TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik | Institut Systemarchitektur | Professur Rechnernetze

WS 2013
LV Informatik-I für Verkehrsingenieure

4. Betriebssysteme

Dr. rer.nat. D. Gütter

Mail: Dietbert.Guetter@tu-dresden.de
WWW: wwwpub.zih.tu-dresden.de/~guetter/

Betriebssystem

Systemsoftware zur Grundausstattung eines Computers

- erbringt Dienstleistungen für den Computer-Bediener, z.B. grafische Nutzeroberflächen
- stellt Routinelösungen für Hardware-Probleme bereit
- erweitert Hardwareleistung und -Auslastung, z.B. durch Organisation von Parallelarbeit mehrerer Komponenten

→ bestimmt wesentlich Leistung und Komfort des Computers mit

Ein optimales Betriebssystem für alle Anforderungen gibt es nicht!
z.B. widersprechen sich (mitunter) Forderungen nach

- max. Zugriffsicherheit und min. Zugriffszeiten
- Echtzeitgarantie und maximaler Systemauslastung
- Komfort und Effizienz, Übersichtlichkeit und Detailoptimierung
- Systemoptimierung und -Weiterentwicklungsoptionen
- Systemoptimierung und Portabilität auf andere Systeme

Lader

Wie kann man eine vorgegebene Hauptspeicherbelegung realisieren?

z.B. durch Eingabe Binärkode über Bedienkonsole

→ zeitaufwendig, hohe Fehlerquote

besser

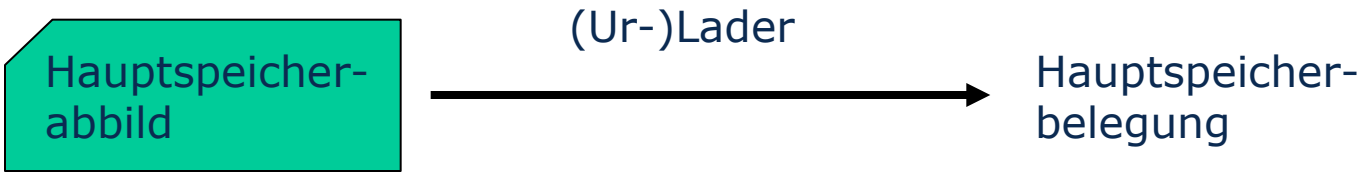
Einmalige Erzeugung
eines maschinenlesbaren Hauptspeicherbelegungplans (Abbild)
auf externem Datenträger

Laden des Hauptspeicherabbildes in den Speicher
durch Maschinenkodeladeprogramm

(Hauptspeicherabbild auf externem Datenträger mehrfach verwendbar)

(Ur-)ladeprogramme

(Ur-)lader
schaffen die Voraussetzung für einen Programmstart



Wie kommt der (Ur-)Lader in den Hauptspeicher ?

- meist EPROM-Speicher, (bei PC im BIOS)
- Urlader lädt i.d.R. nicht Anwendungsprogramme, sondern zunächst leistungsfähigere Systemsoftware (Betriebsssystem).

Betriebsystemlader
sind leistungsfähiger, z.B. können sie auf Dateien zugreifen.

Routineprobleme

Warum soll jeder Programmierer alles neu entwickeln ?

nachnutzbare Objektmoduln

- numerische Standardberechnungen (Trigonometrie usw.)
- …

hauptspeicherresidente Systemroutinen

- Nutzung der peripheren Geräte
- …
  (Gefahr: Systemroutinen können durch
  fehlerhafte Programme überschrieben werden)

Forderung

- Reduktion des Hauptspeicherzugriffsrechtes für Anwenderprogramme
- Privilegierung der Systemroutinen

z.B. PC-BIOS „Zeichenübernahme von Tastatur“

Entlastung Anwendungsprogrammierer

- Test Tastaturtyp
- Zuordnung der Zeichen zu Tasten (nationale Besonderheiten)
- Stellung der Umschalttasten wegen Mehrfachbelegung von Tasten
- Feststellen Tastaturprellen

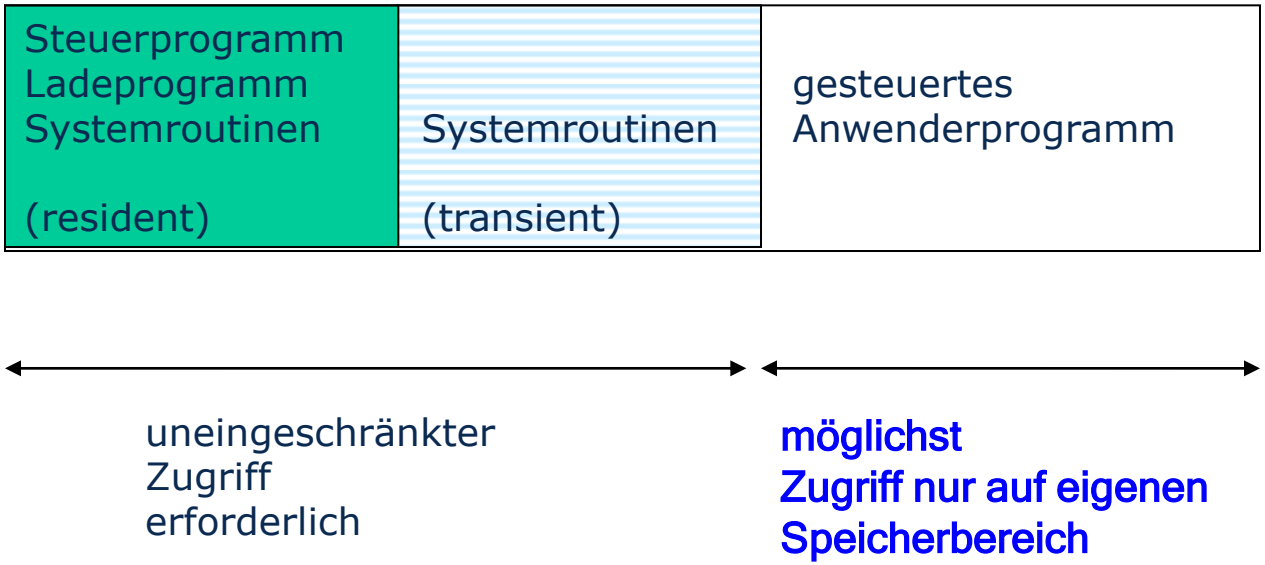
Maschinenkode-API

```
        MOV AH,0        ;Funktionsangabe fuer BIOS im Reg. AH
        INT 16H         ;Aufruf BIOS-Routine
```

BIOS prüft dann, ob eine Tastaturbetätigung vorliegt
und übergibt ggf. den ASCII-Zeichenkode im Register AL

Einfache Betriebssysteme

Hauptspeicheraufteilung



Unterstützung Systemanlauf

Systemanlaufroutine

- Laden der benötigten residenten Teile des Betriebssystems
- Initialisierung von Hardwarekomponenten,

  z.B. Test der Konfiguration
  Herstellen der Arbeitsbereitschaft...

