

Re: msleep() on recursively locked mutexes

Source: <http://unix.derkeiler.com/Mailing-Lists/FreeBSD/hackers/2007-04/msg00217.html>

- *From:* Hans Petter Selasky <hselasky@xxxxxxx>
 - *Date:* Fri, 27 Apr 2007 07:48:49 +0200
-

On Thursday 26 April 2007 23:50, Attilio Rao wrote:

2007/4/26, Julian Elischer <julian@xxxxxxxxxxxxx>:

The reason that mutexes ever recurse in the first place is usually because one piece of code calls itself (or a related piece of code) in a blind manner.. in other words, it doesn't know it is doing so. The whole concept of recursing mutexes is a bit gross. The concept of blindly unwinding them is I think just asking for trouble.

Further the idea that holding a mutex "except for when we sleep" is a generally bright idea is also a bit odd to me.. If you hold a mutex and release it during sleep you probably should invalidate all assumptions you made during the period before you slept as whatever you were

That is not always correct. If you run your code in a separate thread/taskqueue, then you simply wait for this thread/taskqueue to complete somewhere else. This is basically when I need to exit mutexes.

protecting has possibly been raped while you slept. I have seen too many instances where people just called msleep and dropped the mutex they held, picked it up again on wakeup, and then blithely continued on without checking what happened while they were asleep.

Basically, the idea you cannot hold "blocking" locks (mutexes and rwlocks) while sleeping, comes from the difference there is behind turnstiles and sleepqueues.

Turnstiles are thought to serve synchronous events, for short period of time (or rather short) while sleepqueues are thought to serve asynchronous events, so that the path to be protected can be definitively bigger. If you fit in the situation you have to call first a blocking lock and later a sleeping lock, probably you are just using a wrong locking strategy and you should really revisit it.

Re: msleep() on recursively locked mutexes

The suggestion is just for convenience. Usually you don't have a recursed mutex to sleep on. It is just to catch some rare cases where you will end up with a doubly locked mutex, which is not part of the ordinary code path. I don't have such cases in the kernel of the new USB stack, but there are some cases in the USB device drivers, which is due to some mutex locking moves. Those I can fix.

My idea was that by allowing recursive mutexes to sleep, you will end up with less panics in the end for the unwary code developer. You just protect your code with mutexes and if they recurse calling synchronous USB functions, you don't have to care.

As you mention, it is not always possible to drop the blocking lock before to sleep since you can break your critical path and free the way for races of various genre. Even unlocking Giant, that is auto-magically done by sleeping primitives, can lead to very difficult to discover races (I can remind one in tty code, old of some months, that can be a good proof-of-concept for that).

--HPS

freebsd-hackers@xxxxxxxxxxx mailing list

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

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