Subject: [PATCH 00/25] Mount writer count and read-only bind mounts (v7)
Posted by Dave Hansen on Mon, 11 Dec 2006 22:30:00 GMT
View Forum Message <> Reply to Message

I figured I'd run this past the containers list one more time.

These are against 2.6.19-mm1. They compile, boot and pass my
little attached test script.

Changes from previous version
- rework helper names for doing custom 'struct file' creation

---

The following series implements read-only bind mounts. This feature
allows a read-only view into a read-write filesystem. In the process
of doing that, it also provides infrastructure for keeping track of
the number of writers to any given mount. In this version, if there
are writers on a superblock, the filesystem may not be remounted
r/o. The same goes for MS_BIND mounts, and writers on a vfsmount.

This has a number of uses. It allows chroots to have parts of
filesystems writable. It will be useful for containers in the future
and is intended to replace patches that vserver has had out of the
tree for several years. It allows security enhancement by making
sure that parts of your filesystem read-only, when you don't want
to have entire new filesystems mounted, or when you want atime
selectively updated.

This set makes no attempt to keep the return codes for these
r/o bind mounts the same as for a real r/o filesystem or device.
It would require significantly more code and be quite a bit more
invasive.

Using this feature requires two steps:

```
 mount --bind /source /dest
 mount -o remount,ro  /dest
```

I've been using the following script to test that the feature is
working as desired. It takes a directory and makes a regular
bind and a r/o bind mount of it. It then performs some normal
filesystem operations on the three directories, including ones
that are expected to fail, like creating a file on the r/o
mount.

```
#!/bin/sh
DIRS="foo foo-bound foo-bound-ro"
```

```
set -e

function do_umount
{
 (
  umount foo-bound || true;
  umount foo-bound-ro || true;
 ) 2> /dev/null
}

trap do_umount ERR

# just in case the last invocation left them
do_umount

function should_fail
{
 "$@" > /dev/null 2>&1 \
 && echo unexpected success: "$@" \
 || echo GOOD: expected failure: "$@";
}
function should_succeed
{
 "$@" > /dev/null 2>&1 \
 && echo GOOD: expected success: "$@" \
 || echo unexpected failure: "$@"
}
function should_fail_ro
{
 RO=$1
 shift
 if $RO; then
  should_fail "$@"
 else
  should_succeed "$@"
 fi
}
function testdir
{
 RO=$1
 shift;
 i=$1
 should_fail_ro $RO touch $i/$i-file
 should_fail_ro $RO mkdir $i/$i-dir
 should_fail_ro $RO mknod $i/$i-null-chardev c 1 3
 should_fail_ro $RO chmod 777 $i/$i-null-chardev
}
for i in $DIRS; do
```

```
 rm -r ./$i > /dev/null 2>&1 || true
 mkdir -p $i;
done;

mount --bind foo foo-bound/  || exit
mount --bind foo foo-bound-ro  || exit
mount -o remount,ro foo-bound-ro || exit

testdir false foo
testdir false foo-bound
testdir true  foo-bound-ro

should_succeed echo foo > foo-bound/foo-null-chardev
should_succeed echo foo > foo-bound-ro/foo-null-chardev

should_fail chmod 777 foo-bound-ro/foo-dir
should_fail chmod 777 foo-bound-ro/foo-file
should_fail chown nobody foo-bound-ro/foo-dir
should_fail chown nobody foo-bound-ro/foo-file
should_fail rmdir foo-bound-ro/foo-dir

do_umount

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



---

## Subject: [PATCH 01/25] filesystem helpers for custom 'struct file's
Posted by Dave Hansen on Mon, 11 Dec 2006 22:30:02 GMT
View Forum Message <> Reply to Message

Some filesystems forego the vfs and may_open() and create their
own 'struct file's.

This patch creates a couple of helper functions which can be
used by these filesystems, and will provide a unified place
which the r/o bind mount code may patch.

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

```
 lxc-dave/fs/file_table.c      |   30 ++++++++++++++++++++++++++++++
 lxc-dave/fs/hugetlbfs/inode.c |   22 +++++++++----------
 lxc-dave/include/linux/file.h |    8 ++++++++
 lxc-dave/mm/shmem.c           |    7 ++-----
```

```
 lxc-dave/mm/tiny-shmem.c      |   24 +++++++++----------------
 lxc-dave/net/socket.c         |   18 +++++++++---------
 6 files changed, 67 insertions(+), 42 deletions(-)

diff -puN fs/file_table.c~01-24-filesystem-helpers-for-custom-struct-file-s fs/file_table.c
--- lxc/fs/file_table.c~01-24-filesystem-helpers-for-custom-struct-file-s 2006-12-11
14:21:56.000000000 -0800
+++ lxc-dave/fs/file_table.c 2006-12-11 14:21:56.000000000 -0800
@@ -139,6 +139,36 @@ fail:

 EXPORT_SYMBOL(get_empty_filp);

+struct file *alloc_file(struct vfsmount *mnt,  struct dentry *dentry,
+  mode_t mode, const struct file_operations *fop)
+{
+ struct file *file;
+
+ file = get_empty_filp();
+ if (!file)
+  return NULL;
+
+ init_file(file, mnt, dentry, mode, fop);
+ return file;
+}
+
+EXPORT_SYMBOL(alloc_file);
+
+int init_file(struct file *file, struct vfsmount *mnt,
+    struct dentry *dentry, mode_t mode,
+    const struct file_operations *fop)
+{
+ int error = 0;
+ file->f_vfsmnt = mntget(mnt);
+ file->f_dentry = dentry;
+ file->f_mapping = dentry->d_inode->i_mapping;
+ file->f_mode = mode;
+ file->f_op = fop;
+ return error;
+}
+
+EXPORT_SYMBOL(init_file);
+
 void fastcall fput(struct file *file)
 {
  if (atomic_dec_and_test(&file->f_count))
diff -puN fs/hugetlbfs/inode.c~01-24-filesystem-helpers-for-custom-struct-file-s fs/hugetlbfs/inode.c
--- lxc/fs/hugetlbfs/inode.c~01-24-filesystem-helpers-for-custom-struct-file-s 2006-12-11
14:21:56.000000000 -0800
```

```
+++ lxc-dave/fs/hugetlbfs/inode.c 2006-12-11 14:21:56.000000000 -0800
@@ -756,16 +756,11 @@ struct file *hugetlb_zero_setup(size_t s
  if (!dentry)
   goto out_shm_unlock;

- error = -ENFILE;
- file = get_empty_filp();
- if (!file)
-  goto out_dentry;
-
  error = -ENOSPC;
  inode = hugetlbfs_get_inode(root->d_sb, current->fsuid,
     current->fsgid, S_IFREG | S_IRWXUGO, 0);
  if (!inode)
-  goto out_file;
+  goto out_dentry;

  error = -ENOMEM;
  if (hugetlb_reserve_pages(inode, 0, size >> HPAGE_SHIFT))
@@ -774,17 +769,18 @@ struct file *hugetlb_zero_setup(size_t s
  d_instantiate(dentry, inode);
  inode->i_size = size;
  inode->i_nlink = 0;
- file->f_path.mnt = mntget(hugetlbfs_vfsmount);
- file->f_path.dentry = dentry;
- file->f_mapping = inode->i_mapping;
- file->f_op = &hugetlbfs_file_operations;
- file->f_mode = FMODE_WRITE | FMODE_READ;
+
+ error = -ENFILE;
+ file = alloc_file(hugetlbfs_vfsmount, dentry,
+   FMODE_WRITE | FMODE_READ,
+   &hugetlbfs_file_operations);
+ if (!file)
+  goto out_inode;
+
  return file;

 out_inode:
  iput(inode);
-out_file:
- put_filp(file);
 out_dentry:
  dput(dentry);
 out_shm_unlock:
diff -puN include/linux/file.h~01-24-filesystem-helpers-for-custom-struct-file-s include/linux/file.h
--- lxc/include/linux/file.h~01-24-filesystem-helpers-for-custom-struct-file-s 2006-12-11
14:21:56.000000000 -0800
```

```
+++ lxc-dave/include/linux/file.h 2006-12-11 14:21:56.000000000 -0800
@@ -62,6 +62,14 @@ extern struct kmem_cache *filp_cachep;
 extern void FASTCALL(__fput(struct file *));
 extern void FASTCALL(fput(struct file *));

+struct file_operations;
+struct vfsmount;
+struct dentry;
+extern int init_file(struct file *, struct vfsmount *, struct dentry *dentry,
+  mode_t mode, const struct file_operations *fop);
+extern struct file *alloc_file(struct vfsmount *, struct dentry *dentry,
+  mode_t mode, const struct file_operations *fop);
+
 static inline void fput_light(struct file *file, int fput_needed)
 {
  if (unlikely(fput_needed))
diff -puN mm/shmem.c~01-24-filesystem-helpers-for-custom-struct-file-s mm/shmem.c
--- lxc/mm/shmem.c~01-24-filesystem-helpers-for-custom-struct-file-s 2006-12-11
14:21:56.000000000 -0800
+++ lxc-dave/mm/shmem.c 2006-12-11 14:21:56.000000000 -0800
@@ -2493,11 +2493,8 @@ struct file *shmem_file_setup(char *name
  d_instantiate(dentry, inode);
  inode->i_size = size;
  inode->i_nlink = 0; /* It is unlinked */
- file->f_path.mnt = mntget(shm_mnt);
- file->f_path.dentry = dentry;
- file->f_mapping = inode->i_mapping;
- file->f_op = &shmem_file_operations;
- file->f_mode = FMODE_WRITE | FMODE_READ;
+ init_file(file, shm_mnt, dentry, FMODE_WRITE | FMODE_READ,
+   &shmem_file_operations);
  return file;

 close_file:
diff -puN mm/tiny-shmem.c~01-24-filesystem-helpers-for-custom-struct-file-s mm/tiny-shmem.c
--- lxc/mm/tiny-shmem.c~01-24-filesystem-helpers-for-custom-struct-file-s 2006-12-11
14:21:56.000000000 -0800
+++ lxc-dave/mm/tiny-shmem.c 2006-12-11 14:21:56.000000000 -0800
@@ -66,24 +66,19 @@ struct file *shmem_file_setup(char *name
  if (!dentry)
   goto put_memory;

- error = -ENFILE;
- file = get_empty_filp();
- if (!file)
-  goto put_dentry;
-
  error = -ENOSPC;
```

```
  inode = ramfs_get_inode(root->d_sb, S_IFREG | S_IRWXUGO, 0);
  if (!inode)
-  goto close_file;
+  goto put_dentry;

  d_instantiate(dentry, inode);
- inode->i_nlink = 0; /* It is unlinked */
+ error = -ENFILE;
+ file = alloc_file(shm_mnt, dentry, FMODE_WRITE | FMODE_READ,
+   &ramfs_file_operations);
+ if (!file)
+  goto put_inode;

- file->f_path.mnt = mntget(shm_mnt);
- file->f_path.dentry = dentry;
- file->f_mapping = inode->i_mapping;
- file->f_op = &ramfs_file_operations;
- file->f_mode = FMODE_WRITE | FMODE_READ;
+ inode->i_nlink = 0; /* It is unlinked */

  /* notify everyone as to the change of file size */
  error = do_truncate(dentry, size, 0, file);
@@ -91,9 +86,8 @@ struct file *shmem_file_setup(char *name
   goto close_file;

  return file;
-
-close_file:
- put_filp(file);
+put_inode:
+ iput(inode);
 put_dentry:
  dput(dentry);
 put_memory:
diff -puN net/socket.c~01-24-filesystem-helpers-for-custom-struct-file-s net/socket.c
--- lxc/net/socket.c~01-24-filesystem-helpers-for-custom-struct-file-s 2006-12-11
14:21:56.000000000 -0800
+++ lxc-dave/net/socket.c 2006-12-11 14:21:56.000000000 -0800
@@ -355,6 +355,7 @@ static int sock_alloc_fd(struct file **f

 static int sock_attach_fd(struct socket *sock, struct file *file)
 {
+ struct dentry *dentry;
  struct qstr this;
  char name[32];

@@ -362,24 +363,23 @@ static int sock_attach_fd(struct socket
  this.name = name;
```

```
  this.hash = 0;

- file->f_path.dentry = d_alloc(sock_mnt->mnt_sb->s_root, &this);
- if (unlikely(!file->f_path.dentry))
+ dentry = d_alloc(sock_mnt->mnt_sb->s_root, &this);
+ if (unlikely(!dentry))
    return -ENOMEM;

- file->f_path.dentry->d_op = &sockfs_dentry_operations;
+ dentry->d_op = &sockfs_dentry_operations;
  /*
   * We dont want to push this dentry into global dentry hash table.
   * We pretend dentry is already hashed, by unsetting DCACHE_UNHASHED
   * This permits a working /proc/$pid/fd/XXX on sockets
   */
- file->f_path.dentry->d_flags &= ~DCACHE_UNHASHED;
- d_instantiate(file->f_path.dentry, SOCK_INODE(sock));
- file->f_path.mnt = mntget(sock_mnt);
- file->f_mapping = file->f_path.dentry->d_inode->i_mapping;
-
+ dentry->d_flags &= ~DCACHE_UNHASHED;
+ d_instantiate(dentry, SOCK_INODE(sock));
+ init_file(file, sock_mnt, dentry, FMODE_READ | FMODE_WRITE,
+  &socket_file_ops);
+ SOCK_INODE(sock)->i_fop = &socket_file_ops;
  sock->file = file;
  file->f_op = SOCK_INODE(sock)->i_fop = &socket_file_ops;
- file->f_mode = FMODE_READ | FMODE_WRITE;
  file->f_flags = O_RDWR;
  file->f_pos = 0;
  file->private_data = sock;

_
_____________________________________________________
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers
```

---

## Subject: [PATCH 02/25] introduce simple_set_mnt_no_get() helper for NFS
Posted by Dave Hansen on Mon, 11 Dec 2006 22:30:03 GMT
View Forum Message <> Reply to Message

The next patch will introduce a list of all mounts for a given
superblock.  In order to keep the list, we need to make sure
all filesystems attaching a mount to a superblock get added
to this list.

NFS currently bypasses the simple_set_mnt() function, and sets

mnt_sb directly.  This patch makes it use a helper function,
instead.

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

 lxc-dave/fs/namespace.c     |   13 ++++++++++---
 lxc-dave/fs/nfs/super.c     |   10 +++++-----
 lxc-dave/include/linux/fs.h |    1 +
 3 files changed, 16 insertions(+), 8 deletions(-)

diff -puN fs/namespace.c~reintroduce-list-of-vfsmounts-over-superblock fs/namespace.c
--- lxc/fs/namespace.c~reintroduce-list-of-vfsmounts-over-superblock 2006-12-11
14:21:57.000000000 -0800
+++ lxc-dave/fs/namespace.c 2006-12-11 14:21:57.000000000 -0800
@@ -77,13 +77,20 @@ struct vfsmount *alloc_vfsmnt(const char
  return mnt;
 }

-int simple_set_mnt(struct vfsmount *mnt, struct super_block *sb)
+int simple_set_mnt_no_get(struct vfsmount *mnt, struct super_block *sb)
 {
- mnt->mnt_sb = sb;
- mnt->mnt_root = dget(sb->s_root);
+  mnt->mnt_sb = sb;
  return 0;
 }

+EXPORT_SYMBOL(simple_set_mnt_no_get);
+
+int simple_set_mnt(struct vfsmount *mnt, struct super_block *sb)
+{
+  mnt->mnt_root = dget(sb->s_root);
+ return simple_set_mnt_no_get(mnt, sb);
+}
+
 EXPORT_SYMBOL(simple_set_mnt);

 void free_vfsmnt(struct vfsmount *mnt)
diff -puN fs/nfs/super.c~reintroduce-list-of-vfsmounts-over-superblock fs/nfs/super.c
--- lxc/fs/nfs/super.c~reintroduce-list-of-vfsmounts-over-superblock 2006-12-11
14:21:57.000000000 -0800
+++ lxc-dave/fs/nfs/super.c 2006-12-11 14:21:57.000000000 -0800
@@ -653,7 +653,7 @@ static int nfs_get_sb(struct file_system
  }

  s->s_flags |= MS_ACTIVE;
- mnt->mnt_sb = s;

```
+ simple_set_mnt_no_get(mnt, s);
  mnt->mnt_root = mntroot;
  return 0;

@@ -726,7 +726,7 @@ static int nfs_xdev_get_sb(struct file_s
  }

  s->s_flags |= MS_ACTIVE;
- mnt->mnt_sb = s;
+ simple_set_mnt_no_get(mnt, s);
  mnt->mnt_root = mntroot;

  dprintk("<-- nfs_xdev_get_sb() = 0\n");
@@ -903,7 +903,7 @@ static int nfs4_get_sb(struct file_syste
  }

  s->s_flags |= MS_ACTIVE;
- mnt->mnt_sb = s;
+ simple_set_mnt_no_get(mnt, s);
  mnt->mnt_root = mntroot;
  kfree(mntpath);
  kfree(hostname);
@@ -984,7 +984,7 @@ static int nfs4_xdev_get_sb(struct file_
  }

  s->s_flags |= MS_ACTIVE;
- mnt->mnt_sb = s;
+ simple_set_mnt_no_get(mnt, s);
  mnt->mnt_root = mntroot;

  dprintk("<-- nfs4_xdev_get_sb() = 0\n");
@@ -1051,7 +1051,7 @@ static int nfs4_referral_get_sb(struct f
  }

  s->s_flags |= MS_ACTIVE;
- mnt->mnt_sb = s;
+ simple_set_mnt_no_get(mnt, s);
  mnt->mnt_root = mntroot;

  dprintk("<-- nfs4_referral_get_sb() = 0\n");
diff -puN include/linux/fs.h~reintroduce-list-of-vfsmounts-over-superblock include/linux/fs.h
--- lxc/include/linux/fs.h~reintroduce-list-of-vfsmounts-over-superblock 2006-12-11
14:21:57.000000000 -0800
+++ lxc-dave/include/linux/fs.h 2006-12-11 14:21:57.000000000 -0800
@@ -1426,6 +1426,7 @@ extern int get_sb_pseudo(struct file_sys
  struct super_operations *ops, unsigned long,
  struct vfsmount *mnt);
 extern int simple_set_mnt(struct vfsmount *mnt, struct super_block *sb);
```

```
+extern int simple_set_mnt_no_get(struct vfsmount *mnt, struct super_block *sb);
 int __put_super(struct super_block *sb);
 int __put_super_and_need_restart(struct super_block *sb);
 void unnamed_dev_init(void);
```

_

_____

Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

---

Subject: [PATCH 04/25] Add vfsmount writer count
Posted by Dave Hansen on Mon, 11 Dec 2006 22:30:04 GMT
View Forum Message <> Reply to Message

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

---

```
 lxc-dave/fs/namespace.c        |   63 ++++++++++++++++++++++++++++++++++++++++++
 lxc-dave/fs/super.c            |   66 ++++++++++++++++++++++++++++++++++++++++++----
 lxc-dave/include/linux/fs.h    |    2 +
 lxc-dave/include/linux/mount.h |   18 ++++++++++
 4 files changed, 144 insertions(+), 5 deletions(-)

diff -puN fs/namespace.c~03-24-add-vfsmount-writer-count fs/namespace.c
--- lxc/fs/namespace.c~03-24-add-vfsmount-writer-count 2006-12-11 14:21:58.000000000 -0800
+++ lxc-dave/fs/namespace.c 2006-12-11 14:21:58.000000000 -0800
@@ -57,6 +57,7 @@ struct vfsmount *alloc_vfsmnt(const char
  if (mnt) {
   memset(mnt, 0, sizeof(struct vfsmount));
   atomic_set(&mnt->mnt_count, 1);
+  atomic_set(&mnt->mnt_writers, 0);
   INIT_LIST_HEAD(&mnt->mnt_hash);
   INIT_LIST_HEAD(&mnt->mnt_child);
   INIT_LIST_HEAD(&mnt->mnt_mounts);
@@ -894,6 +895,60 @@ out_unlock:
  return err;
 }

+int mnt_make_readonly(struct vfsmount *mnt)
+{
+ int ret = 0;
+
+ WARN_ON(__mnt_is_readonly(mnt));
+
+ /*
+  * This flag set is actually redundant with what
```

```
+  * happens in do_remount(), but since we do this
+  * under the lock, anyone attempting to get a write
+  * on it after this will fail.
+  */
+ spin_lock(&mnt->mnt_sb->s_mnt_writers_lock);
+ if (!atomic_read(&mnt->mnt_writers))
+  mnt->mnt_flags |= MNT_READONLY;
+ else
+  ret = -EBUSY;
+ spin_unlock(&mnt->mnt_sb->s_mnt_writers_lock);
+ return ret;
+}
+
+int mnt_want_write(struct vfsmount *mnt)
+{
+ int ret = 0;
+repeat:
+ if (atomic_add_unless(&mnt->mnt_writers, 1, 0))
+  return 0;
+
+ spin_lock(&mnt->mnt_sb->s_mnt_writers_lock);
+ if (__mnt_is_readonly(mnt)) {
+  ret = -EROFS;
+  goto out;
+ }
+ if (atomic_add_return(1, &mnt->mnt_writers) != 1) {
+  atomic_dec(&mnt->mnt_writers);
+  spin_unlock(&mnt->mnt_sb->s_mnt_writers_lock);
+  goto repeat;
+ }
+ atomic_inc(&mnt->mnt_sb->s_mnt_writers);
+out:
+ spin_unlock(&mnt->mnt_sb->s_mnt_writers_lock);
+ return ret;
+}
+
+void mnt_drop_write(struct vfsmount *mnt)
+{
+ if (!atomic_dec_and_lock(&mnt->mnt_writers,
+     &mnt->mnt_sb->s_mnt_writers_lock))
+  return;
+
+ atomic_dec(&mnt->mnt_sb->s_mnt_writers);
+ spin_unlock(&mnt->mnt_sb->s_mnt_writers_lock);
+}
+
 /*
  * recursively change the type of the mountpoint.
```

```
  */
@@ -962,6 +1017,7 @@ out:
  path_release(&old_nd);
  return err;
 }
+EXPORT_SYMBOL_GPL(mnt_want_write);

 /*
  * change filesystem flags. dir should be a physical root of filesystem.
@@ -985,6 +1041,8 @@ static int do_remount(struct nameidata *

  down_write(&sb->s_umount);
  err = do_remount_sb(sb, flags, data, 0);
+ if (!(sb->s_flags & MS_RDONLY))
+  mnt_flags |= MNT_SB_WRITABLE;
  if (!err)
   nd->mnt->mnt_flags = mnt_flags;
  up_write(&sb->s_umount);
@@ -992,6 +1050,7 @@ static int do_remount(struct nameidata *
   security_sb_post_remount(nd->mnt, flags, data);
  return err;
 }
+EXPORT_SYMBOL_GPL(mnt_drop_write);

 static inline int tree_contains_unbindable(struct vfsmount *mnt)
 {
@@ -1125,6 +1184,8 @@ int do_add_mount(struct vfsmount *newmnt
  if (S_ISLNK(newmnt->mnt_root->d_inode->i_mode))
   goto unlock;

+ if (!(newmnt->mnt_sb->s_flags & MS_RDONLY))
+  mnt_flags |= MNT_SB_WRITABLE;
  newmnt->mnt_flags = mnt_flags;
  if ((err = graft_tree(newmnt, nd)))
   goto unlock;
@@ -1416,6 +1477,8 @@ long do_mount(char *dev_name, char *dir_
   ((char *)data_page)[PAGE_SIZE - 1] = 0;

  /* Separate the per-mountpoint flags */
+ if (flags & MS_RDONLY)
+  mnt_flags |= MNT_READONLY;
  if (flags & MS_NOSUID)
   mnt_flags |= MNT_NOSUID;
  if (flags & MS_NODEV)
diff -puN fs/super.c~03-24-add-vfsmount-writer-count fs/super.c
--- lxc/fs/super.c~03-24-add-vfsmount-writer-count 2006-12-11 14:21:58.000000000 -0800
+++ lxc-dave/fs/super.c 2006-12-11 14:21:58.000000000 -0800
@@ -94,6 +94,8 @@ static struct super_block *alloc_super(s
```

```
   s->s_qcop = sb_quotactl_ops;
   s->s_op = &default_op;
   s->s_time_gran = 1000000000;
+  atomic_set(&s->s_mnt_writers, 0);
+  spin_lock_init(&s->s_mnt_writers_lock);
  }
 out:
  return s;
@@ -577,6 +579,53 @@ static void mark_files_ro(struct super_b
  file_list_unlock();
 }

+static void __sb_mounts_mod_flag(struct super_block *sb, int set, int flag)
+{
+ struct list_head *p;
+
+ spin_lock(&vfsmount_lock);
+ list_for_each(p, &sb->s_vfsmounts) {
+  struct vfsmount *mnt =
+   list_entry(p, struct vfsmount, mnt_sb_list);
+  if (set)
+   mnt->mnt_flags |= flag;
+  else
+   mnt->mnt_flags &= ~flag;
+ }
+ spin_unlock(&vfsmount_lock);
+}
+
+static void sb_mounts_set_flag(struct super_block *sb, int flag)
+{
+ __sb_mounts_mod_flag(sb, 1, flag);
+}
+static void sb_mounts_clear_flag(struct super_block *sb, int flag)
+{
+ __sb_mounts_mod_flag(sb, 0, flag);
+}
+
+static int sb_remount_ro(struct super_block *sb)
+{
+ return fs_may_remount_ro(sb);
+ spin_lock(&sb->s_mnt_writers_lock);
+ if (atomic_read(&sb->s_mnt_writers) > 0) {
+  spin_unlock(&sb->s_mnt_writers_lock);
+  return -EBUSY;
+ }
+
+ sb_mounts_clear_flag(sb, MNT_SB_WRITABLE);
+ spin_unlock(&sb->s_mnt_writers_lock);
```

```
+
+ return 0;
+}
+
+static void sb_remount_rw(struct super_block *sb)
+{
+ spin_lock(&sb->s_mnt_writers_lock);
+ sb_mounts_set_flag(sb, MNT_SB_WRITABLE);
+ spin_unlock(&sb->s_mnt_writers_lock);
+}
+
 /**
  * do_remount_sb - asks filesystem to change mount options.
  * @sb: superblock in question
@@ -588,7 +637,8 @@ static void mark_files_ro(struct super_b
  */
 int do_remount_sb(struct super_block *sb, int flags, void *data, int force)
 {
- int retval;
+ int retval = 0;
+ int sb_started_ro = (sb->s_flags & MS_RDONLY);

 #ifdef CONFIG_BLOCK
  if (!(flags & MS_RDONLY) && bdev_read_only(sb->s_bdev))
@@ -601,13 +651,14 @@ int do_remount_sb(struct super_block *sb

  /* If we are remounting RDONLY and current sb is read/write,
     make sure there are no rw files opened */
- if ((flags & MS_RDONLY) && !(sb->s_flags & MS_RDONLY)) {
+ if ((flags & MS_RDONLY) && !sb_started_ro) {
   if (force)
    mark_files_ro(sb);
-  else if (!fs_may_remount_ro(sb))
-   return -EBUSY;
+  else
+   retval = sb_remount_ro(sb);
+  if (retval)
+   return retval;
  }
-
  if (sb->s_op->remount_fs) {
   lock_super(sb);
   retval = sb->s_op->remount_fs(sb, &flags, data);
@@ -615,6 +666,9 @@ int do_remount_sb(struct super_block *sb
   if (retval)
    return retval;
  }
+ if (!(flags & MS_RDONLY) && sb_started_ro)
```

```
+  sb_remount_rw(sb);
+
  sb->s_flags = (sb->s_flags & ~MS_RMT_MASK) | (flags & MS_RMT_MASK);
  return 0;
 }
@@ -903,6 +957,8 @@ vfs_kern_mount(struct file_system_type *

  mnt->mnt_mountpoint = mnt->mnt_root;
  mnt->mnt_parent = mnt;
+ if (!(mnt->mnt_sb->s_flags & MS_RDONLY))
+  mnt->mnt_flags |= MNT_SB_WRITABLE;
  up_write(&mnt->mnt_sb->s_umount);
  free_secdata(secdata);
  return mnt;
diff -puN include/linux/fs.h~03-24-add-vfsmount-writer-count include/linux/fs.h
--- lxc/include/linux/fs.h~03-24-add-vfsmount-writer-count 2006-12-11 14:21:58.000000000 -0800
+++ lxc-dave/include/linux/fs.h 2006-12-11 14:21:58.000000000 -0800
@@ -968,6 +968,8 @@ struct super_block {
  struct list_head s_io;  /* parked for writeback */
  struct hlist_head s_anon;  /* anonymous dentries for (nfs) exporting */
  struct list_head s_vfsmounts;
+ atomic_t  s_mnt_writers; /* vfsmounts with active writers */
+ spinlock_t  s_mnt_writers_lock; /* taken when mounts change rw state */
  struct list_head s_files;

  struct block_device *s_bdev;
diff -puN include/linux/mount.h~03-24-add-vfsmount-writer-count include/linux/mount.h
--- lxc/include/linux/mount.h~03-24-add-vfsmount-writer-count 2006-12-11 14:21:58.000000000
-0800
+++ lxc-dave/include/linux/mount.h 2006-12-11 14:21:58.000000000 -0800
@@ -28,6 +28,8 @@ struct mnt_namespace;
 #define MNT_NOATIME 0x08
 #define MNT_NODIRATIME 0x10
 #define MNT_RELATIME 0x20
+#define MNT_READONLY 0x40 /* does the user want this to be r/o? */
+#define MNT_SB_WRITABLE 0x80 /* does the SB currently allow writes? */

 #define MNT_SHRINKABLE 0x100

@@ -45,6 +47,7 @@ struct vfsmount {
  struct list_head mnt_mounts; /* list of children, anchored here */
  struct list_head mnt_child; /* and going through their mnt_child */
  atomic_t mnt_count;
+ atomic_t mnt_writers;
  int mnt_flags;
  int mnt_expiry_mark;  /* true if marked for expiry */
  char *mnt_devname;  /* Name of device e.g. /dev/dsk/hda1 */
@@ -65,6 +68,21 @@ static inline struct vfsmount *mntget(st
```

```
 return mnt;
}

+static inline int __mnt_is_readonly(struct vfsmount *mnt)
+{
+ return (mnt->mnt_flags & MNT_READONLY) ||
+        !(mnt->mnt_flags & MNT_SB_WRITABLE);
+}
+
+static inline void __mnt_unmake_readonly(struct vfsmount *mnt)
+{
+ WARN_ON(!__mnt_is_readonly(mnt));
+ mnt->mnt_flags &= ~MNT_READONLY;
+}
+
+extern int mnt_make_readonly(struct vfsmount *mnt);
+extern int mnt_want_write(struct vfsmount *mnt);
+extern void mnt_drop_write(struct vfsmount *mnt);
 extern void mntput_no_expire(struct vfsmount *mnt);
 extern void mnt_pin(struct vfsmount *mnt);
 extern void mnt_unpin(struct vfsmount *mnt);

_
_______________________________________________
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers
```

---

## Subject: [PATCH 06/25] kill open files traverse on remount ro
Posted by Dave Hansen on Mon, 11 Dec 2006 22:30:05 GMT
View Forum Message <> Reply to Message

Now that we have the sb writer count, we don't need to
go looking at all of the individual open files.

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

```
 lxc-dave/fs/file_table.c    |   25 -------------------------
 lxc-dave/fs/super.c         |    1 -
 lxc-dave/include/linux/fs.h |    2 --
 3 files changed, 28 deletions(-)

diff -puN fs/file_table.c~05-24-kill-open-files-traverse-on-remount-ro fs/file_table.c
--- lxc/fs/file_table.c~05-24-kill-open-files-traverse-on-remount-ro 2006-12-11 14:22:00.000000000
-0800
+++ lxc-dave/fs/file_table.c 2006-12-11 14:22:00.000000000 -0800
@@ -293,31 +293,6 @@ void file_kill(struct file *file)
```

```
 }
}

-int fs_may_remount_ro(struct super_block *sb)
-{
- struct list_head *p;
-
- /* Check that no files are currently opened for writing. */
- file_list_lock();
- list_for_each(p, &sb->s_files) {
-  struct file *file = list_entry(p, struct file, f_u.fu_list);
-  struct inode *inode = file->f_path.dentry->d_inode;
-
-  /* File with pending delete? */
-  if (inode->i_nlink == 0)
-   goto too_bad;
-
-  /* Writeable file? */
-  if (S_ISREG(inode->i_mode) && (file->f_mode & FMODE_WRITE))
-   goto too_bad;
- }
- file_list_unlock();
- return 1; /* Tis' cool bro. */
-too_bad:
- file_list_unlock();
- return 0;
-}
-
 void __init files_init(unsigned long mempages)
 {
  int n;
diff -puN fs/super.c~05-24-kill-open-files-traverse-on-remount-ro fs/super.c
--- lxc/fs/super.c~05-24-kill-open-files-traverse-on-remount-ro 2006-12-11 14:22:00.000000000
-0800
+++ lxc-dave/fs/super.c 2006-12-11 14:22:00.000000000 -0800
@@ -606,7 +606,6 @@ static void sb_mounts_clear_flag(struct

 static int sb_remount_ro(struct super_block *sb)
 {
- return fs_may_remount_ro(sb);
  spin_lock(&sb->s_mnt_writers_lock);
  if (atomic_read(&sb->s_mnt_writers) > 0) {
   spin_unlock(&sb->s_mnt_writers_lock);
diff -puN include/linux/fs.h~05-24-kill-open-files-traverse-on-remount-ro include/linux/fs.h
--- lxc/include/linux/fs.h~05-24-kill-open-files-traverse-on-remount-ro 2006-12-11
14:22:00.000000000 -0800
+++ lxc-dave/include/linux/fs.h 2006-12-11 14:22:00.000000000 -0800
@@ -1609,8 +1609,6 @@ extern const struct file_operations read
```

```
 extern const struct file_operations write_fifo_fops;
 extern const struct file_operations rdwr_fifo_fops;

-extern int fs_may_remount_ro(struct super_block *);
-
 #ifdef CONFIG_BLOCK
 /*
  * return READ, READA, or WRITE
_
```

---

Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

---

## Subject: [PATCH 07/25] elevate writer count for chown and friends

Posted by Dave Hansen on Mon, 11 Dec 2006 22:30:06 GMT

View Forum Message <> Reply to Message

chown/chmod,etc... don't call permission in the same way
that the normal "open for write" calls do.  They still
write to the filesystem, so bump the write count during
these operations.

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

```
 lxc-dave/fs/open.c |   37 ++++++++++++++++++++++++++++++++++---
 1 file changed, 33 insertions(+), 4 deletions(-)

diff -puN fs/open.c~06-24-elevate-writer-count-for-chown-and-friends fs/open.c
--- lxc/fs/open.c~06-24-elevate-writer-count-for-chown-and-friends 2006-12-11
14:22:01.000000000 -0800
+++ lxc-dave/fs/open.c 2006-12-11 14:22:01.000000000 -0800
@@ -511,9 +511,12 @@ asmlinkage long sys_fchmod(unsigned int
  err = -EROFS;
  if (IS_RDONLY(inode))
   goto out_putf;
+ err = mnt_want_write(file->f_vfsmnt);
+ if (err)
+  goto out_putf;
  err = -EPERM;
  if (IS_IMMUTABLE(inode) || IS_APPEND(inode))
-  goto out_putf;
+  goto out_drop_write;
  mutex_lock(&inode->i_mutex);
  if (mode == (mode_t) -1)
   mode = inode->i_mode;
```

```
@@ -522,6 +525,8 @@ asmlinkage long sys_fchmod(unsigned int
  err = notify_change(dentry, &newattrs);
  mutex_unlock(&inode->i_mutex);

+out_drop_write:
+ mnt_drop_write(file->f_vfsmnt);
 out_putf:
  fput(file);
 out:
@@ -541,13 +546,16 @@ asmlinkage long sys_fchmodat(int dfd, co
   goto out;
  inode = nd.dentry->d_inode;

+ error = mnt_want_write(nd.mnt);
+ if (error)
+  goto dput_and_out;
  error = -EROFS;
  if (IS_RDONLY(inode))
-  goto dput_and_out;
+  goto out_drop_write;

  error = -EPERM;
  if (IS_IMMUTABLE(inode) || IS_APPEND(inode))
-  goto dput_and_out;
+  goto out_drop_write;

  mutex_lock(&inode->i_mutex);
  if (mode == (mode_t) -1)
@@ -557,6 +565,8 @@ asmlinkage long sys_fchmodat(int dfd, co
  error = notify_change(nd.dentry, &newattrs);
  mutex_unlock(&inode->i_mutex);

+out_drop_write:
+ mnt_drop_write(nd.mnt);
 dput_and_out:
  path_release(&nd);
 out:
@@ -582,7 +592,7 @@ static int chown_common(struct dentry *
  error = -EROFS;
  if (IS_RDONLY(inode))
   goto out;
- error = -EPERM;
+  error = -EPERM;
  if (IS_IMMUTABLE(inode) || IS_APPEND(inode))
   goto out;
  newattrs.ia_valid =  ATTR_CTIME;
@@ -611,7 +621,12 @@ asmlinkage long sys_chown(const char __u
  error = user_path_walk(filename, &nd);
```

```
  if (error)
   goto out;
+ error = mnt_want_write(nd.mnt);
+ if (error)
+  goto out_release;
  error = chown_common(nd.dentry, user, group);
+ mnt_drop_write(nd.mnt);
+out_release:
  path_release(&nd);
 out:
  return error;
@@ -631,7 +646,12 @@ asmlinkage long sys_fchownat(int dfd, co
  error = __user_walk_fd(dfd, filename, follow, &nd);
  if (error)
   goto out;
+ error = mnt_want_write(nd.mnt);
+ if (error)
+  goto out_release;
  error = chown_common(nd.dentry, user, group);
+ mnt_drop_write(nd.mnt);
+out_release:
  path_release(&nd);
 out:
  return error;
@@ -645,7 +665,11 @@ asmlinkage long sys_lchown(const char __
  error = user_path_walk_link(filename, &nd);
  if (error)
   goto out;
+ error = mnt_want_write(nd.mnt);
+ if (error)
+  goto out_release;
  error = chown_common(nd.dentry, user, group);
+out_release:
  path_release(&nd);
 out:
  return error;
@@ -662,9 +686,14 @@ asmlinkage long sys_fchown(unsigned int
  if (!file)
   goto out;

+ error = mnt_want_write(file->f_vfsmnt);
+ if (error)
+  goto out_fput;
  dentry = file->f_path.dentry;
  audit_inode(NULL, dentry->d_inode);
  error = chown_common(dentry, user, group);
+ mnt_drop_write(file->f_vfsmnt);
+out_fput:
```

```
 fput(file);
out:
 return error;
```

–
_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

---

Subject: [PATCH 08/25] elevate mnt writers for callers of vfs_mkdir()
Posted by Dave Hansen on Mon, 11 Dec 2006 22:30:07 GMT
View Forum Message <> Reply to Message

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

```
 lxc-dave/fs/namei.c           |    5 +++++
 lxc-dave/fs/nfsd/nfs4recover.c |    4 ++++
 2 files changed, 9 insertions(+)

diff -puN fs/namei.c~07-24-elevate-mnt-writers-for-callers-of-vfs-mkdir fs/namei.c
--- lxc/fs/namei.c~07-24-elevate-mnt-writers-for-callers-of-vfs-mkdir 2006-12-11
14:22:01.000000000 -0800
+++ lxc-dave/fs/namei.c 2006-12-11 14:22:01.000000000 -0800
@@ -1959,7 +1959,12 @@ asmlinkage long sys_mkdirat(int dfd, con

  if (!IS_POSIXACL(nd.dentry->d_inode))
   mode &= ~current->fs->umask;
+ error = mnt_want_write(nd.mnt);
+ if (error)
+  goto out_dput;
  error = vfs_mkdir(nd.dentry->d_inode, dentry, mode);
+ mnt_drop_write(nd.mnt);
+out_dput:
  dput(dentry);
 out_unlock:
  mutex_unlock(&nd.dentry->d_inode->i_mutex);
diff -puN fs/nfsd/nfs4recover.c~07-24-elevate-mnt-writers-for-callers-of-vfs-mkdir
fs/nfsd/nfs4recover.c
--- lxc/fs/nfsd/nfs4recover.c~07-24-elevate-mnt-writers-for-callers-of-vfs-mkdir 2006-12-11
14:22:01.000000000 -0800
+++ lxc-dave/fs/nfsd/nfs4recover.c 2006-12-11 14:22:01.000000000 -0800
@@ -156,7 +156,11 @@ nfsd4_create_clid_dir(struct nfs4_client
   dprintk("NFSD: nfsd4_create_clid_dir: DIRECTORY EXISTS\n");
   goto out_put;
  }
+ status = mnt_want_write(rec_dir.mnt);
```

```
+ if (status)
+  goto out_put;
  status = vfs_mkdir(rec_dir.dentry->d_inode, dentry, S_IRWXU);
+ mnt_drop_write(rec_dir.mnt);
 out_put:
  dput(dentry);
 out_unlock:
```

_

_____

Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

---

Subject: [PATCH 09/25] elevate write count during entire ncp_ioctl()
Posted by Dave Hansen on Mon, 11 Dec 2006 22:30:08 GMT
View Forum Message <> Reply to Message

Some ioctls need write access, but others don't.  Make a helper
function to decide when write access is needed, and take it.

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

```
 lxc-dave/fs/ncpfs/ioctl.c |   55 ++++++++++++++++++++++++++++++++++++++++++++++++-
 1 file changed, 54 insertions(+), 1 deletion(-)

diff -puN fs/ncpfs/ioctl.c~08-24-elevate-write-count-during-entire-ncp-ioctl fs/ncpfs/ioctl.c
--- lxc/fs/ncpfs/ioctl.c~08-24-elevate-write-count-during-entire-ncp-ioctl 2006-12-11
14:22:02.000000000 -0800
+++ lxc-dave/fs/ncpfs/ioctl.c 2006-12-11 14:22:02.000000000 -0800
@@ -14,6 +14,7 @@
 #include <linux/ioctl.h>
 #include <linux/time.h>
 #include <linux/mm.h>
+#include <linux/mount.h>
 #include <linux/highuid.h>
 #include <linux/smp_lock.h>
 #include <linux/vmalloc.h>
@@ -260,7 +261,7 @@ ncp_get_charsets(struct ncp_server* serv
 }
 #endif /* CONFIG_NCPFS_NLS */

-int ncp_ioctl(struct inode *inode, struct file *filp,
+static int __ncp_ioctl(struct inode *inode, struct file *filp,
        unsigned int cmd, unsigned long arg)
 {
  struct ncp_server *server = NCP_SERVER(inode);
```

```
@@ -821,6 +822,58 @@ outrel:
  return -EINVAL;
 }

+static int ncp_ioctl_need_write(unsigned int cmd)
+{
+ switch (cmd) {
+ case NCP_IOC_GET_FS_INFO:
+ case NCP_IOC_GET_FS_INFO_V2:
+ case NCP_IOC_NCPREQUEST:
+ case NCP_IOC_SETDENTRYTTL:
+ case NCP_IOC_SIGN_INIT:
+ case NCP_IOC_LOCKUNLOCK:
+ case NCP_IOC_SET_SIGN_WANTED:
+  return 1;
+ case NCP_IOC_GETOBJECTNAME:
+ case NCP_IOC_SETOBJECTNAME:
+ case NCP_IOC_GETPRIVATEDATA:
+ case NCP_IOC_SETPRIVATEDATA:
+ case NCP_IOC_SETCHARSETS:
+ case NCP_IOC_GETCHARSETS:
+ case NCP_IOC_CONN_LOGGED_IN:
+ case NCP_IOC_GETDENTRYTTL:
+ case NCP_IOC_GETMOUNTUID2:
+ case NCP_IOC_SIGN_WANTED:
+ case NCP_IOC_GETROOT:
+ case NCP_IOC_SETROOT:
+  return 0;
+ default:
+  /* unkown IOCTL command, assume write */
+  WARN_ON(1);
+ }
+ return 1;
+}
+
+int ncp_ioctl(struct inode *inode, struct file *filp,
+       unsigned int cmd, unsigned long arg)
+{
+ int ret;
+
+ if (ncp_ioctl_need_write(cmd)) {
+  /*
+   * inside the ioctl(), any failures which
+   * are because of file_permission() are
+   * -EACCESS, so it seems consistent to keep
+   *  that here.
+   */
+  if (mnt_want_write(filp->f_vfsmnt))
```

```
+   return -EACCES;
+ }
+ ret = __ncp_ioctl(inode, filp, cmd, arg);
+ if (ncp_ioctl_need_write(cmd))
+  mnt_drop_write(filp->f_vfsmnt);
+ return ret;
+}
+
 #ifdef CONFIG_COMPAT
 long ncp_compat_ioctl(struct file *file, unsigned int cmd, unsigned long arg)
 {
_
```

_____

Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

---

## Subject: [PATCH 11/25] elevate mount count for extended attributes
Posted by Dave Hansen on Mon, 11 Dec 2006 22:30:09 GMT
View Forum Message <> Reply to Message

This basically audits the callers of xattr_permission(), which
calls permission() and can perform writes to the filesystem.

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

 lxc-dave/fs/nfsd/nfs4proc.c |    7 ++++++-
 lxc-dave/fs/xattr.c         |   14 ++++++++++++++
 2 files changed, 20 insertions(+), 1 deletion(-)

```
diff -puN fs/nfsd/nfs4proc.c~10-24-elevate-mount-count-for-extended-attributes fs/nfsd/nfs4proc.c
--- lxc/fs/nfsd/nfs4proc.c~10-24-elevate-mount-count-for-extended-attributes 2006-12-11
14:22:03.000000000 -0800
+++ lxc-dave/fs/nfsd/nfs4proc.c 2006-12-11 14:22:03.000000000 -0800
@@ -626,14 +626,19 @@ nfsd4_setattr(struct svc_rqst *rqstp, st
    return status;
   }
  }
+ status = mnt_want_write(current_fh->fh_export->ex_mnt);
+ if (status)
+  return status;
  status = nfs_ok;
  if (setattr->sa_acl != NULL)
   status = nfsd4_set_nfs4_acl(rqstp, &cstate->current_fh,
         setattr->sa_acl);
  if (status)
```

```
-  return status;
+  goto out;
  status = nfsd_setattr(rqstp, &cstate->current_fh, &setattr->sa_iattr,
     0, (time_t)0);
+out:
+ mnt_drop_write(current_fh->fh_export->ex_mnt);
  return status;
 }

diff -puN fs/xattr.c~10-24-elevate-mount-count-for-extended-attributes fs/xattr.c
--- lxc/fs/xattr.c~10-24-elevate-mount-count-for-extended-attributes 2006-12-11
14:22:03.000000000 -0800
+++ lxc-dave/fs/xattr.c 2006-12-11 14:22:03.000000000 -0800
@@ -12,6 +12,7 @@
 #include <linux/smp_lock.h>
 #include <linux/file.h>
 #include <linux/xattr.h>
+#include <linux/mount.h>
 #include <linux/namei.h>
 #include <linux/security.h>
 #include <linux/syscalls.h>
@@ -237,7 +238,11 @@ sys_setxattr(char __user *path, char __u
  error = user_path_walk(path, &nd);
  if (error)
   return error;
+ error = mnt_want_write(nd.mnt);
+ if (error)
+  return error;
  error = setxattr(nd.dentry, name, value, size, flags);
+ mnt_drop_write(nd.mnt);
  path_release(&nd);
  return error;
 }
@@ -252,7 +257,11 @@ sys_lsetxattr(char __user *path, char __
  error = user_path_walk_link(path, &nd);
  if (error)
   return error;
+ error = mnt_want_write(nd.mnt);
+ if (error)
+  return error;
  error = setxattr(nd.dentry, name, value, size, flags);
+ mnt_drop_write(nd.mnt);
  path_release(&nd);
  return error;
 }
@@ -268,9 +277,14 @@ sys_fsetxattr(int fd, char __user *name,
  f = fget(fd);
  if (!f)
```

```
   return error;
+ error = mnt_want_write(f->f_vfsmnt);
+ if (error)
+  goto out_fput;
  dentry = f->f_path.dentry;
  audit_inode(NULL, dentry->d_inode);
  error = setxattr(dentry, name, value, size, flags);
+ mnt_drop_write(f->f_vfsmnt);
+out_fput:
  fput(f);
  return error;
 }
_
```

---

Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

---

## Subject: [PATCH 12/25] mount_is_safe(): add comment
Posted by Dave Hansen on Mon, 11 Dec 2006 22:30:10 GMT
View Forum Message <> Reply to Message

This area of code is currently #ifdef'd out, so add a comment
for the time when it is actually used.

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

 lxc-dave/fs/namespace.c |    4 ++++
 1 file changed, 4 insertions(+)

```
diff -puN fs/namespace.c~11-24-mount-is-safe-add-comment fs/namespace.c
--- lxc/fs/namespace.c~11-24-mount-is-safe-add-comment 2006-12-11 14:22:04.000000000 -0800
+++ lxc-dave/fs/namespace.c 2006-12-11 14:22:04.000000000 -0800
@@ -700,6 +700,10 @@ static int mount_is_safe(struct nameidat
   if (current->uid != nd->dentry->d_inode->i_uid)
    return -EPERM;
  }
+ /*
+  * We will eventually check for the mnt->writer_count here,
+  * but since the code is not used now, skip it - Dave Hansen
+  */
  if (vfs_permission(nd, MAY_WRITE))
   return -EPERM;
  return 0;
_
```

Subject: [PATCH 14/25] elevate write count over calls to vfs_rename()
Posted by Dave Hansen on Mon, 11 Dec 2006 22:30:11 GMT
View Forum Message <> Reply to Message

This does create a little helper in the NFS code to
make an if() a little bit less ugly.

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

 lxc-dave/fs/namei.c    |    4 ++++
 lxc-dave/fs/nfsd/vfs.c |   23 +++++++++++++++++++----
 2 files changed, 23 insertions(+), 4 deletions(-)

diff -puN fs/namei.c~13-24-elevate-write-count-over-calls-to-vfs-rename fs/namei.c
--- lxc/fs/namei.c~13-24-elevate-write-count-over-calls-to-vfs-rename 2006-12-11
14:22:05.000000000 -0800
+++ lxc-dave/fs/namei.c 2006-12-11 14:22:05.000000000 -0800
@@ -2563,8 +2563,12 @@ static int do_rename(int olddfd, const c
  if (new_dentry == trap)
   goto exit5;

+ error = mnt_want_write(oldnd.mnt);
+ if (error)
+  goto exit5;
  error = vfs_rename(old_dir->d_inode, old_dentry,
        new_dir->d_inode, new_dentry);
+ mnt_drop_write(oldnd.mnt);
 exit5:
  dput(new_dentry);
 exit4:
diff -puN fs/nfsd/vfs.c~13-24-elevate-write-count-over-calls-to-vfs-rename fs/nfsd/vfs.c
--- lxc/fs/nfsd/vfs.c~13-24-elevate-write-count-over-calls-to-vfs-rename 2006-12-11
14:22:05.000000000 -0800
+++ lxc-dave/fs/nfsd/vfs.c 2006-12-11 14:22:05.000000000 -0800
@@ -1566,6 +1566,14 @@ out_nfserr:
  goto out_unlock;
 }

+static inline int svc_msnfs(struct svc_fh *ffhp)
+{
+#ifdef MSNFS
+ return (ffhp->fh_export->ex_flags & NFSEXP_MSNFS);

```
+#else
+ return 0;
+#endif
+}
 /*
  * Rename a file
  * N.B. After this call _both_ ffhp and tfhp need an fh_put
@@ -1627,13 +1635,20 @@ nfsd_rename(struct svc_rqst *rqstp, stru
  if (ndentry == trap)
   goto out_dput_new;

-#ifdef MSNFS
- if ((ffhp->fh_export->ex_flags & NFSEXP_MSNFS) &&
+ if (svc_msnfs(ffhp) &&
   ((atomic_read(&odentry->d_count) > 1)
    || (atomic_read(&ndentry->d_count) > 1))) {
    host_err = -EPERM;
- } else
-#endif
+   goto out_dput_new;
+ }
+
+ host_err = -EXDEV;
+ if (ffhp->fh_export->ex_mnt != tfhp->fh_export->ex_mnt)
+  goto out_dput_new;
+ host_err = mnt_want_write(ffhp->fh_export->ex_mnt);
+ if (host_err)
+  goto out_dput_new;
+
  host_err = vfs_rename(fdir, odentry, tdir, ndentry);
  if (!host_err && EX_ISSYNC(tfhp->fh_export)) {
   host_err = nfsd_sync_dir(tdentry);
_
```



## Subject: [PATCH 15/25] elevate write count files are open()ed
Posted by Dave Hansen on Mon, 11 Dec 2006 22:30:12 GMT
View Forum Message <> Reply to Message

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.

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

lxc-dave/fs/file_table.c | 5 ++++-
lxc-dave/fs/namei.c | 22 ++++++++++++++++++----
lxc-dave/ipc/mqueue.c | 3 +++
3 files changed, 25 insertions(+), 5 deletions(-)

```
diff -puN fs/file_table.c~14-24-tricky-elevate-write-count-files-are-open-ed fs/file_table.c
--- lxc/fs/file_table.c~14-24-tricky-elevate-write-count-files-are-open-ed 2006-12-11
14:22:06.000000000 -0800
+++ lxc-dave/fs/file_table.c 2006-12-11 14:22:06.000000000 -0800
@@ -202,8 +202,11 @@ 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;
diff -puN fs/namei.c~14-24-tricky-elevate-write-count-files-are-open-ed fs/namei.c
--- lxc/fs/namei.c~14-24-tricky-elevate-write-count-files-are-open-ed 2006-12-11
14:22:06.000000000 -0800
+++ lxc-dave/fs/namei.c 2006-12-11 14:22:06.000000000 -0800
@@ -1544,8 +1544,17 @@ 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;
+  if (IS_RDONLY(inode))
+   return -EROFS;
+ }
  /*
   * An append-only file must be opened in append mode for writing.
   */
@@ -1684,14 +1693,17 @@ 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;
@@ -1729,6 +1741,8 @@ ok:
   goto exit;
  return 0;

+exit_mutex_unlock:
+ mutex_unlock(&dir->d_inode->i_mutex);
 exit_dput:
  dput_path(&path, nd);
 exit:
diff -puN ipc/mqueue.c~14-24-tricky-elevate-write-count-files-are-open-ed ipc/mqueue.c
--- lxc/ipc/mqueue.c~14-24-tricky-elevate-write-count-files-are-open-ed 2006-12-11
14:22:06.000000000 -0800
+++ lxc-dave/ipc/mqueue.c 2006-12-11 14:22:06.000000000 -0800
@@ -687,6 +687,9 @@ 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);
  }
_
```



Subject: [PATCH 16/25] elevate writer count for do_sys_truncate()
Posted by Dave Hansen on Mon, 11 Dec 2006 22:30:13 GMT
View Forum Message <> Reply to Message

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

 lxc-dave/fs/open.c |   16 ++++++++++-----
 1 file changed, 11 insertions(+), 5 deletions(-)

```
diff -puN fs/open.c~15-24-elevate-writer-count-for-do-sys-truncate fs/open.c
--- lxc/fs/open.c~15-24-elevate-writer-count-for-do-sys-truncate 2006-12-11 14:22:07.000000000
-0800
+++ lxc-dave/fs/open.c 2006-12-11 14:22:07.000000000 -0800
@@ -241,28 +241,32 @@ static long do_sys_truncate(const char _
  if (!S_ISREG(inode->i_mode))
   goto dput_and_out;

- error = vfs_permission(&nd, MAY_WRITE);
+ error = mnt_want_write(nd.mnt);
  if (error)
   goto dput_and_out;

+ error = vfs_permission(&nd, MAY_WRITE);
+ if (error)
+  goto mnt_drop_write_and_out;
+
  error = -EROFS;
  if (IS_RDONLY(inode))
-  goto dput_and_out;
+  goto mnt_drop_write_and_out;

  error = -EPERM;
  if (IS_IMMUTABLE(inode) || IS_APPEND(inode))
```

```
-  goto dput_and_out;
+  goto mnt_drop_write_and_out;

  /*
   * Make sure that there are no leases.
   */
  error = break_lease(inode, FMODE_WRITE);
  if (error)
-  goto dput_and_out;
+  goto mnt_drop_write_and_out;

  error = get_write_access(inode);
  if (error)
-  goto dput_and_out;
+  goto mnt_drop_write_and_out;

  error = locks_verify_truncate(inode, NULL, length);
  if (!error) {
@@ -271,6 +275,8 @@ static long do_sys_truncate(const char _
  }
  put_write_access(inode);

+mnt_drop_write_and_out:
+ mnt_drop_write(nd.mnt);
 dput_and_out:
  path_release(&nd);
 out:
_
```

---

Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

---

Subject: [PATCH 18/25] elevate write count for do_sys_utime() and touch_atime()
Posted by Dave Hansen on Mon, 11 Dec 2006 22:30:14 GMT
View Forum Message <> Reply to Message

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

 lxc-dave/fs/inode.c  |   20 ++++++++++++++--------
 lxc-dave/fs/utimes.c |   15 +++++++++++----
 2 files changed, 23 insertions(+), 12 deletions(-)

diff -puN fs/inode.c~17-24-elevate-write-count-for-do-sys-utime-and-touch-atime fs/inode.c
--- lxc/fs/inode.c~17-24-elevate-write-count-for-do-sys-utime-and-touch-atime 2006-12-11 14:22:08.000000000 -0800

```
+++ lxc-dave/fs/inode.c 2006-12-11 14:22:08.000000000 -0800
@@ -1167,22 +1167,23 @@ void touch_atime(struct vfsmount *mnt, s

  if (IS_RDONLY(inode))
   return;
- if (inode->i_flags & S_NOATIME)
+ if (mnt && mnt_want_write(mnt))
   return;
+ if (inode->i_flags & S_NOATIME)
+  goto out;
  if (inode->i_sb->s_flags & MS_NOATIME)
-  return;
+  goto out;
  if ((inode->i_sb->s_flags & MS_NODIRATIME) && S_ISDIR(inode->i_mode))
-  return;
+  goto out;

  /*
   * We may have a NULL vfsmount when coming from NFSD
   */
  if (mnt) {
   if (mnt->mnt_flags & MNT_NOATIME)
-   return;
+   goto out;
   if ((mnt->mnt_flags & MNT_NODIRATIME) && S_ISDIR(inode->i_mode))
-   return;
-
+   goto out;
   if (mnt->mnt_flags & MNT_RELATIME) {
    /*
     * With relative atime, only update atime if the
@@ -1193,16 +1194,19 @@ void touch_atime(struct vfsmount *mnt, s
       &inode->i_atime) < 0 &&
       timespec_compare(&inode->i_ctime,
       &inode->i_atime) < 0)
-    return;
+    goto out;
   }
  }

  now = current_fs_time(inode->i_sb);
  if (timespec_equal(&inode->i_atime, &now))
-  return;
+  goto out;

  inode->i_atime = now;
  mark_inode_dirty_sync(inode);
+out:
```

```
+ if (mnt)
+  mnt_drop_write(mnt);
 }
 EXPORT_SYMBOL(touch_atime);

diff -puN fs/utimes.c~17-24-elevate-write-count-for-do-sys-utime-and-touch-atime fs/utimes.c
--- lxc/fs/utimes.c~17-24-elevate-write-count-for-do-sys-utime-and-touch-atime 2006-12-11
14:22:08.000000000 -0800
+++ lxc-dave/fs/utimes.c 2006-12-11 14:22:08.000000000 -0800
@@ -3,6 +3,7 @@
 #include <linux/linkage.h>
 #include <linux/namei.h>
 #include <linux/sched.h>
+#include <linux/mount.h>
 #include <linux/utime.h>
 #include <asm/uaccess.h>
 #include <asm/unistd.h>
@@ -32,6 +33,10 @@ asmlinkage long sys_utime(char __user *
   goto out;
  inode = nd.dentry->d_inode;

+ error = mnt_want_write(nd.mnt);
+ if (error)
+  goto mnt_drop_write_and_out;
+
  error = -EROFS;
  if (IS_RDONLY(inode))
   goto dput_and_out;
@@ -41,7 +46,7 @@ asmlinkage long sys_utime(char __user *
  if (times) {
   error = -EPERM;
   if (IS_APPEND(inode) || IS_IMMUTABLE(inode))
-   goto dput_and_out;
+   goto mnt_drop_write_and_out;

   error = get_user(newattrs.ia_atime.tv_sec, &times->actime);
   newattrs.ia_atime.tv_nsec = 0;
@@ -49,21 +54,23 @@ asmlinkage long sys_utime(char __user *
    error = get_user(newattrs.ia_mtime.tv_sec, &times->modtime);
   newattrs.ia_mtime.tv_nsec = 0;
   if (error)
-   goto dput_and_out;
+   goto mnt_drop_write_and_out;

   newattrs.ia_valid |= ATTR_ATIME_SET | ATTR_MTIME_SET;
  } else {
                error = -EACCES;
                if (IS_IMMUTABLE(inode))
```

```
-                       goto dput_and_out;
+                       goto mnt_drop_write_and_out;

   if (current->fsuid != inode->i_uid &&
       (error = vfs_permission(&nd, MAY_WRITE)) != 0)
-   goto dput_and_out;
+   goto mnt_drop_write_and_out;
  }
  mutex_lock(&inode->i_mutex);
  error = notify_change(nd.dentry, &newattrs);
  mutex_unlock(&inode->i_mutex);
+mnt_drop_write_and_out:
+ mnt_drop_write(nd.mnt);
 dput_and_out:
  path_release(&nd);
 out:
```

–

_____

Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

---

Subject: [PATCH 19/25] sys_mknodat(): elevate write count for vfs_mknod/create()
Posted by Dave Hansen on Mon, 11 Dec 2006 22:30:15 GMT
View Forum Message <> Reply to Message

This takes care of all of the direct callers of vfs_mknod().
Since a few of these cases also handle normal file creation
as well, this also covers some calls to vfs_create().

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

```
 lxc-dave/fs/namei.c         |   12 ++++++++++
 lxc-dave/fs/nfsd/vfs.c      |    4 ++++
 lxc-dave/net/unix/af_unix.c |    4 ++++
 3 files changed, 20 insertions(+)

diff -puN fs/namei.c~18-24-sys-mknodat-elevate-write-count-for-vfs-mknod-create fs/namei.c
--- lxc/fs/namei.c~18-24-sys-mknodat-elevate-write-count-for-vfs-mknod-create 2006-12-11
14:22:09.000000000 -0800
+++ lxc-dave/fs/namei.c 2006-12-11 14:22:09.000000000 -0800
@@ -1899,14 +1899,26 @@ asmlinkage long sys_mknodat(int dfd, con
  if (!IS_ERR(dentry)) {
   switch (mode & S_IFMT) {
   case 0: case S_IFREG:
+   error = mnt_want_write(nd.mnt);
```

```
+   if (error)
+    break;
    error = vfs_create(nd.dentry->d_inode,dentry,mode,&nd);
+   mnt_drop_write(nd.mnt);
    break;
   case S_IFCHR: case S_IFBLK:
+   error = mnt_want_write(nd.mnt);
+   if (error)
+    break;
    error = vfs_mknod(nd.dentry->d_inode,dentry,mode,
      new_decode_dev(dev));
+   mnt_drop_write(nd.mnt);
    break;
   case S_IFIFO: case S_IFSOCK:
+   error = mnt_want_write(nd.mnt);
+   if (error)
+    break;
    error = vfs_mknod(nd.dentry->d_inode,dentry,mode,0);
+   mnt_drop_write(nd.mnt);
    break;
   case S_IFDIR:
    error = -EPERM;
diff -puN fs/nfsd/vfs.c~18-24-sys-mknodat-elevate-write-count-for-vfs-mknod-create fs/nfsd/vfs.c
--- lxc/fs/nfsd/vfs.c~18-24-sys-mknodat-elevate-write-count-for-vfs-mknod-create 2006-12-11
14:22:09.000000000 -0800
+++ lxc-dave/fs/nfsd/vfs.c 2006-12-11 14:22:09.000000000 -0800
@@ -665,6 +665,9 @@ nfsd_open(struct svc_rqst *rqstp, struct
  /* Disallow write access to files with the append-only bit set
   * or any access when mandatory locking enabled
   */
+ err = mnt_want_write(fhp->fh_export->ex_mnt);
+ if (err)
+  goto out_nfserr;
  err = nfserr_perm;
  if (IS_APPEND(inode) && (access & MAY_WRITE))
   goto out;
@@ -1199,6 +1202,7 @@ nfsd_create(struct svc_rqst *rqstp, stru
          printk("nfsd: bad file type %o in nfsd_create\n", type);
   host_err = -EINVAL;
  }
+ mnt_drop_write(fhp->fh_export->ex_mnt);
  if (host_err < 0)
   goto out_nfserr;

diff -puN net/unix/af_unix.c~18-24-sys-mknodat-elevate-write-count-for-vfs-mknod-create
net/unix/af_unix.c
--- lxc/net/unix/af_unix.c~18-24-sys-mknodat-elevate-write-count-for-vfs-mknod-create 2006-12-11
14:22:09.000000000 -0800
```

```
+++ lxc-dave/net/unix/af_unix.c 2006-12-11 14:22:09.000000000 -0800
@@ -816,7 +816,11 @@ static int unix_bind(struct socket *sock
    */
   mode = S_IFSOCK |
          (SOCK_INODE(sock)->i_mode & ~current->fs->umask);
+  err = mnt_want_write(nd.mnt);
+  if (err)
+   goto out_mknod_dput;
   err = vfs_mknod(nd.dentry->d_inode, dentry, mode, 0);
+  mnt_drop_write(nd.mnt);
   if (err)
    goto out_mknod_dput;
   mutex_unlock(&nd.dentry->d_inode->i_mutex);

_
```

---

Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

---

Subject: [PATCH 20/25] elevate mnt writers for vfs_unlink() callers
Posted by Dave Hansen on Mon, 11 Dec 2006 22:30:16 GMT
View Forum Message <> Reply to Message

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

 lxc-dave/fs/namei.c   |    4 ++++
 lxc-dave/ipc/mqueue.c |    5 ++++-
 2 files changed, 8 insertions(+), 1 deletion(-)

```
diff -puN fs/namei.c~19-24-elevate-mnt-writers-for-vfs-unlink-callers fs/namei.c
--- lxc/fs/namei.c~19-24-elevate-mnt-writers-for-vfs-unlink-callers 2006-12-11 14:22:10.000000000
-0800
+++ lxc-dave/fs/namei.c 2006-12-11 14:22:10.000000000 -0800
@@ -2177,7 +2177,11 @@ static long do_unlinkat(int dfd, const c
   inode = dentry->d_inode;
   if (inode)
    atomic_inc(&inode->i_count);
+  error = mnt_want_write(nd.mnt);
+  if (error)
+   goto exit2;
   error = vfs_unlink(nd.dentry->d_inode, dentry);
+  mnt_drop_write(nd.mnt);
  exit2:
   dput(dentry);
  }
diff -puN ipc/mqueue.c~19-24-elevate-mnt-writers-for-vfs-unlink-callers ipc/mqueue.c
```

```
--- lxc/ipc/mqueue.c~19-24-elevate-mnt-writers-for-vfs-unlink-callers 2006-12-11
14:22:10.000000000 -0800
+++ lxc-dave/ipc/mqueue.c 2006-12-11 14:22:10.000000000 -0800
@@ -749,8 +749,11 @@ asmlinkage long sys_mq_unlink(const char
  inode = dentry->d_inode;
  if (inode)
   atomic_inc(&inode->i_count);
-
+ err = mnt_want_write(mqueue_mnt);
+ if (err)
+  goto out_err;
  err = vfs_unlink(dentry->d_parent->d_inode, dentry);
+ mnt_drop_write(mqueue_mnt);
 out_err:
  dput(dentry);


