Windows Boot Environment

Murali Ravirala
Kernel Platform Architecture Team
Microsoft

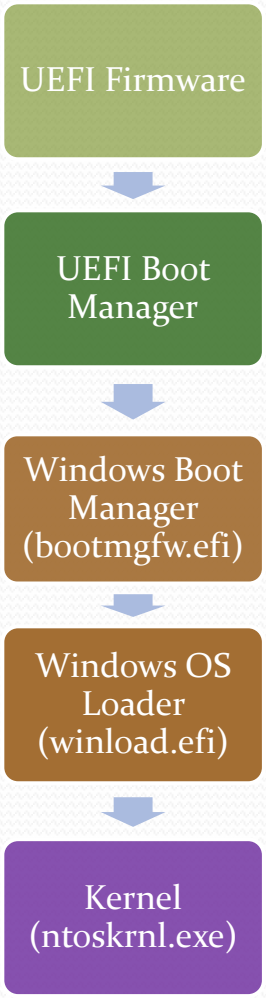
Agenda

- High-level description of Windows boot process
- Roles of different components involved
- Windows UEFI services usage
- Firmware Implementation points

Terms

- Pre-OS space = Boot environment
  - Everything prior to ExitBootServices()
- Boot applications = Windows Boot applications
- Boot manager = Windows Boot manager
- Firmware boot manager = UEFI boot manager
- ESP = EFI system partition
  - Location for Windows boot environment files

Typical Boot flow



Performs CPU and Chipset initialization, load drivers etc.

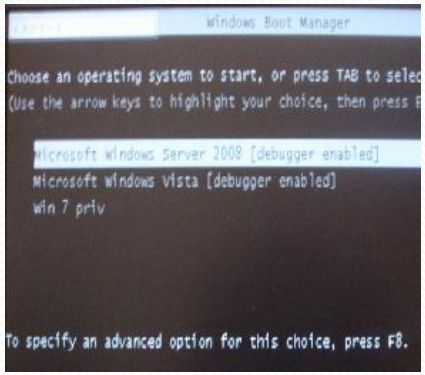
Loads UEFI device drivers (based on NVRAM DriverXxx variable), loads boot application

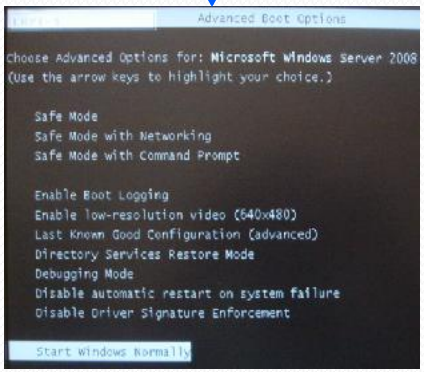
Loads Windows OS loader selected by user

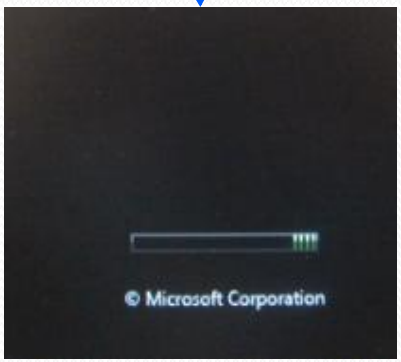
Loads the Windows OS, calls ExitBootServices()

Boot Flow Screens

Windows Boot Manager → Windows OS Loader → OS







Windows Boot manager (Bootmgr)

