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.
```