

Re: Log flood: "scheduling while atomic" (2.6.15.x, 2.6.16.x)

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2006-04/msg06646.html>

- *From:* Andrew Morton <akpm@xxxxxxxx>
 - *Date:* Sat, 29 Apr 2006 18:22:20 -0700
-

Darren Salt <linux@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx> wrote:

I'm seeing bouts of log flooding caused by something presumably not releasing a lock. I've looked at some of the messages, but at around 100/s, I'm not too keen to look through the whole lot :-)

```
scheduling while atomic: swapper/0xafbffff/0
[show_trace+19/32]
[dump_stack+30/32]
[schedule+1278/1472]
[cpu_idle+88/96]
[stext+44/64]
[start_kernel+574/704]
[L6+0/2] 0xc0100199
```

(Trailing parts of some lines have been omitted; it's all repeated data. And some sort of rate-limiting of these messages would be nice, but some other way to draw attention to the problem, e.g. an occasional beep, would be good.)

The most recent instance occurred a few minutes into recording a TV programme (via vdr) from a cx88-based Nova-T. (I'm currently using stock drivers rather than ones built from the v4l-dvb repository.)

Thanks for the report.

The below patch (against 2.6.17-rc3) should, if it still works, tell us which lock didn't get unlocked.

You'll need to enable CONFIG_PREEMPT and CONFIG_DEBUG_PREEMPT and CONFIG_FRAME_POINTER.

Please cc video4linux-list@xxxxxxxx on any result if it looks like v4l is indeed the culprit.

Re: Log flood: "scheduling while atomic" (2.6.15.x, 2.6.16.x)

From: Ingo Molnar <mingo@xxxxxxx>

this implements the "non-preemptible section trace" feature, which prints out a "critical section nesting" trace after stackdumps.

sample output on x86:

Call Trace:

```
[<c0103db1>] show_stack+0x7a/0x90
[<c0103f36>] show_registers+0x156/0x1ce
[<c010412e>] die+0xe8/0x172
[<c010422e>] do_trap+0x76/0xa3
[<c01044fe>] do_invalid_op+0xa3/0xad
[<c01039ef>] error_code+0x4f/0x54
[<c0120be9>] test+0x8/0xa
[<c0120c41>] sys_gettimeofday+0x56/0x74
[<c0102eeb>] sysenter_past_esp+0x54/0x75
```

| preempt count: 00000004]

| 4 levels deep critical section nesting:

```
-----
.. [<c0120bbe>] .... test3+0xd/0xf
....[<c0120bc8>] .. ( <= test2+0x8/0x21)
.. [<c0120bbe>] .... test3+0xd/0xf
....[<c0120bcd>] .. ( <= test2+0xd/0x21)
.. [<c0120bd7>] .... test2+0x17/0x21
....[<c0120be9>] .. ( <= test+0x8/0xa)
.. [<c010407f>] .... die+0x39/0x172
....[<c010422e>] .. ( <= do_trap+0x76/0xa3)
```

sample output on x64:

Call Trace:<ffffffff8012373c>{sys32_gettimeofday+60} <ffffffff80122af3>{cstar_do_call+27}

| preempt count: 00000002]

| 2 level deep critical section nesting:

```
-----
.. [<ffffffff8012372d>] .... sys32_gettimeofday+0x2d/0x110
....[<ffffffff80122af3>] .. ( <= cstar_do_call+0x1b/0x60)
.. [<ffffffff80123737>] .... sys32_gettimeofday+0x37/0x110
....[<ffffffff80122af3>] .. ( <= cstar_do_call+0x1b/0x60)
```

this feature is implemented via a low-overhead mechanism by keeping the caller and caller-parent addresses for each disable_preempt() call site, and printing it upon crashes. Note that every other locking API is thus traced too, such as spinlocks, rwlocks, per-cpu variables, etc. This feature is especially useful in identifying leaked preemption counts, as the missing count is displayed as an

Re: Log flood: "scheduling while atomic" (2.6.15.x, 2.6.16.x)

Re: Log flood: "scheduling while atomic" (2.6.15.x, 2.6.16.x)

extra entry in the stack.

the feature is active when DEBUG_PREEMPT is enabled.

i've also cleaned up preemption-count debugging by moving the debug functions out of sched.c into lib/preempt.c.

also, i have added preemption-counter-imbalance checks to the hardirq and softirq processing codepaths. The behavior of preemption-counter checks is now uniform: a warning is printed with all info we have at that point, and the preemption counter is then restored to the old value.

on x86 i have changed the 4KSTACKS feature to inherit the low bits of the preemption-count across hardirq/softirq-context switching, so that the preemption trace entries of interrupts do not overwrite process level preemption trace entries.

boot- and functionality-tested on x86 and x64. Should work on all architectures, but only x86 and x64 has been updated to print the trace-stack out at stackdump time.

This feature has been part of the -rt tree for quite some time and was very useful in debugging preempt-counter leaks, deadlock and lockup situations.

Signed-off-by: Ingo Molnar <mingo@xxxxxxx>
Signed-off-by: Andrew Morton <akpm@xxxxxxx>

```
arch/i386/kernel/irq.c | 28 ++++++++
arch/i386/kernel/traps.c | 1
arch/x86_64/kernel/process.c | 2
arch/x86_64/kernel/traps.c | 13 +---
include/asm-x86_64/proto.h | 2
include/linux/sched.h | 12 +++
kernel/exit.c | 9 +-
kernel/irq/handle.c | 17 +++++
kernel/sched.c | 35 -----
kernel/softirq.c | 17 +++++
kernel/timer.c | 26 ++++----
lib/Kconfig.debug | 2
lib/Makefile | 2
lib/preempt.c | 101 +++++++++++++++++++++++++++++++++++++
14 files changed, 210 insertions(+), 57 deletions(-)
```

```
diff -puN arch/i386/kernel/irq.c~preempt-tracing arch/i386/kernel/irq.c
--- devel/arch/i386/kernel/irq.c~preempt-tracing 2006-04-29 18:11:20.000000000 -0700
+++ devel-akpm/arch/i386/kernel/irq.c 2006-04-29 18:11:20.000000000 -0700
@@ -55,6 +55,9 @@ fastcall unsigned int do_IRQ(struct pt_r
{
```

Re: Log flood: "scheduling while atomic" (2.6.15.x, 2.6.16.x)

```
/* high bits used in ret_from_code */
int irq = regs->orig_eax & 0xff;
#ifdef CONFIG_DEBUG_PREEMPT
+ u32 count = preempt_count() & PREEMPT_MASK;
#endif
#ifdef CONFIG_4KSTACKS
union irq_ctx *curctx, *irqctx;
u32 *isp;
@@ -95,6 +98,14 @@ fastcall unsigned int do_IRQ(struct pt_r
irqctx->tinfo.task = curctx->tinfo.task;
irqctx->tinfo.previous_esp = current_stack_pointer;

+ /*
+ * Keep the preemption-count offset, so that the
+ * process-level preemption-trace entries do not
+ * get overwritten by the hardirq context:
+ */
#ifdef CONFIG_DEBUG_PREEMPT
+ irqctx->tinfo.preempt_count += count;
#endif
asm volatile(
" xchgl %%ebx,%%esp \n"
" call __do_IRQ \n"
@@ -103,6 +114,9 @@ fastcall unsigned int do_IRQ(struct pt_r
: "0" (irq), "1" (regs), "2" (isp)
: "memory", "cc", "ecx"
);
#ifdef CONFIG_DEBUG_PREEMPT
+ irqctx->tinfo.preempt_count -= count;
#endif
} else
#endif
__do_IRQ(irq, regs);
@@ -165,6 +179,9 @@ extern asmlinkage void __do_softirq(void

asmlinkage void do_softirq(void)
{
#ifdef CONFIG_DEBUG_PREEMPT
+ u32 count = preempt_count() & PREEMPT_MASK;
#endif
unsigned long flags;
struct thread_info *curctx;
union irq_ctx *irqctx;
@@ -181,6 +198,14 @@ asmlinkage void do_softirq(void)
irqctx->tinfo.task = curctx->task;
irqctx->tinfo.previous_esp = current_stack_pointer;

+ /*
+ * Keep the preemption-count offset, so that the
+ * process-level preemption-trace entries do not
+ * get overwritten by the softirq context:
```