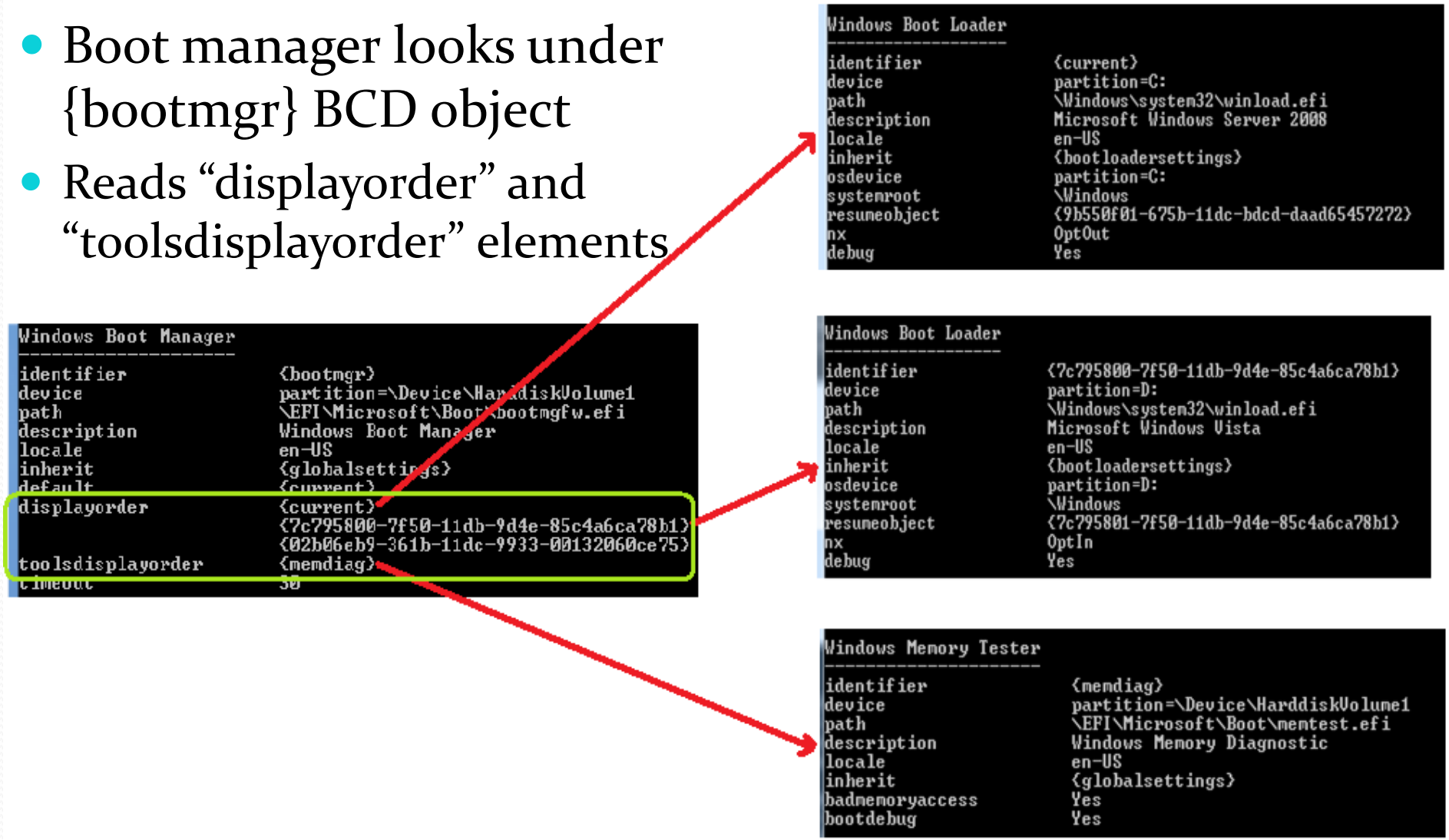
- Loads the Windows boot applications
  - OS loader, Resume loader, memory tester
- Display boot menu and handles user to select
  - Loads the **BCD** store to get a list of boot options
- Locate the OS loader on the device
- Load the appropriate OS loader into memory
- Transfer control to OS loader

