Operating Systems-2 Report

Eslavath Naveen Naik (Roll No: CS22BTECH11021)

March 3, 2024

1 Introduction

This report documents the implementation and comparison of four different mutual exclusion mechanisms in the context of concurrent programming. The mutual exclusion mechanisms considered are Test-and-Set (TAS), Compare-and-Swap (CAS), Bounded Compare-and-Swap (Bounded CAS), and Atomic Increment.

2 Implementation

The assignment implemented four different mutual exclusion mechanisms:

- **Test-and-Set (TAS)**:
  TAS provides mutual exclusion by atomically setting a lock variable. It ensures that only one thread can access a critical section at a time.
- **Compare-and-Swap (CAS)**:
  CAS achieves synchronization among threads by atomically comparing and swapping values. It is commonly used in concurrent programming.
- **Bounded Compare-and-Swap (Bounded CAS)**:
  Bounded CAS is a variant of CAS that limits the number of retries in case of contention.
- **Atomic Increment**:
  Atomic Increment is used as a synchronization mechanism to coordinate access to shared resources among multiple threads.

3 Time vs. Size, N

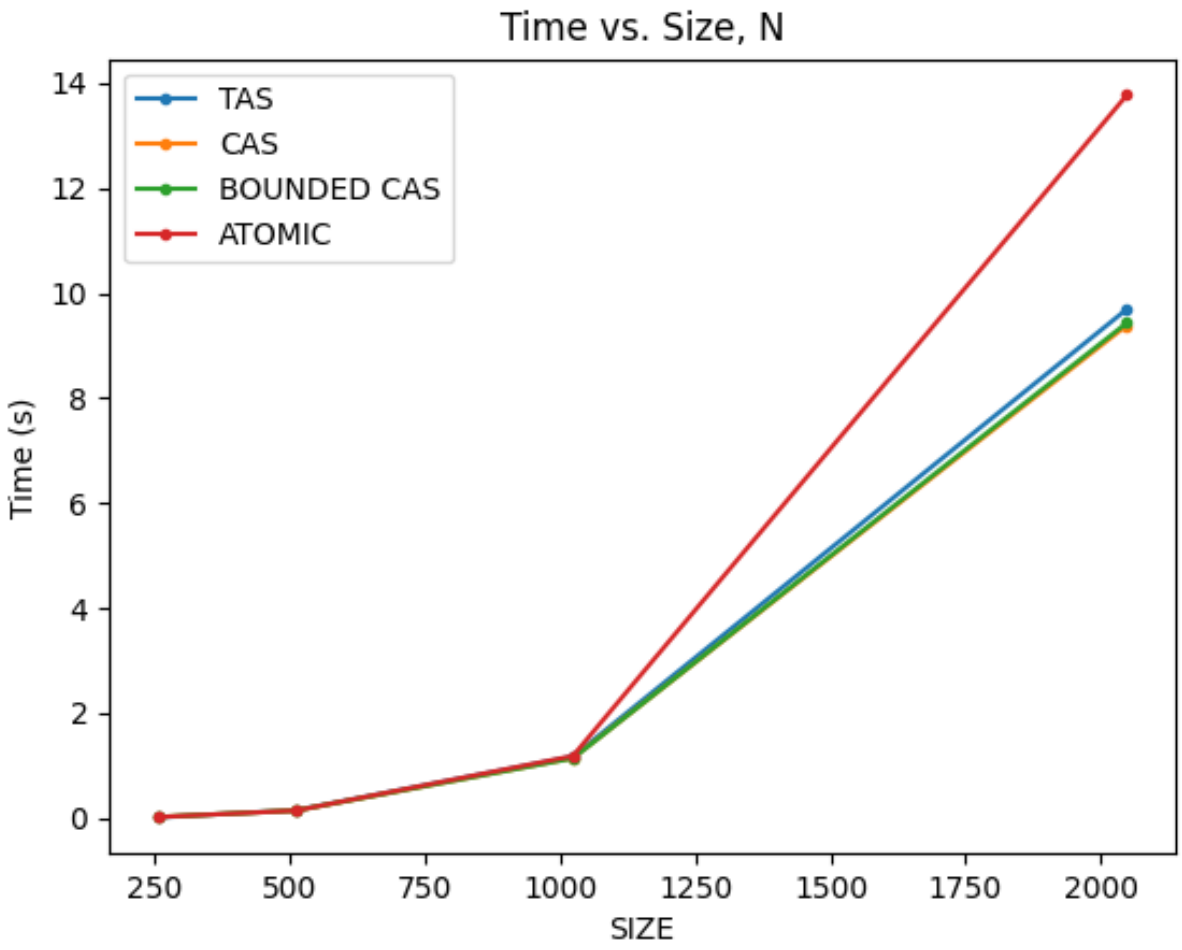

Figure 1: Time vs. Size, N

Size(N)	Time taken by TAS(in sec)	Time taken by CAS(in sec)	Time taken by Bounded CAS(in sec)	Time taken by atomic(in sec)
256	0.0107453	0.0165812	0.017075	0.0157952
512	0.146296	0.140162	0.141343	0.139918
1024	1.17106	1.13944	1.1357	1.18025
2048	9.69494	9.38708	9.43424	13.7757

