So You Want To Torture RCU?
A Round For Those Torturing SW!!!

- Oday test robot
- kernelci
- -next tree
- hulk test robot
- syzkaller
- kselftest

- trinity
- coccinelle
- smatch
- Linux Test Project
- ktest
- And many many more!!!
“Shut Up And Start Torturing!!!”
“Shut Up And Start Torturing!!!”

• Given suitable system running qemu & kvm:

PATH="tools/testing/selftests/rcutorture/bin:$PATH" export PATH

kvm.sh # Default 30 minutes (AKA 30m) on each of 19 scenarios.
kvm.sh --cpus 64 # Run scenarios concurrently in two batches of 30 minutes each.
kvm.sh --allcpus --duration 1d # Weekend run.
kvm.sh --cpus 128 --duration 12h --trust-make # All scenarios in one batch.
Kvm.sh --cpus 16 --configs "2*SRCU-N 2*SRCU-P 2*SRCU-t 2*SRCU-u" # 2x each SRCU scenario.
kvm-again.sh /path/to/old/run/results --duration 45s # No kernel builds.
kvm-remote.sh “sys1 sys2 ... sys20” --cpus 80 --configs “TREE10 15*CFLIST”
# Replace “sys1” etc. with names of systems you can non-interactively ssh to.

https://paulmck.livejournal.com/57769.html
What Does Success Look Like?

RUDE01 ------- 2102 GPs (7.00667/s) [tasks-rude: g0 f0x0 ]
SRCU-N ------- 42229 GPs (140.763/s) [srcu: g549860 f0x0 ]
SRCU-P ------- 11887 GPs (39.6233/s) [srcud: g110444 f0x0 ]
SRCU-t ------- 59641 GPs (198.803/s) [srcu: g1 f0x0 ]
SRCU-u ------- 59209 GPs (197.363/s) [srcud: g1 f0x0 ]
TASKS01 ------- 1029 GPs (3.43/s) [tasks: g0 f0x0 ]
TASKS02 ------- 1043 GPs (3.47667/s) [tasks: g0 f0x0 ]
TASKS03 ------- 1019 GPs (3.39667/s) [tasks: g0 f0x0 ]
TINY01 ------- 43373 GPs (144.577/s) [rcu: g0 f0x0 ] n_max_cbs: 34463
TINY02 ------- 46519 GPs (155.063/s) [rcu: g0 f0x0 ] n_max_cbs: 2197
TRACE01 ------- 756 GPs (2.52/s) [tasks-tracing: g0 f0x0 ]
TRACE02 ------- 559 GPs (1.86333/s) [tasks-tracing: g0 f0x0 ]
TREE01 ------- 8930 GPs (29.7667/s) [rcu: g64765 f0x0 ]
TREE02 ------- 17514 GPs (58.38/s) [rcu: g138645 f0x0 ] n_max_cbs: 18010
TREE03 ------- 15920 GPs (53.0667/s) [rcu: g159973 f0x0 ] n_max_cbs: 1025308
TREE04 ------- 10821 GPs (36.07/s) [rcu: g70293 f0x0 ] n_max_cbs: 81293
TREE05 ------- 16942 GPs (56.4733/s) [rcu: g123745 f0x0 ] n_max_cbs: 99796
TREE07 ------- 8248 GPs (27.4933/s) [rcu: g52933 f0x0 ] n_max_cbs: 183589
TREE09 ------- 39903 GPs (133.01/s) [rcu: g717745 f0x0 ] n_max_cbs: 83002

Plus exit code of zero, which can be useful when bisecting.
Other kvm.sh Parameters?

- **--kconfig**: Specify Kconfig options
- **--bootargs**: Specify kernel-boot parameters
- **--kasan**: Run under KASAN
- **--kcsan**: Run under KCSAN (big binaries!)
  - Also requires very recent compilers
- **--torture**: `rcu`, `lock`, `scf`, `refscale`, `rcuscale`
- **--dryrun scenarios**: Show batches for given CPUs/configs
  - Useful for working out what `--config` argument to specify

“I Cannot Decide What to Torture!!!”

- Use the torture.sh script
- By default, tortures a little of everything
  - Takes about 12 hours on a heavy-duty laptop
- Many arguments to control its torturing

https://paulmck.livejournal.com/61587.html
Arguments to `torture.sh`

- **--do-all**: Everything including KCSAN (disabled by default)
- **--do-none**: Nothing
- **--do-kasan, --do-kcsan**: Enable debugging
- **Select tests:**
- **Select scenarios:**
  - **--configs-rcutorture**, **--configs-locktorture**, **--configs-scftorture**
- **--duration**: Nominal duration: 10m → 11h on 16 CPUs
“But I Want To Use A Debugger!!!”
“But I Want To Use A Debugger!!!”

