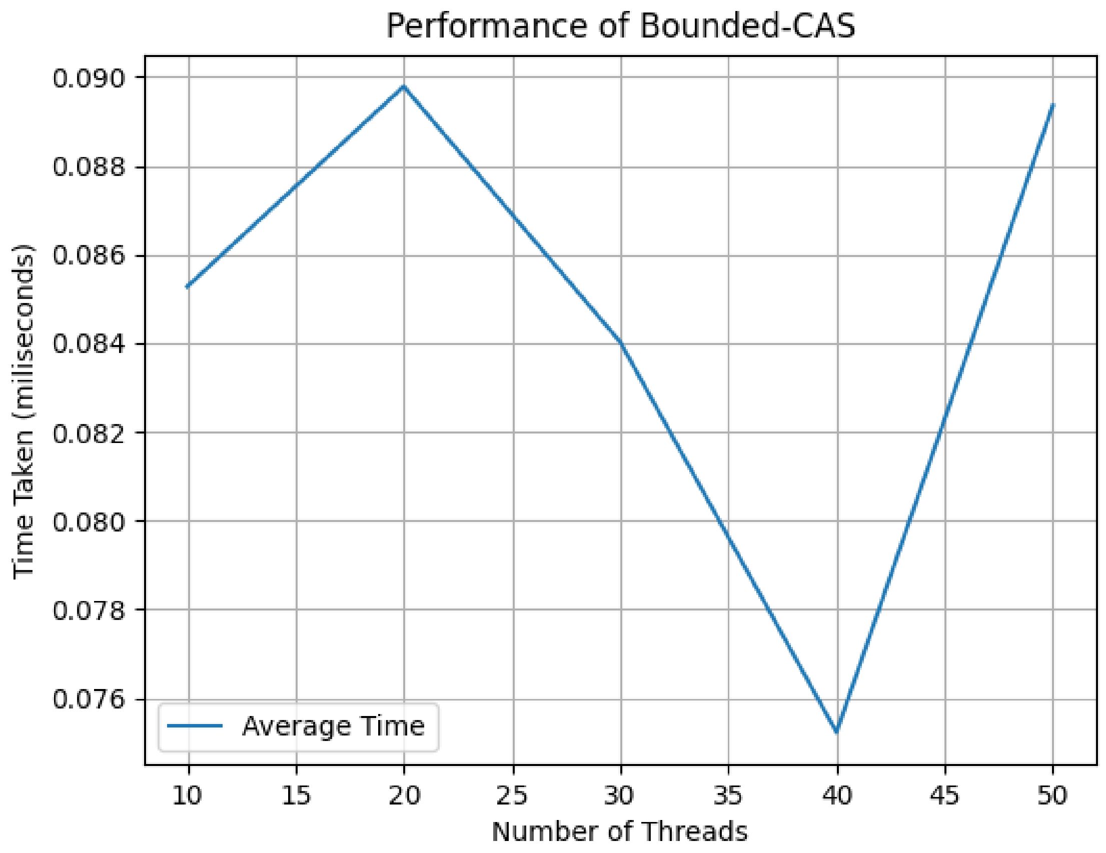
The program implements a multithreaded program using pthreads to simulate a critical section (CS) problem. It reads input parameters from a file, creates threads based on the input, and each thread repeatedly enters and exits the critical section while maintaining a specific order. The critical section execution is controlled using an atomic shared variable. Threads request access to the CS, enter it with a random delay, and then exit. The program records timestamps for each CS entry and exit, calculates the total execution time, and writes the results to an output file.



3. Bounded CAS

Bounded CAS (Compare-and-Swap) is a concurrency control mechanism used in programming and computer science. It ensures atomicity when multiple threads attempt to modify a shared variable concurrently. In a bounded CAS operation, the value of the variable is compared to an expected value. If the current value matches the expected value, the variable is updated to a new value. Otherwise, the operation fails. Bounded CAS is bounded by a maximum number of retries, preventing indefinite looping and ensuring progress in concurrent operations while avoiding the overhead of locking mechanisms like mutexes, making it suitable for high-performance and scalable systems.

This C code simulates concurrency with bounded CAS (Compare-and-Swap) to manage critical sections. It initializes threads and critical sections based on input parameters. Each thread requests access to a critical section, uses CAS to acquire a lock, enters the critical section, performs a task, then releases the lock. The program prints timestamps for request, entry, and exit of each critical section per thread, ensuring mutual exclusion. Random sleep times mimic real-world delays. The output is redirected to "output-bounded-cas.txt". Finally, the code calculates and displays the total execution time in seconds, providing insights into concurrent execution with synchronization.



COMPARISION

The comparison among TAS (Test-and-Set), CAS (Compare-and-Swap), and Bounded CAS (Bounded Compare-and-Swap) mutual exclusion algorithms revolves around fairness, scalability, and overhead considerations.

Fairness:

TAS suffers from potential fairness issues due to thread starvation.

CAS improves fairness by allowing other threads to have a chance to enter the critical section.

Bounded CAS enhances fairness further by limiting a thread's attempts to acquire the lock, preventing monopolization.

Scalability:

TAS faces limitations in high contention scenarios, leading to resource contention.

CAS improves scalability by reducing contention and spin-waiting.

Bounded CAS enhances scalability by limiting lock acquisition attempts, reducing contention, and improving system performance.

Overhead:

TAS generally has lower overhead due to its simple operation.

CAS incurs slightly higher overhead due to compare-and-swap logic.

Bounded CAS has slightly higher overhead than CAS due to managing the bounded counter.

Ultimately, the choice of algorithm depends on specific application needs and system characteristics. TAS suits low-contention scenarios, CAS is preferable for medium to high contention, and Bounded CAS excels in highly concurrent systems with varying contention levels.