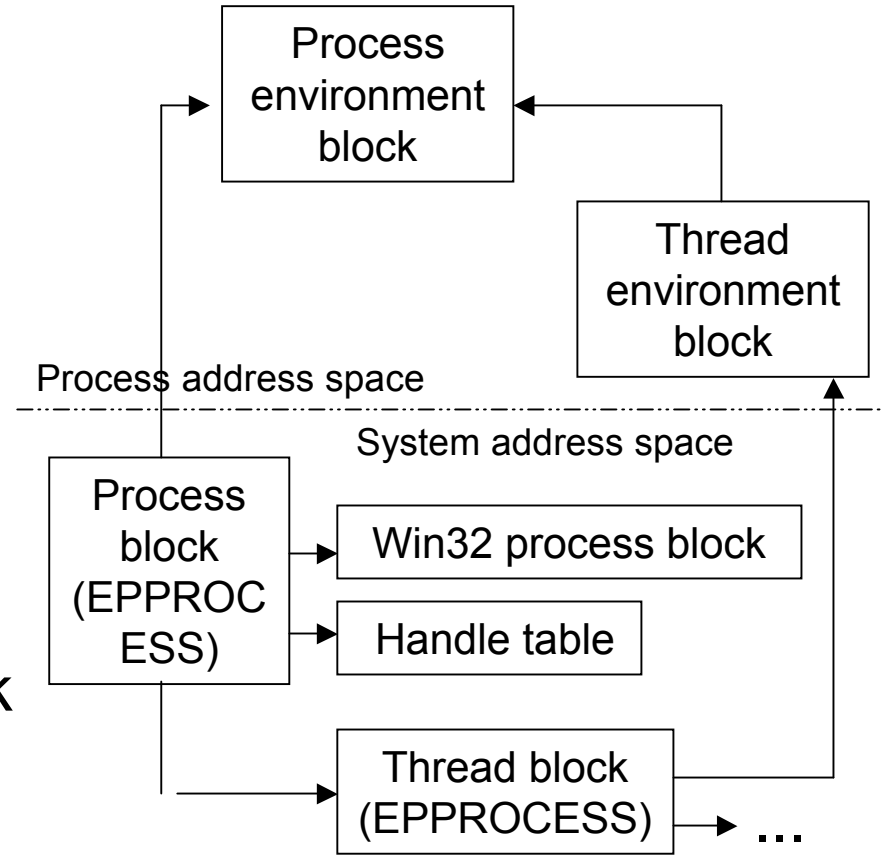
Unit 3: Processes and Threads

3.3. Windows 2000 Process and Thread Internals

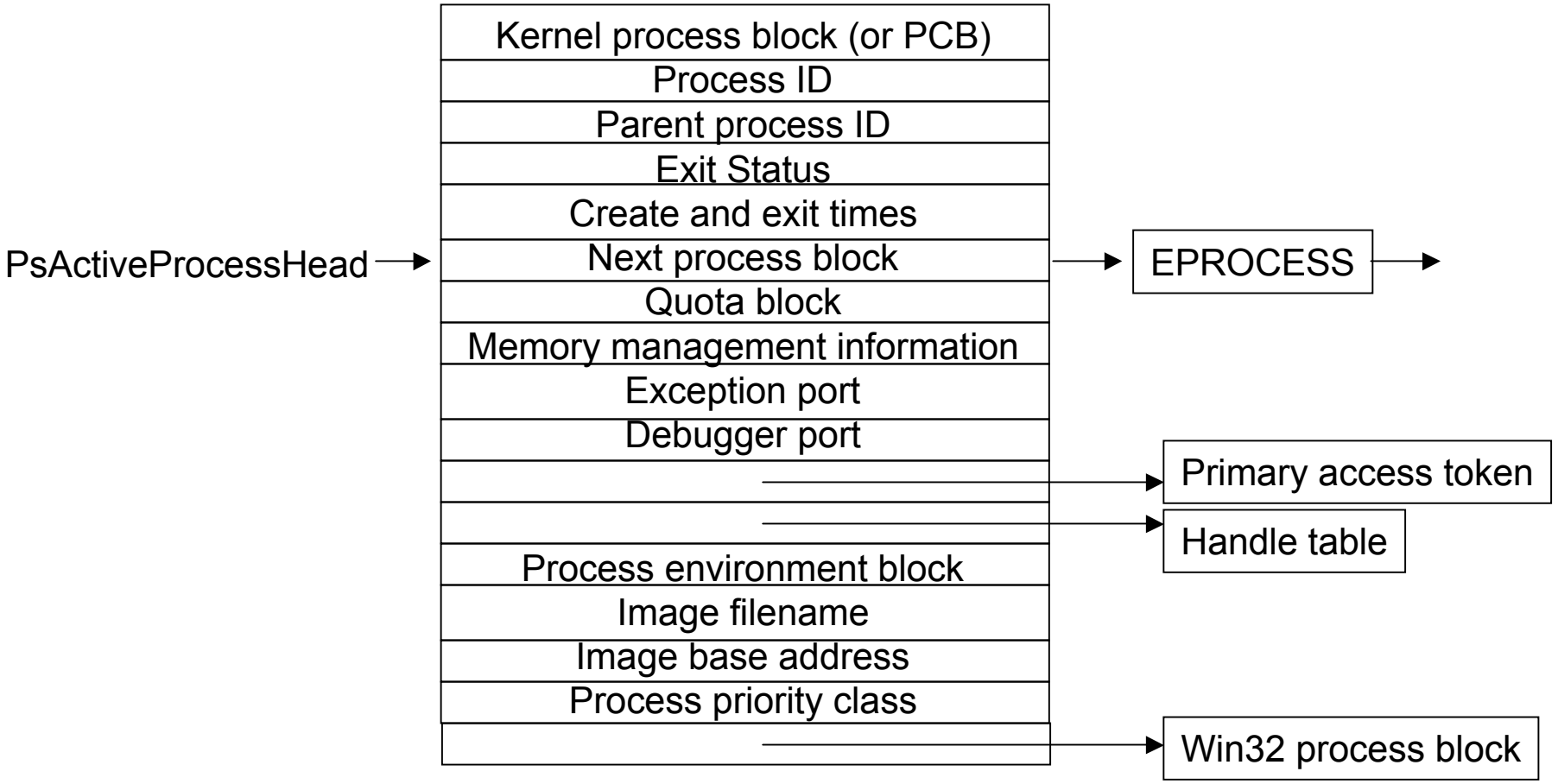
Windows 2000 Process and Thread Internals

Data Structures for each process/thread

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

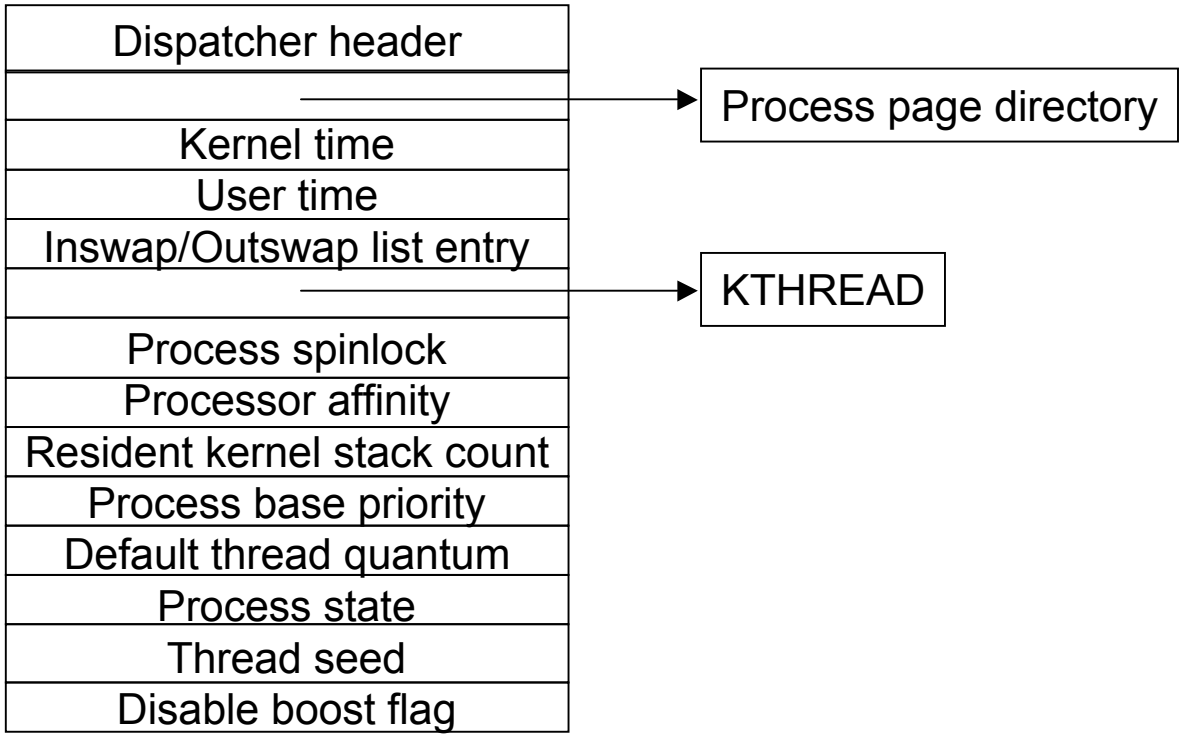


Structure of Executive Process Block



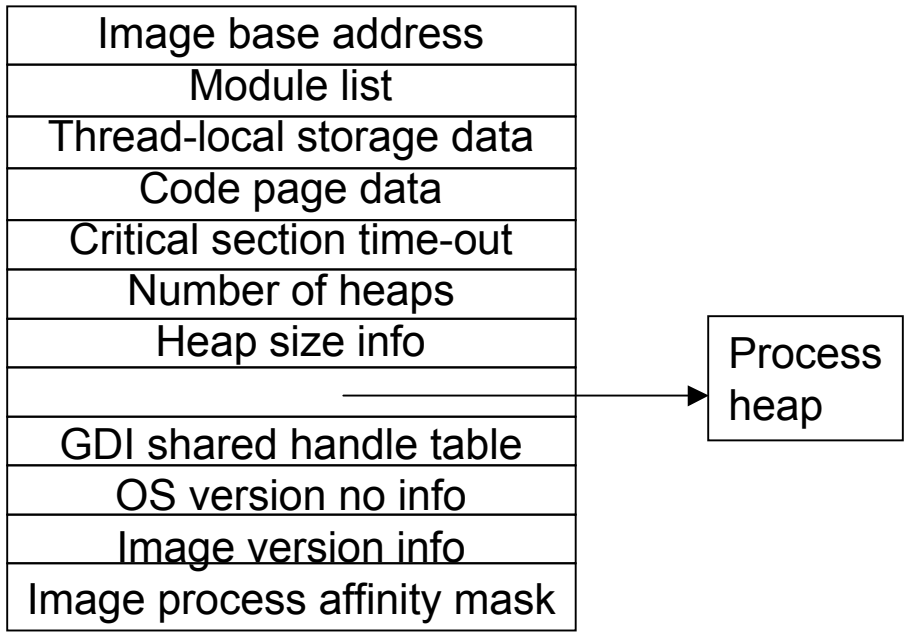
Key substructure: Kernel process block

Sometimes called PCB – process control block



Key substructure: Process Environment Block

- Always mapped at address 0x7FFDF000 (user space)
- Image loader, heap manager, Win32 system DLLs use this info



Process-Related System Variables

Variable	Type	Description
PsActiveProcessHead	Queue header	List head of process blocks
PsIdleProcess	EPROCESS	Idle process block
PsInitialSystemProcess	Pointer to EPROCESS	Pointer to process block of initial system process (PID 2) that contains system threads
PspCreateProcess-NotifyRoutine	Array of 32-bit pointers	Pointers to routines to be called on process creation and deletion (max. 8)
PspCreateProcess-NotifyRoutineCount	DWORD	Count of registered process notification routines
PspCidTable	Pointer to HANDLE_TABLE	Handle table for process and thread client IDs

Process-Related Performance Counters

Object: Counter	Function
Process:%PrivilegedTime	Percentage of time that the threads in the process have run in kernel mode
Process:%ProcessorTime	Percentage of CPU time that threads have used during specified interval %PrivilegedTime + %UserTime
Process:%UserTime	Percentage of time that the threads in the process have run in user mode
Process: ElapsedTime	Total lifetime of process in seconds
Process: ID Process	PID – process IDs are re-used
Process: ThreadCount	Number of threads in a process

Process-Related Functions

Function	Description
CreateProcess	Create new proc.& thread; using callers security ID
CreateProcess-AsUser	Creates new proc.&thread using alternate security ID and then executes specified .EXE
OpenProcess	Returns a handle of specified process object
ExitProcess	Exits current process
TerminateProcess	Terminates a process
FlushInstruction-Cache	Empties another process‘s instruction cache
GetProcessTimes	Obtains another process‘s timing info (%user/%kernel)
GetExitCodeProcess	Returns exit code for another process, indicating how and why process was shut down

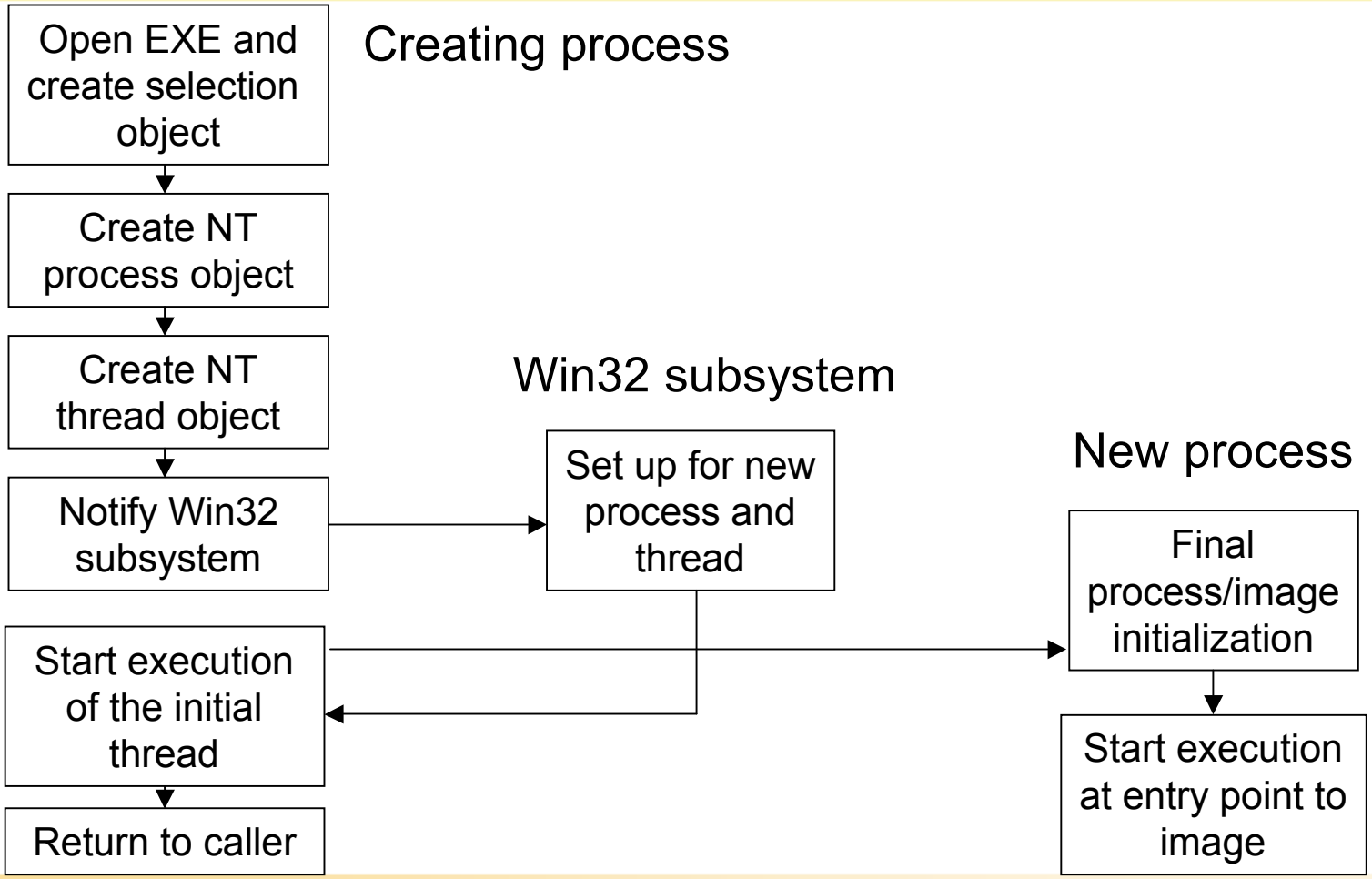
Process-Related Functions (contd.)

Function	Description
GetCommandLine	Returns command-line string passed to the process
GetCurrentProcessID	Returns ID of current process
GetProcessVersion	Returns major/minor versions of Windows version on which the specified process expects to run
GetStartupInfo	Returns contents of STARTUPINFO structure specified during CreateProcess
GetEnvironment-Strings	Returns address of environment block
GetEnvironment-Variable	Returns a specific environment variable
Get/SetProcess-ShutdownParameters	Defines shutdown priority and number of retries for current process

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 Win32 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

The main Stages NT follows to create a process

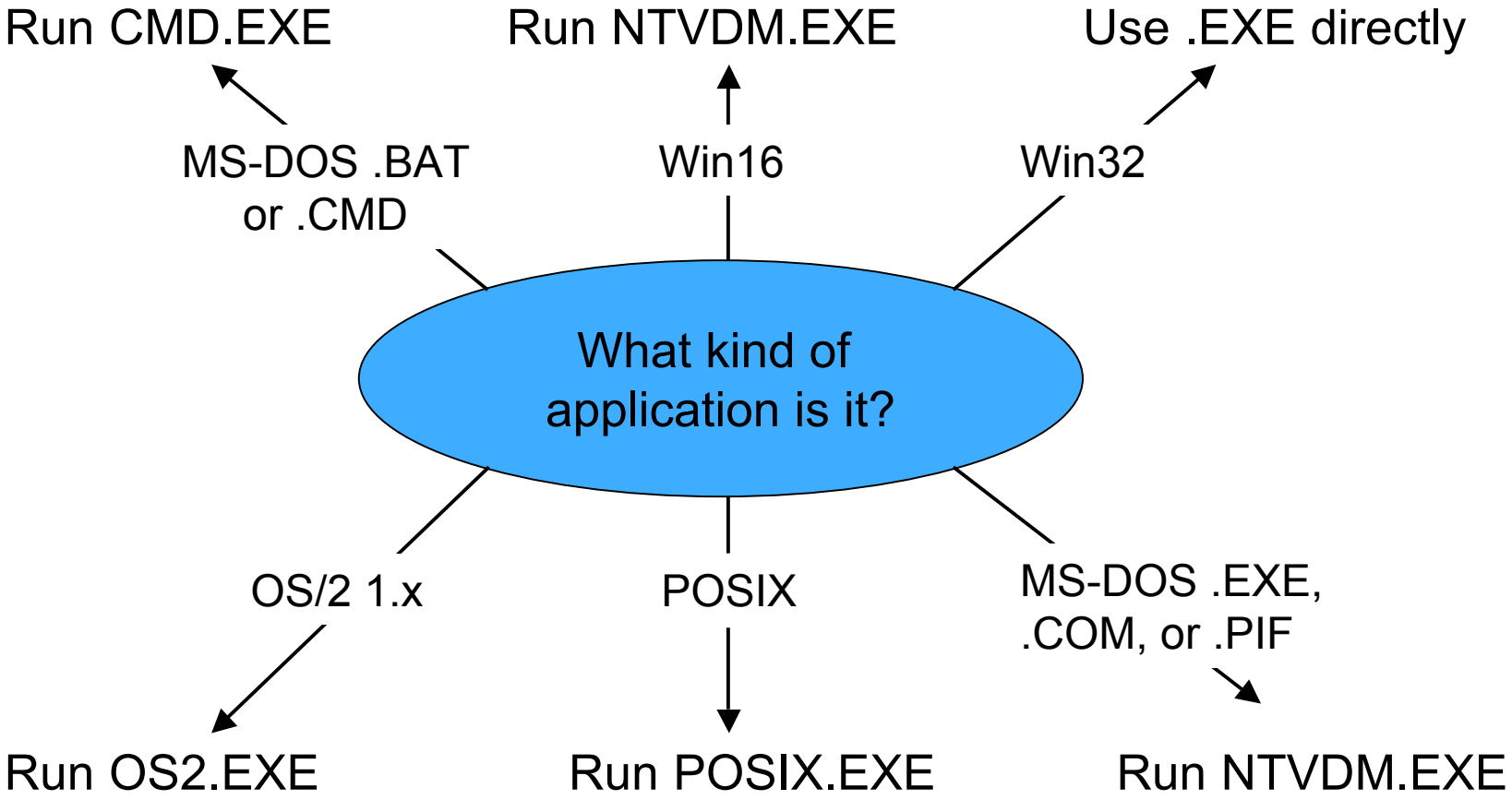


CreateProcess: some notes

- CreationFlags: independent bits for priority class
  -> NT assigns lowest-prio class set
- Default prio class is normal
  unless creator has prio class idle
- If real-time prio class is specified and
  creator has insufficient privileges:
  prio class high is used
- Caller‘s current desktop is used
  if no desktop is specified

Priority classes:
- Real-time
- High
- Normal
- idle

Opening the image to be executed



If executable has no Win32 format...

- CreateProcess uses Win32 „support image“
- No way to create non-Win32 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

Process Creation - next Steps...

- CreateProcess has opened Win32 executable and created a section object to map in proc‘s addr space

Now: create executive proc obj 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

Further Steps...(contd.)

- Create Initial Thread and Its Stack and Context
  - NtCreateThread; new thread is suspended until CreateProcess returns
- Notify Win32 Subsystem about new process

  KERNEL32.DLL sends message to Win32 subsystem including:
  - Process and thread handles
  - Entries in creation flags
  - ID of process‘s creator
  - Flag describing Win32 app (CSRSS may show startup cursor)
- Win32: duplicate handles (inc usage count), set prio 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
    (Win32 creates CREATE_PROCESS_DEBUG_INFO event)
- Image begins execution in user-mode (return from trap)

Thread Internals
Fibers vs. Threads

- NT Threads are scheduled by the kernel
- Kernel mode/user mode threads

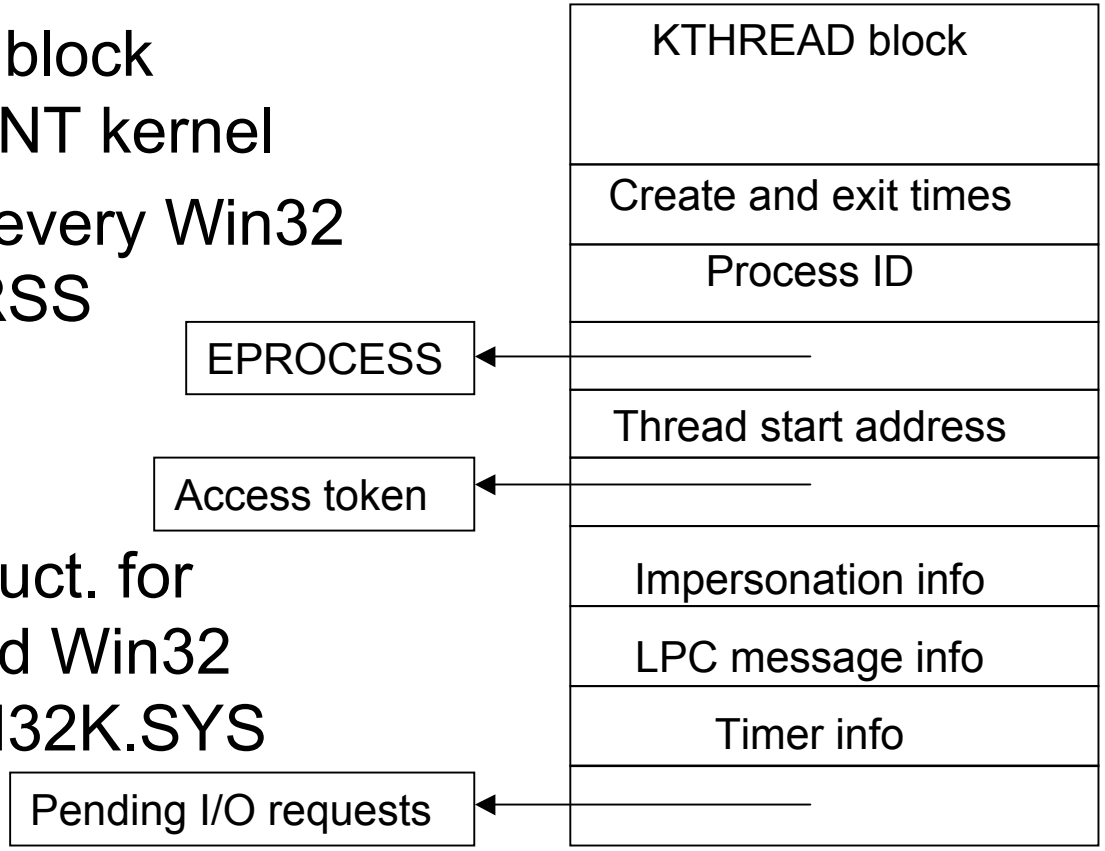
- Many Unix thread implementations are user-space
- NT 3.51/SP3 has introduced Fibers (lightweight threads)
  - Simplify porting of multithreaded Unix apps
  - Programmer schedules Fibers manually (co-operative)
  - Fibers are not scheduled by the system

    ConvertThreadToFiber

    CreateFiber, SwitchToFiber functions

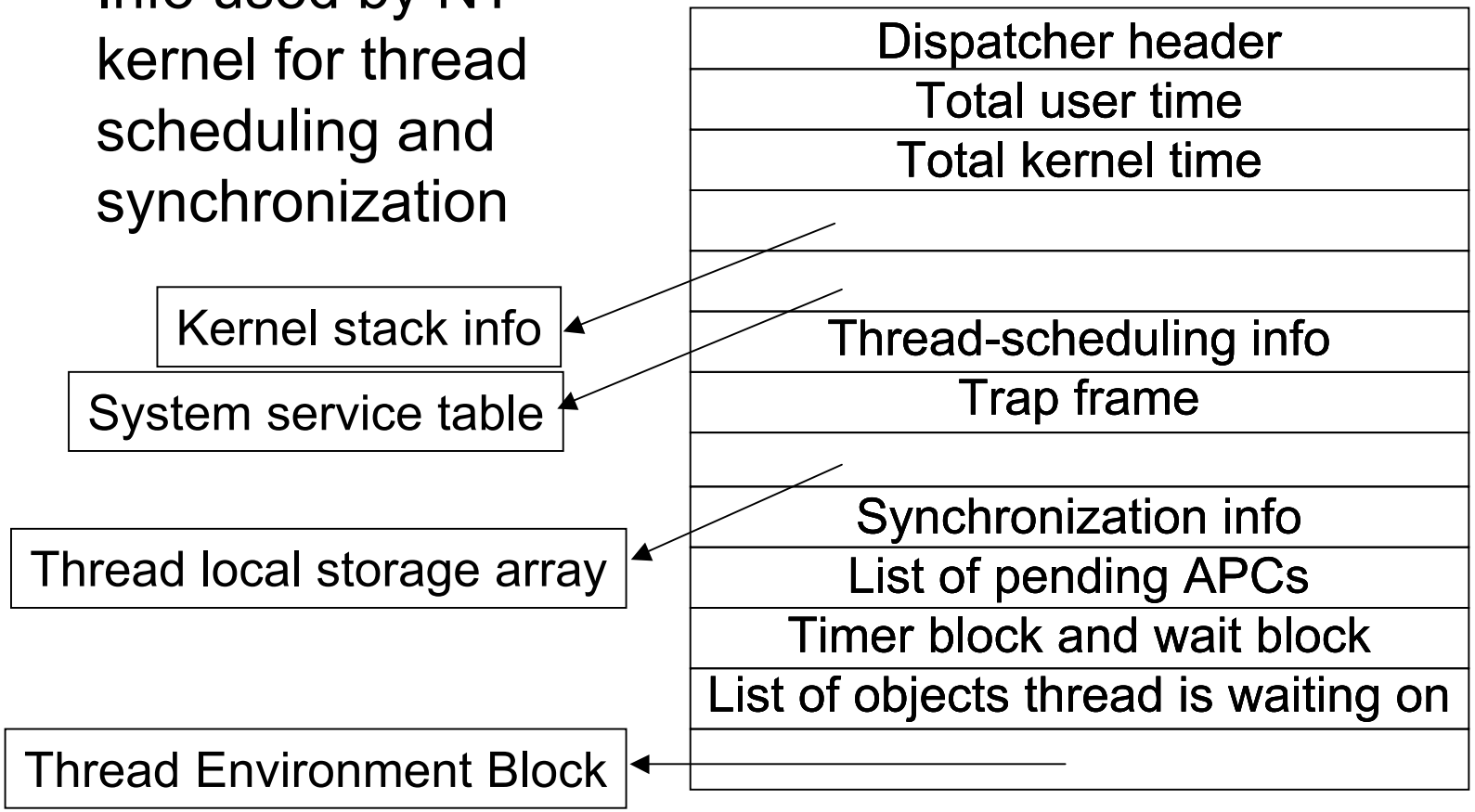
Thread-related Data Structures

- Executive thread block (ETHREAD) – in NT kernel
- Thread block for every Win32 process – in CSRSS
- W32THREAD-struct. for threads that called Win32 subsyst. – in WIN32K.SYS



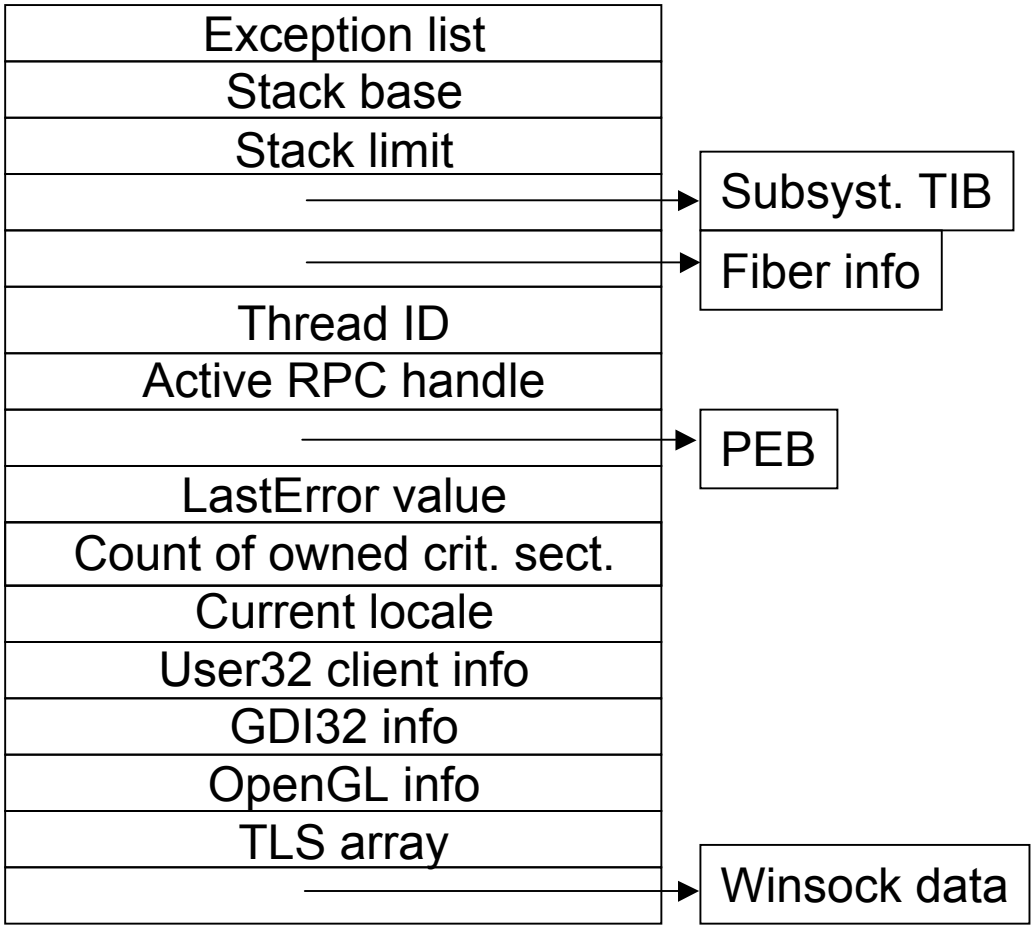
Kernel Thread Block

- Info used by NT kernel for thread scheduling and synchronization



Thread Environment Block

- User mode data structure
- Context for image loader and various Win32 DLLs



Thread-Related System Variables

Variable	Type	Description
PspCreateThread-NotifyRoutine	Array of 32-bit pointers	Pointers to routines to be called on thread creation and deletion (max. 8)
PspCreateThread-NotifyRoutineCount	DWORD	Count of registered thread notification routines

Thread-Related Performance Counters

Object: Counter	Function
Process: Priority Base	Base priority of process: starting priority for thread within process
Thread:%PrivilegedTime	Percentage of time that the thread was run in kernel mode
Thread:%ProcessorTime	Percentage of CPU time that the threads has used during specified interval %PrivilegedTime + %UserTime
Thread:%UserTime	Percentage of time that the thread has run in user mode
Thread: ElapsedTime	Total lifetime of process in seconds
Thread: ID Process	PID – process IDs are re-used
Thread: ID Thread	Thread ID – re-used

Thread-Related Performance Counters (contd.)

Object: Counter	Function
Thread: Priority Base	Base priority of thread: may differ from the thread‘s starting priority
Thread: Priority Current	The thread‘s current dynamic priority
Thread: Start Address	The thread‘s starting virtual address (the same for most threads)
Thread: Thread State	Value from 0 through 7 – current state of thread
Thread: Thread Wait Reason	Value from 0 through 19 – reason why the thread is in wait state

Thread-Related Functions

Function	Description
CreateThread	Create new thread
CreateRemoteThread	Creates thread in another process
ExitThread	Ends execution of a thread normally
TerminateThread	Terminates a thread
GetExitCodeThread	Gets another thread‘s exit code
GetThreadTimes	Returns another thread‘s timing information
Get/SetThreadContext	Returns or changes a thread‘s CPU registers
GetThreadSelectorEntry	Returns another thread‘s descriptor table entry (only on X86 systems)

Thread Startup (in-context thread init.)