• **--gdb is your friend:**
  
  ```
  $ kvm.sh --allcpus --torture lock --configs LOCK05 --gdb
  
  Waiting for you to attach a debug session, for example:
  
  gdb tools/testing/selftests/rcutorture/res/2020.08.27-14.51/LOCK05/vmlinux
  
  After symbols load and the "(gdb)" prompt appears:
  
  target remote :1234
  
  continue
  ```

• Once you have connected, use gdb commands
  
  - But “hbreak” instead of “break”
  
  - The Linux kernel is not fond of software breakpoints

https://paulmck.livejournal.com/58616.html
“I Found a Bug!!! What Now???”
“I Found a Bug!!! What Now???”

- Fix it and post the patch? ;-)
- With RCU, heisenbugs are the common case
  - So make it happen more often!
How to De-Heisenbug Bugs???

• It is not always easy, but here are a few tricks...
How to De-Heisenbug Bugs???

- Adjust CPUs to increase probability by factor of \( M \)
  - \( M \)x fewer runs with \( M \)x more CPUs each
    - If races are between many random kthreads
  - \( M \)x more runs with \( M \)x fewer CPUs each
    - If races are between a few specific kthreads
- This is theory: The real world does what it pleases

https://paulmck.livejournal.com/58187.html
How to De-Heisenbug Bugs???

• Make risky operations happen more frequently!
  - CPU hotplug is one of the usual suspects:
    \texttt{rcutorture.onoff\_interval=200}
  - Long-lived readers (automatic)
  - Full-system idle \texttt{rcutorture.stutter}
  - Callback floods \texttt{rcutorture.fwd\_progress}
  - vCPU preemption \texttt{kvm.sh --jitter \textasciitilde N us-sleep us-spin}
    • See next slide

https://paulmck.livejournal.com/58077.html
The jitter.sh script binds to randomly selected CPUs, forcing preemption, even when that vCPU thinks that it has disabled interrupts.
How to De-Heisenbug Bugs???

• Which scenarios cause the problem most frequently?
  - Use --configs to run those scenarios
  - Use config2csv.sh to compare configurations
  - Double down on suspected accelerators
    • Kconfig options and/or kernel boot parameters
    • Modify kernel or scripts in some cases

https://paulmck.livejournal.com/58077.html
Kernel Module, Not Userspace

Credit: Awakening Conscience, licensed under the Creative Commons Attribution-Share Alike 4.0 International license
Kernel Module, Not Userspace
Kernel Module, Not Userspace

Credit: Awakening Conscience, licensed under the Creative Commons Attribution-Share Alike 4.0 International license
How to De-Heisenbug Bugs???

- Enlist the aid of the laws of physics!!!
- The speed of light is too slow and atoms are too big
  - Hence memory latency and NUMA effects
- For each rcutorture guest OS:
  - Place two CPUs in one hyperthreaded core
  - Place two other CPUs in another core
    - Preferably on some other socket
- Greatly increases probability of some types of period races
  - Accesses take longer only sometimes, raise collision cross section

The effects were not subtle: https://paulmck.livejournal.com/62071.html
Semantics
RCU Semantics (Graphical)

```
rcu_read_lock()
rcu_read_unlock()
synchronize_rcu()
[return]
```

```
rcu_read_lock()
synchronize_rcu()
[return]
Free Old Memory
```

```
rcu_read_lock()

rcu_read_unlock()
Free Old Memory
```

```
rcu_read_lock()

rcu_read_unlock()
Free Old Memory
```

```
rcu_read_lock()

rcu_read_unlock()
Free Old Memory
```

```
rcu_read_lock()

rcu_read_unlock()
Free Old Memory
```

```
rcu_read_lock()

rcu_read_unlock()
Free Old Memory
```

```
rcu_read_lock()

rcu_read_unlock()
Free Old Memory
```

```
rcu_read_lock()

rcu_read_unlock()
Free Old Memory
```
RCU Semantics (API)

- RCU has simple semantics:
  - RCU grace period must wait for all pre-existing RCU readers
- Trivial textbook RCU implementation:

```c
#define rcu_read_lock() __asm__ __volatile__("": : "memory")
#define rcu_read_unlock() __asm__ __volatile__("": : "memory")
#define rcu_dereference(p)\
  ({
    typeof(*p) *__p1 = READ_ONCE(p);\
    __p1;
  })
#define rcu_assign_pointer(p, v) smp_store_release((p), (v))
void synchronize_rcu(void)
{
  int cpu;

  for_each_online_cpu(cpu)
    sched_setaffinity(current->pid, cpumask_of(cpu));
}
```
Just a Few Linux-Kernel Issues...

- Systems with 1000s of CPUs
- Sub-20-microsecond real-time response requirements
- CPUs can come and go ("CPU hotplug")
- If you disturb idle CPUs, you enrage low-power embedded folks
- Forward progress requirements: callbacks, network DoS attacks
- RCU grace periods must provide extremely strong ordering
- RCU uses the scheduler, and the scheduler uses RCU
- Firmware sometimes lies about the number and age of CPUs
- RCU must work during early boot, even before RCU initialization
- Preemption can happen, even when interrupts are disabled (vCPUs!)
- RCU should avoid exploitable ease-of-use issues
Just a Few Linux-Kernel Issues...

- Systems with 1000s of CPUs
- Sub-20-microsecond real-time response requirements
- CPUs can come and go (“CPU hotplug”)
- If you disturb idle CPUs, you enrage low-power embedded folks
- Forward progress requirements: callbacks, network DoS attacks
- RCU grace periods must provide extremely strong ordering
- RCU uses the scheduler, and the scheduler uses RCU
- Firmware sometimes lies about the number and age of CPUs
- RCU must work during early boot, even before RCU initialization
- Preemption can happen, even when interrupts are disabled (vCPUs!)
- RCU should avoid exploitable ease-of-use issues
Not Just a Theoretical Possibility...

[Diagram with text and link]

Credit: Awakening Conscience, licensed under the Creative Commons Attribution-Share Alike 4.0 International license
Many Types of RCU Readers (Old)

rcu_read_lock();
do_something_1();
preempt_disable();
do_something_2();
rcu_read_unlock();
do_something_3();
preempt_enable();

{ local_bh_disable();
do_something_1();
rcu_read_lock();
local_bh_enable();
do_something_2();
preempt_disable();
rcu_read_unlock();
do_something_3();
preempt_enable();
}
Many Types of RCU Readers (New)

rcu_read_lock();
do_something_1();
preempt_disable();
do_something_2();
rcu_read_unlock();
do_something_3();
preempt_enable();

local_bh_disable();
do_something_1();
rcu_read_lock();
local_bh_enable();
do_something_2();
preempt_disable();
rcu_read_unlock();
do_something_3();
preempt_enable();
Many Types of RCU Readers (New)

rcu_read_lock();
do_something_1();
preempt_disable();
do_something_2();
rcu_read_unlock();
do_something_3();
preempt_enable();

local_bh_disable();
do_something_1();
rcu_read_lock();
local_bh_enable();
do_something_2();
preempt_disable();
rcu_read_unlock();
do_something_3();
preempt_enable();

Therefore, rcutorture must randomly generate overlapping readers.
Many Types of RCU Readers (New)

```c
rcu_read_lock();
do_something_1();
preempt_disable();
do_something_2();
rcu_read_unlock();
do_something_3();
preempt_enable();
```

```c
local_bh_disable();
do_something_1();
rcu_read_lock();
local_bh_enable();
do_something_2();
preempt_disable();
rcu_read_unlock();
do_something_3();
preempt_enable();
```

Therefore, rcutorture must randomly generate overlapping readers. **But including nested readers!!!**
Here is Your Elegant Synchronization Mechanism:

Photo by "Golden Trvs Gol twister", CC by SA 3.0
Here is Your Elegant Synchronization Mechanism Equipped to Survive in The Linux Kernel:
Semantics are the Tip of the Iceberg

Formal RCU Semantics

Informal RCU Semantics

Requirements
Semantics are the Tip of the Iceberg

A few short years ago, there were absolutely no formal RCU semantics!!!
Software Engineering
Software Engineering

• RCU contains 17,682 LoC (including comments, etc.)
• 1-3 bugs/KLoC for production-quality code: **18-53 bugs**
  - Best case I have seen: 0.04 bugs/KLoC for safety-critical code
    • Extreme code-style restrictions, single-threaded, formal methods, …
    • And still way more than zero bugs!!!  :-)

• Median age of an RCU LoC is less than four years
  - And young code tends to be buggier than old code!

