Linux kernel client - Bug #1795

break d_lock > s_cap_lock ordering

12/07/2011 08:31 AM - Sage Weil

Status:	Resolved	**Start date:**	12/07/2011
Priority:	Normal	**Due date:**	
Assignee:	Alex Elder	**% Done:**	0%
Category:	fs/ceph	**Estimated time:**	0.00 hour
Target version:	v3.3	**Spent time:**	0.00 hour
Source:		**Reviewed:**	
Tags:		**Affected Versions:**	
Backport:		**ceph-qa-suite:**	
Regression:	No	**Crash signature:**	
Severity:	3 - minor		

Description

```
Dec  7 15:06:27 tgpro1 kernel: =======================================================
Dec  7 15:06:27 tgpro1 kernel: [ INFO: possible circular locking dependency detected ]
Dec  7 15:06:27 tgpro1 kernel: -------------------------------------------------------
Dec  7 15:06:27 tgpro1 kernel: kworker/4:2/3143 is trying to acquire lock:
Dec  7 15:06:27 tgpro1 kernel:  (&sb->s_type->i_lock_key#16){+.+...}, at: [<000ff721>] igrab+0x11/
0x41
Dec  7 15:06:27 tgpro1 kernel:
Dec  7 15:06:27 tgpro1 kernel: but task is already holding lock:
Dec  7 15:06:27 tgpro1 kernel:  (&(&s->s_cap_lock)->rlock){+.+...}, at: [<006d0dd0>] iterate_sessi
on_caps+0x40/0x17d [ceph]
Dec  7 15:06:27 tgpro1 kernel:
Dec  7 15:06:27 tgpro1 kernel: which lock already depends on the new lock.
Dec  7 15:06:27 tgpro1 kernel:
Dec  7 15:06:27 tgpro1 kernel:
Dec  7 15:06:27 tgpro1 kernel: the existing dependency chain (in reverse order) is:
Dec  7 15:06:27 tgpro1 kernel:
Dec  7 15:06:27 tgpro1 kernel: -> #2 (&(&s->s_cap_lock)->rlock){+.+...}:
Dec  7 15:06:27 tgpro1 kernel:        [<00091456>] lock_acquire+0x42/0x59
Dec  7 15:06:27 tgpro1 kernel:        [<0053f187>] _raw_spin_lock+0x24/0x33
Dec  7 15:06:27 tgpro1 kernel:        [<006c180f>] ceph_d_revalidate+0xf4/0x29e [ceph]
Dec  7 15:06:27 tgpro1 kernel:        [<000f3339>] do_lookup+0x223/0x2a6
Dec  7 15:06:27 tgpro1 kernel:        [<000f4d99>] path_lookupat+0x118/0x6e9
Dec  7 15:06:27 tgpro1 kernel:        [<000f5386>] do_path_lookup+0x1c/0x4e
Dec  7 15:06:27 tgpro1 kernel:        [<000f544c>] user_path_at_empty+0x3e/0x69
Dec  7 15:06:27 tgpro1 kernel:        [<000f5484>] user_path_at+0xd/0xf
Dec  7 15:06:27 tgpro1 kernel:        [<000ecf98>] vfs_fstatat+0x40/0x67
Dec  7 15:06:27 tgpro1 kernel:        [<000ed003>] vfs_lstat+0x16/0x18
Dec  7 15:06:27 tgpro1 kernel:        [<000ed019>] sys_lstat64+0x14/0x28
Dec  7 15:06:27 tgpro1 kernel:        [<0053f9ca>] syscall_call+0x7/0xb
Dec  7 15:06:27 tgpro1 kernel:
Dec  7 15:06:27 tgpro1 kernel: -> #1 (&(&dentry->d_lock)->rlock){+.+.-.}:
Dec  7 15:06:27 tgpro1 kernel:        [<00091456>] lock_acquire+0x42/0x59
Dec  7 15:06:27 tgpro1 kernel:        [<0053f187>] _raw_spin_lock+0x24/0x33
Dec  7 15:06:27 tgpro1 kernel:        [<000fca43>] __d_instantiate+0x14/0xbc
Dec  7 15:06:27 tgpro1 kernel:        [<000fcf4e>] d_instantiate+0x30/0x40
Dec  7 15:06:27 tgpro1 kernel:        [<000fe357>] d_alloc_root+0x27/0x2d
Dec  7 15:06:27 tgpro1 kernel:        [<006bd2e2>] open_root_dentry+0xfd/0x143 [ceph]
Dec  7 15:06:27 tgpro1 kernel:        [<006bde68>] ceph_mount+0x338/0x4c4 [ceph]
Dec  7 15:06:27 tgpro1 kernel:        [<000eaac4>] mount_fs+0xe/0x95
Dec  7 15:06:27 tgpro1 kernel:        [<00103266>] vfs_kern_mount+0x4c/0x80
Dec  7 15:06:27 tgpro1 kernel:        [<001046aa>] do_kern_mount+0x2f/0xaf
Dec  7 15:06:27 tgpro1 kernel:        [<001051f0>] do_mount+0x26b/0x2aa
Dec  7 15:06:27 tgpro1 kernel:        [<00105290>] sys_mount+0x61/0x94
```

```
Dec  7 15:06:27 tgpro1 kernel:        [<0053f9ca>] syscall_call+0x7/0xb
Dec  7 15:06:27 tgpro1 kernel:
Dec  7 15:06:27 tgpro1 kernel: -> #0 (&sb->s_type->i_lock_key#16){+.+...}:
Dec  7 15:06:27 tgpro1 kernel:        [<00090b06>] __lock_acquire+0xe7b/0x1428
Dec  7 15:06:27 tgpro1 kernel:        [<00091456>] lock_acquire+0x42/0x59
Dec  7 15:06:27 tgpro1 kernel:        [<0053f187>] _raw_spin_lock+0x24/0x33
Dec  7 15:06:27 tgpro1 kernel:        [<000ff721>] igrab+0x11/0x41
Dec  7 15:06:27 tgpro1 kernel:        [<006d0e00>] iterate_session_caps+0x70/0x17d [ceph]
Dec  7 15:06:27 tgpro1 kernel:        [<006d3bb4>] send_mds_reconnect+0x319/0x46a [ceph]
Dec  7 15:06:27 tgpro1 kernel:        [<006d401f>] ceph_mdsc_handle_map+0x2f2/0x42d [ceph]
Dec  7 15:06:27 tgpro1 kernel:        [<006be00c>] extra_mon_dispatch+0x18/0x1c [ceph]
Dec  7 15:06:27 tgpro1 kernel:        [<006963ed>] dispatch+0x4f7/0x52a [libceph]
Dec  7 15:06:27 tgpro1 kernel:        [<00694459>] con_work+0x14f4/0x16c3 [libceph]
Dec  7 15:06:27 tgpro1 kernel:        [<0007b331>] process_one_work+0x229/0x397
Dec  7 15:06:27 tgpro1 kernel:        [<0007b80e>] worker_thread+0x182/0x2d8
Dec  7 15:06:27 tgpro1 kernel:        [<0007e843>] kthread+0x62/0x67
Dec  7 15:06:27 tgpro1 kernel:        [<00540c86>] kernel_thread_helper+0x6/0xd
Dec  7 15:06:27 tgpro1 kernel:
Dec  7 15:06:27 tgpro1 kernel: other info that might help us debug this:
Dec  7 15:06:27 tgpro1 kernel:
Dec  7 15:06:27 tgpro1 kernel: Chain exists of:
Dec  7 15:06:27 tgpro1 kernel:   &sb->s_type->i_lock_key --> &(&dentry->d_lock)->rlock --> &(&s->s
_cap_lock)->rlock
Dec  7 15:06:27 tgpro1 kernel:
Dec  7 15:06:27 tgpro1 kernel:  Possible unsafe locking scenario:
Dec  7 15:06:27 tgpro1 kernel:
Dec  7 15:06:27 tgpro1 kernel:        CPU0                    CPU1
Dec  7 15:06:27 tgpro1 kernel:        ----                    ----
Dec  7 15:06:27 tgpro1 kernel:   lock(&(&s->s_cap_lock)->rlock);
Dec  7 15:06:27 tgpro1 kernel:                                lock(&(&dentry->d_lock)->rlock);
Dec  7 15:06:27 tgpro1 kernel:                                lock(&(&s->s_cap_lock)->rlock);
Dec  7 15:06:27 tgpro1 kernel:   lock(&sb->s_type->i_lock_key);
Dec  7 15:06:27 tgpro1 kernel:
Dec  7 15:06:27 tgpro1 kernel:  *** DEADLOCK ***
Dec  7 15:06:27 tgpro1 kernel:
Dec  7 15:06:27 tgpro1 kernel: 5 locks held by kworker/4:2/3143:
