Subject: [patch 05/22] elevate write count files are open()ed
Posted by Cedric Le Goater on Thu, 07 Jun 2007 15:25:32 GMT
View Forum Message <> Reply to Message

From: Dave Hansen <hansendc@us.ibm.com>

This is the first really tricky patch in the series. It
elevates the writer count on a mount each time a
non-special file is opened for write.

This is not completely apparent in the patch because the
two if() conditions in may_open() above the
mnt_want_write() call are, combined, equivalent to
special_file().

There is also an elevated count around the vfs_create()
call in open_namei(). The count needs to be kept elevated
all the way into the may_open() call. Otherwise, when the
write is dropped, a ro->rw transisition could occur. This
would lead to having rw access on the newly created file,
while the vfsmount is ro. That is bad.

Some filesystems forego the use of normal vfs calls to create
struct files. Make sure that these users elevate the mnt writer
count because they will get __fput(), and we need to make
sure they're balanced.

Signed-off-by: Dave Hansen <hansendc@us.ibm.com>

---
 fs/file_table.c | 9 ++++++++-
 fs/namei.c | 20 ++++++++++++++++----
 ipc/mqueue.c | 3 +++
 3 files changed, 27 insertions(+), 5 deletions(-)

Index: 2.6.22-rc4-mm2-robindmount/fs/file_table.c
===
--- 2.6.22-rc4-mm2-robindmount.orig/fs/file_table.c
+++ 2.6.22-rc4-mm2-robindmount/fs/file_table.c
@@ -169,6 +169,10 @@ int init_file(struct file *file, struct
  file->f_mapping = dentry->d_inode->i_mapping;
  file->f_mode = mode;
  file->f_op = fop;
+ if (mode & FMODE_WRITE) {
+ error = mnt_want_write(mnt);
+ WARN_ON(error);
+ }

```
  return error;
 }

@@ -207,8 +211,11 @@ void fastcall __fput(struct file *file)
  if (unlikely(S_ISCHR(inode->i_mode) && inode->i_cdev != NULL))
   cdev_put(inode->i_cdev);
  fops_put(file->f_op);
- if (file->f_mode & FMODE_WRITE)
+ if (file->f_mode & FMODE_WRITE) {
   put_write_access(inode);
+  if(!special_file(inode->i_mode))
+   mnt_drop_write(mnt);
+ }
  put_pid(file->f_owner.pid);
  file_kill(file);
  file->f_path.dentry = NULL;
Index: 2.6.22-rc4-mm2-robindmount/fs/namei.c
===================================================================
--- 2.6.22-rc4-mm2-robindmount.orig/fs/namei.c
+++ 2.6.22-rc4-mm2-robindmount/fs/namei.c
@@ -1596,8 +1596,15 @@ int may_open(struct nameidata *nd, int a
    return -EACCES;

   flag &= ~O_TRUNC;
- } else if (IS_RDONLY(inode) && (flag & FMODE_WRITE))
-  return -EROFS;
+ } else if (flag & FMODE_WRITE) {
+  /*
+   * effectively: !special_file()
+   * balanced by __fput()
+   */
+  error = mnt_want_write(nd->mnt);
+  if (error)
+   return error;
+ }
  /*
   * An append-only file must be opened in append mode for writing.
   */
@@ -1736,14 +1743,17 @@ do_last:
  }

  if (IS_ERR(nd->intent.open.file)) {
-  mutex_unlock(&dir->d_inode->i_mutex);
   error = PTR_ERR(nd->intent.open.file);
-  goto exit_dput;
+  goto exit_mutex_unlock;
  }
```

```
  /* Negative dentry, just create the file */
  if (!path.dentry->d_inode) {
+  error = mnt_want_write(nd->mnt);
+  if (error)
+   goto exit_mutex_unlock;
   error = open_namei_create(nd, &path, flag, mode);
+  mnt_drop_write(nd->mnt);
   if (error)
    goto exit;
   return 0;
@@ -1781,6 +1791,8 @@ ok:
   goto exit;
  return 0;

+exit_mutex_unlock:
+ mutex_unlock(&dir->d_inode->i_mutex);
 exit_dput:
  dput_path(&path, nd);
 exit:
Index: 2.6.22-rc4-mm2-robindmount/ipc/mqueue.c
===================================================================
--- 2.6.22-rc4-mm2-robindmount.orig/ipc/mqueue.c
+++ 2.6.22-rc4-mm2-robindmount/ipc/mqueue.c
@@ -686,6 +686,9 @@ asmlinkage long sys_mq_open(const char _
     goto out;
    filp = do_open(dentry, oflag);
   } else {
+   error = mnt_want_write(mqueue_mnt);
+   if (error)
+    goto out;
    filp = do_create(mqueue_mnt->mnt_root, dentry,
       oflag, mode, u_attr);
   }

--
```

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers