Linux-Kernel Memory Ordering Workshop

*Joint work with Jade Alglave, Luc Maranget, Andrea Parri, and Alan Stern*
Changes Since LWN Article

- simpler model: two rounds of simplification vs. strong model
  - Fewer instances of mutually assured recursion
  - Simpler model omits 2+2W, release sequences, and addrpo
    • Will add them back in if compelling use cases arise
  - Simplified cumulativity (weakened B-cumulativity)
  - More complex strong model retained as linux-kernel-hardware.cat because it more closely delineates hardware guarantees
    • Updated from LWN strong model: simplify & handle recent HW changes

- Added a full set of atomic RMW operations

- Added an early implementation of locking
  - spin_trylock(s) equivalent to cmpxchg_acquire(s, 0, 1) emulation
  - spin_unlock(s) equivalent to smp_store_release(s, 0) emulation
  - Large performance advantages over emulation!
Purpose of the Linux Kernel Memory Model

- Hoped-for benefits of a Linux-kernel memory model
  - Memory-ordering education tool (includes RCU)
  - Core-concurrent-code design aid: Automate memory-barriers.txt
  - Ease porting to new hardware and new toolchains
  - Basis for additional concurrency code-analysis tooling
    - For example, CBMC and Nidhugg (CBMC now part of rcutorture)

- Likely drawbacks of a Linux-kernel memory model
  - Extremely limited size: Handful of processes with handful of code
    - Analyze concurrency core of algorithm
    - Maybe someday automatically identifying this core
    - Perhaps even automatically stitch together multiple analyses (dream on!)
  - Limited types of operations (no function call, structures, call_rcu(), …)
    - Can emulate some of these
    - We expect that tools will become more capable over time
    - (More on this on a later slide)
Current Status and Demo

- **Release-candidate memory model:**
  - https://github.com/aparri/memory-model
  - Two rounds of simplification since the LWN article's strong model!
  - Example-driven exposition (outdated but largely accurate);

- **Lots and lots of litmus tests:**
  - https://github.com/paulmckrcu/litmus

- **Demo: How to run model and capabilities**

- **Plan: Add memory model to Linux kernel**
  - In new tools/memory-model directory
Example Simplification: “happens-before” Relation

- **LWN strong-kernel.cat hb:**
  
  ```
  let rec B-cum-propbase = (B-cum-hb ; hb* ) |
  (rfe? ; AB-cum-hb ; hb* )
  and propbase = propbase0 | B-cum-propbase
  and short-obs = ((ncoe|fre) ; propbase+ ; rfe) & int
  and obs = short-obs |
  ((hb* ; (ncoe|fre) ; propbase* ; B-cum-propbase ; rfe) & int)
  and hb = hb0 | (obs ; rfe-ppo)
  ```

- **Current linux-kernel-hardware.cat hb:**
  
  ```
  let rec prop = (overwrite & ext)? ; cumul-fence ; hb* 
  and hb = ppo | rfe | (((hb* ; prop) \ id) & int)
  ```

- **Current linux-kernel.cat hb:**
  
  ```
  let hb = ppo | rfe | ((prop \ id) & int)
  ```
RCU Full Litmus Test: Trigger on Weak CPUs?

C auto/C-RW-G+RW-Rr+RW-Ra
{
}

P0(int *x0, int *x1)
{
    r1 = READ_ONCE(*x0);
    synchronize_rcu();
    WRITE_ONCE(*x1, 1);
}

P1(int *x1, int *x2)
{
    rcu_read_lock();
    r1 = smp_load_acquire(x2);
    WRITE_ONCE(*x0, 1);
    rcu_read_unlock();

    exists
(0:r1=1 \ 1:r1=1 \ 2:r1=1)

P2(int *x2, int *x0)
{
    rcu_read_lock();
    r1 = smp_load_acquire(x2);
    WRITE_ONCE(*x0, 1);
    rcu_read_unlock();
}

https://github.com/paulmckrcu/litmus/blob/master/auto/C-RW-G%2BRW-Rr%2BRW-Ra.litmus
Same RCU Litmus Test: Trigger on Weak CPUs?

P0(int *x0, int *x1)
{
    r1 = READ_ONCE(*x0);
    synchronize_rcu();
    WRITE_ONCE(*x1, 1);
}

P1(int *x1, int *x2)
{
    r1 = READ_ONCE(*x1);
    smp_store_release(x2, 1);
    rcu_read_unlock();
}

P2(int *x2, int *x0)
{
    r1 = smp_load_acquire(x2);
    WRITE_ONCE(*x0, 1);
    rcu_read_unlock();
}

exists (0:r1=1 ∧ 1:r1=1 ∧ 2:r1=1)

https://github.com/paulmckrcu/litmus/blob/master/auto/C-RW-G%2BRW-Rr%2BRW-Ra.litmus
Current Model Capabilities ...