Table 1: Time taken

4 Time vs. rowInc, Row Increment

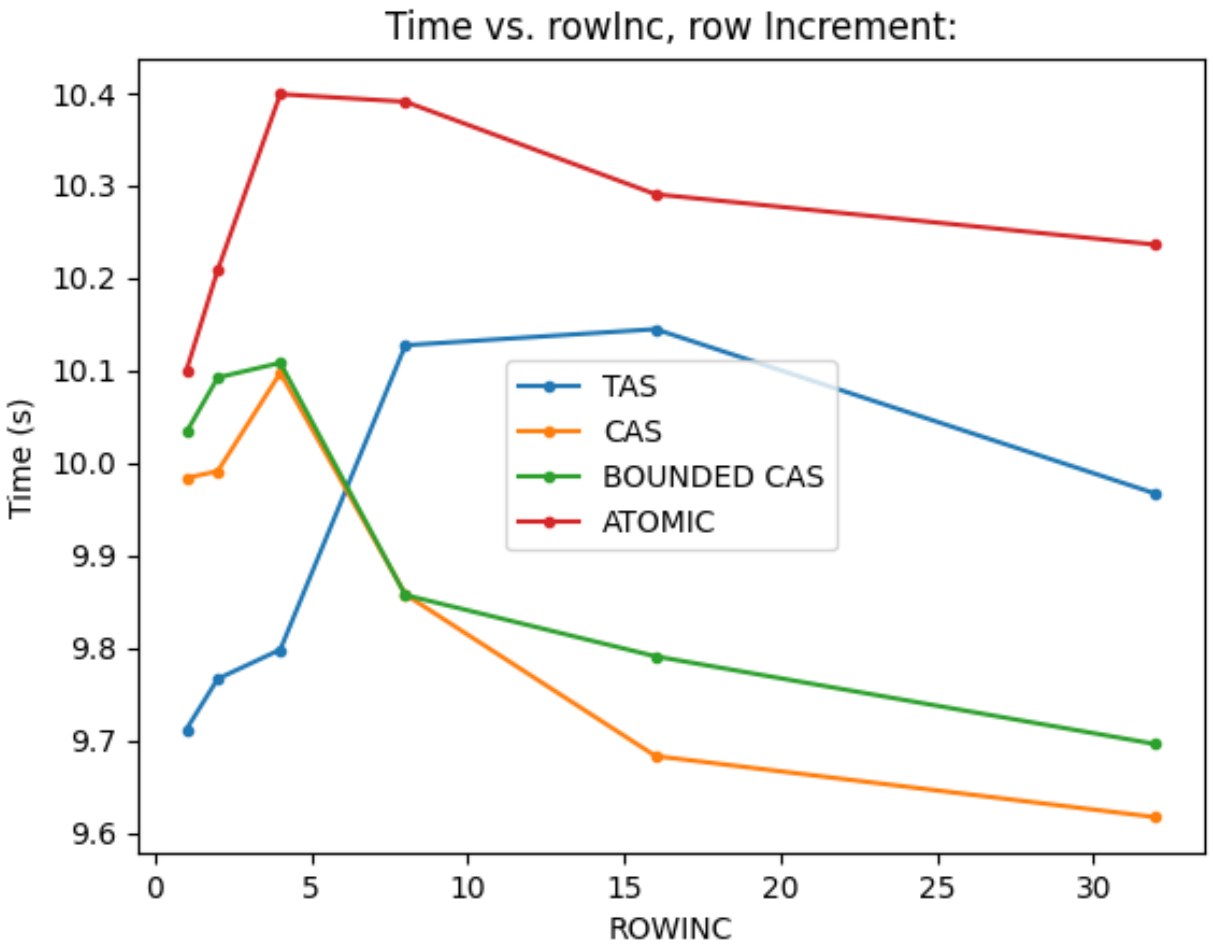

Figure 2: Time vs. rowInc, Row Increment

Size(N)	Time taken by TAS(in sec)	Time taken by CAS(in sec)	Time taken by Bounded CAS(in sec)	Time taken by atomic(in sec)
1	9.71202	9.98387	10.0334	10.1002
2	9.76649	9.991116	10.09244	10.2087
4	9.79809	10.09758	10.10833	10.39907
8	10.1274	9.85784	9.85714	10.39083
16	10.1448	9.68297	9.7908262	10.290736
32	9.9667	9.616799	9.6958362	10.23620

Table 2: Time taken

5 Time vs. Number of threads, K

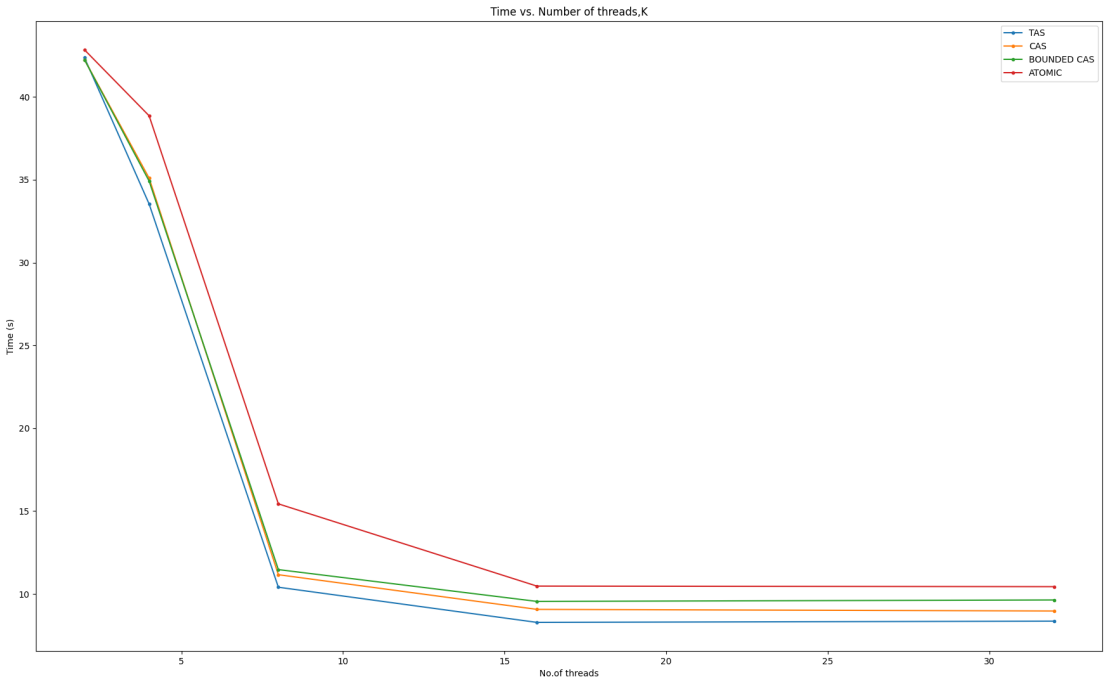

Figure 3: Time vs. Number of threads, K

Size(N)	Time taken by TAS(in sec)	Time taken by CAS(in sec)	Time taken by Bounded CAS(in sec)	Time taken by atomic(in sec)
2	42.3688	42.2473	42.2427	42.844
4	33.5392	35.1018	34.9237	38.8618
8	10.4037	11.1596	11.4673	15.4364
16	8.28132	9.0676	9.54767	10.4719
32	8.35708	8.96837	9.63472	10.4375

