Linux-Kernel Community Validation Practices
Two Definitions and a Consequence

- A non-trivial software system contains at least one bug
- A reliable software system contains no known bugs

Therefore, any non-trivial reliable software system contains at least one bug that you don't know about

Yet there are more than a billion users of the Linux kernel
- In practice, validation is about reducing risk
  - Can formal verification now take a front-row seat in this risk reduction?

What would need to happen for me to include formal verification in my Linux-kernel RCU regression testing?
Current Linux-Kernel Regression Testing

- Stress-test suite example: “rcutorture”
  - http://lwn.net/Articles/154107/, http://lwn.net/Articles/622404/

- “Intelligent fuzz testing”: “trinity”, “syzkaller”
  - https://github.com/google/syzkaller/wiki/Found-Bugs

- Test suite including static analysis: “0-day test robot”
  - https://lwn.net/Articles/514278/

- Integration testing: “linux-next tree”
  - https://lwn.net/Articles/571980/

- Above is old technology – but not entirely ineffective
  - 2010: wait for -rc3 or -rc4.  2013: No problems with -rc1

- Formal verification in design, but not in regression testing
  - http://lwn.net/Articles/243851/, https://lwn.net/Articles/470681/,
  https://lwn.net/Articles/608550/
Formal Verification and Regression Testing: Requirements

(1) Either automatic translation or no translation required
   – Automatic discarding of irrelevant portions of the code
   – Manual translation provides opportunity for human error!

(2) Correctly handle environment, including memory model
   – The QRCU validation benchmark is an excellent cautionary tale

(3) Reasonable memory and CPU overhead
   – Bugs must be located in practice as well as in theory
   – Linux-kernel RCU is 15KLoC (plus 5KLoC tests) and release cycles are short

(4) Map to source code line(s) containing the bug
   – “Something is wrong somewhere” is not helpful: I already know bugs exist

(5) Modest input outside of source code under test
   – Preferably glean much of the specification from the source code itself (empirical spec!)
   – Specifications are software and can have their own bugs

(6) Find relevant bugs
   – Low false-positive rate, weight towards likelihood of occurrence (fixes create bugs!)
Discussion
Ongoing Work

- Ahmed, Groce, and Jensen: Use mutation generation and formal verification to find holes in rcutorture
  - Several holes found, one hiding a real bug
- Liang, Tautschnig, and Kroening: Experiments verifying RCU and uses of RCU using CBMC
- Alglave, Maranget, Parri, Stern, and many arch maintainers: Derive formal memory model for Linux kernel
  - Including RCU, and will drive other tool development
- I hope to someday apply L4's techniques
  - But these currently don't handle all of RCU's code
Formal Validation Tools Used and Regression Testing

- **Promela and Spin**
  - Holzmann: "The Spin Model Checker"
  - I have used Promela/Spin in design for more than 20 years, but:
    - Limited problem size, long run times, large memory consumption
    - Does not implement memory models (assumes sequential consistency)
    - Special language, difficult to translate from C

- **ARMMEM and PPCMEM (2)**
  - Alglave, Maranget, Pawan, Sarkar, Sewell, Williams, Nardelli: "PPCMEM/ARMMEM: A Tool for Exploring the POWER and ARM Memory Models"
    - Very limited problem size, long run times, large memory consumption
    - Restricted pseudo-assembly language, manual translation required

- **Herd (2, 3)**
  - Alglave, Maranget, and Tautschnig: "Herding Cats: Modelling, Simulation, Testing, and Data-mining for Weak Memory"
    - Very limited problem size (but much improved run times and memory consumption)
    - Restricted pseudo-assembly language, manual translation required

Useful, but not for regression testing
Cautiously Optimistic For Future CBMC Version

(1) Either automatic translation or no translation required
   - No translation required from C, discards irrelevant code quite well

(2) Correctly handle environment, including memory model
   - SC and TSO, hopefully will do other memory models in the future

(3) Reasonable memory and CPU overhead
   - OK for Tiny RCU and some tiny uses of concurrent RCU, Tree RCU WIP
   - Jury is out for concurrent linked-list manipulations
   - “If you live by heuristics, you will die by heuristics”

(4) Map to source code line(s) containing the bug
   - Yes, reasonably good backtrace capability

(5) Modest input outside of source code under test
   - Yes, modest boilerplate required, can use existing assertions

(6) Find relevant bugs
   - Jury still out

Formal Verification Challenge
Formal Verification Challenge

- Testing has many shortcomings
  - Cannot find bugs in code not exercised
  - Cannot reasonably exhaustively test even small software systems

- Nevertheless, a number of independently developed test harnesses have found bugs in Linux-kernel RCU
  - Trinity, 0-day test robot, -next testing, mutation testing

- As far as I know, no independently developed formal-verification model has yet found a bug in Linux-kernel RCU
  - Therefore, this challenge:
Formal Verification Challenge

- Can you verify SYSIDLE from C source?
  - Or, better yet, find a bug

- This Verification Challenge 2:

- Mathieu Desnoyers and I verified (separately) with Promela:
  - https://www.kernel.org/pub/linux/kernel/people/paulmck/validation/sysidle/

- But neither Promela/spin is not suitable for regression testing

- Can your formal-verification tool regression-test SYSIDLE?

- Or find some other Linux-kernel bug?
Legal Statement

- This work represents the view of the author and does not necessarily represent the view of IBM.
- 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.
Backup RCU Slides
RCU Removal From Linked List

- Combines waiting for readers and multiple versions:
  - Writer removes the cat's element from the list (list_del_rcu())
  - Writer waits for all readers to finish (synchronize_rcu())
  - Writer can then free the cat's element (kfree())
Waiting for Pre-Existing Readers

- Non-preemptive environment (CONFIG_PREEMPT=n)
  - RCU readers are not permitted to block
  - Same rule as for tasks holding spinlocks

- CPU context switch means all that CPU's readers are done

- Grace period ends after all CPUs execute a context switch
Toy Implementation of RCU: 20 Lines of Code

- Read-side primitives:
  ```c
  #define rcu_read_lock()
  #define rcu_read_unlock()
  #define rcu_dereference(p)  
  ({
      typeof(p) _p1 = (*(volatile typeof(p)*&)p); 
      smp_read_barrier_depends(); 
      _p1; 
  })
  ```

- Update-side primitives
  ```c
  #define rcu_assign_pointer(p, v)  
  ({
      smp_wmb(); 
      (p) = (v); 
  })
  ```
  ```c
  void synchronize_rcu(void)
  {
    int cpu;

    for_each_online_cpu(cpu) 
      run_on(cpu);
  }
  ```

Only 9 of which are needed on sequentially consistent systems
To Probe Deeper (RCU)

- https://queue.acm.org/detail.cfm?id=2488549
  - “Structured Deferral: Synchronization via Procrastination” (also in July 2013 CACM)
  - “User-Level Implementations of Read-Copy Update”
- git://lttng.org/userspace-rcu.git (User-space RCU git tree)
  - Applying RCU and weighted-balance tree to Linux mmap_sem.
  - RCU-protected resizable hash tables, both in kernel and user space
  - Combining RCU and software transactional memory
- http://wiki.cs.pdx.edu/rp/: Relativistic programming, a generalization of RCU
- http://lwn.net/Articles/262464/, http://lwn.net/Articles/263130/, http://lwn.net/Articles/264090/
  - “What is RCU?” Series
  - RCU motivation, implementations, usage patterns, performance (micro+sys)
  - System-level performance for SELinux workload: >500x improvement
  - Comparison of RCU and NBS (later appeared in JPDC)
- http://doi.acm.org/10.1145/1400097.1400099
  - History of RCU in Linux (Linux changed RCU more than vice versa)
  - Harvard University class notes on RCU (Courtesy of Eddie Koher)
To Probe Deeper (1/5)

- **Hash tables:**

- **Split counters:**
  - [http://events.linuxfoundation.org/sites/events/files/slides/BareMetal.2014.03.09a.pdf](http://events.linuxfoundation.org/sites/events/files/slides/BareMetal.2014.03.09a.pdf)

- **Perfect partitioning**
  - Candide et al: “Dynamo: Amazon’s highly available key-value store”
    - [http://doi.acm.org/10.1145/1323293.1294281](http://doi.acm.org/10.1145/1323293.1294281)
    - [http://kernel.org/pub/linux/kernel/people/paulmck/perfbook/perfbook.html](http://kernel.org/pub/linux/kernel/people/paulmck/perfbook/perfbook.html) Section 6.5
  - McKenney: “Retrofitted Parallelism Considered Grossly Suboptimal”
    - Embarrassing parallelism vs. humiliating parallelism
    - [https://www.usenix.org/conference/hotpar12/retro%EF%AC%81tted-parallelism-considered-grossly-sub-optimal](https://www.usenix.org/conference/hotpar12/retro%EF%AC%81tted-parallelism-considered-grossly-sub-optimal)
  - McKenney et al: “Experience With an Efficient Parallel Kernel Memory Allocator”
  - Bonwick et al: “Magazines and Vmem: Extending the Slab Allocator to Many CPUs and Arbitrary Resources”
    - [http://static.usenix.org/event/usenix01/full_papers/bonwick/bonwick_html/](http://static.usenix.org/event/usenix01/full_papers/bonwick/bonwick_html/)
  - Turner et al: “PerCPU Atomics”
To Probe Deeper (2/5)

- **Stream-based applications:**
  - Sutton: “Concurrent Programming With The Disruptor”
    - [http://www.youtube.com/watch?v=UvE389P6Er4](http://www.youtube.com/watch?v=UvE389P6Er4)
  - Thompson: “Mechanical Sympathy”
    - [http://mechanical-sympathy.blogspot.com/](http://mechanical-sympathy.blogspot.com/)

- **Read-only traversal to update location**
  - Arcangeli et al: “Using Read-Copy-Update Techniques for System V IPC in the Linux 2.5 Kernel”
    - [https://www.usenix.org/legacy/events/usenix03/tech/freenix03/full_papers/arcangeli/arcangeli_html/index.html](https://www.usenix.org/legacy/events/usenix03/tech/freenix03/full_papers/arcangeli/arcangeli_html/index.html)
  - Corbet: “Dcache scalability and RCU-walk”
    - [https://lwn.net/Articles/419811/](https://lwn.net/Articles/419811/)
  - Xu: “bridge: Add core IGMP snooping support”
  - Triplett et al., “Resizable, Scalable, Concurrent Hash Tables via Relativistic Programming”
  - Howard: “A Relativistic Enhancement to Software Transactional Memory”
  - McKenney et al: “URCU-Protected Hash Tables”
    - [http://lwn.net/Articles/573431/](http://lwn.net/Articles/573431/)
To Probe Deeper (3/5)

- **Hardware lock elision: Overviews**
  - Kleen: “Scaling Existing Lock-based Applications with Lock Elision”
    - [http://queue.acm.org/detail.cfm?id=2579227](http://queue.acm.org/detail.cfm?id=2579227)

- **Hardware lock elision: Hardware description**
  - POWER ISA Version 2.07
    - [http://www.power.org/documentation/power-isa-version-2-07/](http://www.power.org/documentation/power-isa-version-2-07/)
  - Intel® 64 and IA-32 Architectures Software Developer Manuals
  - Jacobi et al: “Transactional Memory Architecture and Implementation for IBM System z”

- **Hardware lock elision: Evaluations**
  - [http://kernel.org/pub/linux/kernel/people/paulmck/perfbook/perfbook.html Section 16.3](http://kernel.org/pub/linux/kernel/people/paulmck/perfbook/perfbook.html)

- **Hardware lock elision: Need for weak atomicity**
  - Herlihy et al: “Software Transactional Memory for Dynamic-Sized Data Structures”
    - [http://research.sun.com/scalable/pubs/PODC03.pdf](http://research.sun.com/scalable/pubs/PODC03.pdf)
  - Shavit et al: “Data structures in the multicore age”
    - [http://doi.acm.org/10.1145/1897852.1897873](http://doi.acm.org/10.1145/1897852.1897873)
  - Haas et al: “How FIFO is your FIFO queue?”
    - [http://dl.acm.org/citation.cfm?id=2414731](http://dl.acm.org/citation.cfm?id=2414731)
  - Gramoli et al: “Democratizing transactional programming”
    - [http://doi.acm.org/10.1145/2541883.2541900](http://doi.acm.org/10.1145/2541883.2541900)
To Probe Deeper (4/5)

**RCU**
- Desnoyers et al.: “User-Level Implementations of Read-Copy Update”
- McKenney et al.: “RCU Usage In the Linux Kernel: One Decade Later”
- McKenney: “Structured deferral: synchronization via procrastination”
  - http://doi.acm.org/10.1145/2483852.2483867
- McKenney et al.: “User-space RCU” https://lwn.net/Articles/573424/

**Possible future additions**
- Boyd-Wickizer: “Optimizing Communications Bottlenecks in Multiprocessor Operating Systems Kernels”
- McKenney: “N4037: Non-Transactional Implementation of Atomic Tree Move”
- McKenney: “C++ Memory Model Meets High-Update-Rate Data Structures”
To Probe Deeper (5/5)

- RCU theory and semantics, academic contributions (partial list)
  - Gamsa et al., “Tornado: Maximizing Locality and Concurrency in a Shared Memory
    Multiprocessor Operating System”
    • http://www.usenix.org/events/osdi99/full_papers/gamsa/gamsa.pdf
  - McKenney, “Exploiting Deferred Destruction: An Analysis of RCU Techniques”
  - Hart, “Applying Lock-free Techniques to the Linux Kernel”
    • http://www.cs.toronto.edu/~tomhart/masters_thesis.html
  - Olsson et al., “TRASH: A dynamic LC-trie and hash data structure”
    • http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4281239
  - Desnoyers, “Low-Impact Operating System Tracing”
  - Dalton, “The Design and Implementation of Dynamic Information Flow Tracking ...”
  - Gotsman et al., “Verifying Highly Concurrent Algorithms with Grace (extended version)”
  - Liu et al., “Mindicators: A Scalable Approach to Quiescence”
    • http://dx.doi.org/10.1109/ICDCS.2013.39
  - Tu et al., “Speedy Transactions in Multicore In-memory Databases”
    • http://doi.acm.org/10.1145/2517349.2522713
  - Arbel et al., “Concurrent Updates with RCU: Search Tree as an Example”
    • http://www.cs.technion.ac.il/~mayaarl/podc047f.pdf
Backup Promela/PPCMEM/Herd Slides
Promela Model of Incorrect Atomic Increment (1/2)

```
#define NUMPROCS 2

byte counter = 0;
byte progress[NUMPROCS];

proctype incremener(byte me)
{
      int temp;
      temp = counter;
      counter = temp + 1;
      progress[me] = 1;
}
```
Promela Model of Incorrect Atomic Increment (2/2)

```plaintext
15 init {
16   int i = 0;
17   int sum = 0;
18
19   atomic {
20     i = 0;
21     do
22        :: i < NUMPROCS ->
23           progress[i] = 0;
24           run incremerter(i);
25           i++
26        :: i >= NUMPROCS -> break
27     od;
28   }
29   atomic {
30     i = 0;
31     sum = 0;
32     do
33        :: i < NUMPROCS ->
34           sum = sum + progress[i];
35           i++
36        :: i >= NUMPROCS -> break
37     od;
38     assert(sum < NUMPROCS || counter == NUMPROCS)
39   }
40 }
```
PPC IRIW.litmus

""

(* Traditional IRIW. *)

{ 0:r1=1; 0:r2=x;
  1:r1=1; 1:r4=y;
  2:  2:r2=x; 2:r4=y;
  3:  3:r2=x; 3:r4=y;
}

P0 | P1 | P2 | P3
---|---|---|---
stw r1,0(r2) | stw r1,0(r4) | lwz r3,0(r2) | lwz r3,0(r4)
| | sync | sync
| | | lwz r5,0(r4) | lwz r5,0(r2)

exists
(2:r3=1 \ 2:r5=0 \ 3:r3=1 \ 3:r5=0)

Fourteen CPU hours and 10 GB of memory
Herd Example Litmus Test for Incorrect IRIW

```
PPC IRIW-lwsync-f.litmus ""
(* Traditional IRIW. *)
{
0: r1=1; 0: r2=x; 1: r1=1; 1: r4=y; 2: 2: r2=x; 2: r4=y; 3: 3: r2=x; 3: r4=y;
}
P0 | P1 | P2 | P3
stw r1,0(r2) | stw r1,0(r4) | lwz r3,0(r2) | lwz r3,0(r4)
stw r3,0(r2) | lwsync | lwz r5,0(r4) | lwsync
| lwz r5,0(r4) | lwsync |

exists
(2:r3=1 /\ 2:r5=0 /\ 3:r3=1 /\ 3:r5=0)

. . .

Positive: 1 Negative: 15
Condition exists (2:r3=1 /\ 2:r5=0 /\ 3:r3=1 /\ 3:r5=0)
Observation IRIW Sometimes 1 15