

# Unit 3: Processes and Threads

## 3.1. Windows 2000 Process and System Activity

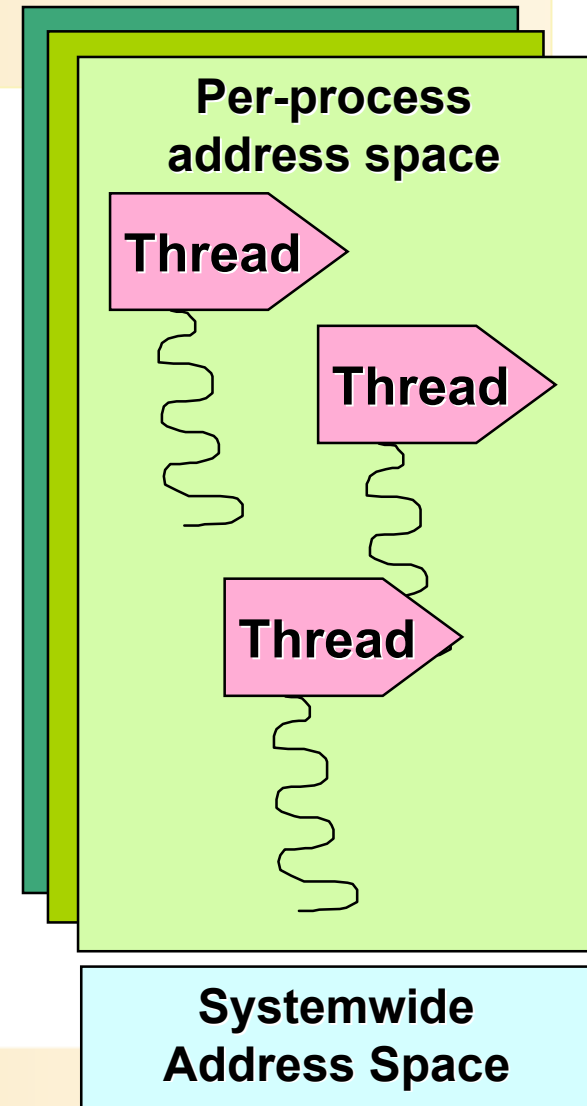
# Windows 2000 Process and System Activity

## Agenda:

- Understanding Process and Thread Activity
- Understanding CPU Time Accounting
- Understanding System Processes
- Process and System Crashes

# Processes and Threads

- What is a process?
  - Represents an instance of a running program
  - Each process has a private memory address space
- What is a thread?
  - An execution context within a process
  - All threads in a process share the same per-process address space
- Every process starts with one thread
  - Running the program's "main" function
  - Can create other threads in the same process
  - Can create additional processes



# Windows 2000 Job Object

- New kernel object to define a group of related processes
  - CreateJobObject()/OpenJobObject()
- Can specify job-wide attributes, quotas, and security limitations
  - Quotas: Total and current CPU time, total and active processes, per-process and per-job CPU time, min and max working set
  - Attributes: CPU affinity, priority class, scheduling class
  - Security limits: No administrators token, only restricted token, only specific token, filter token, no accessing windows outside the job, no reading/writing the clipboard

# Process Information Tools

Many overlapping tools!

Several show one unique  
item that no other tool  
shows

- Tools in Windows 2000:
  - Task Manager, Performance Monitor (perfmon.exe)
  - In \support\debug\
- Tools in Resource Kit:
  - \reskit\qslice.exe - can show process-relative thread activity (GUI)
  - \reskit\pviewer - process and thread details (GUI)
  - \reskit\pview - processes and threads and security details (GUI)
  - \reskit\tlist - shows process tree and thread details (character cell)
  - \reskit\pulist - lists processes and usernames (character cell)
  - \reskit\pstat - process/threads and driver addresses (character cell)
  - \reskit\pmon - process list (character cell)
  - \reskit\apimon.exe - system call and page fault monitoring (GUI)
- Tools from [www.sysinternals.com](http://www.sysinternals.com)
  - ntpmon - shows process/thread create/deletes (and context switches on MP systems only)
  - Handle - displays open handles and loaded DLLs



# Task Manager

- To start: Ctrl+Shift+Esc; or Ctrl+Alt+Del; or right click on empty area of task bar
- Overlaps with other process display utilities
  - Except Win16 process info, only visible here (On Processes tab, click on Options->Show 16-bit tasks)
- Applications tab: List of top level visible windows
  - Windows are owned by threads (right-click on a window and select “go to process”)

- Processes tab: List of processes
- Can configure with View-> Select columns
- Click on column heading to sort by that column
- Right-click on a process name to change priority, end process tree (new in Windows 2000), or (on MP) CPU assignments
- Performance tab: Subset of Windows NT performance counters



# Process Viewer

- Pviewer.exe in Resource Kit, pview.exe in Platform SDK
- Shows start address of each thread
  - Needed to analyze system threads
- Can display remote process list
  - But cannot kill remote processes
    - Use rkill in ResKit!

The screenshot shows the Process Viewer interface for a remote computer named \\mach5. The main window displays a table of processes and a table of threads. The process table shows the following data:

Process	Processor Time	Privileged	User
nddeagnt (0x27)	0:00:00.080	63%	37%
POWERPNT (0xab)	0:14:44.902	11%	89%
PSP (0x9f)	0:00:03.545	75%	25%
PVIEWER (0xb3)	0:00:00.520	56%	44%

The thread table shows the following data:

Thread(s)	Processor Time	Privileged	User
0	0:10:42.183	11%	89%
1	0:00:00.050	60%	40%
2	0:00:00.160	75%	25%
3	0:00:00.040	75%	25%
4	0:00:03.855	44%	56%
5	0:00:00.000	0%	0%

Additional information shown includes Process Memory Used (Working Set: 30276 KB, Heap Usage: 21156 KB), Thread Priority (Normal selected), and Thread Information (User PC Value: 0x77f67e67, Start Address: 0x77f052e0, Context Switches: 393, Dynamic Priority: 14).

Screen snapshot from:  
Programs | Resource Kit |  
Diagnostics | Process Viewer

# Looking at the Process Hierarchy with TLIST -T

- Understanding the parent of a process helps identify what it is and where it came from
- tlist -t shows the tree
  - If parent not alive, left justifies process
    - I.e., cannot see creator if it is gone
  - For example, explorer.exe's parent is dead (it is actually started by userinit.exe, which then exits)
- Windows 2000
  - Perfmon can show parent process id
  - Task Manager has a "kill process tree"

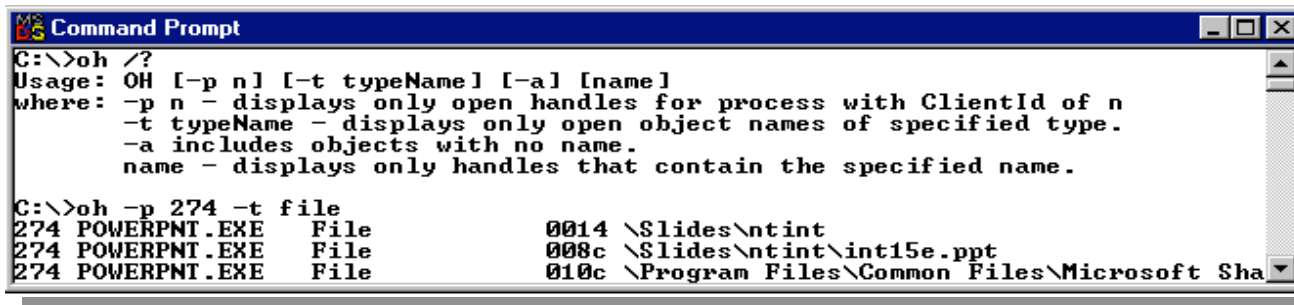
```
Command Prompt
D:\A>tlist /t
System Process (0)
System (2)
  smss.exe (20)
    csrss.exe (24)
      winlogon.exe (34)
        services.exe (40)
          SPOOLSS.EXE (65)
            netdde.exe (72)
              AMGRSRVC.EXE (89)
                clipsrv.exe (93)
                  SDSRV.EXE (63)
                    rpcss.exe (108)
                      TCPSVCS.EXE (112)
                        tapisrv.exe (115)
                          wfxsvc.exe (125)
                            RASMAN.EXE (148)
                              lsass.exe (43)
                                nddeagnt.exe (133)
                                  explorer.exe (142) Program Manager
                                    systray.exe (160)
                                      wfxsnt40.exe (171)
                                        POWERPNT.EXE (188) Microsoft PowerPoint - [int13d.ppt]
                                          notepad.exe (57) int13dqs.txt - Notepad
                                            PVIEW.EXE (203) Process Viewer
                                              PSP.EXE (191) Paint Shop Pro
                                                cmd.exe (215) Command Prompt - tlist /t
                                                  TLIST.EXE (200)

D:\A>
```



# Looking at open Handles

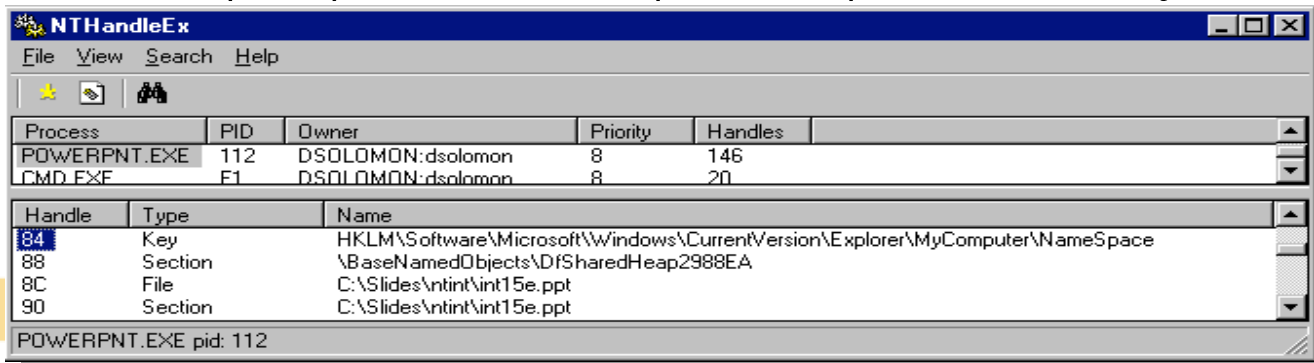
- Handle leaks can show up as system memory leaks!
- Task Manager can show total # handles by process
- Resource Kit “oh” tool (first time run will set an Windows 2000 Global Flag - see gflags.exe in ResKit; reboot required)



```
Command Prompt
C:\>oh /?
Usage: OH [-p n] [-t typeName] [-a] [name]
where: -p n - displays only open handles for process with ClientId of n
       -t typeName - displays only open object names of specified type.
       -a includes objects with no name.
       name - displays only handles that contain the specified name.

C:\>oh -p 274 -t file
274 POWERPNT.EXE File          0014 \Slides\ntint
274 POWERPNT.EXE File          008c \Slides\ntint\int15e.ppt
274 POWERPNT.EXE File          010c \Program Files\Common Files\Microsoft Sha
```

- handleex (GUI) or nthandle (console) from [www.sysinternals.com](http://www.sysinternals.com)



Process	PID	Owner	Priority	Handles
POWERPNT.EXE	112	DSOLOMON:dsolomon	8	146
CMD.EXE	F1	DSOLOMON:dsolomon	8	20

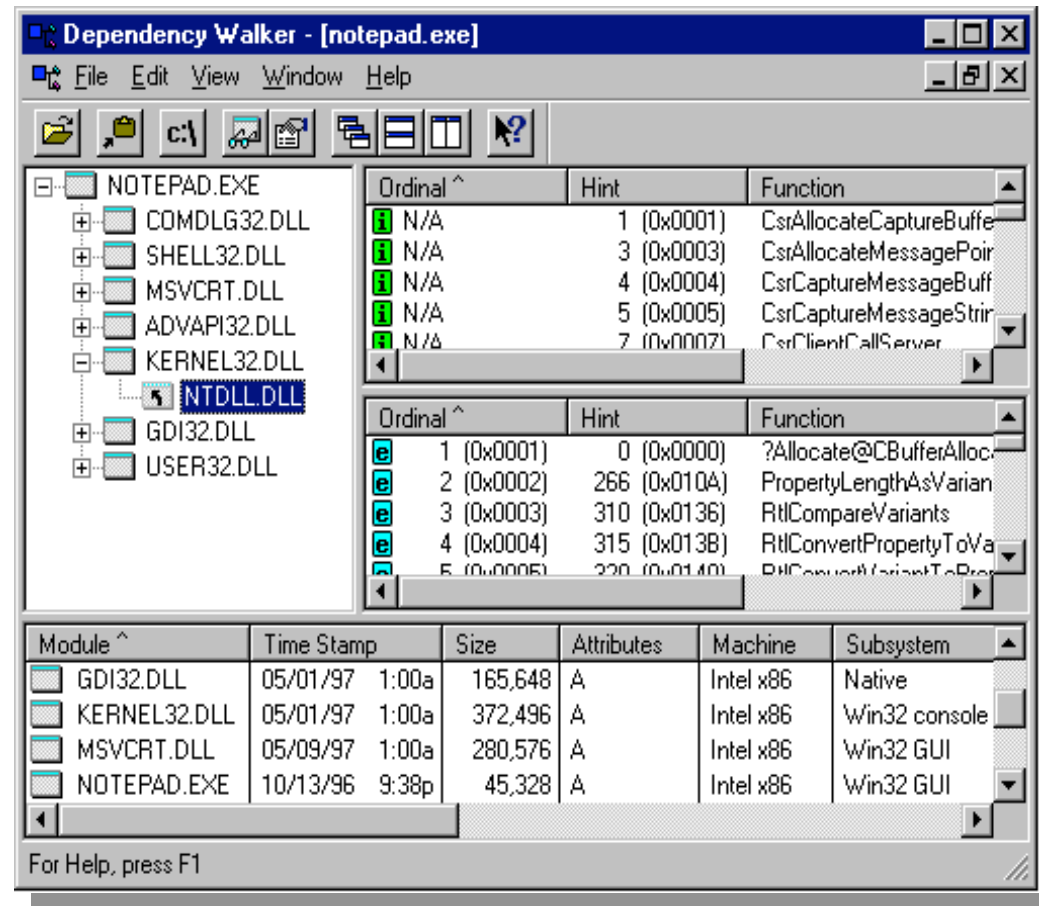
Handle	Type	Name
84	Key	HKLM\Software\Microsoft\Windows\CurrentVersion\Explorer\MyComputer\NameSpace
88	Section	\BaseNamedObjects\DfSharedHeap2988EA
8C	File	C:\Slides\ntint\int15e.ppt
90	Section	C:\Slides\ntint\int15e.ppt

