The Old New Thing

Why doesn't the Low Disk Space warning balloon show up as soon as I run low on disk space

17 Jul 2012 7:00 AM | 66

A customer reported an issue with the title "The notification balloon for Low Disk Space does not appear even if the free disk is very low." They provided the following steps:

- Install Windows 7 64-bit on a SATA drive.
- Copy files to the system drive until disk space becomes low.
- Observe that the notification balloon for Low Disk Space does not immediately appear.
- The balloon appears approximately ten minutes later.

You read through the steps nodding, "uh huh, uh huh", and then you get to the last step and you say, "Wait a second, the subject of your report was that the balloon doesn't appear at all, and now you're saying that it appears after ten minutes. So it *does* appear after all. What is the problem?"

The customer explained that on earlier versions of Windows, the Low Disk Space warning balloon appeared within one minute, whereas in Windows 7 it can take up to ten minutes for the balloon to appear.

Yup, that's right.

In previous versions of Windows, Explorer checked for low disk space once a minute. The Windows performance folks requested that the shell reduce the frequency of checks to improve overall system performance, and the shell team agreed to reduce the frequency to once every ten minutes. (The performance team made other suggestions to reduce the impact of that code that runs every ten minutes.)

So yes, in Windows 7, it may take up to ten minutes for Explorer to report that you are low on disk space. But Explorer never promised that those reports would be timely. Or that they would even appear in the first place. The behavior is not contractual; it's just a courtesy notification.

Related: How full does a hard drive have to get before Explorer will start getting concerned? and How do I disable the low disk space notifications?

Blog - Comment List MSDN TechNet

Comments

mmx
17 Jul 2012 7:13 AM
#

>> The Windows performance folks requested that the shell reduce the frequency of checks to improve overall system performance

Is the operation so expensive to matter ?

As I understand it's only run on local hard disks and, by virtue of being a courtesy

service and not life-threatening-if-the-number-is-wrong, it can also used a cached value.

RP
17 Jul 2012 7:23 AM
#

It's an I/O operation, and I/O is always expensive.

alegr1
17 Jul 2012 7:29 AM
#

It's not like the free space burns by itself. NTFS.SYS can always keep track of it without having to access the disk every time. Looks like the perf team is looking in the wrong places.

For example, if a 500 GB disk is formatted with 4K allocation units, its allocation map is under 16 MB. To go through it once is a trivial operation, and NTFS does that anyway.

> [*The issue isn't the I/O cost. It's the cost of waking up the system once a minute, which takes it out of low-power mode. And consider what happens on a Terminal Server machine with 60 users. -Raymond*]

Dominik
17 Jul 2012 7:38 AM
#

Why not check for low disk space after every 42 MB written to the disk?

GWO
17 Jul 2012 7:41 AM
#

"Is the operation so expensive to matter?" Two words: Network drives

alegr1
17 Jul 2012 7:50 AM
#

>Two words: Network drives

It's not the users that must get a warning about network drives, it's the administrators.

Even then, checking space on a network drive is a trivial operation in regards to the

client: send an SMB request, get a reply. It's less churn than moving a mouse pointer across the taskbar.

Jonathan
17 Jul 2012 7:56 AM
#

I/O can be expensive on:

* iSCSI-mounted disks, which are actually networked to some external storage.

* Physical disks that spin down (like my WD "Green"). I certainly wouldn't want that disk to constantly spin up every 1 min - or 10 for that matter.

And yes, the volume manager could keep a cache of free space in every volume, and remember when it's not populated yet (drive that hasn't spun at all since boot), etc. - but is it worth it for this feature?

Maurits [MSFT]
17 Jul 2012 8:13 AM
#

> The balloon appears approximately ten minutes later.

Approximately five minutes later, surely.

Joshua
17 Jul 2012 8:15 AM
#

Dude, this value is cached _in the boot sector_.

Rich
17 Jul 2012 8:19 AM
#

How could somebody have so much time that they could be bothered to report something like that to MS?

Mike
17 Jul 2012 8:30 AM
#

@Rich: my guess someone in a corporate environment. Someone got burned because they were copying something or whatever and left for lunch and came back to a crashed copy/rip app etc. I worked at a company were users would start experiments that

generate 20TB of data without seeing if the disk array had room. They'd use 20k worth of chemicals fire off a batch job for a gene sequencer and leave for the weekend. They'd come back on Monday and their program crashed because it was out of memory. Users in general will do stupid things and any time you don't hold their hand the same way as they are used to someone will get burned by it and it will be your fault.

