Verwaltungs- und Wirtschaftsakademie Nürnberg

Informatik-Betriebswirt

Version: Dezember 2003 1.1

BETRIEBSSYSTEME

Inhalt

Literatur

Andrew S. Tanenbaum:
"Moderne Betriebssysteme"
Verlag Prentice Hall

H.-J. Siegert/U.Baumgarten:
"Betriebssysteme"
Oldenbourg Verlag

Peter Monadjemi:
"Windows 2000"
Verlag Markt & Technik

Peter Wilke:
"Folienkopien Betriebssysteme SS 2002"
Verwaltungs- und Wirtschaftsakademie Nürnberg

Michael Kofler:
"Linux"
Verlag Addison Wesley

Handbücher der diversen Betriebssystem-Hersteller

EINFÜHRUNG

Computer-Systeme

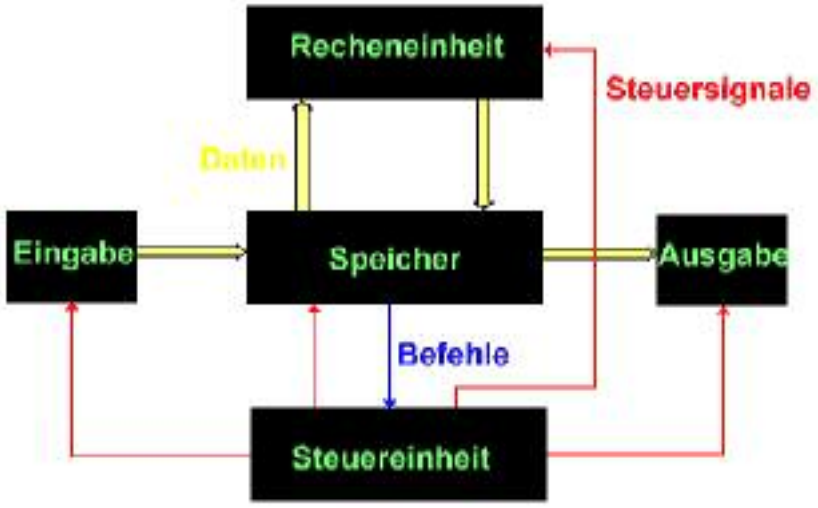
Computer sind Systeme, die aus Hardware und Software bestehen. Jede der beiden Komponenten ist ohne die andere viel weniger wert. Die vom Benutzer gestarteten Programme nehmen, damit sie ablaufen, sowohl die Hardware als auch die Software des Computers in Anspruch. Die Hardwarekomponenten umfassen alle "anfaßbaren" Komponenten. Dazu zählt der Computer selbst und alle Geräte, die daran angeschlossen sind: Bildschirm, Drucker, Diskettenlaufwerk usw. Die Software ist "nicht anfaßbar". Sie benötigt zum Transport, zur Speicherung und zum Funktionieren immer irgendwelche Hardware. Software läßt sich grob in das sog. Betriebsystem und die sog. Anwendersoftware unterteilen.

Trotz allem kann man die Software funktional nicht von der Hardware trennen, da selbst die kleinsten Aufgaben beide in Anspruch nehmen. Man braucht Hardware, um Daten eingeben zu können, aber zugleich auch die Software, die den Datenaustausch zwischen Terminal und Computer steuert. Man braucht Hardware, um Daten ausdrucken zu können, aber auch die Software, die ein Zeichen nach dem anderen vom Computer zum Drucker überträgt. Man braucht Hardware, um mit Zahlen rechnen zu können, aber auch die Software, die Zahlen für die Berechnung aufbereitet. Die beiden Komponenten eines Computersystems sind funktional eng verwoben, trotzdem sollen sie nun nacheinander betrachtet werden.

Hardware

Die Hardware eines einfachen Computers läßt sich in drei Funktionseinheiten unterteilen:

- die Zentraleinheit (CPU, Recheneinheit und Steuereinheit),
- den Speicher (Memory) und
- die Ein- und Ausgabegeräte (E/A-Geräte engl. Input/Output Units oder I/O).



Die CPU (Central Processing Unit) ist das "Herz" eines Computers. Sie bearbeitet Programme, führt Rechnungen aus und steuert die Vorgänge in den anderen Teilen.
Der Speicher nimmt alle Daten auf, die der Computer braucht. Das geht von Daten, die der nächste Programmschritt in einer Mikrosekunde benötigt, bis hin zu ganz selten benötigten Daten.
Schließlich stellen die E/A-Geräte die Verbindung her zwischen dem Computer und seinem Anwender.

Zentraleinheit

Die CPU besteht aus mehreren Teilen, die eng zusammenarbeiten: Recheneinheit und Steuereinheit. Sie enthält mehrere Register für die Aufnahme der Daten, die bei der Ausführung eines Programms verfügbar sein müssen. So nimmt z.B. das Befehlsregister den jeweils auszuführenden Befehl auf. Ein Register kann man sich als schnellen (Zwischen-) Speicher für ein einzelnes Datenwort vorstellen. Nur der anstehende Befehl steht im Befehlsregister, daher gibt es - von der Ausführung her gesehen - keinen Unterschied zwischen langen und kurzen Programmen oder zwischen schwierigen und einfachen. Die Befehle enthalten die Informationen über die Speicherplätze der benötigten Daten; bei der Ausführung des Befehls werden die Daten bereitgestellt und nach der Bearbeitung wieder abgelegt.

Recheneinheit

Die ALU (Arithmetic and Logic Unit) führt alle Berechnungen aus und trifft logische Entscheidungen. Ihre Hauptaufgaben sind es, Vergleiche zwischen Werten vorzunehmen und damit Bedingungen zu überprüfen und arithmetische/logische Operationen auszuführen. Die Rechenfähigkeit beschränkt sich auf die Addition, doch lassen sich daraus alle anderen arithmetischen Operationen aufbauen. Die komplizierten Rechnungen, die der Computer in kurzer Zeit ausführen kann, setzen sich aus einer Vielzahl kleinster Schritte in der ALU zusammen.

Steuereinheit

Die Steuereinheit und die Recheneinheit arbeiten Hand in Hand, die Register der Steuereinheit liefern die Signale zur Steuerung der gewünschten ALU-Funktion. Die Verbindung zwischen den Registern (und von ihnen zu den übrigen Einheiten) wird durch die Steuereinheit hergestellt. Man kann die Steuereinheit als Schaltzentrale des Computers ansehen, die alle Datenströme lenkt.

Speicher

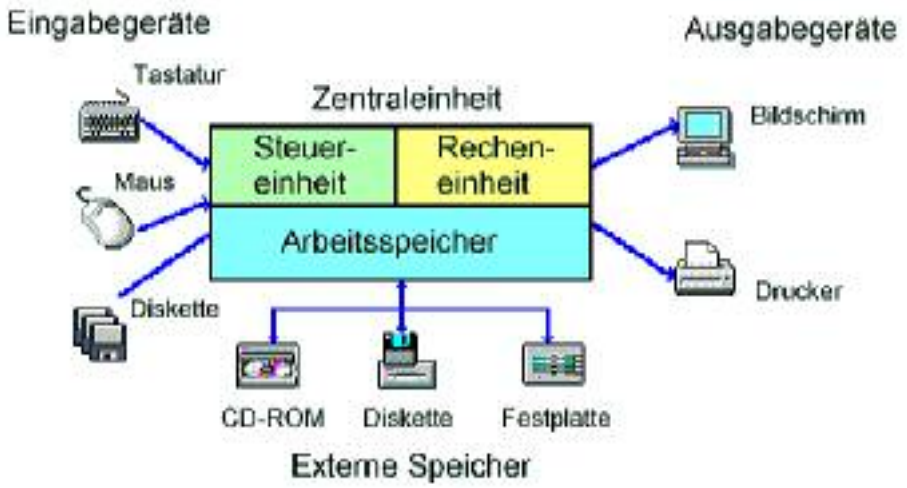
Der Speicher nimmt alle Daten auf, dazu gehören die Programme ebenso wie die von ihnen zu verarbeitenden Werte. Bei der Verarbeitung werden zunächst Zwischenergebnisse im Speicher abgelegt und schließlich die Endergebnisse. Man unterscheidet beim Speicher zwischen dem Hauptspeicher (oder Arbeitsspeicher) und dem externen Speicher. Der Hauptspeicher ist fest im Computer eingebaut, man kann ihn meist durch zusätzliche Speicherkarten erweitern. Im Hauptspeicher befindet sich das Programm, das Sie schreiben oder verändern wollen oder das der Computer bearbeiten soll. Auch die Daten, die beim Ablauf des Programms benötigt werden, werden dort abgelegt. Auf den Hauptspeicher kann die CPU direkt zugreifen, sie kann von dort neue Programmbefehle holen und kann Daten laden oder abspeichern.

Externe Speicher

Im externen Speicher werden die Programme und Werte aufbewahrt, die man zur Zeit nicht benötigt. Als Datenträger werden Festplatten und Disketten (Floppy- Disks) verwendet. Die Diskette wird ins Diskettenlaufwerk eingelegt und kann dann mit Daten "beschrieben" werden. Für das Abspeichern von Programmen vom Hauptspeicher auf den externen Speicher gibt es besondere Kommandos des Betriebssystems, ebenso für das Laden von einem externen Speicher in den Hauptspeicher.

Ein-/Ausgabe-Geräte

Die Eingabe- und Ausgabegeräte stellen die Verbindung zwischen einem Computer und seinem Benutzer her, aber auch zwischen einem Computer und anderen Geräten oder anderen Computern. Ein Ausgabegerät wie der Drucker läßt sich einsetzen, um die Ergebnisse der Programmbearbeitung zu erhalten oder um die Werte auszugeben, die in einem Speicher abgelegt sind. Ein Eingabegerät wird komplementär eingesetzt, mit ihm versorgt man die CPU oder den Hauptspeicher durch die CPU mit neuen Daten.



Vernetzung

Zu den E/A-Geräten gehören auch die Netzwerk-Verbindungen, mit denen sich mehrere Computer zu einem Verbund zusammenschließen lassen. Sie erlauben einen sehr schnellen Datenaustausch zwischen verschiedenen Computersystemen. Ein Vorteil solcher Vernetzung liegt darin, daß mehrere Systeme gemeinsam auf periphere Geräte wie Drucker oder Festplatte zugreifen können. Damit erweitern sich die Fähigkeiten der einzelnen Maschinen ohne entsprechende zusätzliche Kosten.

Software

Damit kennen Sie nun in etwa den Hardwareaufbau eines Computers. Es stellt sich jetzt die Frage nach seinen Einsatzgebieten und nach der Herkunft seiner Fähigkeiten. Seine Universalität erklärt sich aus den sehr elementaren Maschinenbefehlen, der Rechnerarchitektur und der Tatsache, daß der Computer, bevor er überhaupt etwas tun kann, erst durch ein Programm (Software) erklärt bekommen muß, was er zu tun hat. Es gibt eine riesige Vielfalt an Progammen, geschrieben für die unterschiedlichsten Aufgaben, die aber auf ein und derselben Hardware lauffähig sind.

Die Software eines Computersystems kann grob in zwei Teile gegliedert werden: die Anwendungssoftware und das Betriebssystem. Ein Anwenderprogramm ist für die Lösung einer ganz bestimmten Aufgabe geschrieben worden, der Anwender setzt es dann (und nur dann) ein, wenn er eine solche Aufgabe lösen will.

Dagegen ist das Betriebssystem eine nicht spezifische Software. Was die Infrastruktur einer Stadt mit Ihren Straßen, Buslinien, Trambahnen, Telefonnetz , Fernverkehrsanbindung für den Einzelnen und die Geschäftswelt ist, so ist das Betriebssystem die komfortable standardisierte

Umgebung für die Anwendungsprogramme. Möchte man in der Stadt von einem Ort zum anderen Waren transportieren, so benutzt man mit dem Auto die bestehenden Straßen und baut keine. Ähnlich ist es mit den Anwendungsprogrammen. Sie sind für eine spezielle Aufgabe geschaffen worden. Der Ersteller möchte sich aber nicht unbedingt mit der Infrastruktur, der Hardware, auseinendersetzten.

Das Betriebssystem wird ständig benötigt, um die Vorgänge in der Hardware zu steuern und um die Wechselwirkung zwischen dem Computer und seinen Anwendungsprogramme zu koordinieren.

Anwendungsprogramme

Wenn man allgemein von Software spricht, meint man meist Anwendungsprogramme, also Programme, die eine spezielle Aufgabe lösen. Einige typische Anwendungsprogramme sind

- Textverarbeitung,
- Spielprogramme,
- Tabellenkalkulation,
- Lernprogramme und
- Programme für die Verwaltung von Dateien,
- Programme für die Verwaltung großer Datenmengen (Datenbanken),
- Programme zum Steuern von Maschinen,
- Programme zum Regeln industrieller Prozesse,
- Programmiersprachen-Compiler
- und vieles mehr.

Wenn Sie in eine Computerzeitschrift schauen, finden Sie viele weitere Beispiele für Anwendungsprogramme.

Betriebssystem

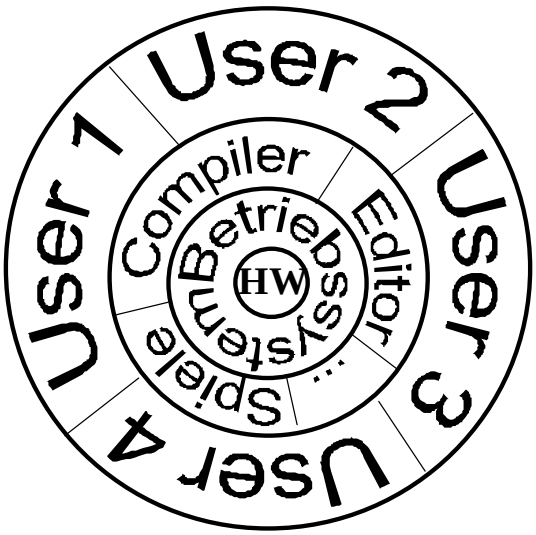
Die "nackte" Hardware eines Computers hat nicht die Fähigkeiten, ein Anwendungsprogramm zu bearbeiten. Auch wenn die CPU die Vorgänge in der Hardware auf einer niedrigen Ebene koordinieren kann, es muß eine Verbindung zwischen dem Anwenderprogramm und der jeweiligen Hardware hergestellt werden. Die Bearbeitung des Programms erfordert den Einsatz verschiedener Eingabe- und Ausgabegeräte. Es ist zunächst extern gespeichert und muß in den Hauptspeicher geladen werden. Diese und viele weitere Abläufe werden vom Betriebssystem gesteuert. Es ist ein sehr

umfangreiches Programm, das die Vorgänge in der Hardware steuert und koordiniert. Der Benutzer braucht sich nicht selbst darum zu kümmern, wie die einzelnen Schritte bei der Bearbeitung seines Programms intern ablaufen.

Wenn ein Computer nur ein einziges Programm zu bearbeiten hätte wie z.B. der Mikroprozessor in einer Waschmaschine, dann bräuchte er kein Betriebssystem. Die Aufgabe eines Betriebssystems ist es, eine Umgebung zu schaffen, in welcher verschiedene Anwendungsprogramme ablaufen können. Es bietet verschiedene Möglichkeiten an, den Computer einzusetzen.

Aufgaben des Betriebssystems

Von den vielen Aufgaben des Betriebssystems sollten Sie einige kennen. Zum Betriebssystem gehört z. B. ein Programm, das den Zugriff auf externe Speicher steuert und die gespeicherten Dateien verwaltet. Bei größeren Systemen und Netzwerken kontrolliert das Betriebssystem z.B. den Zugang zum Computer mit der Eingabe eines Paßwortes und organisiert die gleichzeitige Arbeit mehrerer Benutzer. Das Betriebssystem ist verantwortlich für das Speichern und Laden von Programmen und für die Zuteilung der benötigten Betriebsmittel. Aus der Sicht des Programmierers ist besonders wichtig, daß vom Betriebssystem die Programmierumgebung geschaffen wird. Das Betriebssystem liefert die Umgebung, in der Programme geschreiben werden können, und es steuert den Ablauf dieser Programme. Die Entwicklung immer leistungsfähigerer Betriebssysteme hat es ermöglicht, die Programmier-Arbeit immer komfortabler zu machen.



Grundlagen

Unter einem ***Betriebssystem*** (engl. operating system) versteht man Software, die zusammen mit dem Hardwareeigenschaften des Computers die Basis zum Betrieb bildet und insbesondere die Abarbeitung von Programmen steuert und überwacht.

Definitionen des Betriebssystembegriffe:

DIN 44300:

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

Was ist ein Betriebssystem?

<table>
<tr><td>???</td><td>Anwendungsprogramme</td></tr>
<tr><td>Systemdienstprogramm</td><td rowspan="2">Systemprogramme</td></tr>
<tr><td>Betriebssystem</td></tr>
<tr><td>Maschinensprache</td><td rowspan="3">Hardware</td></tr>
<tr><td>Mikroprogrammierung</td></tr>
<tr><td>Physikalische Geräte</td></tr>
</table>

Virtuelle Maschine

- Verbergen der realen Architektur
- Bereitstellung einer einfach zu handhabenden virtuellen Maschine

Betriebsmittelverwalter

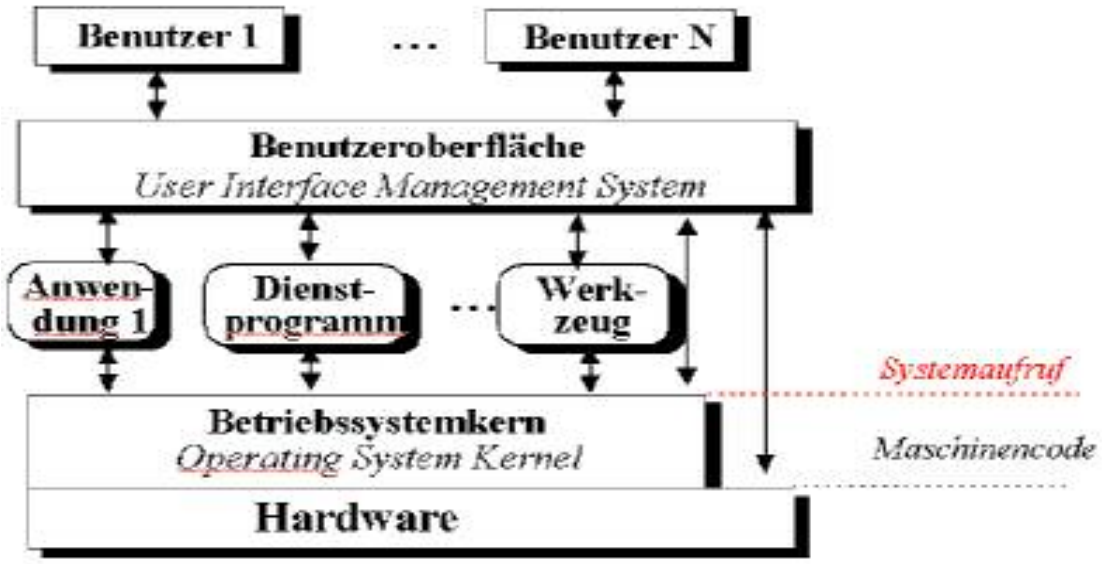
- Verwaltung aller Bestandteile des Rechners
- Geordnete und kontrollierte Zuteilung der Ressourcen
- Verwaltung und Schutz

Ein Betriebssystem hat folgende grundlegende Aufgaben:

- Verbergen der Komplexität der Maschine vor dem Anwender (Abstraktion),
- Bereitstellen einer Benutzerschnittstelle ("Kommandointerpreter", "Shell")
- Bereitstellen einer normierten Programmierschnittstelle (API), ggf. auch Compiler, Linker, Editor
- Verwaltung der Ressourcen der Maschine
    - Prozessor(en)
    - Hauptspeicher
    - Hintergrundspeicher (Platte, Band, etc.)
    - Geräte (Terminal, Drucker, Plotter, etc.)
    - Rechenzeit
- Verfolgung von Schutzstrategien bei dieser Ressourcenbereitstellung
- Koordination von Prozessen

Abstraktion des Maschinebegriffes nach Coy:

- Reale Maschine = Zentraleinheit + Geräte (Hardware)
- Abstrakte Maschine = Reale Maschine + Betriebssystem
- Benutzermaschine = Abstrakte Maschine + Anwendungsprogramm



Historie

Die erste Computergeneration (ca. 1945-1955) besaß kein Betriebssystem.

- Programmierung direkt (Steckbrett, Lochstreifen, Lochkarte)
- Keine Programmiersprachen

Die zweite Generation (ca. 1955-1965) arbeitete mit Stapelverarbeitung. Ein Auftrag wird in geschlossener Form, bestehend aus Programm, Daten und Steueranweisungen, zusammengestellt. Die Resultate erhält der Benutzer erst nach Abschluß der Bearbeitung zurück (meist als Ausdruck).

- Batch-Betrieb (Lochkarten)
- Einfache Job-Control-Sprachen
- Programmiersprachen (Assembler, Fortran, etc.)
- Magnetbänder als Zwischenspeicher

Die dritte Generation (ca. 1965-1980) beherrscht Dialogverarbeitung. Der Benutzer kommuniziert mit dem Computer über Tastatur und Bildschirm, mit deren Hilfe er Programme starten, verfolgen und beeinflussen kann.

- Multiprogramming = Mehrprogrammbetrieb = Mehrere Programme gleichzeitig im Speicher, quasisimultane, zeitlich verschachtelte Bearbeitung auf der Auftragsebene.
- Hauptspeicheraufteilung für mehrere Programme
- Zeitliche Verschachtelung der Programme (z.B.: Prog. A wartet auf Ausgabe, Prog. B rechnet) --> Timesharingbetrieb mit Terminals
- SPOOLING (simultaneous peripheral operation on line) direktes Speichern von Rechenaufträgen auf der Platte, "Selbstbedienung" des BS
- MULTICS als UNIX-Vorgänger
- 1969 das erste UNIX

Die vierte Generation (ab ca. 1975) ist ein Dialogsystem, wie wir es heute kennen.

- UNIX und C
- Multitasking als quasisimultane Ausführung weitgehend unabhängiger Programmabschnitte innerhalb eines Auftrags.
- Personal Computer (CP/M, MS-DOS, etc.)
- Netzwerkbetriebssysteme (Kommunikation mehrerer Computer)

- verteilte Betriebssysteme (mehrere Prozessoren = Multiprocessing) Mehrere Prozessoren bilden ein Computersystem --> Mehrere Programme werden von verschiedenen Prozessoren bearbeitet oder ein Programm von mehreren Prozessoren.

Bei bestimmten Betriebssystemen spielt auch die Verarbeitungszeit eine Rolle. Bei Realzeit-(Echtzeit-)Betriebssystemen für Steuerungs- und Regelungsaufgaben (sog. Prozeßrechner) spielt die Antwortzeit eine Rolle.

Folgende Tabelle gibt einen geschichtlichen Überblick über die Betriebssysteme:

Bis zu den 60-er Jahren	BS auf Lochkarten ; Problem: Speicherpreis
1969	Erstes Interaktive BS: **MULTICS**
1976	Erster Computer des Typs PC und erste Version Unix.
1980	Digital Research bringt den ersten CP/M (8 Bit Prozessor) und Microsoft entwickelt XeniX (BS für 16 und 32 Bit PC's)
1981	PC XT von IBM und DOS 1.0
1984	MS DOS 3.2 und Win 1.0
1987	IBM OS/2
1988	IBM OS/400 für AS/400
1990	Win 3.0
1991	Win 3.1
1993	Windows NT
1995	Win 95 und IBM OS/Warp
1997	Windows NT 4.0
1998	Win 98
2000	Windows 2000
2002	Windows XP

Klassifizierung von Betriebssystemen:

nach der Betriebsart des Rechnersystems

- Stapelverarbeitungs-Betriebssysteme (batch processing)
- Dialogbetrieb-Betriebssysteme (interactive processing, dialog processing)
- Netzwerk-Betriebssysteme (network processing)
- Realzeit-Betriebssysteme (realtime-processing). Ein RT-System muss nicht „sehr schnell“ sein, sondern nur *schnell genug* für die geplante Anwendung. Also nicht ‚Echtzeit‘ sondern ‚Rechtzeit‘. Dazu muss das zeitliche Verhalten des gesamten Systems deterministisch (eindeutig bestimmbar) sein, um eine maximale Reaktionszeit auf bestimmte Ereignisse *garantieren* zu können. Die Analyse der ungünstigsten Fälle ist daher das wichtigste Hilfsmittel um das Verhalten vorhersagen zu können.
- Universelle Betriebssysteme (erfüllen mehrere Kriterien)

nach der Anzahl der gleichzeitig laufenden Programme:

In dieser Klassifikation kommt der Begriff *"Task"* vor. Alternativ kann der deutsche Begriff *"Prozeß"* Verwendung finden. Aus Anwendersicht kann an dieser Stelle auch der Begriff *"Aufgabe"*bzw. *"Auftrag"* verwendet werden.

- Einzelprogrammbetrieb (single-tasking)
  Ein einziges Programm läuft jeweils zu einem bestimmten Zeitpunkt. Mehrere Programme werden nacheinander ausgeführt.
- Mehrprogrammbetrieb (multi-tasking)
  Mehrere Programme werden gleichzeitig (bei mehreren CPUs) oder zeitlich verschachtelt, also quasi-parallel, bearbeitet.

nach der Anzahl der gleichzeitig am Computer arbeitenden Benutzer:

- Einzelbenutzerbetrieb (single-user mode)
  Der Computer steht einem einzigen Benutzer ungeteilt zur Verfügung.
- Mehrbenutzerbetrieb(multi-user mode)
  Mehrere Benutzer teilen sich die Computerleistung, sie sind über Terminals oder Netzwerkverbindungen mit dem Computer verbunden.

nach der Anzahl der verwalteten Prozessoren bzw. Rechner:

Es geht jedoch nicht darum, wieviel Prozessoren allgemein in einem Rechner verwendet werden, sondern wieviel *Universalprozessoren* für die Verarbeitung der Daten zur Verfügung stehen.
Somit ergibt sich folgende Unterscheidung:

- Ein-Prozessor-Betriebssystem
  Die meisten Rechner, die auf der von-Neumann-Architektur aufgebaut sind, verfügen über nur einen Universalprozessor. Aus diesem Grund unterstützen auch die meisten Betriebssysteme für diesen Anwendungsbereich nur einen Prozessor.

- Mehr-Prozessor-Betriebssystem
  Für diese Klassifizierung der Betriebssysteme ist noch keine Aussage über die Kopplung der einzelnen Prozessoren getroffen worden. Auch gibt es keinen quantitativen Hinweis über die Anzahl der Prozessoren, nur, daß es mehr als ein Prozessor ist. Für die Realisierung der Betriebssysteme für die Mehrprozessorsysteme gibt es zwei Herangehensweisen:

  - Jedem Prozessor wird durch das Betriebssystem eine eigene Aufgabe zugeteilt. D.h., es können zu jedem Zeitpunkt nur soviel Aufgaben bearbeitet werden, wie Prozessoren zur Verfügung stehen. Es entstehen Koordinierungsprobleme, wenn die Anzahl der Aufgaben nicht gleich der Anzahl verfügbarer Prozessoren ist.

  - Jede Aufgabe kann prinzipiell jedem Prozessor zugeordnet werden, die Verteilung der Aufgaben zu den Prozessoren ist nicht an die Bedingung gebunden, daß die Anzahl der Aufgaben gleich der Anzahl Prozessoren ist. Sind mehr Aufgaben zu bearbeiten, als Prozessoren vorhanden sind, so bearbeitet ein Prozessor mehrere Aufgaben "quasi-parallel". Sind mehr Prozessoren als Aufgaben vorhanden, dann bearbeiten mehrere Prozessoren eine Aufgabe, falls möglich.

