

---

Subject: [PATCH] Memory shortage can result in inconsistent flocks state

Posted by [Pavel Emelianov](#) on Tue, 11 Sep 2007 12:38:13 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

When the flock\_lock\_file() is called to change the flock from F\_RDLCK to F\_WRLCK or vice versa the existing flock can be removed without appropriate warning.

Look:

```
for_each_lock(inode, before) {
    struct file_lock *fl = *before;
    if (IS_POSIX(fl))
        break;
    if (IS_LEASE(fl))
        continue;
    if (filp != fl->fl_file)
        continue;
    if (request->fl_type == fl->fl_type)
        goto out;
    found = 1;
    locks_delete_lock(before); <<<<<< !
    break;
}
```

if after this point the subsequent locks\_alloc\_lock() will fail the return code will be -ENOMEM, but the existing lock is already removed.

This is a known feature that such "re-locking" is not atomic, but in the racy case the file should stay locked (although by some other process), but in this case the file will be unlocked.

The proposal is to prepare the lock in advance keeping no chance to fail in the future code.

Found during making the flocks pid-namespaces aware.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

---

```
diff --git a/fs/locks.c b/fs/locks.c
index 0db1a14..f59d066 100644
--- a/fs/locks.c
+++ b/fs/locks.c
@@ -732,6 +732,14 @@ static int flock_lock_file(struct file *
    lock_kernel();
    if (request->fl_flags & FL_ACCESS)
```

```

    goto find_conflict;
+
+ if (request->fl_type != F_UNLCK) {
+ error = -ENOMEM;
+ new_fl = locks_alloc_lock();
+ if (new_fl == NULL)
+ goto out;
+ }
+
+ for_each_lock(inode, before) {
+ struct file_lock *fl = *before;
+ if (IS_POSIX(fl))
@@ -753,10 +761,6 @@ static int flock_lock_file(struct file *
+ goto out;
+ }

- error = -ENOMEM;
- new_fl = locks_alloc_lock();
- if (new_fl == NULL)
- goto out;
/*
* If a higher-priority process was blocked on the old file lock,
* give it the opportunity to lock the file.

```

---

Subject: Re: [PATCH] Memory shortage can result in inconsistent flocks state

Posted by [bfields](#) on Wed, 12 Sep 2007 19:06:53 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

On Tue, Sep 11, 2007 at 04:38:13PM +0400, Pavel Emelyanov wrote:

> This is a known feature that such "re-locking" is not atomic,  
> but in the racy case the file should stay locked (although by  
> some other process), but in this case the file will be unlocked.

That's a little subtle (I assume you've never seen this actually  
happen?), but it makes sense to me.

> The proposal is to prepare the lock in advance keeping no chance  
> to fail in the future code.

And the patch certainly looks correct.

I can add it to my (trivial) lock patches, if that's helpful--it'll  
get folded into the branch -mm pulls from and I can pass it along to  
Linus for 2.6.24.

What I don't have that I wish I did is good regression tests for the  
flock or lease code (for posix locks I've been using connectathon,

though that misses some important things too).

--b.

---

---

Subject: Re: [PATCH] Memory shortage can result in inconsistent flocks state  
Posted by [Pavel Emelianov](#) on Thu, 13 Sep 2007 06:04:16 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

J. Bruce Fields wrote:

> On Tue, Sep 11, 2007 at 04:38:13PM +0400, Pavel Emelyanov wrote:

>> This is a known feature that such "re-locking" is not atomic,  
>> but in the racy case the file should stay locked (although by  
>> some other process), but in this case the file will be unlocked.

>

> That's a little subtle (I assume you've never seen this actually  
> happen?), but it makes sense to me.

Well, this situation is hard to notice since usually programs try to finish up when some error is returned from the kernel, but I do believe that this could happen in one of the openvz kernels since we limit the kernel memory usage for "containers" and thus -ENOMEM is a common error.

>> The proposal is to prepare the lock in advance keeping no chance  
>> to fail in the future code.

>

> And the patch certainly looks correct.

>

> I can add it to my (trivial) lock patches, if that's helpful--it'll  
> get folded into the branch -mm pulls from and I can pass it along to  
> Linus for 2.6.24.

Thanks.

> What I don't have that I wish I did is good regression tests for the  
> flock or lease code (for posix locks I've been using connectathon,  
> though that misses some important things too).

>

> --b.

>

---

---

Subject: Re: [PATCH] Memory shortage can result in inconsistent flocks state  
Posted by [Balbir Singh](#) on Thu, 13 Sep 2007 07:16:27 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

On 9/13/07, Pavel Emelyanov <xemul@openvz.org> wrote:  
> J. Bruce Fields wrote:  
>> On Tue, Sep 11, 2007 at 04:38:13PM +0400, Pavel Emelyanov wrote:  
>>> This is a known feature that such "re-locking" is not atomic,  
>>> but in the racy case the file should stay locked (although by  
>>> some other process), but in this case the file will be unlocked.  
>>  
>> That's a little subtle (I assume you've never seen this actually  
>> happen?), but it makes sense to me.  
>  
> Well, this situation is hard to notice since usually programs  
> try to finish up when some error is returned from the kernel,  
> but I do believe that this could happen in one of the openvz  
> kernels since we limit the kernel memory usage for "containers"  
> and thus -ENOMEM is a common error.  
>

The fault injection framework should be able to introduce the same error. Of course hitting the error would require careful setup of the fault parameters.

Balbir

---

---

Subject: Re: [PATCH] Memory shortage can result in inconsistent flocks state  
Posted by [Chuck Ebbert](#) on Thu, 13 Sep 2007 19:27:08 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On 09/11/2007 08:38 AM, Pavel Emelyanov wrote:  
> When the flock\_lock\_file() is called to change the flock  
> from F\_RDLCK to F\_WRLCK or vice versa the existing flock  
> can be removed without appropriate warning.  
>  
> Look:  
>     for\_each\_lock(inode, before) {  
>         struct file\_lock \*fl = \*before;  
>         if (IS\_POSIX(fl))  
>             break;  
>         if (IS\_LEASE(fl))  
>             continue;  
>         if (filp != fl->fl\_file)  
>             continue;  
>         if (request->fl\_type == fl->fl\_type)  
>             goto out;  
>         found = 1;  
>         locks\_delete\_lock(before); <<<<<< !  
>         break;  
>     }

```

>
> if after this point the subsequent locks_alloc_lock() will
> fail the return code will be -ENOMEM, but the existing lock
> is already removed.
>
> This is a known feature that such "re-locking" is not atomic,
> but in the racy case the file should stay locked (although by
> some other process), but in this case the file will be unlocked.
>
> The proposal is to prepare the lock in advance keeping no chance
> to fail in the future code.
>
> Found during making the flock pid-namespaces aware.
>
> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
>
> ---
>
> diff --git a/fs/locks.c b/fs/locks.c
> index 0db1a14..f59d066 100644
> --- a/fs/locks.c
> +++ b/fs/locks.c
> @@ -732,6 +732,14 @@ static int flock_lock_file(struct file *
> lock_kernel();
> if (request->fl_flags & FL_ACCESS)
> goto find_conflict;
> +
> + if (request->fl_type != F_UNLCK) {
> + error = -ENOMEM;
> + new_fl = locks_alloc_lock();
> + if (new_fl == NULL)
> + goto out;
> + }
> +
> for_each_lock(inode, before) {
> struct file_lock *fl = *before;
> if (IS_POSIX(fl))
> @@ -753,10 +761,6 @@ static int flock_lock_file(struct file *
> goto out;
> }
>
> - error = -ENOMEM;
> - new_fl = locks_alloc_lock();
> - if (new_fl == NULL)
> - goto out;
> /*
> * If a higher-priority process was blocked on the old file lock,
> * give it the opportunity to lock the file.

```





