

[PATCH] 2.6, 2.4, Nforce2, Experimental idle halt workaround instead of apic ack delay.

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2004-02/2658.html>

From: Ross Dickson (ross_at_datcreative.com.au)

Date: 02/11/04

To: linux-kernel@vger.kernel.org, Jamie Lokier <jamie@shareable.org>

Date: Thu, 12 Feb 2004 01:22:06 +1000

Greetings,

Inspired by Jamie Lokier posting re halt and idle task here is the result of my attempts to move the hard lockup compensation from the "wake up from C1 disconnect" to the "going into C1 disconnect".

I surmise that between the Athlon and Nforce2 there is an issue surrounding bus drive or pll clock recovery ...etc.... on C1 disconnect and reconnect. Not sure how much of each is responsible for the problems. I have no detailed docs or test gear to dive in and get to the bottom of it. Given the lockups appear in multitude when you change the timer tick rate on a 2.4 kernel from 100Hz to 1000Hz then I think it is reasonable to assume some part does not like doing C1 disconnect cycles that frequently. Also given that a small delay put into the apic timer interrupt routine that typically wakes up the cpu from disconnect can give stability, then we appear to have a second issue of minimum cycle time for back to back C1 disconnect –reconnect cycles.

Approaching from this perspective the following patch implements a new idle thread. One which does not go into C1 disconnect (hlt) if less than 1.6% of the apic timer interval is left to execute. When you think about it, why do we disconnect if we are about to reconnect? It also has a small timing delay to help with back to back disconnect cycles (SMI might put us into one?). The result should be a slightly faster system (then with my apic ack delay patch) when busy but still with disconnect functioning to save power and lower heat with typical loads.

This is EXPERIMENTAL – I have only tested it for a few hours on my EPOX8RGA+ system. At this point it is kernel arg driven although in the future it could be a pci quirk. If it works well then we would ditch the apic ack delay patch, i.e use these mods of process.c in place of my earlier mods to apic.c. For testing this you do not have to back out of the apic.c changes, just drop the "apic_tack=x" kernel arg during the test boots and replace it with the following.

The KERNEL ARG to invoke it is "idle=C1halt".

Linux-Kernel: [PATCH] 2.6, 2.4, Nforce2, Experimental idle halt workaround instead of apic ack delay.

Please share how/if it works..... -performance -cpu temperature -side effects?
Files are also in attached tarball if whitespace problems.

Finally, this is separate to my "io-apic edge for nmi debug" patch. You still need that patch for "nmi_debug=1". Some have reported time of day clock skew when routing the 8254 timer through the io-apic with that patch. My system has no skew problems with that patch and a modified 2.4.24 kernel.

A correct fix has since been authored by Maciej Rozycki and included with the 2.6.3-rc1-mm1 kernel tree. It does not give functioning "nmi_debug=1" with my system as it correctly attempts to route the 8254 timer to where the bios instructs, fails because the bios claims io-apic pin 2 instead of 0 and then falls back to a valid virtual wire mode. I have not yet attempted to combine my io-apic patch (io-apic pin 2 fails so try pin 0) with these later fixes - nor do I know if I should because have no nforce2 tech docs.

Jesse Allen's posting on that patch and his Nforce2
<http://linux.derkeiler.com/Mailing-Lists/Kernel/2004-02/2544.html>

Regards
Ross Dickson

patch for 2.6.3-rc1-mm1

```
---CUT HERE---
--- linux-2.6.3-rc1-mm1/arch/i386/kernel/process.c 2004-02-11 21:29:49.000000000 +1000
+++ linux-2.6.3-rc1-mm1-nf/arch/i386/kernel/process.c 2004-02-11 23:44:04.000000000 +1000
@@ -50,10 +50,11 @@
#include <asm/desc.h>
#include <asm/atomic_kmap.h>
#ifdef CONFIG_MATH_EMULATION
#include <asm/math_emu.h>
#endif
+#include <asm/apic.h>

#include <linux/irq.h>
#include <linux/err.h>

#include <asm/tlbflush.h>
@@ -92,11 +93,11 @@ EXPORT_SYMBOL(enable_hlt);

/*
 * We use this if we don't have any better
 * idle routine..
 */
-void default_idle(void)
+static void default_idle(void)
{
    if (!hlt_counter && current_cpu_data.hlt_works_ok) {
        local_irq_disable();
        if (!need_resched())
```

[PATCH] 2.6, 2.4, Nforce2, Experimental idle halt workaround instead of apic ack delay.

Linux–Kernel: [PATCH] 2.6, 2.4, Nforce2, Experimental idle halt workaround instead of apic ack delay.

```
        safe_halt();
@@ -104,10 +105,29 @@ void default_idle(void)
        local_irq_enable();
    }
}

/*
+ * We use this to avoid nforce2 lockups
+ * Reduces frequency of C1 disconnects
+ */
+static void c1halt_idle(void)
+{
+ if (!hlt_counter && current_cpu_data.hlt_works_ok) {
+ local_irq_disable();
+ /* only hlt disconnect if more than 1.6% of apic interval remains */
+ if( (!need_resched) &&
+ (apic_read(APIC_TMCCT) > (apic_read(APIC_TMICT)>>6))) {
+ ndelay(500); /* helps nforce2 but adds 0.5us hard int latency */
+ safe_halt(); /* nothing better to do until we wake up */
+ }
+ else
+ local_irq_enable();
+ }
+ }
+
+/*
+ * On SMP it's slightly faster (but much more power–consuming!)
+ * to poll the →work.need_resched flag instead of waiting for the
+ * cross–CPU IPI to arrive. Use this option with caution.
+ */
static void poll_idle (void)
@@ -195,20 +215,18 @@ static inline void check_cpu_quiescent(v
+ * The idle thread. There's no useful work to be
+ * done, so just try to conserve power and have a
+ * low exit latency (ie sit in a loop waiting for
+ * somebody to say that they'd like to reschedule)
+ */
+static void (*idle)(void);
void cpu_idle (void)
{
    /* endless idle loop with no priority at all */
    while (1) {
        while (!need_resched()) {
– void (*idle)(void) = pm_idle;
–
        if (!idle)
– idle = default_idle;
–
+ idle = pm_idle ? pm_idle : default_idle;
        check_cpu_quiescent();
        irq_stat[smp_processor_id()].idle_timestamp = jiffies;
    }
}
```