  Das Betriebssystem kann dabei seinerseits auch auf mehrere Prozessoren verteilt sein. Man spricht dann von "verteilten Betriebssystemen".

Betriebssystemkonzepte

Schnittstelle des BS

Systemaufrufe, d.h. die erweiterten Befehle der virtuellen Maschine

Systemaufrufe erzeugen, benutzen und zerstören Software-Objekte, die durch das BS verwaltet werden.

Die wichtigsten dieser Objekte sind *Prozesse* und *Dateien*

Prozesse

Ein *Prozeß* besteht aus:

- einem ausführbaren Programm,
- den Programmdaten,
- dem Kellerinhalt (Stack),
- dem Programmzähler,
- dem Kellerzeiger (Stackpointer),
- Registerinhalten,
- und weiteren Informationen, die zur Programmausführung benötigt werden.

Die *Prozeßtabelle* beinhaltet alle Informationen (bis auf den Inhalt des Adreßraumes) aller Prozesse.

Prozesse werden durch Systemaufrufe erzeugt und beendet.

Prozesse können durch einen speziellen Systemaufruf (fork) Kindsprozeße erzeugen.

Prozeße können Nachrichten austauschen.

Die einfachste form der Nachrichtenübermittlung sind *Signale*.

Die Benutzer- und Gruppenidentikation ermöglicht Schutzfunktionen (Zugang, Abrechnung, Rechte, etc.).

Dateien

Systemaufrufe können Dateien erzeugen, lesen, beschreiben und löschen.

Dateien werden im BS durch *Dateideskriptoren* (File descriptors, handle) verwaltet.

Dateien werden in einer hierarischen Organisationsform, den *Verzeichnissen*, gespeichert.

Jedem Prozeß ist ein aktuelles Arbeitsverzeichnis zugeordnet, von dem aus die relativen Pfadnamen aufgelöst werden.

Von den speziellen Eigenschaften der Ein- und Ausgabegeräten wird abstrahiert, indem Spezialdateien (special files) bereitgestellt werden. Unterschieden wird zwischen:

- *blockorientierte Spezialdateien*
- *zeichenorientierte Spezialdateien*

Besondere Dateideskriptoren sind die Standardeingabe, Standardausgabe und die Standardfehlerausgabe.

Prozeße können durch eine besondere Form der Datei, der Pipe, miteinander kommunizieren.

Pipes sind unidirektionale Datenkanäle zwischen zwei Prozessen. Ein Prozeß schreibt Daten in den Kanal und ein anderer Prozeß liest die Daten in der gleichen Reihenfolge wieder aus. Die Realisierung kann im Hauptspeicher oder als Datei erfolgen.

Systemaufrufe

Benutzerprogramme kommunizieren mit dem BS durch Systemaufrufe.

Jedem Systemaufruf entspricht eine *Bibliotheksfunktion*. Beim Aufruf wird die Kontrolle vom Benutzerprogramm an das BS übergeben und die Bibliotheksfunktion im priveligierten Modus (auch *Supervisor Modus, Kernaufruf*) ausgeführt.

Kommandointerpreter (Shell)

Das BS ist das Programm, das alle Systemaufrufe ausführt.

Der *Kommandointerpreter* ist die Schnittstelle zwischen Benutzer und BS.

Der Kommandointerpreter wird als Kindsprozeß des BS gestartet, wenn der Benutzer seine Arbeit aufnimmt.

Der Benutzer kann nun Programme als Kindsprozeß des Kommandointerpreters starten oder Kommandos eingeben, die vom Kommandointerpreter als Systemaufrufe an das BS weitergegeben werden.

Betriebssystemstrukturen

Monolithische Systeme

Hierbei handelt es sich um BS, die keine ausgezeichnete Struktur besitzen.

Das BS besteht aus einer Menge von Prozeduren.

Monolithische Systeme verfügen über keinen Mechanismen zum Verbergen von Information.

Wenn überhaupt, so lassen sich folgende Bestandteile erkennen:

Ein *Hauptprogramm*, das die angeforderte Dienstprozedur aufruft.

Eine Menge von *Dienstprozeduren*, die die Systemaufrufe ausführen.

Eine Menge von *Hilfsprozeduren*, die die Dienstprozeduren unterstützen.

Geschichtete Systeme

Eine Weiterentwicklung monolithischer Systeme besteht darin, das BS als eine Hierarchie von Schichten zu organisieren.

Beispiel: THE-Betriebssystem

5	Operateur
4	Benutzerprogramme
3	Ein-/Ausgabeverwaltung
2	Operateur-zu-Prozeß-Kommunikation
1	Speicherverwaltung
0	Prozessorvergabe und Multiprogrammierung

Virtuelle Maschinen

Virtuelle Maschinen sind exakte Kopien der Hardware.

Beispiel: VM/370

<table>
<tr><td>Virtuelle 370
CMS</td><td>Virtuelle 370
CMS</td><td>Virtuelle 370
CMS</td></tr>
<tr><td colspan="3">VM/370</td></tr>
<tr><td colspan="3">370 Hardware</td></tr>
</table>

Unterste Ebene: Mehrprogrammbetrieb durch "virtual machine monitor", virtuelle Maschinen als identische Kopien der darunterliegenden Hardware mit Nachbildung von Benutzer/Supervisor-Modi, I/O, Unterbrechungen (Simulation mehrerer /370 Rechner).

Weil jede virtuelle Maschine identisch mit der konkreten Hardware ist, kann auf jeder von ihnen jedes, insbesondere auch unterschiedliche, BS arbeiten, das auch unmittelbar auf der Hardware arbeitet.

Durch die vollständige Trennung der Funktion des Mehrprogrammbetriebs und der Bereitstellung einer virtuellen Maschine wird jeder der Teile deutlich einfacher, wesentlich flexibler und leichter zu warten.

Client-Server-Modell

Moderne BS bestehen auf dem *Kern* und *Benutzerprogrammen*.

Der Kern enthält – nach Möglichkeit - nur die Funktionen, die im priviligierten Modus ablaufen müssen. Dadurch reduziert sich die Größe des Kerns. Solche Kerne werden ***Microkernel*** genannt.

Für das im ***Kernel-Modus*** (priviligierten Modus) laufende Programm gelten keinerlei Einschränkungen, es kann alle möglichen Funktionen des Prozessors verwenden. Programmen, die im ***User-Modus*** (nicht-priviligierten Modus) laufen, ist die Ausführung diverser Befehle verboten, die Auswirkungen auf andere Prozesse des Systems haben könnten.

Die Benutzerprogramme stellen den größten Teil der Funktionalität zur Verfügung.

Die Benutzerprogramme sind spezialisierte Dienste. Komplexe Aufgaben werden durch Zusammenwirken mehrerer Benutzerprogramme erledigt. Dabei fordert ein Programm, der *Client-Prozess*, von einem oder mehreren Programmen, den *Server-Prozeß* an.

Clients und Server laufen als nicht-priveligierte Programme ab.

Über Systemaufrufe an den Kern können sie auf die privilegierten Funktionen zugreifen.

Die Kommunikation zwischen Client und Server verläuft transparent, d.h. es wird prinzipiell nicht zwischen *lokalen* und *verteilten* Diensten unterschieden.

Sehr oft werden die *Mechanismen* im Kern implementiert und die *Strategien* den Servern überlassen.

Zusammenfassung

BS können unter zwei Sichtweisen betrachtet werden:
Betriebsmittelverwalter und *virtuelle Maschine*.

Betreibsmittelverwaltung hat das Ziel, die Ressourcen möglichst effizient zu verwalten.

Virtuelle Maschinen sind leichter zu handhaben als die reale Maschine.

Die wesentlichen Konzepte von BS sind *Prozeß* und *Datei*.

BS können unterschiedlich strukturiert sein:

- Monolithische Systeme
- Hierarchie von Schichten
- Virtuelle Maschine
- Client-Server-Architektur

PROZESSE

Einführung

- Eines der beiden wichtigsten „Konzepte“ eines BS ist das *Prozeßkonzept*.
  - Ein *Prozeß* ist die Abstraktion eines sich in Ausführung befindlichem Programms.
- Pseudo-Parallelität
  - Alle modernen Computer können mehrere Aufgaben quasigleichzeitig erledigen.
  - Die Kontrolle vieler nebenläufiger Prozesse ist eine schwierige Aufgabe!

Das Prozeßmodell

Alle ausführbaren Programme werden in *sequentielle Prozesse* zerlegt.

Jeder Prozeß besitzt - konzeptionell gesehen - seinen eigenen Prozessor.

In der Realität wird der Prozessor zwischen den Prozessen hin und hergeschaltet.

Das Wechseln des Prozessors wird *Mehrprogrammbetrieb* genannt.

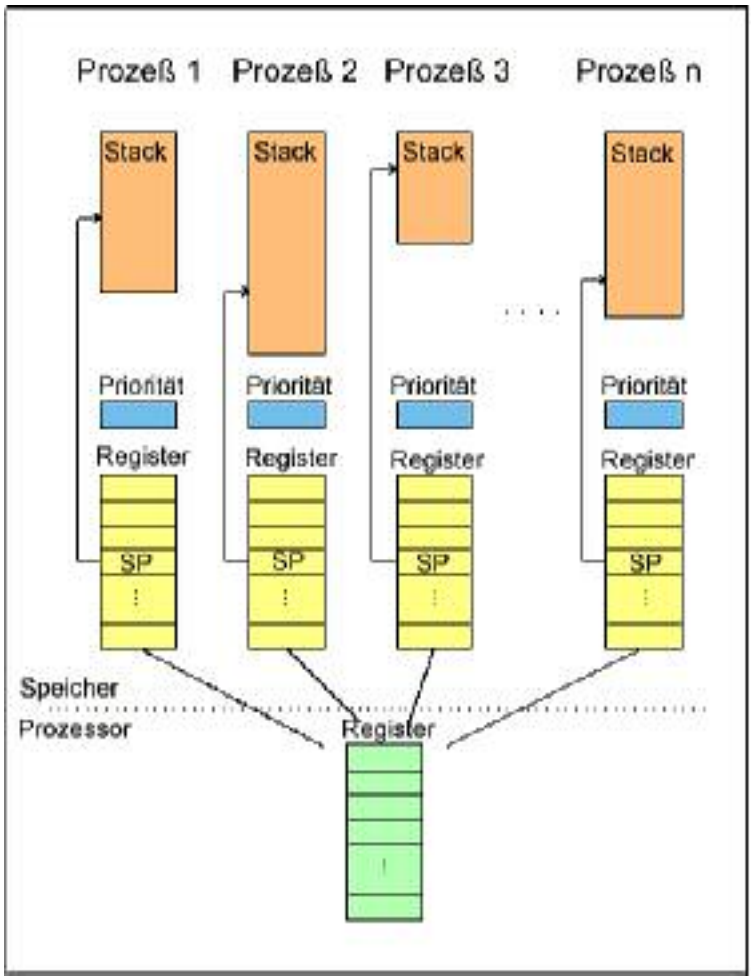
Jeder Prozeß besitz seinen eigenen Befehlszähler, die unabhängig voneinander geführt werden.

Die Ausführungsgeschwindigkeit der Prozesse ist ungleichmässig und nicht reproduzierbar. D.h., es können keine a-priori-Annahmen über den zeitlichen Verlauf gemacht werden.

Unterschied zwischen Prozeß und Programm:

*Ein **Programm** ist ein Algorithmus in einer geeigneten Notation.*

*Ein **Prozeß** ist eine Aktivität, die aus Programm, Eingaben, Ausgaben und einem Zustand besteht.*



Prozeßhierarchien

In den meisten Betriebssystemen wird die Möglichkeit benötigt, während des Betriebs Prozesse zu erzeugen und zu beenden.

Beispiel: UNIX stellt den Systemaufruf *fork* zur Verfügung, mit dem eine identische Kopie des aufrufenden Prozesses erzeugt wird. Vater und Kindsprozeß laufen nebenläufig weiter.

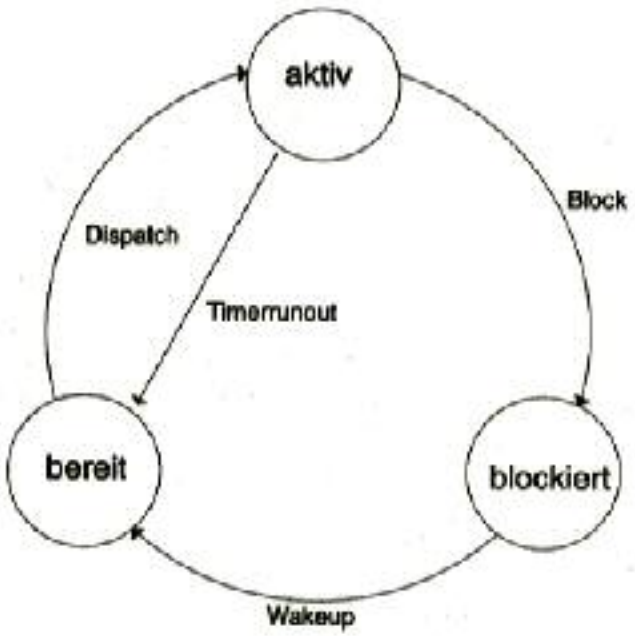
In MS-DOS gibt es einen ähnlichen Systemaufruf, mit dem Unterschied, daß der Vaterprozeß suspendiert wird, bis der Kindsprozeß beendet wird.

Prozeßzustände

Obwohl jeder Prozeß eine unabhängige Einheit mit einem eigenen Befehlszähler und einem eigenen inneren Zustand ist, müssen Prozeße oft miteinander interagieren.

Es gibt folgende Prozeßzustände:

- **aktiv/rechnend** Dem Prozeß wird ein Prozessor zugeteilt.
- **rechenbereit** Der Prozeß ist ausführbar, aber der Prozessor ist einem andere Prozeß zugeteilt.
- **blockiert** Der Prozeß kann solange nicht ausgeführt werden, bis ein externes Ereignis eintritt.



- Dispatch: bereit --> aktiv
  Zuteilung der CPU an einen Prozeß.
- Timerrunout: aktiv --> bereit
  Nach Ablauf einer Zeitscheibe wird dem Prozeß die CPU wieder entzogen.
- Block: aktiv --> blockiert
  Aktiver Prozeß hat eine E/A-Operation angefordert (oder sich selbst für eine bestimmte Zeit verdrängt), bevor seine Zeitscheibe abgelaufen war. Dies ist der einzige Zustandswechsel, den ein Prozeß selbst auslösen kann.
- Wakeup: blockiert --> bereit
  Das Ereignis, auf das der Prozeß gewartet hat ist eingetreten. An den Prozeß eine Nachricht schicken.

Implementierung der Prozesse

Zur Implementierung des Prozeßmodells verwaltet das Betriebssystem eine *Prozeßtabelle*.

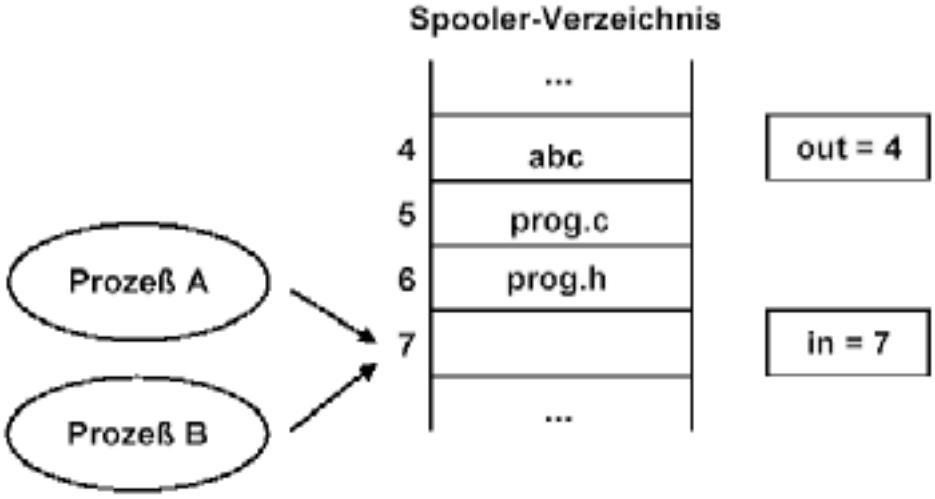
Unterbrechungsbehandlung:

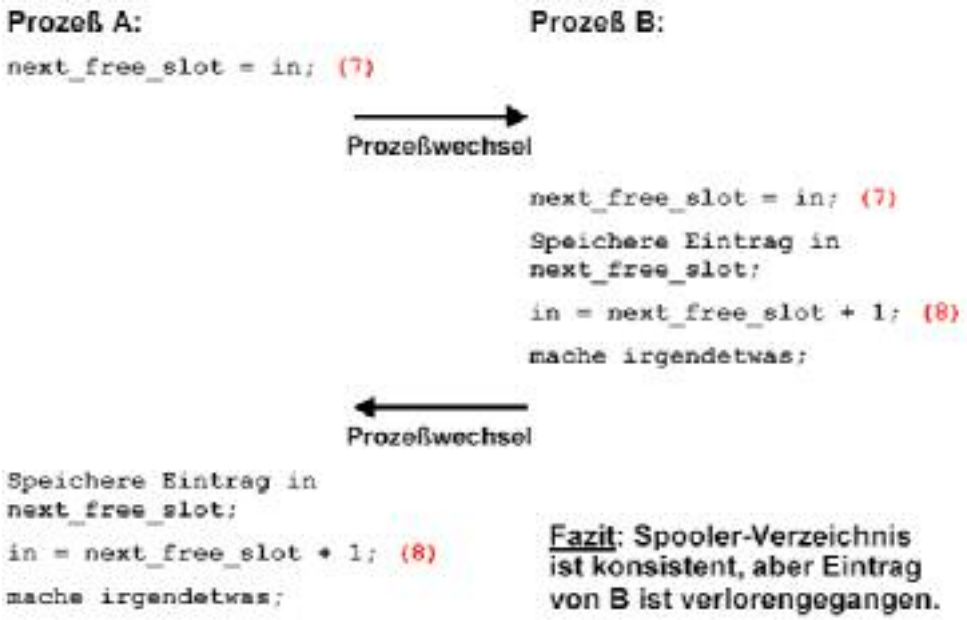
1. Der Programmzähler usw. werden durch die Hardware auf den Keller gelegt.
2. Die Hardware lädt den neuen Programmzähler aus dem Unterbrechungsvektor.
3. Ein Maschinenprogramm rettet die Registerinhalte.
4. Ein Maschinenprogramm bereitet den neuen *Stack* vor.
5. Die Unterbrechung wird behandelt, indem ein Prozeß gestartet wird.
6. Ein Dienstprogramm markiert den neuen Prozeß als rechenbereit.
7. Der Scheduler bestimmt den Prozeß, der als nächster ausgeführt werden soll.
8. Das Dienstprogramm gibt die Kontrolle an das Maschinenprogramm zurück.
9. Das Maschinenprogramm setzt den unterbrochenen Prozeß fort.

Prozesskommunikation

Zeitkritische Abläufe

Beispiel: Drucker-Spooler





Zeitkritische Abläufe sind solche Situationen, in denen zwei oder mehr Prozesse gemeinsam benutzte Daten lesen und schreiben und die Endergebnisse von der zeitlichen Reihenfolge der Lese- und Schreiboperationen abhängig sind.

Kritische Bereiche

Zeitkritische Abläufe lassen sich durch *wechselseitigen Ausschluß* vermeiden.

Der Teil eines Programms, bei dem auf gemeinsam benutzten Speicher zugegriffen wird, wird als *kritischer Bereich* bezeichnet.

- Zu jedem Zeitpunkt darf sich höchstens ein Prozeß in einem kritischen Abschnitt befinden.
- Es dürfen keine Annahmen über die Ausführungsgeschwindigkeit oder die Anzahl der Prozessoren gemacht werden.
- Kein Prozeß, der sich nicht in einem seiner kritischen Abschnitte befindet, darf andere Prozesse blockieren.
- Kein Prozeß soll unendlich lange warten müssen, bis er in seinen kritischen Bereich eintreten kann.

Wechselseitiger Ausschluß mit aktivem Warten

Sperren mit Unterbrechungen

Die einfachste Lösung ist es, wenn ein Prozeß vor dem Eintritt in seinen kritischen Abschnitt alle Unterbrechungen sperrt und erst wieder freigibt, wenn er den kritischen Abschnitt verläßt. Das Sperren aller Unterbrechungen ist innerhalb des Kerns eine nützliche Technik, eignet sich aber nicht für den wechselseitigen Ausschluß von Benutzerprozessen.

Sperrvariablen

Es wird eine logische Variable geführt, die mit 0 den Eintritt in den kritischen Bereich erlaubt und mit 1 sperrt. Der in den kritischen Bereich eintretende Prozeß muß dann vor seinem Eintritt prüfen, ob der Bereich frei ist, die Sperrvariable auf 1 setzen und nach Ende wieder freigeben.
Dieser Ansatz ist ungeeignet, da der Zugriff auf die Sperrvariablen selbst wieder ein zeitkritischer Ablauf ist.

Striktes Alternieren

Striktes Alternieren läßt sich durch Einführung einer Variablen und aktivem Warten realisieren. Obwohl dieser Algorithmus zeitkritische Abläufe ausschließt, ist er nur für fixe Ablaufreihenfolgen geeignet.

Petersons Lösung

Bevor ein Prozeß eine gemeinsam benutzte Variable verwendet, ruft er die Prozedur **enter_region** auf, der er seine Prozeßnummer, 0 oder 1, als Parameter übergibt. Dieser Aufruf bewirkt, daß er wartet, falls es notwendig ist, bis er in seinen kritischen Bereich eintritt. Wenn er die Manipulation der gemeinsam benutzten Variablen beendet hat, ruft er die Prozedur **leave_region** auf und verläßt damit seinen kritischen Abschnitt.

Semaphore

Semaphoren wurde 1965 von E.W. Dijkstra vorgeschlagen

Originalnotation als P- und V-Operationen, jedoch als DOWN und UP leichter lesbar.

Semaphor-Operationen sind atomar.

- DOWN
  - Überprüft, ob der Wert der Semaphore größer als Null ist.
  - Falls ja, wird der Wert um Eins erniedrigt und der Prozeß fortgesetzt.
  - Falls nein, wird der ausführende Prozeß blockiert (schlafen gelegt).
- UP
  - Der Wert der Semaphore wird um Eins erhöht.

- Haben sich Prozesse an der Semaphore blockiert, wird einer ausgewählt und aktiviert (aufgeweckt).
- Semaphoren können sowohl für den *gegenseitigen Ausschluß* (mutual exclusion) wie auch für die *Synchronisation* verwendet werden.

Monitore

Semaphore können zwar alle Synchronisationsprobleme lösen, sind in der Anwendung aber sehr fehlerträchtig.

Es besteht deshalb die Notwendigkeit, höhere Synchronisationsmechanismen einzusetzen.

Monitore wurde 1974/75 von Hoare und Hansen vorgeschlagen.

- Ein *Monitor* besteht aus einer Menge von Prozeduren, Variablen und Datenstrukturen, die in einem besonderen Paket zusammengefasst sind.
- Prozesse können Prozeduren auf einem Monitor aufrufen, wann immer sie es möchten, aber sie können nicht über Prozeduren, die außerhalb des Monitors deklariert sind, auf die internen Datenstrukturen des Monitors zugreifen.
- In einem Monitor kann zu jedem Zeitpunkt nur höchstens ein Prozeß aktiv sein.
- Monitor werden von der verwendeten Programmiersprachen zur Verfügung gestellt. Der gegenseitige Ausschluß wird durch den Compiler unter Verwendung von Semaphoren realisiert. Durch die automatische Erzeugung des Codes ist ein Fehler aber weniger wahrscheinlich als bei manueller Erstellung.

Bedingungsvariablen

Bedingungsvariablen und die auf ihnen definierten Funktionen WAIT und *SIGNAL* ermöglichen es, Prozesse zu blockieren bzw. zu aktivieren.

Die Funktion **WAIT** wird von einem Prozeß aufgerufen, wenn er feststellt, daß er nicht weiterarbeiten kann, Er blockiert sich damit an einer Bedingungsvariablen.

Die Funktion **SIGNAL** auf einer Bedingungsvariablen ausgeführt, aktiviert einen an dieser Bedingungsvariablen blockierten Prozeß.

Ruft ein Prozeß innerhalb des Monitors die WAIT Funktion auf, so blockiert er sich und ein anderer Prozeß kann den Monitor betreten.

Ruft ein Prozeß innerhalb des Monitors die SIGNAL Funktion auf, so muß anschließend der Monitor sofort verlassen werden, damit nicht zwei Prozesse im Monitor aktiv sind. M.a.W., die SIGNAL Funktion muß als letzte Anweisung in der Monitorprozedur stehen.

Monitore-Fazit

Semaphore sind ein sehr einfacher Mechanismus, der einen gemeinsamen Speicher voraussetzt. Dies ist in verteilten Anwendungen nicht gegeben.

Monitore werden nur in wenigen und keiner „gängigen“ Sprache unterstützt.

Nachrichtenaustausch

Der Mechanismus des *Nachrichtenaustausches* benutzt die Primitive SEND und RECEIVE, die als Systemaufrufe implementier sind.

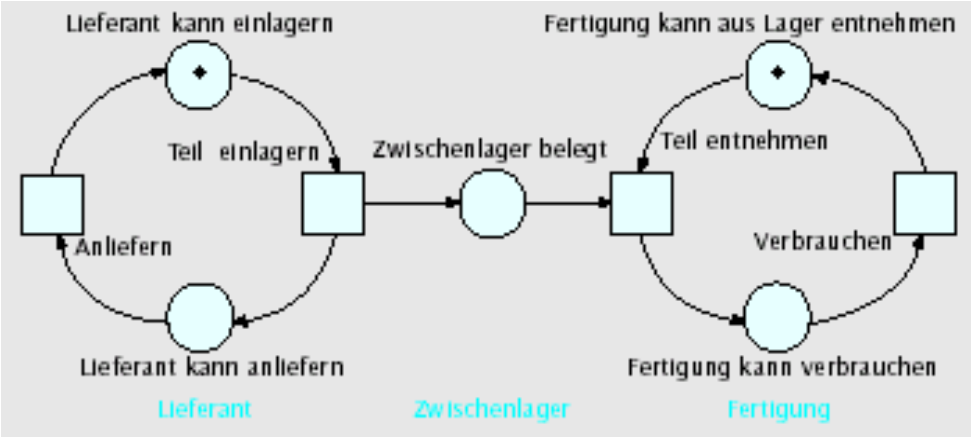
SEND (destination, message) sendet eine Nachrichtan den Empfänger (destination).

RECEIVE (destination, message) empfängt eine Nachricht, und kann sich blockieren, bis die Nachricht eintrifft.

Nachrichtenaustausch ist geeignet, um in verteilten System eingesetzt zu werden. Allerdings treten dort zusätzliche Problem auf:

- es können Nachrichten oder Quittungen verloren gehen,
- Namen müssen eindeutig sein (process@machine.domain),
- Authentifikation,
- Performance.

Beispiel Erzeuger-Verbraucher-Problem



Rendezvous

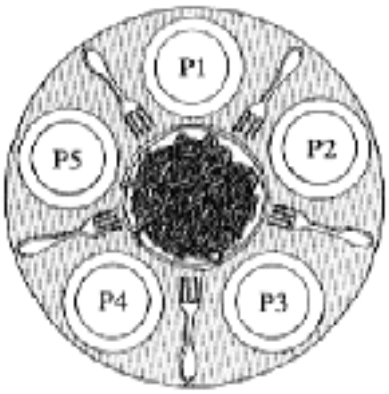
- Bei einem *Rendezvous* wird jede Pufferung von Nachrichten vermieden, die Prozesse synchronisieren sich und tauschen (beim Rendezvous) die Nachrichten direkt miteinander aus.
- Falls der Empfänger schon bereit ist, wenn der Sender senden will, werden die Nachrichten sofort ausgetauscht, anderenfalls blockiert sich der Sender, bis der Empfänger seine Bereitschaft signalisiert.

Prozesskommunikationsprobleme

Das Philosophenproblem (Dijkstra, 1965)

Grundsätzliches:

Fünf Philosophen sitzen um einen Tisch herum. Jeder Philosoph hat einen Teller Spaghetti vor sich. Damit ein Philosoph Spaghetti essen kann, braucht er zwei Gabeln. Zwischen je zwei Tellern liegt eine von insgesamt fünf Gabeln.



Die Aufgabe:

Das Leben der Philosophen besteht aus den sich abwechselnden Phasen *Essen* und *Denken*.

Wenn der Philosoph hungrig wird, versucht er, in einer beliebigen Reihenfolge nacheinander seine linke und rechte Gabel aufzunehmen.

Hat er erfolgreich beide Gabeln aufgenommen, ißt er einige Zeit, legt dann die Gabeln wieder ab und setzt das Denken fort.

Problem: Kann man für jeden Philosophen ein Programm angeben, das die Aufgabe löst?

Frage: Unter Welchen Bedingungen kann etwas schiefgehen?

- Verklemmung (deadlock)
- Verhungern

Das Leser-Schreiber-Problem (Courtois, 1971)

Das Problem:

In einem Prozeßsystem soll es erlaubt sein, daß viele Prozesse gleichzeitig lesend auf einen Datenbestand zugreifen.

Wenn geschrieben werden muß, so darf nur der schreibenden Prozeß auf die Daten zugreifen, sonst niemand.

Das Problem des schlafenden Friseurs

Die Aufgabe:

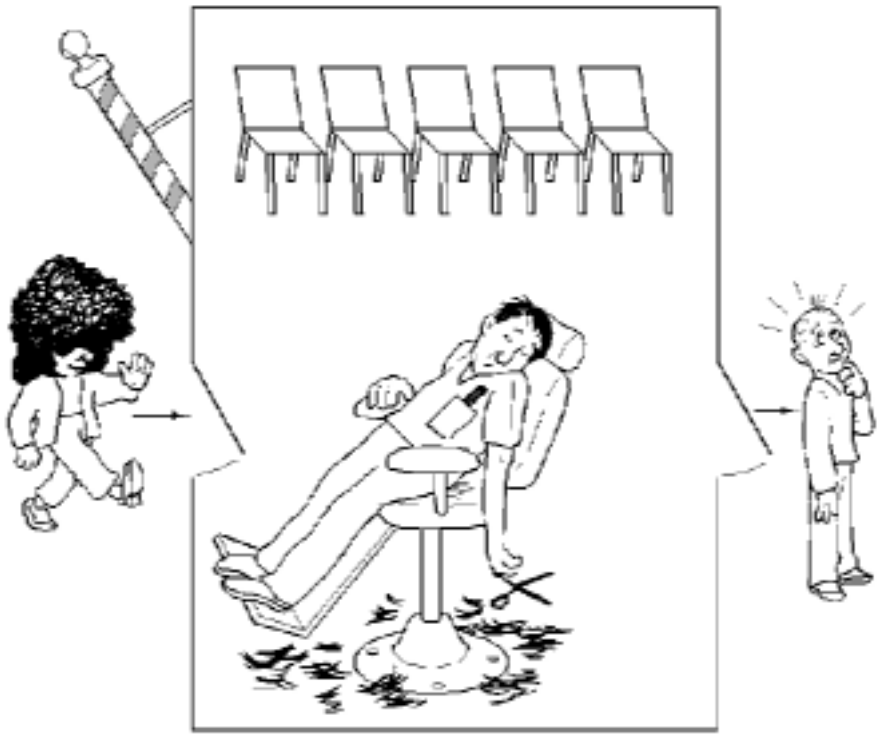
In einem Friseurladen arbeitet ein Friseur und es gibt einen Friseurstuhl und *n* Stühle für wartende Kunden.

Falls kein Kunde anwesend ist, schläft der Friseur.

Ein eintreffender Kunde muß den Friseur wecken.

Falls Kunden eintreffen, während der Friseur Haare schneidet, nehmen die Kunden - falls noch Platz ist - auf den Stühlen Platz oder - sonst - verlassen sie den Laden.

Das Probem: Kann man für Friseur und Kunden Programme angeben, die das Problem lösen, ohne daß zeitkritische Abläufe entstehen?



Scheduling

Vorbemerkung

In den vorangegengenen Beispielen trat öfters die Situation auf, in der mehr als ein Prozeß ausführbar waren. Der Teil des BS, der mit der Entscheidung betraut ist, welcher Prozeß als nächster ausgeführt wird, heißt *Scheduler*.

Der von Scheduler verwendete Algorithmus heißt *Scheduling-Algorithmus*.

Mögliche Aufgaben des Schedulers:

- **Fairneß** Jeder Prozeß erhält einen gerechten Anteil der Prozessorzeit.
- **Effizienz** Der Prozessor wird immer vollständig ausgelastet.
- **Antwortzeit** Die Antwortzeit für die interaktiv arbeitenden Benutzer wird immer minimiert.
- **Verweilzeit** : Die Wartezeit auf die Ausgaben von Stapelaufträgen wird minimiert.
- **Durchsatz** Die Anzahl der Aufträge, die in einem bestimmten Zeitintervall ausgeführt werden, wird maximiert.

Schedulingverfahren, bei dem rechenbereite Prozesse suspendiert werden können, werden ***Pre-emptive-Scheduling*** genannt.

Scheduling-Verfahren, bei dem rechenbereite Prozesse nicht suspendiert werden können, werden *Run-to-Completion-Verfahren* genannt (oder **Non-preemptive**).

Für den praktischen Einsatz in Rechnern sind nur Pre-emptive-Schedulingverfahren von Bedeutung.

Round-Robin-Scheduling

Das älteste, einfachste, weitverbreitetste und fairste Verfahren ist *Round-Robin*.

Jeder Prozeß bekommt ein gewisses Zeitintervall, das *Quantum* genannt wird, für seine Ausführung zugeteilt. Annahme: Alle Prozesse gleich wichtig, also erhalten alle dasselbe Quantum.

Falls das Quantum eines Prozesse abgelaufen ist, wird ihm der Prozessor entzogen und einem anderen Prozeß gegeben.

Wenn sich ein Prozeß blockiert oder seine Ausführung beendet, bevor das Quantum abgelaufen ist, wird ein Prozeßwechsel vollzogen und evtl. Reste des Quantums verfallen.

Die Implementierung ist einfach: Die Prozesse werden in einer Liste geführt. Der erste rechnet, suspendierte Prozesse wandern an das Ende der Liste.

Wichtige Frage: Wie lang soll das Quantum sein?

Fazit: Falls das Quantum zu klein ist, sinkt wegen der häufigen Prozeßwechsel die Prozessorausnutzung, und falls das Quantum zu groß gewählt wurde, erhält man bei kurzen interaktiven Anfragen schlechte Antwortzeiten.

Prioritäts-Scheduling

Beim *Prioritäts-Scheduling* wird jedem Prozeß eine Priorität zugewiesen, und der ausführbereite Prozeß mit der höchsten Priorität wird ausgeführt.

Damit verhindert wird, daß Prozesse mit einer hohen Priorität zu lange ausgeführt werden, erniedrigt der Scheduler die Priorität des gerade ausgeführten Prozesses bei jeder Unterbrechung.

Prioritäten können *statisch* oder *dynamisch* vergeben werden.

Prozesse werden häufig in Prioritätsklassen eingeteilt, wobei dann innerhalb der Klassen Round-Robin-Scheduling betrieben wird und Prioritäts-Scheduling nur zwischen den Klassen.

Shortest-Job-First

Shortest-Job-Fist ist eine Strategie, die besonders gut bei Stapelaufträgen geeignet ist, bei denen die Ausführungszeiten im voraus bekannt sind.

Sind alle Ausführungszeiten zu Beginn bekannt, ist Shortest-Job-First eine optimale Strategie und liefert minimale durchschnittliche Verweilzeiten.

Shortest-Job-First ist nur bedingt auf interaktive Prozesse anwendbar, da diese kein festes Ausführungsmuster besitzen.

Betrachtet man die Zeiten zwischen den Unterbrechungen eines interaktiven Prozesses als Auftrag, stellt sich die Frage, wie man dessen Bearbeitungszeit ermittelt.

Ansatz: Man schließt von der Vergangenheit auf die Zukunft und nimmt immer den Prozess mit der kürzesten geschätzten Ausführungszeit.

Zweistufiges Scheduling

Wenn nicht alle rechenbereiten Prozesse im Hauptspeicher Platz finden, müssen einige Prozesse auf den Hintergrundspeicher ausgelagert werden.

Diese Situation hat sehr große Auswirkungen auf das Scheduling, da die Zeit für den Prozeßwechsel zu einem Prozeß, der auf dem Hintergrundspeicher

abgelegt ist, um einige Größenordnungen höher ist, als zu einem Prozeß, der sich im Hauptspeicher befindet.

Zweistufiges Scheduling ist ein praktisch orientiertes Verfahren, um ausgelagerte Prozesse zu behandeln.

Der Scheduler der unteren Stufe erfüllt die Aufgabe, zwischen den sich im Hauptspeicher befindlichen Prozessen auszuwählen, während der Scheduler der oberen Stufe Prozesse zwischen Haupt- und Hintergrundspeicher transportiert.

Kriterien für den Scheduler der oberen Stufe:

- Wie lange befindet sich ein Prozeß im Haupt- bzw. Hintergrundspeicher?
- Wieviel Prozessorzeit hat ein Prozeß verbraucht?
- Wie hoch ist die Priorität des Prozesses?

Zusammenfassung

BS benutzen ein Modell, das aus sequentiellen Prozessen besteht, die *nebenläufig* ausgeführt werden, um die Auswirkung von Unterbrechungen verbergen zu können.

Ein System heißt ***nebenläufig***, wenn es Zustände besitzt, in denen mehrere lokale Aktionen unabhängig von einander ausgeführt werden können.

Jeder Prozeß besitzt seinen eigenen Zustand und kann als virtueller Prozessor betrachtet werden.

Interaktionen können zu zeitkritischen Abläufen führen, deren Verhalten nicht reproduzierbar ist.

Zur Vermeidung von zeitkritischen Abläufen wurde das Konzept der kritischen Bereiche eingeführt, die Teil des Programmkodes sind. Kritische Bereiche nutzen den wechselseitigen Ausschluß.

Prozesse können über Kommunikationsprimitive miteinander kommunizieren. Damit ist ein wechselseitiger Ausschluß realisierbar.

Die unterschiedlichen Prozeßkommunikationsprimitive sind im Prinzip alle äquivalent und unterscheiden sich nur durch ihre Handhabung.

Es sind viele unterschiedliche Scheduling-Algorithmen vorgeschlagen worden, um unter Berücksichtigung zahlreicher Faktoren den nächsten auszuführenden Prozeß auszuwählen.

SPEICHERVERWALTUNG

Vorbemerkung

Der Teil des Betriebssystem, der den Speicher verwaltet, wird *Speicherverwalter* genannt.

Seine Aufgabe ist:

- die Verwaltung der freien und belegten Speicherbereiche,
- die Zuweisung von Speicherbereichen an die Prozesse, wenn Speicher angefordert wird,
- die Freigabe, wenn der Speicher zurückgegeben wird,
- die Durchführung von Auslagerungen, falls der Hauptspeicher nicht groß genug ist, um alle Prozesse auf einmal zu halten.

Speicherverwaltung ohne Auslagerung

Speicherwaltung ohne Auslagerung der Prozesse auf die Festplatte sind die einfachsten Algorithmen.

Einprogrammbetrieb ohne Auslagerung

Nur ein Prozeß wird ausgeführt.

Diesem Prozeß wird der gesamte Speicher zugeteilt.

Diese Methode wird heute nicht mehr verwendet, da jeder Prozeß die Gerätetreiber für sämtliche Ein-/Ausgabegeräte enthalten muß.

Konsequenz: Nur ein Prozeß kann zu einem Zeitpunkt laufen!

Mehrprogrammbetrieb und Speicherbenutzung

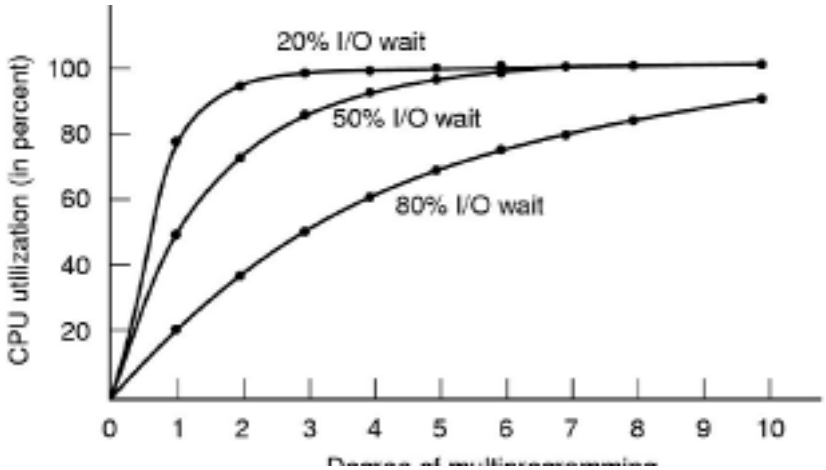
Mehrprogrammbetrieb ist schon aus Gründen der Effizienz unbedingt notwendig. Bsp: 40 msec für Einlesen der Daten, 10 msec für Verarbeitung führt zu 80% wartender CPU.

Modellieren des Mehrprogrammbetriebes

Annahme: Ein Prozeß verbringt den Bruchteil *p* seiner Zeit in einem Ein-/Ausgabewartezustand.

Mit *n* Prozessen auf einmal im Speicher beträgt die Wahrscheinlichkeit, daß alle *n* Prozesse auf eine Ein-/Ausgabe warten p^n.

Die CPU Auslastung ist durch ***Auslastung* = 1 - p^n** gegeben.



Übung: Berechnen Sie die Verbesserung der CPU Auslastung bei Erweiterung mit einem 2. und 3. MByte Speicher, wenn beim Grundausbau von 1 MByte das BS 200 KByte und jedes Benutzerprogramm ebenfalls 200 Kbyte beanspruchen.

Mehrprogrammbetrieb mit fixierten Partitionen

Wie muß der Speicher organisiert werden, damit mehr als ein Prozeß im Speicher sein kann?

- Einteilung des Speichers in *Partitionen*
- Wenn ein Auftrag kommt, wird dieser der Eingabewarteschlange der kleinsten Partition zugewiesen, die groß genug ist, um den Auftrag zu bearbeiten.
- Problem: Wenn die Warteschlange für große Partitionen leer ist, die Warteschlange für kleine Partitionen aber voll sind.

Relokation und Schutz

Das *Relokationsproblem* besteht darin, daß zum Zeitpunkt des Bindens des Programms nicht bekannt ist, an welchen Speicherplatz das Programm zur Ausführungszeit geladen wird.

Der Adreßbereich muß deshalb zu Beginn der Ausführung verschoben (relokiert) werden.

Zum Schutz anderer Prozesse wird die Hardware der Maschine mit einem Basis- und einem Grenzregister ausgestattet.

Das *Basisregister* enthält die Startadresse des rechnenden Prozesses.

Das *Grenzregister* die Länge der Partition.

Damit ist es möglich, Zugriffe auf Speicherzellen außerhalb der zugewiesenen Partion zu erkennen.

Durch Änderung des Basisregisters ist auch eine Relokation zur Laufzeit möglich.

Swapping

Reicht der Speicher nicht aus, um alle bereiten Prozesse im Speicher zu halten, ist es notwendig, Speicherbereiche vom Arbeitsspeicher auf den Hintergrundspeicher (Festplatte) aus- und wieder einzulagern. Dieser Vorgang wird ***Swapping*** genannt.

Mehrprogrammbetrieb mit variablen Partitionen

Um den Speicherverschnitt zu minimieren, werden Partitionen variabler Länge verwendet.

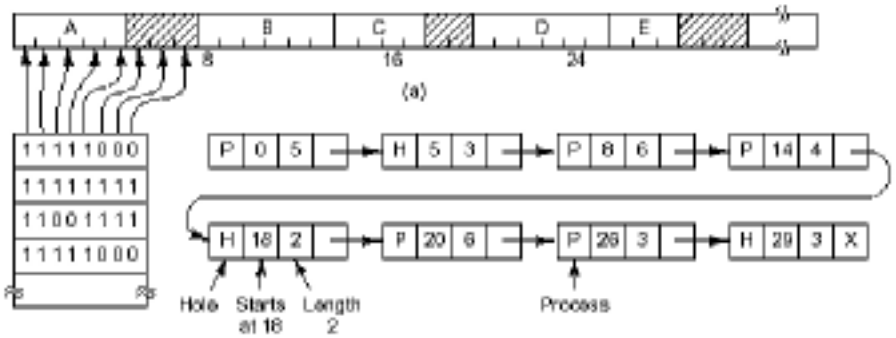
Die mit der Zeit entstehenden kleinen freien Bereiche können durch eine *Speicherbereinigung* (garbage collection) zu einem zusammenhängenden großen Bereich zusammengeführt werden.

Speicherverwaltung mit verkettete Listen

Mit verketteten Listen werden die freien und belegten Speicherbereiche verwaltet.

Jedes Listenelement verwaltet ein Segment, das frei oder belegt ist. Für die Verwaltung wird die Startadresse und die Länge des Segmentes benötigt, sowie Zeiger auf das nächste und vorhergehende Listenelement.

Durch *Verschmelzen* werden die frei werdenden Segmente mit den freien Nachbarn zu größeren Segmenten vereinigt, falls dies möglich ist.



Algorithmen:

First Fit

Durchsuchen der Segmentliste aufsteigend, bis ein Segment gefunden ist, das groß genug ist. Das Segment wird dann in den belegten Teil und den freien Teil aufgeteilt.

Vorteil: Kurze Suchzeiten.

Next Fit

Wie First Fit, nur daß mit der Suche dort begonnen wird, wo das letzte Mal ein Segment gefunden wurde.

Nachteil: Geringfügig schlechtere Performance als First Fit.

Best Fit

Die gesamte Liste wird durchsucht und das Segment mit dem kleinsten Verschnitt ausgewählt. Die Suche läßt sich beschleunigen, wenn die Liste der freien Segmente der Größe nach sortiert wird.

Nachteil: Die Suchzeit ist länger als bei First Fit. Best Fit tendiert dazu, sehr kleine und damit unbrauchbare freie Segmente zu produzieren.

Speicherverwaltung nach dem Buddy System

Idee: Der Rechner benutzt Binärzahlen zur Adressierung. Dies kann ausgenutzt werden, um die Verschmelzung angrenzender freier Segmente zu beschleunigen.

Vorgehen:

Die freien Segmente werden in Listen verwaltet, getrennt nach Segmentgröße von 1,2,4,8,16...*Speichergröße* Bytes.

Zu Beginn ist der gesamte Speicher frei, alle Listen sind leer, bis auf die für den gesamten Speicher.

Die Speichervergabe sucht passende Blöcke. Sind keine Blöcke der geeigneten Größe vorhanden, werden größere Blöcke durch Halbieren auf die benötigte Größe gebracht.

Bei der Freigabe werden benachbarte Blöcke verschmolzen, wenn sich dadurch ein Segment ergibt, daß auch bei der Verteilung entstanden wäre.

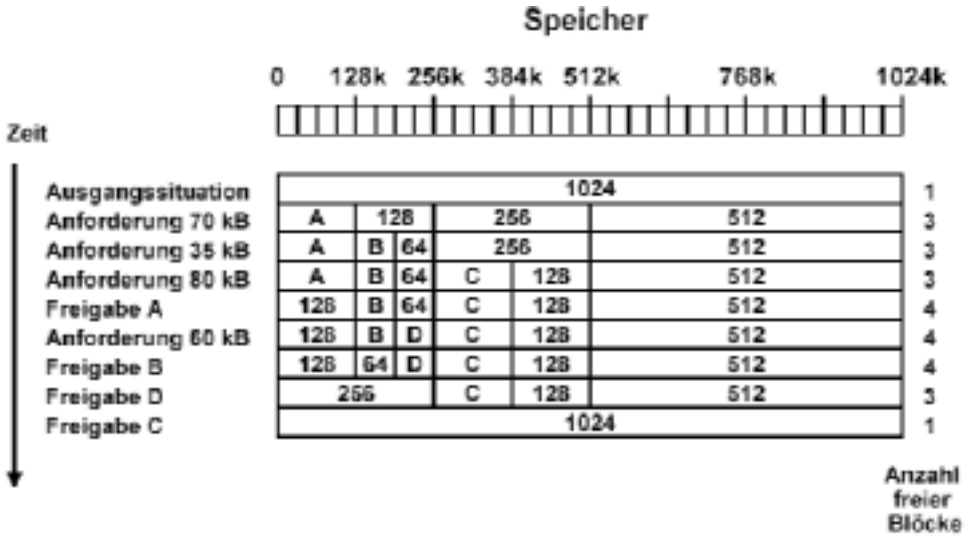
Buddy-Systeme sind extrem ineffizient bei der Ausnutzung des Speichers, es entsteht sogenannte **Fragmentierung**. Das Problem rührt von der Tatsache

her, daß alle Anfragen auf eine Potenz von 2 aufgerundet werden müssen. Ein 35K Prozeß allokiert somit 64K. Die ergänzenden 29K sind verschwendet. Diese Form des Überhangs wird *Interne Fragmentierung* genannt. Die Löcher zwischen den Segmenten wird *Externe Fragmentierung* genannt.

Fragmentierung

Interne Fragmentierung ist der Unterschied zwischen der Größe des zugeteilten Segments und der benötigten Größe.

Externe Fragmentierung sind die Lücken zwischen den zugeteilten Segmenten.



Virtueller Speicher

Problem: Programme sind größer als der verfügbare Speicher.

Die Grundidee des Konzepts des virtuellen Speichers ist es, zu erlauben, daß der Programmcode, die Daten und der Stack zusammen größer sein dürfen als der verfügbare Hauptspeicher.

Das BS hält die gerade benutzten Teile des Prozesses im Arbeitsspeicher und lagert den Rest auf den Hintergrundspeicher(Festplatte) aus.

Paging:

Die vom Programm benutzten Adressen werden als *virtuelle Adressen* bezeichnet und bilden den *virtuellen Adreßraum*.

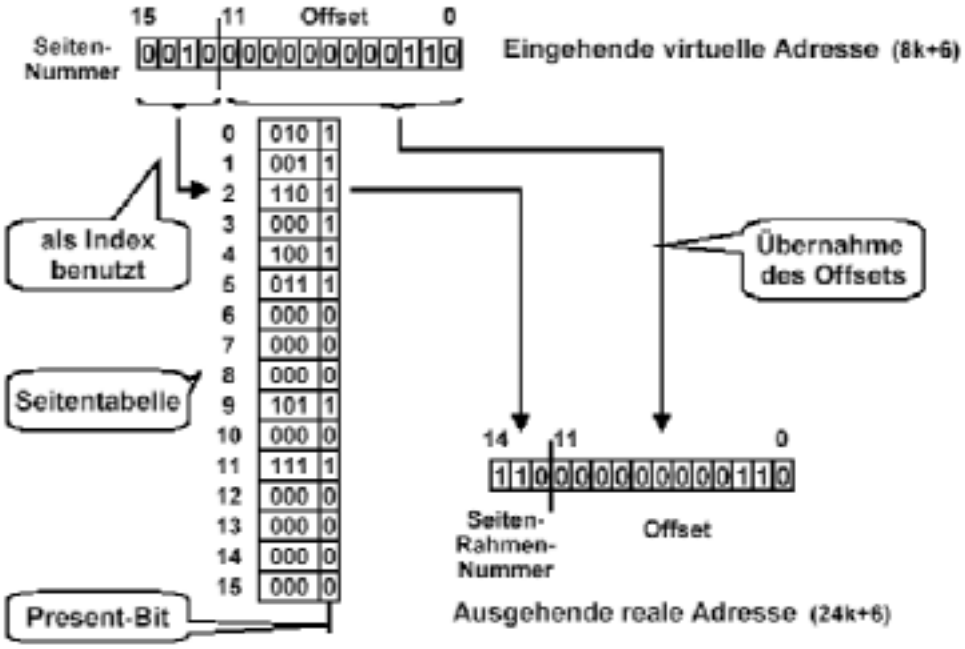
Die virtuelle Adresse wird von der MMU(*m*emory *m*anagement *u*nit) auf die physikalische Adresse umgerechnet.

Der virtuelle Adreßraum ist in gleichgroße Einheiten, den Seiten (pages), eingeteilt.

Die korrespondierenden Einheiten im physikalischen Speicher heißen *Seitenrahmen* oder *Kacheln*.

In der *Seitenkacheltabelle* wird darüber Buch geführt, welche Seiten gerade im Arbeitsspeicher sind und welche nicht.

Wird auf eine Seite zugegriffen, die momentan nicht im Arbeitsspeicher ist, entsteht ein *Seitenfehler* (page fault). Das BS wählt eine Seite aus, lagert diese aus und lagert die benötigte Seite ein.



Seitenkacheltabellen

Es hat sich als praktisch erwiesen, die Umrechnung von virtuellen zu physikalischer Adresse dadurch vorzunehmen, daß die virtuelle Adresse in eine virtuelle Seitennummer (höherwertige Bits) und einen Offset (niederwertige Bits) aufgeteilt wird. Die virtuelle Seitennummer ist ein Verweis in die Seitenkacheltabelle, um einen Eintrag für diese virtuelle Seite zu finden, die die Seitenrahmennummer (falls existent) angibt. Die physikalische Adresse ergibt sich damit aus der Seitenrahmennummer und dem Offset.

Probleme:

Die Seitenkacheltabelle kann extrem lang werden. Bsp.: Bei 32 Bit und einer Seitengröße von 4 KByte benötigt man 10^6 Einträge.

Die Umrechnung muß schnell erfolgen, da bei fast jedem Befehl 2-3 Adressen umgerechnet werden müssen. Lösung: Durch Hardwareunterstützung und/oder Halten der Tabelle im Arbeitsspeicher.

Seitenersetzungsalgorithmen

Vorbemerkungen

Falls die aus dem Arbeitsspeicher zu entfernende Seiten nicht verändert wurde, ist die Kopie auf dem Hintergrundspeicher aktuell und braucht nicht ersetzt zu werden.

Es empfiehlt sich, Seiten auszulagern, die nicht sehr häufig angesprochen werden. Tut man dies nicht, erhöht sich die Chance, daß die ausgelagerte Seite kurze Zeit später wieder eingelagert werden muß.

Algorithmen

Der optimale Algorithmus

Vorgehen: Ersetze die Seite, deren nächste Referenzierung am weitesten in der Zukunft liegt.

Dieser Algorithmus ist nicht realisierbar.

Not Recently Used

NRU benötigt die hardwaremässige Unterstützung von **benutzt** und **modifiziert** Bits zu jeder Seite.

Es entstehen die Klassen:

1. nicht referenziert, nicht modifiziert
2. nicht referenziert, modifiziert
3. referenziert, nicht modifiziert
4. referenziert, modifiziert

Der NRU entfernt zufällig eine der Seiten aus dem Speicher der aus der kleinstnummerierten, nichtleeren Klasse.

NRU ist einfach zu implementieren und hat eine akzeptable Performance.

First In, First Out

Eine Liste verwaltet die Zeitpunkte der Seiteneinlagerung.

Die Seite, die am längsten im Speicher war, wird ausgelagert.

