Subject: [PATCH] Wake up mandatory locks waiter on chmod (v2)
Posted by Pavel Emelianov on Mon, 17 Sep 2007 08:13:24 GMT
View Forum Message <> Reply to Message

When the process is blocked on mandatory lock and someone changes
the inode's permissions, so that the lock is no longer mandatory,
nobody wakes up the blocked process, but probably should.

Switched to use mandatory_lock() static inline function.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
Cc: J. Bruce Fields <bfields@fieldses.org>

```
 fs/attr.c         |    9 +++++++--
 fs/locks.c        |   17 ++++++++++++++++-
 include/linux/fs.h |    1 +
 3 files changed, 24 insertions(+), 3 deletions(-)

diff --git a/fs/attr.c b/fs/attr.c
index ae58bd3..da643b0 100644
--- a/fs/attr.c
+++ b/fs/attr.c
@@ -104,7 +104,7 @@ int notify_change(struct dentry * dentry
 {
  struct inode *inode = dentry->d_inode;
  mode_t mode;
- int error;
+ int error, mandatory;
  struct timespec now;
  unsigned int ia_valid = attr->ia_valid;

@@ -151,6 +151,8 @@ int notify_change(struct dentry * dentry
  if (ia_valid & ATTR_SIZE)
   down_write(&dentry->d_inode->i_alloc_sem);

+ mandatory = (inode->i_flock && mandatory_lock(inode));
+
  if (inode->i_op && inode->i_op->setattr) {
   error = security_inode_setattr(dentry, attr);
   if (!error)
@@ -171,8 +173,11 @@ int notify_change(struct dentry * dentry
  if (ia_valid & ATTR_SIZE)
   up_write(&dentry->d_inode->i_alloc_sem);

- if (!error)
+ if (!error) {
```

```
   fsnotify_change(dentry, ia_valid);
+  if (mandatory)
+   locks_wakeup_mandatory(inode);
+ }

  return error;
 }
diff --git a/fs/locks.c b/fs/locks.c
index a71c589..6481fd9 100644
--- a/fs/locks.c
+++ b/fs/locks.c
@@ -1110,7 +1110,8 @@ int locks_mandatory_area(int read_write,
    break;
   if (!(fl.fl_flags & FL_SLEEP))
    break;
-  error = wait_event_interruptible(fl.fl_wait, !fl.fl_next);
+  error = wait_event_interruptible(fl.fl_wait,
+    !fl.fl_next || !__mandatory_lock(inode));
   if (!error) {
    /*
     * If we've been sleeping someone might have
@@ -1129,6 +1130,20 @@ int locks_mandatory_area(int read_write,

 EXPORT_SYMBOL(locks_mandatory_area);

+void locks_wakeup_mandatory(struct inode *inode)
+{
+ struct file_lock *fl, **before;
+
+ lock_kernel();
+ for_each_lock(inode, before) {
+  fl = *before;
+
+  if (IS_POSIX(fl))
+   locks_wake_up_blocks(fl);
+ }
+ unlock_kernel();
+}
+
 /* We already had a lease on this file; just change its type */
 int lease_modify(struct file_lock **before, int arg)
 {
diff --git a/include/linux/fs.h b/include/linux/fs.h
index 9c519e6..215eea3 100644
--- a/include/linux/fs.h
+++ b/include/linux/fs.h
@@ -1483,6 +1483,7 @@ extern struct kset fs_subsys;
```

```
 extern int locks_mandatory_locked(struct inode *);
 extern int locks_mandatory_area(int, struct inode *, struct file *, loff_t, size_t);
+extern void locks_wakeup_mandatory(struct inode *);

 /*
  * Candidates for mandatory locking have the setgid bit set
```

Subject: Re: [PATCH] Wake up mandatory locks waiter on chmod (v2)
Posted by Trond Myklebust on Mon, 17 Sep 2007 13:55:31 GMT
View Forum Message <> Reply to Message

On Mon, 2007-09-17 at 12:13 +0400, Pavel Emelyanov wrote:
> When the process is blocked on mandatory lock and someone changes
> the inode's permissions, so that the lock is no longer mandatory,
> nobody wakes up the blocked process, but probably should.

Please explain in more detail why we need this patch.

I don't see why changing a file from taking mandatory locks to advisory locks is really a useful operation that we need to support. For one thing, we don't support changing a file from using advisory locking to mandatory locking on-the-fly. Secondly, changing the locking type certainly isn't a documented operation and quite frankly, it doesn't even appear to make sense: if the file needs mandatory locking, then that means that you have a need to protect against some untrusted application that isn't following the locking rules. Then suddenly, you declare that you will trust that application after all???

Cheers,
  Trond

Subject: Re: [PATCH] Wake up mandatory locks waiter on chmod (v2)
Posted by Pavel Emelianov on Mon, 17 Sep 2007 14:16:50 GMT
View Forum Message <> Reply to Message

Trond Myklebust wrote:
> On Mon, 2007-09-17 at 12:13 +0400, Pavel Emelyanov wrote:
>> When the process is blocked on mandatory lock and someone changes
>> the inode's permissions, so that the lock is no longer mandatory,
>> nobody wakes up the blocked process, but probably should.
>
> Please explain in more detail why we need this patch.

>From "this fixes an OOPs/deadlock/leak" POV we do not. This is just an attempt to make the locking code be more consistent and

clean.

On the other hand, as far as I see from the original comment
the code author expected the permissions to change and tried
to handle this case. In this particular code there's a BUG -
task will not be woken up. So this is a fix for this place. If
we don't want to see tasks blocked on no-longer-mandatory
locks, then we should remove the original check and comment.

> I don't see why changing a file from taking mandatory locks to advisory
> locks is really a useful operation that we need to support. For one
> thing, we don't support changing a file from using advisory locking to
> mandatory locking on-the-fly. Secondly, changing the locking type

Actually we have some code trying to do it - no mandatory locking
is allowed for already opened file.

> certainly isn't a documented operation and quite frankly, it doesn't
> even appear to make sense: if the file needs mandatory locking, then
> that means that you have a need to protect against some untrusted
> application that isn't following the locking rules. Then suddenly, you
> declare that you will trust that application after all???

Not trust, but keep things look as they are described in docs. I.e.
two conditions "inode is marked as mandlock" and "task cannot proceed
with opening mand.lock-ed file" must go in pair.

> Cheers,
> Trond

Thanks,
Pavel

Subject: Re: [PATCH] Wake up mandatory locks waiter on chmod (v2)
Posted by Trond Myklebust on Mon, 17 Sep 2007 16:00:50 GMT
View Forum Message <> Reply to Message

On Mon, 2007-09-17 at 18:16 +0400, Pavel Emelyanov wrote:
> Trond Myklebust wrote:
> > On Mon, 2007-09-17 at 12:13 +0400, Pavel Emelyanov wrote:
> >> When the process is blocked on mandatory lock and someone changes
> >> the inode's permissions, so that the lock is no longer mandatory,
> >> nobody wakes up the blocked process, but probably should.
> >
> > Please explain in more detail why we need this patch.
>
> From "this fixes an OOPs/deadlock/leak" POV we do not. This is

> just an attempt to make the locking code be more consistent and
> clean.

Why do you think we get a deadlock or leak? AFAICS if the user turns off mandatory locks on the file, then the existing locks default back into advisory locks which use the same notification mechanism as the mandatory locks.

IOW: the process that is waiting in locks_mandatory_area() will be released as soon as the advisory lock is dropped. If that theory is broken in practice, then that is the bug that we need to fix. We neither want to add a load of locking crap to notify_change(), nor should we need to.

Cheers
  Trond

Subject: Re: [PATCH] Wake up mandatory locks waiter on chmod (v2)
Posted by Pavel Emelianov on Tue, 18 Sep 2007 06:33:26 GMT
View Forum Message <> Reply to Message

Trond Myklebust wrote:
> On Mon, 2007-09-17 at 18:16 +0400, Pavel Emelyanov wrote:
>> Trond Myklebust wrote:
>>> On Mon, 2007-09-17 at 12:13 +0400, Pavel Emelyanov wrote:
>>>> When the process is blocked on mandatory lock and someone changes
>>>> the inode's permissions, so that the lock is no longer mandatory,
>>>> nobody wakes up the blocked process, but probably should.
>>> Please explain in more detail why we need this patch.
>> From "this fixes an OOPs/deadlock/leak" POV we do not. This is
>> just an attempt to make the locking code be more consistent and
>> clean.
>
> Why do you think we get a deadlock or leak? AFAICS if the user turns off

I didn't' tell that.

> mandatory locks on the file, then the existing locks default back into
> advisory locks which use the same notification mechanism as the
> mandatory locks.

True.

> IOW: the process that is waiting in locks_mandatory_area() will be
> released as soon as the advisory lock is dropped. If that theory is
> broken in practice, then that is the bug that we need to fix. We neither
> want to add a load of locking crap to notify_change(), nor should we

> need to.

We have this for inotify already. Adding wakeup for mandatory lock
is not that bad.

Anyway - I noticed, that the system state can become not consistent
and proposed the way to fix it. If this inconsistency is not a big
deal, and nobody cares, than I'm fine with forgetting this patch,
since I have no other arguments to protect it, but "this is just not
very nice without this patch".

> Cheers
> Trond
>
>

Thanks,
Pavel

Subject: Re: [PATCH] Wake up mandatory locks waiter on chmod (v2)
Posted by bfields on Tue, 18 Sep 2007 15:19:57 GMT
View Forum Message <> Reply to Message

On Tue, Sep 18, 2007 at 10:33:26AM +0400, Pavel Emelyanov wrote:
> Trond Myklebust wrote:
> > IOW: the process that is waiting in locks_mandatory_area() will be
> > released as soon as the advisory lock is dropped. If that theory is
> > broken in practice, then that is the bug that we need to fix. We neither
> > want to add a load of locking crap to notify_change(), nor should we
> > need to.
>
> We have this for inotify already. Adding wakeup for mandatory lock
> is not that bad.
>
> Anyway - I noticed, that the system state can become not consistent
> and proposed the way to fix it. If this inconsistency is not a big
> deal, and nobody cares, than I'm fine with forgetting this patch,
> since I have no other arguments to protect it, but "this is just not
> very nice without this patch".

