

Fallstudie: Windows 2000

Geschichte, Programmierung,
Systemstruktur, Prozesse und
Threads, Speicherverwaltung, Ein-
/Ausgabe

Die Geschichte

- 1981: Microsoft programmiert MS-DOS für IBM 8088
 - Viele Ähnlichkeiten mit UNIX
 - Es bekam viele Neuerungen blieb aber ein Kommandozeilenorientiertes Betriebssystem
- 1985: Angeregt durch Lisa von Apple entwickelt MS Windows
 - Erst ab Windows 3.0 wirft Windows Gewinne ab
 - Windows 95 wird das erste richtige Windows Betriebssystem ist aber noch von DOS 7.0 abhängig
 - Win 98 war prinzipiell nur ein 95 Update, da es noch DOS hatte und kein komplettes 32 Bit-System war

- Im Jahr 2000 kam Windows ME (Millennium Edition) es war Win 98 mit installierten Updates
- Größtes Problem: mehrere Prozesse greifen schreibend auf den selben Speicher zu; ein Fehler kann Daten aller Prozesse zerstören
- 1993: Windows NT 3.1 (New Technology)
 - Microsoft wollte ein komplettes 32-Bit System schaffen
 - Benutzer wollten jedoch nicht auf die DOS Programme verzichten
- 96 Upgrade auf NT 4.0
- Kein PnP und Power Management - Dafür Stabiler und Sicherer durch Benutzerfreigaben

Abbildung 11.1 Einige Unterschiede zwischen Windows 98 und Windows NT.

Fähigkeit	Windows 95/98	Windows NT
Reines 32-Bit-System	Nein	Ja
Security?	Nein	Ja
Sichere Dateiverwaltung?	Nein	Ja
Eigener Adressraum pro MS-DOS-Programm?	Nein	Ja
Unicode-fähig?	Nein	Ja
Läuft auf	Intel 80x86	80x86, Alpha, MIPS, ...
Multiprozessorunterstützung?	Nein	Ja
Re-entrant Code im Kern?	Nein	Ja
Plug-and-Play?	Ja	Nein
Power Management?	Ja	Nein
FAT-32-Dateisystem?	Ja	Optional
NTFS-Dateisystem?	Nein	Ja
Win32 API?	Ja	Ja
Laufen alte MS-DOS-Programme?	Ja	Nein
Kritische Bereiche vom Benutzer ungeschützt?	Ja	Nein

- 2000: Windows 2000 wird veröffentlicht
 - Eigentlich Windows NT 5.0
 - Echtes 32-Bit-System für bis zu 32 CPUs
 - Betriebssystem- und Benutzerraum sind getrennt
 - Nun auch PnP, USB, FireWire, IrDA und mehr...
 - Active Directory, Kerberos und Systemüberwachung
 - 4 Versionen: Professional (4GB,2), Server(4GB,4), Advanced Server(8GB,8) und Datacenter Server(64GB,32) – (max. Mem, CPUs)
 - Ca. 30 Millionen Zeilen C-Code
 - Jeden Tag arbeiten hunderte Programmierer für 2000
 - Jeden Tag gibt es eine neue Build-Version

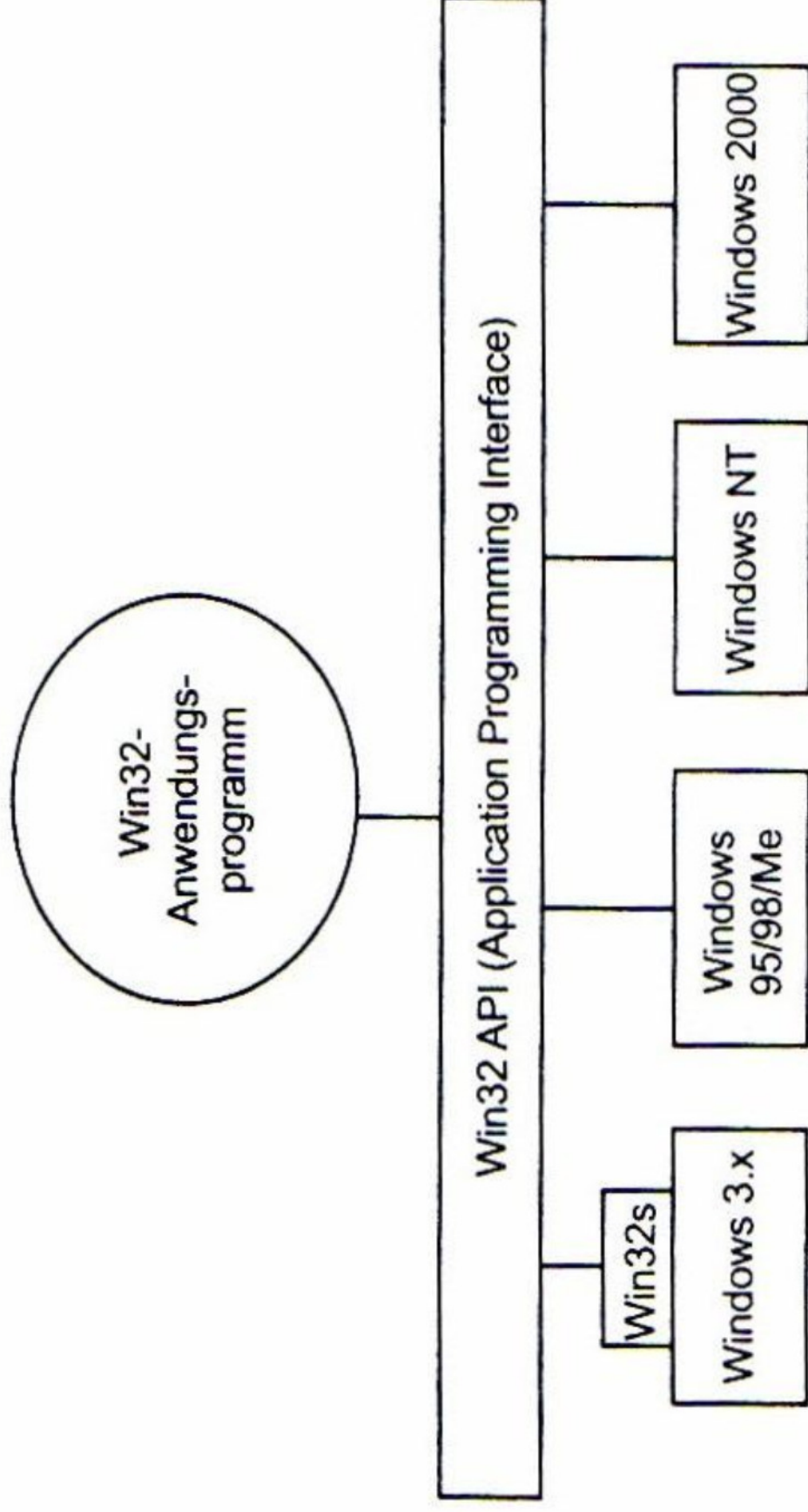
Abbildung 11.3 Ein Vergleich der Größen einiger Betriebssysteme. Der erste Ausdruck in jedem Feld ist die Version; der zweite ist die Größe gemessen in Zeilen Code, wobei K = 1000 und M = 1 000 000. Vergleiche innerhalb einer Spalte sind aussagekräftig; andere Vergleiche nicht, wie im Text erläutert.

Jahr	AT&T	BSD	MINIX	Linux	Solaris	Win NT
1976	V6 9K					
1979	V7 21K					
1980		4.1 38K				
1982	Sys III 58K					
1984		4.2 98K				
1986		4.3 179K				
1987	SVR3 92K		1.0 13K			
1989	SVR4 280K					
1991				0.01 10K		
1993		Free 1.0 235K			5.3 850K	3.1 6M
1994		4.4 Lite 743K		1.0 165K		3.5 10M
1996				2.0 470K		4.0 16M
1997			2.0 62K		5.6 1.4M	
1999				2.2 1M		
2000		Free 4.0 1.4M			5.8 2.0M	2000 29M

Windows-2000-Programmierung

- Programmierschnittstelle Win32 (API)
 - API (Application Programming Interface)
 - Befehlsaufrufe, die immer wieder aktualisiert werden
 - Mit diesen Befehlen kann man das System ansteuern
 - So werden auch Objekte erzeugt (Handles), auf die man zugreifen kann anstatt direkt auf die Ressource
 - Die Schnittstelle Win32 bietet Aufrufe für: Prozesse, Threads, Synchronisation, Speicher-, Datei-, und E/A-Verwaltung, Sicherheit und die graphische Schnittstelle
 - Besonders der GUI-Teil der API ist immens groß
 - Einige API Aufrufe ´sind bei allen Windowssystemen gleich

Abbildung 11.4 Die Win32-Plattform ermöglicht es Programmen, auf nahezu allen Windows-Versionen zu laufen.



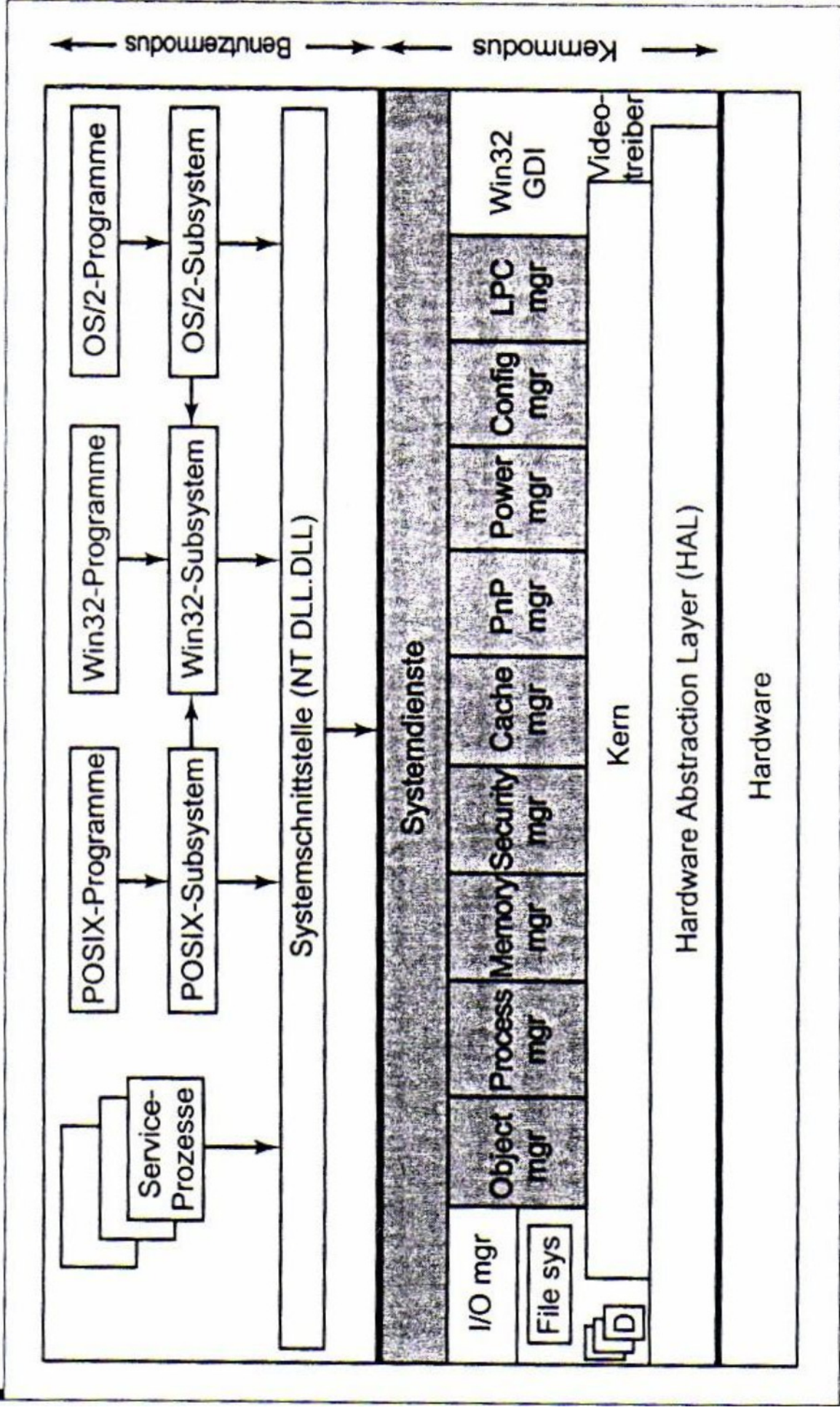
- Die Registrierung (Registry)
 - Früher waren alle Systeminformationen in .ini Dateien
 - Aufgebaut wie ein Dateisystem, jedoch mit Values
 - Außer den Schlüsseln `HARDWARE` und `PERFORMANCE_DATA` werden alle Schlüssel auf der Festplatte gespeichert. `HARDWARE` wird per PnP beim Booten generiert und `PERFORMANCE_DATA` zeigt alle Zähler und Timer im System an.
 - Alle wichtigen Informationen werden gespeichert und jeder Hersteller kann seinen eigenen Subkey ablegen.
 - Alle Dienste und alle Einstellungen werden hier gespeichert
 - Die Schlüssel `HKEY_CLASSES_ROOT`, `HKEY_CURRENT_CONFIG` und `HKEY_CURRENT_USER` sind nur Verweise auf andere Schlüssel der Registrierung.

- In HKEY_CLASSES_ROOT stehen die COM-Objekte (Component Object Model), wo Dateieendungen mit Programmen verknüpft sind.
- Die Registrierung kann von einem Programmierer vollständig genutzt werden, durch API- Aufrufe
- Beim Herunterfahren wird die Registrierung in Dateien gespeichert, die Hives genannt werden. Bei einer Veränderung der Registrierung werden Sicherheitskopien gemacht und beim Kopieren werden atomare Transaktionen durchgeführt.

Abbildung 11.5 Die Root Keys der Registrierung und ausgewählte Unterschlüssel. Die Groß- und Kleinschreibung hat keinen Einfluss, wir folgen hier der Microsoft-Konvention.

Schlüssel	Beschreibung
HKEY_LOCAL_MACHINE HARDWARE SAM SECURITY SOFTWARE SYSTEM	Eigenschaften von Hard- und Software Gerätebeschreibungen und Zuordnung von Hardware zu Treibern Security und Zugangsinformationen für Benutzer Systemweite Sicherheitsrichtlinien Informationen über installierte Anwendungsprogramme Informationen zum Systemstart
HKEY_USERS USER-AST-ID AppEvents Console Control Panel Environment KeyboardLayout Printers Software	Informationen über jeden Benutzer; ein Schlüssel pro Benutzer Einstellungen für Benutzer AST Geräusche bei Ereignissen (neue Mail, Fehler, ...) Einstellungen für die Kommandozeile (Farbe, Schrift, ...) Desktop-Erscheinungsbild, Bildschirmschoner, Mauseinstellungen, ... Umgebungsvariablen Welche Tastatur: 102-Tasten US, QWERTZ, Dvorak etc. Informationen über installierte Drucker Benutzereinstellungen für Microsoft und andere Software
HKEY_PERFORMANCE_DATA	Hunderte von Performancezählern des Systems
HKEY_CLASSES_ROOT	Link zu HKEY_LOCAL_MACHINE\SOFTWARE\CLASSES
HKEY_CURRENT_CONFIG	Link zum aktuellen Hardwareprofil
HKEY_CURRENT_USER	Link zum aktuellen Benutzerprofil

Abbildung 11.7 Die Windows-2000-Struktur (etwas vereinfacht). Der schattierte Bereich ist die Executive. Die mit D bezeichneten Rechtecke sind die Gerätetreiber. Die Dienstprozesse sind die Systemdämonen.



Systemstruktur

- Struktur des Betriebssystems
 - Das System ist in mehrere Bereiche aufgeteilt, die größtenteils in C geschrieben sind.
- I. Die Hardware Abstraction Layer (HAL)
 - Die Maschinenabhängigen Befehle werden in dieser Schicht gespeichert und verarbeitet.
 - Das System muss nicht direkt auf die Hardware zugreifen
 - HAL soll den Unterschied von zwei Chips verstecken
 - Es werden die höheren Schichten vor Busstrukturen und ähnlichem geschützt.

- Es werden Dienste für DMA, Uhren, Synchronisation und Treiber zur Verfügung gestellt.
- Die HAL baut auch nach dem Booten die Kommunikation mit dem BIOS auf um zu erfahren, welche Komponenten im System sind.
- Bei der Installation wird die richtige HAL als hal.dll auf dem System gespeichert.
- DirectX ist eine Komponente, die die HAL bei Multimedia Anwendungen unterstützt, um sie schneller zu machen.

II. Die Betriebssystem-Ebene

- Hier liegt der „Kern“ des Betriebssystems
- Der Kern setzt dort an wo die HAL aufgehört hat.

- Hier wird das Wechseln von Threads, speichern von CPU-Registern, löschen des CPU-Cache usw. gemacht.
- Alle Eingaben der Hardware werden hier analysiert und Unterbrechungsroutrinen werden eingeleitet.
- Alle Prozesse enthalten den Kern in ihrem Adressraum
- Ereignisse und andere Teile des Thread-Management sind auch im Kern verwoben.

III. Die Executive

- Besteht aus 10 Komponenten, jede Komponente ist eine Sammlung aus Prozeduren
- Die Komponenten können sich untereinander aufrufen und tun dies auch oft.
- Ist in der Datei: ntoskrnl.exe

1. Der Objekt-Manager

- Verwaltet alle Objekte, die dem System bekannt sind z.B. Threads, Prozesse, Dateien, Geräte...
- Stellt die Beziehungen zwischen den Objekten her.

2. Der E/A-Manager

- Heimat aller Gerätetreiber
- Ruft den richtigen Treiber für das Gerät auf
- Auch Dateisysteme werden hier verwaltet. FAT und NTFS sind unterschiedliche Treiber.

3. Der Prozess-Manager

- Behandelt Prozesse und Threads
- Erzeugt und Vernichtet Prozesse und Threads
- Schlüssel zum Multitasking

4. Die Speicherverwaltung

- Überwachung der Schutzregeln
- Jeder Prozess darf nur auf seine eigenen Speicherseiten zugreifen
- Behandelt Systemaufrufe für den virtuellen Speicher

5. Der Security-Manager

- Erfüllt den `U.S. Dept. Of Defense Orange Book C2 Standard`
- Behandelt von der authentifizierte Anmeldung über die Zugriffskontrolle bis hin zur Tatsache, dass virtuelle Seiten mit Nullen gefüllt werden müssen, ehe sie benutzt werden dürfen.

6. Der Cache-Manager

- Verwaltung der Blöcke im Cache
- Zugriffsbeschleunigung auf die Blöcke
- Benötigte Blöcke aus Dateisystemen liefern
- Muss immer einen Überblick über alle Blöcke haben

7. Der Plug-and-Play-Manager

- Erhält alle Benachrichtigungen über neue Hardware
- Kümmt sich beim Booten um alle Geräte
- Beim laufenden System nur um USB etc.
- Sucht und installiert passende Treiber
- Treiber sind meistens in den .sys Dateien

8. Der Power-Manager

- Herrscher der Energieverwaltung
- Überwacht die Nutzung und das Abschalten von Geräten wie z.B. den Monitor
- Bei leerer Batterie fordert er Programme auf ihre Daten zu speichern und herunterzufahren

9. Der Konfigurations-Manager

- Kümmt sich um die Registrierung
- Erzeugt und löscht Einträge

10. Der Local-Procedure-Call-Manager

- Bietet schnelle Interprozesskommunikation
- Ist auf Effizienz getrimmt

- Das Win32-GDI-Ausführungsmodul
 - GDI (Graphics Device Interface)
 - Behandelt Bilddaten für Monitor und Drucker
 - Zur Leistungssteigerung im Kern
 - Ist größer als der Rest der Executive
 - Ist in der Datei: win32k.sys
- Die Systemdienste
 - Schnittstelle zur Executive
 - Empfängt API-Befehle und gibt diese an die Executive zur Ausführung weiter
- Die Checked Builds
 - Sie führen bei jedem Treiber einen Plausibilitätstest durch, was das System bremst
 - Hilfe für Autoren von Treibern

IV. Die Gerätetreiber

- Kann ein oder mehrere E/A-Geräte steuern oder nur Prozeduren zur Verfügung stellen
- Gefährlichster Treiber ist der Grafiktreiber, da er direkt auf den Kern einwirken kann
- Treiber werden in eigenen Dateien abgespeichert um so die ntoskrnl.exe nicht zu verändern

Abbildung 11.9 Die Struktur eines Objekts.

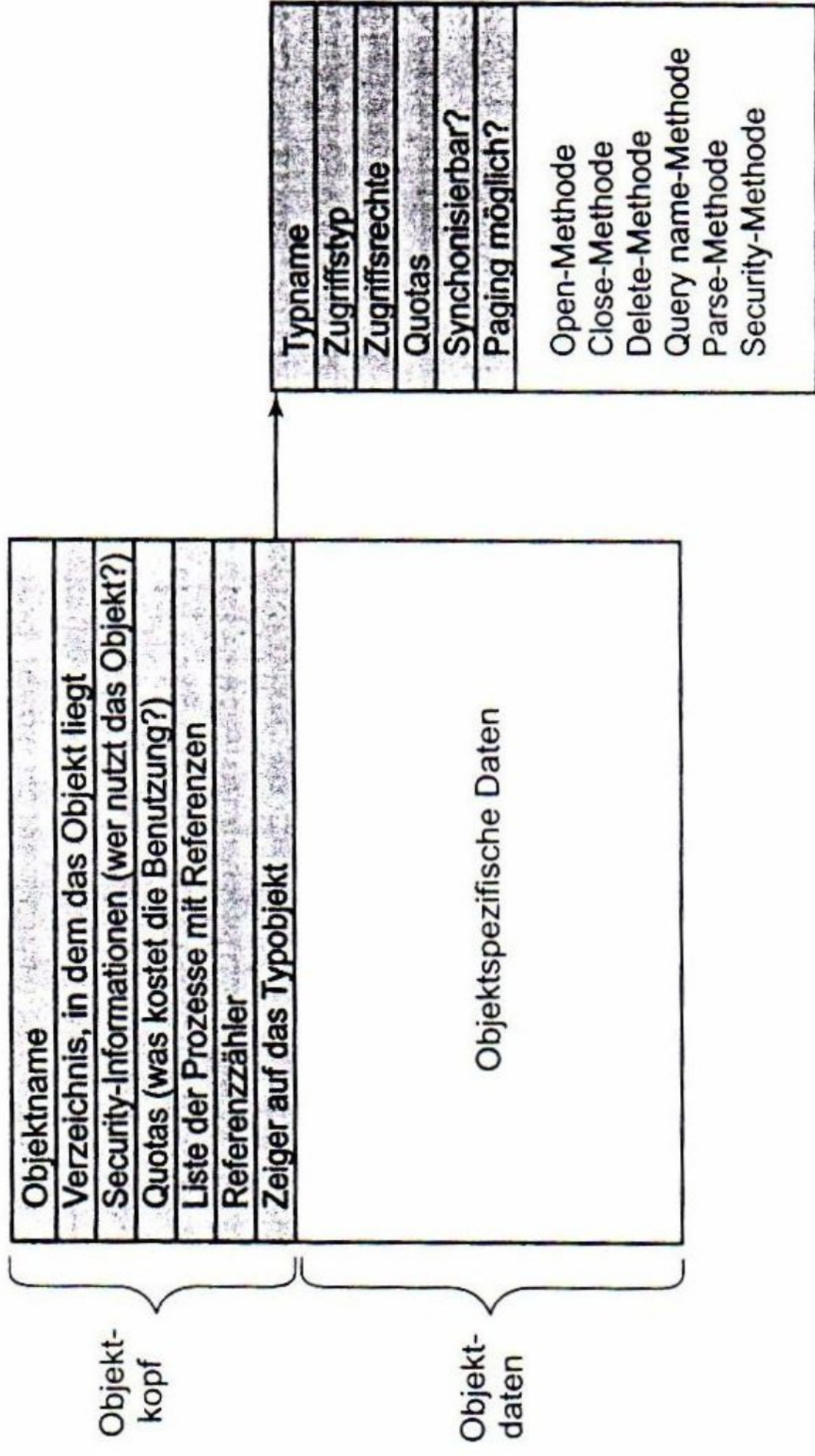


Abbildung 11.10 Einige allgemeine Objekttypen der Executive, die der Objekt-Manager verwaltet.

Typ	Beschreibung
Prozess	Benutzerprozess
Thread	Thread innerhalb eines Prozesses
Semaphor	Zählsemaphor für Interprozesskommunikation
Mutex	Binäres Semaphor für kritische Bereiche
Ereignis	Synchronisationsobjekt mit persistentem Zustand
Port	Mechanismus für Prozessnachrichtenaustausch
Uhr	Objekt, damit Threads für eine gewisse Zeit schlafen können
Queue	Objekt für Benachrichtigung bei asynchroner Kommunikation
Offene Datei	Objekt für eine offene Datei
Access Token	Security-Bezeichner für ein Objekt
Profil	Datenstruktur für Messung der CPU-Belastung
Sektion	Struktur für das Einblenden von Dateien in den virtuellen Speicher
Schlüssel	Schlüssel in der Registry
Objektverzeichnis	Verzeichnis zur Gruppierung im Objektmanager
Symbolischer Link	Zeiger auf ein anderes Objekt durch den Namen
Gerät	Ein-/Ausgabe-Geräteobjekt
Gerätetreiber	Jeder geladene Gerätetreiber besitzt sein eigenes Objekt

Abbildung 11.12 Einige typische Verzeichnisse im Objektnamensraum.

Verzeichnis	Inhalt
??	Anfang, um Geräte wie MS-DOS-Platten (C:) zu finden
Device	Alle erkannten Ein-/Ausgabegeräte
Driver	Objekte für jeden geladenen Gerätetreiber
ObjectTypes	Typobjekte wie in Abbildung 11.11 gezeigt
Windows	Objekte, um Nachrichten an alle Fenster zu schicken
BaseNamedObjs	Vom Benutzer erzeugte Objekte wie Semaphoren, Mutexe, ...
Arcname	.. Vom Bootlader erkannte Partitionsnamen
NLS	National-Language-Support-Objekte
FileSystem	Dateisystemtreiber und Objekte zur Dateisystemerkennung
Security	Objekte des Security-Subsystems
KnownDLLs	Wichtige Shared Libraries, die offen gehalten werden

- Realisierung von Objekten
 - Objekte sind das wichtigste Konzept der Windows-Architektur
 - Ein Objekt ist eine Datenstruktur im RAM
 - Alle Objekte werden beim Lauf des Systems erzeugt
 - Nicht mehr gebrauchte Objekte werden aus dem Speicher gelöscht
 - Der Zähler gibt die Anzahl der Prozesse an, die auf das Objekt zugreifen
 - Wichtige Objekte werden fest im Speicher verankert
 - Jedes Objekt hat einen Typ nachdem bestimmte Operationen auf das Objekt ausgeführt werden
 - Es gibt Standardtypen, jedoch kann das System neue Typen dynamisch erschaffen

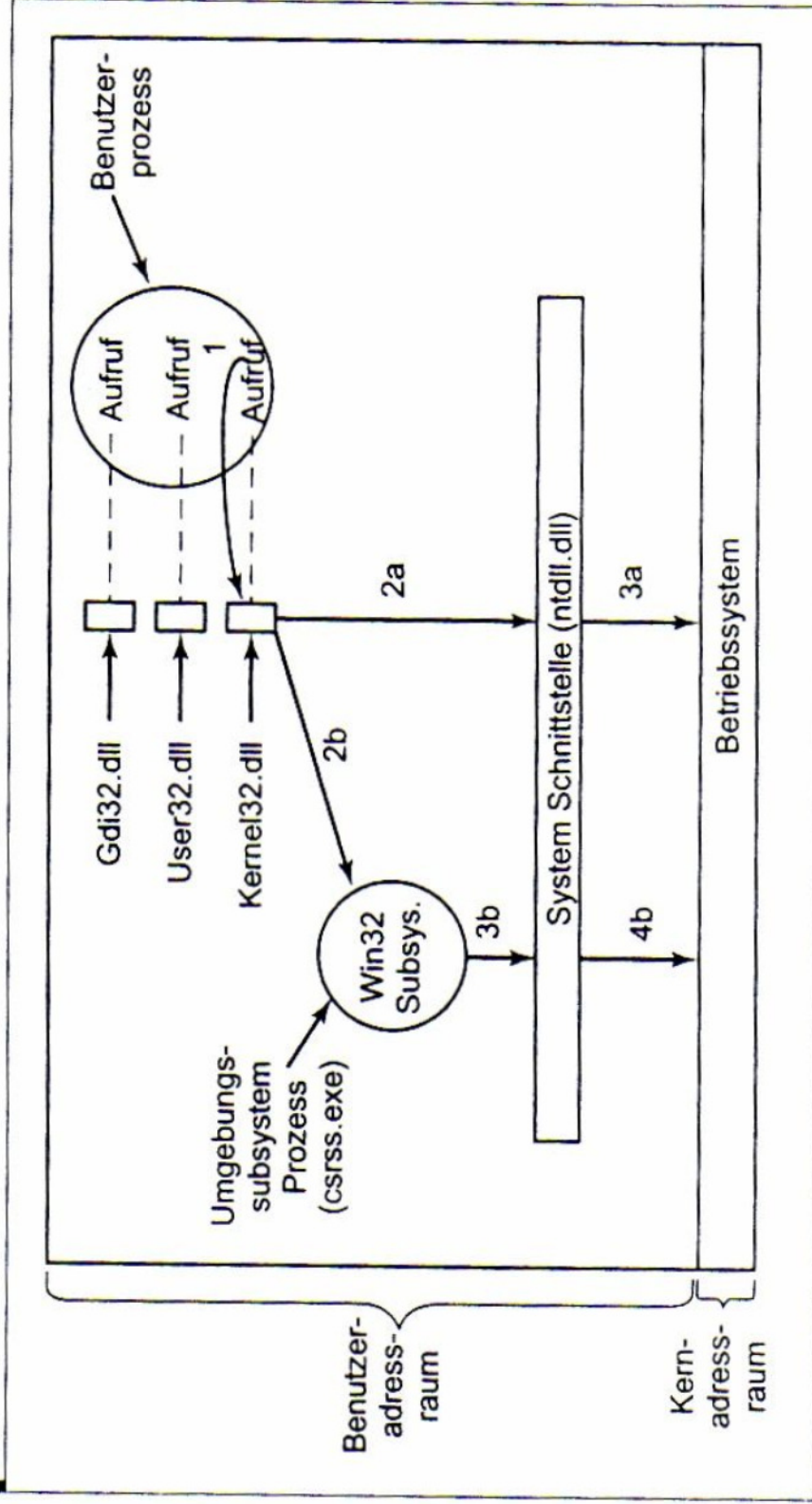
- Nachdem ein Objekt erzeugt wurde erhält der Benutzer die Position des Objekts um darauf zu zureifen zu können
- Ein Objekt kann verschiedene Rechte haben, z.B. read-only

I. Der Namensraum der Objekte

- In den Namensraum werden alle Objekte eingetragen
- Win2000 hat 3 Namensräume für: Objekte, Registrierung und Dateien
- In diesen Räumen schaut man nach, um bestimmte Dinge zu finden, es ist ein Verzeichnis

- Subsysteme für andere Umgebungen
 - Windows unterstützt 3 APIs: Win32, POSIX und OS/2
 - Win32 ist die offizielle Schnittstelle für Windows-Programme
 - Es wird mit DLL (Dynamic Link Libraries) Dateien gearbeitet
 - Mehrere Programme können die gleiche DLL verwenden
 - Es gibt über 800 DLLs, die zusammen über 130MB belegen, die Zahl, der darin enthalten API-Aufrufe ist 13 000
 - Die POSIX-Umgebung ist für minimale UNIX-Unterstützung
 - Bietet nur P1003.1 Funktionalität (Forderung von Regierung)
 - Benutzt größtenteils Win32 Funktionen
 - Für UNIX-Umsteiger bietet Microsoft das Produkt Interix an
 - Das OS/2 Subsystem ist ähnlich beschränkt und ist auch in der Praxis vollkommen nutzlos

Abbildung 11.13 Es wurden verschiedene Wege gewählt, um die Win32-API-Funktionsaufrufe zu implementieren.



Prozesse und Threads in Windows

2000

- Grundlegende Konzepte
 - Es gibt Jobs, Prozesse, Threads und Fiber
 - Job ist einen Zusammenfassung von mehreren Prozessen
 - Jeder Prozess hat einen 4GB großen Adressraum (2GB davon belegt immer das Betriebssystem)
 - Jeder Prozess startet einen Thread
 - Jeder Thread kann 3 Zustände einnehmen (bereit, laufend, blockiert)
 - Ein Thread kann aus dem Benutzermodus in den Kernmodus wechseln, um Systemaufrufe auszuführen

- Ein Thread kann auf alle Objekte seines Prozess zugreifen
- Nach der Ausführung wird ein Thread beendet
- Ein Prozess endet, wenn alle seine Threads beendet sind
- Ein Fiber ist ein leichtgewichtiger Thread
- Diese Fiber werden von Threads erzeugt und werden vom Betriebssystem nicht erkannt und verwaltet
- Da Win2000 Multiprozessor fähig ist müssen alle Prozesse bzw. Threads wiedereintrittsfähig sein
- Win2000 kann maximal 32 CPUs benutzen, da nur 32 Bit Bitmaps zur Adressierung zur Verfügung stehen

Abbildung 11.17 Einige der Win32-Aufrufe zur Verwaltung von Prozessen, Threads und Fibern.

Win32-API-Funktion	Beschreibung
CreateProcess	Erzeuge neuen Prozess
CreateThread	Erzeuge neuen Thread im bestehenden Prozess
CreateFiber	Erzeuge neuen Fiber
ExitProcess	Beende Prozess und alle Threads
ExitThread	Beende diesen Thread
ExitFiber	Beende diesen Fiber
SetPriorityClass	Setze Prioritätenklasse für Prozess
SetThreadPriority	Setze Priorität für einen Thread
CreateSemaphore	Erzeuge neues Semaphore
CreateMutex	Erzeuge neuen Mutex
OpenSemaphore	Öffne bestehendes Semaphore
OpenMutex	Öffne bestehenden Mutex
WaitForSingleObject	Warte auf Semaphore, Mutex etc.
WaitForMultipleObjects	Warte auf eine Menge von Objekten mit geg. Handle
PulseEvent	Setze Ereignis auf signalisiert und dann auf nicht sig.
ReleaseMutex	Gib Mutex frei, damit anderer Thread zugreifen kann
ReleaseSemaphore	Erhöhe Semaphore um 1
EnterCriticalSection	Blockiere Sperre für einen kritischen Bereich
LeaveCriticalSection	Gebe Sperre für kritischen Bereich frei

- API-Aufrufe zum Job, Prozess, Thread und Fiber Management
 - Neue Prozesse werden durch die API-Funktion CreateProzess generiert
 - Diese Funktion hat 10 Parameter (jeder hat viele Optionen)
 - Jeder Prozess ist gleichgestellt mit anderen
 - Ein Prozess kann den Aufruf CreateThread starten
 - Diese Funktion wider rum hat 6 Parameter
 - Da der Kern erzeugt und verwaltet alle Threads
 - Threads können untereinander kommunizieren, mit: Pipes, Named-Pipes, Mailslots, Sockets, Remote-Procedure-Calls und gemeinsam genutzten Dateien
 - Es gibt auch mehrere Werkzeuge zur Synchronisation zwischen Threads (Semaphoren, Mutexe, kritische Bereiche und Ereignisse)

- Es gibt nahezu 100 Win32-API-Aufrufe, die sich mit Prozessen, Threads und Fibern beschäftigen.
- Realisierung von Prozessen und Threads
 - Die Win32-API-Create-Aufrufe ruft eine Prozedur in der kernel32.dll auf:
 1. Erst wird überprüft, ob es eine Win32-Anwendung ist
 2. NTCreatProzess wird ausgeführt und schafft eine leere Prozesshülle
 3. Nun wird NTCreatThread bearbeitet und erstellt den ersten Thread
 4. kernel32.dll gibt nun dem Win32-Subsystem Meldung und alles wird in die Tabellen des Systems eingefügt, außerdem wird der Cursor auf Sanduhr gestellt

5. Jetzt ist der Thread lauffähig und ruft das Laufzeitsystem zur Initialisierung
6. Das Laufzeitsystem setzt die Priorität des Threads fest, teilt den DLLs mit, dass ein neuer Thread gibt und verrichtet andere Arbeiten. Dann startet es den Code des Hauptprogramms.
 - Die Erzeugung eines Threads läuft ähnlich ab
 - Bei beiden Erzeugungen wird ein Handle zurückgeliefert, mit dem man den Prozess oder Thread starten, stoppen, töten oder untersuchen kann.
- Das Scheduling von Threads wird vom Kern übernommen
- Ein aktuelle laufender Thread kann das Scheduling aufrufen, wenn z.B. seine Zeitscheibe abgelaufen ist

- Der Scheduler kann auch noch unter 2 anderen Bedingungen aufgerufen werden: Eine E/A-Operation wird beendet oder eine bestimmte Wartezeit läuft ab.
- Prozesse können an 2 Stellen das Scheduling beeinflussen: Entweder durch die PriorityClass des Prozesses und durch die PriorityClass des einzelnen Thread
- So entstehen 42 Prioritäten für Threads von denen nur 32 vom Betriebssystem erkannt werden
- Die Prioritätenliste wird von oben nach unten abgearbeitet
- Jeder Thread darf eine Zeitscheibe laufen und wird dann an das Ende der Liste gestellt
- Bei mehreren Prozessoren macht jeder Prozessor sein eigenes Scheduling

Abbildung 11.19 Windows 2000 unterstützt 32 Prioritäten für Threads.

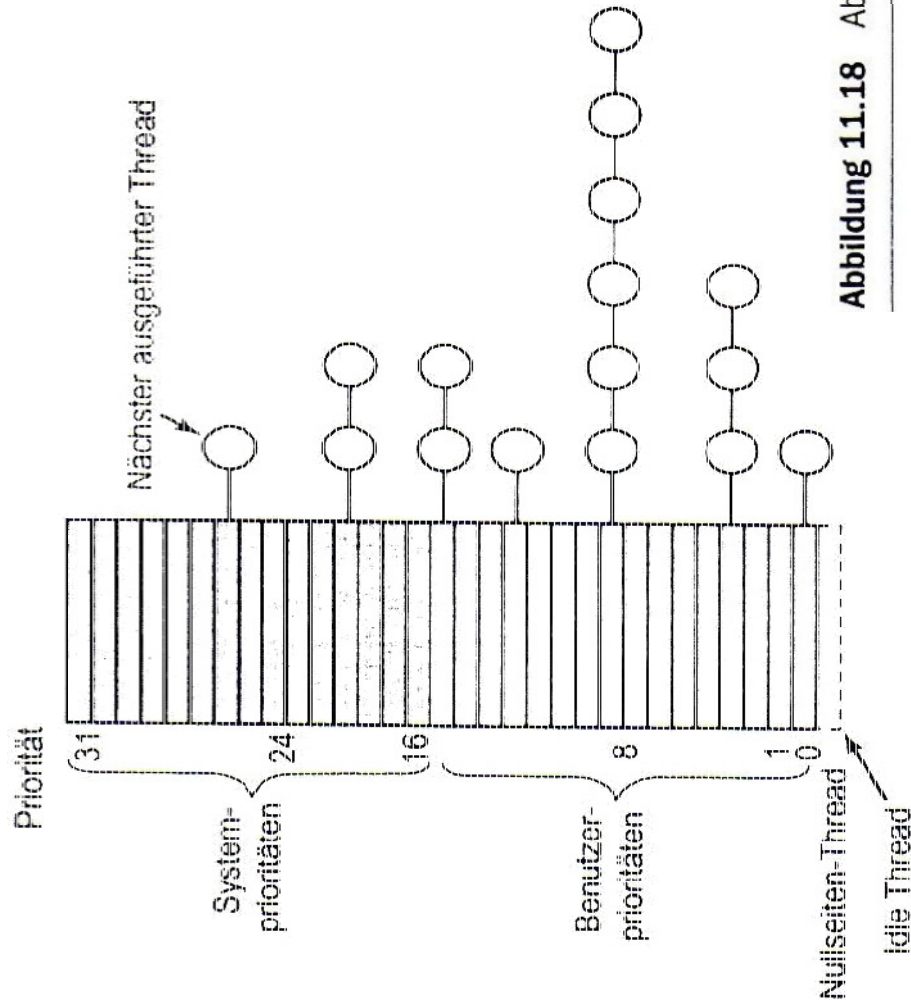


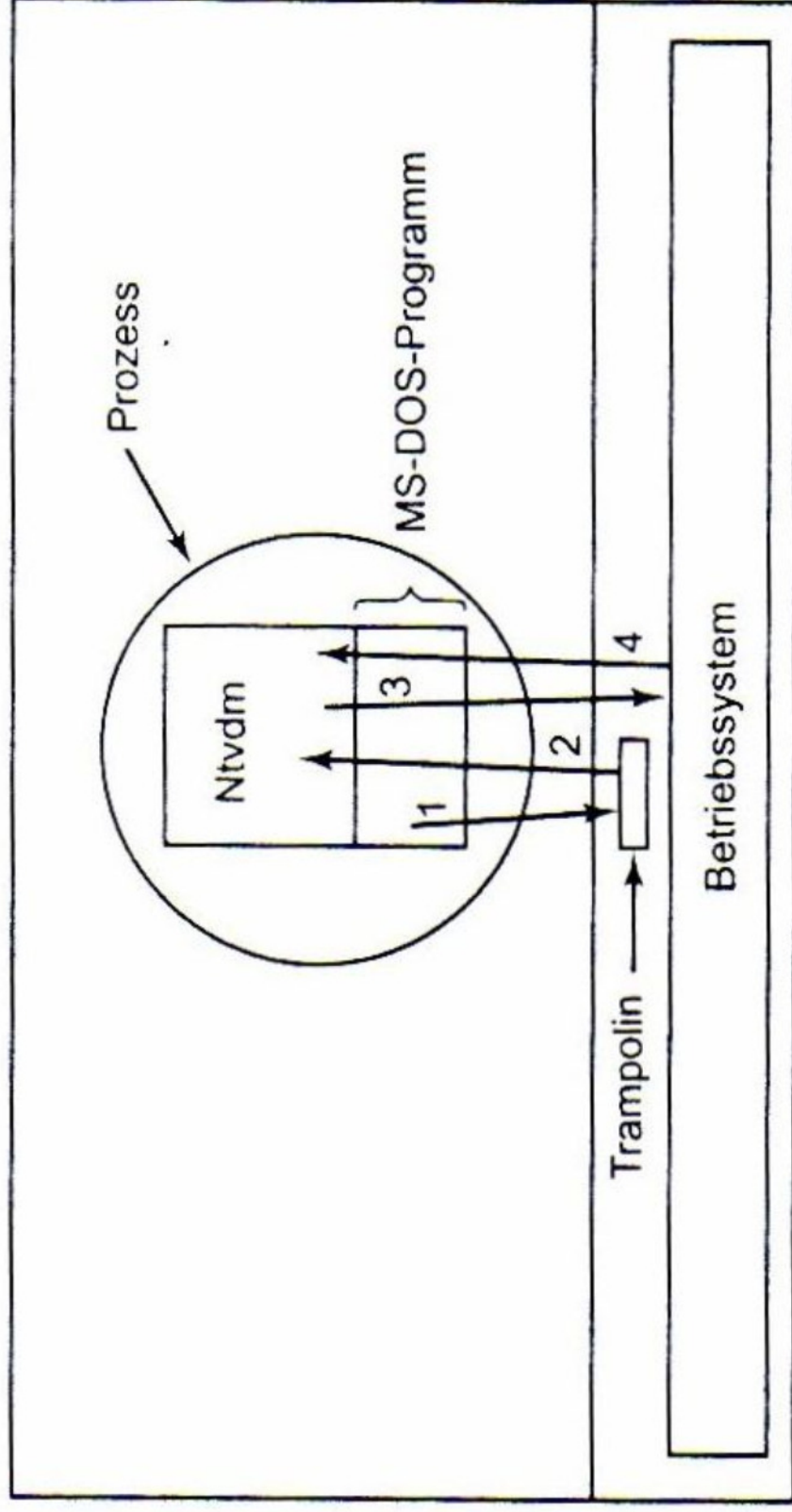
Abbildung 11.18 Abbildung der Win32-Prioritäten auf die Windows-2000-Prioritäten.

		Win32-Prozessklassen-Prioritäten					
Win32-Thread-Prioritäten		Echtzeit	Hoch	Über normal	Normal	Unter normal	Idle
	Zeitkritisch	31	15	15	15	15	15
	Höchste	26	15	12	10	8	6
	Über normal	25	14	11	9	7	5
	Normal	24	13	10	8	6	4
	Unter normal	23	12	9	7	5	3
	Niedrigste	22	11	8	6	4	2
	Idle	16	1	1	1	1	1

- Von 16-31 sind die Prioritäten vom System und vom Admin extra festgelegten Threads
- Benutzerthreads sind im Raum von 0-15
- Der Idle-Thread verbraucht alle CPU-Zeit, die zur Zeit nicht genutzt wird
- Unter bestimmten Umständen kann das System die Priorität eines Threads hoch setzen aber nicht über 15
- Bei bestimmten E/A-Geräten wird die Priorität unterschiedlich erhöht, um die Geräte zu entlasten
- Auch Threads vom Vordergrundprozess werden erhöht
- Diese Erhöhungen gelten jedoch nur für eine Zeitscheibe
- Normalerweise werden Prioritäten langsam heruntergefahren, außer ein Thread hängt fest und ist bereit, dann darf er 2 Scheiben lang mit der höchsten Priorität arbeiten

- Eine Zeitscheibe ist unter Win2000 20ms lang, bei verschiedenen Systemen kann diese Zeit variieren
- MS-DOS-Emulation
 - Erst wird ein Win32-Anwendung gestartet das ntvdm (NT Virtual DOS Machine)
 - Diese Anwendung interpretiert das DOS-Programm
 - Durch ein so genanntes Trampolin bekommen die Aufrufe anstatt zum System zum Emulator
 - Der Emulator untersucht die Aufrufe und verhindert direkten Zugriff auf die Ressourcen des Systems
 - Wenn das Programm Fehler produziert wird es abgerochen

Abbildung 11.21 So werden alte MS-DOS-Programme unter Windows 2000 ausgeführt.



- Starten von Windows 2000
 - Nachdem der Boot-Sektor angesprungen wurde, wird die Datei ntldr in den Speicher geladen und ausgeführt
 - Ntldr liest nun die boot.ini, durch die Informationen aus dieser Datei lädt er die Dateien hal.dll, ntoskrnl.exe und bootvid.dll
 - Jetzt liest ntldr die Registrierung und gibt die Kontrolle an ntoskrnl.exe ab.
 - Nach einer allgemeinen Initialisierung des Betriebssystems wird die Executive aufgerufen
 - Der Balken wächst und der erste Prozess startet: smss.exe (der Session-Manager)
 - Er startet die crss.exe und stellt die Registry zusammen
 - Als letztes erzeugt er den Anmelde Dienst winlogon.exe
 - Jetzt kann der User sich durch die lsass.exe anmelden

- Der richtige Anmeldevorgang wird durch msgina.dll behandelt, um den Vorgang einfacher zu ändern
- Auch der Vaterprozess aller Dienste (service.exe) startet und startet alle Programme gemäß Registrierung
- Letztendlich wird auch die Shell (explorer.exe) gestartet

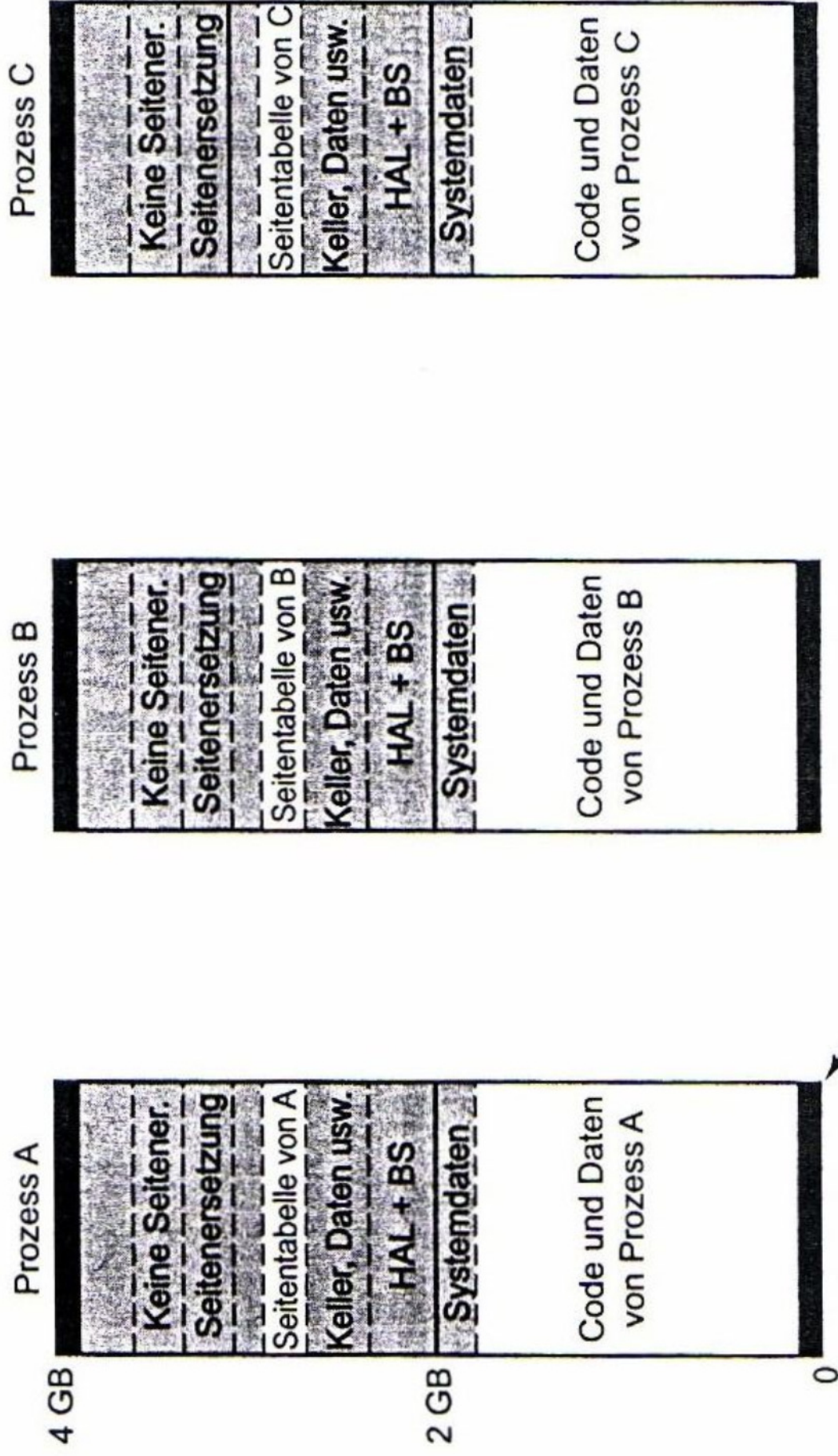
Abbildung 11.22 Prozesse, die während des Hochfahrens gestartet werden. Diejenigen oberhalb der Linie werden immer gestartet. Die unterhalb sind Beispiele für Dienste, die gestartet werden könnten.

Prozess	Beschreibung
idle	Kein echter Prozess, sondern der Idle-Prozess
system	Erzeugt smms.exe und Auslagerungsdatei; lädt DLLs; liest Reg.
smss.exe	Erster Prozess; Initialisierung; erzeugt csrss und winlogon
csrss.exe	Win32-Prozesssubsystem
winlogon.exe	Login-Hintergrundprozess
lsass.exe	Authentifikationsdienst
services.exe	Durchsucht Registry und startet Dienste
Druckserver	Verschiedene Prozesse können den Drucker benutzen
Dateiserver	Behandelt entfernte Anfragen nach lokalen Dateien
Telnet Dämon	Erlaubt entfernten Login
Mail Server	Empfängt und speichert eingehende E-Mail
Fax Server	Empfängt und druckt eingehende Faxe
DNS Server	Internet Domain Name System
Ereignisprotokoll	Protokolliert verschiedene Systemereignisse
Plug-and-play-Manager	Überwacht die Hardware und reagiert auf Veränderungen

Speicherverwaltung

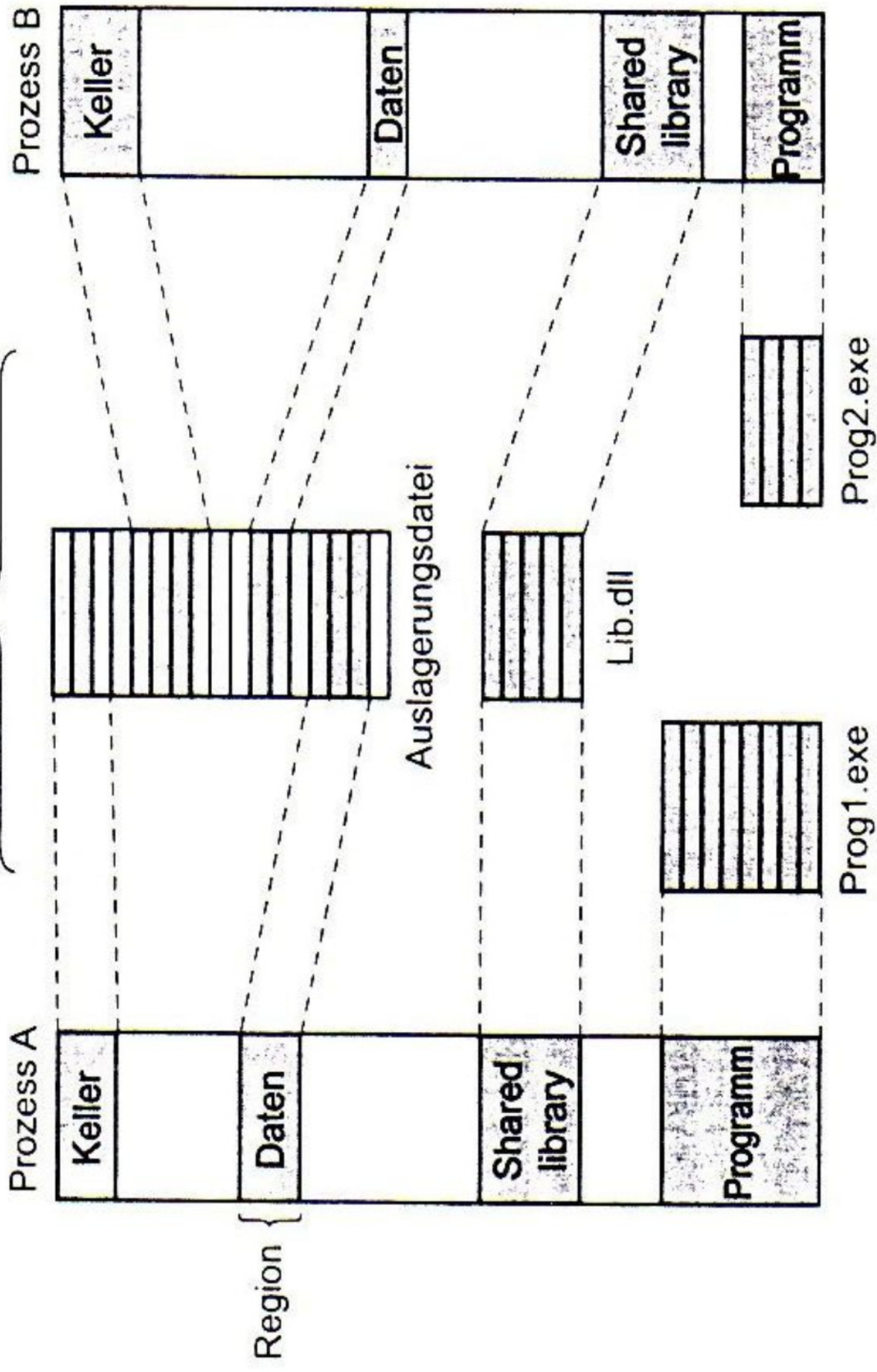
- Grundlegende Konzepte
 - Jeder Prozess hat 4GB adressierbaren Speicherraum
 - Die unteren 2GB sind Benutzerraum, die unteren Kernraum
 - 64KB oben und unten sind ungültig
 - Im letzten Teil der unteren 2GB sind die Timer und Zeitgeber
 - Programme können auf die oberen 2GB, das Betriebssystem, wenn überhaupt nur lesend zugreifen, jedoch können Threads, die in den Kernmodus wechseln so sofort diesen Bereich adressieren
 - Eine virtuelle Speicherseite hat 3 Zustände: frei, reserviert und belegt

Abbildung 11.23 Der virtuelle Adressraum für drei Prozesse. Die weißen Bereiche sind für jeden Prozess privat. Die schattierten Bereiche werden von allen Prozessen geteilt.



- Belegte Seiten können in Auslagerungsdateien gespeichert werden (max. 16 Auslagerungsdateien auf 16 Festplatten)
- Dateien können im Speicher abgebildet und verändert werden, ihre Auslagerungsdatei ist die Originaldatei (nach dem Zugriff wird die Originaldatei überschrieben)
- Alle Prozesse wird die gleiche Seite, mit der neusten Version der Datei, im Speicher zugewiesen
- Eine private Kopie von einer Seite wird an einen Prozess gegeben, wenn andere Prozesse mit dem Original weiter arbeiten müssen oder sollen (Copy-on-Write-Technik)
- Um flexible zu bleiben werden nur relative Adressen im virtuellen Speicher benutzt (Position-Independent-Code)
- Erst bei mehr als 2GB physischem Speicher braucht man Address-Windowing-Extensions oder auch Bank-Switching

Hintergrundspeicher auf der Platte



- Systemaufrufe zur Speicherverwaltung
 - Mit API-Befehlen kann man Seiten im Speicher verankern
 - Man kann die Lesen/Schreiben/Ausführen-Bits einer Region verändern und den Status von Regionen abfragen
 - Auch das Verriegeln und Entriegeln ist möglich
 - Eine Datei kann so im Speicher abgebildet und geöffnet werden
- Implementierung der Speicherverwaltung
 - Kümmt sich fast ausschließlich um Prozesse
 - Ein VAD (Virtual Address Descriptor) wird angelegt, eine Tabelle, in der die Seiten eines Prozesses angelegt sind.
So können auch verstreute Seiten verwaltet werden.

Abbildung 11.26 Ein Seitentabelleneintrag für eine eingelagerte Seite auf einem Pentium.



G: Seite global für alle Prozesse
 L: Große (4 MB) Seite (large)
 D: Seiteninhalt ungültig (dirty)
 A: Auf Seite wurde zugegriffen (accessed)
 C: Caching ein-/ausgeschaltet

Wt: Write through (kein Caching)
 U: Seite sichtbar im Benutzermodus
 W: Seite ist beschreibbar (write)
 V: Gültiger Eintrag (valid)

- Wenn ein Prozess eine Seite anfordert wird diese von der Speicherverwaltung in der Executive in den VADs gesucht
- Die Verwaltung findet oder erzeugt sie, wenn möglich
- Durch Bits wird z.B. gekennzeichnet, ob eine Seite gültig ist oder ob auf sie zugegriffen wurde
- Es gibt 5 Kategorien von Seitenfehlern:
 1. Die referenzierte Seite ist nicht belegt
 2. Eine Schutzverletzung ist aufgetreten
 3. Auf eine geteilt genutzte Seite wurde geschrieben
 4. Der Stack muss wachsen
 5. Die referenzierte Seite ist belegt aber momentan nicht eingelagert
- Die ersten Beiden bringen den Prozess zum abstürzen
- Der dritte Fehler ist nur bei lesbaren Seiten, dann kommt Copy-on-Write

- Beim vierten Fall muss der Seitenrahmen vergrößert werden jedoch mit einer Nullseite um herumschnüffeln zu vermeiden
- Bei Fall Fünf wird die Seite gesucht und eingelagert
- Einmal pro Sekunde überprüft der Balance-Set-Manager, ob es genug freie Seiten gibt
- Bei zu wenig freien Seiten wird der Working-Set-Manager gestartet, der weniger gebrauchte Seiten entfernt
- Der Working-Set-Manager läuft so lange bis genug Seiten frei sind und behandelt auch Teile des Betriebssystems
- Es gibt 5 Listen auf den sich Seiten befinden können:
 1. Zur Zeit benutze Seiten (Working Sets)
 2. Unveränderte Seiten (standby list)
 3. Veränderte Seiten (modified list)
 4. Freie Seiten (free list)
 5. Leere Seiten (zeroed pages list)

- Seiten auf dem Working Set, der modified- oder standby-list gehören noch zu einem Prozess
- Der Swapper-Thread läuft alle 4 Sekunden und sucht Prozesse, deren ganzen Threads im Leerlauf sind. Wenn er einen solchen Prozess findet schiebt er seine Seiten auf die modified- bzw. standby-list
- Die beiden Threads Mapped-Page-Writer und Modified-Page-Writer schreiben veränderte Seiten auf die Platte und verschieben sie, dann in die Unveränderte-Seiten-Liste
- Der Zero-Page-Thread läuft auf niedrigster Priorität und löscht Seiten, die auf der Freien-Liste stehen
- Woher Windows 2000 eine Seite nimmt, wenn eine gebraucht wird, ist so komplex, das die Programmierer es selbst nicht mehr verstehen

Abbildung 11.27 Die verschiedenen Seitenlisten und die Übergänge zwischen ihnen.

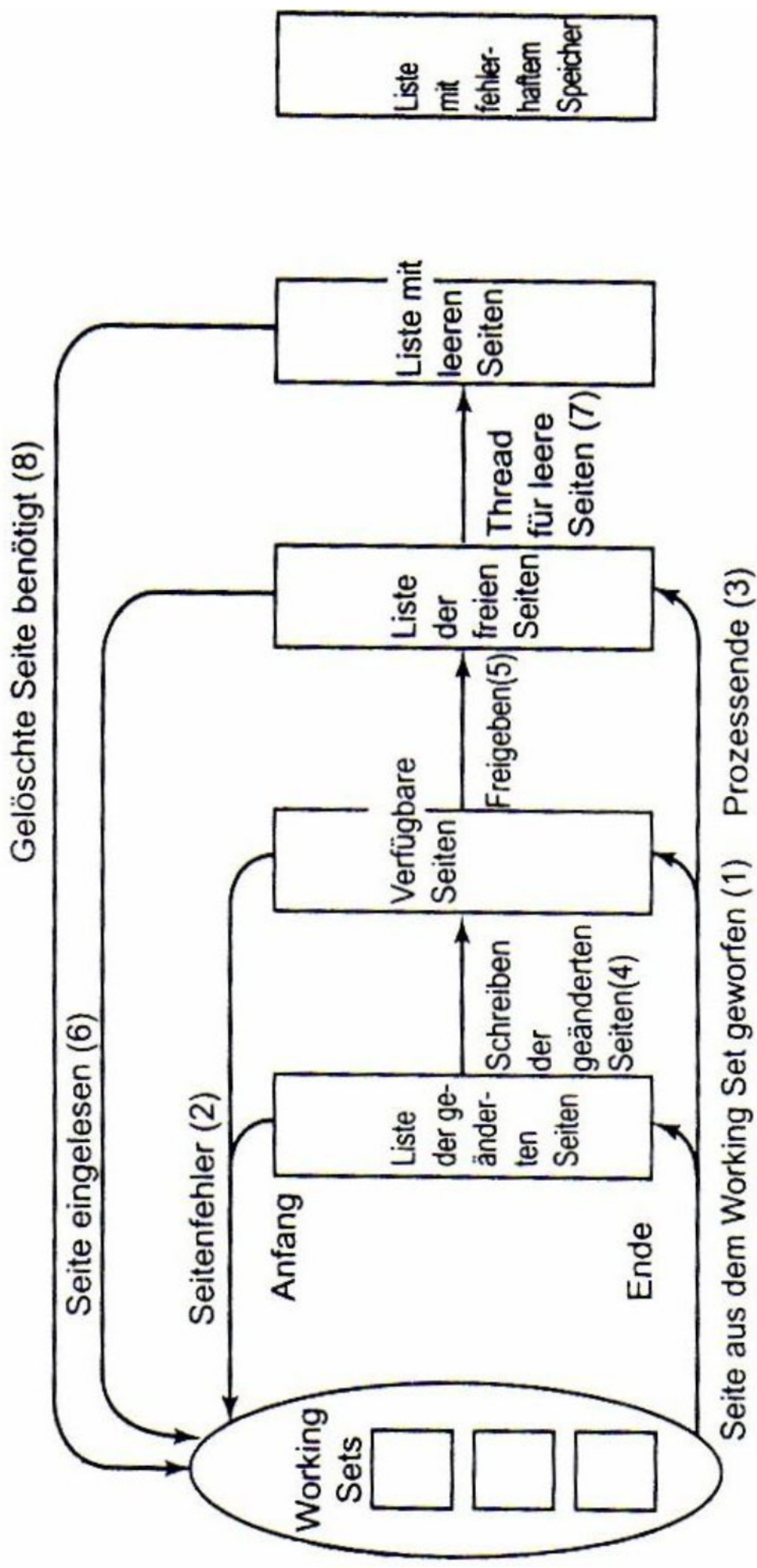
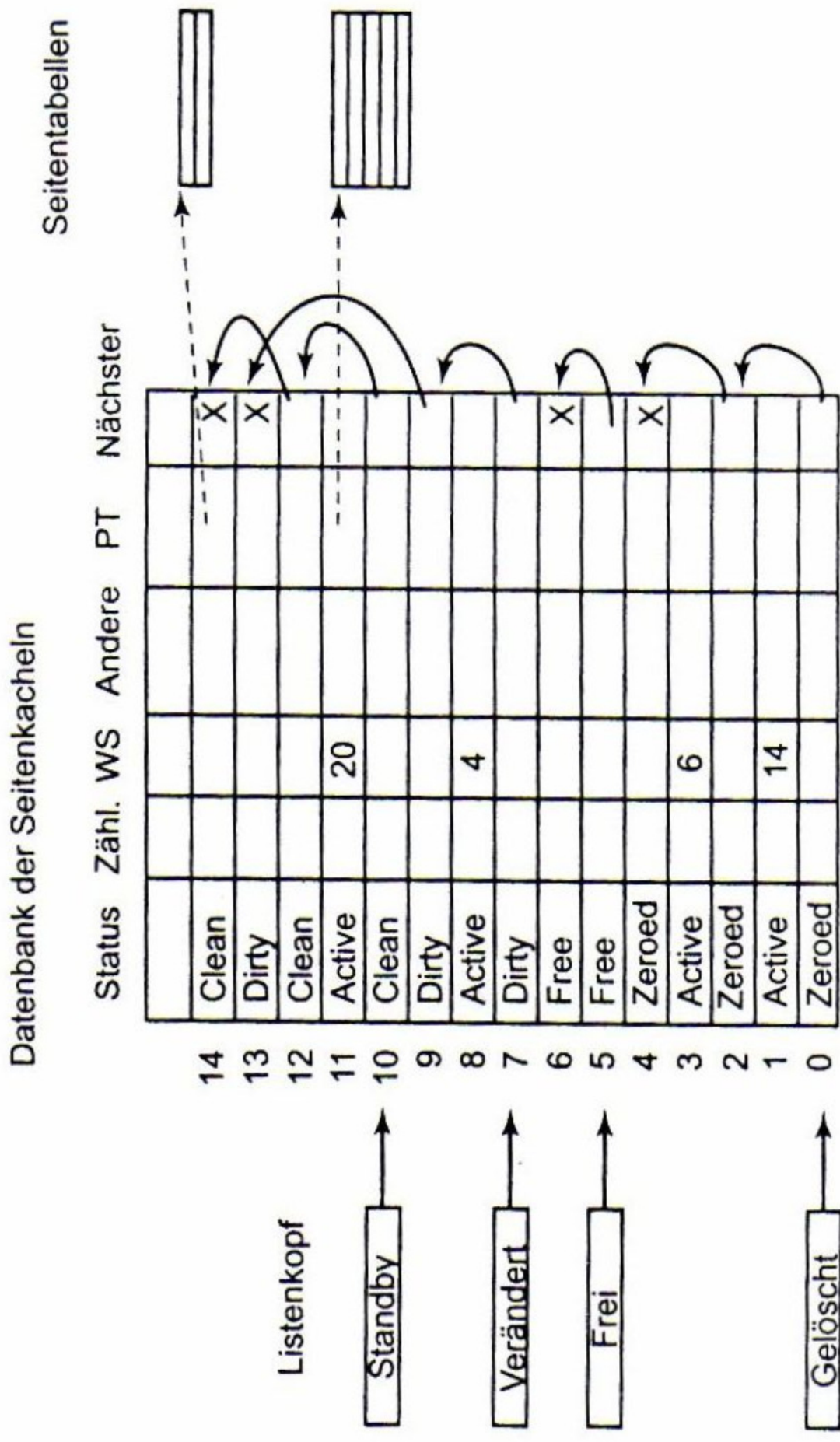


Abbildung 11.28 Einige der wichtigen Felder in der Seitenrahmendatenbank einer gültigen Seite.



Ein-/Ausgabe in Windows 2000

- Grundlegende Konzepte
 - Der E/A-Manager ist eng verbunden mit dem PnP-Manager
 - Der PnP-Manager findet die Hardware, lädt den Treiber und es wird ein Treiberobjekt erzeugt
 - Auch der Power-Manager muss eng mit dem E/A-Manager zusammenarbeiten
 - Beide Manager arbeiten zusammen um Geräte an- und aus zuschalten
 - Auch die Festplatte wird per Treiber vom E/A-Manager verwaltet, er holt die Blöcke für den Speicher von der Platte
 - Threads haben die Möglichkeit synchron mit der E/A zusammen zu arbeiten

- API-Aufrufe für die Ein-/Ausgabe
 - Es gibt über 100 verschiedene APIs für E/A
 - Die wichtigste API ist für das Grafiksystem, in ihr stehen Tausende von Win32-API-Aufrufen z.B. für Rollbalken etc.
 - Zu diesen Aufrufen gehören auch das Malen und Zeichnen oder das Darstellen von Text (der von Windows selbstständig in Bitmaps umgewandelt wird)

Abbildung 11.29 Einige Kategorien der Win32-API-Aufrufe.

API Gruppe	Beschreibung
Fensterverwaltung	Erzeugen, Zerstören und Ändern von Fenstern
Menüs	Erzeugen, Zerstören und Ergänzen von Menüs
Dialogfenster	Dialogbox öffnen und Informationen sammeln
Zeichnen und Malen	Anzeigen von Punkten, Linien und geometrischen Figuren
Text	Anzeigen von Text in spezieller Größe, Art und Farbe
Bitmaps und Icons	Verwalten von Bitmaps und Icons am Bildschirm
Farben und Paletten	Verwaltung der möglichen Farben
Zwischenablage	Informationsaustausch zwischen Anwendungen
Eingabe	Eingabe von der Maus und der Tastatur