Lab 4: Kmdf Power

# Overview

KMDF framework exposes a different model than WDM. There is no 1-1 map between IRP\_MJ functions and EvtXxxx callbacks.

For example

In this lab we'll learn to work with WinDBG to inspect various aspect of Kmdf power management

## Setup

* We will start with lab 3.

# Debug SmplDeviceEvtD0Entry and SmplDeviceEvtD0Exit

## Step 1

* On target machine, disable Sample Device
* Hookup WinDBG to the target machine
* Reload all module with: .reload
* Ask WinDBG to break when SmplDevice loads: sxe ld smpldevice
* g
* On target machine, enable Sample Device
* Verify: Debugger will break when smpldevice loads
* Using the 'x' command, find the D0 routines: x smpldevice!\*d0\*
* Put breakpoints (bu) on the D0 enter and exit
* g

## Step 2: Break on start

* Break on SmplDeviceEvtD0Entry
* In debugger, loads the wdfkd extension
* See dump about the device with !wdfdevice extension command.
* Find out the WDM object behind the WDF object
* Dump it's content using 'dt'. The type is nt!\_DEVICE\_OBJECT

## Step 3: Hibernate

* Enable hibernation. By default VmWare machine do not support hibernation. Using the powercfg.exe utility enable hibernation.
* Using shutdown.exe hibernate
* Verify you break on SmplDeviceEvtD0Exit
* Verity the target state.
* Try to find out which IRP\_MJ the IoManager has sent to the Framework.

## Step 4: start from hibernate

* Start the virtual machine
* You'll have a break when SmplDevice loads. You'll have to reload all symbols again
* g until you break on SmplDeviceEvtD0Entry
* Note that wdf appears on the stack twice
* Find IoCallDriver on the stack.
* Dump the call stack with 'kv' command.
* The second parameter is IRP. Using the !irp extension, dump the content.
* Note that we are 2nd on the irp stack. Above us there is the PnpManager, which is also a device driver.
* Check out what IRP\_MJ is 16, and what IRP\_MN is 2

## Step 5: Sleep

VmWare doesn't support the intermediate power states s1 through s3. An experimental feature exists, which requires manual modification to the \*.vmx file

* Open machine.vmx file
* Repeat steps 4 and 5 with sleep

# Capabilities

In this section we'll set the device capabilities

## Step 1

* Open the SmplDevice at the device manager. On the details tab see the Power Data.
* See the power data for other devices
* Check out power data for other devices

## Step 2: Change default capabilities

* Within WdfDeviceCreate, use the functions WdfDeviceSetPnpCapabilities and make the default non removable.
* Enable D1 using WdfDeviceSetPowerCapabilities.
* Using same method, enable wake from D1, and set DeviceWake to be D1.
* Using same method, set the mapping: S0 -> D0; S1 -> D1; S2 -> D2; S3 -> D3; S4 (hibernate) -> D3; S5 -> D3
* Before creating the default, using WdfDeviceInitSetPowerInRush set current in rush.

# Idle control

In Windows, we can tell power manager to perform idle detection and control. The power manager will detect Idle timeout, and will send IRP\_MJ\_POWER/IRP\_MN\_SET\_POWER when device is idle.

* Using WdfDeviceAssignS0IdleSettings set a timeout of 2 minutes
* Do this by initialize structure WDF\_DEVICE\_POWER\_POLICY\_IDLE\_SETTINGS with IdleCanWakeFromS0, and set IdleTimeout to 120\*1000;
* Put a breakpoint on D0 Exit and verify function is called.
* Using the function WdfDeviceInitSetPowerPolicyEventCallbacks register EvtDeviceArmWakeFromS0 method. Implement SmplDeviceEvtDeviceArmWakeFromS0.
* Put a breakpoint on EvtDeviceArmWakeFromS0. Find in the stack the IRP and verify it is IRP\_MN\_WAIT\_WAKE

# PoFX

PoFX is an alternative to the device level power state. Comment out all steps of sections 1.0 to 1.4

## Step 1:

* Create a new file PoFX.c
* Within PoFX.c Create a new function:  
  VOID SmplPoFxSingleComponentInitialize(\_In\_ WDFDEVICE Device);
* Declare this function in driver.h
* Declare these 3 callbacks in driver.h:

//

// PoFx Callbacks

//

PO\_FX\_COMPONENT\_IDLE\_STATE\_CALLBACK SmplPoFxComponentIdleStateCallback;

PO\_FX\_COMPONENT\_ACTIVE\_CONDITION\_CALLBACK SmplPoFxComponentActiveConditionCallback;

PO\_FX\_COMPONENT\_IDLE\_CONDITION\_CALLBACK SmplPoFxComponentIdleConditionCallback;

* Declare the following structures:

WDF\_POWER\_FRAMEWORK\_SETTINGS Settings;

PO\_FX\_COMPONENT Component;

PO\_FX\_COMPONENT\_IDLE\_STATE IdleStates[FSTATE\_COUNT];

In driver.h define the following:

#define FSTATE\_COUNT 4

#define DEEPEST\_FSTATE\_LATENCY\_IN\_MS 800

#define DEEPEST\_FSTATE\_RESIDENCY\_IN\_SEC 12

## Step 2

* On device.h you need to declare a queue within the DEVICE\_CONTEXT: WDFQUEUE Queue
* Also on device context, declare PoHandle

## Step 3

* Zero (RtlZeroMemory) the Component structure and the IdleStates structure
* Set the following IdleStates
* In Queue.c, store the queue returned from WdfIoQueueCreate

//

// F0

//

IdleStates[0].TransitionLatency = 0;

IdleStates[0].ResidencyRequirement = 0;

IdleStates[0].NominalPower = 0;

//

// F1

//

IdleStates[1].TransitionLatency = WDF\_ABS\_TIMEOUT\_IN\_MS(200);

IdleStates[1].ResidencyRequirement = WDF\_ABS\_TIMEOUT\_IN\_SEC(3);

IdleStates[1].NominalPower = 0;

//

// F2

//

IdleStates[2].TransitionLatency = WDF\_ABS\_TIMEOUT\_IN\_MS(400);

IdleStates[2].ResidencyRequirement = WDF\_ABS\_TIMEOUT\_IN\_SEC(6);

IdleStates[2].NominalPower = 0;

//

// F3

//

IdleStates[3].TransitionLatency = WDF\_ABS\_TI

MEOUT\_IN\_MS(DEEPEST\_FSTATE\_LATENCY\_IN\_MS);

IdleStates[3].ResidencyRequirement = WDF\_ABS\_TIMEOUT\_IN\_SEC(DEEPEST\_FSTATE\_RESIDENCY\_IN\_SEC);

IdleStates[3].NominalPower = 0;

* Set component 0:

//

// Component 0 (the only component)

//

Component.IdleStateCount = FSTATE\_COUNT;

Component.IdleStates = IdleStates;

WDF\_POWER\_FRAMEWORK\_SETTINGS\_INIT(&Settings);

Settings.EvtDeviceWdmPostPoFxRegisterDevice = SmplDeviceEvtWdmPostPoFxRegisterDevice;

Settings.Component = &Component;

Settings.ComponentActiveConditionCallback = SmplPoFxComponentActiveConditionCallback;

Settings.ComponentIdleConditionCallback = SmplPoFxComponentIdleConditionCallback;

Settings.ComponentIdleStateCallback = SmplPoFxComponentIdleStateCallback;

Settings.PoFxDeviceContext = (PVOID) Device;

## Step 4

* Implement SmplPoFxComponentIdleStateCallback according to signature.
* Within SmplPoFxComponentIdleStateCallback call PoFxCompleteIdleState, and DebugPrint.

## Step 5

* Implement SmplPoFxComponentActiveConditionCallback. Within this function call WdfioQueueStart to enable IRP's. The queue should be saved in the device context.

## Step6

* Implement SmplPoFxComponentIdleConditionCallback. Within this function call WdfIoQueueStop

## Step 7

* Within Queue.C implement SmplQueueEvtStatePoFxStopComplete
* PoFxCompleteIdleCondition. The PoHandle is in the device context