CONFIG_RCU_TRACE debugfs Files and Formats

The rcutree implementation of RCU provides debugfs trace output that summarizes counters and state. This information is useful for debugging RCU itself, and can sometimes also help to debug abuses of RCU. The following sections describe the debugfs files and formats.

Hierarchical RCU debugfs Files and Formats

This implementation of RCU provides three debugfs files under the top-level directory RCU: rcu/rcudata (which displays fields in struct rcu_data), rcu/rcugp (which displays grace-period counters), and rcu/rcuhier (which displays the struct rcu_node hierarchy).

The output of "cat rcu/rcudata" looks as follows:

```
rcu_sched:
  0 c=17829 g=17829 pq=1 pqc=17829 qp=0 dt=10951/1 dn=0 df=1101 of=0 ri=36 ql=0
b=10
  1 c=17829 g=17829 pq=1 pqc=17829 qp=0 dt=16117/1 dn=0 df=1015 of=0 ri=0 ql=0
b=10
  2 c=17829 g=17829 pq=1 pqc=17829 qp=0 dt=1445/1 dn=0 df=1839 of=0 ri=0 ql=0
b=10
  3 c=17829 g=17829 pq=1 pqc=17829 qp=0 dt=6681/1 dn=0 df=1545 of=0 ri=0 ql=0
b=10
  4 c=17829 g=17829 pq=1 pqc=17829 qp=0 dt=1003/1 dn=0 df=1992 of=0 ri=0 ql=0
b=10
  5 c=17829 g=17830 pq=1 pqc=17829 qp=1 dt=3887/1 dn=0 df=3331 of=0 ri=4 ql=2
b=10
  6 c=17829 g=17829 pq=1 pqc=17829 qp=0 dt=859/1 dn=0 df=3224 of=0 ri=0 ql=0
b=10
  7 c=17829 g=17830 pq=0 pqc=17829 qp=1 dt=3761/1 dn=0 df=1818 of=0 ri=0 ql=2
b=10
rcu_bh:
  0 c=-275 g=-275 pq=1 pqc=-275 qp=0 dt=10951/1 dn=0 df=0 of=0 ri=0 ql=0 b=10
  1 c=-275 g=-275 pq=1 pqc=-275 qp=0 dt=16117/1 dn=0 df=13 of=0 ri=0 ql=0 b=10
  2 c=-275 g=-275 pq=1 pqc=-275 qp=0 dt=1445/1 dn=0 df=15 of=0 ri=0 ql=0 b=10
  3 c=-275 g=-275 pq=1 pqc=-275 qp=0 dt=6681/1 dn=0 df=9 of=0 ri=0 ql=0 b=10
  4 c=-275 g=-275 pq=1 pqc=-275 qp=0 dt=1003/1 dn=0 df=15 of=0 ri=0 ql=0 b=10
  5 c=-275 g=-275 pq=1 pqc=-275 qp=0 dt=3887/1 dn=0 df=15 of=0 ri=0 ql=0 b=10
  6 c=-275 g=-275 pq=1 pqc=-275 qp=0 dt=859/1 dn=0 df=15 of=0 ri=0 ql=0 b=10
  7 c=-275 g=-275 pq=1 pqc=-275 qp=0 dt=3761/1 dn=0 df=15 of=0 ri=0 ql=0 b=10
```

The first section lists the rcu_data structures for rcu_sched, the second for rcu_bh. Note that CONFIG_TREE_PREEMPT_RCU kernels will have an additional section for rcu_preempt. Each section has one line per CPU, or eight for this 8-CPU system. The fields are as follows:

o The number at the beginning of each line is the CPU number. CPUs numbers followed by an exclamation mark are offline, but have been online at least once since boot. There will be no output for CPUs that have never been online, which can be a good thing in the surprisingly common case where NR_CPUS is substantially larger than the number of actual CPUs.

o "c" is the count of grace periods that this CPU believes have completed. CPUs in dynticks idle mode may lag quite a ways behind, for example, CPU 4 under "rcu_sched" above, which has slept through the past 25 RCU grace periods. It is not unusual to see CPUs lagging by thousands of grace periods.

o "g" is the count of grace periods that this CPU believes have started. Again, CPUs in dynticks idle mode may lag behind. If the "c" and "g" values are equal, this CPU has already reported a quiescent state for the last RCU grace period that it is aware of, otherwise, the CPU believes that it owes RCU a quiescent state.

o "pq" indicates that this CPU has passed through a quiescent state for the current grace period. It is possible for "pq" to be "1" and "c" different than "g", which indicates that although the CPU has passed through a quiescent state, either (1) this CPU has not yet reported that fact, (2) some other CPU has not yet reported for this grace period, or (3) both.

o "pqc" indicates which grace period the last-observed quiescent state for this CPU corresponds to. This is important for handling the race between CPU 0 reporting an extended dynticks-idle quiescent state for CPU 1 and CPU 1 suddenly waking up and reporting its own quiescent state. If CPU 1 was the last CPU for the current grace period, then the CPU that loses this race will attempt to incorrectly mark CPU 1 as having checked in for the next grace period!

o "qp" indicates that RCU still expects a quiescent state from this CPU.

o "dt" is the current value of the dyntick counter that is incremented when entering or leaving dynticks idle state, either by the scheduler or by irq. The number after the "/" is the interrupt nesting depth when in dyntick-idle state, or one greater than the interrupt-nesting depth otherwise.

This field is displayed only for CONFIG_NO_HZ kernels.

o "dn" is the current value of the dyntick counter that is incremented when entering or leaving dynticks idle state via NMI. If both the "dt" and "dn" values are even, then this CPU is in dynticks idle mode and may be ignored by RCU. If either of these two counters is odd, then RCU must be alert to the possibility of an RCU read-side critical section running on this CPU.

This field is displayed only for CONFIG_NO_HZ kernels.

o "df" is the number of times that some other CPU has forced a quiescent state on behalf of this CPU due to this CPU being in dynticks-idle state.

This field is displayed only for CONFIG_NO_HZ kernels.

o "of" is the number of times that some other CPU has forced a quiescent state on behalf of this CPU due to this CPU being offline. In a perfect world, this might neve happen, but it turns out that offlining and onlining a CPU can take several grace periods, and so there is likely to be an extended period of time when RCU believes that the CPU is online when it really is not. Please note that erring in the other direction (RCU believing a CPU is offline when it is really alive and kicking) is a fatal error, so it makes sense to err conservatively.

o "ri" is the number of times that RCU has seen fit to send a reschedule IPI to this CPU in order to get it to report a quiescent state.

o "ql" is the number of RCU callbacks currently residing on this CPU. This is the total number of callbacks, regardless of what state they are in (new, waiting for grace period to start, waiting for grace period to end, ready to invoke).

o "b" is the batch limit for this CPU. If more than this number of RCU callbacks is ready to invoke, then the remainder will be deferred.

There is also an rcu/rcudata.csv file with the same information in comma-separated-variable spreadsheet format.

The output of "cat rcu/rcugp" looks as follows:

rcu_sched: completed=33062 gpnum=33063
rcu_bh: completed=464 gpnum=464

Again, this output is for both "rcu_sched" and "rcu_bh". Note that kernels built with CONFIG_TREE_PREEMPT_RCU will have an additional "rcu_preempt" line. The fields are taken from the rcu_state structure, and are as follows:

o "completed" is the number of grace periods that have completed. It is comparable to the "c" field from rcu/rcudata in that a CPU whose "c" field matches the value of "completed" is aware that the corresponding RCU grace period has completed.

o "gpnum" is the number of grace periods that have started. It is comparable to the "g" field from rcu/rcudata in that a CPU whose "g" field matches the value of "gpnum" is aware that the corresponding RCU grace period has started.

If these two fields are equal (as they are for "rcu_bh" above), then there is no grace period in progress, in other words, RCU is idle. On the other hand, if the two fields differ (as they do for "rcu_sched" above), then an RCU grace period is in progress.

The output of "cat rcu/rcuhier" looks as follows, with very long lines:

