[PATCH v2] hardlockup: detect hard lockups without NMIs using secondary cpus

Colin Cross ccross at android.com
Fri Jan 11 16:51:48 EST 2013

- Previous message: [PATCH] KVM: ARM: Change psci_call return value to bool
- Next message: [PATCH v2] hardlockup: detect hard lockups without NMIs using secondary cpus
- **Messages sorted by:** [date] [thread] [subject] [author]

```
Emulate NMIs on systems where they are not available by using timer
interrupts on other cpus.  Each cpu will use its softlockup hrtimer
to check that the next cpu is processing hrtimer interrupts by
verifying that a counter is increasing.

This patch is useful on systems where the hardlockup detector is not
available due to a lack of NMIs, for example most ARM SoCs.
Without this patch any cpu stuck with interrupts disabled can
cause a hardware watchdog reset with no debugging information,
but with this patch the kernel can detect the lockup and panic,
which can result in useful debugging info.

Signed-off-by: Colin Cross <ccross at android.com>
---
 include/linux/nmi.h |    5 ++-
 kernel/watchdog.c   |  123 ++++++++++++++++++++++++++++++++++++++++++++++++++--
 lib/Kconfig.debug   |   14 +++++-
 3 files changed, 135 insertions(+), 7 deletions(-)

Changes since v1:
    renamed variables to clarify when referring to the next cpu
    factored out function to get next cpu
    fixed int vs unsigned int mismatches
    fixed race condition when offlining cpus

diff --git a/include/linux/nmi.h b/include/linux/nmi.h
index db50840..c8f8aa0 100644
--- a/include/linux/nmi.h
+++ b/include/linux/nmi.h
@@ -14,8 +14,11 @@
  * may be used to reset the timeout - for code which intentionally
  * disables interrupts for a long time. This call is stateless.
  */
-#if defined(CONFIG_HAVE_NMI_WATCHDOG) || defined(CONFIG_HARDLOCKUP_DETECTOR)
+#if defined(CONFIG_HAVE_NMI_WATCHDOG) || defined(CONFIG_HARDLOCKUP_DETECTOR_NMI)
 #include <asm/nmi.h>
+#endif
+
+#if defined(CONFIG_HAVE_NMI_WATCHDOG) || defined(CONFIG_HARDLOCKUP_DETECTOR)
 extern void touch_nmi_watchdog(void);
 #else
 static inline void touch_nmi_watchdog(void)
diff --git a/kernel/watchdog.c b/kernel/watchdog.c
index 75a2ab3..61a0595 100644
--- a/kernel/watchdog.c
+++ b/kernel/watchdog.c
```

```
@@ -44,6 +44,11 @@
 static DEFINE_PER_CPU(bool, hard_watchdog_warn);
 static DEFINE_PER_CPU(bool, watchdog_nmi_touch);
 static DEFINE_PER_CPU(unsigned long, hrtimer_interrupts_saved);
+#endif
+#ifdef CONFIG_HARDLOCKUP_DETECTOR_OTHER_CPU
+static cpumask_t __read_mostly watchdog_cpus;
+#endif
+#ifdef CONFIG_HARDLOCKUP_DETECTOR_NMI
 static DEFINE_PER_CPU(struct perf_event *, watchdog_ev);
 #endif

@@ -179,7 +184,7 @@ void touch_softlockup_watchdog_sync(void)
        __raw_get_cpu_var(watchdog_touch_ts) = 0;
 }

-#ifdef CONFIG_HARDLOCKUP_DETECTOR
+#ifdef CONFIG_HARDLOCKUP_DETECTOR_NMI
 /* watchdog detector functions */
 static int is_hardlockup(void)
 {
@@ -193,6 +198,76 @@ static int is_hardlockup(void)
 }
 #endif

+#ifdef CONFIG_HARDLOCKUP_DETECTOR_OTHER_CPU
+static unsigned int watchdog_next_cpu(unsigned int cpu)
+{
+       cpumask_t cpus = watchdog_cpus;
+       unsigned int next_cpu;
+
+       next_cpu = cpumask_next(cpu, &cpus);
+       if (next_cpu >= nr_cpu_ids)
+               next_cpu = cpumask_first(&cpus);
+
+       if (next_cpu == cpu)
+               return nr_cpu_ids;
+
+       return next_cpu;
+}
+
+static int is_hardlockup_other_cpu(unsigned int cpu)
+{
+       unsigned long hrint = per_cpu(hrtimer_interrupts, cpu);
+
+       if (per_cpu(hrtimer_interrupts_saved, cpu) == hrint)
+               return 1;
+
+       per_cpu(hrtimer_interrupts_saved, cpu) = hrint;
+       return 0;
+}
+
+static void watchdog_check_hardlockup_other_cpu(void)
+{
+       unsigned int next_cpu;
+
+       /*
+        * Test for hardlockups every 3 samples.  The sample period is
+        *  watchdog_thresh * 2 / 5, so 3 samples gets us back to slightly over
+        *  watchdog_thresh (over by 20%).
+        */
+       if (__this_cpu_read(hrtimer_interrupts) % 3 != 0)
+                return;
+
+        /* check for a hardlockup on the next cpu */
+        next_cpu = watchdog_next_cpu(smp_processor_id());
+        if (next_cpu >= nr_cpu_ids)
+                return;
+
+        smp_rmb();
+
+        if (per_cpu(watchdog_nmi_touch, next_cpu) == true) {
+                per_cpu(watchdog_nmi_touch, next_cpu) = false;
+                return;
+        }
+
+        if (is_hardlockup_other_cpu(next_cpu)) {
+                /* only warn once */
+                if (per_cpu(hard_watchdog_warn, next_cpu) == true)
+                        return;
+
+                if (hardlockup_panic)
+                        panic("Watchdog detected hard LOCKUP on cpu %u", next_cpu);
+                else
+                        WARN(1, "Watchdog detected hard LOCKUP on cpu %u", next_cpu);
+
+                per_cpu(hard_watchdog_warn, next_cpu) = true;
+        } else {
+                per_cpu(hard_watchdog_warn, next_cpu) = false;
+        }
+}
+#else
+static inline void watchdog_check_hardlockup_other_cpu(void) { return; }
+#endif
+
 static int is_softlockup(unsigned long touch_ts)
 {
        unsigned long now = get_timestamp(smp_processor_id());
@@ -204,7 +279,7 @@ static int is_softlockup(unsigned long touch_ts)
        return 0;
 }

-#ifdef CONFIG_HARDLOCKUP_DETECTOR
+#ifdef CONFIG_HARDLOCKUP_DETECTOR_NMI

 static struct perf_event_attr wd_hw_attr = {
        .type           = PERF_TYPE_HARDWARE,
@@ -252,7 +327,7 @@ static void watchdog_overflow_callback(struct perf_event *event,
        __this_cpu_write(hard_watchdog_warn, false);
        return;
 }
-#endif /* CONFIG_HARDLOCKUP_DETECTOR */
+#endif /* CONFIG_HARDLOCKUP_DETECTOR_NMI */

 static void watchdog_interrupt_count(void)
 {
@@ -272,6 +347,9 @@ static enum hrtimer_restart watchdog_timer_fn(struct hrtimer *hrtimer)
        /* kick the hardlockup detector */
        watchdog_interrupt_count();

+        /* test for hardlockups on the next cpu */
+        watchdog_check_hardlockup_other_cpu();
+
        /* kick the softlockup detector */
        wake_up_process(__this_cpu_read(softlockup_watchdog));

@@ -396,7 +474,7 @@ static void watchdog(unsigned int cpu)
        __touch_watchdog();
 }

-#ifdef CONFIG_HARDLOCKUP_DETECTOR
+#ifdef CONFIG_HARDLOCKUP_DETECTOR_NMI
 /*
  * People like the simple clean cpu node info on boot.