Re: Log flood: "scheduling while atomic" (2.6.15.x, 2.6.16.x)

```
+ */
+#ifdef CONFIG_DEBUG_PREEMPT
+ irqctx->tinfo.preempt_count += count;
+#endif
/* build the stack frame on the softirq stack */
isp = (u32*) ((char*)irqctx + sizeof(*irqctx));

@@ -192,6 +217,9 @@ asmlinkage void do_softirq(void)
: "0"(isp)
: "memory", "cc", "edx", "ecx", "eax"
);
+#ifdef CONFIG_DEBUG_PREEMPT
+ irqctx->tinfo.preempt_count -= count;
+#endif
}

local_irq_restore(flags);
diff -puN arch/i386/kernel/traps.c~preempt-tracing arch/i386/kernel/traps.c
--- devel/arch/i386/kernel/traps.c~preempt-tracing 2006-04-29 18:11:20.000000000 -0700
+++ devel-akpm/arch/i386/kernel/traps.c 2006-04-29 18:11:20.000000000 -0700
@@ -191,6 +191,7 @@ static void show_trace_log_lvl(struct ta
break;
printk("%s =====\n", log_lvl);
}
+ print_preempt_trace(task, task->thread_info->preempt_count);
}

void show_trace(struct task_struct *task, unsigned long *stack)
diff -puN arch/x86_64/kernel/process.c~preempt-tracing arch/x86_64/kernel/process.c
--- devel/arch/x86_64/kernel/process.c~preempt-tracing 2006-04-29 18:11:20.000000000 -0700
+++ devel-akpm/arch/x86_64/kernel/process.c 2006-04-29 18:11:20.000000000 -0700
@@ -335,7 +335,7 @@ void show_regs(struct pt_regs *regs)
{
printk("CPU %d:", smp_processor_id());
__show_regs(regs);
- show_trace(&regs->rsp);
+ show_trace(current, &regs->rsp);
}

/*
diff -puN arch/x86_64/kernel/traps.c~preempt-tracing arch/x86_64/kernel/traps.c
--- devel/arch/x86_64/kernel/traps.c~preempt-tracing 2006-04-29 18:11:20.000000000 -0700
+++ devel-akpm/arch/x86_64/kernel/traps.c 2006-04-29 18:11:20.000000000 -0700
@@ -196,7 +196,7 @@ static unsigned long *in_exception_stack
* severe exception (double fault, nmi, stack fault, debug, mce) hardware stack
*/

-void show_trace(unsigned long *stack)
+void show_trace(struct task_struct *task, unsigned long *stack)
{
const unsigned cpu = safe_smp_processor_id();
```

Re: Log flood: "scheduling while atomic" (2.6.15.x, 2.6.16.x)

```
unsigned long *irqstack_end = (unsigned long *)cpu_pda(cpu)->irqstackptr;
@@ -260,9 +260,10 @@ void show_trace(unsigned long *stack)
HANDLE_STACK (((long) stack & (THREAD_SIZE-1)) != 0);
#undef HANDLE_STACK
printk("\n");
+ print_preempt_trace(task, task->thread_info->preempt_count);
}

-void show_stack(struct task_struct *tsk, unsigned long *rsp)
+void show_stack(struct task_struct *task, unsigned long *rsp)
{
unsigned long *stack;
int i;
@@ -274,8 +275,8 @@ void show_stack(struct task_struct *tsk,
// back trace for this cpu.

if (rsp == NULL) {
- if (tsk)
- rsp = (unsigned long *)tsk->thread.rsp;
+ if (task)
+ rsp = (unsigned long *)task->thread.rsp;
else
rsp = (unsigned long *)&rsp;
}
@@ -296,7 +297,7 @@ void show_stack(struct task_struct *tsk,
printk("%016lx ", *stack++);
touch_nmi_watchdog();
}
- show_trace((unsigned long *)rsp);
+ show_trace(task, (unsigned long *)rsp);
}

/*
@@ -305,7 +306,7 @@ void show_stack(struct task_struct *tsk,
void dump_stack(void)
{
unsigned long dummy;
- show_trace(&dummy);
+ show_trace(current, &dummy);
}

EXPORT_SYMBOL(dump_stack);
diff -puN include/asm-x86_64/proto.h~preempt-tracing include/asm-x86_64/proto.h
--- devel/include/asm-x86_64/proto.h~preempt-tracing 2006-04-29 18:11:20.000000000 -0700
+++ devel-akpm/include/asm-x86_64/proto.h 2006-04-29 18:11:20.000000000 -0700
@@ -75,7 +75,7 @@ extern void main_timer_handler(struct pt

extern unsigned long end_pfn_map;

-extern void show_trace(unsigned long *rsp);
+extern void show_trace(struct task_struct *task, unsigned long *rsp);
```