Maybe this should be documented, e.g. in fcntl(2). I'm not sure exactly
what we'd say--we probably don't want to commit to the current behavior.
Maybe something like "behavior is undefined when setting or clearing
mandatory locking on a file while it is locked".

--b.

Subject: Re: [PATCH] Wake up mandatory locks waiter on chmod (v2)
Posted by Trond Myklebust on Tue, 18 Sep 2007 16:14:55 GMT
View Forum Message <> Reply to Message

On Tue, 2007-09-18 at 11:19 -0400, J. Bruce Fields wrote:
> Maybe this should be documented, e.g. in fcntl(2). I'm not sure exactly
> what we'd say--we probably don't want to commit to the current behavior.
> Maybe something like "behavior is undefined when setting or clearing
> mandatory locking on a file while it is locked".

The behaviour is pretty much undefined if you set/clear mandatory
locking on the file while some application has it open. It is hard to
see how you can avoid that unless you exclude simultaneous chmod,
read(), write(), and fcntl(SETLK) operations.

Note also that strictly speaking, we're not even compliant with the
System V behaviour on read() and write(). See:

http://www.unix.org.ua/orelly/networking_2ndEd/nfs/ch11_01.htm
and
http://docs.sun.com/app/docs/doc/801-6736/6i13fom0a?l=en&a=view&q=mandatory+lock

According to these docs, we should be wrapping each and every read() and
write() syscall with a mandatory lock. The fact that we're not, and yet
still not seeing any complaints just goes to show how few people are
actually using and relying on this...

Trond

Subject: Re: [PATCH] Wake up mandatory locks waiter on chmod (v2)
Posted by bfields on Tue, 18 Sep 2007 16:52:20 GMT
View Forum Message <> Reply to Message

On Tue, Sep 18, 2007 at 12:14:55PM -0400, Trond Myklebust wrote:
> Note also that strictly speaking, we're not even compliant with the
> System V behaviour on read() and write(). See:
>
> http://www.unix.org.ua/orelly/networking_2ndEd/nfs/ch11_01.htm
> and
> http://docs.sun.com/app/docs/doc/801-6736/6i13fom0a?l=en&a=view&q=mandatory+lock
>
> According to these docs, we should be wrapping each and every read() and
> write() syscall with a mandatory lock. The fact that we're not, and yet
> still not seeing any complaints just goes to show how few people are
> actually using and relying on this...

So currently there's nothing to prevent this:

- write passes locks_mandatory_area() checks
- get mandatory lock
- read old data
- write updates file data
- read new data

You can see the data change even while you hold a mandatory lock that should exclude writes.

Similarly you might think that an application could prevent anyone from seeing the intermediate state of a file while it performs a series of writes under an exclusive mandatory lock, but actually there's nothing to stop a read in progress from racing with acquisition of the lock.

Unless I'm missing something, that makes our mandatory lock implementation pretty pointless. I wish we could either fix it or just ditch it, but I suppose either option would be unpopular.

--b.

Subject: Re: [PATCH] Wake up mandatory locks waiter on chmod (v2)
Posted by Trond Myklebust on Tue, 18 Sep 2007 16:54:56 GMT
View Forum Message <> Reply to Message

On Tue, 2007-09-18 at 12:52 -0400, J. Bruce Fields wrote:
> On Tue, Sep 18, 2007 at 12:14:55PM -0400, Trond Myklebust wrote:
> > Note also that strictly speaking, we're not even compliant with the
> > System V behaviour on read() and write(). See:
> >
> > http://www.unix.org.ua/orelly/networking_2ndEd/nfs/ch11_01.htm
> > and
> > http://docs.sun.com/app/docs/doc/801-6736/6i13fom0a?l=en&a=view&q=mandatory+lock
> >
> > According to these docs, we should be wrapping each and every read() and
> > write() syscall with a mandatory lock. The fact that we're not, and yet
> > still not seeing any complaints just goes to show how few people are
> > actually using and relying on this...
>
> So currently there's nothing to prevent this:
>
> - write passes locks_mandatory_area() checks
> - get mandatory lock
> - read old data
> - write updates file data
> - read new data
>

> You can see the data change even while you hold a mandatory lock that
> should exclude writes.
>
> Similarly you might think that an application could prevent anyone from
> seeing the intermediate state of a file while it performs a series of
> writes under an exclusive mandatory lock, but actually there's nothing
> to stop a read in progress from racing with acquisition of the lock.
>
> Unless I'm missing something, that makes our mandatory lock
> implementation pretty pointless. I wish we could either fix it or just
> ditch it, but I suppose either option would be unpopular.