• We should therefore expect a few tens more bugs!!!
Software Engineering

- RCU contains 17,682 LoC (including comments, etc.)
- 1-3 bugs/KLoC for production-quality code: **18-53 bugs**
  - Best case I have seen: 0.04 bugs/KLoC for safety-critical code
    - Extreme code-style restrictions, single-threaded, formal methods, ...
    - And still way more than zero bugs!!! :-)
- Median age of an RCU LoC is less than four years
  - And young code tends to be buggier than old code!
- We should therefore expect a few tens more bugs!!!
- An rcutorture run that “succeeds” has failed to find them!!!

Linux kernel v5.11
Installed Base
Installed Base

Million-Year Bug? Once In a Million Years!!!
Installed Base

Million-Year Bug? Once In a Million Years!!!
Murphy is a nice guy: Everything that can happen, will...

1975
NHS
Installed Base

Million-Year Bug? Once In a Million Years!!
Murphy is a nice guy: Everything that can happen, will...
...maybe in geologic time

1975
NHS

1
Installed Base

Million-Year Bug? Once in Ten Millennia

1975
NHS

1985
Various

1
10
100
Installed Base

Million-Year Bug? Once per Century

1975 NHS
1985 Various
1995 SQNT
Installed Base

Million-Year Bug? Once a Month
Installed Base

Million-Year Bug? Several Times per Hour

1975 NHS
1985 Various
1995 SQNT
2005 Linux
2015 Linux
2017 Linux
Installed Base

Million-Year Bug? You don't want to know...

1975 NHS
1985 Various
1995 SQNT
2005 Linux
2015 Linux
2017 Linux
IoT?
Installed Base

Million-Year Bug? You don't want to know... But has Murphy transitioned from a nice guy into a homicidal maniac?
Natural Selection
Natural Selection
Natural Selection: Not Just Lifeforms

Software (Randomly Generated)

Validation (Selection!)

Robust Software

Fewer (?) Injected Bugs

Bugs
Natural Selection: Bugs are Software!

Software (Randomly Generated) → Fewer (?) Injected Bugs

Validation (Selection!) → Bugs

Software – And Bugs Adapted to Validation
Natural Selection: Bugs are Software!

- Software (Randomly Generated)
- Fewer (?) Injected Bugs
- Validation (Selection!)
- Software – And Bugs Adapted to Validation
- Bug Reports: Improve Validation
Validate Only Intended Use Cases

Current Validated Use Cases
Major Development Generates Bug

Current Validated Use Cases
After Validation and Bug Fixing

Current Validated Use Cases
After Another Round of Development

Current Validated Use Cases
More Validation and Bug Fixing

Current Validated Use Cases
New Use Cases: Walls of Bugs!!!
Natural Selection: Bugs are Software!

- Software (Randomly Generated)
- Fewer (?) Injected Bugs
- Bugs
- Bug Reports And Paranoia: Improve Validation
- Software – And Bugs Adapted to Validation
- Validation (Selection!)

Software and bugs adapt to validation, resulting in fewer injected bugs.
“Natural Selection” is a Euphemism

If your tests are not failing, they are not helping to improve your software

https://paulmck.livejournal.com/61158.html
Summary

- How to torture RCU, including using gdb
- Tracking down heisenbugs
- The role of RCU semantics
  - Limited but perhaps increasing over time
- Validation via natural selection, good, bad, ugly
Once Again, Applause for Testers!!!
But Watch Out For These Guys!!!
But Watch Out For These Guys!!!

Yes, they tortured software with new workloads, but that wasn’t enough for them!
These Guys *Banished SW*!!!
These Guys *Banished* SW!!! To Mars!
For More Information (1/2)

- Validating RCU in particular and concurrent software in general:
    - “Validation” chapter, especially “Probability and Heisenbugs” section
    - “Formal Verification” chapter
  - “Hunting Heisenbugs” blog posts:
    - https://paulmck.livejournal.com/14639.html
    - https://paulmck.livejournal.com/14969.html
- Linux-kernel source code:
  - kernel/rcu/{rcutorture.c,rcuref.c,rcuscale.c}, kernel/torture.c, kernel/locking/locktorture.c
  - tools/testing/ tools/testing/selftests/rcutorture
For More Information (2/2)

- RCU specification, which is a function of time:
  - Documentation/RCU/Design/Requirements/ in kernel source
  - “RCU Usage In the Linux Kernel: One Decade Later”: