

Re: duplicate read/write locks in net/pfil.c and netinet/ip_fw2.c

Source: <http://unix.derkeiler.com/Mailing-Lists/FreeBSD/arch/2005-08/0156.html>

From: Stephan Uphoff (ups_at_tree.com)

Date: 08/18/05

To: Luigi Rizzo <rizzo@icir.org>

Date: Thu, 18 Aug 2005 10:18:33 -0400

On Thu, 2005-08-18 at 03:57, Luigi Rizzo wrote:

> On Thu, Aug 18, 2005 at 03:32:19AM +0200, Max Laier wrote:

>> On Thursday 18 August 2005 02:02, Luigi Rizzo wrote:

> ...

>>> could you guys look at the following code and see if it makes sense,

>>> or tell me where i am wrong ?

>>>

>>> It should solve the starvation and blocking trylock problems,

>>> because the active reader does not hold the mutex in the critical

> ^^^^^

>

> i meant 'writer', sorry... as max said even in the current implementation

> the reader does not hold the lock.

>

>>> int

>>> RLOCK(struct rwlock *rwl, int try)

>>> {

>>> if (!try)

>>> mtx_lock(&rwl->m);

>>> else if (!mtx_trylock(&rwl->m))

>>> return EBUSY;

>>> if (rwl->writers == 0) /* no writer, pass */

>>> rwl->readers++;

>>> else {

>>> rwl->br++;

>>> cv_wait(&rwl->qr, &rwl->m);

> ^^^^^

>>

>> That we can't do. That's exactly the thing the existing sx(9) implementation

>> does and where it breaks. The problem is that cv_wait() is an implicit sleep

>> which breaks when we try to RLOCK() with other mutex already acquired.

>

> but that is not a solvable problem given that the *LOCK may be blocking.

> And the cv_wait is not an unconditioned sleep, it is one where you release

> the lock right before ans wait for an event to wake you up.

- > *In fact i don't understand why you consider spinning and sleeping*
- > *on a mutex two different things.*

The major difference between sleeping (cv_wait,msleep,..) and blocking on a mutex is priority inheritance.

If you need to be able to use (non-spin) mutexes while holding a [R|W]LOCK and use a [R|W]LOCK while holding a (non-spin) mutex then you need to implement priority inheritance for [R|W]LOCKS.

For the (single) write lock holder tracking priority is easy. (just like a (non-spin) mutex). However priority inheritance to multiple readers is more difficult as one needs to keep track of all holders of the lock.

Keeping track of all readers requires pre-allocated memory resources.

This memory could come from

- 1) A limited global pool
- 2) A limited per [R|W]LOCK pool
- 3) A limited per thread pool
- 4) As a parameter for acquiring a RLOCK

None of the choices are really pretty.

(1),(2) and (3) can lead to limiting reader parallelism when running out of resources. (4) may be practically for some cases since the memory could be allocated from stack. However since the memory must be valid while holding a read lock choice (4) makes some algorithms (than use for example lock crabbing) a bit harder to implement.

> > *Moreover will this break for recursive reads e.g.:*

> >

> > *Thread 1: RLOCK() ... RLOCK() -> cv_wait ...*

> > *Thread 2: WLOCK() -> cv_wait ...*

> >

> > *This is exactly what pfil_hooks must be able to do as the packet filter may*

> > *want to call back to the stack in order to send rejects etc.*

>

> *that's another story (also in issue in ipfw) and the way it is addressed*

> *elsewhere is by releasing and reaquiring the lock. In fact is the topic*

> *that started this thread.*

>

> > *change). The problem is, that we still have to invest 4 mutex operations for*

> > *every access. The current implementation has the same basic problem (though*

> > *it only uses 2 mutex operations for the WLOCK/UNLOCK). Ideally the RLOCK/*

> > *UNLOCK should be free unless there is a writer waiting for the lock. In*

>

> *"free unless" means not free – you always have to check, be it through*

> *an atomic cmpswap or something else. But this is what mtx_lock does*

> *anyways in the fast path.*

>

> *cheers*

> *luigi*

>

freebsd-arch: Re: duplicate read/write locks in net/pfil.c and netinet/ip_fw2.c

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

To unsubscribe, send any mail to "freebsd-arch-unsubscribe@freebsd.org"