Windows Foundation Concepts and Terms

Overview:

- Windows API
- Processes
- Threads
- Virtual memory
- Kernel mode and user mode
- Objects
- Handles
- Security
- Registry

Windows API (aka Win32 API): user-mode system programming interface for Windows OS family (refers to both 32-bit and 64-bit versions of Windows)

Flavors of API:

Component Object Model (COM): Windows API flavor that has Object Linking and Embedding (OLE), the ability to communicate and exchange data

1. Clients communicate with objects called COM server objects through interfaces
2. Component implementation is loaded dynamically rather than statically linked to the client

COM server refers to DLL or EXE that implements COM classes

Windows Runtime (WinRT): Windows API flavor for Windows Apps developers that is built on COM

.NET Framework components:

1. Common Language Runtime (CLR): JIT compiler that translates Common Intermediate Language (CIL) instructions to machine language, garbage collector, type verification, etc. Implement as COM in-process server (DLL)
2. .NET Framework Class Library (FCL): collection of types that implement client server functionality

Services, Functions, Routines

Native system services (system calls): undocumented, underlying services that are callable from user mode (ex NtCreateUserProcess is internal system service for CreateProcess)

Kernel support functions (routines): Subroutines that can only be called from kernel mode

Windows services: Processes started by the Windows service control manager

DLLs: Set of callable subroutines linked together as a binary. Only one in-memory copy of a DLL's code exists among the applications that are referencing it

Processes

Program: static sequence of instructions

Process: container of resources that the program uses during execution

Private virtual address space: set of virtual memory addresses that the process may use

Executable program: the initial code and data that gets mapped into the process' virtual address space

List of open handles: process threads access system resources (examples: semaphores, ports, files, synchronization objects) via these handles

Security context: access token that identifies the process' user, security groups, privileges, attributes, claims, capabilities, User Account Control (UAC) virtualization state, session, limited user account state and also the AppContainer identifier and its sandboxing information

Process ID: unique identifier (aka client ID)

At least one thread of execution: empty process is possible but not useful

Processes points to parent or creator process if parent exists

Parent not same as creator sometimes if parent used broker process

View process tree: tlist /t

Task Manager: right click header row and click Select Columns

Threads: number of threads in each process

Handles: number of handles to kernel objects opened by the process' threads

Status: should usually be running but can be waiting

Suspended state: occurs when all threads in the process are suspended

Call NtSuspendProcess to achieve this

For Win RT apps, suspend occurs when app loses foreground status (user minimized it)

Not Responding: occurs when the User Interface thread has not checked its message queue for at least 5 seconds

Threads:

An entity within the process that Windows schedules for execution. Components:

Set of CPU register contents that represent state of the processor

A stack for kernel mode execution

A stack for user mode execution

Thread-Local Storage (TLS): private storage area for subsystems, run-time libraries, DLLs

Thread ID: unique identifier that is part of the client ID

Security Context (aka token): allows multithreaded server applications to impersonate

Stored in an Access Token object, which contains the process':

Security identification

credentials

Thread Context: architecture-specific structure that includes volatile registers, stacks, private storage area

Access via GetThreadContext, which returns a CONTEXT block

Scheduling threads

Switching execution of threads involves the kernel scheduler

Windows implements two mechanism to reduce scheduling cost:

Fibers (aka lightweight threads):

Applications schedule their own threads

Implemented in Kernel32.dll in user mode → invisible to the kernel

ConvertThreadToFiber: converts thread to a running fiber

Use the fiber to create additional fibers via CreateFiber

Must call SwitchToFiber to start fiber execution

Note: issues sharing TLS, I/O fibers perform poorly, cannot run concurrently on more than one processor

User-mode scheduling (UMS) threads:

Available on 64-bit Windows

Have their own kernel thread state → visible to kernel

Can have multiple UMS threads

Can periodically switch execution contexts by yielding to another thread in user mode (does not involve the scheduler!)

Switches to its dedicated kernel-mode thread (directed context switch) when it needs to perform kernel operations (ex. System calls)

Threads share Process' Virtual Address space

All threads have full read-write access to the process virtual address space

Threads can reference address space of another process if either:

The remote process has a shared memory section (file mapping object) in part of its private address space

The thread's process has the right to open the remote process via ReadProcessMemory, WriteProcessMemory

Processes that run with the same user account have this can get this permission by default unless remote process has certain protections

Thread Access Token

Threads do not have their own access tokens by default but can obtain one

Can impersonate the security context of another process (including processes on a remote Windows system) without affecting other threads in the process

Jobs

An extension to the process model

Functions:

Allow management and manipulation of groups of processes as a unit

Has control over attributes of the associated processes

Provides limits for associated processes

Records accounting information for all associated processes (including terminated processes)

Virtual Memory

Provides each process with the illusion of having its own large, private address space

Memory manager (and hardware) maps (translates) the virtual addresses into physical addresses at run time

Ensures processes do not interfere with OS data

Pages:

Chunks of contiguous physical memory (default size 4KB)

Allows contiguous virtual memory chunks to be mapped to non-contigious chunks in physical memory

Memory Manager:

Frees physical memory for other processes (or for OS) by paging (transferring) memory contents to disk

Loads the information back into memory if a thread accesses virtual address that has been paged to disk

Hardware support assists Memory Manager:

Allows applications to not need to worry about paging

Allows Memory Manager to not need to know about processes or threads

Size of Virtual Address Space

32 bit x86: 4 GB:

Lower half of the address space:

0x00000000 through 0x7FFFFFFF

Allocated for processes

Upper half:

0x80000000 through 0xFFFFFFFF

Allocated for protected OS memory utilization

Increaseuserva qualifier in Boot Configuration Database gives processes with the large address space-aware flag the ability to use up to 3GB of private address space

Allows database servers to keep large portions of db in process address space

Address Windowing Extensions (AWE)

For mapping very large databases

Allows 32-bit app to allocate up to 64 GB of physical memory and then map views (windows) into its 2 GB virtual address space

Developer responsible for managing mapping of virtual to physical memory

64 bit x64:

128 TB address space on Windows 8.1, Server 2012 R2, and later

Unmapped region is about one million times larger than the possible mapped region

Kernel Mode vs User Mode

Windows has two different processor access modes to protect user applications from accessing and/or modifying OS data:

User Mode (Ring 3):

User application code

Kernel Mode (Ring 0):

OS code (system services, device drivers)

Grants access to all system memory and CPU instructions

Processor can only read/write to a page if it is in the access mode that the Page requires

Each page in virtual memory is tagged to indicate a required access mode

Pages in system space can only be accessed from kernel mode

Pages in user address space accessible from both user and kernel mode

Read-only pages not writable from any mode

Some processors mark pages as no-execute (Data Execution Prevention)

Kernel-mode OS and Device Driver code share a single virtual space

32-bit Windows does not protect private read/write system memory

OS and device driver code have complete access to system-space memory

Can bypass Windows security to access objects

Third-party device drivers also have access to all OS data

This risk lead to driver-signing mechanism (Windows 2000)

Warns (or blocks if configured) attempts to add unsigned plug-and-play drivers

Does not affect other types of drivers

64-bit Windows Kernel-Mode Code-Signing (KMCS) policy

All device drivers must be signed with SHA-1 certificate by a major code certification authority (20 authorities exist)

No user (including admin) may install an unsigned driver

Exception: Test Mode

Allows all drivers

Places a Test Mode watermark

Disables certain digital rights management (DRM) features

Windows 10 since 1607s

New Windows 10 drivers must be signed by one of two accepted certification authorities with SHA-2 Extended Validation (EV) Hardware certificate

EV-signed driver must be submitted to Microsoft through System Device (SysDev) portal for attestation signing (Microsoft signs it with a Microsoft signature)

Kernel will accept only Microsoft-signed Windows 10 drivers except in Test Mode

Windows Server 2016

Has same EV requirements as Windows 10

Driver must also pass through Windows Hardware Quality Labs (WHQL) certification as part of the Hardware Compatibility Kit (HCK)

Driver must then be submitted for formal evaluation

Some configurations of Windows have custom signing policies (ex Device Guard)

Might require WHQL signatures on Windows 10 client systems

Might request the omission of WHQL requirement on Windows Server 2016 systems

Switching from user mode to kernel mode

Occurs when user applications make a system service call

Example: ReadFile calls an internal Windows routine that accesses internal data structure, so it must run in kernel mode

Special processor instruction triggers transition

Processor enters the system service and dispatches code in the kernel

Calls internal function in Ntoskrnl.exe or Win32k.sys

Note: mode transition (user to kernel and back) is not a context switch

The same user thread executes in both user and kernel mode

Hypervisor

Specialized and highly privileged component that allows for the virtualization and isolation of all resources on the machine

virtual to physical memory, device interrupts, PCI and USB

Examples: Hyper-V, Xen, KVM, VMware, VirtualBox

Has greater access than the kernel

Can protect and monitor a single host instance to offer guarantees beyond what a kernel can provide

Windows 10 Hyper-V provides virtualization-based security (VBS)

Device Guard: Provides Hypervisor Code Integrity (HVCI) for stronger code-signing (than KMCS alone) and allows customization of signature policy on Windows OS for user mode and kernel mode code

Hyper Guard: Protects key kernel-related and hypervisor-related data structures and code

Credential Guard: Prevents unauthorized access to domain account credentials and secrets, combined with secure biometrics

Application Guard: Provides stronger sandbox for Microsoft Edge

Host Guardian and Shielded Fabric: Use virtual Trust Platform Module (v-TPM) to protect virtual machine from the host

Virtual Trust Levels (VTLs):

Protects VBS technologies from malicious/badly written drivers

VTL 0: OS and its components

VTL 1: VBS technologies

Cannot be affected by kernel mode code

Firmware

Windows components rely on OS security and kernel security

Kernel and OS rely on hypervisor security

Boot Loader responsible for loading components and authenticity checking

Windows 8 and later:

System firmware responsible for root chain of trust that guarantees the boot process

Must be UEFI-based on certified systems

UEFI Standard:

Secure boot implementation

Requires strong guarantees around critical parts of boot-related software

Windows components guaranteed to load securely from the boot process

MSFT manages whitelist and blacklist of the UEFI secure boot component

Windows updates include firmware updates

Trust Platform Module (TPM)

Chip that stores RSA encryption keys specific to the host system for hardware authentication – contains Endorsement Key (EK)

Terminal Services and Multiple Sessions

Terminal Services:

Windows support for multiple interactive user sessions (on a single system)

Allows remote user to establish a session on a server, log in, and run applications (remote desktop)

Can display the entire desktop or individual applications

Session 0: Services Session

Contains system service hosting process

Session 1: Login Session

First login session at the physical console of the machine

Additional sessions created through remote desktop connection program Mstsc.exe or through use of fast user switching

<u>Windows client</u>

only permits single remote user to connect to the machine if no one is logged in to the console

cannot have a remote and a local user at the same time

<u>Windows editions with Windows Media Center</u>

one interactive session and

up to four Windows Media Center Extender sessions

<u>Windows server systems</u>

Support two simultaneous remote connections

Facilitates remote management

Can support more than two remote sessions (must configure as a terminal server)

Fast User Switching (Switch Account option in Windows submenu)

Supports multiple session

Processes running in the session, session-wide data structures, remain active

Objects and Handles

Kernel Object: single, run-time instance of a statically defined object type

Object Type: comprises a system-defined data type, functions for instances, set of object attributes

Ex: process, thread, file, event

Based on lower-level objects that Windows creates and manages

Ex: Process is an instance of the process object type

Object attribute: private member data field in an object

Object method: manipulates objects by reading/changing attributes

Object accomplishes OS tasks:

1. Provide human-readable names for system resources
2. Share resources and data among processes
3. Protect resources from unauthorized access
4. Tracks references to the object for automatic deallocation

Data that needs share, protect, name, made visible to user-mode programs (via system services) are placed in objects

Security

Core security capabilities:

Discretionary (need-to-know) and mandatory protection for all sharable system objects (files, processes, etc)

Security auditing for accountability of users and user actions

User logon authentication

Prevention of user access to uninitialized resources

Access Control over objects:

1. Discretionary access control

Method by which owners of objects grant/deny access to others

When user attempts to access object

Compare user security context with access control list on the object

Windows Server 2012, Windows 8

Dynamic Access Control: attribute-based access control

Resource's access control list identifies required attributes/claims of a user that grant access

Attributes populated through Active Directory

2. Privileged access control
   Allows privileged accounts to access protected objects
   Ex: administrator access files of an employee that left the company
3. Mandatory integrity control
   Protects objects that are being accessed from within the same user account
   Includes protecting elevated administrator account objects from non-elevated

AppContainer: sandbox that hosts Windows Apps and provides isolation

Code in AppContainers can communicate with brokers (non-isolated processes running with user's credentials), other AppContainers, other processes through Windows Runtime interface

Multi-process focus rather than multi-thread focus

Registry

System database that contains the information required to boot and configures the:

System

System-wide software settings that control how Windows operates

Security database

Per-user configuration settings

Can access Windows performance counters via registry functions

Unicode

Uses 16-bit-wide Unicode characters (UTF-16LE) for internal text strings

Many applications deal with 8-bit ANSI character strings

Windows API functions have two version:

-A: function accepts ANSI strings

-W: function accepts Unicode

Kernel Debugging

Examines internal kernel data structures, steps through functions in the kernel

Symbol files generated by the linker

To examine kernel data structure (process list, thread blocks, etc)

Need symbol files for kernel image, Ntoskrnl.exe

Windows debuggers

cdb and ntsd: user mode debuggers

kd: kernel-mode debugger

WinDbg: user and kernel mode debugger with GUI

Invasive Debugging:

DebugActiveProcess function

Can only attach one debugger at a time

Noninvasive:

Debugger opens the process with OpenProcess

Can examine/change memory in target process but cannot set breakpoints

Kernel-mode Debugging:

Open crash dump

Connect to live, running system and examine system state

Connect through named pipe by exposing guest OS's serial port as named pipe device

Local network debugging by exposing host-only network using virtual NIC in guest OS (very fast performance)

Local kernel debugging: connecting to local system and examining system state