The Linux NTFS filesystem driver

Table of contents

Overview

Linux-NTFS comes with a number of user-space programs known as ntfsprogs. These include mkntfs, a full-featured ntfs filesystem format utility, ntfsundelete used for recovering files that were unintentionally deleted from an NTFS volume and ntfsresize which is used to resize an NTFS partition. See the web site for more information.

To mount an NTFS 1.2/3.x (Windows NT4/2000/XP/2003) volume, use the file system type 'ntfs'. The driver currently supports read-only mode (with no fault-tolerance, encryption or journalling) and very limited, but safe, write support.

For fault tolerance and raid support (i.e. volume and stripe sets), you can use the kernel's Software RAID / MD driver. See section "Using Software RAID with NTFS" for details.

Web site

There is plenty of additional information on the linux-ntfs web site at http://www.linux-ntfs.org/

The web site has a lot of additional information, such as a comprehensive FAQ, documentation on the NTFS on-disk format, information on the Linux-NTFS userspace utilities, etc.

Features

- This is a complete rewrite of the NTFS driver that used to be in the 2.4 and
  earlier kernels. This new driver implements NTFS read support and is
  functionally equivalent to the old ntfs driver and it also implements limited
  write support. The biggest limitation at present is that files/directories

```
  cannot be created or deleted.  See below for the list of write features that
  are so far supported.  Another limitation is that writing to compressed files
  is not implemented at all.  Also, neither read nor write access to encrypted
  files is so far implemented.
- The new driver has full support for sparse files on NTFS 3.x volumes which
  the old driver isn't happy with.
- The new driver supports execution of binaries due to mmap() now being
  supported.
- The new driver supports loopback mounting of files on NTFS which is used by
  some Linux distributions to enable the user to run Linux from an NTFS
  partition by creating a large file while in Windows and then loopback
  mounting the file while in Linux and creating a Linux filesystem on it that
  is used to install Linux on it.
- A comparison of the two drivers using:
        time find . -type f -exec md5sum "{}" \;
  run three times in sequence with each driver (after a reboot) on a 1.4GiB
  NTFS partition, showed the new driver to be 20% faster in total time elapsed
  (from 9:43 minutes on average down to 7:53).  The time spent in user space
  was unchanged but the time spent in the kernel was decreased by a factor of
  2.5 (from 85 CPU seconds down to 33).
- The driver does not support short file names in general.  For backwards
  compatibility, we implement access to files using their short file names if
  they exist.  The driver will not create short file names however, and a
  rename will discard any existing short file name.
- The new driver supports exporting of mounted NTFS volumes via NFS.
- The new driver supports async io (aio).
- The new driver supports fsync(2), fdatasync(2), and msync(2).
- The new driver supports readv(2) and writev(2).
- The new driver supports access time updates (including mtime and ctime).
- The new driver supports truncate(2) and open(2) with O_TRUNC.  But at present
  only very limited support for highly fragmented files, i.e. ones which have
  their data attribute split across multiple extents, is included.  Another
  limitation is that at present truncate(2) will never create sparse files,
  since to mark a file sparse we need to modify the directory entry for the
  file and we do not implement directory modifications yet.
- The new driver supports write(2) which can both overwrite existing data and
  extend the file size so that you can write beyond the existing data.  Also,
  writing into sparse regions is supported and the holes are filled in with
  clusters.  But at present only limited support for highly fragmented files,
  i.e. ones which have their data attribute split across multiple extents, is
  included.  Another limitation is that write(2) will never create sparse
  files, since to mark a file sparse we need to modify the directory entry for
  the file and we do not implement directory modifications yet.

Supported mount options
=======================

In addition to the generic mount options described by the manual page for the
mount command (man 8 mount, also see man 5 fstab), the NTFS driver supports the
following mount options:

iocharset=name          Deprecated option.  Still supported but please use
                        nls=name in the future.  See description for nls=name.

nls=name                Character set to use when returning file names.
                        Unlike VFAT, NTFS suppresses names that contain
                        unconvertible characters.  Note that most character
                        sets contain insufficient characters to represent all
                        possible Unicode characters that can exist on NTFS.
                        To be sure you are not missing any files, you are
                        advised to use nls=utf8 which is capable of
                        representing all Unicode characters.

utf8=<bool>             Option no longer supported.  Currently mapped to
                        nls=utf8 but please use nls=utf8 in the future and
                        make sure utf8 is compiled either as module or into
                        the kernel.  See description for nls=name.

uid=
gid=
umask=                  Provide default owner, group, and access mode mask.
                        These options work as documented in mount(8).  By
                        default, the files/directories are owned by root and
                        he/she has read and write permissions, as well as
                        browse permission for directories.  No one else has any
                        access permissions.  I.e. the mode on all files is by
                        default rw------- and for directories rwx------, a
                        consequence of the default fmask=0177 and dmask=0077.
                        Using a umask of zero will grant all permissions to
                        everyone, i.e. all files and directories will have mode
                        rwxrwxrwx.

fmask=
dmask=                  Instead of specifying umask which applies both to
                        files and directories, fmask applies only to files and
                        dmask only to directories.

sloppy=<BOOL>           If sloppy is specified, ignore unknown mount options.
                        Otherwise the default behaviour is to abort mount if
                        any unknown options are found.

show_sys_files=<BOOL>   If show_sys_files is specified, show the system files
                        in directory listings.  Otherwise the default behaviour
                        is to hide the system files.
                        Note that even when show_sys_files is specified, "$MFT"
                        will not be visible due to bugs/mis-features in glibc.
                        Further, note that irrespective of show_sys_files, all
                        files are accessible by name, i.e. you can always do
                        "ls -l \$UpCase" for example to specifically show the
                        system file containing the Unicode upcase table.

case_sensitive=<BOOL>   If case_sensitive is specified, treat all file names as
                        case sensitive and create file names in the POSIX
                        namespace.  Otherwise the default behaviour is to treat
                        file names as case insensitive and to create file names
                        in the WIN32/LONG name space.  Note, the Linux NTFS
                        driver will never create short file names and will
                        remove them on rename/delete of the corresponding long
                        file name.
                        Note that files remain accessible via their short file
                        name, if it exists.  If case_sensitive, you will need
                        to provide the correct case of the short file name.
```

```
disable_sparse=<BOOL>   If disable_sparse is specified, creation of sparse
                        regions, i.e. holes, inside files is disabled for the
                        volume (for the duration of this mount only).  By
                        default, creation of sparse regions is enabled, which
                        is consistent with the behaviour of traditional Unix
                        filesystems.

errors=opt              What to do when critical filesystem errors are found.
                        Following values can be used for "opt":
                          continue: DEFAULT, try to clean-up as much as
                                    possible, e.g. marking a corrupt inode as
                                    bad so it is no longer accessed, and then
                                    continue.
                          recover:  At present only supported is recovery of
                                    the boot sector from the backup copy.
                                    If read-only mount, the recovery is done
                                    in memory only and not written to disk.
                        Note that the options are additive, i.e. specifying:
                           errors=continue,errors=recover
                        means the driver will attempt to recover and if that
                        fails it will clean-up as much as possible and
                        continue.

mft_zone_multiplier=    Set the MFT zone multiplier for the volume (this
                        setting is not persistent across mounts and can be
                        changed from mount to mount but cannot be changed on
                        remount).  Values of 1 to 4 are allowed, 1 being the
                        default.  The MFT zone multiplier determines how much
                        space is reserved for the MFT on the volume.  If all
                        other space is used up, then the MFT zone will be
                        shrunk dynamically, so this has no impact on the
                        amount of free space.  However, it can have an impact
                        on performance by affecting fragmentation of the MFT.
                        In general use the default.  If you have a lot of small
                        files then use a higher value.  The values have the
                        following meaning:
                              Value           MFT zone size (% of volume size)
                                1                12.5%
                                2                25%
                                3                37.5%
                                4                50%
                        Note this option is irrelevant for read-only mounts.
```

Known bugs and (mis-)features
=============================

- The link count on each directory inode entry is set to 1, due to Linux not
  supporting directory hard links. This may well confuse some user space
  applications, since the directory names will have the same inode numbers.
  This also speeds up ntfs_read_inode() immensely. And we haven't found any
  problems with this approach so far. If you find a problem with this, please
  let us know.

Please send bug reports/comments/feedback/abuse to the Linux-NTFS development
list at sourceforge: linux-ntfs-dev@lists.sourceforge.net

Using NTFS volume and stripe sets

For support of volume and stripe sets, you can either use the kernel's
Device-Mapper driver or the kernel's Software RAID / MD driver. The former is
the recommended one to use for linear raid. But the latter is required for
raid level 5. For striping and mirroring, either driver should work fine.

The Device-Mapper driver

You will need to create a table of the components of the volume/stripe set and
how they fit together and load this into the kernel using the dmsetup utility
(see man 8 dmsetup).

Linear volume sets, i.e. linear raid, has been tested and works fine. Even
though untested, there is no reason why stripe sets, i.e. raid level 0, and
mirrors, i.e. raid level 1 should not work, too. Stripes with parity, i.e.
raid level 5, unfortunately cannot work yet because the current version of the
Device-Mapper driver does not support raid level 5. You may be able to use the
Software RAID / MD driver for raid level 5, see the next section for details.

To create the table describing your volume you will need to know each of its
components and their sizes in sectors, i.e. multiples of 512-byte blocks.

For NT4 fault tolerant volumes you can obtain the sizes using fdisk. So for
example if one of your partitions is /dev/hda2 you would do:

```
$ fdisk -ul /dev/hda

Disk /dev/hda: 81.9 GB, 81964302336 bytes
255 heads, 63 sectors/track, 9964 cylinders, total 160086528 sectors
Units = sectors of 1 * 512 = 512 bytes

   Device Boot      Start         End      Blocks   Id  System
   /dev/hda1   *          63     4209029     2104483+  83  Linux
   /dev/hda2         4209030    37768814    16779892+  86  NTFS
   /dev/hda3        37768815    46170809     4200997+  83  Linux
```

And you would know that /dev/hda2 has a size of 37768814 - 4209030 + 1 =
33559785 sectors.

For Win2k and later dynamic disks, you can for example use the ldminfo utility
which is part of the Linux LDM tools (the latest version at the time of
writing is linux-ldm-0.0.8.tar.bz2). You can download it from:
        http://www.linux-ntfs.org/
Simply extract the downloaded archive (tar xvjf linux-ldm-0.0.8.tar.bz2), go
into it (cd linux-ldm-0.0.8) and change to the test directory (cd test). You
will find the precompiled (i386) ldminfo utility there. NOTE: You will not be
able to compile this yourself easily so use the binary version!

Then you would use ldminfo in dump mode to obtain the necessary information:

```
$ ./ldminfo --dump /dev/hda
```

This would dump the LDM database found on /dev/hda which describes all of your dynamic disks and all the volumes on them. At the bottom you will see the VOLUME DEFINITIONS section which is all you really need. You may need to look further above to determine which of the disks in the volume definitions is which device in Linux. Hint: Run ldminfo on each of your dynamic disks and look at the Disk Id close to the top of the output for each (the PRIVATE HEADER section). You can then find these Disk Ids in the VBLK DATABASE section in the <Disk> components where you will get the LDM Name for the disk that is found in the VOLUME DEFINITIONS section.

Note you will also need to enable the LDM driver in the Linux kernel. If your distribution did not enable it, you will need to recompile the kernel with it enabled. This will create the LDM partitions on each device at boot time. You would then use those devices (for /dev/hda they would be /dev/hda1, 2, 3, etc) in the Device-Mapper table.

You can also bypass using the LDM driver by using the main device (e.g. /dev/hda) and then using the offsets of the LDM partitions into this device as the "Start sector of device" when creating the table. Once again ldminfo would give you the correct information to do this.

Assuming you know all your devices and their sizes things are easy.

For a linear raid the table would look like this (note all values are in 512-byte sectors):

```
--- cut here ---
# Offset into   Size of this    Raid type       Device          Start sector
# volume        device                                          of device
0               1028161         linear          /dev/hda1       0
1028161         3903762         linear          /dev/hdb2       0
4931923         2103211         linear          /dev/hdc1       0
--- cut here ---
```

For a striped volume, i.e. raid level 0, you will need to know the chunk size you used when creating the volume. Windows uses 64kiB as the default, so it will probably be this unless you changes the defaults when creating the array.

For a raid level 0 the table would look like this (note all values are in 512-byte sectors):

```
--- cut here ---
# Offset  Size      Raid     Number   Chunk  1st        Start  2nd        Start
# into    of the    type     of       size   Device     in     Device     in
# volume  volume             stripes                    device            device
0         2056320   striped  2        128    /dev/hda1  0      /dev/hdb1  0
--- cut here ---
```

If there are more than two devices, just add each of them to the end of the line.

Finally, for a mirrored volume, i.e. raid level 1, the table would look like this (note all values are in 512-byte sectors):

```
--- cut here ---
# Ofs Size   Raid   Log  Number Region Should Number Source  Start Target Start
# in  of the type   type of log size   sync?  of     Device  in    Device in
# vol volume             params             mirrors        Device       Device
0    2056320 mirror core 2      16     nosync 2    /dev/hda1 0   /dev/hdb1 0
--- cut here ---
```

If you are mirroring to multiple devices you can specify further targets at the end of the line.

Note the "Should sync?" parameter "nosync" means that the two mirrors are already in sync which will be the case on a clean shutdown of Windows. If the mirrors are not clean, you can specify the "sync" option instead of "nosync" and the Device-Mapper driver will then copy the entirety of the "Source Device" to the "Target Device" or if you specified multipled target devices to all of them.

Once you have your table, save it in a file somewhere (e.g. /etc/ntfsvolume1), and hand it over to dmsetup to work with, like so:

```
$ dmsetup create myvolume1 /etc/ntfsvolume1
```

You can obviously replace "myvolume1" with whatever name you like.

If it all worked, you will now have the device /dev/device-mapper/myvolume1 which you can then just use as an argument to the mount command as usual to mount the ntfs volume. For example:

```
$ mount -t ntfs -o ro /dev/device-mapper/myvolume1 /mnt/myvol1
```

(You need to create the directory /mnt/myvol1 first and of course you can use anything you like instead of /mnt/myvol1 as long as it is an existing directory.)

It is advisable to do the mount read-only to see if the volume has been setup correctly to avoid the possibility of causing damage to the data on the ntfs volume.

The Software RAID / MD driver

An alternative to using the Device-Mapper driver is to use the kernel's Software RAID / MD driver. For which you need to set up your /etc/raidtab appropriately (see man 5 raidtab).

Linear volume sets, i.e. linear raid, as well as stripe sets, i.e. raid level 0, have been tested and work fine (though see section "Limitations when using the MD driver with NTFS volumes" especially if you want to use linear raid). Even though untested, there is no reason why mirrors, i.e. raid level 1, and stripes with parity, i.e. raid level 5, should not work, too.

You have to use the "persistent-superblock 0" option for each raid-disk in the NTFS volume/stripe you are configuring in /etc/raidtab as the persistent superblock used by the MD driver would damage the NTFS volume.

Windows by default uses a stripe chunk size of 64k, so you probably want the "chunk-size 64k" option for each raid-disk, too.

For example, if you have a stripe set consisting of two partitions /dev/hda5 and /dev/hdb1 your /etc/raidtab would look like this:

```
raiddev /dev/md0
        raid-level      0
        nr-raid-disks   2
        nr-spare-disks  0
        persistent-superblock   0
        chunk-size      64k
        device          /dev/hda5
        raid-disk       0
        device          /dev/hdb1
        raid-disk       1
```

For linear raid, just change the raid-level above to "raid-level linear", for mirrors, change it to "raid-level 1", and for stripe sets with parity, change it to "raid-level 5".

Note for stripe sets with parity you will also need to tell the MD driver which parity algorithm to use by specifying the option "parity-algorithm which", where you need to replace "which" with the name of the algorithm to use (see man 5 raidtab for available algorithms) and you will have to try the different available algorithms until you find one that works. Make sure you are working read-only when playing with this as you may damage your data otherwise. If you find which algorithm works please let us know (email the linux-ntfs developers list linux-ntfs-dev@lists.sourceforge.net or drop in on IRC in channel #ntfs on the irc.freenode.net network) so we can update this documentation.

Once the raidtab is setup, run for example raid0run -a to start all devices or raid0run /dev/md0 to start a particular md device, in this case /dev/md0.

Then just use the mount command as usual to mount the ntfs volume using for example: `mount -t ntfs -o ro /dev/md0 /mnt/myntfsvolume`

It is advisable to do the mount read-only to see if the md volume has been setup correctly to avoid the possibility of causing damage to the data on the ntfs volume.

Limitations when using the Software RAID / MD driver

Using the md driver will not work properly if any of your NTFS partitions have an odd number of sectors. This is especially important for linear raid as all data after the first partition with an odd number of sectors will be offset by one or more sectors so if you mount such a partition with write support you will cause massive damage to the data on the volume which will only become apparent when you try to use the volume again under Windows.

So when using linear raid, make sure that all your partitions have an even number of sectors BEFORE attempting to use it. You have been warned!

Even better is to simply use the Device-Mapper for linear raid and then you do
not have this problem with odd numbers of sectors.

ChangeLog

Note, a technical ChangeLog aimed at kernel hackers is in fs/ntfs/ChangeLog.

```
2.1.29:
	- Fix a deadlock when mounting read-write.
2.1.28:
	- Fix a deadlock.
2.1.27:
	- Implement page migration support so the kernel can move memory used
	  by NTFS files and directories around for management purposes.
	- Add support for writing to sparse files created with Windows XP SP2.
	- Many minor improvements and bug fixes.
2.1.26:
	- Implement support for sector sizes above 512 bytes (up to the maximum
	  supported by NTFS which is 4096 bytes).
	- Enhance support for NTFS volumes which were supported by Windows but
	  not by Linux due to invalid attribute list attribute flags.
	- A few minor updates and bug fixes.
2.1.25:
	- Write support is now extended with write(2) being able to both
	  overwrite existing file data and to extend files.  Also, if a write
	  to a sparse region occurs, write(2) will fill in the hole.  Note,
	  mmap(2) based writes still do not support writing into holes or
	  writing beyond the initialized size.
	- Write support has a new feature and that is that truncate(2) and
	  open(2) with O_TRUNC are now implemented thus files can be both made
	  smaller and larger.
	- Note: Both write(2) and truncate(2)/open(2) with O_TRUNC still have
	  limitations in that they
	  - only provide limited support for highly fragmented files.
	  - only work on regular, i.e. uncompressed and unencrypted files.
	  - never create sparse files although this will change once directory
	    operations are implemented.
	- Lots of bug fixes and enhancements across the board.
2.1.24:
	- Support journals ($LogFile) which have been modified by chkdsk.  This
	  means users can boot into Windows after we marked the volume dirty.
	  The Windows boot will run chkdsk and then reboot.  The user can then
	  immediately boot into Linux rather than having to do a full Windows
	  boot first before rebooting into Linux and we will recognize such a
	  journal and empty it as it is clean by definition.
	- Support journals ($LogFile) with only one restart page as well as
	  journals with two different restart pages.  We sanity check both and
	  either use the only sane one or the more recent one of the two in the
	  case that both are valid.
	- Lots of bug fixes and enhancements across the board.
2.1.23:
	- Stamp the user space journal, aka transaction log, aka $UsnJrnl, if
```

```
	  it is present and active thus telling Windows and applications using
	  the transaction log that changes can have happened on the volume
	  which are not recorded in $UsnJrnl.
	- Detect the case when Windows has been hibernated (suspended to disk)
	  and if this is the case do not allow (re)mounting read-write to
	  prevent data corruption when you boot back into the suspended
	  Windows session.
	- Implement extension of resident files using the normal file write
	  code paths, i.e. most very small files can be extended to be a little
	  bit bigger but not by much.
	- Add new mount option "disable_sparse".  (See list of mount options
	  above for details.)
	- Improve handling of ntfs volumes with errors and strange boot sectors
	  in particular.
	- Fix various bugs including a nasty deadlock that appeared in recent
	  kernels (around 2.6.11-2.6.12 timeframe).
2.1.22:
	- Improve handling of ntfs volumes with errors.
	- Fix various bugs and race conditions.
2.1.21:
	- Fix several race conditions and various other bugs.
	- Many internal cleanups, code reorganization, optimizations, and mft
	  and index record writing code rewritten to fit in with the changes.
	- Update Documentation/filesystems/ntfs.txt with instructions on how to
	  use the Device-Mapper driver with NTFS ftdisk/LDM raid.
2.1.20:
	- Fix two stupid bugs introduced in 2.1.18 release.
2.1.19:
	- Minor bugfix in handling of the default upcase table.
	- Many internal cleanups and improvements.  Many thanks to Linus
	  Torvalds and Al Viro for the help and advice with the sparse
	  annotations and cleanups.
2.1.18:
	- Fix scheduling latencies at mount time.  (Ingo Molnar)
	- Fix endianness bug in a little traversed portion of the attribute
	  lookup code.
2.1.17:
	- Fix bugs in mount time error code paths.
2.1.16:
	- Implement access time updates (including mtime and ctime).
	- Implement fsync(2), fdatasync(2), and msync(2) system calls.
	- Enable the readv(2) and writev(2) system calls.
	- Enable access via the asynchronous io (aio) API by adding support for
	  the aio_read(3) and aio_write(3) functions.
2.1.15:
	- Invalidate quotas when (re)mounting read-write.
	  NOTE:  This now only leave user space journalling on the side.  (See
	  note for version 2.1.13, below.)
2.1.14:
	- Fix an NFSd caused deadlock reported by several users.
2.1.13:
	- Implement writing of inodes (access time updates are not implemented
	  yet so mounting with -o noatime,nodiratime is enforced).
	- Enable writing out of resident files so you can now overwrite any
	  uncompressed, unencrypted, nonsparse file as long as you do not
	  change the file size.
	- Add housekeeping of ntfs system files so that ntfsfix no longer needs
	  to be run after writing to an NTFS volume.
	  NOTE:  This still leaves quota tracking and user space journalling on
	  the side but they should not cause data corruption.  In the worst
	  case the charged quotas will be out of date ($Quota) and some
	  userspace applications might get confused due to the out of date
	  userspace journal ($UsnJrnl).
2.1.12:
	- Fix the second fix to the decompression engine from the 2.1.9 release
	  and some further internals cleanups.
2.1.11:
	- Driver internal cleanups.
2.1.10:
	- Force read-only (re)mounting of volumes with unsupported volume
	  flags and various cleanups.
2.1.9:
	- Fix two bugs in handling of corner cases in the decompression engine.
2.1.8:
	- Read the $MFT mirror and compare it to the $MFT and if the two do not
	  match, force a read-only mount and do not allow read-write remounts.
	- Read and parse the $LogFile journal and if it indicates that the
	  volume was not shutdown cleanly, force a read-only mount and do not
	  allow read-write remounts.  If the $LogFile indicates a clean
	  shutdown and a read-write (re)mount is requested, empty $LogFile to
	  ensure that Windows cannot cause data corruption by replaying a stale
	  journal after Linux has written to the volume.
	- Improve time handling so that the NTFS time is fully preserved when
	  converted to kernel time and only up to 99 nano-seconds are lost when
	  kernel time is converted to NTFS time.
2.1.7:
	- Enable NFS exporting of mounted NTFS volumes.
2.1.6:
	- Fix minor bug in handling of compressed directories that fixes the
	  erroneous "du" and "stat" output people reported.
2.1.5:
	- Minor bug fix in attribute list attribute handling that fixes the
	  I/O errors on "ls" of certain fragmented files found by at least two
	  people running Windows XP.
2.1.4:
	- Minor update allowing compilation with all gcc versions (well, the
	  ones the kernel can be compiled with anyway).
2.1.3:
	- Major bug fixes for reading files and volumes in corner cases which
	  were being hit by Windows 2k/XP users.
2.1.2:
	- Major bug fixes alleviating the hangs in statfs experienced by some
	  users.
2.1.1:
	- Update handling of compressed files so people no longer get the
	  frequently reported warning messages about initialized_size !=
	  data_size.
2.1.0:
	- Add configuration option for developmental write support.
	- Initial implementation of file overwriting. (Writes to resident files
	  are not written out to disk yet, so avoid writing to files smaller
	  than about 1kiB.)
```

```
        - Intercept/abort changes in file size as they are not implemented yet.
2.0.25:
        - Minor bugfixes in error code paths and small cleanups.
2.0.24:
        - Small internal cleanups.
        - Support for sendfile system call. (Christoph Hellwig)
2.0.23:
        - Massive internal locking changes to mft record locking. Fixes
          various race conditions and deadlocks.
        - Fix ntfs over loopback for compressed files by adding an
          optimization barrier. (gcc was screwing up otherwise ?)
        Thanks go to Christoph Hellwig for pointing these two out:
        - Remove now unused function fs/ntfs/malloc.h::vmalloc_nofs().
        - Fix ntfs_free() for ia64 and parisc.
2.0.22:
        - Small internal cleanups.
2.0.21:
        These only affect 32-bit architectures:
        - Check for, and refuse to mount too large volumes (maximum is 2TiB).
        - Check for, and refuse to open too large files and directories
          (maximum is 16TiB).
2.0.20:
        - Support non-resident directory index bitmaps. This means we now cope
          with huge directories without problems.
        - Fix a page leak that manifested itself in some cases when reading
          directory contents.
        - Internal cleanups.
2.0.19:
        - Fix race condition and improvements in block i/o interface.
        - Optimization when reading compressed files.
2.0.18:
        - Fix race condition in reading of compressed files.
2.0.17:
        - Cleanups and optimizations.
2.0.16:
        - Fix stupid bug introduced in 2.0.15 in new attribute inode API.
        - Big internal cleanup replacing the mftbmp access hacks by using the
          new attribute inode API instead.
2.0.15:
        - Bug fix in parsing of remount options.
        - Internal changes implementing attribute (fake) inodes allowing all
          attribute i/o to go via the page cache and to use all the normal
          vfs/mm functionality.
2.0.14:
        - Internal changes improving run list merging code and minor locking
          change to not rely on BKL in ntfs_statfs().
2.0.13:
        - Internal changes towards using iget5_locked() in preparation for
          fake inodes and small cleanups to ntfs_volume structure.
2.0.12:
        - Internal cleanups in address space operations made possible by the
          changes introduced in the previous release.
2.0.11:
        - Internal updates and cleanups introducing the first step towards
          fake inode based attribute i/o.
2.0.10:
	- Microsoft says that the maximum number of inodes is 2^32 - 1. Update
	  the driver accordingly to only use 32-bits to store inode numbers on
	  32-bit architectures. This improves the speed of the driver a little.
2.0.9:
	- Change decompression engine to use a single buffer. This should not
	  affect performance except perhaps on the most heavy i/o on SMP
	  systems when accessing multiple compressed files from multiple
	  devices simultaneously.
	- Minor updates and cleanups.
2.0.8:
	- Remove now obsolete show_inodes and posix mount option(s).
	- Restore show_sys_files mount option.
	- Add new mount option case_sensitive, to determine if the driver
	  treats file names as case sensitive or not.
	- Mostly drop support for short file names (for backwards compatibility
	  we only support accessing files via their short file name if one
	  exists).
	- Fix dcache aliasing issues wrt short/long file names.
	- Cleanups and minor fixes.
2.0.7:
	- Just cleanups.
2.0.6:
	- Major bugfix to make compatible with other kernel changes. This fixes
	  the hangs/oopses on umount.
	- Locking cleanup in directory operations (remove BKL usage).
2.0.5:
	- Major buffer overflow bug fix.
	- Minor cleanups and updates for kernel 2.5.12.
2.0.4:
	- Cleanups and updates for kernel 2.5.11.
2.0.3:
	- Small bug fixes, cleanups, and performance improvements.
2.0.2:
	- Use default fmask of 0177 so that files are no executable by default.
	  If you want owner executable files, just use fmask=0077.
	- Update for kernel 2.5.9 but preserve backwards compatibility with
	  kernel 2.5.7.
	- Minor bug fixes, cleanups, and updates.
2.0.1:
	- Minor updates, primarily set the executable bit by default on files
	  so they can be executed.
2.0.0:
	- Started ChangeLog.
```