- Initialisierung von Softwarekomponenten,

  z.B. definierte Belegung der Arbeitsdaten

Dienstleistungsprogramme

spezielle Anwendungsprogramme
zur Unterstützung des Rechnerbetriebes

- Hardwaretestprogramme
- Programme zur Wartung und Pflege von Datenträgern
- Programme zur Datenverwaltung
- Programmierunterstützungen.

exakte Abgrenzung
zwischen Dienstleistungsprogrammen und reinen Anwendungsprogrammen
nicht möglich

z.B.	Windows "Arbeitsplatz"	Dienstleistungsprogramm
	Windows "Editor"	Dienstleistungsprogramm ?

Steuerfunktionen

Probleme

- hoher Bedienaufwand
- Endlosschleifen in fehlerhaften Programmen
- irregulärer Programmlauf, z.B. unzulässige Hauptspeicherzugriffe
- irreguläres Programmende, z.B. „Division durch Null“

Lösung

- Steuerung/Überwachung des Laufes der Anwendungsprogramme
- in einfachen Betriebssystemen, z.B. MS-DOS, nur ansatzweise ab Computern der 3. Generation voll verwirklicht

Arbeitsgänge bei Entwicklung eines Programmes

Voraussetzung	Lader resident im Hauptspeicher

Start	Aufgabe
Lader	Laden Editor
Editor	Texteingabe/Ausgabe maschinenlesbarer Quellkode
Lader	Laden Übersetzer
Übersetzer	Eingabe Quellkode / Ausgabe Objektkode
Lader	Laden Verbinder
Verbinder	Umwandlung Objektkode (evtl. mehrere Moduln) in ein Anwender-Maschinenkodeprogramm
Lader	Laden Anwenderprogramm
Anwender-Programm	programmierte Aufgabe, evtl. fehlerhafte Ausführung
Ende	nach fehlerhafter Ausführung evtl. wieder von vorn

Dialogverarbeitung

System-
eingabe

Terminal

Steuerprogramm

Initialisierung

Zyklus

Lesen Steueranweisung (Kommando)
Laden Programmkode in Hauptspeicher
Aufruf

Anwenderprogramm
...
...
...
Rücksprung

Zyklus fortsetzen

Betriebssystem - Architektur

- Strukturierung des gesamten Betriebssystems
- Schnittstellen für Anwendungsprogrammierer
- Bedienerschnittstelle

Steigende Komplexität der Systeme

- hoher Aufwand für Entwicklung
- evtl. unerkannte Fehler

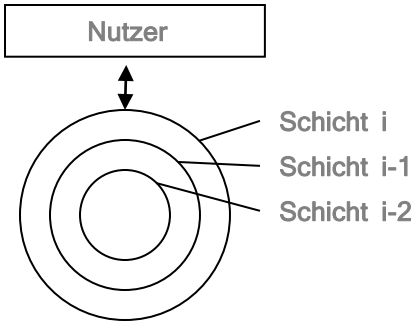
→ Einfache, übersichtliche Strukturen wünschenswert

- Entwicklung schneller und besser in Teamarbeit möglich
- einfachere und zeiteffizientere Testphase des Gesamtsystems
- Senkung des Wartungsaufwandes, leichtere Schulung
- hohe Änderungsfreundlichkeit für Systemweiterentwicklung

Betriebssystem - Diensthierarchien

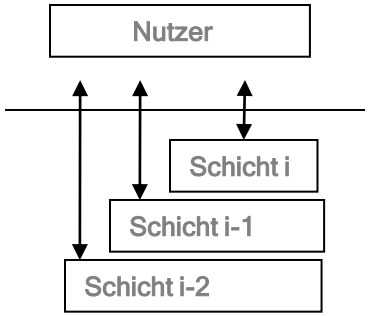
Entwurf eines Betriebssystems

- Aufteilung Gesamtsystems in einfachere Subsysteme
- Subsysteme als hierarchische Schichten
- klare Definition der Dienstleistungen der Subsysteme und der Dienstnutzerschnittstellen



konsistente Schichtung
→ untere Ebenen sind transparent

klarere Struktur



quasikonsistente Schichtung
→ untere Ebenen sind nutzbar

evtl. höhere Effizienz,
nur in Sonderfällen sinnvoll

Betriebssystemkern

Anwendungsprozesse

- gesteuert vom Supervisor
- nichtprivilegierter Status
  (keine E/A-Befehle, HS-Zugriff nur auf programmeigenen Bereich)

Kern (Supervisor, Kernel)

- Zusammenfassung aller permanent erforderlichen BS-Komponenten
- privilegierter Status
  (Nutzung aller Maschinenbefehle, voller Zugriff auf Hauptspeicher)

- stabil (fehlerhafte Anwendungsprozesse können Kern nicht zerstören)
- schwierige Entwicklung des Betriebssystems (Testung, …)
  da alle Komponenten im privilegierten Status
  (fehlerhafte Komponenten → evtl. Zerstörung Betriebssystem)

Betriebssystem – grobe Schichtung

Betriebssystemoberfläche bzw. Bedienerschnittstelle

- hierarchisch oberste Betriebssystemschicht
- möglichst bedienungseffiziente Nutzung der Anwendungsdienste
- Erschwerung von Fehlbedienungen
- leichte Erlernbarkeit

Steuerungsfunktionen

- Prozeßverwaltung
- Betriebsmittelverwaltung

Hardwarezugriff

- Treiber (→ Kap 2.2)
- Hardwareanpassung

Prozesse ←→ Betriebsmittel

Prozess Lauf eines Programmes

- **Prozessverwaltung**
  - Starten, Überwachen, Beenden von Prozessen
  - evtl. Verwalten mehrerer zeitparalleler Prozesse (Multitasking)

Betriebsmittel Ressourcen des Computersystems, welche die Prozesse zur Arbeit benötigen

- **Hauptspeicherverwaltung**
- **Datenverwaltung**

Betriebsmittel

Ressourcen alle für den Lauf eines Programmes erforderlichen Voraussetzungen

reale BM

Wiederspiegelung der physischen Natur der Ressourcen, d.h. es werden keine Funktionen verborgen oder vorgetäuscht.

virtuelle BM

nur scheinbar vorhandene Ressourcen, z.B.

virtueller Hauptspeicher	(evtl. größer als realer HS)
virtuelle RAM-Disk	(Daten im HS; Zugriff wie auf Disk)
virtueller Koprozessor	(z.B. für Gleitkommaoperationen)
virtuelle Drucker	(Druck als Datei auf Festplatte)

Klassifizierung

Hardware-BM z.B Prozessor, Hauptspeicher und periphere Geräte	**Software-BM** z.B. Betriebssystemroutinen und Nachrichten
wiederverwendbare BM z.B. Geräte	**verbrauchbare BM** z.B. Nachrichten
entziehbare BM z.B. Prozessor	**nicht entziehbare BM** z.B. Drucker, Magnetband
nur ungeteilt nutzbare BM, (exklusive BM) z.B. Prozessor, Drucker	**geteilt nutzbare BM** z.B. Magnetplatte, Hauptspeicher

Verwaltung exklusiver BM

Absicherung der konfliktfreien Nutzung exklusiver Betriebsmittel

Zugriff auf BM nur über Betriebssystem **!**

z.B. parallel laufende Programme drucken → verstümmelte Drucklisten

BM-Vektor für jede Betriebsmittel-Klasse

- Anzahl und Identifikation der verfügbaren Einheiten
- Verweise auf spezielle BM-Eigenschaften
- Identifikation des nutzenden Prozesses bzw. Kennzeichen "frei"
- ggf. Liste der auf das BM wartenden Prozesse

Betriebssystemdienste für

- Anforderung von BM
- Nutzung (BM-spezifische Verwaltungsroutinen)
- Freigabe

Semaphortechnik

Semaphor

Datenstruktur mit Zählvariablen Z
Ankeradresse ADR (für Warteliste)

Bei Initialisierung wird Z i.a auf den Wert 1 gesetzt.

Funktion "**Anfordern BM** durch einen Prozeß"

- Z herunterzählen, falls
- Z = 0 Zuteilung BM an Prozeß
  Z < 0 Suspendierung Prozeß; Aufnahme in Warteliste

Funktion "**Freigeben BM** durch Prozeß"

- Z hochzählen, falls
- Z=1 BM als "frei" kennzeichnen
  sonst BM anderem Prozeß in Warteliste zuteilen

Hauptspeicherverwaltung

HSV bestimmt wesentlich Systemleistungsfähigkeit,
u.U. bringt Speichervergrößerung mehr als schnellerer Prozessor.

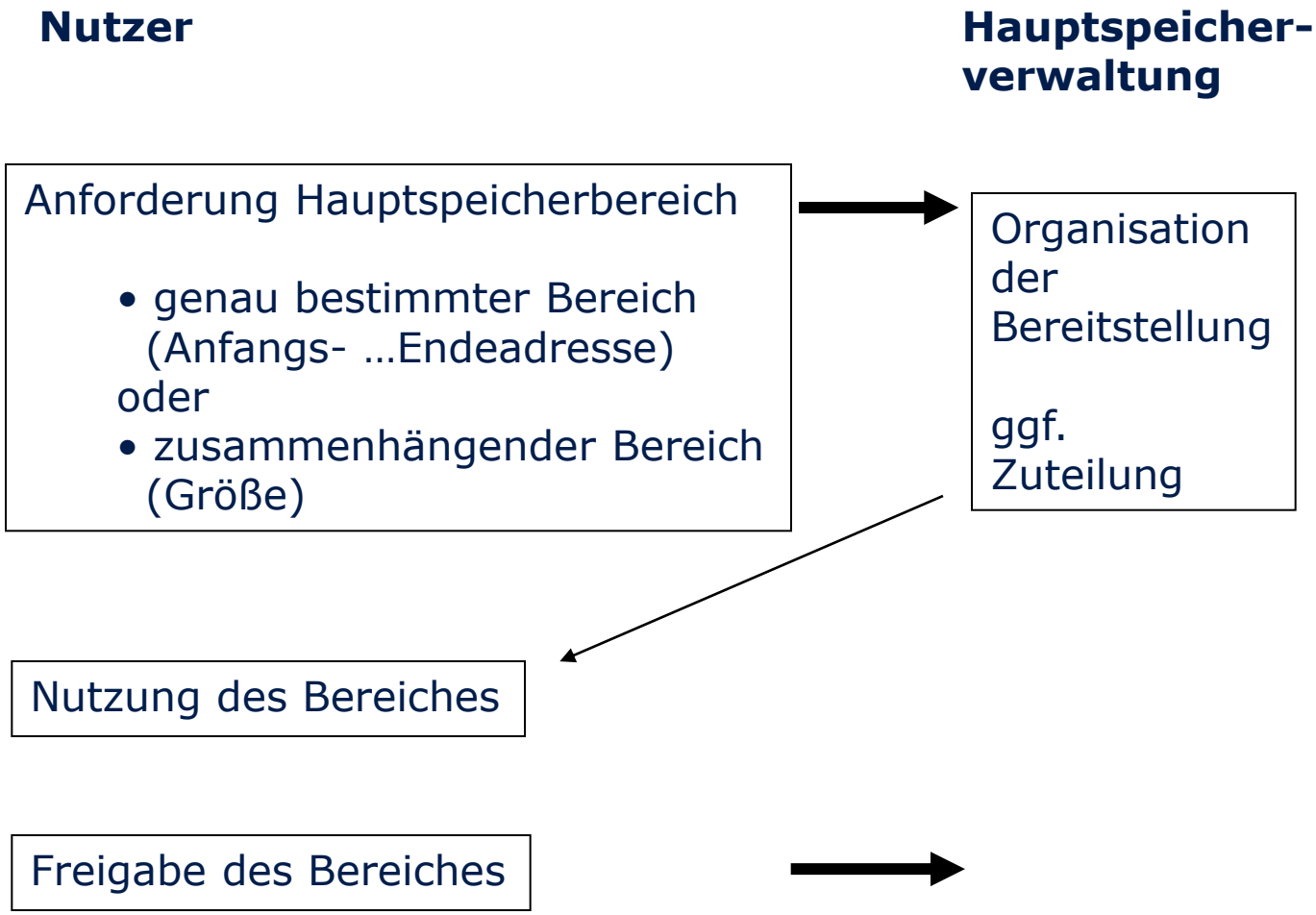
HSV unterstützt

- Laden von Treibern und Programmen durch Betriebssystem
- Nachladen von Daten durch Anwenderprozesse

Optimierungsziele

- vollständige Ausnutzung des vorhandenen Speichers
  sparsam zuteilen, Fragmentierung vermeiden
- Verwaltung mehrerer Speicheranforderungen
- Zugriffsschutz (s. Kap.2.2)
- Effiziente Implementierung
  schnelle Algorithmen, geringer Speicherbedarf

Zusammenwirken Nutzer - HSV



Speicherbelegungstabellen

zeigen an

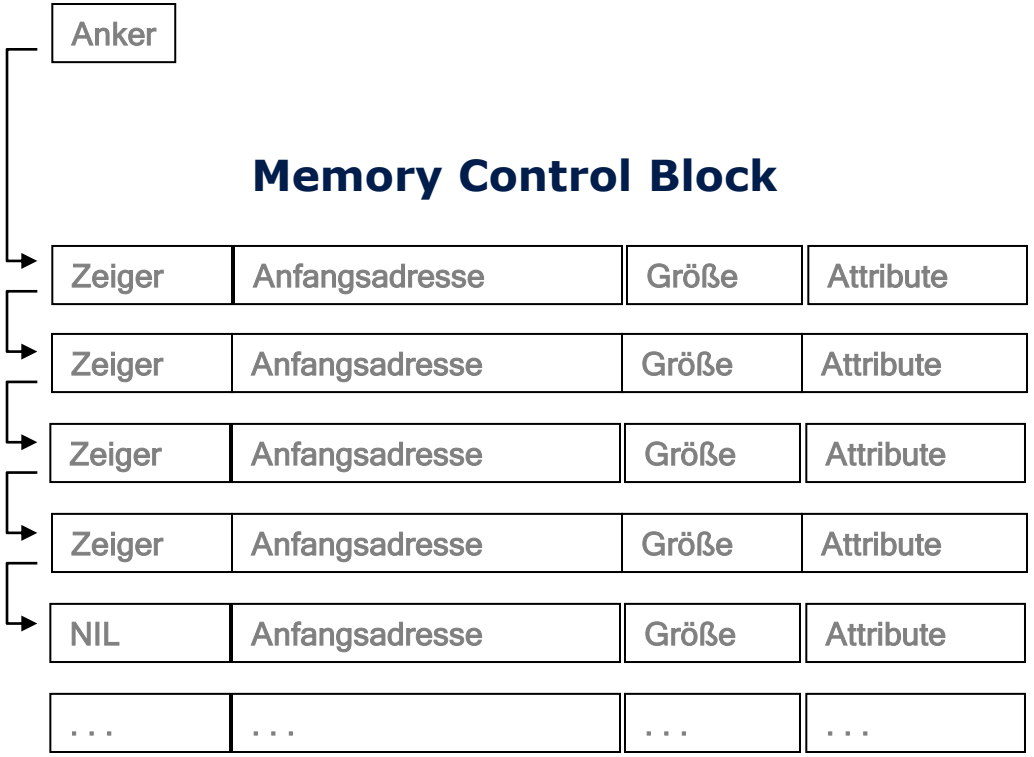
- welche Teile des Hauptspeichers frei sind
- welche Teile an Nutzerprozesse vergeben wurden

Realisierung

- Listen von Steuerblöcken
- Belegungsvektoren (Bit Map)

Hauptspeicher - Steuerblöcke

Verwaltung des Hauptspeichers für Treiber usw.



Hauptspeicher – Bit Maps

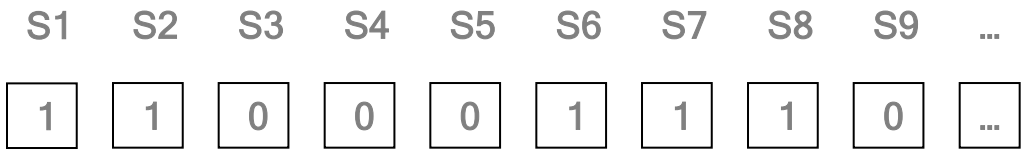
Hauptspeicher-Einteilung in (gleichgroße) Segmente

jedem Segment
wird in der Reihenfolge der Adressierung ein Bit zugeordnet

Vermerk frei/belegt

z.B. belegt: S1;S2;S6;S7;S8;…
frei: S3;S4;S5;S9;…

Bit Map



Fragmentierung

Hauptspeicheranforderungen können u.U. nicht befriedigt werden,
obwohl insgesamt ausreichend freier Speicher existiert,
weil dieser in viele kleine freie Speicherbereiche zerstückelt ist.

z.B.

Anforderung: zusammenhängender Speicherbereich für 7 Segmente

Speicherbelegung

frei:	31 Segmente		
zugeteilt:	16 Segmente		
Max. Freibereich:	5 Segmente	→	Anforderung nicht realisierbar

Vergabestrategien

z.B.

- schnelle Suche
  d.h. die Vergabe des ersten geeigneten Hauptspeicherbereiches
  nach sequentieller Suche in den Speicherbelegungstabellen

- optimale Suche
  d.h. die Vergabe des kleinsten geeigneten Bereiches
  (Vermeiden von Hauptspeicherzerstückelung)

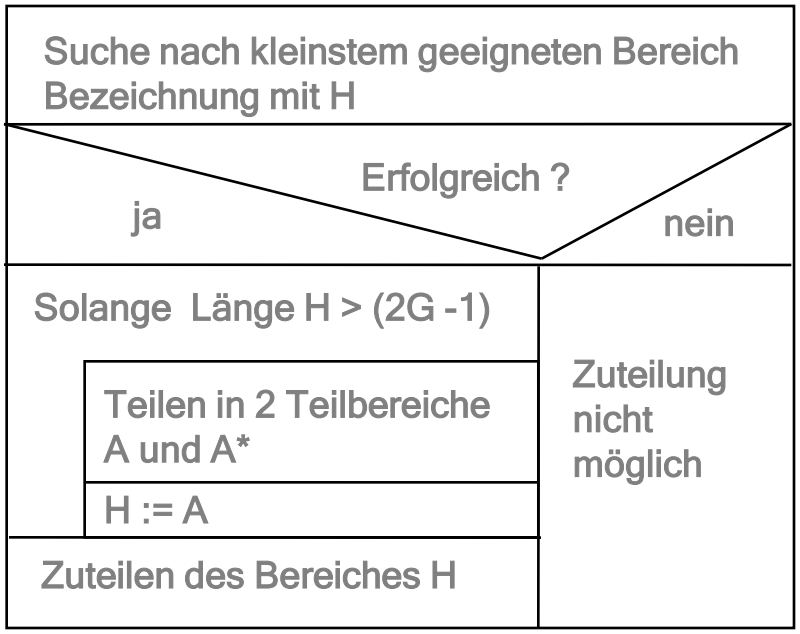
- stochastische Suche
  d.h. die Vergabe irgendeines geeigneten Bereiches

- Hauptspeicherteilung (Buddy-Technik)
  d.h. schrittweise Halbierung
  des Speicherbereiches bei Hauptspeicheranforderungen
  und Zusammenfassung freier Speicherbereiche bei Rückgabe

Buddy-Technik

anfänglich wird nur ein Speicherbereich verwaltet
bei Anforderungen schrittweise Teilung des Hauptspeichers,

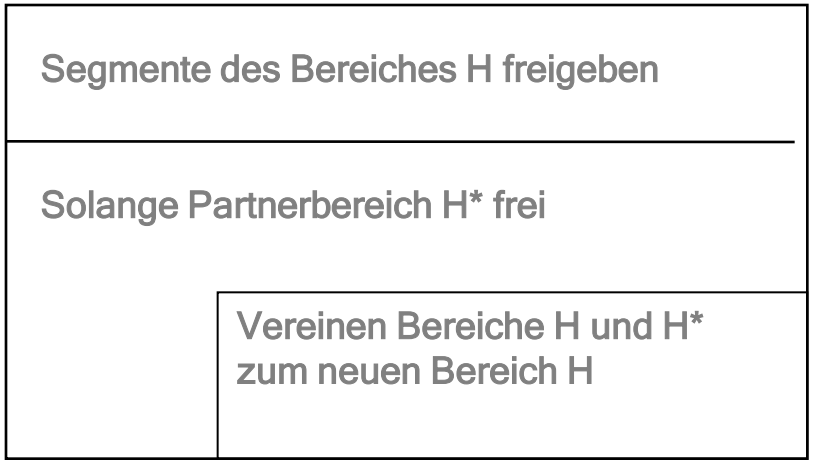
Struktogramm: Hauptspeicheranforderung der Größe G



Buddy-Technik (2)

bei Freigaben Zusammenfassung freier Speicherbereiche

Struktogramm: Freigabe Speicherbereich H

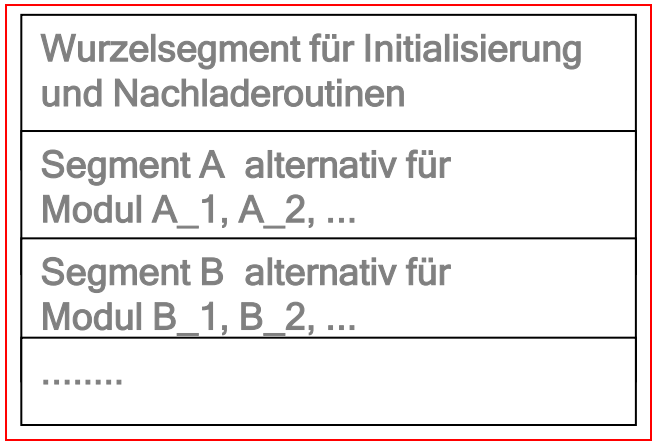


Überlagerungsstrukturen

Programmiertechnik für Programme mit zu großem Hauptspeicherbedarf

- Unterteilung Gesamtprogramm in Moduln, getrennte Übersetzung
- Festlegungen von Segmenten im Hauptspeicher
- Zuordnung von (evtl. mehreren) Moduln zu Segmenten

Laden der Moduln in Segmente zur Laufzeit,
organisiert durch Wurzelsegment und Betriebssystem



gute Planung schwierig

Ziel:

Reduzierung HS-Bedarf
ohne großen Laufzeitanstieg

Verwaltung eines virtueller Hauptspeichers

Hardware gewährleist bei modernen Prozessoren eine Unterstützung der **gestreuten Adressierung**

- Programmkode nutzt virtuellen Adressraum
- Abarbeitungsprozess im realen Adressraum
- Hauptspeicherverwaltung muß Zuordnung organisieren
  Seiten ←→ Rahmen

→ **Seitentabellen**
Zuordnungsverzeichnis der Rahmenadressen zu Seitenadressen

Seitennummer	Rahmennummer
. . .	. . .
. . .	. . .

Seitentabellen

Ermittlung Rahmenadresse muß(!) effizient organisiert sein

- Seitentabellen können sehr lang sein,
  da für jede Seitenadresse ein Eintrag existieren muß.

- Inverse Tabellen sind erheblich kürzer, da
  nur so viele Einträge erforderlich, wie Rahmen belegt sind.
  Allerdings ist die Zugriffszeit auf inverse Tabellen größer.

- **Optimierungsproblem**

  kleine Seiten → große Tabellen

  große Seiten → kleine Tabellen, aber
  viel Platzverlust durch „halbleere" Seiten

Seitenwechseltechnik - Paging

erforderlich, falls virtueller Speicher größer als realer Speicher

- Rahmen des Realspeichers anfänglich frei
  Seiten auf externen Datenträgern, meist Magnetplatten
- Laden Programmkode-Seiten in verfügbare Rahmen

- evtl. alle verfügbaren Rahmen belegt,
  dann Freimachen Rahmen mit einer "alten" Seite

- Algorithmen für Seitenwechsel werden sehr häufig ausgeführt,
  d.h. sie bestimmen wesentlich die Systemeffizienz
  (Abwägung Entscheidungsgüte gegen zeitlichen Aufwand)

  Wechselstrategien — älteste Seite (FIFO-Strategie)
  Seite mit geringster Nutzungshäufigkeit
  vorausschauend
  . . .

Seitenwechseltechnik - Zeitprobleme

nur Lesezugriffe auf „alte" Seite

- Rahmen können überschrieben werden, da Original verfügbar auf Platte

auch Schreibzugriffe auf „alte" Seite

- Seiteninhalt verändert, Rückschreiben auf Platte erforderlich, sonst keine Konsistenz
- Zugriffszeit auf externe Speicher >> Hauptspeicherzugriffszeit d.h.
  Paging nur sinnvoll, wenn Seitenwechsel relativ selten erfolgen.

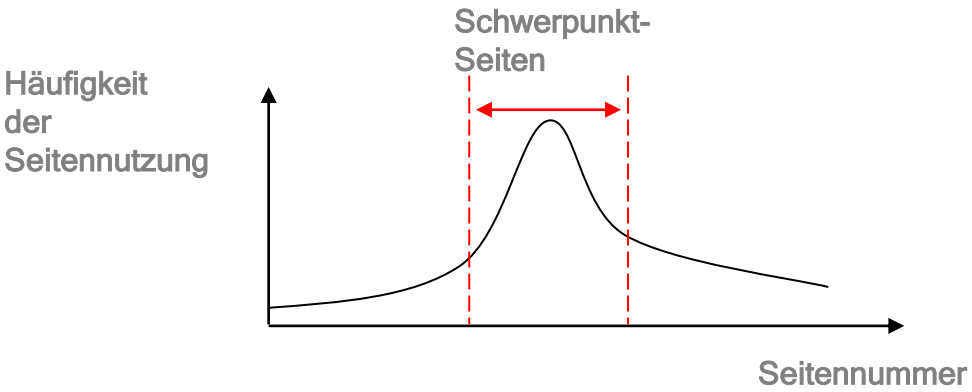
Seitenflattern

extrem häufige Seitenwechsel → extrem lange Programmlaufzeiten

Grund: schlechte Seitenwechselalgorithmen, zu kleiner Realspeicher
ungünstige Eigenschaften des Applikationsprogrammes

Minimierung Seitenwechsel

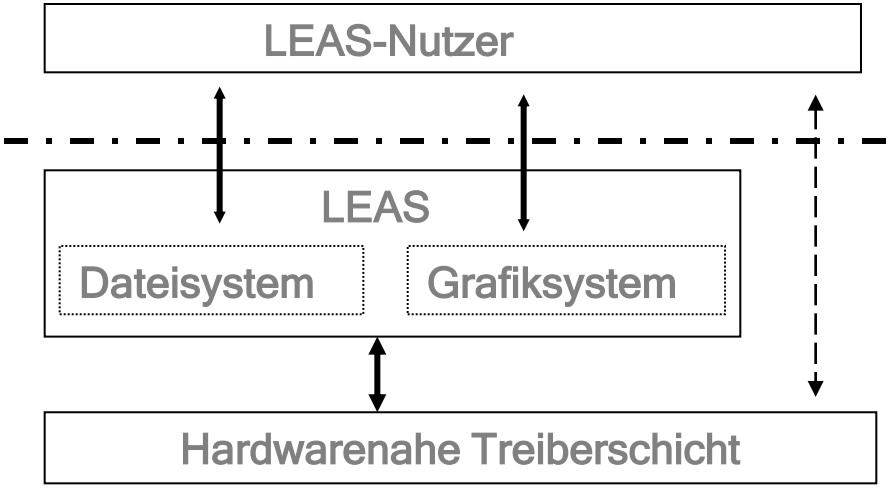
- häufigst genutzte Seiten möglichst permanent im Realspeicher
  Problem: Nutzungshäufigkeit nur statistisch vorhersehbar
- gute Programme konzentrieren Nutzung auf Schwerpunktseiten
- optimierende Compiler erforderlich



Datenverwaltung

Realisierung des E/A-Systems betriebssystemspezifisch,
Unterschiede betreffen Arbeitsteilung zwischen BS-Komponenten
sowie den Grad der Abschottung gerätespezifischer Funktionen

