

## Sparse "context" checking..

**Source:** <http://linux.derkeiler.com/Mailing-Lists/Kernel/2004-10/10190.html>

---

**From:** Linus Torvalds ([torvalds\\_at\\_osdl.org](mailto:torvalds_at_osdl.org))

**Date:** 10/31/04

Date: Sat, 30 Oct 2004 20:20:53 -0700 (PDT)  
To: Kernel Mailing List <[linux-kernel@vger.kernel.org](mailto:linux-kernel@vger.kernel.org)>

I just committed the patches to the kernel to start supporting a new automated correctness check that I added to sparse: the counting of static "code context" information.

The sparse infrastructure is pretty agnostic, and you can count pretty much anything you want, but it's designed to test that the entry and exit contexts match, and that no path through a function is ever entered with conflicting contexts.

In particular, this is designed for doing things like matching up a "lock" with the pairing "unlock", and right now that's exactly what the code does: it makes each spinlock count as "+1" in the context, and each spinunlock count as "-1", and then hopefully it should all add up.

It doesn't always, of course. Since it's a purely static analyser, it's unhappy about code like

```
int fn(arg)
{
    if (arg)
        spin_lock(lock);
    ...
    if (arg)
        spin_unlock(lock);
}
```

because the code is not statically deterministic, and the stuff in between can be called with or without a lock held. That said, this has long been frowned upon, and there aren't that many cases where it happens.

Right now the counting is only enabled if you use sparse, and add the "-Wcontext" flag to the sparse command line by hand – and the spinlocks have only been annotated for the SMP case, so right now it only works for CONFIG\_SMP. Details, details.

Also, since sparse does purely local decisions, if you actually `_intend_`

## Linux-Kernel: Sparse "context" checking..

to grab a lock in one function and release it in another, you need to tell sparse so, by annotating the function that acquires the lock (with "\_\_acquires(lockname)") and the function that releases it (with, surprise surprise, "\_\_releases(lockname)") in the declaration. That tells sparse to update the context in the callers appropriately, but it also tells sparse to expect the proper entry/exit contexts for the annotated functions themselves.

I haven't done the annotation for any functions yet, so expect warnings. If you do a checking run, the warnings will look something like:

```
CHECK kernel/resource.c
kernel/resource.c:59:13: warning: context imbalance in 'r_start' - wrong count at exit
kernel/resource.c:69:13: warning: context imbalance in 'r_stop' - unexpected unlock
```

which just shows that "r\_start" acquired a lock, and sparse didn't expect it to, while "r\_stop" released a lock that sparse hadn't realized it had. In this case, the cause is pretty obvious, and the annotations are equally so.

A more complicated case is

```
CHECK kernel/sys.c
kernel/sys.c:465:2: warning: context imbalance in 'sys_reboot' - different lock contexts for basic block
```

where that "different lock contexts" warning means that sparse determined that some code in that function was reachable with two different lock contexts. In this case it's actually harmless, since what happens in this case is that the code after rebooting the machine is unreachable, and sparse just doesn't understand that.

But in other cases it's more fundamental, and the lock imbalance is due to dynamic data that sparse just can't understand. The warning in that case can be disabled by hand, but there doesn't seem to be that many of them. A full kernel build for me has about 200 warnings, and most of them seem to be the benign kind (ie the kind above where one function acquires the lock and another releases it, and they just haven't been annotated as such).

The sparse thing could be extended to `_any_` context that wants pairing, and I just wanted to let people know about this in case they find it interesting..

Linus

-

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in the body of a message to [majordomo@vger.kernel.org](mailto: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/>