It gets even better when you throw mmap() into the mix :-)

Trond

Subject: Re: [PATCH] Wake up mandatory locks waiter on chmod (v2)
Posted by bfields on Tue, 18 Sep 2007 17:40:16 GMT
View Forum Message <> Reply to Message

On Tue, Sep 18, 2007 at 12:54:56PM -0400, Trond Myklebust wrote:
> On Tue, 2007-09-18 at 12:52 -0400, J. Bruce Fields wrote:
> > So currently there's nothing to prevent this:
> >
> >   - write passes locks_mandatory_area() checks
> > - get mandatory lock
> > - read old data
> >   - write updates file data
> > - read new data
> >
> > You can see the data change even while you hold a mandatory lock that
> > should exclude writes.
> >
> > Similarly you might think that an application could prevent anyone from
> > seeing the intermediate state of a file while it performs a series of
> > writes under an exclusive mandatory lock, but actually there's nothing
> > to stop a read in progress from racing with acquisition of the lock.
> >
> > Unless I'm missing something, that makes our mandatory lock
> > implementation pretty pointless. I wish we could either fix it or just
> > ditch it, but I suppose either option would be unpopular.
>
> It gets even better when you throw mmap() into the mix :-)

Hm. Documentation/mandatory.txt claims that it mandatory locks and
mmap() with MAP_SHARED exclude each other, but I can't see where that's
enfoced. That file doesn't make any mention of the above race.

So for now I think someone should update that file and fcntl(2) to
mention these problems and to recommend rather strongly against using
mandatory locking.

--b.

Subject: Re: [PATCH] Wake up mandatory locks waiter on chmod (v2)
Posted by Hugh Dickins on Tue, 18 Sep 2007 18:38:52 GMT
View Forum Message <> Reply to Message

On Tue, 18 Sep 2007, J. Bruce Fields wrote:
> On Tue, Sep 18, 2007 at 12:54:56PM -0400, Trond Myklebust wrote:
> >
> > It gets even better when you throw mmap() into the mix :-)
>
> Hm. Documentation/mandatory.txt claims that it mandatory locks and
> mmap() with MAP_SHARED exclude each other, but I can't see where that's
> enfoced. That file doesn't make any mention of the above race.

I believe the locks_verify_locked() call from mm/mmap.c prevents
mmap'ing shared-write a file with mandatory locks in force;
and the mapping_writably_mapped() calls from fs/locks.c prevent
mandatory locking on a file while it's mmap'ed shared-write.

Though I think there's no lock to prevent those checks racing,
so it's not quite watertight.

Hugh

Subject: [PATCH 1/2] Documentation: move mandatory locking documentation to filesystems/
Posted by bfields on Tue, 25 Sep 2007 16:55:51 GMT
View Forum Message <> Reply to Message

Shouldn't this mandatory-locking documentation be in the
Documentation/filesystems directory?

Give it a more descriptive name while we're at it, and update 00-INDEX
with a more inclusive description of Documentation/filesystems (which
has already talked about more than just individual filesystems).

Signed-off-by: J. Bruce Fields <bfields@citi.umich.edu>

On Tue, Sep 18, 2007 at 01:40:16PM -0400, bfields wrote:
> Hm. Documentation/mandatory.txt claims that it mandatory locks and
> mmap() with MAP_SHARED exclude each other, but I can't see where that's
> enfoced. That file doesn't make any mention of the above race.
>
> So for now I think someone should update that file and fcntl(2) to
> mention these problems and to recommend rather strongly against using
> mandatory locking.

Anyone have an objection if I submit these two patches to Linus for
2.6.24?

--b.

```
 Documentation/00-INDEX                      |    4 +-
 Documentation/filesystems/00-INDEX          |    2 +
 Documentation/filesystems/mandatory-locking.txt |  152 ++++++++++++++++++++++++
 Documentation/locks.txt                     |   10 +-
 Documentation/mandatory.txt                 |  152 ----------------------
 5 files changed, 160 insertions(+), 160 deletions(-)
 create mode 100644 Documentation/filesystems/mandatory-locking.txt
 delete mode 100644 Documentation/mandatory.txt

diff --git a/Documentation/00-INDEX b/Documentation/00-INDEX
index 43e89b1..910473c 100644
--- a/Documentation/00-INDEX
+++ b/Documentation/00-INDEX
@@ -145,7 +145,7 @@ fb/
 feature-removal-schedule.txt
  - list of files and features that are going to be removed.
 filesystems/
- - directory with info on the various filesystems that Linux supports.
+ - info on the vfs and the various filesystems that Linux supports.
 firmware_class/
  - request_firmware() hotplug interface info.
 floppy.txt
@@ -240,8 +240,6 @@ m68k/
  - directory with info about Linux on Motorola 68k architecture.
 magic-number.txt
  - list of magic numbers used to mark/protect kernel data structures.
-mandatory.txt
- - info on the Linux implementation of Sys V mandatory file locking.
 mca.txt
  - info on supporting Micro Channel Architecture (e.g. PS/2) systems.
 md.txt
diff --git a/Documentation/filesystems/00-INDEX b/Documentation/filesystems/00-INDEX
index 59db1bc..73c8e94 100644
```

```
--- a/Documentation/filesystems/00-INDEX
+++ b/Documentation/filesystems/00-INDEX
@@ -52,6 +52,8 @@ isofs.txt
  - info and mount options for the ISO 9660 (CDROM) filesystem.
 jfs.txt
  - info and mount options for the JFS filesystem.
+mandatory
+ - info on the Linux implementation of Sys V mandatory file locking.
 ncpfs.txt
  - info on Novell Netware(tm) filesystem using NCP protocol.
 ntfs.txt
diff --git a/Documentation/filesystems/mandatory-locking.txt
b/Documentation/filesystems/mandatory-locking.txt
new file mode 100644
index 0000000..bc449d4
--- /dev/null
+++ b/Documentation/filesystems/mandatory-locking.txt
@@ -0,0 +1,152 @@
+ Mandatory File Locking For The Linux Operating System
+
+  Andy Walker <andy@lysaker.kvaerner.no>
+
+      15 April 1996
+
+
+1. What is  mandatory locking?
+------------------------------
+
+Mandatory locking is kernel enforced file locking, as opposed to the more usual
+cooperative file locking used to guarantee sequential access to files among
+processes. File locks are applied using the flock() and fcntl() system calls
+(and the lockf() library routine which is a wrapper around fcntl().) It is
+normally a process' responsibility to check for locks on a file it wishes to
+update, before applying its own lock, updating the file and unlocking it again.
+The most commonly used example of this (and in the case of sendmail, the most
+troublesome) is access to a user's mailbox. The mail user agent and the mail
+transfer agent must guard against updating the mailbox at the same time, and
+prevent reading the mailbox while it is being updated.
+
+In a perfect world all processes would use and honour a cooperative, or
+"advisory" locking scheme. However, the world isn't perfect, and there's
+a lot of poorly written code out there.
+
+In trying to address this problem, the designers of System V UNIX came up
+with a "mandatory" locking scheme, whereby the operating system kernel would
+block attempts by a process to write to a file that another process holds a
+"read" -or- "shared" lock on, and block attempts to both read and write to a
+file that a process holds a "write " -or- "exclusive" lock on.
```