Table 3: Time taken

6 Time vs. Algorithms

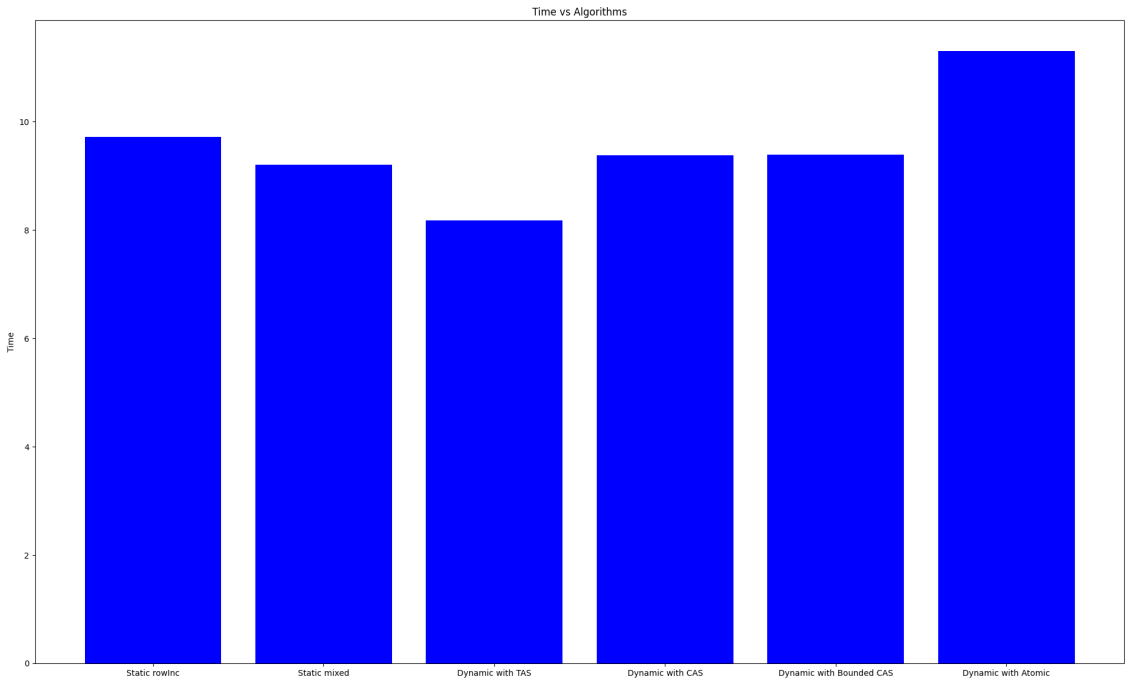

Figure 4: Time vs. Algorithms

Algorithms	Time (sec)
Static rowInc	9.7201
Static mixed	9.20807
Dynamic with TAS	8.18319
Dynamic with CAS	9.3826
Dynamic with Bounded CAS	9.39112
Dynamic with Atomic	11.3057

Table 4: Time taken by different algorithms

7 Code Overview

The code reads input from a file containing a square matrix and other parameters such as the number of threads and the number of rows each thread should process. It implements matrix multiplication using multiple threads and different mutual exclusion mechanisms. The matrix multiplication operation is divided into chunks, and each thread processes a chunk of rows. The time taken by each mechanism to perform the matrix multiplication operation is measured using the `std::chrono` library. Finally, the result matrices and the time taken for each mechanism are written to an output file for analysis.

8 Observations

- Different mutual exclusion mechanisms exhibit varying performance characteristics depending on factors such as contention, overhead, and system configuration.
- Understanding the behavior and performance of these mechanisms is crucial for designing efficient and scalable concurrent systems.
- Further experimentation and analysis may be required to optimize the code and explore the behavior of these mechanisms under various workload conditions.

9 Conclusion

The report provides insights into the implementation and comparison of mutual exclusion mechanisms in concurrent programming. Understanding the trade-offs and performance characteristics of these mechanisms is essential for designing robust and efficient concurrent systems. Overall, the exploration of mutual exclusion mechanisms contributes to a deeper understanding of operating systems and concurrent programming paradigms.