  * Reduce the watchdog noise by only printing messages
@@ -472,9 +550,44 @@ static void watchdog_nmi_disable(unsigned int cpu)
        return;
 }
 #else
+#ifdef CONFIG_HARDLOCKUP_DETECTOR_OTHER_CPU
+static int watchdog_nmi_enable(unsigned int cpu)
+{
+       /*
+        * The new cpu will be marked online before the first hrtimer interrupt
+        * runs on it.  If another cpu tests for a hardlockup on the new cpu
+        * before it has run its first hrtimer, it will get a false positive.
+        * Touch the watchdog on the new cpu to delay the first check for at
+        * least 3 sampling periods to guarantee one hrtimer has run on the new
+        * cpu.
+        */
+       per_cpu(watchdog_nmi_touch, cpu) = true;
+       smp_wmb();
+       cpumask_set_cpu(cpu, &watchdog_cpus);
+       return 0;
+}
+
+static void watchdog_nmi_disable(unsigned int cpu)
+{
+       unsigned int next_cpu = watchdog_next_cpu(cpu);
+
+       /*
+        * Offlining this cpu will cause the cpu before this one to start
+        * checking the one after this one.  If this cpu just finished checking
+        * the next cpu and updating hrtimer_interrupts_saved, and then the
+        * previous cpu checks it within one sample period, it will trigger a
+        * false positive.  Touch the watchdog on the next cpu to prevent it.
+        */
+       if (next_cpu < nr_cpu_ids)
+               per_cpu(watchdog_nmi_touch, next_cpu) = true;
+       smp_wmb();
+       cpumask_clear_cpu(cpu, &watchdog_cpus);
+}
+#else
 static int watchdog_nmi_enable(unsigned int cpu) { return 0; }
 static void watchdog_nmi_disable(unsigned int cpu) { return; }
-#endif /* CONFIG_HARDLOCKUP_DETECTOR */
+#endif /* CONFIG_HARDLOCKUP_DETECTOR_OTHER_CPU */
+#endif /* CONFIG_HARDLOCKUP_DETECTOR_NMI */

 /* prepare/enable/disable routines */
 /* sysctl functions */
diff --git a/lib/Kconfig.debug b/lib/Kconfig.debug
index aaf8baf..f7c4859 100644
--- a/lib/Kconfig.debug
+++ b/lib/Kconfig.debug
@@ -191,15 +191,27 @@ config LOCKUP_DETECTOR
```

```
          The overhead should be minimal.  A periodic hrtimer runs to
          generate interrupts and kick the watchdog task every 4 seconds.
          An NMI is generated every 10 seconds or so to check for hardlockups.
+         If NMIs are not available on the platform, every 12 seconds the
+         hrtimer interrupt on one cpu will be used to check for hardlockups
+         on the next cpu.

          The frequency of hrtimer and NMI events and the soft and hard lockup
          thresholds can be controlled through the sysctl watchdog_thresh.

-config HARDLOCKUP_DETECTOR
+config HARDLOCKUP_DETECTOR_NMI
        def_bool y
        depends on LOCKUP_DETECTOR && !HAVE_NMI_WATCHDOG
        depends on PERF_EVENTS && HAVE_PERF_EVENTS_NMI

+config HARDLOCKUP_DETECTOR_OTHER_CPU
+       def_bool y
+       depends on LOCKUP_DETECTOR && SMP
+       depends on !HARDLOCKUP_DETECTOR_NMI && !HAVE_NMI_WATCHDOG
+
+config HARDLOCKUP_DETECTOR
+       def_bool y
+       depends on HARDLOCKUP_DETECTOR_NMI || HARDLOCKUP_DETECTOR_OTHER_CPU
+
 config BOOTPARAM_HARDLOCKUP_PANIC
        bool "Panic (Reboot) On Hard Lockups"
        depends on HARDLOCKUP_DETECTOR
--
1.7.7.3
```

- Previous message: [PATCH] KVM: ARM: Change psci_call return value to bool
- Next message: [PATCH v2] hardlockup: detect hard lockups without NMIs using secondary cpus
- **Messages sorted by:** [date] [thread] [subject] [author]

More information about the linux-arm-kernel mailing list