+
+The System V mandatory locking scheme was intended to have as little impact as
+possible on existing user code. The scheme is based on marking individual files
+as candidates for mandatory locking, and using the existing fcntl()/lockf()
+interface for applying locks just as if they were normal, advisory locks.
+
+Note 1: In saying "file" in the paragraphs above I am actually not telling
+the whole truth. System V locking is based on fcntl(). The granularity of
+fcntl() is such that it allows the locking of byte ranges in files, in addition
+to entire files, so the mandatory locking rules also have byte level
+granularity.
+
+Note 2: POSIX.1 does not specify any scheme for mandatory locking, despite
+borrowing the fcntl() locking scheme from System V. The mandatory locking
+scheme is defined by the System V Interface Definition (SVID) Version 3.
+
+2. Marking a file for mandatory locking
+---------------------------------------
+
+A file is marked as a candidate for mandatory locking by setting the group-id
+bit in its file mode but removing the group-execute bit. This is an otherwise
+meaningless combination, and was chosen by the System V implementors so as not
+to break existing user programs.
+
+Note that the group-id bit is usually automatically cleared by the kernel when
+a setgid file is written to. This is a security measure. The kernel has been
+modified to recognize the special case of a mandatory lock candidate and to
+refrain from clearing this bit. Similarly the kernel has been modified not
+to run mandatory lock candidates with setgid privileges.
+
+3. Available implementations
+----------------------------
+
+I have considered the implementations of mandatory locking available with
+SunOS 4.1.x, Solaris 2.x and HP-UX 9.x.
+
+Generally I have tried to make the most sense out of the behaviour exhibited
+by these three reference systems. There are many anomalies.
+
+All the reference systems reject all calls to open() for a file on which
+another process has outstanding mandatory locks. This is in direct
+contravention of SVID 3, which states that only calls to open() with the
+O_TRUNC flag set should be rejected. The Linux implementation follows the SVID
+definition, which is the "Right Thing", since only calls with O_TRUNC can
+modify the contents of the file.
+
+HP-UX even disallows open() with O_TRUNC for a file with advisory locks, not
+just mandatory locks. That would appear to contravene POSIX.1.

+
+mmap() is another interesting case. All the operating systems mentioned
+prevent mandatory locks from being applied to an mmap()'ed file, but HP-UX
+also disallows advisory locks for such a file. SVID actually specifies the
+paranoid HP-UX behaviour.
+
+In my opinion only MAP_SHARED mappings should be immune from locking, and then
+only from mandatory locks - that is what is currently implemented.
+
+SunOS is so hopeless that it doesn't even honour the O_NONBLOCK flag for
+mandatory locks, so reads and writes to locked files always block when they
+should return EAGAIN.
+
+I'm afraid that this is such an esoteric area that the semantics described
+below are just as valid as any others, so long as the main points seem to
+agree.
+
+4. Semantics
+------------
+
+1. Mandatory locks can only be applied via the fcntl()/lockf() locking
+   interface - in other words the System V/POSIX interface. BSD style
+   locks using flock() never result in a mandatory lock.
+
+2. If a process has locked a region of a file with a mandatory read lock, then
+   other processes are permitted to read from that region. If any of these
+   processes attempts to write to the region it will block until the lock is
+   released, unless the process has opened the file with the O_NONBLOCK
+   flag in which case the system call will return immediately with the error
+   status EAGAIN.
+
+3. If a process has locked a region of a file with a mandatory write lock, all
+   attempts to read or write to that region block until the lock is released,
+   unless a process has opened the file with the O_NONBLOCK flag in which case
+   the system call will return immediately with the error status EAGAIN.
+
+4. Calls to open() with O_TRUNC, or to creat(), on a existing file that has
+   any mandatory locks owned by other processes will be rejected with the
+   error status EAGAIN.
+
+5. Attempts to apply a mandatory lock to a file that is memory mapped and
+   shared (via mmap() with MAP_SHARED) will be rejected with the error status
+   EAGAIN.
+
+6. Attempts to create a shared memory map of a file (via mmap() with MAP_SHARED)
+   that has any mandatory locks in effect will be rejected with the error status
+   EAGAIN.
+

```
+5. Which system calls are affected?
+---------------------------------
+
+Those which modify a file's contents, not just the inode. That gives read(),
+write(), readv(), writev(), open(), creat(), mmap(), truncate() and
+ftruncate(). truncate() and ftruncate() are considered to be "write" actions
+for the purposes of mandatory locking.
+
+The affected region is usually defined as stretching from the current position
+for the total number of bytes read or written. For the truncate calls it is
+defined as the bytes of a file removed or added (we must also consider bytes
+added, as a lock can specify just "the whole file", rather than a specific
+range of bytes.)
+
+Note 3: I may have overlooked some system calls that need mandatory lock
+checking in my eagerness to get this code out the door. Please let me know, or
+better still fix the system calls yourself and submit a patch to me or Linus.
+
+6. Warning!
+-----------
+
+Not even root can override a mandatory lock, so runaway processes can wreak
+havoc if they lock crucial files. The way around it is to change the file
+permissions (remove the setgid bit) before trying to read or write to it.
+Of course, that might be a bit tricky if the system is hung :-(
+
diff --git a/Documentation/locks.txt b/Documentation/locks.txt
index e3b402e..fab857a 100644
--- a/Documentation/locks.txt
+++ b/Documentation/locks.txt
@@ -53,11 +53,11 @@ fcntl(), with all the problems that implies.
 1.3 Mandatory Locking As A Mount Option
 ---------------------------------------

-Mandatory locking, as described in 'Documentation/mandatory.txt' was prior
-to this release a general configuration option that was valid for all
-mounted filesystems. This had a number of inherent dangers, not the least
-of which was the ability to freeze an NFS server by asking it to read a
-file for which a mandatory lock existed.
+Mandatory locking, as described in 'Documentation/filesystems/mandatory.txt'
+was prior to this release a general configuration option that was valid for
+all mounted filesystems.  This had a number of inherent dangers, not the
+least of which was the ability to freeze an NFS server by asking it to read
+a file for which a mandatory lock existed.

 From this release of the kernel, mandatory locking can be turned on and off
 on a per-filesystem basis, using the mount options 'mand' and 'nomand'.
diff --git a/Documentation/mandatory.txt b/Documentation/mandatory.txt
```

```
deleted file mode 100644
index bc449d4..0000000
--- a/Documentation/mandatory.txt
+++ /dev/null
@@ -1,152 +0,0 @@
- Mandatory File Locking For The Linux Operating System
-
-  Andy Walker <andy@lysaker.kvaerner.no>
-
-      15 April 1996
-
-
-1. What is  mandatory locking?
-------------------------------
-
-Mandatory locking is kernel enforced file locking, as opposed to the more usual
-cooperative file locking used to guarantee sequential access to files among
-processes. File locks are applied using the flock() and fcntl() system calls
-(and the lockf() library routine which is a wrapper around fcntl().) It is
-normally a process' responsibility to check for locks on a file it wishes to
-update, before applying its own lock, updating the file and unlocking it again.
-The most commonly used example of this (and in the case of sendmail, the most
-troublesome) is access to a user's mailbox. The mail user agent and the mail
-transfer agent must guard against updating the mailbox at the same time, and
-prevent reading the mailbox while it is being updated.
-
-In a perfect world all processes would use and honour a cooperative, or
-"advisory" locking scheme. However, the world isn't perfect, and there's
-a lot of poorly written code out there.
-
-In trying to address this problem, the designers of System V UNIX came up
-with a "mandatory" locking scheme, whereby the operating system kernel would
-block attempts by a process to write to a file that another process holds a
-"read" -or- "shared" lock on, and block attempts to both read and write to a
-file that a process holds a "write " -or- "exclusive" lock on.
-
-The System V mandatory locking scheme was intended to have as little impact as
-possible on existing user code. The scheme is based on marking individual files
-as candidates for mandatory locking, and using the existing fcntl()/lockf()
-interface for applying locks just as if they were normal, advisory locks.
-
-Note 1: In saying "file" in the paragraphs above I am actually not telling
-the whole truth. System V locking is based on fcntl(). The granularity of
-fcntl() is such that it allows the locking of byte ranges in files, in addition
-to entire files, so the mandatory locking rules also have byte level
-granularity.
-
-Note 2: POSIX.1 does not specify any scheme for mandatory locking, despite
```

-borrowing the fcntl() locking scheme from System V. The mandatory locking
-scheme is defined by the System V Interface Definition (SVID) Version 3.
-
-2. Marking a file for mandatory locking

