# Determing cause of access denied - USN Journal

Posted by Will Steele - 01 Mar 2010 - 18:44

---

I have a Vista SP1 workstation on which I would like to shrink the primary partition. I have run a few tools to try and consolidate data to the front of the drive so I can then run the shrink functionality of disk management. The tools have shown me that the problem lies in a few USN journal files near the end of the drive that will not move (or allow deletion.) When I run the fsutil command logged in under the local machine administrator account against the drive I get this error (command and error shown below):

C:UsersAdministrator>fsutil usn deletejournal /D C:
Error: Access is denied.

Someone suggested on the Technet forums I then ran the shell as system using this command:

C:usersadministrators>psexec -s fsutil usn deletejournal /D C:
Error: Access is denied.

fsutil exited on MYMACHINE with error code 1.

As I suspected, system privileges did not allow me to delete the entries. So, I am looking to use WinDbg to figure out what process or system structure is preventing the utility from deleting these files. I suspect it is a "feature" but I don't know if it is a bug. How would you debugging experts approach this issue?

==========

# Re: Determing cause of access denied - USN Journal

Posted by Robert Kuster - 04 Apr 2010 - 23:48

---

Hi Will.

Some solutions to this issue have been provided here:
www.mydefrag.com/forum/index.php?topic=1131.0

To peek under the hood:

Behind the scenes "fsutil usn deletejournal /D C:" translates into a call too fsutil!DeleteUsnJournal. Its pseudo-code looks like this:

```
fsutil!DeleteUsnJournal()
{
  fsutil!IsVolumeLocalNTFS(..)
  hVolume = kernel32!CreateFileW(sVolume);   // sVolume: "\.c:", OR "\.d:", OR..
  kernel32!DeviceIoControl(hVolume, FSCTL_DELETE_USN_JOURNAL, ..)
  fsutil!DisplayError(..)
}
```

In short you can attach WinDbg to fsutil.exe, set a breakpoint on DeleteUsnJournal, and debug through it. It can actually only fail on two places:

1) as it tries to open a handle to the volume
2) in the call too DeviceIoControl( FSCTL_DELETE_USN_JOURNAL )

After you find the failed function you could execute a !gle (get last error) in WinDbg. With some luck you will get a hint about what failed. With less luck you might end up debugging FSCTL_DELETE_USN_JOURNAL throughout kernel mode. The stack in the kernel looks something like this:

```
1: kd> kb
ChildEBP RetAddr Args to Child
acb11adc b9dbdca6 88928a38 88666008 acb11b20 Ntfs!NtfsDeleteUsnJournal
acb11af0 b9da7adc 88928a38 88666008 acb11b20 Ntfs!NtfsUserFsRequest+0x325
acb11b04 b9da7a36 88928a38 88666008 8a6c2020 Ntfs!NtfsCommonFileSystemControl+0x44
acb11b78 804ef19f 8a6c2020 88666008 806e6428 Ntfs!NtfsFsdFileSystemControl+0x116
acb11b88 80658128 8a7294d8 8a729590 88666008 nt!IopfCallDriver+0x31
acb11bac b9e26ee5 8a6a4040 00000000 acb11bf0 nt!IovCallDriver+0xa0
acb11bbc 804ef19f 8a729590 88666008 806e6428 sr!SrFsControl+0x121
acb11bcc 80658128 889f04c0 88666008 00000000 nt!IopfCallDriver+0x31
acb11bf0 b9e43837 88666000 889f04c0 8a786040 nt!IovCallDriver+0xa0
acb11c1c 804ef19f 889f04c0 88666008 806e6428 fltmgr!FltpFsControl+0xd7
acb11c2c 80658128 8a70e130 806e6410 88666008 nt!IopfCallDriver+0x31
acb11c50 8057f982 886661bc 8a70e130 88666008 nt!IovCallDriver+0xa0
acb11c64 805807f7 889f04c0 88666008 8a70e130 nt!IopSynchronousServiceTail+0x70
acb11d00 805792a8 00000788 00000000 00000000 nt!IopXxxControlFile+0x5c5
acb11d34 8054163c 00000788 00000000 00000000 nt!NtFsControlFile+0x2a
acb11d34 7c90e514 00000788 00000000 00000000 nt!KiFastCallEntry+0xfc
0007faf8 7c90d3aa 7c80174d 00000788 00000000 ntdll!KiFastSystemCallRet
0007fbf8 7c9100b8 00090330 0007fcd0 7c910041 ntdll!ZwFsControlFile+0xc

1: kd> !irp 88666008
Irp is active with 10 stacks 10 is current (= 0x886661bc)
No Mdl: System buffer=887e9400: Thread 88600020: Irp stack trace.
cmd flg cl Device File Completion-Context
> 4 0 8a6c2020 8a70e130 00000000-00000000
FileSystemNtfs
...
```

A good point to start you investigation is nt!NtFsControlFile as obviously something between it and Ntfs!NtfsDeleteUsnJournal fails.
The following breakpoint will stop the kernel for any FSCTL_DELETE_USN_JOURNAL request:

```
bu nt!NtFsControlFile ".if( poi(@esp + 18) == 0x000900f4 ) { .echo *****
FSCTL_DELETE_USN_JOURNAL *****; } .else { g;};"
```

Explanation:
> the sixth parameter is the dwIoControlCode which was passed to DeviceIoControlCode
> FSCTL_DELETE_USN_JOURNAL == 0x000900f4

You can also check out these articles:
Keeping an Eye on Your NTFS Drives: the Windows 2000 Change Journal Explained
Keeping an Eye on Your NTFS Drives, Part II: Building a Change Journal Application

I hope this helps,
Robert

==========