

# [BUG -rt] Double OOPs – thread\_info free race / printk recursive lock

---

*Source:* <http://linux.derkeiler.com/Mailing-Lists/Kernel/2006-08/msg01565.html>

---

- *From:* Darren Hart <[dvhltc@xxxxxxxxxx](mailto:dvhltc@xxxxxxxxxx)>
  - *Date:* Fri, 4 Aug 2006 10:43:05 -0700
- 

We've seen very rarely over the last few months, on various -rt kernels. The latest reproduction is on 2.6.16-rt22 (+some minor fixups). Analysis of the vmcore produced by kdump suggests two problems:

- 1) An invalid pointer dereference in cache\_flusharray() which causes the page fault.
- 2) Then printk calls kmallocc when trying to print the oops, which grabs a recursive lock and prints a different oops.

I'm focusing on the first oops here. The executive summary is that when we try to free a task's thread\_info struct we end up trying to free a page that is already free. I didn't see anything in more recent -rt patches that seem to address this issue. Any thoughts as to why we end up trying to double free the thread\_info struct?

A summary of our analysis:

crash sys and bt follow:

```
KERNEL: /usr/lib/debug/lib/modules/2.6.16-rtj12.7smp/vmlinux
DUMPFILE: vmcore
CPUS: 4
DATE: Wed Aug 2 17:35:17 2006
UPTIME: 00:20:47
LOAD AVERAGE: 34.84, 22.95, 19.53
TASKS: 2523
NODENAME: hammervm22
RELEASE: 2.6.16-rtj12.7smp
VERSION: #1 SMP PREEMPT Fri Jul 28 17:16:36 PDT 2006
MACHINE: i686 (1993 Mhz)
MEMORY: 5.5 GB
PANIC: "kernel BUG at kernel/rtmutex.c:639!"
PID: 21
COMMAND: "softirq-tasklet"
TASK: cae6e6e0 [THREAD_INFO: cae76000]
CPU: 1
STATE: TASK_RUNNING (PANIC)
```

```
crash> bt
```

[BUG -rt] Double OOPs – thread\_info free race / printk recursive lock

```
PID: 21 TASK: cae6e6e0 CPU: 1 COMMAND: "softirq-tasklet"
#0 [cae77ab4] crash_kexec at c013c073
#1 [cae77b08] die at c0103a7f
#2 [cae77b40] do_invalid_op at c0103c81
#3 [cae77c08] error_code (via invalid_op) at c01033b9
EAX: 0000001d EBX: cae76000 ECX: cae77c44 EDX: c0336542 EBP:
00000020
DS: 007b ESI: c9db8560 ES: 007b EDI: 00000020
CS: 0060 EIP: c0321085 ERR: ffffffff EFLAGS: 00010292
#4 [cae77c3c] rt_lock_slowlock at c0321085
#5 [cae77c94] __kmalloc at c015afc4
#6 [cae77ca8] soft_cursor at c01e0af4
#7 [cae77cd8] bit_cursor at c01e0999
#8 [cae77d5c] fbcon_cursor at c01dca2a
#9 [cae77d94] hide_cursor at c021c765
#10 [cae77da4] vt_console_print at c021ed73
#11 [cae77dc0] __call_console_drivers at c011dd25
#12 [cae77ddc] call_console_drivers at c011de8b
#13 [cae77dfc] release_console_sem at c011e25f
#14 [cae77e14] vprintk at c011e17c
#15 [cae77e7c] printk at c011df69
#16 [cae77e88] do_page_fault at c0322c99
#17 [cae77ee0] error_code (via page_fault) at c01033b9
EAX: 00000000 EBX: c5af2870 ECX: 0000002c EDX: cae6e6e0 EBP:
80000000
DS: 007b ESI: c5b5ba30 ES: 007b EDI: cae77f5c
CS: 0060 EIP: c03217a5 ERR: ffffffff EFLAGS: 00010286
#18 [cae77f14] rt_lock at c03217a5
#19 [cae77f18] cache_flusharray at c015adf2
#20 [cae77f34] __cache_free at c015aea8
#21 [cae77f4c] kfree at c015b15f
#22 [cae77f68] free_task at c011aeb8
#23 [cae77f74] rcu_process_callbacks at c0140c36
#24 [cae77f84] __tasklet_action at c0122854
#25 [cae77f98] tasklet_action at c01228ae
#26 [cae77fa4] ksoftirqd at c0122a45
#27 [cae77fc4] kthread at c012e80d
#28 [cae77fe8] kernel_thread_helper at c0100f8f
```

A bit of analysis on the original fault. Note the beginning of the call chain:

```
#22 [cae77f68] free_task at c011aeb8
#23 [cae77f74] rcu_process_callbacks at c0140c36
#24 [cae77f84] __tasklet_action at c0122854
#25 [cae77f98] tasklet_action at c01228ae
#26 [cae77fa4] ksoftirqd at c0122a45
#27 [cae77fc4] kthread at c012e80d
#28 [cae77fe8] kernel_thread_helper at c0100f8f
```

As part of a call back, we are attempting to free a task structure. The

## [BUG -rt] Double OOPs – thread\_info free race / printk recursive lock

address of the task structure is:  
0xf314f770

One of the first things task\_free() does is call kfree(task->thread\_info).  
In this case task->thread\_info is:  
0xf20b8000

kfree does a 'c = virt\_to\_cache(objp)' on the passed address.  
virt\_to\_cache() is a simple routine:  
static inline struct kmem\_cache \*virt\_to\_cache(const void \*obj)  
{  
struct page \*page = virt\_to\_page(obj);  
return page\_get\_cache(page);  
}

Note that virt\_to\_page(0xf20b8000) gives a page struct at 0xc5af2840.  
page\_get\_cache() simply returns the page->lru.next field which is used as a  
pointer to a kmem\_cache structure. Here is the page structure:

```
struct page {  
    flags = 2148007936,  
    _count = {  
        counter = -1  
    },  
    _mapcount = {  
        counter = -1  
    },  
    {  
        ptl = {  
            lock = {  
                wait_lock = {  
                    raw_lock = {  
                        slock = 1  
                    },  
                    break_lock = 0  
                },  
                wait_list = {  
                    prio_list = {  
                        next = 0x0,  
                        prev = 0x0  
                    },  
                    node_list = {  
                        next = 0x0,  
                        prev = 0x0  
                    }  
                },  
                owner = 0x0  
            },  
            break_lock = 0  
        }  
    },  
};
```

[BUG -rt] Double OOPs – thread\_info free race / printk recursive lock

```
index = 0,  
lru = {  
next = 0xc5b5ba30,  
prev = 0xc5b27310  
}  
}
```

Note the count value of -1 which normally indicates a free page. Hence, it looks like we are trying to free a page that is already free. If the page is already free, the lru.next field does not contain the expected data (a kmem\_cache pointer). This starts a bad chain of events as we start keying off fields in the bad kmem\_cache structure.

Examination of the lru.next pointer confirms that it is another page (@ 0x5ba00) with the same page\_flags 0x80008000 (#define PG\_buddy 19 /\* Page is free, on buddy lists \*/)

Looks like root cause may be trying to free twice. More investigation needed.

--

Darren Hart  
IBM Linux Technology Center  
Realtime Linux Team

-

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

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

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