CS 554: Advanced Topics in Concurrent Computing Systems
Final Project Proposal

Correctness Analysis of Classic Mutual Exclusion Algorithms Under different Memory Models

Team Members
Amit Maheshwar Varanasi
Dhanush Bommavaram

1. Project Overview

We propose to analyze the correctness of two foundational mutual exclusion algorithms, **Peterson's Algorithm** and **Dekker's Algorithm** under various memory models. Our project will empirically validate both the algorithms, and formally analyze if and why these algorithms fail under different implementations and memory models.
This project bridges theoretical understanding of memory models with practical systems programming, demonstrating how even well-established algorithms can exhibit unexpected behavior.

2. Memory Models and Tools

Memory Models to Analyze:

- memory_order_seq_cst (SC+NA)
- memory_order_acq_rel (RA+NA)
- memory_order_relaxed (RA+NA+RLX)
- no atomic operations (NA)

Tools and Languages:

- **C/C++ with pthreads** for implementation
- **C11/C++11 atomics**
- **x86/Arm processor** for empirical testing
- **GDB/objdump** for examining generated assembly
- **CppMem** for detecting race conditions

3. Results to produce

For each algorithm:

1. Try to prove/disprove the data race conditions when using Peterson's algorithm and Dekker's algorithm with and without atomic operations under various memory models,
    a. Algorithm implemented with atomic operations
        i. memory_order_seq_cst (SC+NA)
        ii. memory_order_acq_rel (RA+NA)
        iii. memory_order_relaxed (RA+NA+RLX)
        iv. no atomic operations (NA)
    b. Algorithm implemented without atomic operations

    Initially prove using event structures and also verify example implementation of algorithms with CppMem.
2. Examine generated assembly code.

4. Why This Project Interests Us

As students in an advanced concurrency course, we're fascinated by how subtle the gap between theory and practice can be. These algorithms are taught as correct solutions in undergraduate courses, yet they fail on real hardware highlighting the critical importance of understanding memory models in systems programming.
This project appeals to us because:

- It combines **formal reasoning** (correctness proofs) with **empirical validation**.
- It's **practical and relevant** understanding these issues is crucial for writing correct concurrent code
- It has **clear success criteria** we either detect violations or we don't
- It deepens our understanding of the **hardware-software interface**

5. Conclusion

This project provides an ideal balance of theoretical rigor and practical implementation. By analyzing two classic algorithms under multiple memory models, we will gain deep insights into concurrent programming challenges while producing tangible, reproducible results. The 4 week timeline with two team members allows us to thoroughly explore both algorithms while maintaining focus and quality.
We believe this project will not only demonstrate our learnings from the course but also contribute to our understanding of real-world concurrent systems knowledge.