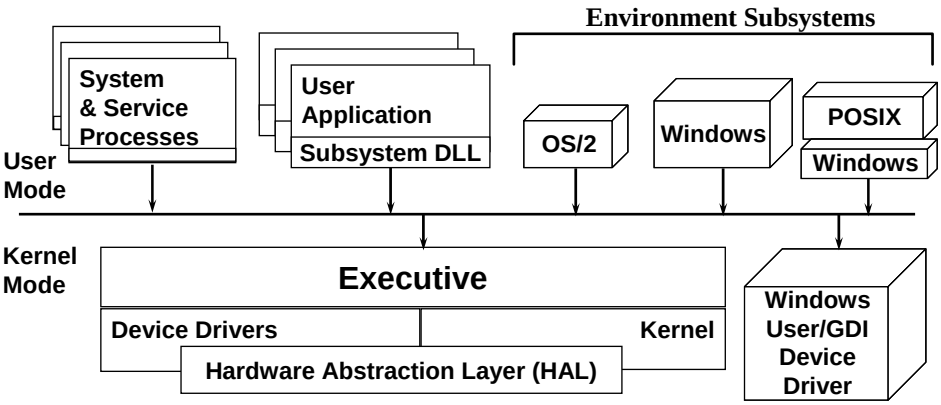
Unit OS2:
Operating System Principles

2.3. Windows on Windows - OS Personalities

Roadmap for Section 2.3.

- Environment Subsystems
- System Service Dispatching
- Windows on Windows - 16bit
- Windows on Windows - 64bit

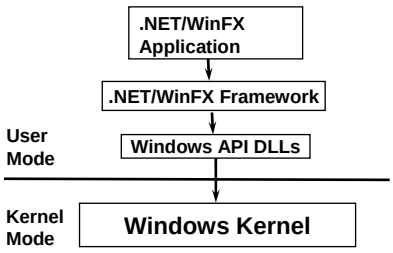
Multiple OS Personalities



4

What about .NET and WinFX?

- WinFX is the .NET Framework that will ship with Longhorn
- Both .NET and WinFX are built on standard Windows APIs
  - They are not a subsystem
  - They do not call undocumented Windows system calls



5

Environment Subsystems

- Environment subsystems provide exposed, documented interface between application and Windows native API
  - Each subsystem defines a different set of APIs & semantics
  - Subsystems implement these by invoking native APIs
    - i.e., subsystem “wraps” and extends Windows native API
    - Example: Windows CreateFile in Kernel32.Dll calls native NtCreateFile
- .exe’s and .dll’s you write are associated with a subsystem
  - Specified by LINK /SUBSYSTEM option
  - Cannot mix calls between subsystems

6

Environment Subsystems

- Three environment subsystems originally provided with NT:
  - Windows –Windows API (originally 32-bit, now also 64-bit)
  - OS/2 - 1.x character-mode apps only
    - Removed in Windows 2000
  - Posix - only Posix 1003.1 (bare minimum Unix services - no networking, windowing, threads, etc.)
    - Removed in Windows XP/Server 2003 – enhanced version ships with Services For Unix 3.0
    - Ships with Windows Server 2003 R2
- Of the three, Windows provides access to the majority of OS native functions
- Of the three, Windows is required to be running
  - System crashes if Windows subsystem process exits
  - POSIX and OS/2 subsystems are actually Windows applications
  - POSIX & OS/2 start on demand (first time an app is run)
    - Stay running until system shutdown

7

Subsystem Information in Registry

- Subsystems configuration and startup information is in:

  HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control
  \Session Manager\SubSystems

  Values:

  - Required - list of value names for subsystems to load at boot time
  - Optional - list of value names for subsystems to load when needed
  - Windows - value giving filespec of Windows subsystem (csrss.exe)

    csrss.exe Windows APIs required - always started when Windows boots
  - Kmode - value giving filespec of Win32K.Sys
    (kernel-mode driver portion of Windows subsystem)
  - Posix - file name of POSIX subsystem

    psxss.exe Posix APIs optional - started when first Posix app is run
- Some Windows API DLLs are in "known DLLs" registry entry:

  HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\KnownDLLs
  - Files are opened as mapped files
  - Improves process creation/image startup time

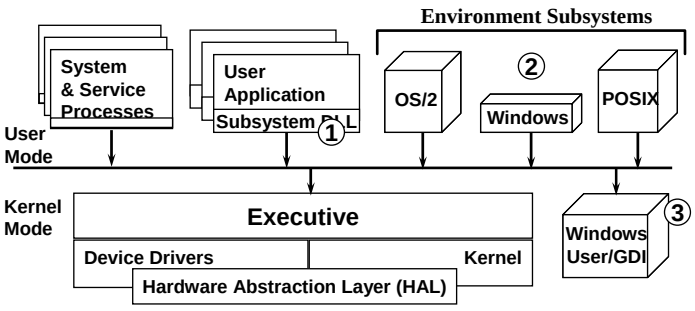
8

Subsystem Components

① API DLLs
- for Windows: Kernel32.DLL, Gdi32.DLL, User32.DLL, etc.

