Paul E. McKenney – IBM Distinguished Engineer, Linux Technology Center

03 Jan 2011

Multi-Core Memory Models and Concurrency Theory

A View from the Linux Community

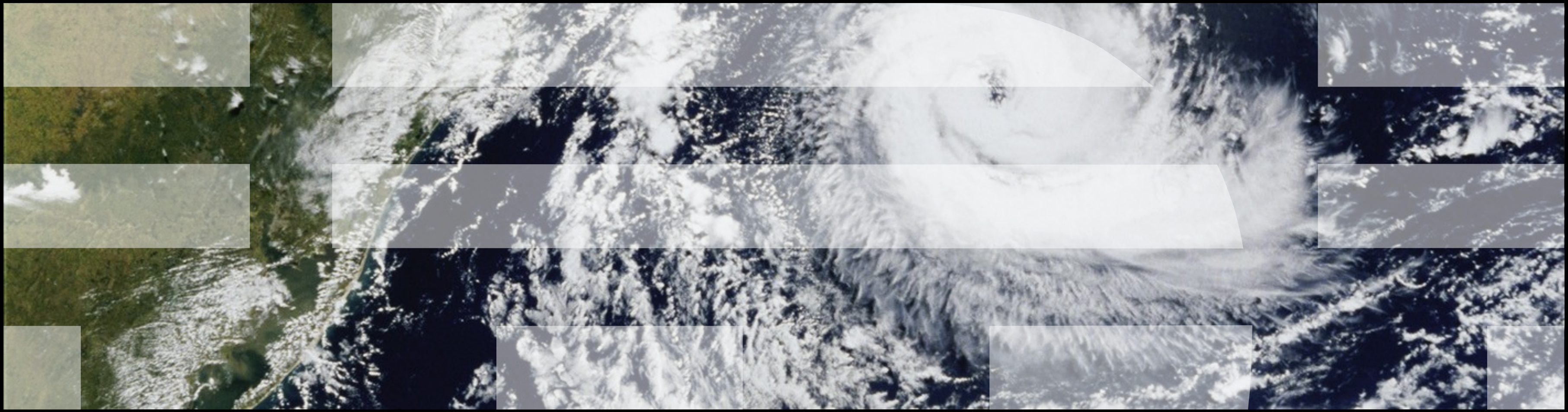


Table of contents

How did I get this way?

- Sole proprietor doing real-time systems on 8-bit and 16-bit microprocessors
- SRI International: systems administration and Internet research
- Sequent Computer Systems: DYNIX 3 and DYNIX/ptx on Balance and Symmetry
  - 1990: 30-way SMP, primarily via locking: parallel memory allocation
    - Workload moving from scientific/technical to RDBMS transactional
  - 1993: Clustering requires scalable distributed lock manager: RCU (AKA “rclock”)
    - Workload almost entirely RDBMS transactional
  - 1996-7: NUMA-Q enables scalable 64-way system: counting requires distribution
    - Workload still almost entirely RBMS transactional
  - Note: Symmetry and NUMA-Q are single CPU architecture: x86
- IBM: AIX
  - 2000: More of the same, except that Power brings weaker memory ordering
- IBM: Linux
  - 2001: Some changes...

Starting the Linux journey

- 2001: 2-4 CPU systems (SGI does 100s with special patchset), lots of CPU architectures
  - Locking with much coarse-grained locking
    - Infrastructure, web serving, small DBMS, …
- 2001-present: a simple matter of applying scalability lessons learned in the 1990s?

The Linux journey

- 2001: 2-4 CPU systems (SGI does 100s with special patchset), lots of CPU architectures
  - Locking with much coarse-grained locking
    - Infrastructure, web serving, small DBMS, …
- 2001-present: a simple matter of applying scalability lessons learned in the 1990s?
  - Not quite!!!
  - Several complicating factors: workloads, real-time response, energy efficiency
- Workloads:
  - Old way: When a certain RDBMS runs, we are done!!!
  - New way: Everything from smartphones to supercomputers. We are *never* done.
- Real-time response:
  - Old way: If 90% of transactions complete in under two seconds, we are good!!!
  - New way: Make that 100% in a few tens of microseconds. But no pressure!!!
- Energy efficiency:
  - Old way: Energy costs are a negligible fraction of total costs
  - New way: Hand-held battery-powered SMP systems. Yes, they are already here.

Paths Forward

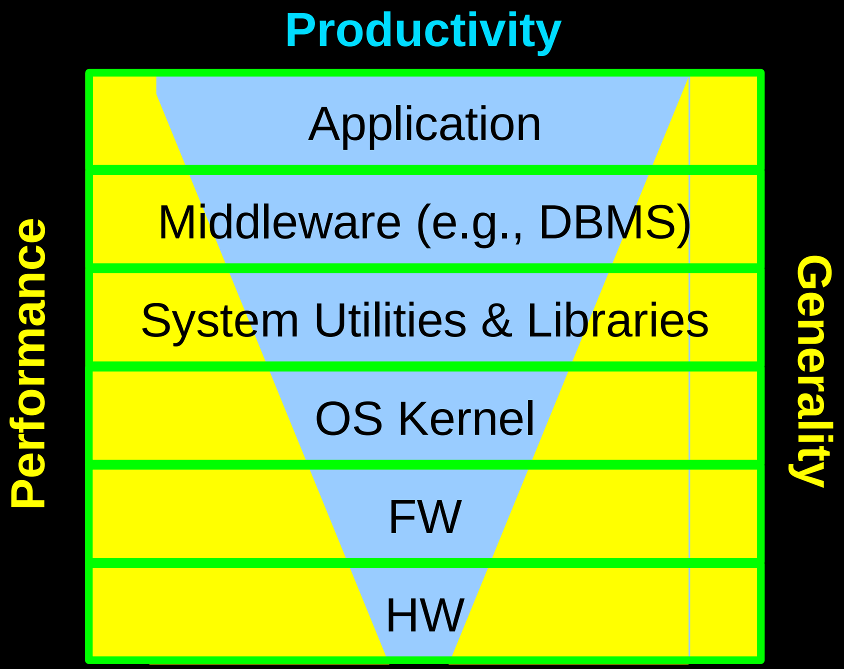
Some assertions based on two decades of parallel experience

- Hardware properties must be taken into account
- Specialization will be required
  - What general-purpose synchronization mechanism provides excellent performance, scalability, real-time response, and energy efficiency?
  - Only one concurrency mechanism is like only one mechanical fastener
- Validation of concurrent software can benefit from hardware-validation tools and experience
  - Example: validation of first real-time RCU implementation
- Different levels of abstraction require different paradigms

Different levels of abstraction require different paradigms

- RCU implementations validated “on bare metal” – in conjunction with severe testing
- *Uses* of RCU handled much differently! Key point: RCU efficiently removes data races
- Given an RCU read-side critical section:
  - `rcu_read_lock();` R_1; R_2; R_3; …; `rcu_read_unlock();`
- And given an RCU update:
  - M_1; M_2; M_3; …; `synchronize_rcu();` D_1; D_2; D_3; …;
- Then the following must hold:
  - $\forall m,i(M_m \to R_i) \vee \forall i,n(R_i \to D_n)$
- The following pair of derived formulas are usually more helpful for validating RCU uses:
  - $\exists i,m(R_i \to M_m) \Rightarrow \forall j,n(R_j \to D_n)$
  - $\exists n,i(D_n \to R_i) \Rightarrow \forall m,j(M_m \to R_j)$
- Joint work with Mathieu Desnoyers, Alan Stern, Michel Dagenais, and Jonathan Walpole
  - http://www.rdrop.com/users/paulmck/RCU/urcu-supp.2010.12.14a.pdf
  - To appear in IEEE Transactions on Parallel and Distributed Systems

Different levels of abstraction require different paradigms



Use of different paradigms

- Low-level critical code: Do what is required for performance
  - Combinatorial explosion limits the size of algorithm that can be proven
    - But spinlocks, sequence locks, RCU, etc. can be usually verified reasonably
  - Use higher-level semantics to prove uses of such algorithms
  - Analogy with mechanical engineering: use beam equations, not Schroedinger equations!!!
    - In real life: use rated loads for specific beams
- Interactions with the outside world:
  - You don't know what the ordering is in any case: why fabricate one?
  - “Stop worrying and learn to love non-determinism, non-linearizability, non-strong non-commutativity, conflicting operations, ...”
  - Added benefits: real-time response, keep up with networking HW, ...

