Using RCU's CPU Stall Detector

The CONFIG_RCU_CPU_STALL_DETECTOR kernel config parameter enables
RCU's CPU stall detector, which detects conditions that unduly delay
RCU grace periods. The stall detector's idea of what constitutes
"unduly delayed" is controlled by a set of C preprocessor macros:

RCU_SECONDS_TILL_STALL_CHECK

	This macro defines the period of time that RCU will wait from
	the beginning of a grace period until it issues an RCU CPU
	stall warning. This time period is normally ten seconds.

RCU_SECONDS_TILL_STALL_RECHECK

	This macro defines the period of time that RCU will wait after
	issuing a stall warning until it issues another stall warning
	for the same stall. This time period is normally set to thirty
	seconds.

RCU_STALL_RAT_DELAY

	The CPU stall detector tries to make the offending CPU print its
	own warnings, as this often gives better-quality stack traces.
	However, if the offending CPU does not detect its own stall in
	the number of jiffies specified by RCU_STALL_RAT_DELAY, then
	some other CPU will complain. This delay is normally set to
	two jiffies.

When a CPU detects that it is stalling, it will print a message similar
to the following:

```
INFO: rcu_sched_state detected stall on CPU 5 (t=2500 jiffies)
```

This message indicates that CPU 5 detected that it was causing a stall,
and that the stall was affecting RCU-sched. This message will normally be
followed by a stack dump of the offending CPU. On TREE_RCU kernel builds,
RCU and RCU-sched are implemented by the same underlying mechanism,
while on TREE_PREEMPT_RCU kernel builds, RCU is instead implemented
by rcu_preempt_state.

On the other hand, if the offending CPU fails to print out a stall-warning
message quickly enough, some other CPU will print a message similar to
the following:

```
INFO: rcu_bh_state detected stalls on CPUs/tasks: { 3 5 } (detected by 2, 2502
jiffies)
```

This message indicates that CPU 2 detected that CPUs 3 and 5 were both
causing stalls, and that the stall was affecting RCU-bh. This message
will normally be followed by stack dumps for each CPU. Please note that
TREE_PREEMPT_RCU builds can be stalled by tasks as well as by CPUs,
and that the tasks will be indicated by PID, for example, "P3421".
It is even possible for a rcu_preempt_state stall to be caused by both
CPUs -and- tasks, in which case the offending CPUs and tasks will all
be called out in the list.

Finally, if the grace period ends just as the stall warning starts
printing, there will be a spurious stall-warning message:

```
INFO: rcu_bh_state detected stalls on CPUs/tasks: { } (detected by 4, 2502
jiffies)
```

This is rare, but does happen from time to time in real life.

So your kernel printed an RCU CPU stall warning. The next question is
"What caused it?" The following problems can result in RCU CPU stall
warnings:

- A CPU looping in an RCU read-side critical section.

- A CPU looping with interrupts disabled. This condition can
result in RCU-sched and RCU-bh stalls.

- A CPU looping with preemption disabled. This condition can
result in RCU-sched stalls and, if ksoftirqd is in use, RCU-bh
stalls.

- A CPU looping with bottom halves disabled. This condition can
result in RCU-sched and RCU-bh stalls.

- For !CONFIG_PREEMPT kernels, a CPU looping anywhere in the kernel
without invoking schedule().

- A bug in the RCU implementation.

- A hardware failure. This is quite unlikely, but has occurred
at least once in real life. A CPU failed in a running system,
becoming unresponsive, but not causing an immediate crash.
This resulted in a series of RCU CPU stall warnings, eventually
leading the realization that the CPU had failed.

The RCU, RCU-sched, and RCU-bh implementations have CPU stall
warning. SRCU does not have its own CPU stall warnings, but its
calls to synchronize_sched() will result in RCU-sched detecting
RCU-sched-related CPU stalls. Please note that RCU only detects
CPU stalls when there is a grace period in progress. No grace period,
no CPU stall warnings.

To diagnose the cause of the stall, inspect the stack traces.
The offending function will usually be near the top of the stack.
If you have a series of stall warnings from a single extended stall,
comparing the stack traces can often help determine where the stall
is occurring, which will usually be in the function nearest the top of
that portion of the stack which remains the same from trace to trace.
If you can reliably trigger the stall, ftrace can be quite helpful.

RCU bugs can often be debugged with the help of CONFIG_RCU_TRACE.