② Subsystem process
- for Windows: CSRSS.EXE (Client Server Runtime SubSystem)

③ For Windows only: kernel-mode GDI code
- Win32K.SYS - (this code was formerly part of CSRSS)



9

Role of Subsystem Components

① API DLLs

- Export the APIs defined by the subsystem
- Implement them by calling Windows “native” services, or by asking the subsystem process to do the work

② Subsystem process

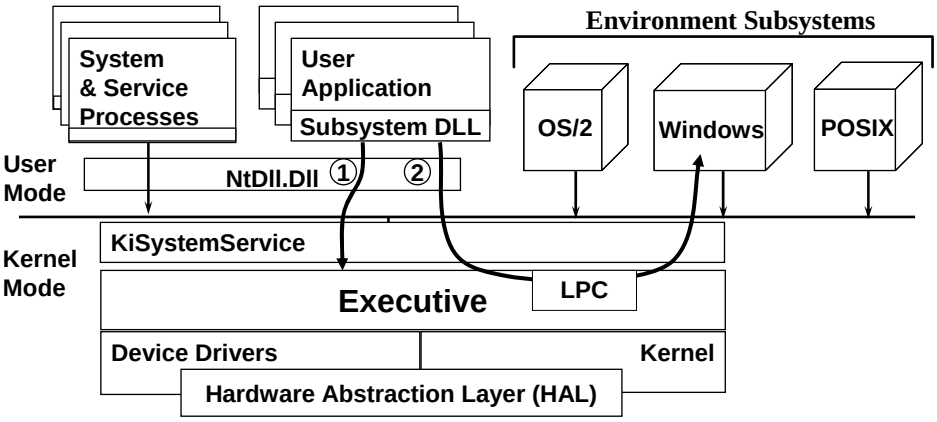
- Maintains global state of subsystem
- Implements a few APIs that require subsystem-wide state changes
  - Processes and threads created under a subsystem
  - Drive letters
  - Window management for apps with no window code of their own (character-mode apps)
  - Handle and object tables for subsystem-specific objects

③ Win32K.Sys

- Implements Windows User & GDI functions; calls routines in GDI drivers
- Also used by Posix and OS/2 subsystems to access the display

10

Simplified Architecture
(3.51 and earlier)

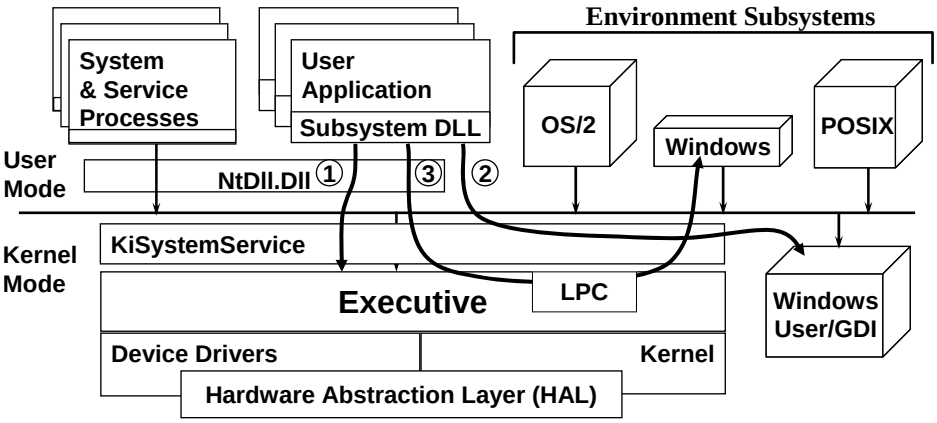


① most Windows Kernel APIs

② all other Windows APIs, including User and GDI APIs

11

Windows Simplified Architecture



① most Windows Kernel APIs
② most Windows User and GDI APIs (these were formerly part of CSRSS)
③ a few Windows APIs

12

Role of CSRSS.EXE
(Windows Subsystem Process)

- A few Windows APIs are implemented in this separate process
  - In 3.51 and earlier:
    - Nearly all User and GDI APIs were implemented in CSRSS
    - CSRSS had a thread for every application thread that created a window
    - GDI drivers (video, printer) were user mode, mapped into this process
    - This was done for protection, esp. to keep GDI drivers in user mode
- CSRSS in NT 4.0 and later: role is greatly diminished
  - Maintains system-wide state information for all Windows “client” processes
  - Several Windows services LPC to CSRSS for “setup and teardown” functions
    - Process and thread creation and deletion
    - Get temporary file name
    - Drive letters
  - Security checks for file system redirector
  - Window management for console (character cell) applications …
  - … including NTVDM.EXE

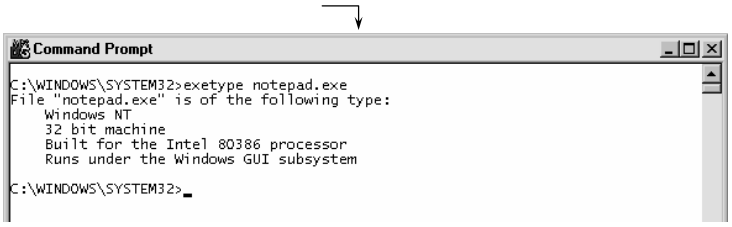
13

Header of Executable File Specifies Subsystem Type

- Subsystem for each .exe specified in image header
  - see winnt.h (in Platform SDK)

```
IMAGE_SUBSYSTEM_UNKNOWN     0   // Unknown subsystem
IMAGE_SUBSYSTEM_NATIVE      1   // Image doesn't require a subsystem
IMAGE_SUBSYSTEM_WINDOWS_GUI 2   // Windows subsystem (graphical app)
IMAGE_SUBSYSTEM_WINDOWS_CUI 3   // Windows subsystem (character cell)
IMAGE_SUBSYSTEM_OS2_CUI     5   // OS/2 subsystem
IMAGE_SUBSYSTEM_POSIX_CUI   7   // Posix subsystem
```

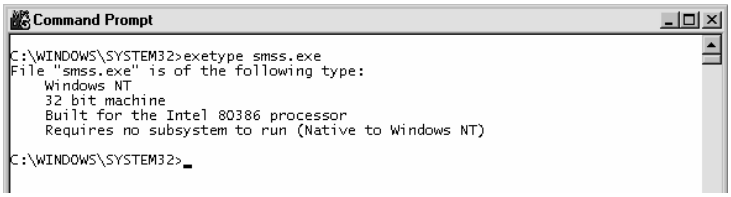
  - or exetype image.exe (2000 Resource Kit)



14

Native Images

- .EXEs not linked against any subsystem
  - Interface to Windows executive routines directly via NTDLL.DLL
- Two examples:
  - smss.exe (Session Manager -- starts before subsystems start)
  - csrss.exe (Windows subsystem)



15

Lab: Subsytems & Images

- Look at subsystem startup information in registry
- Using EXETYPE, look at subsystem types for:
  - \windows\system32\notepad.exe, cmd.exe, csrss.exe

16

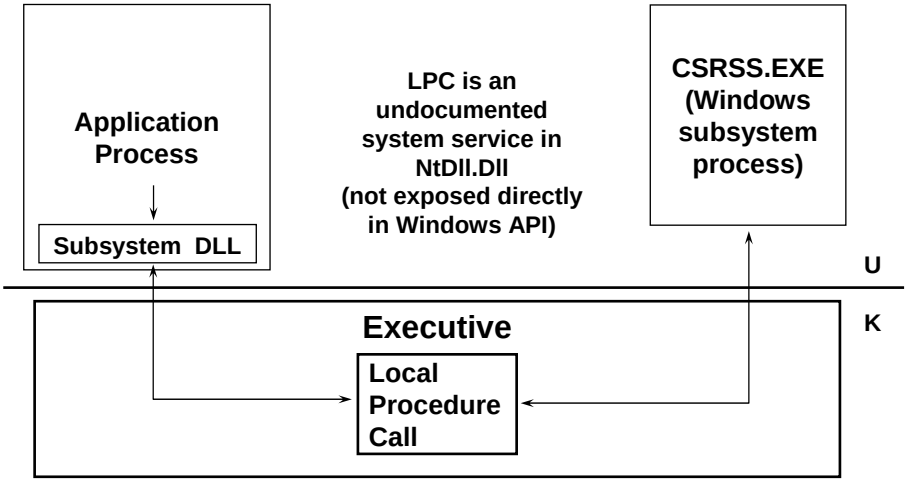
POSIX.1 Subsystem

- Original POSIX subsystem implemented only POSIX.1
  - ISO/IEC 9945-1:1990 or IEEE POSIX standard 1003.1-1990
  - POSIX.1 compliance as specified in Federal Information Processing Standard (FIPS) 151-2 (NIST)
  - POSIX Conformance Document in \HELP in Platform SDK
- Support for impl. of POSIX.1 subsystem was mandatory for NT
  - fork service in NT executive
  - hard file links in NTFS
- Limited set of services
  - such as process control, IPC, simple character cell I/O
  - POSIX subsystem alone is not a complete programming environment
- POSIX.1 executable cannot
  - create a thread or a window
  - use remote procedure calls (RPCs) or sockets

17

Invoking (a few) Windows Services

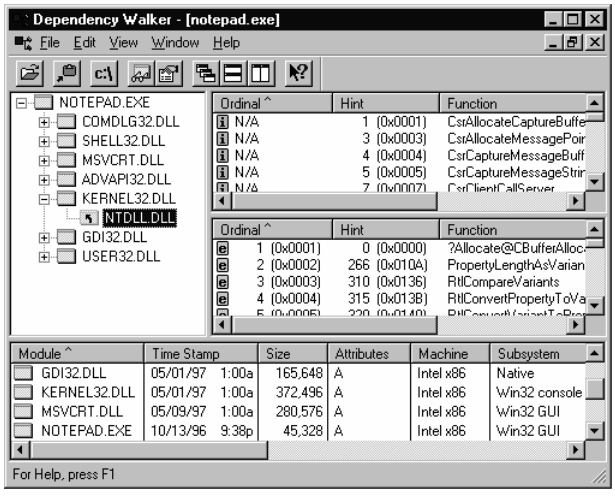
- **Some system calls still require communication with the Windows subsystem process**



