

Re: rwlocks, correctness over speed.

## Re: rwlocks, correctness over speed.

---

*Source:* <http://unix.derkeiler.com/Mailing-Lists/FreeBSD/arch/2007-11/msg00090.html>

---

- *From:* Max Laier <max@xxxxxxxxxxxxxxxx>
  - *Date:* Fri, 23 Nov 2007 12:28:33 +0100
- 

On Friday 23 November 2007, Alfred Perlstein wrote:

- \* Stephan Uphoff <ups@xxxxxxxx> [071123 00:46] wrote:

Alfred Perlstein wrote:

- \* Max Laier <max@xxxxxxxxxxxxxxxx> [071122 14:40] wrote:

On Thursday 22 November 2007, Attilio Rao wrote:

2007/11/22, Max Laier  
<max@xxxxxxxxxxxxxxxx>:

rwlocks are  
already  
used in  
places that  
do recursive  
reads. The  
one place  
I'm certain  
about is  
pfil(9) and  
we need a  
proper  
sollution for  
this. Before  
rwlocks  
were used, I  
had a  
handrolled  
locking that  
supported  
both  
read/write  
semantics  
and

Re: rwlocks, correctness over speed.

starvation  
avoidance –  
at the cost  
of failing to  
allow further  
read access  
when a  
writer asked  
for access.  
This  
however,  
was quite  
application  
specific and  
not the most  
efficient  
implementation  
either.

I'm not a pfil(9) expert, but  
for what I've heard, rwlocks  
should  
be really good for it,  
shouldn't them?

The concept is that if we  
want to maintain fast paths  
for rwlock  
we cannot do too much  
tricks there. And you can  
really deadlock  
if you allow recursion on  
readers...

How about adding `rwlock_try_rlock()` which  
would do the following:

- 1) Only variant to allow[1] read recursion  
and ...
- 2) ... only if no outstanding write requests
- 3) Let the caller deal with failure

This can be implemented statically, so no  
overhead in the fast  
path. The caller is in the best position to  
decide if it is  
recursing or not – could keep that info on the  
stack – and can  
either fail[2] or do a normal `rwlock_rlock()`  
which would wait for

Re: rwlocks, correctness over speed.

the writer to enter and exit.

[2] In most situation where you use read locks you can fail or roll back carefully as you didn't change anything – obviously. In pfil – for instance – we just dropped the packet if there was a writer waiting.

[1] "allow" in terms of WITNESS – if that can be done.

The problem is that there is no tracking in the common case (without additional overhead), so you can't know if you're recursing, unless you track it yourself.

–Alfred

I talked with Attilio about that on IRC.

Most common cases of writer starvation (but not all) could be solved by keeping a per thread count of shared acquired rwlocks.

If a rwlock is currently locked as shared/read AND a thread is blocked on it to lock it exclusively/write – then new shared/read locks will only be granted to thread that already has a shared lock. (per thread shared counter is non zero)

To be honest I am a bit twitchy about a lock without priority propagation – especially since in FreeBSD threads run with user priority in kernel space and can get preempted.

Stephan

That's an interesting hack, I guess it could be done.

I would still like to disallow recursion.

Can we come to a concensus on that?

As long as we properly take care of all the fallout before doing so, I'm all for it.

Attached is a diff to switch pfil over to rmlocks which works for me. I'll post it to –net and do the switch in a few days unless I hear

Re: rwlocks, correctness over speed.

Re: rwlocks, correctness over speed.

otherwise.

--

/"\ Best regards, | mlaier@xxxxxxxxxxxx  
\ / Max Laier | ICQ #67774661  
X <http://pf4freebsd.love2party.net/> | mlaier@EFnet  
\ / ASCII Ribbon Campaign | Against HTML Mail and News  
Index: pfil.c

```
=====
RCS file: /home/ncvs/src/sys/net/pfil.c,v
retrieving revision 1.14
diff -u -r1.14 pfil.c
--- pfil.c 2 Feb 2006 03:13:15 -0000 1.14
+++ pfil.c 23 Nov 2007 09:06:12 -0000
@@ -34,7 +34,7 @@
#include <sys/errno.h>
#include <sys/lock.h>
#include <sys/malloc.h>
-#include <sys/rwlock.h>
+#include <sys/rmlock.h>
#include <sys/socket.h>
#include <sys/socketvar.h>
#include <sys/system.h>
@@ -66,11 +66,12 @@
 pfil_run_hooks(struct pfil_head *ph, struct mbuf **mp, struct ifnet *ifp,
 int dir, struct inpcb *inp)
{
+ struct rm_priotracker rmpt;
struct packet_filter_hook *pfh;
struct mbuf *m = *mp;
int rv = 0;

- PFIL_RLOCK(ph);
+ PFIL_RLOCK(ph, &rmpt);
KASSERT(ph->ph_nhooks >= 0, ("Pfil hook count dropped < 0"));
for (pfh = pfil_hook_get(dir, ph); pfh != NULL;
pfh = TAILQ_NEXT(pfh, pfil_link)) {
@@ -80,7 +81,7 @@
break;
}
}
- PFIL_RUNLOCK(ph);
+ PFIL_RUNLOCK(ph, &rmpt);

*mp = m;
return (rv);
@@ -104,7 +105,7 @@
}
PFIL_LIST_UNLOCK();

- rw_init(&ph->ph_mtx, "Pfil hook read/write mutex");
```

Re: rwlocks, correctness over speed.

Re: rwlocks, correctness over speed.

```
+ PFIL_LOCK_INIT(ph);
PFIL_WLOCK(ph);
ph->ph_nhooks = 0;
```

```
@@ -143,7 +144,7 @@
free(pfh, M_IFADDR);
TAILQ_FOREACH_SAFE(pfh, &ph->ph_out, pfil_link, pfnext)
free(pfh, M_IFADDR);
- rw_destroy(&ph->ph_mtx);
+ PFIL_LOCK_DESTROY(ph);
```

```
return (0);
}
Index: pfil.h
```

```
=====
RCS file: /home/ncvs/src/sys/net/pfil.h,v
retrieving revision 1.16
diff -u -r1.16 pfil.h
--- pfil.h 8 Jun 2007 12:43:25 -0000 1.16
+++ pfil.h 23 Nov 2007 10:07:52 -0000
@@ -37,7 +37,7 @@
#include <sys/_lock.h>
#include <sys/_mutex.h>
#include <sys/lock.h>
-#include <sys/rwlock.h>
+#include <sys/rmlock.h>
```

```
struct mbuf;
struct ifnet;
@@ -69,7 +69,7 @@
pfil_list_t ph_out;
int ph_type;
int ph_nhooks;
- struct rwlock ph_mtx;
+ struct rmlock ph_lock;
union {
u_long phu_val;
void *phu_ptr;
@@ -93,10 +93,13 @@
struct pfil_head *pfil_head_get(int, u_long);
```

```
#define PFIL_HOOKED(p) ((p)->ph_nhooks > 0)
-#define PFIL_RLOCK(p) rw_rlock(&(p)->ph_mtx)
-#define PFIL_WLOCK(p) rw_wlock(&(p)->ph_mtx)
-#define PFIL_RUNLOCK(p) rw_runlock(&(p)->ph_mtx)
-#define PFIL_WUNLOCK(p) rw_wunlock(&(p)->ph_mtx)
+#define PFIL_LOCK_INIT(p) \
+ rm_init(&(p)->ph_lock, "Pfil hook read/write mutex", LO_RECURSABLE)
+#define PFIL_LOCK_DESTROY(p) rm_destroy(&(p)->ph_lock)
+#define PFIL_RLOCK(p, t) rm_rlock(&(p)->ph_lock, (t))
+#define PFIL_WLOCK(p) rm_wlock(&(p)->ph_lock)
```

Re: rwlocks, correctness over speed.

Re: rwlocks, correctness over speed.

```
+#define PFIL_RUNLOCK(p, t) rm_runlock(&(p)->ph_lock, (t))  
+#define PFIL_WUNLOCK(p) rm_wunlock(&(p)->ph_lock)  
#define PFIL_LIST_LOCK() mtx_lock(&pfil_global_lock)  
#define PFIL_LIST_UNLOCK() mtx_unlock(&pfil_global_lock)
```

***Attachment: signature.asc***

*Description:* This is a digitally signed message part.