Validating Low-Level Critical Code

- Combination of stress testing and mechanical proofs
- Mechanical proofs
  - Few Linux kernel programmers do proofs, but the number is growing
    - Paul E. McKenney, Mathieu Desnoyers, and a few others
  - Use Promela/spin (historical choice, might choose differently today)
  - Explicitly represent non-determinism
    - Model full permutation of possible orderings (below), or...
    - Explicitly model abstract representation of cache and/or store buffer
    - Some too-large models converted to VHDL and subjected to hardware validation

```
do :: 1 -> sum = ctr[0]; i = 1; break
   :: 1 -> sum = ctr[1]; i = 0; break
od;
sum = sum + ctr[i];
```

How Does the Linux Kernel Community Cope?

Kernel-Community Approaches to Concurrency (Subset 1/2)

- Organizational mechanisms
  - Maintainers and quality assurance: recognition and responsibility
  - Informal apprenticeship/mentoring model
  - Design/code review required for acceptance
  - Aggressive pursuit of modularity and simplicity
- Use sane idioms and abstractions
  - Locking, sequence locking, sleep/wakeup, memory fences, RCU, ...
  - Conventional use of memory-ordering primitives, for example:
    - Susmit's message passing (MP): sync + dependency
    - Susmit's write-to-read causailty (WRC): sync + dependency
  - This avoids Susmit's PPOCA, RSW, RDW, …
    - Hard to even express in core kernel code
  - Needing to know too much about the underlying memory model indicates broken abstraction, broken design, or both

Kernel-Community Approaches to Concurrency (Subset 2/2)

- Static source-code analysis
  - “checkpatch.pl” to enforce coding standards
  - “sparse” static analyzer to check lock acquire/release mismatches
  - “coccinelle” to automate inspection and generation of bug fixes
- Dynamic analysis
  - “lockdep” deadlock detector (also checks for misuse of RCU)
  - Tracing and performance analysis
  - Assertions
- Aggressive automation
  - “git” source-code control system: from weeks to minutes for rebases and merges
- Testing
  - In-kernel test facilities such as rcutorture
  - Out-of-kernel test suites

Kernel-Community Approaches to Concurrency

- To err is human, and therefore...
  - People/organizational mechanisms are at least as important as concurrency technology
  - Use multiple error-detection mechanisms
  - For core of RCU, validation starts at the very beginning:
    - Write a design document: safety factors and conservative design
    - Consult with experts, update design as needed
    - Write code in pen on paper: Recopy until last two copies identical
    - Do proofs of correctness for anything non-obvious
    - Do full-up functional and stress testing
    - Document the resulting code (e.g., publish on LWN)
  - If I do all this, then there are probably only a few bugs left
    - And I detect those at least half the time

An Important Question

Given a Randomly Selected Human Being...

- *Any* human being: head of state, rock star, street person, farmer, researcher, student, CEO, diplomat, janitor, plumber, housewife, toddler, juvenile delinquent, bureaucrat, mafia don, warlord, mercenary soldier, terrorist, policeman, lawyer, doctor, kernel hacker, hardware architect, concurrency-theory researcher, application developer, …

Given a Randomly Selected Human Being...

- *Any* human being: head of state, rock star, street person, farmer, researcher, student, CEO, diplomat, janitor, plumber, housewife, toddler, juvenile delinquent, bureaucrat, mafia don, warlord, mercenary soldier, terrorist, policeman, lawyer, doctor, kernel hacker, hardware architect, concurrency-theory researcher, application developer, …

- What one change would you make to this person's life?

And This is Why This Workshop Is So Important!!!

QUESTIONS?

Legal Statement