FIFO wird in dieser reinen Form selten eingesetzt, da auch intensiv genutzte Seiten ausgelagert werden.

Second Chance

Second Chance vermeidet es - wenn es geht - intensiv genutzte Seiten aus dem Speicher zu entfernen.

Soll eine Seite ersetzt werden, deren Benutzt-Bit gesetzt ist, wird das Benutzt-Bit gelöscht und der Einlagerungszeitpunkt auf die aktuelle Zeit gesetzt und die Seiten ans Ende der Liste verschoben.

Ist das Benutzt-Bit nicht gesetzt, ist die Seite alt und unbenutzt und kann sofort ersetzt werden.

Least Recently Used

Idee: Wenn Seiten in der (jüngeren) Vergangenheit häufig benutzt wurden, wird angenommen, daß die Seiten auch in der (näheren) Zukunft benutzt werden.

Es wird die Seite entfernt, die am längsten unbenutzt ist.

Die Implementierung von LRU ist schwierig, da eine Liste der nach Referenzzeitpunkten sortierten Seiten benötigt wird. Bei jeder Referenz muß diese Liste aktualisiert werden, was sehr zeitaufwendig ist oder teure Spezialhardware benötigt.

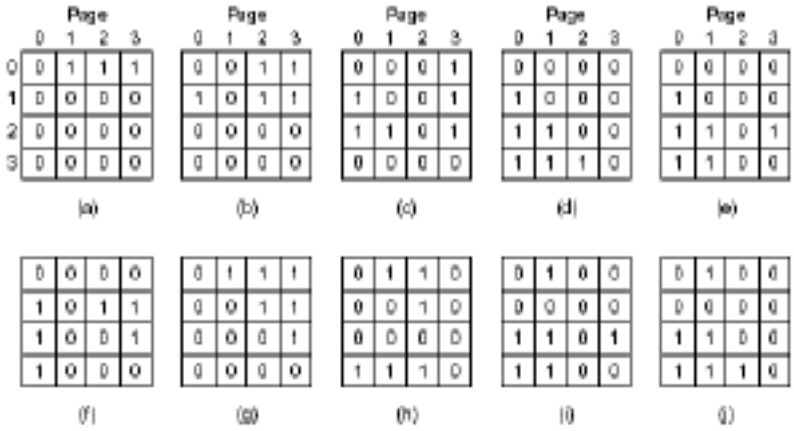


Figure 4-3. LRU using a matrix when pages are referenced in the order 0, 1, 2, 3, 2, 1, 0, 3, 2, 3.

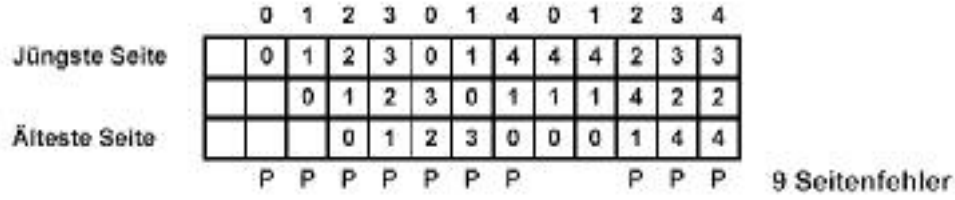
Belady´s Anomalie

Intuitiv vermutet man den Zusammenhang, daß umso weniger Seitenfehler auftreten je mehr Seitenrahmen zur Verfügung stehen.

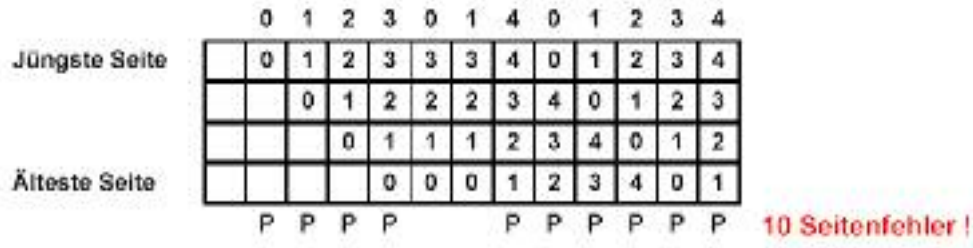
Belady hat 1969 ein Gegenbeispiel gefunden, daß mit 3 Seitenrahmen weniger Fehler erzeugt als mit vier:

- **Beispiel:**
  - **5 Seiten (0, ..., 4).**
  - **Folge der Seitenzugriffe: 0,1,2,3,0,1,4,0,1,2,3,4.**
  - **Seitenersetzungsalgorithmus: FIFO.**
  - **(a) 3 Seitenrahmen.**
  - **(b) 4 Seitenrahmen.**

(a) 3 Seitenrahmen:



(b) 4 Seitenrahmen:



Das Arbeitsbereichmodell

Unter ***Lokalität*** versteht man die Eigenschaft eines Prozesses, daß zur Ausführungszeit nicht immer auf alle Seiten zugegriffen wird, sondern gehäuft auf einige wenige. Z.B. wird bei einer Schleife der Code innerhalb der Schleife, und damit die entsprechenden Seiten, angesprochen, während der übrige Code momentan nicht im Arbeitsspeicher benötigt wird. Bei den Daten wird ebenfalls oft auf eng beieinander liegende Bereiche zugegriffen, z.B. bei Reihungen oder Verbunden.

Die Menge der Seiten, die ein Prozeß augenblicklich benutzt, wird ***Arbeitsbereich*** (working set) genannt.

Falls eine Seite in den letzten ***n*** Beobachtungszeiträumen nicht angesprochen wurde, wird sie aus dem Arbeitsbereich herausgenommen.

Das BS versucht, die Seiten des Arbeitsbereiches im Hauptspeicher zu halten.

Segmentierung

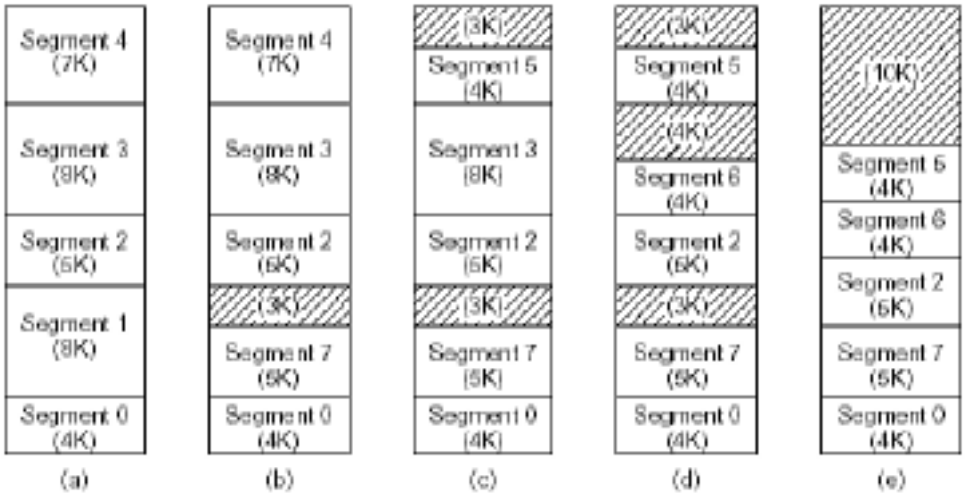
Der Speicherbedarf variiert einerseits mit den aktiven Prozessen, andererseits haben die Prozesse selbst einen variablen Platzbedarf.

Eine allgemeine Lösung ist die Einführung von unabhängigen Adreßräumen, den ***Segmenten***.

Jedes Segment besteht aus einer linearen Sequenz von Adressen, von 0 bis zum Maximum.

Die Segmentlänge kann sich während der Ausführung ändern. Da jedes Segment einen getrennten Adreßraum bildet, können unterschiedliche Segmente sich nicht gegenseitig behindern.

Durch die Segmentierung ist auch die Benutzung gemeinsamer Prozeduren und Daten, sog. *shared memory, shared libraries*, möglich. Eine Einbindung dieser gemeinsam benutzten Teile in den Adreßraum jedes Prozesses ist nicht notwendig.



Implementierung von reiner Segmentierung

Die Implementierung der Segmentierung unterscheidet sich von Paging grundlegend: Seiten haben eine feste Größe, Segmente nicht!

Zusammenfassung

Im einfachsten Fall kann das System keine Seiten ersetzen und ein Programm verbleibt nach dem Laden im Speicher bis es terminiert.

Swapping ermöglicht die Verwaltung mehrerer Prozesse, falls genügend Platz vorhanden ist.

Virtueller Adreßraum macht die Programmierung unabhängig vom tatsächlich vorhandenen physikalischen Speicher.

Der Adreßraum wird typischerweise in Einheiten gleicher Größe, den Seiten, eingeteilt. Der physikalische Speicher wird in Seitenrahmen gleicher Größe wie die Seiten eingeteilt. Die Seitentabellen beschreiben die Zuordnung von Seiten zu Seitenrahmen.

Es gibt zahlreiche Seitenersetzungsalgorithmen, jeder mit Vor- und Nachteilen.

DATEISYSTEME

Einführung

Dateien sind das zweite, wichtige Konzept eines Betriebssystems.

Probleme

Die Speicherkapazität eines Prozesses ist auf den virtuellen Adreßraum beschränkt, was für Anwendungen zu gering sein kann.

Informationen dürfen nicht verloren gehen, wenn der Rechner ausfällt oder ein Prozeß stirbt.

Mehrere Prozesse sollten gleichzeitig auf Informationen (oder Teile von diesen) zugreifen können.

Anforderungen

1. Es muß möglich sein, sehr große Mengen von Informationen zu speichern.

2. Die Informationen müssen die Terminierung der Prozesse, die diese verwenden, überleben.

3. Es muß für mehrere Prozesse möglich sein, gleichzeitig auf die Informationen zuzugreifen.

Datei

Die Informationen werden auf Platten und anderen externen Speichermedien in Einheiten, sog. *Dateien*, gespeichert.

Die in Dateien gespeicherte Informationen müssen *persistent* sein, d.h. nicht von der Erzeugung oder Terminierung von Prozessen abhängen.

Dateisystem

Dateien werden durch das *Dateisystem*, einem Teil des BS, verwaltet.

Das Dateisystem bestimmt die Struktur, Benennung, Zugriff, Benutzung, Schutz, Implementierung etc. der Dateien.

Dateien

Benennung von Dateien

Dateien sind ein Abstraktionsmechanismus.

Die genauen Regeln für die Benennung von Dateien variieren von System zu System.

Alle wichtigen BS erlauben die Verwendung von Zeichenketten von ein bis acht Buchstaben, Ziffern und einigen Sonderzeichen.

Einige Dateisysteme unterscheiden zwischen Groß- und Kleinschreibung.

Viele Dateisysteme unterstützen zwei- oder mehrteilige Dateinamen, wobei die beiden Bestandteile durch einen Punkt getrennt werden.

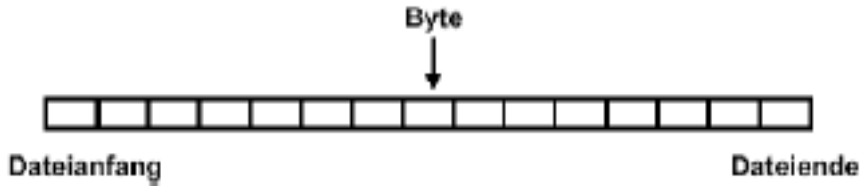
Üblicherweise wird der letzte Bestandteil des Dateinamens als *Dateinamenerweiterung* bezeichnet und macht zusätzliche Angaben über die Datei.

Dateistruktur

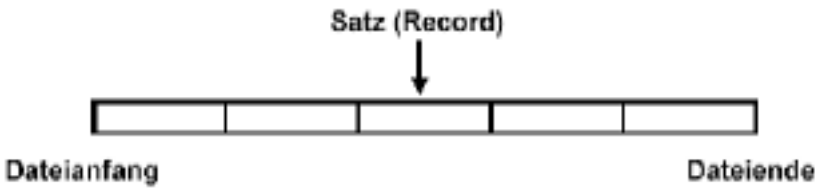
Dateien können auf verschiedene Weise strukturiert werden

- Folge von Bytes
- Folge von Datensätzen
- Baum von Datensätzen

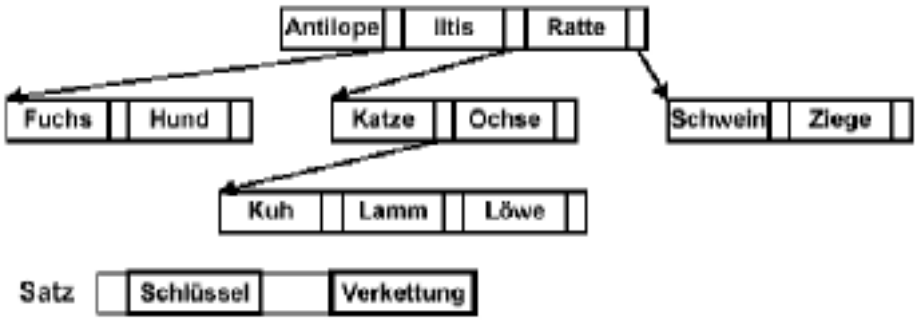
Durch Betrachtung einer Datei als Folge von Bytes wird die maximale Flexibilität erreicht, d.h. das BS gibt keine Unterstützung und steht keiner Verwendung im Weg.



Die Grundidee der Strukturierung einer Datei als Folge von Datensätzen (engl. records) besteht darin, daß eine Leseoperation einen ganzen Datensatz liest und eine Schreiboperation einen ganzen Datensatz ersetzt oder an das Ende der Datei anhängt.



Bei der baumartigen Strukturierung besteht eine Datei aus einem Baum von Datensätzen, die nicht alle dieselbe Länge haben müssen, von denen jeder ein Schlüsselfeld an einer ganz bestimmten Position im Datensatz enthält. Die Datensätze sind nach dem Schlüssel sortiert in einem Baum angeordnet.



Dateitypen

Viele BS unterstützen verschiedene Dateitypen:

- gewöhnliche Dateien,
- Verzeichnisse,
- zeichenorientierte Spezialdateien und
- blockorientierte Dateien.

Gewöhnliche Dateien sind in der Regel ASCII- oder Binärdateien.

ASCII-Dateien lassen sich besonders einfach verarbeiten, ausdrucken und zum Datenaustausch verwenden.
Binärdateien verwenden nicht nur die ASCII-Zeichen, sondern alle möglichen Bitkombinationen. Gewöhnlich haben Binärdateien eine interne Struktur.

Beispiele: Grafik-Dateien. Darstellung der Farben: RGB 255.255.255

Dateizugriff

Beim *sequentiellen Zugriff* kann ein Prozeß alle Bytes oder Sätze einer Datei beginnend am Anfang der Reihe nach lesen. Das Lesen kann beliebig oft wiederholt werden, es muß aber immer am Anfang der Datei begonnen werden.

Beim *Direktzugriff* können die Bytes und Sätze einer Datei in einer beliebigen Reihenfolge gelesen werden.

Für die Positionierung gibt es zwei Verfahren: Entweder wird die Position beim Lese/Schreibbefehl mit angegeben oder es existiert ein separater Positionierungsbefehl (z.B. seek).

Bei neueren BS werden alle Dateien als Direktzugriffsdateien organisiert, der sequentielle Zugriff ist dann nur ein Spezialfall davon.

Dateiattribute

Jede Datei hat einen Namen und ihre Daten. Zusätzlich verbinden alle BS weitere Informationen mit jeder Datei, die *Dateiattribute*.

Mögliche Dateiattribute:

Feld	Bedeutung
Schutz	Wer darf wie auf die Datei zugreifen
Paßwort	Paßwort, das für den Zugriff benötigt wird
Erzeuger	Identifikation der Erzeugers der Datei
Eigentümer	Aktueller Eigentümer der Datei
Read-Only-Flag	0 für Lesen/Schreiben, 1 für nur Lesen
Versteckt-Flag	0 für normal, 1 für versteckt
System-Flag	0 für normale Dateien, 1 für Systemdateien
Archiv-Flag	0 für Sicherungskopie erstellt, 1 für Sicherung muß gemacht werden
ASCII/Binär-Flag	0 für ASCII-Datei, 1 für Binärdatei
Flag für wahlfreien Zugriff	0 für nur sequentiellen Zugriff, 1 für wahlfreien Zugriff
Tempörär-Flag	0 für normal, 1 für löschen bei Prozeßende
Sperre-Flag	0 für nicht gesperrt, 1 für gesperrt
Satzlänge	Anzahl der Bytes in einem Satz
Schlüsselposition	Offset des Schlüssels im Satz
Schlüssellänge	Anzahl der Bytes des Schlüsselfeldes
Erzeugungsdatum	Datum und Uhrzeit der Datelerzeugung
Zeit des letzten Zugriffs	Datum und Uhrzeit des letzten Zugriffs
Zeit der letzten Änderung	Datum und Uhrzeit der letzten Änderung
Aktuelle Größe	Anzahl der Bytes der Datei
Maximalgröße	Größe, bis zu der die Datei maximal wachsen darf

Dateioperationen

CREATE Eine Datei wird ohne Daten erzeugt. Damit wird angekündigt, daß ein Dateiinhalt entstehen wird, und es werden einige Attribute festgelegt.

DELETE Wenn die Datei nicht mehr benötigt wird, muß sie gelöscht werden, um den Plattenplatz freizugeben. Zu diesem Zweck gibt es immer einen Systemaufruf.

OPEN Bevor eine Datei benutzt werden kann, muß sie geöffnet werden. Der Zweck des Systemaufrufs OPEN besteht darin, durch das System die Attribute und die Liste der Plattenadressen in den Arbeitsspeicher zu holen, um in nachfolgenden Aufrufen schnell auf die Datei zugreifen zu können.

CLOSE Wenn alle Zugriffe beendet worden sind, werden die Attribute und die Plattenadressen nicht länger benötigt, so daß die Datei geschlossen werden kann, um den in den internen Tabellen belegten Platz freizugeben. Dies wird in vielen Systemen dadurch gefördert, dass den Prozessen eine obere Schranke für die Anzahl der geöffneten Dateien auferlegt wird.

READ Aus einer Datei werden Daten gelesen, normalerweise von der aktuellen Position. Der Aufrufer muß angeben, wieviele Daten benötigt werden, und er muß einen Puffer zur Zwischenspeicherung bereithalten.

WRITE Daten werden in eine Datei geschrieben, wiederum in der Regel an die aktuelle Position. Wenn die aktuelle Position das Ende der Datei ist, steigt die Größe der Datei. Wenn die aktuelle Position in der Mitte der Datei liegt, werden die bestehenden Daten überschrieben und gehen für immer verloren.

APPEND Dieser Aufruf stellt eine abgespeckte Version des WRITE dar. Es können Daten lediglich am Ende einer Datei angefügt werden. In den Systemen, die nur eine minimale Anzahl von Systemaufrufen bereitstellen, gibt es diesen Aufruf normalerweise nicht.

SEEK In Direktzugriffsdateien ist es erforderlich, die Position anzugeben, von der die Daten genommen werden sollen. Ein gebräuchlicher Ansatz besteht in dem Systemaufruf SEEK, durch den ein Zeiger auf die aktuelle Position auf eine spezielle Stelle in der Datei gesetzt wird. Nachdem dieser Aufruf beendet ist, können Daten von dieser Position gelesen oder geschrieben werden.

GET ATTRIBUTES Die Prozesse brauchen für ihre Arbeit die Dateiattribute. Dieser Systemaufruf liefert die Attribute einer Datei.

SET ATTRIBUTES Einige der Attribute können durch den Benutzer festgelegt und nach der Erzeugung einer Datei verändert werden, was durch diesen Systemaufruf ermöglicht wird.

RENAME Es geschieht häufig, daß ein Benutzer den Namen einer existierenden Datei ändern möchte, was durch diesen Aufruf ermöglicht wird.

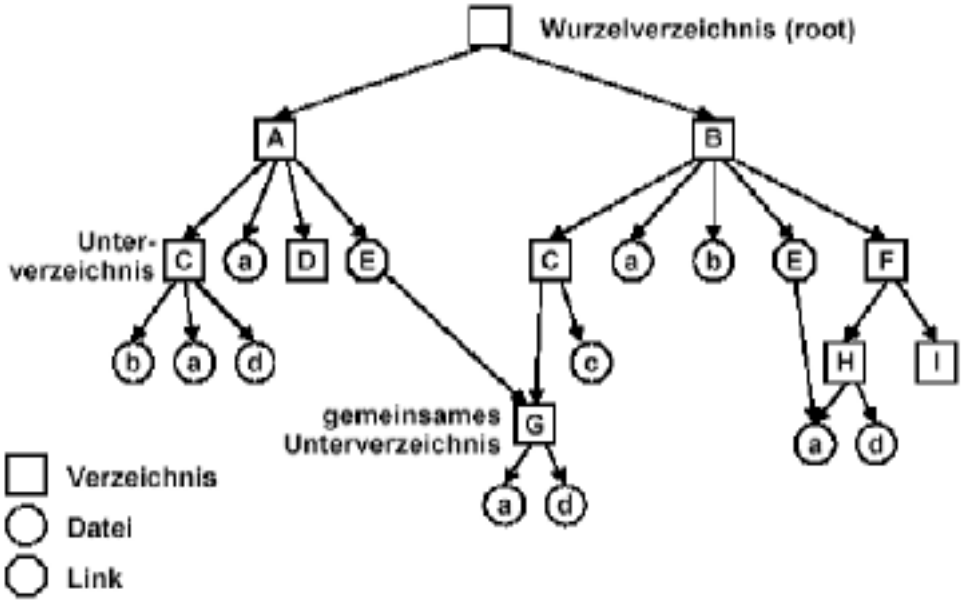
Verzeichnisse

Hierarchische Verzeichnissysteme

Ein Verzeichnis enthält typischerweise einen Eintrag pro Datei.

Wenn eine Datei geöffnet werden soll, durchsucht das BS das Verzeichnis, bis es den Namen der zu öffnenden Datei findet. Dann extrahiert es die Attribute und die Plattenadresse, entweder direkt aus dem Verzeichniseintrag oder aus der referenzierten Datenstruktur, und legt sie in einer Tabelle im Hauptspeicher ab. Alle weiteren Referenzen auf diese Datei benutzen diese Informationen in dem Hauptspeicher.

Die Anzahl der Verzeichnisse variiert von System zu System.



Die Verzeichnisse werden hierarchisch angeordnet.

Pfadnamen

Wenn das Dateisystem als ein Verzeichnisbaum organisiert wird, wird ein Weg benötigt die Dateinamen zu spezifizieren. Es werden gewöhnlich zwei verschiedene Methoden benutzt:

1. Jeder Datei wird ein *absoluter Dateiname* zugeordnet, bestehend aus dem Pfad vom Wurzelverzeichnis zur Datei. Absolute Dateinamen beginnen immer im Wurzelverzeichnis und sind immer eindeutig.

2. *Relative Pfadnamen* basieren auf dem Konzept des *Arbeitsverzeichnisses* (auch „aktuelles Verzeichnis“ genannt). Ein Benutzer kann ein Verzeichnis als Arbeitsverzeichnis auszeichnen, in diesem Fall werden alle Pfadnamen, die nicht im Wurzelverzeichnis beginnen, relativ zu dem Arbeitsverzeichnis benutzt.

Die meisten BS, die ein hierarchisches Dateisystem unterstützen, haben zwei besondere Einträge in jedem Verzeichnis, **. und ..**
(ausgesprochen „Punkt“ und „Punkt Punkt“).
Punkt referenziert auf das aktuelle Verzeichnis, Punkt Punkt referenziert auf das Elternverzeichnis.

Verzeichnisoperationen

CREATE Ein Verzeichnis wird erzeugt. Es ist außer Punkt und Punkt Punkt leer, welche automatisch vom System erstellt werden.

DELETE Ein Verzeichnis wird gelöscht. Nur ein leeres Verzeichnis kann gelöscht werden. Ein Verzeichnis, das nur noch Punkt und Punkt Punkt enthält, gilt als leeres Verzeichnis, da diese Dateien normalerweise nicht gelöscht werden können.

OPENDIR Ein Verzeichnis muß geöffnet werden, bevor es gelesen werden kann.

CLOSEDIR Nachdem ein Verzeichnis gelesen wurde, sollte es geschlossen werden, um Tabellenplatz freizugeben.

READDIR Dieser Aufruf holt den nächsten Eintrag aus einem geöffneten Verzeichnis.

LINK Linking ist eine Technik, die es einer Datei erlaubt, in mehr als einem Verzeichnis zu erscheinen. Dieser Systemaufruf spezifiziert eine existierende Datei und einen Pfadnamen und erzeugt einen Link von einer existierenden Datei auf den Namen, der durch den Pfad angegeben wird. Auf diese Art und Weise erscheint dieselbe Datei in mehreren Verzeichnissen.

UNLINK Ein Verzeichniseintrag wird entfernt. Falls die zu entfernende Datei nur in einem Verzeichnis vorhanden ist (der normale Fall), wird sie aus dem Dateisystem entfernt. Falls sie in mehreren Verzeichnissen erscheint, wird nur der angegebene Pfadname gelöscht. Die anderen bleiben erhalten.

Dateisystemimplementierungen

Die Implementierung von Dateien

Kontinuierliche Allokation

Jede Datei wird als kontinuierlicher Block von Daten auf der Platte gespeichert.

Dieses Verfahren ist einfach zu implementieren, da die Verwaltung der Dateiblöcke auf die Speicherung einer einzelnen Nummer reduziert wird, der Plattenadresse des ersten Blocks.

Der Durchsatz ist ausgezeichnet, da die gesamte Datei in einer einzigen Operation von der Platte gelesen werden kann.

Die Allokation ist nicht durchführbar, wenn die maximale Größe der Datei zum Zeitpunkt der Erzeugung bekannt ist.

Die Fragmentierung der Platte verschwendet Platz, der anderweitig genutzt werden könnte. Die Verdichtung ist gewöhnlich unerschwinglich teuer.

Allokation mittels einer verknüpften Liste

Wird das erste Wort in jedem Block als Zeiger auf den nächsten Block genutzt (der Rest des Blocks enthält die Daten), kann die Datei als verkettete Liste verwaltet werden.

Jeder Plattenblock kann benutzt werden; es geht kein Platz durch Fragmentierung verloren.

In der Liste wird die Plattenadresse des ersten Blocks gespeichert, der Rest kann dann ab der Startposition gefunden werden.

Das sequentielle Lesen ist einfach, der wahlfreie Zugriff extrem langsam.

Durch den Speicherverbrauch des Zeigers ist der für die Daten zur Verfügung stehende Platz von der Größe her keine Zweierpotenz mehr. Der Einsatz dieser Größe ist wenig effizient, da praktisch alle Programme in Blöcken lesen und schreiben, deren Größe eine Zweierpotenz ist.

Allokation mittels einer verknüpften Liste und Indexeinsatz

Beide Nachteile einer Allokation mittels einer verknüpften Liste können durch die Speicherung der Zeigerworte jedes Plattenblocks in einer Tabelle oder einem Index im Speicher eliminiert werden.

Beispiel: MS-DOS FAT-Filesystem

Durch die Verwendung dieser Organisation ist der gesamte Block für Daten verfügbar.

Der wahlfreie Zugriff ist sehr einfach. Die zu durchlaufende Kette kann komplett im Hauptspeicher gehalten werden.

Die gesamte Tabelle muß während der ganzen Arbeitszeit im Speicher gehalten werden.

I-Nodes

Zu jeder Datei wird ein I-Node eingerichtet.