Gabe
17 Jul 2012 8:57 AM
#

The issue here is not that the disk has to spin up, otherwise the disk could never spin down in the first place. I think the perf hit is in working set. A code path that runs once per minute is likely to not ever get paged out.

alegr1
17 Jul 2012 9:44 AM
#

>I think the perf hit is in working set. A code path that runs once per minute is likely to not ever get paged out.

InvalidArgumentException. There is no need to have code "paged out". There is a need to keep most recently used code in memory at an expense for least recently used code. 1 minute is a lot of time, compared to page-in latency. Code that does free space check has trivial size (<<<1M). It can be easily paged in in <100ms. If the OS is idle, the code will be in memory and won't even need the OS drive spin-up.

>And yes, the volume manager could keep a cache of free space in every volume, and remember when it's not populated yet (drive that hasn't spun at all since boot), etc. - but is it worth it for this feature

I'm pretty sure that volume mounting DOES access the disk, requiring a spin-up and DOES get the amount of free space in the process.

Adam Rosenfield
17 Jul 2012 9:54 AM
#

Updating once per minute is fine, but updating once per second will cause secondary effects due to code not being paged out: blogs.msdn.com/.../55256.aspx

alegr1
17 Jul 2012 10:22 AM
#

A toxic case of free space query is quote management. If quotes are enabled, each user may see their own free space number. Although this may be done cached, as well.

alegr1
17 Jul 2012 10:24 AM
#

>Updating once per minute is fine, but updating once per second will cause secondary effects due to code not being paged out

That made difference. In Windows 95 age. Not now.

xpclient
17 Jul 2012 10:51 AM
#

Disk space filling up is a common problem in Windows 7 due to the disaster that is Component Based Servicing.

> [*It wouldn't be a problem if you could disable auto-sort in Explorer, though. - Raymond*]

xpclient
17 Jul 2012 11:42 AM
#

Nope that doesn't have to do anything with my complaint about CBS. CBS is the worst disk gobbling up and slow component in Windows.

lefty
17 Jul 2012 11:52 AM
#

My newspaper used to hit my doorstep at 4:00am. Now it doesn't get here until 5:30.

What's up with that? I need my stuff *now*.

Anonymous Coward
17 Jul 2012 1:35 PM
#

I remember turning this off because it solved a problem where the floppy drive would make noise every now and then. As a side effect it also solved a minor issue of my hard drive never falling asleep.

I can't remember where I picked up this tip (I thought it was this site, but I can't find it again) but even though I'm reasonably technical, I could never have solved it by myself. I guess I'd have been listening to bzzz-ter-dum-ter-dum-ter-dum-click for ever.

Maurits [MSFT]
17 Jul 2012 2:07 PM
#

Why does this feature poll at all? Can't the check be tied to disk writes?

> [*And you would do this... how? -Raymond*]

Anonymous Coward
17 Jul 2012 2:18 PM
#

@Maurits - You want a shell notification to be tied in with all kernel disk writes?!

Matt
17 Jul 2012 2:37 PM
#

@Raymond

Why not do it somewhat like this?

DWORD dw;

HANDLE hFile = CreateFile("\\?\Devices\HarddiskVolume1\Events\LowDisk");

ReadFile(hFile, &dw);

> [*Um, disk free space is a file system concept, not a volume concept. And now the kernel needs to know what is effectively a UI value (what level disk space is considered "low" for a particular volume). And of course you wouldn't be able to eject or format any drives because Explorer would always have a handle open! -Raymond*]

Maurits [MSFT]
17 Jul 2012 4:32 PM
#

Implementation details left as an exercise.

> [*"Hey, I can't reformat my hard drive because Explorer is monitoring it for low disk space." -Raymond*]

Nick
17 Jul 2012 4:51 PM
#

[And of course you wouldn't be able to eject or format any drives because Explorer would always have a handle open! -Raymond]

Be careful... that's some pretty serious troll bait right there :)

Maurits [MSFT]
17 Jul 2012 5:36 PM
#

The existence of poor implementations does not necessarily imply the impossibility of a proper implementation.

cheong00
17 Jul 2012 7:43 PM
#

["Hey, I can't reformat my hard drive because Explorer is monitoring it for low disk space." -Raymond]

I'm pretty sure "Disk Format" command would have been written to unmount the drive before starting formatting, just like convert.exe have option to force, if it's necessary.

Mark
17 Jul 2012 7:45 PM
#

Indeed, but the fact that no one has found a proper one yet...

Crescens2k
17 Jul 2012 10:17 PM
#

@Nick:

Why is that troll bait? On its own, Explorer doesn't keep handles open. If you end up not being able to do something to a drive because of an open handle, it is something else. For example, an anti-virus holding the file handle open for a bit longer due to the file system scan, playing something off of a removable drive, and the media player taking a bit longer to release the handle than it would appear. You just close an Explorer window and the thumbnail generator is working away. You forget you have a console window open that is located in a directory on that drive.

In fact, I would say the biggest cause of not being able to eject a drive right after you stop using a file on it is the anti-virus. How many times have you stopped using stuff on a removable drive, tried to eject it and it fails then tried to eject it again and it succeeds?

@cheong00:

Yes, force options are one thing, but isn't it annoying to force something?

@alger1:

Mounting a volume does things like read the bootsector, do the initial consistancy check (replaying the last few events if the dismount wasn't clean) and then brings the volume online. So yes, it does access the disk during volume mount.

cheong00
17 Jul 2012 10:59 PM
#

@Crescens2k: On the other hand, I could think of some cases that adding option (or just some warning prompt) for "disk format" to force unmount useful. (When trying to format a partition where "indexing service" is running, or have shared folder that's using by remote users, for example.)

Matt
18 Jul 2012 1:15 AM
#

@Raymond:

OK, how about (in ntos):

```
//

// create an event during boot

//

HANDLE hLowDiskConditionEvent;

void DuringBoot()

{

 hLowDiskConditionEvent = NtCreateEvent(..., L"\\?\KernelObjects\LowDiskCondition");

}

//

// there are lots of syscalls that return configuration values (e.g. whatever backs GetSystemMetrics for instance). Let's just hijack one to return the disk threshold

//

SIZE_T cached = ~0;

NtGetSystemMetrics(int iVal, SIZE_T* result;)

{

 ..
```

```
 if(iVal == SM_LOWDISKTHRESHOLD)
 {
   if(cached == ~0 || cacheExpiredForSomeReason)
     cached = ReadRegKey("HKLM\blah\LowDiskConditionThreshold");
   *result = cached;
 }
 ..
}
//
// In the filesystem driver
//
NTSTATUS HandleSomeFileWrite(...)
{
 DoTheWrite();
 if(cachedVolumeSpace < NtGetSystemMetrics(SM_LOWDISKTHRESHOLD))
 {
   NtOpenEvent(hEvent, L"\\?\KernelObjects\LowDiskCondition");
   NtSetEvent(hEvent);
 }
}
//
// And finally in explorer.exe (or whomever else might care about low-disk events).
//
void LowDiskEventThreadMain()
{
 hEvent = OpenEvent(GENERIC_READ, "\\?\KernelObjects\LowDiskCondition");
 WaitForSingleObject(hEvent, TIMEOUT_WHENEVER);
 // oh noes! we're low on disk space. Find out which disk is low by querying them all now
and then show the balloon.
}
```

Someone
18 Jul 2012 2:17 AM
#

"In fact, I would say the biggest cause of not being able to eject a drive right after you stop using a file on it is the anti-virus."

In my experience, this is not true because online-virus checks are performed as part of the file operation: When reading, during CreateFile or at the first ReadFile, when writing, during WriteFile() or at CloseHandle(), causing this calls to take more time.

The background operations of the Explorer and/or of the Indexing Service are to blame: When they try to create the icons for many large video files in an Explorer window, this can take forever. And once initiated, it will continue for some time even when you change the folder in Explorer. Also, if the virus scanner is configured to check also at Read operations, this file operations caused by Explorer and/or Indexing Service triggers the Virus scanner, slowing down the extracting further more.

AC
18 Jul 2012 2:19 AM
#

>And of course you wouldn't be able to eject or format any drives because Explorer would always have a handle open! -Raymond

On XP, that was the case about 99% of the time anyway.

Neil
18 Jul 2012 3:31 AM
#

Obligatory rant: I installed so much stuff on my Windows 8 Consumer Preview virtual machine that I consumed the VM's default volume allocation and I didn't get a single notification. (At the time, I was trying to run a script which didn't have any error-checking, so it just appeared to be misbehaving. Fortunately I then happened to invoke another, cleverer, process that alerted me.)

alegr1
18 Jul 2012 7:19 AM
#

[And of course you wouldn't be able to eject or format any drives because Explorer would always have a handle open! -Raymond]

Vista bug.

alegr1

18 Jul 2012 7:27 AM
#

>The background operations of the Explorer and/or of the Indexing Service are to blame: When they try to create the icons for many large video files in an Explorer window, this can take forever

In a few cases (bailing out because of some error?) Explorer would keep a handle to a directory or to a file, making it impossible to delete a directory.

But for what it's worth, Explorer keep A LOT of handles to the directories open, because it's constantly monitoring volumes for changes. Just run HANDLES.EXE and see. The handles are opened with SHARE_DELETE_FILE, though.

alegr1
18 Jul 2012 7:33 AM
#

[The issue isn't the I/O cost. It's the cost of waking up the system once a minute, which takes it out of low-power mode.]

This is why SetWaitableTimer has bResume argument.

[And consider what happens on a Terminal Server machine with 60 users. -Raymond]

A terminal server is supposed to be able to sustain activity of all users at all times, not just a single call to NTFS.SYS once per minute per user which doesn't even have to access the disk.

> [*bResume is for waking from suspend, which is not the same as waking from a low-power state. The point about Terminal Server is that you have all these threads waking up and paging in stacks constantly. -Raymond*]

Gabe
18 Jul 2012 7:35 AM
#

Matt: How does your code account for the fact that there are actually 4 different thresholds, and they differ depending on the size of the volume? How does it handle different per-user quotas?

Keep in mind that this solution doesn't prevent the need for the polling code because some filesystems might not implement the LowDiskCondition event you describe, so you're adding to the existing code, not replacing it. Furthermore, you now have to build some hysteresis or rate-limiting into the system so that the user doesn't get pelted with low disk space warnings when something is causing the space on the disk to alternate above and below the limit.

In all, this seems like a fairly complex solution to a rather trivial problem. Very few people need low disk space warnings at a time resolution smaller than 10 minutes, and adding complexity to the kernel and every disk filesystem driver is not worth the minor benefit.

alegr1
18 Jul 2012 7:39 AM
#

//hEvent = OpenEvent(GENERIC_READ, "\\?\KernelObjects\LowDiskCondition");

Is not going to work (other than the backslashes has to be doubled).

A disk may show different amount of free and available space to each user, because of NTFS quotes. The event has to be per user. This is why FSCTL would be better.

Also, GENERIC_READ is for files, not for events. You want SYNCHRONIZE.

ErikF
18 Jul 2012 9:56 AM
#

Making the kernel responsible for monitoring disk space for a user-mode program is problematic because it starts adding dependencies between the FS drivers and Win32 (specifically the registry). As the file system is loaded before the registry, you would then need two different functions to access the disk: one for before Win32 is available and one for after.

All this just to let the user know a couple of minutes sooner that a disk is running out of space? I'd say that it doesn't make the -100 bar, let alone making it a candidate for the next version of Windows!

Matt
18 Jul 2012 10:13 AM
#

Nice to see other people joining in with suggesting ideas, rather than shooting other ones down...

@alegr:

>>> "the backslashes has to be doubled", GENERIC_READ/SYNCHRONIZE

I left out the extra slashes for readability, but thanks for the pedantry.

>>> "The event has to be per user"

Why? If the disk is low on space for me, it's low on space for all of the other users as well. But if you really want it to be per-user, just read the quota from a HKCU regkey instead of a HKLM one. You probably want to have both, because running low on quota isn't the same as running low on disk space, and explorer should probably show different messages for those events.

I should also clarify that my idea doesn't absolve explorer from needing to do work. It just means it doesn't have to poll. Think of the event as a "Hey, some filesystem says a low disk event might be going on for some user on some drive". The user then has to decide which one.

@gabe:

>>> How does your code account for the fact that there are actually 4 different thresholds.

```
for(int i=0; i < 4; i++)
{
  What I just said.
}
```

>>> and they differ depending on the size of the volume?

So? The filesystem gets to choose when to signal the event. It's easy for each filesystem and each volume to choose different criteria.

>>> How does it handle different per-user quotas?

If you really want to, adapt this solution to read from HKCU and then you have per-volume per-user quotas as an event, and explorer can wait on that.

@erikf:

>>> Making the kernel responsible for monitoring disk space for a user-mode program is problematic because it starts adding dependencies between the FS drivers and Win32 (specifically the registry)

a) The kernel is already (deliberately and permanently) tied to Win32

b) The registry is loaded pretty early on in boot (before any user-mode processes or non-boot kernel-mode drivers load). This doesn't sound like a hard problem to rectify. I could also be a pedant and point out that winload.exe loads the registry using the boot-fs driver before winload loads the kernel fs driver (in order to see what other boot-drivers need to be loaded), so the registry actually gets loaded before any of ntos.

c) It doesn't add any dependencies. It exposes an event. It doesn't care if or who listens to the event. They can subscribe if they want, or not if they don't. That's the joy of event-based programming.

>>> "All this just to let the user know a couple of minutes sooner that a disk is running out of space?"

No. My objection wasn't the slight delay. It's the principle of polling to watch for an event. If polling is the answer, you're probably asking the wrong question.

[*This makes the file system responsible for determining the notification thresholds. What if Windows 9 wants to change the threshold or add a new fifth threshold at 1GB? What if there are two clients subscribed to the low disk space notification, one from Windows 7 and one from Windows 9? Which threshold does the file system use? It's also strange to have the file system aware of what is basically a UI feature. If you really want to do this via a file system notification, it would have to be a per-handle asynchronous IOCTL, like FSCTL_LOW_DISK_SPACE where the input parameter describes the threshold at which to complete the IOCTL. -Raymond*]

Joshua

18 Jul 2012 10:37 AM
#

[it would have to be a per-handle asynchronous IOCTL, like FSCTL_LOW_DISK_SPACE where the input parameter describes the threshold at which to complete the IOCTL. -Raymond]

If you built it, I'd use it.

> [*Given that this involves a kernel change, its starting point is probably minus 500 points. -Raymond*]

alegr1
18 Jul 2012 11:07 AM
#

[bResume is for waking from suspend, which is not the same as waking from a low-power state]

Last time I checked, an idle desktop still needs to update the clock on the taskbar every minute. Whatever C-state the system was, it's now back to C0.

alegr1
18 Jul 2012 11:10 AM
#

[The point about Terminal Server is that you have all these threads waking up and paging in stacks constantly. -Raymond]

If marginal activity affects your system performance, you need to rethink your system specs. Oh no, an user clicks the text format button every minute, the TS is so overwhelmed. Oh no, the Word autosave is every minute, the TS churns to a stop. This only means you should not run a 60 users TS on a 512 MB box.

Gabe
18 Jul 2012 11:20 AM
#

Matt: Your solution doesn't eliminate polling; it merely tells you when to poll. To do it "properly", you need a special API like Raymond's FSCTL method. That would allow Explorer to be notified exactly when each of its messages needs to be displayed or hidden. And now that I think about it, if you have lots of filesystems you might even need a whole extra thread to wait for all these signals.

Pros: Polling would only be required for filesystems that don't support the notification mechanism. There would be no need for the kernel to know about Explorer's arbitrary thresholds. Users would be able to change their thresholds without having to alert the filesystem about a change in system metrics. Other programmers would be able to use the mechanism without being beholden to Explorer's preset thresholds.

Cons: Such a system is a global solution to a local problem. I'm sure there are many parts of Explorer that could be improved with support from the kernel, but imagine what would happen if everybody with a problem went to the kernel to solve it.

alegr1
18 Jul 2012 11:40 AM
#

[Given that this involves a kernel change, its starting point is probably minus 500 points. -Raymond]

This didn't stop ReadDirectoryChanges implementation.

> [*I didn't say "minus infinity points." -Raymond*]

Not important
18 Jul 2012 11:59 AM
#

Raymond - I hope you realize that, even though the shell honors its contract, this is a terrible user experience. 10 minutes is way to late and the notification is worthless. (If I start a bulk copy operation which will put me over the disk space, the operation will fail well before the notification appears).

Nick
18 Jul 2012 1:49 PM
#

@Crescens2k:

The reason a comment about Explorer holding handles open is troll bait is because (1) it IS partially true, and (2) many people get very much up in arms about it. Regarding (1), the biggest offender are handles to Thumbs.db. I can't count the times I get errors about not being able to delete a folder because Thumbs.db is "open in another program". It's always fixed by checking Process Explorer to verify it's Explorer, closing the Explorer window I'm using and opening a new one, and then trying the delete operation again.

@alegr1:

"the backslashes [have been] doubled"

HUZZAH! :)

640k
18 Jul 2012 3:43 PM
#

@Raymond: ["Hey, I can't reformat my hard drive because Explorer is monitoring it for low

disk space." -Raymond]

Do you now see why the NT architecture is worse than windows 95?

Prevention to modifying files & dirs because of open handles is STUPID, and people HATE it.

It does NOT prevent changes (physical removable media is an obvious example), it only make it harder to perform them, therefore you cant trust open handles anyway.

640k
18 Jul 2012 3:50 PM
#

And I may add, I've lost count on how many times I've implemented retry-logic for recursively removing folders. At least in a dozen different languages. None of ms frameworks/apis has this included.

Maurits [MSFT]
18 Jul 2012 4:38 PM
#

> also strange to have the file system aware of what is basically a UI feature

I think an OnDiskSpaceRunningOut callback would be of use to (say) database management systems as well. Flush the tempdb, send alerts to sysadmins.

alegr1
18 Jul 2012 4:39 PM
#

>Do you now see why the NT architecture is worse than windows 95?

Ugh, Windows 95 was worse in that. It didn't support FILE_SHARE_DELETE at all.

Joshua
18 Jul 2012 4:42 PM
#

640k wrote:

> Prevention to modifying files & dirs because of open handles is STUPID, and people HATE it.

So do developers. I _really_ wanna find where in the kernel that check is and NOP it out (as if all handles have FILE_SHARE_DELETE). Any filesystem that has hard links should be able to delete a file that is open. The disk space will be reclaimed when the last handle is closed.

alegr1
18 Jul 2012 4:52 PM
#

>Any filesystem that has hard links should be able to delete a file that is open.

It doesn't work the way you think it works. If you issue Delete on a file with FILE_SHARE_DELETE, it only will be marked for deletion on the last handle closed, as if you opened it with FILE_FLAG_DELETE_AFTER_CLOSE. The name will still appear in the directory. You won't be able to create a duplicate file with the same name right away. You won't also be able to open that file name again.

Maurits [MSFT]
18 Jul 2012 5:04 PM
#

> Which threshold does the file system use

There are many ways to answer this question.

One is to have the file system raise the event every time the disk utilization goes up by a percent. Then the client can ignore all of the events except the one that corresponds to its own threshold.

Another option is to tell the driver what the exact threshold is.

Gabe
18 Jul 2012 11:09 PM
#

Maurits: You can't use 1% of drive space as a threshold. I can get a 2TB HD for about $100. When it has 1% free, it will still have 20GB available. I don't want to be notified when there's 20GB free! Explorer's 200MB threshold is 0.01% free. The 1MB threshold corresponds to half of a millionth of the disk space being free.

Matt
19 Jul 2012 1:35 AM
#

@Raymond:

>>> "This makes the file system responsible for determining the notification thresholds."

No it doesn't. It makes NtGetSystemParameters or whomever determine the thresholds. The volume manager merely triggers the notification - which it must do, since it is a volume notification, and only the volume manager knows when the amount of free-space in the volume changes.

But then again, maybe you're one of the people who phones up the library every ten minutes saying "Has the book I ordered come in yet? No? OK I'll phone back in ten minutes". A better way would be to phone them up and say "Please phone/text/email me when my book comes into the library" or at the very least "please phone me when you have a book delivery, so I can ask whether the book I wanted came in with that delivery". That way you're not phoning them all the time, wasting your and their time, and you also get your notification an average of 5 minutes and at best 9 minutes 59 seconds earlier than with the every-ten-minute phone call. Everyone wins.

>>> "What if Windows 9 wants to change the threshold or add a new fifth threshold at 1GB?"

Then Windows9 changes the implementation of NtGetSystemParameters(SM_GETLOWDISKTHRESHOLDS) to reflect that.

>>> "What if there are two clients subscribed to the low disk space notification, one from Windows 7 and one from Windows 9? "

That's why you wrap them in a lovely user-mode api in kernel32 or user32. Explorer can call that.

>>> "Which threshold does the file system use?"

The one that NtGetSystemParameters tells it to use. Win32k already has lots of ways to abstract random configuration values out of system components.

>>> "It's also strange to have the file system aware of what is basically a UI feature."

It's not a UI feature, it's an event. If the UI wishes to use this event for showing a balloon tip, great. If the Task Scheduler wants to use it as a trigger for starting a process, that's cool too. If explorer wants to use it as a trigger to start deleting or compressing temporary files on disk, sure that's great as well, or if a database wants to use it as a warning then who am I to say no? It's only a UI feature because you're looking at it with your UI-feature goggles on.

@gabe:

>>> "if you have lots of filesystems you might even need a whole extra thread to wait for all these signals."

WaitForMultipleObjectsEx

>>> "You can't use X% ... Explorer uses Y%!"

Read Maurits' post, reading "a" in "goes up by a percent" as an algebraic term, as in, "goes up by n percent". If you're going to attack ideas, attack the idea, not the finger-in-the-air values used for descriptive purposes.

> [*Say you have a machine running Windows Server 2008. NtGetSystemParameters will return the Windows Server 2008 threshold. You have a client running Windows 7. The client connects to your server and says, "Let me know when the disk space gets low." The server will call NtGetSystemParameters and retrieve the server's threshold, which is not the same as the threshold that Windows 7 wants. You then get people saying, "Sometimes I'm warned when the disk space drops below 2GB, and sometimes I don't get warned until it drops below 1GB." In your library case, suppose somebody wants to know when there are fewer than 3 copies of the book available. They call the library and say "Let me know when the book availability gets low" and the library looks at its NtGetSystemParameters(BOOK_ALERT_LEVEL) and sees that the value is 2. The book count drops to 2. No call is made. Customer later casually notices that*

the book count is 2 and says, "Hey, I told you I wanted to know when the book count dropped below 3!" Also, the threshold needs to be adjusted so it doesn't notify constantly if disk space fluctuates just above/below the threshold. The correct way to do this (if it is done at all) is to have an IOCTL that can be fired per handle called "Complete when disk space goes below X or above Y." But you still have to poll, because you may have lost connectivity to the server, which closed the handle! - Raymond]

Maurits [MSFT]
19 Jul 2012 8:12 AM
#

Gabe's point is valid; users will be far more interested in "0.03% available" vs. "0.02% available" than in "47.03% available" and "47.02% available."

Going up a level, I wonder if what the user would actually want is something like "it looks like you've been writing a lot of data to this drive recently; if you keep it up you'll run out of space in about three minutes."

Obvious solution: have WriteFile run slower and slower as the disk runs out of space. That way the disk will never be full.

JamesJohnston
19 Jul 2012 9:03 AM
#

"And I may add, I've lost count on how many times I've implemented retry-logic for recursively removing folders. At least in a dozen different languages. None of ms frameworks/apis has this included."

Is that why Git on Windows (MSysGit) seems to be unable to fully clean a code repository when there is a lot to clean? Suppose I have dozens of directories that need to be removed as part of cleaning. Sometimes I might have one or two directories remaining after the clean, that should have been deleted. I always wonder what's up with that? I have to clean twice to be sure everything is clean. What causes that?

Klimax
19 Jul 2012 10:05 AM
#

@Matt 18 Jul 2012 10:13 AM:

"The kernel is already (deliberately and permanently) tied to Win32"

This is absolutely wrong as it can only get. Kernel is second lowest layer in NT architecture separate from HAL and upper levels. See: p.blog.csdn.net/.../o_WinNtKernel2-3.bmp for illustration. Win32 is subsystem as is POSIX. Kernel provides services to them, through another few layers but itself is independent of them. (As can be seen subsystems are user-mode , which is quite different thing from kernel mode.)

For more see Windows Internals by Mark Russinovich. (Very detailed)

f0dder
19 Jul 2012 12:17 PM
#

@JamesJohnston: "GIT on Windows", or TortoiseGIT? Tortoise causes this often, whether it be git or svn or hg. Haven't seen it from a shell, though.

I'm personally pretty fond of Windows' "gestapo-locking", having run into various "funny episodes" on *u*x... it sure does suck when you're dealing with badly programmed applications (including explorer!), but the alternative is worse.

Matt
19 Jul 2012 3:43 PM
#

@Klimax

>>> "This is absolutely wrong as it can only get ... For more see Windows Internals by Mark Russinovich. (Very detailed)"

I worked with the guy who co-wrote that book, and you're wrong. NT used to be decoupled, but has been deliberately tied to Win32 since Vista. They are still developed by different teams, but it is not possible to separate them. ntoskrnl loads win32k.sys during boot, and that is not configurable.

Joshua
19 Jul 2012 9:15 PM
#

@alegrl1: The existence of incorrect implementations does not preclude correct implementations. Besides, I could always rename it to .nfsXXXXXXXXX first.

Klimax
20 Jul 2012 10:16 AM
#

@Matt:

Really? Anonymous commenter supposedly says some things, which are not backed by anything. Got something to back supposed tying?

(For now I'll go with the books over Anon poster)

Anyway: Win32k is loaded by Csrss.exe which is started by Smss which is started by kernel(). Second, fact that something would load/start something doesn't make it tied to it... Also requriment of Windows subystem is not in kernel.

So at minimum one thing you posted is already shown to be wrong, what else can be then wrong in your post?

Matt
21 Jul 2012 11:29 AM
#

@Klimax

OK. I'll play ball.

Firstly, let's actually read the Windows Internal's book, rather than just assume it agrees with you. I quote (page 50, WinInternals Part 1 in Chapter "Environment Subsystems and Subsystem Dlls" subchapter "Windows Subsystem"):

"Although Windows was [originally] designed to support multiple, independent environment subsystems, from a practical perspective, having each subsystem implement all the code to handle windowing and display I/O would result in a large amount of duplication of system functions that, ultimately, would negatively affect both system size and performance. Because Windows was the primary subsystem, the Windows designers decided to locate these basic functions there and have the other subsystems call on the Windows subsystem to perform display I/O. Thus, the SUA subsystem calls services in the Windows subsystem to perform display I/O.

As a result of this design decision, the Windows subsystem is a required component for any Windows system, even on server systems with no interactive users logged in. Because of this, the process

is marked as a critical process (which means if for any reason it exits, the system crashes)."

Secondly, if you disagree with both the book AND me, we can just run strings.exe against ntoskrnl.exe and you'd see win32k in there, plain as day, including the path that windows loads it from.

And finally, in an ironic twist to your claim "So at minimum one thing you posted is already shown to be wrong", I'll be a pedant and point out that Win32k isn't actually loaded by CSRSS at all (in fact, csrss.exe is loaded from win32k.sys). Win32k.sys is loaded by ntoskrnl in response to an NtSetSystemInformation syscall from inside Smss.exe. Csrss is the user-mode part of win32k.sys (once again, you'd know this if you actually read the book you're using to try and disagree with me).

And just in response to which ever smart alec actually DOES follow this through and points out that the second parameter to the NtSetSystemInformation syscall is a UNICODE_STRING version of "\\SystemRoot\System32\win32k.sys" parameter and foolishly thinks this is proof that win32k.sys's loading is actually decoupled from NT (which it used to be before vista - kinda like how I said "NT used to be decoupled from Windows before Vista"), let's actually follow the call in WinDbg, IDA or whatever disassembler takes your pick.

NtSetSystemInformation(

__in DWORD p1,

__in void* p2,

```
 __in size_t sizeP2)

{

 switch(p1)

 {

   case SystemExtendServiceTableInformation:

     if(CurrentMode != KernelMode)

     {

       if(!Param2IsAValidUnicodeString()) return E_GOAWAY;

       if(!IsSmssExe()) return E_GOAWAY;

       if(0 == memcmp(Param2,"\\SystemRoot\\System32\\win32k.sys")) return
E_GOAWAY;

     }

     MmLoadSystemImage(ParameterWhichMustBeWin32kDotSys);

 }

}
```

Thanks for playing. Better luck next time.

Klimax
22 Jul 2012 5:27 AM
#

"Windows subsystem is a required component " =! "Kernel is tied to it." (So, misinterpretation of book noted) Also you're still trying to peddle, that loading == tied => Still wrong.

Also you missed "The Windows executive is the upper layer of Ntoskrnl.exe. (The kernel is the lower layer.)" It's possible to have components well defined yet in one executable.

"And finally, in an ironic twist to your claim "So at minimum one thing you posted is already shown to be wrong", I'll be a pedant and point out that Win32k isn't actually loaded by CSRSS at all (in fact, csrss.exe is loaded from win32k.sys). Win32k.sys is loaded by ntoskrnl in response to an NtSetSystemInformation syscall from inside Smss.exe. Csrss is the user-mode part of win32k.sys (once again, you'd know this if you actually read the book you're using to try and disagree with me)."

One thing I had wrong (misread sentence) is that csrss starts win32k, that is started by Smss. (still not kernel)

From Windows Internals 5th edition; 13.1.5 Smss, Csrss, and Wininit p.987:

"17. At this point, SmpLoadDataFromRegistry returns to SmpInit, which returns to the main

thread entry point. Smss then creates the number of initial sessions that were defined

(typically,

only one, session 0, but you can change this number through the NumberOfInitialSessions registry

value in the Smss registry key mentioned earlier) by calling SmpCreateInitialSession, which

creates an Smss process for each user session. This function's main job is to call SmpStartCsr to

start Csrss in each session.

18. As part of Csrss's initialization, it loads the kernel-mode part of the Windows subsystem

(Win32k.sys). The initialization code in Win32k.sys uses the video driver to switch the screen to

the resolution defined by the default profile, so this is the point at which the screen changes from

the VGA mode the boot video driver uses to the default resolution chosen for the system"

Anyway your whole reply is still in contradiction to whole chapter 13. (Currently part2 with it of 6th edition is unreleased.) Your supposed RI is bit suspect, but can't prove it as I don't have necesary skills with debuggers nor access to advenced tools. (but I'll take a look at procmon and its bootlogging)

"Thanks for playing. Better luck next time."

It didn't end yet...