18

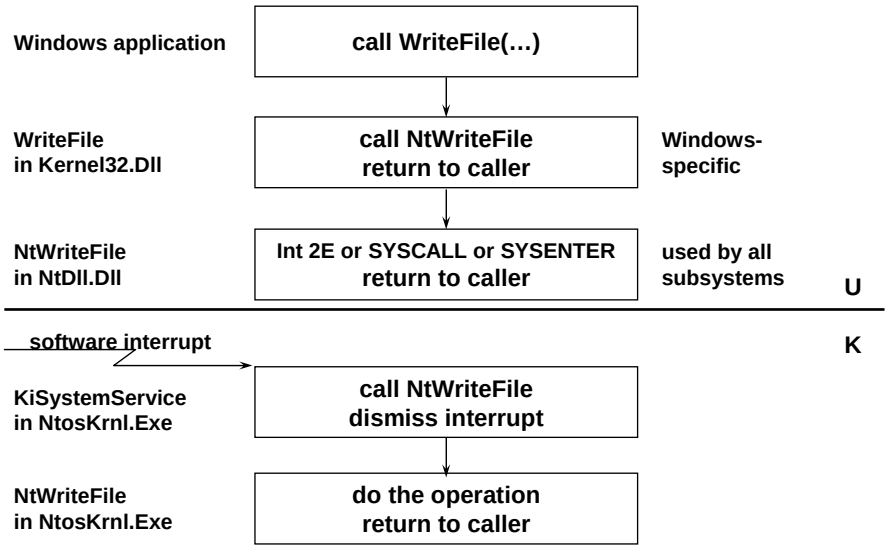
System Call Dispatching

- NTDLL.DLL provides interface for native system calls



19

Example: Invoking a Windows Kernel API



20

Invoking System Functions from User Mode

- Kernel-mode functions (“services”) are invoked from user mode via a protected mechanism
  - x86: INT 2E (as of XP, faster instructions are used where available: SYSENTER on x86, SYSCALL on AMD)
  - i.e., on a call to an OS service from user mode, the last thing that happens in user mode is this “change mode to kernel” instruction
  - Causes an exception or interrupt, handled by the system service dispatcher (KiSystemService) in kernel mode
  - Return to user mode is done by dismissing the interrupt or exception
- The desired system function is selected by the “system service number”
  - Every Windows function exported to user mode has a unique number
  - This number is stored in a register just before the “change mode” instruction (after pushing the arguments to the service)
  - This number is an index into the system service dispatch table
  - Table gives kernel-mode entry point address and argument list length for each exported function

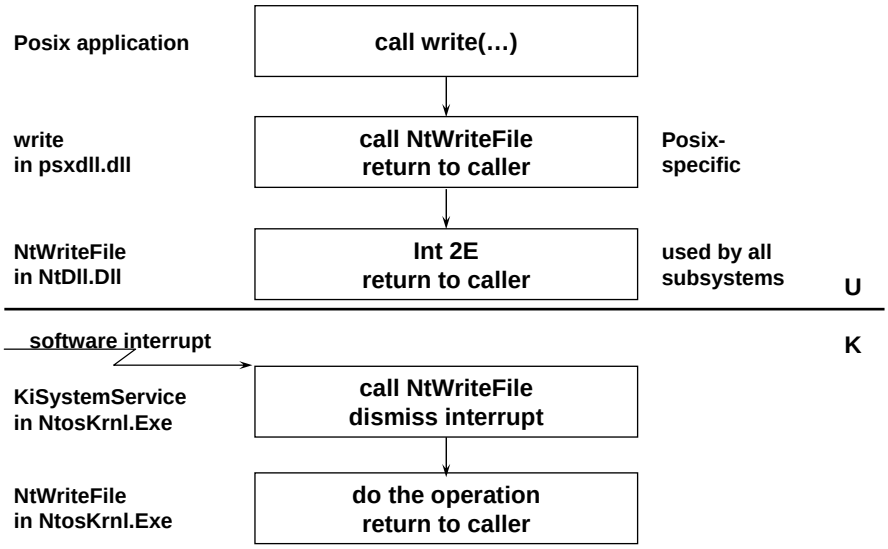
21

Invoking System Functions from User Mode

- All validity checks are done after the user to kernel transition
  - KiSystemService probes argument list, copies it to kernel-mode stack, and calls the executive or kernel routine pointed to by the table
  - Service-specific routine checks argument values, probes pointed-to buffers, etc.
  - Once past that point, everything is “trusted”
- This is safe, because:
  - The system service table is in kernel-protected memory; and
  - The kernel mode routines pointed to by the system service table are in kernel-protected memory; therefore:
  - User mode code can’t supply the code to be run in kernel mode; it can only select from among a predefined list
  - Arguments are copied to the kernel mode stack before validation; therefore:
  - Other threads in the process can’t corrupt the arguments “out from under” the service

22

Example: Invoking a Posix API



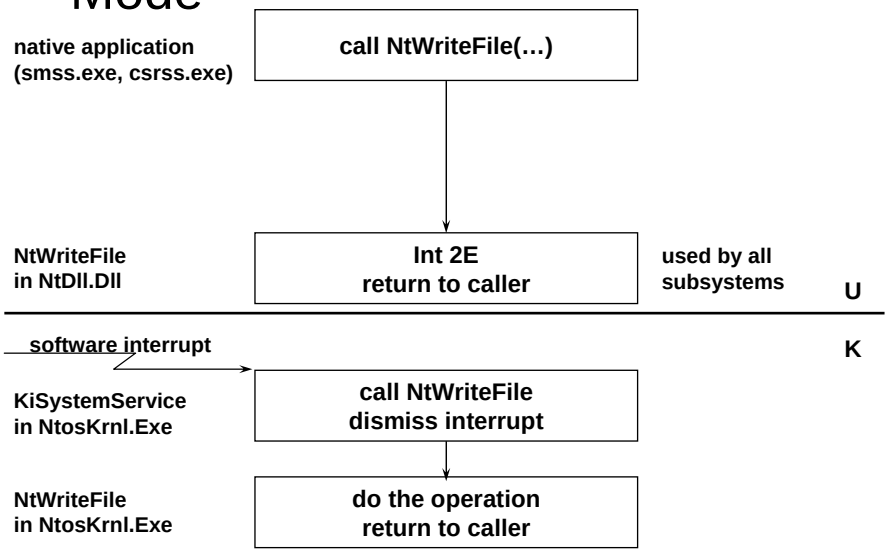
23

Ntdll.dll

- Interface to Windows system calls (285 calls starting with “Nt”-some have “Zw” aliases)
  - These user-mode routines have the same function names and arguments as the kernel mode routines they invoke
    - e.g. NtWriteFile in NtDll.Dll invokes NtWriteFile in NtosKrnl.Exe
  - Majority are not supported or documented
    - 7 are (partially) documented in the Platform SDK:
      - NtQuerySystemInformation, NtQuerySystemTime, NtQueryInformationProcess, NtQueryInformationThread, NtCreateFile, NtOpenFile, NtWaitForSingleObject
    - The DDK describes 25 of them as “Zw” routines (such as ZwReadFile)
      - These entry points call the corresponding “Nt” interface via the system call interface
        - Thus, “previous mode” is kernel mode, which means no security checks
      - Kernel mode code could also call NtReadFile directly
- Other user-mode support routines
  - Image loader (“Ldr”)
  - Debug infrastructure (“Dbg”)
  - Csrss support routines (“Csr”)
  - RTL routines (“Rtl”)
  - Tracing routines (“Etw”) [new as of Windows Server 2003]

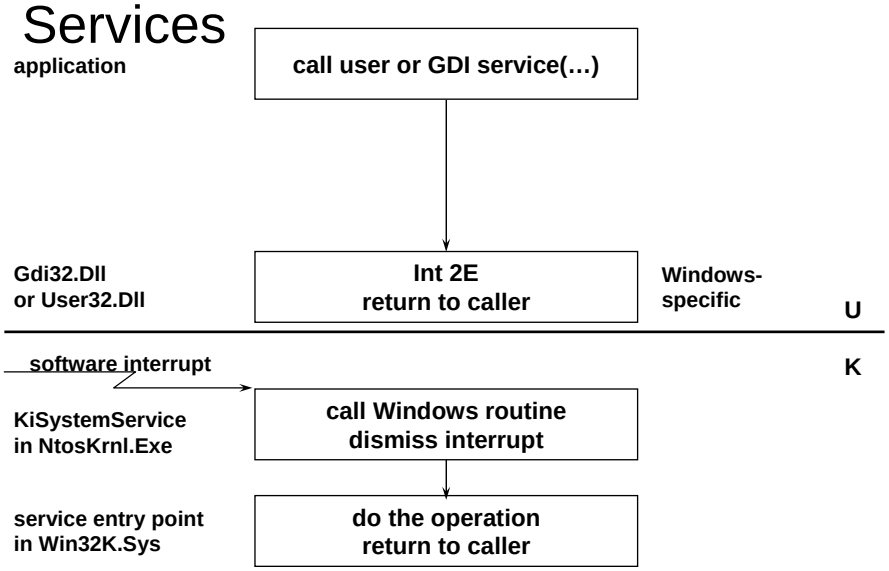
24

Calling a “Native” API from User Mode



25

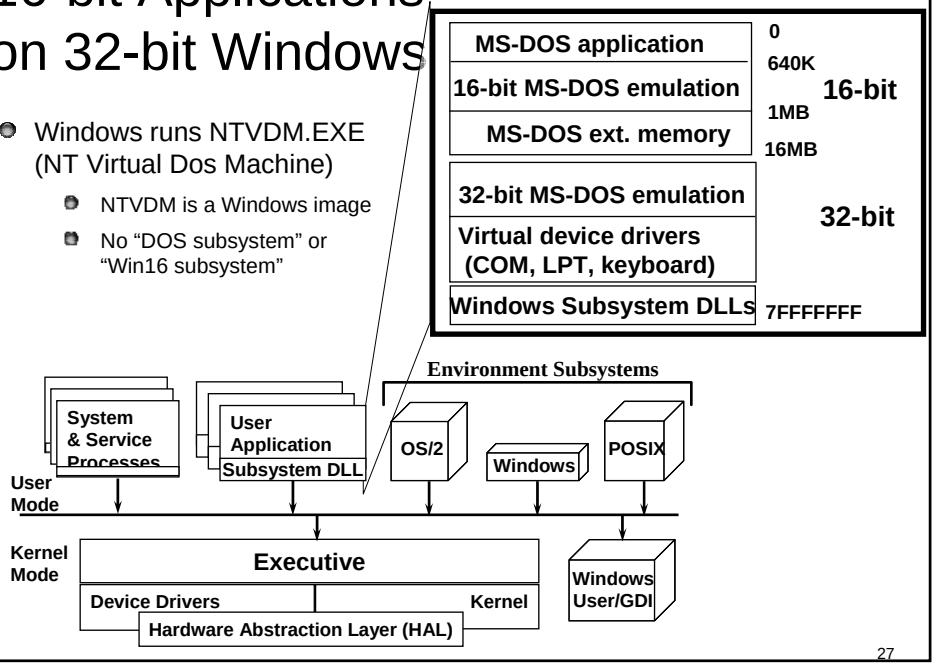
Invoking (most) User and GDI Services



26

16-bit Applications on 32-bit Windows

- Windows runs NTVDM.EXE (NT Virtual Dos Machine)
  - NTVDM is a Windows image
  - No “DOS subsystem” or “Win16 subsystem”

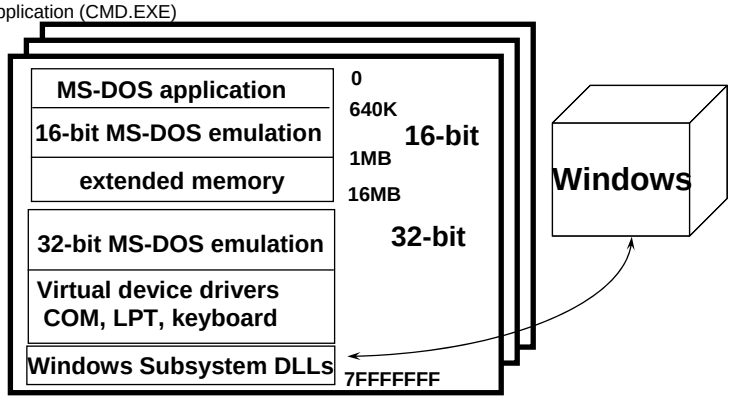


27

DOS 16-bit Applications
e.g. command.com, edit.com (NT4 had qbasic.exe)

- **Windows runs NTVDM.EXE (NT Virtual DOS Machine)**
  - See \System\CurrentControlSet\Control\WOW\cmdline
- **Each DOS app has a separate process running NTVDM**
  - DOS & Windows 16-bit drivers not supported
  - Note: Windows "command prompt" is not a "DOS box", despite icon; it's a Windows console application (CMD.EXE)

Example: three DOS apps running in three NTVDM processes

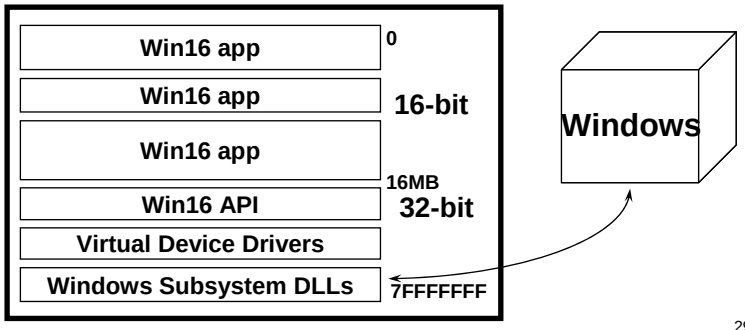


28

Windows 16-Bit Applications
e.g. sysedit.exe, winhelp.exe

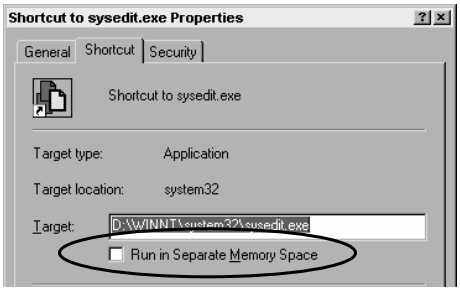
- Windows also runs NTVDM.EXE
  - See \CurrentControlSet\Control\WOW\wowcmdline
- NTVDM loads wowexec.exe
  - WOW = "Windows on Windows"
  - Win16 calls are translated to Win32 (Windows API)

example: three Win16 apps (and wowexec.exe) running in one NTVDM process



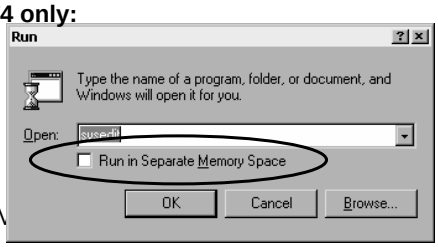
29

Windows 16-bit Applications
Multitasking Details



- By default:
  - Each Win16 app runs in a separate thread in the common NTVDM process
  - They cooperatively multitask among themselves (Win16 Yield API)...
  - …and the one (if any) that wants to run, preemptively multitasks with all other threads on Windows
  - necessary to meet serialization assumptions of some Win16 apps

NT4 only:



- Option to run Win16 apps in separate VDM
  - "Run in Separate Memory Space" = run in separate process
  - default set by \CurrentControlSet\Control\WOW\DefaultSeparateVDM
  - Win16 apps run this way preemptively multitask with all other threads, including the un-Yield'ed thread in a shared Win16 NTVDM (if any)

30

Monitoring 16-bit Applications

- To most of Windows, an NTVDM process is just another process
- Task Manager
  - "tasks" are simply the names of top-level windows - Win16 windows included
  - "processes" display identifies Win16 apps within NTVDM processes
    - by reading the NTVDM process's private memory (undocumented interface)
  - does not identify the DOS apps within each NTVDM process
- TLIST (resource kit)
  - does identify the DOS apps within each NTVDM process (by window title)
  - but for a shared Win16 NTVDM process, only shows one window title
- QuickView, exetype
  - identifies DOS, Win16, etc., application .exe's

31

Lab: 16-bit Applications

- DOS applications:
  - Run command.com and edit.com
  - look at process list in Task Manager Process tab - cannot differentiate which NTVDM.EXE is which
  - From Applications tab, right click on window -> goto process (now can map which NTVDM.EXE process is which)
- Windows 3.1 applications:
  - Run winhelp.exe twice (do not check “run in separate memory space”)
  - Run winhelp.exe once and check “run in separate memory space”
  - Bring up Task Manager Process tab - make sure “Show 16-bit Tasks” is checked on the View menu
  - Look at Task Manager Process tab and see 16-bit applications identified inside the two NTVDMs

32

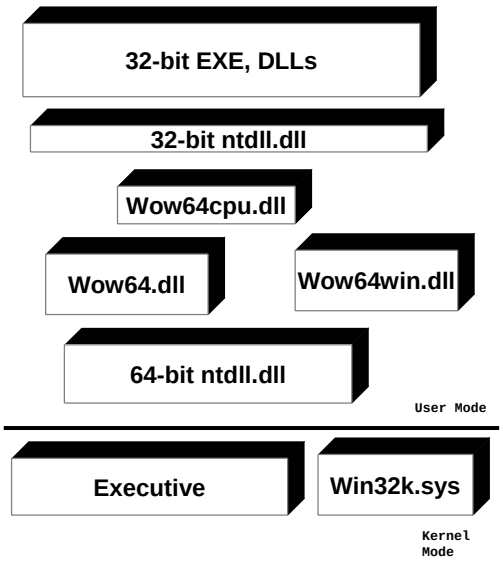
Wow64

- 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
    - Issues the native 64-bit system call
    - Returns any data from the 64-bit system call
- *IsWow64Process()* function can tell a 32-bit process if it is running under Wow64
- Performance
  - On x64, instructions executed by hardware
  - On IA64, instructions have to be emulated
    - New Intel IA-32 EL (Execution Layer) does binary translation of Itanium to x86 to speed performance
      - Downloadable now – bundled with Server 2003 SP1

33

Wow64 Components

- Wow64.dll - provides core emulation infrastructure and thunks for Ntoskrnl.exe entry-point functions; exception dispatching
- Wow64win.dll - provides thunks for Win32k.sys entry-point functions
- Wow64cpu.dll – manages thread contexts, supports mode-switch instructions



34

Wow64 Limitations

- Cannot load 32-bit DLLs in 64-bit process and vice versa
- Does not support 32-bit kernel mode device drivers
  - Drivers must be ported to 64-bits
  - Special support required to support 32-bit applications using DeviceIoControl to driver
    - Driver must convert 32-bit structures to 64-bit

Wow64 Feature Support on 64-bit Windows	Platforms	
	IA64	x64
16-bit Virtual DOS Machine (VDM) support	N/A	N/A
Physical Address Extension (PAE) APIs	N/A	Yes
GetWriteWatch() API	N/A	Yes
Scatter/Gather I/O APIs	N/A	Yes
Hardware accelerated with DirectX version 7,8 and 9	Software-Emulation Only	Yes

35

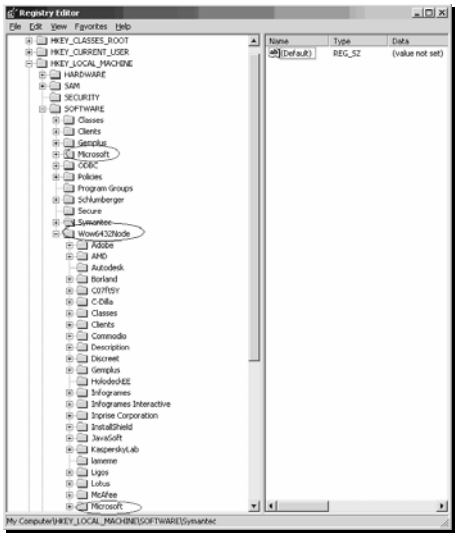
Wow64 File Locations

- Location of system files
  - 64-bit system files are in \windows\system32
  - 32-bit system files are in \windows\syswow64
  - 32-bit applications live in “\Program Files (x86)”
  - 64-bit applications live in “\Program Files”
- File access to %windir%\system32 redirected to %windir%\syswow64
- Two areas of the registry redirected (see next slide)

36

Wow64 Registry Redirection

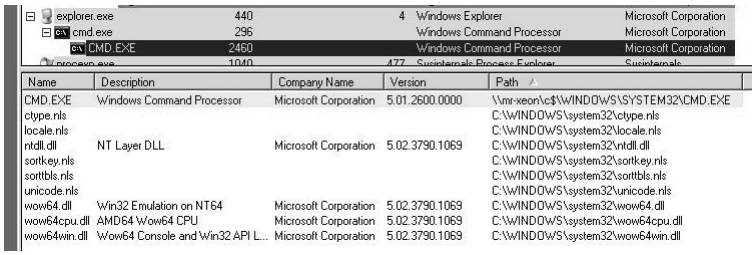
- Two registry keys have 32-bit sections:
  - HKEY_LOCAL_MACHINE\Software
  - HKEY_CLASSES_ROOT
  - Everything else is shared
- 32-bit data lives under \Wow6432Node
  - When a Wow64 process opens/creates a key, it is redirected to be under Wow6432Node



37

Example: Cmd.exe on 64-bit System

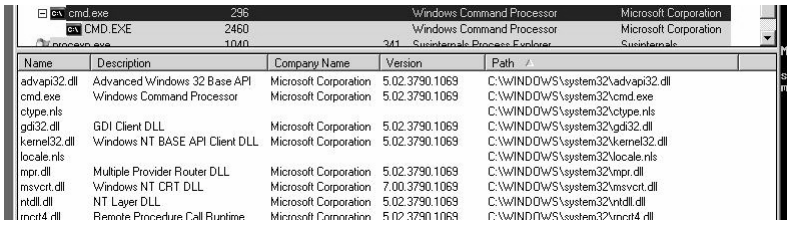
- 32-bit Cmd.exe process:



Process	PID		Description	Company
explorer.exe	440	4	Windows Explorer	Microsoft Corporation
cmd.exe	296		Windows Command Processor	Microsoft Corporation
CMD.EXE	2460		Windows Command Processor	Microsoft Corporation

Name	Description	Company Name	Version	Path
CMD.EXE	Windows Command Processor	Microsoft Corporation	5.01.2600.0000	\\mr-xeon\c$\WINDOWS\SYSTEM32\CMD.EXE
ctype.nls				C:\WINDOWS\system32\ctype.nls
locale.nls				C:\WINDOWS\system32\locale.nls
ntdll.dll	NT Layer DLL	Microsoft Corporation	5.02.3790.1069	C:\WINDOWS\system32\ntdll.dll
sortkey.nls				C:\WINDOWS\system32\sortkey.nls
sorttbls.nls				C:\WINDOWS\system32\sorttbls.nls
unicode.nls				C:\WINDOWS\system32\unicode.nls
wow64.dll	Win32 Emulation on NT64	Microsoft Corporation	5.02.3790.1069	C:\WINDOWS\system32\wow64.dll
wow64cpu.dll	AMD64 Wow64 CPU	Microsoft Corporation	5.02.3790.1069	C:\WINDOWS\system32\wow64cpu.dll
wow64win.dll	Wow64 Console and Win32 API L...	Microsoft Corporation	5.02.3790.1069	C:\WINDOWS\system32\wow64win.dll

- 64-bit Cmd.exe process:



Process	PID		Description	Company
cmd.exe	296		Windows Command Processor	Microsoft Corporation
CMD.EXE	2460		Windows Command Processor	Microsoft Corporation

Name	Description	Company Name	Version	Path
advapi32.dll	Advanced Windows 32 Base API	Microsoft Corporation	5.02.3790.1069	C:\WINDOWS\system32\advapi32.dll
cmd.exe	Windows Command Processor	Microsoft Corporation	5.02.3790.1069	C:\WINDOWS\system32\cmd.exe
ctype.nls				C:\WINDOWS\system32\ctype.nls
gdi32.dll	GDI Client DLL	Microsoft Corporation	5.02.3790.1069	C:\WINDOWS\system32\gdi32.dll
kernel32.dll	Windows NT BASE API Client DLL	Microsoft Corporation	5.02.3790.1069	C:\WINDOWS\system32\kernel32.dll
locale.nls				C:\WINDOWS\system32\locale.nls
mpr.dll	Multiple Provider Router DLL	Microsoft Corporation	5.02.3790.1069	C:\WINDOWS\system32\mpr.dll
msvcrt.dll	Windows NT CRT DLL	Microsoft Corporation	7.00.3790.1069	C:\WINDOWS\system32\msvcrt.dll
ntdll.dll	NT Layer DLL	Microsoft Corporation	5.02.3790.1069	C:\WINDOWS\system32\ntdll.dll
rpcrt4.dll	Remote Procedure Call Runtime	Microsoft Corporation	5.02.3790.1069	C:\WINDOWS\system32\rpcrt4.dll

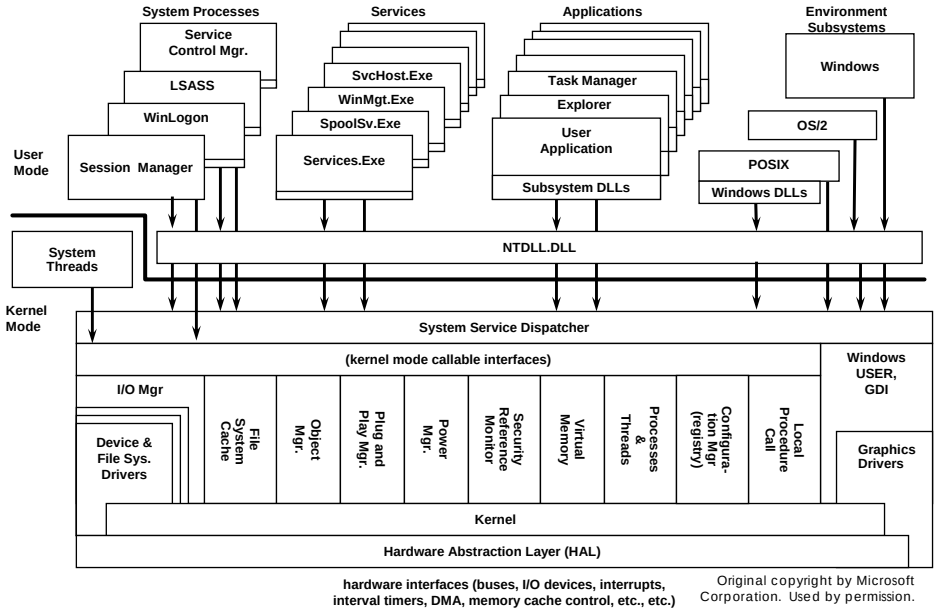
38

Four Contexts for Executing Code

- Full process and thread context:
  - User applications
  - Windows Services
  - Environment subsystem processes
  - System startup processes
- Have thread context but no “real” process:
  - Threads in “System” process
- Routines called by other threads/processes:
  - Subsystem DLLs
  - Executive system services (NtReadFile, etc.)
  - GDI32 and User32 APIs implemented in Win32K.Sys (and graphics drivers)
- No process or thread context
  - (“arbitrary thread context”)
  - Interrupt dispatching
  - Device drivers

39

Windows Architecture



40

Further Reading

- Mark E. Russinovich and David A. Solomon, Microsoft Windows Internals, 4th Edition, Microsoft Press, 2004.
- Chapter 2 - System Architecture
  - Environment Subsystems and Subsystem DLLs (from pp. 53)
  - NTDLL.DLL (from pp. 63)
  - Executive (from pp. 65)

41