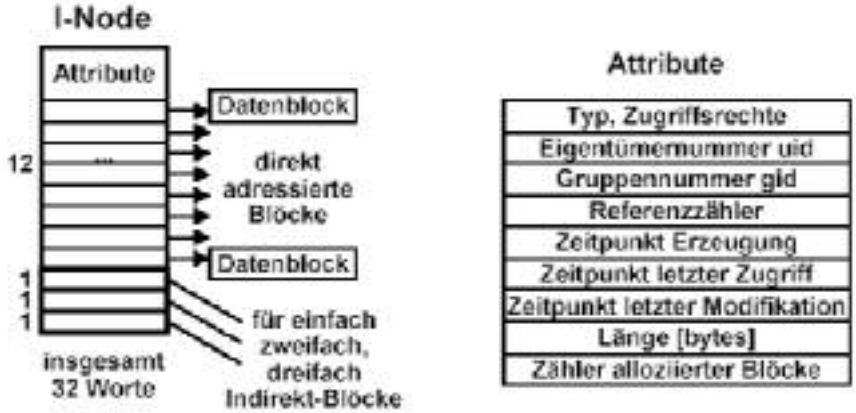
I-Nodes (Index-Knoten) sind kleine Tabellen, welche die Attribute und die Plattenadressen der Dateiblöcke enthalten.

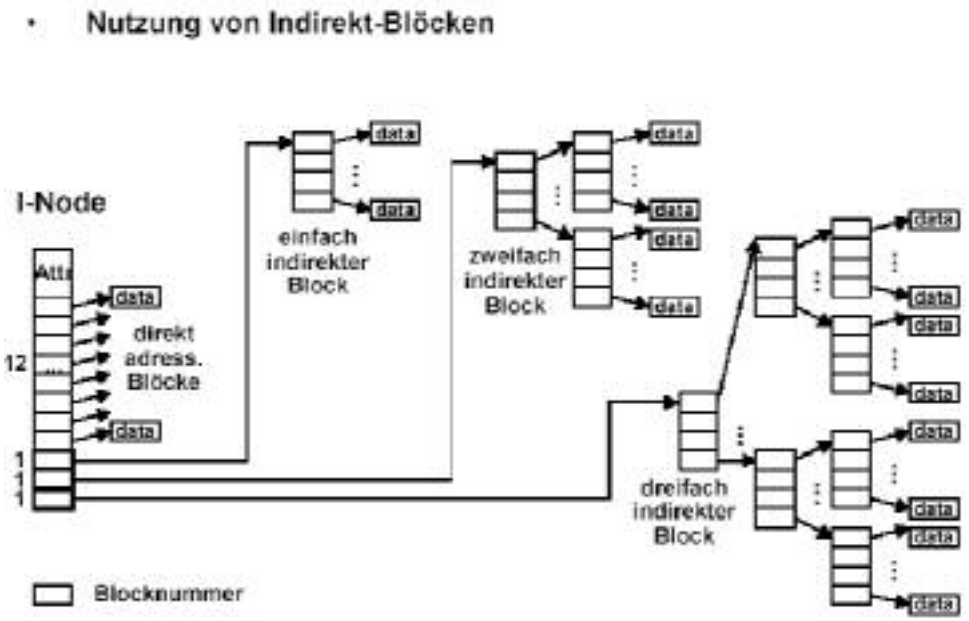
Die ersten Plattenadressen werden im I-Node selbst gespeichert, so daß für kleine Dateien die notwendige Information im I-Node selber steckt, welche durch das Öffnen der Datei von der Platte in den Hauptspeicher geholt wird.

Bei größeren Dateien ist eine der Adressen in dem I-Node die Adresse eines Plattenblockes, der auch *einfacher indirekter Block* genannt wird. Dieser Block enthält weitere Plattenadressen.

Falls benötigt, werden weiter indirekte Blöcke eingesetzt.

- **Beispiel: BSD UNIX Fast File System**





Die Implementierung von Verzeichnissen

Bevor eine Datei gelesen werden kann, muß die Datei geöffnet werden. Das BS benutzt den Pfadnamen, um den Verzeichniseintrag zu lokalisieren, der dann die notwendigen Informationen zum Auffinden der Plattenblöcke liefert.

Abhängig vom System kann diese Information

- die Plattenadresse der gesamten Datei (kontinuierliche Allokation),
- die Nummer des ersten Blocks (beide Schemata der verknüpften Liste) oder
- die Nummer des I-Nodes sein.

Die Hauptaufgabe des Verzeichnissystems ist die Lokalisierung der Daten aufgrund des Dateinamens.

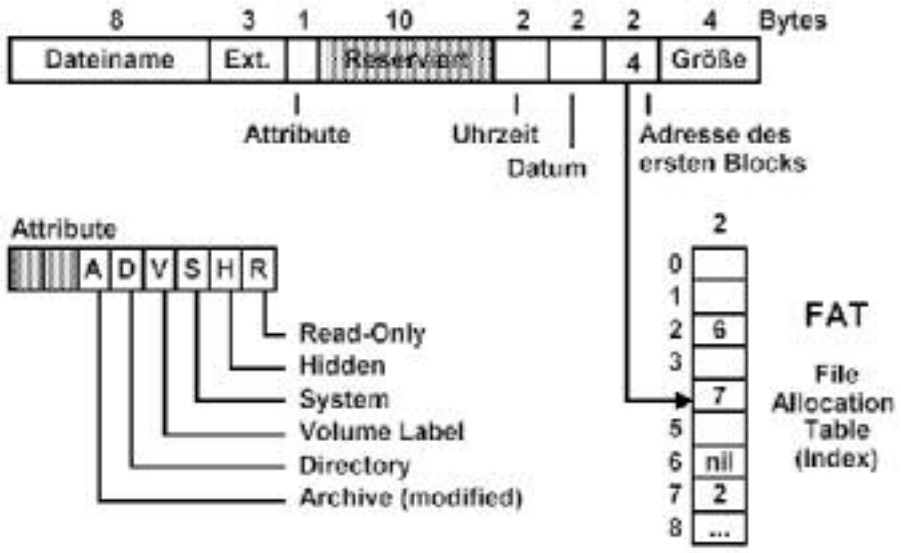
Die Attribute können

- im Verzeichniseintrag oder
- im I-Node gespeichert werden.

Verzeichnisse unter MS-DOS

- Hierarchischer Verzeichnisbaum
- Ein Eintrag ist 32 Bytes lang
- Enthalten sind
    - Dateiname
    - Attribute
    - Nummer des ersten Plattenblocks

- **Verzeichniseintrag:**



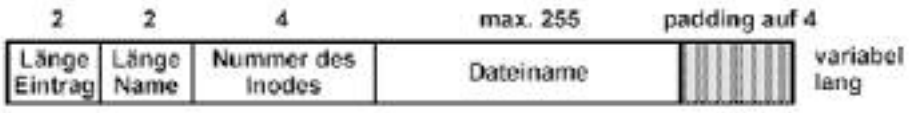
Die erste Plattenblocknummer wird als Index auf eine Tabelle vom Typ „Verknüpfte Liste“ genutzt, mit Durchlaufen der Kette können alle Blöcke gefunden werden.
Verzeichnisse können geschachtelt werden.

Verzeichnisse unter Unix

Jeder Eintrag enthält nur:

- Dateiname und
- I-Node-Nummer

- **Verzeichniseintrag BSD UNIX Fast File System (ufs):**



Alle weiteren Informationen werden im I-Node gehalten:

- Typ,
- Größe,
- Zeiten,
- Benutzerzugehörigkeit und
- Plattenblöcke

Wenn eine Datei geöffnet wird, erkennt das Dateisystem den angegebenen Dateinamen und lokalisiert die Plattenblöcke:

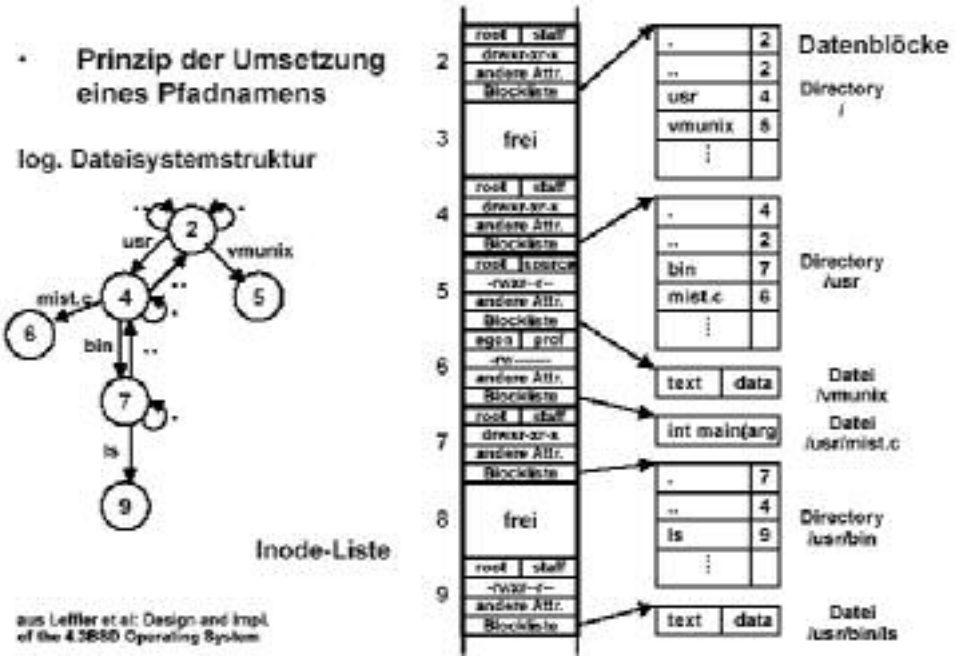
1. Das Wurzelverzeichnis wird lokalisiert. Dieser I-Node liegt auf einer fixierten Stelle auf der Platte.

2. Dann wird die erste Komponente des Pfades innerhalb des Wurzelverzeichnisses gesucht.

3. I-Nodes können einfach lokalisiert werden, da ihre Nummer auf einen fixierten Platz auf der Platte verweist.

4. Von diesem I-Node ausgehend, lokalisiert das System das Verzeichnis für die erste Komponente und sucht die nächste Komponente in diesem Verzeichnis. usw.

Relative Pfadnamen werden in der gleichen Art und Weise gesucht wie absolute, es wird nur vom Arbeitsverzeichnis aus gestartet anstatt vom Wurzelverzeichnis.



Gemeinsam benutzte Dateien

Die Nutzung gemeinsam benutzter Dateien wird bequemer, wenn Verweise auf die Datei in mehreren Verzeichnissen enthalten sind.

Die Verbindung zwischen dem Verzeichnis und der gemeinsam benutzten Datei wird *Link* genannt. Das Dateisystem selber ist nun ein *gerichteter azyklischer Graph* (engl. directed acyclic graph DAG) und kein Baum mehr.

Die gemeinsame Benutzung der Dateien führt zu einigen Problemen:

- Enthält ein Verzeichnis die Plattenadressen, wird bei einem Link eine Kopie der Plattenadresse angelegt. Damit werden Änderungen nur in dem Verzeichnis wirksam, auf das der Benutzer zugreift, womit ein gemeinsame Nutzung nicht mehr möglich ist.

- Das Problem wird gelöst, indem man entweder die Plattenblöcke nicht in den Verzeichnissen einträgt, sondern in einer separaten Datenstruktur, z.B. den I-Nodes (Unix-Lösung), oder die Verbindung wird mittels einer vom System erzeugten Datei vom Typ LINK erzeugt, die nur den Pfadname der Datei enthält. Dieser Weg wird *symbolisches Linken* genannt.

- Die erste Methode hat den Nachteil, daß Links auf gelöschte Dateien entstehen können, wenn die Datei vom Eigentümer gelöscht wird.

- Die zweite Methode hat den Nachteil des benötigten Speicherplatzes für die I-Nodes.

Bei Verwendung von Links haben Dateien zwei oder mehr Pfade die zu ihnen führen, d.h. Programme, die Daten in einem Verzeichnis starten und in den Unterverzeichnissen suchen, werden eine Datei mehrfach lokalisieren.

Plattenplatzmanagement

Zwei generelle Strategien zu Speicherung von *n* Bytes:

1. *n* zusammenhängende Bytes der Platte werden allokiert oder

2. die Datei wird in eine Anzahl von (nicht notwendigerweise) gleichgroßen Blöcken aufgeteilt.

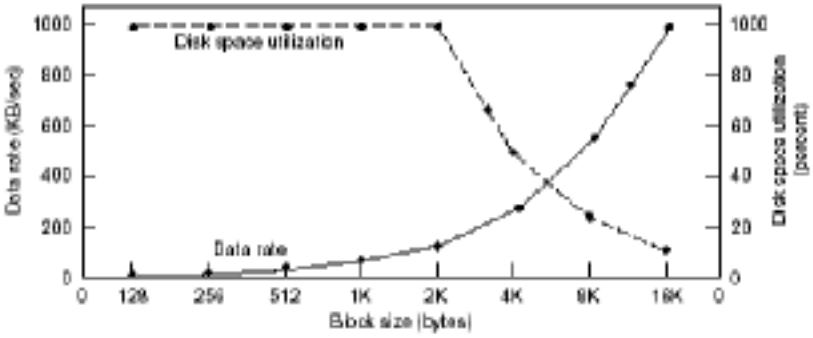
Problem: Bei der ersten Methode tritt ein Problem auf, wenn die Datei wächst und sie deswegen auf der Platte verlagert werden muß.

Blockgrößen

Untersuchungen zeigen, daß die durchschnittliche Dateigröße in UNIX-Umgebungen ca. 1K beträgt.

Das Einlesen jeder Allokationseinheit benötigt eine Suche und eine Rotationsverzögerung, so daß das Lesen umso langsamer wird, je kleiner die Allokationseinheiten sind.

Diagramm Plattenplatzeffizienz

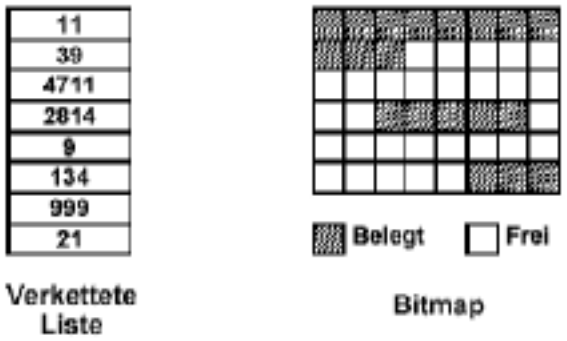


Der normale Kompromiß ist die Wahl einer Blockgröße von 512, 1K oder 2K Bytes.

Verwaltung der freien Blöcken

Zwei Standard-Methoden:

1. Verkettete Liste von Plattenblöcken, wobei jeder Block so viele freie Plattenblocknummern enthält wie möglich.
2. Bitmap zur Verwaltung der freien Blöcke



Falls genügend Hauptspeicher vorhanden ist, um die Bitmap im Hauptspeicher zu halten, ist diese Methode vorzuziehen.

Plattenquantum (Quota)

Der Systemadministrator kann jedem Benutzer eine Obergrenze für den von ihm belegten Plattenplatz und die Anzahl der Dateien zuweisen.

Der Platzbedarf einer Datei wird dem Eigentümer über die Dateiattribute seinem Quantum belastet, ebenso wird die Anzahl der Dateien verwaltet.

Eine Überschreitung des Quantums quittiert das System mit der Weigerung weitere Dateien anzulegen oder zusätzlichen Plattenplatz zu belegen.

Es gibt Systeme, die zwischen *harten* und *weichen* Grenzen unterscheiden.

Die Zuverlässigkeit des Dateisystems

Die Zerstörung des Dateisystems hat meistens katastrophale Folgen.

Es sind Maßnahmen möglich, die das Ausmass der unwiderruflichen Zerstörung des Dateisystems auf ein möglichst geringes Maß reduzieren.

Maßnahmen:

Fehlerblockverwaltung

- Platten haben oft fehlerhafte Blöcke
- Hardwarelösung: Auszeichnung eines Blocks als defekt, der Controller verwendet solche Blöcke nicht
- Softwarelösung: Im Dateisystem wird eine Datei der fehlerhaften Blöcke geführt, so daß sie nicht mehr für Datenblöcke verwendet werden können.

Backups

- Bei kleinen Datenmengen reichen die üblichen Backup-Medien i.d.R. aus.
- Bei großen Datenmengen, d.h. ab ca. 1 GB, ist die Sicherung der gesamten Datenmenge schwierig und zeitaufwendig.
- *Inkrementelle Datensicherung* reduziert den Aufwand.
    - Die einfachste Form ist die periodische Durchführung einer kompletten Sicherung und die tägliche Sicherung der Dateien, die sich seit der letzten kompletten Sicherung geändert haben.
    - Dazu muß eine Liste der Sicherungsdaten für jede Datei auf der Platte verwaltet werden.

Dateisystemkonsistenz

- Es können zwei Arten von Konsistenzchecks durchgeführt werden:
  1. Blöcke und
  2. Dateien.
- Um die Blockkonsistenz zu überprüfen, bildet das Programm eine Tabelle mit zwei Zählern pro Block, beide werden mit 0 initialisiert. Der erste Zähler verwaltet, wie oft ein Block in einer Datei präsent ist; der zweite Zähler zählt, wie oft der Block in der Freiliste (bez. in der Bitmap der freien Blöcke) vorhanden ist.

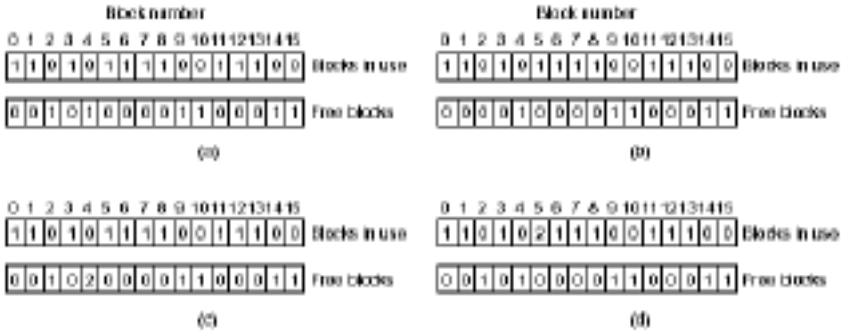
- Das Programm liest dann alle I-Nodes. Beginnend bei einem I-Node ist es möglich, eine Liste von allen Blocknummern, die in der entsprechenden Datei verwendet werden, aufzustellen. Sobald jede Blocknummer gelesen wurde, wird der Zähler in der ersten Tabelle inkrementiert. Das Programm überprüft dann die Freiliste oder die Bitmap, um alle Blöcke zu finden, die nicht verwendet werden. Jedes Auftreten eines Blockes in der Freiliste bewirkt eine Inkrementierung des Zählers in der zweiten Tabelle.

- Wenn das Dateisystem konsistent ist, besitzt jeder Block entweder eine 1 in der ersten oder in der zweiten Tabelle.

- Fehlerbehandlung:
  - **Verlorene Blöcke** Verlorene Blöcke werden der Freiliste zugeordnet.

  - **Duplikate in der Freiliste** Ein Eintrag wird aus der Freiliste entfernt.

  - **Duplikate belegter Blöcke** Dies ist der schlimmste Fall, da Blöcke in zwei oder mehreren Dateien präsent sind. Einträge von Blocks in der Freiliste werden gelöscht. Für jedes Duplikat werden freie Blöcke allokiert, der Inhalt dorthin kopiert und den betroffenen Dateien zugeordnet. Der Inhalt der Blöcke wird voraussichtlich unbrauchbar sein.



- Die Verzeichnisstruktur wird ähnlich überprüft.

Durchsatz

Ein Plattenzugriff ist wesentlich langsamer als ein Speicherzugriff (ca. 100.000 mal langsamer).

Deswegen werden viele Dateisysteme so konstruiert, daß die Anzahl der Plattenzugriffe auf ein Minimum reduziert werden.

Die am häufigsten angewandte Technik ist der *Blockcache* oder *Puffercache*.

- Cache ist eine Sammlung von Blöcken, die auf die Platte gehören, aber aus Gründen der Performance im Speicher gehalten werden.
- Der gebräuchlichste Algorithmus zur Verwaltung des Cache ist, alle Leseanfragen zu überprüfen, ob der benötigte Block im Cache ist:
- Ist der Block im Cache, kann die Leseanfrage ohne Plattenzugriff beantwortet werden.
- Ist der Block nicht im Cache, so wird er von der Platte in den Cache und dann in den Arbeitsspeicher kopiert.
- Falls ein Block in einen vollen Cache geladen werden muß, muß ein Block auf die Platte geschrieben werden, falls er nach der Einlagerung verändert wurde, und der Platz im Cache wird freigegeben.
- Für die Auswahl des Blockes sind Strategien wie bei der Seitenersetzung anwendbar:
  - FIFO
  - Second-Chance
  - LRU
  - etc.
- Um das Dateisystem möglichst intakt zu halten, ist es wenig sinnvoll, die Datenblöcke lange im Cache zu halten, bevor sie wieder zurückgeschrieben werden. Deshalb ist reines LRU wenig erstrebenswert.
- UNIX erlaubt mit den Befehl *sync* das sofortige Schreiben aller modifizierten Blöcke auf die Platte (sog. *Nonwrite-Through-Cache*).
- MS-DOS schreibt modifizierte Blöcke zu bald wie möglich auf die Platte (sog. *Write-Through-Cache*)
- Der Vorteil der MS-DOS-Lösung ist, daß bei Entfernen eines Datenträgers ohne vorherige Abmeldung (un-mount) keine Daten verloren gehen und auch das Dateisystem intakt bleibt.
- Der Vorteil der UNIX-Lösung ist die wesentlich bessere Effizienz.

Neben dem Caching ist die Reduzierung der Plattenarmbewegungen eine weitere wichtige Technik. Dabei werden die zu allokierenden Blöcke auf der Platte möglichst nah an dem vorhergehenden Block zu platzieren.

Sicherheit

Vorbemerkung

Dateisysteme enthalten häufig Informationen, die für ihre Benutzer von beträchtlichem Wert sind.

Der Schutz der Informationen gegenüber einer nicht authorisierten Benutzung ist ein wichtiger Aspekt in allen Dateisystemen.

Sicherheitsumgebung

Sicherheit bezeichnet die technischen, organisatorischen, juristischen und politischen Aspekte um Daten derart zu sichern, daß sie nicht von nicht authorisierten Personen gelesen oder verändert werden können.

Schutzmechanismen bezeichnet die speziellen Mechanismen eines Betriebssystems, die dem Schutz von Informationen im Rechner dienen.

Ursachen für Datenverlust:

- Naturkatastrophen und Kriege,
- Hard- und Softwarefehler,
- Menschliches Fehlverhalten,
- etc.

Eindringlinge:

- Gelegentliches Ausspionieren durch nicht autorisierte Benutzer,
- Herumschnüffeln von Insidern,
- Gezielter Versuch, Geld zu verdienen,
- Kommerzielle oder militärische Spionage.

Schutz der Privatspähre

Berühmte Sicherheitsmängel

lpr Option zum Löschen von Dateien beim Druckkommando lpr

link Link core auf passwd und Dump

mkdir Ausnutzen der Systemgeschwindigkeit um Eigentümer einer Datei zu werden

Trojanische Pferde Veränderung eines Programms, z.B. Editor, so daß neben der ursprünglichen Funktionalität zusätzliche Funktionen zur Umgehung der Systemsicherheit vorhanden sind, z.B. Dateien stehlen oder kopieren.

Paging TENEX erlaubte den Aufruf von bestimmten Programmen bei einem Seitenfehler, dadurch konnten Passwörter zeichenweise entschlüsselt werden, in dem die Paßwörter auf zwei Seiten verteilt wurden.

Internet-Wurm

Am 2.11.88 wurde von Robert Tappan Morris, Student an der Cornell Universität, ein Wurmprogramm in das Internet entlassen.

Durch diese Aktion wurden tausende von Computern in Universitäten, Unternehmen und staatlichen Forschungseinrichtungen auf der ganzen Welt lahmgelegt, bevor das Programm ausfindig und unschädlich gemacht werden konnte.

Morris hatte zwei Fehler in UNIX entdeckt, durch die es möglich war, nicht authorisierten Zugriff zu Maschinen im gesamten Internet zu bekommen.

Er schrieb ein sich selbst replizierendes Programm, einen *Wurm*, das diese Fehler ausnutzte und sich innerhalb von Sekunden auf alle erreichbaren Rechner ausbreitete. Er arbeitete Monate an diesem Programm und verbesserte es, damit dessen Attacken möglichst versteckt blieben.

Es ist nicht bekannt, ob die Version vom 2.11.88 nur ein Test oder das reale Experiment war.

Wenige Stunden nach der Freigabe waren die meisten SUN und DEC Rechner im Internet lahmgelegt.

Technisch betrachtet bestand der Wurm aus zwei Programmen, einem Bootstrap und dem eigentlichen Wurm:

Das Bootstrap-Programm bestand aus 99 C-Programmzeilen, genannt l1.c. Es wurde auf dem angegriffenen System compiliert und ausgeführt. Wenn es dann lief, stellte es eine Verbindung zu dem System her, von dem es kam, lud von diesem das eigentliche Wurmprogramm und führte es aus.

Nachdem der Wurm sich getarnt hatte, durchsuchte der Wurm auf dem neuen Rechner die Routingtabellen, um zu sehen, mit welchen Rechnern dieser verbunden war, um dann den Versuch zu unternehmen, das Bootstrap-Programm auf diese Maschinen weiterzuverbreiten.

Drei Methoden dienten dazu, neue Rechner zu infizieren:

1. Eine entfernte Shell wurde mit dem Befehl **rsh** ausgeführt. Einige Rechner führten die rsh ohne weitere Authentifikation aus und luden das Wurmprogramm.

2. Das Programm **finger** liefert Informationen über einen Benutzer an eine bestimmten Maschine. Das Wurmprogramm rief nun das Programm finger mit einer speziell codierten Zeichenkette der Länge 536 Bytes auf, was den Puffer des Programm überlaufen lies und den Stack überschrieb. Nach Rückkehr ins Hauptprogramm wurden die Anweisungen auf den Stack ausgeführt, um eine Shell auszuführen. Wenn das erfolgreich war, dann stand dem Wurm eine Shell auf dem attackierten System zur Verfügung.

3. Durch einen Fehler im **sendmail** Programm war es möglich, eine Kopie des Bootstrapprogramms zu versenden und auszuführen.

Einmal etabliert, versuchte der Wurm Benutzerpassworte zu knacken. Jedes geknackte Passwort erlaubte es dem Wurm, Zugang zu all den Maschinen zu bekommen, zu denen der Eigentümer des Paßwortes eine Zugangsberechtigung hatte.

Wurde Zugang zu einer Maschine erhalten, auf der der Wurm bereits vorhanden war, wurde in 1/7 der Fälle weiter infiziert und sonst abgebrochen. Dies führt nach kurzer Zeit zu einen Überlastung der Maschinen. Ohne diese Re-Infizierung wäre der Wurm wahrscheinlich unentdeckt geblieben.

Morris wurde gefaßt, als einer seiner Freunde mit dem Reporter John Markoff, New York Times, sprach, und versehentlich den Login-Namen rtm nannte.

Morris wurde festgenommen und von einem Gericht zu einer Strafe von 10.000 $, 3 Jahren Haft und 400 Stunden gemeinnütziger Arbeit verurteilt. Seine Justizkosten überstiegen vermutlich 150.000 $. Das Urteil wurde sehr kontrovers diskutiert.

Generische Sicherheitsattacken

- Fordere Speicherseiten, Plattenplatz, oder Medien an und lies diese einfach. Viele BS löschen nicht die Inhalte, bevor diese Betriebsmittel zugeteilt werden.

- Versuche illegale Systemaufrufe, legale Systemaufrufe mit illegalen Parametern oder legale Systemaufrufe mit legalen aber unvernünftigen Parametern. Viele System können so verwirrt werden.

- Beginne mit der Anmeldung beim System und drücke auf halbem Weg durch den Dialog die DEL-, RUBOUT- oder BREAK-Taste. In einigen Systemen wird dadurch das Programm zur Überprüfung der Paßwörter abgebrochen und der Zugang als erfolgreich angesehen.

- Versuche komplexe Betriebssystemstrukturen, die im Benutzeradressraum verwaltet werden, zu modifizieren, um Schaden in Bezug auf die Sicherheit anzurichten.

- Verwirre den Benutzer durch Bildschirmausgaben wie *login:* um die Eingabe von Login-Namen und Paßwörter zu provozieren.

- Durchsuche die Handbücher nach Einträge wie *„Tu nicht X“* und versuche so viele Variante von *X* wie möglich.

- Überzeuge einen Systemprogrammierer, das System so zu verändern, daß es bestimmte lebenswichtige Sicherheitsüberprüfungen für jeden Benutzer für Prozesse mit Deiner Benutzerkennung übergeht (*Falltür* oder *trap door*).

- Versuche das Personal zu überlisten oder zu bestechen, um an Informationen zu kommen.

Viren

Ein *Virus* ist ein Programmstück, das sich an ein gültiges Programmstück anfügt, mit der Absicht, andere Programme zu infizieren.

Typische Arbeitsweise:

Zunächst wird ein nützliches neues Programm geschrieben, das versteckt den Viruscode enthält, und verbreitet.

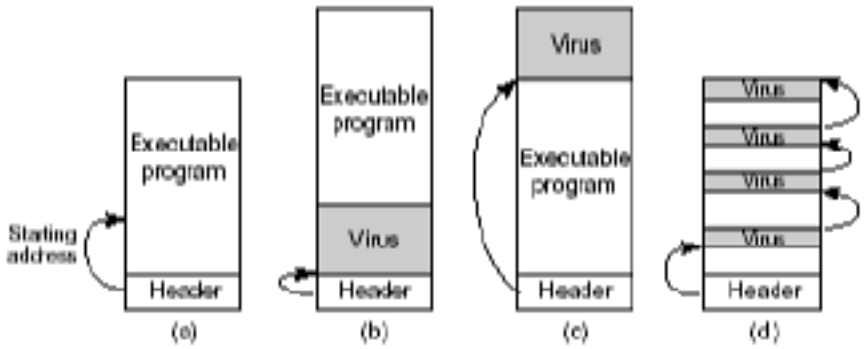
Wird das Programm gestartet, so beginnt es unmittelbar damit, alle Binärprogramme, die sich auf der Festplatte befinden, daraufhin zu untersuchen, ob sie bereits infiziert sind. Wird ein noch nicht infiziertes Programm gefunden, wird der Virus-Code am Ende des Programme angehängt und ein Sprungbefehl zum Beginn des Virus-Codes eingefügt. Nach der Terminierung des Virus-Codes springt dieser zurück an die zuvor erste Anweisung. Bei jeder Ausführung des infizierten Programms wird damit versucht, weitere Programme zu infizieren.

Zusätzlich zu der Infektion anderer Programme kann der Virus weitere Dinge tun, z.B. Daten löschen, Hardware beschädigen oder den Bootvorgang des Systems sabotieren.

Die Infektion durch Viren ist leichter zu verhindern als die Infektion zu heilen. Es gibt kommerzielle Anti-Viren-Programme.

Ein genereller Ansatz besteht darin, erst die gesamte Festplatte unter Einbeziehung des Bootsektors neu zu formatieren. Daran anschließend kann sämtliche vertrauenswürdige Software geladen werden und die Prüfsumme für jede Datei gebildet werden, mit deren Hilfe dann später Abweichungen erkannt werden können. Damit werden Infektionen nicht verhindert, aber schnell erkannt.

Infektionen können dadurch erschwert werden, daß das Verzeichnis, in dem sich die Binärprogramme befinden, für normale Benutzer nicht mehr zum schreibenden Zugriff zur Verfügung steht.



Entwurfsprinzipien für die Sicherheit

- Der Entwurf für ein System sollte öffentlich sein. Geheimhaltung dient nur der Täuschung der Entwickler und Betreiber.
- Zugriffe sollten standardmäßig abgelehnt werden.
- Zugriffsrechte sollten bei jedem Zugriff überprüft werden und nicht nur einmal pro Sitzung oder Operationstyp.
- Man sollte jedem Prozeß sowenig Zugriffsrechte einräumen wie möglich (least priviledgeed principle).
- Der Schutzmechanismus sollte einfach, einheitlich und in den untersten Schichten des BS verankert sein und nicht nachträglich integriert werden.
- Das gewählte Schema muß psychologisch akzeptabel sein.

Benutzerauthentifikation

Paßworte

Das am häufigsten eingesetzte Verfahren zur Authentifizierung.

Der Paßwortschutz ist leicht zu verstehen und ebenso leicht zu implementieren.

Die Paßwortauthentifikation ist leicht zu durchbrechen, da die Kombination von Benutzernamen und Paßwort relativ leicht ist. Eine Untersuchung zeigte, daß 86% aller Paßwörter eines Systems aus Vornamen, Nachnamen, Straßen- und Städtenamen und gängigen Wörtern bestanden.

Man kann eine Zufallszahl in der Paßwortdatei speichern, die dem Benutzerpaßwort angehängt wird, bevor die Verschlüsselung stattfindet. Die Zufallszahl wird geändert, sobald sich das Paßwort ändert. Ist die Zufallszahl *N* Bit lang, so vergrößert sich die Liste der potentiellen Paßwörter um den Faktor 2^n, womit der Angriff über vorher verschlüsselte Paßwörter meistens nutzlos wird.

Der Rechner sollte nur Paßwörter akzeptieren, die eine gewisse Länge (z.B. 8 Zeichen) haben und mind. je einen Klein-, Großbuchstaben und ein Sonderzeichen enthalten.

Regelmässiges ändern der Paßwörter, im Extremfall Einmal-Paßwörter, begrenzen den Schaden bei Entdeckung der Paßwörter.

Eine Alternative zur Eingabe von Paßwörtern ist das Stellen von Fragen an den Benutzer, deren Antworten verschlüsselt gespeichert sind. Die Fragen sind speziell auf den Benutzer zugeschnitten und werden zufällig ausgewählt.

Ein andere Alternative ist die Herausforderungsantwort, wobei der Benutzer eine Funktion definiert, die zur Authentifizierung verwendet werden soll, z.B. x^2. Der Computer gibt ein Argument vor und der Benutzer antwortet mit dem Resultat, das der Computer dann vergleicht. Diese Funktion kann zeitlich variieren, z.B. von der Uhrzeit.

Physikalische Identifikation

Der Benutzer benötigt einen physikalischen Gegenstand, z.B. eine Codekarte, um sich zu authentifizieren.

Eine Alternative ist die Messung schwer fälschbarer physikalischer Eigenschaften, z.B.Fingerabdrücke, Stimmproben etc.

Bei der Überprüfung von Unterschriften ist es sinnvoller die Reihenfolge in der die Linie gezeichnet werden zu überprüfen statt die Unterschrift im Ganzen.

Recht einfach ist die Überprüfung der Fingerlänge.

Blut oder Urinproben würden z.B. recht sichere Identifikationen ermöglichen, wären aber wohl kaum akzeptabel.

Gegenmaßnahmen

- Zugang nur über spezielle Terminals,
- Rückruf bei Zugang über Telefonwählleitungen,
- Erhöhung der Wartezeit nach jedem erfolglosen Zugangsversuch,
- Protokollierung der Zugangsversuche,
- Aufstellen von Fallen und Ködern.

Schutzmechanismen

Es sind zu unterscheiden:
Politik Wessen Daten vor wem geschützt werden sollen.
Mechanismus Wie setzt das System eine Politik durch.

Schutzdomänen

In einem Rechensystem sind viele *Objekte* zu finden, die geschützt werden müssen, z.B. CPU, Speichersegmente, Prozesse, Dateien etc.

Jedes Objekt wird durch einen eindeutigen Namen identifiziert und verfügt über eine Menge von Operationen, die auf ihm ausgeführt werden können.

Eine *Domäne* ist eine Menge von (Objekt, Rechte)-Paaren.
Jedes Paar gibt ein Objekt und eine Teilmenge der Operationenmenge an, die auf diesem Objekt ausgeführt werden können.

Ein *Recht* bedeutet in diesem Kontext, die Erlaubnis für die Ausführung einer Operation zu besitzen.

Zu jedem Zeitpunkt befindet sich ein Prozeß in irgendeiner Domäne.

Während der Ausführung kann der Prozeß die Domäne wechseln. Die Regeln hängen vom jeweiligen System ab.

In UNIX wird die Domäne eines Prozesses durch die Benutzeridentifikation und die Gruppenidentifikation festgelegt. Es ist möglich, daß ein gegebenes Paar (uid,gid) eine vollständige Liste aller Objekte zusammen mit der Art des Zugriffs zum Lesen, Schreiben und Ausführen anzugeben (Dateien, Spezialdateien), auf die zugegriffen werden kann. Zwei Prozesse mit

derselben Kombination von (uid, gid) haben auf genau dieselbe Objektmenge Zugriff.

Zudem besteht UNIX aus zwei Hälften: einem Benutzerteil und einem Kernteil. Wenn ein Prozeß einen Systemaufruf durchführt, schaltet er vom Benutzerteil in den Kernteil. Der Kernteil hat Zugriff auf eine andere Menge von Objekten als der Benutzerteil. So kann zum Beispiel auf alle Seiten im Arbeitsspeicher zugreifen, auf die gesamte Platte etc. In diesem Sinn stellt ein Systemaufruf einen Domänenwechsel dar.

Führt ein Prozeß den Systemaufruf EXEC mit eine Datei aus, für die das SETUID oder SETGID gesetzt ist, so erhält er die neue effektive Benutzer- oder Gruppenidentifikation, was einem Domänenwechsel entspricht.

Zugriffskontrollisten

Jedem Objekt wird eine geordnete Liste zugeordnet, die *Zugriffskontrolliste* oder *ACL.*

Jeder Eintrag in der ACL gibt die Benutzeridentifikation, die Gruppenidentifikation und die erlaubten Zugriffe an.

Capabilities

Bei diesem Verfahren wird jedem Prozeß eine Liste mit den Objekten, auf die er zugreifen darf, zusammen mit den Informationen, welche Operationen zulässig sind, zugeordnet. Diese Liste wird als *Capability Liste* bezeichnet und jedes einzelne Element wird *Capability* genannt.

Die Capabilty-Listen müssen vor Verfälschungen geschützt gespeichert werden. Möglichkeiten dazu sind:

1. Spezielle Hardware-Architektur, bei der jedes Wort mit einem Markierungsbit versehen ist, mit dem gekennzeichnet wird, ob das Wort eine Capability ist oder nicht.

2. Die Capabilities werden vollständig innerhalb des BS verwaltet.

3. Die Capabilities werden verschlüsselt im Benutzeradressraum gespeichert. Diese Möglichkeit ist besonders für verteilte System gut geeignet.

Zusammenfassung

Sicht von außen:

Ein Dateisystem ist eine Menge von Dateien und Verzeichnissen zusammen mit den für sie definierten Operationen.

Dateien können gelesen oder beschrieben werden, Verzeichnisse können erzeugt und zerstört werden und Dateien können von einem in ein anderes Verzeichnis bewegt werden.

Die Benennung von Dateien, die Struktur, die Typisierung, der Zugriff und die Attribute sind ebenfalls wichtige Entwurfsfestlegungen.

Die meisten modernen Dateisysteme unterstützen ein hierarchisches Dateisystem, in dem Verzeichnisse beliebig tief geschachtelt werden können.

Sicht von innen:

Zuordnung der Plattenblöcke zu den Dateien, gemeinsame Nutzung von Dateien, Verwaltung des freien Speicherplatzes

Strukturierung von Verzeichnissen, inkl. Speicherung von Namen, Attributen und Plattenadressen und I-Nodes

Sicherheit

Schutz

EIN-/AUSGABE

Einführung

Eine der Hauptaufgaben des Betriebssystems ist es, alle Ein-/Ausgabe-Geräte eines Computers zu überwachen und zu steuern.

Der Code für die Ein-/Ausgabe stellt einen beträchtlichen Teil des gesamten Betriebssystems dar.

In diesem Kapitel geht es um die Verwaltung der I/O, nicht die konkrete Programmierung.

I/O-Hardware

I/O-Geräte

Block-orientierte Geräte Ein block-orientiertes Gerät speichert Informationen in Blöcken fester Größe, von denen jeder eine eigene Adresse besitzt. Die Blockgrößen reichen von ca. 128 bis 1024 Bytes. Eine wesentliche Eigenschaft ist, daß jeder Block unabhängig von den anderen gelesen werden kann.

Beispiel: Festplatten, Disketten.

Zeichen-orientierte Geräte Ein zeichen-orientiertes Gerät erzeugt oder akzeptiert Zeichenströme, ohne dabei auf irgendeine Blockstruktur zu achten. Es ist nicht adressierbar und kennt keine Suchoperation.

Beispiel: Terminals, Zeilendrucker, Mäuse, Netzwerkschnittstellenkarten, Magnetbandgeräte.

Das o.g. Klassifikationsschema ist nicht perfekt, da eine ganze Reihe von Geräten nicht eingeordnet werden können,

Beispiel: Uhren, Monitore.

Steuerwerke

Die I/O-Einheiten bestehen aus einer mechanischen und einer elektronischen Komponente.

Lassen sich die beiden Komponenten trennen, so wird die elektronische Komponente als *Steuerwerk* (Controller) bezeichnet.

Die mechanische Komponente ist das Gerät selbst.

Das Steuerwerk ist normalerweise über ein Kabel mit dem Gerät verbunden.

Falls die Schnittstelle standardisiert ist, können auch mehrere Geräte gesteuert werden.

Die Schnittstelle zwischen Steuerwerk und Gerät ist meistens eine Schnittstelle auf sehr niedrigem Niveau.

Beispiel: Lesen von der Platte

- Von der Platte kommt ein serieller Bit-Strom, der mit einem Vorspann beginnt, gefolgt von den Bytes des Blocks, und abgeschlossen von einer Prüfsumme oder einem fehlerkorrigierenden Code.
- Die Aufgabe des Steuerwerks ist es, den seriellen Bit-Strom in Byte-Blöcke zu konvertieren und vice-versa, sowie ggf. die notwendigen Fehlerkorrekturen durchzuführen.
- Der Block wird typischerweise in einem Puffer innerhalb des Steuerwerks Bit für Bit gesammelt. Ist die Prüfsumme o.k., wird der Block in den Hauptspeicher kopiert.

Speicherbasierte I/O

Jedes Steuerwerk besitzt ein paar Register, die zur Kommunikation mit dem Prozessor benutzt werden. Sind diese Register Teil des normalen Adreßraums, spricht man von *speicher-basierter I/O* (memory mapped I/O).

Das BS führt Kommandos aus, indem es Befehle in die Steuerwerksregister schreibt.

Wenn ein Befehl akzeptiert worden ist, kann sich der Prozessor anderen Aufgaben widmen.

Ist die Bearbeitung beendet, löst das Steuerwerk eine Unterbrechung aus.

Direkter Speicherzugriff (DMA)

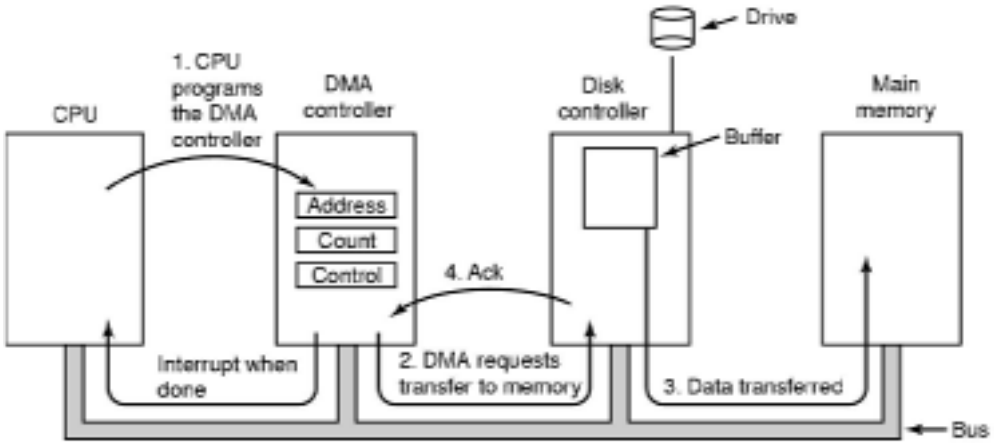
Viele Steuerwerke, insbesondere für block-orientierte Geräte, unterstützen den *direkten Speicherzugriff* (DMA Direct Memory Access).

Der Prozessor übergibt dem Steuerwerk die Blockadresse auf der Platte und die Speicheradresse, an die der gelesene Block kopiert werden soll, sowie die Anzahl der zu lesenden Bytes.

Nachdem das Steuerwerk vom Gerät den ganzen Block gelesen hat und die Prüfsumme überprüft hat, kopiert es die geforderte Anzahl Bytes an die angegebene Adresse und löst eine Unterbrechung aus, wenn der Vorgang beendet ist.

Die Pufferung der Daten im Steuerwerk ist wegen der Unterschiede in den Datenübertragungsraten notwendig.

Um die Übertragung mehrerer Blöcke zu optimieren, kann man den Abstand (Interleave) aufeinanderfolgender Blöcke auf der Platte an die Gegebenheiten anpassen.



I/O-Software

Ziele der I/O-Software:

Geräteunabhängigkeit: z.B. Disketten wie Festplatten behandeln.

Einheitliches Benennungsschema zur Identifikation von Dateien und Geräten mittels Pfadnamen

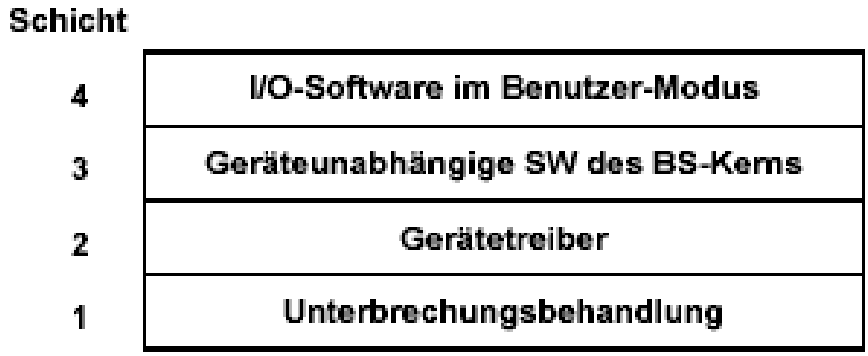
Fehlerbehandlung so nah an der Hardware wie möglich

Synchrone und asynchrone Übertragung

Zugriff regeln: gemeinsam und exklusiv benutzbare Geräte

Strukturierung der Software in vier Schichten:

1. Unterbrechungsbehandlung
2. Gerätetreiber
3. Geräte-unabhängige Betriebssystem-Software
4. Benutzer-Software



Unterbrechungsbehandlung

Die beste Möglichkeit Unterbrechungen zu verbergen ist, daß jeder Prozeß, der eine I/O-Operation ausführt, blockiert wird, bis die I/O-Operation beendet wird und die Unterbrechung auftritt.

1. Unterbrechungsbehandlungs-Schicht:

Die Geräteprozesse können sich durch:

- Semaphoren mittels Down,
- das Warten auf eine Nachricht mit RECEIVE oder ein
- WAIT-Kommando in einem Monitor selbst blockieren.

Nach Eintritt eines Ereignisses erfolgt die Entblockierung des Prozesses durch das Betriebssystem mittels eines Semaphore-UPs einer SEND-Nachricht oder eines Signals.

Gerätetreiber

Die Aufgabe eines Gerätetreibers ist es, eine abstrakte Anfrage von der geräte-unabhängigen Software entgegenzunehmen und dafür zu sorgen, daß die Anfrage ausgeführt wird.

Der gesamte geräte-abhängige Code gehört in den Gerätetreiber.

Der Gerätetreiber behandelt einen Gerätetyp oder höchstens eine Klasse von eng verwandten Geräten.

Ein Gerätetreiber setzt die Kommandos in die Register des Steuerwerkes ab und überprüft die korrekte Ausführung.

2. Gerätetreiber-Schicht:

Die Software dieser Schicht muß sich um die Details der diversen Geräte kümmern, z.B. den Interleave-Faktor, um korrekt arbeiten zu können. Die

Treiber beschreiben die Register des Rechenwerks, sie regeln die Queue-Abarbeitung von Kommandos der Form „lies Block *n*“ und schlafen bei Nichtbenutzung.

Geräte-unabhängige Betriebssystem-Software

Die Aufgabe der geräteunabhängigen Software ist es, I/O-Funktionen auszuführen, die allen Geräten gemeinsam sind, und eine einheitliche Schnittstelle zur Benutzer-I/O-Software zur Verfügung zu stellen.

Ein wesentlicher Punkt in einem BS ist die Benennung der Objekte.

Die geräte-unabhängige Software ist für die Umsetzung von symbolischen Gerätenamen in Treibernamen verantwortlich.

In UNIX bestimmt ein Gerätename, z.B. /dev/tty0, eindeutig den I-Knoten einer Spezialdatei. Der I-Knoten enthält die *Gerätehauptnummer* (major device number), die den zugehörigen Treiber festlegt, und die *Gerätenebennummer* (minor device number), die als Parameter an den Treiber übergeben wird, um das Gerät zu bestimmen.

Die Spezialdateien, die die Geräte darstellen, werden unter UNIX durch die üblichen rwx-Rechte geschützt.

Die Pufferung der Daten, Verwaltung der Freispeicherliste sind eine weitere wichtige Aufgaben.

Zugriffssynchronisation exklusiv-nutzbarer Geräte.

Behandlung von Fehlern, die von den Treibern nicht behandelt werden können.

3. Geräteunabhängige Software-Schicht:

Ausführung von I/O-Funktionen, die für alle Geräte gemeinsam gelten, wie z.B. die Pufferung, (das Fehlerhandling), der Geräteschutz, die Gerätebenennung, ...

Benutzer-Software

Bibliotheken

Der größte Teil der I/O-Software ist im Betriebssystem enthalten.

Ein kleiner Teil besteht aus Bibliotheken, die zu Benutzerprogrammen dazugebunden werden können.

Systemaufrufe werden normalerweise über Bibliotheksroutinen aufgerufen.

Spooling-System

Spooling ist die Möglichkeit, exklusiv nutzbare Geräte in einem Mehrprogrammsystem zu behandeln.

Ein spezieller Prozeß, der *Spooling-Dämon*, und ein spezielles Verzeichnis, das *Spooling-Verzeichnis*, bilden das Spooling-System.

Um Daten zu spoolen, werden diese in das Spool-Verzeichnis kopiert, verschoben oder dort generiert. Es ist Aufgabe des Dämons, der als einziger eine Schreibberechtigung für die Spezialdatei des Gerätes besitzt, die Daten auszugeben.

Spooling wird beim Drucken, bei Netzwerken u.a. benutzt.

4. Benutzer-I/O-Software:

Die Software, die nicht zum Betriebssystem gehört, die aber der Ein/Ausgabe dient, z.B. Bibliotheken wie <stdio.h> von C. Auch Spooling-Dämonen gehören hier dazu.

Festplatten

Vorteile von Festplatten gegenüber dem Arbeitsspeicher:

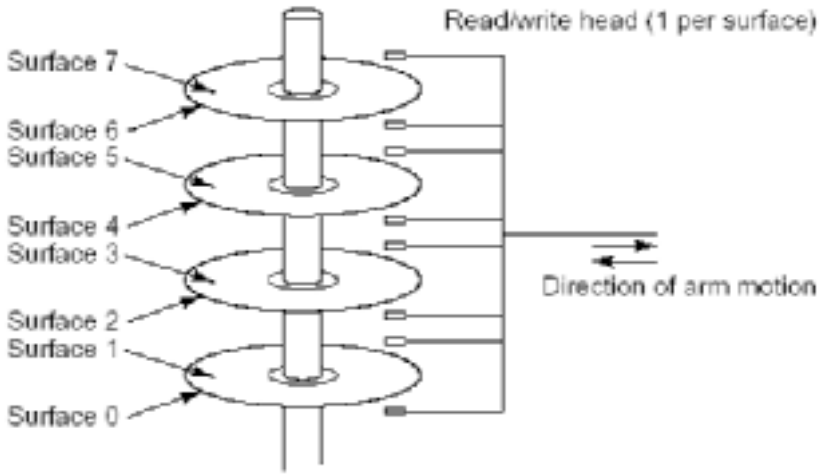
- die verfügbare Speicherkapazität ist wesentlich größer,
- der Preis pro Bit ist viel geringer, und
- die Informationen gehen nicht verloren, wenn der Strom abgeschaltet wird.

Festplatten-Hardware

Alle Festplatten sind zylinder-organisiert.

Jeder Zylinder enthalt so viele Spuren, wie Schreib-/Leseköpfe vorhanden sind.

Die Spuren sind in Sektoren unterteilt, wobei die Anzahl der Sektoren idR. zwischen 8 und 32 liegt.



Plattenarm-Scheduling-Algorithmen

Die Zeit, die benötigt wird, um einen Plattenblock zu lesen oder zu schreiben, wird durch drei Faktoren bestimmt:

1. die Dauer des Suchvorgangs (die Zeit für die Armbewegung),

2. die Umdrehungsverzögerung, und

3. Übertragungszeit.

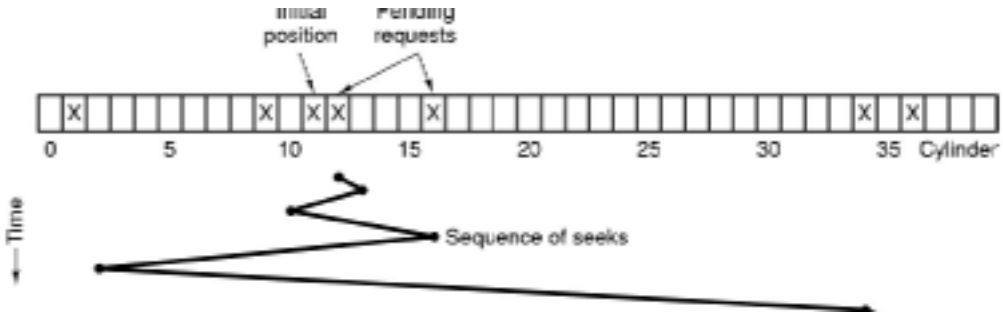
Algorithmen:

FCFS/FIFO First Come First Served

- Der Plattentreiber akzeptiert immer nur eine Anfrage.
- Die Anfragen werden in der Reihenfolge des Eintreffens bearbeitet.
- Vorteil: fair
- Nachteil: langsam

SSF Shortest Seek Time First

- Der Plattentreiber akzeptiert beliebig viele Anfragen, die in einer Tabelle gespeichert werden.
- Als nächste zu bearbeitenden Anfrage wird diejenige ausgewählt, bei der die erforderliche Armbewegung am kleinsten ist.
- Vorteil: schnelle Antwortzeiten
- Nachteil: Bei stark belasteten Platten tendiert der Arm dazu, sich in der Mitte aufzuhalten, Anfragen nach peripheren Zylindern werden sehr selten bearbeitet.



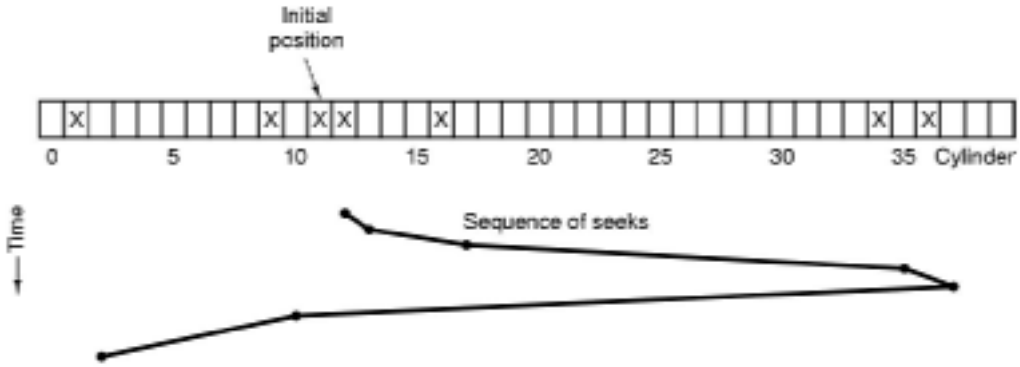
Aufzugsalgorithmus (elevator algorithm)

Ein Bit gibt an, in welche Richtung (up, down) sich der Arm bewegt.

Der Arm bewegt sich solange in eine Richtung, bis alle Anforderungen auf seinem Weg erfüllt sind.und kehrt dann um..

Die Obergrenze für die Armbewegungen ist 2x die Anzahl der Zylinder, um alle Anfragen zu bearbeiten.

Vorteil: akzeptable Geschwindigkeit und fair



RAID-Systeme

RAID (Redundant Array of Inexpensive Disks)

Zusammenschaltung vieler Platten

Beispiel: 38 Platten für 32 Datenbits und 6 Parity Bits

Hamming Codierung bei Verwendung der Bits 1,2,4,8,16,32 als Paritätsbits, damit kann der Ausfall einer Platte kompensiert werden.

Fehlerbehandlung

Fehlerursachen bei Festplatten:

- Programmierfehler
- flüchtige Prüfsummenfehler
- permanente Prüfsummenfehler
- Fehler beim Suchvorgang
- Fehler des Steuerwerks

Uhren

Uhren werden auch *Timer* genannt.

Uhren werden durch einen Treiber betrieben.

Einfache Uhren

Einfache Uhren sind an die Stromversorgung (110V / 230V) gekoppelt und lösen bei jedem Spannungszyklus (50 oder 60 Hz) eine Unterbrechung aus.

Kompliziertere Uhren

Kompliziertere Uhren bestehen aus einem Kristalloszillator, einem Zähler und einem Holding-Register.

Wenn ein Quarz genau geschnitten ist und unter Spannung gehalten wird, erzeugt der Quarz periodische Signale mit einer sehr hohen Genauigkeit.

Die Frequenz liegt in Abhängigkeit des gewählten Kristalls zwischen 5 und 100 MHz.

Bei jedem Signal wird der Zähler um Eins erniedrigt. Falls der Wert Null erreicht, wird eine Unterbrechung ausgelöst.

Programmierbare Uhren können in verschiedenen Modi betrieben werden:

- Einmalmodus
- Wiederholungsmodus

Die Unterbrechungsfrequenz kann frei gewählt werden. Z.B. für einen 1 MHz Kristall wird der Zähler jede Mikrosekunde erniedrigt.

Mit einem 16-Bit-Register können Unterbrechungen mit Raten zwischen 1 und 65535 Mikrosekunden ausgelöst werden.

Uhr-Software

Aufgaben:

- das Verwalten der Uhrzeit,
- das Verhindern, daß Pozesse länger ausgeführt werden, als ihnen gestattet wurde,
- das Buchführen über die Prozessornutzung,
- die Behandlung des ALARM-Systemaufrufs durch Benutzerprozesse,
- das Bereitstellen von Uhren zur Überwachung von Betriebssystemaktivitäten und

- das Profiling, das Führen von Statistiken und andere Überwachungen.

Terminals

Terminal-Hardware

Anschlußarten:

RS-232-Anschluß

Geräte, die aus einer Tastatur und einem Bildschirm bestehen und zur Kommunikation die serielle Schnittstelle verwenden, über die Daten bitweise übertragen werden.

Da sowohl Computer als auch die Terminals intern mit ganzen Zeichen arbeiten, sind Bausteine, die *UART*s (Universal Asynchronous Receiver Transmitter), für die Konvertierung entwickelt worden.

Speicher-basiert

Diese Terminals sind fester Bestandteil des Rechners, die Schnittstelle ist das *Video-RAM*. Dieser Speicher ist Teil des Adreßraums des Rechners und wird durch den Prozessor auf dieselbe Art wie der restliche Speicher adressiert.

Speicher-basierte Terminals

Das *Video-Steuerwerk* (Graphik-Kontroller) liest das Video-RAM aus und erzeugt für den Bildschirm die entsprechenden Video-Signale.

Der Bildschirm generiert einen Elektronenstrahl bei Monochrom und drei Strahlen bei Farbmonitoren.

Jeder Bildpunkt des Monitors wird im Video-RAM als ein Bit (Monochrom) oder bis zu 24 bzw. 32 Bit dargestellt. Die Darstellung am Monitor ist völlig flexibel (Zeichengröße, Zeichensätze, Grafiken, Farben,...).

Die Tastatur ist vom Monitor getrennt und wird über eine serielle Schnittstelle angeschlossen.

Die meisten Tastaturen liefern nur eine Tastennummer, die dann vom Treiber noch in das Zeichen umgesetzt werden muß. Damit sind z.B. nationale Tastaturbelegungen möglich.

Input-Software

Die Hauptaufgabe des Tastaturtreibers ist es, die Eingaben von der Tastatur zu sammeln. Dabei sind zwei Modi möglich:

Zeichen-orientiert Eingaben unverändert entgegennehmen und weitergeben (raw mode)

Zeilen-orientiert Der Treiber übernimmt das Editieren innerhalb einer Zeile und gibt die Daten zeilenweise weiter (cooked mode)

Die Zeichen werden meistens zunächst gepuffert und dann vom Treiber verarbeitet, abhängig vom Modus.

Die Pufferung erfolgt in separaten Puffern für jedes angeschlossene Terminal.

Es wird von den Benutzern als komfortabel angesehen, wenn die eingegeben Zeichen am Monitor protokolliert werden (Echo-Funktion). Dies kann hardware- oder softwaremäßig realisiert werden.

Probleme des Treibers:

- Eingabe über 80 Zeichen
- Tabulatoren
- Zeilenvorschübe (UNIX: nur Zeilenvorschub, MS-DOS: Wagenrücklauf/Zeilenvorschub, MacOS: nur Wagenrücklauf)
- Löschzeichen (Ctrl-H, Backspace, Delete)
- Steuerzeichen:

Zeichen	Bedeutung unter UNIX
Backspace	Lösche ein Zeichen
@	Lösche aktuelle Zeile
\	Maskierung - akzeptiere nächstes Zeichen ohne Sonderbedeutung
Crtl-S	Halte Ausgabe an
Crtl-Q	Setze Ausgabe fort
DEL	Unterbrich Prozeß (SIGINT)
Crtl-\	Erzwinge Core-Dump (SIGQUIT)
Crtl-D	Dateiende

Output-Software

Die Ausgabe ist wesentlich einfacher, aber die Treiber für RS-232 und speicher-basierte Terminals unterscheiden sich vollkommen

Bei RS-232-basierte Terminals wird jedem Terminal ein Puffer zugeordnet. Programme schreiben die Ausgaben in den Puffer, der Treiber gibt sie dann zeichenweise am Monitor aus.

Bei speicher-basierten Terminals werden die Zeichen direkt in das Video-RAM gesendet. Dabei muß der Treiber die Steuerzeichen interpretieren.

Bildschirm-orientierte Programme (z.B. Editoren) müssen den Bildschirm auf komplexe Art und Weise manipulieren. Viele Treiber bieten dazu spezielle Steuersequenzen an:

Bewege der Cursor um eine Position nach rechts (bez. links, unten, oben).

Bewege den Cursor nach Position (*x*,*y*).

Füge ein Zeichen oder eine Zeile an der Position des Cursors ein.

Lösche ein Zeichen oder eine Zeile an der Position des Cursors.

Verschiebe den Bildschirminhalt um *n* Zeilen nach oben oder unten.

Lösche den Bildschirminhalt von der Position des Cursors bis zum Ende der Zeile oder des Bildschirms.

Wechsle in einen bestimmten Modus (inverse Darstellung, Unterstreichung, Blinken, normaler Modus, ...)

Erzeuge, schließe, bewege, ... ein Fenster.

Zusammenfassung

Ein beträchtlicher Teil des Betriebssystems befaßt sich mit I/O

I/O-Hardware

I/O-Software

Strukturierung der Software in vier Schichten:

1. Unterbrechungsbehandlung
2. Gerätetreiber
3. Geräte-unabhängige Betriebssystem-Software
4. Benutzer-Software

Unterbrechungsbehandlung

Gerätetreiber

Gerätearten: Festplatten, Uhren, Terminals

VERKLEMMUNGEN

Einführung

Viele Betriebsmittel können zu einem Zeitpunkt nur von einem Prozeß benutzt werden.

Alle BS müssen eine Möglichkeit zur Verfügung stellen, mit der einem Prozeß temporär ein exklusiver Zugriff auf ein bestimmtes Betriebsmittel erlaubt wird.

Bei vielen Anwendungen benötigt ein Prozeß mehrere Betriebsmittel gleichzeitig im exklusiven Zugriff.

In einem Mehrprogrammsystem können dabei schwierige Probleme auftreten. Beispiel:

> Prozeß A fordert die Erlaubnis an, den Drucker benutzen zu dürfen, und die Erlaubnis wird erteilt.
>
> Prozeß B fordert danach die Erlaubnis an, das Bandgerät benutzen zu dürfen, und die Erlaubnis wird erteilt.
>
> Prozeß A fordert die Erlaubnis an, das Bandgerät benutzen zu dürfen, die Erlaubnis wird verweigert, bis das Bandgerät freigegeben wird.
>
> Prozeß B fordert die Erlaubnis an, den Drucker benutzen zu dürfen, die Erlaubnis wird verweigert, bis der Drucker freigegeben wird.
>
> Beide Prozesse sind blockiert und verbleiben für immer in diesem Zustand.

Eine solche Situation wird ***Deadlock*** oder ***Verklemmung*** genannt.

Deadlocks treten nicht nur beim Zugriff auf exklusiv genutzte Geräte sondern auch in vielen anderen Situationen auf:

In einem Datenbanksystem müssen zur Vermeidung zeitkritischer Abläufe eine Reihe von Sperren auf die benutzten Datensätze gesetzt werden. Falls ein Prozeß A den Datensatz R1 sperrt, und Prozeß B den Datensatz R2, so entsteht ein Deadlock, wenn die Prozesse eine Sperre auf den Datensatz des jeweils anderen zu setzen versuchen.

Betriebsmittel

Deadlocks können entstehen, wenn Prozessen ein exklusiver Zugriff auf ein Betriebsmittel gestattet worden ist.

Ein *Betriebsmittel* ist ein Objekt, auf das der Zugriff durch das BS erteilt wird.

Bei manchen Betriebsmitteln gibt es mehrere Instanzen, jede Kopie kann verwendet werden, um eine Anforderung nach diesem Betriebsmittel zu befriedigen. Ein Betriebsmittel kann zu jedem Zeitpunkt nur von höchstens einem Prozeß genutzt werden.

Es gibt zwei Arten von Betriebsmitteln:

Unterbrechbare Betriebsmittel

Unterbrechbare Betriebsmittel können einem Prozeß entzogen werden, und der Prozeß kann nach Wiederzuteilung seine Berechnung fortsetzen.

Beispiel.: Speicher

Ununterbrechbare Betriebsmittel

Die ununterbrechbaren Betriebsmittel können einem Prozeß nicht entzogen werden, ohne daß dessen bisherige Berechnungen abgebrochen werden müssen.

Beispiel.: Drucker, CD-Brenner

Kritische Situationen in Verbindung mit unterbrechbaren Betriebsmitteln können meistens aufgelöst werden, wenn die Betriebsmittel temporär entzogen werden.

Deadlocks entstehen in Verbindung mit ununterbrechbaren Betriebsmitteln.

Nutzung eines Betriebsmittel erfolgt in den Schritten:

1. Anfordern des Betriebsmittels,
2. Benutzen des Betriebsmittels,
3. Freigeben des Betriebsmittels.

Falls ein Betriebsmittel bei Anforderung nicht verfügbar ist, blockiert sich der anfordernde Prozeß.

Verklemmungen

Eine Menge von Prozessen befindet sich in einem „Deadlock“-Zustand, falls jeder Prozeß der Menge auf ein Ereignis wartet, das nur ein anderer Prozeß der Menge auslösen kann.

Für eine Deadlock-Situation müssen die folgenden vier Bedingungen erfüllt sein:

Die Bedingungen des wechselseitigen Ausschlusses. Jedes Betriebsmittel wird entweder von genau einem Prozeß belegt oder ist frei.

Die Belegungs- und Wartebedingung. Ein Prozeß, der bereits Betriebsmittel belegt, kann weitere Betriebsmittel anfordern.

Die Ununterbrechbarkeitsbedingung. Die Betriebsmittel, die von einem Prozeß belegt werden, können nicht entzogen werden, sondern müssen explizit vom belegenden Prozeß freigegeben werden.

Die zyklische Wartebedingung. Es muß eine zyklische Kette aus zwei oder mehr Prozessen existieren, so daß jeder Prozeß ein Betriebsmittel anfordert, das von dem nächsten Prozeß in der Kette belegt wird.

Alle vier Bedingungen müssen erfüllt sein, damit eine Deadlock- Situation eintreten kann. Falls eine Bedingung nicht erfüllt ist, ist keine Deadlock-Situation möglich.

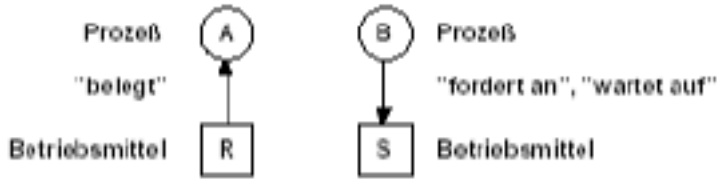
Modellierung mit Betreibsmittelgraphen

Die Graphen besitzen zwei Arten von Knoten: Prozesse, die als Kreise dargestellt werden, und Betriebsmittel, die als Quadrate dargestellt werden.

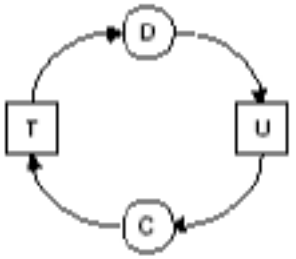
Eine Kante von einem Betriebsmittel zu einem Prozeß bedeutet, daß das Betriebsmittel von diesem Prozeß belegt ist.

Eine Kante von einem Prozeß auf ein Betriebsmittel bedeutet, daß der Prozeß blockiert ist, weil er auf die Zuteilung dieses Betriebsmittels wartet.

- Betriebsmittelzuteilungsgraph (Holt, 1972):



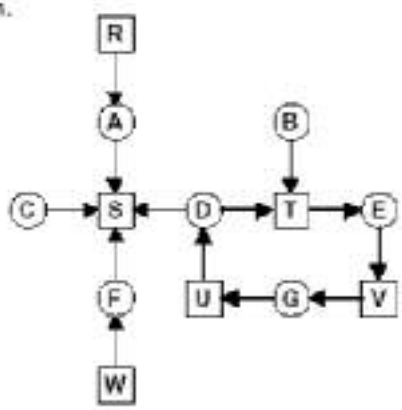
Eine einfache Deadlock-Situation:



Zyklus!

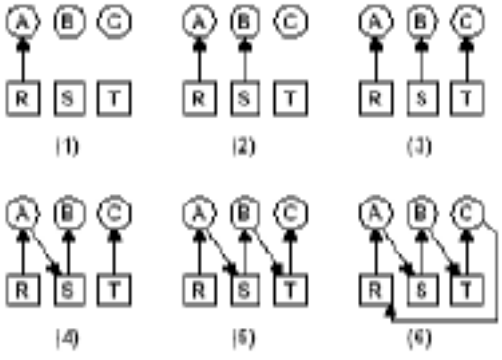
Ein Zyklus im Betriebsmittel-zuteilungsgraphen bedeutet die Existenz eines Deadlocks.

1. A belegt R und fordert S an.
2. B fordert T an.
3. C fordert S an.
4. D belegt U und fordert S und T an.
5. E belegt T und fordert V an.
6. F belegt W und fordert S an.
7. G belegt V und fordert U an.



- Eine mögliche Ausführungsfolge bei Nebenläufigkeit:

1. A fordert R an.
2. B fordert S an.
3. C fordert T an.
4. A fordert S an.
5. B fordert T an.
6. C fordert R an.



Deadlock!

Grundsätzlich gibt es vier Strategien, mit denen Deadlock- Situationen behandelt werden können:

1. Ignorieren des gesamten Problems,

2. die Erkennung und Behebung,

3. die dynamische Verhinderung durch vorsichtige Betriebsmittelzuteilung

und

4. die Vermeidung durch konzeptionelles Verbieten einer der vier notwendigen Bedingungen.

Der Vogel-Straussalgorithmus

Ignorieren ist die einfachste Strategie:

- Mathematiker finden es unakzeptabel
- Ingenieure fragen, wie oft das Problem auftritt, wie oft das System aus anderen Gründen abstürzt, und wie schwerwiegend ein Deadlock ist.

Beispiele endlicher Betriebsmittel in UNIX:

- Prozeßtabelle
- I-Nodes

Erkennung und Behebung

Der einfachste Fall: Nur genau ein Betriebsmittel jeder Klasse

Man konstruiert den Betriebsmittelgraphen

Enthält der Graph einen Zyklus liegt ein Deadlock vor.

Algorithmus zu Untersuchung eines Graphen auf Zyklen:

1. Für jeden Knoten *N* des Graphen führe die folgenden fünf Schritte mit *N* als Startknoten aus:
2. Initialisiere *L* mit der leeren Liste und setze alle Kanten auf unmarkiert.
3. Füge den aktuellen Knoten an das Ende der Liste von *L* und überprüfe, ob der Knoten zweimal in der Liste enthalten ist. Falls dies der Fall ist, dann erhält der Graph einen Zyklus (aus den in *L* enthaltenen Knoten) und der Algorithmus terminiert.

4. Überprüfe, ob von dem aktuellen Knoten noch nicht markierte Kanten ausgehen. Falls ja, gehe zu Schritt 5, sonst zu Schritt 6.
5. Wähle eine der unmarkierten Kanten aus und markiere sie. Setze den Endknoten dieser Kante als neuen aktuellen Knoten und gehe zu Schritt 3.
6. Es ist nun eine Sackgasse erreicht. Gehe zurück zu dem vorherigen Knoten, also zu dem Knoten, der zuvor der aktuelle Knoten war, und setze den Algorithmus mit Schritt 3 fort. Falls dieser Knoten der Startknoten ist, enthält der Graph keinen Zyklus und der Algorithmus terminiert.

Es gibt bessere als den vorgestellten Algorithmus zur Lösung des Problems. An dieser Stelle genügt uns jedoch die Existenz eines Algorithmus.

Deadlock-Erkennung mit mehreren Betriebsmitteln in jeder Klasse

Wenn es mehrere Instanzen einer Betriebsmittelklasse gibt, wird ein Matrix-basierter Algorithmus eingesetzt.

Im ersten Schritt sucht der Algorithmus nach einem Prozeß, der bis zu seiner Beendigung ausgeführt werden kann. Ein solcher Prozeß ist dadurch charakterisiert, daß alle seine Betriebsmittelanforderungen durch die aktuell verfügbaren erfüllt werden können. Bei Beendigung gibt der Prozeß alle seine Betriebsmittel frei.

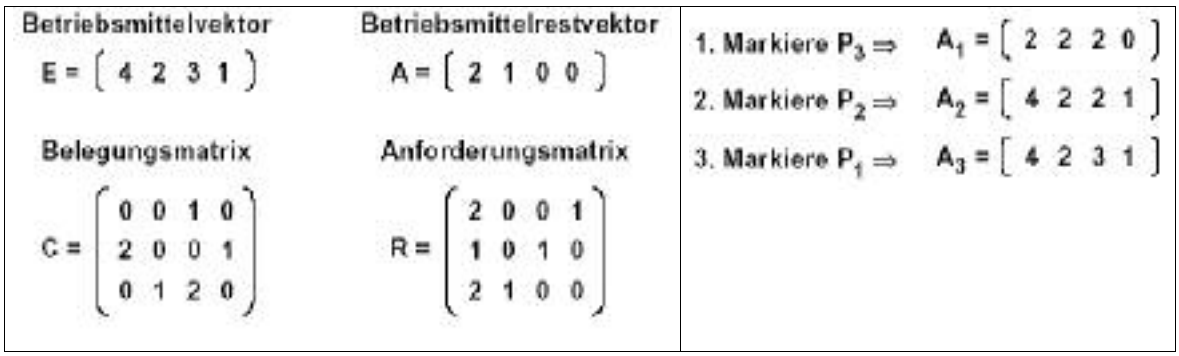
Falls alle Prozesse vollständig ausgeführt werden können, ist keiner an einem Deadlock beteiligt.

Falls einer oder mehrere niemals ausgeführt werden können, befinden sie sich in einem Deadlock-Zustand.

Deadlocks können also erkannt werden. Aber wann soll die Suche durchgeführt werden?

Wird bei jeder Anforderung überprüft, wird der Deadlock zum frühestmöglichen Zeitpunkt erkannt, aber sehr viel Prozessorzeit verbraucht.

Wenn die Überprüfung nur durchgeführt wird, wenn die Prozessorauslastung gering ist, fällt der Verlust an Rechenleistung nicht sehr ins Gewicht und wenige ausführbare Prozesse deuten auf Blockaden hin.



Deadlock-Behebung

Deadlock-Behebung mittels Unterbrechung Temporärer Entzug eines Betriebsmittels und Zuteilung an einen anderen Prozeß.

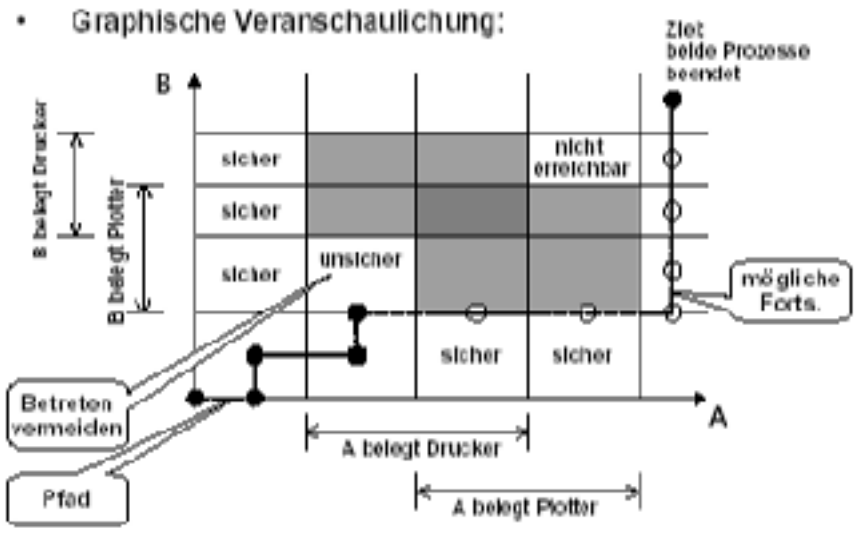
Deadlock-Behebung mittels teilweiser Wiederholung Prozesse schreiben ihren Berechnungszustand an sog. Checkpunkten in eine Datei. Aufgrund dieser Beschreibung kann eine Berechnung dann fortgesetzt werden. Der Deadlock wird behoben indem ein Prozeß, der die benötigten Betriebsmittel belegt, auf seinen letzten Checkpunkt zurückgesetzt wird und ihm die Betriebsmittel entzogen werden.

Deadlock-Behebung mittels Prozeßabbruch Ein Prozeß wird abgebrochen und seine Betriebsmittel werden entzogen. Können die anderen Prozesse ihre Arbeit dann immer noch nicht fortsetzen, müssen weitere Prozesse abgebrochen werden.

Verhinderung

Ein Prozeß wird typischerweise seine Betriebsmittel nicht auf einmal sondern in mehreren Schritten anfordern.

Kann ein Deadlock durch geeignete Zuteilungen verhindert werden? Ja, wenn gewisse Informationen vorher bekannt sind.



Sichere und unsichere Zustände

Ein Zustand heißt *sicher*, wenn er kein Deadlock-Zustand ist und es eine Möglichkeit gibt, alle vorliegenden Anforderungen zu erfüllen, indem die Prozesse in irgendeiner Reihenfolge ausgeführt werden.

Ein unsicherer Zustand ist kein Deadlock-Zustand.

Der Unterschied zwischen einem sicheren und einem unsicheren Zustand ist, daß das System von einem sicheren Zustand aus gewährleisten kann, daß alle Prozesse ihre Ausführung beenden können, wohingegen von einem unsicheren Zustand aus diese Garantie nicht gegeben werden kann.

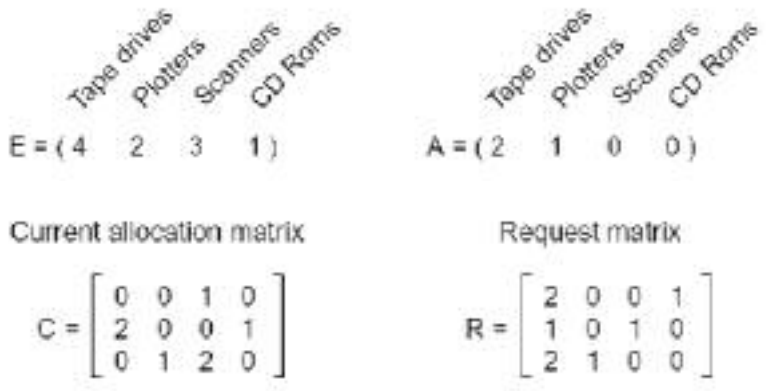
Der Bankieralgorithmus

- Der Algorithmus geht zurück auf Dijkstra (1965).
- Analogie:
  - Bankier — Betriebssystem
  - Kunden — Prozesse
  - Geld — Einzige Betriebsmittelklasse
  - Kreditbetrag — Betriebsmittel
- Idee:
  - Der Bankier hat insgesamt eine geringere Geldmenge zur Verfügung als die Summe aller Kreditzusagen, die er macht.
  - Er weiß, daß nicht alle Kunden gleichzeitig ihren gesamten Kreditrahmen ausschöpfen werden.
  - Er kann eine Kreditforderung verzögern und erst später zuteilen.
  - Er teilt einen Kredit nur dann zu, wenn der daraus resultierende Zustand "sicher" ist, d.h. wenn er letztlich alle Kreditforderungen der Kunden in deren Kreditrahmen befriedigen kann.

Der folgende Scheduling-Algorithmus kann Deadlocks verhindern.

Gegeben seien:

- die Belegungsmatrix *C*
- die Anforderungsmatrix *R*
- der Vektor *E* der insgesamt verfügbaren Betriebsmittel
- der Vektor *P* der belegten Betriebsmittel
- der Vektor *A*, der aktuell verfügbaren Betriebsmittel



Algorithmus, zur Überprüfung, ob ein Zustand sicher ist.

1. Suche eine Zeile in *R*, in der alle Betriebsmittelanforderungen kleiner oder gleich *A* sind. Falls keine solche Zeile existiert, wird das System einen Deadlock-Zustand erreichen, da kein Prozeß seine Berechnungen vollständig ausführen kann.
2. Nimm an, daß der Prozeß der ausgewählten Zeile alle benötigten Betriebsmittel angefordert hat und später seine Ausführungen beendet. Markiere den Prozeß als terminiert, und füge die von ihm belegten Betriebsmittel zum Vektor *A* hinzu.
3. Wiederhole die Schritte 1. und 2. solange, bis entweder alle Prozesse als terminiert markiert sind, und damit der Anfangszustand sicher ist, oder bis ein Deadlock-Zustand auftritt, und damit der Anfangszustand unsicher ist.

Beispiel: Der Rechner hat 10 Magnetbandstationen

Zustand 1			Zustand 2			Zustand 3		
Prozess	bel.	Max.	Prozess	bel.	Max.	Prozess	bel.	Max.
A	0	6	A	1	6	A	1	6
B	0	5	B	1	5	B	2	5
C	0	4	C	2	4	C	2	4
D	0	7	D	4	7	D	4	7
sicher			sicher			unsicher		
10 frei			2 frei			1 frei		

Ein Zustand ist dann sicher, wenn das BS mindestens bei einem Prozeß seine Maximalforderung erfüllen kann (die anderen müssen u. U. warten).

Zustand 1 ist sicher, da jeder Prozeß befriedigt werden kann.

Zustand 2 ist sicher, da Prozeß C befriedigt werden kann.

Zustand 3 ist unsicher, da keiner der der Prozesse voll befriedigt werden kann.

Das Schema kann auf beliebig viele Betriebsmittel erweitert werden.

Achtung: Der Algorithmus macht unrealistische Annahmen:

Prozesse kennen ihren Betriebsmittelbedarf selten im voraus.

Die Anzahl der Prozesse in einem realen System ist nicht fest.

Betriebsmittel können ausfallen, so daß nicht die angenommene Anzahl zur Verfügung steht.

Bankier-Algorithmus:

- Durchführung:
  1. Überprüfe bei jeder neuen Anforderung eines Prozesses nach Betriebsmitteln, ob die Zuteilung zu einem sicheren Zustand führt.

     Zur Überprüfung der Sicherheit eines Zustands: vgl. 6.5.2.
  2. Falls ja: Teile zu.
  3. Falls nein: Blockiere den anfordernden Prozeß.
  4. Bei Beendigung eines Prozesses werden die diesem Prozeß zugeteilten Betriebsmittel wieder frei. Alle blockierten Prozesse werden geweckt und ihre Anforderung gemäß 1. neu bearbeitet.

Vermeidung

Deadlocks können in realen Systemen nicht verhindert werden, da die notwendigen Informationen über zukünftige Betriebsmittelanforderungen nicht verfügbar sind.

In realen Systemen versucht man, Deadlocks prinzipiell zu vermeiden.

Wechselseitiger Ausschluß

Falls die Betriebsmittel nie einem Prozeß exklusiv zugeteilt werden, können keine Deadlock-Zustände entstehen.

Spooling kann nicht für alle Betriebsmittelarten eingesetzt werden, z.B. Prozeßtabellen, ebenso kann die Konkurrenz um Speicherplatz für das Spooling selbst wieder zu Deadlocks führen.

Trotzdem ist diese Idee häufig anwendbar. Die Zuteilung eines Betriebsmittels sollte soweit wie möglich vermieden werden, und es sollte versucht werden, daß so wenige Prozesse wie möglich ein Betriebsmittel tatsächlich anfordern.

Belegungs- und Wartebedingung

Falls man es vermeiden kann, daß Prozesse die bereits Betriebsmittel belegen, weitere anfordern, können keine Deadlock-Zustände mehr auftreten.

Eine Möglichkeit, dies zu erreichen, ist, von allen Prozessen zu fordern, daß sie ihre benötigten Betriebsmittel vor Beginn der eigentlichen Ausführung anfordern. Falls alle Betriebsmittel zur Verfügung stehen, können sie dem Prozeß zugeteilt werden, und der Prozeß kann seine Berechnungen vollständig ausführen. Anderenfalls wird dem Prozeß keines der geforderten Betriebsmittel zugeteilt, und der Prozeß muß auf die Zuteilung warten.

Dieser Ansatz verschwendet Betriebsmittel, da sie auch belegt werden, wenn sie gar nicht gebraucht werden; aber Deadlocks werden vermieden.

Eine Alternative ist es, daß vor Zuteilung eines weiteren Betriebsmittels alle belegten Betriebsmittel freigegeben werden müssen. Danach kann der Prozeß dann alle Betriebsmittel neu anfordern.

Ununterbrechbarkeitsbedingung

Das Entziehen eines Betriebsmittel ist in besten Fall schwierig, im schlechtesten Fall unmöglich.

Zyklische Wartebedingung

Das Eintreten der zyklischen Wartebedingung kann auf verschiedene Arten vermieden werden.

Einfach: Jeder Prozeß hat zu jedem Zeitpunkt nur Anspruch auf ein Betriebsmittel. Benötigt er ein weiteres, muß er zunächst das belegte freigeben.

Alternative: Alle Betriebsmittel werden global nummeriert. Dann lautet die Regel: „Den Prozessen werden nur Betriebsmittel neu zugeteilt, die eine höhere Nummer als die bereits zugeteilten Betriebsmittel haben.“ Damit kann der Betriebsmittelgraph keinen Zyklus enthalten.

Es ist schwierig (wenn nicht unmöglich), eine Nummerierung zu finden, die alle Benutzer zufrieden stellt.

Resumee:

Bedingung	Ansatz
Wechselseitiger Ausschluß	Spooling
Belegungs- und Wartebedingung	Betriebsmittelanforderung zu Beginn der Berechnung
Ununterbrechbarkeitsbedingung	Betriebsmittel entziehen
zyklische Wartebedingung	Betriebsmittel numerisch ordnen

Zusammenfassung

Deadlock-Zustände sind ein Problem in jedem Betriebssystem.

Sie treten auf, wenn jeweils ein Betriebsmittel exklusiv zugeteilt worden ist und alle ein Betriebsmittel anfordern, das bereits von einem Prozeß dieser Menge belegt ist. Alle Prozesse sind dann blockiert, und keiner kann jemals ausgeführt werden.

Deadlocks können verhindert werden, indem überprüft wird, ob der nachfolgende Zustand sicher oder unsicher ist.

Ein Zustand ist sicher, falls es eine Folge von Ereignissen gibt, so daß alle Prozesse ihre Ausführung beenden können.

Der Bankiersalgorithmus verhindert Deadlocks, indem er solche Anforderungen zurückstellt, die das System in einen unsicheren Zustand versetzen.

Deadlocks können konzeptionell vermieden werden, indem das System entsprechend konstruiert wird.

Wenn es Prozessen nur gestattet wird, höchstens ein Betriebsmittel zu besitzen, ist die zyklische Wartebedingung nie erfüllt.

Wenn Prozesse nummerierte Betriebsmittel nur in einer aufsteigenden Reihenfolge anfordern dürfen, werden Deadlocks vermieden.

WINDOWS VERSUS LINUX

Windows NT/2000

Windows NT (New Technology) ist ein Server-basiertes Multitasking-Betriebssystem, also mit Netzwerkunterstützung. Unbestreitbarer Vorteil ist jedoch, daß es portable konzipiert ist und so auf verschiedenen Prozessoren implementierbar ist (Intel, Mips, DEC Alpha, etc.), was außer Unix bei keinem anderen der vorgestellten BS der Fall ist. Applikationen können somit Quelltextkompatibel erstellt werden. Interessant ist bei NT die Server-Anwendung. NT-3.x-Arbeitsstationen können etwa genausoviel wie Windows-95-Rechner. Die über das "normale" Windows hinausgehenden Eingenschaften sind:

- preemptive multitasking
- Aufteilung von Programmen in threads
- für Mehrprozessorsysteme geeignet
- virtuelle Speicherverwaltung (je Prozeß bis zu 4 GByte)
- Benutzerverwaltung, Zugriffsrechte, Autentifizierung
- Unterstützung großer Platten (bis 17'000'000 GByte)
- Unterstützung verschiedener Dateisysteme (FAT von DOS, HPFS von OS/2, NTFS von NT)
- Dateisystem NTFS (NT File System)
    - Zugriffsrechte
    - Fehlertoleranz (Wiederherstellung nach Systemabsturz)
    - Links, transaktionsourientierte Zugriffe
    - lange Dateinamen, Unterscheidung Groß-/Kleinschreibung
- dynamisch ladbare Gerätetreiber
- Netzwerkunterstützung (auch für Novell, TCP/IP, OS/2)

Das System zeigt sich gut strukturiert. Die Basis wird von drei Schichten gebildet:

- Hardware Abstraction Layer (HAL)
- Kern
- NT-Executive (Systemdienste)

Darauf setzen Subsysteme auf, die verschiedene Betriebssystem-Emulationen bereitstellen. Für DOS, OS/2 oder Windows-Anwendungen wir jeweils ein eigenes, abgeschottetes 32-Bit-Subsystem angelegt. Fehlerhafte Programme oder Verletzung von Zugriffsrechten beeinflußt die Arbeit des Servers nicht. Alle Netzzugriffe werden vom "Advanced Server" behandelt, einer übergeordneten Verwaltungsinstanz.

Konfigurationsdaten werden nicht in einzelnen Dateien, sondern in einer einzigen Konfigurationsdatenbank (Registry) gespeichert. Das System und die Treiber greifen auf diese Datenbank zurück. Beim Bootvorgang überprüft das System die Funktion aller Komponenten.

Der Nachfolger **Windows NT 4.x** ist das leistungsstarke 32-bit-Betriebssystem von Microsoft. Es ist speziell optimiert für den Netzwerkbetrieb in Verbindung mit dem Client-Server-Konzept. Microsoft vertreibt Windows NT in zwei Versionen, den Windows NT Server und die Windows NT Workstation. Beide Versionen werden auf getrennten CD-ROMs ausgeliefert. Die Windows-NT-Server-CD-ROM ist für den Aufbau eines kompletten NT-Netzwerk-Servers bestimmt, die Windows NT-Workstation-CD-ROM kann dazu genutzt werden, Arbeitsplatz-PCs mit dem Windows-NT-Workstationsystem auszustatten. Windows NT 4.0 stellt sich in der Oberfläche von Windows 95 dar, hat jedoch einen völlig anderen inneren Aufbau.

In Windows NT 4.0 ist der Microsoft Internet Explorer (Web-Browser) und bei Windows NT Server auch der Internet Information Server (IIS, Web-Server) integriert. Der Internet Information Server bietet die Errichtung eines eigenen Web-Servers mit den Diensten WWW, Gopher und FTP. Dieser Dienst integriert sich nahtlos in die BackOffice-Strategie von Microsoft. Windows NT 4.0 enthält einen DNS-Server (Domain Name Service) der in der alten Windows NT 3.51 Version oft vermisst wurde. Mit dem Remote Programm Load (RPL) ist es möglich, diskless Workstations (ohne Festplatte) unter Windows95 vom Windows NT-Server zu booten. Datenbankanbindungen über ODBC können über Internet Server realisiert werden. Wichtige Highlights von Windows NT 4.0 sind neben dem kostenlosen Internet Explorer und dem Exchange Client die Unterstützung einer grossen Anzahl neuer Treiber, höhere Grafikleistung durch die Verlagerung von Teilen des GDI in den Kernel, Einrichtung von Hardware Profiles und einer DirectX Schnittstelle um Spiele auf Windows NT lauffähig zu machen.

Windows NT unterstützt sowohl preemptives Multitasking und Multithreading wie auch Multiprocessing (Verteilen von Programmteilen auf mehrere CPUs). Es ist skalierbar auf bis zu 32 CPUs (Windows NT Workstation kann max. 2 CPUs ansprechen, für Windows NT Server gibt es je nach Prozessoranzahl unterschiedliche Lizenzen). Pro System werden 4 GByte RAM unterstützt, jeder Anwendung kann bis zu 2 GByte virtueller Arbeitsspeicher zugewiesen werden. Datenspeicher werden vom System bis zu 402 Mio. Terabyte untersützt. Microsoft gibt als Systemvoraussetzung

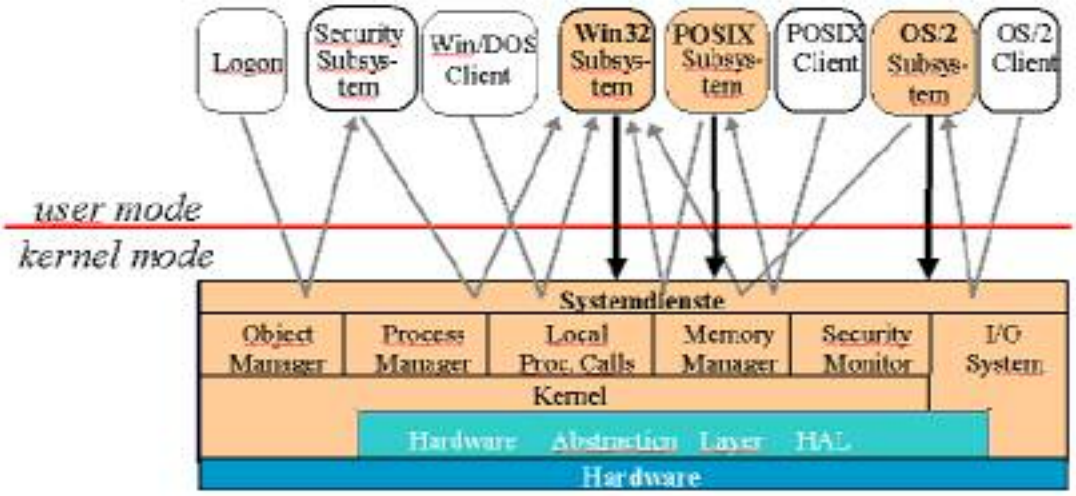
mind. eine 486 CPU, 16 MByte RAM und eine 500 MByte Festplatte an. Es hat sich jedoch gezeigt, dass ein Arbeiten mit akzeptabler Performance erst ab einer Pentium 100 CPU, 64 MByte RAM und einer 1 GByte Platte möglich ist. Windows NT unterstützt die Prozessorplattformen Intel x86/Pentium, und DIGITAL Alpha AXP/21x64. Die Software für beide Plattformen ist auf der CD enthalten und dort in verschiedenen Verzeichnissen untergebracht. Windows NT kann Anwendungen, die für IBMs Presentation Manager (bis Version 1.3) sowie POSIX 1003.1 geschrieben wurden, ausführen.

Am weitesten verbreitet ist Windows NT auf der Intel-Architektur. Windows NT auf Alpha-Architektur wird meist dann eingesetzt, wenn man auf sehr hohe CPU- oder I/O-Leistung Wert legt. Die Alpha-Version wird jedoch nicht mehr weiterentwickelt.

Im Lieferumfang ist die Unterstützung der Protokolle TCP/IP, NetBEUI, IPX/SPX, DLC und AppleTalk enthalten. Windows NT enthält Telnet und FTP-Clients sowie in der Server-Version einen FTP-Server Dienst. Das Dynamic Host Configuration Protocol (DHCP) ermöglicht die dynamische Einrichtung und Verwaltung von TCP/IP Adressen. Windows Internet Naming Service (WINS) ordnet den TCP/IP Adressen Namen zu. Für den Administrator und Benutzer wird es dadurch leichter in einem TCP/IP Netzwerk zu arbeiten. Es ist ebenfalls leicht möglich Windows NT in ein NetWare Netzwerk zu integrieren. Der Client Service für NetWare (CSNW) ermöglicht den Zugriff auf die Datei- und Druck-Services eines NetWare 3.x Servers. Der Gateway Service für NetWare (GSNW) bietet Arbeitsstationen im Windows NT Server-Netzwerk den Zugriff auf NetWare Server.

Windows NT unterstützt Fernzugriffe mit den Protokollen NetBEUI, IPX/SPX und TCP/IP; dies kann über ISDN, X25 oder analoge Telefonleitungen realisiert werden. Als Client erlaubt Windows NT Unix über PPP/SLIP, NetWare, LanRovers, Windows 3.x, Windows95 sowie LAN Manager. Über diesen Remote Access Service (RAS) sind bis zu 256 gleichzeitige Verbindungen erlaubt. Mit RaRAS (Routing and Remote Access Service) bietet Microsoft einen Software-basierten Multiprotokollrouter für Windows NT Server 4.0. RaRAS unterstützt Routing von TCP/IP und IPX. Als Routing-Protokolle werden RIP und OSPF unterstützt, sowie statisches Routing. Bei der Authentisierung über PAP/CHAP greift RaRAS auf die Windows NT Domain-User-Authentisierung zurück. Unterstützt werden zudem RADIUS-Clients. Zentraler Bestandteil von RaRAS ist der Routing Table Manager. Hier werden die Routing-Tabellen verwaltet. Für Konfiguration und Management steht eine grafische Oberfläche zur Verfügung.

Windows NT ist zwar ein Multitasking-, aber im Gegensatz zu Unix kein Multiuser-Betriebssystem. Dieses Defizites hatte sich der amerikanische Softwarehersteller Citrix Systems angenommen. Bereits 1992 schlossen Citrix und Microsoft ein Abkommen über eine strategische Partnerschaft zur Entwicklung des Multiuser NT WinFrame. WinFrame ist die Grundlage für "Application Publishing", einem neuen Weg für moderne Client/Server-Architekturen im PC Umfeld. Die Grundidee von Application Publishing ist nicht neu. Das Prinzip erinnert in weiten Teilen an Unix basierende Netze mit X-Window Terminals. Bei Application Publishing laufen die Anwendungen nicht auf dem einzelnen Arbeitsplatz-PC, sondern auf dem Server. Der Arbeitsplatzrechner bekommt nur die Fensterdarstellung über das Netz zugespielt, braucht also weder installierte Anwendungen noch hohe Rechenleistung oder Speicherausbau.



Microsoft Windows 2000, bisher unter der Bezeichnung Windows NT 5.0 bekannt, wird um etliche neue Eigenschaften und Funktionen erweitert. Dazu gehören die Bereiche Administrierbarkeit, Skalierbarkeit und Erweiterbarkeit sowie Storage und Hardware Management. Microsoft wird Windows 2000 wie NT in drei Versionen anbieten: Windows 2000 Professional entspricht der Windows NT Workstation, Windows 2000 Server dem NT Server und die NT Enterprise Edition wird als Windows 2000 Advanced Server weitergeführt.

Microsoft Windows 2000 implementiert Active Directory als zentrale Plattform, die den Zugriff und Management auf Netzwerk- und Systemressourcen vereinfacht. Weitere Features sind ein zentralisiertes Konfigurationsmanagement und die konfigurierbare und erweiterbare Microsoft Management Console (MMC).

Windows 2000 unterstützt max. 4 GByte physischen Speicher, bei 64-bit CPUs (Digital Alpha, Intel Merced) können 32 GByte adressiert werden. Mit

dem Microsoft Cluster Server können zwei Server im Verbund arbeiten. Dabei überwachen sich die Geräte gegenseitig um bei einem Ausfall eines Servers ohne Unterbrechung den Betrieb aufrecht zu halten. Während dem normalen Betrieb können die Server die Arbeitslast untereinander aufteilen, um eine höhere Produktivität zu erreichen.

Das Dateisystem NTFS implementiert nun auch eine Quotierung, mit der den Benutzern der maximal zur Verfügung stehende Plattenplatz festgelegt werden kann. Die NTFS-Erweiterung EFS (Encryption File System) ermöglicht die Verschlüsselung sensibler Daten auf Datei- oder Directoryebene. Wegen der Exportbeschränkungen für Kryptoverfahren handelt es sich aber um sogenannte "schwache Verschlüsselung".

Plug-and-Play hält nun auch bei Windows 2000 Einzug. Dies ermöglicht dann auch den problemlosen Betrieb von PC-Cards in mobilen Rechnern. Zusätzlich soll durch Erweiterung des Windows Driver Models (WDM) erreicht werden, dass in Windows 98 und Windows 2000 identische Treibersoftware zum Einsatz kommen kann.

Unix (und Derivate)

Unix ist eines der ältesten Betriebssysteme (UrUnix 1969). Es ist in vielen Eigenschaften beispielgebend für andere Systeme gewesen (Auch DOS, Windows und OS/2 haben bei UNIX "abgeschaut"). Inzwischen ist es für nahezu jede Hardwareplattform verfügbar. Eigentlich muss hier von einer Betriebssystemfamilie gesprochen werden, denn praktisch jeder Workstationhersteller liefert sein eigenes Unix aus, das sich zumindest in der Benutzerschnittstelle sehr unterscheidet. Es gibt hier allerdings eine Tendenz, die Vielfalt an Oberflächen zu überwinden, da einzelne Hersteller angefangen haben, ihr System auf Fremdarchitekturen zu portieren. Die Unix-Implementationen lassen sich in zwei Standards zusammenfassen: Berkeley Unix (BSD) sowie AT&T's System V Release 4 (SVR4). Letzteres ist momentan dabei, den Vorrang zu gewinnen. Neu entstehende Unix Versionen folgen diesem Standard. Im allgemeinen gilt: ist ein Programm für einen der beiden Standards geschrieben, so lässt es sich ohne allzugrosse Probleme auf ein anderes System des gleichen Standards portieren. Auch bei den verwendeten Benutzeroberflächen (GUI - Graphical User Interface) gibt es unterschiedliche Standards. Die neueren folgen aber alle der X11 Definition. Seit einigen Jahren klar auf dem Vormarsch ist die - ebenfalls auf X11 basierende - MOTIF Definition. Mehr und mehr Unix Implementationen bedienen sich dieser Oberfläche, während Konkurrenten wie OPENLOOK eher rückläufig sind.

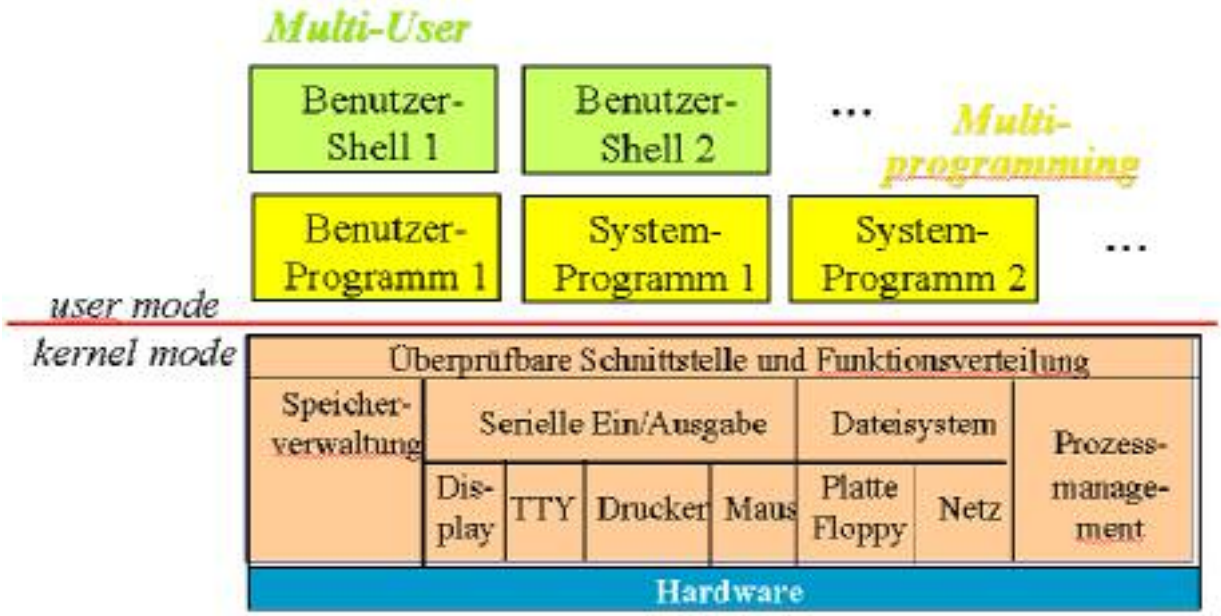
Die wichtigsten Eigenschaften in Kürze:

- UNIX ist ein portables, einfach aufgebautes Betriebssystem
    - Multitasking-BS (Multiprocessing-BS)
    - Multiuser-BS (Mehrbenutzer-BS)
    - dialogorientiert
- UNIX ist ein Werkzeugkasten
    - viele hundert Dienstprogramme
    - flexibel: kleine Tools sind schnell erstellt
- UNIX ist geeignet für Mikrocomputer der Oberklasse, Mini-Computer, Großrechner
- UNIX ist für Puristen
    - Befehlsname sind kryptisch (ls, pwd, cat, awk, grep, ..)
    - Motto 1:"Keine Nachricht ist eine gute Nachricht!"
    - Motto 2: "Wer will schon schlechte Nachrichten?"
- mit grafischer Oberfläche (X Window) bedienbar wie Windows
- UNIX ist in Schichten strukturiert
    - Shell (Kommandointerpreter) mit mächtiger Scriptsprache
    - Kern
    - Treiber
- Aufgaben des Kerns
    - Prozeß-Scheduling
    - Prozeß-Umschaltung
    - Prozeß-Kommunikation
    - Dateisystem verwalten
    - Ein-/Ausgabesteuerung
    - Gerätesteuerung (device driver)
    - Zugangskontrolle und Abrechnung
    - alle Systemdienste für Programmier-Schnittstellen
- Das Dateisystem ist hierarchisch strukturiert

- lange Dateinamen
- Normale Dateien (normal files)
- Verzeichnisse (directories)
- Spezialdateien (special files) = Geräteschnittstelle
- Named Pipes
- Links
- Jede Datei besitzt 12 voneinander unabhängige Schutzbits

Netzwerkbetrieb über TCP/IP ist bei UNIX von Anfang an möglich gewesen. Die meisten Netzwerkdienste anderer Systeme basieren auf den UNIX-Diensten. Insbesonder die 'Internet-Connectivity', die jetzt in aller Munde ist, basiert auf UNIX. So ist es z. B. auch möglich, Platten anderer Rechner über das Netz einzubinden.

Mehr als 90% des Codes ist in C programmiert --> portabel. Für unzählige Applikationen ist auch Quellcode erhältlich (für eigene Anpassungen).



Linux

Linux ist ein frei verfügbares Multitasking- und Multiuser-Betriebssystem auf UNIX-Basis für Systeme mit Intel-Prozessoren. Erfunden wurde Linux von Linus Torvalds und weiterentwickelt von einer Vielzahl von

Entwicklern in aller Welt. Linux wurde von Anfang an unter die GPL, der General Public License gestellt. Es kann frei und kostenlos verteilt, eingesetzt und erweitert werden. Entwickler haben so Einblick in sämtliche Quellcodes und können dadurch sehr einfach neue Funktionen integrieren bzw. Programmierfehler schnell finden und eliminieren. Treiber für neue Adapter (SCSI Controller, Grafikkarten etc.) können dadurch sehr schnell integriert werden. Inzwischen hat Linux mit vergleichbaren UNIX-Implementierungen gleichgezogen - oft ist es sogar robuster und stabiler als kommerzielle Produkte.

Linux kann auf zwei verschiedene Arten bezogen werden: Alle benötigten Teile können kostenlos aus dem Internet geladen werden. Einfacher ist der Einsatz einer sogenannten Distribution, diese werden von verschiedenen Firmen angeboten und enthalten neben einer Vielzahl von Anwendungen ein Installationsprogramm, welches die Installation von Linux wesentlich vereinfacht. Zu empfehlen sind die Distributionen von RedHat und S.u.S.E.

Linux wird mittlerweile von mehreren Millionen Anwendern weltweit erfolgreich eingesetzt. Die Benutzergruppen reichen von privaten Anwendern über Schulungsfirmen, Universitäten, Forschungszentren bis hin zu kommerziellen Anwendern und Firmen, die in Linux eine echte Alternative zu anderen Betriebssystemen sehen.

UNIX Gut gelöst:	Verbesserungsfähig:

Windows Gut gelöst:	Verbesserungsfähig: