Windows Internals

Alon Fliess
Chief Architect, CodeValue (http://www.codevalue.net)
alonf@codevalue.net
http://blogs.microsoft.co.il/blogs/alon
About Me

• Alon Fliess:
  – Chief Software Architect & Co-Founder at CodeValue Ltd.
  – More than 25 years of hands-on experience
  – Microsoft Regional Director & Microsoft MVP
  – Active member of several Patterns & Practices councils
  – Renowned speaker at both international and domestic events
About CodeValue

• A leading software company
• ~150 employees: more than 130 technology experts
• Provides high quality software development solutions
  – Turn-Key projects
  – Software development and consultation
  – Tailor-made courses and training
• Fields of expertise include:
  – Desktop & LOB applications
  – Cloud Computing
  – Advanced Mobile & Web Technologies
  – User Experience (UX) & User Interface (UI)
  – Application Lifecycle Management (ALM) and DevOps
  – Embedded & IoT
About OzCode

• An innovative debugging extension for Visual-Studio
• Simplify & visualize complex statements
• Compare objects and collections
• Search and filter collections
• Focus on relevant data
• http://oz-code.com
Course Details

• Objectives
  – Understand Windows features and architecture
  – Uncover the internal algorithms used by Windows relevant to developers
  – Enhance the ability to design and implement optimized software for the Windows platform

• Target audience
  – Developers & IT Pros
Resources

• Books
  – Windows Internals / David Solomon & Mark Russinovich / 4th, 5th, 6th,... editions
  – Windows Sysinternals Administrator’s Reference / Mark Russinovich & Aaron Margosis
  – Programming the Windows Driver Model / Walter Oney / 2nd edition
  – Developing Drivers with the Windows Driver Foundation / Orwick & Smith

• Web
  – www.sysinternals.com
  – www.osr.com
  – www.microsoft.com/whdc
  – http://nirsoft.net/
  – http://msdn.microsoft.com
  – http://www.alex-ionescu.com/
Course Contents

• 1: System Architecture
• 2: Kernel Mechanisms
• 3: Management Mechanisms
• 4: Processes, Threads & Jobs
• 5: Memory Management
• 6: Security
• The I/O Subsystem
• Networking
• 7: Introduction to Windows Universal Apps
• A: Introduction to COM
• B: Introduction to WinDbg
Module 1

SYSTEM ARCHITECTURE
Agenda

- Windows NT History
- Tools
- Basic Concepts
- Windows NT Design Goals
- System Architecture
- Summary
Windows NT History

- Windows NT 3.1 (July 1993)
- Windows NT 3.5 (September 1994)
- Windows NT 3.51 (May 1995)
- Windows NT 4.0 (July 1996)
- Windows 2000 (December 1999)
- Windows XP (August 2001)
- Windows Server 2003 (March 2003)
- Windows Vista (January 2007)
- Windows Server 2008 (February 2008)
- Windows 7 & 2008 R2 (October 2009)
- Windows 8.1 & Server 2012 R2
- Windows 10 & Server 2016
Tools – Where are they Coming From?
WINDOWS BASED TOOLS
Windows Built-In Tools
Task Manager

![Task Manager Window]
Windows Built-In Tools
Resource Monitor
Windows Built-In Tools

Performance Monitor
Windows Built-In Tools
Windows Troubleshooting Platform

Troubleshoot computer problems
Click on a task to automatically troubleshoot and fix common computer problems. To view more troubleshooters, click on a category or use the Search box.

- **Programs**
  - Run programs made for previous versions of Windows

- **Hardware and Sound**
  - Configure a device
  - Use a printer
  - Troubleshoot audio recording
  - Troubleshoot audio playback

- **Network and Internet**
  - Connect to the Internet
  - Access shared files and folders on other computers

- **System and Security**
  - Fix problems with Windows Update
  - Run maintenance tasks
  - Improve power usage

See also
- Security and Maintenance
- Help and Support
- Recovery
Windows Admin Tools

- Control Panel
- MMC
- System Configuration
- System Information
- RegEdit
- ...

[Image: System Configuration dialog box]
SYSTEM INTERNALS TOOLS
System Internals

- The Sysinternals web site was created in 1996 by Mark Russinovich and Bryce Cogswell
- Microsoft acquired Sysinternals in July, 2006
- Targets both IT pros and developers
- If you have a question about a tool or how to use it, please visit the [Sysinternals Forum](#)
System Internals Tools

Process Explorer
System Internals Tools

Process Monitor
Additional Important SysInternals Tools

- **AccessEnum** - shows you who has what access to directories, files and Registry keys
- **Autoruns** – See who automatically loads a dll or auto starts an exe
- **DebugView** – Intercept debug output calls, very useful for debug sessions
- **PipeList** – Displays the named-pipes with their properties in the system
- **ProcDump** – Very useful process dump tool, dump process memory on various occasions
- **SysMon** - Monitor and log system activity to the Windows event log
- **PsTools** - CLI utilities for handling processes running on local or remote computers, rebooting computers, dumping event logs, and more.
- **RAMMap** - Physical memory usage analysis utility
- **TCPView** - Active socket command-line viewer
- **VMMMap** - A process virtual and physical memory analysis utility
- **WinObj** – The Object Manager namespace viewer
Windows SDK

DEBUGGING TOOLS FOR WINDOWS
Overview

• Debugging Tools for Windows is a collection of debugging and diagnostic tools. It contains:
  – Four Windows debuggers (WinDbg, KD, CDB and NTSD)
  – Logger and Log Viewer (Logger.exe, Logviewer.exe)
  – ADPlus (Autodump+, Adplus.vbs) - Automatically create memory dump files and log files
  – GFLogs – Used to control System debug settings
  – UMDH - User-Mode Dump Heap utility (Umdh.exe)
  – Symbol & Source Server files
  – Remote Debugging tools
  – The debugger.chm help file
• In the next chapter we will deep dive into the debuggers
Global Flags

- GFlags (the Global Flags Editor), gflags.exe, enables and disables advanced debugging, diagnostic, and troubleshooting features
WINDOWS SDK AND WINDOWS DRIVER KIT TOOLS
SDK/WDK Tools Overview

• The Windows SDK & WDK contain:
  – Tools
  – Code samples
  – Documentation
  – C/C++/.NET compilers, headers, and libraries

• Enable developers create applications & device drivers that run on the Windows platform
VISUAL STUDIO TOOLS
Visual Studio Tools Overview

• With Visual Studio comes a large set of tools for building, debugging (local & remote), trace and test applications

• Some of the tools are accessible from within Visual Studio
  – Others are command-line or external UI based tools

• Partial tool list:
  – Spy++
  – InteliTrace
  – VS Profiler
  – C/C++ and .NET static Analysis
  – Error Lookup
APPLICATION COMPATIBILITY TOOLKIT
ADK – Windows 10 & Deployment Kit

• Tools for:
  – Customize Windows images for large-scale deployment
    • Windows PE (Windows Preinstallation Environment)
  – Test the quality and performance of the system & applications
  – Contain the ACT – Application Compatibility Toolkit
    • Creating an inventory for your organization, including installed applications, computers, and devices
    • Collecting compatibility data
    • Creating mitigation packages to fix the compatibility issues
Select the features you want to install

Click a feature name for more information.

- Application Compatibility Toolkit (ACT)
- Deployment Tools
- Windows Preinstallation Environment (Windows PE)
- Imaging And Configuration Designer (ICD)
- User State Migration Tool (USMT)
- Volume Activation Management Tool (VAMT)
- Windows Performance Toolkit
- Windows Assessment Toolkit
- Windows Assessment Services - Client
- Microsoft SQL Server 2012 Express

Application Compatibility Toolkit (ACT)

Size: 26.1 MB
Tools to evaluate and mitigate application compatibility issues before deploying a new version of Windows.

ACT requires access to a database. The database must be SQL Server 2005 (or Express Edition) or later. You can install SQL Server or use an existing installation.

Estimated disk space required: 6.5 GB
Disk space available: 291.7 GB
Windows Assessment Console

Choose a job, or a single assessment, and run it to learn more about your computer.

- Battery life during connected standby
- Battery life during full video playback
- Battery life during idle periods
- Windows Store apps performance
- Browsing experience
- Media experience
- Startup and shutdown experience
- Hardware performance
- Run Individual Assessments

Select a single assessment.

Battery life during connected standby

Measures battery life and reports energy efficiency issues during connected standby periods. For connected standby-capable devices only.

- Job details
  - Version: 10.0.10240.10584
  - Estimated run time: Varies
  - Supported architecture: x86, x64, ARM
  - Earliest supported version(s) of Windows: Windows 8, Windows RT
  - Run mode: Will run silently, and may not display any indication of assessment activity.

This job will run the following assessments:

1. Connected Standby energy efficiency
   - This workload enters Connected Standby and measures the energy efficiency of the computer
     - Approximate run time: Varies
     - Version: 10.0.10240.16384
     - Supported architecture: x86, x64, ARM
     - Author: Microsoft Corporation
     - More information

Configure  Run
Compatibility Administrator

This is an open working database. Fixes can be created for this database.

More information about Compatibility Administrator:
Download the latest version of the Application Compatibility Toolkit
3\textsuperscript{RD} PARTY TOOLS
3rd Party Tools Overview

- There are plenty of good 3rd party tools
  - Some of them cost money
    - Intel VTune profiler
    - SCI Tools – Understand
    - Parasoft C/C++ Quality Solution
    - PVS-Studio
    - .Net Tools such as JetBrains Resharper & CodeValue OzCode
  - Some of them are free
    - NirSoft tools
    - Dependency Walker
    - MiniDumpWizard & MiniDumpView
Dependency Walker

• This is an old utility that still works:
Tool Summary

• In this section we’ve been introduced to the Windows Developer Debugging Toolbox
• Tools come from many places, most of them are free
• Pick the set of tools that you are most comfortable with
  – Study it, know the pros and cons of each tool
• Understand the System mechanism that the tool analyzes
Basic Concepts

- Abstraction Hierarchy
- User Mode vs. Kernel Mode
- The Windows API
- Services and routines
- Processes, threads & Jobs
- Virtual Memory
- Objects and handles
- The Registry
The Computer

• No, it is not Computer Science 101
  – However there are some concepts that need dusting off
  – In order to fix the machine, we need to understand it!
• From the software point of view, the Computer is built from a collection of resources
  – The CPU is a resource responsible for executing code
  – The RAM is a resource that holds volatile information and lets the CPU execute code and consume data
  – I/O (Disk, Network) – is the (slow) resource that can provide or consume data
The CPU

• The CPU is the resource that can execute code
• Modern Computers usually have one or more CPUs that contain one or more cores
  – A core is a CPU inside the CPU chip
• The CPU can read instructions from memory (RAM) and execute them one-by-one
• The CPU fetches data from memory
• The CPU stores data into memory
The Random Access Memory

• Volatile memory stores information that came from I/O devices or from the CPU
• The CPU can work only against RAM (Physical Memory)
• The Memory is a resource, usually broken into pages
• I/O devices can work directly with the Memory
• I/O devices use interrupts to alert the CPU
• Code Instructions can also alert the CPU
Address Space, Bus & Bytes

- To fetch data from memory the CPU uses a bus
- The CPU puts the data address on the address bus
- The possible Address Space is derived from the number of lines in the address bus
  - To be more accurate, the size of the address register

- On 32 bit machines we have maximum 4GB ram
- On 64 bit machines we have maximum of 18 EB = 18,446,744,073,709,551,616
  - 1 EB = $10^{18}$ bytes = 1 billion gigabytes = 1 million terabytes

- Address is the location of the data
  - The data does not necessarily exist there!
Abstraction Layers

• People think and speak in abstractions
  – There are just too many tiny details
• When we say CPU, we actually mean the many components and busses and gates inside the CPU
• We say instruction, but a CPU instruction is built from many other tiny instructions that control the many components inside the CPU
• So we will continue to talk using abstractions and concepts, but...
• When a problem arises, sometime the abstraction breaks. We need to get to the real thing!
Assembly Language

• A collection of instructions that serve as the building blocks for a program for the CPU

• Most CPU Instructions are read/execute/write
  – Other instructions are: control, compute, etc.

• Each command can be executed in a certain CPU mode:
  – If the CPU reads a command that is forbidden to execute in the current CPU mode, an exception occurs
    • Exception is a software interrupt
"An operating system (OS) is software, consisting of programs and data, that runs on computers and manages computer hardware resources and provides common services for efficient execution of various application software."

-Wikipedia
User Mode vs. Kernel Mode

• Process access modes
  • User mode
    – Allow access to non-operating system code & data only
    – No access to the hardware
    – Protects user applications from crashing the system
  • Kernel mode
    – Privileged mode for use by the kernel and device drivers only
    – Allows access to all system resources
    – Can potentially crash the system
The Windows API

• Application Programming Interface for all Windows versions
• Documented in the Windows SDK (formerly Platform SDK)
• Each version implements a different subset of the API
• Now collectively called the Windows API
  — Previously referred to as the “Win32 API”
  — 64 bit windows introduced Win64
• Contains functions in the following areas
  — Base services, user interface services, component services, graphics and multimedia, messaging and collaboration, networking
• API style
  — Flat C functions
  — COM (Component Object Model)
The .NET Framework & Tools

• An higher abstraction layer
  – On top of Win32 API
• Object Oriented APIs
• Management:
  – Hosting, Lifetime, threading, memory, security, configuration, Metadata
• Support many programming language
• Call native code and APIs using .NET Interop
UWP & WinRT

- For Windows Store Application
- Windows 8/8.1 ➔ WinRT ➔ Windows Runtime
- Windows 10 ➔ UWP ➔ Universal Windows Platform
  - A superset of WinRT
- A set of unmanaged OO APIs
  - Hosting, Lifetime, threading, memory, security, configuration, Metadata
- Base on COM
  - Native & Managed language support
Processes

• Process
  – A set of resources used to execute a program

• A process consists of
  – A private virtual address space in which memory can be allocated and used
  – An executable program, referring to an image file on disk which contains the initial code and data to be executed
  – A table of handles to various objects, such as files, events, threads, and others
  – A security context, called an access token, used for security checks when accessing shared resources
  – One or more threads that execute code
Processes in Task Manager
Threads

• Thread
  – Entity that is scheduled by the kernel to executes code

• A thread contains
  – The state of CPU registers
  – Current access mode (user mode or kernel mode)
  – Two stacks, one in user mode and one in kernel mode
  – A private storage area, called Thread Local Storage (TLS)
  – Optional security token
  – Optional message queue and Windows the thread creates
  – A priority, used in thread scheduling
  – A state: running, ready, waiting
Jobs

- A job object allows groups of processes to be managed as a unit
- Job Limits:
  - Memory, execution time, number of active processes, CPU affinity, priority, UI restriction, and more
- Job Notifications (process has exited)
- Resource Accounting (I/O counters)
- Nested Jobs (Windows 8/8.1/10)
The Windows Loader

• The Windows loader routines reside in ntdll.dll
  – In WinDbg type: x ntdll!Ldr*

• Whenever there is a need to map an exe or Dll file to the process, the loader does it
  – The public API to handle images from code are in the ImageHlp.dll

• In order to load a Dll or Executable into runnable program we need a protocol
  – A way to extract information from a file and create from it a live program
The Portable Executable is the format for all DLL, EXE, SYS, OCX, and even PDB files.

The PE itself is a container for other format named Common Object File Format (originated in the Unix O/S).

The data is saved in headers followed by sections:
- Headers contain metadata
- Sections contain data

The ImageHlp.dll provides APIs to handle and manipulate PE files.
Virtual Memory

• Each process “sees” a flat linear memory
• Internally, virtual memory may be mapped to physical memory, but may also (temporarily) stored on disk
• Processes access memory regardless of where it actually exists
  – The memory manager handles mapping of virtual to physical pages
  – Processes cannot (and need not) know the actual physical address of a given address in virtual memory
Virtual Memory Mapping

Process A

Virtual memory

Physical memory

Virtual memory

Process B

Disk
Virtual Memory Layout

x86 (32 bit)

- 2 GB User Process Space
- 2 GB System Space

x64 ≤ Windows 7 (44 bit)

- 8 TB User Process Space
- 8 TB System Space

X64 ≥ Windows 8.1 (48 bit)

- 128 TB User Process Space
- 128 TB System Space

Windows Internals 60
The (Call) Stack

• The stack is used to:
  – Store local variables
  – Handle arguments passing to a function
  – Keeping the return address
  – Housekeeping for exception handling (x86)

• Each thread has a stack with predefined size of 1 MB
  – This is the reserved size of the stack
  – The default size for the reserved and initially committed is specified in the PE header (linker: /STACK:reserve,commit)
  – `CreateThread` can allocate bigger stack size
The Call Stack – Deep Dive

- The stack expands and shrinks during the thread execution
  - The stack is built according to the function calling convention and the exception handling mechanism
  - There are 6 common calling conventions:
    - __stdcall
    - __cdecl
    - this call
    - __fastcall
    - __naked
    - X64 calling convention
Example of x86 __cdecl

`Void Foo(int a, int b, int c)`

```c
{
    int x, y, z;
}
```

<table>
<thead>
<tr>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>16( + ebp)</td>
<td>third function argument (c)</td>
</tr>
<tr>
<td>12 + ebp)</td>
<td>second function argument (b)</td>
</tr>
<tr>
<td>8 + ebp)</td>
<td>first function argument (a)</td>
</tr>
<tr>
<td>4 + ebp)</td>
<td>old EIP (the function's &quot;return address&quot;)</td>
</tr>
<tr>
<td>0( + ebp)</td>
<td>old EBP (previous function's base pointer)</td>
</tr>
<tr>
<td>-4(+ ebp)</td>
<td>first local variable (x)</td>
</tr>
<tr>
<td>-8( + ebp)</td>
<td>second local variable (y)</td>
</tr>
<tr>
<td>-12( + ebp)</td>
<td>third local variable (z)</td>
</tr>
</tbody>
</table>
The x64 Calling Convention

Passing parameters

- All arguments are right justified in registers
- All stack parameters are 8 bytes aligned
- Any parameter that's not 1, 2, 4, or 8 bytes (including structs) is passed by reference
- structs and unions of 8, 16, 32, or 64-bits are passed as if they were integers of the same size
- The first 4 integer parameters are passed (in left to right order) in rcx, rdx, r8, r9
- Further integer parameters are passed on the stack by pushing them in right to left order (parameters to the left at lower addresses)
- The *this* pointer is passed in rcx
The x64 Calling Convention

Passing parameters

• All arguments are right justified in registers
• All stack parameters are 8 byte aligned
• Any parameter that's not 1, 2, 4, or 8 bytes (including structs) is passed by reference
• Structs and unions of 8, 16, 32, or 64-bits are passed as if they were integers of the same size
• The first 4 integer parameters are passed (in left to right order) in rcx, rdx, r8, r9
• Further integer parameters are passed on the stack by pushing them in right to left order (parameters to the left at lower addresses)
• The *this* pointer is passed in rcx
The x64 Calling Convention

Passing parameters

• The first floating point parameters are passed (left to right) in xmm0 to xmm3

• Further floating point parameters are passed on the stack, right to left order (parameters to the left at lower addresses)

• Return value:
  – On rax, if the return value is an integer or a pointer
  – On xmm0 if it is a floating point value

• Function must preserve: rbx, rbp, rdi, rsi, r12, r13, r14, r15, xmm6 - xmm15 and the x87 register stack

• Function may destroy: rax, rcx, rdx, r8, r9, r10, and r11 and xmm0 - xmm5
The x64 Calling Convention - Stack Handling

- The caller must reserve 32 bytes (4 64-bit values) on the stack
  - This space allows the rcx, rdx, r8, r9 registers to be easily copied to a well-known stack location if there is a need
- The caller is responsible for cleaning up the stack
  - Usually the compiler reserves enough stack space for the function that requires the most stack space and just adjust positioning within that stack space to fit all functions that it is calling
- A function can be a leaf or a frame function
  - Leaf functions don't need to support the stack unwinding process that is part of exception handling but it is limited (can’t call other functions)
  - Frame functions must handle many tasks such as building a stack frame, and save some register values
- For more info look at: http://software.intel.com/en-us/articles/introduction-to-x64-assembly/
The Heap

• A heap is an abstraction for managing memory allocations

• In Windows we have several heaps:
  – The process global heap, default size is 1MB
    • you can change it using the linker flag /Heap(reserve,commit)
  – The process’ custom heaps created by the HeapCreate API
  – The C++ Runtime heap
  – The kernel page and non-page pools
Objects and Handles

- Objects are runtime instances of static structures (object type)
  - Examples: process, mutex, event, desktop, file
- Reside in system memory space
- Kernel code can obtain direct pointer to an object
- User mode code can only obtain a handle to an object
  - Shields user code from directly accessing an object
- Objects are reference counted
- The Object Manager is the entity responsible for creating, obtaining and otherwise manipulating objects
The Registry

- Global hierarchical repository of data
- Machine wide data as well as user specific data
- Persistent as well as volatile data
- Also “contains” performance data
  - Not really in the registry
  - But the registry API is used to query this data

- Main hives
  - HKEY_LOCAL_MACHINE – machine wide settings
  - HKEY_CURRENT_USER – user specific settings
Virtualization (Hyper-V)

Hyper-V High Level Architecture

Root Partition
- VMWPs
- VMMS
- WMI
- VSPs
- VID
- WinHv
- VMBus

Enlightened Windows Child Partition
- User Applications
- VSCs/ICs
- WinHv
- VMBus

Enlightened Linux Child Partition
- User Applications
- Linux VSCs/ICs
- I/O Stack
- Drivers
- LinuxHv
- VMBus

Unenlightened Child Partition
- User Applications
- Kernel

Hypervisor
- Hypercalls
- MSR
- APIC
- Scheduler
- Address Management
- Partition Manager

Processors

Memory
Containers (Windows 10/Server 2016)
History

• Design started in 1989 by David Cutler
• First released as Windows NT 3.1 at 1993
• Many SKUs, Many flavors
  – From the smallest IoT device to the larger Surface Hub screen and the strong server
Windows NT Design Goals

• Separate address space per process
  – One process cannot (easily) corrupt another’s memory
• Protected kernel
  – User mode applications cannot crash kernel
• Preemptive multitasking and multithreading
• Multiprocessing support
• Internationalization support using Unicode
• Security throughout the system
• Integrated networking
Windows NT Design Goals

• Powerful file system (NTFS)
  – Supports protection, compression and encryption

• Run most 16 bit Windows and DOS apps
  – On 32 bit systems (NTVDM is not supported in 64-bit versions of Windows)

• Run POSIX 1003.1 and OS/2 applications

• Portable across processors and platforms

• Be a great client as well as server platform
Windows Editions

• Windows XP Home
  – Designed as a replacement for the Windows 9x/ME family (“Consumer Windows”)

• Windows Professional (2000, XP), Vista, 7, 8, 8.1
  – Main desktop (client) OS

  – Server platforms

• Other variants

• New Variants (Windows 10):
  – Xbox One, HoloLens, Surface Hub, Windows IoT
Windows Numeric Versions

- Windows NT 4 (NT 4.0)
- Windows 2000 (NT 5.0)
- Windows XP (NT 5.1)
- Windows Server 2003, 2003 R2, XP 64 (NT 5.2)
- Windows Vista, Server 2008 (NT 6.0)
- Windows 7, Server 2008 R2 (NT 6.1)
- Windows 8, Server 2012, Windows Phone 8 (NT 6.2)
- Windows 8.1, Server 2012 R2, Windows Phone 8.1 (NT 6.3)
- Windows 10, Server 2016 (NT 10.0)
- These values are obtained using GetVersionEx
Professional vs. Server

- Same core system files
- Differences
  - Number of processors supported
  - **Maximum amount of RAM** than can be used
  - Maximum of **concurrent network connections** supported for file and print sharing (10 on professional, **20 for Windows 7**)
  - Some services only appear in Server versions
  - Other system policies and default settings (e.g. thread quantum)
- To query the OS type, use `GetVersionEx` (Win32) or `RtlGetVersion` (WDK)
Product Data Type

• The actual product type is stored in the registry
  – HKLM\SYSTEM\CCS\Control\ProductOptions

• Values
  – **ProductType**
    • **WinNT** – 2000 Professional, XP Home, XP professional, Vista, 8
    • **ServerNT** – Server (domain controller)
    • **LanmanNT** – Server (not a domain controller)
  – **ProductSuite**
    • Distinguishes the various server types, and between XP Home and Professional editions

• Memory limits for the various editions can be found at
Kernel Mode Components

- Hardware abstraction Layer (HAL)
  - Isolates the kernel and device drivers from platform specific issues
- Kernel
  - Thread scheduling, interrupt & exception dispatching, multiprocessor support, synchronization primitives
- Device Drivers
  - Loadable kernel modules that handle I/O requests for hardware devices and buses
- Executive
  - Virtual memory manager, object manager, security, IPC, plug & play, power manager, configuration manager
- Win32K.SYS
  - The Windows subsystem kernel component
  - Handles user interface and graphics
User Mode Components

• User applications
  – Executables under one of 3 subsystems: Win32 (also Win 16, DOS), POSIX, OS/2

• System processes
  – Logon, Session manager, Service Control Manager

• Services
  – Normal Win32 processes that also interact with the SCM

• Subsystem process
  – A single process per subsystem (per session) handling subsystem specific issues

• Subsystem DLLs
  – Libraries implementing the API for a subsystem
    – Win32: kernel32.dll, user32.dll, gdi32.dll, advapi32.dll

• NTDLL.DLL
  – Implements the (undocumented) native API that subsystem DLLs use
Core System Files

- **Ntoskrnl.exe**
  - Executive and kernel
  - Original is NtOsKrnl.Exe (single CPU) or NtKrn1Mp.Exe (multi CPU)
- **NtKrn1Pa.exe**
  - Executive and kernel (32 bit) with support for Physical Address Extension (PAE)
  - Original is NtKrn1Pa.Exe (single CPU) or NtKrPaMp.Exe (multi CPU)
- **Hal.dll**
  - Hardware Abstraction Layer
- **Win32k.sys**
  - Kernel component of the Win32/Win64 subsystem
- **NtD11.dll**
  - System support routines and Native API dispatcher to executive services
- **Kernel32.dll, user32.dll, gdi32.dll, advapi32.dll**
  - Core Windows subsystem DLLs
- **CSRSS.exe (Client Server Runtime SubSystem)**
  - The Win32/Win64 subsystem process
Symmetric Multiprocessing

- SMP
  - All CPUs are the same and share main memory and have equal access to peripheral devices (no master/slave)
- Basic architecture supports up to 32/64 CPUs
  - Windows 7 64 bit & 2008 R2 support up to 256 cores
  - Windows 8 / 2012 supports up to 640 cores
- Actual number of CPUs determined by licensing and product type
  - Prior NT 6.0: Number of licensed CPUs
    HKLM\System\CCS\Control\Session Manager
      • LicensedProcessors (DWORD)
  - NT 6.0 and above ([Licensed Processors](#)):
    • HKLM\System\CCS\Control\ProductOptions\ProductPolicy
NUMA

- NUMA (Non Uniform Memory Architecture) systems
  - Groups of physical processors ("nodes") that have local memory
    - Connected to the larger system through a cache-coherent interconnect bus
  - Still an SMP system (e.g. any processor can access all of memory)
    - But node-local memory is faster
- Scheduling algorithms take this into account
  - Try to schedule threads on processors within the same node
  - Try to allocate memory from local memory for processes with threads on the node
- New Windows APIs to allow applications to optimize
Environment Subsystems

• A subsystem is a special view of the OS
  – Exposes services via subsystem DLLs

• Windows 2000 ships with Win32, OS/2 and POSIX 1003.1 (POSIX-1)

• Windows XP and later have Windows and a basic POSIX
  – An enhanced POSIX version is available with the “Services for UNIX” product

• Windows 10 has a new Ubuntu based capabilities
  – For providing Bash and other tools for developers
  – It does not implemented as a sub-system

• The Windows subsystem must always be running
  – Owner of keyboard, mouse and display

• Other subsystems configured to load on demand
Subsystems in the Registry

- Windows 8.1 snapshot
Subsystem DLLs

- Every image belongs to exactly one subsystem
  - Value stored in image PE header
    - Can view with the exetype.exe utility (originally from Win2K resource kit), can be found as a GitHub project
    - the dependency walker (depends.exe) or dump bin utility (dumpbin.exe)
  - Allows the Windows Loader to make correct decisions
- An image of a certain subsystem calls API functions exposed through the subsystem DLLs
  - E.g. kernel32.dll, user32.dll, etc. for the Windows subsystem
- Some images belong to no subsystem
  - “Native” images (Which API functions do they call?)
The Native API

• Implemented by NTDLL.DLL
  – Used by subsystem DLLs and “native” images
  – Undocumented interface
  – Lowest layer of user mode code

• Contains
  – Various support functions
  – Dispatcher to kernel services
    • Most of them accessible using Windows API “wrappers”
NTDLL.DLL Kernel Gate

• 32 bit dispatching code (XP & Pentium II and up)
  • 32 bit (XP Pentium I & Windows 2000)
    – Uses int 0x2e instead of sysenter
  • 64 bit dispatching code is similar (using the syscall instruction)
Function Call Flow

• E.g. *ReadFile*

![Diagram of function call flow](image-url)
System Service Table(s)

• 2 built-in, up to 4 supported
  – Bits 12-13 select the table
  – Lower 12 bits select the service
• Each thread has a pointer to its system service table
• EAX serves as an index to the system service required
  – 32 bit Windows holds the actual addresses
  – 64 bit Windows holds offsets
    • Lower 4 bits used as argument count
    • Must be masked off to get actual offset
Default System Tables

• **KeServiceDescriptorTable**
  – Contains only kernel related entries

• **KeServiceDescriptorTableShadow**
  – Contains both kernel related entries and USER and GDI entries

• When a thread is first created, its table pointer is set to **KeServiceDescriptorTable**

• The first time it makes any GDI or USER call, its table pointer is changed
Executive

• The upper layer of NtOskrn1.exe

• Functions
  – System service functions callable from user mode
    • Accessed through NtDll.dll
    • Most can be called from the Windows API
  – Callable by device drivers and documented in the WDK
  – Exported but undocumented for use by NtOskrn1.exe
  – Internal functions
Executive Components

• Configuration manager
  – Responsible for managing the system registry

• Object manager
  – Manages objects created by kernel or user code

• Process and thread manager
  – Creates and terminates processes and threads
  – The underlying support is provided by the lower-layer Kernel

• Security Reference Monitor (SRM)
  – Implements local security enforcement, including object protection and auditing

• I/O manager
  – Implements device independent I/O and dispatches calls to appropriate device drivers
Executive Components

• Plug & Play (PnP) manager
  – Handles hardware identification, enumeration, resource allocation and loading of appropriate drivers

• Power manager
  – Coordinates power events and generates I/O requests to drivers based on state changes

• Cache manager
  – Manages caching for file based I/O

• Memory manager
  – Implements virtual memory handling for use by other system components
Kernel

• The lower layer of the Kernel
• Implements all low level activity, such as interrupt dispatching, thread scheduling and processor synchronization

• Implements a set of kernel objects
  – Control objects
    • Controlling various OS functions, e.g. APC, DPC
  – Dispatcher objects
    • Have synchronization capabilities, e.g. mutex, event
    • Executive adds management over these primitive objects, e.g. security, handle management and reference counting
Hardware Abstraction Layer

• Isolates the kernel and device drivers from the underlying hardware platform
  – E.g. interrupt controller

• Allows easy access to hardware by device drivers

• Various HALs are on the Windows install media
  – The appropriate one is copies to the System32 directory
    and is always named Hal.Dll
Device Drivers

• Loadable kernel modules (usually with SYS extension)
• The only documented way to add code that runs in the kernel
• Types (partial list)
  – Hardware device drivers – manage a physical device
  – File system device drivers – translate file I/O to device specific requests
  – Protocol drivers – implement a specific network protocol (TCP/IP, IPX/SPX, etc.)
  – User programmed driver, simply for the ability to execute code in kernel mode
System Processes

• Idle process
• System process
• Session Manager (\Smss\Exe)
• Windows subsystem (\Csrss\Exe)
• Logon process (\Winlogon\Exe)
• Service control manager (SCM) (\Services\Exe)
• Local security authentication server (\Lsass\Exe)
Idle Process

- Always has a PID of 0
- Not a real process (does not run any executable image)
- One thread per CPU (core)
- Accounts for idle time
System Process

- Has a fixed PID (8 on Win2K, 4 on XP and later)
- Home of system threads
  - Threads created by the kernel and device drivers
  - Execute code in system space only
  - Created using the `PsCreateSystemThread` kernel API (documented in the WDK)
  - Allocate memory from the system pools
  - Windows 10/Server 2016 – Memory size includes processes’ compressed pages
Session Manager

• Running the image \windows\system32\smss.exe
• The first user mode process created by the system
• Tasks
  – Create additional page files
  – Creating system environment variables
  – Launches the subsystem processes (normally just csrss.exe)
  – Launches itself in other sessions
    • That instance loads WINLOGON and CSRSS in that session
    • Then terminates
• Finally
  – waits forever for csrss.exe instances to terminate
    • If any of them dies, crashes the system
  – Waits for subsystem creation requests
  – Waits for terminal services session creation requests
Winlogon

- Running the image \windows\system32\winlogon.exe
- Handles interactive logons and logoffs
- If terminated, logs off the user session
- Notified of a user request by the Secure Attention Sequence (SAS), typically Ctrl+Alt+Del
- Authenticates the user by presenting a username / password dialog
  - Can be replaced
- Sends captured username and password to LSASS
  - If successfully authenticated, initiates the user’s session
LSASS

• Running the image `\windows\system32\Lsass.exe`
• Calls the appropriate authentication package
• Upon successful authentication, creates a token representing the user’s security profile
• Returns information to Winlogon
Service Control Manager (SCM)

- Running the image `\windows\system32\services.exe`
- Responsible for starting, stopping and interacting with service processes
- Services
  - Similar to UNIX “daemon processes”
  - Normal Windows executables, that interact with the SCM
  - Can be started automatically when the system starts up without an interactive logon
  - Can run under “special” accounts
    - LocalSystem, NetworkService, LocalService
Summary

• User mode code has limited access to the system
• Kernel mode has full and total access to all system resources
• Applications run under a specific subsystem
• The primary subsystem is the Windows subsystem
Windows NT 6.x Versions

• Windows 6.0
  – Windows Vista
  – Windows Server 2008
  – Windows Home Server 2011
• Windows 6.1
  – Windows 7
  – Windows Server 2008 R2
  – Windows Thin PC
• Windows 6.2
  – Windows 8
• Windows 6.3
  – Windows 8.1
  – Windows Server 2012 R2
What’s New in 6.0

• Lots of kernel improvements
  – In the memory manager, process scheduler and I/O scheduler
  – Many new security features such as BitLocker, ASLR, and an improved Windows Firewall

• Windows Aero desktop theme

• DWM, WDDM, DirectX 10, IE 7, IIS 7

• UAC, ReadyBoost, SuperFetch, IPv6

• Kernel Transaction, Transactional NTFS & Registry

• Restart Manager, I/O Priority, ETW

• Server Core, Self-healing NTFS

• And many more...
What’s New in 6.1

• Many Kernel Changes
  – 256 Cores, Core & Socket Parking, Hot Locks Removal, Better Windows Memory Utilization, Faster Registry Access

• Power Consumption Improvements
  – Trigger Start Services, Timers, Power Consumption Profiling

• Client Side Enhancements
  – Taskbar, Multi-Touch, Ribbon, Shell Libraries, Federated Search, Direct2D & DirectWrite

• Tons of features
  – Compatibility, troubleshooting platform, Mount and boot from VHD, Sensor & Location
Power Consumption Profiling
Powercfg /energy

DEMO
What’s New in 6.2

- Windows 8 and Windows Server 2012
- Better Tablet Support, ability to run on ARM, Support 640 cores
- **WinRT** – A new Windows programming model
  - Object Oriented Class Library
  - Ref-Count lifetime management
  - .NET like Metadata Support
  - Natively supports C++, .NET and JavaScript
- **Windows Store App (Metro)** – A new UI approach for Windows applications
  - Chrome-less UI Tile based that supports touch and mouse
  - App development with XAML/C++|.NET or HTML5 + JavaScript
  - Handles application start, switch, suspend and shutdown
  - Support application inter-communication through contracts
  - App distribution & installation through Microsoft App Store
Windows Store Application

DEMO
What’s New in 6.3

- Windows 8.1, Server 2012 R2
  - Internet Explorer 11 ([WebGL](#), [SPDY](#))
  - PowerShell 4.0
    - New commands for managing the Start screen, Windows Defender & Windows components
  - Start Screen Enhancement (Tile Sizes, All Apps, Search, Desktop Background)
  - Boot to desktop & Visible Desktop Start Button
  - New technology support
    - [NFC](#) printing, [Wi-Fi Direct](#) printing, [Miracast](#), [Mobile broadband tethering](#), Auto-triggered [VPN](#)
  - Many new (6000) XAML and WinJS APIs
  - [DirectX](#) 11.2
  - Better Windows Store Application Multitasking
NT 6.x Kernel Changes

• Windows 7 (x64) and Server 2008 R2 are based on same kernel (OS version number 6.1)
  – Windows 8/2012 (6.2), Windows 8.1/2012 R2 (6.3)
• Enhancements in many areas
  – Multi- and Many-Core Processing
  – Power Efficiency
  – Security
  – Management
  – Virtualization
  – Componentization and Layering
• And, lots of Windows 7 specific features
  – TaskBar, Direct Write/2D/3D, GPGPU, Touch, others
NUMA Processor Group Support

• Segmented specification – “groups” of CPUs
  – Windows 6.X Supports more than 64 Logical Processors
    • 256 on Windows 7/2008 R2 and 640 on Windows 8/2012
  – CPUs identified in software by Group#: CPU#
  – Allows backward compatibility with 64-bit affinity
  – New applications have full CPU range using new APIs

• Permits better locality during scheduling than a “flat” specification
Remove Coarse-Grained Locks

• Locks serialize access to data structures
  – Prevents multiple threads from simultaneously modifying data
  – Inhibits scaling because threads must wait for their turn (contention)

• Examples:
  – Dispatcher Lock
  – Memory Manager PFN Lock
  – Cache Manager Virtual Address Control Block (VACB) Lock
  – Object Manager Type Lock

• Either replaced with lock-free algorithms or with finer-grained synchronization mechanisms
Memory Optimizations

• Desktop Window Manager (DWM) re-architecture reduces memory footprint per window by 50%

• Registry moved from mapped files to paged pool
  – Improves performance because views into registry file don’t need to be mapped and unmapped

• Working set management improvements:
  – Working set is amount of RAM memory manager assigns to process or kernel memory type
  – Memory manager uses more information for better tuning
  – System cache, paged pool, and pageable system code each have own working set
Power Efficiency

• Keep idle and stay idle
  – Minimize running services and tasks
  – Avoid background processing
  – Let LPs and sockets stay idle so that they enter deep sleep (C states)

• Core Parking
  – Move load from one CPU to another
    • The CPU and the Socket go to sleep, conserving power

• Trigger-Started Services
  – To reduce the number of background running processes
Virtual Accounts

• Want better isolation than existing service accounts
  – Don’t want to manage passwords

• Virtual accounts are like service accounts:
  – Process runs with virtual SID as principle
    • Can ACL objects to that SID
  – System-managed password
  – Show up as computer account when accessing network

• Services can specify a virtual account
  – Account name must be “NT SERVICE\<service>”
    • Service control manager verifies that service name matches account name
  – Service control manager creates a user profile for the account

• Also used by IIS app pool and SQL Server
Native VHD Support

• Foundational support for booting from VHD and for Attach/Removal of VHDs
  – Orderly and surprise removal of volumes
  – Support for nested volumes
  – Servicing for mounted (offline) VHD volumes

• VHD operations
  – Create / Attach / Remove

• Tools and APIs:
  – Win32 APIs
  – VDS APIs (DCOM Remotable)
  – Hyper-V WMI for management operations

• Performance goal: within 10% of native
Create and Attach VHD

DEMO
VHD Boot in Windows
Server Core Architecture

Server Core Server Roles and Optional Features

- DNS
- DHCP
- AD
- File
- AD LDS
- Print
- Media Server
- IIS
- Hyper-V
- .NET
- PS
- ASP.NET
- TS
- NAS
- ADFS
- WDS
- Etc...

Server
With .NetFx, Shell, Tools, etc.

Server Core
Security, TCP/IP, File Systems, RPC, plus other Core Server Sub-Systems

GUI, Shell, IE, Media, Mail, Etc.

