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