

Re: Locking fundamentals

Source: <http://unix.derkeiler.com/Mailing-Lists/FreeBSD/hackers/2006-12/msg00205.html>

- *From:* "Attilio Rao" <attilio@xxxxxxxxxxx>
 - *Date:* Fri, 22 Dec 2006 23:25:53 +0100
-

2006/12/22, Robert Watson <rwatson@xxxxxxxxxxx>:

On Thu, 21 Dec 2006, Attilio Rao wrote:

>> I disagree. There are many uses of atomic operations in the kernel that are
>> not for locks or refcounts. It's a bad idea to use locks if you can achieve
>> the same thing locklessly, with atomic operations.

>

> I can agree with you about this but atomic instructions/memory barriers
> should be used very carefully (and if you know what you are going to do). It
> is very simple to write down a wrong semantic using them.

Agreed here also --- it is easy to use atomic operations incorrectly, especially with decreasing memory consistency properties. Simple statistics are about the only place they're safe to use without very careful thinking. There are some nice examples in the updated version of Andrew Birrell's threading paper of just how easy it is to shoot feet using atomic operations.

>> I would personally also add "critical sections"
>> (`critical_enter()/critical_exit()`) at level I. They can be used instead of
>> locks when you know your data will only be accessed on one CPU, and you
>> only need to protect it from (non-FAST) interrupt handlers.

>

> From this point of view, we would also add `sched_pin()/sched_unpin()` which
> are used in order to avoid thread migration between CPUs (particularly
> helpfull in the case we have to access safely to some per-CPU datas).
> However, probably one of the most important usage we do of
> `critical_section` is in the spin mutex implementation (which linked to
> interrupt disabling would avoid deadlock in spin mutex code).

`sched_bind()`, `sched_pin()`, and friends are also potentially dangerous unless used very carefully: they work alright if the thread is dedicated to a single purpose and written entirely with that synchronization model in mind, or if the period of pinning is brief (and it doesn't matter what CPU it's on).

However, as soon as you have multiple code modules interacting with each other, you have to be very careful about the thread moving around when not expected by calling code, starvation issues, etc. Pinning a thread briefly using a critical section to access data on the whatever the current CPU may be is a fine (and good) strategy, but service components of the kernel should not

Re: Locking fundamentals

employ extended pinning unless it's been carefully thought out, especially with regard to other components it may call, or that may call it. In that sense, of course, it's not different than locking, only we have a weaker assertion/consistency verification system. It's frequently the interactions between components/subsystems that make synchronization most difficult.

...and please note that it is worthwhile just for preemptive kernels.

Attilio

--

Peace can only be achieved by understanding – A. Einstein

freebsd-hackers@xxxxxxxxxxx mailing list

<http://lists.freebsd.org/mailman/listinfo/freebsd-hackers>

To unsubscribe, send any mail to "freebsd-hackers-unsubscribe@xxxxxxxxxxx"