Betriebssysteme

Vorlesungsskript

Prof. Dr. Hellberg

Inhaltsverzeichnis

1 Betriebssysteme

1.1 Überblick und Einordnung

Die Hardware eines Computersystems reicht im Allgemeinen nicht aus, um darauf effizient Anwendungen zu entwickeln bzw. ablaufen zu lassen.

Definition

Ein **Betriebssystem (Operating System)** stellt das Bindeglied zwischen der Hardware eines Computers einerseits und dem Anwender bzw. seinen Programmen andererseits dar. Es umfasst Programme, die zusammen mit den Eigenschaften des Computers „die Grundlage der möglichen Betriebsarten dieses Systems bilden und insbesondere die Abwicklung von Programmen steuern und überwachen".

Die Entwicklung von Betriebssystemen begann Mitte der 50er Jahre beim Einsatz von Computern der 2. Generation.

Ausführlich werden allgemeine Konzepte, Architekturen und Funktionsprinzipien von Betriebssystemen später behandelt.

1.2 Anforderungen und Aufgaben

Ziele beim Einsatz eines Betriebssystems (vgl. Betriebsart):

- „optimale" Auslastung der vorhandenen Betriebsmittel,
- Erfüllung von Nutzeranforderungen, z. B. Beachtung von Prioritäten.

Definition

Virtuelle (abstrakte) Maschine: Das Betriebssystem stellt sowohl für den Bediener als auch für den Programmierer *Dienste* zur Verfügung, die die Hardware in dieser Form nicht bieten kann.

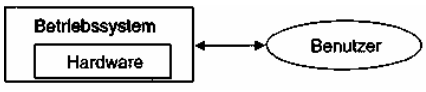


Bild 7.1 Betriebssystem als virtuelle Maschine

Abbildung 1: Betriebssystem als virtuelle Maschine

1.3 Aufgaben eines Betriebssystems

- abhängig von Einsatzgebiet, Zielen und speziellen Forderungen der Nutzer
- Anpassung der Leistungen der Hardware an die Bedürfnisse der Benutzer (Abstraktion der Hardware): Das Betriebssystem erweitert die Hardware-Funktionalität, z. B. durch Dateiverwaltung.
- Organisation, Steuerung und Kontrolle des gesamten Betriebsablaufs im System: Zuordnung der Benutzeraufträge zu entsprechenden Ausführungseinheiten (z.B. Prozesse), geeignete Ablaufplanung unter Beachtung möglicher Wechselwirkungen zwischen den Ausführungseinheiten.
- Verwaltung und ggf. geeignete Zuteilung von Betriebsmitteln (Ressourcen) an verschiedene Ausführungseinheiten.
- Kontrolle und Durchsetzung von Schutzmaßnahmen (z. B. Zugriffsrechte), insbesondere zur Sicherung der Integrität von Daten und Programmen, vor allem im Mehrnutzer-Betrieb.
- Protokollierung des gesamten Ablaufgeschehens im System, Erfassung und Analyse von Leistungsdaten zur Systemoptimierung.

1.4 Klassifikation

Neben den nachfolgend beschriebenen Kriterien gibt es weitere Bewertungsgrößen.

Die Betriebsart (Nutzungsform, Operation mode) eines Computers legt die Art und Weise der Kommunikation mit dem Benutzer fest.

1.4.1 Klassifikation nach Betriebsarten

- **Stapelverarbeitung (batch processing):** Bearbeitung einer Folge von Stapelaufträgen. Ein *Stapelauftrag (Job)* wird vom Benutzer mit allen erforderlichen Programmen, Daten und Anweisungen zur Ablaufsteuerung (Job Control Language, JCL) zusammengestellt. Er wird ohne Interaktion des Benutzers vollständig abgearbeitet.

  Beispiele: IBM OS/370, OS/390, MVS; BS 2000.

- **Dialogbetrieb (interactive processing):** Ständiger Wechsel zwischen Aktionen des Benutzers (z. B. Kommandoeingaben) und solchen des Systems (z. B. Kommandoausführung). Der Nutzer kann den Arbeitsablauf im Dialog jederzeit beeinflussen.

  Beispiele: MS-DOS, MS Windows 95/98/ME/NT/2000/XP; UNIX/Linux.

- **Echtzeitverarbeitung (real time processing):** Einsatz eines Computersystems zur Steuerung und Überwachung von technischen Prozessen, wobei durch das Echtzeit-Betriebssystem vor allem die Rechtzeitigkeit (Einhaltung von Zeitbedingungen) gesichert werden muss.

  Beispiele: VxWorks, VRTX, pSOS, LynxOS, Enea OSE, MS Windows CE, QNX, OSEK/VDX (speziell im Automobilbau).

- **Verteilte Verarbeitung (distributed processing):** Ein verteiltes System besteht aus mehreren miteinander gekoppelten Computern. Das Betriebssystem dient hier vorrangig der Verteilung von Daten, Ressourcen und Arbeitslast.

Beispiele für:

- verteilte Betriebssysteme: Amoeba, CHORUS, MACH
- Netzwerk-Betriebssysteme: Novell Netware
- Parallelrechner-Betriebssysteme: UNICOS, SPP-UX, KSR-OS

Viele Betriebssysteme bieten mehrere Betriebsarten an, ggf. durch ergänzende Komponenten (z. B. Dialogbetrieb und Stapelverarbeitung bei IBM MVS mit TSO, oder Dialog- und Echtzeitbetrieb bei VMS).

1.4.2 Klassifikation nach Anzahl der Benutzer

Anzahl der Benutzer, die zur selben Zeit bedient werden können.

- **Einzelnutzer-System (single user System):** Ein solches System führt keine Erkennung bzw. Verwaltung mehrerer Nutzer durch.

  Beispiele: MS-DOS, MS Windows 3.X/95/98/ME.

- **Mehrnutzer-System (multi user System):** Es steht (gleichzeitig) mehreren Benutzern zur Verfügung, z. B. über mehrere Terminals.

  Beispiele: UNIX, IBM OS/390, OS/400, BS2000, OpenVMS.

1.4.3 Klassifikation nach Anzahl der Aufträge

Anzahl der Aufträge, die gleichzeitig bearbeitet werden können:

- **Einzelprozess-System (single tasking System):** Das System kann jeweils nur einen einzigen Auftrag bearbeiten. Ein weiterer Auftrag wird erst nach Beendigung des aktuellen angenommen.

  Beispiele: CP/M, MS-DOS.

- **Mehrprozess-System (multi tasking System / multiprogramming System):** Es kann gleichzeitig mehrere verschiedene Aufträge verwalten und ggf. parallel oder zumindest quasi-parallel (zeitgeschachtelt) bearbeiten. Eine modifizierte Variante findet sich bei Multi-Threading-Systemen.

  Beispiele: MS Windows 95/98/ME/NT/2000/CE/XP, UNIX/Linux, IBM OS/390, OS/400, OS/2, BS2000, OpenVMS, VxWorks, VRTX, LynxOS, Enea OSE.

2 Architektur von Betriebssystemen

2.1 Entwurfskriterien

Kriterien beim Entwurf und der Implementierung von Betriebssystemen (sie dienen auch zur qualitativen Bewertung eines Systems):

- Modularität und Orthogonalität
- inkrementelle Erweiterbarkeit und Konsistenz
- statische oder dynamische Konfigurierung/Rekonfigurierung
- Skalierbarkeit
- Zuverlässigkeit und Fehlertoleranz
- Portierbarkeit
- Transparenz und Virtualisierung

☞ In der Praxis können nicht alle diese Größen zu einem Optimum geführt werden. Oft ist ein Kompromiss zu den Einsatzzielen und Benutzeranforderunben nötig.

2.2 Hauptkomponenten

Gestaltungsvarianten von Betriebssystemen:

- **Sammlung von Funktionen:** Das Betriebssystem stellt zwar Unterprogramme bereit, übt aber kaum Steuerungsfunktionen aus. Damit ist meist nur Einzelnutzer- und Einzelprozessbetrieb möglich.
- **Privilegierte Kontrollinstanz:** Zur Realisierung von Mehrnutzer- und/ oder Mehrprozessbetrieb muss das Betriebssystem die Steuerung wichtiger Vorgänge übernehmen. Dadurch ergibt sich eine gewisse Strukturierung aus den zu realisierenden Aufgaben.

Typische Funktionskomplexe universeller Betriebssysteme:

- Kommunikation mit der Umgebung
- Auftragsverwaltung
- Benutzerverwaltung
- Prozessverwaltung und -koordinierung
- Betriebsmittelverwaltung
- (Haupt-)Speicherverwaltung
- Eingabe-/Ausgabe-Steuerung
- Dateiverwaltung

☞ In Betriebssystemen für spezielle Einsatzbereiche sind einige Komponenten besonders ausgeprägt oder fehlen teilweise. Letzteres betrifft auch Betriebssysteme für mobile Computer (z. B. EPOC, PalmOS, MS Windows CE).

Definition

Als **Systemsoftware** werden alle Programme zusammengefasst, die eine effiziente und komfortable Benutzung eines Computers ermöglichen. Neben dem Betriebssystem gehören dazu ergänzende, hardwareunabhängige Dienst- und Hilfsprogramme.

Systemsoftware umfasst u. a.:

- **Programmierumgebungen** (integrierte Software-Entwicklungssysteme) mit Editoren, Compiler bzw. Interpreter, Verbinder (Linker) und Lader, Testhilfen (Debugger) sowie entsprechenden Bibliotheken und weiteren Verwaltungsprogrammen.
- **Dienstprogramme** (tools, Utilities) für typische Arbeitsvorgänge (z. B. Suchen, Kopieren oder Sortieren von Daten) sowie zur Installation/Konfigurierung und Administration. Vielfach werden dazu auch Browser eingesetzt.
- Programme zur Realisierung spezieller Betriebsformen.

Man unterscheidet beim interaktiven Betrieb mitunter zwischen *Teilnehmerbetrieb* (die Benutzer arbeiten hierbei mit individuellen Programmen) und *Teilhaberbetrieb* (die Nutzer arbeiten gleichzeitig mit individuellen Daten, jedoch mit demselben Programm). Beim Spezialfall *Transaktionsverarbeitung (transaction processing, TP)* werden *TP-Monitore* benutzt.

2.2.1 Architekturmodelle

Architekturmodelle verdeutlichen die Anordnung der Komponenten des Betriebssystems und ihr funktionales Zusammenwirken. Sie geben auch Aufschluss über die Portabilität des Systems. Bekannte Modelle sind:

- **Monolithische Architektur:** Alle wesentlichen Komponenten des Systems sind zu einem homogenen Gebilde zusammengefügt, das zwar u. U. effizient, aber nicht flexibel anpassbar ist.
- **Kern-Schale-Architektur:** Das System besteht aus dem privilegierten *Kern (kernel),* der die wichtigsten Komponenten vereint (z.B. Prozessverwaltung), und einer *Schale (shell)* für ergänzende Bestandteile (z.B. Kommando-Interpreter). Typischer Vertreter dieses Modells ist UNIX.

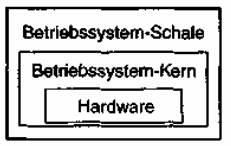


Bild 7.2 Kern-Schale-Modell

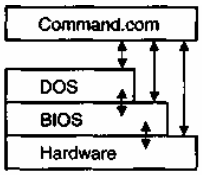


Bild 7.3 Quasi-konsistente Schichten (Beispiel MS-DOS, vereinfacht)

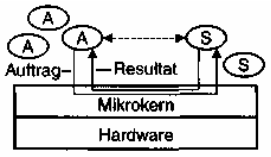
Abbildung 2: Kern-Schale-Modell und Quasi-Konsistente Schichten.

- **Hierarchische Schichten (Mehrschichtenmodell):** Das System wird modularisiert und in einzelne Schichten geteilt. Zwischen diesen sind *Schnittstellen* definiert, so dass sie austauschbar sind. Sie können mit abgestuften Privilegien ausgestattet sein. Diese Architektur ist weit verbreitet, z. B. bei MS-DOS, OS/2, OpenVMS.

  ☞ In der Hierarchie bietet jede Schicht ihre Dienstleistungen der (den) darüber liegenden Schicht(en) an und greift gleichzeitig zur Erfüllung ihrer Aufgaben auf die Dienste der tiefer liegenden Schicht(en) zurück. Die unterste Schicht ist im Allgemeinen die Hardware, die oberste oft ein Kommandointerpreter. *Konsistente Schichten* sind leicht austauschbar, das strenge Durchlaufen aller Schichten führt aber oft zu Effizienzverlusten. Bei *quasi-konsistenten Schichten (Treppenstufenmodelle*) kann dagegen bei Bedarf auf verschiedene tiefer liegende Schichten zugegriffen werden. Dies führt jedoch u. U. zu unkontrollierbaren Abläufen.

- **Mikrokern (micro kernel):** Er bildet nur noch eine Art Infrastruktur mit minimalem Funktionsumfang. Alle anderen Betriebssystemfunktionen werden durch Systemprozesse außerhalb des Kerns erbracht, die flexibel modifiziert oder erweitert werden können. Damit lassen sich günstig Client-Server-Modelle realisieren. Mikrokerne findet man z.B. bei MACH, CHORUS, QNX/Neutrino, z.T. bei MS Windows NT.

  ☞ Mikrokerne enthalten nur noch elementare Funktionen, z. B. zur Speicherverwaltung, IPC, Prozessverwaltung, Scheduling-Mechanismus, sowie einige hardwarenahe E/A-Funktionen. Systemprozesse realisieren dagegen z. B. die Dateiverwaltung



A Anwendungsprozess
S Server (Dienstleistungsprozess)

Bild 7.4 Mikrokern und ergänzende Systemprozesse (Server-Prozesse)

Abbildung 3: Mikrokern und ergänzende Systemprozesse (Server-Prozesse)

- **Virtuelle Maschinen**: Durch konsequente Abstraktion und Virtualisierung wird auf der Hardware zunächst ein „Basis-Betriebssystem“ *(virtual machine monitor)* aufgesetzt, auf dem jedoch keine klassischen Anwendungen ablaufen, sondern (wiederum) virtuelle Maschinen in Form verschiedener Betriebssysteme. Diese sind damit von der realen Hardware völlig entkoppelt.

  Beispiel: IBM VM/370, auf dem verschiedene Betriebssysteme laufen können (z. B. OS/MVS, MVS/XA, MVS/ESA, AIX). Die Idee der virtuellen Maschine spielt heute auch im Zusammenhang mit Java eine Rolle.

  ☞ Beim Entwurf als auch bei der Implementierung vieler Betriebssysteme werden auch Prinzipien der Objektorientierung berücksichtigt.

2.2.2 Schnittstellen

Die Nutzung der Dienste eines Betriebssystems erfolgt im Allgemeinen entweder über die Benutzungsschnittstelle oder über die Programmierschnittstelle.

Definition

Die **Benutzungsschnittstelle (user interface)** dient zur Interaktion des Bedieners mit dem System. Sie wird mit Hilfe eines Kommando-Interpreters realisiert, der Bedienhandlungen auswertet.

Interaktionsmöglichkeiten für Kommando-Interpreter:

- **Kommandosprachen**: Aufträge (Kommandos) werden durch Texteingaben hinter einem Bereitschaftszeichen *(prompt)* ausgelöst.

  ☞ Viele Kommando-Interpreter bieten umfangreiche Programmierkonstrukte an, um komplexe Arbeitsabläufe durch Kommandofolgen zu automatisieren (z. B. Batch Files unter MS-DOS, Shell Scripts unter UNIX).

- **Text-Menüs, Bildschirm-Masken** erlauben eine gute Bedienerführung mit strengen Eingabekontrollen.

- **Grafische Benutzungsoberfläche (Graphical User Interface, GUI):** Die Bedienung erfolgt einfach mittels grafischer Eingabegeräte, Ausgaben erscheinen z. B. in Bildschirmfenstern (Windows).

  ☞ Beim „Schreibtisch-Modell“ (desktop, workplace) sind Objekte (z. B. Dateien) durch *Sinnbilder (icons,* Symbole) repräsentiert. Arbeitsvorgänge werden auf der grafischen Oberfläche visualisiert (z.B. Kopieren einer Datei durch Verschieben eines Symbols mit der Maus, *„drag and drop“).*

  ☞ Die grafische Benutzungsschnittstelle kann fester Bestandteil des Betriebssystems (z. B. bei MS Windows 95/98/ME/NT) oder aber (frei wählbare) Systemkomponente (z. B. bei UNIX/Linux) sein. Der letzte Ansatz bietet hohe Flexibilität durch beliebige *Window-Manager,* die Aussehen, Bedienung und „Verhalten“ der Fenster *(„look and feel“)* bestimmen und auf einem (standardisierten) Fenstersystem basieren, z.B. OSF/Motif auf X-Windows.

Definition

Die **Programmierschnittstelle** (Application Programming Interface, **API**) definiert in ihrer Syntax und Semantik die Funktionen des Betriebssystems in Form von Systemdiensten (system Services).

Diese Systemdienste stehen für den Programmierer einer Anwendung im Allgemeinen in Funktionsbibliotheken (z. B. DLLs bei MS Windows) für die jeweils benutzte Programmierumgebung bereit und sind z. T. standardisiert (POSIX).

☞ Der Aufruf von Systemdiensten erfolgt unter Nutzung spezieller Unterbrechungsmechanismen bzw. privilegierter Befehle (Trap, Supervisor Call SVC) und ist Bestandteil des hardwareabhängigen *Application Binary Interface (ABI).*

☞ Auch die Funktionen eines Window-Systems können von Anwendungsprogrammen benutzt werden, um diese mit grafischen Bedienelementen auszustatten. Ein solches mehrschichtiges Grafiksubsystem bietet verschiedene Funktionen *(intrinsics)* zur Realisierung von typischen Grafikelementen *(widgets),* z.B. für Schaltknöpfe *(buttons)* oder Schiebebalken *(scroll bars)* an.

3 Prozesse

3.1 Prozesskonzept

Definition

Ein **Programm (program)** ist eine statische Folge von *Anweisungen* in einer Programmiersprache unter Nutzung von *Daten.* Es dient zur Codierung eines *Algorithmus* und liegt im Allgemeinen in Form einer Datei vor.

Definition

Ein (sequentieller) **Prozess (process, auch: task)** ist eine dynamische Folge von Aktionen (Zustandsänderungen), die durch Ausführung eines Programms auf einem Prozessor zustande kommt. Ein Prozess ist insbesondere durch seinen *zeitlich veränderlichen Zustand* gekennzeichnet. Er wird im Betriebssystem infolge eines Auftrags erzeugt.

☞ Im Stapelbetrieb umfasst der Begriff *Job* einen (großen) Auftrag, der aus mehreren Teilaufgaben (tasks) und zugehörigen Daten besteht.

Ein **Programm** kann mehreren Prozessen zugeordnet sein. Ein Prozess ist aus der *Sicht des Benutzers* als *„Aktivitätsträger“* bzw. *„Ausführungseinheit“* charakterisiert. Die Aktivität eines Prozesses zeigt sich in der Transformation von Eingangs- in Ausgangsoperanden, deren Werte z. B. in CPU-Registern oder Speicherzellen vorliegen können.

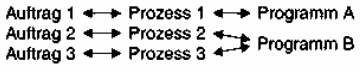


Bild 7.5 Beispiele für den Zusammenhang zwischen Auftrag, Prozess und Programm

Abbildung 4: Beispiele für den Zusammenhang zwischen Auftrag, Prozess und Programm

Adressraum. Er umfasst alle einem Prozess zugeordneten Speicherbereiche für Programmcode, statische sowie dynamische Daten und Stack. Das Betriebssystem kann dafür einen *Zugriffsschutz* realisieren.

Der Auftrag zur Erzeugung eines Prozesses kann u. a. ausgelöst werden:

- von einem Bediener, z. B. durch Kommandoeingabe
- direkt aus der Umgebung, z. B. durch Interrupts
- von anderen Prozessen, z. B. durch Aufruf von Systemdiensten

☞ Nach Erfüllung seines Auftrages wird ein Prozess i. d. R. beendet, Ausnahmen bilden periodische Prozesse (z.B. Dienstleistungs- bzw. Server-Prozesse), die nach Erfüllung eines Auftrages auf neue Aufträge warten.

3.2 Verwaltung paralleler Prozesse

Definition

Wenn zu einem Zeitpunkt mehrere Prozesse im System existieren, d.h., wenn sie sich zwischen Erzeugung und Abschluss befinden, werden sie **parallele Prozesse** genannt, unabhängig davon, ob ein Ein- oder Mehrprozessorsystem zugrunde liegt.

Aus der *Sicht des Betriebssystems* stellen Prozesse *Verwaltungseinheiten* bzw. *Objekte* dar, die im Prinzip parallel ablaufen können (Multi-Tasking). Steht jedoch nur ein Prozessor zur Verfügung, kann die Bearbeitung der Prozesse nur quasi-parallel erfolgen. Diese **Nebenläufigkeit (concurrency)** wird z.B. durch eine geeignete *zeitliche Verschachtelung* der Prozesse erreicht (Scheduling).

Das Betriebssystem verwaltet sowohl **Anwendungsprozesse (Benutzerprozesse)** als auch **Systemprozesse** (zur Ausführung von Betriebssystemleistungen, die ggf. aus dem Betriebssystem-Kern ausgelagert wurden). Alle diese Prozesse arbeiten im Allgemeinen im so genannten **Benutzermodus (user mode)**, in dem nur nichtprivilegierte Befehle des Prozessors verfügbar sind.

Die Routinen des Betriebssystem-Kerns laufen dagegen im **Systemmodus (system mode)**, in dem auch privilegierte Befehle zulässig sind. Die unerlaubte Verwendung solcher Befehle im Benutzermodus wird von vielen Betriebssystemen als Fehler erkannt.

3.3 Prozesszustände

Zu einem Zeitpunkt können nur so viele Prozesse im System wirklich voranschreiten, wie Prozessoren zur Verfügung stehen. Die Prozessverwaltung basiert deshalb auf einem **Zustandsmodell**, das die **Zustände**, die ein Prozess während seiner Existenz (ggf. auch mehrfach) annehmen kann, und die zulässigen **Zustandsübergänge** enthält.

Typische Grundzustände eines Prozesses:

- **aktiv (auch rechnend, running):** Der betreffende Prozess hat einen Prozessor zugeteilt bekommen und schreitet voran (er „läuft“).
- **bereit (auch laufbereit, ready):** Der Prozess wartet nur noch auf die Zuteilung eines Prozessors (keine weiteren Wartebedingungen).
- **wartend (auch blockiert, suspendiert, waiting):** Der Prozess muss auf die Erfüllung (mindestens) einer Wartebedingung warten.
- **nicht existent (non existing):** „Formaler“ Zustand für Prozesse, die noch nicht oder nicht mehr im System existieren.

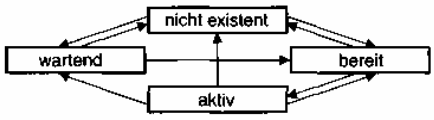


Bild 7.6 Beispiel eines Prozess-Zustandsmodells

Abbildung 5: Beispiel eines Prozess-Zustandmodells

☞ In der Literatur findet man weitere Bezeichnungen für Prozesszustände. Mitunter wird „aktiv“ als Zusammenfassung der Zustände „rechnend“ und „bereit“ benutzt. In realen Systemen besteht der Zustand „wartend“ oft aus vielen Unterzuständen

Beispiele sinnvoller (zulässiger) Zustandsübergänge:

- **nicht existent → wartend (oder bereit):** Ein Prozess wird erzeugt.
- **wartend bereit:** Alle Wartebedingungen für den Prozess wurden erfüllt, er wird laufbereit und erwartet die Prozessorzuteilung.
- **bereit → aktiv:** Der Prozess erhält einen Prozessor zugeteilt (Scheduling) und kann in seinem Ablauf voranschreiten.
- **aktiv → bereit:** Der aktive Prozess wird verdrängt, ihm wird der Prozessor zugunsten eines anderen Prozesses entzogen.
- **aktiv → wartend:** Im Ablauf des Prozesses tritt eine Wartebedingung auf (z. B. Warten auf Eingabedaten), er muss deshalb auf die Erfüllung dieser Bedingung warten und verliert den Prozessor.
- **aktiv → nicht existent:** Der Prozess hat seine Arbeit beendet und meldet sich beim Betriebssystem ab.
- **bereit (oder wartend) → nicht existent:** Der Prozess wird von einem anderen Prozess oder vom Betriebssystem abgebrochen und entfernt.

Die Zustandsübergänge werden durch interne Funktionen des Betriebssystem-Kerns realisiert. Für die Prozesse (bzw. den Programmierer) stehen diese nur innerhalb komplexerer Systemdienste am API zur Verfügung.

3.4 Prozessbeschreibung

Aus der Sicht des Benutzers ist ein Prozess vor allem durch den zugehörigen Auftrag bzw. sein Programm gekennzeichnet.

Definition

Zur Verwaltung im Betriebssystem besitzt jeder Prozess einen eigenen **Kontext (context)**, der alle relevanten Informationen über den Prozess enthält

Typische Bestandteile des Prozess-Kontextes:

- **Benutzer-Kontext:** Inhalt des (virtuellen) Adressraumes des Prozesses, quasi das „Speicherabbild" des Prozesses.
- **Hardware-Kontext (Register-Kontext, „Registerabbild"):** Inhalt aller Prozessorregister (Befehlszähler, Stackpointer usw.).
- **System-Kontext:** Inhalt der betriebssysteminternen Verwaltungsdaten über den Prozess.

Zum System-Kontext gehören u. a. folgende Informationen:

- **Prozess-Identifikator:** Name oder Nummer des Prozesses,
- **Auftrag/Auftraggeber:** z. B. Pfadname des benutzten Programms, Name des auftraggebenden Benutzers,
- **Ablaufpriorität**, sofern sie vom Betriebssystem berücksichtigt wird,
- **Verwandschaftsbeziehungen**: Identifikatoren von Vater- und Sohnprozessen, falls eine Prozesshierarchie verwaltet wird (z. B. UNIX),
- **Zustand**: Zustand des Prozesses (Prozess-Zustandsmodell),
- **Zugriffsrechte** des Prozesses für Dateien, Speicherbereiche usw.,
- **Betriebsmittel**: Beschreibung angeforderter und bereits zugeteilter Betriebsmittel, ggf. Auslastung (Betriebsmittelkonten).

☞ Die Speicherung des Kontextes erfolgt im Allgemeinen mit Hilfe einer komplexen Datenstruktur, dem **Prozesskontrollblock** (Process Control Block, **PCB**).

3.5 Prozesswechsel

Die Änderung der Prozessorzuteilung bedeutet praktisch einen **Prozesswechsel (task switch)** bzw. **Kontextwechsel (context switch):** Der aktive Prozess gibt den Prozessor entweder *freiwillig* ab oder bekommt ihn *entzogen.* Dieser Vorgang wird vom **Prozessumschalter (dispatcher)** gesteuert. Die Auswahl des Prozesses, dem der Prozessor als nächstes zugeteilt wird, erfolgt durch den **Scheduler** anhand einer gewählten Strategie.

☞ Prozessumschalter und Scheduler benutzen als zentrale Komponenten des Betriebssystemkerns das zugrunde liegende Zustandsmodell. Ein Prozesswechsel kann i. d. R. nur nach einer Unterbrechung stattfinden (Hardwareunterbrechung/Interrupt oder Softwareunterbrechung bzw. Systemdienstaufruf).

Ein Prozesswechsel erfordert im Betriebssystem u. a. folgende Aktionen:

1. Sicherung des gesamten Kontextes des unterbrochenen Prozesses.
2. Änderung des Zustandes des unterbrochenen Prozesses in „bereit“ oder „wartend“ in Abhängigkeit vom Grund der Unterbrechung.
3. Scheduling: Auswahl des nächsten zu aktivierenden Prozesses.
4. Änderung des Zustandes dieses Prozesses von „bereit“ in „aktiv“.
5. Wiederherstellung des (gesicherten) Kontextes dieses Prozesses. Durch Laden der Prozessorregister wird dieser Prozess fortgesetzt.

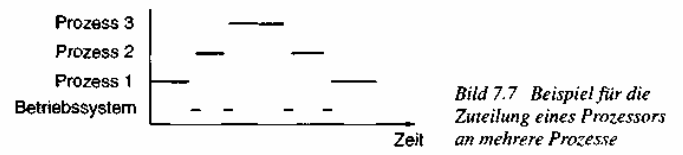


Bild 7.7 Beispiel für die Zuteilung eines Prozessors an mehrere Prozesse

Abbildung 6:Beispiel für die Zuteilung eines Prozessors an mehrere Prozesse

☞ Die PCBs werden in Vektoren/Tabellen und/oder Listen verwaltet. Bei Listen erfolgt der Zustandsübergang eines Prozesses durch Umkettung seines PCB aus einer Liste in eine andere. Die vom System dafür benötigte *Umschaltzeit* (context switch time) dient oft zur Bewertung von

Betriebssystemen.

Kernsperren verhindern ggf. eine Unterbrechung von Abläufen innerhalb des Betriebssystem-Kerns (z. B. während des Prozesswechsels).

