Unit OS4: Scheduling and Dispatch

4.3. Windows Process and Thread Internals

Roadmap for Section 4.3.

- Windows Process and Thread Internals
- Thread Block, Process Block
- Flow of Process Creation
- Thread Creation and Deletion
- Process Crashes
- Windows Error Reporting

2

Windows Process and Thread Internals

Data Structures for each
  process/thread:

- Executive process block
  (EPROCESS)
- Executive thread block
  (ETHREAD)
- Win32 process block
- Process environment block
- Thread environment block

Process environment block → Thread environment block

Process address space
System address space

Process block (EPROCESS) → Win32 process block
→ Handle table
Thread block (ETHREAD) → ...
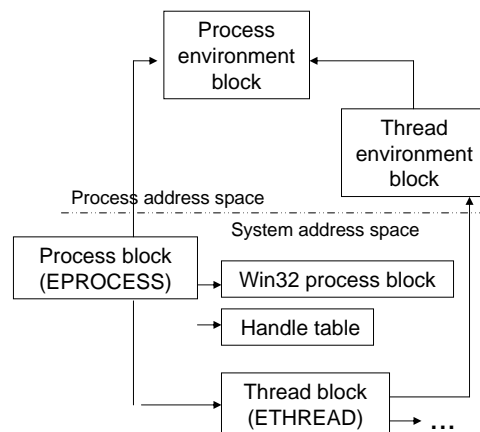
Process

- Container for an address space and threads
- Associated User-mode Process Environment Block (PEB)
- Primary Access Token
- Quota, Debug port, Handle Table etc
- Unique process ID
- Queued to the Job, global process list and Session list
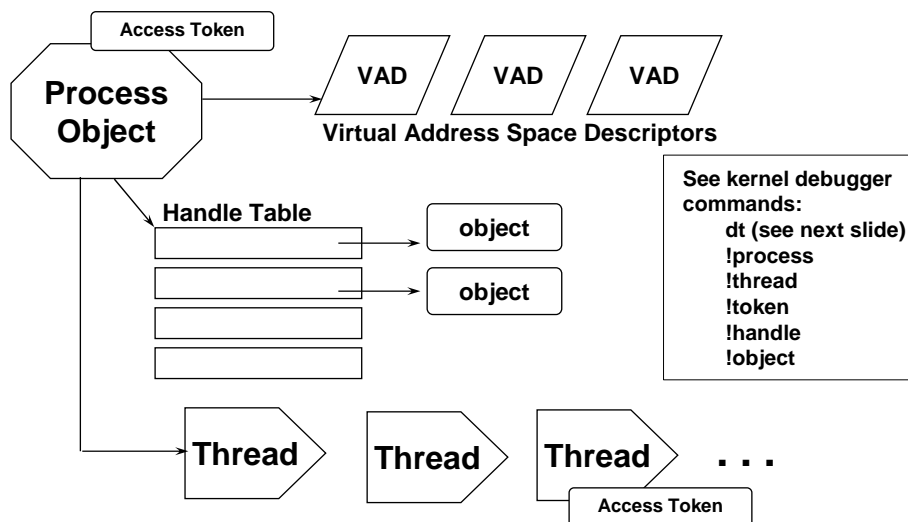- MM structures like the WorkingSet, VAD tree, AWE etc

Thread

- Fundamental schedulable entity in the system
- Represented by ETHREAD that includes a KTHREAD
- Queued to the process (both E and K thread)
- IRP list
- Impersonation Access Token
- Unique thread ID
- Associated User-mode Thread Environment Block (TEB)
- User-mode stack
- Kernel-mode stack
- Processor Control Block (in KTHREAD) for CPU state when not running

5

Processes & Threads
Internal Data Structures

Access Token

Process Object

VAD   VAD   VAD

Virtual Address Space Descriptors

See kernel debugger commands:
    dt (see next slide)
    !process
    !thread
    !token
    !handle
    !object

Handle Table

object

object

Thread   **Thread**   **Thread**   . . .

Access Token

6

Process Block Layout

Kernel Process Block (or PCB)
Process ID
Parent Process ID
Exit Status
Create and Exit Time
Next Process Block	→ EPROCESS
Quota Block
Memory Management Information
Exception Port
Debugger Port
●	→ Primary Access Token
●	→ Handle Table
Process Environment Block
Image File Name
Image Base Address
Process Priority Class
●	→ Win32 Process Block

Dispatcher Header
●	→ Process Page Directory
Kernel Time
User Time
Inwwap/Outswap List Entry
●	→ KTHREAD . . .
Process Spin Lock
Processor Affinity
Resident Kernel Stack Count
Process Base Priority
Default Thread Quantum
Process State
Thread Seed
Disable Boost Flag

Thread Block

ETHREAD

KTHREAD
Create and Exit Time
Process ID
●	→ EPROCESS
Thread Start Address
●	→ Access Token
Impersonation Information
LPC Message Information
Timer Information
●	→ Pending I/O Requests

KTHREAD

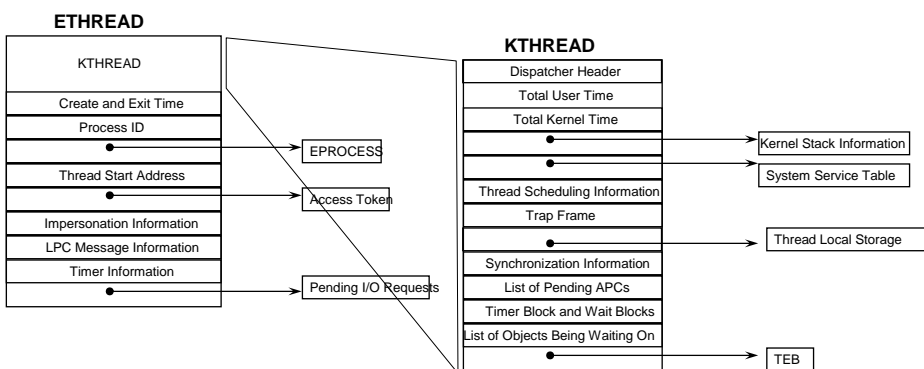| Dispatcher Header |
|---|
| Total User Time |
| Total Kernel Time |
| ● | → Kernel Stack Information |
| ● | → System Service Table |
| Thread Scheduling Information |
| Trap Frame |
| ● | → Thread Local Storage |
| Synchronization Information |
| List of Pending APCs |
| Timer Block and Wait Blocks |
| List of Objects Being Waiting On |
| ● | → TEB |

8
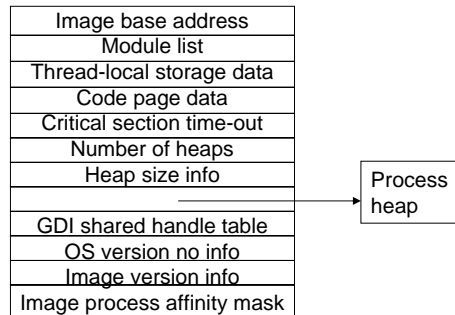
4

Process Environment Block
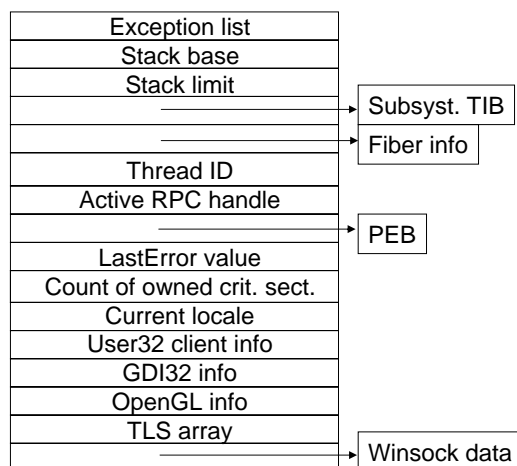
- Mapped in user space

- Image loader, heap manager, Windows system DLLs use this info

- View with !peb or dt nt!_peb

Image base address
Module list
Thread-local storage data
Code page data
Critical section time-out
Number of heaps
Heap size info
GDI shared handle table
OS version no info
Image version info
Image process affinity mask

Process heap

Thread Environment Block

- User mode data structure

- Context for image loader and various Windows DLLs

- View with !teb or dt nt!_teb

| Exception list |
| --- |
| Stack base |
| Stack limit |
| |
| |
| |
| Thread ID |
| Active RPC handle |
| |
| LastError value |
| Count of owned crit. sect. |
| Current locale |
| User32 client info |
| GDI32 info |
| OpenGL info |
| TLS array |
| |

Subsyst. TIB

Fiber info

PEB

Winsock data

Flow of CreateProcess()

1. Open the image file (.EXE) to be executed inside the process
2. Create Windows NT executive process object
3. Create initial thread (stack, context, Win NT executive thread object)
4. Notify Windows subsystem of new process so that it can set up for new proc.& thread
5. Start execution of initial thread (unless CREATE_SUSPENDED was specified)
6. In context of new process/thread: complete initialization of address space (load DLLs) and begin execution of the program

11

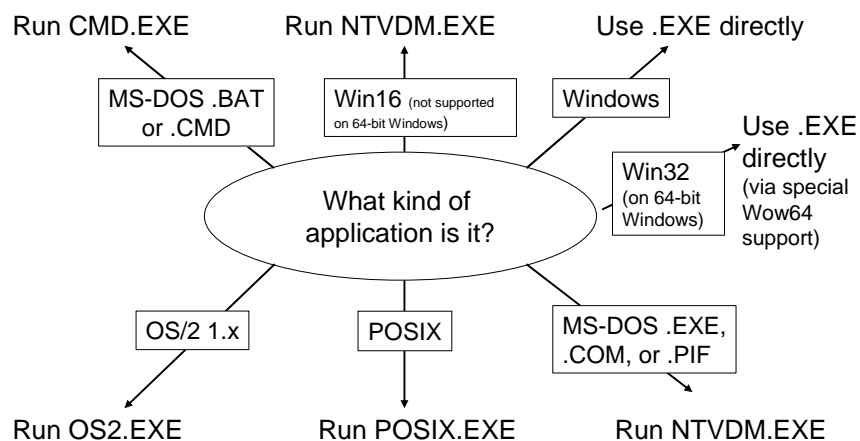# The main Stages Windows follows to create a process
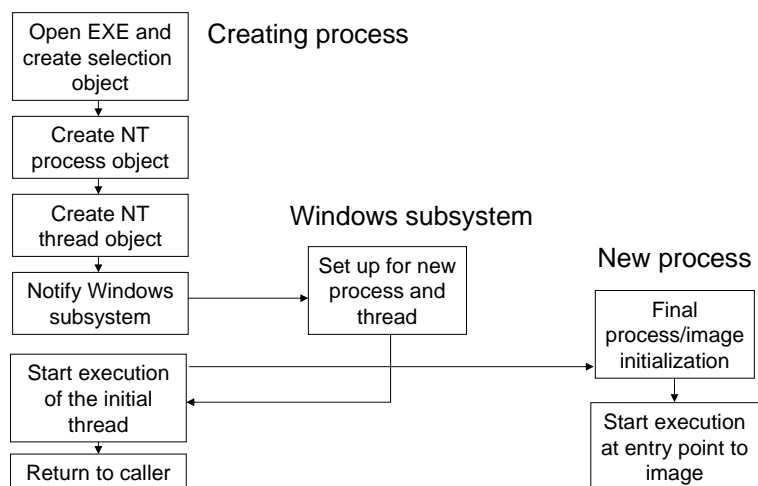
12

CreateProcess: some notes

- CreationFlags: independent bits for priority class
  -> NT assigns lowest-priority class set

- Default priority class is normal
  unless creator has priority class idle

- If real-time priority class is specified and
  creator has insufficient privileges:
  priority class high is used

- Caller's current desktop is used
  if no desktop is specified

Opening the image to be executed

Run CMD.EXE   Run NTVDM.EXE   Use .EXE directly

| MS-DOS .BAT or .CMD | Win16 (not supported on 64-bit Windows) | Windows |

Win32 (on 64-bit Windows) → Use .EXE directly (via special Wow64 support)

What kind of application is it?

| OS/2 1.x | POSIX | MS-DOS .EXE, .COM, or .PIF |

Run OS2.EXE   Run POSIX.EXE   Run NTVDM.EXE

If executable has no Windows format...

- CreateProcess uses Windows „support image"

- No way to create non-Windows processes directly

    - OS2.EXE runs only on Intel systems

    - Multiple MS-DOS apps may share virtual dos machine

    - .BAT of .CMD files are interpreted by CMD.EXE

    - Win16 apps may share virtual dos machine (VDM)
      Flags: CREATE_SEPARATE_WOW_VDM
                      CREATE_SHARED_WOW_VDM
      Default: HKLM\System...\Control\WOW\DefaultSeparateVDM

    - Sharing of VDM only if apps run on same desktop under same security

- Debugger may be specified under (run instead of app !!)

    \Software\Microsoft\WindowsNT\CurrentVersion\ImageFileExecutionOptions

15

Process Creation - next Steps...

- CreateProcess has opened Windows executable and created a section object to map in proc's addr space

Now: create executive process object via NtCreateProcess

  - Set up EPROCESS block

  - Create initial process address space (page directory, hyperspace page, working set list)

  - Create kernel process block (set inital quantum)

  - Conlude setup of process address space (VM, map NTDLL.DLL, map lang support tables, register process: PsActiveProcessHead)

  - Set up Process Environment Block

  - Complete setup of executive process object

16

Further Steps...(contd.)

- Create Initial Thread and Its Stack and Context
  - NtCreateThread; new thread is suspended until CreateProcess returns
- Notify Windows Subsystem about new process
  KERNEL32.DLL sends message to Windows subsystem including:
  - Process and thread handles
  - Entries in creation flags
  - ID of process's creator
  - Flag describing Windows app (CSRSS may show startup cursor)
- Windows: duplicate handles (inc usage count), set priority class, bookkeeping
  - allocate CSRSS proc/thread block, init exception port, init debug port
  - Show cursor (arrow & hourglass), wait 2 sec for GUI call, then wait 5 sec for window

CreateProcess: final steps

Process Initialization in context of new process:
- Lower IRQL level (dispatch -> **A**sync.**P**roc.**C**all. level)
- Enable working set expansion
- Queue APC to exec *LdrInitializeThunk* in NTDLL.DLL
- Lower IRQL level to 0 – APC fires,
  - Init loader, heap manager, NLS tables, TLS array, crit. sect. Structures
  - Load DLLs, call DLL_PROCESS_ATTACH func
- Debuggee: all threads are suspended
  - Send msg to proc's debug port
    (Windows creates CREATE_PROCESS_DEBUG_INFO event)
- Image begins execution in user-mode (return from trap)

Process Rundown Sequence

1. DLL notification
   - unless TerminateProcess used
2. All handles to executive and kernel objects are closed
3. Terminate any active threads
4. Process's exit code changes from STILL_ACTIVE to the specified exit code

```
BOOL GetExitCodeProcess(
            HANDLE hProcess,
            LPDWORD lpdwExitCode);
```

5. Process object & thread objects become signaled
6. When handle and reference counts to process object == 0, process object is deleted

Creation of a Thread

1. The thread count in the process object is incremented.
2. An executive thread block (ETHREAD) is created and initialized.
3. A thread ID is generated for the new thread.
4. The TEB is set up in the user-mode address space of the process.
5. The user-mode thread start address is stored in the ETHREAD.

Creation of a Thread

6. KeInitThread is called to set up the KTHREAD block.

   - The thread's initial and current base priorities are set to the process's base priority, and its affinity and quantum are set to that of the process.
   - KeInitThread allocates a kernel stack for the thread and initializes the machine-dependent hardware context for the thread, including the context, trap, and exception frames.
   - The thread's context is set up so that the thread will start in kernel mode in KiThreadStartup.
   - Finally, KeInitThread sets the thread's state to Initialized and returns to PspCreateThread.

7. Any registered systemwide thread creation notification routines are called.

8. The thread's access token is set to point to the process access token,

   - an access check is made to determine whether the caller has the right to create the thread.

9. Finally, the thread is readied for execution.

Thread Rundown Sequence

1. DLL notification
   - unless TerminateThread was used
2. All handles to Windows User and GDI objects are closed
3. Outstanding I/Os are cancelled
4. Thread stack is deallocated
5. Thread's exit code changes from STILL_ACTIVE to the specified exit code

```
BOOL GetExitCodeThread(
            HANDLE hThread,
            LPDWORD lpdwExitCode);
```

6. Thread kernel object becomes signaled
7. When handle and reference counts == 0, thread object deleted
8. If last thread in process, process exits

Start of Thread Wrapper

- All threads in all Windows processes appear to have one of just two different start addresses, regardless of the .EXE running
  - One for thread 0 (start of process wrapper), the other for all other threads (start of thread wrapper)
- These "wrapper" functions are what Process Viewer shows as Thread Start Address for Windows apps
- Start of process & start of thread wrappers have same behavior
  - ◆ Provides default exception handling, access to debugger, etc.
  - ◆ Forces thread exit when thread function returns
- To find "real" Windows start address, use TLIST <processname> (or Kernel Debugger !thread command)

Windows Start of Process/Thread Function(conceptual model)

```
void BaseProcessStart  [or BaseThreadStart - basically the same]  (
    LPTHREAD_START_ROUTINE lpStartAddr,
    LPVOID lpvThreadParm)
{
    __try {
        DWORD dwThreadExitCode = lpStartAddr(lpvThreadParm);
        ExitThread(dwThreadExitCode);
    }
    __except(UnhandledExceptionFilter(
            GetExceptionInformation())) {
        ExitProcess(GetExceptionCode());
    }
}
```

Windows Unhandled Exception Filter

```
if process has a debugger attached
    return EXCEPTION_CONTINUE_SEARCH
if AUTO=0 {                        // run debugger automatically?
    Display message box;           // no - ask user what to do
    if(clicked OK)
            ExitProcess();
}

// either AUTO=1, or (AUTO=0 and user clicked CANCEL),
// so run debugger
GetProfileString("AEdebug","debugger",...);
hEvent = CreateEvent( ... );
hProcess = CreateProcess(...);  // Create debugger
    - pass process id, event to signal
WaitForMultipleObjects( [hEvent, hProcess] );
return EXCEPTION_CONTINUE_SEARCH;
```
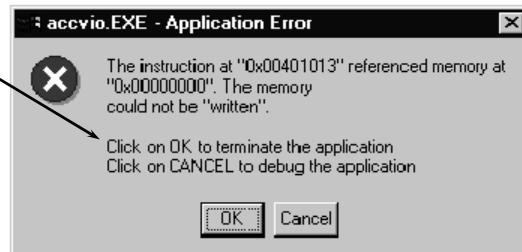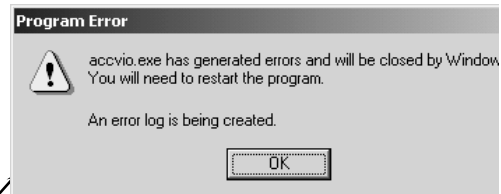
◆ **Implication: you can connect a debugger (VC++ or WinDbg) to a running process**

```
C:\> msdev -p pid
```

Process Crashes (Windows 2000)

- 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
  system is
  Auto=1; Debugger=DRWTSN32.EXE

- Default with VC++ is
  Auto=0, Debugger=MSDEV.EXE

Program Error

accvio.exe has generated errors and will be closed by Window
You will need to restart the program.

An error log is being created.

[OK]

accvio.EXE - Application Error

The instruction at "0x00401013" referenced memory at
"0x00000000". The memory
could not be "written".

Click on OK to terminate the application
Click on CANCEL to debug the application

[OK] [Cancel]

Process Crashes (Windows XP & Windows Server 2003)

- On XP & Server 2003, when an unhandled exception occurs:
  - System first runs DWWIN.EXE
    - DWWIN creates a process microdump and XML file and offers the option to send the error report
  - Then runs debugger (default is Drwtsn32.exe)

---

accvio.EXE

accvio.EXE has encountered a problem and needs to close. We are sorry for the inconvenience.

If you were in the middle of something, the information you were working on might be lost.

Please tell Microsoft about this problem.
We have created an error report that you can send to us. We will treat this report as confidential and anonymous.

To see what data this error report contains, click here.

[Send Error Report] [Don't Send]

Windows Error Reporting

- Configurable with System Properties->Advanced->Error Reporting
  - HKLM\SOFTWARE \Microsoft\PCHealth \ErrorReporting
- Configurable with group policies
  - HKLM\SOFTWARE \Policies\Microsoft \PCHealth

Error Reporting

You can choose to have software errors reported to Microsoft to help improve future products.

○ Disable error reporting
  ☑ But notify me when critical errors occur

◉ Enable error reporting
  ☑ Windows operating system
  ☑ Programs   [Choose Programs...]

[OK] [Cancel]

Further Reading

- Mark E. Russinovich and David A. Solomon, Microsoft Windows Internals, 4th Edition, Microsoft Press, 2004.
  - Chapter 6 - Processes, Thread, and Jobs (from pp. 289)
  - Process Internals (from pp. 289)
  - Flow of Create Process (from pp. 300)
  - Thread Internals (from pp. 313)

Source Code References

- Windows Research Kernel sources
  - \base\ntos\ke
    - procobj.c - Process object
    - thredobj.c, thredsup.c – Thread object
  - \base\ntos\ke\i386, amd64 – thredini.c – architecture specific thread initialization
  - \base\ntos\inc\ke.h, ps.h – structure/type definitions