Windows Internals
What’s New in Windows 10.X

2500+ new platform features
Windows Core

• The refactored common core
  – One hardware platform, Universal hardware driver, Standard network and I/O
The convergence journey

Converged OS kernel

Unified core and app platform

Easy for users to get & stay current
One Store + One Dev Center

Reuse Existing Code

One SDK + Tooling

Adaptive User Interface

Natural User Inputs

One Universal Windows Platform

Universal Apps

Devices + IoT

Mobile

PC

XBox

Surface Hub

HoloLens
Windows 10 operating system

Universal Windows Platform

Bridging technologies

- Win32 desktop
- Web hosted
- Android
- iOS

Windows Presentation Foundation (WPF)
- .Net languages
- C++
- .Net runtime

Web Forms (WF)
- .Net languages
- C++ & CX
- .Net runtime

Microsoft Foundation Class (MFC)
- C++
- .Net languages

XAML
- DirectX
- HTML
- WWA

 languages

C++

.NET runtime

Obj.C
Java
Web
Win32
Universal Windows Platform

- A single API surface
- A guaranteed API surface
- The same on all devices
# Windows 10 for IoT Devices

<table>
<thead>
<tr>
<th>One Windows Platform</th>
<th>Secure</th>
<th>Connected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 10 IoT for Industry Devices</td>
<td>One Windows core scalable across devices</td>
<td>Enterprise-grade security</td>
</tr>
<tr>
<td>Windows 10 IoT for Mobile Devices</td>
<td>One universal app and driver platform</td>
<td>Advanced Customization and lockdown capabilities</td>
</tr>
<tr>
<td>Windows 10 IoT for Small Devices</td>
<td>One management and deployment approach</td>
<td>Continued innovation and support</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Local connectivity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cloud connectivity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Device services</td>
</tr>
</tbody>
</table>
Windows 10 IoT Core

• Use the same API, SDK, Visual Studio to develop Universal App
  – Extend UWP with device specific API
  – Run your desktop app on the PI
  • Extend it to control and to be controlled by hardware devices
  – You can Use Arduino Wiring Pi on Windows 10 IoT devices
    • C++ library that mimic the Arduino library
Nano Server (Server 2016)

• A remotely administered server operating system optimized for private clouds and datacenters
  – Similar to Windows Server in Server Core mode, but significantly smaller
  – Has no local logon capability
  – Only supports 64-bit applications, tools, and agents
  – Takes up far less disk space, sets up significantly faster, and requires far fewer updates and restarts

• Nano Server is ideal for a number of scenarios:
  – As a “compute” host for Hyper-V virtual machines
  – As a storage host for Scale-Out File Server
  – As a DNS server
  – As a web server running Internet Information Services (IIS)
  – As a host for applications that are developed using cloud application patterns
What’s New in NT 10.X

• Security
  – VSM – Virtual Secure mode
    • Enables isolated User Mode (protect user mode from the kernel)
  – IOMMU – I/O Memory Management Unit
    • MMU for devices, as opposed to processors
    • Hardware based protection against DMA-access
    • Protects against buggy drivers and malicious code
  – CFG – Control Flow Guard
  – Windows Hello
What’s New in NT 10.X

- Memory Compression
  - Can compress pages in memory and result in less Hard Page Faults
  - Does not prevent the use of pagefile but lowers the need to use it
- Lightweight Suspend & Deep Sleep
- Universal Drivers Development
- Cortana
- Continuum
- Developer Mode & Device Developer Portal
- Many internal mechanisms improvements such as Windows Loader, Kernel Timers & Scheduler
Agenda

- Trap Dispatching
- Object Management
- Synchronization
- Kernel Event Tracing
- Wow64
- Kernel Transaction Manager
- Diagnostics
- Summary
Trap Dispatching

• Traps
  – Interrupts or exceptions
  – Divert code execution outside the normal flow

• Trap dispatching
  – Kernel mechanisms for capturing an executing thread when an interrupt or exception occurs and transferring control to a handling routine

• Interrupt
  – Asynchronous event, unrelated to the current executing code

• Exception
  – Synchronous call to certain instructions
  – Reproducible under the same conditions
Hardware Interrupts

CPU

Interrupt Controller

INTA

INTR

IRQ 0

IRQ 1

Device 1

Device 2

Device n

Data bus

IRQ number
Interrupt Dispatching

- Kernel or User mode code
- Interrupt
- Record CPU state (trap frame)
- Mask equal or lower IRQL interrupts
- Call appropriate ISR
- Restore CPU state
- Interrupt Service Routine
Hardware Interrupts Requests

- The device needing service signals the line connected to the interrupt controller (PIC or APIC)
- The interrupt controller signals the CPU on a single line
- The CPU requests the interrupt vector number from the interrupt controller
- The appropriate ISR is called based on its entry in the Interrupt Dispatch Table (IDT)
Interrupt Request Level (IRQL)

- Each interrupt has an associated Interrupt Request Level (IRQL)
  - Can be considered its priority
  - For hardware interrupts, mapped by the HAL
- Each processor's context includes its current IRQL
  - A CPU always runs the highest IRQL code
- Servicing an interrupt raises the processor IRQL to the level of the interrupt's IRQL
  - This masks all interrupts at that IRQL and lower
- Dismissing an interrupt restores the processor's IRQL to that prior to the interrupt
  - Allowing any previously masked interrupts to be serviced
IRQL Levels

HIGH_LEVEL (31)
POWER_LEVEL (30)
IPI_LEVEL (29)
CLOCK_LEVEL (28)
PROFILE_LEVEL (27)
Device n
  .
  .
Device 1
DISPATCH_LEVEL (2)
APC_LEVEL (1)
PASSIVE_LEVEL (0)

High/Profile (15)
IPI/Power (14)
Clock (13)
Synch (2003) (12)
Device n
  .
Device 1
DISPATCH_LEVEL (2)
APC_LEVEL (1)
PASSIVE_LEVEL (0)

x86

x64

Windows Internals
IRQL Levels

• **PASSIVE_LEVEL (0)**
  – The “normal” IQRL level
  – User mode code always runs at this level
• **APC_LEVEL (1)**
  – Used for special kernel APCs (see I/O System module)
• **DISPATCH_LEVEL or DPC_LEVEL (2)**
  – The kernel scheduler runs at this level
    • If the CPU runs code at this (or higher) level, no context switching will occur on that CPU until IRQL drops below this level
    • Also no waiting on kernel objects (requires scheduler)
• Page faults can only be handled in IRQL < DPC_LEVEL
  – Code running at this or higher IRQL must always access non-paged memory
IRQL Levels

• Device IRQL (DIRQL) (3-11/26)
  – Reserved for hardware devices
  – The level that an ISR runs at
  – Always greater than DISPATCH_LEVEL (2)

• HIGH_LEVEL (x86=31, x64=15)
  – The highest level possible
  – If code runs at this level, nothing can interfere on that CPU
  – However, other CPUs are not affected
  – Use only when absolutely necessary!

• Other levels exist for kernel internal use
IRQL vs. Thread Priorities

• IRQL is an attribute of a CPU
• Thread priority is an attribute of a thread
• With IRQL $\geq$ DISPATCH_LEVEL (2)
  – Thread priority has no meaning
  – The currently executing thread will execute forever, until IRQL drops below 2
• IRQLs can be changed (in kernel mode only) with KeRaiseIrql and KeLowerIrql
Interrupts and IRQLs

User mode code (0)

ISR 1 (5)

ISR 2 (8)
Mapping Interrupts to IRQLs

• IRQL is a software-only entity
• IRQL is not the same as IRQ
  – IRQ is an interrupt line connected to the interrupt controller
• The HAL is responsible for translating IRQs to vector numbers and IRQLs
• Vector is assigned in a round-robin fashion
• IRQL = Vector / 16
Software Interrupts

- The kernel uses software interrupts for various tasks
  - Initiate thread dispatching
  - Non-time critical interrupt processing
  - Handling timer expirations
  - Asynchronously execute a procedure in a context of a specific thread
Deferred Procedure Call (DPC)

• Used to defer processing from higher (device) IRQL to a lower (dispatch) level
• Implemented via DPC objects and software interrupts
• DPC object defines a procedure and arguments
• Executes specified procedure at Dispatch IRQL (2)
• Used heavily for driver "after interrupt" functions
  – ISR calls KeInsertQueueDpc or IoRequestDpc to queue a DPC
DPC Queue

• A list of "work requests"
  – One queue per CPU
    • But one CPU can process a DPC from another CPU's queue
  – Implicitly ordered by time (FIFO)
    • Can be somewhat manipulated with the Importance field
  – Each specifies procedure and arguments

• Processed after all higher-IRQL work (interrupts) completed
The DPC Processing Loop

- DPCs are queued and cannot interrupt each other on any one CPU
- Each CPU in the system might be executing a different DPC at the same time

Software interrupt at DPC level

- Attempt to remove a DPC object from the DPC queue
- Did we get one?
  - Yes
    - DPC routine executes and returns
  - No (queue empty)
    - Call DPC routine with arguments from the DPC object
DPC Processing Notes

• Idle CPUs can help by processing DPCs from another CPU’s DPC queue

• A DPC can be targeted at a different CPU than the executing one by calling `KeSetTargetProcessorDPC/Ex` API
  – Still may be processed by another (idle) CPU

• DPC importance can be controlled by calling `KeSetImportanceDpc`
  – Low, medium (default), high (goes to head of queue)
Exception Dispatching

• Synchronous event resulting from certain code
• Examples
  – Divide by zero, access violation, stack overflow, invalid instruction
• Structured Exception Handling (SEH)
  – A mechanism used to handle, and possibly resolve, exceptions
• Exceptions are connected to entries in the IDT
  – An exception dispatcher in the kernel is responsible for “disposing” of the exception
Exception Handling

• Some exceptions are handled transparently
  – E.g. a breakpoint is an exception, which is transferred to the appropriate debugger

• Some exceptions are filtered back to user mode for possible handling
  – E.g. accessing a user mode address that is not mapped

• Frame based exception handlers are searched
  – If the current frame has none, the previous frame is searched and so on

• Unhandled exceptions from kernel mode generate a “bug check” a.k.a. “Blue Screen of Death”
Resolving Exceptions

Exception occurred

Switch to kernel mode (if was in user mode)

Create a Trap Frame

Exception occurred in kernel mode?

Yes → Look for a handler

Found one?

Yes → Execute handler → Done

No → Crash system

No
Resolving Exceptions

Exception occurred in user mode

- Debugger attached?
  - Yes: Send message to Debug port
  - No: Search frame base handlers

Search frame base handlers

- Found one?
  - Yes: Execute handler
  - No: "First chance" handled By debugger?
    - Yes: Done
    - No: Send message to Debug port
Resolving Exceptions

- No frame based handler found
  - Debugger attached?
    - Yes
      - Send message to Debug port
    - No
      - “Second chance” handled By debugger?
        - Yes
          - Done
        - No
          - Subsystem handles exception?
            - Yes
              - Send message to Exception port
            - No
              - Send message to Error port
    - No
      - Send message to Exception port

- Terminate process
Structured Exception Handling (SEH)

- Exposed for developers by extended C keywords
  - __try
    - Wraps a block of code that may throw exceptions
  - __except
    - Possible block for handling exceptions in the preceding __try block
  - __finally
    - Execute code whether an exception occurred or not
  - __leave
    - Jumps to the __finally clause
  - Allowed blocks are __try/__finally and __try/__except
    - However, can be nested to any level

- Works in kernel mode and user mode
- Custom exceptions can be raised with **RaiseException** (Win32)
CRITICAL_SECTION cs;

void f()
{
    EnterCriticalSection(&cs);
    __try
    {
        // do work while critical section held
    }
    __finally
    {
        LeaveCriticalSection(&cs);
    }
}
SEH and High Level Exceptions

• How does SEH relate to C++, .NET or Java exceptions?
• C++ (at least Microsoft’s’s compiler), .NET and Java use SEH internally
  – With a custom exception code
  – The `EXCEPTION_RECORD` holds the extra custom information that is understood by the particular runtime

• What about finally in C++?
  – No `finally` keyword
  – However, can be simulated with the idiom Resource Acquisition Is Initialization (RAII)
CRITICAL_SECTION cs;

class AutoCS {
    CRITICAL_SECTION* pcs;
public:
    AutoCS(CRITICAL_SECTION* c) : pcs(c) {
        EnterCriticalSection(pcs);
    }
    ~AutoCS() {
        LeaveCriticalSection(pcs);
    }
};

void f() {
    AutoCS lock(&cs);
    // access shared resource...
}
System Crash

• “Blue Screen of Death”
• Occurs when code running in kernel mode raises an unhandled exception
• First steps in diagnosis
  – Make sure a crash dump file is written to disk
  – don’t automatically restart
• Set in the Startup Recovery dialog in System applet / Advanced Settings / Startup & Recovery
Startup And Recovery

Windows Internals
“Blue Screen” Information

- A blue screen shows
  - The code of the crash (e.g. 0x0A)
  - The textual name of the code
    - \texttt{IRQL\_NOT\_LESS\_OR\_EQUAL}
  - Four numbers –
    their meaning depends on the exact error code
    - Go to the debugging tools help and search for the stop code to learn more
  - A short stack trace leading to the crash
    - Typically ending in \texttt{KeBugCheckEx} call
    - For simple scenarios, culprit is on the stack
Object Manager

• Executive component responsible for managing executive objects
• Manages creating, deleting, protecting and tracking objects
• Maintains a hierarchical namespace of objects
  – Can be viewed partially with the WinObj utility (from www.sysinternals.com)
• User mode clients can obtain handles to objects
  – Cannot touch actual memory structure
• Kernel mode clients can do either
Object Manager Namespace
Object Kinds

• Executive objects
  – High level objects, usually encapsulating a kernel object
  – Add reference tracking, security, handle management

• Kernel objects
  – Primitive objects
  – Implement the actual functionality of the object
  – Provide synchronization mechanisms
Object Structure

- Objects Name
- Object Directory
- Security Descriptor
- Quota Charges
- Open Handle Count
- Open Handles List
- Object Type
- Reference Count

Owned by Object Manager

Owned by kernel

Owned by Executive

Kernel Object

Object Data

Type Object
- Type Name
- Access Type
- Synchronizable?
- Pageable?
- Object Methods
Object Types

- Symbolic link - Allows referencing objects indirectly
- Process - A virtual address space and control information necessary for execution of thread objects
- Thread - An executable entity within a process
- Job - A collection of processes managed as a group
- Section (File mapping) - A region of shared memory
- File - Represents an open file or another I/O device
- Token - A security profile of a thread or process
- Event - A flag object, used for synchronization or notification
- Semaphore - A counter providing a resource gate for maximum allowed threads accessing a shared resource
Object Types

• Mutex (mutant) - A synchronization object used to serialize access to a shared resource
• Timer - An object allowing notification after some time period elapsed
• IoCompletion (I/O completion port) - An object for notifications of I/O operation completion
• Key - An object representing registry data
• Window Station - A management object, containing desktops, a clipboard and an atom table
• Desktop - An object contained within a window station, contains a drawing surface, windows and hooks
• Transaction (many types) – Kernel Transaction Manager types
• And more… (Some undocumented yet)
Type Object Attributes

• Pool type
  – Indicates whether objects of this type need to be allocated from paged or non-paged pool

• Access type
  – The type of access a thread can request when opening an object
    • E.g. Read, Write, Synchronize

• Synchronization
  – Indicates whether a thread can wait on objects of this type
    • Job, Process, Thread, File, Timer, Event, Semaphore, Mutex, I/O Completion Port

• Methods
  – One or more routines that the Object Manager calls at certain points of the object’s lifetime
Object Handles

• When a process creates or opens an object, it receives a handle to the object
  – Used as an opaque, indirect pointer to the underlying object
  – Allows sharing objects across processes

• Each process has a private handle table
  – Sharing is possible through
    • Process handle inheritance
    • Opening an object by name
    • Duplicating a handle (DuplicateHandle Windows API)

• Viewing process handles
  – Use Process Explorer (GUI), handle.exe (Console) (from www.SysInternals.com)
Process Handle Table

• Pointed to by the **EPROCESS** object
  – A handle is an index to that table
  – Starting at 4 with increments of 4

• Implemented as a 3 level layout
Process Explorer Handle View
Handle Entry Layout

- **Object header** is always a multiple of 8 (x86) / 16 (x64) bytes
- **Flags**
  - P – protect from close
  - I – Is handle inherited in a child process?
  - A – auditing on close
  - L – synchronization lock
- **Access Mask**
  - Determines what actions can be performed using this handle
Sharing Objects

• A handle is private to its containing process

• Sometimes an object needs to be shared between processes

• Sharing is possible through
  – Process handle inheritance
  – Opening an object by name
  – Duplicating a handle
Handle Usage

- User mode processes retrieve a handle by calling an appropriate Create* function or an Open* function
  - A handle is returned upon success
- User mode processes close a handle using the CloseHandle API
  - Entry is deleted from the handle table
- Kernel mode code can obtain handles that reside in the kernel system space and is visible in any process context
- Alternatively, kernel mode code can obtain a direct pointer to the underlying object
  - Using ObReferenceObjectByHandle
  - Must release reference by calling ObDereferenceObject
Object Names

• Object can have names
  – Allows object lookup by name
  – Allows easy sharing between processes
• When an object is created, it can (optionally) have a name
  – Not allowed for all object types created from user mode
• Objects are stored in the object manager namespace
• Can be viewed with the WinObj tool
Object Namespace Directories

- Global?? - Symbolic links
- BaseNamedObjects – Global Mutexes, semaphores, events, timers and sections that have a name (named objects)
- Sessions/[0-...]/BaseNamedObjects – Local named objects
- Sessions/[0-...]/AppContainerNamedObject – Local Universal Windows Platform named objects
- Device - Device objects
- Driver - Driver objects
- FileSystem - File system driver objects and file system recognizer device objects
- KnownDLLs - Section names and paths for known DLLs (stored also in the registry), loaded automatically at startup
- ObjectTypes - All known object types
- Windows - Windows subsystem ports and window stations
WinObj.Exe - BaseNamedObjects

[Diagram of the Sysinternals WinObj tool, showing BaseNamedObjects and related objects and files.]
Object Names and Sessions

- In a terminal session environment, each session should have its own objects
  - From Windows XP, each logon (switch user) has its own session
  - From Windows Vista, services run in session 0
  - From Windows 8, Windows Store app (AKA UWP on Windows 10) has their own sub folder:
    - The object manager creates a Sessions directory with a session ID subdirectory under: Sessions/[0-…]/AppContainerNamedObject
      - Named objects are created under that directory for that session
      - Redirection is automatic
- Sessions can access the interactive session objects using the prefix "Global\" when using object names to retrieve a handle
User and GDI Objects

• The Object Manager is responsible for kernel objects only
• User and GDI objects are managed by Win32k.sys
• The API functions in user32.dll and gdi32.dll don’t go through Ntdll.dll
  – They invoke the sysenter/syscall instructions directly
User and GDI Objects

• User objects
  – Windows (HWND), menus (HMENU) and hooks (HHOOK)

• User object handles
  – No reference/handle counting
  – Private to a Window Station

• GDI objects
  – Device context (HDC), pen (HPEN), brush (HBRUSH), bitmap (HBITMAP) and others

• GDI object handles
  – No reference/handle counting
  – Private to a process
  – Cannot be shared across processes
Thread Synchronization

- Threads sometimes need to coordinate work
- Canonical example
  - Accessing a linked list concurrently from multiple threads
- Synchronization is based upon waiting for some condition to occur
- The kernel provides a set of synchronization (dispatcher) primitives on which threads can wait efficiently
Kernel Dispatcher Objects

• Maintain a state (signaled or non-signaled)
  – The meaning of “signaled” is dependent on object type

• Can be waited to change to the signaled state
  – Windows API: `WaitForSingleObject, WaitForMultipleObjects` and their variants
  – Kernel mode: `KeWaitForSingleObject, KeWaitForMultipleObjects`

• Dispatcher object types
  – Process, thread, event, mutex, semaphore, timer, file, I/O completion port

• Higher level wrappers exist
  – MFC: `CSyncObject` (abstract base of `CMutex, CSemaphore` and others)
  – .NET: `WaitHandle` (abstract base of `Mutex, Semaphore` and others)
“Signaled” Meaning

• Process - The process has terminated
• Thread - The thread has terminated
• Mutex - The mutex is free
• Event - The event flag is raised
• Semaphore - The semaphore count is below its maximum
• File, I/O completion port - I/O operation completed
• Timer - Interval time expires
Mutex

• Mutual exclusion
• Called Mutant in kernel terminology
• Allows a single thread to enter a critical region
• The thread that enters the critical region (its wait has succeeded) is the owner of the mutex
• Releasing the mutex allows one (single) thread to acquire it and enter the critical section
• Recursive acquisition is ok (increments a counter)
  – If the owning thread does not release the mutex before it terminates, the kernel releases it and the next wait succeeds with a special code (abandoned mutex)
Semaphore

• Maintains a counter (set at creation time)
• Allows x callers to “go through” a gate
• Does not maintain any ownership
• When a thread succeeds a wait, the semaphore counter decreases
  – When the counter reaches zero, subsequent waits do not succeed (state is non-signaled)
  – Releasing the semaphore increments its counter, releasing a thread that is waiting
Event

• Maintains a Boolean flag
• Event types
  – Manual reset (Notification)
  – Auto reset (Synchronization)
• When set (signaled) threads waiting for it succeed the wait
  – Manual reset event releases any number of threads
  – Auto reset event releases just one thread
    • And the event goes automatically to the non-signaled state
• Useful when no other object fits the bill
Critical Section

- User mode replacement for a mutex
- Can be used to synchronize threads within a single process
  - Operates on a structure of type `CRITICAL_SECTION`
- Cheaper than a mutex when no contention exists
  - No transition to kernel mode in this case
- Uses `EnterCriticalSection` and `LeaveCriticalSection` API functions
  - No way to specify a timeout other than infinite or zero
    - Zero is accomplished with `TryEnterCriticalSection`
- .NET
  - A similar effect is achieved with the `lock` C# keyword
  - Calls the framework’s `Monitor.Enter/Exit` in a `try/finally` block
Executive Resource

• Similar to a mutex
• Supports multiple-reader shared access
• Not exported to the Windows API
  – Win32 API has the Slim Reader Writer Lock
• Not a dispatcher object
  – Structure allocated from non-paged pool
  – Has a special API that can be used in kernel mode (documented in the WDK)
• Functions
  – `ExAcquireResourceExclusiveLite`, `ExAcquireResourceSharedLite`, `ExReleaseResourceLite`
Push Locks

• Similar to executive resources
• Advantages
  – Acquisition for shared access is faster the executive resources
  – Storage for EX_PUSH_LOCK can be allocated from paged or non-paged pool
    • Structure is smaller than ERESOURCE
• Disadvantages
  – Cannot be acquired recursively
  – No function to check if lock is currently exclusively acquired
• API
  – FltInitializePushLock, FltDeletePushLock
  – FltAcquirePushLockExclusive, FltAcquirePushLockShared
  – FltReleasePushLock
Asynchronous Procedure Call (APC)

• An object representing a callback function to be called by a specific thread
• APC forces a specified thread to run a specified routine in that thread
  – Makes the thread ready to run if it was waiting
  – At the end of the APC routine, the thread might go back to waiting (if it was in that state earlier)
• APC takes precedence over "normal" thread code
  – Only within that thread (does not affect its priority)
• Kernel mode API is exported but not documented
APC Types

• User mode APCs
  – Used for I/O completion callback routines (e.g. `ReadFileEx`, `WriteFileEx`)
  – Only deliverable when thread goes into "alertable state" (e.g. with `SleepEx`, `WaitForSingleObjectEx`, `MsgWaitForMultipleObjectsEx`)
  – Can be explicitly queued with `QueueUserApc` (Win32)

• Normal kernel APCs
  – Run in kernel mode, at IRQL PASSIVE_LEVEL (0)
  – Preempts user mode code (and user mode APCs)

• Special kernel APCs
  – Run in kernel mode, at IRQL APC_LEVEL (1)
  – Used for I/O completion report from "arbitrary thread context"
Critical Regions and Guarded Regions

• Critical region
  – User mode APCs and normal kernel mode APCs disabled
  – Enter with KeEnterCriticalSection
  – Leave with KeLeaveCriticalSection

• Guarded region
  – All APCs are disabled
  – Enter with KeEnterGuardedRegion
  – Leave with KeLeaveGuardedRegion

• Calls to Enter function and Leave functions must match
Fast Mutex

• Also called Executive Mutex
• Not exported to the Windows API
• Similar to a normal mutex
  – Better performance
• Differences
  – After acquisition IRQL is raised to APC_LEVEL
  – No recursive acquisition allowed (no ownership)
• API
  – `ExAcquireFastMutex`, `ExAcquireFastMutexUnsafe`
    • Unsafe version requires normal kernel mode APCs to be disabled (IRQL is APC_LEVEL)
  – `ExReleaseFastMutex`, `ExReleaseFastMutexUnsafe`
Guarded Mutexes

• Introduced in Windows Server 2003

• Similar to a fast mutex
  – Starting with Windows 8, implemented the same as fast mutex
  – Prior to Windows 8, a bit faster than a fast mutex

• Represented by the KGUARDED_MUTEX structure
# Kernel Synchronization Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Exposed to device drivers</th>
<th>Exposed to user mode</th>
<th>Disables normal kernel APCs</th>
<th>Disables special kernel APCs</th>
<th>Supports recursive acquisition</th>
<th>Supports shared and exclusive acquisition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel Mutex</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Kernel Semaphore</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Fast Mutex</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Guarded Mutex</td>
<td>Yes (2K3+)</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Push lock</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Executive Resource</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
High-\textbf{IRQL} Synchronization

- “Normal” kernel code runs at IRQL = PASSIVE\_LEVEL
- Interrupts may intervene at any time
  - Their IRQL is always \textgreater\ DISPATCH\_LEVEL
- Accessing global kernel data must be synchronized
- Solution: raise the CPU IRQL to at least that of the potentially accessing interrupt
  - Fine for one CPU
CPU Synchronization

- IRQL \(\geq\) DISPATCH\_LEVEL
- Thread scheduler cannot interrupt the current thread on the CPU
  - So another thread cannot be scheduled on that CPU
- Another CPU may try to access the shared resource
- Synchronization in this case involves using IRQL changes and spin locks
The Spin Lock

• Synchronization on MP systems uses IRQLs within each CPU and spin locks to coordinate among the CPUs

• A spin lock is just a data cell in memory
  – It is accessed with a test and modify operation, atomic across all processors

• Not exposed (and not needed) to user mode applications
Spin Lock Concepts

• Spin lock acquisition and release routines implement a one-owner-at-a-time algorithm
• Where do spin locks come from?
  – Some are defined in the system
  – Some are automatically associated with I/O devices and related objects
  – Device drivers can create and use additional spin locks
• A spin lock is either free or owned by a specific CPU
• A CPU should own a spin lock that protects shared data before manipulating it
Acquiring a Spin Lock

- IRQL is implicit in the choice of routine
  - `KeAcquireSpinLock` uses IRQL=DISPATCH_LEVEL
  - `KeAcquireSpinLockAtDpcLevel` does not change the IRQL
  - `KeSynchronizeExecution` and interrupt dispatcher use `SyncIrql` found in interrupt object
  - `ExInterlockedXxx` routines use IRQL=HIGH_LEVEL

- Spin locks should not be requested if already owned
  - Causes a deadlock!
Spinlock “Busy Wait”

• Busy waiting causes repeated read attempts at the spinlock
• CPU is running continuously
• On Pentium 4 or newer
  – A new machine instruction (pause/yield) is inserted into such busy wait loops
  – Less power is consumed
  – Used as a hint to the CPU of a hyper threaded core, causing more CPU time to be allocated to the other core
Queued Spin Locks

- From Windows XP
- Efficient version of dispatch-level spin locks (IRQL DISPATCH_LEVEL)
- Ensures that the spin lock is acquired on a first-come first-serve CPU basis
- Check `KeAcquireInStackQueuedSpinLock` and `KeReleaseInStackQueuedSpinLock` in the WDK
- Useful when it’s likely that more than one CPU will try to acquire the spin lock
Passive-Level Interrupt Service Routines

• Starting with Windows 8, a driver can use the `IoConnectInterruptEx` routine to register a passive-level Interrupt Service routine (ISR)
  – This can be use only for devices that can communicate in low IRQL
  – Memory mapped I/O
• For example I²C
• This enable slow interrupt handling since the ISR execution time is very limited
• More Information [here](#)
System Worker Threads

• Threads created by Windows that reside under the System process
• Device drivers may create additional threads for their own use (by calling the function `PsCreateSystemThread`)
• These threads always execute in kernel mode
• Work Item
  – A request by the system or a device driver to run a routine in IRQL 0
  – Carried out by a system worker thread
  – Requested by calling `ExQueueWorkItem` or `IoQueueWorkItem`
Worker Threads Creation

• Worker threads are created in three priority levels
  – Delayed worker threads (priority 12)
    • Process work items that are not considered time critical and can have their stack paged out while waiting for requests
  – Critical worker threads (priority 13)
    • Process time critical work items
    • On Server systems have their stack in memory at all times
  – Hypercritical worker thread (priority 15)
    • A single thread used to free terminated thread objects

• The number of critical and delayed worker threads is changing according to work item load
  – Initial number is based on OS version and the registry
Windows Global Flags

- Global flags maintained by the OS in a global variable named NtGlobalFlags, used for debugging, tracing and other information
  - Initialized from the registry at boot time
  - HKLM\System\CCS\Control\Session Manager in the GlobalFlags value
- Each process can maintain its own global flags settings
- Flags can be changed with the Gflags.exe utility (Windows SDK and Debugging Tools)
Global Flags

[Image of Global Flags settings window with various options and checkboxes for different functionalities, such as 'Stop on exception', 'Enable heap allocation checking', 'Enable heap deallocation checking', and options for kernel special pool tags and object reference tracing.]
Kernel Event Tracing

• The kernel provides a mechanism to record data by kernel and user mode components, and exposed through Event Tracing for Windows (ETW)

• ETW entities
  – Provider
    • Produces event data
    • Defines a GUID for its event classes and registers them with ETW
  – Controller
    • Starts or stops logging sessions
  – Consumer
    • Selects trace sessions to receive data, either in real-time or from a log file
Example ETW Providers

- Disk I/O (disk class driver)
- File I/O (file system drivers)
- Hardware configuration (Plug & Play manager)
- Image Load/Unload (system image loader in the kernel)
- Page Faults (memory manager)
- Process create / delete (process manager)
- Registry activity (configuration manager)
- Context switches (kernel dispatcher)
- System calls (kernel dispatcher)
- Interrupts (kernel dispatcher)
Using ETW

• Can use the “Data Collector Sets” node in Performance Monitor
  – Can select provider(s)
  – Can start/stop a tracing session

• To generate a human readable output, use the `tracerpt.exe` command line tool
  – Can generate (e.g.) a CSV file, viewable in a text editor or MS Excel
Instrumentation Design Workflow

1. Design
   Create Instrumentation Manifest (ECMangen)

2. Instrumentation Definition
   XML Manifest
   CTRPP / MC preprocessors
   .rc
   .c
   .h

3. Generate Code

4. Instrument
   Instrument provider APIs

5. Build
   New Provider
TraceLogging

• The new Windows 10 event tracing framework for user-mode applications and kernel-mode drivers
  – An abstraction on top of ETW
  – Tracing an event is as simple as an API call
  – Events are self-describing and do not require any additional binaries
    • All the metadata about the event is recorded in the event
  – Activities inside a single process can be easily expressed through start and stop events
  – TraceLogging is compatible with existing instrumentation support
    • The new ETW APIs continue to support the old providers
  – TraceLogging offers the same event tracing API for Windows 10, Xbox One, and Windows 10 Mobile
  – TraceLogging APIs are accessible from C, C++, .NET, and Windows Runtime
WINDOWS X64 & WOW 64
64?

• No more out of memory exceptions (address space)
  – 32 bit = 2GB + 2GB or 3GB + 1GB
  – 64 bit = 8TB + 8TB

• Actually x64 == 48bit today

• Canonical Form:
  – Bits 48-63 is signed extended of bit 47

8TB = 8192GB

2GB
x86-64 (A.K.A x64, x86_64 and amd64)

• The original specification was created by AMD
• Has been implemented by AMD, Intel, VIA, & others
• Fully backwards compatible with 16-bit and 32-bit x86 code
  – Existing x86 executables run with no compatibility or performance penalties
  – existing applications that are converted to x64 to take advantage of new features of the processor may achieve performance improvements
Windows 64

• First versions Server 2003/XP 64 (NT 5.2)
• Server 2008R2 is the first 64-bit only OS from MS
• No more VDM & Wow32
  – Virtualization (Hyper-V, VMWare, Oracle Virtual Box)
• Old (and new) x86 32bit applications run with no performance penalty
• 64bit development feels like 32bit development
Windows PE32+ File Format

- The **Portable Executable** is the format for all DLL, EXE, SYS, OCX, and even PDB files.
- It has been changed to support x64 PE:
  - The file header Machine type has `IMAGE_FILE_MACHINE_[AMD64]||[IA64]||[ARM64]`.
- The optional header magic number determines whether an image is a PE32 or PE32+ executable.
- PE32+ images allow for a 64-bit address space while limiting the image size to 2 gigabytes.
dumpbin /headers c:\windows\system32\notepad.exe
dumpbin /headers c:\windows\SysWow64\notepad.exe

DEMO
**Wow64**

- Allows execution of Win32 binaries on 64-bit Windows
  - Wow64 intercepts system calls from the 32-bit application, Converts 32-bit data structure into 64-bit aligned structures and Issues the native 64-bit system call
- The **IsWow64Process** function can tell a 32-bit process if it is running under Wow64, in .NET:
  - `Environment.Is64BitOperatingSystem && !Environment.Is64BitProcess`
- Address space is 2GB or 4GB (if image is linked with the `/LARGEAADDRESSAWARE`)
  - Use **Editbin** to edit the flag after compilation/linking
- Performance
  - On x64, 32 bit instructions executed by hardware
  - On IA64 (obsolete), instructions have to be emulated
- Device drivers must be native 64 bit
Wow64 Restrictions

• A 64 bit process cannot load a 32 bit DLL and vice versa
  – Except resource-only DLLs, which can be loaded cross-architecture

• Some APIs are not supported by Wow64 processes
  – E.g. ReadFileScatter, WriteFileGather, AWE functions
Wow64 Architecture

• Wow64.dll – core emulation infrastructure and thunks for Ntoskrnl.exe functions; exception dispatching

• Wow64win.dll – intercepts GUI system calls for Win32k.sys

• Wow64Cpu.dll – manages thread contexts, supports mode-switch instructions
Wow64 File System Redirection

• System directories names have not changed in 64 bit Windows (e.g. `\Windows\System32` contains native 64 bit images)

• 32 bit applications must use their own directories
  – `\Windows\System32` maps to `\Windows\Syswow64`
  – 32 bit apps installed in `\Program Files (x86)`
  – 64 bit apps installed in `\Program Files`

• Some directories are not redirected
Wow64 Registry Redirection

• Components trying to register as 32 bit and 64 bit will clash
• 32 bit components are redirected to the Wow64 registry node (Wow6432Node)
  – HKLM\Software
  – HKEY_CLASSES_ROOT
  – HKCR\Software\Classes
• New flags for Registry APIs allow access to the 64 bit or 32 bit nodes
  – KEY_WOW64_64KEY – open a 64 bit key
  – KEY_WOW64_32KEY – open a 32 bit key
Wow64 Registry Reflection

• Some keys used with COM (Component Object Model) are mirrored
  – HKLM\Software\Classes
  – HKLM\Software\Ole
  – HKLM\Software\Rpc
  – HKLM\Software\COM3
  – HKLM\Software\EventSystem

• HKLM\Software\Classes\CLSID is mirrored only for LocalServer32 keys only
  – i.e. Out of process servers (can be activated by 32 bit clients)
  – InProcServer32 keys are not mirrored
Avoid Potential Problems

• No NTVDM - 16-bit binaries are no longer supported
  – Except for some kinds of installers
• ALL drivers must be compiled and target 64-bit Windows
• 32-bit applications install to
  – "C:\Program Files (x86)"
• While 64 applications use "C:\Program Files"
  – Automatically handled by using
    • `SHGetFolderPath(CSIDL_PROGRAM_FILES)`
    • `Environment.GetFolderPath(Environment.SpecialFolder.ProgramFiles)`
  – Use `GetSystemWow64Directory` to get the SysWow64 (System32 of 32 bit applications)
IPC Wow64 and 64 bit processes

- It just works!!!
  - Sharing Kernel Objects (Handles)
  - Using HWND and Windows Messages
  - Using RPC, .NET Remoting, WCF, Sockets
  - D/COM (with x86/x64) proxy/stub
  - CreateProcess can create 32/64 bit process
  - Shared Memory (Content has to be pointer size independent)
Programming x64

- **Programming Guide for 64-bit Windows**
- 64bit development feels like 32bit development
- For native developer:
  - Pointer size has changed
  - Data alignment is even more important
  - Packing can waste a lot of memory
  - Pointer truncating is a real headache
  - Polymorphic Types and API helps targeting both 32 and 64 bit
# X86/X64 Programming Model

<table>
<thead>
<tr>
<th>Data model</th>
<th>char</th>
<th>short</th>
<th>int</th>
<th>long</th>
<th>long long</th>
<th>pointers</th>
<th>OSes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILP32</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>32</td>
<td>64?</td>
<td>32</td>
<td>Windows, Linux, Solaris, OS X, <em>BSD, AIX, HP-UX, other UN</em>Xes</td>
</tr>
<tr>
<td>LLP64</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>32</td>
<td>64</td>
<td>64</td>
<td>Windows</td>
</tr>
</tbody>
</table>

`sizeof(size_t) != sizeof(int)` and `sizeof(size_t) != sizeof(long)`
Size Does Matter

• Pointers and variable that may hold pointer data (such as WPARAM & LPARAM) has changed to 64 bit
• Non-pointer types such int, long, DWORD, stay the same (32 bit)
• If you care about the exact size use:
  – DWORD32, DWORD64, INT32, INT64, LONG32, LONG64, UINT32, UINT64, ULONG32, ULONG64, POINTER_32, POINTER_64
• Polymorphic size types:
  – Void *, INT_PTR, UINT_PTR
Windows Header File are Type Polymorphic

- **LRESULT CALLBACK WndProc**
  ```
  HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam
  ```

- **From ...\Include\shared\minwindef.h:**
  - /* Types use for passing & returning polymorphic values */
  - typedef UINT_PTR WPARAM;
  - typedef LONG_PTR LPARAM;
  - typedef LONG_PTR LRESULT;

- **From ...\Include\shared\basetsd.h:**
  - Next Page
Windows Header File are Type Polymorphic

- From `\Include\shared\basetsd.h`:
  - `#if defined(_WIN64)`
    - `typedef __int64 INT_PTR, *PINT_PTR;`
    - `typedef unsigned __int64 UINT_PTR, *PUINT_PTR;`
    - `typedef __int64 LONG_PTR, *PLONG_PTR;`
    - `typedef unsigned __int64 ULONG_PTR, *PULONG_PTR;`
    - `#define __int32 64 __int64`
  - `#else`
    - `typedef _W64 int INT_PTR, *PINT_PTR;`
    - `typedef _W64 unsigned int UINT_PTR, *PUINT_PTR;`
    - `typedef _W64 long LONG_PTR, *PLONG_PTR;`
    - `typedef _W64 unsigned long ULONG_PTR, *PULONG_PTR;`
    - `#define __int32 32 __int32`
  - `#endif`
Predefined Macros

The compiler defines these macros:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>_WIN64</td>
<td>A 64-bit platform.</td>
</tr>
<tr>
<td>_WIN32</td>
<td>A 32-bit platform. This value is also defined by the 64-bit compiler for backward compatibility.</td>
</tr>
<tr>
<td>_WIN16</td>
<td>A 16-bit platform</td>
</tr>
<tr>
<td>_M_IA64</td>
<td>Intel Itanium platform</td>
</tr>
<tr>
<td>_M_IX86</td>
<td>x86 platform</td>
</tr>
<tr>
<td>_M_X64</td>
<td>x64 platform</td>
</tr>
</tbody>
</table>
Data Alignment

• Allocate on aligned boundaries
  – Use `__declspec(align)` to align data to specific boundary
  – Example: `__declspec(align) int abc;`

• Use the C++ 11 `alignas` new keyword

• Can use dynamically with `aligned_malloc()` family of runtime routines

• x64 handles alignment faults in hardware
Be ware of Padding (Packing Difference)

struct node {
    char *l;
    char s;
    struct tnode *prev;
    int i;
    struct node *next;
}

32bit

64bit
Smart Reordering Types Can Save a lot of Space

```c
struct node {
    char *l;
    char s;
    struct tnode *prev;
    int i;
    struct node *next;
} // Original

struct node {
    char *l;
    struct tnode *prev;
    struct node *next;
    struct node *next;
} // Tuned
```
How to Correctly Cast a Pointer to int in a 64-bit Application

• The most general answer is – in no way
  – Putting a 64-bit pointer into a 32-bit variable causes cutting of high-order bits and therefore incorrect program behavior

• However, there is a specific case when you may store a pointer in 32-bit types
  – HANDLE, HWND, HBRUSH, Etc.

• Use the following helper function to safe convert:
Helper Functions

- `void * Handle64ToHandle( const void * POINTER_64 h )`
- `void * POINTER_64 HandleToHandle64( const void *h )`
- `long HandleToLong( const void *h )`
- `unsigned long HandleToUlong( const void *h )`
- `void * IntToPtr( const int i )`
- `void * LongToHandle( const long h )`
- `void * LongToPtr( const long l )`
- `void * Ptr64ToPtr( const void * POINTER_64 p )`
- `int * PtrToInt( const void *p )`
- `long * PtrToLong( const void *p )`
- `void * POINTER_64 PtrToPtr64( const void *p )`
- `short * PtrToShort( const void *p )`
- `unsigned int * PtrToUint( const void *p )`
- `unsigned long * PtrToUlong( const void *p )`
- `unsigned short * PtrToUshort( const void *p )`
- `void * UIntToPtr( const unsigned int ui )`
- `void * ULongToPtr( const unsigned long ul )`
X64 Managed Applications

• You can choose to target Any-CPU, X86 or x64
• For assemblies (DLL) choose any-cpu unless you have a specific native code dependency or the assembly contains native code (C++/CLI)
• For executable, use specific hardware if you depend on native implementation (native DLL, P/Invoke, etc.)
  – Otherwise use Any-CPU
• The CorFlags.exe utility can show/change target CPU architecture
.NET Framework

• On x64 System there are two versions of each .NET framework (from framework 2.0)
  – A 32 bit framework (C:\Windows\Microsoft.NET\Framework)
    • V1.0, V1.1, V2.0, V3.0, V3.5 V4.0, V4.5, V4.5.2, V 4.5.1, V4.6, V4.6.1, V4.6.2
  – A 64 bit framework (C:\Windows\Microsoft.NET\Framework64)
    • V2.0, V3.0, V3.5 V4.0, V4.5, V4.5.2, V 4.5.1, V4.6, V4.6.1, V4.6.2
  – The GAC and NGEN also have different locations for Any-CPU, x86, x64
Which CLR Host My Application

- On x64 System:
IIS Settings

- .NET Sites and services are hosted in IIS under the application pool process
  - Choose the application pool bit-ness to match your needs
.NET X64 and Interop

• `System.IntPtr` & `System.UIntPtr` are polymorphic types

• Structures and Classes Layout has to be matched to the Native x86/x64 structure
  – Use polymorphic .NET types and/or the `[StructLayout]` attribute

• Be aware of x64 when you use Windows resources such as registry, shared memory, file-system
X64 Summary

• We learned about Windows x64
  – The new hardware architecture
  – Windows 32 on Windows 64
  – The new PE32+ file format
  – Native development considerations
  – X64 .NET framework
Transaction

• Atomic **transaction** is a reliability principle
  – It is an all or nothing operation
    • the system will be in a good state after the execution of the transaction

• Transaction follows the **ACID** criteria:
  – Atomicity – all or nothing
  – Consistency – Application moves from one consistent state to another consistent state
  – Isolation – No transaction can “see” or “interfere” with another
  – Durability – Committed transaction will be committed!!!

• Transaction timeouts even solve deadlocks
Distributed Atomic Transaction

- A **transaction with many parties**, sometime across process and machine boundaries
  - **Two phase commit** protocol

- Transaction Roles
  - Transaction Manager (TM)
    - **Microsoft Transaction Server**, Kernel Transaction Manager
    - Managers communicate between themselves
  - Resource Manager (RM)
    - The client of the TM, responsible to the state of the resource
Two Phase Commit

**Transaction Manager**
- Send Prepare to all RMs
- Force write “Commit” record to TM log
- Lazy write “Committed” record to TM log

**Resource Manager**
- Prepare
- Prepared
- Commit
- Committed
-lazy write “Committed” record to RM log and Release locks

**Flow**
- Send Prepare to all RMs
- Prepare
- Prepared
- Commit
- Committed

**Steps**
1. Send Prepare to all RMs
2. Prepare
3. Prepare and Force write “Prepared” record to RM log
4. Prepared
5. Commit
6. Lazy write “Committed” record to RM log
7. Committed
8. Lazy write “Committed” record to TM log
9. Lazy write “Committed” record to TM log
10. Release locks
Windows KTM

• A kernel based implementation of a distributed transaction system
  – Can participate in any MTS based transaction
  – Provide API to create your own Resource Manager
  – Two built in Kernel RMs:
    • Transactional NTFS
    • Transactional Registry
  – Play nice with the kernel handle concept
Creating Transaction

• **CreateTransaction**
  – Returns handle to a kernel transaction object
  – You can set the transaction timeout value
  – You can ask for a non distributed transaction
  – For distributed transaction:
    • you need to add the DTC’s SID in the Security Descriptor that you can set using the `lpTransactionAttributes`
  – CloseHandle closes the transaction handle
    • If you close the last handle without committing the transaction, the KTM will rollback the transaction

• **OpenTransaction**, **GetTransactionId**
  – Opens transaction handle from transaction Id
Commit & Rollback

• To commit a transaction call \texttt{CommitTransaction}
• To Rollback a transaction call \texttt{RollbackTransaction}
  – Or close the handle without any commit
• To successfully commit a transaction, no party should call to Rollback and the transaction should be committed on time
Kernel Resource Managers

• KTM provides all you need to develop your own resource manager

• However in most cases you will use the KTM with one of the two existing managers
  – The Transactional NTFS
  – The Transactional Registry

• We will start with the registry, and then we will dive into transactional NTFS
Transactional Registry

- **RegCreateKeyTransacted**
  - Creates a registry key and associates it with a transaction
    - If the key already exists, the function opens it
- **RegDeleteKeyTransacted**
  - Deletes a sub-key and its values from the registry as a transacted operation
- **RegOpenKeyTransacted**
  - Opens the specified registry key and associates it with a transaction
- Having registry keys under a transaction means that all registry operations will be committed or rolled back
Transactional NTFS

• With transactional NTFS you can create a distributed transaction with a scope that includes a database, communication channels and the file system
  – Using TxNTFS you can update a UNC entry in a database while creating this file – all in one atomic transaction

• There are two API types for TxNTFS
  – Old API that takes KTM handle
    • ReadFile, WriteFile, ...
  – New API with the *Transacted, that takes a transaction handle
    • MoveFileTransacted
TxNTFS Isolation

- TxNTFS is optimized for success
  - The Resource Manager makes the changes
  - Changes are hidden from non-transactional reader
  - At commit the Resource Manager make the changes available
  - Non transactional writer can’t create a handle if the file is already open for transactional writer and vice versa
  - Transactional Reader can choose to see the dirty or committed state
    - `CreateFileTransacted(...pusMiniVersion)`

<table>
<thead>
<tr>
<th>TXFS_MINIVERSION_*</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMITTED_VIEW</td>
<td>The view of the file as of its last commit</td>
</tr>
<tr>
<td>DIRTY_VIEW</td>
<td>The view of the file in the transaction (modify)</td>
</tr>
<tr>
<td>DEFAULT_VIEW</td>
<td>A transaction that is modifying the file gets the dirty view. A transaction that is not modifying the file gets the committed view.</td>
</tr>
</tbody>
</table>
Distributed Transaction

- Kernel transaction can take part with DTC based transaction
  - Combine TxNTFS & Transactional Registry with SQL Server, Oracle DB, MSMQ, IBM MQSeries
  - The famous scenario of using the file system to store large files and keeping its path in the DB becomes reliable

- Instead of creating kernel transaction handle
  - Ask the handle from the DTC
    - Query `IKernelTransaction` from the `ITransaction` interface
    - `HRESULT GetHandle(HANDLE * pHandle);`
IKernelTransaction (Native)

#include "stdafx.h"
#include <Windows.h>
#include <ktmw32.h>
#include <TxDtc.h>
#include <xolehlp.h>
#pragma comment (lib, "ktmw32.lib")
#pragma comment (lib, "xolehlp.lib")

int _tmain(int argc, _TCHAR* argv[])
{
    CoInitialize(NULL);
    ITransactionDispenser *pITransactionDispenser;
    ITransaction *pITransaction;
    IKernelTransaction *pKernelTransaction;
    HANDLE hTransactionHandle;
HRESULT hr = DtcGetTransactionManagerEx(NULL, NULL,
   IID_ITransactionDispenser, OLE_TM_FLAG_NONE,
   NULL, (void**) &pITransactionDispenser);

hr = pITransactionDispenser->BeginTransaction(
   NULL, ISOLATIONLEVEL_READCOMMITTED,
   ISOFLAG_RETAIN_BOTH, NULL, &pITransaction);

hr = pITransaction->QueryInterface(IID_IKernelTransaction,
   (void**) &pKernelTransaction);

hr = pKernelTransaction->GetHandle(&hTransactionHandle);
HANDLE hAppend = CreateFileTransacted(L"test.txt", FILE_APPEND_DATA, FILE_SHARE_READ, NULL, OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL, hTransactionHandle, NULL, NULL);

DWORD nWritten;
wchar_t text[] = L"Transaction Rocks!";
WriteFile(hAppend, text, sizeof(text), &nWritten, nullptr);
IKernelTransaction (Native)

```c
hr = pITransaction->Commit(FALSE, XACTTC_SYNC_PHASEONE, 0);

CloseHandle(hAppend);
pKernelTransaction->Release();
pITransaction->Release();
pITransactionDispenser->Release();
CoUninitialize();
return 0;
```

KTM & .NET

• .NET does not support KTM out of the box
  – However we can Interop to get `IKernelTransaction`
  • Using `System.Transaction` interop mechanisms
    – Once we have the `IKernelTransaction.GetHandle` method, we can use a KTM transaction handle

• Interop to `CreateFileTransacted` can provide a transacted file handle
  – .NET stream supports creating a stream from a Win32 file handle

• Using `System.Transaction` + KTM enables strong scenarios such as distributed WCF based transaction with Databases and TxNTFS.
KTM Summary

Transaction == Retry Boundaries

- Windows NT 6.x provides new transaction capabilities such as KTM, TxRegisry, TxNTFS
- KTM can take part in DTC transaction
- Using System.Transaction + Interop, we can have a KTM transaction within .NET code
Enhanced I/O in NT 6.x

- NT 6.X provides new I/O capabilities
  - I/O Priority
  - I/O Bandwidth reservation
  - Better (asynchronous) I/O cancelation
  - A new ability to mount a Virtual Hard Drive
  - A new ability to boot from a VHD file
I/O Priority

• In Windows I/O is bound to the thread that issues the API
  – Each thread has a list of pending I/O Request Packets (IRP)

• Windows NT 6.x has added I/O priority level to the IRP, stored in the Flags field. Drivers should support this flag
  – There are 5 levels:
    • Critical, High, Normal, Low, Very Low
    • High not implemented
    • Critical is only for use by memory manager

• Windows execute at least one of Low or Very Low I/O request every second
I/O Priority APIs

• There are several methods to set the I/O priority:
  – Setting to thread or process to “Background Mode”
    • `SetThreadPriority`, `SetPriorityClass`
      – `THREAD_MODE_BACKGROUND_BEGIN`
      – `THREAD_MODE_BACKGROUND_END`
      – `PROCESS_MODE_BACKGROUND_BEGIN`
      – `PROCESS_MODE_BACKGROUND_END`
  – Providing a hint for the file handle
    • `SetFileInformationByHandle`
      – With `FILE_IO_PRIORITY_HINT_INFO`
  – Using Multimedia Class Scheduling Service
  – Using Kernel call: `IoSetIoPriorityHint`
I/O Bandwidth Reservation

• Streaming applications can request I/O bandwidth guarantees
• Specified on individual files
• I/O system reports back to application
  – Optimal transfer size
  – Number of outstanding I/Os they should maintain
Reserving Bandwidth for Streaming

- An application that requires throughput of 200 bytes per second from the disk would make the following call:

  ```c
  SetFileBandwidthReservation( hFile, 1000, 200, FALSE, &transferSize, &outstandingRequests );
  ```

- **TransferSize**: The minimum size of the I/O request
- **OutstandingRequests**: The number of outstanding TransferSize chunks. For maximum throughput
- To get the current setting
  - call the [GetFileBandwidthReservation](#) API
I/O Cancellation Support

• Until NT 6.x, opens could not be cancelled
  – Example: You browse to an off-line network share in a File Save dialog and hang for the duration of the network timeout

• In NT 6.x, opens and other synchronous I/O can be cancelled
  – CancelSynchronousIo cancels a pending synchronous I/O issued by another thread
  – CancelIoEx permits canceling I/Os from any thread
    • The old CancelIo could only cancel all I/Os issued by the calling thread, hence only for asynchronous operations

• Windows 7/2008 file open/save dialogs all implement cancellation
Summary

• Windows is an object based system managed by the Object Manager executive component
• Synchronization primitives exist for inter processor and inter thread scenarios
• The scheduler only works when IRQL is less than DISPATCH_LEVEL
• Transaction provides retry boundaries
• Windows 6.x solved and enhanced I/O operations
Module 3

MANAGEMENT & DIAGNOSTICS MECHANISMS
Agenda

• The Registry
• Windows Services
• Windows Management Instrumentation
• Windows Error Report
• Summary
The Registry

• Hierarchical repository of system / user configuration data
  – Some stored in files, some built dynamically and stored in memory

• Access
  – REGEDIT.EXE
    • Originally written for Windows 95
    • Enhanced significantly in Windows 2000 and XP
  – Programmatically
    • APIs in Win32 and the Kernel

• Activity
  – Can watch with Process Monitor from SysInternals
Registry: The Hives

- **HKEY_LOCAL_MACHINE (HKLM)**
  - Contains machine specific configuration (not user related)
- **HKEY_CURRENT_USER (HKCU)**
  - Contains per-user information
- **HKEY_CLASSES_ROOT (HKCR)**
  - Contains file extension associations and COM (Component Object Model) registration
- **HKEY_USERS**
  - Contains sub-keys for each user ever logged on to the system
- **HKEY_CURRENT_CONFIG**
  - Contains current hardware configuration
- **HKEY_PERFORMANCE_DATA**
  - Contains performance counters data
  - Not really part of the registry
HKEY_CURRENT_USER

- Contains user specific information
- Maintained in a file named NtUser.Dat stored under \Documents and Settings\<user name>
- Sub-keys
  - AppEvents – sound/event associations
  - Console – command window settings
  - Control Panel – screen saver, desktop scheme, keyboard and mouse settings, etc.
  - Environment – environment variable definitions
  - Keyboard Layout
  - Network – network drives mappings
  - Printers – printers connections settings
  - Software – user specific software preferences
HKEY_CLASSES_ROOT

- Contains file extension association (used by the shell – Explorer.Exe) and COM servers registration (classes, interfaces, type libraries, applications, etc.)
- Actual data comes from two sources
  - HKLM\Software\Classes
  - Per user class registration in HKCU\Software\Classes stored in \Users\<username>\LocalSettings\Application Data\Microsoft\Windows\UsrClass.dat
    - The addition of per-user classes is new to Windows 2000
- Sub-keys
  - CLSID – COM classes registration
  - Interface – COM interfaces proxy\stub registration
  - TypeLib – Registered type libraries
  - AppID – COM server applications registration (not the same as COM+ applications)
HKEY_LOCAL_MACHINE

- Contains information relevant to the machine regardless of the logged on user
- Sub-keys
  - Hardware – contains device descriptions detected during the boot process
  - SAM – contains local users and group information
    - Actually a link to HKLM\Security\SAM
  - Security – system-wide security policy and user rights assignments
  - Software – System-wide configuration for system boot as well as third party software settings (directories, passwords, etc.)
  - System – system-wide configuration needed to boot the system, such as device drivers and Win32 services to load
Hive Store Path

• Non volatile registry data is stored in files
  – HKLM\System
    • System32\Config\System
  – HKLM\SAM
    • System32\Config\SAM
  – HKLM\Security
    • System32\Config\Security
  – HKLM\Software
    • System32\Config\Software
# Registry Data Types

- **Most useful types:**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REG_SZ</td>
<td>Null terminated, fixed length, Unicode string</td>
</tr>
<tr>
<td>REG_MULTI_SZ</td>
<td>Multiple null-terminated Unicode strings</td>
</tr>
<tr>
<td>REG_EXPAND_SZ</td>
<td>Variable length Unicode string that can hold embedded environment variables</td>
</tr>
<tr>
<td>REG_BINARY</td>
<td>Arbitrary length binary data</td>
</tr>
<tr>
<td>REG_DWORD</td>
<td>32 bit number</td>
</tr>
<tr>
<td>REG_DWORD_BIG_ENDIAN</td>
<td>32 bit number in big endian format (high byte first)</td>
</tr>
<tr>
<td>REG_LINK</td>
<td>Symbolic link to another key</td>
</tr>
<tr>
<td>REG_QWORD</td>
<td>64 bit number</td>
</tr>
</tbody>
</table>
Registry Access APIs

• User mode (Windows API)
  – `RegCreateKeyEx`, `RegOpenKeyEx`, `RegCloseKey`
  – `RegDeleteKey`
  – `RegSetValueEx`, `RegQueryValueEx`,
  – `RegEnumKeyEx`, `RegEnumValue`

• Kernel mode
  – `ZwCreateKey`, `ZwOpenKey`, `ZwClose`
  – `ZwDeleteKey`
  – `ZwSetValueKey`, `ZwQueryValueKey`
  – `ZwEnumerateKey`, `ZwEnumerateValueKey`
Windows Services

• A Service is a program that provides some service without being tied to the logged on user
  – A service may run without any user logging in

• Service examples
  – IIS service (listening on HTTP port 80, etc.)
  – SQL server (listening for database requests via some mechanism)
Service Communication

• A service must register itself with the Service Control Manager (SCM)

• The SCM is responsible for starting, stopping and otherwise manipulating services
  – Communicate via a named pipe

• A Service Control Application (SCP) may start, stop and otherwise control a service by issuing commands to the SCM
Service Configuration

• A service application must be installed
  – By calling the `CreateService` API
• Inserts a new key into the registry under `HKLM\System\CCS\Services`
  – Technically, the entries in that key correspond to services and device drivers
• Use the Services MMC plug-in to view service only information
Service (& Drivers) Registry Key
Services MMC Snap-in
Service Key Parameters

• **Start**
  – When to start the service / device driver?

• **ErrorControl**
  – What to do if starting fails?

• **Type**
  – Device driver or service?

• **Group, Tag**
  – Used to control load order within a start group

• **ImagePath**
  – The path of the service / device driver
Service Key Parameters

• DependOnService, DependOnGroup
  – Dependencies of the driver / service

• ObjectName
  – Account to run service in

• DisplayName
  – Name displayed in Services MMC

• Description
  – Service description displayed in Services MMC
Service Key Parameters

• **FailureActions**
  – Actions the SCM should take if service process crashes

• **FailureCommand**
  – Command line program to be executed, if appropriate failure action selected

• **Security**
  – Security descriptor of service / driver object
## Service Key Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>SERVICE_BOOT_START (0)</td>
<td>Driver is loaded very early in the boot process</td>
</tr>
<tr>
<td></td>
<td>SERVICE_SYSTEM_START (1)</td>
<td>Driver is loaded during kernel initialization</td>
</tr>
<tr>
<td></td>
<td>SERVICE_AUTO_START (2)</td>
<td>Drivers and services started by the SCM after it loads (Services.exe)</td>
</tr>
<tr>
<td></td>
<td>SERVICE_DEMAND_START (3)</td>
<td>Drivers and services that load on demand using the StartService API</td>
</tr>
<tr>
<td></td>
<td>SERVICE_DISABLED (4)</td>
<td>Driver or service is not loaded</td>
</tr>
<tr>
<td>ErrorControl</td>
<td>SERVICE_ERROR_IGNORE (0)</td>
<td>Error is ignored</td>
</tr>
<tr>
<td></td>
<td>SERVICE_ERROR_NORMAL (1)</td>
<td>A warning is displayed</td>
</tr>
<tr>
<td></td>
<td>SERVICE_ERROR_SEVERE (2)</td>
<td>If last known good isn’t used, the system boots with last known good configuration</td>
</tr>
<tr>
<td></td>
<td>SERVICE_ERROR_CRITICAL (3)</td>
<td>If last known good not used, boot with last known good. If already in last known good – crash the system with a blue screen</td>
</tr>
</tbody>
</table>
## Service Key Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>SERVICE_KERNEL_DRIVER (1)</td>
<td>Kernel device driver</td>
</tr>
<tr>
<td>SERVICE_FILE_SYSTEM_DRIVER (2)</td>
<td>Kernel mode file system driver</td>
<td></td>
</tr>
<tr>
<td>SERVICE_WIN32_OWN_PROCESS (16)</td>
<td>Service running in its own dedicated process</td>
<td></td>
</tr>
<tr>
<td>SERVICE_WIN32_SHARE_PROCESS (32)</td>
<td>Service sharing a process with other services</td>
<td></td>
</tr>
<tr>
<td>SERVICE_INTERACTIVE_PROCESS (256)</td>
<td>Flag indicating service is allowed to create a UI and interact with the logged on user</td>
<td></td>
</tr>
<tr>
<td><strong>ImagePath</strong></td>
<td>Path to .sys or .exe file</td>
<td>If empty, default for drivers is System32\Drivers folder and filename is registry key plus sys extension For services, looks using the PATH environment variable</td>
</tr>
<tr>
<td><strong>ObjectName</strong></td>
<td>Account to run service</td>
<td>May be a specific user, but usually a predefined account, e.g. LocalSystem, LocalService, NetworkService</td>
</tr>
</tbody>
</table>
Service Architecture

Main thread:
1. `StartServiceCtrlDispatcher`
2. `Service control handler`
3. `Main`

Service thread:
1. `RegisterServiceCtrlHandler(Ex)`
2. `Handle client request(s)`

SCM creates:
1. `StartServiceCtrlDispatcher`
Controlling Services

- Service Control Programs, such as the Services MMC use the Windows API to control services
  - `OpenSCManager`
    - Opens a connection to the SCM
  - `OpenService`, `CreateService`
    - Opens a connection to an existing service or installs a new service
  - `StartService`
    - Starts a service
  - `ControlServiceEx`
    - Sends other commands to the service (stop, pause, etc.)
  - `QueryServiceStatus`, `DeleteService`

- Other tools
  - `sc.exe`
    - Command line tool used to control many aspects of services
Service Accounts

• LocalSystem
  – The most powerful on the local computer
  – Use with care

• NetworkService
  – Allows a service to authenticate outside the local computer
  – Has less privileges locally

• LocalService
  – Similar to NetworkService, but can only access network elements accepting anonymous access
Virtual Service Accounts

• Want better isolation than existing service accounts
  – Don’t want to manage passwords

• Virtual accounts are like service accounts:
  – Process runs with virtual SID as principle
    • Can ACL objects to that SID
  – System-managed password
  – Show up as computer account when accessing network

• Services can specify a virtual account
  – Account name must be “NT SERVICE\<service>”
    • Service control manager verifies that service name matches account name
  – Service control manager creates a user profile for the account

• Also used by IIS app pool and SQL Server
Interactive Services Before Vista

• Services are normally created under an invisible Window Station
  – Cannot interact with the user

• A flag can be used to create the service under WinSta0 (the interactive Window Station)
  – May be dangerous for a LocalSystem account service because of its high privileges
Interactive Services NT 6.0 and Above

• On Vista/2008 and up: If a Services wishes to interact with the user
  – The Interactive Service Detection service (UI0Detect.exe) launches an instance of itself in the user’s desktop, displaying a message box
  – “Show me that message” switches the user’s desktop to the Windows service desktop for the interaction
Window Station & Desktop

- A container of desktops, clipboard and atom table
  - A desktop is a container of windows, menus and hooks
Shared Service Processes

• Some services run in their own process
• Some services are sharing a single process
  – Less system overhead of extra processes
  – If one service crashes, it brings down all other services in that process
  – All services running in a shared process run with the same account
SvcHost.Exe

- Generic service host used by Windows
- The services themselves are DLLs
- Must have the DLL path under the *Parameters* subkey
SvcHost.Exe

• Determines which DLLs should share the same process by looking at HKLM\Software\Microsoft\Windows NT\CurrentVersion\Svchost

• Multi strings comprising of services under same service host
SvcHost.Exe

Registry Editor

Name | Type | Data
--- | --- | ---
\(\textbf{(Default)}\) | REG_SZ | (value not set)
\texttt{Axmsvsv} | REG_SZ | AxmsSV
\texttt{DcomLaunch} | REG_SZ | PowerLSM BrokerInfrastructure PlugPlay DeviceInstall DcomLaunch
\texttt{defrogsvc} | REG_SZ | defrogsvc
\texttt{GService} | REG_SZ | vmich/ebean/vmcrv
\texttt{imgsv} | REG_SZ | Xtsvc
\texttt{LocalService} | REG_SZ | ns Win32Service Host w32time EventSystem RemoteRegistry WinHttpAutoProxySvc Sstp
\texttt{LocalServiceAndNoImpersonation} | REG_SZ | TimeBroker, SSDPSrv upnpHost SCardSvc BTH-FSrv QWAVE ftdisup wbISrv W5Service Wcm
\texttt{LocalServiceNetworkRestricted} | REG_SZ | DHCP eventlog AudioSrv wuscvs Lmi-Hosts AppIDSvc WcmSvc wcmvsc homegroupprovider WinReg
\texttt{LocalServiceNoNetwork} | REG_SZ | Dps PLA BFE mpxsvc NcedAutoSetup WwanSvc
\texttt{LocalServicePeerNet} | REG_SZ | PNRPSvc p2pimsvc p2pns PnpAutoReg
\texttt{LocalSystemNetworkRestricted} | REG_SZ | WsSystemHost WsRpc Netman trkvs AudioEndpointBuilder UDFSvcs immon hide
\texttt{netsvcs} | REG_SZ | Aellookupsvc CertPropSvc SCPolicySvc lamanserver gpsvc IKEEXT FastUserSwitching
\texttt{NetworkService} | REG_SZ | CryptSvc nlasvc lanmanworkstation DHCPTermService NdpAgent WinRM WECSVC DHCPSVC
\texttt{NetworkServiceAndNoImpersonation} | REG_SZ | KtmRm
\texttt{NetworkServiceNetworkRestricted} | REG_SZ | PeerDist
\texttt{PeerDist} | REG_SZ | PeerDistSvc
\texttt{Print} | REG_SZ | PrintNotify
\texttt{regsvc} | REG_SZ | RemoteRegistry
\texttt{RPCSS} | REG_SZ | RPEPNative RpcsS
\texttt{rsdsvc} | REG_SZ | sdsvc
\texttt{snpv} | REG_SZ | swpv
\texttt{termsvcs} | REG_SZ | TermService
\texttt{WbiSvcGroup} | REG_SZ | WbiSvc
\texttt{wcsvs} | REG_SZ | WcsPlugInService
\texttt{WsrSvcGroup} | REG_SZ | wersvc
Trigger Start Services

• Introduced in Windows 7
• Service can start with a certain “trigger”
• Cannot be configured using the Services MMC
  – Must call the `ChangeServiceConfig2` API function (or a comparable tool)
• Possible triggers
  – Computer joins a domain
  – Device arrival
  – Firewall port open
  – Group or user policy change
  – IP address availability
  – Network protocol (Windows 8 and later)
  – Custom, based on Event Tracing for Windows (ETW) event
Trigger Start API

SERVICE_TRIGGER trigger = {0};
trigger.dwTriggerType =
    SERVICE_TRIGGER_TYPE_IP_ADDRESS_AVAILABILITY;
trigger.dwAction =
    SERVICE_TRIGGER_ACTION_SERVICE_START;
trigger.pTriggerSubtype =
    (GUID*)&NETWORK_MANAGER_FIRST_IP_ADDRESS_ARRIVAL_GUID;
SERVICE_TRIGGER_INFO info = {0};
info.cTriggers = 1;
info.pTriggers = &trigger;
ChangeServiceConfig2 (hService,
    SERVICE_CONFIG_TRIGGER_INFO, &info);
Scheduled Tasks

• Tasks have always been launched by triggers
• Windows NT 6.0 extends the variety of task triggers and conditions
• Task
  – Actions – what it does
  – Triggers – what triggers it
  – Conditions – what must hold
Task Triggers And Conditions

• Which user to use for launching the task?
• What triggers the task?
  – Schedule (calendar), delay, repeat, or auto-expire
  – At log on, start up, lock, or unlock
  – On an event log entry
• Start only if
  – Computer is idle, on AC power, or connected to a specific network connection
• Do what?
  – Run a program, send e-mail, show a message
Creating Tasks API

```csharp
ITaskServices scheduler = new TaskSchedulerClass();
scheduler.Connect(null, null, null, null);

ITaskFolder rootFolder = scheduler.GetFolder("\\");

ITaskDefinition task = scheduler.NewTask(0);
IExecAction action = (IExecAction) task.Actions.Create(
    _TASK_ACTION_TYPE.TASK_ACTION_EXEC);
```
Task UI
Unified Background Processes Manager (UBPM)

Public API

SCM (services.exe)

Task Scheduler (Schedsvc.dll)

Unified Background Process Manager (UBPM) (Services.exe)

ETW Driven

Services...

Tasks...
UBPM Summary

• Embrace service design goals: performance, security, power consumption
• Does your service have to run all the time?
• Retire “old” Windows XP-style services and consider: Automatic (Delayed Start) services, Trigger Start services, scheduled tasks
• Give your background activity the minimum security privileges
• Strive to minimize background work on idle
Control Sets

• A control set is a set of system configuration parameters
  – HKLM\System\CurrentControlSet is a symbolic link to one of the control sets
    • E.g. ControlSet001, ControlSet002
• One of the sets is the current
• Another set is the Last Known Good
• Values located at HKLM\System\Select
Control Sets in the Registry
Last Known Good

• The SCM is also responsible for saving the current control set as a successful boot
  – HKLM\System\CurrentControlSet
• A successful boot
  – Auto start services started successfully
  – Successful user logon
• Unsuccessful boot
  – An boot/system/auto-start driver has crashed the system
  – A service failed to start and has ErrorControl set to Severe or Critical
The meaning of a successful boot can be changed by a boot verification program

- Pointed to by the registry key HKLM\System\CurrentControlSet\Control\BootVerificationProgram
  - That program’s installer must disable WinLogon’s report of a successful boot by writing 0 to the key HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\ReportBootOk
  - Must call `NotifyBootConfigStatus` to indicate a successful or unsuccessful boot
    - An unsuccessful boot causes a restart
Windows Diagnostics & Management Overview

• What are my system and application doing?
• A short introduction of
  – Performance counters
  – Event Tracing for Windows (ETW)
  – WMI
  – Windows Error Report
• Integrating these diagnostics tools in your application can improve maintenance and reliability
Windows Performance Counters

• Provide information as to how well the operating system or an application, service, or driver is performing

• Numeric information grouped into categories, counters and instances

• Application can provide counter data, or consume this data

• Counter can be accessed from code, scripting & tools
Creating Counters

• Native:
  – Create a manifest
  – Use the CTRPP preprocessing tool
  – PerfAutoInitialize, PerfCreateInstance, PerfSet*Counter*Value

• Managed:
  – System.Diagnostics namespace
  – PerformanceCounterCategory class
  – CounterCreationDataCollection class
  – PerformanceCounter class
Windows Management Instrumentation

• WMI is an implementation of the Web Based Enterprise Management (WBEM) standard
  – An extensible data management and collection facility with the required flexibility to connect to local and remote systems

• WMI entities are object based
  – Properties and methods
  – Events
WMI Namespace View

- WMI CIM Studio (downloadable as part of the [WMI Administrative Tools](https://msdn.microsoft.com/en-us/library/windows/desktop/aa394567(v=vs.85).aspx), fix it for Windows 7 and above)
Windows Error Report

• WER collects mini dumps and other information and sends it to Microsoft
• WER is automatically activated when a process crashes
  – You can create and send a WER report even if your application didn’t crash
• You can also disable a WER report from your application (WerAddExludedApplication)
• Open an account at Winqual to read these reports
• Demos: C:\Program Files\Microsoft SDKs\Windows\v7.0\Samples\winbase\windowserrorreporting
When the problem event occurs, Windows invokes WER.

WER collects the data, builds a report, and prompts the user for consent and sends it to Microsoft (Watson Server).

If the Watson server requests additional data, WER collects the data and prompts the user for consent.

If the application has registered for recovery and restart, WER executes the registered callback functions.

If a response to the problem is available from Microsoft, the user is notified.
Generating Custom Report

Call the `WerReportCreate` to create the report.

Call the `WerReportSetParameter` to set the report parameters.

Call the `WerReportAddFile` to add files and/or the `WerReportAddDump` to add a minidump.

Call the `WerReportSubmit` function to send the report.

Call the `WerReportCloseHandle` to free resources.
Windows Troubleshooting Platform

• WTP provides ISVs, OEMs, and administrators the ability to write troubleshooting packs
  – Used to discover and resolve issues found on the computer
• using WTP you can automate the process of fixing the most common detectable issues
WTP Architecture

Windows Troubleshooting Platform

Process 1
- GUI
- CMD

Windows Troubleshooting Run-time Engine
- Results and Reports

PowerShell Scripts
- Get-Diagnostics
- Update-DiagReport
- Write-DiagProgress

Platform Credentials
- Update-Diagnostics
- Update-DiagRootCause

Process 2
- PowerShell Runtime

Windows Troubleshooting Platform

Identification Metadata
- Contains name and description of the pack, a list of root causes, and the security signature

Detection Scripts
- Checks current configuration with expected configuration

Resolution Scripts
- Sets the current configuration to what is expected

Verification Scripts
- Verifies that current configuration is now what is expected

Localized Resources
- Contains localized display strings in MUI format
Timer **Coalescing**

- Staying idle requires minimizing timer interrupts
- Before, periodic timers had independent cycles even when period was the same
- New timer APIs permit timer coalescing
  - Application or driver specifies tolerable delay
- Timer system shifts timer firing to align periods on a coalescing interval:
  - 50ms, 100ms, 250ms, 1s
Coalesce wait able Timer (>= NT 6.1)

- BOOL SetWaitableTimerEx(
  HANDLE hTimer,
  const LARGE_INTEGER *lpDueTime,
  LONG lPeriod,
  PTIMERAPCROUTINE pfnCompletionRoutine,
  LPVOID lpArgToCompletionRoutine,
  PREASON_CONTEXT WakeContext,
  ULONG TolerableDelay);

Windows Internals
329
UI Coalescable Timer (>=NT 6.2)

• UINT_PTR WINAPI SetCoalescableTimer( HWND hwnd,
  UINT_PTR nIDEEvent,
  UINT uElapse,
  TIMERPROC lpTimerFunc,
  ULONG uToleranceDelay);
Intelligent Timer Tick Distribution

• Before, primary timer interrupt on LP 0 propagated timer to all other LPs
  – Timer interrupt updates process and thread runtimes, checks for thread quantum end
  – Even if LP was idle, it had to service interrupt

• Now, timer system propagates timer only to processors that aren’t idle (also called tick skipping)
  – Non-timer interrupts still wake LP
Wait Chain Traversal (>= NT 6.0)

• WCT enables diagnosing of hangs and deadlocks

• A wait chain is an alternating sequence of threads and synchronization objects
  – A thread waits for an object using synchronization API
  – Once a thread took the object it owns it
  – If a thread waits for a lock that is owned by another thread, we can say that
    the first thread waits for the second thread

• WCT currently supports the following synchronization primitives:
  – Critical sections, Mutexes, GUI (SendMessage), Wait operations
    on processes and threads, ALPC and COM.
Deadlock Detection

BOOL WINAPI GetThreadWaitChain(
    HWCT WctHandle,
    DWORD_PTR Context,
    DWORD Flags,
    DWORD ThreadId,
    LPDWORD NodeCount
    PWAITCHAIN_NODE_INFO NodeInfoArray,
    LPBOOL IsCycle
    );

Windows Internals 333
Analyze Wait Chain

Task Manager

Analyze wait chain

SimpleDeadlock.exe is in deadlock. You can end the process to resolve the deadlock immediately, or you can wait to see if the issue resolves itself.

More about wait chains

End process  Cancel
Summary

• The registry is a hierarchical database used to store machine wide and user based information

• Applications are encouraged to use other mechanisms if possible (e.g. XML files)

• Services are Windows processes that can (among other things) start without an interactive logon

• Windows implements most services as DLLs hosted in a common host (Svchost)

• We have also seen
  – performance counters
  – Windows Error Report
  – Windows Troubleshooting Platform
  – Background processing
  – Timer enhancement
  – Deadlock detection
Agenda

• Processes
• Access Tokens
• Threads
• Thread Scheduling
• Thread Pools
• Synchronization
• Jobs
• Summary
Process

• Management and containment object. Owns:
  • Private virtual address space (2GB/3GB on x86, 8TB / x64 – up to Windows 8, 128 TB Windows 8.1 and above)
  • Working set (physical memory owned by process)
  • Private handle table to kernel objects
  • Security token
    – Has a priority class - Affects all threads running in that process
    – Created from user mode by the Win32 APIs **CreateProcess, CreateProcessAsUser, CreateProcessWithLongonW**
    – Terminated when
      • All threads within that process terminate
      • One of the threads in the process calls **ExitProcess** (Win32)
      • Killed with **TerminateProcess** (Win32)
Process Creation

• Flow of Process Creation
  – Open image file
  – Create kernel Executive Process object
  – Create initial thread
  – Notify CSRSS of new process and thread
  – Complete process and thread initialization
    • Load required DLLs and Initialize
    • DllMain function called with DLL_PROCESS_ATTACH reason
  – Start execution of main entry point (_mainCRTStartup calls: main, WinMain)
Default DLL Search Paths

• Same logic as used by `LoadLibrary`
  – Directory of executable
  – The System directory (`GetSystemDirectory`, e.g. `c:\Windows\System32`)
  – The 16 bit System directory (on 32 bit systems) (e.g. `c:\Windows\System`)
  – The Windows directory (`GetWindowsDirectory`, e.g. `c:\Windows`)
  – The current directory of the process
  – Directories specified in the PATH environment variable

• If safe DLL search mode is not active (XP SP2+) or Windows version is XP SP1 or lower, the current directory is the second item in the above list

• A developer can alter the search path using `SetDefaultDllDirectories`
  – For Windows 7, Windows Server 2008 R2, Windows Vista, and Windows Server 2008, KB2533623 must be installed
  – This provide a protection from most DLL injection related security problems
Creating Process Example

```cpp
#include "stdafx.h"
#include <Windows.h>
#include <iostream>
using namespace std;

int main()
{
    STARTUPINFO si = { sizeof(si) }; 
    PROCESS_INFORMATION pi;

    wchar_t name[] = L"Notepad.exe";
    BOOL created = CreateProcess(nullptr, name, 
        nullptr, 0, FALSE, 0, nullptr, 
        nullptr, &si, &pi);
    if (!created)
    {
        DWORD dwErrorCode = GetLastError();
        wcerr << L"Failed to start process (error number = " 
        << dwErrorCode << ")" << endl;
        return dwErrorCode;
    }

    wcout << L"Process started... pid = " << 
        pi.dwProcessId << L" Main thread id = " << 
        pi.dwThreadId << endl;

    DWORD result = WaitForSingleObject(pi.hProcess, 
        10000);

    if (result == WAIT_TIMEOUT)
        wcout << L"Process still running..." << endl;
    else
        wcout << L"Process exited." << endl;

    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
    return 0;
}
```

//Continue here ...

```cpp
wcout << L"Process started... pid = " << 
    pi.dwProcessId << L" Main thread id = " << 
    pi.dwThreadId << endl;

DWORD result = WaitForSingleObject(pi.hProcess, 
    10000);

if (result == WAIT_TIME_OUT) 
    wcout << L"Process still running..." << endl;
else 
    wcout << L"Process exited." << endl;

CloseHandle(pi.hProcess);
CloseHandle(pi.hThread);
return 0;
```
Access Tokens

• Security context of a process or thread
• A process runs under an access token with the security settings of the specific user
  – CreateProcess (Win32) uses the logged on user
  – CreateProcessAsUser (Win32) uses whatever token is specified in the call (assuming a token for that user can be obtained)
• A thread by default runs under the access token of its process
  – Can assume another token temporarily by using impersonation
Thread

• Instance of a function executing code
  – Owns
    • Context (registers, etc.), 2 stacks (user mode and kernel mode)
    • Optionally, message queue and Windows
    • security token
  – Scheduling state
    • Priority (0-31), State (Ready, Wait, Running)
    • Current access mode (user or kernel)
  – Created using CreateThread, CreateRemoteThreadEx (Win32)
  – Destroyed when
    • Thread function returns to win32 (ntdll.dll)
    • The thread calls ExitThread (Win32)
    • Terminated with TerminateThread (Win32)
    • The thread’s process is exited or terminated
Thread Stacks

• Every user mode thread has two stacks
  – In kernel space (small, 12K (x86), 24K (x64))
    • Resides in physical memory (most of the time)
  – In user space (may be large)
    • By default 1MB is reserved, 64KB committed
    • A guard page is placed just below the last committed page, so that the stack can grow
    • Can change the initial size
      – Using linker settings as new defaults
      – On a thread by thread basis in the call to `CreateThread`, `CreateRemoteThreadEx`
Creating Thread Example

```cpp
DWORD WINAPI Thread(PVOID param)
{
    wcout << L"Thread started id = " << GetCurrentThreadId() << endl;
    PROCESSOR_NUMBER processorNumber;
    GetThreadIdealProcessorEx(GetCurrentThread(), &processorNumber);
    wcout << L"Ideal processor is " << processorNumber.Number << endl;
    for (int i = 1; i <= 10; i++)
    {
        cout << i << endl; Sleep(rand() % 2000);
    }
    return 10;
}

void CreateThreadWithCreateThreadAPI()
{
    DWORD id, code;
    HANDLE hThread = CreateThread(nullptr, 0, Thread, nullptr, 0, &id);
    wcout << L"Thread " << GetCurrentThreadId() << L" free as a thread..." << endl;
    WaitForSingleObject(hThread, INFINITE);
    GetExitCodeThread(hThread, &code);
    wcout << L"Thread finished. result = " << code << endl;
    CloseHandle(hThread);
}
```
void CreateThreadWithCreateRemoteThreadAPI()
{
    DWORD id;
    LPPROC_THREAD_ATTRIBUTE_LIST pAttrlist;
    SIZE_T attrsize;
    //Get the required attribute list size
    InitializeProcThreadAttributeList(nullptr, 1, 0, &attrsize);
    pAttrlist = static_cast<LPPROC_THREAD_ATTRIBUTE_LIST>(_malloca(attrsize));
    memset(pAttrlist, 0, attrsize);
    InitializeProcThreadAttributeList(pAttrlist, 1, 0, &attrsize);
    PROCESSOR_NUMBER processorNumber;  processorNumber.Group = 0;
    processorNumber.Number = 0;
    UpdateProcThreadAttribute(pAttrlist, 0, PROC_THREAD_ATTRIBUTE_IDEAL_PROCESSOR,
        &processorNumber, sizeof(processorNumber), nullptr, nullptr);
    //Create new thread with Ideal processor 0
    HANDLE hThread = CreateRemoteThreadEx(GetCurrentProcess(), nullptr, 0,
        Thread, nullptr, 0, pAttrlist, &id);
    ...
Thread Priorities (Win32 View)

- **Idle Priority Class**
- **Normal Priority Class**
- **High Priority Class**
- **Realtime Priority Class**

**Normal Priorities**

- Above Normal Priority Class
- Normal Priority Class
- Below Normal Priority Class

**Realtime Priorities**

- Realtime Priority Class

Priority:

1 4 6 8 10 13 15 16 24 31
Thread Priorities

• Win32 view
  – Process has a priority class: Idle(4), Below Normal(6), Normal(8), Above Normal(10), High(13), Realtime(24)
  – Thread priority is an offset from that base priority (-2, -1, 0, 1, 2 plus 2 special levels)
    • This is a Win32 constraint
    • More special levels for Real Time range

• Kernel view
  – Thread priority is an absolute value (0-31)
    • The scheduler does not care about processes
Thread Scheduling

- Priority based, preemptive, time-sliced
  - Highest priority thread runs first
  - If time slice (quantum) elapses, and there is another thread with the same priority in the Ready state – it runs
    - Otherwise, the same thread runs again
  - If thread A runs, and thread B (with a higher priority) receives something it waited upon (message, kernel object signaling, etc.), thread A is preempted and thread B becomes the Running thread

- Voluntary switch
  - A thread entering a wait state is dropped from the scheduler’s Ready list

- Typical time slice
  - 20 msec (UP, Professional), 120 msec (UP, Servers)
  - 30 msec (MP, Professional), 180 msec (MP, Servers)

- On an MP system with $n$ CPU cores, $n$ concurrent threads may be running
Thread States

- **Init**
- **Ready**
- **Standby**
- **Running**
- **Waiting**
- **Terminate**

**Transition**

- Preempt
- Preemption, quantum end
- Voluntary switch

- Kernel stack outswap
Thread States Details

- **Init** - Thread has just been created and is initializing
- **Ready** - Thread wants to run but no available CPU exists at the moment
- **Standby** - Represents a short state where context switch occurs
- **Running** - Thread is consuming a CPU and performing work
- **Waiting** - Thread is waiting for some event, kernel object, etc.
  - Does not consume any CPU time
- **Transition** - Thread becoming ready, but kernel stack needs to be read from page file back to RAM
- **Terminate** - Thread has terminated (for whatever reason)
The Scheduler

• Scheduling routines are called when scheduling events occur
  – Interval Timer interrupts checks for quantum end and timed wait completion
  – Hardware interrupt for I/O Completion
  – Changing state of waitable object other threads are waiting on
  – Entering a wait on one or more objects
  – Entering Sleep
The Quantum

- Each thread starts with a quantum number
  - 6 on Home/Professional, 36 on Servers
- Scheduler clock tick is typically
  - 10 msec (uniprocessor)
  - 15 msec (multiprocessor)
- Can determine with `clockres.exe` utility from SysInternals
- Quantum can be modified by using the registry or a Job (see later)
Quantum Accounting

- Pre NT 6.0
  - Thread quantum is decremented by 3 on clock tick
    - This means that the quantum is typically 30msec on Home/Professional and 180msec on Servers
  - Thread quantum is decremented by 1 when wait is complete
    - For threads with priority less than 14
    - Other threads get a replenished quantum
  - When Quantum reaches zero (or less)
    - Preempt thread and restore quantum value
  - Interrupt serviced during a thread’s run
    - Implicitly counts against that thread’s quantum
Quantum Accounting

• NT 6.0 and up
  – Quantum number itself is not used
  – Thread running time is used
    • Interrupt handling is not charged to the thread
Quantum Control

- Registry key: HKLM\SYSTEM\CCS\Control\PriorityControl
- Value: Win32PrioritySeparation

- Short vs. Long
  - 1 = long, 2 = short
  - 0, 3 = default (long for Server, short for Client)

- Variable vs. Fixed
  - 1=boost priority of foreground process, 2=don’t boost
  - 0, 3=default (boost for Client, don’t boost for Server)

- Foreground quantum boost
  - Index into a table

<table>
<thead>
<tr>
<th></th>
<th>Short</th>
<th>Long</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>Fixed</td>
<td>18</td>
<td>18</td>
</tr>
</tbody>
</table>
Performance Options

• System applet in Control Panel
• Programs
  – Short, variable quantum
  – Default for Professional
• Background services
  – Long, fixed quantum
  – Default for Servers
Quantum Boosting

• On a system configured for short, variable quantum
  – The foreground process gets triple quantum (6 clock ticks)
  – For any process with a priority class above Idle
Priority Boosts

• Windows boosts the priority of threads in a number of scenarios
  – Completion of I/O operations
  – After waiting for Executive events or semaphores
  – During waiting for an executive resource
  – After threads in the foreground process complete a wait operation
  – When GUI threads wake up because of windowing activity
  – When a thread is starved
Completion of I/O Request or Wait

- Occurs when an I/O or wait completes
  - Can be specified by driver or Executive
    - `KeSetEvent` (Event, Increment... )
    - `IoCompleteRequest` (Irp, PriorityBoost )

- After a boost, thread runs for one quantum at that priority
  - Then drops one level, runs another quantum
  - Then drops another level, etc., until back to base priority

- Recommended boost values defined in `<ntddk.h>`

```c
#define IO_SERIAL_INCREMENT 2
#define EVENT_INCREMENT 1
#define IO_KEYBOARD_INCREMENT 6
```
Thread Priority Boost and Decay

Priority Boost

- Priority boost upon wait completion
- Preempted (before quantum end)

Time

Base Priority

Run | Wait | Run | Wait | Run

Windows Internals
Foreground Process Wait Boost

• Foreground process
  – The process which contains the thread who is the owner (and creator) of the foreground window

• After a thread running in the foreground process completes a wait on a kernel object

• Receives a boost in the amount of the value set in the registry for foreground priority boost
  – +2 by default
GUI Thread Wakeup

• GUI threads receive a priority boost of 2 when they wake up due to a Window message arriving
• Provided by Win32k.sys
• Improves their chance of running sooner, giving a better responsiveness to the user
Priority Inversion Resolution

- **Priority Inversion**
  - High-priority thread waits on something locked by a lower priority thread which can’t run because of a middle priority thread running

- **Boosts thread to avoid priority inversion**
  - Threads staying in ready state a long time (four seconds) get a big boost to priority 15
    - Get to run for 3 quantum at this special boost
    - Then priority drops to base

- **Technically, starvation avoidance**

- **Implemented by the balance set manager**
  - Scans at most 16 threads per pass
  - Boosts at most 10 threads per pass
Multi Processor Systems

• In a multi CPU / multicore / hyper-threaded system
  – Licensing accounts against physical processors (not logical)
• When choosing a processor for a thread
  – If a physical CPU’s logical processors are idle, one of them would be selected
• NUMA systems should be handled with respect to the nodes
Multiprocessor Support

• Ideal Processor – Soft Affinity
  – Every thread has an ideal processor
  – Default value set in round-robin within each process
    • A random starting value
  – Can Override with SetThreadIdealProcessor or SetThreadIdealProcessorEx
  – On hyper-threaded systems, the next ideal processor selected is from the next physical CPU (not logical)
Hard Affinity

- Threads can run on any CPU unless hard affinity is set for that thread
  - `SetThreadAffinityMask`
  - The mask is a bit mask of allowed CPUs to run that thread
    - Default is process affinity mask, which defaults to all processors
  - Calling `SetProcessAffinityMask` changes priority mask for all threads running under that process
    - And future created thread in that process
- Using hard affinity may result in threads getting less CPU time
**Processor Groups** (>= NT 6.1)

- To support more than 64 logical cores, the scheduling mechanisms and APIs had to change
  - Instead of having one 64 bit flag to provide logical core id, we now have Processor Groups:
Processor Group Characteristics

- Processor allocating to groups happens in boot time
- Each logical processor is assigned to a single group
- All the logical processors in a core, and all the cores in a physical processor, are assigned to the same group if possible
- Physical processors that are physically close to one another are assigned to the same group
- A process can have affinity for more than one group at a time
  - However, a thread can be assigned to only a single group at any time, and that group is always in the affinity of the thread’s process
- An interrupt can only target processors of a single group
- In NUMA architectures, a group can contain processors from one or more nodes
  - All the processors in a node are assigned to the same group whenever possible
Group, Process, and Thread Affinity

- Windows initially assigns each process to a single group in a round-robin manner across the groups in the system.
- The first thread of a process initially runs in the group to which Windows assigns the process.
  - An application can override this default by using the `SetThreadGroupAffinity`.
- Each newly created thread is by default assigned to the same group as the thread that created it.
  - A new thread can be assigned to a group using the `CreateRemoteThreadEx` API.
    - Pass the `PROC_THREAD_ATTRIBUTE_GROUP_AFFINITY` extended attribute together with a `GROUP_AFFINITY` structure.
- Only the system process is assigned a multi-group affinity at startup time.
- A single thread can never be assigned to more than one group at any time.
  - However, a thread can change the group to which it is assigned.
CPU Sets (NT 10.X)

• Provide granular “soft” affinity and information about CPU cores
  – On most systems each CPU Set ID will map directly to a single logical processor
  – Compatible with OS power management
    • Provides information about the energy consumption of a core (EfficiencyClass)
  – Provide information about core parked status
  – Provide information about system reserved CPU Sets
    • Can’t be assign to the application threads
  – A thread affinitized to a given CPU Set will typically execute on one of the processors in its list of selected CPU Set IDs
CPU Sets APIs

- GetProcessDefaultCpuSets
- GetSystemCpuSetInformation
- GetThreadSelectedCpuSets
- SetProcessDefaultCpuSets
- SetThreadSelectedCpuSets
- CPU_SET_INFORMATION_TYPE
- SYSTEM_CPU_SET_INFORMATION
A thread is ready

Y

Is there an idle CPU?

Y

Is ideal CPU idle?

Y

Use it

Y

Use it

N

Is ideal CPU running lower priority thread?

Y

Use it

N

Add it to the ideal CPU’s ready queue

N

Find and use first numbered idle CPU

Y

Use it

N

Is current CPU idle?

Y

Use it

N

Is previous CPU idle?

Y

Use it

N

Is there an idle CPU?

N

Use it

Windows Internals
Waitable Objects

- Kernel supports various waitable objects, which can result in thread pre-emption
  - Process, Thread, Mutex (Mutant), Timer, File, Semaphore, Event (manual reset (notification) and auto reset (synchronization)), Job, Console, Change notification

- Waiting functions
  - Win32: `WaitForSingleObject(Ex)`, `WaitForMultipleObjects(Ex)`, `MsgWaitForMultipleObjects(Ex)`, `SignalObjectAndWait`
  - Kernel: `KeWaitForSingleObject`, `KeWaitForMultipleObjects`
Thread Pools

• Windows 2000 introduced a thread pool mechanism in user mode
  – One thread pool per process
  – Implemented entirely in user mode by ntdll.dll
    – Functions: `QueueUserWorkItem`, `RegisterWaitForSingleObject`
      – No way to prioritized work requests
• Windows Vista and up added support for private thread pools
Private **Thread Pools**

- Internally called Worker Factories
  - Kernel object: TpWorkerFactory
- More than one can be created per process
- Part of the implementation moved to kernel mode
  - Implemented behind the scenes using I/O completion ports
- Functions: `CreateThreadPool`, `CreateThreadPoolWork`, `SubmitThreadPoolWork`, `SetThreadPoolWait`, many others
**MMCSS** – Multimedia Class Scheduler Service (>= NT 6.0)

- Running as “Standard User” implies max thread priority – 15
- Standard Users want to play Audio, Video, Games and capture Audio and Video
  - They want great smooth experience
- **MMCSS** – A service that boost thread priorities and other characteristics
  - All the thread need to do is to announce that it is going to do Multimedia
Tasks & Registry

![Registry Editor]

- MCI Extensions
- MCI32
- MiniDumpAuxiliaryDlls
- MsiCorruptedFileRecovery
- Multimedia
  - SystemProfile
    - Tasks
      - Audio
      - Capture
      - Distribution
      - Games
      - Low Latency
      - Playback
      - Pro Audio
      - Window Manager
- (Default)
- Affinity
- Background Only
- Clock Rate
- GPU Priority
- Priority
- Scheduling Category
- SFIO Priority

Registry Key: `Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Multimedia\SystemProfile\Tasks\Audio`
Using MMCSS

- HANDLE WINAPI `AvSetMmThreadCharacteristics`(
  LPCTSTR TaskName, LPDWORD TaskIndex);

- HANDLE WINAPI `AvSetMmMaxThreadCharacteristics`(
  LPCTSTR FirstTask, LPCTSTR SecondTask,
  LPDWORD TaskIndex);

- BOOL WINAPI `AvRevertMmThreadCharacteristics`(
  HANDLE AvrtHandle);
MMCSS Remarks

• The MMCSS service itself (mmcss.dll) runs as priority 27

• It boosts the multimedia threads into the range registered with the task as appears in the registry
  • This may be even to the real-time priority range

• MMCSS also deals with IO priority and networking throttling
MMCSS Remarks

• The MMCSS service itself (mmcss.dll) runs as priority 27
• It boosts the multimedia threads into the range registered with the task as appears in the registry
  • This may be even to the real-time priority range
• MMCSS also deals with IO priority and networking throttling
Thread Ordering (and timing) Service

• The mission:
  – Getting Real-time accuracy and task ordering under non-real-time O/S such as Windows

• The method:
  – Use MMCSS to get the resources
  – Use Thread Ordering Service to get timing and order

• The thread ordering service controls the execution of one or more client threads.

• It ensures that each client thread runs once during the specified period and in relative order
Using Thread Ordering Group

• The *parent thread* creates one or more thread ordering groups by calling the `AvRtCreateThreadOrderingGroup` API
  – Use this function to specify the period for the thread ordering group and a time-out interval

• Additional client threads call the `AvRtJoinThreadOrderingGroup` function to join an existing thread ordering group
  – Add a thread as a predecessor or successor to the parent thread in the execution order
Waiting to be executed

- Client threads should call `AvRtWaitOnThreadOrderingGroup` in their execution loop.
  - The predecessor threads are executed one at a time in the order that they joined the group, then the parent thread and then all successor.
  - Each time a thread returns to the `AvRtWaitOnThreadOrderingGroup`, the next thread starts.
  - If all threads complete their execution before a period ends, all threads wait until the remainder of the period elapses before any are executed again.
Remarks

• When the client need no longer run as part of the group, it calls the `AvRtLeaveThreadOrderingGroup`

• The parent thread calls the `AvRtDeleteThreadOrderingGroup` to delete the group

• The group is also destroyed if the parent thread does not complete its execution before the period plus the time-out interval elapse
Job Objects

• Kernel object introduced in Windows 2000
• Allows groups of one (or more) processes to be managed as a unit
• System enforces Job quotas and security
  – Total and process CPU time, working sets, CPU affinity and priority class, quantum length (for long, fixed quantums only)
  – Security limits
  – UI limits
Creating a Job Object

• Use the `CreateJobObject` function
  – When the job is created, no processes are associated with the it

• To associate a process with a job, use the `AssignProcessToJobObject` function
  – After a process is associated with a job, the association cannot be broken
  – All new sub-processes will be associated with the Job
    • `CreateProcess` can ask to break-away from the job, if allowed
  – (NT >= 6.2) A process can be associated with more than one job in a hierarchy of nested jobs
Using the Job object

- To determine if a process is running in a job: `IsProcessInJob`
- Kill all processes within the job: `TerminateJobObject`
- Setting limits: `SetInformationJobObject`
- Getting information: `QueryInformationJobObject`
  - the job’s user time and kernel time
  - number of active processes
  - number of total processes created within the job
  - number of total processes terminated within the job
- Getting Notification: `Set Completion Port` using `SetInformationJobObject`
**Nested Jobs** (>= NT 6.2)

- An application can use nested jobs to manage subsets of processes
  - Nested jobs also enable an application that uses jobs to host other applications that also use jobs
Nested Jobs

• Nested jobs is created when a process that is already in a job assign a process to another Job

• The effective limit for child job can be more restrictive than the limit of its parent
  – It cannot be less restrictive

• For a nested job, event messages are sent to every I/O completion port associated with any job in the parent job chain of the job that triggered the message

• Parent Job termination terminates all nested jobs
Summary

• Process is a container of resources for running a program
• Thread is the actual entity scheduled by the kernel to run on a CPU
• Job allows managing process(es) as a single unit while applying constraints
Agenda

• Memory Manager Features
• Virtual Memory vs. Physical Memory
• Virtual Memory Mapping
• Memory Mapped Files and Shared Memory
• Summary
Memory Manager Fundamentals

• Each process sees a virtual address space
  – 2 GB (32 bit), 8 TB (x64 <= NT 6.2), 128 TB (x64 >= 6.3)

• Memory Manager tasks
  – Mapping virtual addresses to physical addresses
  – Using page files to back up pages that cannot fit in physical memory
  – Provide memory management services to other system components
Managing Memory

- Memory is managed by chunks called Pages
- Page size is determined by CPU type
  - A compromise between fine and coarse
- Two page sizes are supported
- Allocations, de-allocations and other memory block attributes (e.g. protection) are always per page

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Small page size</th>
<th>Large page size</th>
</tr>
</thead>
<tbody>
<tr>
<td>x86</td>
<td>4 KB</td>
<td>4 MB (2 MB on PAE systems)</td>
</tr>
<tr>
<td>x64</td>
<td>4 KB</td>
<td>2 MB</td>
</tr>
<tr>
<td>IA-64</td>
<td>8 KB</td>
<td>16 MB</td>
</tr>
</tbody>
</table>
Virtual Page States

- Each page in virtual memory may be in one of three states
- Free
  - Unallocated page
  - Any access causes an Access Violation exception
- Committed
  - Allocated page that can be accessed
  - Has a backup on disk
- Reserved
  - Unallocated page causing Access Violation on access
  - Address range will not be used for future allocations unless specifically requested
- Can view with VMMap (SysInternals)
The Virtual Address Descriptor (VAD) Tree

• A range tree that describes memory ranges used by a process
  – The `!vad` WinDbg extension command and the `vmmmap` SysInternals tool show this tree
• When process reserve or commit a range of pages, a new VAD entry is created
  – The real allocation will occur only on demand
### Viewing Process Address Space

<table>
<thead>
<tr>
<th>Type</th>
<th>32bit notepad</th>
<th>64bit notepad on W8 8TB</th>
<th>64bit notepad on W8.1 128TB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>13396 K</td>
<td>8648 K</td>
<td>8648 K</td>
</tr>
<tr>
<td>Image</td>
<td>5734 K</td>
<td>60148 K</td>
<td>60148 K</td>
</tr>
<tr>
<td>Mapped File</td>
<td>15 K</td>
<td>19912 K</td>
<td>19912 K</td>
</tr>
<tr>
<td>Shareable</td>
<td>58 K</td>
<td>20000 K</td>
<td>20000 K</td>
</tr>
<tr>
<td>Heap</td>
<td>4904 K</td>
<td>4004 K</td>
<td>4004 K</td>
</tr>
<tr>
<td>Managed Heap</td>
<td>12 K</td>
<td>12 K</td>
<td>12 K</td>
</tr>
<tr>
<td>Stack</td>
<td>28 K</td>
<td>2560 K</td>
<td>2560 K</td>
</tr>
<tr>
<td>Private Data</td>
<td>82 K</td>
<td>1416 K</td>
<td>1416 K</td>
</tr>
<tr>
<td>Page Table</td>
<td>418 K</td>
<td>444 K</td>
<td>444 K</td>
</tr>
<tr>
<td>Unusable</td>
<td>2668 K</td>
<td>3272 K</td>
<td>3272 K</td>
</tr>
<tr>
<td>Free</td>
<td>1961728 K</td>
<td>135291355072 K</td>
<td>135291355072 K</td>
</tr>
</tbody>
</table>

**Viewing Process Address Space**

- **32bit notepad**
  - PID: 42282
  - Committed: 2GB

- **64bit notepad on W8 8TB**
  - PID: 3744
  - Committed: 8TB

- **64bit notepad on W8.1 128TB**
  - PID: 42106
  - Committed: 128TB
Locking Memory

• Memory used by processes and kernel may be paged out temporarily

• The memory manager allows locking pages in RAM, preventing paging

• Processes can call `VirtualLock` to request a block of memory to stay in physical memory
  – The memory manager will comply if possible
  – Can increase maximum size with `SetProcessWorkingSetSize(Ex)`

• Device drivers can call `MmProbeAndLockPages`
  – No hard wired limits and will not be unlocked if memory pressure is high
Sharing Pages

- Kernel32.DLL code
- Process A
- EXE code
- Process B
- Kernel32.DLL code
- RAM
- Process B code
- Process A code
- EXE code
Sharing Pages

• Code pages are shared between processes
  – 2 or more processes based on the same images
  – DLL code
    • However, DLLs must be loaded in same address

• Data pages (read/write) are shared at first
  – But with special protection called Copy-On-Write
  – If one process changes the data, an exception is caught by the
    Memory Manager, which creates a private copy of the accessed
    page for that process
    • Removing the Copy-On-Write protection
Sharing Data Without COW

- For use in DLLs only

```c
#pragma data_seg("shared")
int count = 0;
__declspec(dllexport) double epsilon = .000001;
#pragma data_seg()

#pragma comment(linker, "/section:shared,RWS")
```
Protecting Memory

• Process memory is automatically protected against read/write from other processes
  – Unless the process shares pages, or
  – Another process has virtual memory read or write access handle and uses \texttt{ReadProcessMemory} or \texttt{WriteProcessMemory}

• Processors support hardware controlled memory protection
  – Can be set with the \texttt{VirtualAlloc/Ex}, \texttt{VirtualProtect/Ex} functions
Protection Options

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAGE_NOACCESS</td>
<td>No access allowed</td>
</tr>
<tr>
<td>PAGE_READONLY</td>
<td>Only read is allowed. Other operations cause Access Violation exception</td>
</tr>
<tr>
<td>PAGE_READWRITE</td>
<td>Read and write allowed</td>
</tr>
<tr>
<td>PAGE_EXECUTE</td>
<td>Only code execution is allowed</td>
</tr>
<tr>
<td>PAGE_EXECUTE_READ</td>
<td>Execution and reading is allowed</td>
</tr>
<tr>
<td>PAGE_EXECUTE_READWRITE</td>
<td>All access is allowed</td>
</tr>
<tr>
<td>PAGE_WRITECOPY</td>
<td>Copy on write. Write access will cause a private copy to be set for the caller</td>
</tr>
<tr>
<td>PAGE_EXECUTE_WRITECOPY</td>
<td>Same as PAGE_WRITECOPY with execution support</td>
</tr>
<tr>
<td>PAGE_GUARD</td>
<td>Guard page. Access will cause EXCEPTION_GUARD_PAGE exception. Can be combined with other attributes. Used primarily for a thread’s stack</td>
</tr>
</tbody>
</table>

- On processors that do not support execution protection, Execute is the same as Read
Virtual Address Space Layout

• Each process sees its own private address space
• System space is part of the entire address space that is visible (but not accessible by user mode code)
• The layout depends on the “bitness” of the OS and the specific process
x86 Address Space Layout

- **System cache**
  - Paged pool
  - Non paged pool

- **Process page tables**
  - Hyperspace

- **Kernel & executive**
  - HAL
  - Drivers

- **Application code**
  - Global variables
  - Per thread stacks
  - DLL code

- **1 GB system space**

- **3 GB user address space**
3GB User Address Space (x86)

- Windows can be configured with the /3GB boot.ini switch (Windows 2003 and lower)
  - Or using the Boot Configuration Database (BCD) on Vista and up
- Each process gets a 3GB address space
- No returned pointer will be above 2GB unless the image is compiled with the LARGEADDRESSAWARE linker flag
  - To prevent breaking applications that depend on 31 bit addresses
x64 Address Space Layout (<= NT 6.2)

- **Mapping initialized by loader**: $\text{FFFF80000000}$
  - unmapped
- **System working set (512GB-4K)**: $\text{FFFF78000000}$
- **Shared system page**: $\text{FFFF78000000}$
- **Hyperspace Process page tables (512GB)**: $\text{FFFF70000000}$
- **4 level page table map (512GB)**: $\text{FFFF68000000}$
- **Start of system space**: $\text{FFFF08000000}$
  - unmapped
- **Per process space (~8 TB)**: $\text{000007FF000000}$
- **Reserved for HAL (2GB)**: $\text{FFFFFFF80000000}$
- **Non paged pool (128GB)**: $\text{FFFFAC00000000}$
- **System PTE pool (128GB)**: $\text{FFFFAA00000000}$
- **System mapped views (max 1TB)**: $\text{FFFFA800000000}$
- **Start of paged pool area (128GB)**: $\text{FFFFFA80000000}$
- **System cache (1TB)**: $\text{FFFFFA80000000}$
- **System working set (512GB)**: $\text{FFFFF900000000}$
- **Session space (512GB)**: $\text{FFFFF900000000}$
- **Non paged pool (128GB)**: $\text{FFFFAC00000000}$
- **System PTE pool (128GB)**: $\text{FFFFAA00000000}$
- **System mapped views (max 1TB)**: $\text{FFFFA800000000}$
- **Start of paged pool area (128GB)**: $\text{FFFFFA80000000}$
- **System cache (1TB)**: $\text{FFFFFA80000000}$
- **System working set (512GB)**: $\text{FFFFF900000000}$
- **Session space (512GB)**: $\text{FFFFF900000000}$
### x64 Address Space Layout (>= NT 6.3)

<table>
<thead>
<tr>
<th>Start</th>
<th>End</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFFFFFF000<code>00000000</code></td>
<td>FFFFFFFF07`FFFFFFF</td>
<td>8TB</td>
<td>Memory Hole</td>
</tr>
<tr>
<td>FFFFFFF080<code>00000000</code></td>
<td>FFFFFFFF0FF`FFFFFFF</td>
<td>168TB</td>
<td>Unused Space</td>
</tr>
<tr>
<td>FFFFFFFB00<code>00000000</code></td>
<td>FFFFFFFF0FFFFFFF</td>
<td>16TB</td>
<td>System Cache</td>
</tr>
<tr>
<td>FFFFFFFC00<code>00000000</code></td>
<td>FFFFFFFF0FFFFFFF</td>
<td>16TB</td>
<td>Paged Pool</td>
</tr>
<tr>
<td>FFFFFFFD00<code>00000000</code></td>
<td>FFFFFFFF0FFFFFFF</td>
<td>16TB</td>
<td>System PTEs</td>
</tr>
<tr>
<td>FFFFFFFE00<code>00000000</code></td>
<td>FFFFFFFF0FFFFFFF</td>
<td>16TB</td>
<td>Nonpaged Pool</td>
</tr>
<tr>
<td>FFFFFFFF00<code>00000000</code></td>
<td>FFFFFFFF06F`FFFFFFF</td>
<td>6.5TB</td>
<td>Unused Space</td>
</tr>
<tr>
<td>FFFFFFFF680<code>00000000</code></td>
<td>FFFFFFFF0FFFFFFF</td>
<td>512GB</td>
<td>PTE Space</td>
</tr>
<tr>
<td>FFFFFFFF700<code>00000000</code></td>
<td>FFFFFFFF07F`FFFFFFF</td>
<td>512GB</td>
<td>HyperSpace</td>
</tr>
<tr>
<td>FFFFFFFF780<code>00000000</code></td>
<td>FFFFFFFF00000FFF</td>
<td>4K</td>
<td>Shared User Data</td>
</tr>
<tr>
<td>FFFFFFFF780<code>00001000</code></td>
<td>FFFFFFFF0000000000</td>
<td>~3GB</td>
<td>System PTE WS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Start</th>
<th>End</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFFFFFFF780<code>C00000000</code></td>
<td>FFFFFFFF0FFFFFFF</td>
<td>1GB</td>
<td>WS Hash Table</td>
</tr>
<tr>
<td>FFFFFFFF781<code>00000000</code></td>
<td>FFFFFFFF0791`3FFFFFFF</td>
<td>65GB</td>
<td>Paged Pool WS</td>
</tr>
<tr>
<td>FFFFFFFF791<code>40000000</code></td>
<td>FFFFFFFF0799`3FFFFFFF</td>
<td>32GB</td>
<td>WS Hash Table</td>
</tr>
<tr>
<td>FFFFFFFF799<code>40000000</code></td>
<td>FFFFFFFF07A9`7FFFFFFF</td>
<td>65GB</td>
<td>System Cache WS</td>
</tr>
<tr>
<td>FFFFFFFF7A9<code>80000000</code></td>
<td>FFFFFFFF07B1`7FFFFFFF</td>
<td>32GB</td>
<td>WS Hash Table</td>
</tr>
<tr>
<td>FFFFFFFF7B1<code>80000000</code></td>
<td>FFFFFFFF07FF`FFFFFFF</td>
<td>314GB</td>
<td>Unused Space</td>
</tr>
<tr>
<td>FFFFFFFF800<code>00000000</code></td>
<td>FFFFFFFF08FF`FFFFFFF</td>
<td>1TB</td>
<td>System View PTEs</td>
</tr>
<tr>
<td>FFFFFFFF900<code>00000000</code></td>
<td>FFFFFFFF097F`FFFFFFF</td>
<td>512GB</td>
<td>Session Space</td>
</tr>
<tr>
<td>FFFFFFFF980<code>00000000</code></td>
<td>FFFFFFFF0A70`FFFFFFF</td>
<td>1TB</td>
<td>Dynamic VA Space</td>
</tr>
<tr>
<td>FFFFFFFF80<code>00000000</code></td>
<td>FFFFFFFF0AFF`FFFFFFF</td>
<td>512GB</td>
<td>PFN Database</td>
</tr>
<tr>
<td>FFFFFFFF<code>FFC00000</code></td>
<td>FFFFFFFF`FFFFFFF</td>
<td>4MB</td>
<td>HAL Heap</td>
</tr>
</tbody>
</table>
Virtual Address Translation

- Hardware translates each virtual address to a physical address

[Diagram]

- Page directory
- Page tables
- Translation lookaside buffer (TLB)

Virtual address Translation (hardware)

- Virtual page number
- Byte within page

If page is not valid...

Physical page number
- Byte within page

Page fault (exception handled by software)

Recently used page table entries
x86 Virtual Address Translation

- Virtual address
  - 31 bits
  - 10 bits
  - 10 bits
  - 12 bits

- Page directory (one per process)
- PDE (Page Directory Entry)
- PTE (Page Table Entry)
- Page table(s)
- Page
- Byte within page
- RAM

KPROCESS
CR3

1024 entries
1024 entries
1024 entries
Page Directory

• One per process
  – Mapped to virtual address 0xC0300000 (0xC0600000 on PAE systems)

• Physical address of page directory stored in KPROCESS structure

• While a thread is executing, the CR3 register stores its address
  – When a thread context switch occurs between threads of different processes, CR3 is reloaded from the appropriate KPROCESS instance
x86 PDE and PTE

- Each entry is 32 bits (64 bits on PAE)
- Upper 20 bits (25 bits on PAE) is the Page Frame Number (PFN)
- Bit 0 is the Valid bit
  - If set, indicates that the page is in RAM
  - Otherwise, accessing the page caused a page fault
- Other bits exist (see next slide)
Valid x86 PTE/PDE layout

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved (Large page if PDE)</td>
</tr>
<tr>
<td>12</td>
<td>Global</td>
</tr>
<tr>
<td>11</td>
<td>Reserved</td>
</tr>
<tr>
<td>10</td>
<td>Reserved</td>
</tr>
<tr>
<td>9</td>
<td>Reserved</td>
</tr>
<tr>
<td>8</td>
<td>Valid</td>
</tr>
<tr>
<td>7</td>
<td>Write (writable on MP)</td>
</tr>
<tr>
<td>6</td>
<td>Cache disabled</td>
</tr>
<tr>
<td>5</td>
<td>Write through</td>
</tr>
<tr>
<td>4</td>
<td>Owner</td>
</tr>
<tr>
<td>3</td>
<td>Accessed</td>
</tr>
<tr>
<td>2</td>
<td>Dirty</td>
</tr>
<tr>
<td>1</td>
<td>Owner</td>
</tr>
<tr>
<td>0</td>
<td>Valid</td>
</tr>
</tbody>
</table>

- **Dirty** – page has been written to
- **Large page** – this maps a large page (2MB)
- **Accessed** – page has been read
- **Owner** – user mode or kernel mode accessible
Physical Address Extensions (PAE)

- Intel Pentium Pro and up processors support a new PAE mode
- Current hardware allows access to up to 64GB of physical memory
- Virtual address translation contains an extra level of indirection
- Each PTE/PDE is 64 bits of which 25 are the PFN
  - Thus maximum is $2^{(25+12)}=2^{37}=128\text{GB}$
Address Windowing Extensions

- Temporary solution (on 32 bit) to providing access to large amounts of physical memory (>4GB)
- Requires the PAE kernel
- System cache uses the extra memory even if applications don’t
- Applications can allocate the above physical memory and map it to a window in their process address space
  - Call `AllocateUserPhysicalPages`
  - Call `VirtualAlloc` with MEM_PHYSICAL and MEM_RESERVE flags
  - Map to a window with `MapUserPhysicalPages`
Page Faults

• Valid PTE/PDE results in the CPU accessing data in physical memory
• If PTE/PDE is invalid, the CPU throws a page fault exception
  – Expects the OS to handle it and instruct it to retry the memory access
• Windows has to get the data from disk, fix the required PTE/PDE and let the CPU try again
• Page fault types
  – Hard page fault – requires disk access
  – Soft page fault – does not require disk access
  • Example: a needed shared DLL is simply directed to the process by pointing PTE to it
## Some Reasons For Faults

<table>
<thead>
<tr>
<th>Reason for fault</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accessing a page that is not in RAM but in a page file or mapped file</td>
<td>Allocate a physical page, read the data from disk and add page to the working set</td>
</tr>
<tr>
<td>Accessing a page that is in the modified or standby page list</td>
<td>Move the page to the working set</td>
</tr>
<tr>
<td>Accessing a page that is not committed</td>
<td>Access violation</td>
</tr>
<tr>
<td>Accessing a page from user mode that can only be accessed from kernel mode</td>
<td>Access violation</td>
</tr>
<tr>
<td>Writing to a page that is read only</td>
<td>Access violation</td>
</tr>
<tr>
<td>Accessing a demand zero page</td>
<td>Add a zeroed page to the working set</td>
</tr>
<tr>
<td>Writing to a guard page</td>
<td>Guard page violation (if part of a thread’s stack, commit more memory and add to working set)</td>
</tr>
<tr>
<td>Writing to a copy-on-write page</td>
<td>Make a process private page copy and replace in working set</td>
</tr>
<tr>
<td>Executing code in page marked no-execute</td>
<td>Access violation</td>
</tr>
</tbody>
</table>
Invalid PTEs

• The CPU throws a page fault exception when the Valid bit (bit 0) in a PTE is clear
• Windows uses the other PTE bits to indicate where the required page can be found
• Example: a page that resides in a page file (x86 w/o PAE)
Page File(s)

• Backup storage for writeable, non-shareable committed memory
  – Up to 16 page files are supported
  – On different partitions
  – Initial size and maximum size can be set
    • Using the System applet in Control Panel
  – Named PageFile.Sys on disk (root partition)
  – Created contiguous on boot
  – Initial value should be maximum of normal usage

• Page file information in the registry
  HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management\PagingFiles
Page File Sizes

• If selecting “system managed size” in System applet

<table>
<thead>
<tr>
<th>RAM size</th>
<th>Initial page file size</th>
<th>Maximum page file size</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 1GB</td>
<td>1.5 * RAM</td>
<td>3 * RAM</td>
</tr>
<tr>
<td>&gt;= 1GB</td>
<td>1 * RAM</td>
<td>3 * RAM</td>
</tr>
</tbody>
</table>

• When page files space run low
  – “System running low on virtual memory”
    • First time: Before page file expansion
    • Second time: When committed bytes reaching commit limit
  – “System out of virtual memory”
    • Page files are full
Commit Charge

- Commit charge represents the memory than can be committed
  - In RAM and existing page file(s)
- Contributors to the commit charge
  - Private committed memory (VirtualAlloc with MEM_COMMIT flag)
    - No RAM or page file is used until memory is actually touched
    - Until then, considered demand zero pages
  - Page file backed memory mapped file allocated with MapViewOfFile
  - Copy on write mapped memory
  - Kernel non-paged and paged pools
  - Kernel mode stacks
  - Page tables and yet-to-be-created page tables
  - Allocations with Address Windowing Extensions (AWE) functions
- The commit limit is basically the amount of RAM plus maximum size of all page files
Working Sets

• Process Working Set
  – The subset of the process’ committed memory that resides in physical memory

• System working set
  – The subset of system memory residing in physical memory

• Systems with terminal services
  – Some kernel memory is on a per session basis and as such working set as well

• Demand paging
  – When a page is needed from disk, more than one is read at a time to reduce I/O
  – 3 pages for code based images, 7 pages for others
Page Frame Number Database

• PFN database describes the state of all physical pages
• Valid PTEs point to entries in the PFN database, and a PFN entry points back to the PTE
• The structure layout of a PFN entry depends on the state of the page
• Kernel debugger: `!memusage`, `!PFN`
Standby Page List
Free Page List
Zero Page List
Bad Page List

Standby Page List
page read from disk or kernel allocations

Free Page List
zero page thread

Zero Page List

Bad Page List

Modified Page List
modified page writer

“soft” page faults

working set replacement

Private pages at process exit

Process Working Sets

demand zero page faults

“soft” page faults

working set replacement

Private pages at process exit

Page Dynamics

Windows Internals

426
## Physical Page States

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active (Valid)</td>
<td>The page is part of a working set (either process or kernel) or is not a part of any working set (kernel non-paged pool) with a valid PTE pointing to it</td>
</tr>
<tr>
<td>Transition</td>
<td>Temporary state when the page is in the middle of I/O. The PTE is encoded so that collided pages can be recognized and handled</td>
</tr>
<tr>
<td>Standby</td>
<td>The page belonged to a working set but was removed. It has not been modified since it was last written to disk. The PTE still refers to the page but is marked invalid and in transition</td>
</tr>
<tr>
<td>Modified</td>
<td>Similar to Standby, but the page was modified, so cannot be discarded until it’s written back to disk</td>
</tr>
<tr>
<td>Modified no write</td>
<td>Similar to modified, but will not be written to disk. The cache manager does this at the request of file system drivers</td>
</tr>
<tr>
<td>Free</td>
<td>The page is free but contains garbage data in it</td>
</tr>
<tr>
<td>Zeroed</td>
<td>A free page that is zeroed out</td>
</tr>
<tr>
<td>Bad</td>
<td>Some hardware error, so page cannot be used</td>
</tr>
</tbody>
</table>
Standby and Modified Page Lists

- Used to
  - Avoid writing pages back to disk too soon & avoid releasing pages to the free list too soon
- The system can replenish the free page list by taking pages from the top of the standby page list
  - This breaks the association between the process and the physical page
- Pages move from the modified list to the standby list
  - Modified pages’ contents are copied to the pages’ backing stores (usually the paging file) by the modified page writer (see next slide)
  - The pages are then placed at the bottom of the standby page list
  - On Windows NT 10.X pages are compressed into the System process user space
    - Only in a case of low memory, compressed pages are written to the page file
- Pages can be faulted back into a process from the standby and modified page list
  - The SPL and MPL form a system-wide cache of “pages likely to be needed again”
Modified Page Writer

• Moves pages from modified to standby list, and copies their contents to disk
  – i.e., this is what writes the paging file and updates mapped files (including the file system cache)

• Two system threads (priority 17)
  – One for mapped files, one for the paging file

• Triggered when
  – Memory is overcommitted (too few free pages)
  – Or modified page threshold is reached
    • Currently 800 pages
  – Does not flush entire modified page list
Zero Page List

• Large uninitialized data regions are mapped to demand zero pages

• On first reference to such a page, a page is allocated from the zero page list
  – No need to read zeroes from disk to provide the “data”
  – After modification, these pages are mapped to the paging file

• Zero page list is replenished by the “zero page thread”
  – The only thread running at priority 0
  – One per system (even on SMP)
  – Takes pages from the free page list, fills them with zeroes, and puts them on the zero page list while the CPU is otherwise idle
  – Usually is waiting on an event - which is set when, after resolving a fault, system notices that zero page list is too small
Memory Compression (NT 10.X)

- Working set (RAM)
- Modified list
- Standby list
- Free list
- Pagefile (Disk)
- SYSTEM-process (User Mode memory)
- Default Store
- Kernel Mode memory

TRIM
COMPRESS
WRITE
Repurposing

The unneeded free pages go to free
Memory compression

• Can compress pages in memory and result in less Hard Page Faults
  – Does not prevent the use of pagefile but lowers the need to use it

• History
  – Readyboost technology (Windows 7) used a compressed cache on a USB stick to improve performance to store data also written to the pagefile
  – Windows 8 evolved this by moving the cache into regular memory
    • But data was also still written out to the pagefile
  – The Windows 8.1 changed this by the cache switching to now being its own compressed pagefile in RAM that could be paged out to disk if required
  – Windows 10 changes this to now apply to all systems and all types of applications
    • Any data you now find in your disk-based pagefile is compressed after initially being compressed in the RAM based pagefile
    • It is only written to disk as a lack of resources required
Memory Compression – Why?

• Unused memory is unused resources
• Memory is 100,000 times faster than a spindle and also 50 times faster than an SSD
• SSD’s wear out
• After utilizing this most people don’t need to page out to disk at all (statistically)
Memory Compression – How?

• Memory gets put into stores
  – Modern/Universal apps have their own store pre process
  – If app doesn’t have its own then the SYSTEM-process’ store is used
    • This might surprise some as the memory footprint of SYSTEM has been a few kilos for the past 25 years
Memory Compression – Price?

• Compression/Decompression requires CPU so is it worth it?
  – Yes! It definitely is!!!
  – Decompression rates:
    • 3 Year old laptop = 300MB/s per core
    • Current laptop with 8 cores = 7GBps
    • Lowest end Windows phone = 250-300MB/s

• Compression and decompression are low priority tasks as long as the computer is not critically low on memory
Memory Compression

• Also organizes memory so that it’s not fragmented
  – When pages enter a store that store is created a contiguous reservation in pagefile so that when that store is paged into pagefile they will not be fragmented

• Pages are compressed in page file as well
  – Less reads needed to get the same amount of data
  – Decompressed outside of the pagefile
Secure Enclave

enclave

[ˈɛnklɛv]  🎧

NOUN

1. a portion of territory surrounded by a larger territory whose inhabitants are culturally or ethnically distinct.
"the besieged Muslim enclave of Srebrenica"

synonyms: area of land · area · region · enclave · country · state · [more]

- a place or group that is different in character from those surrounding it:
"the engineering department is traditionally a male enclave"
Enclave (NT 10 Update 2)

- Provide an APIs to use the new Intel SGX Skylake CPU capability – Software Guard Extensions
  - The system BIOS also needs to support the feature
- A method baked by the CPU to isolate a region of memory pages
- Only code executed inside the enclave can reach the data inside the enclaved region
  - First check: IsEnclaveTypeSupported
  - Then create: CreateEnclave (Kernel NtCreateEnclave)
  - Load data with: LoadEnclaveData (NtLoadEnclaveData)
  - Initialize it (begin code execution): InitializeEnclave (NtInitializeEnclave)
  - Destroy it: VirtualFree (NtVirtualFree)
Memory Manager Changes

• To support the feature in Windows 10 Update 2, the kernel adds a new Enclave Page List
• Internally, however, since the memory manager has actually run out of list identifiers, the kernel actually identifies these pages as being “bad” pages currently suffering an in-page error
  – This actually makes sense – the memory manager knows never to use bad pages, so calling enclave pages “bad” is another way to keep them at bay
System Memory Usage

• Kernel and driver memory usage breaks down into:
  – Non-pageable code
  – Pageable code
  – File system cache
  – Non-paged pool
  – Paged pool
System Memory Pools

- The kernel provides two general memory pools for use by the kernel itself and device drivers
  - Non-paged pool
    - Memory always resides in RAM (never paged out)
    - Can be accessed at any IRQL
  - Paged pool
    - Memory can be swapped to disk
    - Should be accessed at IRQL < DPC_LEVEL (2) only

- Pool sizes depend on the amount of RAM and the OS type (Professional vs. Servers)
  - Can be altered (up to some maxima) in registry
    - HKLM\System\CurrentControlSet\Control\Session Manager\Executive
  - Task Manager displays current sizes (in RAM)
**System Pools in Task manager**

1. “Kernel Memory Paged” = physically resident size of paged pool

2. “Kernel Memory Nonpaged” = physical size of non-paged pool
System Memory Pools APIs

- **ExAllocatePool** (obsolete)
  - Allocate memory from the paged or non-paged pool
- **ExAllocatePoolWithTag**
  - Allocate memory and “tag” it with a 4-byte value
  - Used to track memory leaks
- **ExFreePoolWithTag**
  - deallocates a block of pool memory allocated with the specified tag
- Pool usage (and tags) can be viewed with **PoolMon.exe**
Memory Mapped Files

• Internally called Section objects
• Allows the creation of “views” into the file
  – Returns a memory pointer for data manipulation
• Implies shared memory capabilities
  – This is the usual case with mapping EXEs and DLLs
• Also can create “pure” shared memory
  – Backed up by the system paging file
  – When memory mapped file object destroyed, memory is “recycled”
Memory Mapped Files API

- **Win32**
  - `CreateFileMapping` – create a file mapping object based on a specific file (previously created with `CreateFile`) or based on system paging file
  - `OpenFileMapping` – open an existing MMF based on its name (NOT the filename)
  - `MapViewOfFile` (Ex) – create a “view” into the MMF
- **Kernel**
  - `ZwCreateSection` – creates a section object (if based on a file, call `ZwCreateFile` first)
  - `ZwMapViewOfSection` – map a “view” into system space
Large Pages

• Large pages allow mapping using a PDE only (no need for PTEs)
  – Advantage is better use of the translation look aside buffers
• Windows maps by default large pages for Nt0sKrnl.Exe and Hal.Dll as well as core system data (initial non paged pool and PFN database)
• Potential disadvantage
  – Single protection to entire page
  – Thus, kernel is protected with read/write (can’t protect with read only because data is there as well)
Applications Using **Large Pages** (>= NT 5.2)

- Applications can use large pages for their allocations by specifying the `MEM_LARGE_PAGE` in calls to the `VirtualAlloc` function
- Size and alignment must be multiple of large page size
  - Can be determined by calling the `GetLargePageMinimum` function
- Must have the `SeLockMemoryPrivilege` privilege
Memory APIs in User Mode

High Level

C/C++ runtime API

Heap API

Virtual API

Low Level
The Heap Manager

• Allocating in page granularity is sometimes too much
  – Need fine grained control

• The heap manager manages smaller allocations (8/16 bytes minimum)

• API exported from Ntdll.dll for user mode and Ntoskrnl.exe from kernel mode

• The HeapXxx Windows API functions are a thin wrapper over the native Ntdll.dll functions
Heap Types

• One heap is always created with a process, called the Default Process Heap
  – Can be accessed using `GetProcessHeap`

• Additional heaps can be created using the `HeapCreate` function

• A heap can be fixed in size or growable
  – The default heap is growable

• Low Fragmentation Heap (LFH)
Windows Heaps

Default Process Heap
C Runtime Heap
Application Private Heaps

ntdll.dll Heap Manager

Virtual Memory Manager
Main Heap API Functions

- **HeapCreate** – Creates a new heap with the specified committed and reserved sizes (can be expandable)
- **HeapDestroy** – Destroy a heap
- **HeapAlloc**, **HeapFree**, **HeapReAlloc** – Allocate, free or re-allocate from a heap
- **HeapWalk** – Enumerate the contents of a heap
- **GetProcessHeap**, **GetProcessHeaps** – Return a handle to the default heap / an array of all process heaps
The Heap Data Structure

• The heap is built from two main components
  – Front-End Allocator
    • Fast, fixed size based memory allocator
  – Back-End Allocator
    • Complex, second chance memory allocator

• When a request is not satisfied by the Front-End allocator, it goes to the Back-End allocator

• Currently ( >= NT 6.x) there are two Front End allocators:
  – Look aside list allocator
  – Low fragmentation allocator

• Additional deep information:
  – Understanding Low Fragmentation Heap
The LAL Heap Data Structure

- In general the front-end allocator is built from an array of free look aside lists with fixed size blocks.
The Back End Allocator Data Structure

- The back-end maintains one varied block size **free** list and 126 fixed size look aside **free** lists.
Each Block Data Structure

Mandatory Metadata For Heap Management:
- Current Block Size (2)
- Previous Block Size (2)
- Segment Index (1)
- Flags (1)
- Unused (1)
- Tag Index (1)
- User Area (This is for us 😊)

Optional Metadata For Debugging Support:
- Suffix Bytes (16)
- Fill Pattern (In Debug mode) (???)
- Heap Extra (8)
Heap Allocation Algorithm

 Caller asks for a block of memory

 LAL Block Available?

 Yes: Remove from LAL & return to caller
 No:

 Find large enough block and split it to two

 Exact size block exists in the Back End Allocator?

 Yes: Mark block as busy, remove from list & return to caller
 No: Return the busy half to the caller

 Block Found?

 Yes: Mark one half as busy, add the second block to the appropriate list
 No: Allocate another segment

 Allocate another segment

 Create new block, mark it as busy

 Return the new block to the caller
Summary

• The Memory Manager provides various services to applications and system components
• Virtual memory is translated to physical memory while using page files as backup
• Sharing pages is implemented to minimize RAM usage and share data between processes
• User mode applications have plenty of memory related API and services
Module 6
SECURITY
Agenda

• Windows Security Features
• Security Components
• User Access Control (UAC)
• Access Tokens
• Protecting Objects
• Privileges
• New Security features in NT 10.X
• Summary
Windows Security Features

• Design for C2 level security of the DoD
• Permissions can be applied to all shareable resources
  – Including NTFS file system (but not FAT)
• NTFS encryption protects data from unauthorized physical access
• Support for Kerberos authentication
• Support for Digital Certificates
• Cryptographic API for encryption and hashing
Authentication and Authorization

• Authentication
  – Be sure that the user is who she claim she is
  – It is also about identity

• Authorization
  – Allow and deny read/write/control access to information, settings and resources

• There are other aspects such as integrity, Denial of Service, Cryptography, Intrusion Detection, Malware (Viruses, Worms) and more
Security Components Overview

Kernel Mode:
- System Service Dispatcher
- Hardware Abstraction Layer (HAL)
- Kernel
- Device & File Sys. Drivers
- I/O Mgr
- File System Cache
- Object Mgr
- Plug and Play Mgr
- Power Mgr
- Security Reference Monitor
- Virtual Memory
- Threads & Processes
- Configuration Mgr (registry)
- Local Procedure Call
- Graphics Drivers
- Windows USER, GDI
- LSASS
- LSA Server
- SAM Server
- MSVC1_0.dll
- Kerberos.dll
- Event Logger
- Active Directory
- SAM
- WinLogon
- LogonUI
- LSA Policy

User Mode:
- LogonUI
- LSA Policy
- MSVC1_0.dll
- Kerberos.dll
- Event Logger
- Active Directory
- SAM
- WinLogon
- LogonUI
- LSA Policy

Windows Internals

463
Security System Components

• Security Reference Monitor (SRM)
  – Part of the Executive, providing access token definition, access checks on objects, manipulating privileges and auditing messages

• Local security authority subsystem (Lsass)
  – Running the image `\Windows\System32\Lsass.exe`
  – Responsible for local security policy, user authentication and sending events to the Event Log
  – Most functionality implemented by the Local security authority service (Lsasrv.dll)

• Lsass policy database
  – Local system security policy stored under HKLM\SECURITY
Security System Components

- Security Account Manager (SAM) service
  - A set of functions to manage the users and groups on the local system
  - Implemented in `\Windows\System32\SamSrv.dll` (runs under `Lsass.Exe`)

- SAM database
  - On non-domain controller systems, contains local users and groups (along with passwords)
  - On domain controller systems, stores admin recovery account data
  - Stored in the registry under `HKLM\SAM`
Security System Components

• Active Directory
  – A directory service storing objects data in a computer domain
  – Stored on computers designated as Domain Controllers
  – Runs as a service (\Windows\System32\Ntdsa.dll) under Lsass

• Authentication packages
  – DLLs that run in the context of Lsass or other processes that implement an authentication policy (e.g. returning whether a user may log in)
  – The returned information allows Lsass to generate a token
Security System Components

• Logon process
  – Running the image `\Windows\System32\Winlogon.Exe`
  – Responsible for responding to the SAS (Secure Attention Sequence, by default Ctrl+Alt+Del) and for managing interactive logon sessions (e.g. running the user’s shell)
Authentication in Windows

• It begins with an instance of **WinLogon** process
  – In NT 6.x There is one WinLogon per session

• Usually at logon the user supply the user name and password
  – WinLogon supports **custom credential provider**
  – This enables scenarios such as Smart Card, Biometric or even face recognition based logon
Credential Providers

• Prior to Windows Vista (< NT 6.0)
  – WinLogon loaded the Graphics Identification and Authentication (GINA) DLL to display a logon UI
  – Custom GINA hard to write
  – Only one can be configured

• Windows Vista (>= 6.0) and later
  – The LogonUI.Exe process to display logon UI
  – Loads logon providers from registry info
  – Microsoft provides Interactive (authui.dll) and Smartcard (Smart-cardcredentialprovider.dll) providers

• Windows NT 10.X provides “Windows Hello”
Why Is Logon Secure?

• Winlogon creates the Winlogon desktop which is the only one visible
• Pressing SAS takes to that desktop
• Keyboard handler in the kernel disables hooks when SAS is detected
• Deregistering the SAS is only possible by the thread that registered it
Logon Sequence

Winlogon calls CSP to get security info

Winlogon sends returned data to Lsass

Local logon (Y) or Legacy domain?

Use Kerberos

Match credentials?

Use MSVC1_0

N

Y

N

Logon failed

N

Y

Lsass retrieves account privileges from security DB or Active Directory

Lsass creates a token for the session

Winlogon attaches it to the first process in the session (subsequent processes get a copy of that token)
Non-Interactive Logon

• A Network logon
  – Also called Non-Interactive logon

• Non-interactive authentication can only be used after an interactive authentication has taken place.

• In non-interactive authentication, the user does not input logon data,
  – Instead, previously established credentials are used

• The local LSA uses SSPI, a standard security provider interface to pass the current logon information

• This enables Active Directory and Workgroup non-interactive logons
Additional Logon

• The LogonUser function attempts to log a user on to the local computer
  – The result is a primary access token
  – Use the CreateProcessAsUser to create a process for this user
  – Use ImpersonateLoggedInUser to impersonate a thread
    • Call RevertToSelf to revert to self

• You can combine LogonUser & CreateProcessAsUser with CreateProcessWithLogonW
SID – Security Identifier

• A **SID** is a unique value used to identify a trustee
  – A **trustee** is one of User, Group, Domain, Well Known Group, Computer

• SIDs are issued by authority such as Domain Controller, or LSA

• Windows uses SIDs in:
  – Security Descriptors to identify the owner on an object
  – Access Control Entry, to identify the target trustee
  – Access Token, to identify the user and the groups

  **S-1-5-21-2280383352-4165795427-4139450486-1001**
Well Known SIDs

• Windows defines some built in groups
• Examples

<table>
<thead>
<tr>
<th>SID</th>
<th>Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S-1-1-0</td>
<td>Everyone</td>
<td>All users</td>
</tr>
<tr>
<td>S-1-2-0</td>
<td>Local</td>
<td>Users who log on physically</td>
</tr>
<tr>
<td>S-1-5-18</td>
<td>Local System</td>
<td>Local System account</td>
</tr>
<tr>
<td>S-1-5-20</td>
<td>Network Service</td>
<td>Network Service account</td>
</tr>
<tr>
<td>S-1-5-19</td>
<td>Local Service</td>
<td>Local Service account</td>
</tr>
<tr>
<td>S-1-5-32-544</td>
<td>Administrators</td>
<td>Administrators group</td>
</tr>
</tbody>
</table>
Authorization - The Access Token

- The Token is used for authorization
  - To check access to **securable object**
  - To perform a system task that requires privileges

- The token contains (partial list):
  - The SID for the user's account
  - SIDs for each group of which the user is a member
  - A logon SID that identifies the current logon session
  - A list of the privileges held by the user and the user's groups
  - The SID for the primary group
  - The default DACL
  - Whether the token is a primary or impersonation
Using the **Access Token**

• Every process has a primary token
  – This is usually a copy of the logon token

• Each thread has a copy of the primary token
  – A thread can have another token – the impersonation token
    • Impersonation can be used in a server application to change the thread security context to match the client context

• To work with a token call: **OpenProcessToken**
Privileges

- The right to perform a system-related **operation**
  - Setting System times, add device, create page file,…
- For each user the system has a privileges db
  - The Access Token holds the list of privileges
  - Privilege can be enabled or disabled
- When trying to perform a privileged operation
  - The system checks the user's access token
    - The token should have the necessary privileges
    - And the privileges have to be enabled
Setting Privileges DB
User Account Control

• UAC is a security feature of \( \geq \) NT 6.x
  – It runs user applications in a security safe environment
  – It has changed the ecosystem to support running as a standard user

• UAC is based on a restricted filtered token
  – The user is not part of the Administrator group
  – The user has much less privileges

• Even if you add a privilege, it may not be there!

• We will talk more about UAC later
UAC Filtered Token
ProcExp + standard user notepad.exe + admin notepad.exe

DEMO
Debugging Privileges
Securable Object Access Control

• A securable object is an object that can have a security descriptor

• All named Windows objects are securable
  – Some unnamed objects, such as process and thread objects, can have security descriptors too
  • You can pass Security Attribute that has a Security Descriptor when you create objects such as file & process

• Each type of object defines its own set of specific access rights

• Each type of object defines its mapping of generic access rights
Security Descriptor

• The security DB of an object
  – The SID of the owner and the primary group of the object
  – A list of access rights that tell who can do what with the object ➔ DACL = List of ACEs
  – An information that tells the system the type of action that generates security audit event

• The SID is kept in the object
  – For example each NTFS file has a security NTFS stream

• The SID may contain inherited ACEs
  – This enables security setting of a container such as a directory to propagate to the child tree
Access Mask

• Each Access Control Entry (ACE) has:
  – The target User or Group SID
  – The type of ACE (Allowed or Denied)
  – The Access Mask

• Checking for access rights
  – a bitwise mask comparison process

<table>
<thead>
<tr>
<th>G</th>
<th>R</th>
<th>W</th>
<th>E</th>
<th>A</th>
<th>Reserved</th>
<th>Standard Rights</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0 9 8 7 6 5</td>
<td>AS</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1 1 1 1 0 0</td>
<td>GA</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>9</td>
<td>1 8 7 6 5 4</td>
<td>GE</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>9</td>
<td>8</td>
<td>7 6 5 4 3</td>
<td>GW</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 0 0 0 0</td>
<td>GR</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 0 0 0 0</td>
<td></td>
</tr>
</tbody>
</table>

AS

Object Specific Access Rights
Accessing Securable Object

File test.txt

DACL

Access Denied

Access Allowed

GROUP: USERS
Alon
Aaron

Administrators
Access Allow
Read, Write, Delete

HANDLE hFile = CreateFile("test.txt", GENERIC_READ, ...)

HANDLE hFile = CreateFile("test.txt", GENERIC_WRITE, ...)

Access Denied

Access Allowed

Order Of ACES Matter
DEMO

File System Security Settings
Object Handle Access Mask

• Most of Win32 Object handle creation APIs ask for access mask (dwDesiredAccess)
  – OpenProcess, CreateFile, OpenThread, RegCreateKeyEx, ...

• You should request only the access rights that you need
  – This prevents using the handle in an unintended way

• Most of the time use the generic access rights
  – This is simpler than specifying all the corresponding standard and specific rights

• The MAXIMUM_ALLOWED can be use to request that the object be opened with all the access rights that are valid for the caller
Object Handle Access Mask

• Each **handle** in the process handle table has a pointer to the object, some flags and an access mask

• The Object Handle Access Mask is set when the handle is created
  – Use **DuplicateHandle** to get another handle to an object with different access rights

• Since the access mask is part of the handle:
  – There is no farther DACL check when accessing the object
  – Impersonated client can access an object using any opened handle, even if this client can’t create such an handle
    • If the handle has been opened prior the impersonation
Security Descriptor Definition Language

• A text based language that enables us to define a security descriptor information
  – It also provides a way to store and transfer security information (serialization)
  – `ConvertSecurityDescriptorToStringSecurityDescriptor`
  – `ConvertStringSecurityDescriptorToSecurityDescriptor`
  – `ObjectSecurity.SetSecurityDescriptorSddlForm`

• Example:
  – "O:AOG:DAD:(A;;RPWPCCDCLCSWRCWDWO;G;S-1-0-0)"

• To learn more about SDDL language look [here](#)
Windows Authorization API

• Win32 has a large set of APIs, data structures and enums that let you manipulate and read securable objects

• Common use of these API are:
  – Create object with SID that contains DACL
  – Create and manipulate handle with specific Access Mask
  – Impersonate and Revert thread identity
  – Manipulate objects and container security settings
Windows Security in .NET

- Two namespaces provide Windows access control abilities in the .NET framework
  - `System.Security.Principal`
  - `System.Security.AccessControl`

- To deal with securable object you have two options:
  - The `ObjectSecurity` class (and its derived chain)
    - For Win32 objects that has a .NET wrapper
  - The `GenericSecurityDescriptor` (and its derived chain)
    - For Win32 objects that need to be interop to.

- Look at the `.NET Security Workshop Access Control`
Session Zero Isolation

• Prior to Windows NT 6.x users applications and services ran in the same session
  – Sessions were introduced in Windows 2000 to support terminal service

• The main reason for sessions is isolation
  – Since isolation is a security feature, in Windows NT 6.x services run in Session zero, while users runs on session 1, 2, ...
    • Terminal services & Switch User

• Session Zero Isolation makes an headache
  – Can’t send Windows Messages, Can’t Display UI, Kernel Object are private to session, and more...
System Processes

• Session 0 is dedicated to system processes and services only
  – First interactive user session is 1

• When a user logs on
  – SMSS.EXE starts a new instance of itself to configure the new session
    • Creates the Windows subsystem process (CSRSS.EXE) and WinLogon.Exe

• Interactive Services
  – Work as expected
Sessions in XP/W2K/WS03

Window Station

Session 0

Services

1st User's Window

1st User's Window

1st User's Window

Screen Saver

Login

Shatter Attack

Windows Internals

CodeValue College
Sessions in Vista/7/8

Session 0
- Window Station
- Service
- Service

Session 1
- Window Station
- 1st User's Window
- 1st User's Window
- Screen Saver
- Login
- Secure
The purpose of the Windows integrity mechanism is to restrict the access permissions of applications that are running under the same user account and that are less trustworthy.

There are four integrity levels:

<table>
<thead>
<tr>
<th>Level</th>
<th>Typical process</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>Services</td>
</tr>
<tr>
<td>High</td>
<td>Elevated user apps</td>
</tr>
<tr>
<td>Medium</td>
<td>Default: Normal user apps</td>
</tr>
<tr>
<td>Low</td>
<td>IE Protected Mode</td>
</tr>
</tbody>
</table>
Object Integrity Labels

• Every securable object has one
• Includes Level and Policy
• Policies can include:
  – No-Write-Up - Lower IL can’t write to object
  – No-Read-Up: -Lower IL can’t read object
  – No-Execute-Up - Lower IL can’t execute object
• Defaults:
  – Objects: Medium + No-Write-Up
  – Processes: No-Write-Up + No-Read-Up
• The integrity labels are stored as an ACE with a special SID in the SACL on the object
Complicated? Let’s See Again

• Object can have an integrity label
  – Stored in its Security Descriptor

• Processes run at an integrity level (IL)
  – Stored in its Access Token

• Process cannot access object if its IL is lower than the object’s label
  – Part of the access check
<table>
<thead>
<tr>
<th>Integrity Level</th>
<th>Mandatory Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S-1-16-0</td>
<td>Untrusted process</td>
<td>Used for anonymous processes in some cases</td>
</tr>
<tr>
<td>S-1-16-4096</td>
<td>Low Level</td>
<td>Used for Internet Explorer</td>
</tr>
<tr>
<td>S-1-16-8192</td>
<td>Medium Level</td>
<td>Default for standard users</td>
</tr>
<tr>
<td>S-1-16-12288</td>
<td>High Level</td>
<td>Used for administrators</td>
</tr>
<tr>
<td>S-1-16-16384</td>
<td>System Level</td>
<td>Used for system processes and services</td>
</tr>
<tr>
<td>S-1-16-20480</td>
<td>Protected process</td>
<td>Used for certain protected processes, such as DRM processes</td>
</tr>
</tbody>
</table>
SDDL for Mandatory Labels

#define SDDL_MANDATORY_LABEL TEXT("ML") // Integrity label

// The SDDL strings for the mandatory label policy flags, which are in the access mask, are the following:
#define SDDL_NO_WRITE_UP TEXT("NW")
#define SDDL_NO_READ_UP TEXT("NR")
#define SDDL_NO_EXECUTE_UP TEXT("NX")

// The SDDL SID strings for the integrity levels are the following:
#define SDDL_ML_LOW TEXT("LW")
#define SDDL_ML_MEDIUM TEXT("ME")
#define SDDL_ML_HIGH TEXT("HI")
#define SDDL_ML_SYSTEM TEXT("SI")

• Example: "S:(ML;;NW;;;LW)"
Process Explorer to the Rescue
Sharing Kernel Objects

• Sharing Kernel Object Between a service and user application is a bit tricky
  – We need to create the object in the Global namespace
  – We need to set the Integrity Level Label to Medium
  – We need to make sure that the object can be accessed by the client (DACL)

• The problem:
  – Any wrong setting will block the access
  – We will see how to do it in the following exercise
User Access Control (UAC)

• Goals
  – Running applications with standard user rights
    • Not as administrators
  – Allow applications to elevate to administrator rights when needed
  – Allow legacy applications to run with standard user rights even when assuming administrative rights
Running as a Standard User

• Windows Vista adds
  – Privileges that allow more granularity for various activities
    • E.g. changing time zone as opposed to changing time
  – More configurations options
    • e.g. wireless settings, power options, installing critical updates
  – Virtualization of file and registry to allow legacy applications to access global elements (e.g. %ProgramFiles% folder) without really compromising the system

• Windows 7 adds two more levels for UAC
Virtualization

- Applications should not write user-specific data in global locations in the file system or in the HKLM\Software registry hive.
- When a legacy application tries to write to a global location, an access denied is returned.
  - Redirected to a per user area.
    - E.g. C:\Users\<username>\AppData\Local\VirtualStore
- When reading data, first checks the per user area, and if not found, global area.
- "Legacy" means
  - 32 bit, runs with standard user rights and has no manifest file indicating it’s a Vista or later application.
Administrator Approval Mode

• When a user logs in
  – If is a member of any administrators group
    • Two tokens are created
  – Otherwise
    • A standard user token is created

• When a process needs to run elevated
  – An administrator approves by clicking (AAM, Consent elevation)
  – A non-administrator needs to supply username and password of an administrator (Over The Shoulder elevation, OTS)
Running Elevated

• How to run a process with elevated credentials?
  – Right-click on the file in Windows Explorer and select “Run as Administrator”
  – Call the `ShellExecuteEx` API with the “runas” verb
  – Add a manifest file requesting administrative rights

```xml
<trustInfo xmlns="urn:schema-microsoft-com:asm.v3">
  <security>
    <requestedPrivileges>
      <requestedExecutionLevel Level="requireAdministrator" uiAccess="false"/>
    </requestedPrivileges>
  </security>
</trustInfo>
```
The UAC Manifest

```bash
> mt.exe -inputresource:App.EXE;#1 -out:extracted.manifest
```

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
  <trustInfo xmlns="urn:schemas-microsoft-com:asm.v3">
    <security>
      <requestedPrivileges>
        <requestedExecutionLevel level="asInvoker"
          uiAccess="false"/>
      </requestedPrivileges>
    </security>
  </trustInfo>
</assembly>
```

<table>
<thead>
<tr>
<th>Permission</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>asInvoker</code></td>
<td>Launch with the same token as the parent process</td>
</tr>
<tr>
<td><code>highestAvailable</code></td>
<td>Launch with the highest token that the user has</td>
</tr>
<tr>
<td><code>requireAdministrator</code></td>
<td>Highest token of the User provided User is a member of Administrators group</td>
</tr>
</tbody>
</table>
Executing an Elevated Process

• When executing an image requesting administrative rights
  – The Application Information Service (AIS, AppInfo.Dll) launches Consent.Exe
    • Captures the screen into a bitmap
    • Applies some fade effect
    • Switches to a desktop only accessible to the LocalSystem account
    • Displays the appropriate elevation dialog box
      – Signed by Microsoft or a Windows component – blue stripe
      – Signed by other than Microsoft – gray stripe
      – Unsigned – orange stripe
    • If the user declines the elevation, an access denied is returned to the process creator
    • Otherwise, AIS calls `CreateProcessAsUser` and redirects the original process to be the parent of the new process
User Interface Privilege Isolation

- Windows Vista introduced the concept of Integrity Levels
- By default, a process cannot send a window message to another process with a higher integrity level
- This mechanism is called User Interface Privilege Isolation (UIPI)
- Running as Standard User sets integrity level to Normal
- Running as Administrator sets integrity level to High
- Use the ChangeWindowMessageFilter & ChangeWindowMessageFilterEx to allow incoming messages
- Use UAC manifest: uiAccess to bypasses
Object Namespace

- protects named objects from unauthorized access
  - `CreatePrivateNamespace`
    - The `lpAliasPrefix` parameter of serves as the name of the namespace
  - The function requires an isolation boundary
    - use the `CreateBoundaryDescriptor` and `AddSIDToBoundaryDescriptor` functions
  - each namespace is uniquely identified by its name and boundaries
    - Two namespaces with the same name but different boundaries are different
  - Sample: Microsoft SDKs\Samples\winbase\services\pvtnamespace
Other Security Mechanisms

- Windows Firewall
- Nx (DEP)
- /GS
- Safe CRT functions
- ASLR
- BitLocker
- Crypto API
- Protected Process Light (>= NT 6.3)
- Virtual Secure Mode (VSM)(NT 10.X)
- Many more...
- Windows 10 Security Features
Summary

• Security considerations must be factored into the design of a system
• All kernel based objects can be protected
• Windows uses a Security Descriptor to determine who can do what with an object
I/O SYSTEM
Agenda

- The I/O System
- Device Drivers
- The Windows Driver Model (WDM)
- The Windows Driver Foundation (WDF)
- The Universal Driver Model
- Plug & Play
- IRP Processing
- I/O System Objects
- Power Management
- Driver Verifier
I/O Subsystem Design Features

• As many "setup" functions as possible are performed before driver is invoked
  – Reduced security risks - all security tests, buffer access methods, etc. are done by kernel-supplied code
  – Reduced risk of system crashes

• Many kernel mode functions needed by drivers are performed by system-supplied routines
  – Driver only decides which to call, and in what order
  – Drivers are easier to write, and more reliable

• I/O subsystem provides rich "feature set" for drivers
  – Hides many architectural and platform differences
I/O System Components

- Applications
- Windows Services
- User mode PnP Manager
- Setup Components
- User mode
- Kernel mode
- WDM WMI routines
- PnP Manager
- Power Manager
- I/O Manager
- I/O System
- Drivers
- HAL

- .inf files
- .cat files
- registry

Windows Internals
General Driver Types

• Function driver
  – Manages a hardware device
• Bus driver
  – Manages a bus (PCI, USB, Firewire, etc.)
• Filter driver(s)
  – Sits on top of a function driver (upper filter) or a bus driver (lower filter)
  – Allows interception of requests
More Device Drivers Types

• Virtual Device Driver
  – A user mode driver used to allow (some) access to hardware from DOS/Win16
• Kernel Device Driver
  – Video Device Driver
    • Translates GDI, DirectX and OpenGL commands for a specific video card
  – File System Driver
  – Plug & Play Device Driver
    • Handles Plug & Play requests from the Plug & Play Manager
  – WDM Device Driver
    • A Plug & Play device driver, which also handles Power Management and optionally WMI
  – Legacy Device Driver
    • Non Plug & Play Device Driver (usually NT 4 driver)
    • Software Driver
Kernel Device Driver

• The only one that can touch hardware, handle interrupts, etc.
  – Has a SYS extension
  – Its routines always run in kernel mode
  – Always uses the kernel mode stack
    • Limited in size: 12KB (32 bit), 24KB (64 bit)
    • No documented way to enlarge it
  – Unhandled exceptions will crash the system
    • Producing the infamous “Blue Screen of Death”
Kernel Device Driver

• Usually invoked by a user mode code (**ReadFile**, **WriteFile**, **DeviceIoControl**)

• Fully interruptible, but not always pre-emptible

• System handles all device independent aspects of I/O
  – No need for assembly

• Layered model
Anatomy of a Driver

- A driver exports functionality, callable by the I/O system.
Driver and **Device Objects**

- Drivers are represented in memory using a DRIVER_OBJECT structure
  - Created by the I/O system
  - Provided to the driver in the `DriverEntry` function
  - Holds all exported functions
- Device objects are created by the driver on a per-device basis
  - Represented by the `DEVICE_OBJECT` structure
  - Typically created in the Driver’s `AddDevice` routine
  - Several can be associated with a single driver object
- I/O system is device-centric, not driver-centric
Invoking a Driver

- **ReadFile**: call `ReadFile` (user mode)
  - Return to caller
- **NtReadFile**: call `NtReadFile` (kernel mode)
  - int $0xE$/sysenter
  - Return to caller

**Kernel32.DLL**

- **NtReadFile**: call `NtReadFile`
- **NtOskrnl.EXE**: check parameters
  - call driver
  - block, or not
  - return to caller
- **NtOskrnl.EXE**: initiate I/O
  - return to caller

**CodeValue College**
Accessing Devices

- A client that wants to communicate with a device, must open a handle to the device
  - `CreateFile` or `CreateFile2` from user mode
    - The `System.IO.FileStream` class in .NET
  - `ZwCreateFile` from kernel mode
- `CreateFile` accepts a "filename" which is actually a device symbolic link
  - "file" being just one specific case
  - For devices, the name should have the format `\\.\name`
    - Cannot access non-local device
    - Must use double backslashes "\\.\name" in C/C++ or in C++ 11:
      - `R"(\\.\name)"`
Asynchronous I/O

• The I/O manager supports an asynchronous model
  – Client initiates request, may not block, and get a notification later
• Device drivers must be written with asynchrony in mind
  – Should start an operation, mark the IRP as pending and return immediately
• The I/O manager supports several ways of receiving a notification when the operation completes
• To use I/O asynchronously, `CreateFile` must be called with the `FILE_FLAG_OVERLAPPED` flag
• Other I/O functions must provide a non-null OVERLAPPED structure pointer
What is WDM?

• A model for writing device drivers
  – Source compatibility between Windows 98/ME and Windows 2000/XP
  – Standard handling for Plug & Play (P&P), Power Management and Windows Management Instrumentation (WMI)
  – Supports a wide range of buses (PCI, USB, IEEE1394 and more)
    • Extensible to support future buses
  – Supports a wide range of device classes (HID, Scanners, Modems, etc.)

• Not included in WDM
  – File system drivers
  – Video drivers
What is WDF?

• A framework and a set of tools for the development of Windows device drivers

• Framework(s)
  – Kernel Mode Driver Framework (KMDF) – developing drivers running in kernel mode
  – User Mode Driver Framework (UMDF) – developing drivers running in user mode

• KMDF is a replacement for the Windows Driver Model (WDM)

• UMDF allows creation of certain drivers running in user mode
Why WDF?

• What’s wrong with WDM?
  – Well, nothing... but...

• WDM
  – Originally created to get source-level compatibility between Windows 2K/XP and Windows 98/ME
  – Very flexible but complex
  – Poor DDI design
  – Very little default behavior support
  – Plug & Play and power management requires a lot of hard-to-write boilerplate code

• Most driver types must run in kernel mode
  – Potentially risking a blue screen
  – Harder to write and debug
Windows Driver Foundation

• User mode and kernel mode drivers
  – Same object model and concepts

• Consistent object model
  – Properties, methods and events

• Much boilerplate code and default handling already part of the library
  – Drivers only need to register for interesting events

• Object hierarchy simplifying object lifetime management

• Versionable with side by side support

• Simple but not simpler, and extensible
Universal Windows Derivers (NT 10.X)

• Enable developers to create a single driver that runs across multiple different device types
  – Desktop, Mobile, IoT Core, Windows Server 2016, Xbox One, HoloLens
  – The driver model equivalent to the UWP user mode API
  – A Universal Windows driver calls only device driver interfaces (DDIs) that are part of UWP
    • These DDIs are marked as Universal on the corresponding MSDN reference pages
  – Use the WDM or WDF driver model
  – New tools to package and install universal driver on the various platforms
What is Plug & Play?

• Automatic and dynamic recognition of installed hardware
  – Hardware detected at initial system installation
  – Recognition of PnP hardware changes between boots
  – Run-time response to PnP hardware changes

• Dynamic loading and unloading of drivers in response to hardware insertion or removal

• Hardware resource allocation and reallocation
  – PnP manager may reconfigure resources at run-time in response to new hardware requesting resources that are already in use
Device Enumeration

• Upon boot, the P&P Manager performs enumeration of buses and devices
  – Starts from an imaginary “Root” device
  – Scans the system recursively to walk the device tree
    • Creates a PDO (Physical Device Object) for each physical device
    • Loads lower filter drivers (if exist)
      – They create their FiDOs (Filter Device Object)
    • Loads “the driver”
      – It should create the FDO (Functional Device Object)
    • Loads upper filter drivers (if exist)
      – They create their FiDOs
Device Node ("DevNode")

- Represents a stack of devices
  - PDO: Physical Device Object
    - Created by the bus driver
  - FiDO: Filter Device Object
    - Optional lower/upper device objects
  - FDO: Functional Device Object
    - The "actual" WDM driver created device object
Important Registry Keys

- “Hardware” (Instance) keys
  - HKLM\System\CurrentControlSet\Enum
  - Information about a single device

- “Class” keys
  - HKLM\System\CurrentControlSet\Control\Class
  - Information about all devices of same type

- “Software” (Service) key
  - HKLM\System\CurrentControlSet\Services\drivername
    - Information about a specific driver
What is an IRP?

• A structure representing some request
  – The IRP structure is defined in `<wdm.h>`
  – Contains all details needed to handle the request (codes, buffers, sizes, etc.)
  – Always allocated from non-paged pool
  – Accompanied by a set of structures of type IO_STACK_LOCATION
    • Number of structures is the number of the devices in this DevNode
    • Complements the data in the IRP
  – Broadly speaking, the data needed for handling the request is “split” between the actual IRP object and the “current” I/O stack location
IRP Flow

P&P manager

I/O Manager

Power Manager

Processing on the way down

Register completion routine

Register completion routine

Processing on the way up

Call completion routine

Call completion routine

Complete request

Windows Internals

FiDO

FDO

PDO

Complete request
Typical IRP Processing

App calls (e.g.) ReadFile

Validate request

Dispatch routine

Start I/O routine

ISR routine

DPC routine

User mode

Kernel mode

Requesting thread context

Arbitrary thread context

Device interrupt

DPC (software) interrupt

User mode

Kernel mode

Requesting thread context

Arbitrary thread context

Device interrupt

DPC (software) interrupt
Referencing User Buffers

- Buffers provided in user space are not generally accessible from an arbitrary thread context and/or high IRQL (>=2)
- The I/O system provides ways to mitigate that
- Buffered I/O
  - Transfer is to and from an intermediate buffer in system address space
  - I/O Manager does all of the setup work
- Direct I/O
  - Transfer is to or from user's physical pages
  - I/O Manager does most of the setup work
- Neither I/O
  - No help from the I/O manager
Buffered I/O

Kernel space

User space

q=Irp->AssociatedIrp.SystemBuffer

copy

RAM
Direct I/O

Kernel space

User space

Irp->MdlAddress

q=MmGetSystemAddressForMdlSafe(Irp->MdlAddress, ...)

MDL

RAM
I/O System Objects

- Driver object (**DRIVER_OBJECT**)
  - Represents the driver for the I/O system
  - Includes dispatch routine pointers, **AddDevice** routine pointer, Unload routine pointer, etc.
  - Created by the kernel, passed to the driver and filled by it

- Device object (**DEVICE_OBJECT**)
  - Defines a specific device (usually hardware)
  - Associated with File objects
  - Allows for driver-defined extensions
  - Provides a DPC object for after Interrupt processing
  - Created by the driver using **IoCreateDevice**
    - Several may be created
I/O System Objects

• Interrupt object
  – Completely opaque
  – Represents an interrupt source
  – Associates an interrupt vector with an Interrupt Service Routine (ISR)
  – Created by calling `IoConnectInterrupt`

• Adapter object
  – Defines characteristics of a DMA controller
  – Either system DMA or bus master controller
  – Supports arbitration for access to DMA related resources
  – Created with `IoGetDmaAdapter`
Asynchronous Procedure Call (APC)

- An object representing a callback function to be called by a specific thread
- Example: after an interrupt may need to execute code in a particular (non-arbitrary) thread
  - e.g. reporting I/O completion
- APC forces a specified thread to run a specified routine in that thread
  - Makes the thread ready to run if it was waiting
  - At the end of the APC routine, the thread might go back to waiting (if it was in that state earlier)
- APC takes precedence over "normal" thread code
  - Only within that thread (does not affect its priority)
- Kernel mode API is exported but not documented
APC Types

• Special kernel APCs
  – Run in kernel mode, at IRQL APC_LEVEL (1)
  – Always deliverable
  – Used for I/O completion report from "arbitrary thread context"

• User mode APCs
  – Used for I/O completion callback routines (see ReadFileEx, WriteFileEx)
  – Only deliverable when thread goes into "alertable state" (see docs for e.g. SleepEx, WaitForSingleObjectEx, MsgWaitForMultipleObjectsEx)
  – Can be explicitly queued with QueueUserApc (Win32)
I/O Prioritization

• To support I/O Prioritization in the Kernel:
  – The I/O manager has IRP queue for each priority
  – There are mechanisms to prevent I/O starvations
    • In case of a detection of a starvation, the PsBoostThreadIo and IoBoostThreadIoPriority are called
  – IRP has a priority field
    • This is a hint, not all device driver use it
    • However Idle class I/O is managed by the system storage class driver, so it is automatically applied to all storage devices
Power Management

• Why manage power?
  – Increase battery life in portable systems
  – Biggest savings in servers

• Energy Star requirements
  – When idle, computer can consume only 15 watts

• Why implement Energy Star?
  – Requirement for selling to US government
  – Many Europeans and Asian governments have similar requirements
Power Management Until Win2K

• Mainly managed by hardware
• Advanced Power Management (APM) BIOS
  – On x86 machines runs “hidden”
    • Unable to hook
  – Can interrupt the CPU at any time
• Problems
  – Scheme is x86-specific
  – Optimized for DOS – not NT
  – Does not allow devices to wake the system
WDM Power States

- System power states
  - Working (S0)
    - System is generally operational and running at full speed
  - Sleeping (S1-S3)
    - System appears off and is not performing any computational tasks. CPU is off. Memory remains powered up
  - Hibernate (S4)
    - Physical memory is written to disk. CPU is off
  - Off (S5)
    - No power to the system
- Waking
  - Transition from sleep state to Working state
WDM Power States

- **D0** - Mandatory, Highest level of power consumption and performance
- **D1**
  - Less power consumption than D0 but more than D2
  - Restore latency shorter than D2
- **D2**
  - Less power consumption than D1
  - Restore latency longer than D1
- **D3 (Hot & Cold)** - Mandatory
  - No power to device
  - Longest restore time
  - The D3hot sub-state requires the device to remain accessible on its parent bus so that it can respond to bus-specific software commands
    - This requirement, and the power used to meet it, are removed in D3cold
Power Transitions

System Power States

Device Power States

S0
S1
S2
S3
S4
S5
D0
D1
D2
D3
hot/cold

Windows Internals
Power Control Panel

- Specifies policy to the Power Manager
- ACPI machines can program the power button and sleep button (lid switch on laptops)
  - Shutdown, standby, hibernate, off
  - None (Sleep button only)
- Power Manager battery monitor alarm
  - Place machine in standby, hibernate or off state
  - Notification messages
  - Run program
How The System Powers Down

Power Manager sends IRP_MN_QUERY_POWER To all devices

Everyone agrees?

No

Power Manager sends IRP_MN_SET_POWER With old power state

Yes

Power Manager sends IRP_MN_SET_POWER With new power state
Installing Drivers

• Use an INF file
  – Required for WDM drivers
  – Format is new, but loosely based on Windows 95 INF file format

• When a new type of hardware is detected by the PnP Manager, the system device installer searches the `%SystemRoot%\INF` for a suitable INF file
  – If not found, a dialog box will pop up requesting the user to provide one

• INF file is not “run” from beginning to end
Driver Verifier

• A tool that allows monitoring device drivers activities and operations
• Different UI in Windows 2000 and XP
• Can be operated from the command line as well
  – Can even change some settings without a restart
• Does not require any special code or awareness from the driver writer
• Can monitor any driver
Driver Verifier Options

• Automatic Checks
  – Improper IRQL usage, improper Spin Lock acquisition or release, driver resources

• Special memory pool
  – The driver’s allocations will be made from a special pool and monitored for overruns, underruns and illegal usage

• Forcing IRQL checking
  – Forces paging of all paged driver code/data, forcing checks of correct behavior in IRQL and Spin Lock usage

• Low resource simulation
  – Causes random failure in memory allocations
  – Checks driver’s ability to cope
Driver Verifier Options

• Memory Pool Tracking
  – Checks whether the driver has freed all its allocated memory when unloaded

• I/O Verification
  – IRPs are allocated from a special pool, monitors driver’s I/O handling

• DMA Verification
  – Monitors DMA buffer and map registers usage

• Deadlock Detection
  – Monitors usage of spin locks, mutexes and fast mutexes to detect possible deadlocks

• SCSI Verification
  – Monitors SCSI routine usage, excessive delays, etc.
Summary

• The I/O system provides a consistent model of flow for use by applications
• Device drivers are used to implement the device specific functionality
  – Integrate with the rest of the system
• The plug & play manager controls automatic recognition and loading of drivers
Module 8

NETWORKING
Agenda

• Introduction to Windows Networking
• Networking Components
• Networking APIs
• Summary
Windows Networking

- Network support is integrated with the I/O system and the Windows API
- The network stack consists of services, APIs, protocol and network device drivers
- Broadly implements the classic OSI (Object System Interconnection) networking model
Networking Components

• Networking APIs
  – Protocol-independent way for apps to communicate across a network
  – Socket (TCP/UDP), Named pipe, mailslot, File sharing, RDP, etc.

• Transports Driver Interface
  – NDIS protocol drivers
  – Process IRPs coming from TDI clients
  – Can add protocol specific headers (e.g. TCP, UDP)
Networking Components

• NDIS library (ndis.sys)
  – Kernel mode DLL that encapsulates network adapter drivers

• NDIS miniport drivers
  – Kernel mode driver that interfaces with a specific hardware network card
  – They are wrapped by the NDIS library so they are easier to write

• Winsock Kernel (WSK) – transport independent kernel-mode networking API
  – Replaces the old TDI
Networking Components

- The **Windows Filtering Platform**
  - A set of API and system services that provide a platform for creating network filtering applications
  - Replaces the TDI Filter interface
  - Windows Firewall with Advanced Security (WFAS) is implemented using WFP
  - Provides a packet filtering infrastructure for both IP 4/6
    - Allows for data filtering, modification, and re-injection
  - Many other **features**
Networking APIs

- **Windows Sockets** (Winsock) and **Winsock Kernel (WSK)**
- **Remote Procedure Calls (RPC)**
- **Web Access APIs**
  - **WinInet** – FTP, HTTP
  - **WinHttp** - HTTP
- **Named pipes** and **Mailslots**
- **Distributed Component Object Model (DCOM)**
- **Message queuing** (**MSMQ**)
- **Peer-to-Peer**
Network Redirectors

- Windows supports multiple redirectors for accessing remote resources such as shared file
- **Multiple Provider Router (MPR)**
  - A DLL (`\Windows\System32\Mpr.dll`) which determines which network to access when using the Windows WNet API for browsing remote file systems
- **Multiple UNC Provider (MUP)**
  - A driver (`\Windows\System32\Drivers\Mup.sys`) which determines which network to access when using the Windows I/O API
Protocol Drivers

• Kernel mode drivers responsible for translating requests into the underlying protocol
• Separating APIs from protocols allows using more than one protocol to implement an API
• In Windows NT 6.X Microsoft rewrite the whole network stack around TCP/IP (IP V4 and IP V6)
• \Windows\System32\Drivers\Tcpip.sys
  – TCP/UDP/IP/ARP/ICMP and IGMP
NDIS Drivers

• NDIS (Network Driver Interface Specification) was developed to allow protocol drivers to communicate with network adapter drivers in a device independent way

• Network drivers that comply with the specification are NDIS drivers (or miniport drivers)

• Windows supplies Ndis.sys, a kernel mode helper library for communicating with and implementing an NDIS miniport driver
Summary

• Networking support is integrated with the rest of the system
• Various networking APIs exist to facilitate different programming models
• Windows provides extensibility and flexibility to further grow the networking system
Appendix A

INTRODUCTION TO UNIVERSAL WINDOWS PLATFORM
The Windows Runtime (>= NT 6.2)

- The new runtime for Windows Store applications
- Based on an enhanced version of COM
  - Reference counted objects
  - Metadata with same format as CLR’s metadata
- Projected to other environments
  - C++/CX – most direct projection
  - .NET platform (any language)
  - JavaScript
- Subset of Win32 and .NET work with Store apps
WinRT Objects

• COM objects implementing IUnknown and a new interface, IInspectable

```cpp
class IInspectable : public IUnknown {
public:
    virtual HRESULT __stdcall GetIids(
        ULONG *iidCount,
        IID **iids) = 0;

    virtual HRESULT __stdcall GetRuntimeClassName(
        HSTRING *className) = 0;

    virtual HRESULT __stdcall GetTrustLevel(
        TrustLevel *trustLevel) = 0;
};
```
Language Projections

Windows Runtime Object (or Component)
Written in C++, C#, VB

- Windows Metadata
- Projection
  - C++ App
- Projection
  - CLR
- Projection
  - C#/VB App
- Projection
  - Chakra
- Projection
  - HTML App
VC++ Projections for WinRT

C++/CX:
Component Extensions
Language extensions
Well aligned with C++/CLI

Consumption: ^, ref new

Authoring: ref/value/interface
class, property, delegate,
event, generic interfaces

WRL:
Windows Runtime Library
Library-like syntax
No exceptions (HRESULT model)

Consumption: ComPtr<> , make<> 

Authoring: Lots of MIDL and attributes
# C++/CX Extensions

<table>
<thead>
<tr>
<th>Key Bindings</th>
<th>Feature</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1. Data Types</strong></td>
<td>ref class</td>
<td>Reference type</td>
</tr>
<tr>
<td></td>
<td>value class</td>
<td>Value type</td>
</tr>
<tr>
<td></td>
<td>interface class</td>
<td>Interface</td>
</tr>
<tr>
<td></td>
<td>property</td>
<td>Property with get/set</td>
</tr>
<tr>
<td></td>
<td>event</td>
<td>“Delegate property” with add/remove/raise</td>
</tr>
<tr>
<td></td>
<td>delegate</td>
<td>Type-safe function pointer</td>
</tr>
<tr>
<td></td>
<td>generic</td>
<td>Type-safe generics</td>
</tr>
<tr>
<td><strong>2. Allocation</strong></td>
<td>ref new</td>
<td>Reference-counted allocation</td>
</tr>
<tr>
<td><strong>3. Pointer &amp; Reference</strong></td>
<td>^</td>
<td>Strong pointer (“hat” or “handle”)</td>
</tr>
<tr>
<td></td>
<td>%</td>
<td>Strong reference</td>
</tr>
</tbody>
</table>
Application Binary Interface (ABI)

• WinRT boundary

• Components adhering to the ABI are consumable in other languages
  – Simpler with the provided projections

• Public surface must conform to WinRT types
  – Implementation can use any type
WinRT Components

- DLLs with metadata (.winmd files)
- Cannot be distributed stand alone through the store
- Cannot be shared between applications
- Only sealed classes are allowed
- Only WinRT types can be passed across the interop boundary
- Can use the resulting library from C++, C# or JavaScript Store apps
Asynchrony in WinRT

• Many WinRT APIs are asynchronous in nature
  – Call to start an operation
  – Returns immediately
  – Register to be notified when result is available
    • Handle the result

• Basic rule: if something may consume more than 50msec, make it asynchronous

• Some APIs provide asynchronous access only
Universal Windows Platform

- One Operating System
  - One Windows core for all devices
- One App Platform
  - Apps run across every family
- One Dev Center
  - Single submission flow and dashboard
- One Store
  - Global reach, local monetization
  - Consumers, Business & Education
Windows 10 operating system

Universal Windows Platform

C++
.Net languages
.Net runtime

C++ & CX
.Net languages

XAML
DirectX
HTML

Bridging technologies

Obj.C iOS
Java Android
Web hosted
Win32 desktop

.MFC
.WPF
.WF

.WPF
.WF
.MFC
Windows 10 IoT Core

• Use the same API, SDK, Visual Studio to develop Universal App
  – Extend UWP with device specific API
  – Run your desktop app on the PI
    • Extend it to control and to be controlled by hardware devices
  – You can Use Arduino Wiring Pi on Windows 10 IoT devices
    • C++ library that mimic the Arduino library
Setup Raspberry Pi 2 Device

Set up a new device

First, let's get Windows 10 IoT Core on your device.

Device type
- Raspberry Pi 2
- Windows 10 IoT Core for Raspberry Pi 2

Insert an SD card into your computer.
Note that this process will erase all content on your card.

View the list of recommended SD cards

Drive
- 62Gb Mass Storage Device USB Device

I accept the software license terms

Download and install

Flashing your SD card
18 MB downloading - 3%

Cancel
NOOBS OS Selection Menu

- Raspbian [RECOMMENDED]
  A community-created port of Debian jessie for the Raspberry Pi

- OpenELEC_RPI2
  OpenELEC is a fast and user-friendly Kodi Entertainment Center distribution.

- Data Partition
  Adds an empty 512MB ext4 format partition to the partition layout.

- OSMC_PI2
  A fast and feature filled open source media center

- Windows 10 IoT Core
  Windows for devices. Enables you to build projects and apps.

Disk space

- Needed: 0 MB
- Available: 28324 MB
Hello World – IoT Windows Universal App

• Start Coding
RPi 2 & 3 GPIO
using Windows.Devices.Gpio;

public void GPIO()
{
    // Get the default GPIO controller on the system
    GpioController gpio = GpioController.Default();
    if (gpio == null)
        return; // GPIO not available on this system

    // Open GPIO 5
    using (GpioPin pin = gpio.OpenPin(5))
    {
        // Latch HIGH value first. This ensures a default value when the pin is set as output
        pin.Write(GpioPinValue.High);

        // Set the IO direction as output
        pin.SetDriveMode(GpioPinDriveMode.Output);

    } // Close pin - will revert to its power-on state

    // Timer Tick Function
    private void Timer_Tick(object sender, object e)
    {
        if (pinValue == GpioPinValue.High)
        {
            pinValue = GpioPinValue.Low;
            pin.Write(pinValue);
            LED.Fill = redBrush;
        }
        else
        {
            pinValue = GpioPinValue.High;
            pin.Write(pinValue);
            LED.Fill = grayBrush;
        }
    }
}
RGB Blinky Demo

• Using C++ and the WiringPi library
• Using Universal App to Control an RGB Led
• RPI GPIO Schema
Windows app

- A single app package
  - Running on any device
  - Testing for capabilities
  - Adjusting to devices
Adaptive design

• Responsive design
  – Flexible layout responds to small changes
  – Many controls handle basic responsiveness

• Adaptive design
  – Smart layout adjusts to large changes
  – Features like visual states aid in this design

• Tailored design
  – A device-specific app can simplify design
  – Some devices have unique design languages
Adaptive design
Tailored design
Continuum for convertibles and Phones
Adaptive code

• A compatible binary across devices
  – Universal API with device-specific implementation

• Light up our app with capabilities
  – Testing for capabilities and namespaces
Test capabilities at runtime

- Use Adaptive Code to light-up your app on specific devices

```javascript
var api = "Windows.Phone.UI.Input.HardwareButtons";
if (Windows.Foundation.Metadata.ApiInformation.IsTypePresent(api)) {
}
```
Demo

HELLO UWP
Get Started

• This short lecture will provide the basic knowledge
• However this is defiantly not enough
  – Start here: https://dev.windows.com/en-us/getstarted
  – Download visual studio Community 2015
  – Follow this “Create a Hello World app” lab
  – Learn more about the XAML Platform
  – Learn C# - this is for absolute beginners

• Feel free to ask question – alonf@codevalue.net
Visual Studio IDE

• Every project type
  – Desktop, Windows, Phone, Service, Web, Game, More...

• Every developer task
  – Code edit, Architecture design, UX design, Debug, Profile, Review, Test, More...

• Every development language
  – C++/CX, C#, Visual Basic, JavaScript, XAML, HTML, More...

• Visual Studio Online
  – Source repository, project management, bug tracking, More...
Blend for Visual Studio

• The XAML Developer’s IDE
  – Always part of Visual Studio
  – Uses the Visual Studio shell
  – Full auto-complete & intellisense

• Validation
• Snippets
• Peek
  – File & solution management
  – Resource management
  – Data management
  – Animation
  – States
Developer unlock

For developers

Use developer features
These settings are intended for development use only.

- Don't use developer features
- Sideload apps

Device discovery

- Off

Settings

Update & security

Use developer features
These settings are intended for development use only.

- Don't use developer features
- Sideload apps

- Developer mode

Learn more.
.NET Native

• Next generation compiler in the cloud
  – Every Windows apps, only Windows app (right now)

• Apps use the standard C++ optimizer
  – As optimizer performance improves, so does .Net native

• Apps with .Net bootstrapper
  – Includes garbage collection

• There is no runtime
  – This is machine code
Real benefits with .Net Native

• 50% faster average startup time
• 14% less average memory usage
C#

• A multi-paradigm programming language encompassing strong typing, imperative, declarative, functional, generic, object-oriented and component-oriented programming disciplines

• Developed by Microsoft and approved as a standard by Ecma (ECMA-334) and ISO (ISO/IEC 23270:2006)

• C# is intended to be a simple, modern, general-purpose, object-oriented programming language

• The most recent version is C# 6.0, which was released on July 20, 2015
C# Basics

• The platform – Compiler, Jitter, CLR, GC, Libraries (references), .NET Native
• Types, Value types and reference types (boxing), Metadata (ILDASM)
• Class and Interfaces, explicit interface, polymorphism (abstract, override, new), partial
• Methods, ref and out parameters, params, lambda expression
• Exceptions, checked and unchecked, finally
• Control Flow (if-else, while, do-while, for, foreach, switch)
• Operator overloading
• Properties
• Delegates & Events
• Attributes
• Generic, constraints
• Enumerators (yield return)
• LINQ
• Task, Async/Await

UI Designer and XAML Fundamentals
UI Designer and XAML Fundamentals

• Visual Studio
  – UI Designer
  – Toolbox
  – Document Outline
  – Properties View
  – Device Emulator

• What is XAML?
• Basic XAML
• Type Converters
• Markup Extensions
• Naming Elements
• XAML Rules
• Summary
UI Designer

- Visual Studio UI Designer provides quick and easy way to produce rich UI interface
- Drag and drop UI controls from the toolbox into the designer
- The UI designer provides an easy way to select, resize, align, transform and more, using only the mouse
- The UI designer generates XAML and automatically updated when XAML changes
Toolbox

• The toolbox groups UI controls available at design time
• 3rd party and custom controls are also available from the toolbox
• Simple drag and drop a control from the toolbox to the designer canvas
• Element from the toolbox can be also drag and drop directly into XAML
• Controls can be searched within the search box and can be sorted
• Additional groups can be created
Device View

- The device view provides an easy way to change page layout properties
- Page orientation: Landscape or Portrait
- Screen resolutions:
Document Outline

• The document outline view displays the logical tree of UI elements, each element name or type and an icon of the element’s type.

• Each element in the hierarchy can be design-time locked and hidden.

• The document outline is very useful to navigate between UI elements, especially in complex XAML files.
Properties View

• The properties view provides an easy a rapid way to:
  – Search for an element’s property by its name
  – Set simple property value using plain text
  – Set complex property value using designers
  – Easily select color brushes, styles, font size, transformations and more
  – Register element’s events
  – Arrange properties in groups
  – Create data bindings
  – Reset to default values
What is XAML?

• XML based language
• Enable separation of UI and behavior (code)
• Windows Phone related tools emit XAML
• XAML allows
  – Creation of objects
  – Setting of properties
  – Connection to events
  – Custom behaviors
• XAML cannot call methods directly
XAML vs. Code

• Anything that can be done in XAML can be done in code
  – But not vice versa
• XAML is usually shorter and more concise than the equivalent code
  – Thanks to type converters and markup extensions
• XAML should be used for initial UI
• Code will handle events and change items dynamically
Simple XAML Example

• Visual Studio UI designer generates XAML on each control picked from the toolbox
• XAML Can be visually viewed in the UI designer
XAML Namespaces

• The default XAML namespace is assigned a value that is mapped to some of the runtime namespaces contain UI elements
• Other XAML namespaces may be mapped to custom namespaces and other runtime namespaces
• The “x” namespace is mapped to a special namespace, contains XAML parser specific types
• XAML namespace can be defined on each element level
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <Button x:Name="buttonOk"
            Width="200"
            Height="200"
            Content="OK"
            Click="buttonOk_Click" />
</Grid>
Elements and Attributes

• Elements with type names only designate object creation (via the default constructor)

• Attributes indicate property or event values
  – Event values are event handlers (methods) names

• Complex properties are designated using a `<Type.Property>` element
ContentPanel - place additional content here

```xml
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <Button x:Name="buttonOk"
        Width="200"
        Height="200"
        Content="OK"
        Click="buttonOk_Click">
        <Button.Background>
            <LinearGradientBrush EndPoint="0.5,1"
                StartPoint="0.5,0">
                <GradientStop Color="#FFB2D9FF" Offset="0.004"/>
                <GradientStop Color="#FFB0D8FF" Offset="1"/>
                <GradientStop Color="#FF0A85FF" Offset="0.571"/>
            </LinearGradientBrush>
        </Button.Background>
    </Button>
</Grid>
```
XAML And Code Behind

• A root element, usually **Page** or **UserControl** classes, can have code behind file
• The name of the code behind file is correlated to the XAML file name
• For example: MainPage.xaml and MainPage.xaml.cs

```xml
<Page
    x:Class="UWPDemo.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:UWPDemo"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    mc:Ignorable="d">
    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}"
        <Image x:Name="image" HorizontalAlignment="Left" VerticalAlignment="Top" Stretch="Fill" Opacity="0.5"/>
        <Button x:Name="button" Content="Button" HorizontalAlignment="Left" Height="61" Width="127"/>
    </Grid>
</Page>
```
Child Elements

• Child elements (that are not property elements) can be one of
  – The **Content** property of the object
    • A property adorned with the attribute `Windows.UI.Xaml.Controls.ContentProperty`
  – Collection items
    • The object implements `IList` or `IDictionary`
  – A value that can be type-converted
Content Property

- A single property that is designated with the `ContentProperty` attribute on the type
- Allows shortening the markup

```xml
<Button Content="OK" />
</Button>

```

```xml
<Button>OK</Button>

```

```xml
<Button>
  <Button.Content>
    <Rectangle Fill="Blue"/>
  </Button.Content>
</Button>

```

```xml
<Button>
  <Rectangle Fill="Blue"/>
</Button>
```
Collection Items

- List (IList)

```
<ListBox>
<ListBox.Items>
<ListBoxItem Content="Item 1"/>
<ListBoxItem Content="Item 2"/>
</ListBox.Items>
</ListBox>
```

- ResourceDictionary

```
<ResourceDictionary>
<SolidColorBrush x:Key="br1" Color="Aqua" /> 
<Rectangle x:Key="rc1" Fill="Brown" />
</ResourceDictionary>
```
Summary of XAML Rules

- XML Element – create a new instance
- XML attribute – set a property or register an event
  - Type converter may execute
- **Type.Property** – set a “complex” property
- **ContentProperty** attribute – no need to specify **Type.Property**
- Property of type **IList** or **IDictionary**
  - Add child elements (XAML calls appropriate **Add** method)
  - Need a **x:Key** in case of a dictionary
Parsing and Using XAML

- Visual Studio compiles the XAML file to XBF (Binary XAML format) and embeds it as a resource
- The XBF file is parsed at runtime and the object tree created by the `InitializeComponent` method of the parent’s element class

```csharp
[GeneratedCodeAttribute("Microsoft.Windows.UI.Xaml.Build.Tasks"," 14.0.0.0")]
[global::System.Diagnostics.DebuggerNonUserCodeAttribute()]
public void InitializeComponent()
{
    if (_contentLoaded)
    {
        return;
    }
    _contentLoaded = true;

    Uri resourceLocator = new Uri("ms-appx:///MainPage.xaml");
    Application.LoadComponent(this, resourceLocator, ComponentResourceLocation.Application);
}
```
Naming Elements

• Elements can be named using the `x:Name` XAML attribute
• The code-behind file will contain a field with that name
• Elements deriving from `FrameworkElement` contain a `Name` property that can be used in code to locate elements
  – `x:Name` and `Name` cannot be set on the same element
## XAML Keywords

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Valid on</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>x:Class</td>
<td>Root element</td>
<td>The class that derives from the element type</td>
</tr>
<tr>
<td>x:ClassModifier</td>
<td>Root element, must be used with x:Class</td>
<td>The class visibility (public by default)</td>
</tr>
<tr>
<td>x:FieldModifier</td>
<td>Element, must be used with x:Name</td>
<td>Visibility of the field created behind the element</td>
</tr>
<tr>
<td>x:Key</td>
<td>Element that its parent implements IDictionary</td>
<td>Key in a dictionary</td>
</tr>
<tr>
<td>x:Name</td>
<td>Element</td>
<td>The element’s name, used for a field name for that element</td>
</tr>
<tr>
<td>X:Uid</td>
<td>Element</td>
<td>Identifies elements that should use localized resources</td>
</tr>
</tbody>
</table>
Mapping custom types to XAML namespaces

• You can define your own custom types in C# and then reference your custom types in XAML markup

• To use XAML for custom types - those that come from libraries other than the Windows Runtime core libraries:
  – You must declare and map a XAML namespace with a prefix
  – Use that prefix in element usages to reference the types that were defined in your library
  – You declare prefix mappings as xmlns attributes

• For example:
  – the attribute syntax to map a prefix myTypes to the namespace myCompany.myTypes is
  – xmlns:myTypes="using:myCompany.myTypes"
  – The representative element usage is: <myTypes:CustomButton/>
XAML Markup Extensions

- Represent some kind of "shortcut" that enables a XAML file to access a value or behavior that isn't simply declaring elements based on backing types
- In XAML attribute syntax, curly braces "{" and "}" indicate a XAML markup extension usage
- A XAML parser calls code that provides behavior for that particular markup extension
  - That code provides an alternate object or behavior result that the XAML parser needs
- Examples:
  - `{x:Bind} {Binding} {StaticResource} {ThemeResource} {TemplateBinding} {RelativeSource} {CustomResource} {x:Null}`
<Canvas.Resources>
  <Style TargetType="Border" x:Key="PageBackground">
    <Setter Property="BorderBrush" Value="Blue"/>
    <Setter Property="BorderThickness" Value="5"/>
  </Style>

  <Border Style="{StaticResource PageBackground}">
    ...
  </Border>
</Canvas.Resources>
XAML and .NET Events

• XAML has a syntax for attaching event handlers to objects in the markup

• You specify the name of the event as an attribute name on the object where the event is handled
  – For the attribute value, you specify the name of an event-handler function that you define in code

• The XAML processor uses this name to create a delegate representation in the loaded object tree, and adds the specified handler to an internal handler list

<Button Click="showUpdatesButton_Click">Show updates</Button>
Summary

• XAML is mainly used to create a Windows app user interface
• It declaratively allows object creation, property and event assignment
• A code-behind file will usually contain the procedural logic
• Sharing with designers is easier
• Tools such as Expression Blend generate XAML that is immediately usable
BASIC CONCEPTS
Agenda

• Logical and Visual Trees
• Dependency Properties
• Attached Properties
• Routed Events
• Attached Events
• Resources
• Summary
Logical Tree

• A tree of elements/controls making up the user interface

```xml
<Page x:Class="UWPDemo.MainPage"
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <StackPanel>
    <Button Content="OK" Background="Red" Margin="6"/>
    <ListBox Margin="6">
      <ListBoxItem Margin="2">
        <Border BorderThickness="4">
          <TextBlock Text="Hello" FontSize="20"/>
        </Border>
      </ListBoxItem>
      <ListBoxItem Content="Item 2"/>
    </ListBox>
  </StackPanel>
</Page>
```
Visual Tree

• The actual visual objects making up a logical tree
  – Generated from the control templates
Traversing the Trees

• The visual trees can be examined using the static classes
  \texttt{Windows.UI.Xaml.Media.VisualTreeHelper}

• \texttt{VisualTreeHelper}
  – \texttt{GetParent}, \texttt{GetChild}, \texttt{GetChildrenCount}
Dependency Properties

• **Dependency properties** are the workhorse of any XAML based technology
  – Provide the basis for many of UWP’s features
    • E.g. property changed, data binding, animations, styles, default value
  – Value may be provided by many entities, each with its own priority
    • Highest active level wins out
• Must be declare in a class derived from **DependencyObject**
• Must be “registered” by calling the **DependencyProperty.Register** method
• Most UWP properties are dependency
• UWP can handle Dependency Properties in a **new way**
// IsSpinningProperty is the dependency property identifier 
// no need for info in the last PropertyMetadata parameter, so we pass null
public static readonly DependencyProperty IsSpinningProperty = DependencyProperty.Register(
    "IsSpinning", typeof(Boolean),
    typeof(MainPage), null
);

// The property wrapper, so that callers can use this property through a simple
// MainPageInstance.IsSpinning usage rather than requiring property system APIs
public bool IsSpinning
{
    get { return (bool)GetValue(IsSpinningProperty); } 
    set { SetValue(IsSpinningProperty, value); } 
}
Dependency Property Precedence

- Animation
- Local value (also resource or data binding)
- Templated Property (Control or Data template)
- Style setters
- Default value from property metadata
Attached Properties

• Special kind of dependency properties
• May be “attached” to objects of different types than the declaring type
  – Declared with the static `DependencyProperty.RegisterAttached` method
• Allows “context” properties
  – E.g. `Canvas.Left` for elements that happen to be in a Canvas element
  – Can be set on any object
• XAML
  – An attribute with `Type.Property` syntax is used
• In code
  – The type exposes a `SetXxx` and a `GetXxx` with the element reference
Attached Properties Example

• XAML

```xml
<Canvas>
    <Button x:Name="cmdOK" Canvas.Left="30" Canvas.Top="20"
            Content="OK" Padding="10" FontSize="26">
    </Button>
</Canvas>
```

• Code

```csharp
Canvas.SetLeft(cmdOK, 30);
Canvas.SetTop(cmdOK, 20);
```

```csharp
cmdOK.SetValue(Canvas.LeftProperty, 30);
cmdOK.SetValue(Canvas.TopProperty, 30);
```
DependencyObject

• Represents an object that participates in the dependency property system
• Base class of many important UI-related classes: **UIElement**, **Geometry**, **FrameworkTemplate**, **Style**, and **ResourceDictionary**
• The dependency property system's primary function is to compute the values of properties, and to provide system notification about values that have changed
• Dependency & Attach property hosting support for the existing Windows Runtime dependency properties and custom properties
• The **Dispatcher** property for advanced threading scenarios
Routed Events

• UWP events are not implemented via the default .NET event implementation

• A routed event is an event that is potentially passed on (routed) from a child object to each of its successive parent objects

• The Windows Runtime supports the concept of a routed event for a set of events that are present on most UI elements
  – These events are for input and user interaction scenarios
  – They are implemented on the `UIElement` base class
  – For example:
    • `DoubleTapped`
    • `KeyDown`
    • `KeyUp`
Binary Resources

• The resources that can be used by any .NET application
  – Usually store bitmaps, icons, etc.

• Packaging
  – Embedded inside an Assembly
  – Loose files that are known to the application at compile time
  – Loose files that are not known to the application at compile time

• May be localized
Defining Binary Resources

• Binary resources can be embedded using Visual Studio by adding the resource and selecting an action type
  – **Resource**
    • Embedded inside the assembly
  – **Content**
    • Remains as a loose file, but an `AssemblyAssociatedContentFile` attribute is added with the relative path of the file

• Don’t use the **Embedded Resource** action! (UWP can’t use such resources)
Accessing Binary Resources

- Binary resources that are added to the project as Content or Resource can be referenced easily, even if loose files are involved (as long as they are in the folder of the executable or a subfolder).

```xml
<Button VerticalAlignment="Center" HorizontalAlignment="Center">
    <StackPanel Orientation="Horizontal">
        <Image Source="Assets/apple.png" Width="48" Height="48" Margin="4"/>
        <TextBlock Text="This is an Apple" FontSize="24"
            VerticalAlignment="Center" Margin="4"/>
    </StackPanel>
</Button>
```
Logical Resources

• Arbitrary named .NET objects, stored in the **Resources** collection property of an element
  – Typically for sharing the resource among child objects

• The **FrameworkElement** type defines a **Resources** property (of type **ResourceDictionary**
Creating and Using Resources

• Add a **Resources** property to some element
  – Usually a **Page** or the **Application**
  – Any child element can reference those resources

• Add the objects with a **x:Key** attribute (must be unique in this resource dictionary)

• Use the **StaticResource** markup extension with the resource key name
Resources Example

```xml
<StackPanel Margin="4" Orientation="Horizontal">
    <Button Margin="4" Content="1" Padding="4">
        <Button.Background>
            <LinearGradientBrush>
                <GradientStop Offset="0" Color="Blue" />
                <GradientStop Offset="1" Color="Yellow" />
            </LinearGradientBrush>
        </Button.Background>
    </Button>
    <Button Margin="4" Content="2" Padding="4">
        <Button.Background>
            <LinearGradientBrush>
                <GradientStop Offset="0" Color="Blue" />
                <GradientStop Offset="1" Color="Yellow" />
            </LinearGradientBrush>
        </Button.Background>
    </Button>
    <Button Margin="4" Content="3" Padding="4">
        <Button.Background>
            <LinearGradientBrush>
                <GradientStop Offset="0" Color="Blue" />
                <GradientStop Offset="1" Color="Yellow" />
            </LinearGradientBrush>
        </Button.Background>
    </Button>
</StackPanel>
```

```xml
<StackPanel Margin="4" Orientation="Horizontal">
    <StackPanel.Resources>
        <StackPanel.Resources>
            <LinearGradientBrush x:Key="back">
                <GradientStop Offset="0" Color="Blue" />
                <GradientStop Offset="1" Color="Yellow" />
            </LinearGradientBrush>
        </StackPanel.Resources>
    </StackPanel.Resources>
    <Button Margin="4" Content="1" Padding="4" Background="{StaticResource back}" />
    <Button Margin="4" Content="2" Padding="4" Background="{StaticResource back}" />
    <Button Margin="4" Content="3" Padding="4" Background="{StaticResource back}" />
</StackPanel>
```
Layout

- Layout is the arranging of user interface elements within some container
- Older technologies (e.g. Windows Forms) mostly used exact position and sizes
  - Limited in flexibility and adaptability
- UWP provides several layout panels that can control dynamically size and placement of elements
- Elements may advertise their size and position needs
- Two layout kind:
  - Static layout: Explicit pixel sizes and positions (Canvas)
  - Fluid layout: shrink, grow and reflow to adapt the visual space available
Size and Position of Elements

- Element sizing and positioning is determined by the element itself and its logical parent.
- A child element may request various settings.
- The parent panel does not have to comply.
Element Size

- **Width** and **Height** properties (from `FrameworkElement`)
  - Control the exact size of the element
  - Default value is `Double.NaN`
    - Meaning: be as large as it needs to be
    - usually a bad idea to use these properties
      - Prevents smart resizing by panel
      - Reasonable only in a Canvas

- **MinWidth**, **MinHeight**, **MaxWidth**, **MaxHeight** properties
  - Defaults are 0 (`MinWidth`, `MinHeight`), `Double.PositiveInfinity` (“Infinity” in XAML) (`MaxWidth`, `MaxHeight`)
More Size Properties

• Read only properties (UIElement)
  – DesiredSize
    • Set indirectly by elements to report their desired size to their parent
    • Internally used by panels
  – RenderSize
    • The actual size the element is rendered with
  – ActualWidth, ActualHeight
    • Just the components of RenderSize (RenderSize.Width, RenderSize.Height)
Margin and Padding

• Both of type **Thickness** (value type)
  – Maintains the properties **Left, Top, Right, Bottom** indicating distance from the corresponding edge

• **Margin** (from FrameworkElement)
  – The amount of space to add around the element

• **Padding** (from Control, and Border)
  – The amount of space to add around the content of the control

• In XAML, can supply one, two or four numbers
Visibility

• Visibility of elements is determined by the **Visibility** property (from UIElement) of the **Windows.UI.Xaml.Visibility** enumeration
  
  – **Visible**
    • The element is rendered and participates in layout
  
  – **Collapsed**
    • The element is invisible and does not participate in layout
Element Positioning

• Element position is determined by the containing panel policy
  – Simple X,Y is not the usual case

• Elements can indicate their position constraints or preferences to their containing panel
Alignment

- Alignment indicates what should be done with any extra space given to an element
- **HorizontalAlignment**
  - **Left**, **Right**, **Center**, **Stretch**
- **VerticalAlignment**
  - **Top**, **Bottom**, **Center**, **Stretch**

```xml
<StackPanel Margin="4">
  <Button HorizontalAlignment="Left" Background="Red">Left</Button>
  <Button HorizontalAlignment="Center" Background="Orange">Center</Button>
  <Button HorizontalAlignment="Right" Background="Yellow">Right</Button>
  <Button HorizontalAlignment="Stretch" Background="Lime">Stretch</Button>
</StackPanel>
```
Content Alignment

• Similar to element alignment
• What to do with extra space when the content is smaller than its control
• HorizontalContentAlignment
• VerticalContentAlignment

```xml
<StackPanel Margin="4">
  <Button HorizontalAlignment="Stretch" HorizontalContentAlignment="Left" Background="Red">Left</Button>
  <Button HorizontalAlignment="Stretch" HorizontalContentAlignment="Center" Background="Orange">Center</Button>
  <Button HorizontalAlignment="Stretch" HorizontalContentAlignment="Right" Background="Yellow">Right</Button>
  <Button HorizontalAlignment="Stretch" HorizontalContentAlignment="Stretch" Background="Lime">Stretch</Button>
</StackPanel>
```
Flow Direction

• The **FlowDirection** property indicates the flow of layout
  – **LeftToRight** (the default)
  – **RightToLeft**
• The **RightToLeft** setting reverses the meaning of “left” and “right”
Layout Panels

- **Layout panels** derive from the abstract `Windows.UI.Xaml.Controls.Panel` class
- Maintain a **Children** property of type `UIElementCollection` (its `ContentProperty`)
- Each child element can be a panel as well
  - Allows creation of complex and adaptive user interfaces
- UWP provides several built in panels
  - Custom layout panels can be created as well
UWP Layout Panels

- **Main layout panels**
  - **Canvas**
    - Arranges children in a 2D coordinate system
  - **StackPanel**
    - Arranges children in a horizontal or vertical “stack”
  - **VariableSizedWrapGrid**
    - Provides a grid-style layout panel where each tile/cell can be variable size based on content
  - **Grid**
    - Arranges children in a flexible grid
  - **RelativePanel**
    - Defines an area within which you can position and align child objects in relation to each other or the parent panel
Canvas

- Doesn’t support fluid UI
  - you control all aspects of positioning and sizing child elements
  - You typically use it for special cases like creating graphics or to define small static areas of a larger adaptive UI
  - You can use code or visual states to reposition elements at runtime
- Elements are positioned absolutely using Canvas.Top and Canvas.Left attached properties
- Layering can be explicitly specified using the Canvas.ZIndex
- If an element's size is not set explicitly, it sizes to its content
- Child content is not visually clipped if larger than the panel
- Child content is not constrained by the bounds of the panel
Canvas Example

```xml
<Page x:Class="CanvasDemo.MainPage"
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 xmlns:local="using:CanvasDemo"
 xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
 mc:Ignorable="d">

 <Canvas x:Name="_canvas" PointerPressed="OnPress" Background="{ThemeResource ApplicationPageBackgroundThemeBrush}"
>
  <Rectangle Fill="Blue" Width="100" Height="100" Canvas.Left="100" Canvas.Top="100"/>

 </Canvas>

</Page>
```

```csharp
private void OnPress(object sender, PointerRoutedEventArgs e)
{
    var ellipse = new Ellipse
    {
        Width = Radius, Height = Radius,
        Fill = new SolidColorBrush(Colors.BlueViolet),
    };
    _canvas.Children.Add(ellipse);
    var position = e.GetCurrentPoint(_canvas).Position;
    Canvas.SetLeft(ellipse, position.X - Radius/2);
    Canvas.SetTop(ellipse, position.Y - Radius/2);
}
```
StackPanel

- Stacks its elements in a vertical or horizontal “stack”
- **Orientation** property
  - *Vertical* (default) or *Horizontal*
- Alignment is ignored in the direction of stacking
- In the direction specified by Orientation property, an element sizes to its content
<Page x:Class="StackPanelDemo.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:local="using:StackPanelDemo"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
mc:Ignorable="d"
Width="400"
Height="200">
  <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.ColumnDefinitions>
      <ColumnDefinition/>
    </Grid.ColumnDefinitions>
    <StackPanel Orientation="Horizontal" Grid.Column="0">
      <Button Margin="4" Background="Red" VerticalAlignment="Stretch">One</Button>
      <Button Margin="4" Background="Orange" VerticalAlignment="Stretch">Two</Button>
      <Button Margin="4" Background="LightGreen" VerticalAlignment="Stretch">Three</Button>
    </StackPanel>
    <StackPanel Orientation="Vertical" Grid.Column="1">
      <Button Margin="4" Background="Red" HorizontalAlignment="Stretch">One</Button>
      <Button Margin="4" Background="Orange" HorizontalAlignment="Stretch">Two</Button>
      <Button Margin="4" Background="LightGreen" HorizontalAlignment="Stretch">Three</Button>
    </StackPanel>
  </Grid>
</Page>
The **Grid**

- The most versatile and useful panel
- Usually used as the top-level panel
  - Visual Studio and Expression Blend windows/user controls start this way
- Supports fluid resizing of child elements
  - You can use code or visual states to reposition and reflow elements
- Arranges its children in a multi-row and multi-column way
  - Their sizes and number can be manipulated in interesting ways
  - Somewhat similar to an HTML table
- Child content is visually clipped if larger than the panel
Creating a Grid

• For rows
  – Set the `RowDefinitions` property
  – Add a `RowDefinition` object for each row
  – Set any special properties

• For columns
  – Set the `ColumnDefinitions` property
  – Add a `ColumnDefinition` object for each column
  – Set any special properties

• For each element
  – Set the `Grid.Row` and `Grid.Column` attached properties (default is 0, 0)
Sizing Rows and Columns

• By default, all rows are of equal height and all columns are of equal width
  – Can change the height of a row using the `RowDefinition.Height` property
  – Can change the width of a column using the `ColumnDefinition.Width` property
  – Each one of type `GridLength`
  – The unit is controlled by the `GridUnitType` property
    • *Auto* – size as required by content
    • *Pixel* – (double value) size is the number specified
    • *Star* – size is a weighted proportional (default)
      – “*”, “2*”, etc. in XAML

• Spanning
  • A row may span more than one column and vice versa
  • Can be set by the `Grid.RowSpan` and `Grid.ColumnSpan` attached properties
    – Default for both is 1
Grid Sample

<table>
<thead>
<tr>
<th>Column_1</th>
<th>Auto</th>
<th>The column will size to fit its content.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column_2</td>
<td>*</td>
<td>After the Auto columns are calculated, the column gets part of the remaining width. Column_2 will be one-half as wide as Column_4.</td>
</tr>
<tr>
<td>Column_3</td>
<td>44</td>
<td>The column will be 44 pixels wide.</td>
</tr>
<tr>
<td>Column_4</td>
<td>*2</td>
<td>After the Auto columns are calculated, the column gets part of the remaining width. Column_4 will be twice as wide as Column_2.</td>
</tr>
</tbody>
</table>

```xml
<Grid>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="Auto"/>
    <ColumnDefinition/>
    <ColumnDefinition Width="44"/>
    <ColumnDefinition Width="2*"/>
  </Grid.ColumnDefinitions>
  <TextBlock Text="Column 1 sizes to its content." FontSize="24"/>
</Grid>
```
RelativePanel

- A layout container that is useful for creating UIs that do not have a clear linear pattern
- Elements are arranged in relation to the edge or center of the panel, and in relation to each other
- Child content is visually clipped if larger than the panel

```xml
<RelativePanel BorderBrush="Gray" BorderThickness="10">
  <Rectangle x:Name="RedRect" Fill="Red" MinHeight="100" MinWidth="100"/>
  <Rectangle x:Name="BlueRect" Fill="Blue" MinHeight="100" MinWidth="100" RelativePanel.RightOf="RedRect"/>
  <!-Width is not set on the green and yellow rectangles. It's determined by the RelativePanel properties. -->
  <Rectangle x:Name="GreenRect" Fill="Green" MinHeight="100" Margin="0,5,0,0" RelativePanel.Below="RedRect"
    RelativePanel.AlignLeftWith="RedRect" RelativePanel.AlignRightWith="BlueRect"/>
  <Rectangle Fill="Yellow" MinHeight="100" RelativePanel.Below="GreenRect"
    RelativePanel.AlignLeftWith="BlueRect" RelativePanel.AlignRightWithPanel="True"/>
</RelativePanel>
```
VariableSizedWrapGrid

- Elements are arranged in rows or columns
- Elements are automatically wrap to a new row or column
  - When the MaximumRowsOrColumns value is reached
- Whether elements are arranged in rows or columns is specified by the Orientation property
- Elements can span multiple rows and columns using:
  - VariableSizedWrapGrid.RowSpan and VariableSizedWrapGrid.ColumnSpan
- Elements are sized as specified by the ItemHeight and ItemWidth properties
  - If these properties are not set, the item in the first cell sizes to its content, and all other cells inherit this size
- Child content is visually clipped if larger than the panel
<Page>
  x:Class="VariableSizedWrapGridDemo.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="using:VariableSizedWrapGridDemo"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  mc:Ignorable="d">
    <VariableSizedWrapGrid MaximumRowsOrColumns="3" Orientation="Horizontal"
      Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
      <Rectangle Margin="4" Fill="Red" Height="40" Width="40"/>
      <Rectangle Margin="4" Fill="Orange" Height="40" Width="40"/>
      <Rectangle Margin="4" Fill="Blue" Height="80" Width="80"/>
      <Rectangle Margin="4" Fill="Green" Height="100" Width="140"/>
      <Rectangle Margin="4" Fill="Purple" Height="20" Width="20"/>
      <Rectangle Margin="4" Fill="Black" Height="30" Width="30"/>
    </VariableSizedWrapGrid>
  </Page>
What is a Control?

• **Controls** are elements capable of receiving focus and handling input
  – You add a control to your app UI.
  – You set properties on the control, such as width, height, or foreground color
  – You hook up some code to the control so that it does something

• Many controls are available “out of the box”

• Custom controls can be created
  – User controls that wrap one or more controls and expose higher level properties
  – Custom controls that derive from an existing control and extend its functionality
Example: Static Text

• The TextBlock Element
  – The **Text** property
  – Font related properties
    • **FontSize**, **FontFamily**, etc.
  – **TextAlignment**, **TextTrimming**, **TextDecorations**

• An optional collection of “inlines”
  – Replacement for the **Text** property
  – Inheriting from the abstract **Inline** class
  – **Hyperlink**, **Bold**, **Run**, **Span**, **Underline**, **Italic**, **LineBreak**
Example: TextBlock

```xml
<TextBlock FontSize="16" Margin="4">
  <Run Text="Hello" />
  <Bold>Hello
    <Italic>Hello</Italic>
  </Bold>
  <LineBreak />
  <Hyperlink>Go to my web site</Hyperlink>
  <LineBreak />
  <Underline>This line is underlined</Underline>
  <LineBreak />
  <Span FontSize="20">
    Hello in Bigger Font
  </Span>
</TextBlock>
```

UWPDemo

Hello **Hello Hello**

[Go to my web site]

This line is underlined

Hello in Bigger Font
Other Controls

• Here you can find the full [control list](#)
DATA BINDING
What is **Data Binding**?

- A way for your app's UI to display data, and optionally to stay in sync with that data

- Data binding allows you to separate the concern of data from the concern of UI, for better:
  - Readability, Testability, and Maintainability

- You can choose to use either the `{x:Bind}` or `{Binding}` markup extension

- `{x:Bind}` is new for Windows 10 and it has better performance
  - `{Binding}` has more features

- **Data binding** means tying two arbitrary objects

- Typical scenario is a non-visual object (or collection) to a visual element
  - Any changes to the non-visual object are reflected in the visual element (and optionally vice versa)
Data Binding Concepts

• Source
  – The data object to bind to

• Property Path
  – The property on the source object to use
  – May be a nested property, an array element or an indexer

• Target
  – The target object to modify
  – Must be a dependency property

• Binding Mode
  – Typically one way or two way (target update source)
Using Data Binding

• Typically done in XAML using the \{Binding\} markup extension
  – The Binding class is the workhorse behind the scenes
  – Set on the target property

• \{x:Bind\} – a new binding markup extension
  – Provides compile-time syntax validation
  – Provides better performance
    • Where Binding used reflection at runtime to handle binding, x:Bind is able to
      produce strongly typed code at compile-time to handle the bindings
  – The defaults is Mode=OneTime whereas Binding will default to
    Mode=OneWay
  – It requires defining the type of the variable using x:DataType
Binding Direction

• The **Binding** object allows specifying how the target / source properties are updated

• **Mode** property (of type enum **BindingMode**)

<table>
<thead>
<tr>
<th>Binding Mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>OneWay</td>
<td>The target property is updated by source property changes</td>
</tr>
<tr>
<td>TwoWay</td>
<td>OneWay + the source property is updated by changes of the target property</td>
</tr>
<tr>
<td>OneTime</td>
<td>Target is updated by the source the first time they are bound</td>
</tr>
</tbody>
</table>
Binding to Objects

• **Source**
  – Reference to the source object

• **RelativeSource**
  – Sets the source based on the “location” of the target in the layout tree
  – Useful in data and control templates

• **DataContext**
  – Used by default if **Source** or **RelativeSource** is not specified
  – Searches up the element tree if not found on target element
Change Notifications

• An object must notify when one of its properties changes
  – By defining the property as a dependency property
  – Or by implementing the `INotifyPropertyChanged` interface
• Raise the `PropertyChanged` event
The DataContext

- Sometimes many elements bind to the same object
  - Perhaps with different properties
- The object may be specified as the `DataContext` property on any common parent element
- Whenever the `Source` or `RelativeSource` properties are not specified in the `Binding`, a data context object is searched up the element hierarchy
  - If found, becomes the binding source object
- Can be used programmatically without the need to create the source object in XAML
Bind Example

```xml
<x:Class="DataBindingDemo.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:local="using:DataBindingDemo"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
mc:Ignorable="d"
FontSize="36">
  <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.ColumnDefinitions>
      <ColumnDefinition/>
      <ColumnDefinition/>
      <ColumnDefinition/>
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto"/>
      <RowDefinition/>
    </Grid.RowDefinitions>
    <TextBlock Grid.Row="0" Grid.ColumnSpan="3" HorizontalAlignment="Center" Text="{x:Bind Counter, Mode=OneWay}>0</TextBlock>
    <Button Grid.Row="1" Grid.Column="0" Click="OnUp" HorizontalAlignment="Center" VerticalAlignment="Top">+</Button>
    <Button Grid.Row="1" Grid.Column="1" Click="OnDown" HorizontalAlignment="Center" VerticalAlignment="Top">-</Button>
    <Button Grid.Row="1" Grid.Column="2" Click="OnReset" HorizontalAlignment="Center" VerticalAlignment="Top">R</Button>
  </Grid>
</Page>
```
public sealed partial class MainPage : INotifyPropertyChanged
{
    private int _counter;
    
    public MainPage()
    {
        InitializeComponent();
        DataContext = this;
    }

    public int Counter
    {
        get { return _counter; }
        set
        {
            if (_counter == value)
                return;
            _counter = value;
            OnPropertyChanged();
        }
    }

    private void OnUp(object sender, RoutedEventArgs e) { ++Counter; }
    private void OnDown(object sender, RoutedEventArgs e) { --Counter; }
    private void OnReset(object sender, RoutedEventArgs e) { Counter = 0; }

    public event PropertyChangedEventHandler PropertyChanged;

    private void OnPropertyChanged([CallerMemberName] string propertyName = null)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
}
Data Templates

- A data template is a piece of UI that describes how to display a source object.
- Various elements have properties of type `DataTemplate` just for this:
  - `ContentControl` has a `ContentTemplate` property.
  - `ItemsControl` has a `ItemTemplate` property.
  - `HeaderedContentControl` and `HeaderedItemsControl` have a `HeaderTemplate` property.
- With data binding, the source object is automatically set to the current object that is being rendered.
Value Converters

• A value converter can completely alter the way the source is interpreted into the target
• Often used to match source and target that are of incompatible types
  – E.g. show a red background when the price of a book is greater than 50
• A value converter implements the `IValueConverter` interface
• Create an instance of the converter as a resource and use in a binding expression

```csharp
public interface IValueConverter {
    object Convert(object value, Type targetType, object parameter, CultureInfo culture);
    object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture);
}
```
Converter Parameters

• The “parameter” object is null by default
  – Can be set using the `ConverterParameter` property of the Binding object
  – Not “bindable” in itself!

• The culture info supplied is by default set to “en-US”
  – A type converter exists that can be used via the `ConverterCulture` property of the Binding object (e.g. “fr-CA”, “he-IL”)

• The return value can be `Binding.DoNothing`, which cancels the binding operation
Data Template Selectors

• A way to replace a data template completely
  – Cannot be done in XAML alone
• Derive a class from `DataTemplateSelector`
• Override the `SelectTemplate` method
• Set the `ItemTemplateSelector` property on the `ItemsControl` element to an instance of the custom template selector
Data Template Selector Example

```xml
<Application.Resources>
  <local:AlternateTemplateSelector x:Key="Selector" EvenTemplate="evenTemplate" OddTemplate="oddTemplate" />
  <DataTemplate x:Key="evenTemplate">
    <Border Background="Red">
      <TextBlock Margin="2" Foreground="White" FontSize="20" Text="{Binding}"/>
    </Border>
  </DataTemplate>
  <DataTemplate x:Key="oddTemplate">
    <Border Background="Yellow">
      <TextBlock Margin="2" Foreground="DarkBlue" FontSize="15" Text="{Binding}"/>
    </Border>
  </DataTemplate>
</Application.Resources>

<grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
  <ListBox Name="_list" ItemTemplateSelector="'{StaticResource Selector}'" HorizontalContentAlignment="Stretch"/>
</grid>

public class AlternateTemplateSelector : DataTemplateSelector
{
  public string OddTemplate { get; set; }
  public string EvenTemplate { get; set; }
  protected override DataTemplate SelectTemplateCore(object item, DependencyObject container)
  {
    return (DataTemplate)(Application.Current.Resources[ (int)item % 2 == 0 ? EvenTemplate : OddTemplate]);
  }
}

public MainPage()
{
  this.InitializeComponent();
  var data = new List<int>();
  Random r = new Random();
  for (int i = 1; i < 100; i++)
  {
    data.Add(r.Next(1, 100));
    _list.ItemsSource = data;
  }
}
Styles

• A style (instance of the `Windows.UI.Xaml.Style` class) is a collection of dependency property setters (and triggers)

• Usually defined as a resource

• Can be applied to any element with the `Style` property

• Any specific property changed by the element that conflicts with the style takes precedence over the style setting
<Page x:Class="UWPDemo.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <Page.Resources>
        <Style x:Key="FancyButton" TargetType="Button">
            <Setter Property="Button.Background" Value="Purple" />
            <Setter Property="Button.FontSize" Value="20" />
            <Setter Property="Button.FontWeight" Value="Bold" />
            <Setter Property="Button.Foreground" Value="Yellow" />
            <Setter Property="Button.Margin" Value="4" />
        </Style>
    </Page.Resources>
    <StackPanel Margin="4">
        <Button Content="Click Me" Style="{StaticResource FancyButton}" />
        <Button Content="Hello" Style="{StaticResource FancyButton}" />
        <Button Content="Me Fancy" Style="{StaticResource FancyButton}" />
    </StackPanel>
</Page>
ADAPTIVE UI
Scaling algorithm
$4 \times 4$
Effective pixel
Ignore scale, resolution, & dpi
Design in Effective Pixels

XAML is already in Effective Pixels
Planning your design

Phone
Viewing Distance: 16.3"

Tablets and 2 in 1
Viewing Distance: 20"

Small and Large Laptops
Viewing Distance: 24.5"

Small and Large Desktop Monitors
Viewing Distance: 28"

TV
Viewing Distance: 84"
Snap points

- Phone: epx 320
- Phablet & Tablet: 548, 720
- Desktop: 1024
Design Techniques for Adaptive UI
Use standard responsive/adaptive design techniques

1. Reposition
2. Resize
3. Reflow
4. Reveal
5. Replace
6. Re-architect
Adaptive design

• Build a page that adapts to different screen sizes and orientations
  – Use Visual States and Adaptive Triggers to change layout
  – Use RelativePanel to position blocks of content relative to peers, re-positioning in different visual states
Tailored design

• Build unique experiences on different devices
Adaptive UI – The **VisualStateManager**

- Manages visual states and the logic for transitions between visual states for controls
- Each VisualState has a name that is representative of a UI state that can be changed by the user, or changed by control logic
- One way to handle the current VSM state is to handle the **SizeChanged** event and to command the VSM to **GoToState**
- However, VSM states can be triggered using the **AdaptiveTrigger** element
  - Represents a declarative rule that applies visual states based on window properties
<Page x:Class="AdaptiveUI.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:AdaptiveUI"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    mc:Ignorable="d">
    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <VisualStateManager.VisualStateGroups>
            <VisualStateGroup x:Name="VisualStateGroup">
                <VisualState x:Name="VisualStateMin320">
                    <VisualState.StateTriggers>
                        <AdaptiveTrigger MinWindowWidth="320"/>
                    </VisualState.StateTriggers>
                </VisualState>
            </VisualStateGroup>
        </VisualStateManager.VisualStateGroups>
    </Grid>
</Page>
VSM Sample

<VisualState.Setters>
  <Setter Target="Hero.(Grid.ColumnSpan)" Value="2" />
  <Setter Target="Hero.(Grid.RowSpan)" Value="1" />
  <Setter Target="Metadata.(Grid.Column)" Value="0" />
  <Setter Target="Metadata.(Grid.Row)" Value="1" />
  <Setter Target="Metadata.(Grid.ColumnSpan)" Value="2" />
  <Setter Target="Metadata.(Grid.RowSpan)" Value="1" />
  <Setter Target="Description.(RelativePanel.RightOf)" Value="" />
  <Setter Target="Description.(RelativePanel.Below)" Value="Username" />
  <Setter Target="Description.Margin " Value="0,12,0,0" />
  <Setter Target="ImageName.FontSize" Value="20" />
</VisualState.Setters>
</VisualState>
VSM Sample

<VisualState x:Name="VisualStateMin548">
  <VisualState.StateTriggers>
    <AdaptiveTrigger MinWindowWidth="548"/>
  </VisualState.StateTriggers>
  <VisualState.Setters>
    <Setter Target="Hero.(Grid.ColumnSpan)" Value="2"/>
    <Setter Target="Hero.(Grid.RowSpan)" Value="1"/>
    <Setter Target="Metadata.(Grid.Column)" Value="0"/>
    <Setter Target="Metadata.(Grid.Row)" Value="1"/>
    <Setter Target="Metadata.(Grid.ColumnSpan)" Value="2"/>
    <Setter Target="Metadata.(Grid.RowSpan)" Value="1"/>
    <Setter Target="Description.(RelativePanel.RightOf)" Value="Avatar"/>
    <Setter Target="Description.(RelativePanel.Below)" Value=""/>
    <Setter Target="Description.Margin" Value="12,0,0,0"/>
    <Setter Target="ImageName.FontSize" Value="20"/>
  </VisualState.Setters>
</VisualState>
<VisualState x:Name="VisualStateMin1024">
  <VisualState.StateTriggers>
    <AdaptiveTrigger MinWindowWidth="1024"/>
  </VisualState.StateTriggers>
  <VisualState.Setters>
    <Setter Target="Hero.(Grid.ColumnSpan)" Value="1"/>
    <Setter Target="Hero.(Grid.RowSpan)" Value="2"/>
    <Setter Target="Metadata.(Grid.Column)" Value="1"/>
    <Setter Target="Metadata.(Grid.Row)" Value="0"/>
    <Setter Target="Metadata.(Grid.ColumnSpan)" Value="1"/>
    <Setter Target="Metadata.(Grid.RowSpan)" Value="2"/>
    <Setter Target="LeftCol.Width" Value="2*"/>
    <Setter Target="RightCol.Width" Value="1*"/>
    <Setter Target="Description.(RelativePanel.RightOf)" Value="Avatar"/>
    <Setter Target="Description.(RelativePanel.Below)" Value=""/>
    < Setter Target="Description.Margin" Value="12,0,0,0"/>
    <Setter Target="ImageName.FontSize" Value="24"/>
  </VisualState.Setters>
</VisualState>
VSM Sample

```xml
<VisualStateGroup>
    <VisualStateManager.VisualStateGroups>
    <Grid.RowDefinitions>
        <RowDefinition />
        <RowDefinition />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition x:Name="LeftCol" />
        <ColumnDefinition x:Name="RightCol" />
    </Grid.ColumnDefinitions>
    <Image x:Name="Hero" Grid.Column="0" Source="Assets/airtime.jpg" Stretch="UniformToFill"
        HorizontalAlignment="Center" VerticalAlignment="Center" />
    <ScrollViewer x:Name="Metadata" VerticalScrollBarVisibility="Auto">
        <RelativePanel Grid.Column="1" Background="LightBlue" Padding="12">
            <Image x:Name="Avatar" Source="Assets/avatar.jpg" Width="100" Height="100" HorizontalAlignment="Left" />
            <TextBlock x:Name="Username" RelativePanel.Below="Avatar"
                RelativePanel.AlignHorizontalCenterWith="Avatar" Text="phutureproof" />
        </RelativePanel>
    </ScrollViewer>
</VisualStateManager.VisualStateGroups>
</VisualStateGroup>
```
VSM Sample

<StackPanel x:Name="Description">
  <TextBlock x:Name="ImageName" Foreground="White" FontWeight="Light" Text="Airtime" />
  <TextBlock Text="9/15/15" />
</StackPanel>
</RelativePanel>
</ScrollViewer>
</Grid>
</Page>
The Application Object

• Every UWP application is represented by an instance of `Windows.UI.Xaml.Application` – A singleton

• The static property `Application.Current` returns that instance

• The Visual Studio wizard creates a XAML file for the application
  – Useful for application level resources
  – Also can set the `StartupUri` property to indicate which window to show on startup
COMMANDS AND MVVM
Agenda

• Introduction to the MVVM Pattern
• Commands
• UWP Command Support
• Implementing Commands for MVVM
• Simple MVVM Framework
• Summary
The MVVM Pattern

- Model – View – ViewModel
- Based on similar principles of Model View Controller (MVC) and Model View Presenter (MVP)
- Natural pattern for XAML based applications
  - Data binding is key
- Enables developer-designer workflow
- Increases application testability
MVVM Participants

• Model
  – Business logic and data
  – May implement change notification for properties and collections

• View
  – Data display and user interactivity
  – Implemented as a Page, UserControl, DataTemplate or custom control
  – Has little or no code behind

• ViewModel
  – UI logic and data for the View
  – Abstracts the Model for View usage
  – Exposes commands (ICommand) to be used by the View
  – Implements change notifications
  – Maintains state for the View (communicates via data binding)
The View

- Provides the user interface and interaction
- The DataContext property points to the View Model
- Updated using property changes from the ViewModel
- Binds to commands (on ICommandSource elements) provided by the ViewModel
The ViewModel

- Exposes properties the View binds to
- Can be an adapter if some functionality missing from Model classes
- Exposes commands to be invoked by the view
- Maintains state for the View
- Implements change notifications (INotifyPropertyChanged)
  - Uses ObservableCollection<T> that already implements INotifyCollectionChanged
The Model

• Responsible for business logic and data, e.g.
  – Data Transfer Objects (DTO)
  – POCOs (Plain Old CLR Objects)
  – Generated entity objects
  – Generated proxy objects

• May provide change notifications
• Provides validation if appropriate
Introduction to Commands

• Handling routed events and executing some code is fine for simple applications
• Sometimes the same code needs to execute from unrelated events (e.g. mouse click, keyboard shortcut, menu item, toolbar)
• Maintaining UI state (e.g. enabled/disabled) becomes difficult
• Higher level functionality, such as an undo / redo system is not possible
• Solution: use commands (the “Command” design pattern) with some support from UWP
The Command

- A command is an object implementing the `System.Windows.Input.ICommand` interface.

```csharp
public interface ICommand {
    event EventHandler CanExecuteChanged;

    bool CanExecute(object parameter);
    void Execute(object parameter);
}
```
Commands for MVVM

• MVVM frameworks typically provide a basic ICommand implementation that uses a delegate
  – Class typically called DelegateCommand or RelayCommand
• Other implementations possible
  – E.g. the CompositeCommand class from PRISM that holds a list of commands
• Using commands in MVVM
  – Some controls expose a Command and CommandParameter properties that can be bound to a command exposed by the ViewModel
  – ButtonBase, Hyperlink and MenuItem expose these
    • Technically part of the ICommandSource interface
Wiring the View and the View Model

• The View’s DataContext must be set to its supporting ViewModel

• Some options
  – The View can create an instance of the right VM (even in XAML)
  – The ViewModel can be injected using some dependency injection technique (e.g. Unity or MEF)
  – Use some global ViewModel locator object
  – A Main VM can be set explicitly on the main View, and other VMs can be exposed as properties, which will be bound by child views
Summary

• Commands allow high level segregation of tasks
• The MVVM pattern is common in WPF to separate logic from UI and increase testability
Asynchronous Operations

• UWP, like other UI technologies, is mostly single threaded
• Synchronous operations are easy to use, but long operations may freeze the UI (“Not responding”)
  – Unacceptable user experience
• Asynchronous operations on the UI thread can be scheduled using the Dispatcher object responsible for that UI thread
  – Accessible using `Dispatcher.CurrentDispatcher` static property if on the UI thread
  – Or by calling `DispatcherObject.Dispatcher` instance property (which is inherited by all UWP elements)
Using the Dispatcher

• An operation can be scheduled using the **Invoke** or **BeginInvoke** methods
  – Accept at least a delegate and an optional **DispatcherPriority**
  – .NET 4.5 adds **InvokeAsync** variants

• **DispatcherPriority** enumeration
  – Indicates the priority to process the operation
  – Lower priorities can assume higher priority operations have already completed
  – Default is **DispatcherPriority.Normal**
Updating the UI

• When doing work on a background thread (either explicitly created or using the thread pool or task), some result may need to update UWP elements
  – Accessing the object directly will cause an exception

• Need to marshal the required operation to the UI thread using the Dispatcher
  – Call `Invoke` (for synchronous invocation) or `BeginInvoke/InvokeAsync` (for asynchronous invocation)
  – Specify a priority for the update operation (usually `DispatcherPriority.Normal`)
Async Patterns in .NET

• Asynchronous Programming Model (APM)
  – Existed since .NET 1.0
• Event Asynchronous Pattern (EAP)
  – Introduced in .NET 2.0
• Task Asynchronous Pattern (TAP)
  – New to .NET 4.5 and C# 5.0
Asynchrony with C# 5.0

• Synchronous

```csharp
private void OnGetData(object sender, RoutedEventArgs e) {
    _cmdGet.IsEnabled = false;
    var wc = new WebClient();
    _text.Text = wc.DownloadString(new Uri("http://msdn.microsoft.com"));
    _cmdGet.IsEnabled = true;
}
```

• Asynchronous with C# 5.0

```csharp
private async void OnGetData(object sender, RoutedEventArgs e) {
    _cmdGet.IsEnabled = false;
    var wc = new WebClient();
    _cmdGet.IsEnabled = true;
}
```
Asynchronous methods

• Marked with the `async` modifier
• Must return `void`, `Task` or `Task<T>`
  – Avoid returning `void`
    • Does not allow callers to be notified of completion
• Use `await` to cooperatively yield control
• Are resumed when awaited operation completes
• Can await anything that implements the “awaiter pattern”
• Execute in the synchronization context of their caller
  – Depends on the implementing waiter
• Allow composition using regular programming constructs
HANDLING PAGE NAVIGATION
The Navigation Model

- The UWP app navigation model consists of:
  - Launching, activating, resuming, suspending, and terminating apps
  - Moving between apps
  - Moving between pages in an app
  - Moving between views within an app

- The UWP navigation model uses **Frame** and **Page** objects
  - The frame hosts the pages and keeps the navigation history
  - You can also pass data between pages as you navigate
Shell-drawn back button for Mobile and Tablet
if (Frame.CanGoBack)
{
    // Setting this visible is ignored on Mobile and when in tablet mode!
}
Desktop, Windowed mode:
Or provide your own on-canvas Back Button
Navigation in Depth

• Refer to this location for more navigation details
<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Accessibility</strong></td>
<td>Create Universal Windows apps that are accessible to the widest possible range of device types and user audiences, including people who have impairments or disabilities.</td>
</tr>
<tr>
<td><strong>App data and settings</strong></td>
<td>Find out how and where a Universal Windows app stores both app data and user data. Learn from scenarios that describe the different storage options and the situations where each provides the best experience.</td>
</tr>
<tr>
<td><strong>App-to-app communication</strong></td>
<td>Learn how Universal Windows apps (including Windows web apps) can launch other apps and exchange data and files. Complex tasks that would normally require a user to manage multiple apps can now be handled seamlessly.</td>
</tr>
<tr>
<td><strong>Audio, video, and camera</strong></td>
<td>Capture photos and videos from a capture device, such as a webcam, and render audio streams in an app.</td>
</tr>
<tr>
<td><strong>Contacts and calendar</strong></td>
<td>Let your users access their Windows contacts and calendar appointments from your app, so they can share content, email, and calendar info, or send messages without having to switch between Windows apps.</td>
</tr>
<tr>
<td><strong>Controls, layouts, and text</strong></td>
<td>Learn about user interface components, like text, buttons, lists, and windows that are available and the various options and layouts available to support a fully-interactive user experience.</td>
</tr>
<tr>
<td><strong>Custom user interactions</strong></td>
<td>Find out about the user interaction platform, the input sources (including touch, touchpad, mouse, pen/stylus, and keyboard), modes (touch keyboard, mouse wheel, pen eraser, and so on), and user interactions supported by Universal Windows apps.</td>
</tr>
<tr>
<td><strong>Data binding</strong></td>
<td>Synchronize the UI elements of your Universal Windows app with different sources of data, including databases, files, and internal objects, to provide a data-driven user experience.</td>
</tr>
<tr>
<td><strong>Debugging, testing, and performance</strong></td>
<td>Learn about the testing and debugging cycle and how to use the related tools provided with Microsoft Visual Studio or as separate downloads. Make sure your Universal Windows app delivers the experience you intend and is ready for publication to the Windows Store.</td>
</tr>
<tr>
<td>Topic</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Devices, sensors, and power</strong></td>
<td>Integrate different devices like printers, cameras, and sensors into your Universal Windows app to provide a robust and flexible connected-device experience for your users.</td>
</tr>
<tr>
<td><strong>Files, folders, and libraries</strong></td>
<td>Learn how to read and write text and other data formats in files, and manage files and folders. Also find info about reading and writing app settings, about file and folder pickers, and about special, &quot;sandboxed&quot; locations such as the Video/Music library.</td>
</tr>
<tr>
<td><strong>Games</strong></td>
<td>Understand the basics of creating games on the new Universal Windows Platform (UWP), and how to provide visuals that match the ambition of your game with Microsoft DirectX 12.</td>
</tr>
<tr>
<td><strong>Globalization and localization</strong></td>
<td>Adapt your Universal Windows app for additional languages, markets, cultures, and regions. This info guides you through a set of best practices for development, and points you to more details about how to prepare your app for international markets.</td>
</tr>
<tr>
<td><strong>Graphics and animation</strong></td>
<td>Enhance your Universal Windows app with UI graphics and animations that keep users visually engaged and interested in the user experience.</td>
</tr>
<tr>
<td><strong>Launching, resuming, and background tasks</strong></td>
<td>Create background tasks and register for system-generated events to provide functionality even when your Universal Windows app is suspended or not running.</td>
</tr>
<tr>
<td><strong>Maps and location</strong></td>
<td>Learn how your Universal Windows app can tap into the Bing Maps service and produce accurate map visuals that now include aerial 3D imagery and street-level views.</td>
</tr>
<tr>
<td><strong>Monetize your app</strong></td>
<td>Create free apps, trials (both time-based and feature-based), paid apps, and in-app products, to give your customers the option to try your app for free and make purchase decisions during their experience with your app.</td>
</tr>
<tr>
<td><strong>Navigation</strong></td>
<td>Learn about the various options you have to support navigation between pages and content in your app.</td>
</tr>
<tr>
<td>Topic</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Navigation</td>
<td>Learn about the various options you have to support navigation between pages and content in your app.</td>
</tr>
<tr>
<td>Networking and web services</td>
<td>Create a connected, or network-aware, Universal Windows app that can use available network connections to do things like fetch RSS feeds, engage in multiplayer games, or interact with nearby devices.</td>
</tr>
<tr>
<td>Packaging apps</td>
<td>Understand the app package that contains the files that constitute your Universal Windows app, and how you work with it to deploy, manage, and update your app through the Windows Store. Also learn about app capabilities, which must be declared in the app package manifest for access to specific resources.</td>
</tr>
<tr>
<td>Porting apps to Windows 10</td>
<td>Bring an existing app to the UWP where you can create a single app package that not only targets the Windows-based devices of your choosing, but also capitalizes on features and user experiences unique to each device type.</td>
</tr>
<tr>
<td>Security</td>
<td>Manage sensitive user info and help secure app data and resources while keeping the user experience intact. Features like basic password protection, roaming credentials, single sign-on, Microsoft account authentication, and cryptography are all at your disposal.</td>
</tr>
<tr>
<td>Threading and async programming</td>
<td>Use asynchronous programming to help your app stay responsive by allowing it to continue to run and respond to the UI while it completes other work that might take an extended amount of time.</td>
</tr>
<tr>
<td>Tiles, badges, and notifications</td>
<td>Design tiles (including secondary tiles and lock-screen apps), badges, and toast notifications that your Universal Windows app can use to communicate new, real-time info to users in the form of text, images, or both.</td>
</tr>
<tr>
<td>Windows Runtime components</td>
<td>Learn more about these self-contained objects that you can initialize and use from any language, including C#, Visual Basic, JavaScript, and C++. For example, you could create a Windows Runtime component in C++ that uses a third-party library to perform a computationally expensive operation, or simply reuse some Visual Basic or C# code in your Universal Windows app.</td>
</tr>
<tr>
<td>XAML platform</td>
<td>Get started with the basic concepts of the XAML programming language. Or, if you're already familiar with XAML, jump ahead and learn how to implement Windows Runtime features in XAML using Visual Studio to create a great Universal Windows app.</td>
</tr>
</tbody>
</table>
Appendix B

INTRODUCTION TO COM
Agenda

• What is the Component Object Model?
• Interfaces and IUnknown
• Other COM Entities
• The Active Template Library (ATL)
• Using ATL for Client Access
• Using ATL to Implement COM Classes
• Object Reuse
• Threading Models and Apartments
• Summary
Component Object Model

• Binary standard interface specification for objects
  – Based closely on C++ vtable mechanism for dispatch
  – Ideally language independent
    • Can use with C, C++, VB, .NET, etc.

• A set of rules and a supporting runtime for building component based systems

• Location transparency
  – Fast access to in-process objects
  – Seamless inter-process communication to out-of-process objects

• Interface based programming
What is a COM Object?

• A COM object
  – Separates its interface(s) from its implementation
  – Provides multiple services through separate interfaces

• All COM objects must implement the **IUnknown** interface
  – Includes a *QueryInterface()* function that clients can use to acquire pointers to other interfaces
  – Also includes *AddRef()* and *Release()* functions for reference counting
The IUnknown Interface

• Base interface
  – Must be implemented by all COM objects
  – All interfaces must extend IUnknown

```cpp
class IUnknown {
public:
    virtual HRESULT __stdcall QueryInterface(const IID& iid, void** ppvObject) = 0;
    virtual ULONG __stdcall AddRef() = 0;
    virtual ULONG __stdcall Release() = 0;
};
```
GUIDs

• Every entity in COM is identified by a GUID
  – 128 bit Global Unique Identifiers, statistically unique across space and time

• Used to identify interfaces, classes, type libraries and more

• Can generate with the Guidgen.exe tool
  – Calls CoCreateGuid internally
  – Several formats available
**HRESULTs**

- Interface functions should not throw Win32 or C++ exceptions
  - Not meaningful to all types of clients
  - Cannot cross process boundaries
- Interface methods should return an HRESULT
  - 32-bit value that contains success or error code
  - AddRef() and Release() are only exceptions to this rule
### Common HRESULTs

<table>
<thead>
<tr>
<th>HRESULT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>Standard success code ((=0))</td>
</tr>
<tr>
<td>S_FALSE</td>
<td>Partial success in some sense ((=1))</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>Generic failure code</td>
</tr>
<tr>
<td>E_POINTER</td>
<td>Bad pointer supplied</td>
</tr>
<tr>
<td>E_UNEXPECTED</td>
<td>Unexpected call at this time</td>
</tr>
<tr>
<td>E_NOINTERFACE</td>
<td>The requested interface is not supported</td>
</tr>
<tr>
<td>E_NOTIMPL</td>
<td>Functionality not implemented (yet)</td>
</tr>
<tr>
<td>E_OUTOFMEMORY</td>
<td>Not enough memory to complete the operation</td>
</tr>
<tr>
<td>E_INVALIDARG</td>
<td>Argument is invalid</td>
</tr>
</tbody>
</table>
# HRESULT Macros

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCEEDED</td>
<td>Returns true if the severity bit is off</td>
</tr>
<tr>
<td>FAILED</td>
<td>Returns true if the severity bit is on</td>
</tr>
<tr>
<td>HRESULT_CODE</td>
<td>Returns the information field (e.g., error code) of the HRESULT</td>
</tr>
<tr>
<td>HRESULT_FACILITY</td>
<td>Returns the facility field of the HRESULT</td>
</tr>
<tr>
<td>HRESULT_SEVERITY</td>
<td>Returns the severity field of the HRESULT</td>
</tr>
<tr>
<td>MAKE_HRESULT</td>
<td>Constructs an HRESULT from individual fields</td>
</tr>
<tr>
<td>HRESULT_FROM_WIN32</td>
<td>Constructs an HRESULT from a Win32 error code</td>
</tr>
</tbody>
</table>
QueryInterface Rules

- **QueryInterface** must be

<table>
<thead>
<tr>
<th>Symmetric</th>
<th>Transitive</th>
<th>Reflexive</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Symmetric" /></td>
<td><img src="image" alt="Transitive" /></td>
<td><img src="image" alt="Reflexive" /></td>
</tr>
</tbody>
</table>

- When **IUnknown** is requested, **QueryInterface** must return the same pointer value
  - Serves as the identity of the COM object
Reference Counting

• A COM object determines its own lifetime
  – Maintains a reference count
  – Incremented by a call to `AddRef()` or a successful call to `QueryInterface()`
  – Decremented by a call to `Release()`
  – When all reference counts drop to zero, deletes itself

• Clients of COM objects must follow the rules
  – Call `AddRef()` after duplicating an interface pointer
  – Call `Release()` when an interface is no longer required
  – Failure to follow these rules may lead to memory leaks or crashes
COM Servers

- One or more COM classes can be implemented as
  - A DLL (in-proc) server, or
  - An EXE (local) server
- Each class of COM object has an associated class object (class factory)
  - Usually implements the standard interface `IClassFactory`
  - Creates instance of the associated COM object
- To implement a DLL server
  - Implement the class object
  - Implement the exported DLL global functions (see later)
  - Generate a GUID to identify the COM object's class
  - Build the DLL
  - Register the COM object with the Registry (see later)
Activation APIs

- Client wants to create an instance of a COM class
  - Calls `CoGetClassObject` to get a class object (class factory)
    - Typically requesting the `IClassFactory` interface
  - Calls `IClassFactory::CreateInstance` to get an actual instance

```c
HRESULT CoGetClassObject(
    __in REFCLSID rclsid, // GUID of COM class
    __in DWORD dwClsContext, // CLSCTX_ALL
    __in_opt COSERVERINFO *pServerInfo, // NULL for local machine
    __in REFIID riid, // class factory interface ID
    __out LPVOID *ppv // result
);
```
CoGetClassObject Example

CoInitialize(0);

IClassFactory* pCF;
HRESULT hr = CoGetClassObject(CLSID_Account, CLSCTX_ALL, NULL, IID_IClassFactory, (void**)&pCF);
if(SUCCEEDED(hr)) {
    // acquired class factory, create object
    IAccount* pAcc;
    hr = pCF->CreateInstance(NULL, IID_IAccount, (void**)&pAcc);
    pCF->Release();

    if(SUCCEEDED(hr)) {
        // do something with pAcc
        pAcc->Deposit(100);
        // interface no longer needed
        pAcc->Release();
    }
}
Activation APIs

- The previous code works, but a bit tedious
- Most class factories implement IClassFactory
- Call `CoCreateInstance` as a convenience that always requests IClassFactory in its internal call to `CoGetClassObject`
  - Always uses the local machine

```c
HRESULT CoCreateInstance(
    __in  REFCLSID rclsid,   // requested class ID
    __in  LPUNKNOWN pUnkOuter,  // for aggregation (see later)
    __in  DWORD dwClsContext,
    __in  REFIID riid,         // requested interface
    __out LPVOID *ppv          // result
);  
```
CoInitialize();

IAccount* pAcc;
HRESULT hr = CoCreateInstance(CLSID_Account, NULL, CLSCTX_ALL, 
    IID_IAccount, (void**)&pAcc);
if(SUCCEEDED(hr)) {
    // use account
    pAcc->Deposit(100);

    // interface no longer needed
    pAcc->Release();
}
How a COM Object is Created

Client

Ole32.Dll

CoCreateInstance

search CLSID in registry

COM Server DLL

Load DLL

CoGetClassObject

Class Factory

<< Creates >>

IClassFactory pointer

COM Instance

<< Creates >>

interface pointer

IClassFactory::CreateInstance

Interface pointer

IXxx::SomeMethod

IXxx::Release

Client Ole32.Dll
CoCreateInstance
search CLSID in registry
COM Server DLL
Load DLL
CoGetClassObject
Class Factory
<< Creates >>
COM Instance
interface pointer
IXxx::SomeMethod
IXxx::Release
Registering the COM Class

- COM classes must be registered
  - Minimum is the CLSID to DLL/EXE mapping
  - Can provide a “user-friendly” ProgID
Implementing a COM Class in C++

• Create a Win32 DLL Project
• Define all relevant GUIDs
• Define required interfaces
• Implement the actual class
• Implement a Class Factory
• Implement the following global functions
  – DllGetClassObject (must)
  – DllRegisterServer (highly recommended)
  – DllUnregisterServer (highly recommended)
  – DllCanUnloadNow (optional)
Create a Win32 DLL Project

• Open Visual Studio, Select File->New Project...
  – C++ / Win32 -> Win32 Project
Create a Win32 DLL Project

• Select an Application Type of DLL
Define Relevant GUIDs

- Use the GuidGen.Exe tool

```cpp
// GUIDs.h

#pragma once

#include <InitGuid.h>

// {D28CD8CD-18DE-456b-B544-4D771F96B6F9}
DEFINE_GUID(CLSID_SimpleAccount,
            0xd28cd8cd, 0x18de, 0x456b, 0xb5, 0x44, 0x4d, 0x77, 0x1f,
            0x96, 0xb6, 0xf9);

// {B66AB2DF-D047-46ba-9AAD-F7BE32B1140F}
DEFINE_GUID(IID_IAccount,
            0xb66ab2df, 0xd047, 0x46ba, 0x9a, 0xad, 0xf7, 0xbe, 0x32,
            0xb1, 0x14, 0xf);
```
Define Interfaces

This is typically defined with the Interface Definition Language (IDL):

```c
#include <Unknwn.h>

struct IAccount : IUnknown {
    STDMETHOD(Deposit)(double amount) = 0;
    STDMETHOD(Withdraw)(double amount) = 0;
    STDMETHOD(GetBalance)(double* balance) = 0;
};
```
Implement the Actual COM Class

// SimpleAccount.h

#include "GUIDs.h"
#include "Interfaces.h"

class CSimpleAccount : public IAccount {
public:
    CSimpleAccount() : m_Balance(0), m_RefCount(1) {
    }

    // IUnknown
    STDMETHOD(QueryInterface)(REFIID riid, void** ppv);
    STDMETHOD_(ULONG, AddRef)();
    STDMETHOD_(ULONG, Release)();

    // IAccount
    STDMETHOD(Deposit)(double amount);
    STDMETHOD(Withdraw)(double amount);
    STDMETHOD(GetBalance)(double* balance);

private:
    double m_Balance;
    unsigned mRefCount;
};
// SimpleAccount.cpp

#include "SimpleAccount.h"
#include "GUIDs.h"

// IUnknown

STDMETHODIMP CSimpleAccount::QueryInterface(REFIID riid, void** ppv) {
    if(riid == IID_IUnknown || riid == IID_IAccount) {
        AddRef();
        return *ppv = static_cast<IAccount*>(this), S_OK;
    }
    return E_NOINTERFACE;
}

STDMETHODIMP_(ULONG) CSimpleAccount::AddRef() {
    return ++m_RefCount;
}

STDMETHODIMP_(ULONG) CSimpleAccount::Release() {
    if(--m_RefCount == 0) {
        delete this;
        return 0;
    }
    return m_RefCount;
}
// IAccount

STDMETHODIMP CSimpleAccount::Deposit(double amount) {
    if (amount <= 0) return E_INVALIDARG;
    m_Balance += amount;
    return S_OK;
}

STDMETHODIMP CSimpleAccount::Withdraw(double amount) {
    if (amount <= 0) return E_INVALIDARG;
    if (m_Balance < amount) return E_UNEXPECTED;

    m_Balance += amount;
    return S_OK;
}

STDMETHODIMP CSimpleAccount::GetBalance(double* balance) {
    if (balance == NULL) return E_POINTER;
    *balance = m_Balance;
    return S_OK;
}
Implement the Class Factory

class CSimpleAccountCF : public IClassFactory {
public:
    // IClassFactory methods
    STDMETHODIMP.CreateInstance(IUnknown *pUnkOuter, REFIID riid, void **ppvObject);
    STDMETHODIMP(LockServer)(BOOL fLock);

    // IUnknown methods
    STDMETHODIMP(QueryInterface)(REFIID riid, void** ppv);
    STDMETHOD(UULONG, AddRef)() { return 2; }    // never dies
    STDMETHOD(UULONG, Release)() { return 1; }    // never dies
};

// implement IClassFactory

STDMETHODIMPIMP CSimpleAccountCF::CreateInstance(IUnknown* pOuter, REFIID riid, void** ppv) {
    CSimpleAccount* pAcc = new CSimpleAccount;
    HRESULT hr = pAcc->QueryInterface(riid, ppv);
    pAcc->Release();
    return hr;
}

STDMETHODIMPIMP CSimpleAccountCF::LockServer(BOOL fLock) {
    return S_OK;    // don't keep track
}
HINSTANCE g_hInstDll;

BOOL WINAPI DllMain(HINSTANCE hDll, DWORD reason, LPVOID) {
    switch(reason) {
    case DLL_PROCESS_ATTACH:
        g_hInstDll = hDll;
        break;
    }
    return TRUE;
}

// maintain a singleton class factory
CSimpleAccountCF g_SimpleAccountCF;

STDAPI DllGetClassObject(REFCLSID rclsid, REFIID riid, void** ppv) {
    if(rclsid == CLSID_SimpleAccount)
        return g_SimpleAccountCF.QueryInterface(riid, ppv);
    return CLASS_E_CLASSNOTAVAILABLE;
}
Implementing Global Functions

STDAPI DllRegisterServer() {
    HKEY hKey = NULL;
    DWORD code = RegCreateKeyExW(HKEY_CLASSES_ROOT,
        L"CLSID\{D28CD8CD-18DE-456b-B544-4D771F96B6F9}\InProcServer32",
        0, 0, 0, KEY_ALL_ACCESS, 0, &hKey, NULL);
    if(hKey == NULL) return HRESULT_FROM_WIN32(code);
    WCHAR path[MAX_PATH] = {0};
    GetModuleFileNameW(g_hInstDll, path, MAX_PATH);
    RegSetValueExW(hKey, L"", 0, REG_SZ, (const BYTE*)path,
        lstrlenW(path) * sizeof(WCHAR));
    RegCloseKey(hKey);
    return S_OK;
}

STDAPI DllUnregisterServer() {
    RegDeleteKeyW(HKEY_CLASSES_ROOT,
        L"CLSID\{D28CD8CD-18DE-456b-B544-4D771F96B6F9}\InProcServer32");
    return S_OK;
}

- Use regsvr32.exe with the DLL path to register
  - Calls DllRegisterServer
  - Use /u to unregister (calls DllUnregisterServer)
A Simple Client

```cpp
#include "..\SimpleAccountSvr\GUIDs.h"
#include "..\SimpleAccountSvr\Interfaces.h"

void _tmain(int argc, _TCHAR* argv[]) {
    CoInitialize(0);

    IAccount* pAcc;
    HRESULT hr = CoCreateInstance(CLSID_SimpleAccount, NULL, CLSCTX_ALL,
                                   IID_IAccount, (void**)&pAcc);
    if(SUCCEEDED(hr)) {
        pAcc->Deposit(100);
        pAcc->Withdraw(60);
        double balance;
        pAcc->GetBalance(&balance);
        cout << "Current balance: " << balance << endl;

        // finished working with object
        pAcc->Release();
    }

    CoUninitialize();
}
```
Implementation Notes

• The “interesting” implementation is of the CSimpleAccount class
• Most other code is boilerplate and would probably be repeated in other projects
  – Class factory implementation
  – IUnknown implementation
  – Registration / unregistration code
  – DllGetClassObject implementation
• Enter the Active Template Library (ATL)
What is ATL?

• A collection of C++ template classes, macros and functions that make it easier to develop COM components
  – Highly optimized with no run time library
  – Takes care of “boilerplate” code such as class factories, IUnknown implementation and registration

• Provides client side support as well with smart pointers and other COM entity wrappers
  – BSTR, VARIANT, SAFEARRAY
Building a COM Component with ATL

• Create an ATL project
• Add a COM class
  – Select interface type and other attributes
• Implement methods (can use a helping wizard)
• Build and you’re done
  – No need to explicitly implement IUnknown
  – No need to implement a class factory
  – No need to implement registration code
Create an ATL Project
Create an ATL Project

• Select a server type
  – Typically a DLL
New ATL Project Contents

- Two projects are actually created
- One is the main project where our code resides
- The other (ending with PS) is a proxy/stub DLL
  - Code automatically generated based on interface definitions in the main project
  - Only needed if cross apartment / process communication may occur
  - Can do without it when working with the Type Library Marshalar (not covered in this course)
  - Typically unnecessary when working with Media Foundation as we’ll implement MF interfaces and not define our own
- Can usually delete the PS project
Main Files in an ATL Project

• *ProjectName*.IDL
  – Interface, methods and COM class definitions (more on that later)
• *ProjectName*.cpp
  – Global functions implementation
• dllmain.cpp
  – DllMain implementation
• *ProjectName*.def
  – Module definition file with exports
• *ProjectName*.rgs
  – Project registry script (more on this later)
Adding a COM Class

- Right click project, select “Add Class”, select “ATL Simple Object”, then click Add
Configuring a COM Class

• Select a Simple name as a base name
• Select first interface to implement (IUnknown is present automatically)

• C++ section
  – Affects the generated source code

• COM section
  – Reflected in the IDL and type library
Configuring a COM Class

• Custom interface inherits from IUnknown

• Dual interface inherits from IDispatch
  – Provides automation capabilities
Interface Definition Language (IDL)

• Used to define COM interfaces, methods, properties and other entities
• Has no implementation, just declarations
• Allows placements of attributes
• Compiled by the Microsoft IDL (MIDL) compiler
  – Generates appropriate header files
  – Generates a type library
• Used to generate a proxy/stub DLL (if needed)
A Typical IDL File

import "oaidl.idl";
import "ocidl.idl";

[ object,
  uuid(C084AF0-4875-4243-AFB4-1C194EC03A85),
  helpstring("IAccount Interface"),
  pointer_default(unique) ]

interface IAccount : IUnknown {
  [helpstring("method Deposit")] HRESULT Deposit([in] DOUBLE amount);
  [helpstring("method Withdraw")] HRESULT Withdraw([in] DOUBLE amount);
  [helpstring("method GetBalance")] HRESULT GetBalance([out,retval] DOUBLE* balance);
};

[ uuid(52BEAE9B-BD95-42A5-BBC5-57D220F606AC),
  version(1.0),
  helpstring("AccountSvr 1.0 Type Library") ]

library AccountSvrLib {
  importlib("stdole2.tlb");
  [ uuid(E29B6DC7-6A8C-4F25-9589-4AA563C27380),
    helpstring("SimpleAccount Class") ]
  coclass SimpleAccount {
    [default] interface IAccount;
  };
Compiling an IDL File

• IDL files are compiled by the MIDL compiler
  – A C/C++ header file needed for server compilation (and optionally for the client too)
  – A type library file (TLB extension)
    • Provides a “universal header file” usable by any language (C++, VB, .NET languages, others)
    • Added to the final DLL/EXE as a resource in an ATL project
  – A GUIDs file
  – More files for creating a proxy/stub DLL
Adding Members to an Interface

• Right click the interface in “Class View” and select “Add Method” or “Add Property”

• Adds code in 3 locations
  – IDL file
  – H file
  – CPP file
A Typical Header File

```cpp
class ATL_NO_VTABLE CSimpleAccount :
    public CComObjectRootEx<CComSingleThreadModel>,
    public CComCoClass<CSimpleAccount, &CLSID_SimpleAccount>,
    public IAccount{
public:
    CSimpleAccount() {
    }

DECLARE_REGISTRY_RESOURCEID(IDR_SIMPLEACCOUNT)
BEGIN_COM_MAP(CSimpleAccount)
    COM_INTERFACE_ENTRY(IAccount)
END_COM_MAP()

DECLARE_PROTECT_FINAL_CONSTRUCT()

HRESULT FinalConstruct() {
    return S_OK;
}

void FinalRelease() {
}

public:
    STDMETHOD(Deposit)(DOUBLE amount);
    STDMETHOD(Withdraw)(DOUBLE amount);
    STDMETHOD(GetBalance)(DOUBLE* balance);
};

OBJECT_ENTRY_AUTO(__uuidof(SimpleAccount), CSimpleAccount)
```

- **Implements IUnknown**
- **Define class factory**
- **Our custom interface**
- **Registry script resource ID**
  - For registering this as a COM Class in the registry
- **Map for implementing IUnknown::QueryInterface**
- **Post construction**
- **Interface methods**
- **Export class for external clients**
A Typical Registry Script File (RGS)

HKCR
{
    AccountSvr.SimpleAccount.1 = s 'SimpleAccount Class'
    {
        CLSID = s '{E29B6DC7-6A8C-4F25-9589-4AA563C27380}'
    }
    AccountSvr.SimpleAccount = s 'SimpleAccount Class'
    {
        CLSID = s '{E29B6DC7-6A8C-4F25-9589-4AA563C27380}'
        CurVer = s 'AccountSvr.SimpleAccount.1'
    }
    NoRemove CLSID
    {
        ForceRemove {E29B6DC7-6A8C-4F25-9589-4AA563C27380} = s 'SimpleAccount Class'
        {
            ProgID = s 'AccountSvr.SimpleAccount.1'
            VersionIndependentProgID = s 'AccountSvr.SimpleAccount'
            InprocServer32 = s '%MODULE%'
            {
                val ThreadingModel = s 'Apartment'
            }
            'TypeLib' = s '{52BEAE9B-BD95-42A5-BBC5-57D220F606AC}'
        }
    }
}
ATL Smart Pointers

• Automate the acquiring, releasing and accessing of interface pointers
  • `CComPtr<T>`
    – Basic smart pointer
  • `CComQIPtr<T>`
    – Supports QueryInterface with a constructor or operator=
• Overload operators for convenient access to the underlying interface pointer (->, *, &)
• Destructor calls `IUnknown::Release`
Client Code with Smart Pointers

• `#include <atlbase.h>`

```cpp
CComPtr<IAccount> spAcc;
spAcc.CoCreateInstance(__uuidof(Account)); // calls CoCreateInstance
if (spAcc) {
    spAcc->Deposit(100);
    spAcc->Withdraw(1000);
    // no need to release interface pointer
    // can explicitly release with spAcc.Release() or spAcc = NULL
}
```

`__uuidof(Account)` can be used instead of `CLSID_Account`
Looking At COM Components

• Can use the OLE/COM Object Viewer (oleview.exe)
  – View registry information conveniently
  – Can create instances
  – Can “decompile” type libraries back into IDL
ATL Class Relationships

CComObjectRootBase

CComObjectRootEx<typename T>

CComCoClass<typename T, &CLSID_Account>

<<interface>>

IAccount

CAccount

CComObject<typename T>

Handles reference counting based on threading model (see later)
Either CComSingleThreadModel or CComMultiThreadModel

Defines class factory and aggregation model

Our custom interface

Our component class
An abstract base class

Implements IUnknown
Concrete class instantiated by class factory
COM Object Reuse

• Traditional OOP reuse is through containment or inheritance

• Containment
  – COM supports containment easily
  – A COM object can be a client of another COM object
  – Can even implement same interface and provide delegation

• Inheritance
  – As a binary standard, COM cannot support true inheritance, as not enough common ground exists between languages
  – As an alternative, COM supports aggregation
Containment vs. Aggregation

Containment

Aggregation

IX

IA

delegates

or uses

IX

IY

delegates

Controlling IUnknown

IX

IY

IA
Aggregation Example

• The Account class aggregates a Calculator class

```cpp
class ATL_NO_VTABLE CSimpleAccount :
   public CComObjectRootEx<CComSingleThreadModel>,
   public CComCoClass<CSimpleAccount, &CLSID_SimpleAccount>,
   public IAccount
{
  public:

  DECLARE_REGISTRY_RESOURCEID(IDR_SIMPLEACCOUNT)
  BEGIN_COM_MAP(CSimpleAccount)
      COM_INTERFACE_ENTRY(IAccount)
      COM_INTERFACE_ENTRY_AGGREGATE(IID_ICalculator, m_spUnkCalc.p)
  END_COM_MAP()

  DECLARE_GET_CONTROLLING_UNKNOWN()
  DECLARE_PROTECT_FINAL_CONSTRUCT()

  HRESULT FinalConstruct() {
      return m_spUnkCalc.CoCreateInstance(CLSID_Calculator, GetControllingUnknown());
  }

  CComPtr<IUnknown> m_spUnkCalc;
  ...
};

CComPtr<IAccount> spAcc;
spAcc.CoCreateInstance(__uuidof(SimpleAccount));
if(spAcc) {
    CComQIPtr<ICalculator> spCalc(spAcc);
    ATLASSERT(spCalc);
    // do something with spCalc...
    // can go the other way around
    CComQIPtr<IAccount> spAcc2(spCalc);
    ATLASSERT(spAcc2);
}
```
Processes & Threads

• Process
  – A management and containment object, providing resources to execute a program
  – Manages a private virtual address space
    • By default 2 GB (32 bit), 8 TB (64 bit)

• Thread
  – Entity scheduled by the kernel to execute code
  – Has a stack, priority, optional message queue
  – Has a scheduling state (running, ready, waiting)
COM & Threading

• Client code may have multiple threads
  – May create objects on many threads
  – May access an object concurrently from multiple threads
  – Object may call back to the client

• Objects may have multiple threads
  – Worker threads
  – May access object state

• Some objects are thread safe, some are not

• COM Apartments provide a solution
COM Apartments

• An “Apartment” groups objects with the same concurrency requirements
• A process may have any number of apartments
• Every object is associated with exactly one apartment
• Objects may be called by threads in their apartment only
• Cross apartment access requires a proxy
Apartments may be created explicitly, or implicitly by COM
Apartment Types

- **Single Threaded Apartment (STA)**
  - Only one thread can live in the apartment
  - Any number of STAs may exist in a process
  - Method calls from other apartments serialized by the thread’s message queue
    - Implies thread affinity
    - Thread safety implicitly provided

- **Multithreaded Apartment (MTA)**
  - Any number of threads can reside there
  - Zero or one MTA per process
  - Any thread in the MTA can call any object in there
    - No thread affinity
    - No thread safety

- **Thread Neutral Apartment (TNA)**
  - No threads may live there
  - Zero or one TNA per process
  - No thread safety, no thread affinity
  - Mostly useful in COM+ scenarios
Entering an Apartment

• A thread that wants to work with COM must enter an apartment by calling `CoInitializeEx`.
  - `COINIT_APARTMENTTHREADED` enters a new STA.
  - `COINIT_MULTITHREADED` enters the lone MTA.
  - `CoInitialize(0)` is equivalent to `CoInitializeEx(0, COINIT_APARTMENTTHREADED)`.

• Cannot switch apartments unless
  - Calls `CoUninitialize` to get out of its apartment.
Objects & Apartments

- Objects may have their own concurrency requirements
- A COM class residing in a DLL indicates its threading model via the registry

<table>
<thead>
<tr>
<th>Setting</th>
<th>Apartment Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single (or nothing)</td>
<td>Main STA</td>
<td>First STA in process (legacy setting)</td>
</tr>
<tr>
<td>Apartment</td>
<td>STA</td>
<td></td>
</tr>
<tr>
<td>Free</td>
<td>MTA</td>
<td></td>
</tr>
<tr>
<td>Both</td>
<td>STA or MTA</td>
<td>In the calling thread’s apartment</td>
</tr>
<tr>
<td>Neutral</td>
<td>TNA</td>
<td></td>
</tr>
</tbody>
</table>
Apartments & Marshaling

• When a thread calls \textbf{CoCreateInstance}
  – If the threading model of the class is where that thread is located, the object is created in that apartment and the thread gets back a direct pointer
  – Otherwise, a proxy is created (based on the interface requested) that marshals calls correctly to the target apartment

• A \texttt{ThreadingModel=“Both”} setting means the object is always created in the creator’s apartment

• Explicit marshaling is sometimes needed
Apartments Example

• STA client / STA object

1. `CoCreateInstance (...) &p);`
2. Method call
3. `CoCreateInstance (...) &q);`
4. Method call

```
STA client / STA object
p
CoCreateInstance (... &p);

Method call
MyObject

q
CoCreateInstance (... &q);

Method call
MyObject
```
Apartments Example

• MTA client / STA object

CoCreateInstance (... , &p);

Method call

SendMessage

MyObject

Proxy

MTA

STA
Explicit Marshaling

- An interface pointer is marshaled by using the **CoMarshalInterface** and **CoUnmarshalInterface** functions
  - Usually happens behind the scenes
  - Won’t be covered in this course

- Simpler alternatives exist
  - **CoMarshalInterThreadInterfaceInStream**, **CoGetInterfaceAndReleaseStream**
    - Suitable for some scenarios

- A more convenient way to marshal interfaces is using the Global Interface Table (GIT)
The Global Interface Table (GIT)

- One per process
- Can marshal direct interface pointers or pointers to proxies
- Accessed with the IGlobalInterfaceTable interface
  - Class id is CLSID_StdGlobalInterfaceTable

```c
interface IGlobalInterfaceTable : IUnknown {
HRESULT RegisterInterfaceInGlobal(
    [in, iid_is(riid)] IUnknown *pIUnk,
    [in] REFIID riid,
    [out] DWORD *pdwCookie);
HRESULT RevokeInterfaceFromGlobal(
    [in] DWORD dwCookie);
HRESULT GetInterfaceFromGlobal(
    [in] DWORD dwCookie,
    [in] REFIID riid,
    [out, iid_is(riid)] void **ppv);
}
```

- Marshal an interface into the GIT
- Get back a neutral cookie
- Revoke registration
- Unmarshal an interface into the target apartment (proxy created if needed)
Using the GIT Example

```cpp
CoInitialize(0); // STA

CComPtr<IAccount> spAcc;
spAcc.CoCreateInstance(__uuidof(SimpleAccount));
if(spAcc) {
    CComPtr<IGlobalInterfaceTable> spGIT;
    spGIT.CoCreateInstance(CLSID_StdGlobalInterfaceTable);
    DWORD cookie;
    spGIT->RegisterInterfaceInGlobal(spAcc, __uuidof(IAccount), &cookie);

    CreateThread(0, 0, MyThread, (void*)cookie, 0, NULL);
}

DWORD WINAPI MyThread(PVOID param) {
    CoInitializeEx(0, COINIT_MULTITHREADED); // MTA

    CComPtr<IAccount> spAcc;
    CComPtr<IGlobalInterfaceTable> spGIT;
    spGIT.CoCreateInstance(CLSID_StdGlobalInterfaceTable);
    spGIT->GetInterfaceFromGlobal((DWORD)param, __uuidof(IAccount), (void**) &spAcc);
    ATLASSERT(spAcc);
    // do something with spAcc...

    return 0;
}
```
The Free Threaded Marshalar (FTM)

• COM classes marked “Both”
  – Can live in an STA or MTA
    • Never created with a proxy
    • But a proxy might be created if object accessed from a different apartment

• How can such an object make sure it never gets a proxy in-process?
  – Aggregate the Free Threaded Marshalar (FTM)
  – Implements the IMarshal interface and always returns a direct pointer when unmarshaling
Aggregating the FTM

- Can use `CoCreateFreeThreadedMarshaler` - ATL wizard can help if just creating the class

```cpp
BEGIN_COM_MAP(CSomeClass)
...
COM_INTERFACE_ENTRY_AGGREGATE(IID_IMarshal, m_pUnkMarshaler.p)
END_COM_MAP()

HRESULT FinalConstruct() {
    return CoCreateFreeThreadedMarshaler(
        GetControllingUnknown(), &m_pUnkMarshaler.p);
}
```
FTM Issues

• An object aggregating the FTM should not store pointers to other COM objects that do not use the FTM
  – Can lead to COM rule violation (such as the unhelpful `RPC_E_WRONGTHREAD` HRESULT)

• Solution
  – Use the GIT to get back cookies and store those
  – Cookies are apartment agnostic
  – Create an interface pointer on the fly whenever needed with the help of the GIT and the cookie
Summary

• COM is a binary standard for object communication
• Interface based programming
• Dynamic loading and location transparency
• ATL helps with building COM servers and clients
• The Apartments model helps in dealing with concurrency issues
INTRODUCTION TO WINDBG
WinDbg Overview

• WinDbg is a standalone GUI debugger
  – No need to install – simply xcopy
  – Used by Microsoft to debug Windows itself
  – User mode or kernel mode debugger

• UI windows
  – Command – most important window
  – Call Stack, Processes & Threads, Source, Locals, Watch, Registers, others

• Command window can do anything
  – Some shortcuts available through the GUI
WinDbg Workspaces

- WinDbg uses Workspace to keep the debugging session settings
  - This is somehow similar to VS modes
- There are several workspaces
  - A base workspace – no debugging session
  - Default user-mode workspace
  - Remote default workspace
  - Kernel-mode workspace (& Processor-specific) workspace
  - Named workspace
- For each debugging session (including dump analysis) a new workspace is created
- Workspaces are cascaded, i.e. when debugging user-mode app, the setting is accumulated.
Starting Process Debugging

• WinDbg can debug multiple processes at the same time
  – Create a new process
    .create <exe_path>
    • Or File->Open Executable...
  – Attach to a running process
    • .attach <process_id>
    • Or File->Attach to a Process...

• By default, stops at Initial Breakpoint
Starting WinDbg from cmd.exe

```
windbg.exe <app_to_start>
Windbg.exe –p <PID>
Windbg.exe –pn <exe_name>
Windbg.exe –z <crash_dump_file>
```
Stop Debugging

• Stop the debugging session by:
  – q command – quit
  – .detach
  – From the menu:
Debugging Information Windows

• Most of the WinDBG windows resemble the Visual Studio debugger main windows
  – Watch, Local, Registers, Disassembly, Process & Threads and Call Stack
  – Many of them provide more information

• The **Command Window** is the main debugger information window

• The **Scratch Pad** is like notepad

• The Command Browser is a GUI version of the Command Windows

• The Source windows display the source code
Debugger Commands

- The command prompt shows the ordinal process number : thread number
- To break long command execution use the Debug->Break menu, or CTRL+BREAK
- `cls` command clears the command window text
- Debugger Command Programs
  - small application that consists of debugger commands and control flow tokens,
    - `if, .else, .elsif, .foreach, .for, .do, .break, .while, ...
    - `.block { }`
  - You can execute a program from the command window or from a script
WinDbg Command Types

• Regular commands
  – Intrinsic to WinDbg
  – Have no prefix
  – Work on the debugged process

• Meta commands
  – Work on the debugger itself or the process of debugging itself
  – Prefixed with a dot (.)

• Extension commands (“bang” commands)
  – Supplied by extension (custom) DLLs
  – Prefixed with an exclamation mark (!)
  – Some extension DLLs are loaded automatically
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>B[A</td>
<td>C</td>
</tr>
<tr>
<td>D[type]&lt;range&gt;</td>
<td>Dump memory</td>
</tr>
<tr>
<td>DT ...</td>
<td>Dump using type information</td>
</tr>
<tr>
<td>DV [name]</td>
<td>Dump local variables</td>
</tr>
<tr>
<td>E[type] &lt;add&gt; &lt;vals&gt;</td>
<td>Enter memory values</td>
</tr>
<tr>
<td>G[H</td>
<td>N [...] P T</td>
</tr>
<tr>
<td>K</td>
<td>KP</td>
</tr>
<tr>
<td>LM[k</td>
<td>l</td>
</tr>
<tr>
<td>LN &lt;expr&gt;</td>
<td>list nearest symbols</td>
</tr>
<tr>
<td>Q</td>
<td>Quit</td>
</tr>
<tr>
<td>R [[&lt;reg&gt;[= &lt;expr&gt;]]</td>
<td>view or set registers</td>
</tr>
<tr>
<td>S&lt;opts&gt; &lt;range&gt; &lt;values&gt;</td>
<td>Search memory</td>
</tr>
<tr>
<td>SX [...]</td>
<td>Event filter</td>
</tr>
<tr>
<td>U &lt;range&gt;</td>
<td>Unassemble</td>
</tr>
<tr>
<td>X [&lt;*</td>
<td>module&gt;!<em>]&lt;</em></td>
</tr>
</tbody>
</table>
Meta (Dot) Commands

- Control the behavior of the debugger, or the flow of execution

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.help</td>
<td>List all meta-command</td>
</tr>
<tr>
<td>.hh</td>
<td>Open the Debugger help viewer</td>
</tr>
<tr>
<td>.attach</td>
<td>Attach to process</td>
</tr>
<tr>
<td>.detach</td>
<td>Detach from process</td>
</tr>
<tr>
<td>.kill</td>
<td>End debugging session and close the target application</td>
</tr>
<tr>
<td>.beep</td>
<td>Speaker beep</td>
</tr>
<tr>
<td>.create</td>
<td>Create Process</td>
</tr>
<tr>
<td>.dump</td>
<td>Create dump file</td>
</tr>
<tr>
<td>.echotime</td>
<td>Display the current time</td>
</tr>
<tr>
<td>.ecxr</td>
<td>Display exception context</td>
</tr>
<tr>
<td>.eventlog</td>
<td>Display recent debug events</td>
</tr>
</tbody>
</table>
Extension Commands

• Extension commands are dll based debugger commands
  – They extend the debugger
  – The can be tailored to specific needs
  – You can develop your own

• Use .load to load extension Dll

• To use extension command
  – ![module.]extension [arguments]
## Some User-Mode Ext. Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>!analyze</td>
<td>Displays information about the current exception</td>
</tr>
<tr>
<td>!help</td>
<td>List extension commands</td>
</tr>
<tr>
<td>!address</td>
<td>Displays the address space layout</td>
</tr>
<tr>
<td>!cpuid</td>
<td>Displays CPU version info for all CPUs</td>
</tr>
<tr>
<td>!error [err]</td>
<td>Display Win32 error</td>
</tr>
<tr>
<td>!exchain</td>
<td>Display exception chain</td>
</tr>
<tr>
<td>!for_each_frame &lt;cmd&gt;</td>
<td>Executes command for each frame in current thread</td>
</tr>
<tr>
<td>!for_each_local &lt;cmd&gt;</td>
<td>Executes command for each local variable</td>
</tr>
<tr>
<td>!gle</td>
<td>Displays last error &amp; status for current thread</td>
</tr>
<tr>
<td>!obja &lt;address&gt;</td>
<td>Displays OBJECT_ATTRIBUTES[32</td>
</tr>
<tr>
<td>!std_map &lt;address&gt;</td>
<td>Displays a std::map&lt;&gt;</td>
</tr>
<tr>
<td>![u]str &lt;address&gt;</td>
<td>Display string</td>
</tr>
<tr>
<td>!gflag</td>
<td>Sets or displays the global flags</td>
</tr>
</tbody>
</table>
# Threads

<table>
<thead>
<tr>
<th>Thread identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>~.</td>
<td>The current thread</td>
</tr>
<tr>
<td>~#</td>
<td>The thread that caused the current exception or debug event</td>
</tr>
<tr>
<td>~*</td>
<td>All threads in the process</td>
</tr>
<tr>
<td>~Number</td>
<td>The thread whose ordinal is Number.</td>
</tr>
<tr>
<td>~[TID]</td>
<td>The thread whose thread ID is TID. (The brackets are required and you cannot add a space between the second tilde and the opening bracket.)</td>
</tr>
<tr>
<td>~[Expression]</td>
<td>The thread whose thread ID is the integer to which the numerical Expression resolves.</td>
</tr>
<tr>
<td>Process identifier</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>.</td>
<td>The current process</td>
</tr>
<tr>
<td>#</td>
<td>The process that caused the current exception or debug event</td>
</tr>
<tr>
<td>*</td>
<td>All processes</td>
</tr>
<tr>
<td>Number</td>
<td>The process whose ordinal is Number.</td>
</tr>
<tr>
<td>~[PID]</td>
<td>The process whose process ID is PID</td>
</tr>
<tr>
<td>[Expression]</td>
<td>The process whose process ID is the integer to which the numerical Expression resolves</td>
</tr>
<tr>
<td>Number s</td>
<td>Select process whose ordinal is number</td>
</tr>
</tbody>
</table>
Self Repeating Commands

• The j (Execute If-Else) command
  `bp `loop.cpp:143`` "j (poi(i)>0n20) ' '; 'gc' "`

• The z (Execute While) command
  `r$t0 = 0 ?? (wchar_t *)envp[@$t0]; r$t0 = @$t0 + 1; z (@$t0 < 10)`

• The ~e (for each thread) command
  `~*e kb 2`

• The !list extension command
  `!list "-t ntdll!_LIST_ENTRY.Flink -e -m 3 -x \"dd @$extret 14; dt ntdll!_RTL_CRITICAL_SECTION_DEBUG @$extret-0x8\" ntdll!RtlCriticalSectionList"`
Aliases

- There are three kinds of aliases:
  - user-name
    - `as` chl kernel32!CreateHardLinkW
    - `.echo` chl
    - `ad` chl
    - `as v pt;kb`
  - fixed-name
    - 10 predefined aliases `u0` - `u9`
  - automatic-aliases
    - like pseudo registers
- Use the `{{}}` to interpret an alias
Evaluating Expressions

- WinDbg has two different expression types
  - C++ and MASM (MASM is the default)
  - In MASM, all symbols are treated as addresses

- Use the `.expr /s c++`

- The `?` evaluate an expression

- The `??` Evaluate C++ expression regardless .expr

- Mixing: `@@c++( expr )` or `@@masm( expr )`

- By default numbers are decimal, 010 – octal, 0x10 – hex and you can use the L, U, and I64 or any C++ suffixes

- Use the regular [C++ operators](https://en.cppreference.com/w/cpp/language/operator一览)
MASM Operators

• The default expression mode is MASM
• There are set of useful operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hi</td>
<td>Hi 16 bit</td>
</tr>
<tr>
<td>low</td>
<td>Low 16 bit</td>
</tr>
<tr>
<td>not</td>
<td>1 if the argument is zero, zero for non-zero value</td>
</tr>
<tr>
<td>by</td>
<td>Low-order byte</td>
</tr>
<tr>
<td>wo</td>
<td>Low-order word</td>
</tr>
<tr>
<td>dwo</td>
<td>Double word from the specified address</td>
</tr>
<tr>
<td>qwo</td>
<td>Quad word from the specified address</td>
</tr>
<tr>
<td>poi</td>
<td>Pointer sized data from the specified address</td>
</tr>
<tr>
<td>$scmp(&quot;S1&quot;,&quot;S2&quot;)</td>
<td>Like strcmp ($sicmp is like stricmp(</td>
</tr>
</tbody>
</table>
Reading Variable Values

Source: int Add(int a, int b) { int r = a + b; return r; }

> .expr /s masm
Current expression evaluator: MASM - Microsoft Assembler expressions
> dd r L1
00000000`0018f660 00000003
> ?? r
Evaluate expression: 1635936 = 00000000`0018f660
> ? dwo(r)
Evaluate expression: 3 = 00000000`00000003
> ? poi(r)
Evaluate expression: -3689348818177884157 = cccccccc`00000003
> ?? r
int 3
> .expr /s C++
Current expression evaluator: C++ - C++ source expressions
> ? r
Evaluate expression: 3 = 00000000`00000003
> ? &r
Evaluate expression: 1635936 = 00000000`0018f660
## Macros for C++ Expressions

<table>
<thead>
<tr>
<th>Macro</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>#CONTAINING_RECORD(Address, Type, Field)</td>
<td>Returns the base address of an instance of a structure</td>
</tr>
<tr>
<td>#FIELD_OFFSET(Type, Field)</td>
<td>Returns the byte offset of a field in a known structure type</td>
</tr>
<tr>
<td>#RTL_CONTAINS_FIELD (Struct, Size, Field)</td>
<td>Indicates whether the given byte size includes the desired field</td>
</tr>
<tr>
<td>#RTL_FIELD_SIZE(Type, Field)</td>
<td>Returns the size of a field in a structure of known type</td>
</tr>
<tr>
<td>#RTL_NUMBER_OF(Array)</td>
<td>Returns the number of elements in an array</td>
</tr>
<tr>
<td>#RTL_SIZEOF_THROUGH_FIELD(Type, Field)</td>
<td>Returns the size of a structure of known type, up through and including a specified field</td>
</tr>
</tbody>
</table>
Configuring Symbols

• Debugging symbols come in several flavors
  – Full program database (PDB files)
  – Public symbols only (PDB files)
  – Exported symbols only (in the DLL itself)

• Select File->Symbol File Path…
  – Add search folders as appropriate

• Automatically uses the _NT_SYMBOL_PATH environment variable

• To get the symbols of the OS DLLs automatically, add the following string
  
  SRV*c:\Windows\Symbols*http://msdl.microsoft.com/download/symbols
Symbol Commands

```
.reload [options] [module]
   - Reload all symbol files (useful if symbol path updated)
   - Useful options
     /f (forces loading of symbols now)

lm [options]
   - Lists all loaded modules
   - Shows in parenthesis the symbol loading status
   - WinDbg uses deferred symbol loading
     lm v m [module] (shows detailed info for module)

ld <module_name>
   - Loads symbols for the specified module
```
Examining Symbols

x [options] <module!symbol>

x [options] *

- Searches and displays all symbols matching a pattern
- Module and symbol can contain wildcards
- x * displays all local variables

• Options

/t (display data type if known)
/v (display type and size of symbol)
/s size (displays only symbols matching size in bytes)
Symbol Syntax & Matching

- `[module]![symbol]`
  - `??` DumpFiles!zero
- What is the result of: `?face`?
- Some commands can take a string wildcard
  - `*` represents zero or more characters
  - `?` represents any single character
  - `[ ]` contain a list of characters represent any single character in the list. You can use a hyphen ( - ) to specify a range.
  - `#` represents zero or more of the preceding characters.
  - `+` represents one or more of the preceding characters. \ is used as escape character
- `x kernel32!Create*[WA]`
The ss command is used to set the current suffix value:

```
0:00> ss n
0:00> bp kernel32!CreateHardLink
Could't resolve error at 'kernel32!CreateHardLink'
0:00> ss a
0:00> bp kernel32!CreateHardLink
0:00> bl
 0 e 00000000`76c2bcc0 0001 (0001) kernel32!CreateHardLinkA
0:00> ss w
0:00> bp kernel32!CreateHardLink
0:00> bl
 0 e 00000000`76c2bcc0 0001 (0001) kernel32!CreateHardLinkA
 1 e 00000000`76c2b9c0 0001 (0001) kernel32!CreateHardLinkW
```
Pseudo Registers

- Variables maintained by the debugger that hold certain values
  - All pseudo registers begin with the $ sign
  - To distinguish from other symbols use the @
  - Use the r command to set a value
    - You don’t need the @ sign when using the r command
  - \( r \quad \$t1 = @$t1 + 1 \)
<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$exp</td>
<td>The last expression that was evaluated</td>
</tr>
<tr>
<td>$ra</td>
<td>The return address that is currently on the stack</td>
</tr>
<tr>
<td>$ip</td>
<td>The instruction pointer register (eip or rip)</td>
</tr>
<tr>
<td>$exentry</td>
<td>The address of the entry point of the current process</td>
</tr>
<tr>
<td>$retreg</td>
<td>The primary return value register (eax, rax)</td>
</tr>
<tr>
<td>$csp</td>
<td>The current call stack pointer (esp, rsp)</td>
</tr>
<tr>
<td>$p</td>
<td>The value that the last d* (Display Memory) command</td>
</tr>
<tr>
<td>$peb</td>
<td>The address of the PEB block</td>
</tr>
<tr>
<td>$teb</td>
<td>The address of the TEB block</td>
</tr>
<tr>
<td>$tpid $tid</td>
<td>The Process and Thread Ids</td>
</tr>
<tr>
<td>$bp number</td>
<td>The address of breakpoint number ($bp1)</td>
</tr>
<tr>
<td>$frame</td>
<td>The current frame index</td>
</tr>
<tr>
<td>$dbgtime</td>
<td>The current time</td>
</tr>
<tr>
<td>$callret</td>
<td>The return value of the last function that .call called</td>
</tr>
</tbody>
</table>
User Defined Pseudo Registers

• There are 20 user-defined pseudo-registers
  – $t0, t1, \ldots, t19$
• Variables that you can read and write through the debugger
• You can store any integer value
• The default preset value is 0
• They can be especially useful as loop variables
• They can be very useful inside scripts
Pseudo Register Example

• What does the following line print?

```c
.for (r $t1=1; @$t1 <= @@C++(10U); r $t1 = @$t1 + 1) {
  .for (r $t2 = 1; @$t2 <= @@C++(10U); r $t2 = @$t2 + 1) {
    .printf "%d\t", @@C++(@$t1 * @$t2);
  }
} .echo;
```
Log File

• You can save the Debugger command window to a log file:
  .logopen [filename]
  .logopen /t c:\temp\mylog.txt

• To append to existing log use:
  .logappend [filename]

• To close the file use:
  .logclose

• .logfile displays the log file status
Breakpoints

• The breakpoint window can be found under the Edit top-level menu
• The command edit-box is like small command window line
• You can specify breakpoint addresses, module and routine offsets, or source file and line number
## Controlling Breakpoints

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F9</td>
<td>Set a breakpoint in the source window</td>
</tr>
<tr>
<td><code>.bpcmds</code></td>
<td>List all breakpoints with the commands that created them</td>
</tr>
<tr>
<td><code>bp</code></td>
<td>Set a new breakpoint</td>
</tr>
<tr>
<td><code>bm</code></td>
<td>Set new breakpoints on symbols that match a specified pattern</td>
</tr>
<tr>
<td><code>ba</code></td>
<td>Set a processor breakpoint (data breakpoint)</td>
</tr>
<tr>
<td><code>bc</code></td>
<td>Breakpoint clear</td>
</tr>
<tr>
<td><code>bd</code></td>
<td>Disable breakpoint</td>
</tr>
<tr>
<td><code>be</code></td>
<td>Re-enable one or more disabled breakpoints</td>
</tr>
<tr>
<td><code>br</code></td>
<td>Change the ID of an existing breakpoint</td>
</tr>
<tr>
<td><code>bs</code></td>
<td>change the command associated with an existing breakpoint</td>
</tr>
<tr>
<td><code>bsc</code></td>
<td>Change a breakpoint condition</td>
</tr>
</tbody>
</table>
Processor Breakpoints Example

```plaintext
?? (wchar_t *)name._Bx._Alias
Evaluate expression: 2030176 = 00000000`001efa60
>ba r4 0x1efa60
>bl
1 e 00000000`001efa60 r 4 0001 (0001) 0:****
>g
breakpoint 1 hit
DataBreakPoint!std::char_traits<wchar_t>::assign+0x1b:
00000001`3f8a1a6b 5f pop rdi
```
Complex Breakpoints

• Break only from thread 2 after 3 passes
  ~2  bu  MyFunc  3
• Break on member method:
  bp  @@C++(Person::GetName)
• Breakpoint with command
  ba  w4  0x71a578a8  "k;g"
  bu  MyDll!func  ".dump  c:\tmp\dump.dmp; g"
• Conditional Breakpoint
  bp  'x.cpp:14'  "j (poi(i)>5) ''; 'gc' "
Walking the Stack

• Open the Call Stack window
  – View->Call Stack

• Issue the k command

• Common variants
  kP (lists all function parameters)
  kb (lists first 3 parameters to each function)
  k [n] (lists no more than n stack frames)
Displaying Memory

d[a|b|c|d|D|f|p|q|u|w|W] [/c width] [range]
  - a=ASCII, b=byte & ASCII, c=DWORD & ASCII, d=DWORD, D=double, f=float, p=pointer size, q=QWORD, u=Unicode, w=WORD, W=WORD & ASCII

dt [options] [address]
  - Displays types and/or data
    -a <count> (displays array elements)
  - Check the docs

!address [address | -summary -filter]
  - Displays status of a memory block (or entire process address space)
Additional Useful Commands

- ~* k – call stack for all threads
- .pcmd -s "r $tpid, $tid" - issue a command whenever the target stops executing
- .call ShowRestartMessage(envp) – call a function
- .reload - Reload modules and symbols
- !dlls - list of loaded modules
- lm vm kernel32 – verbose info
- !dh ntdll – display PE header
- !runaway 7 – time consumed by thread
- .dml_start – Start Debugger Markup Language
- kM – DML stack command
Using dt – Display Type

• The **Display Type** command is very powerful

  dt [-DisplayOpts] [-SearchOpts] [module!]Name [[-SearchOpts] Field] [Address] [-l List]
  dt [-DisplayOpts] Address [-l List]
  dt -h

```
0:000> dt nt!_PEB CriticalSectionTimeout. 000007fffffff8000
ntdll!_PEB
  +0x0c0 CriticalSectionTimeout : 0xfffffe86\`079b8000
    +0x000 LowPart : 0x79b8000
    +0x004 HighPart : -6035
    +0x000 u : <unnamed-tag>
    +0x000 QuadPart : -25920000000000
```
Event Filters

- Like the exception window of Visual Studio
- It is based on the WaitForDebugEvent events
WinDbg Scripts

• A script file is a text file that contains a sequence of debugger commands
• For CDB & NTSD, a ntsd.ini file script will start when starting the debugger
• The `${argn}` can be used inside a script to read input arguments
• Scripts can call other scripts
• Run script with: `<, $><, $$<, $$><, $$>a<
• For more information:
Example

$$ Get module list LIST_ENTRY in $t0.$

r? $t0 = &@$peb->Ldr->InLoadOrderModuleList

$$ Iterate over all modules in list.

.r? $t1 = *(ntdll!_LDR_DATA_TABLE_ENTRY**)@$t0;
(@$t1 != 0) & (@$t1 != @$t0);
   r? $t1 = (ntdll!_LDR_DATA_TABLE_ENTRY*)@$t1->InLoadOrderLinks.Flink

{ $$ Get base address in $Base.$

   as /x $/v:$Base} @$c++(@$t1->DllBase)

$$ Get full name into $Mod.$

   as /msu $/v:$Mod} @$c++(&@$t1->FullDllName)

.block

   { .echo ${$Mod} at ${$Base}

} ad ${/v:$Base}

   ad ${/v:$Mod}

}
Kernel Debugging

• Kernel debugging can assist user-mode debugging
  – Memory, IO, Thread & Process information from the kernel can reveal problems in user-mode code
• You can do a local or remote kernel debugging
• You can remote debug a local virtual machine
• On Window NT 6.x a `bcd` setting is required:
  – `bcdedit -debug`
  – `bcdedit -dbgsettings` ...
Remote Debugging With WinDbg

• On the target (Server) machine run:
dbgsrv.exe –t tcp:port=6160
  • it needs the dbgeng.dll & dbghlp.dll
    – Open the firewall for dbgsrv.exe

• On the host (client) machine run WinDbg
  WinDbg –premote tcp:server=<machine ip or name>, port = 6160

• Use the Attach to process to start debugging
Minidump Files

• A minidump is a snapshot of a process
  – May be created at any time, not just when a process crashes

• Minidump types
  – Kernel mindumps (not relevant for this course)
  – Basic (usually enough for native processes)
  – Full (required to get useful info for managed processes)

• Minidump creation
  .dump [options] <filename>
  – On Vista & 2008 use Task Manager
  – ADPlus
  – ProcDump (SysInternals)
Minidump Creation

- **WinDbg**
  
  \texttt{dump [options] filename}
  
  - Options
    
    - \texttt{/ma} (full minidump)
    - \texttt{/o} (overwrite existing file)
    - \texttt{/u} (ensure unique filename)
    - \texttt{/c} (add a comment)
    - \texttt{/b} (compress to CAB)

- **ADPlus**
  
  - Hang mode – noninvasive attach
  - Crash mode – attaches the CDB debugger
  - Common options
    
    - \texttt{-hang} (hang mode)
    - \texttt{-crash} (crash mode)
    - \texttt{-pn} (specify process name including extension)
    - \texttt{-p} (specify process ID)
    - \texttt{-c} (specify XML config file to read options from)
    - \texttt{-quiet} (don’t show various confirmation dialogs)
Opening a Minidump

- In WinDbg, File->Open Crash Dump...
- WinDbg –z <dump_file>
- Issue the “magical” command
  - !analyze –v
- Can use most other WinDbg/SOS commands
- You can also open a user mode dump file with Visual Studio
Application Verifier

- The `!avrf` extension controls the settings of App Verifier and displays a variety of output produced by Application Verifier

```bash
>!avrf
Verifier package version >= 3.00
Application verifier settings (80643027):
  - full page heap
  - Handles
`g
VERIFIER STOP 0000000000000300: pid 0x315C: Invalid handle exception for current stack trace.

  00000000C0000008 : Exception code.
  00000000002CFA30 : Exception record. Use .exr to display it.
  00000000002CF540 : Context record. Use .cxr to display it.
  0000000000000000 : Not used.
```
Application Verifier

0:000> !avrf -cnt
WaitForSingleObject calls: 0
WaitForSingleObjectEx calls: 0
WaitForMultipleObjects calls 0
WaitForMultipleObjectsEx calls: 0
Waits with timeout calls: 0
Waits with timeout failed: 0
CreateEvent calls: 2
CreateEvent calls failed: 0
Heap allocation calls: 81
Heap allocations failed: 0
CloseHandle called with null handle: 0
CloseHandle called with pseudo handle: 0
Heaps created: 2
Heaps destroyed: 0
...
Resources

• Debugging Tools for Windows

• WinDbg Info: http://windbg.info/

• Common Commands
  – http://windbg.info/doc/1-common-cmds.html

• Application Verifier
Summary

• In this appendix we got deep into the powerful WinDbg Debugger

• To mitigate hard problems we need:
  ✓ Tools
  ✅ Deep Understanding
  – Experience
Thank You