Subject: [PATCH 1/5] Cleanup macros for distinguishing mandatory locks
Posted by Pavel Emelianov on Mon, 17 Sep 2007 07:50:15 GMT
View Forum Message <> Reply to Message

The combination of S_ISGID bit set and S_IXGRP bit unset is
used to mark the inode as "mandatory lockable" and there's a
macro for this check called MANDATORY_LOCK(inode). However,
fs/locks.c and some filesystems still perform the explicit
i_mode checking. Besides, Andrew pointed out, that this macro
is buggy itself, as it dereferences the inode arg twice.

Convert this macro into static inline function and switch
its users to it, making the code shorter and more readable.

The __mandatory_lock() helper is to be used in places where
the IS_MANDLOCK() for superblock is already known to be true.

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

```
 fs/locks.c         |   14 ++++----------
 fs/nfsd/nfs4state.c |    2 +-
 fs/nfsd/vfs.c       |    2 +-
 fs/read_write.c     |    2 +-
 include/linux/fs.h  |   21 +++++++++++++++++++----
 5 files changed, 24 insertions(+), 17 deletions(-)

diff --git a/include/linux/fs.h b/include/linux/fs.h
index 291d40b..9c519e6 100644
--- a/include/linux/fs.h
+++ b/include/linux/fs.h
@@ -1488,12 +1488,25 @@ extern int locks_mandatory_area(int, str
  * Candidates for mandatory locking have the setgid bit set
  * but no group execute bit -  an otherwise meaningless combination.
  */
-#define MANDATORY_LOCK(inode) \
- (IS_MANDLOCK(inode) && ((inode)->i_mode & (S_ISGID | S_IXGRP)) == S_ISGID)
+
+static inline int __mandatory_lock(struct inode *ino)
+{
+ return (ino->i_mode & (S_ISGID | S_IXGRP)) == S_ISGID;
+}
+
+/*
+ * ... and these candidates should be on MS_MANDLOCK mounted fs,
+ * otherwise these will be advisory locks
+ */
```

```
+
+static inline int mandatory_lock(struct inode *ino)
+{
+ return IS_MANDLOCK(ino) && __mandatory_lock(ino);
+}

 static inline int locks_verify_locked(struct inode *inode)
 {
- if (MANDATORY_LOCK(inode))
+ if (mandatory_lock(inode))
   return locks_mandatory_locked(inode);
  return 0;
 }
@@ -1504,7 +1517,7 @@ static inline int locks_verify_truncate(
         struct file *filp,
         loff_t size)
 {
- if (inode->i_flock && MANDATORY_LOCK(inode))
+ if (inode->i_flock && mandatory_lock(inode))
   return locks_mandatory_area(
    FLOCK_VERIFY_WRITE, inode, filp,
    size < inode->i_size ? size : inode->i_size,
diff --git a/fs/locks.c b/fs/locks.c
index f59d066..a71c589 100644
--- a/fs/locks.c
+++ b/fs/locks.c
@@ -1116,7 +1116,7 @@ int locks_mandatory_area(int read_write,
     * If we've been sleeping someone might have
     * changed the permissions behind our back.
     */
-   if ((inode->i_mode & (S_ISGID | S_IXGRP)) == S_ISGID)
+   if (__mandatory_lock(inode))
     continue;
   }

@@ -1755,9 +1755,7 @@ int fcntl_setlk(unsigned int fd, struct
  /* Don't allow mandatory locks on files that may be memory mapped
   * and shared.
   */
- if (IS_MANDLOCK(inode) &&
-     (inode->i_mode & (S_ISGID | S_IXGRP)) == S_ISGID &&
-     mapping_writably_mapped(filp->f_mapping)) {
+ if (mandatory_lock(inode) && mapping_writably_mapped(filp->f_mapping)) {
   error = -EAGAIN;
   goto out;
  }
@@ -1881,9 +1879,7 @@ int fcntl_setlk64(unsigned int fd, struc
  /* Don't allow mandatory locks on files that may be memory mapped
```

```
   * and shared.
   */
- if (IS_MANDLOCK(inode) &&
-     (inode->i_mode & (S_ISGID | S_IXGRP)) == S_ISGID &&
-     mapping_writably_mapped(filp->f_mapping)) {
+ if (mandatory_lock(inode) && mapping_writably_mapped(filp->f_mapping)) {
   error = -EAGAIN;
   goto out;
  }
@@ -2077,9 +2073,7 @@ static void lock_get_status(char* out, s
   out += sprintf(out, "%6s %s ",
         (fl->fl_flags & FL_ACCESS) ? "ACCESS" : "POSIX ",
         (inode == NULL) ? "*NOINODE*" :
-        (IS_MANDLOCK(inode) &&
-         (inode->i_mode & (S_IXGRP | S_ISGID)) == S_ISGID) ?
-        "MANDATORY" : "ADVISORY ");
+         mandatory_lock(inode) ? "MANDATORY" : "ADVISORY ");
  } else if (IS_FLOCK(fl)) {
   if (fl->fl_type & LOCK_MAND) {
    out += sprintf(out, "FLOCK  MSNFS     ");
diff --git a/fs/nfsd/nfs4state.c b/fs/nfsd/nfs4state.c
index 6256492..a0635d7 100644
--- a/fs/nfsd/nfs4state.c
+++ b/fs/nfsd/nfs4state.c
@@ -2030,7 +2030,7 @@ static inline int
 io_during_grace_disallowed(struct inode *inode, int flags)
 {
  return nfs4_in_grace() && (flags & (RD_STATE | WR_STATE))
-  && MANDATORY_LOCK(inode);
+  && mandatory_lock(inode);
 }

 /*
diff --git a/fs/nfsd/vfs.c b/fs/nfsd/vfs.c
index 70f2c86..3c703a7 100644
--- a/fs/nfsd/vfs.c
+++ b/fs/nfsd/vfs.c
@@ -65,7 +65,7 @@
  * locks on them because there is no way to know if the accesser has
  * the lock.
  */
-#define IS_ISMNDLK(i) (S_ISREG((i)->i_mode) && MANDATORY_LOCK(i))
+#define IS_ISMNDLK(i) (S_ISREG((i)->i_mode) && mandatory_lock(i))

 /*
  * This is a cache of readahead params that help us choose the proper
diff --git a/fs/read_write.c b/fs/read_write.c
index 507ddff..124693e 100644
```

```
--- a/fs/read_write.c
+++ b/fs/read_write.c
@@ -205,7 +205,7 @@ int rw_verify_area(int read_write, struc
  if (unlikely((pos < 0) || (loff_t) (pos + count) < 0))
   goto Einval;

- if (unlikely(inode->i_flock && MANDATORY_LOCK(inode))) {
+ if (unlikely(inode->i_flock && mandatory_lock(inode))) {
   int retval = locks_mandatory_area(
    read_write == READ ? FLOCK_VERIFY_READ : FLOCK_VERIFY_WRITE,
    inode, file, pos, count);
```