

Re: msleep() on recursively locked mutexes

Source: <http://unix.derkeiler.com/Mailing-Lists/FreeBSD/hackers/2007-04/msg00229.html>

- *From:* Hans Petter Selasky <hselasky@xxxxxxxxxxxxx>
 - *Date:* Fri, 27 Apr 2007 19:17:29 +0200
-

On Friday 27 April 2007 15:14, Daniel Eischen wrote:

On Fri, 27 Apr 2007, Hans Petter Selasky wrote:

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.

I know there are reasons why we have `msleep()`, but use of `msleep()` and recursive mutexes should raise be frowned upon. If you want to sleep, that's why we have `condvar`'s. And if you are releasing a synchronization lock in a different thread, then why aren't you using `condvar`'s wrapped by `mutexes()` to protect the internal state?

Re: msleep() on recursively locked mutexes

When I find time, I will convert my new USB stack into using condvar's, but they will do the same as msleep(), and that is to exit the passed mutex.

When you hold a mutex, it should be for a very short time. And I agree with the other comment that all drivers should be multi-thread safe, so we shouldn't add cruft to allow for non MT-safe drivers.

Yes, and no.

Mutexes are used to get the CPU out of the code. Therefore you should not lock/unlock all the time, to ensure that the locked time is as short as possible. Because then you get double work checking the state after that you lock a mutex again. Surely, in a "static" environment there is nothing to check. But in a dynamic environment you need to check that "descriptors" of all kinds are still present, after that you lock a mutex. Unlocking a mutex allows "anything" to happen. Keeping a mutex locked prevents certain things from happening.

First of all: Where is FreeBSD's locking strategy document? We should have a global strategy when we write device drivers, so that various modules can be connected together without races and locking order reversals.

In my view there are two categories of mutexes.

1) Global mutexes.

2) Private mutexes.

Rule: The Global mutex is locked last.

How do we organize this.

1a) The global mutex protects interrupts/callbacks into a hardware driver. This is the lowest level.

2a) Private mutexes protects sub-devices of the hardware driver. This might be as simple as an IF-QUEUE.

I have chosen the following model:

P0 indicates process 0. P1 indicates an optional second process.

Up-call:

```
P0 lock(2);
P0 lock(1);
P0 unlock(1);
```

Re: msleep() on recursively locked mutexes

Re: msleep() on recursively locked mutexes

P0 unlock(2);

Down-call:

P1 lock(1);

P1 wakeup P0 // for example

P1 unlock(1);

P0 //callback:

P0 lock(2); // the new USB stack currently uses P1 here

P0 unlock(2);

Sure, in the up-call you lock #2 longer than lock #1, that is why lock #1 has got to be as short as possible. But it does not make sense to make lock #2 lock shorter than lock #1. I cannot see that the system will go faster that way.

In the downcall I see no problems at all. #1 does its things as fast as possible. In parallel #2 can still execute, until the "callback".

I hope you understand my semantics.

What do you want to change from what I have explained?

Any comments?

My conclusion: If you want more parallelism, then use more mutexes. Don't try to push an existing mutex by unlocking it!

--HPS

freebsd-hackers@xxxxxxxxxxx mailing list

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

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