Boot Configuration Data (BCD) store

- Stores configuration information required to boot
- Replaces legacy boot.ini (BIOS) and efinvr.exe (on Itanium)
- BCD is a container for BCD objects
  - Each boot application is represented by a BCD object
  - Object are identifed by GUIDs or aliases ({bootmgr}, {default})
  - BCD object is a container of BCD elements
  - Elements contain configuration setting for a boot application
- Located at (ESP)\EFI\Microsoft\Boot\BCD

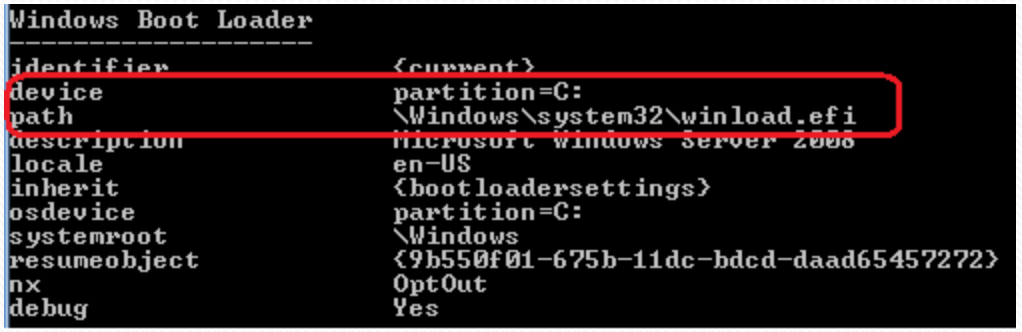
Displaying Boot menu

- Boot manager looks under {bootmgr} BCD object
- Reads “displayorder” and “toolsdisplayorder” elements



Loading OS loader

- Boot entry provides the path to the loader
  - “device” and “path” elements in the BCD store



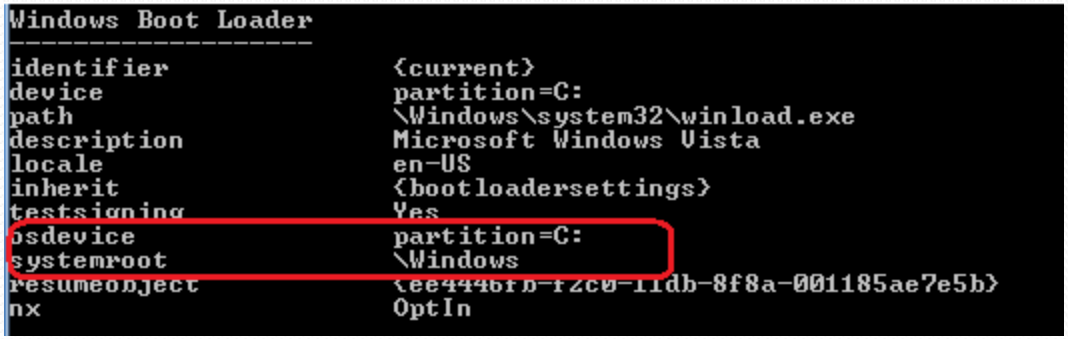
- Loading OS loader into memory
  - Bootmgr understands the NTFS file system
  - Locates file on the disk and reads it into memory

Windows OS Loader

- Load all files needed by the kernel to initialize
- Setup the execution environment for the kernel
- Terminate boot services
- Transfer control to the kernel

Loading OS binaries

- What files are loaded?
  - Kernel + other kernel components required for initialization (ntoskrnl.exe, hal.dll, kdcom, …)
  - All drivers marked as boot start
  - The system hive
- Boot entry provides the path to the OS files
  - BCD entry has “osdevice” and “systemroot” elements



OS Environment Setup

- Setting up OS environment involves:
  - Initializing page tables for kernel
  - Performing architecture specific initialization
  - Setting up runtime services to operate in virtual mode

- Kernel page table initialization
  - OS loader executes in the paging context of the kernel
  - Kernel address space built as files are loaded and mapped
- Architecture-specific initialization
  - Allocate and initialize GDT for kernel
  - Allocate the IDT (initialized by kernel)
  - Allocate kernel stacks

Virtual Addresses for Runtime Services

- OS calls runtime services in virtual mode
- OS loader creates virtual address mappings for all runtime regions
- Informs the firmware of virtual address mappings
  - SetVirtualAddressMap service is used
  - Invoked after calling ExitBootServices()

Execution context: Bootmgr

- Execution context includes:
  - GDT, IDT, stack and page table mappings
- Boot manager executes in firmware context
  - GDT, IDT and stack initialized by firmware
  - Page tables created by firmware
  - Firmware established mapping of physical memory (identity mapping)

Execution context: OS loader

- OS loader executes in an alternate context
  - Building the context for the kernel, so executes in a separate context
- Loader context:
  - GDT, IDT and stack is initialized by OS loader on entry
  - Page tables initialized by OS loader
  - Non-identity mapping of physical memory (VA != PA)
  - Boot services/Runtime services are identity mapped (might change in future)

Resume loader

- Restores OS context from the hibernation file
- Hibernation file (hiber file)
  - Contains state of physical memory and processors
  - Created by kernel before putting system in S4
- All pages that were in use by OS must be restored
  - Runtime memory map must not conflict with OS memory map
  - Otherwise OS or firmware may corrupt each other's data

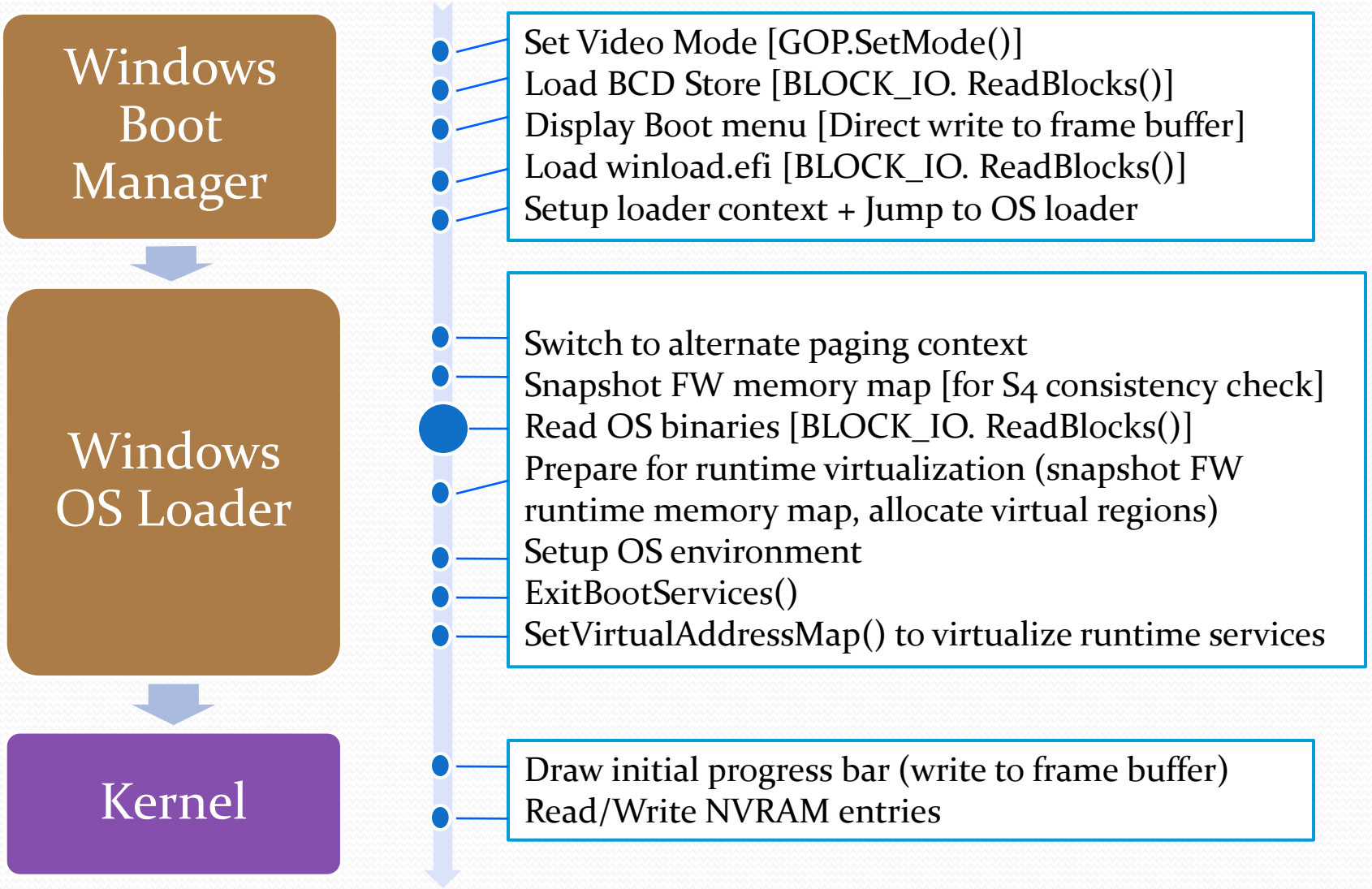
S4 Resume requirements

- Firmware must ensure that physical memory is consistent across S4 sleep transitions
  - OS physical memory during boot must be available to OS during resume
    - Required to restore physical memory across S4 transition
    - Runtime firmware memory must be consistent in size and location between boot and resume
  - Windows will fail to resume from S4 if these conditions are not satisfied

Resume loader

- Firmware memory map is captured by loader and verified by resume application
  - Fail resume if memory map is inconsistent
- Restores virtual address mappings for all runtime services code/data
- Informs the firmware of virtual address mappings
  - SetVirtualAddressMap service is used
  - Invoked after calling ExitBootServices()

Windows Boot Timeline Detail

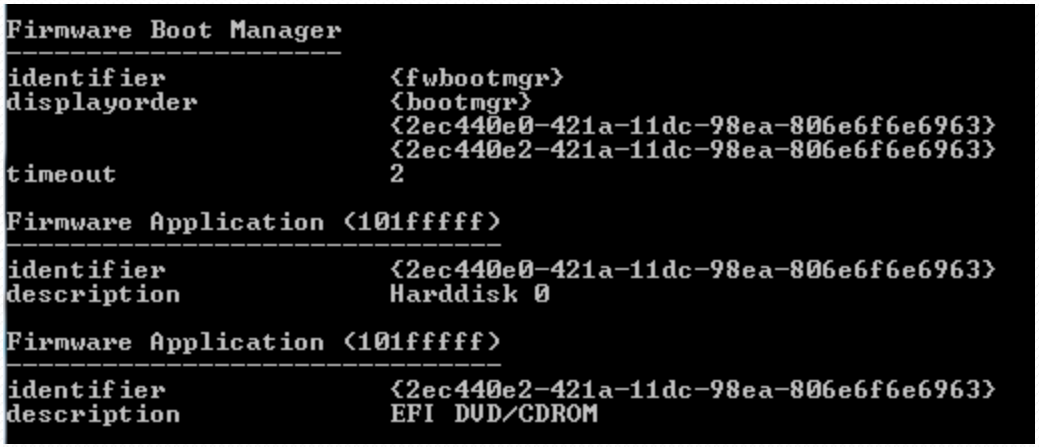


BCD store and NVRAM

- BCD abstracts all the information in the NVRAM
- Provides consistent interfaces to manipulate boot entries
- NVRAM boot entries are cached in the BCD store
- BCD has 1:1 mappings for some UEFI global variables
  - BootOrder → "displayorder"
  - Timeout → "timeout"
  - BootNext → "bootsequence"
  - All variables encapsulated by {fwbootmgr} object

BCD store and NVRAM

- Boot#### is represented by a BCD object
- Any time {fwbootmgr} is manipulated, NVRAM is automatically updated
- Windows only creates one additional NVRAM entry for Windows Boot manager



Windows UEFI Usage

Display Protocol Usage

- Boot environment display
  - Boot applications switch the system into graphics mode
  - Required for localized text to be rendered
  - GOP and UGA protocols are supported
  - UGA is deprecated, so long-term choice should be GOP
  - Windows requires **`1024x768`** or **`800x600`** resolution with **`32`**-bit or **`24`**-bit color (BGR frame format)
  - Fallback to text mode + English if requirements not met
  - Text mode output requires Simple Text Output protocol

Display Protocol Usage

- GOP does not support runtime calls
  - Boot applications will set the video mode
  - Preserve mode and frame buffer after ExitBootServices() and until high-res graphics driver takes over
  - Firmware may not manipulate frame buffer after mode is set by OS Loader
- VGA support still requires INT `10`h support
  - Windows Server 2008 supports headless systems with no VGA

I/O Protocol Usage

- Boot environment input
  - Only keyboard is supported as the input device
  - Simple Text Input protocol is required to read keyboard input
- Boot environment disk I/O
  - Windows uses Block I/O Protocol and Device Path Protocol to boot from a block IO device.
    - Windows boot applications have filesystem support for NTFS, FAT, UDFS, CDFS
  - Block I/O protocol is used extensively by the OS loader and Resume loader.

Other Protocols

- For BitLocker™ support, Windows uses the EFI TCG Protocol
- For PXE boot, Windows uses the EFI PXE Base Code Protocol

Runtime Services Usage

- Minimal amount of runtime services are used at OS runtime.
- Windows uses only UEFI variable services
- Windows philosophy is give preference to OS native drivers followed by ACPI runtime support
  - Only use UEFI runtime services when required (and not supported by other preferred options)
- Windows uses following variable services:
  - GetVariable/SetVariable
  - GetNextVariableName

WHEA Error reporting

- For WHEA error record persistence, Windows uses QueryVariableInfo variable service
- Assumes implementation of UEFI **2.1** hardware error record persistence
- Minimum storage requirement must be guaranteed for error records
- **1**KB on x64; **100**KB on Itanium
- Additional info. available Microsoft's WHDC site:
- **http://www.microsoft.com/whdc/system/pnppwr/whea/default.mspx**

Windows on UEFI Implementation issues

Implementation Issues

- S4 Resume memory map issue
  - Resume failure if runtime memory map is inconsistent
- Runtime services invocation by OS
  - Invoked in the context of the OS with interrupts on and paging enabled.
- Runtime services virtualization range
  - Services may be virtualized in high virtual address region.
  - Do not assume addresses are below certain value (< 4GB)

Implementation Issues

- Runtime services execution time
  - Bugcheck `0x101`(CLOCK_WATCHDOG_TIMEOUT) possible if runtime services uses SMM and takes too long
  - May cause secondary processor to miss some clock interrupts (leads system to believe processor is hung)
- Simultaneous runtime services invocation possible
  - OS will invoke one runtime service at a time normally
  - On NMI or MCA exceptions, OS may invoke runtime services to persist error record
  - Other runtime operations may have been in flight at that time

Implementation Issues

- Interrupt status prior to ExitBootServices()
  - Interrupts are turned OFF while boot application runs
  - Interrupts are turned ON before making a firmware call
  - Do not assume that interrupts are always ON.

IA64 Differences

- Virtual address mappings for runtime services created during OS (HAL) initialization
  - Fallback to using physical mode if runtime services virtual mappings cannot be setup
- Does not use alternate paging contexts in the boot environment

Summary

- Different components involved in Windows boot and their timeline
- How UEFI Protocols and services are used by Windows
- Implementation issues

Questions???

THE END!