

## Re: [RFC PATCH -rt 2/2] RCU priority boosting additions to rcutorture

---

*Source:* <http://linux.derkeiler.com/Mailing-Lists/Kernel/2007-01/msg06593.html>

---

- *From:* Josh Triplett <josh@xxxxxxxxxx>
  - *Date:* Thu, 25 Jan 2007 11:06:35 -0800
- 

Paul E. McKenney wrote:

On Thu, Jan 25, 2007 at 12:47:04AM -0800, Josh Triplett wrote:

One major item: this new test feature really needs a new module parameter to enable or disable it.

CONFIG\_PREEMPT\_RCU\_BOOST is the parameter -- if not set, then no test. This parameter is provided by the accompanying RCU-boost patch.

It seems useful for rcutorture to use or not use the preempting thread independently of CONFIG\_PREEMPT\_RCU\_BOOST. That would bring you from two cases to four, and the two new cases both make sense:

\* CONFIG\_PREEMPT\_RCU\_BOOST=n, but run rcutorture with the preempting thread. This configuration allows you to demonstrate the need for CONFIG\_PREEMPT\_RCU\_BOOST, by showing what happens when you need it and don't have it.

\* CONFIG\_PREEMPT\_RCU\_BOOST=y, but run rcutorture without the preempting thread. This configuration allows you to test with rcutorture while running a \*real\* real-time workload rather than the simple preempting thread, or just test basic RCU functionality.

A simple boolean module\_param would work here.

At some point, we may want to add the ability to run multiple preempting threads, but that doesn't need to happen for this patch.

Paul E. McKenney wrote:

```
diff -urpNa -X dontdiff
linux-2.6.20-rc4-rt1/kernel/rcutorture.c
linux-2.6.20-rc4-rt1-rcubtorture/kernel/rcutorture.c
---- linux-2.6.20-rc4-rt1/kernel/rcutorture.c 2007-01-09
```

Re: [RFC PATCH -rt 2/2] RCU priority boosting additions to rcutorture

```
10:59:54.000000000 -0800
+++ linux-2.6.20-rc4-rt1-rcubtorture/kernel/rcutorture.c
2007-01-23 11:27:49.000000000 -0800
```

```
+static int rcu_torture_preempt(void *arg)
+{
+ int completedstart;
+ time_t gcstart;
+ struct sched_param sp;
+
+ sp.sched_priority = MAX_RT_PRIO - 1;
+ sched_setscheduler(current, SCHED_RR, &sp);
+ current->flags |= PF_NOFREEZE;
+
+ do {
+ completedstart = rcu_torture_completed();
+ gcstart = xtime.tv_sec;
+ while ((xtime.tv_sec - gcstart < 10) &&
+ (rcu_torture_completed() == completedstart))
+ cond_resched();
+ if (rcu_torture_completed() == completedstart)
+ rcu_torture_preempt_errors++;
+ schedule_timeout_interruptible(shuffle_interval * HZ);
```

Why call `schedule_timeout_interruptible` here without actually handling interruptions? So that you can send it a signal to cause the shuffle early?

It allows you to kill the process in order to get the module unload to happen more quickly in case someone specified an overly long interval.

I didn't actually know that you could kill a kthread from userspace. :)

That rationale makes sense.

But now that you mention this, a simple one-second sleep is probably appropriate here.

OK.

```
+ } while (!kthread_should_stop());
+ return NULL;
+}
+
+static void rcu_preempt_start(void)
```

Re: [RFC PATCH -rt 2/2] RCU priority boosting additions to rcutorture

Re: [RFC PATCH -rt 2/2] RCU priority boosting additions to rcutorture

```
+{
+ rcu_preeempt_task = kthread_run(rcu_torture_preeempt,
NULL,
+ "rcu_torture_preeempt");
+ if (IS_ERR(rcu_preeempt_task)) {
+ VERBOSE_PRINTK_ERRSTRING("Failed to create
preeempter");
```

This ought to include the errno value, PTR\_ERR(rcu\_preeempt\_task).

Good point — what I should do is return this value so that rcu\_torture\_init() can return it, failing the module-load process and unwinding.

Even better, yes.

```
+ rcu_preeempt_task = NULL;
+ }
+}
+
+static void rcu_preeempt_end(void)
+{
+ if (rcu_preeempt_task != NULL) {
```

if (rcu\_preeempt\_task) would work just as well here.

True, but was being consistent with usage elsewhere in this file.

Fair enough; don't worry about it for this patch, then. I'll deal with that particular style cleanup later, throughout rcutorture.

```
static struct rcu_torture_ops rcu_ops = {
.init = NULL,
.cleanup = NULL,
@@ -267,7 +334,9 @@ static struct rcu_torture_ops
rcu_ops =
.completed = rcu_torture_completed,
.deferredfree = rcu_torture_deferred_free,
.sync = synchronize_rcu,
- .stats = NULL,
+ .preeemptstart = rcu_preeempt_start,
+ .preeemptend = rcu_preeempt_end,
+ .stats = rcu_preeempt_stats,
.name = "rcu"
};
```

Re: [RFC PATCH -rt 2/2] RCU priority boosting additions to rcutorture

```
@@ -306,6 +375,8 @@ static struct rcu_torture_ops
rcu_sync_o
.completed = rcu_torture_completed,
.deferredfree = rcu_sync_torture_deferred_free,
.sync = synchronize_rcu,
+ .preemptstart = NULL,
+ .preemptend = NULL,
.stats = NULL,
.name = "rcu_sync"
};
```

Much like other common structures such as struct file\_operations, no need to explicitly specify members as NULL here; any member you don't specify will get a NULL value. That avoids the need to update every use of this structure whenever you add a new member used by only some of them.

Untrusting, aren't I? ;-)

Heh. I have that problem as well; I always hesitate to trust the compiler to initialize values.

I removed all the "= NULL" entries.

Thanks.

```
@@ -856,6 +935,8 @@ rcu_torture_cleanup(void)
kthread_stop(stats_task);
}
stats_task = NULL;
+ if (cur_ops->preemptend != NULL)
```

if (cur\_ops->preemptend) would work as well.

True, though again there is a lot of existing "!= NULL" in this file and elsewhere. Many thousands of them through the kernel. ;-)

As before, don't worry about it for this patch then.

I will run this through the mill and repost.

Re: [RFC PATCH -rt 2/2] RCU priority boosting additions to rcutorture

Re: [RFC PATCH -rt 2/2] RCU priority boosting additions to rcutorture

Thanks!

– Josh Triplett

–

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in the body of a message to majordomo@xxxxxxxxxxxxxxxx

More majordomo info at <http://vger.kernel.org/majordomo-info.html>

Please read the FAQ at <http://www.tux.org/lkml/>