Nonblocking Synchronization

Fuad Tabba

Department of Computer Science
University of Auckland

April 1st, 2008

Outline

Locks: The Good

An easy to understand solution to the critical section problem...

- Easy to reason about.
- Well understood, been around since ages ago!
- Implemented in most programming languages.

Locks: The Bad

Locks have lots of problems, some easier to solve than others...

- Deadlock
- Priority Inversion
- Convoying

Locks: The Ugly

Locks pose correctness and performance issues...

- Cannot tolerate faults
- Not easy to use correctly
- Software engineering: break encapsulation and limit composability
- Tradeoff: ease of use vs performance

Nonblocking Synchronization

Do not use locks... correctly!

- `Nonblocking:` execution is not indefinitely postponed...
- `Wait-free:` everyone progresses (strongest)
- `Lock-free:` the system progresses
- `Obstruction-free:` single uncontended thread progresses (weakest)

It's not about avoiding faults, it's about making progress.

Nonblocking Synchronization

Definitions are vague

- Lock-free and Nonblocking are sometimes used synonymously
- Deadlocks are never allowed
- Livelocks and starvation can occur (Lock-free and Obstruction-free)
- Need some level of hardware support (i.e. CAS)
- More difficult to write than lock-based algorithms

Some Examples

Nonblocking algorithms in theory and practice...

- Nonblocking counter
- Nonblocking stack
- Nonblocking linked-list
- RCU (Read-Copy Update) (?)

References and Further Reading

Refer to the following for more information

- Wikipedia: Nonblocking Synchronization
- Paul E. McKenney: What is RCU, Really?
- Angela Demke Brown: Avoiding Locks