POWERPNT.EXE pid: 112

# DLL Usage

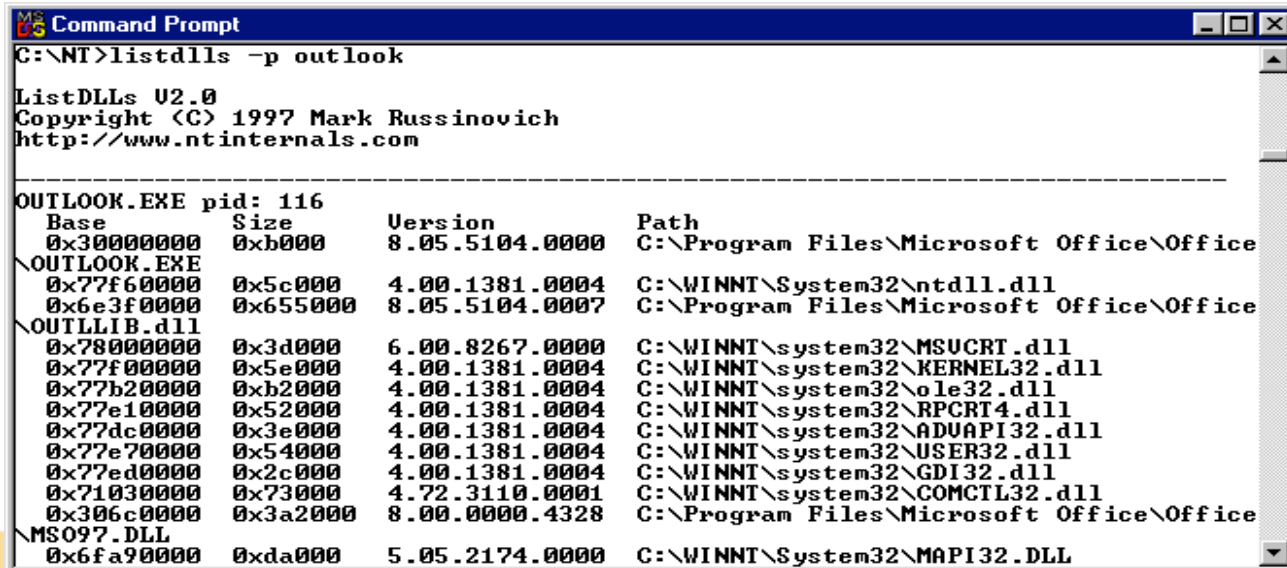
## Static references

- Depends.exe in Resource Kit
- Displays static linkage from EXE to DLLs



# DLL Usage Actual files

- To diagnose DLL conflicts, you need to know which DLLs were loaded and from where
- `tlist <processname>` or `tlist <processid>` lists the DLLs, but not the path
- `listdlls` from [www.sysinternals.com](http://www.sysinternals.com) lists full path



```
MS Command Prompt
C:\NT>listdlls -p outlook

ListDLLs U2.0
Copyright (C) 1997 Mark Russinovich
http://www.ntinternals.com

-----
OUTLOOK.EXE pid: 116
Base      Size      Version   Path
0x30000000 0xb000    8.05.5104.0000 C:\Program Files\Microsoft Office\Office
\OUTLOOK.EXE
0x77f60000 0x5c000   4.00.1381.0004 C:\WINNT\System32\ntdll.dll
0x6e3f0000 0x655000  8.05.5104.0007 C:\Program Files\Microsoft Office\Office
\OUTLLIB.dll
0x78000000 0x3d000   6.00.8267.0000 C:\WINNT\system32\MSUCRT.dll
0x77f00000 0x5e000   4.00.1381.0004 C:\WINNT\system32\KERNEL32.dll
0x77b20000 0xb2000   4.00.1381.0004 C:\WINNT\system32\ole32.dll
0x77e10000 0x52000   4.00.1381.0004 C:\WINNT\system32\RPCRT4.dll
0x77dc0000 0x3e000   4.00.1381.0004 C:\WINNT\system32\ADVAPI32.dll
0x77e70000 0x54000   4.00.1381.0004 C:\WINNT\system32\USER32.dll
0x77ed0000 0x2c000   4.00.1381.0004 C:\WINNT\system32\GDI32.dll
0x71030000 0x73000   4.72.3110.0001 C:\WINNT\system32\COMCTL32.dll
0x306c0000 0x3a2000  8.00.0000.4328 C:\Program Files\Microsoft Office\Office
\MSO97.DLL
0x6fa90000 0xda000   5.05.2174.0000 C:\WINNT\System32\MAPI32.DLL
```

# Agenda

- Understanding Process and Thread Activity
- Understanding CPU Time Accounting
- Understanding System Processes
- Process and System Crashes

# Kernel Mode Versus User Mode

- A processor state
  - Controls access to memory
  - Each memory page is tagged to show the required mode for reading and for writing
    - Protects the system from the users
    - Protects the user (process) from themselves
    - System is not protected from system
  - Code regions are tagged “no write in any mode”
  - Controls ability to execute privileged instructions
  - A Windows NT abstraction
    - Intel: Ring 0, Ring 3
- Associated with threads
  - Threads can change from user to kernel and back
  - Part of saved context, along with registers, etc.
  - Does not affect scheduling
- PerfMon counters:
  - “Privileged Time” and “User Time”
  - 4 levels of granularity: thread, process, processor, system

# Getting Into Kernel Mode

Code is run in kernel mode for one of three reasons:

## 1. Requests from user mode

- Via the system service dispatch mechanism
- Kernel-mode code runs in the context of the requesting thread

## 2. Interrupts from external devices

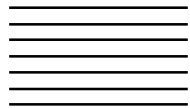
- Windows 2000 interrupt dispatcher invokes the interrupt service routine
- ISR runs in the context of the interrupted thread (so-called “arbitrary thread context”)
- ISR often requests the execution of a “DPC routine,” which also runs in kernel mode
- Time not charged to interrupted thread

## 3. Dedicated kernel-mode system threads

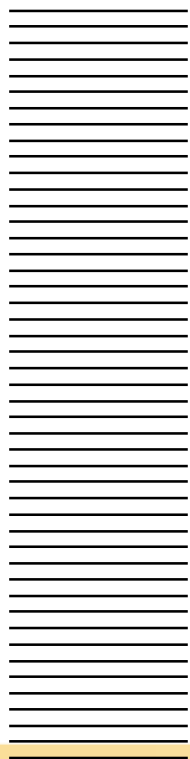
- Some threads in the system stay in kernel mode at all times (mostly in the “System” process)
- Scheduled, preempted, etc., like any other threads

# Interrupt Dispatching

user or  
kernel mode  
code



interrupt !



kernel mode

Interrupt dispatch routine

Disable interrupts

Record machine state (trap  
frame) to allow resume

Mask equal- and lower-IRQL  
interrupts

Find and call appropriate  
ISR

Dismiss interrupt

Restore machine state  
(including mode and  
enabled interrupts)

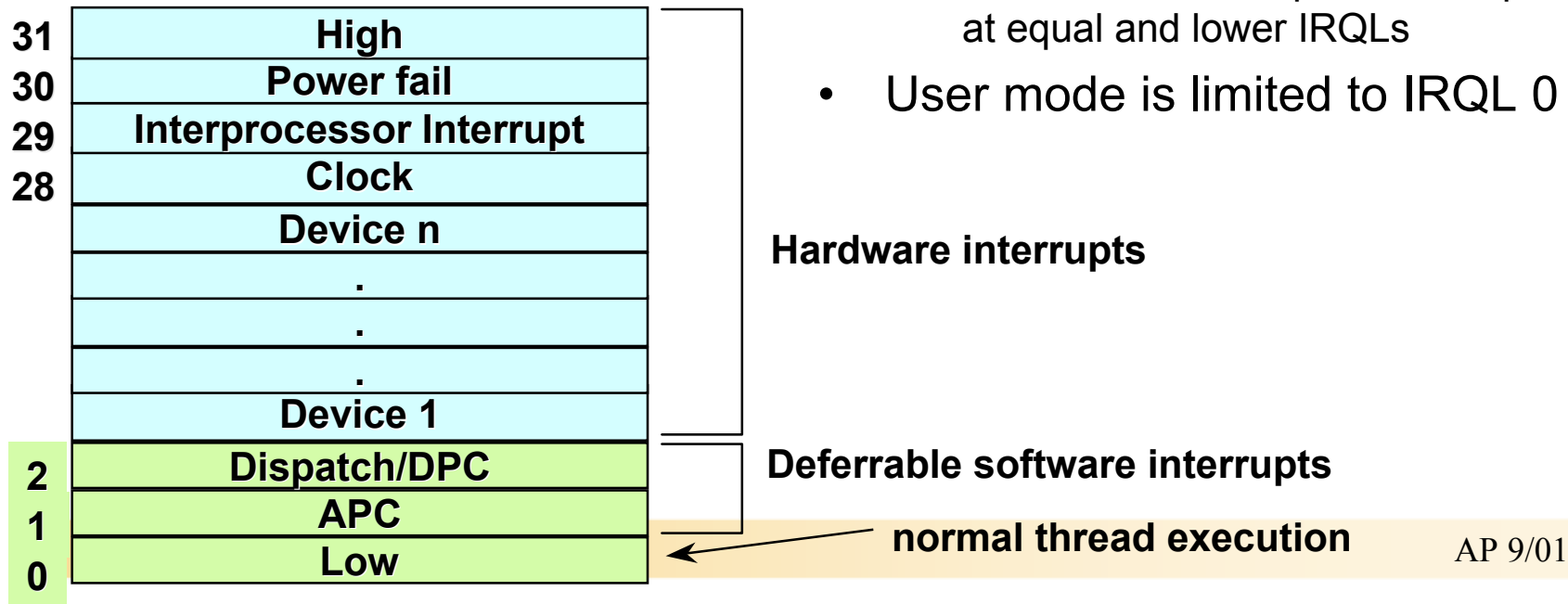
Note, no thread or  
process context  
switch!

Interrupt service routine

Tell the device to stop  
interrupting  
Interrogate device state,  
start next operation on  
device, etc.  
Request a DPC  
Return to caller

# Interrupt Precedence via IRQLs

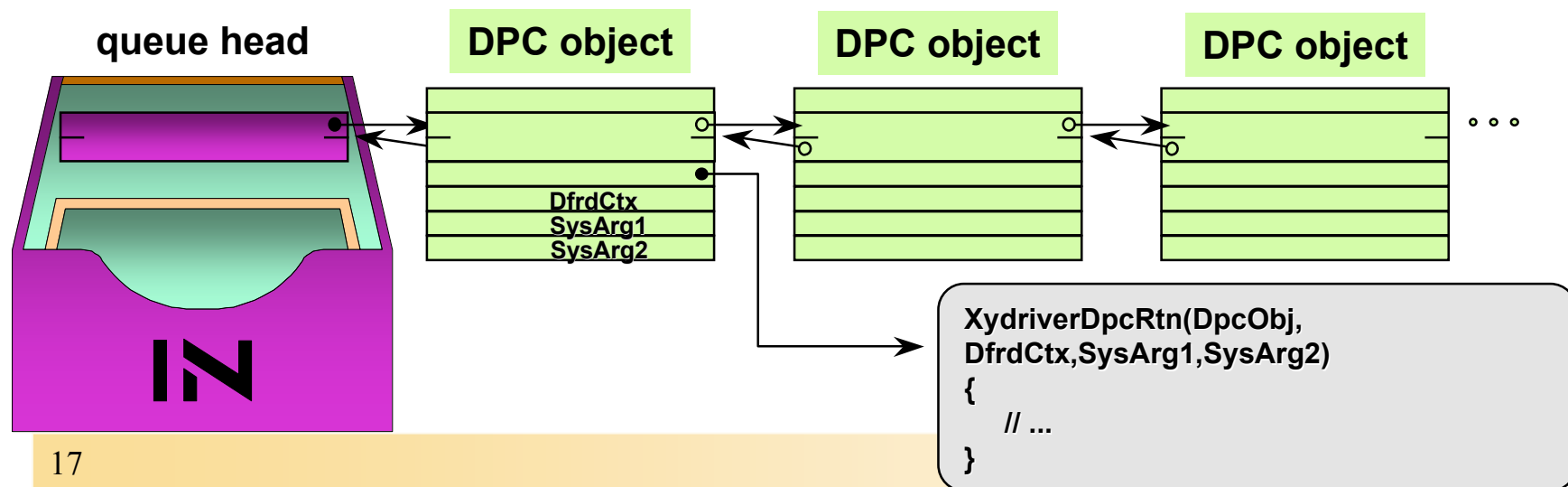
- IRQL = Interrupt Request Level
  - The “precedence” of the interrupt with respect to other interrupts
  - Different interrupt sources have different IRQLs
  - Not the same as IRQ
- IRQL is also a state of the processor
- Servicing an interrupt raises processor IRQL to that interrupt’s IRQL
  - This masks subsequent interrupts at equal and lower IRQLs
- User mode is limited to IRQL 0





# Deferred Procedure Calls (DPCs)

- A list of “work requests”
  - One queue per processor (but processors can run each others’ DPCs)
  - Implicitly ordered by time of request (FIFO)
- Used to defer processing from higher (device) interrupt level to a lower (dispatch) level
  - Used heavily for driver “after interrupt” functions
  - Used for quantum end and timer expiration



# Accounting for Kernel-Mode Time

“Processor Time” =

total busy time of processor  
(equal to elapsed real time - idle  
time)

“Processor Time” =

“User Time” + “Privileged Time”

“Privileged Time” =

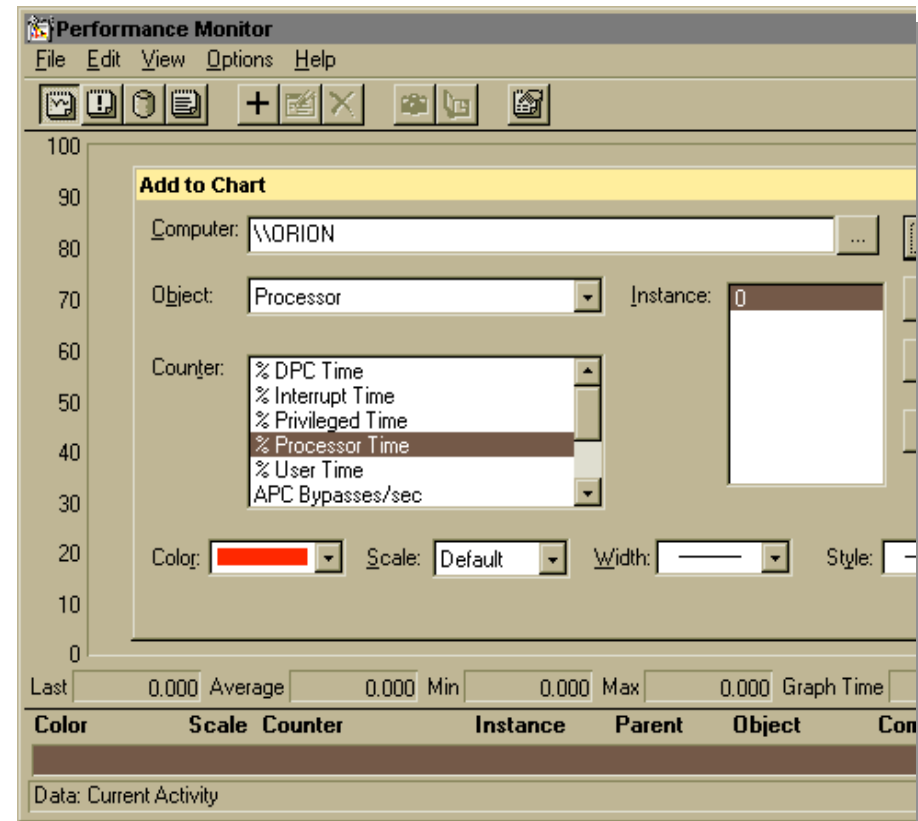
time spent in kernel mode

“Privileged Time” includes:

- Interrupt Time
- DPC Time

Again note:

Interrupts and DPCs are not  
charged to any process or thread



Screen snapshot from: Programs |  
Administrative Tools | Performance Monitor  
click on “+” button, or select Edit | Add to chart...

# Agenda

- Understanding Process and Thread Activity
- Understanding CPU Time Accounting
- Understanding System Processes
- Process and System Crashes

# System Process Tree

(Idle)	Process id 0 Part of the loaded system image Home for idle thread(s) (not a real process nor real threads)
(System)	Process id 2 (8 in Win2000) Part of the loaded system image Home for kernel-defined threads (not a real process) Thread 0 (routine name Phase1Initialization) launches the first “real” process, smss.exe (and then becomes the zero page thread)

**Note: These two processes have different names in different utilities (they are not running a real .EXE)**

# System Threads

- Subroutines in OS and some drivers that need to run as real threads
  - E.g., need to run concurrently with other system activity, wait on timers, perform background “housekeeping” work
  - For details, see DDK documentation on PsCreateSystemThread()
- What process do they appear in?
  - Windows NT 4.0: The “System” process (PID 2)
  - Windows 2000: windowing system threads appear in “csrss.exe” (Win32 subsystem process) - rest in “System” (PID 8)

# Examples Of System Threads

- Core operating system (NTOSKRNL.EXE)
  - Modified Page Writer
  - Balance Set Manager
  - Swapper (kernel stack, working sets)
  - Cache manager lazy writer
  - Zero page thread (thread 0, priority 0)
  - General pool of worker threads (ExQueueWorkItem())
- File server (SRV.SYS)
- Floppy driver (FLOPPY.SYS)

# Identifying System Threads

- To really understand what's going on, must find which driver a thread "belongs to":
  1. Use PerfMon to monitor individual thread activity
  2. Get relative thread # and look up "Start address" (address of thread function) in Pviewer
  3. Run `\ntreskit\pstat` to find which driver thread is in (look for what driver starts near the thread start address – may have to compute ending address of driver)

# Identifying System Threads (contd.)

- If thread is in NTOSKRNL.EXE, must find name of subroutine:
  1. Dump NTOSKRNL.DBG (or NTKRNLMP.DBG) with Kernel Debugger by opening any crash dump file and typing “x \*”
    - note: Values vary for each service pack
  2. Look up address
- For details, see Chapter 2 of D.Solomon „Inside Windows NT“, MS Press, 1998.
  - Available as the free sample chapter on [mspress.microsoft.com](http://mspress.microsoft.com)

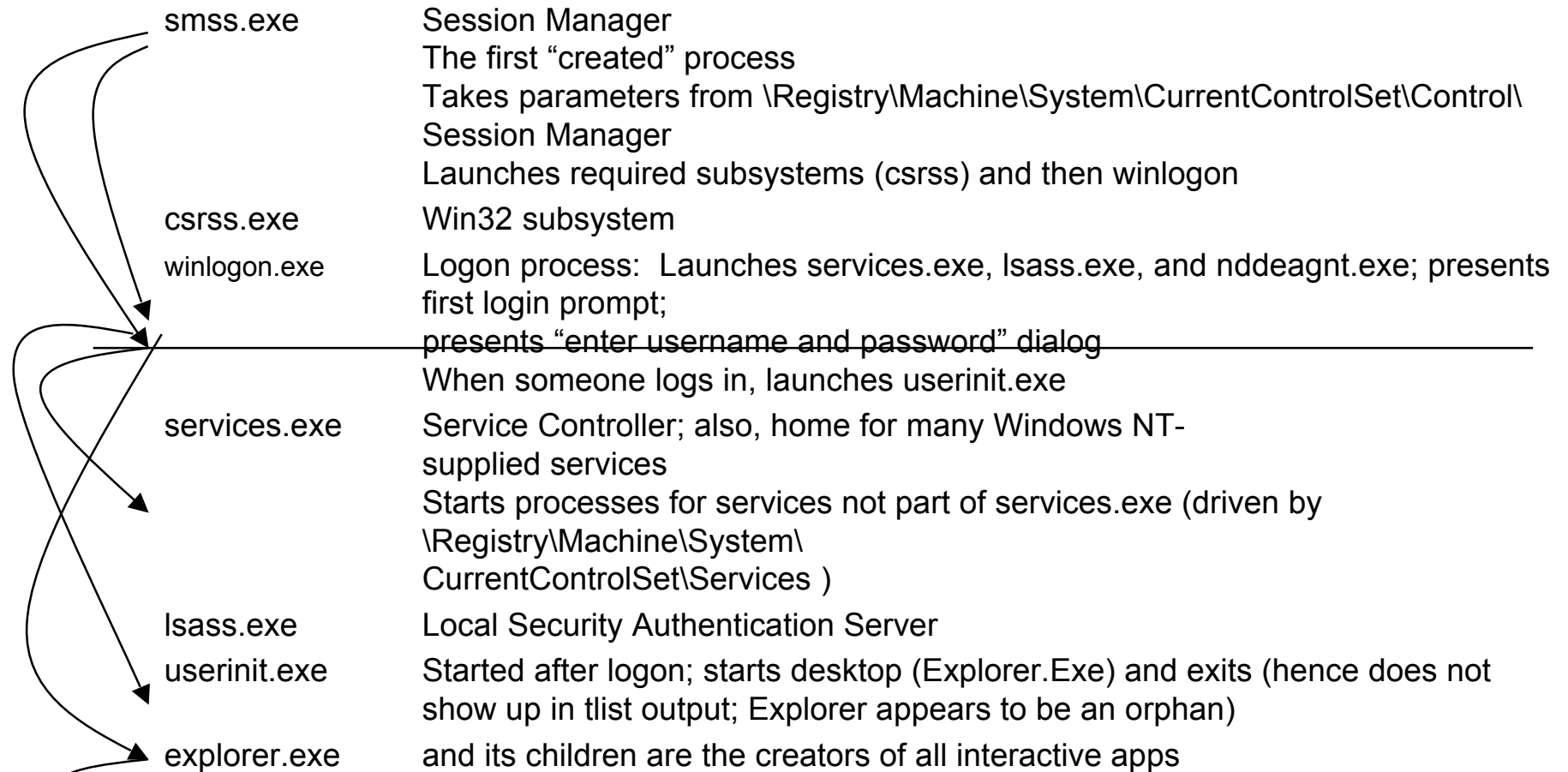


# Threads in NTOSKRNL.EXE

Observed on Intel Windows NT® Workstation 4.0

<b>Routine Name</b>	<b>Priority</b>	<b>Notes</b>
<b>Phase1Initialization</b>	<b>0</b>	<b>First thread in life of system; becomes zero page thread</b>
<b>ExpWorkerThread</b>	<b>9-16</b>	<b>Pool of worker threads</b>
<b>MiDereferenceSegmentThread</b>	<b>18</b>	<b>Dereferences segments; also expands paging file</b>
<b>MiModifiedPageWriter</b>	<b>17</b>	<b>Writes modified pages to paging file</b>
<b>KeBalanceSetManager</b>	<b>16</b>	<b>Reclaims memory from processes, with aid of . . .</b>
<b>KeSwapProcessOrStack</b>	<b>23</b>	<b>Scheduled by balance set manager</b>
<b>FsRtlWorkerThread</b>	<b>16, 17</b>	<b>Dedicated worker threads for FSDs</b>
<b>SepRmCommandServerThread</b>	<b>15</b>	<b>Security Reference Monitor Command Server</b>
<b>MiMappedPageWriter</b>	<b>17</b>	<b>Writes modified pages to mapped files</b>

# System Process Tree



# Win32 Subsystem Process (`csrss.exe`)

- Contains user-mode part of windowing system
  - Majority is in `WIN32K.SYS` (kernel-mode driver)
- Rarely invoked - only at:
  - Process creation and deletion
  - Thread creation and deletion
  - Get temporary file name
  - Drive letters
  - Security checks for file system redirector
  - Window management for console (character cell) applications
  - Some support for 16-bit DOS support (`NTVDM.EXE`)

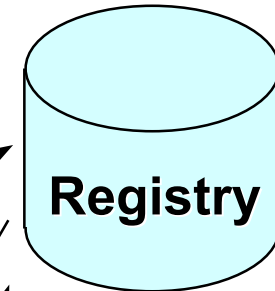
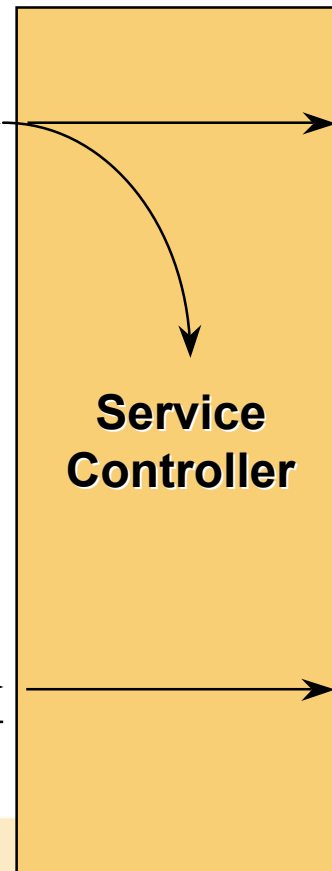
# Service Processes

- Install time

- Setup application tells Service Controller about the service



CreateService



System boot / initialization

- SCM reads registry, starts services as directed

- Management / maintenance

- Control panel can start and stop services and change startup parameters



# Mapping Service Processes to Service Names

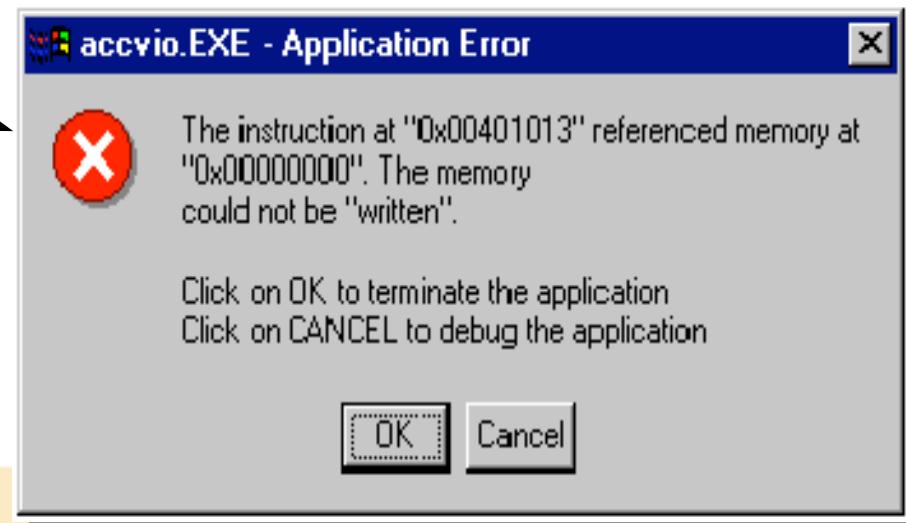
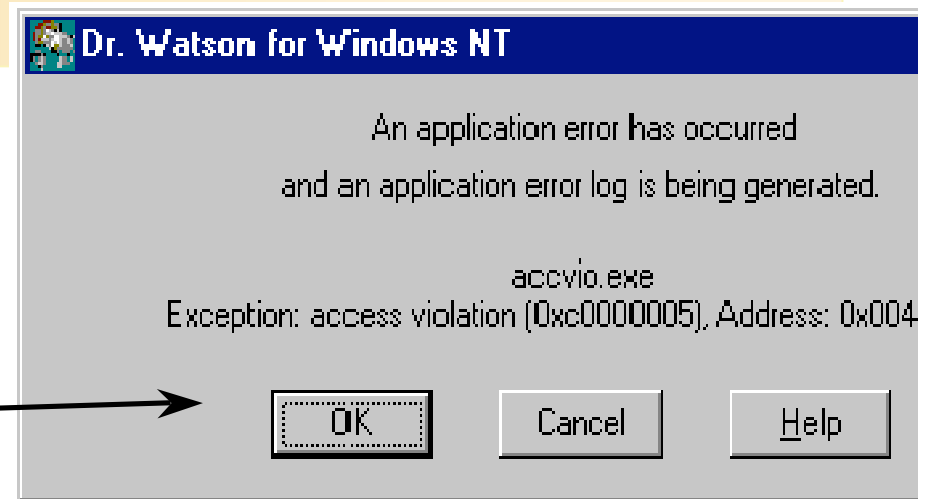
- Not always a 1 to 1 mapping
  - Some service processes contain more than one service
  - E.g., Event Log service is in lsass.exe, Workstation and Server are in services.exe
- Look up .EXE name or service name in registry:
  - HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services
  - One key per installed service
- Mandatory information kept on each service:
  - Type of service (Win32, Driver...)
  - Imagename of service .EXE
  - Start type (automatic, manual, or disabled)
- Optional information:
  - Display Name, Dependencies, Account and password to run under

# Agenda

- Understanding Process and Thread Activity
- Understanding CPU Time Accounting
- Understanding System Processes
- Process and System Crashes

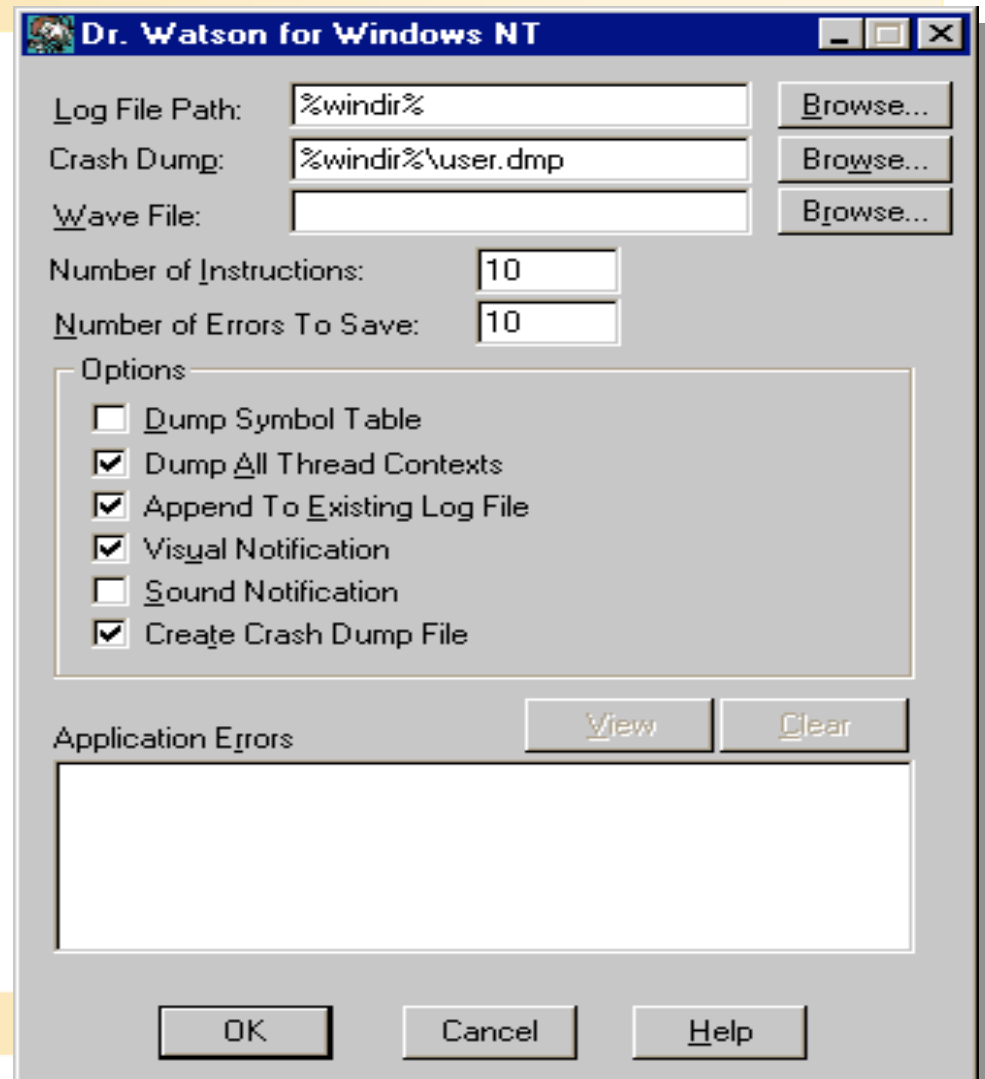
# Process Crashes

- Registry defines behavior for unhandled exceptions
  - HKLM\Software\Microsoft\Windows NT\CurrentVersion\AeDebug
  - Debugger=filespec of debugger to run on app crash
  - Auto 1=run debugger immediately  
0=ask user first
- Default on retail Windows 2000 system is **Auto=1**;  
Debugger=DRWTSN32.EXE
- Default with VC++ is **Auto=0**, Debugger=MSDEV.EXE



# Dr. Watson

- Default is to run this automatically
- Can customize by running DRWTSN32.EXE
- Look for its log files (“drwtsn32.log”)





# Why does Windows 2000 crash

- Unhandled exception in device driver or kernel function
  - Memory access violation, etc.
- Call to a kernel routine results in reschedule when interrupt request level (IRQL) is DPC/dispatch or higher
- Page fault on memory backed by paging file when IRQL is DPC/dispatch or higher
  - Memory manager would have to wait for I/O operation
  - Waits cannot occur at DPC/dispatch IRQL level or higher (would require re-schedule)
- Device driver or OS function crashes system
  - Via call to KeBugCheck() – reaction on corruption of system integrity
- Hardware error occurs (NMI, machine check)

# System Crashes

- Few outside of Microsoft perform true Windows 2000 crash dump analysis
- Often the victim is on the stack, not the culprit
  - See article on blue screen on [www.sysinternals.com](http://www.sysinternals.com)
- But simply looking up crash code (bugcheck code) may be enough
  1. Look up explanation in Windows 2000 messages help file (`\ntreskit\NTMSG.S.HLP`)
    - Click on “Kernel”, then “STOP”
  2. Do a search in TechNet for hex stop code

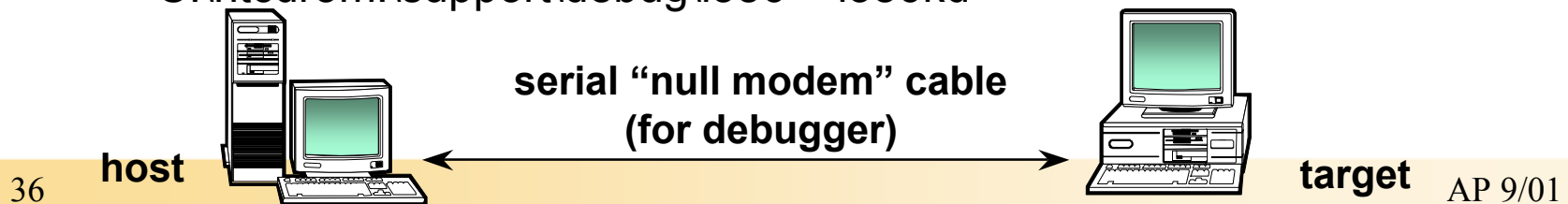
# Crash Debugging Tools

- Kernel Debuggers
  - (I386KD.EXE, WinDBG.EXE, KD.EXE)
  - Available with platform SDK and Windows 2000 DDK
- Open a crash dump for interactive analysis
  - Also can examine a live Windows 2000 system, but requires 2 computers
  - dumpchk: Checks validity of dump file
  - Available in \support\tools of Windows 2000 CD-ROM
- Limited documentation available:
  - See Resource Kit/Windows NT Workstation Resource Guide (“Windows NT Debugger” chapter) for limited documentation

# Windows 2000 Kernel Debugger

- To open a dump, minimally need one symbol table file:
  - NTOSKRNL.DBG (or NTKRNLMP.DBG if a MP system)  
Must match rev of Windows 2000 (service pack level)
- Two modes of operation:
  - Open a crash dump file:

```
e:\> set _NT_SYMBOL_PATH=e:\support\debug\i386\symbols
e:\> i386kd -Z dumpfilename
```
  - Connect to a live system via null modem cable
    - mMust boot target system with /DEBUG
  - C:\> set \_NT\_SYMBOL\_PATH= e:\i386\symbols  
C:\> set \_NT\_DEBUG\_PORT=COMn default COM1  
C:\> set \_NT\_DEBUG\_BAUD\_RATE=nnnnn default 19200  
C:\ntcdrom:\support\debug\i386 > i386kd



# Debug Boot Options

- /DEBUG - useful to break into a “hung” system
  - Kernel debugger loads at boot time and attempts to connect
    - If no host, boot continues
  - Pro: Can “break in” to target system (run debugger on host and type Cntrl/C)
  - Cons:
    - Takes away a COM port for life of system
    - Windows NT checks to see if host debugger wants to connect each clock tick (negligible impact)
    - System crash: Debugger waits to connect to host, then writes crash dump file
    - Debug output from any driver will cause debugger to activate (hangs system if no host connected)
- /CRASHDEBUG - useful to look at a crash on a system that cannot take a crash dump
  - Kernel debugger loads only when system crashes
    - COM port not taken away while system is up
    - No issue with debug output from drivers
  - But, cannot “break in” to target if hung

# Windows 2000 Internals Information Sources

- MSDN Library
  - Platform SDK API documentation
  - Windows NT Device Driver Kit (DDK) documentation
  - Win32 Knowledge Base - has some Windows NT internals articles
- Past Windows NT/2000 conferences audio/video tapes ([www.mobiletape.com](http://www.mobiletape.com))
- [www.sysinternals.com](http://www.sysinternals.com)
  - Windows 2000 internals articles and tools
- [www.microsoft.com/hwdev](http://www.microsoft.com/hwdev)
  - hardware developers and driver writers
- [www.microsoft.com/hwdev/ntifskit](http://www.microsoft.com/hwdev/ntifskit)
  - Installable File System Developers Kit
- [comp.os.ms-windows.programmer.nt.kernel-mode](mailto:comp.os.ms-windows.programmer.nt.kernel-mode)
  - drivers newsgroup
- [www.cmkrnl.com](http://www.cmkrnl.com) - Windows 2000 device driver FAQ