rtnl mutex, the network stack big kernel lock

Red Hat

Florian Westphal
4096R/AD5FF600 fw@strlen.de
80A9 20C5 B203 E069 F586
AE9F 7091 A8D9 AD5F F600

netdev 2.2, Seoul, November 2017

Agenda

1. Intro: What is rtnetlink?
2. rtnetlink then and now
3. problems and challenges

What is rtnetlink?

- kernels network configuration interface
- ancient by kernel standards: rtnetlink.c added 20 years ago
- `CONFIG_RTNETLINK` removed in 2001 (always enabled ever since)
- used by almost everything related to network configuration
  - ipv4, ipv6, can, decnet, bridge, mpls, . . .
  - adding/removing interfaces, tunnels, neigh entries, ip addresses, ipv6 address labels, routes, qdiscs, . . .

```
rtnl_register(PF_INET, RTM_NEWROUTE, inet_rtm_newroute,
              NULL);
...
void rtnl_register(int protocol, int msgtype,
                   rtnl_doit_func, rtnl_dumpit_func);
```

rtnetlink in Linux 4.13

```
static void rtnetlink_rcv(struct sk_buff *skb)
{
    rtnl_lock();
    netlink_rcv_skb(skb, &rtnetlink_rcv_msg);
    rtnl_unlock();
}
```

- `rtnetlink_rcv_msg` decodes request (contains family/type), then invokes doit or dumpit callback
- callbacks decode/validate netlink messages and perform desired action

What is rtnl mutex used for?

1. serializes all rtnetlink requests
2. serializes with other userspace apis (sysfs, ioctl, ...) to network configuration
3. protects list of net namespaces

As a consequence:

- one request at a time, e.g. adding ip address must wait for user listing interface properties
- dump requests (fib, tc classifier list, interfaces)... are also serialized

`rtnl_mutex` can be held for very long times:

- `schedule()` (incl. `GFP_KERNEL` allocations)
- `synchronize_rcu(_net)`

rtnetlink: caveats

- callbacks rely on rtnl mutex being held
- `rtnl_lock` guarantees consistency during a dump
- can't blindly avoid rtnl mutex
- allow to annotate handler: `RTNL_DOIT_UNLOCKED`
- then start to push `rtnl_lock` down

rtnetlink in Linux 4.14

```
static void rtnetlink_rcv(struct sk_buff *skb)
{
   netlink_rcv_skb(skb, &rtnetlink_rcv_msg);
}

rtnetlink_rcv_msg():
 flags = handlers[type].flags;
 doit = handlers[type].doit;
 if (flags & RTNL_FLAG_DOIT_UNLOCKED)
      return doit(skb, nlh, extack);

 rtnl_lock();
 err = doit(skb, nlh, extack);
 rtnl_unlock();
 return err;
```

converting users

- a few low-hanging fruits: RTM_GETROUTE, ipv6 address labels
- handlers that don't change anything or use different lock internally
- main problem: even if handler doesn't modify anything it still needs to provide consistent data
- link ops, af ops: depend on RTNL mutex
- other places that make assumptions on rtnl presence (e.g. for upper/lower device in stacked setups)

```
rtnl_fill_ifinfo:
 if (nla_put_string(skb, IFLA_IFNAME, dev->name) ||
     nla_put_u32(skb, IFLA_TXQLEN, dev->tx_queue_len)
```

e.g. don't want to return garbled name to userspace
How to guarantee consistency without RTNL mutex?

converting users (2): rtnl af ops

- address family specific operations
- only a few instances of these exist
- no callback implementation needs to sleep → convert to rcu
- patch is straightforward
- no advantage – still locked via rtnl
- but needed to make more rtnl pushdowns possible

converting users (3): rtnl link ops

- link specific operations
- lots of instances
- at least some callbacks depend on rtnl
- need a way to prevent module unload/link ops removal while callback is active
- "standard solution": `.owner = THIS_MODULE;`
- however, turns out nothing needs to be done at all, provided doit callback either
  1. acquires RTNL mutex, or
  2. takes reference count of the device that the `link_ops` are assigned to, or
  3. uses rcu read lock + `dev_get_by_index_rcu`
- ... because link op unregister removes all affected devices (refcount must drop to 0)

general problems

- lot of call paths, large amount of code (netdev ops!)
- e.g., ”can i call `netdev_ops->ndo_fdb_add()` without mutex”?
  - `dev_get_phys_port_name()`?
  - `dev_num_vf()`?
  - `ndo_get_vf_port()`?
- not just because of races:
  - module removal
  - parallel changes create new problems
  - not-so-obvious dependencies, netdev notifiers in particular

problems (2): devinet

- ip address assignment, among other things
- also has legacy ioctl based interface
- handlers acquire RTNL mutex to serialize requests
- when a new address is assigned, a notifier call chain gets invoked
- allows in-kernel users (e.g. ipvlan) to **veto** the new address
- requires serialization vs. other address changes in same family

problems (3): IP FIB

- again rtnetlink, again RTNL mutex
- FIB lookups already rcu safe
- replace RTNL mutex with new FIB mutex?
  - creates potential for ABBA deadlocks
  - so only feasible if strict ordering is guaranteed
  - common add/delete ops should only grab new FIB mutex
- FIB changes also occur indirectly by kernel (e.g. device link state change)
  - notifiers are called with rtnl mutex already held
  - so we now acquire new FIB mutex while also holding RTNL one
  - . . . acquiring RTNL mutex while holding FIB mutex would deadlock
- second issue: dump consistency checks

problems (4): IP FIB (continued)

- netlink dumps can be large
- can span multiple messages, i.e. dump request → read(), read(), read(), ..
- locks have to be dropped before returning to userspace
- dumps can thus be inconsistent if changes happen in between
- → `NLM_F_DUMP_INTR` flag set in that case
- fib notifier increments a counter, if counter changed at end of dump: inconsistent result

problems (5): IP FIB (continued)

- can't just make counter `atomic_t`, consider:
  1. A: a new FIB entry gets added
  2. B: a dump request starts, fetches current counter
  3. A: the new FIB entry is linked into the list
  4. B: the dump request finishes, fetches counter
  5. A: `call_fib4_notifiers()` is invoked and increments the sequence counter
  6. B: dump appears consistent
- possible way out: `seqcount_t`

problems (6): lockless dumps

- was already tried a few years back
- large parts of rtnl dump functions make mutex assumptions
  - qdisc info – we would crash if other cpu replaces qdisc while another dumps it
  - xdp information
  - SR-IOV information
  - link stats

Summary

- network config path has many dependencies, e.g. via notifiers
- makes it hard to remove rtnl locking
- initial work completed
  - handlers can indicate they do not need rtnl mutex
  - a few simple handlers do so, e.g. `ip route get ..`
- current focus: no rtnl mutex when dumping

Any questions?