Chapter 1

WDF KMDF Driver Development Introduction

Common OS architecture concepts

- Overall Architecture
  - Monolithic
  - Layered
  - Microkernel (Client-Server)
- User Mode vs. Kernel Mode
- Certain CPU instructions not accessible in User Mode
  - e.g. Halting the processor
  - e.g. Accessing hardware registers
- Virtual memory with certain ranges not accessible in User Mode
  - e.g. kernel address space
  - e.g. other processes' address spaces (process isolation)

History and design goals

- Designed by David Cutler (former DEC / VAX / VMS)
- Portability
  - Available on different platforms (x86, DEC-Alpha, PowerPC, MIPS, IA64)
  - Hardware Abstraction Layer (HAL)
    New "plug in" extension capability for core system components (e.g. interrupt controller) in Windows 8
  - WDM Bus drivers (up to NT4 part of the HAL)
- Microkernel concepts
  - Not a Microkernel operating system
  - Similar modularity in kernel design
  - Large OS components in user mode (Subsystems)

Windows components in user mode and kernel mode

- User mode
  - Applications
  - Subsystems
  - Services
- Interface from user mode to kernel mode
  - Windows Native API
  - Not documented
  - Should not be used in shipping products
- Kernel mode
  - WDM Drivers
  - Executive

Windows Executive

- Kernel-Mode managers and libraries
  - Kernel
  - Io Manager
  - Plug’n’Play manager
  - Power manager
  - Memory manager
  - Hardware abstraction layer
- Executable files
  - Windows\system32\ntoskrnl.exe
  - Windows\system32\Hal.dll

WDM driver model

- Direct interaction with executive components
- Using a subset of the function calls exported by ntoskrnl.exe
- Component indicated by prefix letters
  - IoXxx IO manager
  - MmXxx memory manager
  - PoXxx power manager
  - ObXxx object manager
  - KeXxx kernel
  - RtlXxx runtime library
- Rich set of features

WDM driver architecture fundamentals

- File based
  - Drivers are accessed as if they were files
- Packet based
  - Each I/O operation is described by an IO-Request-Packets (IRP)
  - PnP, Power and other events are also described by IRPs
- Layered
  - Usually more than one driver layered above one hardware
  - IRPs can be handed through these layers in a device stack of layered drivers
- Asynchronous

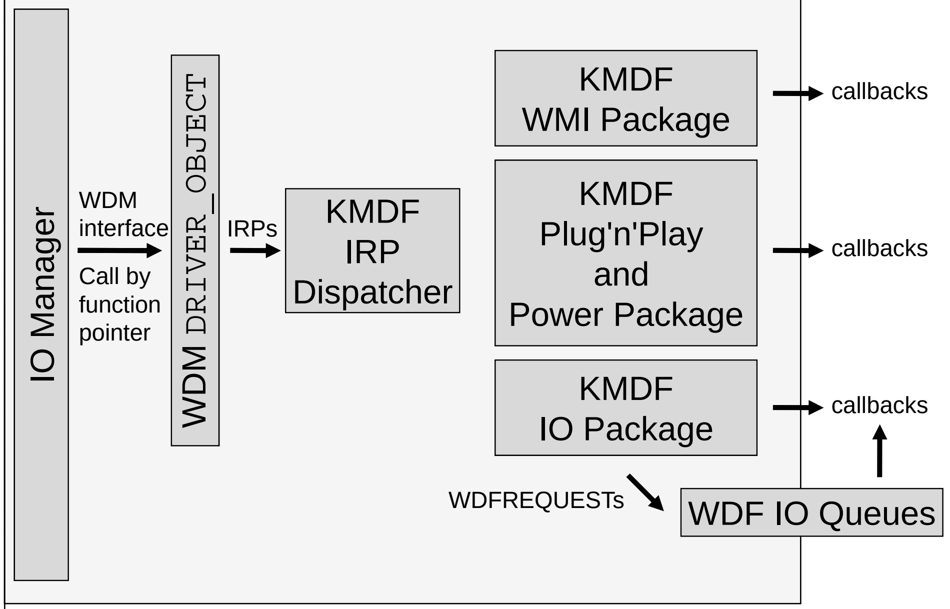
Windows driver types

- Kernel mode WDM Plug'n'Play drivers
  - WDM device function drivers (incl. WDF-KMDF)
  - WDM device filter drivers (incl. WDF-KMDF)
  - WDM software drivers (incl. WDF-KMDF)
  - System supplied port drivers
- Kernel mode Miniport drivers
- Kernel mode legacy Non-Plug’n’Play drivers
- Kernel mode File System Drivers (FSD)
- User mode drivers
  - WDF-UMDF drivers
  - Printer driver components
  - Display driver components

Windows Driver Foundation (WDF)

- WDF is a library for simplifying WDM driver development
- WDF supports user mode drivers and kernel mode drivers
- WDF provides a consistent object model for user mode and kernel mode drivers
- WDF object model supports
  - Handling, queuing and cancellation of IO requests
  - Full Plug'n'Play state machine with default implementation
  - Full Power Management state machine with default implementation
  - Many more infrastructure tasks

WDF KMDF Overview



IO Manager

WDM interface

Call by function pointer

WDM DRIVER_OBJECT

IRPs

KMDF IRP Dispatcher

KMDF WMI Package

KMDF Plug'n'Play and Power Package

KMDF IO Package

callbacks

callbacks

callbacks

WDFREQUESTs

WDF IO Queues

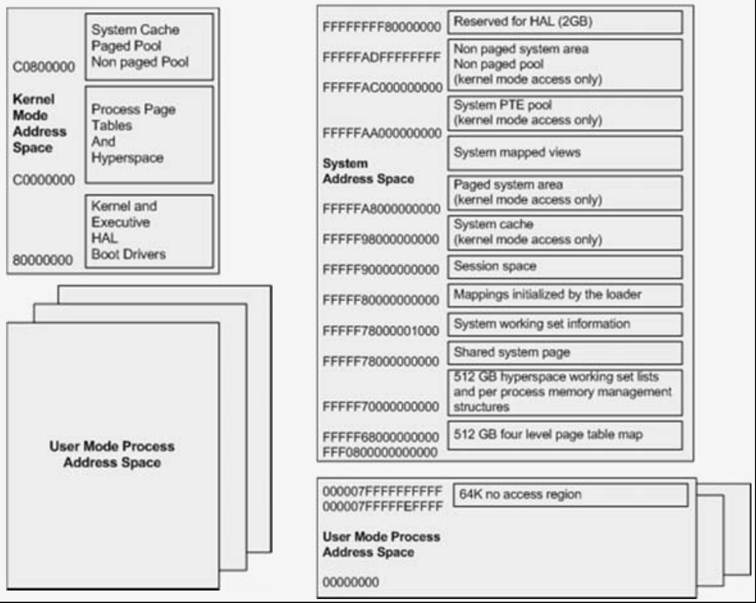
Process model

- Process isolation
- Memory protection
  - No access to operating system memory
  - No access to other processes' address spaces
  - Memory cleanup after process termination
- Process specific object handles
  - Handle cleanup after process termination

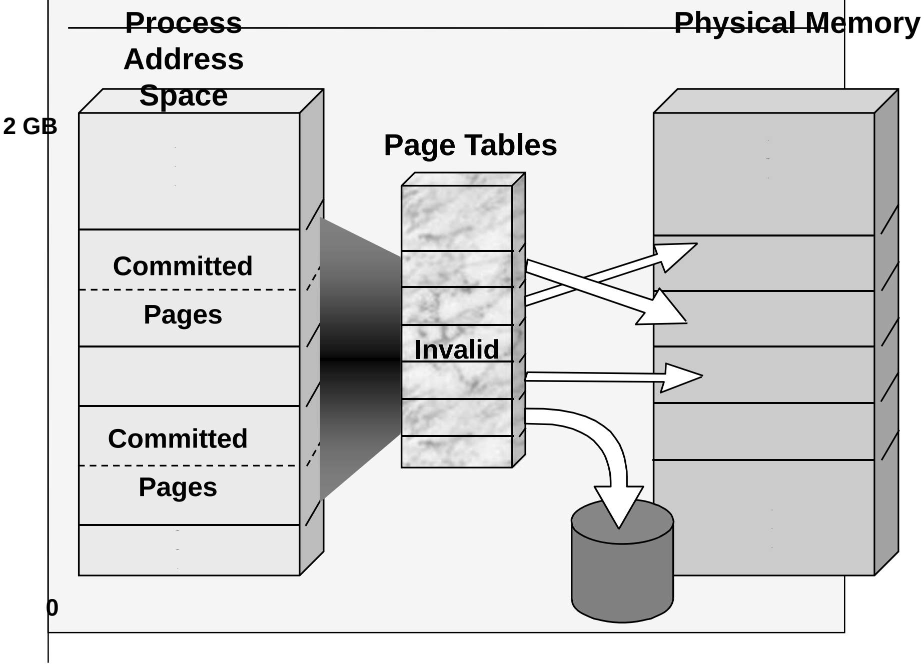
Virtual Memory

- Extension of physical RAM by paging file
- Protection
  - Protection of operating system memory
  - Process isolation
- One single privileged address space for the operating system
  - Applications cannot corrupt the OS
- Separate isolated address space for each user mode application
  - Applications cannot corrupt each other

X86 and x64 address space



Virtual Memory Fundamentals



Object based architecture

- Object based resource access and usage
  - Polymorphic CreateXxx(…) functions returning handles
  - Interprocess resource usage through names in Object Manager’s namespace
- Tool for viewing Object Manager's namespace: winobj.exe (download from Sysinternals)
- Polymorphic synchronization through dispatcher objects

Windows trap handling

- Hardware independent trap handling model

- Distinguishing four different traps
  - Exceptions
  - System calls
  - Hardware interrupts
  - Virtual memory exceptions

- Exceptions raised by violating thread
  - Can be handled by the application itself using SEH (Structured Exception Handling)
  - Default handler shows dialog box and calls `ExitProcess()`