-
-A file is marked as a candidate for mandatory locking by setting the group-id
-bit in its file mode but removing the group-execute bit. This is an otherwise
-meaningless combination, and was chosen by the System V implementors so as not
-to break existing user programs.
-
-Note that the group-id bit is usually automatically cleared by the kernel when
-a setgid file is written to. This is a security measure. The kernel has been
-modified to recognize the special case of a mandatory lock candidate and to
-refrain from clearing this bit. Similarly the kernel has been modified not
-to run mandatory lock candidates with setgid privileges.
-
-3. Available implementations

-
-I have considered the implementations of mandatory locking available with
-SunOS 4.1.x, Solaris 2.x and HP-UX 9.x.
-
-Generally I have tried to make the most sense out of the behaviour exhibited
-by these three reference systems. There are many anomalies.
-
-All the reference systems reject all calls to open() for a file on which
-another process has outstanding mandatory locks. This is in direct
-contravention of SVID 3, which states that only calls to open() with the
-O_TRUNC flag set should be rejected. The Linux implementation follows the SVID
-definition, which is the "Right Thing", since only calls with O_TRUNC can
-modify the contents of the file.
-
-HP-UX even disallows open() with O_TRUNC for a file with advisory locks, not
-just mandatory locks. That would appear to contravene POSIX.1.
-
-mmap() is another interesting case. All the operating systems mentioned
-prevent mandatory locks from being applied to an mmap()'ed file, but HP-UX
-also disallows advisory locks for such a file. SVID actually specifies the
-paranoid HP-UX behaviour.
-
-In my opinion only MAP_SHARED mappings should be immune from locking, and then
-only from mandatory locks - that is what is currently implemented.
-
-SunOS is so hopeless that it doesn't even honour the O_NONBLOCK flag for
-mandatory locks, so reads and writes to locked files always block when they
-should return EAGAIN.
-

-I'm afraid that this is such an esoteric area that the semantics described
-below are just as valid as any others, so long as the main points seem to
-agree.
-
-4. Semantics

-
-1. Mandatory locks can only be applied via the fcntl()/lockf() locking
- interface - in other words the System V/POSIX interface. BSD style
- locks using flock() never result in a mandatory lock.
-
-2. If a process has locked a region of a file with a mandatory read lock, then
- other processes are permitted to read from that region. If any of these
- processes attempts to write to the region it will block until the lock is
- released, unless the process has opened the file with the O_NONBLOCK
- flag in which case the system call will return immediately with the error
- status EAGAIN.
-
-3. If a process has locked a region of a file with a mandatory write lock, all
- attempts to read or write to that region block until the lock is released,
- unless a process has opened the file with the O_NONBLOCK flag in which case
- the system call will return immediately with the error status EAGAIN.
-
-4. Calls to open() with O_TRUNC, or to creat(), on a existing file that has
- any mandatory locks owned by other processes will be rejected with the
- error status EAGAIN.
-
-5. Attempts to apply a mandatory lock to a file that is memory mapped and
- shared (via mmap() with MAP_SHARED) will be rejected with the error status
- EAGAIN.
-
-6. Attempts to create a shared memory map of a file (via mmap() with MAP_SHARED)
- that has any mandatory locks in effect will be rejected with the error status
- EAGAIN.
-
-5. Which system calls are affected?

-
-Those which modify a file's contents, not just the inode. That gives read(),
-write(), readv(), writev(), open(), creat(), mmap(), truncate() and
-ftruncate(). truncate() and ftruncate() are considered to be "write" actions
-for the purposes of mandatory locking.
-
-The affected region is usually defined as stretching from the current position
-for the total number of bytes read or written. For the truncate calls it is
-defined as the bytes of a file removed or added (we must also consider bytes
-added, as a lock can specify just "the whole file", rather than a specific
-range of bytes.)

-
-Note 3: I may have overlooked some system calls that need mandatory lock
-checking in my eagerness to get this code out the door. Please let me know, or
-better still fix the system calls yourself and submit a patch to me or Linus.
-
-6. Warning!

-
-Not even root can override a mandatory lock, so runaway processes can wreak
-havoc if they lock crucial files. The way around it is to change the file
-permissions (remove the setgid bit) before trying to read or write to it.
-Of course, that might be a bit tricky if the system is hung :-(
-
--
1.5.3.1.139.g9346b

Subject: [PATCH 2/2] locks: add warning about mandatory locking races
Posted by bfields on Tue, 25 Sep 2007 16:56:29 GMT
View Forum Message <> Reply to Message

The mandatory file locking implementation has long-standing races that
probably render it useless. I know of no plans to fix them. Till we
do, we should at least warn people.

Signed-off-by: J. Bruce Fields <bfields@citi.umich.edu>

 Documentation/filesystems/mandatory-locking.txt | 21 ++++++++++++++++++++-
 1 files changed, 20 insertions(+), 1 deletions(-)

diff --git a/Documentation/filesystems/mandatory-locking.txt
b/Documentation/filesystems/mandatory-locking.txt
index bc449d4..8ac5cfb 100644
--- a/Documentation/filesystems/mandatory-locking.txt
+++ b/Documentation/filesystems/mandatory-locking.txt
@@ -3,7 +3,26 @@
   Andy Walker <andy@lysaker.kvaerner.no>

       15 April 1996
-
+   (Updated September 2007)
+
+0. Why should I avoid mandatory locking?
+--
+
+The Linux implementation is prey to a number of difficult-to-fix race
+conditions which in practice make it not dependable:
+

+ - The write system call checks for a mandatory lock only once
+ at its start. It is therefore possible for a lock request to
+ be granted after this check but before the data is modified.
+ A process may then see file data change even while a mandatory
+ lock was held.
+ - Similarly, an exclusive lock may be granted on a file after
+ the kernel has decided to proceed with a read, but before the
+ read has actually completed, and the reading process may see
+ the file data in a state which should not have been visible
+ to it.
+ - Similar races make the claimed mutual exclusion between lock
+ and mmap similarly unreliable.

1. What is mandatory locking?

--
1.5.3.1.139.g9346b

Subject: Re: [PATCH 1/2] Documentation: move mandatory locking documentation to filesystems/
Posted by Randy Dunlap on Tue, 25 Sep 2007 17:12:40 GMT
View Forum Message <> Reply to Message

On Tue, 25 Sep 2007 12:55:51 -0400 J. Bruce Fields wrote:

> Shouldn't this mandatory-locking documentation be in the
> Documentation/filesystems directory?

Agreed.

> Give it a more descriptive name while we're at it, and update 00-INDEX
> with a more inclusive description of Documentation/filesystems (which
> has already talked about more than just individual filesystems).

OK.
One correction below.

Then
Acked-by: Randy Dunlap <randy.dunlap@oracle.com>

> Signed-off-by: J. Bruce Fields <bfields@citi.umich.edu>
>
> ---
>
> On Tue, Sep 18, 2007 at 01:40:16PM -0400, bfields wrote:
> > Hm. Documentation/mandatory.txt claims that it mandatory locks and

> > mmap() with MAP_SHARED exclude each other, but I can't see where that's
> > enfoced. That file doesn't make any mention of the above race.
> >
> > So for now I think someone should update that file and fcntl(2) to
> > mention these problems and to recommend rather strongly against using
> > mandatory locking.
>
> Anyone have an objection if I submit these two patches to Linus for
> 2.6.24?
>
> --b.
>
> Documentation/00-INDEX | 4 +-
> Documentation/filesystems/00-INDEX | 2 +
> Documentation/filesystems/mandatory-locking.txt | 152 ++++++++++++++++++++++++
> Documentation/locks.txt | 10 +-
> Documentation/mandatory.txt | 152 ----------------------
> 5 files changed, 160 insertions(+), 160 deletions(-)
> create mode 100644 Documentation/filesystems/mandatory-locking.txt
> delete mode 100644 Documentation/mandatory.txt
>
> diff --git a/Documentation/00-INDEX b/Documentation/00-INDEX
> index 43e89b1..910473c 100644
> --- a/Documentation/00-INDEX
> +++ b/Documentation/00-INDEX
> @@ -145,7 +145,7 @@ fb/
> feature-removal-schedule.txt
>   - list of files and features that are going to be removed.
> filesystems/
> - - directory with info on the various filesystems that Linux supports.
> + - info on the vfs and the various filesystems that Linux supports.
> firmware_class/
>   - request_firmware() hotplug interface info.
> floppy.txt
> @@ -240,8 +240,6 @@ m68k/
>   - directory with info about Linux on Motorola 68k architecture.
> magic-number.txt
>   - list of magic numbers used to mark/protect kernel data structures.
> -mandatory.txt
> - - info on the Linux implementation of Sys V mandatory file locking.
> mca.txt
>   - info on supporting Micro Channel Architecture (e.g. PS/2) systems.
> md.txt
> diff --git a/Documentation/filesystems/00-INDEX b/Documentation/filesystems/00-INDEX
> index 59db1bc..73c8e94 100644
> --- a/Documentation/filesystems/00-INDEX
> +++ b/Documentation/filesystems/00-INDEX
> @@ -52,6 +52,8 @@ isofs.txt

> - info and mount options for the ISO 9660 (CDROM) filesystem.
> jfs.txt
> - info and mount options for the JFS filesystem.
> +mandatory

  mandatory-locking.txt

> + - info on the Linux implementation of Sys V mandatory file locking.
> ncpfs.txt
> - info on Novell Netware(tm) filesystem using NCP protocol.
> ntfs.txt

~Randy
Phaedrus says that Quality is about caring.

Subject: Re: [PATCH 1/2] Documentation: move mandatory locking documentation to filesystems/
Posted by bfields on Tue, 25 Sep 2007 17:24:52 GMT
View Forum Message <> Reply to Message

On Tue, Sep 25, 2007 at 10:12:40AM -0700, Randy Dunlap wrote:
> One correction below.
...
> > --- a/Documentation/filesystems/00-INDEX
> > +++ b/Documentation/filesystems/00-INDEX
> > @@ -52,6 +52,8 @@ isofs.txt
> > - info and mount options for the ISO 9660 (CDROM) filesystem.
> > jfs.txt
> > - info and mount options for the JFS filesystem.
> > +mandatory
>
> mandatory-locking.txt

Thanks, fixed.

--b.