Re: Log flood: "scheduling while atomic" (2.6.15.x, 2.6.16.x)

Re: Log flood: "scheduling while atomic" (2.6.15.x, 2.6.16.x)

```
extern void show_registers(struct pt_regs *regs);

extern void exception_table_check(void);
diff -puN include/linux/sched.h~preempt-tracing include/linux/sched.h
--- devel/include/linux/sched.h~preempt-tracing 2006-04-29 18:11:20.000000000 -0700
+++ devel-akpm/include/linux/sched.h 2006-04-29 18:12:02.000000000 -0700
@@ -638,6 +638,14 @@ extern unsigned int max_cache_size;

#endif /* CONFIG_SMP */

#ifdef CONFIG_DEBUG_PREEMPT
#define MAX_PREEMPT_TRACE 25
extern void print_preempt_trace(struct task_struct *task, u32 count);
#else
static inline void print_preempt_trace(struct task_struct *task, u32 count)
+{
+}
#endif

struct io_context; /* See blkdev.h */
void exit_io_context(void);
@@ -884,6 +892,10 @@ struct task_struct {
atomic_t fs_excl; /* holding fs exclusive resources */
struct rcu_head rcu;

#ifdef CONFIG_DEBUG_PREEMPT
+ void *preempt_off_caller[MAX_PREEMPT_TRACE];
+ void *preempt_off_parent[MAX_PREEMPT_TRACE];
#endif
/*
* cache last used pipe for splice
*/
diff -puN kernel/exit.c~preempt-tracing kernel/exit.c
--- devel/kernel/exit.c~preempt-tracing 2006-04-29 18:11:20.000000000 -0700
+++ devel-akpm/kernel/exit.c 2006-04-29 18:11:20.000000000 -0700
@@ -888,10 +888,11 @@ fastcall NORET_TYPE void do_exit(long co
tsk->it_prof_expires = cputime_zero;
tsk->it_sched_expires = 0;

- if (unlikely(in_atomic()))
- printk(KERN_INFO "note: %s[%d] exited with preempt_count %d\n",
- current->comm, current->pid,
- preempt_count());
+ if (unlikely(in_atomic())) {
+ printk(KERN_ERR "BUG: %s[%d] exited with nonzero preempt_count %d!\n",
+ tsk->comm, tsk->pid, preempt_count());
+ print_preempt_trace(tsk, preempt_count());
+ }

acct_update_integrals(tsk);
if (tsk->mm) {
```

Re: Log flood: "scheduling while atomic" (2.6.15.x, 2.6.16.x)

```
diff -puN kernel/irq/handle.c~preempt-tracing kernel/irq/handle.c
--- devel/kernel/irq/handle.c~preempt-tracing 2006-04-29 18:11:20.000000000 -0700
+++ devel-akpm/kernel/irq/handle.c 2006-04-29 18:11:20.000000000 -0700
@@ -85,7 +85,24 @@ fastcall int handle_IRQ_event(unsigned i
local_irq_enable();

do {
+#ifdef CONFIG_DEBUG_PREEMPT
+ u32 in_count = preempt_count(), out_count;
+#endif
ret = action->handler(irq, action->dev_id, regs);
+#ifdef CONFIG_DEBUG_PREEMPT
+ out_count = preempt_count();
+ if (in_count != out_count) {
+ printk(KERN_ERR "BUG: irq %d [%s] preempt-count "
+ "imbalance: in=%08x, out=%08x!\n",
+ irq, action->name, in_count, out_count);
+ print_preempt_trace(current, out_count);
+ /*
+ * We already printed all the useful info,
+ * fix up the preemption count now:
+ */
+ preempt_count() = in_count;
+ }
+#endif
if (ret == IRQ_HANDLED)
status |= action->flags;
retval |= ret;
diff -puN kernel/sched.c~preempt-tracing kernel/sched.c
--- devel/kernel/sched.c~preempt-tracing 2006-04-29 18:11:20.000000000 -0700
+++ devel-akpm/kernel/sched.c 2006-04-29 18:12:36.000000000 -0700
@@ -50,8 +50,9 @@
#include <linux/times.h>
#include <linux/acct.h>
#include <linux/kprobes.h>
-#include <asm/tlb.h>
+#include <linux/kallsyms.h>

+#include <asm/tlb.h>
#include <asm/unistd.h>

/*
@@ -2877,38 +2878,6 @@ static inline int dependent_sleeper(int
}
#endif

-#if defined(CONFIG_PREEMPT) && defined(CONFIG_DEBUG_PREEMPT)
-
-void fastcall add_preempt_count(int val)
- {
- /*
```

```

- * Underflow?
- */
- BUG_ON((preempt_count() < 0));
- preempt_count() += val;
- /*
- * Spinlock count overflowing soon?
- */
- BUG_ON((preempt_count() & PREEMPT_MASK) >= PREEMPT_MASK-10);
-}
-EXPORT_SYMBOL(add_preempt_count);
-
-void fastcall sub_preempt_count(int val)
-{
- /*
- * Underflow?
- */
- BUG_ON(val > preempt_count());
- /*
- * Is the spinlock portion underflowing?
- */
- BUG_ON((val < PREEMPT_MASK) && !(preempt_count() & PREEMPT_MASK));
- preempt_count() -= val;
-}
-EXPORT_SYMBOL(sub_preempt_count);
-
-#endif
-
static inline int interactive_sleep(enum sleep_type sleep_type)
{
return (sleep_type == SLEEP_INTERACTIVE ||
diff -puN kernel/softirq.c~preempt-tracing kernel/softirq.c
--- devel/kernel/softirq.c~preempt-tracing 2006-04-29 18:11:20.000000000 -0700
+++ devel-akpm/kernel/softirq.c 2006-04-29 18:11:20.000000000 -0700
@@ -93,7 +93,24 @@ restart:

do {
if (pending & 1) {
#ifdef CONFIG_DEBUG_PREEMPT
+ u32 in_count = preempt_count(), out_count;
#endif
h->action(h);
#ifdef CONFIG_DEBUG_PREEMPT
+ out_count = preempt_count();
+ if (in_count != out_count) {
+ printk(KERN_ERR "BUG: softirq %ld preempt-count"
+ " imbalance: in=%08x, out=%08x!\n",
+ (long)(h - softirq_vec),
+ in_count, out_count);
+ print_preempt_trace(current, out_count);
+ /*
+ * Fix up the bad preemption count:

```

Re: Log flood: "scheduling while atomic" (2.6.15.x, 2.6.16.x)

```
+ */
+ preempt_count() = in_count;
+ }
+#endif
rcu_bh_qsctr_inc(cpu);
}
h++;
diff -puN kernel/timer.c~preempt-tracing kernel/timer.c
--- devel/kernel/timer.c~preempt-tracing 2006-04-29 18:11:20.000000000 -0700
+++ devel-akpm/kernel/timer.c 2006-04-29 18:13:35.000000000 -0700
@@ -34,6 +34,7 @@
#include <linux/cpu.h>
#include <linux/syscalls.h>
#include <linux/delay.h>
+#include <linux/kallsyms.h>

#include <asm/uaccess.h>
#include <asm/unistd.h>
@@ -436,6 +437,7 @@ static inline void __run_timers(tvec_base
while (!list_empty(head)) {
void (*fn)(unsigned long);
unsigned long data;
+ int in_count, out_count;

timer = list_entry(head->next, struct timer_list, entry);
fn = timer->function;
@@ -444,17 +446,19 @@ static inline void __run_timers(tvec_base
set_running_timer(base, timer);
detach_timer(timer, 1);
spin_unlock_irq(&base->lock);
- {
- int preempt_count = preempt_count();
- fn(data);
- if (preempt_count != preempt_count()) {
- printk(KERN_WARNING "huh, entered %p "
- "with preempt_count %08x, exited"
- " with %08x?\n",
- fn, preempt_count,
- preempt_count());
- BUG();
- }
+ in_count = preempt_count();
+ fn(data);
+ out_count = preempt_count();
+ if (in_count != out_count) {
+ print_symbol(KERN_ERR "BUG: %s", (long)fn);
+ printk(KERN_ERR "(%p) preempt-count imbalance: "
+ "in=%08x, out=%08x!",
+ fn, in_count, out_count);
+ print_preempt_trace(current, out_count);
+ }
+ */
```

Re: Log flood: "scheduling while atomic" (2.6.15.x, 2.6.16.x)

```
+ * Fix up the bad preemption count:
+ */
+ preempt_count() = in_count;
+ }
spin_lock_irq(&base->lock);
+ }
diff -puN lib/Kconfig.debug~preempt-tracing lib/Kconfig.debug
--- devel/lib/Kconfig.debug~preempt-tracing 2006-04-29 18:11:20.000000000 -0700
+++ devel-akpm/lib/Kconfig.debug 2006-04-29 18:11:20.000000000 -0700
@@ -92,6 +92,8 @@ config DEBUG_SLAB_LEAK
config DEBUG_PREEMPT
bool "Debug preemptible kernel"
depends on DEBUG_KERNEL && PREEMPT
+ select FRAME_POINTER
+ select KALLSYMS_ALL
default y
help
If you say Y here then the kernel will use a debug variant of the
diff -puN lib/Makefile~preempt-tracing lib/Makefile
--- devel/lib/Makefile~preempt-tracing 2006-04-29 18:11:20.000000000 -0700
+++ devel-akpm/lib/Makefile 2006-04-29 18:11:20.000000000 -0700
@@ -25,7 +25,7 @@ lib-$(CONFIG_SEMAPHORE_SLEEPERS) += sema
lib-$(CONFIG_GENERIC_FIND_NEXT_BIT) += find_next_bit.o
lib-$(CONFIG_GENERIC_HWEIGHT) += hweight.o
obj-$(CONFIG_LOCK_KERNEL) += kernel_lock.o
-obj-$(CONFIG_DEBUG_PREEMPT) += smp_processor_id.o
+obj-$(CONFIG_DEBUG_PREEMPT) += smp_processor_id.o preempt.o

ifneq ($(CONFIG_HAVE_DEC_LOCK),y)
lib-y += dec_and_lock.o
diff -puN /dev/null lib/preempt.c
--- /dev/null 2003-09-15 06:40:47.000000000 -0700
+++ devel-akpm/lib/preempt.c 2006-04-29 18:11:20.000000000 -0700
@@ -0,0 +1,101 @@
+/*
+ * lib/preempt.c
+ *
+ * DEBUG_PREEMPT variant of add_preempt_count() and sub_preempt_count().
+ * Preemption tracing.
+ *
+ * (C) 2005 Ingo Molnar, Red Hat
+ */
+#include <linux/module.h>
+#include <linux/hardirq.h>
+#include <linux/kallsyms.h>
+
+/*
+ * Add a value to the preemption count, and check for overflows,
+ * underflows and maintain a small stack of callers that gets
+ * printed upon crashes.
+ */
```

Re: Log flood: "scheduling while atomic" (2.6.15.x, 2.6.16.x)

```
+void fastcall add_preempt_count(int val)
+{
+ unsigned int count = preempt_count(), idx = count & PREEMPT_MASK;
+
+ /*
+ * Underflow?
+ */
+ BUG_ON(count < 0);
+
+ preempt_count() += val;
+
+ /*
+ * Spinlock count overflowing soon?
+ */
+ BUG_ON(idx >= PREEMPT_MASK-10);
+
+ /*
+ * Maintain the per-task preemption-nesting stack (which
+ * will be printed upon crashes). It's a low-overhead thing,
+ * constant overhead per preempt-disable.
+ */
+ if (idx < MAX_PREEMPT_TRACE) {
+ void *caller = __builtin_return_address(0), *parent = NULL;
+
+ #ifdef CONFIG_FRAME_POINTER
+ parent = __builtin_return_address(1);
+ if (in_lock_functions((unsigned long)parent)) {
+ parent = __builtin_return_address(2);
+ if (in_lock_functions((unsigned long)parent))
+ parent = __builtin_return_address(3);
+ }
+ #endif
+ current->preempt_off_caller[idx] = caller;
+ current->preempt_off_parent[idx] = parent;
+ }
+}
+EXPORT_SYMBOL(add_preempt_count);
+
+void fastcall sub_preempt_count(int val)
+{
+ unsigned int count = preempt_count();
+
+ /*
+ * Underflow?
+ */
+ BUG_ON(val > count);
+
+ /*
+ * Is the spinlock portion underflowing?
+ */
+ BUG_ON((val < PREEMPT_MASK) && !(count & PREEMPT_MASK));
+
+}
```