Dec  7 15:06:27 tgpro1 kernel:  #0:  (ceph-msgr){++++.+}, at: [<0007b2e5>] process_one_work+0x1dd/
0x397
Dec  7 15:06:27 tgpro1 kernel:  #1:  ((&(&con->work)->work)){+.+.+.}, at: [<0007b2e5>] process_one
_work+0x1dd/0x397
Dec  7 15:06:27 tgpro1 kernel:  #2:  (&s->s_mutex){+.+.+.}, at: [<006d393e>] send_mds_reconnect+0x
a3/0x46a [ceph]
Dec  7 15:06:27 tgpro1 kernel:  #3:  (&mdsc->snap_rwsem){+++++.}, at: [<006d3a36>] send_mds_reconn
ect+0x19b/0x46a [ceph]
Dec  7 15:06:27 tgpro1 kernel:  #4:  (&(&s->s_cap_lock)->rlock){+.+...}, at: [<006d0dd0>] iterate_
session_caps+0x40/0x17d [ceph]
Dec  7 15:06:27 tgpro1 kernel:
Dec  7 15:06:27 tgpro1 kernel: stack backtrace:

Dec  7 15:06:27 tgpro1 kernel: Call Trace:
Dec  7 15:06:27 tgpro1 kernel:  [<0008f840>] print_circular_bug+0x219/0x226
Dec  7 15:06:27 tgpro1 kernel:  [<00090b06>] __lock_acquire+0xe7b/0x1428
Dec  7 15:06:27 tgpro1 kernel:  [<00091456>] lock_acquire+0x42/0x59
Dec  7 15:06:27 tgpro1 kernel:  [<000ff721>] ? igrab+0x11/0x41
Dec  7 15:06:27 tgpro1 kernel:  [<0053f187>] _raw_spin_lock+0x24/0x33
Dec  7 15:06:27 tgpro1 kernel:  [<000ff721>] ? igrab+0x11/0x41
Dec  7 15:06:27 tgpro1 kernel:  [<000ff721>] igrab+0x11/0x41
Dec  7 15:06:27 tgpro1 kernel:  [<006d0e00>] iterate_session_caps+0x70/0x17d [ceph]
Dec  7 15:06:27 tgpro1 kernel:  [<006d5475>] ? ceph_mdsc_submit_request+0x55/0x55 [ceph]
Dec  7 15:06:27 tgpro1 kernel:  [<00694d13>] ? ceph_pagelist_append+0xbc/0xf9 [libceph]
Dec  7 15:06:27 tgpro1 kernel:  [<006d3bb4>] send_mds_reconnect+0x319/0x46a [ceph]
Dec  7 15:06:27 tgpro1 kernel:  [<0008eb6f>] ? trace_hardirqs_on_caller+0x10b/0x13c
Dec  7 15:06:27 tgpro1 kernel:  [<0008ebab>] ? trace_hardirqs_on+0xb/0xd
Dec  7 15:06:27 tgpro1 kernel:  [<006d401f>] ceph_mdsc_handle_map+0x2f2/0x42d [ceph]
Dec  7 15:06:27 tgpro1 kernel:  [<00c40015>] ? 0xc40014
Dec  7 15:06:27 tgpro1 kernel:  [<006be00c>] extra_mon_dispatch+0x18/0x1c [ceph]
Dec  7 15:06:27 tgpro1 kernel:  [<006963ed>] dispatch+0x4f7/0x52a [libceph]
```

```
Dec  7 15:06:27 tgpro1 kernel:  [<0008ebab>] ? trace_hardirqs_on+0xb/0xd
Dec  7 15:06:27 tgpro1 kernel:  [<00694459>] con_work+0x14f4/0x16c3 [libceph]
Dec  7 15:06:27 tgpro1 kernel:  [<0007b331>] process_one_work+0x229/0x397
Dec  7 15:06:27 tgpro1 kernel:  [<0007b2e5>] ? process_one_work+0x1dd/0x397
Dec  7 15:06:27 tgpro1 kernel:  [<00692f65>] ? ceph_fault+0x262/0x262 [libceph]
Dec  7 15:06:27 tgpro1 kernel:  [<0007b80e>] worker_thread+0x182/0x2d8
Dec  7 15:06:27 tgpro1 kernel:  [<00002b40>] ? setup_sigcontext+0x24/0x24
Dec  7 15:06:27 tgpro1 kernel:  [<00002b40>] ? setup_sigcontext+0x24/0x24
Dec  7 15:06:27 tgpro1 kernel:  [<0007b68c>] ? rescuer_thread+0x1ed/0x1ed
Dec  7 15:06:27 tgpro1 kernel:  [<0007e843>] kthread+0x62/0x67
Dec  7 15:06:27 tgpro1 kernel:  [<0007e7e1>] ? __init_kthread_worker+0x42/0x42
Dec  7 15:06:27 tgpro1 kernel:  [<00540c86>] kernel_thread_helper+0x6/0xd
```

History

#1 - 12/12/2011 03:19 AM - Amon Ott

Seems fixed here now with git branch wip-d-lock.

#2 - 01/03/2012 10:48 AM - Sage Weil

- Target version deleted (v3.2)

#3 - 01/03/2012 10:52 AM - Sage Weil

- translation missing: en.field_position set to 1

#4 - 01/03/2012 11:14 AM - Sage Weil

- Target version set to v3.3

- translation missing: en.field_position deleted (3)

- translation missing: en.field_position set to 1

- translation missing: en.field_position changed from 1 to 699

#5 - 01/12/2012 04:25 PM - Alex Elder

- Status changed from New to In Progress

Discussed this with Sage. The problem arose because dentry_lease_is_valid() is using the MDS session's s_cap_lock for protecting the atomic update of s_cap_gen and s_cap_ttl. But it must do so while holding the dentry lock, and that violates the lock ordering: s_cap_lock -> i_lock -> d_lock

Will introduce a new inner lock, d_lock -> s_gen_lock, which will be used to protect only atomic updates of s_cap_gen and s_cap_ttl.

#6 - 01/13/2012 11:54 AM - Alex Elder

I have posted a series of four proposed patches to the list to address this, along with a few other issues identified while looking at this.

The first one introduces a new spinlock, s_gen_ttl_lock, which is used instead of s_cap_lock to ensure the two fields are updated or sampled atomically where this is needed.

The second replaces the use of the special value 0 for the ttl field to indicate "some time in the past," instead using (jiffies - 1).

The last two change the types for these two fields to atomic types to guarantee operations on each is atomic.

Will commit to master once they're reviewed.

#7 - 01/13/2012 02:28 PM - Alex Elder

OK, after several iterations and some discussion we concluded the last two patches (turning these things into atomics) were not actually needed.

#8 - 01/13/2012 03:53 PM - Sage Weil

- Assignee set to Alex Elder

#9 - 01/13/2012 07:14 PM - Alex Elder

I was hitting some odd behavior while testing. Will try again over the weekend or early next week.

Also, a note: Sage said this warning can be triggered by causing an MDS restart. Lockdep doesn't work for UML builds so it has to be done in a "real" kernel. It's possible to script this in teuthology, but maybe not obvious.

#10 - 01/18/2012 05:57 AM - Alex Elder

- Status changed from In Progress to Resolved

I have verified under UML that no new problems arise with the fix in place. I have not verified the lockdep warnings are addressed, but I have pretty good confidence this has been addressed. The fix is committed to master.

Marking this resolved.