1 Einführung

Als Einstieg in das Thema wollen wir feststellen, wo in einem Rechner das Betriebssystem einzuordnen ist, welchen Zwecken es dient und wie ein Betriebssystem als Begriff definiert werden kann.

1.1 Zweck

Der Begriff »Betriebssystem« kann auf unterschiedliche Art und Weise erfasst werden. So kann man sich fragen: Was leistet ein Betriebssystem? Es realisiert hauptsächlich zwei Grundfunktionen.

- *Erweiterte Maschine*: Das Betriebssystem verbirgt viele kleine, applikationsunabhängige Teilfunktionen und kann damit einfacher benutzt, d.h. auch programmiert werden als die blanke Rechnerhardware. Die erweiterte Maschine ist eine Abstraktion auf hohem Niveau und entspringt einer Top-down-Sicht.
- *Betriebsmittelverwalter*: Das Betriebssystem verwaltet die zeitliche und räumliche Zuteilung von Rechnerressourcen. Im Mehrprogrammbetrieb wird im Zeitmultiplex der Prozessor zwischen verschiedenen ablauffähigen Programmen hin und her geschaltet. Im Raummultiplex wird der verfügbare Speicher auf geladene Programme aufgeteilt. Ausgehend von den Ressourcen entspricht dies mehr einer Bottom-up-Sicht.

Genauer betrachtet erfüllt ein Betriebssystem sehr viele Zwecke. Es kann mehrere oder sogar alle der folgenden Funktionalitäten realisieren:

- *Hardwareunabhängige Programmierschnittstelle*: Programme können auf diese Weise unverändert auf verschiedenen Computersystemen ablaufen (auf Quellcodeebene gilt dies sogar für unterschiedliche Prozessorfamilien mit differierenden Instruktionssätzen).
- *Geräteunabhängige Ein-/Ausgabefunktionen*: Programme können ohne Änderung unterschiedliche Modelle eines Peripheriegeräts ansprechen.

- *Ressourcenverwaltung*: Mehrere Benutzer bzw. Prozesse können gemeinsame Betriebsmittel ohne Konflikte nutzen. Die Ressourcen werden jedem Benutzer so verfügbar gemacht, wie wenn er exklusiven Zugriff darauf hätte.
- *Speicherverwaltung*: Mehrere Prozesse/Applikationen können nebeneinander im Speicher platziert werden, ohne dass sie aufeinander Rücksicht nehmen müssen (jeder Prozess hat den Speicher scheinbar für sich allein). Zudem wird bei knappem Speicher dieser optimal auf alle Nutzer aufgeteilt.
- *Massenspeicherverwaltung (Dateisystem)*: Daten können persistent gespeichert und später wieder gefunden werden.
- *Parallelbetrieb (Multitasking)*: Mehrere Prozesse können quasiparallel auf einem Einprozessorsystem ablaufen. Konzeptionell erscheint dieses als Multiprozessorsystem, indem versteckt vor den Anwendungen parallele Abläufe sequenzialisiert werden.
- *Interprozesskommunikation*: Prozesse können mit anderen Prozessen Informationen austauschen. Die Prozesse können dabei entweder auf dem gleichen Rechner ablaufen (lokal) oder auf verschiedenen Systemen (verteilt) ausgeführt werden.
- *Sicherheitsmechanismen*: Es können sowohl Funktionen für die Datensicherung, d.h. die fehlerfreie Datenverarbeitung, als auch Datenschutzkonzepte implementiert sein. Der Datenschutz kann zum Beispiel durch das Löschen freigegebener Bereiche im Hauptspeicher und auf Plattenspeichern sichergestellt werden, da damit empfindliche Informationen nicht in falsche Hände fallen können. Auch die Zugangskontrolle zum Rechner (Anmeldedialoge, Benutzerverwaltung) dient dem Datenschutz.

Die geräteunabhängige Ein-/Ausgabe war eine der wichtigsten Errungenschaften bei der erstmaligen Einführung von Betriebssystemen. Früher war es notwendig, dass die Applikationen die Eigenheiten der angeschlossenen Peripheriegeräte kennen mussten. Mit einem Betriebssystem stehen logische Kanäle zur Verfügung, die Ein-/Ausgaben über standardisierte Funktionen bereitstellen (siehe Abb. 1–1). Die logischen Kanäle werden häufig mittels sprechender Textnamen identifiziert.

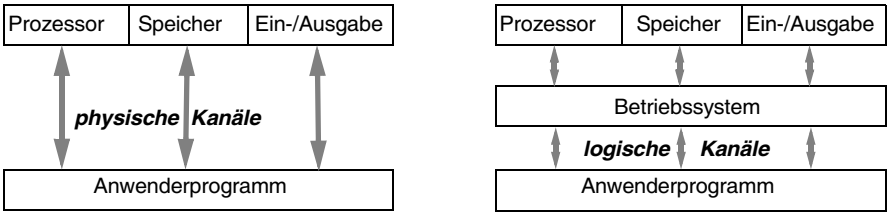

Abb. 1–1 *Ein-/Ausgabe ohne und mit Betriebssystem*

1.2 Definitionen

Leider existiert keine allgemein verbindliche Definition eines Betriebssystems. Welche Komponenten zu einem Betriebssystem gehören und welche nicht, lässt sich nicht endgültig festlegen. Nachfolgend sind drei unterschiedliche Definitionen stellvertretend vorgestellt, die dabei helfen, ein Betriebssystem zu charakterisieren. Eine erste, etwas schwer lesbare Definition nach DIN 44 300 beschreibt ein Betriebssystem wie folgt (Ausschnitt):

> *... die Programme eines digitalen Rechnersystems, die zusammen mit den Eigenschaften dieser Rechenanlage die Basis der möglichen Betriebsarten des Rechnersystems bilden und insbesondere die Abwicklung von Programmen steuern und überwachen.*

Eine zweite, der Literatur entnommene Definition lautet:

> *Ein Betriebssystem ist eine Menge von Programmen, welche die Ausführung von Benutzerprogrammen auf einem Rechner und den Gebrauch der vorhandenen Betriebsmittel steuern.*

Eine dritte Definition betrachtet das Betriebssystem als *Ressourcenverwalter*, wobei die Ressource hauptsächlich die darunter liegende Hardware des Rechners ist. Ein Computersystem lässt sich hierbei als eine strukturierte Sammlung von Ressourcenklassen betrachten, wobei jede Klasse durch eigene Systemprogramme kontrolliert wird (siehe Tab. 1–1).

	Zentrale Ressourcen	**Periphere Ressourcen**
Aktive Ressourcen	Prozessor(en)	Kommunikationseinheiten 1. Endgeräte (Tastaturen, Drucker, Anzeigen, Zeigegeräte etc.) 2. Netzwerk (entfernt, lokal) etc.
Passive Ressourcen	Hauptspeicher	Speichereinheiten 1. Platten 2. Bänder 3. CD-ROM/DVD etc.

