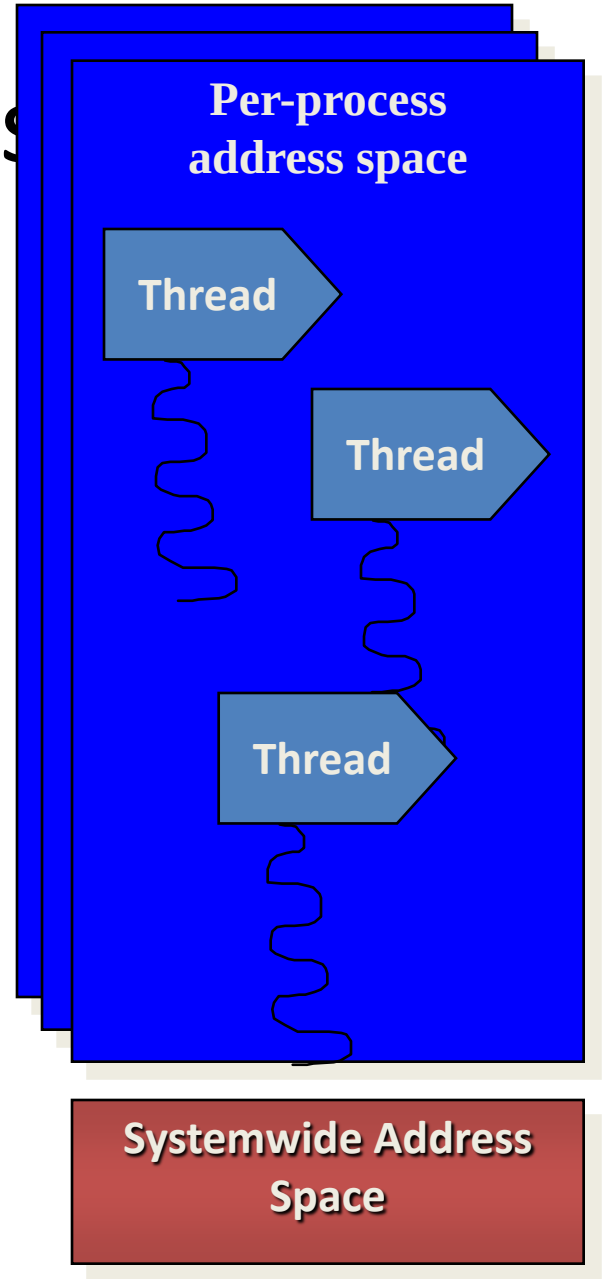
Programování ve Windows
Cvičení 2 – Aplikace a Procesy

Andrea Číková

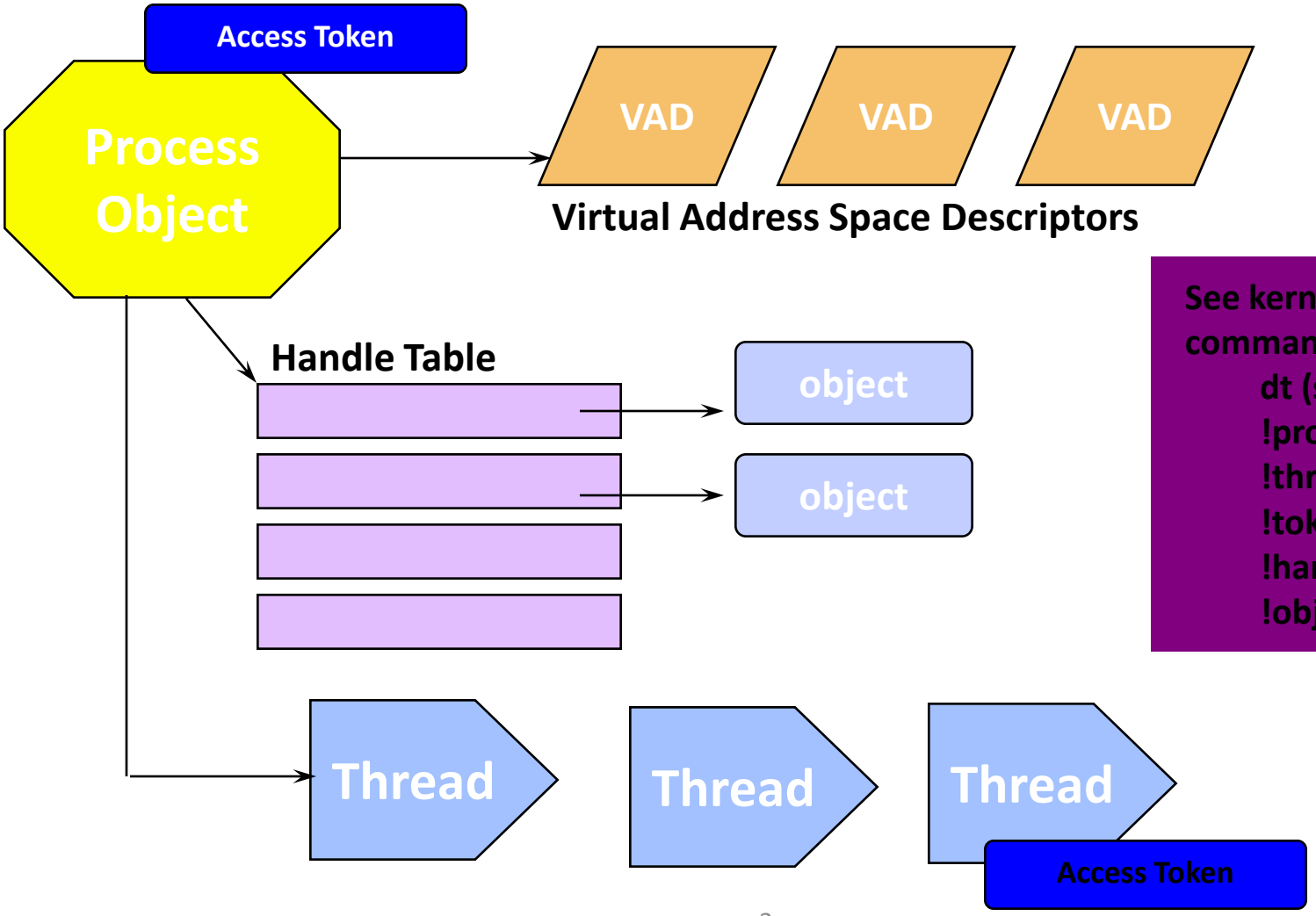
Martin Osovský

Windows Process

- What is a process?
  - Represents an instance of a running program
    - you create a process to run a program
    - starting an application creates a process
  - Process defined by:
    - Address space
    - Resources (e.g. open handles)
    - Security profile (token)
- Every process starts with one thread
  - First thread executes the program's "main" function
    - Can create other threads in the same process
    - Can create additional processes



Processes & Threads
Internal Data Structures



See kernel debugger commands:

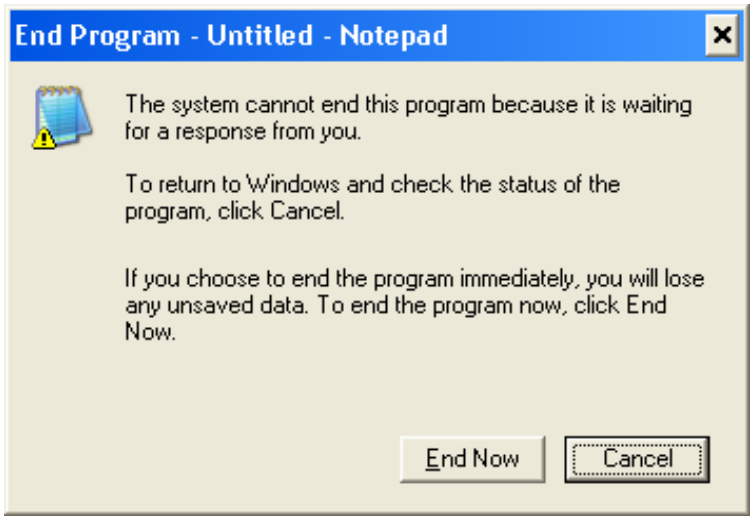
- dt (see next slide)
- !process
- !thread
- !token
- !handle
- !object

Per-Process Data

- Each process has its own…
  - Virtual address space (including program code, global storage, heap storage, threads' stacks)
  - processes cannot corrupt each other's address space by mistake
  - Working set (physical memory "owned" by the process)
  - Access token (includes security identifiers)
  - Handle table for Windows kernel objects
  - Environment strings
  - Command line
  - These are common to all threads in the process, but separate and protected between processes

Why Do Processes Exit?
(or Terminate?)

- Normal: Application decides to exit (ExitProcess)
  - Usually due to a request from the UI
  - or: C RTL does ExitProcess when primary thread function (main, WinMain, etc.) returns to caller
    - this forces TerminateThread on the process's remaining threads
    - or, any thread in the process can do an explicit ExitProcess



- Orderly exit requested from the desktop (ExitProcess)
  - e.g. "End Task" from Task Manager "Tasks" tab
  - Task Manager sends a WM_CLOSE message to the window's message loop…
  - …which should do an ExitProcess (or equivalent) on itself
- Forced termination (TerminateProcess)
  - if no response to "End Task" in five seconds, Task Manager presents End Program dialog (which does a TerminateProcess)
  - or: "End Process" from Task Manager Processes tab
- Unhandled exception
  - Covered in Unit 4.3 (Process and Thread Internals)

Process APIs

- CreateProcess
- OpenProcess
- GetCurrentProcessId - returns a global ID
- GetCurrentProcess - returns a handle
- ExitProcess
- TerminateProcess - no DLL notification
- Get/SetProcessShutdownParameters
- GetExitCodeProcess
- GetProcessTimes
- GetStartupInfo

Process Creation

- No parent/child relation in Win32
- *CreateProcess()* – new process with primary thread

```
BOOL CreateProcess(
   LPCSTR lpApplicationName,
   LPSTR lpCommandLine,
   LPSECURITY_ATTRIBUTES lpProcessAttributes,
   LPSECURITY_ATTRIBUTES lpThreadAttributes,
   BOOL bInheritHandles,
   DWORD dwCreationFlags,
   LPVOID lpEnvironment,
   LPCSTR lpCurrentDirectory,
   LPSTARTUPINFO lpStartupInfo,
   LPPROCESS_INFORMATION lpProcessInformation)
```

Parameters

- fdwCreate:
  - CREATE_SUSPENDED, DETACHED_PROCESS, CREATE_NEW_CONSOLE, CREATE_NEW_PROCESS_GROUP
- lpStartupInfo:
  - Main window appearance
  - Parent‘s info: GetStartupInfo
  - hStdIn, hStdOut, hStdErr fields for I/O redirection
- lpProcessInformation:
  - Ptr to handle & ID of new proc/thread

```
typedef struct _PROCESS_INFORMATION {
        HANDLE hProcess;
        HANDLE hThread;
        DWORD dwProcessId;
        DWORD dwThreadId;
} PROCESS_INFORMATION;
```

UNIX & Win32 comparison

- Windows API has no equivalent to fork()
- CreateProcess() similar to fork()/exec()
- UNIX $PATH vs. lpCommandLine argument
  - Win32 searches in dir of curr. Proc. Image; in curr. Dir.; in Windows system dir. (GetSystemDirectory); in Windows dir. (GetWindowsDirectory); in dir. Given in PATH
- Windows API has no parent/child relations for processes
- No UNIX process groups in Windows API
  - Limited form: group = processes to receive a console event