Eine *vollständige Kernsperre* ist einfach und sicher, aber unproduktiv und setzt voraus, dass im Kern keine Wartezustände auftreten. Die beste Reaktionsfähigkeit besteht *ohne Kernsperren,* dazu muss jedoch der ganze Kern reentrant (wiedereintrittsfähig) sein. Möglich sind auch *teilweise Sperren* für einige besonders kritische Teile des Kerns.

„**Leerprozess**“ (**idle task**, Dummy-Prozess). Er besteht meist nur aus einer Warteschleife, so dass er jederzeit problemlos verdrängt werden kann. Er wird nur dann benötigt, wenn beim Scheduling kein anderer Prozess laufbereit ist. Ansonsten müsste innerhalb des Betriebssystem-Kerns (im Systemmodus) „aktiv gewartet werden *(„busy waiting“).*

Systemdienste, die *direkt* die Prozessverwaltung beeinflussen, umfassen insbesondere Erzeugung/Start (oft mit Vererbung von Eigenschaften des „Vaterprozesses“ an den „Sohnprozess“) und Beendigung von Prozessen sowie die Änderung von Eigenschaften (z. B. Priorität). Außerdem greifen Dienste zur Koordinierung *indirekt* in die Prozessverwaltung ein.

Systemdienst	**Unix**	**Windows NT (Win32-API)**
Prozess erzeugen	fork(); exec()	CreateProcess()
Prozess (sich selbst) beenden	Exit()	ExitProcess()

Tabelle 7.1 Beispiele für Systemdienste zur Prozesserzeugung und -beendigung

3.6 Threads

Wegen ihres umfangreichen Kontextes sind Prozesse oft „schwergewichtig“ (z.B. in UNIX). Um feinere Parallelität, insbesondere auch *innerhalb einer Anwendung* zu erreichen, wurden „leichtgewichtige“ Prozesse bzw. Threads entwickelt.

Definition

Ein **Thread** ist ein *Aktivitätsträger* (sequentieller *„Ausfiihrungsfaden')* mit minimalem Kontext (Stack und Register) innerhalb einer *Ausführungsumgebung* (Prozess). Jeder Prozess besitzt in diesem Fall mindestens einen (initialen) Thread. Alle Threads, die zu ein und demselben Prozess gehören, benutzen denselben Adressraum sowie weitere Betriebsmittel dieses Prozesses gemeinsam.

Durch die gemeinsame Umgebung verringert sich der Aufwand beim Wechsel zwischen Threads innerhalb *desselben* Prozesses erheblich. Andererseits lassen sich Zugriffe der Threads innerhalb eines Prozesses nicht mehr anhand von Adressraumgrenzen kontrollieren, was zugunsten höherer Parallelität meist akzeptiert wird. Oft ist zusätzliche Synchronisation erforderlich. Einige Systeme erlauben die Vereinbarung von lokalen Speicherbereichen *(Thread Local Storage TLS).* Ein Wechsel zwischen Threads, die zu *verschiedenen* Anwendungen gehören, erfordert jedoch noch immer einen „teuren“ Adressraumwechsel und damit höheren Aufwand.

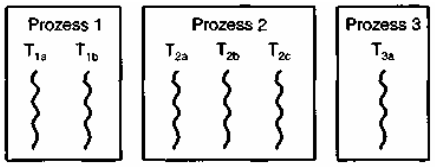


Bild 7.8 Beispiel für Anwendungen/Prozesse mit mehreren Threads (T)

Abbildung 7: Beispiel für Anwendungen / Prozesse mit mehreren Threads

In der Praxis eignen sich viele Anwendungs- und Systemprozesse dafür, in Form paralleler Threads implementiert zu werden. Geeignete Kooperationsstrukturen entsprechen grundsätzlich denen paralleler Prozesse. Threads werden von verschiedenen Betriebssystemen z.T. direkt unterstützt (z. B. MACH, MS Windows NT/2000/XP, IBM OS/2). Für weitere Betriebssysteme (u. a. für UNIX) werden Thread-Bibliotheken angeboten, mit denen sich Threads außerhalb des Betriebssystems auf Anwendungsebene implementieren lassen.

4 Koordinierung paralleler Prozesse

4.1 Wechselwirkungen zwischen Prozessen

Definition

Unter **Koordinierung** versteht man die geeignete Beeinflussung der Abläufe in einem System paralleler Prozesse (bzw. Threads), um *Wechselwirkungen* (z. B. *Konkurrenz* oder *Kooperation)* zwischen ihnen zu beherrschen. Dies erfolgt im Allgemeinen durch den Programmierer mittels *Synchronisation* und/oder *Kommunikation.*

Definition

Synchronisation (synchronization) bedeutet eine solche Beeinflussung paralleler Prozesse, dass sie hinsichtlich ihres zeitlichen Ablaufs in eine gewisse Reihenfolge gebracht werden, die zufällig oder vorbestimmt sein kann.

Ein **Verarbeitungsschritt (sequentieller Teilprozess)** umfasst alle Operationen im Abstand zwischen zwei Wechselwirkungen eines Prozesses. **Determiniertheit** eines Systems paralleler Prozesse besteht dann, wenn die globalen Ergebnisse der Verarbeitungsschritte aller Prozesse unabhängig von ihrer Ablaufgeschwindigkeit sind. Ein solches Prozesssystem liefert bei gleichen Eingabedaten und gleichen inneren Zuständen für alle möglichen Ablauffolgen stets dieselben Ergebnisse.

4.2 Konkurrenz zwischen Prozessen

Konkurrenz tritt auf, wenn mehrere Prozesse unabhängig voneinander zeitweilig dasselbe Betriebsmittel exklusiv benutzen wollen.

Definition

Durch **wechselseitigen Ausschluss (mutual exclusion)** bzw. Sperrsynchronisation wird aus der Menge der um das selbe Betriebsmittel konkurrierenden Prozesse *einem* der Zugriff erlaubt und *alle anderen* werden vorübergehend von der Nutzung dieses Betriebsmittels ausgeschlossen. Diejenigen Verarbeitungsschritte eines Prozesses, in deren Verlauf ein exklusives Betriebsmittel genutzt werden soll und die nicht parallel zu denselben Schritten eines anderen Prozesses ausgeführt werden dürfen, bilden einen kritischen Abschnitt (critical region, critical section).

Beispiel: Ein typisches Beispiel für einen kritischen Abschnitt stellen Schreibzugriffe zu einem gemeinsamen Speicherbereich dar. Für die Sperrsynchronisation ist irrelevant, welcher der Prozesse seinen kritischen Abschnitt *zuerst* betritt.

Wechselseitiger Ausschluss wird meist durch **Protokolle (Steueralgorithmen)** realisiert, die vor dem Betreten *(Vorprotokoll)* und nach dem Verlassen des kritischen Abschnitts (*Nachprotokoll)* abgearbeitet werden und folgenden **Qualitätsanforderungen** genügen sollten:

- **Sicherheit:** Zu einer Zeit darf sich höchstens ein einziger Prozess in einem kritischen Abschnitt desselben Betriebsmittels befinden.
- **Lebendigkeit:** Jedem Prozess, der einen kritischen Abschnitt betreten will, muss dies irgendwann gelingen.

Mittel zur Implementierung des wechselseitigen Ausschlusses:

- **Unterbrechungssperren**, **Verdrängungssperren**: Sie verhindern „radikal“ ungewollte Prozesswechsel (und ggf. auch Interrupts).
- **Sperrvariablen (spin locks):** Sie beschreiben einen (kurzen) kritischen Abschnitt durch ihren Wert als „frei“ bzw. „belegt“.

Prozesse müssen diese Variable im Vorprotokoll abfragen und ggf. durch zyklische Nachfrage *(aktives Warten, busy waiting)* verharren. Die Zugriffe erfordern komplexe unteilbare („atomare“) Maschinenbefehle wie z.B. „Test_and_Set“. Andernfalls müssen in den Prozessen aufwendige Algorithmen implementiert werden (Dekker, Dijkstra, Peterson,*)*.

- **Semaphore**: Spezielle Koordinationselemente, die eine Steuervariable und eine Warteschlange für Prozesse integrieren („Ampeln“).

Die Steuervariable kann vom Typ Boolean *(binäre Semaphore,* auch *Mutex)* oder Integer sein *(allgemeine/zählende Semaphore)*. Sie gibt an, ob bzw. wie viele Prozesse den kritischen Abschnitt noch betreten dürfen. Semaphore werden im Betriebssystem implementiert, wartepflichtige Prozesse geraten in die Warteschlange. Die Protokolloperationen (nach Dijkstra p(S) und v(S),) sind über Systemdienste am AP1 nutzbar, die aus der Sicht der Prozesse *unteilbar (atomar)* ablaufen. Einige Betriebssysteme bieten weiterhin Dienste zur Abfrage der Steuervariablen ohne Wartepflicht und zur Anforderung mit Zeitüberwachung beim Warten. Auch ein korrekter Einsatz von Semaphoren birgt die Gefahr von *Verklemmungen*.

- **Hochsprachliche Koordinationsmittel**: Sie sind leichter zu benutzen und durch Systemsoftware (z. B. Compiler) kontrollierbar.

Konzepte wie die so genannten Monitore gibt es nur in experimentellen Programmiersprachen oder mit Modifikationen (z. B. in Java).

Systemdienst	Unix	Windows NT (Win32-API)
Semaphor erzeugen	Semget ()	CreateSemaphore(), OpenSemaphore
Semaphor anfordern	Semop()	WaitForSingleObject()
Semaphor freigeben	Semop()	ReleaseSemaphore()
Semaphor vernichten	Semctl()	CloseHandle()

Tabelle 7.2 Beispiele für Semaphor-Systemdienste in UNIX und Windows NT

Beispiel: Einsatz eines binären Semaphors *S* zum wechselseitigen Ausschluss eines kritischen Abschnitts (kA) in zwei Prozessen P_1 und P_2.

Innerhalb von *p(S)* wird anhand der Steuervariablen geprüft, ob der kA frei ist. Ist dies der Fall, wird die Steuervariable auf „belegt" gesetzt und der Prozess darf den kA betreten. Andernfalls wird er in die Warteliste bezüglich *S* eingereiht und ein anderer, laufbereiter Prozess wird aktiviert. Beim Verlassen des kA mittels *v(S)* wird geprüft, ob weitere Prozesse in der Warteliste dieses Semaphors stehen. Ist das der Fall, wird einer von ihnen freigesetzt und kann den kA betreten. Ansonsten wird die Steuervariable wieder auf „frei" gesetzt. A_{11}, A_{12}, A_{21} und A_{22} sind irgendwelche Verarbeitungsschritte außerhalb des kA. Unabhängig davon, zu welchem Zeitpunkt P_1 und P_2 ihre Operation *p(S)* erreichen, wird darin gesichert, dass stets maximal ein Prozess im kA arbeitet. Bei quasi-gleichzeitiger Anforderung wird im Allgemeinen ein Prozess zufällig ausgewählt.

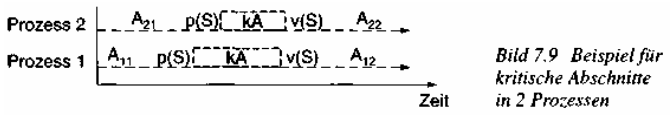


Bild 7.9 Beispiel für kritische Abschnitte in 2 Prozessen

Abbildung 8: Beispiel für kritische Abschnitte in 2 Prozessen

4.3 Kooperation von Prozessen

Kooperation von Prozessen bedeutet im Allgemeinen ein zielgerichtetes Zusammenwirken, z. B. zur arbeitsteiligen Erfüllung einer komplexen Aufgabe.

Definition

Ereignissynchronisation ist eine Form der Koordinierung paralleler Prozesse mit dem Ziel, gewünschte *Präzedenzen (Reihenfolgebeziehungen)* zu realisieren. Diese ergeben sich aus der zu lösenden Aufgabe und hängen von bestimmten *Ereignissen* in den Prozessen ab.

☞ Präzedenzen *innerhalb eines* Prozesses ergeben sich aus der Sequenz seiner Programmausführung und bedürfen daher keiner zusätzlichen Synchronisation!

Beispiel: Erzeuger-Verbraucher-Problem (producer consumer problem): Von zwei zyklisch arbeitenden Prozessen „erzeugt" der eine Daten (z. B. durch Eingaben) und der zweite „verbraucht" Daten (z. B. durch Visualisierung). Beide Prozesse sollen so miteinander kooperieren, dass der erste seine Daten über einen gemeinsamen Puffer an den zweiten übergibt. Ziel ist dabei die unbedingte Einhaltung von Präzedenzen (der Puffer sollte erst ausgelesen werden, nachdem er vollständig gefüllt und umgekehrt erst wieder gefüllt werden, nachdem er vollständig geleert wurde), unabhängig von der Geschwindigkeit beider Prozesse. Ansonsten soll maximale Parallelität der Prozesse gewahrt bleiben.

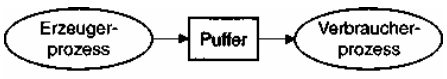


Bild 7.10 Erzeuger-Verbraucher-Problem

Abbildung 9: Erzeuger-Verbraucher-Problem

Typische **Grundstrukturen** der Ereignissynchronisation:

- **Folge**: Ein Verarbeitungsschritt eines Prozesses kann erst begonnen werden, nachdem ein bestimmter Schritt eines zweiten Prozesses abgeschlossen wurde.
- **Gleichlauf**: Zwei Prozesse warten an einer bestimmten Stelle wechselseitig aufeinander.

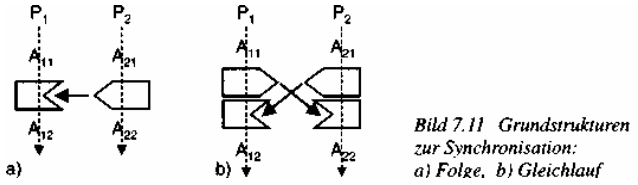


Bild 7.11 Grundstrukturen zur Synchronisation: a) Folge, b) Gleichlauf

Abbildung 10: Unbekannt.

Einfache Mittel zur **Ereignissynchronisation** (prozedurorientierte Synchronisation mit Hilfe globaler Datenstrukturen):

- **(private) Semaphore**: Hierbei können die Semaphor-Operationen von einem Prozess zum „Signalisieren" und von einem anderen Prozess zum Prüfen (und ggf. Warten) benutzt werden.
- **Ereignisvariablen (event counter/flags):** Datenstrukturen mit binärer oder zählender Steuervariable und Prozess-Warteliste sowie Operationen zum Warten auf ein Ereignis, zum Signalisieren eines Ereignisses mit Freisetzung aller wartenden Prozesse und zur Abfrage des Wertes der Steuervariablen.
- **Signale:** Sie stellen eine asynchrone Prozessunterbrechung *(Alarm, Software-Interrupt)* dar, die insbesondere von anderen Prozessen wie auch vom Betriebssystem zur Meldung besonderer Situationen ausgelöst werden kann. Der betroffene Prozess kann darauf u. a. mit einer individuellen Signalbehandlungsroutine reagieren.

Definition

(Interprozess-)Kommunikation (Interprocess Communication, IPC) bedeutet einen Austausch von Informationen zwischen kooperierenden parallelen Prozessen. Häufig werden dazu Nachrichten (Botschaften, messages) benutzt (nachrichtenbasierte Synchronisation).

Kriterien zur Klassifizierung von Kommunikationsverfahren:

- **Art der Nachricht:** Übertragung der gesamten Information oder nur eines Verweises bzw. Zugriffsrechts darauf. Der Nachrichtenkopf kann zusätzliche Angaben (z. B. Prioritäten, Zeitfristen) enthalten.
- **Beteiligung des Betriebssystems:** Bei *impliziter Kommunikation* regeln die Prozesse den Nachrichtenaustausch selbst. *Explizite Kommunikation* erfolgt unter Steuerung des Betriebssystems.

- **Zahl der kommunizierenden Prozesse:** Bei *direkter Übermittlung* sind zwischen Sende- und Empfangsprozessen Relationen der Form 1 : 1 *(Uni-cast), m :* 1 oder 1 : *n* (hierbei *Broadcast* an alle Empfänger oder *Multicast* an einige Empfänger) realisierbar. Bei indirekter Übermittlung lassen sich *(m :* n)-Formen bilden.
- **Synchronisation:** Der Nachrichtenaustausch kann synchron (blocking send) oder asynchron (no-wait send) erfolgen. Im ersten Fall müssen Sender und Empfänger vor Beginn der Übermittlung mittels Gleichlauf synchronisiert werden, die Übertragung kann nach einer Rückantwort des Empfängers (bidirektionale Kommunikation) oder ohne (unidirektional) erfolgen. Dazu existieren z.T. Konstrukte *(‚Rendezvous")* in höheren Programmiersprachen (z. B. Occam, Ada). Bei asynchronen Verfahren ist eine Abstimmung von Sender und Empfänger nicht nötig.

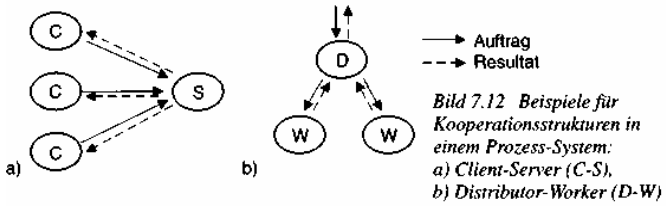


Abbildung 11: Beispiele für Kooperationsstrukturen in einem Prozess-System

Mittel zur Interprozesskommunikation:

- **gemeinsame Dateien:** relativ langsame implizite Kommunikation unter Nutzung von Systemdiensten zum Dateizugriff, ggf. mit zusätzlicher Synchronisation (z. B. durch Semaphore).
- **gemeinsame Speicherbereiche (shared memory):** schnelle implizite Kommunikation unter Nutzung gewöhnlicher Speicherzugriffe. Dazu gibt es Systemdienste zum Anlegen und Entfernen gemeinsamer Speicherbereiche und ggf. zur expliziten Synchronisation.
- **Kanäle, Pipelines (pipes, FIFOs):** Unidirektionaler Nachrichtenaustausch in Form eines Datenstroms (über Puffer bzw. Dateien).
- **Briefkästen (mailboxes) bzw. Nachrichtenwarteschlangen (message queues):** globale Nachrichten-Übergabestellen für indirekte, asynchrone Kommunikation mit Diensten zum Anlegen/Entfernen eines Briefkastens bzw. einer Nachrichtenwarteschlange und zum Absenden/Empfangen von Nachrichten. Die Nachrichten können darin mittels verschiedener Strategien (z.B. FIFO, Priorität, Zeitüberwachung) verwaltet werden. Ports entsprechen modifizierten Briefkästen zur *n :* 1-Kommunikation.
- **Sockets:** Endpunkte für Kommunikationsverbindungen mit jeweils einem zugeordneten Prozess. Leistungsfähiges, universelles Kommunikationsmittel, auch für verteilte Systeme geeignet.
- **entfernte Prozeduraufrufe (Remote Procedure Call, RPC):** Damit kann ein Prozess Dienstleistungen eines *anderen* Prozesses über einen „gewöhnlichen" *Prozeduraufruf* in Anspruch nehmen. Dahinter verbirgt sich jedoch ein bidirektionaler Nachrichtenaustausch zwischen beiden Prozessen über Adressraum- bzw. Rechnergrenzen hinweg! Implementierung durch Laufzeitbibliotheken.

☞ Nachrichtenbasierte Verfahren werden oft als *Message Passing* bezeichnet.

5 Betriebsmittel

5.1 Klassifikation

Definition

Betriebsmittel (resources) sind alle Komponenten (Objekte) eines Computersystems, die zur Erfüllung von Aufträgen benötigt werden.

Eigenschaften von Betriebsmitteln:

- **Realisierungsform**: *Hardware-Betriebsmittel* (z. B. Hauptspeicher, Prozessor), *Soßware-Betriebsmittel* (z. B. Programmcode, Dateien).
- **Benutzbarkeit**: *Lokale* (oder *dedizierte) Betriebsmittel* stehen jeweils nur einem einzigen Prozess zur Verfügung, *globale* (oder *gemeinsame) Betriebsmittel* sind von allen Prozessen nutzbar.
- **Wiederverwendbarkeit**: *Wiederverwendbare Betriebsmittel* sind nach ihrer Nutzung durch andere Prozesse in derselben Weise verwendbar (z. B. Eingabedateien). *Verbrauchbare Betriebsmittel* werden durch ihre Nutzung quasi zerstört (z. B. Signale, Nachrichten).
- **Benutzungsweise**: Parallel oder mehrfach nutzbare Betriebsmittel können gleichzeitig von mehreren Prozessen benutzt werden (z. B. reentrante Programmteile). *Exklusiv nutzbare Betriebsmittel* besitzen innere Zustände (z. B. die Register des Prozessors) und sind deshalb von mehreren Prozessen nur nacheinander verwendbar.
- **Entziehbarkeit**: *Entziehbare Betriebsmittel* können einem Prozess zeitweise entzogen werden, wobei ihr Zustand für eine erneute Zuteilung gesichert werden muss (z. B. Prozessor, Hauptspeicher). Bei *nicht entziehbaren Betriebsmitteln* wäre diese Sicherung nicht oder nur mit hohem Aufwand möglich (z. B. Drucker, Festplatte).

5.2 Verwaltung

Einige Betriebsmittel müssen vom System verwaltet werden. Deren Zuteilung an Prozesse ist eine zentrale Aufgabe von Betriebssystemen.

Definition

Ablaufplanung (scheduling) ist die Vorgehensweise des Betriebssystems bei der Zuteilung von Betriebsmitteln an Prozesse. Die Auswahl eines geeigneten Prozesses wird im Allgemeinen anhand einer Zuteilungsstrategie vom *Scheduler,* einer speziellen Kernkomponente, vorgenommen.

☞ Das Scheduling kann auf mehreren Ebenen bzw. für verschiedene Betriebsmittel erfolgen. Es dient der Optimierung des Systemverhaltens anhand einer Zielfunktion, z. B. Fairness gegenüber allen Prozessen, maximale Auslastung einzelner Betriebsmittel, maximaler Durchsatz (Aufträge pro Zeit) oder minimale Antwortzeiten. Besonders wichtig ist die Zuteilung des *Prozessors* an einen der laufbereiten Prozesse.

Eine echte *„Planung"* der Zuteilungsreihenfolge ist praktisch nur dann möglich, wenn der Bedarf der Prozesse vorab (weitgehend) bekannt ist, z. B. im Stapelbetrieb für Jobs, oder für periodische Prozesse. Bei *statischer Verwaltung* erhält ein Prozess alle benötigten Betriebsmittel bei seiner Erzeugung vom System zugeteilt und gibt sie bei seiner Beendigung wieder an das System zurück.

In allen anderen Fällen erfolgt das Scheduling *„operativ"* durch *dynamische Zuteilung,* abhängig von der aktuellen *Lastsituation.* Dabei fordert ein Prozess ein Betriebsmittel bei Bedarf an, benutzt es eine gewisse Zeit exklusiv und gibt es im Allgemeinen danach wieder zurück (Achtung: hierbei besteht Verklemmungsgefahr).

Zur Anforderung und Freigabe stehen den Prozessen im Allgemeinen Systemdienste am API zur Verfügung. Fordert ein Prozess ein bereits belegtes Betriebsmittel an, so muss er warten, bis dieses wieder frei ist (passives Warten in einer Warteschlange). Zur Implementierung eignet sich deshalb die Sperrsynchronisation, z. B. durch binäre Semaphore (für *Einzelanforderungen)* oder allgemeine Semaphore *(gleichzeitige* Anforderung mehrerer *gleichartiger* Betriebsmittel). Die *gleichzeitige* Anforderung *verschiedener* Betriebsmittel wird z. B. mit Hilfe eines Vektors möglich, in dem die Forderungen markiert werden.

Kriterien für Scheduling-Strategien:

- **Verdrängung (preemption):** Bei *nicht verdrängenden (nonpreemptive,* auch *run-to-completion) Strategien* wird über die Neuzuteilung erst entschieden, wenn der aktive Prozess die Ressource selbst freigibt. Bei *verdrängenden (preemptive) Strategien* kann der aktive Prozess an einer beliebigen Stelle unterbrochen und ihm das Betriebsmittel entzogen werden.
- **Verdrängungsursache:** Bei einer *zeitgesteuerten (zyklischen) Strategie* wird die Verdrängung durch eine Zeitunterbrechung, bei *ereignisgesteuerten Strategien* durch ein Ereignis ausgelöst.

Typische Strategien zum (Prozessor-)Scheduling:

- **First Come First Served (FCFS):** nichtverdrängende Strategie mit Zuteilung in der Reihenfolge des Eintreffens der Prozesse bzw. der Aufträge.
- **Shortest Job First (SJF), Shortest Processing Time (SPT):** Bei Kenntnis der zu erwartenden Prozessorbelegungszeit erfolgt die Zuteilung an den Prozess mit der kürzesten Rechenzeit (z. B. im Stapelbetrieb).
- **Round Robbin (RR), Time-Sharing- (oder Time-Slice-)Verfahren:** Reihum erhält jeder Prozess den Prozessor für ein gewisses Zeitintervall (Zeitscheibe). Nach Ablauf dieser Frist wird der Prozess verdrängt. Falls er weitere Prozessorzeit benötigt, wird er erneut ans Ende der Warteliste eingeordnet. Typisch für den Dialogbetrieb.
- **Feste Prioritäten (Fixed External Priorities, FEP):** Zuordnung von Prioritäten (Vorrangzahlen) an die Prozesse. Die Betriebsmittelzuteilung erfolgt an den jeweils höchstpriorisierten Prozess. Niedrig priorisierte Prozesse können dabei „verhungern" (starvation).
- **Dynamische Prioritäten:** Die Prioritäten sind von „außen" (z. B. mittels Systemdienstaufruf) und/oder von „innen" (z.B. durch zyklische Neuberechnung im Betriebssystem) veränderbar: Shortest Elapsed Time, SET: Priorität fällt mit wachsender bisheriger Bearbeitungszeit; Highest Response Ratio Next, HRN: Priorität in Abhängigkeit vom Quotienten aus Verweilzeit und Bearbeitungszeit; Shortest Remaining Processing Time, SRT: Priorität wächst mit verbleibender Restbearbeitungszeit eines Prozesses.
- **Terminabhängige Zuteilung:** dynamische Prioritäten durch Neuberechnung anhand von Zeitkriterien, z. B. Zuteilung an den Prozess, dessen einzuhaltender Antwortzeitpunkt am nächsten liegt (Earliest Deadline First, EDF) oder an denjenigen mit dem geringsten Spielraum (Differenz aus Restantwortzeit und Restbearbeitungszeit, Least Laxity First, LLF), z. B. bei der Echtzeitverarbeitung.

  Prioritätenverfahren bergen die Gefahr der *Prioritätenumkehr (priority inversion),* dies kann von einigen Betriebssystemen durch *Prioritätenvererbung (priority inheritance)* gemildert werden (z. B. VxWorks).

In der Praxis werden mitunter verschiedene Strategien kombiniert, z. B. bei Multi Level Queueing (MLQ) oder bei Multi Level Feedback Queueing, MLFQ. Typisch sind Kombinationen aus Time Sharing und Prioritäten. Außerdem kann bei geeigneter Betriebssystemarchitektur eine Trennung von *Scheduling-Mechanismus* und einzelnen

Scheduling-Strategien vorgenommen werden: Strategien können flexibel außerhalb des Kerns (z.B. durch spezielle Systemprozesse) implementiert werden.

5.3 Verklemmungen

Definition

Eine **Verklemmung (deadlock)** ist ein Zustand in einem System paralleler Prozesse, bei dem einige Prozesse derart wechselseitig aufeinander warten, dass keiner von ihnen mehr voranschreiten kann.

Der *permanente* Wartezustand bei Verklemmungen resultiert daraus, dass jeder der verklemmten Prozesse auf Bedingungen wartet, die nur von *anderen* Prozessen erfüllt werden könnten, die jedoch *ihrerseits* auf solche Bedingungen warten, häufig im Zusammenhang mit Betriebsmittelanforderungen. Zur Beschreibung dienen u. a. **Betriebsmittelgraphen**.

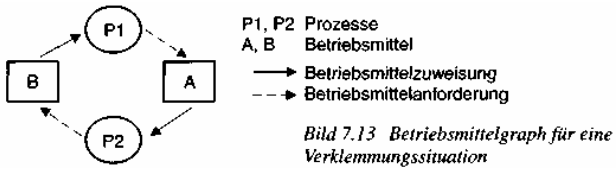


Abbildung 12: Betriebsmittelgraph für eine Verklemmungssituation

Im Bild repräsentieren die Achsen den Zeitverlauf *je eines* Prozesses, die Kurven zeigen jeweils einen konkreten Gesamtablauf *beider Prozesse,* der sich durch (zufällige) Prozesswechsel ergibt. Im Beispiel enthalten beide Prozesse je zwei *überlappende* kritische Abschnitte kAl und kA2, die korrekt durch Semaphore koordiniert sind. Dadurch spannen sich Flächen auf, die von der Kurve nicht durchlaufen werden dürfen. Es hängt von der zufälligen Lage dieser „Sperrflächen" ab, ob für die Kurve eine „Falle" entsteht. Ablauffolgen, die in diese Falle (im Bild die kreuzweise schraffierte Fläche) geraten, führen zu einer Verklemmung!

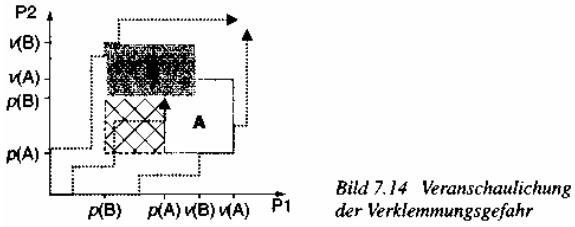


Abbildung 13: Veranschaulichung der Verklemmungsgefahr

Bedingungen zum Auftreten von Verklemmungen:

1. Die Betriebsmittel werden *exklusiv* genutzt (kritische Abschnitte).
2. Bereits belegte Betriebsmittel können *nicht entzogen* werden.
3. Die *Prozesse fordern* Ressourcen *nach, ohne* die *belegten freizugeben.*
4. Die Prozesse fordern Betriebsmittel *in unterschiedlicher Reihenfolge* an, so dass eine geschlossene Kette (Kreis) entsteht.

Die Bedingungen I bis 3 sind für das Auftreten von Verklemmungen zwar *notwendig,* dennoch muss es nicht zwangsläufig dazu kommen. *Hinreichend* ist erst die Bedingung 4, die *zufällig* eintreten kann. Daher sind Verklemmungen nicht durch Tests auszuschließen, sondern nur durch theoretische Untersuchungen unter Nutzung von Modellen (z. B. PETRI-Netze).

Verhinderung von Verklemmungen (prevention): Maßnahmen mit dem Ziel, das Eintreten einer der Verklemmungsbedingungen auszuschließen, indem den Prozessen Beschränkungen auferlegt werden.

So werden z. B. einige Betriebsmittel nur einem Prozess fest zugeordnet. Mit diesem Dienstleistungsprozess könnten andere Prozesse kommunizieren und damit das Betriebsmittel indirekt nutzen, z. B. einen Drucker (Spooling).

Vermeidung von Verklemmungen (avoidance): Das Betriebssystem analysiert bei jeder Betriebsmittelanforderung, ob nach Zuteilung im weiteren Verlauf (im ungünstigsten Fall) nur *sichere, verklemmungsfreie* Zustände eintreten würden. Kann dies nicht garantiert werden, wird die aktuelle Anforderung „sicherheitshalber" nicht erfüllt, auch wenn dies zur Zeit möglich wäre (Beispiel Bankier-Algorithmus*)*.

Entdeckung und Beseitigung (detect and delete): Das Betriebssystem untersucht zyklisch den Zustand aller Prozesse (und Betriebsmittel) auf mögliche Verklemmungen. Eine Verklemmung kann nur durch vorzeitigen Abbruch mindestens eines verklemmten Prozesses aufgelöst werden.

In vielen Betriebssystemen sind keine oder nur vereinzelte Gegenmaßnahmen zu Verklemmungen implementiert. Praktisch wird oft eine (optionale) Zeitüberwachung derjenigen Systemdienste angeboten, die zu Wartezuständen führen können, damit bei Zeitüberschreitung (timeout) eine Fehlerbehandlung eingeleitet werden kann.

6 Speicherverwaltung

6.1 Aufgaben

Der Hauptspeicher kann sowohl *räumlich* als auch *zeitlich geteilt* genutzt werden.Seine Verwaltung erfolgt oft gesondert, weil sie stark von der Hardware, z. B. der MMU und verschiedenen Adressierungsarten abhängt.

Aufgaben der Speicherverwaltung:

- Bereitstellung und Zuteilung von Speicher an Prozesse, ggf. Entzug,
- effiziente Organisation freier und belegter Speicherbereiche,
- Verbergen von Beschränkungen (Virtualisierung),
- Schutz vor fehlerhaften/unerlaubten Zugriffen.

Die Zuordnung (allocation) von Speicher an Prozesse ist *statisch* (vor dem Start des Prozesses) oder *dynamisch* (bei Bedarf) möglich.

6.2 Einfache Speicherverwaltung

Möglichkeiten der räumlichen Aufteilung des Speichers:

- getrennte Bereiche für das Betriebssystem und die Anwendung(en),
- getrennte Bereiche *(Partitionen)* für jeden Prozess.
- Die Aufteilung kann in Bereiche *fester* oder *variabler Größe* erfolgen.

Einige typische Probleme bei der Speicherverwaltung:

- **Fragmentierung**: Speicherverschnitt durch nicht nutzbare Teile.
  - *Interne Fragmentierung* ergibt sich, wenn einem Prozess verfahrensbedingt mehr Speicher zugeteilt wird, als er benötigt.
  - *Externe Fragmentierung* tritt auf, wenn der gesamte Speicher so in freie und belegte Teile zerstückelt ist, dass kein genügend großer, zusammenhängender, freier Bereich zu finden ist. Durch *Defragmentierung (Kompaktifizierung, garbage collection)* erfolgt eine Umordnung, wobei alle belegten Teile „dicht“ ans Ende des Speichers und alle freien Teile zusammen an den Anfang verschoben werden.
- **Verschiebbarkeit (relocation):** Programmteile müssen verschiebbar sein, damit sie in jedem beliebigen freien Bereich gespeichert werden können. Dies erfordert entsprechende Adressierungsverfahren und einen geeigneten Programm-Lader.

Typische Datenstrukturen zur Speicherverwaltung:

- **Belegungsvektor (bitmap):** Kennzeichnung des Zustandes (belegt/frei) eines Bereiches fester Größe durch je ein zugeordnetes Bit.
- **Verkettete Listen:** Beschreibung je eines Speicherbereichs (Größe, Belegungszustand, ggf. zugeordneter Prozess) durch ein Listenelement. Belegte und freie Bereiche können in einer gemeinsamen oder in getrennten Listen (z.B.,*Freispeicherliste")* stehen.

Verfahren	Suche
First Fit	Vom Beginn an nach dem ersten ausreichend großen Bereich
Next Fit	Wie First Fit, aber beginnend an der jeweils aktuellen Position
Best Fit	Vollständig nach dem kleinsten passenden Bereich
Worst Fit	Vollständig nach dem größten passenden Bereich

Tabelle 7.3 Einige Verfahren zur Suche nach passenden Speicherbereichen

Definition

Swapping: Verfahren, das u. a. im Time-Sharing bei Speichermangel einen unwichtigen (z. B. zur Zeit blockierten) Prozess auswählt und alle von ihm belegten Bereiche vorübergehend auf einem Hintergrundspeicher (Festplatte) auslagert. Der *Swap-Bereich* kann als Dateisystem oder als Datei angelegt werden (vorab oder bei Bedarf).

6.3 Virtueller Speicher

Definition

Ein **virtueller Speicher** ist ein Konzept, bei dem von der *Größe* des Hauptspeichers und der *Position,* die einzelne Bereiche darin einnehmen, abstrahiert wird. Stattdessen wird dem Anwender/Programmierer ein unbegrenzter (virtueller) Speicher vorgespiegelt. Die Umwandlung der *virtuellen* in *reale Adressen,* die Positionierung der Speichereinheiten sowie deren automatische Ein- und Auslagerung unter Nutzung eines externen Speichers erfolgen unter Steuerung des Betriebssystems.

☞ Virtueller Speicher ist quasi eine Automatisierung der **Überlagerungstechnik**. Auch sie erlaubt es, Prozesse ablaufen zu lassen, deren gesamter Speicherbedarf größer als der verfügbare Hauptspeicher ist.

Voraussetzung ist hier jedoch eine vorherige Zerlegung in solche Teile *(overlays),* die zeitweise nicht gemeinsam benötigt werden und deshalb durch andere überlagert werden können. Die baumförmige Überlagerungsstruktur muss vom Programmierer konzipiert und mittels spezieller Steueranweisungen für Linker und Lader formuliert werden.

Die Organisation des virtuellen Speichers erfolgt durch Aufteilung des virtuellen Adressraums der Prozesse und des realen Hauptspeichers in:

- **Segmente**: verschieden große Speichereinheiten je nach Gegebenheiten der Prozesse bzw. Festlegungen des Programmierers. Interne Fragmentierung kann vermieden werden, externe jedoch nicht. Zur Adresstransformation erhält jeder Prozess eine *Segmenttabelle.*

  ☞ Die Segmentierung wird sowohl von Prozessoren (Segmentregister) als auch von Compilern unterstützt. Damit sind differenzierter Zugriffsschutz wie auch gemeinsame Nutzung von Code-Segmenten durch mehrere Prozesse möglich (vgl. „shared libraries“). Ein Segment wird als zusammenhängender Bereich verwaltet. Segmente, die wegen zusätzlichem Speicherbedarf der Prozesse „wachsen“, müssen ggf. umgelagert werden.

- **Seiten (pages) und Seitenrahmen (page frames):** einheitlich große Speichereinheiten, die vom Betriebssystem bestimmt und für den Programmierer nicht sichtbar sind. Jeder Rahmen des Hauptspeichers kann eine beliebige Seite aus dem virtuellen Adressraum eines Prozesses aufnehmen. Externe Fragmentierung wird vermieden, nicht jedoch interne. Zur Adresstransformation besitzt jeder Prozess eine **Seitentabelle**.

  ☞ Die optimale Seitengröße ist ein Kompromiss zwischen der Häufigkeit von Seitenwechseln und der Größe der internen Fragmentierung (im Mittel eine halbe Seite je Prozess), sie liegt in der Praxis oft zwischen 2 und 32 KByte.

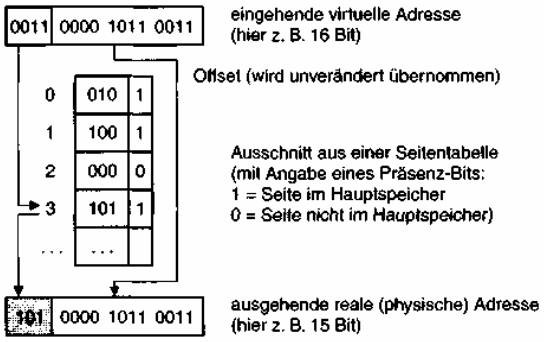


Abbildung 14: Prinzip der Adressformation mittels Seitentabelle (einstufig)

Ablauf bei virtuellem Speicher mit Seitentechnik (paging):

1. Bei jedem Zugriff zu einer Seite erfolgt eine Zerlegung der virtuellen Adresse in ein oder mehrere Indexanteile und eine Verschiebung (offset). Der Index verweist in die Seitentabelle, aus der bestimmbar ist, ob die Seite bereits im Hauptspeicher liegt.
2. Ist das der Fall, kann die Adresstransformation und nachfolgend der Speicherzugriff durchgeführt werden. Ansonsten wird der aktuelle Befehl mit einer speziellen Ausnahme (exception) abgebrochen, die einen **Seitenzugriffsfehler (page fault)** signalisiert. Damit wird das sog. *„Laden nach Bedarf" (demand paging)* realisiert.
3. Zur Behandlung des Seitenzugriffsfehlers muss die referenzierte Seite vom externen Speicher in einen freien Rahmen des Hauptspeichers geladen und die Seitentabelle aktualisiert werden. Da jede Seite in einen beliebigen freien Rahmen gespeichert werden kann, ist die Strategie **zur Platzierung (placement)** sehr einfach.
4. Steht kein freier Rahmen zur Verfügung, muss erst eine Seite ausgelagert werden. Die Strategie zur **Ersetzung (replacement)** beeinflusst stark das Leistungsverhalten des Systems.
5. Nach einem Seitenzugriffsfehler und seiner Behandlung muss der zuvor erfolglos abgebrochene Befehl wiederholt werden.

Der Algorithmus	**ersetzt diejenige Seite, die**
First In First Out (FIFO)	sich am längsten im Hauptspeicher befindet
Not Recently Used (NRU)	in letzter Zeit nicht benutzt wurde
Least Recently Used (LRU)	am längsten nicht benutzt wurde
Not Frequently Used (NFU)	am wenigsten genutzt wurde
Least Frequently Used (LFU)	in letzter Zeit am wenigsten genutzt wurde

Tabelle 7.4 Einige Seitenersetzungsstrategien

Die Implementierung erfordert Hardwareunterstützung zur Speicherung von Informationen zur Seitenpräsenz/Gültigkeit, zur Zugriffshäufigkeit, zu Schreibzugriffen und zum Zugriffsschutz. Da die Tabellen sehr umfangreich werden können, sind weitere Organisationsformen üblich, wie *mehrstufige* oder *invertierte Seitentabellen* bzw. zusätzliche Puffer in der MMU, z. B. ein *Assoziativspeicher (Translation*

LookAside Buffer, TLAB). Der Transport von Seiten zwischen Extern- und Hauptspeicher wird meist von einem speziellen Dienstprozess des Betriebssystems *(Seitenwechselprozess)* übernommen.

Definition

Seitenflattern (thrashing) kennzeichnet einen (Überlast-)Zustand des Systems infolge häufiger Lade- und Ersetzungsvorgänge. Dazu kann es kommen, wenn nicht alle aktuell von Prozessen geforderten Seiten im Hauptspeicher gehalten werden können. Es gibt Modelle und Verfahren, um diesem Effekt vorzubeugen.

Definition

Das **Lokalitätsprinzip** besagt, dass sich Zugriffe eines Prozesses zeitweise auf einige Seiten konzentrieren. Dies wird durch bestimmte Programmiertechniken (z. B. Modularisierung) unterstützt.

Definition

Die **Arbeitsmenge (working set)** umfasst alle Seiten, die ein Prozess im Moment zum Voranschreiten braucht. Günstig ist es demnach, wenn die Arbeitsmengen möglichst vieler Prozesse im Hauptspeicher verbleiben. Einfluss auf die Effizienz des virtuellen Speichers hat auch die Organisation des externen Speichers, auf dem die ausgelagerten Seiten untergebracht sind.

7 Ein-/Ausgabe-System

7.1 Anforderungen und Struktur

Die Steuerung der Ein-/Ausgabe (E/A) erfordert vor allem die Abstraktion von technischen Details und Eigenschaften der E/A-Geräte und die Bildung geeigneter Schnittstellen. Das E/A-System weist oft eine (quasi-konsistente) Schichtenstruktur auf.

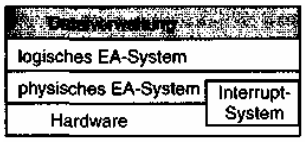


Bild 7.16 Schichtenstruktur des Ein-/Ausgabe-Systems (Die Dienste des E/A-Systems sind i. Allg. über Funktionsbibliotheken zugänglich.)

Abbildung 15: Sichtenstruktur des Ein-/Ausgabe-Systems

7.2 Physisches Ein-/Ausgabe-System

Das **physische Ein-/Ausgabe-System** steuert alle Vorgänge zur Ein-/Ausgabe von Daten mit den physischen (real angeschlossenen, peripheren) Geräten des Systems. Diese Vorgänge sind *gerätespezifisch* und werden im Allgemeinen durch **Gerätetreiber (device driver)** erbracht.

Viele Steuerfunktionen sind in *E/A-Prozessoren (Controller)* realisiert, auf die über spezielle E/A-Befehle und Port-Adressen oder über gewöhnliche Speicherbefehle und Hauptspeicheradressen *(Memory Mapped I/O)* zugegriffen werden kann. Eine softwaregesteuerte Einstellung der Port-Adressen erlaubt den Anschluss peripherer Geräte ohne weitere Eingriffe *(‚Plug & Play')*.

Aufgaben eines Gerätetreibers:

- Initialisierung eines peripheren Gerätes,
- Vorbereitung und Initiierung einer E/A-Operation,
- Übergabe bzw. Übernahme von Daten,
- Abbruch einer E/A-Operation,
- Kontrolle angestoßener E/A-Operationen, ggf. Fehlerbehandlung,
- Reaktion auf Ausnahmesituationen/Unterbrechungen,
- Synchronisation mit dem auftraggebenden Prozess.

Beispiel: Festplattentreiber müssen den Schreib-/Lesekopf positionieren. Dazu gibt es diverse Algorithmen (z. B. *Shortest Seek Time First, SSTF* oder den *Fahrstuhl-Algorithmus).* Außerdem ist ggf. ein *Interleave-Faktor* zu beachten.

E/A-Geräte können *zeichenorientiert* (z. B. Tastatur, Bildschirm) oder *blockorientiert* (z. B. Magnetbandspeicher, Festplatten) arbeiten. Das Zusammenspiel zwischen Treiber und Gerät ist möglich durch:

- **Hardwaresynchronisation**: Bei sehr schnellen Geräten behält der Treiber die Steuerung bis zum Abschluss einer E/A-Operation.
- **Softwaresynchronisation**: Bei langsamen Geräten muss auf das Ende der E/A-Operation gewartet werden, entweder durch zyklische Abfrage von Statusmeldungen (**polling**) oder durch **Interrupts**.

Polling ist wegen des aktiven Wartens oft unbefriedigend. Bei Interrupt-Steuerung wird der Treiber bei einer Wartesituation blockiert und später durch ein Interrupt wieder bereit gesetzt bzw. aktiviert.

Definition

Interrupts sind spezielle Signale, die eine asynchrone Unterbrechung der Befehlsabarbeitung durch den Prozessor veranlassen. Ein Interrupt kann durch eine spezifische Routine (Interrupt Service Routine, ISR, Interrupt-handler) behandelt werden.

Nach Unterbrechung durch ein Interrupt speichert der Prozessor die Adresse der Unterbrechungsstelle und verzweigt zur weiteren Behandlung über eine *Interrupt-Vektor-Tabelle* zu derjenigen ISR, die dem auslösenden Interrupt zugeordnet ist. Die dafür benötigte Zeit wird *Interrupt-Latenzzeit* genannt. ISRs sollten kurz sein. Da es nicht immer gelingt, die Interrupt-Behandlung vollständig innerhalb der ISR durchzuführen, wird häufig ein Teil der Behandlung durch einen speziellen Prozess (oder Thread) realisiert, der mit der ISR zu synchronisieren ist. Nach Abschluss der ISR wird die Arbeit an der Unterbrechungsstelle fortgesetzt.

☞ Interrupts können sowohl von der Hardware, insbesondere von peripheren Geräten, als auch von der Software durch spezielle *Interrupt-Befehle (Traps, Supervisor Calls)* ausgelöst werden. Letztere werden vor allem benutzt, um durch Systemdienstaufrufe in den Betriebssystem-Kern zu gelangen.

Treiber erhalten Aufträge von Anwendungsprozessen bzw. höheren Systemschichten ebenso wie „Interrupt-Anstöße“ von den Geräten. Sie können als Prozeduren wie auch als Prozesse bzw. Threads implementiert sein. Die Implementierung als Prozess bzw. Thread erlaubt eine höhere Parallelität. Wegen ihrer gerätespezifischen Funktionen gehören Treiber im Allgemeinen nicht zum Betriebssystem-Kern. Stattdessen sind sie (dynamisch) installierbar.

7.3 Logisches Eingabe-/Ausgabe-System

Definition

Das **logische Eingabe-/Ausgabe-System** stellt für Anwendungen oder höhere Systemschichten Funktionen zur *geräteunabhängigen* E/A zur Verfügung und benutzt dazu Dienste des physischen E/A-Systems.

Typische **Funktionen** des logischen E/A-Systems:

- Verwaltung logischer Geräte (z. B. Konsole, Plattenlaufwerk),
- Bereitstellung geräteunabhängiger Dateneinheiten, z. B. *Block* bzw. *Cluster*, ggf. Pufferung,
- Speicherplatzverwaltung auf externen Speichermedien.

Auch das logische E/A-System wird durch Aufruf von Systemdiensten bzw. über Bibliotheksfunktionen genutzt. Außerdem sind spezielle E/A-Dienstleistungsprozesse außerhalb des Betriebssystem-Kerns

möglich, z.B. *SPOOL*-Prozesse *(Simultaneous Peripheral Operations OnLine),* Sie erlauben bei langsamen E/A-Geräten (z. B. Drucker), die Daten auf einen schnellen Externspeicher (z.B. Festplatte) zwischenzulagern. Damit lassen sich beim Stapelbetrieb Wartezeiten verkürzen. Im Mehrprozessbetrieb kann die Zuordnung eines exklusiv nutzbaren Betriebsmittels an einen solchen Server-Prozess Verklemmungen verhindern.

8 Dateiverwaltung

8.1 Dateikonzept

Definition

Eine **Datei (file)** ist eine Menge von logisch zusammengehörenden Daten, die auf einem geeigneten Medium permanent gespeichert werden kann und über einen Bezeichner identifizierbar ist.

Die Dateiverwaltung ist eine Komponente des Betriebssystems, die in Zusammenarbeit mit dem E/A-System dem Benutzer bzw. Programmierer komfortable Dienste anbietet.

Aufgaben der Dateiverwaltung:

- Abbildung der Nutzerdaten auf Verwaltungseinheiten (Satz, Block),
- Realisierung von Zugriffsverfahren (sequentiell, direkt/wahlfrei),
- Zuordnung von Speicherplatz für Dateien, Verwaltung belegter und freier Teile des Speichermediums,
- Gewährleistung von Datenschutz.

Definition

Ein **Dateisystem (file System)** umfasst die Menge aller von einem Betriebssystem in derselben Art verwalteten Dateien einschließlich aller dafür erforderlichen Verwaltungsinformationen.

Beispiele für Dateisysteme:

FAT (MS-DOS), NTFS (Windows NT), HPFS (OS/2), ext2fs (Linux), NFS (Sun Network File System), CDFS (CD-ROM).

8.2 Dateiorganisation

Die **Dateiorganisation** beschreibt die Strukturierung bzw. Anordnung der Daten und die möglichen Zugriffsformen.

Die **logische Struktur** einer Datei ist für ein Programm z. B. sichtbar als:

- **Folge von Bytes (byte stream):** unstrukturierte Folge von Zeichen,
- **Folge von Datensätzen (records):** Dateneinheit für eine Dateioperation des logischen E/A-Systems von fester oder variabler Länge, die in einzelne Felder strukturiert sein kann.

Bytefolgen sind ein einfaches, anwendungsneutrales und zugleich flexibles Mittel zur Dateiverwaltung. Eine Anwendung kann daraus komplexere (Satz-) Strukturen aufbauen. Außerdem lassen sich damit E/A-Vorgänge vereinheitlichen (Abbildung logischer E/A-Geräte auf Dateien, z.B. in UNIX). Eine Datensatz-Organisation dagegen wird vor allem von klassischen Betriebssystemen zur kommerziellen Datenverarbeitung angeboten (z.B. IBM OS/390, BS2000, VMS).

Aus der Struktur der Datei ergeben sich verschiedene Zugriffsformen:

- **Sequentieller Zugriff** auf die Bytes bzw. Sätze einer Datei (mittels Dateizeiger, file pointer).
- **Direkter (wahlfreier, random) Zugriff** auf Sätze fester Länge.
- **Indexsequentieller Zugriff** unter Nutzung eines Satzschlüssels.

Beispiele: ISAM (Indexed Sequential Access Method), VSAM (Virtual Sequential Access Method), B- bzw. B*-Bäume.

8.3 Speicherplatzzuordnung und -Verwaltung

Für die Abbildung von Dateien auf Speichermedien schafft das physische E/ASystem Voraussetzungen, z. B. durch *Formatierung* von Plattenspeichern in Zylinder, Spuren und Sektoren sowie effiziente Zugriffsalgorithmen.

Als Speicherplatz-Verwaltungseinheit benutzt das logische E/A-System **Blöcke** fester Größe (**cluster**), typischerweise zwischen 1/2 und 64 KByte groß. Eine Datei wird auf eine Menge fortlaufend numerierter Blöcke abgebildet. Zwischen Sätzen und Blöcken erfolgt ggf. eine Umwandlung mittels *Blockung* (mehrere Sätze in einem Block) oder *Segmentierung* (mehrere Blöcke für einen Satz).

Verfahren zur Zuordnung (allocation) von Blöcken zu Dateien:

- aufeinander folgende (zusammenhängende) Blöcke für die gesamte Datei *(contiguos files)* oder für größere Teile einer Datei.
- interne Verkettung der Blöcke untereinander durch einen Zeiger.
- externe Verkettung der Blöcke in einer gesonderten Datenstruktur (Index, z. B. MS-DOS: *File Allocation Table (FAT),* UNIX: *Inode).*

Die Dateiverwaltung nutzt ähnliche Techniken wie die Hauptspeicherverwaltung. Die nötigen Verwaltungsdaten werden auf dem Datenträger selbst gespeichert. Zur *Defragmentierung* werden z. T. Hilfsprogramme angeboten.

8.4 Verzeichnisse

Definition

Ein **Dateiverzeichnis** (Ordner, Katalog, **directory**) ist eine Datenstruktur mit Informationen über Dateien zu dem Zweck, diese effizient und geordnet auf einem Datenträger verwalten zu können.

Dem *Benutzer* dienen Verzeichnisse zur Herstellung einer logischen Ordnung von Dateien unabhängig von ihrer wirklichen Lage auf dem Datenträger. Dazu enthält ein Verzeichniseintrag (auch *File Control Block, FCB)* direkt oder indirekt über weitere Datenstrukturen zahlreiche Informationen wie z. B. Name bzw. Pfadbeschreibung, ggf. Dateityp, Zugriffsrechte, Besitzverhältnisse u. ä., Dateigröße, Erstellungs- und Modifikationsdatum sowie weitere Attribute. In manchen Systemen sind auch gemeinsam benutzbare Dateien möglich (z. B. durch *Links).*

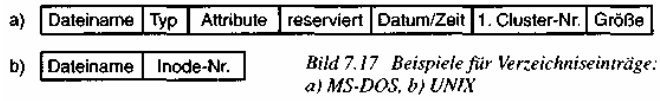


Bild 7.17 Beispiele für Verzeichniseinträge: a) MS-DOS, b) UNIX

Abbildung 16: Beispiele für Verzeichniseinträge

Konventionen für Namen sowie Anordnung und „Wirkungsbereich" von Verzeichnissen differieren in einzelnen Systemen. So findet man *einstufige* (Volume Table Of Contents, VTOC), *benutzereigene,* meist jedoch *mehrstufige, hierarchische Dateiverzeichnisse* für einen Datenträger oder für das gesamte System. Verzeichnisse können in gesonderter Form als auch in Form von Verzeichnisdateien gespeichert sein. Mehrstufige Verzeichnisse haben oft die Struktur eines Baumes (oder eines gerichteten Graphen), wobei der oberste Knoten im Allgemeinen. als *Wurzel (root)* bezeichnet wird. Innerhalb eines Dateisystems kann die Eindeutigkeit von Dateinamen durch *Pfadangaben (path names)* gesichert werden. Diese können *absolut* (beginnend ab der Wurzelposition) oder *relativ* zum aktuellen Arbeitsverzeichnis (current/working directory) sein.

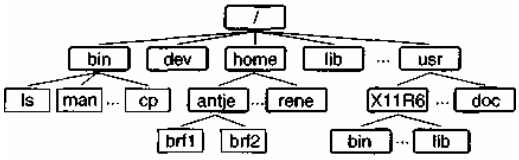


Bild 7.18 Beispiel für ein hierarchisches Dateiverzeichnis

Abbildung 17: Beispiel für ein hierarchisches Dateiverzeichnis

Das Betriebssystem benutzt Verzeichnisse, um die vom Benutzer verwendeten Dateinamen auf die internen Strukturen zur Organisation des Zugriffs und des Speicherplatzes abzubilden.

8.5 Datenträger-Organisation

Zur Erkennung und Organisation des Dateisystems setzen Betriebssysteme eine gewisse Strukturierung des *Datenträgers* bzw. einer *Partition* voraus. Dazu wird eine Information über Position und Umfang der Verwaltungsdaten des Dateisystems benötigt. Diese ist meist in einem bestimmten Block (z. B. *Superblock, Boot-Sektor*) vermerkt, der oft an einer festen Position auf dem Datenträger steht. Auch der *Anfangslader (Bootstrap Loader*) für das Betriebssystem befindet sich an einer definierten Position des Datenträgers.

Zu den zentralen Verwaltungsdaten des Dateisystems zählen der Name des Datenträgers, die Zuordnung belegter und die Lage freier Blöcke sowie das Wurzelverzeichnis oder aber ein Verweis darauf. Aus Sicherheitsgründen werden diese Daten zum Teil redundant gespeichert. Die Verwaltung mehrerer verschiedener Dateisysteme auf einem Datenträger ist möglich, sie erfordert eine Aufteilung in logische Geräte bzw. Partitionen und die Speicherung dieser Information auf dem Datenträger (z. B. in einer *Partitionstabelle).*

8.6 Sicherheit und Zugriffsschutz

Datenverlust vorzubeugen erfordert vom Benutzer u. a. inkrementelle oder vollständige Sicherung von Dateien, Verzeichnissen oder ganzen Datenträgern *(backup)*. Programme erlauben dazu z.B. periodisches automatisches Erstellen einer Sicherungskopie der aktuell bearbeiteten Datei oder automatisches Sichern der alten Version. Durch Hardware-Redundanz und besondere Softwarelösungen kann die Datensicherung weiter verbessert werden, z. B. durch *RAID-Verfahren*.

Maßnahmen der Dateiverwaltung zur Datensicherheit:

- Registrierung fehlerhafter Blöcke des Datenträgers,
- Speicherung von Zeitinformationen über jede Datei für Backups,
- Sicherung der Konsistenz des Dateisystems,
- Rekonstruktion fehlerhafter Dateien durch Protokollierung,
- Sofortiges Schreiben von Blöcken *(write through)* aus dem Cache,
- Datei-Sperren (file locking), Satzsperren (record locking),
- *Transaktionen*: ein Vorgang findet entweder vollständig korrekt oder gar nicht und ohne Veränderungen statt.

Der Zugriffsschutz zu Dateien spielt insbesondere beim Mehrnutzer-Betrieb bzw. in verteilten Systemen eine wesentliche Rolle. Es gibt viele Verfahren zur Vergabe und Kontrolle verschiedener *Zugriffsrechte* (z. B. Lesen, Schreiben, Ausführen) an Benutzer.

8.7 Leistungsverbesserungen

Die Leistungsfähigkeit der Dateiverwaltung wird wesentlich durch die Häufigkeit von Zugriffen auf den Datenträger beeinflusst.

Maßnahmen zur Vermeidung bzw. Beschleunigung von Dateizugriffen:

- effiziente Organisation und „günstige" Lage von Dateiverwaltungsinformationen (Tabellen, Listen, Verzeichnisse), z. B. durch Organisation von Daten in *Zylindergruppen, Bänder/Stripes* u. Ä.
- Realisierung eines *Pufferspeichers (cache),* um Nutzer- und Verwaltungsdaten möglichst lange im Hauptspeicher zu halten.

Beim *Caching* führen Lesezugriffe immer in den Cache. Bei sequentiellem Lesen können sie z. T. sogar „vorausschauend" erfolgen *(read ahead).* Analog zum virtuellen Speicher wird im Cache eine Ersetzungsstrategie realisiert, z. B. FIFO oder LRU.

Schreiboperationen in den Cache sollten zusätzlich auf dem peripheren Speicher ausgeführt werden, z. B. durch:

- sofortiges Schreiben (write through) nach jedem Schreibzugriff,
- verzögertes Schreiben (write behind, write-on-close),
- zyklisches Schreiben modifizierter Daten (cyclic write).

8.8 Systemdienste zur Dateiverwaltung

Systemdienste zur Dateiverwaltung erfordern im Allgemeinen die Angabe eines *Dateideskriptors (file descriptor/handle)* zur Identifizierung.

Typische Systemdienste zur Arbeit mit Dateien:

- **Erzeugen** einer neuen Datei, Festlegung von Eigenschaften,
- **Löschen** einer Datei, Freigabe des belegten Speicherplatzes,
- **Öffnen** zur Vorbereitung folgender Dateizugriffe: Laden und Prüfen von Verwaltungsinformationen (z. B. Lage der Datei, Rechte),
- **Schließen**: abschließende Aktionen nach Bearbeitung einer Datei, z. B. Aktualisierung von Verwaltungsinformationen (Verzeichnisse),
- **Lesen von Daten** aus der Datei in einen Puffer des Prozesses,
- **Schreiben von Daten** aus einem Puffer des Prozesses in die Datei,
- **Suchen** einer bestimmten Position in der Datei,
- Lesen und ggf. Verändern von Dateieigenschaften.

Einige Dienste sind auch für den Bediener als Kommandos (Dienstprogramme) benutzbar, z.B. Anlegen, Umbenennen und Löschen von Dateien/Verzeichnissen.

9 Einsatz von Betriebssystemen

9.1 Installation und Konfigurierung

Die Installation eines Betriebssystems auf einem Computer ist Voraussetzung für seine Benutzbarkeit. Bestandteil der Installation ist die Konfigurierung.

Definition

Konfigurierung ist die Auswahl und Zusammenstellung von gewünschten Hardware/Software-Komponenten zu einem funktionsfähigen System, einer entsprechenden Konfiguration (configuration).

Bei der Konfigurierung werden nur die konkret benötigten bzw. gewünschten Teile des Betriebssystems vom Datenträger in den Hauptspeicher geladen und dort zu einer in Funktionalität und Umfang konkret angepassten und jederzeit erneut ladbaren Form des Betriebssystems zusammengefügt. Die Auswahlentscheidungen können sowohl vom Bediener abgefragt als auch zum Teil vom Installationsprogramm *(Setup-Programm)* selbstständig getroffen werden.

Nach erfolgter Installation muss das Betriebssystem die vorhandene bzw. gewählte Hard- und Softwarekonfiguration speichern, so dass sie nach erneutem Startvorgang sofort bekannt ist. Dazu werden oft Systemdateien angelegt, z. B. config.sys bei MS-DOS oder Registry bei Windows95/98/NT. Diese Dateien erfordern bei ihrer Veränderung meist detaillierte Systemkenntnisse.

Definition

Eine **Rekonfigurierung** des Systems bedeutet die Änderung seiner Konfiguration, z. B. durch Hinzufügen und/oder Herausnehmen einzelner Komponenten. Eine Änderung der Hardware-Konfiguration zieht i. d. R. eine Rekonfigurierung des Betriebssystems nach sich.

So erfordert der Einbau eines neuen Peripheriegerätes im Allgemeinen die Installation eines passenden Treibers. Eine derartige Rekonfigurierung kann bei einigen Systemen sogar dynamisch zur Laufzeit des Systems erfolgen.

9.2 Boot-Vorgang

Da das Betriebssystem nach dem Anschalten des Computers ohne weitere Bedienhandlungen arbeitsbereit sein sollte, muss es zuvor selbst erst (ohne Betriebssystemunterstützung!) geladen und gestartet werden.

Das Laden des Betriebssystems erfolgt in mehreren Schritten **(bootstrapping, „booten“).** Beim Einschalten des Computers wird infolge einer besonderen Unterbrechung *(Reset-Signal, Interrupt)* ein systemspezifisches Programm gestartet, das an einer definierten Stelle im ROM bzw. EPROM des Computers steht (z. B. das sog. *ROM BIOS* bei PCs). Dieses testet die Hardware *(Power On Self Test),* ermittelt und speichert die Hardware-Konfiguration und initialisiert einige wichtige Komponenten (z. B. Controller).

Danach wird ein relativ kleines betriebssystemspezifisches Programm (Urlader, Anfangslader, bootstrap loader) von einer definierten Position („Boot-Sektor“) desjenigen externen Speichers geladen und gestartet, der als dafür vorgesehen erkannt wurde. Der Urlader wiederum dient dem Nachladen weiterer Betriebssystem-Teile bzw. Konfigurationsdaten vom Datenträger. Auch dies erfolgt schrittweise, zunächst werden Dateien mit definierten Namen bzw. von definierter Stelle (z. B. im Wurzelverzeichnis) geladen. Erst diese Dateien vervollständigen das Betriebssystem, so dass es nun selbst die Steuerung übernehmen kann.

Einige Systeme erlauben es, während des Bootens eines von mehreren Betriebssystemen auszuwählen, das gestartet werden soll. Diese Systeme können sich auch in verschiedenen logischen Bereichen (Partitionen) derselben Festplatte befinden. Die genaue Aufteilung und Belegung der Partitionen durch verschiedene Betriebssysteme muss vom Boot-Programm durch das Lesen eines definierten Blockes (z. B. *Master Boot RecordMBR, Master File Table MFT)* erkannt werden.

9.3 Administration

Bei der Nutzung eines Betriebssystems sind je nach Betriebsart bzw. Einsatzfall verschiedene administrative Aufgaben zu lösen, z.B.:

- *Wartung und Pflege des Betriebssystems:* Installation von neuen Versionen (updates) bzw. von Systemergänzungen oder -korrekturen (patches, bugfixes), sowie die Konfigurierung bzw. Rekonfigurierung des Betriebssystems und ggf. der Hardware.
- *Benutzerverwaltung (Anlegen von Nutzerprofilen, accounts):* Vergabe von Nutzerkennungen (Login-Namen), Zugriffsrechten, Beschränkungen (z. B. disk quotas) und deren Eintragung in zentrale Dateien.
- *Überwachung und Sicherung des laufenden Betriebs:* Überwachung von Ereignismeldungen, Login-Versuchen, Abrechnungsdaten; Datensicherung (backups), Gewährleistung der Sicherheit im Betrieb.
- *Organisatorische und Planungsmaßnahmen:* Organisation/Durchführung von Nutzerschulungen, ggf. Erstellung von Hilfsprogrammen/Dokumentationen, Unterstützung des Managements durch Berichte, Konzeptionen, Kontakt zu Systemherstellern bzw. -lieferanten.

9.4 Leistungsbewertung

Bewertungsgrößen für Betriebssysteme sind z. B.:

- **Einsatzfähigkeit/Nutzen:** Betriebsarten, mögliche Benutzer- und Prozessanzahl, unterstützte bzw. benötigte Hardware, Portabilität, Installationsaufwand, Integrierbarkeit in das bestehende Umfeld, Stabilität, Bedienkomfort/Benutzerschnittstelle, Umfang weiterer Systemsoftware.
- **qualitative technische Merkmale:** Innovationsgrad, Konformität zu Standards, Architektur, Skalierbarkeit, Zuverlässigkeit, Sicherheit, Prozess-Zustandsmodell, Scheduling-Verfahren, virtueller Speicher, API.
- **quantitative technische Merkmale:** Prozess-Wechsel-Zeit, Dauer möglicher Kernsperren, Interrupt-Latenzzeit, Umfang des API.
- **betriebswirtschaftliche Größen:** Kosten für Anschaffung, Betrieb, Updates, Support usw. (vgl. *Total Cost of Ownership, TCO).*

Möglichkeiten zur Leistungsbewertung (**performance evaluation**) eines Betriebssystems:

- **Messung relevanter Daten,** z. B. Prozesswechsel-Dauer, ggf. unter Nutzung von *Benchmarks (Lastprofilen),*
- **analytische Modelle,** z.B. Bedienungsmodelle zur Untersuchung von Scheduling- oder Seitenersetzungsstrategien,
- **grafische Modelle** zur Beschreibung und qualitativen Bewertung, z.B. PETRI-Netze zum Nachweis der Verklemmungsfreiheit,
- **Simulation** typischer Vorgänge und Verhaltensweisen, z.B. Ermittlung von Engpässen, Dimensionierung von Wartelisten.

☞ Einige Kenngrößen von Betriebssystemen werden durch die Hardware beeinflusst!

9.5 Schutz und Sicherheit

Schutzmechanismen in Betriebssystemen richten sich oft nach der Betriebsart bzw. der Einsatzumgebung. Im Vordergrund stehen dabei *Integrität* und *Verfügbarkeit* des Systems.

Zur Realisierung von Schutzmechanismen werden alle Verwaltungseinheiten im System in jeweils identifizierbare *aktive Einheiten (Subjekte,* z.B. Benutzer, Prozesse) *und passive Einheiten (Objekte, z.* B. Speicherbereiche, Dateien) klassifiziert. Dadurch lassen sich Operationen ableiten, die von Subjekten über einer Menge von Objekten ausgeführt werden dürfen und einen *Schutzbereich (protection domain)* bilden. Ziel ist es, dem Betriebssystem die Zugriffskontrolle und -Steuerung zu ermöglichen und dafür zu sorgen, dass diese nicht umgangen werden können.

Typische **Schutzkonzepte**:

- **Zugriffskontroll-Listen (Access Control List, ACL):** Hierin wird für jedes Objekt festgelegt, wer welche Operation darüber ausführen darf (z. B. Schreibrecht eines Prozesses auf eine Datei).
- **Berechtigungen (capabilities):** Sie sind einem Subjekt zugeordnet, von ihm jedoch nicht änderbar. Sie geben an, welche Objekte durch welche Operationen bearbeitet werden dürfen.

☞ Einfache praktische Mittel sind z. B. (verschlüsselte) Passwörter für Benutzer und verschiedene Zugriffsrechte für Dateien, diese reichen aber oft nicht aus.

Potentielle Sicherheitslücken entstehen in der Praxis u. a. durch:

- inkonsistente Betriebssystemzustände infolge unerwarteter Abläufe (z.B. unsinnige oder unzulässige Eingaben),
- beabsichtigte oder unbeabsichtigte Schlupflöcher in Systemprogrammen, undokumentierte Systemdienste bzw. Parameter,
- fehlerhafte Betriebssystemergänzungen/-anpassungen,
- fehlerhafte Identifizierung, Authentisierung bzw. Autorisierung.

☞ Angreifer nutzen solche Lücken oft aus, um durch Trojanische Pferde, Wurm- oder Viren-Programme im Betriebssystem oder in Anwendungen Daten auszuspähen oder zu manipulieren.

10 Betriebssysteme für spezielle Einsatzgebiete

10.1 Echtzeit-Betriebssysteme

Bei der **Echtzeitverarbeitung** *(real time processing)* werden Vorgänge in einem (technischen Basis-) System durch einen Computer überwacht und gesteuert. Deshalb bestehen besondere Anforderungen an das zeitliche Verhalten *(Reaktionszeit, Antwortzeit*), auch unter hoher Verarbeitungslast *(Hochlastfestigkeit),* sowie an die Verfügbarkeit.

☞ So genannte *harte Echtzeitbedingungen* erfordern, unter allen Umständen Zeitüberschreitungen zu vermeiden, z.B. in der Medizin-, Fahrzeug- und Flugtechnik (Steuerung von Antiblockiersystem oder Airbag). *Weiche Echtzeitbedingungen* liegen vor, wenn einzelne oder geringfügige Zeitüberschreitungen tolerierbar sind, ohne dass größere Schäden entstehen (z. B. bei der Telefonvermittlung).

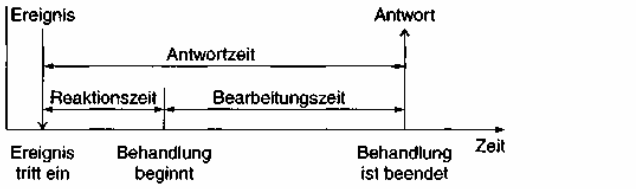


Bild 7.19 Reaktionszeit und Antwortzeit (In der Reaktionszeit ist die durch die Hardware bestimmte Interrupt-Latenzzeit enthalten, → 7.6.2.)

Abbildung 18: Reaktionszeit und Antwortzeit

Besonderheiten von Echtzeit-Betriebssystemen:

- *Zeitverwaltung* zur Überwachung von Zeitpunkten und Zeitfristen,
- *Unterbrechungssystem* zur Annahme von Aufträgen durch Ereignisse (Interrupts) aus der technischen Umgebung,
- *Scheduling* durch präemptive Verfahren, die zeitliche Abhängigkeiten bzw. Fristen berücksichtigen, z. B. EDF, LLF, oder eine Einplanung garantieren, z. B. bei periodischen Prozessen durch periodenmonotones Scheduling (Rate Monotonie Scheduling, RMS),
- *Synchronisation/Kommunikation* durch Koordinierungsmittel (z. B. Ereigniszähler) mit minimalen Verzögerungen (Latenzzeiten),
- *Betriebsmittel:* Arbeitsfähigkeit bei beschränkten Ressourcen, insbesondere bei eingebetteten Systemen (embedded Systems). Geeignete Betriebssysteme belegen hierbei oft weniger als 100 KByte Speicher (RAM/ROM).
- *Software-Architektur:* Das Betriebssystem sollte ROM-fähig sein, damit es auf einem Festwertspeicher untergebracht werden kann.
- *Robustheit, Zuverlässigkeit, Verfügbarkeit* und *Stabilität* im Dauerbetrieb, im Allgemeinen ohne Bedienereingriffe.

☞ Der Markt für Echtzeit-Betriebssysteme ist durch zahlreiche Produkte diverser Hersteller für unterschiedlichste Hardware-Plattformen (inkl. MikroController) gekennzeichnet, z.B. Enea OSE, VRTX, LynxOS, VxWorks, pSOS, WindowsCE, OSEK/VDX (im Automobilbau), außerdem EPOC und PalmOS für mobile Computer und Kommunikationsgeräte, Eine Standardisierung der APIs wird durch Standards angestrebt, z.B. POSIX.

10.2 Netzwerk-Betriebssysteme

Knoten eines Netzwerkes können als *Server* arbeiten, um Dienstleistungen für alle anderen Systeme zu erbringen. Ein **Netzwerk-Betriebssystem** (auch: **Server-Betriebssystem**) muss als besondere Form verteilter Betriebssysteme:

- die gekoppelten (auch autonom arbeitsfähigen) Systeme verwalten,
- die Kommunikation zwischen auftraggebenden Systemen *(clients)* und dienstleistendem System *(server)* ermöglichen und steuern.

☞ Eine typische Aufgabe eines Netzwerk-Betriebssystems ist die Organisation eines *Datei-Servers (file server)* mit Daten und Programmen zur gemeinsamen Benutzung durch alle Systeme im Netz. Hier gibt es im Allgemeinen keine Ortstransparenz, so dass beim Dateizugriff die konkrete Pfadangabe nötig ist.

Als Netzwerk- bzw. Server-Betriebssystem eignen sich viele universelle Betriebssysteme wie z.B. WindowsNT/2000 (Server), OS/2, UNIX/Linux, OS/390, OS/400, die Mehrprozess- und Mehrnutzerbetrieb erlauben. Andererseits gibt es spezielle Systeme (z. B. Novell Netware /7.80/), die als reines Netzwerk-Betriebssystem konzipiert und optimiert sind.

10.3 Verteilte Systeme

„Echte“ verteilte Systeme verfolgen das Ziel, dem Benutzer den Eindruck eines einzigen homogenen Systems zu vermitteln. Dabei ist entweder jeder Knoten des Netzes mit demselben Betriebssystem ausgestattet oder es gibt eine Verteilungskomponente. Deshalb bestehen besondere Forderungen, wie z.B. nach Transparenz, Flexibilität, Zuverlässigkeit und Skalierbarkeit. Beispiele verteilter Betriebssysteme sind Amoeba, MACH und Chorus.

☞ Beim Entwurf der Dienste ist zu beachten, dass in einem verteilten System ein Verbindungsweg oder ein Netzknoten ausfallen kann. Geeignete Gegenmaßnahmen erfordern einen zusätzlichen Implementierungsaufwand im Betriebssystem.

Besonderheiten und Probleme verteilter Betriebssysteme:

- *Zeit:* Eine Uhrsynchronisation aller Systeme ist nötig.
- *Synchronisation:* Typisch sind Dienstleistungsprozesse und Transaktionen, keine zentralen Datenstrukturen (Semaphore).
- *Kommunikation:* Nutzung von RPC und Sockets auf Basis standardisierter Protokolle (OSI, TCP/IP).
- *Anordnung der Prozessoren/Netzknoten:* Dazu gibt es verschiedene Modelle (z. B. Workstation-Modell, Prozessor-Pool).
- *Verteilte Dateisysteme* für transparenten Zugriff zu File-Servern, z. B. Network File System (NFS, Sun) oder Novell-Filesystem.

☞ Durch (transparente) *Migration* können Dateien zu einem anderen Knoten des Netzes verlagert werden. Bei der *Replikation* werden wegen höherer Verfügbarkeit und Fehlertoleranz eigenständige Kopien von Dateien erstellt.

10.4 Betriebssysteme für Parallelrechner

Auch Parallelrechner sind spezielle verteilte Systeme. Von Bedeutung für das Betriebssystem ist es, ob die Prozessoren über gemeinsamen Speicher verfügen (speichergekoppelte Systeme) oder nicht.

Bei **UMA-Architekturen (symmetrische Multiprozessoren)** können alle Prozessoren den Code ein und desselben Betriebssystems abarbeiten. Es gibt gemeinsamen Speicher und für das Scheduling eine einzige Warteliste laufbereiter Prozesse/Threads für alle Prozessoren.

Bei **NUMA-Architekturen** wird der Unterschied zwischen lokalen und entfernten Speicherzugriffen meist vom Betriebssystem verborgen, so dass ein *verteilter gemeinsamer Speicher (distributed shared memory)* entsteht. Beim Scheduling sind ggf. Beziehungen eines Prozesses zu einem bestimmten Prozessor *(Affinität)* bzw. Prozessgruppen *(‚gang scheduling")* zu berücksichtigen, um eine ungünstige Zuordnung zu vermeiden.

Auf **UMA- und NUMA-Architekturen** werden z. B. einige Mach-basierende Systeme (z. B. SPP-UX, KSR-OS), verschiedene UNIX-Implementierungen und z. T. Windows NT/2000 eingesetzt.

Parallelrechner mit ausschließlich lokalem Speicher auf jedem Knoten (**NORMA**) ähneln echten verteilten Systemen, allerdings mit sehr schnellem Kommunikationssystem. Auf jedem Knoten kann ein (minimales) Betriebssystem als Laufzeitumgebung mit der jeweiligen Anwendung verbunden werden. Die nötigen Kommunikationsdienste werden in Bibliotheken (z.B. *PVM oder MPI*) angeboten. Ein Hochleistungs-parallelrechner kann über eine Workstation als Frontend bedient werden, die dann z.B. einen Stapelbetrieb organisiert. Die entsprechenden Systeme sind im Allgemeinen herstellerspezifisch (z.B. Parsytec Parix, Meiko CS, UNICOS MAX).

10.5 Fallstudien universeller Betriebssysteme

Im Folgenden sind einige häufig benutzte *universelle* Betriebssysteme aufgelistet und grob charakterisiert.

MS-DOS (Microsoft): „klassisches" PC-Betriebssystem

Nur Single-Tasking- und Single-User-Betrieb; Architektur: Treppenstufenmodell; Dateisystem FAT (File Allocation Table); beschränkte Speicherverwaltung.

Windows 95/98/ME (Microsoft): PC-Betriebssystem

Vorrangig für private Anwender; grafische Benutzeroberfläche; Single-User-Betrieb; Multi-Tasking bzw. Multi-Threading mittels Prioritäten und Zeitscheiben; FAT-Dateisystem mit Erweiterungen (FAT32/VFAT); virtueller Speicher, verbesserte Treiberschnittstelle („Plug & Play")-

Windows NT/2000 (Microsoft): Betriebssystem für PCs und Workstations, für Einzelsysteme als auch für Server (Netzwerk)

Windows NT vorrangig für professionelle Anwender; grafische Benutzeroberfläche; Multi-Tasking-(Multi-Threading-)Betrieb; in der Server-Version Multi-User-Betrieb möglich; verdrängendes Scheduling nach Prioritätsklassen und Time-Sharing; Architektur mit (Mikro-)Kern, Systemschichten, Serverprozessen und Subsystemen für Fremdanwendungen; Virtueller Speicher (paging); eigene Dateisysteme (NTFS, New Technology File System; DFS, Distributed File System) und Unterstützung von FAT; verschiedene Schutz-und Sicherheitsmechanismen, begrenzt skalierbar/multiprozessorfähig.

Windows 2000 als Nachfolger von Windows NT vorrangig für professionelle, geschäftliche Anwendungen; mit verschiedenen Ausbaustufen (von Professional bis Data Center Server); zusätzliche Sicherheitsfunktionen; höhere Skalierbarkeit.

Windows XP (Microsoft):

Früherer Codename „Whistler“; als jüngster Nachfolger von Windows NT/2000 und ebenso zur Ablösung von Windows 95/98/ME positioniert; neuartiges Registrierungsverfahren; Einsatz sowohl für private Anwender als auch im professionellen/kommerziellen Umfeld; verbesserte Stabilität und Geschwindigkeit; modifizierte grafische Benutzeroberfläche; verstärkte Integration von Internet-basierenden Funktionen.

OS/2 Warp (IBM): Betriebssystem für PCs, Workstations und Server

Grafische Benutzeroberfläche und integrierte Spracherkennung; Multi-Tasking-(Multi-Threading-) Betrieb; mit LAN-Manager auch Multi-User-Betrieb; verdrängendes Scheduling nach Prioritäten; Architektur mit Kern, Systemschichten und Client-Server-Unterstützung; Virtueller Speicher (paging); eigenes Dateisystem (HPFS, High Performance File System) und Unterstützung von FAT; diverse Schutz- und Sicherheitsmechanismen; Skriptsprache REXX.

BeOS (Be Inc.): Betriebssystem für PCs

Multi-Tasking-, (Multi-Threading)-, Single-User-Betrieb (ab V5.0 Multi-User); Mikro-Kern-Architektur; virtueller Speicher; eigenes Dateisystem (BeFS) mit Journaling-Funktion; optimiert für Multimedia-Anwendungen.

MacOS (Apple): Betriebssystem für Apple Macintosh Computer

Multi-Tasking Betrieb; in höheren Versionen Multi-User-Unterstützung; grafische Benutzeroberfläche, zahlreiche Publishing-, Multimedia- und Internet-Tools; aktuelle Version Mac OS X basiert auf einem neuen, als „Open Source Project“ entwickelten Kern („Darwin“) und bietet eine neue grafische Benutzeroberfläche.

UNIX (Systeme zahlreicher Hersteller, z. B. IBM AIX, HP-UX, Sun Solaris, Compaq True64Unix, SCO Unix, SGI IRIX, u. a.)

„Klassisches“ Multi-Tasking- und Multi-User-Betriebssystem für Workstations/Server, aber auch für PCs und Mainframes portiert; System V Release 4 (SVR4) und BSD-UNIX als Quasi-Standards; Kern-Schale-Architektur mit Unterstützung des Client-Server-Modells durch zahlreiche Systemprozesse („Dämonen“); verschiedene austauschbare Shells mit zahlreichen Tools und Programmiermöglichkeiten; Multi-Threading über Bibliotheken möglich; Scheduling mittels Prioritäten und Zeitscheiben; virtueller Speicher (swapping, paging), Dateiverwaltung mittels Inodes, verschiedene Dateisysteme; verschiedene grafische Benutzeroberflächen auf Basis von X11 möglich, z. B. OSF/Motif; diverse Schutz- und Sicherheitsmechanismen; sehr gut skalierbar für Multiprozessorsysteme.

Linux (frei verfügbare Alternative zu kommerziellen UNIX-Systemen)

Entwicklung durch Programmierer in aller Welt als „Open Source Project“ (Quellcode ist via Internet öffentlich verfügbar), meist in Form von GNU-Versionen entsprechender UNIX-Programme; neben freier Verfügbarkeit im Internet auch Vertrieb von so genannten Linux-Distributionen (z. B. von S.u.S.E., Red Hat oder Debian); für verschiedene PC-Plattformen, aber auch auf Server und Mainframes portiert (z.B. auf IBM S/390); neben privatem Gebrauch inzwischen auch verstärkte kommerzielle Anwendung.

OS/390 (IBM): Betriebssystem für IBM-Anlagen (Mainframes, S/390)

Multi-Tasking- und Multi-User-Betriebssystem zur kommerziellen Datenverarbeitung („High-End-Enterprise-Server“); Multiprozessor-Unterstützung; Virtueller Speicher; hochverfügbares, interaktives System (Parallel Sysplex für Hochverfügbarkeits-Cluster); diverse Schutzmechanismen und laufende Authorisierungskontrolle; integrierte UNIX-Schnittstelle; zahlreiche Management-Dienste und Systemsoftware-Komponenten (z. B. Transaktionsmonitor); seit 2001: z/OS als Nachfolger für neue Server-Linien.

OS/400 (IBM): Betriebssystem für mittlere IBM-Anlagen (AS/400)

Multi-Tasking- und Multi-User-Betriebssystem, vor allem für Abteilungsrechner/Server („Midrage-Bereich“); objektorientierte Architektur; Multi-

Threading-Unterstützung; Ein-Ebenen-Speicher-Modell; grafische Benutzeroberfläche; Multiprozessor-Unterstützung; diverse Schutzmechanismen; Internet-Server-Integration; zahlreiche Management-Dienste und Systemsoftware-Komponenten.

BS2000, BS2000/OSD (Siemens): Betriebssystem für verschiedene Hardwareplattformen (Mainframes, z. B. S/390, SR2000)

Multi-Tasking- und Multi-User-Betriebssystem für „High-End-Server“; Stapel-, Dialog- und Transaktionsbetrieb möglich; Version /OSD als „offenes System“ UNIX-konform; Multiprozessor-Unterstützung und Hochverfügbarkeitslösung HIPLEX; diverse Schutz- und Sicherheitsmechanismen; umfangreiche Systemsoftware.

VMS, Open VMS (Compaq/Digital); Betriebssystem für mittlere Systeme/Server (auf VAX- bzw. Alpha-Plattformen)

Multi-Tasking-, (Multi-Threading-) und Multi-User-Betrieb; geeignet für Multiprozessor-Systeme und Cluster; strenge Sicherheits- und Schutzmechanismen; hochverfügbar; verschiedene Dateisysteme (z.B. Spiralog) und Connektivität zu Windows NT.

Details zu diesen Systemen sollten (ebenso wie zu Betriebssystemen für spezielle Einsatzgebiete) ggf. den Veröffentlichungen der jeweiligen Hersteller entnommen werden.