_
```



---

Subject: [PATCH 21/25] do_rmdir(): elevate write count
Posted by Dave Hansen on Mon, 11 Dec 2006 22:30:17 GMT
View Forum Message <> Reply to Message

Elevate the write count during the vfs_rmdir() call.

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

 lxc-dave/fs/namei.c |    5 +++++
 1 file changed, 5 insertions(+)

```
diff -puN fs/namei.c~20-24-do-rmdir-elevate-write-count fs/namei.c
--- lxc/fs/namei.c~20-24-do-rmdir-elevate-write-count 2006-12-11 14:22:10.000000000 -0800
+++ lxc-dave/fs/namei.c 2006-12-11 14:22:10.000000000 -0800
@@ -2097,7 +2097,12 @@ static long do_rmdir(int dfd, const char
  error = PTR_ERR(dentry);
  if (IS_ERR(dentry))
   goto exit2;
+ error = mnt_want_write(nd.mnt);
+ if (error)
+  goto exit3;
  error = vfs_rmdir(nd.dentry->d_inode, dentry);
+ mnt_drop_write(nd.mnt);
+exit3:
```

```
  dput(dentry);
 exit2:
  mutex_unlock(&nd.dentry->d_inode->i_mutex);
```

_
_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

---

Subject: [PATCH 23/25] elevate writer count for custom struct_file
Posted by Dave Hansen on Mon, 11 Dec 2006 22:30:18 GMT
View Forum Message <> Reply to Message

Some filesystems forego the use of normal vfs calls to create
struct files.  Make sure that these users elevate the mnt writer
count.  These probably don't have any real meaning because there
is no real backing store for these mounts, but it is here for
consistency.

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

 lxc-dave/fs/file_table.c |    4 ++++
 1 file changed, 4 insertions(+)

```
diff -puN fs/file_table.c~22-24-elevate-writer-count-for-custom-struct-file fs/file_table.c
--- lxc/fs/file_table.c~22-24-elevate-writer-count-for-custom-struct-file 2006-12-11
14:22:12.000000000 -0800
+++ lxc-dave/fs/file_table.c 2006-12-11 14:22:12.000000000 -0800
@@ -164,6 +164,10 @@ 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;
 }
```

_
_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: [PATCH 24/25] honor r/w changes at do_remount() time
Posted by Dave Hansen on Mon, 11 Dec 2006 22:30:19 GMT
View Forum Message <> Reply to Message

Originally from: Herbert Poetzl <herbert@13thfloor.at>

This is the core of the read-only bind mount patch set.

Note that this does _not_ add a "ro" option directly to
the bind mount operation.  If you require such a mount,
you must first do the bind, then follow it up with a
'mount -o remount,ro' operation.

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

 lxc-dave/fs/namespace.c |   24 ++++++++++++++++++++++--
 lxc-dave/fs/open.c      |    2 +-
 2 files changed, 23 insertions(+), 3 deletions(-)

```
diff -puN fs/namespace.c~23-24-honor-r-w-changes-at-do-remount-time fs/namespace.c
--- lxc/fs/namespace.c~23-24-honor-r-w-changes-at-do-remount-time 2006-12-11
14:22:12.000000000 -0800
+++ lxc-dave/fs/namespace.c 2006-12-11 14:22:12.000000000 -0800
@@ -399,7 +399,7 @@ static int show_vfsmnt(struct seq_file *
  seq_path(m, mnt, mnt->mnt_root, " \t\n\\");
  seq_putc(m, ' ');
  mangle(m, mnt->mnt_sb->s_type->name);
- seq_puts(m, mnt->mnt_sb->s_flags & MS_RDONLY ? " ro" : " rw");
+ seq_puts(m, __mnt_is_readonly(mnt) ? " ro" : " rw");
  for (fs_infop = fs_info; fs_infop->flag; fs_infop++) {
   if (mnt->mnt_sb->s_flags & fs_infop->flag)
    seq_puts(m, fs_infop->str);
@@ -1023,6 +1023,23 @@ out:
 }
 EXPORT_SYMBOL_GPL(mnt_want_write);

+static int change_mount_flags(struct vfsmount *mnt, int ms_flags)
+{
+ int error = 0;
+ int readonly_request = 0;
+
+ if (ms_flags & MS_RDONLY)
+  readonly_request = 1;
+ if (readonly_request == __mnt_is_readonly(mnt))
+  return 0;
+
+ if (readonly_request)
+  error = mnt_make_readonly(mnt);
```

```
+ else
+  __mnt_unmake_readonly(mnt);
+ return error;
+}
+
 /*
  * change filesystem flags. dir should be a physical root of filesystem.
  * If you've mounted a non-root directory somewhere and want to do remount
@@ -1044,7 +1061,10 @@ static int do_remount(struct nameidata *
   return -EINVAL;

  down_write(&sb->s_umount);
- err = do_remount_sb(sb, flags, data, 0);
+ if (flags & MS_BIND)
+  err = change_mount_flags(nd->mnt, flags);
+ else
+  err = do_remount_sb(sb, flags, data, 0);
  if (!(sb->s_flags & MS_RDONLY))
   mnt_flags |= MNT_SB_WRITABLE;
  if (!err)
diff -puN fs/open.c~23-24-honor-r-w-changes-at-do-remount-time fs/open.c
--- lxc/fs/open.c~23-24-honor-r-w-changes-at-do-remount-time 2006-12-11 14:22:12.000000000
-0800
+++ lxc-dave/fs/open.c 2006-12-11 14:22:12.000000000 -0800
@@ -401,7 +401,7 @@ asmlinkage long sys_faccessat(int dfd, c
     special_file(nd.dentry->d_inode->i_mode))
   goto out_path_release;

- if(IS_RDONLY(nd.dentry->d_inode))
+ if(__mnt_is_readonly(nd.mnt) || IS_RDONLY(nd.dentry->d_inode))
   res = -EROFS;

 out_path_release:
_
```



---

Subject: [PATCH 25/25] add kerneldoc for i_nlink helpers
Posted by Dave Hansen on Mon, 11 Dec 2006 22:30:20 GMT
View Forum Message <> Reply to Message

I left this until the end of the series because some of the
descriptions here really don't make any sense until this
series is complete.

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

 lxc-dave/include/linux/fs.h |   44 ++++++++++++++++++++++++++++++++++++++++++
 1 file changed, 44 insertions(+)

diff -puN include/linux/fs.h~24-24-add-kerneldoc-for-i-nlink-helpers include/linux/fs.h
--- lxc/include/linux/fs.h~24-24-add-kerneldoc-for-i-nlink-helpers 2006-12-11 14:22:13.000000000 -0800
+++ lxc-dave/include/linux/fs.h 2006-12-11 14:22:13.000000000 -0800
@@ -1242,6 +1242,14 @@ static inline void mark_inode_dirty_sync
  __mark_inode_dirty(inode, I_DIRTY_SYNC);
 }

+/**
+ * inc_nlink - directly increment an inode's link count
+ * @inode: inode
+ *
+ * This is a low-level filesystem helper to replace any
+ * direct filesystem manipulation of i_nlink.  Currently,
+ * it is only here for parity with dec_nlink().
+ */
 static inline void inc_nlink(struct inode *inode)
 {
  inode->i_nlink++;
@@ -1253,6 +1261,18 @@ static inline void inode_inc_link_count(
  mark_inode_dirty(inode);
 }

+/**
+ * check_nlink - check an inode's status after direct
+ *    i_nlink modification.
+ * @inode: inode
+ *
+ * Some filesystems can not make simple incremental changes
+ * to i_nlink, most notably clustered ones.  They must do
+ * direct manipulation of i_nlink.  This function must be
+ * called after such modifications are complete to make
+ * sure that the VFS knows that the inode is going to go
+ * away.
+ */
 static inline void check_nlink(struct inode *inode)
 {
  if (inode->i_nlink)
@@ -1262,12 +1282,36 @@ static inline void check_nlink(struct in
  atomic_inc(&inode->i_sb->s_mnt_writers);
 }

```
+/**
+ * drop_nlink - directly drop an inode's link count
+ * @inode: inode
+ *
+ * This is a low-level filesystem helper to replace any
+ * direct filesystem manipulation of i_nlink.  In cases
+ * where we are attempting to track writes to the
+ * filesystem, a decrement to zero means an imminent
+ * write when the file is truncated and actually unlinked
+ * on the filesystem.
+ */
 static inline void drop_nlink(struct inode *inode)
 {
  inode->i_nlink--;
  check_nlink(inode);
 }

+/**
+ * clear_nlink - directly zero an inode's link count
+ * @inode: inode
+ *
+ * This is a low-level filesystem helper to replace any
+ * direct filesystem manipulation of i_nlink.  See
+ * drop_nlink() for why we care about i_nlink hitting zero.
+ *
+ * Note that we could do the i_state flag directly in here,
+ * but we call check_nlink() to keep the number of places
+ * where the flag is set to exactly one.  The compiler
+ * should get rid of the superfluous i_nlink check.
+ */
 static inline void clear_nlink(struct inode *inode)
 {
  inode->i_nlink = 0;
_
```



---

Subject: Re: [PATCH 01/25] filesystem helpers for custom 'struct file's
Posted by Christoph Hellwig on Tue, 12 Dec 2006 10:30:40 GMT
View Forum Message <> Reply to Message

On Mon, Dec 11, 2006 at 02:30:02PM -0800, Dave Hansen wrote:
>
>
> Some filesystems forego the vfs and may_open() and create their

> own 'struct file's.
>
> This patch creates a couple of helper functions which can be
> used by these filesystems, and will provide a unified place
> which the r/o bind mount code may patch.

Can you get rid of init_file by reordering the code in the socket
and shmem code a little bit?  I'd prefer to completly get rid
of the get_empty_filp export long-term.

Also why do you export init_file and alloc_file?  There don't seem
to be any modular users.

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers