

Spread the word!

[Follow @arvisam](#)
[Like](#)

0

[Tweet](#)

3

Debugging story: Slowness due to NTFS short file (8.3) name generation


[Arvind Shyamsundar](#) 19 Jun 2013 10:04 PM

[Q](#)

When I teach production debugging to my customers, I always tell them that to be successful you need to **not only know the right tool** and command syntax, but also know the right **methodology**. And perhaps even more importantly when debugging certain types of issues, knowledge of Windows Internals and the ability to '**connect the dots**' is a savior.

Recently, I had one such experience wherein an issue with an application turned out to be related to a NTFS behavior (which I had previously read about in my SQL Server work) and had I not 'connected the dots' I would not have been able to remediate the issue quickly. Wondering what it is all about? Read on!

Scenario

So this was a strange problem: a server had hundreds of processes which were all seemingly 'hung' and not doing anything. The server in question was hosting a vendor application, which had spawned those processes, but normally those processes should execute quickly and not accumulate the way they had done.

This appeared to me like the processes were hung in some way. Since the application was deployed as a service, these child processes were on a non-visible desktop, so maybe they had popped up an UI element – that was my first theory. But as with any user mode process hanging situation, I decided to collect hang mode dumps and look at those to be sure about the root cause. I took sample dumps from 2 processes using [DebugDiag 1.2](#).

Dump analysis

Once the dumps were available, I opened them in WinDbg (x86, as these were 32-bit application processes.) Here are the call stacks of the dump for the first process:

```
0:000> kL
ChildEBP RetAddr
0018d368 74cd9a04 ntdll!NtSetInformationFile+0x12
0018d43c 74cecd44 kernel32!MoveFileWithProgressTransactedW+0x334
0018d474 74cecc5c kernel32!MoveFileWithProgressTransactedA+0x5b
0018d494 74d3d93f kernel32!MoveFileWithProgressA+0x1b
0018d4b0 00401302 kernel32!MoveFileA+0x16
WARNING: Stack unwind information not available. Following frames may be wrong.
0018d4cc 00401aa2 fooapp+0x7777
0018ff94 76fb9ef2 kernel32!BaseThreadInitThunk+0xe
0018ffd4 76fb9ec5 ntdll!_RtlUserThreadStart+0x70
0018ffec 00000000 ntdll!_RtlUserThreadStart+0x1b
```

The above call stack is due to a file being moved. In this case by dumping the parameters to MoveFileA, we found that the file was being renamed in the same folder (the file extension was being changed from .FOO to .BAR – file extensions changed to protect the identity 🙄). The call stack for the thread in the second process is given below:

```
0:000> kL
ChildEBP RetAddr
0018d38c 74c7c5dc ntdll!NtCreateFile+0x12
0018d430 74cc3f86 KERNELBASE!CreateFileW+0x35e
0018d45c 74cc53e4 kernel32!CreateFileWImplementation+0x69
0018d48c 004012b3 kernel32!CreateFileA+0x37
WARNING: Stack unwind information not available. Following frames may be wrong.
0018d4c0 004019df fooapp+8888
0018ff94 76fb9ef2 kernel32!BaseThreadInitThunk+0xe
0018ffd4 76fb9ec5 ntdll!_RtlUserThreadStart+0x70
0018ffec 00000000 ntdll!_RtlUserThreadStart+0x1b
```

The above is a call stack of the application creating a new file. By looking at both of these, it appeared to me that the hard disk was performing slowly. So we extracted the file name (the first parameter to CreateFileA routine) and found that it was a new file within a folder on the D: drive. When we ran performance counter logs for the D: drive, it appeared to be just normal.

So it was **quite puzzling** to see the **calls to creating and renaming files taking so long** despite the physical drive performing quite well.

Root cause

We then looked a bit further by browsing to the folder on D: drive which was being accessed by the above application. We could not even list the contents (Explorer would appear unresponsive.) It then came to our mind that when we have a **large number of similarly named files in the same folder**, (which was exactly the case with this application) then NTFS has to work extra hard to generate to unique 8.3 naming convention names (so called 'short file name'.) This [TechNet article](#) mentions a threshold of 300000 files above which this short name generation can become a bottleneck. The algorithm for the short file name generation is also [documented here](#) and there is another very interesting but [really old KB article](#) which shows how to achieve the name conversion in FoxPro 😊

Now, these names are more of a legacy remnant of DOS and the 16-bit world, and in most servers there should not be any reason to use them. So thankfully [we have a way](#) to **disable the generation of these short file names**. I then tried to evaluate the effect that disabling this has. My test results are given below.

Testing

I executed a series of tests with a simple PowerShell script which creates and renames file in a tight loop. For each case, the folder already contained over a million files were already created in the same folder. My tests were done on a Windows 8 laptop with a single spindle SATA hard disk.

| Test Description | File Count per process | Avg. time (8dot3 ON) | Avg. time (8dot3 OFF) | % Improvements |
|-------------------------------------|------------------------|----------------------|-----------------------|----------------|
| Single process creating files | 25000 | 126028 | 102617 | 19% |
| 5 process creating + renaming files | 25000 | 567611 | 375946 | 34% |
| 20 process creating files | 25000 | 2420367 | 1422348 | 41% |

Conclusion

The benefits of disabling 8dot3 name generation are obvious from the test results above, and validate the guidance given in the KB articles mentioned previously. And keep in mind these results were obtained on Windows 8 and Windows 2008 R2, wherein the algorithm for 8dot3 name generation has been further tuned. The gains might be even more dramatic on Windows 2003.

So in summary, we would like to **confirm the benefits of disabling 8dot3 name generation** on servers, where there are large number of similarly named files in the same folder. And for SQL Servers using FILESTREAM storage, this [setting is recommended](#) as well.

Further Reading

For more information on NTFS, I encourage everyone to take a look at the presentation that **Neal Christiansen** (Principal SDE, Microsoft) delivered in a community meeting. The presentation is found here: [NTFS - The workhorse file system for the Windows Platform](#). Neal also has presented two excellent talks on Channel9, which can be accessed [here](#) and [here](#). Amazing information, take a look!

Jose Barreto, Principal PM at Microsoft [has blogged](#) about the performance impact of 8.3 names. His [TechEd 2011 presentation](#) is also worth a look.

On a side note, [Adi Oltean](#) talks about potential security considerations when the 8.3 filename is used, do take a look!

Spread the word!

Follow [@arvisam](#)

Like

0

Tweet

3

Comments