[PATCH] 2.6, 2.4, Nforce2, Experimental idle halt workaround instead of apic ack delay.

Linux-Kernel: [PATCH] 2.6, 2.4, Nforce2, Experimental idle halt workaround instead of apic ack delay.

```
        idle();
    }
    schedule();
@@ -260,16 +278,18 @@ void __init select_idle_routine(const st

static int __init idle_setup (char *str)
{
    if (!strcmp(str, "poll", 4)) {
        printk("using polling idle threads.\n");
- pm_idle = poll_idle;
+ idle = poll_idle;
    } else if (!strcmp(str, "halt", 4)) {
        printk("using halt in idle threads.\n");
- pm_idle = default_idle;
+ idle = default_idle;
    } else if (!strcmp(str, "C1halt", 6)) {
+ printk("using C1 halt disconnect friendly idle threads.\n");
+ idle = c1halt_idle;
    }
-
    return 1;
}

__setup("idle=", idle_setup);
---CUT HERE---
```

patch for 2.4.24:

```
---CUT HERE---
--- linux-2.4.24/arch/i386/kernel/process.c 2003-12-01 15:10:12.000000000 +1000
+++ linux-2.4.24-nf/arch/i386/kernel/process.c 2004-02-11 22:43:27.000000000 +1000
@@ -78,20 +78,38 @@ void enable_hlt(void)

/*
 * We use this if we don't have any better
 * idle routine..
 */
-void default_idle(void)
+static void default_idle(void)
{
    if (current_cpu_data.hlt_works_ok && !hlt_counter) {
        __cli();
- if (!current->need_resched)
+ if(!current->need_resched)
            safe_halt();
        else
            __sti();
    }
}
+/*
+ * We use this to avoid nforce2 lockups
+ * Reduces frequency of C1 disconnects
```

[PATCH] 2.6, 2.4, Nforce2, Experimental idle halt workaround instead of apic ack delay.

Linux–Kernel: [PATCH] 2.6, 2.4, Nforce2, Experimental idle halt workaround instead of apic ack delay.

```
+ */
+static void c1halt_idle(void)
+{
+ if (current_cpu_data.hlt_works_ok && !hlt_counter) {
+ __cli();
+ /* only hlt disconnect if more than 1.6% of apic interval remains */
+ if( (!current->need_resched) &&
+ (apic_read(APIC_TMCCT) > (apic_read(APIC_TMICT)>>6))) {
+ ndelay(500); /* helps nforce2 but adds 0.5us hard int latency */
+ safe_halt(); /* nothing better to do until we wake up */
+ }
+ else
+ __sti();
+ }
+}

/*
 * On SMP it's slightly faster (but much more power-consuming!)
 * to poll the ->need_resched flag instead of waiting for the
 * cross-CPU IPI to arrive. Use this option with caution.
@@ -121,21 +139,22 @@ static void poll_idle (void)
 * The idle thread. There's no useful work to be
 * done, so just try to conserve power and have a
 * low exit latency (ie sit in a loop waiting for
 * somebody to say that they'd like to reschedule)
 */
+static void (*idle)(void);
+
+void cpu_idle (void)
+{
+ /* endless idle loop with no priority at all */
+ init_idle();
+ current->nice = 20;
+ current->counter = -100;
+
+ while (1) {
- void (*idle)(void) = pm_idle;
+ if (!idle)
- idle = default_idle;
+ idle = pm_idle ? pm_idle : default_idle;
+ while (!current->need_resched)
+ idle();
+ schedule();
+ check_pgt_cache();
+ }
@@ -143,13 +162,15 @@ void cpu_idle (void)

static int __init idle_setup (char *str)
{
+ if (!strncmp(str, "poll", 4)) {
+ printk("using polling idle threads.\n");

```

[PATCH] 2.6, 2.4, Nforce2, Experimental idle halt workaround instead of apic ack delay.

Linux-Kernel: [PATCH] 2.6, 2.4, Nforce2, Experimental idle halt workaround instead of apic ack delay.

```
- pm_idle = poll_idle;
+ idle = poll_idle;
+ } else if (!strcmp(str, "C1halt", 6)) {
+ printk("using C1 halt disconnect friendly idle threads.\n");
+ idle = c1halt_idle;
+ }
-
+ return 1;
+ }

__setup("idle=", idle_setup);
---CUT HERE---
```

-
To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
the body of a message to majordomo@vger.kernel.org
More majordomo info at <http://vger.kernel.org/majordomo-info.html>
Please read the FAQ at <http://www.tux.org/lkml/>

- application/x-tgz attachment: [nforce2-idle.tgz](#)