- READ_ONCE() and WRITE_ONCE()
- smp_store_release() and smp_load_acquire()
- rcu_assign_pointer(), rcu_dereference() and lockless_dereference()
- rcu_read_lock(), rcu_read_unlock(), and synchronize_rcu()
  - Also synchronize_rcu_expedited(), but same as synchronize_rcu()
- smp_mb(), smp_rmb(), smp_wmb(), smp_read_barrier_depends(), smp_mb__before_atomic(), and smp_mb__after_atomic()
- xchg(), xchg_relaxed(), xchg_release(), xchg_acquire(), cmpxchg(), cmpxchg_relaxed(), cmpxchg_release(), and cmpxchg_acquire()
  - Plus a great many atomic_*() functions, see linux-kernel.def for list
- spin_lock(), spin_unlock(), and spin_trylock()
And Limitations

- Compiler optimizations not modeled
- No arithmetic
- Single access size, no partially overlapping accesses
- No arrays or structs (but can do trivial linked lists)
- No dynamic memory allocation
- No interrupts, exceptions, I/O, or self-modifying code
- No functions
- No asynchronous RCU grace periods, but can emulate them:
  - Separate thread with release-acquire, grace period, and then callback code
- Locking is new and lightly tested
  - Compare suspicious results to emulations with xchg() and report any bugs!
How to Run Models

• Download herd tool as part of diy toolset
  – http://diy.inria.fr/sources/index.html

• Build as described in INSTALL.txt
  – Need ocaml v4.01.0 or better: http://caml.inria.fr/download.en.html
    • “make world.opt” – Or install from your distro (easier and faster!)
    • Recent ocaml needs opam, see diy's README

• Memory model (https://github.com/aparri/memory-model):
  – linux.def: Support pseudo-C code
  – linux-kernel.cfg: Specify Linux-kernel model
  – linux-kernel.bell: “Bell” file defining events and relationships
  – linux-kernel.cat: “Cat” file defining actual memory model
  – linux-kernel-hardware.cat: Complex model more closely describing HW

• Various litmus tests (https://github.com/paulmckrcu/litmus):
  – herd7 -conf linux-kernel.cfg C-RW-R+RW-Gr+RW-Ra.litmus
  – herd7 -conf linux-kernel.cfg C-RW-R+RW-G+RW-R.litmus
Repeat of Earlier Litmus Test: Trigger on Weak CPUs?

P0(int *x0, int *x1)
{
    r1 = READ_ONCE(*x0);
    synchronize_rcu();
    WRITE_ONCE(*x1, 1);
}

P1(int *x1, int *x2)
{
    rcu_read_lock();
    r1 = READ_ONCE(*x1);
    smp_store_release(x2, 1);
    rcu_read_unlock();
}

P2(int *x2, int *x0)
{
    rcu_read_lock();
    r1 = smp_load_acquire(x2);
    WRITE_ONCE(*x0, 1);
    rcu_read_unlock();
}

exists (0:r1=1 ∧ 1:r1=1 ∧ 2:r1=1)

https://github.com/paulmckrcu/litmus/blob/master/auto/C-RW-G%2BRW-Rr%2BRW-Ra.litmus
Running Litmus Test on Earlier Slide

$ herd7 -conf strong.cfg litmus/auto/C-RW-G+RW-Rr+RW-Ra.litmus

Test auto/C-RW-G+RW-Rr+RW-Ra Allowed

States 7

0:r1=0; 1:r1=0; 2:r1=0;
0:r1=0; 1:r1=0; 2:r1=1;
0:r1=0; 1:r1=1; 2:r1=0;
0:r1=0; 1:r1=1; 2:r1=1;
0:r1=1; 1:r1=0; 2:r1=0;
0:r1=1; 1:r1=0; 2:r1=1;
0:r1=1; 1:r1=1; 2:r1=0;

No Witnesses

Positive: 0 Negative: 7

Condition exists (0:r1=1 ∧ 1:r1=1 ∧ 2:r1=1)

Observation auto/C-RW-G+RW-Rr+RW-Ra Never 0 7

Hash=0cb6fa9aabafe5e4e28d1332afa966e3

Cannot happen
But Wait! There Are Prizes!!!

- First person to find a bug in the memory model
  - For example, a litmus test allowed by hardware with mainline Linux support, where that litmus test is prohibited by the memory model
  - Prize: Libre Computer Potato kickstarter board

- First person using memory model to find a bug in the kernel
  - For example, a missing smp_mb()
  - Consolation category: Missing comment in arch code relying on arch-specific behavior
  - Prize: Libre Computer Potato kickstarter board

- Best litmus test (counter-intuitive, biggest kernel example, ...)
  - Prize: Libre Computer Potato kickstarter board

- And a surprise consolation prize!!!
Another RCU Litmus Test: Trigger on Weak CPUs?

P0(int *x0, int *x1)
{
    r1 = READ_ONCE(*x0);
    synchronize_rcu();
    WRITE_ONCE(*x1, 1);
}

P1(int *x1, int *x2)
{
    r1 = READ_ONCE(*x1);
    WRITE_ONCE(*x2, 1);
    rcu_read_unlock();
}

P2(int *x2, int *x0)
{
    rcu_read_lock();
    r1 = READ_ONCE(*x2);
    WRITE_ONCE(*x0, 1);
    rcu_read_unlock();
}

exists (0:r1=1 ∧ 1:r1=1 ∧ 2:r1=1)

A Hierarchy of Litmus Tests: Rough Rules of Thumb

- Only one thread or only one variable: No ordering needed!
- Dependencies and rf relations everywhere
  - No additional ordering required
- If all rf relations, can replace dependencies with acquire
  - Some architecture might someday also require release, so careful!
- If only one relation is non-rf, can use release-acquire
  - Dependencies/rmb/wmb/READ_ONCE() sometimes replace acquire
  - But be safe – actually run the model to find out exactly what works!!!
- If two or more relations are non-rf, strong barriers needed
  - At least one between each non-rf relation
  - But be safe – actually run the model to find out exactly what works!!!

But for full enlightenment, see memory model itself
- https://github.com/aparri/memory-model
A Hierarchy of Memory Ordering: Rough Overheads

- **Read-write dependencies:**
  - Free everywhere

- **Read-read address dependencies:**
  - Free other than on DEC Alpha

- **Release/acquire chains and read-read control dependencies:**
  - Lightweight: Compiler barrier on x86 and mainframe, special instructions on ARM, lightweight isync or lwsync barriers on PowerPC

- **Restore sequential consistency:**
  - Full memory barriers
    - Expensive pretty much everywhere
    - But usually affect performance more than scalability
Litmus Test Exercises (1/4)

- All rf relations and dependencies
  - C-LB+ldref-o+o-ctrl-o+o-dep-o.litmus

- All rf relations but one dependency removed
  - C-LB+ldref-o+o-o+o-dep-o.litmus

- Message passing with read-to-read address dependency
  - C-MP+o-assign+o-dep-o.litmus

- Message passing with lockless_dereference()
  - C-MP+o-assign+ldref-o.litmus

- All rf relations, acquire load instead of one dependency
  - C-LB+ldref-o+acq-o+o-dep-o.litmus
Litmus Test Exercises (2/4)

- All rf relations, but all dependencies replaced by acquires
  - C-LB+acq-o+acq-o+acq-o.litmus

- One co relation, the rest remain rf relations
  - C-WWC+o+acq-o+acq-o.litmus

- One co, rest remain rf, but with release-acquire
  - C-WWC+o+o-rel+acq-o.litmus

- One co, one fr, and only one remaining rf relation
  - C-Z6.0+o-rel+acq-o+o-mb-o.litmus

- One co, one fr, one rf, and full memory barriers
  - C-Z6.0+o-mb-o+acq-o+o-mb-o.litmus
**Litmus Test Exercises (3/4)**

- One co, one fr, one rf, and all but one full memory barriers
  - C-3.SB+o-o+o-mb-o+o-mb-o.litmus

- One co, one fr, one rf, and all full memory barriers
  - C-3.SB+o-mb-o+o-mb-o+o-mb-o.litmus

- IRIW, but with release-acquire
  - C-IRIW+rel+rel+acq-o+acq-o.litmus

- Independent reads of independent writes (IRIW), full barriers
  - C-IRIW+o+o+o-mb-o+o-mb-o.litmus
Litmus Test Exercises (4/4): Kernel vs. Hardware

- **Only co: 2+2W**
  - C-2+2W+o-r+o-r.litmus
  - C-2+2W+o-wmb-o+o-wmb-o.litmus
  - herd7 -conf linux-kernel.cfg <file>.litmus
  - herd7 -conf linux-kernel.cfg -cat linux-kernel-hardware.cat <file>.litmus

- **Weaker B-cumulativity**
  - https://www.kernel.org/pub/linux/kernel/people/paulmck/LWNLinuxMM/C-wmb-is-B-cumulative.litmus

- **No release sequences (also a difference from C11)**
  - C-Mprelseq+o-r+rmwinc+a-o.litmus, C-relseq.litmus, C-relseq-not-B-cumulative.litmus

- **Additional exercises in the Examples.html file:**
## Quick Guide to Linux Kernel Memory Model

- **"rcu-path"**: Constraints on ordering based on RCU read-side critical sections and grace periods
- **"pb"**: Propagates-before, or constraints based on order of stores reaching memory (including effects of barriers)
- **"hb"**: Happens-before, or constraints based on temporal ordering
- **"ppo"**: Preserved program order, or intra-thread constraints on instruction execution
- **"coherence"**: SC Per-Variable
- **"RMW"**: Atomic Operations
"Non-Multicopy Atomic": Writes Unsynchronized

Can have r1==1 && r2==1 && r3==0

What would prohibit this outcome? (C-WRC-o+o-data-o+o-rmb-o.litmus)
Lack of Ordering For Read-Read Dependencies

```c
p->a = 1;
WRITE_ONCE(gp, p);
```

```c
p = READ_ONCE(gp);
BUG_ON(p && p->a != 1);
```

Can you write one litmus test demonstrating this and another prohibiting this?
Legal Statement

- This work represents the view of the authors and does not necessarily represent the view of their employers.
- IBM and IBM (logo) are trademarks or registered trademarks of International Business Machines Corporation in the United States and/or other countries.
- Linux is a registered trademark of Linus Torvalds.
- Other company, product, and service names may be trademarks or service marks of others.
Questions?