The Old New Thing

The apocryphal history of file system tunnelling

15 Jul 2005 10:00 AM | 34

One of the file system features you may find yourself surprised by is tunneling, wherein the creation timestamp and short/long names of a file are taken from a file that existed in the directory previously. In other words, if you delete some file "File with long name.txt" and then create a new file with the same name, that new file will have the same short name and the same creation time as the original file. You can read this KB article for details on what operations are sensitive to tunnelling.

Why does tunneling exist at all?

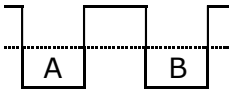
When you use a program to edit an existing file, then save it, you expect the original creation timestamp to be preserved, since you're editing a file, not creating a new one. But internally, many programs save a file by performing a combination of save, delete, and rename operations (such as the ones listed in the linked article), and without tunneling, the creation time of the file would seem to change even though from the end user's point of view, no file got created.

As another example of the importance of tunneling, consider that file "File with long name.txt", whose short name is say "FILEWI~1.TXT". You load this file into a program that is not long-filename-aware and save it. It deletes the old "FILEWI~1.TXT" and creates a new one with the same name. Without tunnelling, the associated long name of the file would be lost. Instead of a friendly long name, the file name got corrupted into this thing with squiggly marks. Not good.

But where did the name "tunneling" come from?

From quantum mechanics.

Consider the following analogy: You have two holes in the ground, and a particle is in the first hole (A) and doesn't have enough energy to get out. It only has enough energy to get as high as the dotted line.



You get distracted for a little while, maybe watch the Super Bowl halftime show, and when you come back, the particle somehow is now in hole B. This is impossible in classical mechanics, but thanks to the wacky world of quantum mechanics, it is not only possible, but **actually happens**. The phenomenon is known as tunneling, because it's as if the particle "dug a tunnel" between the two holes, thereby allowing it to get from one hole to another without ever going above the dotted line.

In the case of file system tunneling, it is information that appears to violate the laws of classical mechanics. The information was destroyed (by deleting or renaming the file), yet somehow managed to reconstruct itself on the other side of a temporal barrier.

The developer who was responsible for implementing tunneling on Windows 95 got kind of carried away with the quantum mechanics analogy: The fragments of information about recently-deleted or recently-renamed files are kept in data structures called "quarks".

Blog - Comment List MSDN TechNet

Comments

Nate
15 Jul 2005 10:46 AM
#

I can see why you would want the long file name to be tunneled, because of this scenario of when a file is edited with a non-LFN aware application, but why is the creation date preserved? If an application does a savetemp/delete/rename operation to save a file, the expected behavior from such an app would be to be to have a shiny new creation date.

Janne
15 Jul 2005 10:59 AM
#

I'm guessing this "safe save" was used by a large number of old DOS (and old Windows) programs to avoid the risk of losing all versions of a file if the computer crashed (or suffered a power failure) during the write operation, since FAT wasn't a journaled file system like NTFS. And, if my memory isn't lacking, FAT didn't HAVE a "creation date" until Windows 95, so that's when the problem first came into existance.

Tim Smith
15 Jul 2005 11:13 AM
#

It goes well beyond "if the computer crashes". The last thing you want to do is delete your original and then hope your software will be able to write the new file. If it fails, you have lost all the users work.

Assume your program will fail during the serialization of the new file. What will the user be able to do when the program fails? Sure, it is rare, but it is also very easy to implement a backup system that helps protect against failures.

A regular viewer
15 Jul 2005 11:13 AM
#

Totally OT, did you use a tool to generate that neat image using <table> tags, if so which one? If you hand coded that image/table, a darned good job.

Arlie Davis
15 Jul 2005 11:24 AM
#

Perhaps the *app* expects a new, shiny date, but the *user* certainly does not.

This is the right behavior in 99.9% of all situations. The tunneling behavior only kicks in during short time windows, when certain sequences of operations occur.

James Curran
15 Jul 2005 11:35 AM
#
um.. if this is the "apocryphal" history.... What's the real one?

PatriotB
15 Jul 2005 11:44 AM
#
Unfortunately, tunneling didn't help Visual C++ 6. Everytime you save a file, the create date is wiped out and replaced with the current date/time.

Looks like its fixed in VS.NET and newer versions.

vince
15 Jul 2005 11:51 AM
#
Why not just have the application get the original creation time, and then when it creates the new file, update the creation time of the new file to be the same as the old one?

This is the way *NIX editors work, and it seems like the task is much more appropriate to userspace than kernel space.

Especially if your app makes the (reasonable) assumption that if it deletes an old file and creates a new one with the same name that is should have the new creation date.

I cringe to think of how many other silly hacks like this are cluttering up the windows kernel.

vince
15 Jul 2005 12:21 PM
#
How embarassing, I misunderstood what typical *NIX editors do (yes I had checked code before posting my previous post, but misunderstood).

*NIX filesystems actually don't store the creation time at all, so this is a non-problem for editors. What confused me is when an editor makes a backup copy when you start making changes to a file, it does preserve the access/modified times for you.

I still stand by my claim that having the OS doing what windows does is needless bloat though and unwisely causes hidden side-effects to simple code.

AC
15 Jul 2005 12:27 PM
#
Wow... Short file names. It's a good thing that I run "fsutil behavior set disable8dot3 1" immediately after installing a fresh copy of XP. Significantly improves the installation of programs with 10s of 1000s of files (like Visual Studio).

Alan De Smet
15 Jul 2005 12:27 PM
#

Re: why not have the application handle setting the creation time correctly?

Because while you are a conscientious developer who always does the Right Thing, there are many developers who are sloppy, who don't care, or just accidentally overlooked subtle rule 47.A.iv "On replacing files".

While applications should be responsible for many things, ultimately it's up the OS to try and provide some level of consistency. This is how "silly hacks" like protected memory, preemptive multi-tasking, and OS enforced file permissions get into kernels.

I will note that strictly speaking Unix-like systems don't have a "create" time; they have a "change" time which tracks the last change to the inode. And enough programs replace files by clobbering the original that the ctime is usually the same as m(odified)time.

I'm not sure I agree with this particular hack; but I'm sympathetic to the design. Ultimately the lack of reliable ctimes on Unix has never been a problem for me.

Ben Cooke
15 Jul 2005 12:59 PM
#

I tried disabling short filenames for a while. I was surprised to find that it was me rather than any of my software that was incompatible. I have this habit -- that I hadn't really noticed until it didn't work anymore -- of performing the sequence Ctrl+Esc, R, c:\progra~1 (or c:\docume~1). Without the short filenames stored, that doesn't work anymore.

I also use a similar trick to install programs whose installers are intolerant of paths containing spaces. (Usually, they don't bother to call GetLongFileName either)

Is there a way, I wonder, to have progra~1 and docume~1 work without bothering to store them anywhere else? I suppose I could just create a junction point at c:\progra~1 pointing at c:\program files, but then c:\progra~1 will show up in directory listings, and stupid programs like the lame virus scanner I use will end up checking it all twice. Bleh.

mikeb
15 Jul 2005 1:10 PM
#

Nate: >> but why is the creation date preserved <<

This is done because the file systems on 32-bit Microsoft platforms support 3 time stamps for files: Creation, Last Access, and Last Modified. So the creation time stamp should be preserved when the editor 'saves' the file, but the other time stamps will be updated.

vince: >> Why not just have the application get the original creation time, and then when it creates the new file, update the creation time of the new file to be the same as the old one? <<

I'd guess that the primary reason is so that legacy applications that were written for platforms (MS-DOS, Win16) that have no concept of a Creation timestamp - only a Last modified timestamp - would work in the expected manner on newer filesystems.

Michael Geary
15 Jul 2005 1:36 PM
#

"I'm guessing this 'safe save' was used by a large number of old DOS (and old Windows) programs to avoid the risk of losing all versions of a file if the computer crashed (or suffered a power failure) during the write operation, since FAT wasn't a journaled file system like NTFS."

Whoa there. I don't think NTFS journaling will protect your data if a program does an "unsafe" save by directly overwriting the original file data in place. The "safe" save technique is needed on NTFS as much as any other filesystem. (Or am I mistaken?)

Michael Geary
15 Jul 2005 1:38 PM
#

"I'm guessing this 'safe save' was used by a large number of old DOS (and old Windows) programs to avoid the risk of losing all versions of a file if the computer crashed (or suffered a power failure) during the write operation, since FAT wasn't a journaled file system like NTFS."

Whoa there. I don't think NTFS journaling will protect your data if a program does an "unsafe" save by directly overwriting the original file data in place. The "safe" save technique is needed on NTFS as much as any other filesystem. (Or am I mistaken?)

waleri
15 Jul 2005 1:41 PM
#

Yet another problem with save/delete/rename scenario is that if edited file is actualy a hard link, the link is broken. Whether this is a good or bad thing, depends entirely on your own point of view. Actually, I've never used an editor that can handle this situation correctly, which means that all those editors creates a new file and modify creation date

Mike
15 Jul 2005 2:06 PM
#

Somebody should probably mention the ReplaceFile api which does the delete/rename/fixup in a more controlled way and handles stuff like getting the security reset correctly.

Mike Dimmick
15 Jul 2005 2:37 PM
#

I was wondering why I didn't know about ReplaceFile, but I see it first appeared in Windows 2000.

NTFS journalling does not protect user data, it only protects filesystem data. All that NTFS guarantees is that a valid consistent filesystem will be available after recovery.

"Transactional NTFS" is scheduled in for Longhorn, but this still won't protect every user data write. It's opt-in, and from what I've heard, moderately expensive in terms of time. The only way it can work, IMO, is if it logs all writes (both the before and after images, so the write can be rolled back if required) to the log file, then makes the changes in cache.

Chris
15 Jul 2005 2:42 PM
#

I'm with vince, this is cringe-worthy. Apple didn't do hacks like this when moving Mac OS 9 from 32 ASCII to 255 Unicode characters; app upgrades are required for preservation of long file names. Admittedly, 32 is a lot of characters in the first place, so the problem is less acute, but you really don't want to have to live with this sort of hack forever. The downside is too user-visible.

Cyrille Chépélov
15 Jul 2005 3:24 PM
#

Michael Geary: last I, uh, experienced it, NTFS exhibits at best a metadata-journalling behaviour.

To afford doing what you describe, you need a file system with complete transaction semantics, and an implementation that supports those. To the extent that its debugging went enough ahead, reiser4 might be an example, but leveraging its atomic capabilities requires a significant userland cooperation (meaning that in a Win32 land, with exatons of stuff to support, you simply can't do it)

Now, give me an atomic TernaryRenameAndCopyAttributes(LPCTSTR FileName, LPCTSTR BackupName, LPCTSTR ReplacementName); :-)

informator
15 Jul 2005 4:23 PM
#

To Mike Dimmick:

Yes, and don't forget that on IDE drives (and most of SATA drives) disk controller's cache data are not written through even if you're using unbuffered IO/write through flag. The reliable way here is to issue FlushFileBuffers after log write to make sure that log write data are in the media, then write data and issue another FlushFileBuffers call. This is required because when FlushFileBuffers is called, the pending writes in the controller's cache may be written to the media in non-chronological order (this is for IDE/most of SATA drives).

Bryan
15 Jul 2005 4:57 PM
#

Except that some (but very few) IDE drives don't flush their caches even when the OS tells them to -- I believe this was a performance hack, though I don't know for sure.

I do know that it's what prompted the Win95 or 98 (can't remember which) "shutdown update". The update merely prolonged the OS shutdown, though, to try to get the problematic drive(s) to flush its/their cache before it/they lost power.

Tim Smith
15 Jul 2005 9:07 PM
#

ReplaceFile - Drool...

In my last job I had to support 95+/NT4+ so I often lost track of new API. Now that I don't have those restrictions, I really need to review the API library.

I have some code to fix so it start uses ReplaceFile.

ipoverscsi
15 Jul 2005 10:16 PM
#

I would seriously like to thank you for pointing out this tunneling phenomenon. I once wrote a program that looked for new files by checking the creation time that consistently failed because of this issue. It took me a while to figure out that tunneling was occurring, but I assumed it was merely a fluke of accessing the file over a share. My coworkers, of course, didn't believe me that this was occurring. Now I can finally say 'I told you so.' :)

Norman Diamond
16 Jul 2005 7:55 AM
#

Friday, July 15, 2005 10:59 AM by Janne
> I'm guessing this "safe save" was used by a
> large number of old DOS (and old Windows)
> programs to avoid the risk of losing all
> versions of a file if the computer crashed
> (or suffered a power failure) during the
> write operation,

I also think so, and this actually made a large number of old DOS programs smarter than a large number of old Unix programs. In 1976 I used the "ed" editor in Unix ("vi" hadn't been invented yet) to make a few changes to a file of around 1,000 lines, typed the "w" command, and didn't get any feedback. When Unix came back up, it had never heard of my file, not the old version, not the new version. A few years later I saw companies advertising database systems running on Unix and I couldn't believe that anyone would use such a system for anything important. A few more years later, I finally got used to the idea that Unix was becoming somewhat reliable. (A few more years later, Windows 95 taught me that Unix had not been the world's most unreliable OS.)

Friday, July 15, 2005 2:37 PM by Mike Dimmick
> NTFS journalling does not protect user data,
> it only protects filesystem data.

Agreed. After a BSOD, files that had been open for writing often have a few sectors of binary zeroes instead of data that the user thought they had written.

Carlos
16 Jul 2005 11:52 AM
#
Friday, July 15, 2005 2:37 PM by Mike Dimmick
"Transactional NTFS is scheduled in for Longhorn, but this still won't protect every user data write. It's opt-in, and from what I've heard, moderately expensive in terms of time. The only way it can work, IMO, is if it logs all writes (both the before and after images, so the write can be rolled back if required) to the log file, then makes the changes in cache."

You can do better than this, and I believe reiser4 does. You write any changes to free sectors on the disk. When the write is committed you update the metadata to point at the new data instead of the old data (or write the metadata changes to the transaction log). If the write is rolled-back you put the new sectors back on the free list.

John
17 Jul 2005 1:02 AM
#
> Transactional NTFS is scheduled for Longhorn

Netware had transactional filesystem support (TTS) back in the early 1990's. It's amazing that NTFS still hasn't caught up.

Dean Harding
17 Jul 2005 8:01 PM
#
> Netware had transactional filesystem support (TTS) back
> in the early 1990's. It's amazing that NTFS still hasn't
> caught up.

I just had a look at TTS, and it's really nothing like TNTFS. TTS only works for certain files and for certain changes, whereas TNTFS works for any file and any type of change. Also, it uses the new "kernel" transactions, so anything else that supports kernel transactions can be modified in the same transaction (for example, the registry also supports them). They also participate in COM+ and DTS transactions, so you can join them to database transactions and have the whole operation atomic.

> I'm with vince, this is cringe-worthy. Apple didn't do hacks
> like this when moving Mac OS 9 from 32 ASCII to 255 Unicode
> characters; app upgrades are required for preservation
> of long file names.

Yes, well, we all know Apple's stance on backwards-compatibility...

William
17 Jul 2005 10:25 PM
#
"Quarks"... so could we expect the color theory being implemented somewhere? :)

vince
18 Jul 2005 12:09 AM
#
Dean Harding Said:
> Yes, well, we all know Apple's stance on
> backwards-compatibility...

Was that meant as sarcasm? Apple has done an admirable job of keeping backwards compatibility going, including changes of architecture (m68k to ppc, the forthcoming ppc to x86) and vast changes in OS design (classic under OSX).

nikos
18 Jul 2005 9:51 AM
#
waleri:

>Yet another problem with save/delete/rename
>scenario is that if edited file is actualy
>a hard link, the link is broken

this is my pet peeve too
most editors (including Visual studio) cause it, and in the old days at least the MFC CFile class was to blame that does all this rename/delete in the background whenever you save.

surely some other "safety" alternative could be possible, e.g. saving a backup in TEMP or whatever

CN
21 Jul 2005 10:36 AM
#
Carlos:

I imagine that the actual implementation uses the free sector approach. After all, this should be quite comparable to what the current Volume Shadow feature does.

(Hey, you could do semi-transactions today by quickly manipulating volume shadows. The performance would be a joke, but it would be kind of fun.)

GregM
27 Jul 2005 1:05 PM

#

Mike, thank you for mentioning ReplaceFile. I would have used this API a few weeks ago had I known about it.

余啊雷

8 Aug 2005 9:07 AM

#

Today I experienced some strange unexpected log file roll-overs when logging messages using the...