# Unit OS4: Scheduling and Dispatch

4.6. Lab description

# Copyright Notice



- These materials are part of the *Windows Operating System Internals Curriculum Development Kit,* developed by David A. Solomon and Mark E. Russinovich with Andreas Polze
- Microsoft has licensed these materials from David Solomon Expert Seminars, Inc. for distribution to academic organizations solely for use in academic environments (and not for commercial use)

# Labs for 4.2

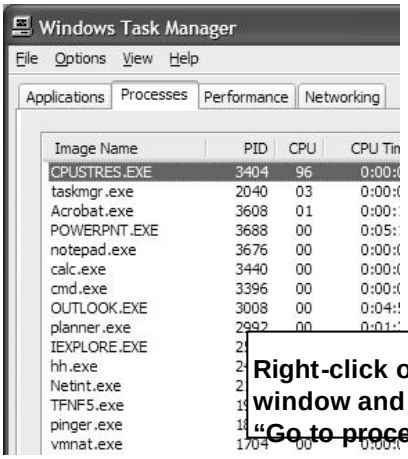
- The following is a superset of the labs presented in module

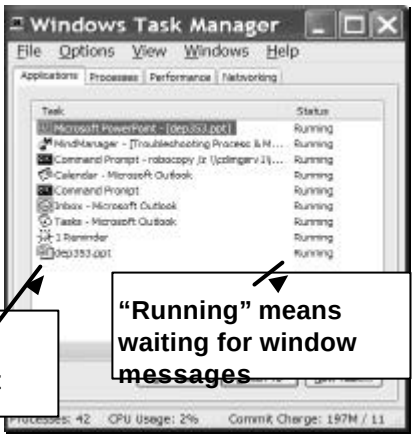
  "4.2. Windows Processes and Threads"



# Task Manager: Processes vs Applications Tabs

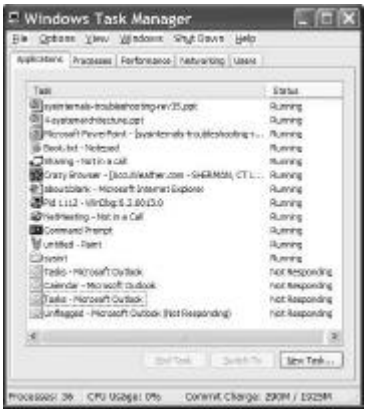
- Processes tab: List of processes
- Applications tab: List of top level visible windows



# What Are Task Manager's "Applications"?

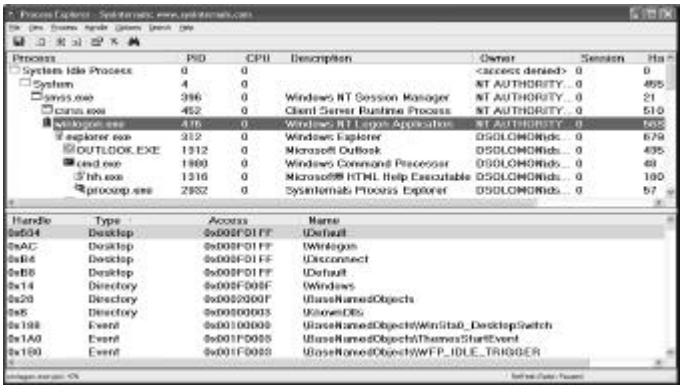
- A meaningless term at the OS level
  - Not a list of processes
  - Not a list of "tasks" (another meaningless term)
  - It's a list of top level visible windows in your session that meet certain criteria
- What does the status column mean?
  - Running:
    - Windows don't run—threads do
    - Running displayed only when owning thread is waiting for a window message (e.g. not running!)
  - Not Responding: not waiting for window messages
- To map a window to a process, right-click on a window and select "Go to process"






# Process Explorer (Sysinternals)

- "Super Task Manager"
  - Shows full image path, command line, environment variables, parent process, security access token, open handles, loaded DLLs & mapped files



# Lab: The Process List

1. Run Process Explorer & maximize window
2. Run Task Manager – click on Processes tab
3. Arrange windows so you can see both
4. Notice process tree vs flat list in Task Manager
   - If parent has exited, process is left justified
5. Sort on first column (“Process”) and note tree view disappears
6. Click on View->Show Process Tree (or CTRL+T) to bring it back
7. Notice description and company name columns



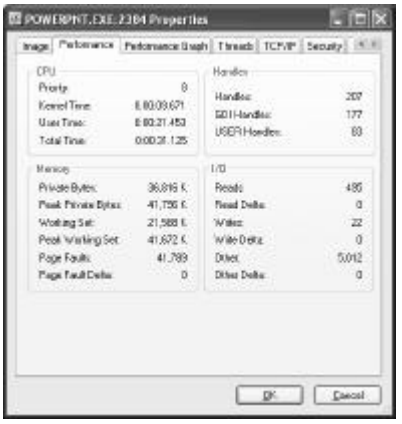
# Lab: Refresh Highlighting

1. Change update speed to paused by pressing space bar
2. Run Notepad
3. In ProcExp, hit F5 and notice new process
4. Exit Notepad
5. In ProcExp, hit F5 and notice Notepad in red

- Uses
  - Understanding process startup sequences
  - Detecting appearance of processes coming and going

# Process Performance

- Click on Performance Tab of process properties
  - Note: all these numbers can be configured as columns






# Thread Details

- Process Explorer "Threads" tab shows which thread(s) are running
  - Start address represents where the thread began running (not where it is now)
  - Click Module to get details on module containing thread start address

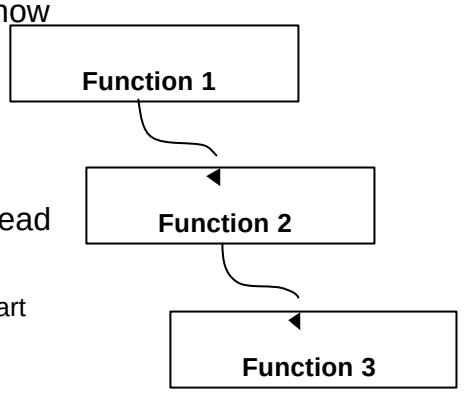
# Thread Start Functions

- Process Explorer can map the addresses within a module to the names of functions
  - This can help identify which component within a process is responsible for CPU usage
- Requires access to:
  - Symbol file for that module
  - Proper version of Dbghelp.dll
- By default, Process Explorer looks for:
  - Dbghelp.dll: in the default Windows Debugging Tools install directory
  - Symbols: _NT_SYMBOL_PATH environment variable
  - Can also specify with Options->Configure Symbols



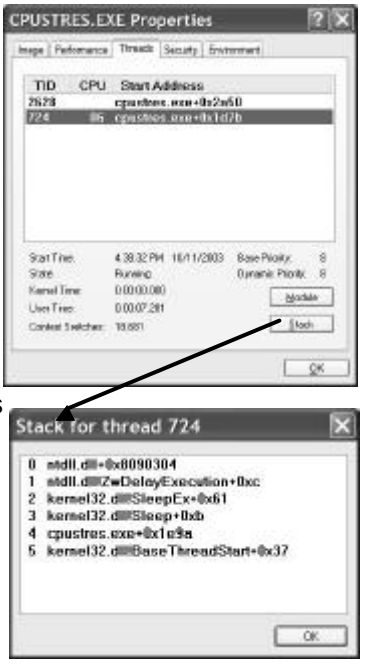
# Call Stacks

- Process Explorer can also show the thread call stack
  - Represents sequence of functions called
- Important if start address doesn't indicate what the thread is doing
  - E.g. if it's a generic library start routine



# Call Stacks

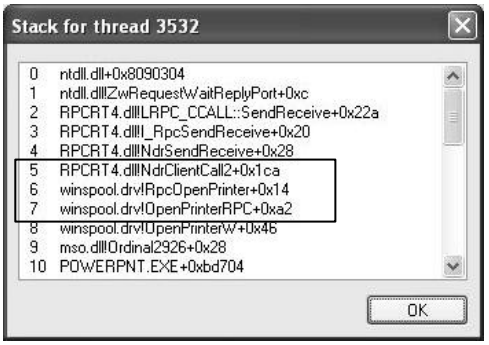
- Click Stack to view call stack
  - Lists functions in reverse chronological order
- Note that start address on Threads tab is different than first function shown in stack
  - This is because all user threads start in a Windows library function which calls the programmed start address






# Example: Viewing Stacks

- Problem: Powerpoint was hanging for 1 minute on startup
- Thread stack shows waiting on a printer driver



# Suspending Processes

- Process Explorer can suspend a process
- Why would you want to do this?
  - You've started a long running job but want to pause it to do something else
    - Lowering the priority still leaves it running…
  - You've started a long download but want to have your network bandwidth temporarily
  - Some multi-service system process activity is due to other processes calling upon their services
    - Suspend a process that is consuming CPU time to see what that does to the system process in question



# Lab: Suspend

- Start Notepad
- From a command prompt:
  1. Suspend Notepad process with Process Explorer
  2. Try to switch back to Notepad (should not respond)
  3. Open Task Manager and look at Notepad's status in the applications tab ☺
  4. Resume Notepad

# Process Explorer Lab: Column Selection And Username

- Notice additional details show for each process (icon, description)
- Click on View->Select Columns
  - Add username column
- Compare username column in Task Manager with Process Explorer – what is the difference?
- Deselect View->Show Processes From All Users

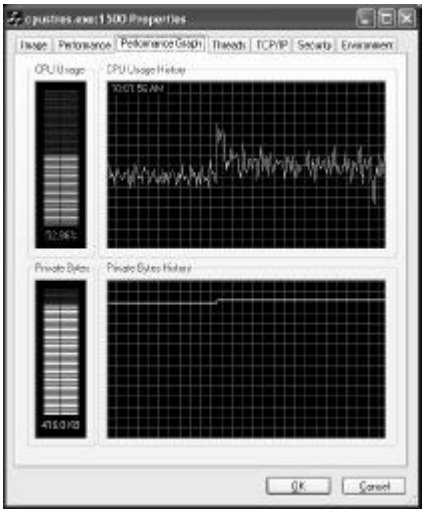


# Process Explorer Lab: Command Line

- Double click on date/time in task bar (lower right of screen)
- In Process Explorer, hit F5 to refresh
- Find new process created (RUNDLL32.EXE)
- Examine command line arguments
- Example: cmd.exe process was consuming lots of CPU time
  - Command line argument showed which .BAT file was running

# Examining CPU Time

- Open process properties and look at CPU usage history on the performance graph page
- Hover the mouse over a point to see the time of that value






# Process Explorer Lab: Environment Variables

- Click on Environment Tab of process properties

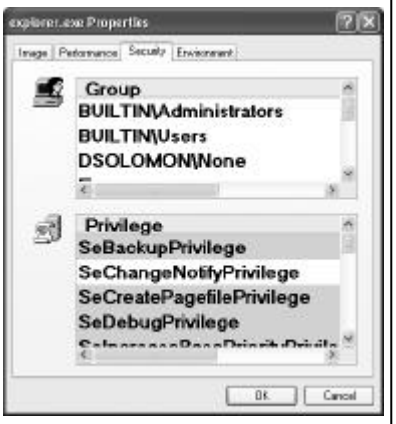
# Process Explorer Lab: Environment Variables

- Open a command prompt
- Run Notepadexe from command prompt
- Type “set dave=smarter_than_mark”
- In ProcExp, hit F5 and examine environment variables for Cmd.exe and Notepad.exe
  - Notice Notepad.exe does not know that Dave is smarter than Mark ☺

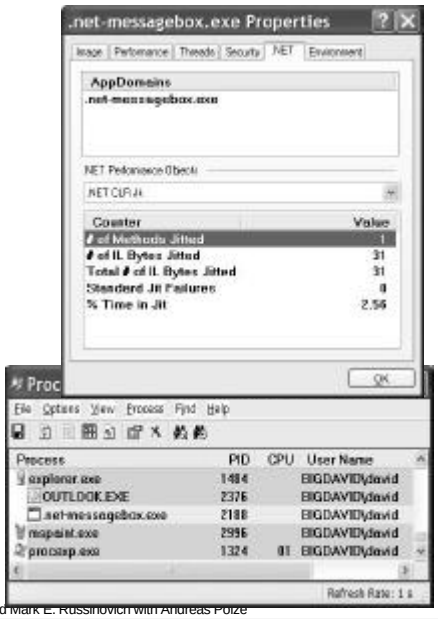


# Security

- Click on Security tab of process properties
- Shows rest of access token (username is on image tab)
  - Groups list
    - Includes OS-assigned groups
  - Privileges (user rights)
    - Disabled by default
    - Programs turn these on when needed
- This is really a “Resultant Set of Groups” and “Resultant Set of Privileges” page



# .NET Information




- Process Explorer is aware of .NET processes
  - Can highlight with Options->Highlight .NET Processes
- Process properties have .NET tab
  - Shows details about .NET process (CLR, Appdomains)
- Can also add .NET-specific columns to process list



# Windows Status

- If you really like Task Manager's Applications tab:
  - Add the Window Title column
  - Add the Window Status column
    - Uses the same Windows function as Task Manager to determine status


# Lab: Window Process Finder

- Use the Window process finder toolbar button to identify the owner of a window
- Lab:
  1. Open Regedit and modify HKLM\System\CurrentControlSet\Control\ ProductOptions\ProductType
  2. Move the window process finder target over the resulting popup to see what process owns the window



# Lab: Viewing Process Activity

1. Task Manager:
   - Applications tab: find the process that owns a window (right mouse click on window title)
   - Process tab: add a few additional columns: Virtual Memory size, Handle count, Thread count
     - Windows 2000: add I/O counters; right click on a process & notice “end process tree” option
2. Look at process hierarchy with Process Explorer
   - Start a command prompt, then run Notepad from command prompt, and find new process in process list
   - Exit the command prompt and notice Notepad process moves to bottom of list since it has no parent

# PS Tools

- Group of 12 process/system control tools
  - Where'd the "Ps" come from?
    - The UNIX process listing tool is named "ps"
    - The first PsTool was a UNIX "ps"-equivalent, PsList
- They all work on Windows NT4/2000/XP/2003
- They all work remotely as well as locally
  - Require admin rights to remote system
    - Can specify credentials with "-u" switch
- None require manual remote software installation



# PS Tools

- Psfile – lists & closes remote file opens
- Psshutdown – remote shutdown, lock workstation, log off user
- Psexec – run an app on a remote system
- Pslist – list processes & threads
- Psuptime – system up time
- Psinfo – display general system info
- Psgetsid – displays computer or user SIDs
- Psservice – service process control (like SC in XP)
- Psloglist – dumps event log in text
- PsSuspend – suspend a process
- PsKill – kill processes
- Psloggedon – lists local and remote logon sessions
- Pspassword – change local/remote passwords

# PsKill

- The perfect complement to PsList is PsKill
  - Similar to Resource Kit Kill and Remote Kill
  - See a process running on a remote (or local) system with PsList, kill it with PsKill
- Unlike Task Manager, PsKill lets you kill *any* process if you're an admin
  - Uses "Debug" privilege
- Uses auto-installed remote service and TerminateProcess API



# PsList/PsKill Lab

1. Open a command prompt
2. Try Pslist on your machine
   - c:\sysint\pslist
   - c:\sysint\pslist –t (tree view)
   - c:\sysint\pslist -s (autorefresh)
3. Look at process list on your neighbor's machine
   - c:\sysint\pslist \\computername
4. Kill Explorer.exe on your neighbor's workstation
   - c:\sysint\pskill \\computer explorer.exe

# PsExec

- Remotely execute programs
  - Executes console programs interactively
  - Allows you to start programs as yourself , in alternate user credentials, or in the System account
- With PsExec you can:
  - Launch a remote command prompt to effect a light-weight telnet
  - Remote-enable “local only” command-line tools like IpConfig
- Uses auto-installed remote service



# PsExec Lab

1. Open a command prompt
2. Run Regedit under System account:
   psexec -s -i c:\windows\regedit.exe
3. Start Notepad interactively on another workstation (or to yourself if not on a network):
   - c:\sysint\psexec –i \\computer notepad.exe
   - Find the Notepad process you created by examining the process tree with pslist on the remote system
     - Notice parent service process

# Lab Series 4.3.

- The following labs capitalize on the materials presented in Section

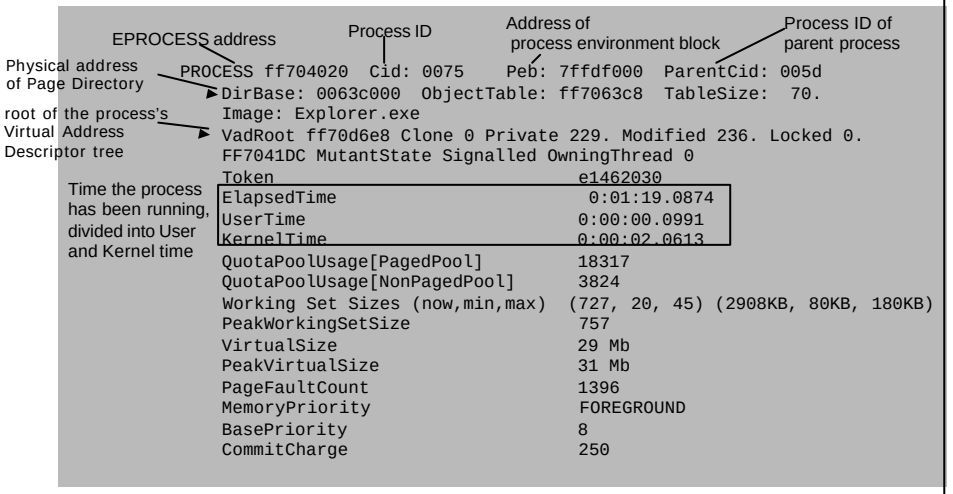
  "4.3. Windows Process and Thread Internals"



# Process/Thread Kernel Debugger Commands

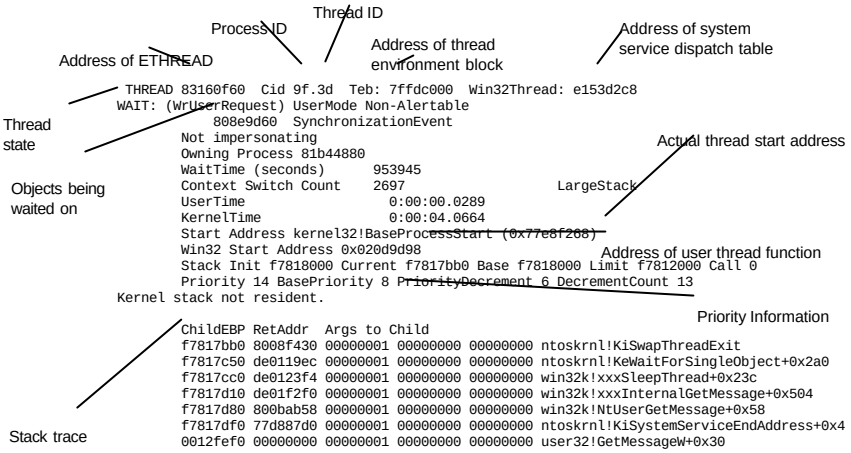
- !process [/s Session] [Address/Pid [Flags]]
  - !process – display current process (not full details)
  - !process 342 – display full details of process 342
  - !process 829fa030 – display process identified by EPROCESS address
  - !process 0 0 – summary display of all processes
  - !process 0 7 – full details of all processes
- !thread [Address [Flags]]
  - !thread – current thread
  - !thread 826e8898 – display thread identified by ETHREAD address
- To view user stack, must set process context:
  - .process <address of EPROCESS>
  - .context <address of page directory (Dirbase)>
- !peb [Address]
- !teb [Address]

# Process Block (!process)






# Thread Block (!thread)



# Dumping Structures with Kernel Debugger

- "dt" ("Display Type") command can format most kernel structures
  - Syntax: "dt StructureName address –r"
  - dt nt!_* - displays all OS structures known to dt
  - Note: relies on type information in symbol files
    - Public symbols have this for XP, Windows Server 2003, and Windows 2000 SP4 and later
- Process/thread-related structures:
  - nt!_EPROCESS
  - nt!_ETHREAD
  - nt!_PEB
  - nt!_TEB
  - nt!_TOKEN
  - nt!_JOB



# Process Block Layout

```
lkd> dt nt!_EPROCESS
   +0x000 Pcb              : _KPROCESS
   +0x06c ProcessLock      : _EX_PUSH_LOCK
   +0x070 CreateTime       : _LARGE_INTEGER
   +0x078 ExitTime         : _LARGE_INTEGER
   +0x080 RundownProtect   : _EX_RUNDOWN_REF
   +0x084 UniqueProcessId  : Ptr32 Void
   +0x088 ActiveProcessLinks : _LIST_ENTRY
   +0x090 QuotaUsage       : [3] Uint4B
   +0x09c QuotaPeak        : [3] Uint4B
   +0x0a8 CommitCharge     : Uint4B
   +0x0ac PeakVirtualSize  : Uint4B
   +0x0b0 VirtualSize      : Uint4B
                           .
                           .
```

➢ **NOTE: Add "-r" to recurse through substructures**

# Thread Block (!strct ethread)

```
lkd> dt nt!_ETHREAD
   +0x000 Tcb            : _KTHREAD
   +0x1c0 CreateTime     : _LARGE_INTEGER
   +0x1c0 NestedFaultCount : Pos 0, 2 Bits
   +0x1c0 ApcNeeded      : Pos 2, 1 Bit
   +0x1c8 ExitTime       : _LARGE_INTEGER
   +0x1c8 LpcReplyChain  : _LIST_ENTRY
   +0x1c8 KeyedWaitChain : _LIST_ENTRY
   +0x1d0 ExitStatus     : Int4B
   +0x1d0 OfsChain       : Ptr32 Void
   +0x1d4 PostBlockList  : _LIST_ENTRY
   +0x1dc TerminationPort : Ptr32 _TERMINATION_PORT
   +0x1dc ReaperLink     : Ptr32 _ETHREAD
```

- **NOTE: Add "-r" to recurse through substructures**



# Lab Series 4.4.

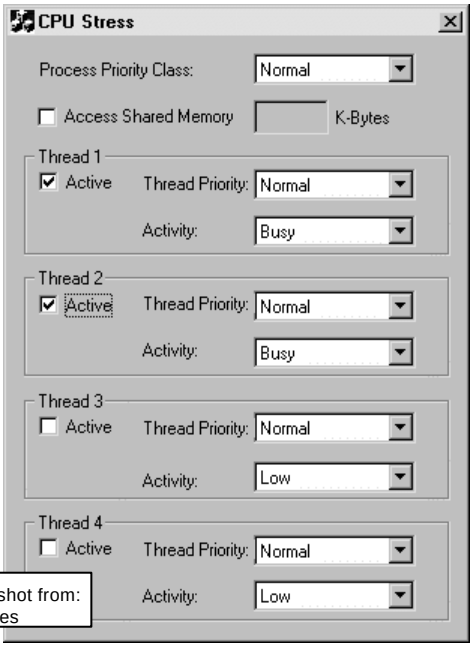
- The following labs capitalize on the materials presented in Section

  "4.4. Windows Thread Scheduling"

# Watching Scheduling

CPUSTRES.EXE - Creating a Test Case

- Run: cpustres.exe (Resource Kit)

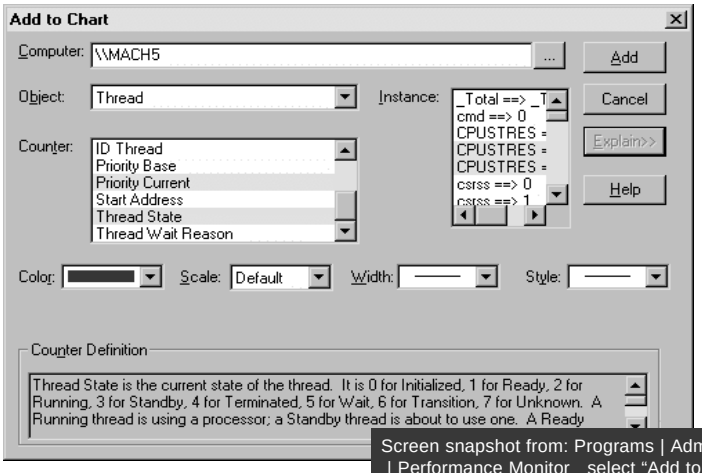



Screen snapshot from: Run… cpustres



# Watching the Scheduler

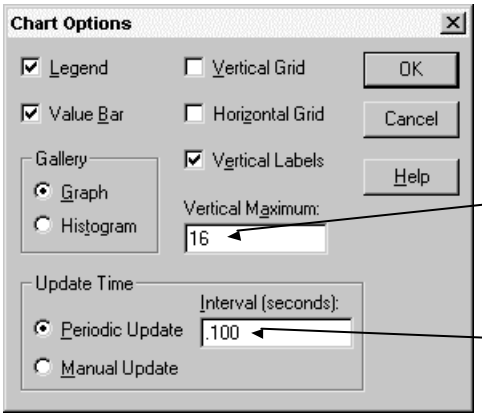
## Performance Monitor - Threads Object




Screen snapshot from: Programs | Admin. Tools | Performance Monitor select “Add to Chart”, and Object: Thread. use Ctrl-leftClick to select multiple items in a selection box

# Watching the Scheduler

## Performance Monitor - Options | Chart




**Set chart maximum vertical scale to 16**

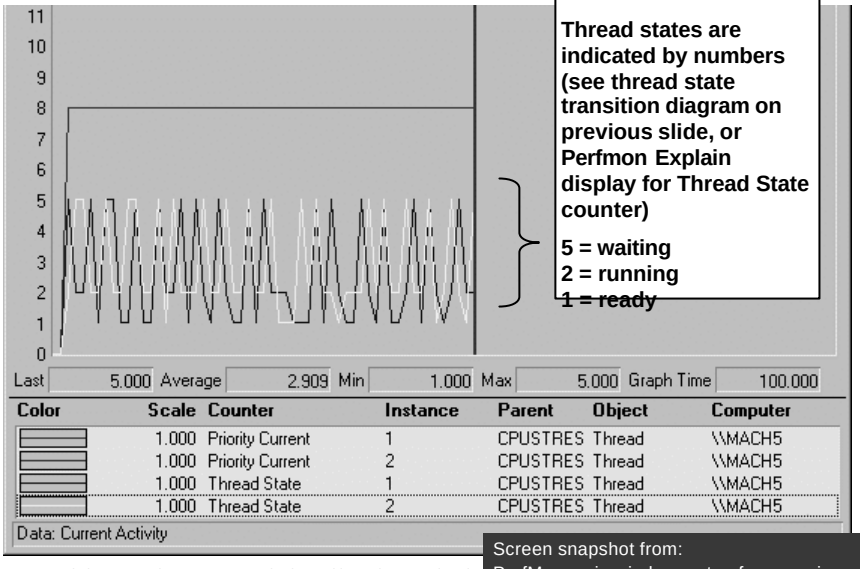
**Set update interval to 0.1 seconds or less**

Screen snapshot from: Performance Monitor Options menu | Chart command



# Watching the Scheduler

## Performance Monitor




**Thread states are indicated by numbers (see thread state transition diagram on previous slide, or Perfmon Explain display for Thread State counter)**

**5 = waiting**
**2 = running**
**1 = ready**

Screen snapshot from:
PerfMon main window, setup from previous slide

# Lab Series 4.5.

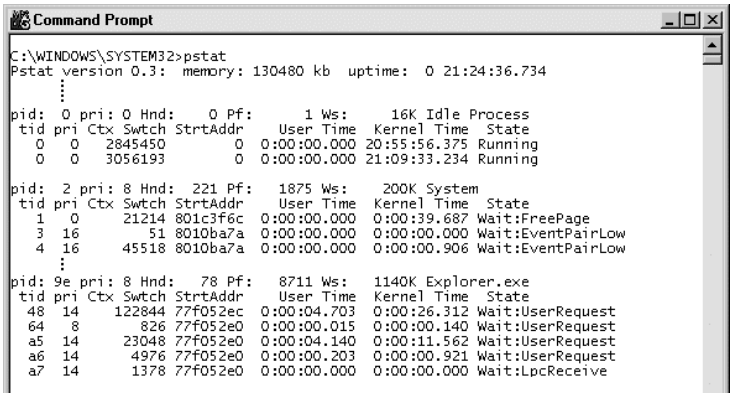
- The following labs capitalize on the materials presented in Section

  "4.5. Advanced Windows Scheduling"



# Looking at Waiting Threads

- For waiting threads, user-mode utilities only display the wait reason
- Example: pstat

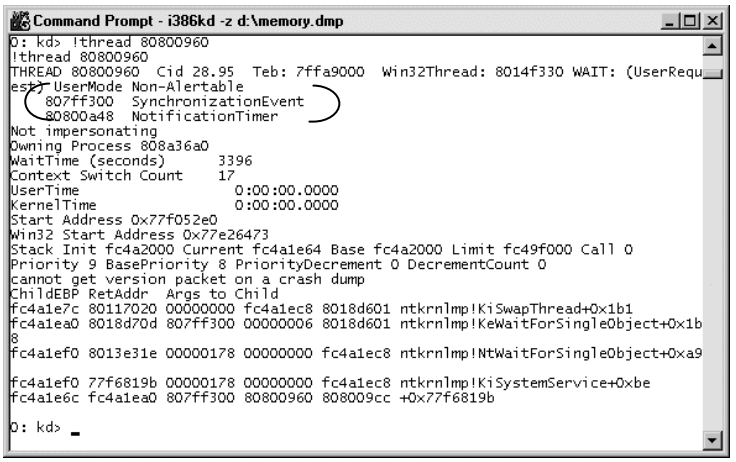


```
Command Prompt

C:\WINDOWS\SYSTEM32>pstat
Pstat version 0.3:  memory: 130480 kb  uptime:  0 21:24:36.734
       ⋮
pid:  0 pri: 0 Hnd:    0 Pf:      1 Ws:     16K Idle Process
 tid pri Ctx Swtch StrtAddr    User Time  Kernel Time  State
   0   0   2845450        0  0:00:00.000 20:55:56.375 Running
   0   0   3056193        0  0:00:00.000 21:09:33.234 Running

pid:  2 pri: 8 Hnd:  221 Pf:   1875 Ws:    200K System
 tid pri Ctx Swtch StrtAddr    User Time  Kernel Time  State
   1   0     21214 801c3f6c  0:00:00.000  0:00:39.687 Wait:FreePage
   3  16        51 8010ba7a  0:00:00.000  0:00:00.000 Wait:EventPairLow
   4  16     45518 8010ba7a  0:00:00.000  0:00:00.906 Wait:EventPairLow
       ⋮
pid: 9e pri: 8 Hnd:   78 Pf:   8711 Ws:   1140K Explorer.exe
 tid pri Ctx Swtch StrtAddr    User Time  Kernel Time  State
  48  14    122844 77f052ec  0:00:04.703  0:00:26.312 Wait:UserRequest
  64   8       826 77f052e0  0:00:00.015  0:00:00.140 Wait:UserRequest
  a5  14     23048 77f052e0  0:00:04.140  0:00:11.562 Wait:UserRequest
  a6  14      4976 77f052e0  0:00:00.203  0:00:00.921 Wait:UserRequest
  a7  14      1378 77f052e0  0:00:00.000  0:00:00.000 Wait:LpcReceive
```

- To find out <u>what</u> a thread is waiting on, must use kernel debugger

# Looking at Wait Queues

- !thread command to kernel debugger
  - Lists addresses of objects being waited on (if a mutex, shows owner)
  - !irpfind can search IRPs for an event object address



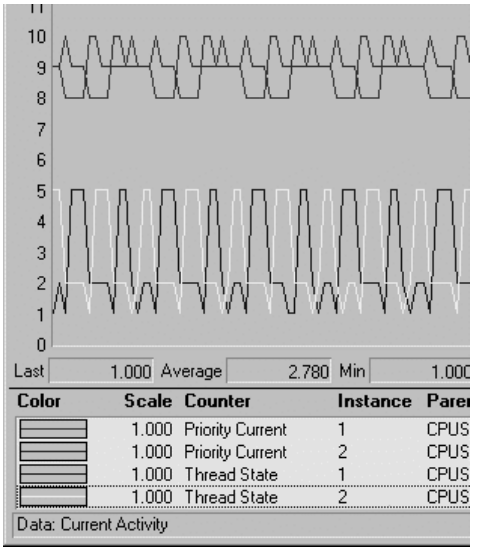



# Lab: Foreground Priority Boosts

- See Book “EXPERIMENT: Watching Foreground Priority Boosts and Decays”, p.351
- See Book “EXPERIMENT: Watching Priority Boosts on GUI Threads”, p.353

# Priority Boost and Decay
## Demo with CpuStres and PerfMon

- CpuStres settings:
  - two active threads
  - activity level = busy (about 25% wait time)
  - normal process priority class, normal thread priorities
- Usually only visible in PerfMon if target app owns foreground window (hence longer quantum)
- These are showing +2 boost (from 8 to 10) for foreground apps after wait completion






# Lab: CPU Starvation Resolution

- See Book EXPERIMENT: Watching Priority Boosts for CPU Starvation, p.355
  - CpuStres with two compute-bound threads (“maximum” activity level)
  - One is at lower priority than the other
- See Book EXPERIMENT: “Listening to Priority Boosting”, p.357