How Windows Shuts Down

blogs.msdn.microsoft.com/ntdebugging/2007/06/08/how-windows-shuts-down

ntdebug June 8, 2007

Hi my name is Bryan, I'm a escalation engineer on the Microsoft CPR platforms team. A common problem scenario involves shutting down Windows. When troubleshooting problems during shut down we have to look at the Winlogon.exe process which can be tricky and must be done correctly.

Troubleshooting Techniques

Common Settings

Settings that would affect how Windows shuts down are mostly in HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon, and are documented thoroughly at http://www.microsoft.com/technet/prodtechnol/windows2000serv/reskit/regentry/12314.mspx. The setting to really pay attention is:

DisableCAD (REG_DWORD). When set to 1 the GINA will bypass the Ctrl-Alt-Del dialog and go straight to the logon prompt. *The system cannot be shutdown when the logon prompt at the console is displayed.*

Another common setting that is sometimes needed for security reasons is in HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management.

ClearPageFileAtShutdown (REG_DWORD). When set to 1 the memory manager will clear the paging file on shutdown. *If the paging file is large this can significantly the time it takes for the system to shut down*.

Winlogon Event Notification

Prior to Windows Vista/2008 Server, Winlogon Notification Packages can be used to reliably monitor the state of Winlogon. These are simple export DLLs that are registered with entry points for each event monitored by the package. Winlogon explicitly calls these entry points when a specific event occurs. Each package handles the notification within the context of the Winlogon process. Keep in mind that a bug in a notification package can cause problems in Winlogon and render the system inaccessible. You can find more information about Winlogon Notification Packages at http://msdn2.microsoft.com/en-us/library/aa380545.aspx.

Starting with Windows Vista support for Winlogon Notification Packages has been pulled. Most of this functionality is still provided through SCM event notifications. See the following link for more information about this.

http://technet2.microsoft.com/WindowsVista/en/library/6ec4ec6d-6b84-44c9-b3af-116589a42b861033.mspx?mfr=true

You can find information about how to write a service here.

http://msdn2.microsoft.com/en-us/library/ms685969.aspx

However these notifications are performed asynchronously so the exact moment of each state within Winlogon cannot be reliably known.

Winlogon Logging

Winlogon logging is obtained by applying the checked build of Winlogon and setting the following registry values under HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon.

DebugFlags (REG_SZ). This is a comma-separated list of debug flags to capture. The list of flags is Error, Warning, Trace, Init, Timeout, Sas, State, MPR, CoolSwitch, Profile, DebugLsa, DebugSpm, DebugMpr, DebugGo, Migrate, DebugServices, Setup, SC, Notify,

and Job.

LogFile (REG_SZ). This is the path to the log file where the events should be written. If this value is missing then the events are written to the debug port.

These options can also be placed in the win.ini file. This option and other debug logging options for Winlogon can be found in these articles.

http://support.microsoft.com/default.aspx?scid=kb;EN-US;232575

http://support.microsoft.com/default.aspx?scid=kb;EN-US;221833

Checked builds of Windows binaries are available through MSDN subscriptions. The checked binary applied to the system must match the service pack level of the system.

When looking at the Winlogon logs there are three check points to identify. They are the three ExitWindowsEx calls made during shutdown. These log entries were made during a shutdown initiated with “shutdown –r –f –t 0”.

328.372> Winlogon-Trace: Doing remote-initiated (Administrator) **Reboot=true**, **Force=true**

328.372> Winlogon-Trace: Starting user thread for Logoff, flags = 3807

328.764> Winlogon-Trace: Doing immediate shutdown, LastGinaRet = b, Flags = 3804

328.764> Winlogon-Trace: Calling **ExitWindowsEx**(0x3804, 0)

.

.

.

328.1528> Winlogon-Trace: **ExitWindowsEx** called to shut down COM processes

.

.

.

328.332> Winlogon-Trace: Starting shutdown

328.332> Winlogon-Trace: Starting user thread for Logoff, flags = 7

.

.

.

328.284> Winlogon-Trace: Calling **ExitWindowsEx**(0x7, 0)

If shutdown stalls then based on the log entries you should be able to identify which part of shutdown it is stuck in.

Debugging

In order to debug Winlogon you will need to first install the debugging tools for Windows (http://www.microsoft.com/whdc/devtools/debugging/default.mspx). It contains information about how to set up the debugger for Winlogon debugging. ***If the debugger is not set up correctly on Winlogon the system can easily get in an unusable state and the system will need to be reinstalled.***

You should debug Winlogon whenever a problem manifests itself on the Winlogon desktop where the SAS and logon prompt dialogs are displayed. When debugging Winlogon you should start by looking at the main process thread. If shutdown (or any other Winlogon activity) is hung that thread will show what happened to prevent shutdown from completing. Also look at these Winlogon flags which provide some indication of whether or not the main thread is even processing a shutdown.

0105fe8c winlogon!g_fReadyForShutdown

0105fdf8 winlogon!ShutdownHasBegun

01062b3c winlogon!ShutdownInProgress

01062b30 winlogon!ShutdownTime

I will point out how these variables are modified.

Shutdown Sequence of Events

If you are debugging a shutdown issue you first need to understand the sequence of events that take place during shutdown. This will help isolate what activity may be failing.

RPC Call

An application, local or remote, tells Windows to shut down by calling the Win32 API ***InitiateSystemShutdownEx*** (http://msdn2.microsoft.com/en-us/library/aa376874.aspx). This API creates an RPC call over the named pipe *InitShutdown*. The remote connect and RPC call looks like this in a network trace.

11:08:40.025CLIENTSERVERSMBSMB: C; Nt Create Andx, FileName = \InitShutdown

11:08:40.027CLIENTSERVERMSRPCMSRPC: c/o Request: unknown Call=0x1 Opnum=0x2 Context=0x0 Hint=0x20

A local connect would not need to go over the network, but it uses the same mechanism to make this call.

The server side of this RPC call is handled by the function ***BaseInitiateShutdownEx*** in the Winlogon.exe process. This RPC worker performs the following actions.

1.Checks the privilege level of the client. If the client privilege test fails then the error status code is returned. This will typically be ERROR_ACCESS_DENIED (0x5).

2.Parses the shutdown command sent by the RPC client. If the command is malformed then the status error code of ERROR_INVALID_PARAMETER (0x57) is returned.

3.Checks the Winlogon flags *ShutdownInProgress*, *ShutdownHasBegun*, *g_fReadyForShutdown*, and the current state of the terminal desktop to see if we can shutdown. Winlogon cannot begin the shutdown if a shutdown is in progress or if it is not ready. Winlogon also will not start a shutdown if the force flag was not specified in the command and the desktop is locked. If the *ShutdownInProgress* or *ShutdownHasBegun* flag is set then the return value is ERROR_SHUTDOWN_IN_PROGRESS (0x45B). If the *g_fReadyForShutdown* is not set then the return value is ERROR_NOT_READY (0x15).

Debugger output:

dd winlogon!g_fReadyForShutdown l 1

0105fe8c00000001

dd winlogon!ShutdownInProgress l 1

01062b3c00000000

dd winlogon!ShutdownHasBegun l 1

0105fdf800000000

4.Winlogon initializes the shutdown data including the global variable *ShutdownTime*. If this variable has been set then we know we've gotten this far.

Debugger Output:

dq winlogon!ShutdownTime l 1

01062b3001c7a859`baee0060

.formats 01c7a859`baee0060

Evaluate expression:

Time:Wed Jun6 12:42:54.506 2007 (GMT-4)

5.If Winlogon has determined that it can proceed then the *ShutdownInProgress* flag is set. Subsequent shutdown requests from this point will fail in step 3 when it checks this flag.

6.Generates the shutdown audit event if auditing was enabled. Note that shutdown may fail at some point after this even though the audit log is generated.

7.Creates another thread to continue working on the shutdown. At this point the RPC worker thread returns to the caller.

To this point there has been no visible indication on the server being shutdown. If the RPC worker fails for some reason then the client application will get an indication of the failure. If the RPC worker successfully hands the shutdown request to the next thread then the client application will get ERROR_SUCCESS (0) as the return code. *The client application will get no indication of a failure after that point*.

Worker Thread

The RPC worker thread hands control to the ***LogoffThreadProc*** thread. If the delay specified in the shutdown command was 0 then the *ShutdownInProgress* flag is cleared and the *ShutdownHasBegun* is set. In the Winlogon log you will see a line that starts with “Doing immediate shutdown”. Otherwise you will not see any visible indication at this point that a shutdown is occurring.

If the delay specified in the shutdown command was greater than 0 then the countdown shutdown dialog is displayed.

When that dialog completes the *AbortShutdown* flag is checked. This flag would get set as a result of a call to ***AbortSystemShutdown*** (http://msdn2.microsoft.com/en-us/library/aa376630.aspx). If it is set then the shutdown is aborted. Otherwise the *ShutdownInProgress* flag is cleared, the *ShutdownHasBegun* flag is set and if the shutdown was initiated by a system process then we shut the system down immediately at this point with no further clean up.

After these flags are updated the *ExitWindowsInProgress* flag is set and we call the Win32 API ***ExitWindowsEx*** (http://msdn2.microsoft.com/en-us/library/ms893047.aspx). In the Winlogon log you will see a line that starts with “Calling ExitWindowsEx”.

Debugger Output:

dd winlogon!ExitWindowsInProgress l 1

0105fd84 00000001

The Win32 API ***ExitWindowsEx*** makes an RPC call to CSRSS.EXE. CSRSS synchronously sends a WM_QUERYENDSESSION message to all Windows applications.

When an application gets this message it indicates that shutdown can continue and CSRSS then sends the WM_ENDSESSION message. After that the process is terminated. If the application indicates that it cannot be terminated then CSRSS stops processing any further applications and waits for the interactive user to close the application. The ***ExitWindowsEx*** call will fail with error ERROR_OPERATION_ABORTED (0x3E3) and the Winlogon flags are reset so that a new shutdown request can be processed.

An application that prevents shutdown from proceeding in this manner can be seen visual since it will be the foreground window on the desktop. To confirm which application returned did this you will need to live debug CSRSS. A return code of 3 from either winsrv!ConsoleClientShutdown or winsrv!UserClientShutdown will indicate the application did this.

Debugger Output:

0:002> pc

eax=00000000 ebx=7c81a3ab ecx=7ffdb000 edx=75a58ca0 esi=75a58ca0 edi=00164600

eip=75a564de esp=0052fe40 ebp=0052fe68 iopl=0nv up ei pl zr na po nc

cs=001bss=0023ds=0023es=0023fs=003bgs=0000efl=00000246

CSRSRV!CsrShutdownProcesses+7e:

75a564de ff5740 call dword ptr [edi+0x40]{**winsrv!UserClientShutdown** (75a9db1f)}
ds:0023:00164640=75a9db1f

; Step past the call.

0:002> p

eax=00000003 ebx=7c81a3ab ecx=7ffdb000 edx=75a58ca0 esi=75a58ca0 edi=00164600

eip=75a564e1 esp=0052fe4c ebp=0052fe68 iopl=0nv up ei pl zr na po nc

cs=001bss=0023ds=0023es=0023fs=003bgs=0000efl=00000246

CSRSRV!CsrShutdownProcesses+81:

75a564e1 8bf8movedi,eax

; The first parameter is a structure that contains the process ID.

75a564d5 ff75f4pushdword ptr [ebp-0xc]

75a564d8 ff750cpushdword ptr [ebp+0xc]

75a564db ff75f8pushdword ptr [ebp-0x8]

75a564de ff5740calldword ptr [edi+0x40]

; Get the pointer to this structure.

0:002> dd ebp-8 l 1

dd ebp-8 l 1

0052fe60**0018a530**

; The first DWORD is the process ID. The second DWORD is the thread ID.

0:002> dd 0018a530 l 1

dd 0018a530

0018a530**0000066c**

; Break into kernel mode so we can look at all the processes.

0:002> .breakin

.breakin

Break instruction exception - code 80000003 (first chance)

nt!RtlpBreakWithStatusInstruction:

8081db0e ccint3

; Get the process object with that process ID.

kd> !process 0000066c 0

Searching for Process with Cid == 66c

PROCESS ff62a638SessionId: 0Cid: 066cPeb: 7ffdf000ParentCid: 0108

DirBase: 0390d000ObjectTable: e1658e38HandleCount:51.

Image: test.exe

Console (text-based) applications are asynchronously sent a separate CTRL_SHUTDOWN_EVENT notification. This means that ***ExitWindowsEx*** will proceed no matter how the application chooses to handle this notification.

Services.exe (and all Windows services) is a console application which receives this notification from CSRSS. Services.exe registers a control handler ***ScShutdownNotificationRoutine*** which calls ***ScShutdownAllServices*** on shutdown. This function traverses through all Windows services that are not stopped or stopping and that accept the shutdown notification and sends them the SERVICE_CONTROL_SHUTDOWN notification. Each service has 20 seconds by default to shutdown. However a service may

request more time by calling ***SetServiceStatus*** with a wait hint and updated check point. It can do this so long as it continues to respond within the current timeout period. However since Services.exe received this as an asynchronous message from CSRSS.EXE it will not prevent the system from shutting down.

After ***ExitWindowsEx*** returns control to Winlogon the *ExitWindowsInProgress* flag is cleared and ***LogoffThreadProc*** exits.

MainLoop

The state of Winlogon is controlled by the main process thread in a function called ***MainLoop***. As ***LogoffThreadProc*** sets the *ShutdownHasBegun* flag and calls ***ExitWindowsEx*** the ***MainLoop*** function picks up on this change of state and begins executing its shutdown code. Since ***MainLoop*** is responsible for interaction with the user this is the first place where the user will get visible confirmation from Winlogon that the system is shutting down.

When ***MainLoop*** sees that the Winlogon state has changed to shutting down it takes the following actions.

1.Signal the shell that we are shutting down. This causes the Explorer shell to disappear.

2.Checks to see if there are any updates to the user's profile.

3.Send out the logoff notification event.

4.Delete network connections.

5.Play the logoff sound.

6.Play the system exit sound.

7.Creates a ***KillComProcesses*** thread. This calls ***ExitWindowsEx*** and will wait up to 15 minutes for this to complete.

8.Save and unload the user’s profile.

9.Delete RAS connections.

10.Send out the shutdown notification event.

11.Stop Windows file protection.

12.Creates another ***LogoffThreadProc*** thread which again calls ***ExitWindowsEx***.

13.Call the shutdown function in the GINA. This displays the Windows is shutting down dialog.

14.Wait for any remaining system processes to complete. If we are stuck here then we would need to look at System, smss.exe, or csrss.exe. One of those will be stuck in some action.

15.Shut down the system.

Winlogon uses the native API ***NtShutdownSystem*** in step 15. If there is a hang after this point then you will see the main thread in Winlogon stuck in this call and the real problem is likely the result of a device driver in the System process. This call will never return.

shutdown.jpg