c=6902 g=6903 s=2 jfq=3 j=72c7 nfqs=13142/nfqsng=0(13142) fqlh=6 oqlen=0

```
1/1 .>. 0:127 ^0
3/3 .>. 0:35 ^0    0/0 .>. 36:71 ^1    0/0 .>. 72:107 ^2    0/0 .>. 108:127 ^3

3/3f .>. 0:5 ^0    2/3 .>. 6:11 ^1    0/0 .>. 12:17 ^2    0/0 .>. 18:23 ^3
0/0 .>. 24:29 ^4    0/0 .>. 30:35 ^5    0/0 .>. 36:41 ^0    0/0 .>. 42:47 ^1
0/0 .>. 48:53 ^2    0/0 .>. 54:59 ^3    0/0 .>. 60:65 ^4    0/0 .>. 66:71 ^5
0/0 .>. 72:77 ^0    0/0 .>. 78:83 ^1    0/0 .>. 84:89 ^2    0/0 .>. 90:95 ^3
0/0 .>. 96:101 ^4    0/0 .>. 102:107 ^5    0/0 .>. 108:113 ^0    0/0 .>. 114:119
^1    0/0 .>. 120:125 ^2    0/0 .>. 126:127 ^3
rcu_bh:
c=-226 g=-226 s=1 jfq=-5701 j=72c7 nfqs=88/nfqsng=0(88) fqlh=0 oqlen=0
0/1 .>. 0:127 ^0
0/3 .>. 0:35 ^0    0/0 .>. 36:71 ^1    0/0 .>. 72:107 ^2    0/0 .>. 108:127 ^3

0/3f .>. 0:5 ^0    0/3 .>. 6:11 ^1    0/0 .>. 12:17 ^2    0/0 .>. 18:23 ^3
0/0 .>. 24:29 ^4    0/0 .>. 30:35 ^5    0/0 .>. 36:41 ^0    0/0 .>. 42:47 ^1
0/0 .>. 48:53 ^2    0/0 .>. 54:59 ^3    0/0 .>. 60:65 ^4    0/0 .>. 66:71 ^5
0/0 .>. 72:77 ^0    0/0 .>. 78:83 ^1    0/0 .>. 84:89 ^2    0/0 .>. 90:95 ^3
0/0 .>. 96:101 ^4    0/0 .>. 102:107 ^5    0/0 .>. 108:113 ^0    0/0 .>. 114:119
^1    0/0 .>. 120:125 ^2    0/0 .>. 126:127 ^3
```

This is once again split into "rcu_sched" and "rcu_bh" portions,
and CONFIG_TREE_PREEMPT_RCU kernels will again have an additional
"rcu_preempt" section. The fields are as follows:

- "c" is exactly the same as "completed" under rcu/rcugp.

- "g" is exactly the same as "gpnum" under rcu/rcugp.

- "s" is the "signaled" state that drives force_quiescent_state()'s
  state machine.

- "jfq" is the number of jiffies remaining for this grace period
  before force_quiescent_state() is invoked to help push things
  along. Note that CPUs in dyntick-idle mode thoughout the grace
  period will not report on their own, but rather must be check by
  some other CPU via force_quiescent_state().

- "j" is the low-order four hex digits of the jiffies counter.
  Yes, Paul did run into a number of problems that turned out to
  be due to the jiffies counter no longer counting. Why do you ask?

- "nfqs" is the number of calls to force_quiescent_state() since
  boot.

- "nfqsng" is the number of useless calls to force_quiescent_state(),
  where there wasn't actually a grace period active. This can
  happen due to races. The number in parentheses is the difference
  between "nfqs" and "nfqsng", or the number of times that
  force_quiescent_state() actually did some real work.

- "fqlh" is the number of calls to force_quiescent_state() that
  exited immediately (without even being counted in nfqs above)
  due to contention on ->fqslock.

- "oqlen" is the number of callbacks on the "orphan" callback

list. RCU callbacks are placed on this list by CPUs going offline, and are "adopted" either by the CPU helping the outgoing CPU or by the next rcu_barrier*() call, whichever comes first.

o Each element of the form "1/1 0:127 ^0" represents one struct rcu_node. Each line represents one level of the hierarchy, from root to leaves. It is best to think of the rcu_data structures as forming yet another level after the leaves. Note that there might be either one, two, or three levels of rcu_node structures, depending on the relationship between CONFIG_RCU_FANOUT and CONFIG_NR_CPUS.

  o The numbers separated by the "/" are the qsmask followed by the qsmaskinit. The qsmask will have one bit set for each entity in the next lower level that has not yet checked in for the current grace period. The qsmaskinit will have one bit for each entity that is currently expected to check in during each grace period. The value of qsmaskinit is assigned to that of qsmask at the beginning of each grace period.

    For example, for "rcu_sched", the qsmask of the first entry of the lowest level is 0x14, meaning that we are still waiting for CPUs 2 and 4 to check in for the current grace period.

  o The characters separated by the ">" indicate the state of the blocked-tasks lists. A "T" preceding the ">" indicates that at least one task blocked in an RCU read-side critical section blocks the current grace period, while a "." preceding the ">" indicates otherwise. The character following the ">" indicates similarly for the next grace period. A "T" should appear in this field only for rcu-preempt.

  o The numbers separated by the ":" are the range of CPUs served by this struct rcu_node. This can be helpful in working out how the hierarchy is wired together.

    For example, the first entry at the lowest level shows "0:5", indicating that it covers CPUs 0 through 5.

  o The number after the "^" indicates the bit in the next higher level rcu_node structure that this rcu_node structure corresponds to.

    For example, the first entry at the lowest level shows "^0", indicating that it corresponds to bit zero in the first entry at the middle level.

The output of "cat rcu/rcu_pending" looks as follows:

```
rcu_sched:
  0 np=255892 qsp=53936 rpq=85 cbr=0 cng=14417 gpc=10033 gps=24320 nf=6445
nn=146741
```

```
  1 np=261224 qsp=54638 rpq=33 cbr=0 cng=25723 gpc=16310 gps=2849 nf=5912
nn=155792
  2 np=237496 qsp=49664 rpq=23 cbr=0 cng=2762 gpc=45478 gps=1762 nf=1201
nn=136629
  3 np=236249 qsp=48766 rpq=98 cbr=0 cng=286 gpc=48049 gps=1218 nf=207 nn=137723
  4 np=221310 qsp=46850 rpq=7 cbr=0 cng=26 gpc=43161 gps=4634 nf=3529 nn=123110
  5 np=237332 qsp=48449 rpq=9 cbr=0 cng=54 gpc=47920 gps=3252 nf=201 nn=137456
  6 np=219995 qsp=46718 rpq=12 cbr=0 cng=50 gpc=42098 gps=6093 nf=4202 nn=120834
  7 np=249893 qsp=49390 rpq=42 cbr=0 cng=72 gpc=38400 gps=17102 nf=41 nn=144888
rcu_bh:
  0 np=146741 qsp=1419 rpq=6 cbr=0 cng=6 gpc=0 gps=0 nf=2 nn=145314
  1 np=155792 qsp=12597 rpq=3 cbr=0 cng=0 gpc=4 gps=8 nf=3 nn=143180
  2 np=136629 qsp=18680 rpq=1 cbr=0 cng=0 gpc=7 gps=6 nf=0 nn=117936
  3 np=137723 qsp=2843 rpq=0 cbr=0 cng=0 gpc=10 gps=7 nf=0 nn=134863
  4 np=123110 qsp=12433 rpq=0 cbr=0 cng=0 gpc=4 gps=2 nf=0 nn=110671
  5 np=137456 qsp=4210 rpq=1 cbr=0 cng=0 gpc=6 gps=5 nf=0 nn=133235
  6 np=120834 qsp=9902 rpq=2 cbr=0 cng=0 gpc=6 gps=3 nf=2 nn=110921
  7 np=144888 qsp=26336 rpq=0 cbr=0 cng=0 gpc=8 gps=2 nf=0 nn=118542
```

As always, this is once again split into "rcu_sched" and "rcu_bh" portions, with CONFIG_TREE_PREEMPT_RCU kernels having an additional "rcu_preempt" section. The fields are as follows:

o "np" is the number of times that __rcu_pending() has been invoked for the corresponding flavor of RCU.

o "qsp" is the number of times that the RCU was waiting for a quiescent state from this CPU.

o "rpq" is the number of times that the CPU had passed through a quiescent state, but not yet reported it to RCU.

o "cbr" is the number of times that this CPU had RCU callbacks that had passed through a grace period, and were thus ready to be invoked.

o "cng" is the number of times that this CPU needed another grace period while RCU was idle.

o "gpc" is the number of times that an old grace period had completed, but this CPU was not yet aware of it.

o "gps" is the number of times that a new grace period had started, but this CPU was not yet aware of it.

o "nf" is the number of times that this CPU suspected that the current grace period had run for too long, and thus needed to be forced.

  Please note that "forcing" consists of sending resched IPIs to holdout CPUs. If that CPU really still is in an old RCU read-side critical section, then we really do have to wait for it. The assumption behing "forcing" is that the CPU is not still in an old RCU read-side critical section, but has not yet responded for some other reason.

o "nn" is the number of times that this CPU needed nothing. Alert readers will note that the rcu "nn" number for a given CPU very closely matches the rcu_bh "np" number for that same CPU. This is due to short-circuit evaluation in rcu_pending().