Tab. 1–1 *Ressourcenklassen*

Ein Betriebssystem lässt sich auch mit einer Regierung (*government*) vergleichen. Wie diese realisiert das Betriebssystem keine nützliche Funktion für sich alleine, sondern stellt eine Umgebung zur Verfügung, in welcher andere Beteiligte nützliche Funktionen vollbringen können. Einige Autoren (z.B. K. Bauknecht, C. A. Zehnder) ziehen die Begriffe *Systemsoftware* bzw. *Systemprogramme* der Bezeichnung *Betriebssystem* vor. In diesem Sinne ist folgende Beschreibung dieser Autoren abgefasst:

»Die Systemprogramme, oft unter dem Begriff *Betriebssystem* zusammengefasst, lassen sich gemäß Abbildung 1–2 gruppieren. Die eigentlichen *Steuerprogramme* sind für folgende Funktionen zuständig:

- *Steuerung aller Computerfunktionen* und Koordination der verschiedenen zu aktivierenden Programme.
- *Steuerung der Ein-/Ausgabeoperationen* für die Anwendungsprogramme.
- *Überwachung und Registrierung* der auf dem Computersystem ablaufenden Aktivitäten.
- *Ermittlung und Korrektur* von Systemfehlern.«

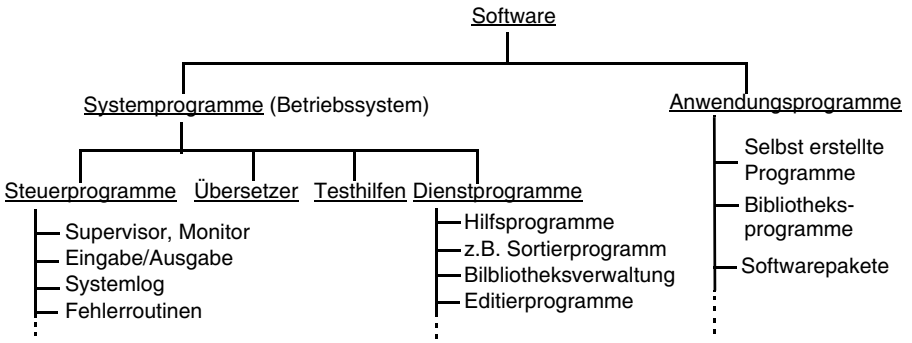

Abb. 1–2 *Softwaregliederung*

Auffallend bei dieser Definition ist der Einbezug von *Übersetzern* (Compiler, Binder), *Testhilfen* und *Dienstprogrammen*. Für klassische Betriebssysteme (z.B. Unix und GNU-Tools) trifft dies vollumfänglich zu, während moderne Betriebssysteme oft die Bereitstellung von Übersetzungs-Tools irgendwelchen Drittherstellern überlassen bzw. diese als separate Applikation ausliefern (z.B. Windows und Visual C/C++).

1.3 Einordnung im Computersystem

In einem Rechner stellt das Betriebssystem eine Softwareschicht dar, die zwischen den Benutzerapplikationen einerseits und der Rechnerhardware andererseits liegt (siehe Abb. 1–3). Das Betriebssystem selbst besteht aus einem *Betriebssystemkern* und einer Sammlung von Programmen, die *Betriebssystemdienste* bereitstellen. Je nach Betrachtungsweise zählen dazu auch Programme zur Softwareentwicklung, wie Editoren und Compiler. Häufig wird nur der Betriebssystemkern als Betriebssystem bezeichnet, während der Begriff *Systemprogramme* für das Gesamtpaket inklusive der Programmentwicklungswerkzeuge benutzt wird.

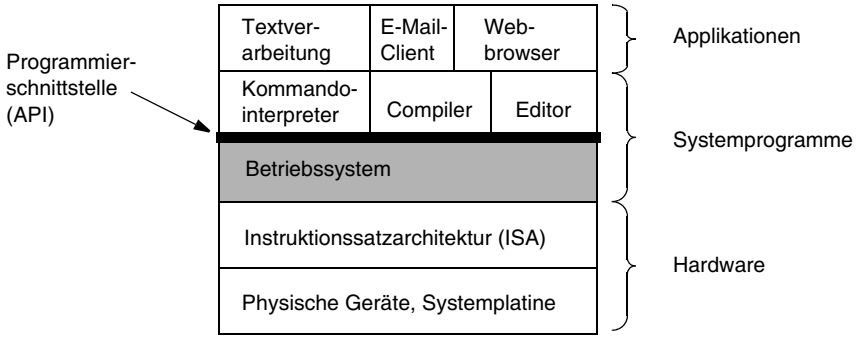

Abb. 1–3 *Schichtenmodell eines Rechners*

Das Betriebssystem setzt auf der Prozessorarchitektur auf, die durch einen Satz von Maschinenbefehlen und den Registeraufbau charakterisiert wird (sog. Instruktionssatzarchitektur, ISA). Die Systemplatine mit all ihren Bausteinen und den angeschlossenen Peripheriegeräten stellt die Arbeitsumgebung des Prozessors dar. Diese muss ebenfalls dem Betriebssystem in all ihren Details bekannt sein. Von zentraler Bedeutung für den Softwareentwickler ist die Programmierschnittstelle des Betriebssystems (*Application Programming Interface, API*). Die dort zur Verfügung gestellte Funktionalität kann in Benutzerapplikationen eingesetzt werden. Aus Anwendungssicht unterscheiden sich Betriebssysteme in der *Programmierschnittstelle*, in den unterstützten *Dateiformaten für ausführbare Dateien* und der *Maschinensprache*, in die ihr Code übersetzt wurde. Zudem kann oft der Funktionsumfang, d.h. die installierten Systemteile, während des Installationsvorgangs unterschiedlich gewählt werden. Wie bereits erwähnt, setzt das Betriebssystem direkt auf der Rechnerhardware auf und muss diese daher genau kennen. Denn es verwaltet folgende Hardwareelemente:

- Prozessor
- Arbeitspeicher (*main memory*)
- Massenspeicher (*mass storage*), z.B. Disketten, Festplatten, CD-ROM, DVD
- Benutzerschnittstelle (*user interface*)
- Kommunikations- und andere Peripheriegeräte (LAN, WLAN usw.)

Die Betriebssystemtheorie beruht damit auf den Prinzipien der Computertechnik. Computertechnik befasst sich mit:

1. Rechner-Grundmodellen (Von-Neumann-, Harvard-Architektur)
2. Funktionsweise des Prozessors (Instruktionssatz, Registeraufbau)
3. Speichern und ihren Realisierungen (Primär- und Sekundärspeicher)
4. Peripheriegeräten (Tastatur, Bildschirm, Schnittstellenbausteine usw.)

Um die hardwarenahen Teile des Betriebssystems oder nur schon den exakten Ablauf der Programmausführung zu verstehen, ist es daher unerlässlich, sich mit

ein paar Details der Computertechnik zu befassen. In diesem Buch sind an verschiedenen Stellen kleine Exkurse in die Computertechnik enthalten, die jeweils notwendiges Grundwissen zur Verfügung stellen. Dies kann nicht einen Grundkurs in Prozessortechnik ersetzen. Deswegen kann es sinnvoll sein, mittels entsprechender Bücher über Prozessorgrundlagen tiefere Einsichten in das Thema zu bekommen. Besonders sei dies empfohlen, wenn überhaupt kein entsprechendes Vorwissen vorliegt. Wir beschränken uns hier auf das Verständnis computertechnischer Funktionsweisen, soweit sie für das Verständnis des Betriebssystems nötig sind, und lassen die Realisierungsdetails der Hardwareelemente auf der Seite.

1.4 Betriebssystemarten

Ein Betriebssystem stellt eine Umgebung zur Verfügung, in der applikationsspezifische Programme ablaufen können. Eine derartige Umgebung kann recht unterschiedlich realisiert werden:

- Als Laufzeitsystem (*Run-Time System*) einer Programmiersprache (ADA, Modula-2)
- Als virtuelle Maschine zur Ausführung eines Zwischencodes (z.B. Java Virtual Machine)
- Als Basisprogramm eines Rechners (z.B. Unix, Windows)
- Als (sprachunabhängige) Programmbibliothek (z.B. Mikrokontroller-Betriebssysteme)

Häufig findet man auch Kombinationen dieser vier Varianten. Sprach-Laufzeitsysteme können Fähigkeiten zur Verfügung stellen, die ansonsten nur Bestandteil von Betriebssystemen sind. Dies betrifft beispielsweise Multitasking-Funktionen (z.B. in Java, Ada, Modula-2) und die Speicherverwaltung (verschiedene Sprachen).

1.4.1 Klassische Einteilungen

Eine einfache Klassifizierung von Betriebssystemen richtet sich nach den Anwendungsgebieten:

- *Stapelverarbeitung (batch processing)*: Programme werden einzeln gestartet. Dies kann direkt durch einen Operator geschehen oder indirekt über die Interpretation einer Stapelbefehlsdatei. Das ist eine klassische Form von Großrechnerbetriebssystemen z.B. zur Ausführung von Buchhaltungsprogrammen über Nacht.
- *Time-Sharing-Betrieb*: Die zur Verfügung stehende Rechenleistung wird in Form von Zeitscheiben (*time slices, time shares*) auf die einzelnen Benutzer aufgeteilt mit dem Ziel, dass jeder Benutzer scheinbar den Rechner für sich

alleine zur Verfügung hat. Die Zuteilung der Zeitscheiben kann nach unterschiedlichen Strategien erfolgen, die alle eine möglichst gerechte Aufteilung anstreben. Historisch gesehen sind Time-Sharing-Systeme die Nachfolger der Batch-Systeme mit der Neuerung, dass sie alle Benutzer interaktiv arbeiten lassen.

- *Echtzeitbetrieb*: Die Rechenleistung wird wiederum auf mehrere Benutzer oder zumindest Prozesse aufgeteilt. Als wesentliches Element werden jetzt aber unterschiedliche Prioritäten berücksichtigt mit dem Ziel, die wichtigsten Prozesse innerhalb gewisser Zeitschranken auszuführen. Echtzeitsysteme sind reaktive Systeme, indem sie auf Signale aus der Umgebung (Interrupts, Meldungen) möglichst rasch reagieren sollen.

Moderne Betriebssysteme fallen mehr oder weniger in die Gruppe der Echtzeitsysteme, weswegen diese für uns im Vordergrund stehen. Eine weitere Klassifizierung unterteilt Betriebssysteme nach unterstützter Rechnerstruktur:

- Einprozessorsysteme
- Multiprozessorsysteme
- Verteiltes System

Je nach Auslegung des Computersystems kann ein Betriebssystem eine oder mehrere dieser Rechnerstrukturen unterstützen.

Beispiele:
Windows unterstützt Einprozessorsysteme und Multiprozessorsysteme. Das Betriebssystem Amoeba ermöglicht das transparente Arbeiten auf einem verteilten System. Für den Benutzer präsentiert es sich wie ein Einzelrechner, besteht in der Tat aber aus mehreren über ein Netzwerk verbundenen Computern.

1.5 Betriebssystemarchitekturen

1.5.1 Architekturformen

Solange es nur um die Systemprogrammierung geht, kann man sich mit einer Black-Box-Betrachtung des Betriebssystems zufrieden geben. Nach außen ist damit nur die Programmierschnittstelle sichtbar, jedoch nicht das Systeminnere (siehe A in Abb. 1–4). Dies entspricht einem klassischen Ideal des Software Engineering, das aussagt, dass die Schnittstelle das Maß aller Dinge ist und die Implementierung dahinter beliebig austauschbar sein soll. Trotzdem kann es hilfreich sein, die Innereien eines Betriebssystems zu kennen, damit man nicht Gefahr läuft, quasi gegen die Implementierung zu programmieren. Dies könnte zum Bei-

spiel in einem überhöhten Ressourcenverbrauch oder einer unbefriedigenden Ausführungsgeschwindigkeit resultieren. Daneben ist es stets interessant, unter die »Motorraumhaube« eines Betriebssystems zu gucken. Mit anderen Worten, es geht um eine White-Box-Betrachtung (siehe B in Abb. 1–4).

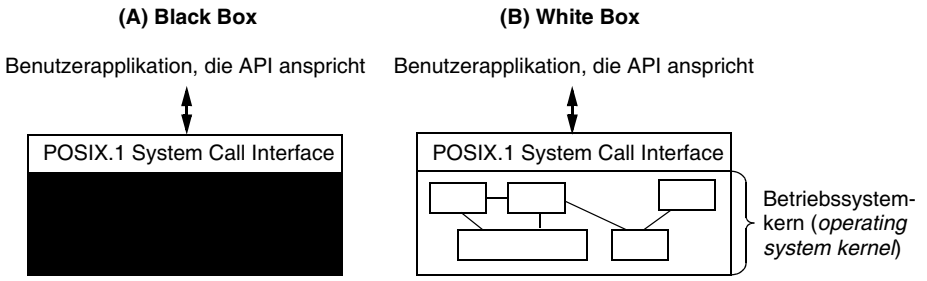

Abb. 1–4 *Black- und White-Box-Betrachtung (Beispiel: Unix)*

Damit verbunden sind auch die Entwurfs- und Konstruktionsprinzipien, die einen erst dann interessieren, wenn man über die Black-Box-Betrachtung hinausgeht. Eine wesentliche Frage ist dabei die Art und Weise, wie die Betriebssystemsoftware strukturiert ist. In der Theorie kennt man *drei Grundstrukturen*, denen sich konkrete Betriebssysteme zuordnen lassen: *monolithische, geschichtete* und *Mikrokernsysteme*. Zuerst soll jedoch auf die Funktionsweise und Bedeutung der Benutzer-/Kernmodus-Umschaltung eingegangen werden, da sie bei der Betrachtung der Grundstrukturen eine zentrale Rolle spielt.

1.5.2 Benutzer-/Kernmodus

Als hardwarenahe Softwarekomponente ist ein Betriebssystem eng mit den Möglichkeiten der unterliegenden Plattform verbunden. Es haben sich mit den Jahren unterschiedliche Leistungsklassen von Prozessoren und zugehöriger Hilfslogik etabliert:

- *Mikrokontroller*: Es handelt sich hierbei um einfache Mikroprozessoren, die primär in eingebetteten Systemen (*embedded systems*) eingesetzt werden. Um die Kosten gering zu halten, verfügen sie lediglich über einen Prozessor mit wenig oder gar keinen weiterführenden Mechanismen zur Unterstützung eines Betriebssystems. Hingegen sind sie zusammen mit verschiedenen Peripherieeinheiten (E/A, Kommunikation, Zeitgeber usw.) in einen einzigen Halbleiterchip integriert, was Kosten und Platz spart.
  Beispiele: Intel 8051, Siemens 80C166, Motorola HC6805
- *Einfache Universalmikroprozessoren*: Sie entsprechen in vielen Punkten den Mikrokontrollern, enthalten jedoch auf dem gleichen Chip keine Peripherieeinheiten. In dieser Gruppe finden wir vor allem die älteren Prozessortypen.
  Beispiele: Intel 8080/85/86, Motorola 6800, 68000

- *Leistungsfähige Universalmikroprozessoren*: Diese Rechnerchips verfügen über eine ganze Reihe von Hardwareelementen, die ein Betriebssystem unterstützen. Dazu zählen eine MMU (*Memory Management Unit*) und Mechanismen für einen privilegierten Betriebsmodus für die Systemsoftware (Privilegiensystem).

Da in modernen Desktop-Rechnern und Servermaschinen nur die letzte der obigen Gruppen zum Einbau gelangt, konzentrieren wir uns auf deren Möglichkeiten. Für die Betrachtung der Grundstrukturen sind sowohl das *Privilegiensystem* des Prozessors als auch die Fähigkeiten der MMU wichtig. Durch das Privilegiensystem sollen gewisse Operationen und Zugriffe geschützt werden, damit ein Programmierfehler in einem Anwendungsprogramm nicht das ganze Computersystem durcheinander bringt. Insbesondere bei Multitasking-Anwendungen und Multiuser-Betrieb (mehrere gleichzeitige Benutzer) ist ein solcher Schutz erwünscht. So wird in den meisten Betriebssystemen der Zugriff auf Hardwareteile mittels dieser Schutzfunktionen dem normalen Anwender (bzw. Benutzerapplikationen) verwehrt. Die Grundidee einer Hardwareunterstützung besteht darin, unterschiedliche Betriebsarten zu ermöglichen, wobei jede Betriebsart in ihren Pflichten und Rechten genau definiert ist. Im Minimum beinhaltet dies:

- Einen Kernmodus (*kernel mode, supervisor mode*) mit »allen Rechten« für Betriebssystemcode
- Einen Benutzermodus (*user mode*) mit »eingeschränkten Rechten« für Applikationscode

Das Ziel besteht darin, die Applikationen untereinander und den Betriebssystemcode gegen diese zu schützen. Dies bedeutet, dass eine Applikation nicht das ganze System lahm legen darf. Komfortablere Lösungen unterstützen teilweise mehr als zwei Betriebsarten, die auch Privilegienstufen (*privilege level*) genannt werden.

	Benutzermodus	**Kernmodus**
Ausführbare Maschinenbefehle	Begrenzte Auswahl	Alle
Hardwarezugriff	Nein bzw. nur mithilfe des Betriebssystems	Ja, Vollzugriff
Zugriff auf Systemcode bzw. Daten	Keiner bzw. nur lesend	Exklusiv

Tab. 1–2 *Vergleich zwischen Benutzer- und Kernmodus*

Die Möglichkeiten des Privilegiensystems werden stets in Kombination mit der Speicherverwaltung genutzt. So könnte die MMU dafür sorgen, dass nur im Kernmodus ein Zugriff auf Systemcode und Daten möglich ist. Den Benutzerprozessen ist mithilfe von diesem hardwaregestützten Mechanismus in der Regel weder ein direkter Zugriff auf Hardwareteile noch ein Überschreiben von Systemcode oder Systemdaten möglich. Mit der Kenntnis der Fähigkeiten der Benut-

zer-/Kernmodus-Umschaltung lassen sich die nachfolgend aufgeführten drei Typen von Aufbaustrukturen klarer in ihren Eigenschaften unterscheiden. Oberstes Ziel ist, das unabsichtliche Überschreiben von Systemdaten und Code zu verhindern, um unkontrollierte Systemabstürze zu vermeiden. Im Idealfall werden Schutzmechanismen auch für die Abschottung verschiedener Systemteile untereinander eingesetzt. Die Grenze des Sinnvollen ist allerdings darin zu sehen, dass eine teilweise lahm gelegte Systemsoftware aus Sicht des Anwenders oft nicht besser ist als ein Totalabsturz. Es kann jedoch manchmal nützlich sein, nicht vertrauenswürdige Teile der Systemsoftware, wie Erweiterungen oder Treiber von Drittherstellern, in ihrem Schadenspotenzial zurückzubinden. Ein sekundäres Ziel für den Einsatz der Benutzer-/Kernmodus-Umschaltung können Maßnahmen zur Eindämmung von Systemmanipulationen sein, die zum Ziel haben, vertrauenswürdige Daten zu missbrauchen. So ist Sicherheitssoftware fundamental von den Sicherheitseigenschaften einer Systemplattform abhängig, die softwareseitig durch das benutzte Betriebssystem gegeben ist.

1.5.3 Monolithische Systeme

Die Struktur dieser Systeme besteht darin, dass sie keine oder nur eine unklare Struktur haben (Abb. 1–5). Meist handelt es sich um evolutionär gewachsene Betriebssysteme, bei denen es anfänglich unwichtig war, einzelne Teilfunktionen klar mit Schnittstellen voneinander abzugrenzen. Beispiel dafür sind MS-DOS und ältere Unix-Varianten (inklusive Linux).

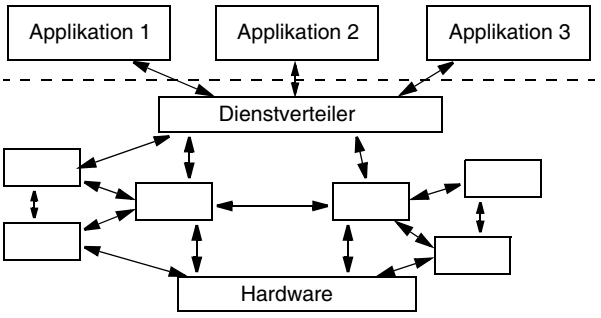

Abb. 1–5 *Beispiel für eine monolithische Betriebssystemstruktur*

1.5.4 Geschichtete Systeme

Bei dieser Strukturierungsform sind die Betriebssystemfunktionen in viele Teilfunktionen gegliedert, die hierarchisch auf mehrere Schichten verteilt sind. Die Festlegung der Schichtenstruktur ist vom einzelnen Betriebssystem abhängig, eine Standardstruktur gibt es nicht.

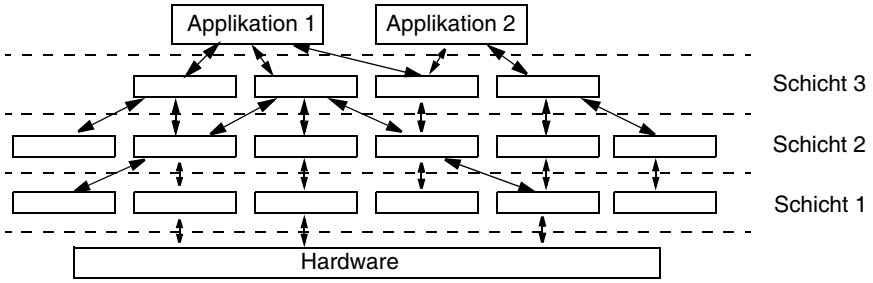

Abb. 1–6 *Beispiel einer geschichteten Struktur*

Wie in Abbildung 1–6 zu sehen ist, bauen Funktionen einer höheren Schicht strikt nur auf Funktionen einer tieferen Schicht auf. Jede Schicht realisiert eine bestimmte Funktionsgruppe. Systemaufrufe passieren nach unten alle Schichten, bis sie auf die Hardware einwirken. Eingabedaten durchlaufen umgekehrt alle Schichten von unten bis oben zur Benutzerapplikation. Die Schicht 1 in Abbildung 1–6 könnte zum Beispiel eine Hardware-Abstraktionsschicht sein, die eine allgemeine Betriebssystemimplementierung auf eine bestimmte Hardwareplattform anpasst. Beispiele für geschichtete Betriebssysteme sind neuere Unix-Varianten und OS/2.

1.5.5 Mikrokernsysteme

Nur die allerzentralsten Funktionen sind in einem Kernteil zusammengefasst, alle übrigen Funktionen sind als Serverdienste separat realisiert (z.B. Dateidienste, Verzeichnisdienste). Der Mikrokern enthält lediglich die vier Basisdienste Nachrichtenübermittlung (*message passing*), Speicherverwaltung (*virtual memory*), Prozessorverwaltung (*scheduling*) und Gerätetreiber (*device drivers*). Diese sind dabei in ihrer einfachsten Form realisiert. Weiter gehende Funktionen sind in den Serverprozessen enthalten, die im Benutzermodus ausgeführt werden. Dadurch werden die komplexeren Teile in klar abgegrenzte Teile aufgesplittet, wovon man sich eine Reduktion der Komplexität verspricht. Da zudem ein Großteil des Betriebssystemcodes auf die Benutzerebene (Benutzermodus) verschoben wird, sind die empfindlichsten Teile des Systemcodes gegen fehlerhafte Manipulationen aus dieser Richtung geschützt (Kernmodus). Ebenso ist es nur dem eigentlichen Kern erlaubt, auf die Hardware zuzugreifen. Beansprucht ein Benutzerprozess einen Systemdienst, so wird die Anforderung als Meldung durch den Mikrokern an den zuständigen Serverprozess weitergeleitet. Entsprechend transportiert der Mikrokern auch die Antwort an den anfordernden Prozess zurück (siehe Abb. 1–7). Vorteilhaft ist das derart verwendete Client/Server-Modell für verteilte Betriebssysteme. Für den Benutzerprozess bleibt es verborgen (transparent), ob der Serverprozess lokal oder an einem entfernten Ort ausgeführt wird.

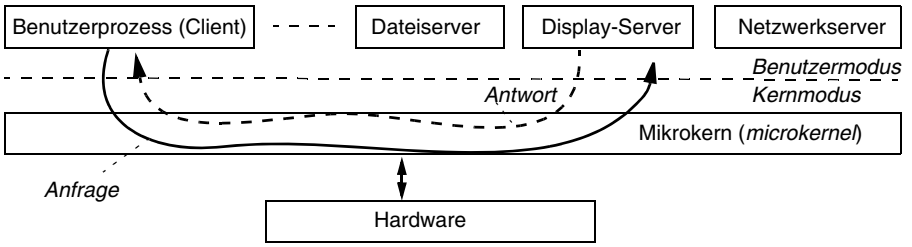

Abb. 1–7 *Mikrokern nach dem Client/Server-Prinzip*

Kommerzielle Mikrokernbetriebssysteme verlagern neben den vier erwähnten Grunddiensten zusätzliche Funktionen in den Mikrokern, da eine Mikrokern-Implementation in ihrer reinen Form infolge des Client/Server-Meldungsverkehrs zumindest für gewisse Systemdienste unangenehm ineffizient ausfallen würde. Betrachtet man beispielsweise den Datenverkehr zwischen einem Benutzerprozess und dem Display-Server (siehe Abb. 1–7), so muss für eine bestimmte Operation auf dem grafischen Desktop insgesamt viermal eine Umschaltung zwischen Benutzer- und Kernmodus stattfinden. Dies verlangsamt besonders einfachere Operationen nicht unbeträchtlich, die selbst wenig Rechenzeit benötigen. Es stellt sich zudem die Frage, ob eine Systemarchitektur, in der z.B. der Mikrokern unbeeinträchtigt weiterläuft, wenn die gesamte grafische Oberfläche infolge von Fehlzugriffen nicht mehr ansprechbar ist, letztlich viel mehr bringt. Eine derartige Situation führt nämlich über alle Systemkomponenten hin betrachtet meist doch zu einem nicht erholungsfähigen Betriebszustand. Beispiele derartiger Systemarchitekturen sind das Mac OS X (basierend auf Mach kernel) und Amoeba.

1.5.6 Beispiele von Systemarchitekturen

Unix System V

Der innere Aufbau des Unix-Betriebssystems (System V Release 3 hier als Beispiel betrachtet) spiegelt die zwei zentralen Unix-Konzepte *Dateien (files)* und *Prozesse (processes)* über entsprechende Subsysteme wider. Diese sind in Abbildung 1–8 als klar abgegrenzte logische Blöcke zu sehen. In der Realität sind die Abgrenzungen aber nicht so eindeutig, da einige Module auf interne Funktionen anderer Module einwirken (monolithische Struktur). In der Abbildung 1–8 sind die drei Schichten *Benutzerebene (user level)*, *Kernebene (kernel level)* und *Hardwareebene (hardware level)* dargestellt. Zuoberst stehen die Benutzerprogramme, die entweder direkt (über Trap-Interrupt von Assemblersprache) oder mithilfe von API-Funktionen aus einer Programmbibliothek (Hochsprache) die Systemdienste nutzen. Für die Interrupt-Verarbeitung liegen Kernroutinen vor, die bei einer Programmunterbrechung aufgerufen werden.

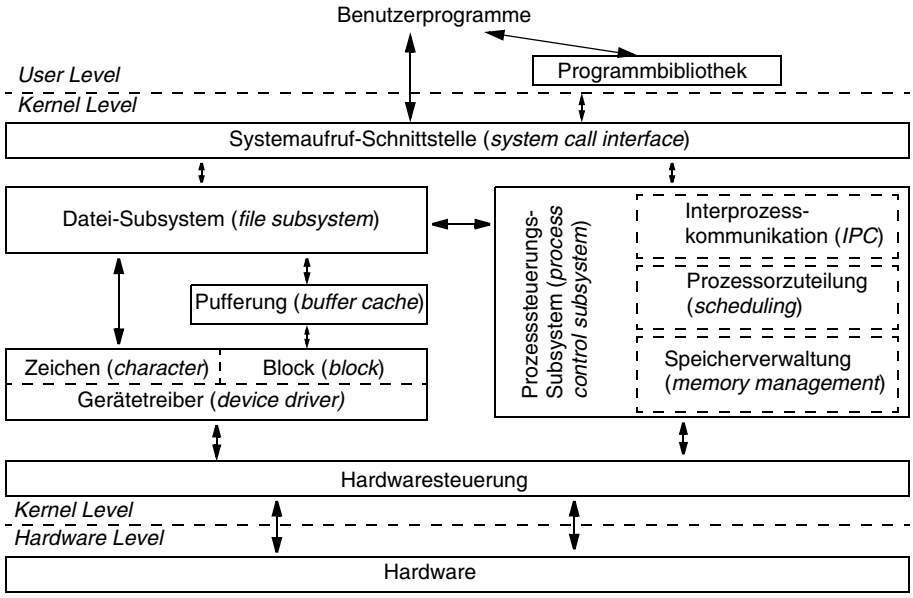

Abb. 1–8 *Interne Struktur des Unix (Kern des System V Release 3)*

Erwähnenswert ist, dass viele Unix-Kommandos gleichartig wie Benutzerprogramme implementiert sind, indem sie als ausführbare Dateien vorliegen und die Programmierschnittstelle zur Kommunikation mit dem Kern benutzen (Beispiel: Unix Kommandointerpreter, *shells*). Dadurch konnte der Kern kompakt und überschaubar gestaltet werden. Die Steuerung von Peripheriegeräten erfolgt durch die Treiber, die entweder zeichenorientiert arbeiten (*character device driver*) oder ganze Datenblöcke manövrieren (*block device driver*). Letztere Gruppe von Treibern kann mithilfe eines Puffers Daten sowohl beim Lesen als auch beim Schreiben zwischenpuffern. Unix ist in C programmiert, ergänzt mit wenigen hardwarenahen Teilen in der Assemblersprache der unterliegenden Hardwareplattform.

Windows 7

Das Betriebssystem Windows 7 besitzt eine Architekturmischform, die sowohl Elemente der Mikrokernidee als auch der geschichteten Strukturierung realisiert. Jedoch unterscheidet es sich nicht groß von vielen Unix-Systemen, indem ein Großteil des Systemcodes einschließlich der Treiber in der gleichen Ausführungsumgebung abläuft und damit unter sich keinen Schutz gegen Fehlzugriffe genießt. Hingegen sind viele Dienste und Hilfsfunktionen in separate Prozesse ausgelagert und daher genauso gegeneinander geschützt wie Benutzerprozesse unter sich.

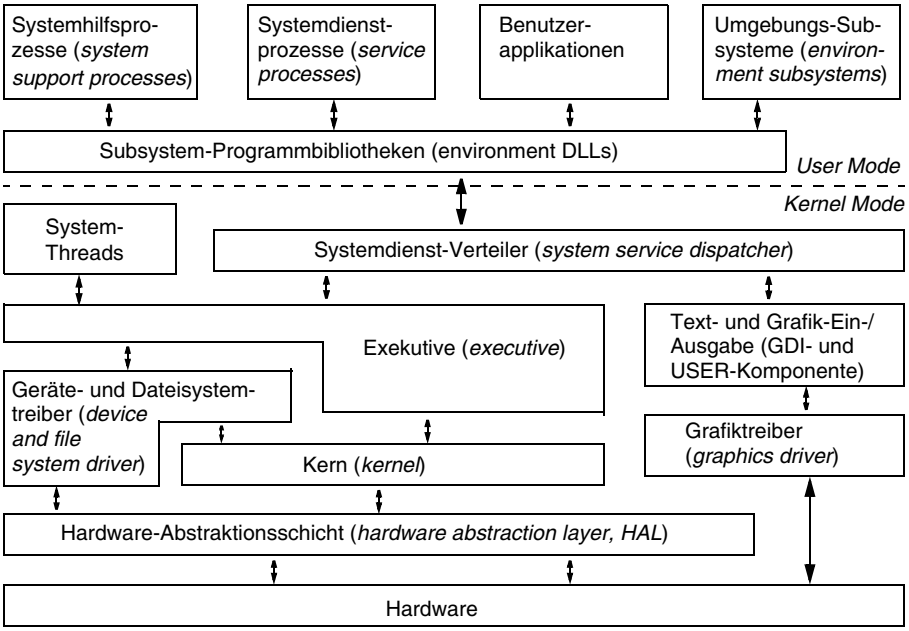

Abb. 1–9 *Interne Struktur von Windows 7*

Alle Systemteile im Kernmodus sind insgesamt gegen böswillige Benutzerprozesse abgeschottet. Damit unterscheidet es sich deutlich von früheren Windows-Produkten der Reihe 3.x/95/98/ME, die keinen vollständigen Schutz des Systemcodes vor Manipulationen durch fehlerhafte Applikationen boten. Dies ist jedoch eine fundamentale Anforderung für ein stabiles und robustes Betriebssystem. Windows 7 ist in C, zu kleineren Teilen in C++ programmiert. Wenige Softwareteile, die direkt die Hardware ansprechen, sind auch in Assemblersprache kodiert.

Nachdem Windows über viele Jahre nur als Gesamtsystem installier- und ladbar war, wurde mit Windows Server 2008 eine Version ohne grafische Oberfläche geschaffen (*windows core*), da ein GUI für den Serverbetrieb nicht unbedingt notwendig ist. Mit Windows 7 wurde zudem ein *MinWin* definiert, das nur aus dem eigentlichen Windows Kernel, den Netzwerkprotokollen, dem Dateisystem und einem minimalen Satz von Diensten (*core services*) besteht und in 40 MB Hauptspeicher Platz findet. MinWin kann unabhängig vom restlichen Windows-Code geladen und getestet werden, womit die höheren Schichten des Systems besser abgekoppelt werden. Entsprechend wurde auch systemintern eine *MinWin API* definiert, die nun von den höheren Schichten genutzt wird.

Google Chrome OS

Google realisiert mit diesem Betriebssystem eine neue Idee, wie Anwender mit Programmen arbeiten, nämlich webbasiert. Man ist auch versucht zu sagen, dass das Chrome OS die Versprechungen einlöst, die für das Web 2.0 gemacht wurden. Die Architekturidee besteht im Wesentlichen darin, aus serverbasierten Applikationen (*Cloud Services, Cloud Computing*) und kostengünstigen Netbooks eine Systemlösung zu realisieren, die übliche Anwendungen, wie Office-Programme, Kalender- und Informationsdienste, dem Anwender als Webapplikationen zur Verfügung stellt, ohne dass er diese Applikationen auf seinem Computer installieren muss. Der Zugriff auf die Cloud Services erfolgt via Chrome Webbrowser. Traditionelle GUI-Applikationen lassen sich nicht installieren, da als Benutzeroberfläche der Webbrowser dient, der seinerseits auf einem Linux-Kernel aufsetzt. Damit der Benutzer zwischendurch ohne Internetanbindung arbeiten kann, ermöglicht ihm das ebenfalls von Google stammende Browser-Plug-in *Gears*, seine Daten lokal zu speichern. Um einen möglichst schnellen Arbeitsbeginn zu erreichen, setzt das Chrome OS auf einer umfangreichen Firmware auf, die für eine blitzartige Initialisierung der Hardware sorgt. So gesehen liegt eine Drei-Schichten-Architektur vor: zuunterst die Firmware, dann der Linux-Kernel und zuoberst der Chrome Webbrowser.

1.5.7 Abstraktionen aus Benutzer- und Entwicklersicht

Abstraktionen sind ein wichtiges Mittel, um die Arbeit mit Computern zu vereinfachen. Sie spielen in zweifacher Hinsicht eine Rolle:

- *Abstraktion aus Benutzersicht:* Vorhandene Programme einschließlich des Betriebssystems sollen möglichst intuitiv bedienbar sein.
- *Abstraktion aus System- und Entwicklersicht*: Programme sollen möglichst einfach und flexibel entworfen werden. Flexibilität schließt dabei die Unabhängigkeit von konkreter Rechnerplattform und Peripherie ein.

Abstraktion aus Benutzersicht

Die Schnittstelle zwischen dem Computer und dem Menschen war und ist ein Thema, das einer laufenden Weiterentwicklung unterliegt. Dabei steht die Frage »Wie sollen die zu verarbeitenden Informationen nach außen visualisiert und rechnerintern dargestellt werden?« im Fokus. Die Visualisierung wird ergänzt durch Dialogverfahren zwischen Benutzer und Gerät, was gemeinhin unter folgenden Begriffen verstanden wird:

- Benutzerschnittstelle (*User Interface, UI*)
- Mensch-Maschine-Schnittstelle (*Man Machine Interface, MMI* bzw. *Human Machine Interface, HMI*)

Aus Sicht des Benutzers stellt diese Schnittstelle das dar, was er von einem Rechner wahrnehmen kann, also was offensichtlich ist. Daher wird oft auch der Begriff der Bedienoberfläche verwendet, um damit die dahinter liegende Existenz von verborgener rechnerinterner Datenverarbeitung anzudeuten. Leider ist es noch nicht gelungen, allgemein gültige Abstraktionen der Datenverarbeitung zu finden, die einerseits der menschlichen Begriffs- und Vorstellungswelt entsprechen und andererseits alle Anwendungsgebiete von Universalrechnern abdecken. Man muss hier von einem offenkundigen Spannungsfeld sprechen, das durch folgende Ansprüche geschaffen wird:

- *Universalität der Rechneranwendung*: z.B. Büroanwendungen, Programmentwicklung, wissenschaftliche Rechnernutzung, Informationssysteme für Unternehmensdaten
- *Optimale Darstellung von Objekten der Datenverarbeitung*: z.B. für wenig versierte Anwender, für Softwareentwickler

Von der Idee her wären ganz unterschiedliche Bedienoberflächen für Benutzer, die sich mit Computerinternas nicht befassen wollen, und für Softwareentwickler, die viele Details genau wissen müssen, denkbar. Tatsache ist jedoch, dass die grundlegende Bedienoberfläche eines Universalrechners für alle Benutzer weitgehend gleich sichtbar ist. Erst durch die Ausgestaltung der einzelnen Applikationen werden im Rahmen der Möglichkeiten des Betriebssystems individuelle Bedienoberflächen geschaffen. Wesentlich ist hier der Punkt, dass Betriebssysteme keine völlige Freiheit bei der Gestaltung der Bedienoberfläche erlauben, sondern stets einen Grundsatz an Funktionalität und Darstellungsmöglichkeiten vorgeben. Dies prägt die Bedienschnittstelle in einem wesentlichen Grad, da meist nur beschränkte Möglichkeiten vorhanden sind, diese Darstellungselemente flexibel anzupassen.

Nun wollen wir uns näher mit diesen Fähigkeiten eines Betriebssystems befassen. Sieht man von den allerersten Rechnern ab, so kann man erkennen, dass die Gestaltung der Bedienoberfläche sich an *Metaphern* orientiert. Eine Metapher ist ein übertragener bildlicher Ausdruck, z.B. indem ein Symbol für einen Papierkorb auf der Bedienoberfläche sinnbildlich für das (vorläufige) Wegwerfen von Daten steht. Genau wie bei einem richtigen Papierkorb kann man ein dort deponiertes Dokument wieder »herausfischen«. Erst das Leeren des Papierkorbs stellt einen unwiderrufbaren Löschvorgang dar.

Gegenstände aus dem Alltag werden für die Visualisierung von Vorgängen in einem Rechner eingesetzt, die abstrahiert zwar gleichartig, konkret aber völlig andersartig ablaufen. Besonders auffallend ist hier die Anlehnung an die Büroarbeit, in der heute tatsächlich ein PC ein kaum noch wegzudenkendes Bürogerät darstellt. Die Sinnbilder selbst repräsentieren dabei schon auf das Wesentliche zusammengefasste Objekte. So wird der Bildschirm als Desktop (desk = Schreibtisch, top = Oberseite) bezeichnet, wobei das Deckblatt als Arbeitsfläche nachge-

bildet wird. Und es stört auch niemanden, dass der Papierkorb auf dem virtuellen Schreibtisch oben darauf steht. Metaphern sind ebenfalls wichtig bei der permanenten Datenspeicherung auf Rechnern. Dies kommt durch die in der Tabelle 1–3 aufgelisteten Analogien zum Ausdruck (Aufzählung nicht vollständig).

Computerbegriff	Alltagsbegriff	Weitere Alltagsbegriffe
file	Akte	Datei, elektronisches Dokument
folder	Hängemappe	Ordner, Verzeichnis

Tab. 1–3 *Metaphern*

An dem Begriff *folder* ist die Wandlung der Begriffe über die Zeit ersichtlich. Er wurde erst in neuerer Zeit eingeführt und ersetzt den Begriff des *directory* (Verzeichnis). Der Grund dürfte in der einsichtigeren grafischen Darstellung einer Hängemappe liegen. Auffallend ist der teilweise nicht konsistente Gebrauch der Metaphern, was die Rechnerbedienung nicht gerade einfacher erscheinen lässt. Betrachtungen über die Gestaltung der Rechnerbedienung stellen ein Wissensgebiet für sich dar, sodass an dieser Stelle für Details auf weiterführende Literatur verwiesen sei.

Abstraktion aus System- und Entwicklersicht

Frühe Programmentwicklungen waren sehr stark auf die verwendete Rechnerplattform zugeschnitten, sodass sie mit viel Aufwand für die Verwendung auf nur leicht abweichenden Rechnerkonfigurationen vorbereitet werden mussten. Daher ist die Abstraktion von Ressourcen ein Konzept, das die einfache Verbreitung von Standardprogrammen sehr stark erleichtert. Ressourcenabstraktion heißt im Einzelnen:

- Abstraktion von der konkreten Instruktionssatzarchitektur, d.h. Prozessortyp
- Abstraktion von den konkreten Eigenschaften zentraler Rechnerhardware, d.h. Auslegung der Systemplatine (z.B. Zeitgeberhardware, physische Adressraumaufteilung)
- Abstraktion von den physikalischen und logischen Eigenschaften von Ein-/Ausgabeschnittstellen (z.B. IDE, SCSI, serielle und parallele Ein-/Ausgabe, Netzwerkanschluss)
- Abstraktion von den physikalischen und logischen Eigenschaften von angeschlossenen Peripheriegeräten (Drucker, Scanner, Festplatten, DVD-Laufwerk usw.)

Nützlich ist dies, indem Betriebssysteme portabel entwickelt werden können und sich so mit vergleichsweise geringem Aufwand auf eine andere Rechnerplattform anpassen lassen. Vor allem profitiert aber die Applikationsentwicklung davon, indem die Anschlussart und das konkrete Fabrikat eines Peripheriegerätes in der

Regel in den Anwendungen nicht mehr berücksichtigt werden muss. Es fällt dem Betriebssystem anheim, die notwendigen Datentransformationen vorzunehmen, sodass sich Programm und Peripherie verstehen. Neben den bereits erwähnten Abstraktionen von der konkreten *Rechnerplattform* und angeschlossener *Peripherie* findet das Abstraktionskonzept auch bei der eigentlichen Programmausführung seinen Einsatz. Neben der Abstraktion von *Ressourcen* existiert eine Abstraktion der *Programmausführung*.

- Wird ein Programm, das ja nichts anderes als eine Verfahrensvorschrift darstellt, auf einem Rechner ausgeführt, so wird dies als Prozessausführung bezeichnet.
- Eine Prozessausführung, auch als Prozessablauf bezeichnet, kann unterbrochen werden. Gründe dafür sind die parallele Ausführung von anderen Prozessen oder auch zeitkritische Interrupt-Serviceroutinen zur sofortigen Behandlung von Peripherieereignissen.
- Die Ausführung paralleler Prozesse erfolgt derart, dass für jeden Prozess (Programminstanz) der Eindruck entsteht, dass er den Rechner für sich alleine benutzt.

Die *Prozessabstraktion* (bzw. das Prozessmodell) erlaubt dem Betriebssystem, verschiedene Programminstanzen zu verwalten. Dies wird mit dem Begriff Prozessverwaltung bzw. Prozessmanagement umschrieben. Da die Programmausführung stets im Zusammenhang mit Ressourcen erfolgt, können diese nach Bedarf zugewiesen und entzogen werden. Dies wird innerhalb des Betriebssystems durch die Ressourcenverwaltung bzw. das Ressourcenmanagement erledigt. Eine besondere Bedeutung fällt dabei den zentralen Ressourcen CPU und Arbeitsspeicher zu. Für die Ausführung paralleler Prozesse werden hier Multiplexverfahren angewendet.

- Die CPU wird im Zeitmultiplex (*time multiplex*) einzelnen, ablaufbereiten Prozessen zugewiesen. Mögliche Multiplexverfahren werden in Abschnitt 3.4 erläutert.
- Der Arbeitsspeicher (*main memory*) wird im Raummultiplex (*space multiplex*) anfordernden Prozessen zugeteilt. Typische Zuteilungsverfahren werden in Abschnitt 7.5 angesprochen.

Eine Abstraktion beinhaltet ein generalisiertes Modell eines Betriebsmittels, das nur die unbedingt notwendigen Details enthält. Damit vereinfacht es die Anwendung des Betriebssystems und der darunter liegenden Hardware. Durch diese Generalisierung werden aber auch ganz spezifische Eigenschaften des Betriebsmittels versteckt. Will man diese nutzen, so muss die Abstraktion umgangen werden.

In der Geschichte der Betriebssysteme wurden unterschiedliche Formen der Abstraktion gefunden, die als Teil der Philosophie konkreter Produkte wahrnehmbar sind. Für die erfolgreiche Systemprogrammierung ist es wesentlich, dass

man ein gutes Verständnis der eingesetzten Abstraktionen erwirbt. Die Unix-Betriebssystemfamilie baut sehr stark auf den zwei Abstraktionen *Datei (file)* und *Prozess (process)* auf. So werden Peripheriegeräte einfach als *spezielle Dateien (special file)* behandelt. Für die Benennung von Interprozesskommunikationselementen wird ebenfalls auf das Dateisystem zurückgegriffen (*Named Pipes*, s.a. Folgekapitel). Die Windows-Betriebssysteme benutzen sogenannte *Systemobjekte* zur Beschreibung unterschiedlichster Ressourcen, wie Dateien, Geräte und Prozesse. Auch die Interprozesskommunikation wird mittels Systemobjekten abgewickelt. Details zu diesem Konzept sind in Abschnitt 2.2.4 enthalten.