

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

*Source:* <http://linux.derkeiler.com/Mailing-Lists/Kernel/2004-02/6520.html>

---

*From:* Ross Dickson ([ross\\_at\\_datcreative.com.au](mailto:ross_at_datcreative.com.au))

*Date:* 02/25/04

To: "Prakash K. Cheemplavam" <[PrakashKC@gmx.de](mailto:PrakashKC@gmx.de)>

Date: Wed, 25 Feb 2004 22:38:55 +1000

On Monday 23 February 2004 11:37, Prakash K. Cheemplavam wrote:

> *Oh before I forget, I had to resolve a reject by hand, but I \*think\* I*  
> *did it right. (And yes, I used your corrected version of the patch.)*  
> *Well, maybe you take a look over the patch and rediff. :-)*  
>  
> *Prakash*  
>  
>  
>

Hi Prakash,

Patches attached rediffed for 2.6.3 and 2.6.3-mm3.

Late comers see start of this thread for more detail.

Also Arjen's page at <http://atlas.et.tudelft.nl/verwei90/nforce2/index.html>

Note that the ioapic patch is not essential for 2.6.3-mm3, only if you want to use `nmi_debug=1`.

2.6.3-mm3 sets up 8254 timer OK in a virtual wire mode without this patch.

It will be interesting to see if there is any clock skew associated with these patches, I get no skew on 2.4.24.

The kernel arg to invoke the idle halt workaround is still "idle=C1halt".

I have increased slightly the `ndelay` time from 500ns to 600ns – I had some lockups on startup and shutdown with certain removable IDE drives, this seemed to help. Feel free to adjust the value if you want to.

Also if you want snappier performance you can try different values instead of 6 in this line:

```
+ (apic_read(APIC_TMCCT) > (apic_read(APIC_TMICT)>>6))) {
```

The 6 yields a divide by  $2^6 = 64$  for the 1.6% ( $1/64 = 1.6\%$ )

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

If you try say 5 for  $2^5 = 32$  for 3.1% or 4 for 6% or 3 for 12.5% then if less than this percentage of time remains in the slice the system won't go into hlt disconnect thus avoiding the reconnect – recovery time associated with that cycle that would have been.

I have found a need for less ndelay time with the more the percentage (lower number)– there appears to be a tradeoff between the two.

3 for 12.5% or 2 for 25% might even work with no ndelay time i.e. the ndelay line removed or commented from the code, for some.

Of course expect a bit more heat under moderate loads with an increase in the percentage but some may prefer it for their application.

Happy hacking,  
Ross.

For 2.6.3 ioapic

---CUT HERE---

```
--- linux-2.6.3/arch/i386/kernel/io_apic.c.original 2004-02-18 13:57:22.000000000 +1000
+++ linux-2.6.3/arch/i386/kernel/io_apic.c 2004-02-24 11:57:04.000000000 +1000
@@ -2188,12 +2188,56 @@ static inline void check_timer(void)
        check_nmi_watchdog();
    }
    return;
}
clear_IO_APIC_pin(0, pin1);
- printk(KERN_ERR "..MP-BIOS bug: 8254 timer not connected to IO-APIC\n");
+ printk(KERN_ERR "..MP-BIOS bug: 8254 timer not connected to IO-APIC INTIN%d\n",pin1);
+ }
+
+
+ #if defined(CONFIG_ACPI_BOOT) && defined(CONFIG_X86_UP_IOAPIC)
+ /* for nforce2 try vector 0 on pin0
+ * Note 8259a is already masked, also by default
+ * the io_apic_set_pci_routing call disables the 8259 irq 0
+ * so we must be connected directly to the 8254 timer if this works
+ * Note2: this violates the above comment re Subtle but works!
+ */
+ printk(KERN_INFO "..TIMER: Is timer irq0 connected to IO-APIC INTIN0? ...n");
+ if (pin1 != -1) {
+ extern spinlock_t i8259A_lock;
+ unsigned long flags;
+ int tok, saved_timer_ack = timer_ack;
+ /*
+ * Ok, does IRQ0 through the IOAPIC work?
+ */
+ io_apic_set_pci_routing ( 0, 0, 0, 0, 0); /* connect pin */
+ unmask_IO_APIC_irq(0);
+ timer_ack = 0;
+
+
+ /*
```

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

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

```
+ * Ok, does IRQ0 through the IOAPIC work?
+ */
+ spin_lock_irqsave(&i8259A_lock, flags);
+ Dprintk("..TIMER 8259A ints disabled?, imr1:%02x, imr2:%02x\n", inb(0x21), inb(0xA1));
+ tok = timer_irq_works();
+ spin_unlock_irqrestore(&i8259A_lock, flags);
+ if (tok) {
+ if (nmi_watchdog == NMI_IO_APIC) {
+ disable_8259A_irq(0);
+ setup_nmi();
+ enable_8259A_irq(0);
+ check_nmi_watchdog();
+ }
+ printk(KERN_INFO "..TIMER: works OK on IO-APIC INTIN0 irq0\n" );
+ return;
+ }
+ /* failed */
+ timer_ack = saved_timer_ack;
+ clear_IO_APIC_pin(0, 0);
+ io_apic_set_pci_routing ( 0, pin1, 0, 0, 0);
+ printk(KERN_ERR "..MP-BIOS: 8254 timer not connected to IO-APIC INTIN0\n");
+ }
+ #endif

    printk(KERN_INFO "...trying to set up timer (IRQ0) through the 8259A ... ");
    if (pin2 != -1) {
        printk("\n..... (found pin %d) ...", pin2);
        /*
---CUT HERE---
```

For 2.6.3 idleC1halt

---CUT HERE---

--- linux-2.6.3/arch/i386/kernel/process.c.original 2004-02-18 13:57:11.000000000 +1000

+++ linux-2.6.3/arch/i386/kernel/process.c 2004-02-24 11:25:56.000000000 +1000

@@ -48,10 +48,11 @@

#include <asm/irq.h>

#include <asm/desc.h>

#ifdef CONFIG\_MATH\_EMULATION

#include <asm/math\_emu.h>

#endif

+#include <asm/apic.h>

#include <linux/irq.h>

#include <linux/err.h>

asmlinkage void ret\_from\_fork(void) \_\_asm\_\_("ret\_from\_fork");

@@ -87,11 +88,11 @@ EXPORT\_SYMBOL(enable\_hlt);

/\*

\* We use this if we don't have any better

\* idle routine..

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

```

*/
–void default_idle(void)
+static void default_idle(void)
{
    if (!hlt_counter && current_cpu_data.hlt_works_ok) {
        local_irq_disable();
        if (!need_resched())
            safe_halt();
@@ –99,10 +100,29 @@ 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(600); /* helps nforce2 but adds 0.6us 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)
@@ –136,20 +156,18 @@ 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 */
    while (1) {
        while (!need_resched()) {
– void (*idle)(void) = pm_idle;

```

```

-
-         if (!idle)
- idle = default_idle;
-
+ idle = pm_idle ? pm_idle : default_idle;
+         irq_stat[smp_processor_id()].idle_timestamp = jiffies;
+         idle();
+     }
+     schedule();
+ }
@@ -200,16 +218,18 @@ void __init select_idle_routine(const st

```

```

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

```

\_\_setup("idle=", idle\_setup);

---CUT HERE---

For 2.6.3-mm3 ioapic

---CUT HERE---

```

--- linux-2.6.3-mm3/arch/i386/kernel/io_apic.c.original 2004-02-24 12:26:32.000000000 +1000
+++ linux-2.6.3-mm3/arch/i386/kernel/io_apic.c 2004-02-24 16:40:56.000000000 +1000
@@ -2206,12 +2206,54 @@ static inline void check_timer(void)
+         timer_ack = !cpu_has_tsc;
+     }
+     return;
+ }
+     clear_IO_APIC_pin(0, pin1);
- printk(KERN_ERR "..MP-BIOS bug: 8254 timer not connected to IO-APIC\n");
+ printk(KERN_ERR "..MP-BIOS bug: 8254 timer not connected to IO-APIC INTIN%d\n",pin1);
+ }
+
+
+ #if defined(CONFIG_ACPI_BOOT) && defined(CONFIG_X86_UP_IOAPIC)
+ /* for nforce2 try vector 0 on pin0
+ * Note 8259a is already masked, also by default

```

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

```
+ * the io_apic_set_pci_routing call disables the 8259 irq 0
+ * so we must be connected directly to the 8254 timer if this works
+ */
+ printk(KERN_INFO "..TIMER: Is timer irq0 connected to IO-APIC INTIN0? ...\n");
+ if (pin1 != -1) {
+ extern spinlock_t i8259A_lock;
+ unsigned long flags;
+ int tok;
+ /*
+ * Ok, does IRQ0 through the IOAPIC work?
+ */
+ io_apic_set_pci_routing ( 0, 0, 0, 0, 0); /* connect pin */
+ unmask_IO_APIC_irq(0);
+
+ /*
+ * Ok, does IRQ0 through the IOAPIC work?
+ */
+ spin_lock_irqsave(&i8259A_lock, flags);
+ Dprintf("..TIMER 8259A ints disabled?, imr1:%02x, imr2:%02x\n", inb(0x21), inb(0xA1));
+ tok = timer_irq_works();
+ spin_unlock_irqrestore(&i8259A_lock, flags);
+ if (tok) {
+ if (nmi_watchdog == NMI_IO_APIC) {
+ disable_8259A_irq(0);
+ setup_nmi();
+ enable_8259A_irq(0);
+ if (check_nmi_watchdog() < 0);
+ timer_ack = !cpu_has_tsc;
+ }
+ printk(KERN_INFO "..TIMER: works OK on IO-APIC INTIN0 irq0\n" );
+ return;
+ }
+ /* failed */
+ clear_IO_APIC_pin(0, 0);
+ io_apic_set_pci_routing ( 0, pin1, 0, 0, 0);
+ printk(KERN_ERR "..MP-BIOS: 8254 timer not connected to IO-APIC INTIN0\n");
+ }
+}endif
```

```
printk(KERN_INFO "...trying to set up timer (IRQ0) through the 8259A ... ");
if (pin2 != -1) {
    printk("\n..... (found pin %d) ...", pin2);
    /*
```

---CUT HERE---

For 2.6.3-mm3 idleC1halt

---CUT HERE---

```
--- linux-2.6.3-mm3/arch/i386/kernel/process.c.original 2004-02-24 12:26:32.000000000 +1000
+++ linux-2.6.3-mm3/arch/i386/kernel/process.c 2004-02-24 15:54:26.000000000 +1000
@@ -49,10 +49,11 @@
#include <asm/desc.h>
```

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

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

```
#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>

asmlinkage void ret_from_fork(void) __asm__("ret_from_fork");
@@ -88,11 +89,11 @@ 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())
            safe_halt();
@@ -100,10 +101,29 @@ 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(600); /* helps nforce2 but adds 0.6us 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.
```

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

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

```
*/
static void poll_idle (void)
@@ -137,20 +157,18 @@ 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 */
    while (1) {
        while (!need_resched()) {
- void (*idle)(void) = pm_idle;
-
            if (!idle)
- idle = default_idle;
-
+ idle = pm_idle ? pm_idle : default_idle;
            irq_stat[smp_processor_id()].idle_timestamp = jiffies;
            idle();
        }
        schedule();
    }
@@ -201,16 +219,18 @@ void __init select_idle_routine(const st

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

__setup("idle=", idle_setup);

---CUT HERE---
```

Attached patches also in tarballs if whitespace problems.

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

–

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-lockup-patches-rd-2.6.3.tgz](#)
- 

- application/x-tgz attachment: [nforce2-lockup-patches-rd-2.6.3-mm3.tgz](#)