

Project Title:

**[The Search-Insert-Delete Problem]**

Submitted by:

**[Urwah Imran] [2021-CE-15]**

**[Samreen Razzaq] [2021-CE-13]**

**[Ayesha] [2021-CE-18]**

**[Aiman Malik] [2021-CE-23]**

Submitted to:

**[Dr. Ali Hammad and Ma’am Darakhshan Abdul Ghaffar]**

Course:

**[CMPE-331L: Operating Systems]**

Semester:

**[5th]**

Date of Submission:

**[December 27, 2023]**

**Department of Computer Engineering**

**University of Engineering and Technology, Lahore**

Table of Contents

[Abstract 3](#_Toc154246208)

[1. Introduction 3](#_Toc154246209)

[2. Problem statement 3](#_Toc154246210)

[3. Solution to the problem 3](#_Toc154246211)

[4. Results and Conclusion 3](#_Toc154246212)

[**5.** Future work and Applications in Operating systems 3](#_Toc154246213)

[References 3](#_Toc154246214)

# 

# Abstract

The “Search-Delete-Insert-Problem”, a concurrency challenge in operating systems. In this problem, three types of threads, namely searchers, inserters, and deleters, share access to a singly linked list. Searchers examine the list concurrently, while inserters add new items to the list, and deleters remove items from anywhere in the list. However, these operations must be categorized into three-way mutual exclusion, where searchers can run concurrently, insertions require mutual exclusion, and deletion should be mutually exclusive with searches and insertions

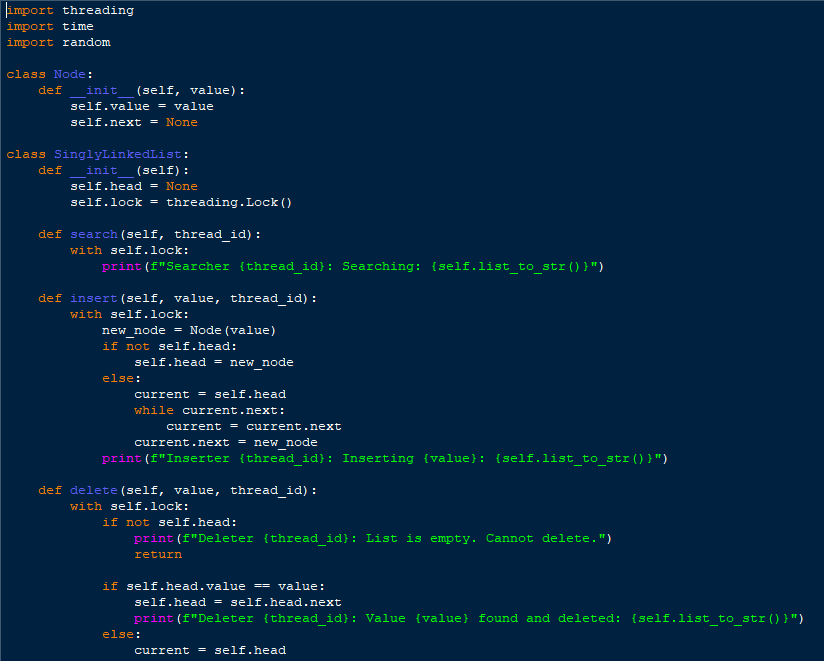
# Introduction

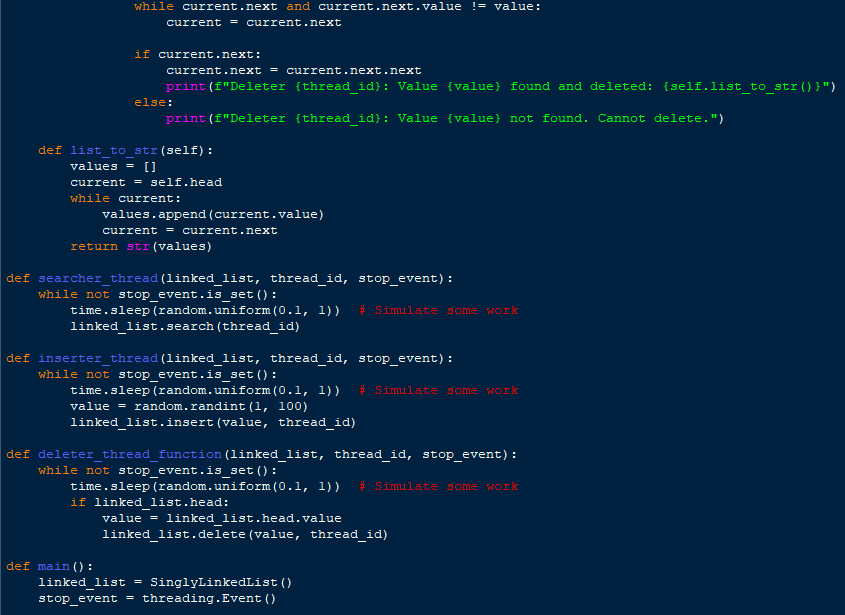
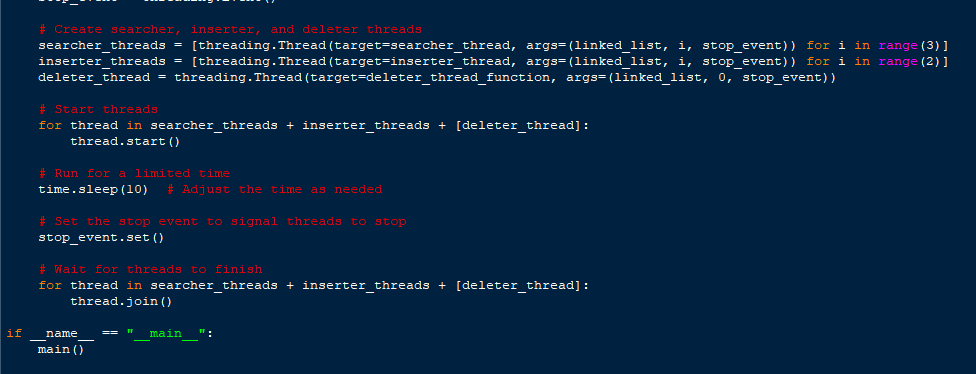
The search-insert-delete problem, drawn from Andrews's Concurrent Programming [1], encapsulates a complex challenge in concurrent systems where three types of threads—searchers, inserters, and deleters—share access to a singly-linked list. Searchers operate concurrently, merely examining the list, while inserters add new items exclusively to prevent simultaneous insertions. Interestingly, an inserter can proceed in parallel with any number of searches. Deleters, responsible for removing items from the list, must have exclusive access, limiting them to one process at a time and necessitating mutual exclusion with searches and insertions. This intricate scenario demands a nuanced approach to thread coordination and synchronization, exploring solutions that strike a delicate balance between enabling concurrency and ensuring the required mutual exclusion.

# Problem statement

The problem entails designing a concurrent system in which searchers, inserters, and deleters can access a singly-linked list. Searchers operate concurrently without interfering with each other, inserters must ensure mutual exclusion to prevent conflicts, and deleters need exclusive access, excluding searches and insertions. The task is to create a system that enforces this three-way categorical mutual exclusion effectively.

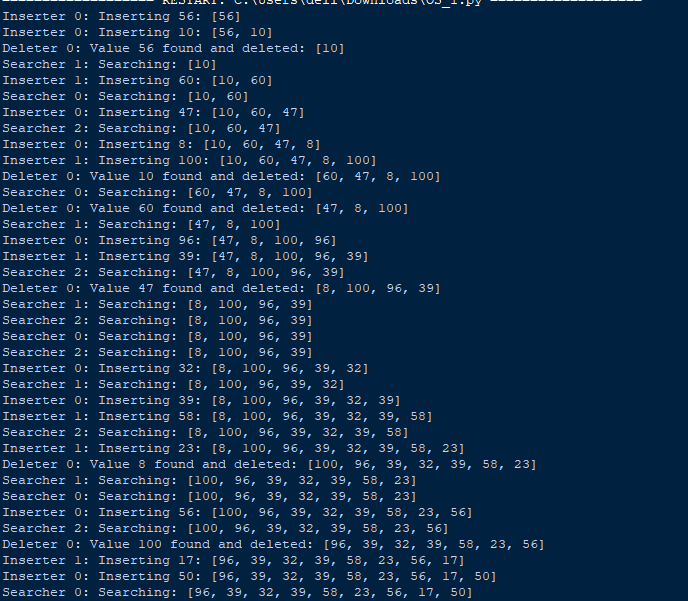
# Solution to the problem

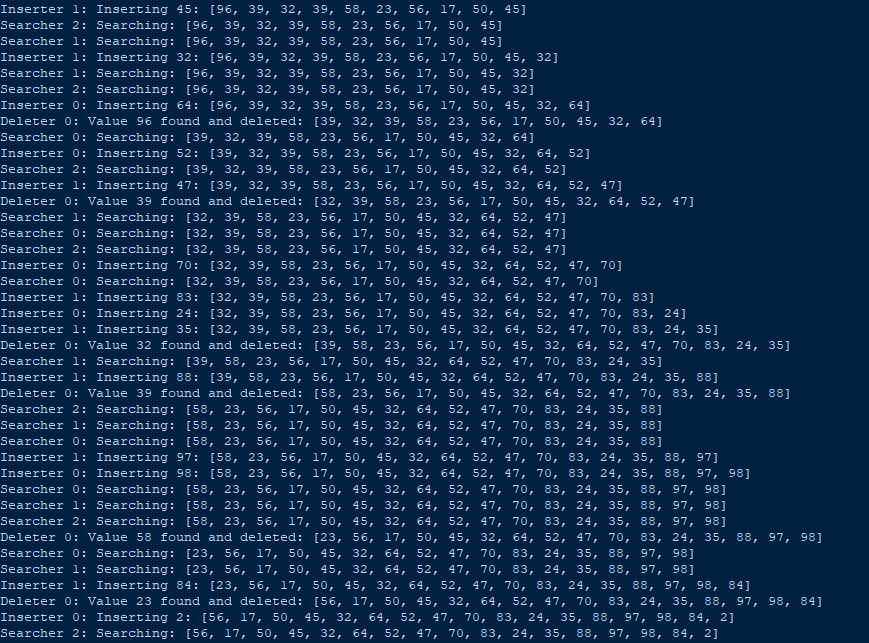
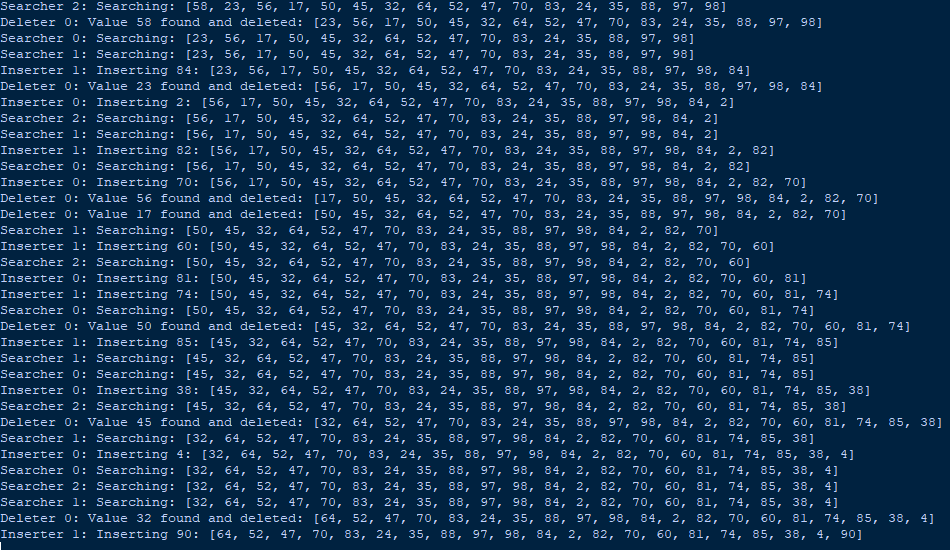
The solution to the search-insert-delete problem involves carefully orchestrating the concurrent actions of searchers, inserters, and deleters on a singly-linked list. The provided code leverages semaphores and light switches to enforce mutual exclusion and ensure the correct execution of each thread type. For searchers, a light switch mechanism is used, allowing multiple searchers to operate concurrently while excluding deleters. Inserters utilize a combination of a light switch and a semaphore to ensure mutual exclusion during insertion, permitting only one inserter to modify the list at a time. Deleters, on the other hand, use semaphores to enforce exclusive access, acquiring both the "noSearcher" and "noInserter" semaphores before entering their critical section. The careful coordination of these synchronization mechanisms guarantees that searchers, inserters, and deleters operate harmoniously, adhering to the specified constraints and ensuring the integrity of the singly-linked list. However, the solution acknowledges the potential for starvation, a common challenge in categorical exclusion problems, and emphasizes the difficulty in achieving both efficiency and maximum concurrency while avoiding this issue.



# 

# Results and Conclusion





**Conclusion**

In conclusion, the search-insert-delete problem, as presented in Andrews's Concurrent Programming, highlights the intricate challenges associated with managing concurrent access to a singly-linked list by searchers, inserters, and deleters. The provided solution, employing semaphores and light switches, offers a structured approach to achieving mutual exclusion and concurrency, striking a delicate balance to ensure the integrity of the shared data structure. The solution, while effective, acknowledges the potential for issues such as starvation and emphasizes the ongoing difficulty in finding solutions that optimize both efficiency and maximum concurrency. Looking forward, future work in the field of operating systems could explore more advanced synchronization techniques and their application in dynamic, distributed environments, further enhancing the practicality and scalability of solutions to such complex concurrency problems. The search-insert-delete problem serves as a valuable case study in understanding the nuances of thread coordination and resource management, contributing to the broader discourse on concurrent programming challenges and solutions in operating systems.

1. **Future work and Applications in Operating systems**

Future work in the realm of operating systems pertaining to the search-insert-delete problem could explore more advanced synchronization techniques and algorithms to enhance performance and address potential limitations. Researchers might investigate the applicability of lock-free or wait-free data structures to further reduce contention and enhance concurrency in scenarios with searchers, inserters, and deleters. Additionally, exploring the integration of these synchronization mechanisms into modern operating systems, considering factors such as scalability and adaptability to diverse hardware architectures, would be valuable. Furthermore, the application of this problem to real-world scenarios could be extended to systems with dynamic lists, distributed environments, or varying levels of thread priority, introducing additional complexities that would require innovative solutions. Practical implementations and benchmarks on different operating systems could be conducted to assess the effectiveness of the proposed solutions in diverse computing environments. Overall, future work in this domain should aim to advance the understanding and practical applicability of concurrent programming solutions for efficient and scalable management of shared resources in operating systems.

# References

[[1] https://youtu.be/1KyMN3IB4BE?si=Jeks4udTkamWKRbW](https://youtu.be/1KyMN3IB4BE?si=Jeks4udTkamWKRbW)

[2] <https://www.geeksforgeeks.org/search-insert-and-delete-in-a-sorted-array/>