Oft Bereitstellung des Zugriffs auf zwei Anwenderschnittstellen
Treiber-Schnittstelle für spezielle Fälle
und LEAS, bestehend aus Dateisystem und Grafiksystem



Grafiksystem

Nutzerschnittstelle

- weitgehend einheitlich für Bildschim und Drucker
  Geräteumlegung dadurch vereinfacht

- verschiedene Modi
  Textausgabemodus (ASCII-Zeichen)
  Grafikausgabemodus (Pixel)
    - Farbauflösung, z.B. True Color (16 bit pro Pixel)
    - Pixelauflösung, z.b. (1024x768 bzw. 72 dpi)
    - installierte Zeichensätze im Grafikmodus (z.B. „Arial“, „18 dot“)

- Grafikroutinen für einfache geometrische Probleme
  Kreise, Linien, Hintergrundkachelung, Blinken

- Anpassung an unterschiedliche Auflösungen, Farbtiefen, …
  z.B. Umwandlung Farbbild in Schwarz/Weiß-Bild

Fenstertechnik

Fenster = Ausschnitt eines virtuellen Bildschirms,
i.a. mehrere Fenster auf realen Bildschirm

- Anwendungsprogrammierer kann Fensterinhalt ohne Rücksicht auf die anderen Fenster beschreiben.

- Bediener bestimmt Aussehen

  Fenster kann ganzen Bildschirm einnehmen
  Fenster kann auf Sinnbildgröße verkleinert werden
  mehrere Fenster können sich überdecken

- Grafiksystem verwaltet Fensterinhalte

  z.B. Retten von Inhalten verdeckter Fenster
  ggf. Wiederherstellen von Fensterinhalten
  wenn Fenster zu klein, Ausschnitt zeigen und Rollbalken

Dateisystem

Programmierung des Datenzugriff auf externe Geräte erfordert
auf Maschinenbefehlsebene detaillierte gerätespezifische Kenntnisse.

Dateisystem

- gewährleistet einheitlichen Datenzugriff auf alle externen Geräte
- erleichtert Anwendungsprogrammierung
  verkürzt Entwicklungs- und Testzeiträume

Datei

- Datenbestand, der über einen Dateinamen identifiziert wird
- Zugriff über Betriebssystemdienste nach Zugriffsmethoden,
  die von der Struktur der Daten abhängen

Dateizugriffsmethoden

Datenstrukturen

- unstrukturierte sequentielle Bytefolge
- Folge von Datensätzen

Zugriffsmethoden (für Lese- und Schreiboperationen)

- **sequentielle** Zugriffsmethode
  logische Bearbeitungsreihenfolge der Daten
  entspricht der physischen Anordnung der Daten

- **direkte** Zugriffsmethode
  Dateneinheiten der Datei sind eindeutig nummeriert.
  Bearbeitungsreihenfolge beliebig (über Angabe der Nummer)

- **...**

Dateizugriffsmethoden (2)

Auswahl der Zugriffsmethode

- i.a. durch die Anwendung bestimmt
- aber auch z.T. durch Geräte oder Betriebssysteme erzwungen
- direkter Zugriff auf Tastatur, Magnetbanddateien, … sinnlos
- Datenbanken nicht mit sequentiellen ZGM realisierbar
- UNIX, MS-DOS, MS-Windows, …

  alle Geräte werden bewusst im Sinne des einheitlichen Zugriffes als Dateien mit einer flachen Struktur angesehen.

  → nur sequentielle und direkte ZGM nutzbar

Dateisystem Aufgaben

- Bereitstellung einer Schnittstelle

  zur programmtechnischen Nutzung der Geräteperipherie
  Abstraktion von speziellen Geräteeigenschaften
  unabhängig von konkreter Hardware

- Bereitstellung abstrakter logischer Geräte für Programmierer

  z.B. Standardeingabe, Standardausgabe, Massenspeicher, Grafikausgabegerät

  Zuordnung zu physischen Geräte wird erst zur Laufzeit festgelegt.
  Diese Flexibilität ermöglicht ohne Programmänderung

  Gerätewechsel bei Havarien,
  Nutzung verschiedenartiger Geräte (Drucker, Festplatte),
  Programmlauf auf anderen Rechnern mit anderer Ausstattung

- Effizienzverbesserungen, wie interne Pufferung und Caching

Logische Geräte

- **Standardausgabe** (stdout)
  Programm-Ausgabe (i.a. zeichenweise)
  z.B. auf Drucker oder Bildschirm
- **Standardeingabe** (stdin)
  Bedienereingabe (i.a. zeichenweise)
  z.B. von Tastatur oder Magnetplatte
- **Fehlerausgabe** (stderr)
  Ausgabe von Systemfehlernachrichten
  evtl. auf dasselbe Gerät wie Standardausgabe
- **Daten-E/A, direkt**
  adressierbare Ein- und Ausgabe von Datensätzen
  z.B. bei Magnetplatte
- **Daten-E/A, sequentiell**
  sequentielle Ein- oder Ausgabe von Datensätzen
  Leserichtung festlegbar
  z.B. bei Magnetband

Dateien als logische Geräte

- logische Geräte als Datenstromlieferanten bzw. –konsumenten
  Zugriff (hardwareunabhängig) nach verschiedenen Methoden

    - sequentiell
    - Direkt

  Unterscheidung von Eingabe-, Ausgabe- und Arbeitsdateien

- Bezeichnung log. Geräte je nach Betriebssystem unterschiedlich

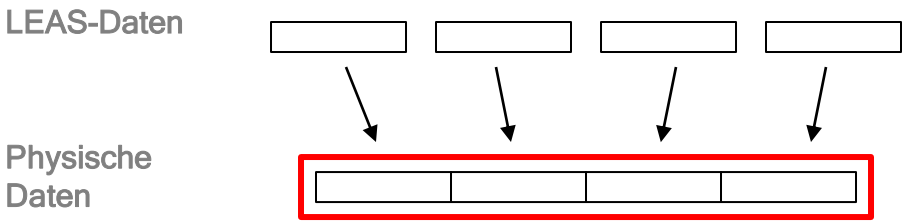
  CON, PRN, …
  STDIN, STDOUT
  SYS001, SYS002, …

- Zuordnung von logischen zu physischen Geräten durch
  Bedienerkommandos bzw. Steueranweisungen

Datenstrukturierung

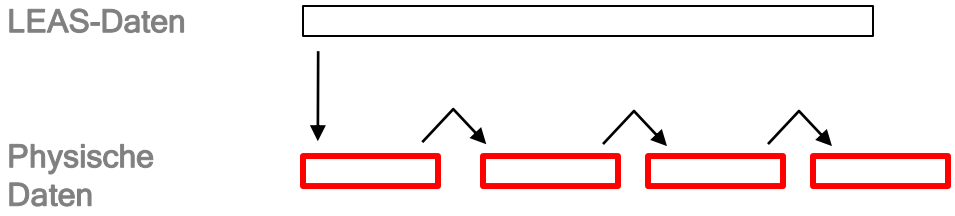
Blockung

Zusammenfassung mehrerer logischer Dateneinheiten des LEAS zu einer physischen Dateneinheit auf Datenträger



Segmentierung

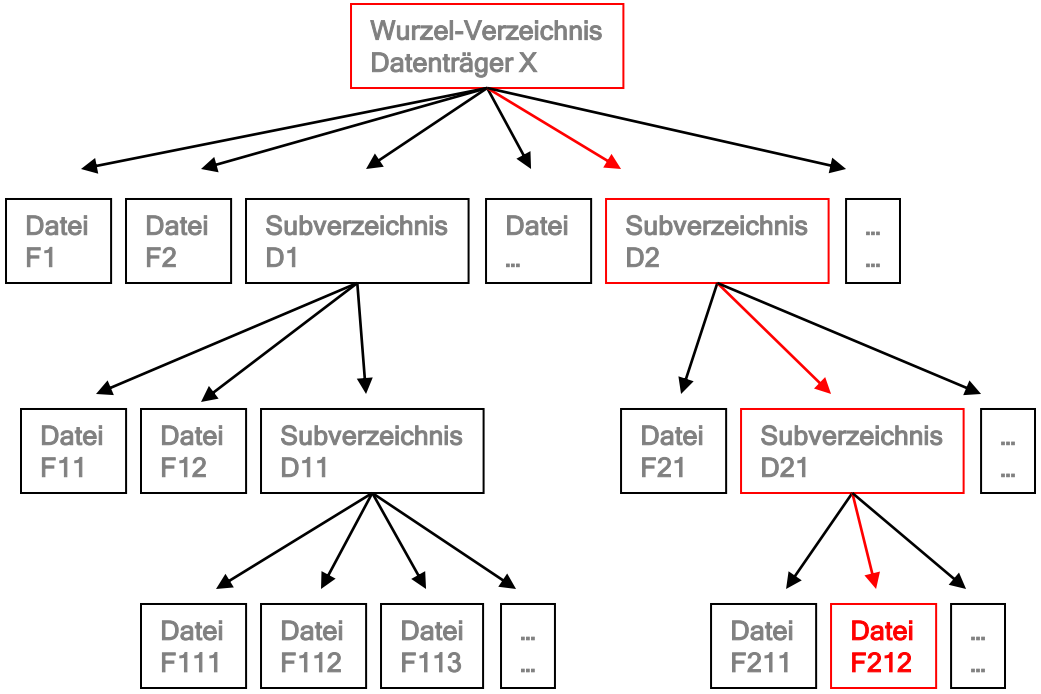
Zerlegung/Verkettung einer logischen Dateneinheit des LEAS in mehrere physische Dateneinheiten



(Unix), MS-DOS, Windows, …

Dateinamen müssen eindeutig sein

→ bei größeren Datenträgern Einteilung in Verzeichnisse sinnvoll



Dateikennzeichnung über Zugriffspfad\Dateiname, z.B. X:\D2\D21\F211

Ausgewählte Dateisysteme

MS-DOS, Windows, …

- **FAT** (File Allocation Table) für kleine Datenträger
  Datei-ID besteht aus Name und Typ (max. 8 bzw. max. 3 Zeichen)

- **NTFS** (New Technology File System)
  für leistungsfähige Systeme (ab Windows XP), unterstützt
  - (fast) unbegrenzt große Datenträger
  - lange Dateinamen
  - flexible Zugriffsrechte auf Dateiebene
  - Konsistenz nach Abstürzen

ext2, ext3 meistverwendete Linux-Dateisysteme
Leistungsmerkmale vergleichbar NTFS

NVFS (Non Volatile File System) Dateisystem für Palm-Rechner
Ziel: Verschleißreduktion beim Flash Memory

Zugriffsfunktionen

1. Verzeichnis Anlegen, Löschen, Auswählen, Dateisuche
2. bestehende Datei Löschen, Umbenennen

Bei diesen Funktionen

Inputparameter: Verzeichnis-, bzw. Dateiname(n)

Rückmeldung: O.K.

bzw. Fehlermeldung,
z.B. „falscher Name“

Dateisystem Handle

3. bestehende Datei Öffnen, neue Datei Erzeugen

Inputparameter:	Dateiname
interne Arbeit	BS legt Dateiverwaltungsblock an
	Nummer des Blockes ist das sog. **Handle**
	im Block „Bearbeitungszeiger auf Anfang“
Rückmeldung:	O.K. mit Handle
	bzw. Fehlermeldung, z.B. „falscher Name“

4. geöffnete Dateien - Eingabe, Ausgabe, Schließen, Positionieren

Inputparameter:	Handle, evtl. E/A-Adresse, …
Rückmeldung:	O.K. bzw. Fehlermeldung, z.B. „Dateiende“

Datei Schließen nach Beendigung der Dateinutzung erforderlich !

Abschluß z.B.	Rest Ausgabepuffer auf Magnetplatte schreiben
	und Dateiverwaltungsblock freigeben.

Datenträgerorganisation

Festlegungen zur Datenanordnung auf Datenträgern,
abhängig von Datenträgertyp und Betriebssystem

Lage und Inhalt von **Datenträgerkennsätze**

- Datenträgeridentifikation
- Nutzeridentifikation
- Informationen zur Speicherbelegung (Dateien und freier Speicher)

Aus diesen Kennsätzen ergeben sich dann
Position der Dateien und physischen Dateiblöcke auf Datenträger.
z.B.

Magnetband: Datenträgerkennsatz mit Band-/Nutzer-Name
gefolgt von einer Serie von Dateien,
(jeweils Dateikennsatz und mehrere Dateiblöcke)
Bereichstrennung durch spezielle Bandmarken
Bandende durch Doppelmarke gekennzeichnet

Datenträgerorganisation - Magnetplatten

Grundformatierung (-Initialisierung): vollständiges Beschreiben

- Datenträgerkennsatz — Besitzer, …, Grundstruktur
- evtl. Boot Record — Lader für Betriebsssystem
- Verzeichnisse — Informationen zur Datei-Lokalisierung, d.h. Dateinamen und phys. Zuordnung,
  evtl. Reservierungen freier Blöcke
  Erstellungsdatum,
  Angaben zu Zugriffsschutz-statistik

- i.a. mehrere Dateiverzeichnisse auf einem Datenträger
  häufig Unterstützung einer Verzeichnis-Hierarchie
- Dateien bzw. „freie" Blöcke

Datenträgerorganisation beeinflußt wesentlich

- durchschnittliche Zugriffsgeschwindigkeit auf Nutzdaten
- Wartungsaufwand (Reorganisation)

Dateisystem FAT (File Allocation Table)

Ursprünglich entwickelt für MS-DOS, heute Austauschformat

Sektoren	physische Datenblöcke (i.a 512 Byte)
Cluster	Zusammenfassung mehrerer Sektoren Numerierung (ab 1) ab Außenspur 0 im Zylinder 0
Partitionierung	Unterteilung Festplatte in mehrere logische Bereiche (in LV nicht weiter behandelt)

Datenträger

1. Sektor	Urladereintrag (**Boot Record**) für Formatinformationen und BS-Ladeprogramm (Laden unterschiedlicher Betriebssysteme möglich)
danach	**FAT**, evtl. zur Sicherheit mehrfach abgespeichert **Wurzelverzeichnis** **Cluste**r, frei oder belegt (Dateien/Subverzeichnisse)

FAT–12 1.Sektor

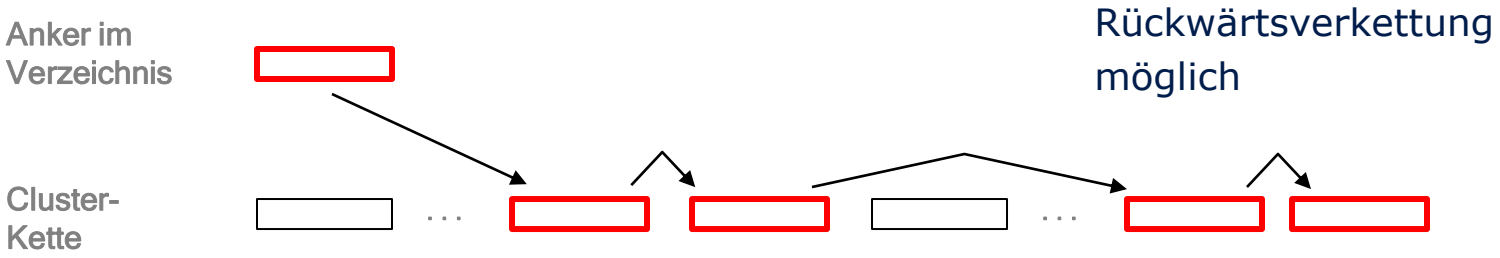
Position	Parameter	Inhalt	Bemerkung
00 - 02	Sprungbefehl zur Laderoutine	EB3C90	
03 - 0A	Name des Datenträgers	"SYSTEM01"	
0B - 0C	Bytes / Sektor	0002	512
0D	Sektoren / Cluster	01	
0E - 0F	Anzahl reservierter Sektoren	0100	1
10	Anzahl der FAT	02	
11 - 12	Anzahl Wurzelverzeichniseinträge	E000	14
13 - 14	Gesamtzahl der Sektoren (Begrenzung auf max. 32 MB)	400B	2880 ergibt 1,44 MB
15	Datenträgertyp	F0	Diskette
16 - 17	Sektoren / FAT	0900	9
18 - 19	Sektoren / Spur	1200	18
1A - 1B	Spuren / Zylinder	0200	2 (oben/unten)
1C - 1D	Anzahl verborgener Sektoren	0000	
1E - ...	Urladeprogramm	...	Maschinenbefehle

FAT–12 Verzeichnis

Position	**Inhalt**	Bemerkung
00	unterschiedlich meist (1.Zeichen Dateiname)	1. Ende Verzeichnis 05 o. E5 Eintrag frei (Datei gelöscht) 2E Subverzeichnis sonst (1.Zeichen Dateiname)
01 - 07	Rest des Dateinamens	u.U. mit Leerzeichen aufgefüllt
08 - 0A	Dateityp	u.U. mit Leerzeichen aufgefüllt
0B	Attribute	Bit 0...7 (ja=1, nein=0) Bit 0 Nur-Lesedatei Bit 1 versteckte Datei Bit 2 Systemdatei Bit 3 Datenträgername Bit 4 Subverzeichnis Bit 5 Backup nicht aktuell Bit 6,7 reserviert
0C - 15	reserviert	
16 - 17	Zeit des letzten Schreibzugriffs	
18 - 19	Datum der letzten Änderung	
1A - 1B	Nummer des 1. Dateiclusters	weiter Cluster-Kette in FAT
1C - 1F	Filegröße in Byte	

FAT–12

Position	Inhalt	Bemerkung
00	Datenträgertyp	s. Urladereintrag
01 - 02	FFFF	
03 - …	unterschiedlich	12-bit-Clustertabelle ab Byte 3



Spezielle Zeiger in FAT-12	Bemerkung
000	Cluster ist frei
002 - FEF	Nummer des Folge-Clusters innerhalb einer Datei
FF0 - FF6	reserviert
FF7	Cluster fehlerhaft
FF8 - FFF	letzter Cluster in Kette

FAT – Kritik , Verbesserungen, Alternativen

Anlegen/Löschen/… von Dateien
führen nach gewisser Nutzungsdauer zur Fragmentierung (Zerstückelung) des Speicherraumes, dadurch mehr Kammbewegungen, größere Zugriffszeiten, höherer Verschleiß

→ regelmäßige **Defragmentierung** des Datenträgers erforderlich

Längenbegrenzung der Dateinamen auf 8.3-Format

→ **VFAT** (Virtual FAT)
virtuelle Dateinamen bis 255 Zeichen
(1 Datei belegt mehrere Einträge im Verzeichnis)
interne Alias-Namen weiter im 8.3-Format

FAT-Systeme haben keinen Nutzerschutz

FAT - Begrenzung der Datenträgergröße

FAT-12 12-bit-Zeiger (4096 verschiedene Werte)
1 Sektor/Cluster (512 Byte/Cluster)
max. 2 MB Datenträgergröße (4096*512 Byte)

FAT-16 16-bit-Zeiger
max. 32 MB bei 1 Sektor/Cluster; 2 GB bei 64 Sektoren/Cluster
max. 2^{16} Dateien, viele kleine Cluster, hoher Verschnitt

FAT-32 28-bit-Zeiger
max. 8 TB bei 64 Sektoren/Cluster; max. 4 GB Dateigröße

→ gut geeignet als Transfersystem (USB-Sticks, …),
da von allen Betriebssystemen unterstützt
http://msdn.microsoft.com/de-de/windows/hardware/gg463084

exFAT kaum Begrenzungen, nicht abwärtskompatibel,
wird nicht von allen Betriebsssystemen unterstützt

Prozeß

Vorgang der Abarbeitung *eines* Maschinenprogrammes

- Prozesse werden durch das Betriebsystem verwaltet.
- Sie können unterbrochen werden.
- Eine spätere Fortsetzung muß das Betriebssystem garantieren.

Die Prozeßverwaltung des Betriebssystems ermöglicht die (Quasi-)Parallelarbeit mehrerer Prozesse.

Betriebsmodi

Stapelverarbeitung

- Automatisierte Auftragsbearbeitung
- Multiprozeßbetrieb; mehrere Aufgaben
- Ziel: bestmögliche Auslastung aller Systemkomponenten

Dialogverarbeitung

- interaktive Systeme
- Multiprozeßbetrieb; mehrere Nutzer; mehrere Aufgaben
- Ziel: kurze Reaktionszeiten; hohe Prozeßanzahl

Echtzeitverarbeitung

- zeitkritische Aufgaben
- Multiprozeßbetrieb; schnelle, flexible Prozeßumschaltung
- Ziel: Echtzeitgarantie, Zuverlässigkeit

Zeitparallele Abarbeitung von Programmen

prinzipiell möglich,
ohne wesentliche Laufzeiterhöhung der einzelnen Programme

Betriebssystem muß dazu

- jedem Programm einen virtuellen Prozessor bereitstellen
- und konfliktfreie Nutzung der Systemressourcen gewährleisten

Prozeß zeitlich begrenzter Vorgang der Abarbeitung eines Programmes auf einem virtuellen Prozessor

echte Parallelarbeit von Prozessen,
falls jedem virtuellen Prozessor ein realer Prozessor zugeordnet ist

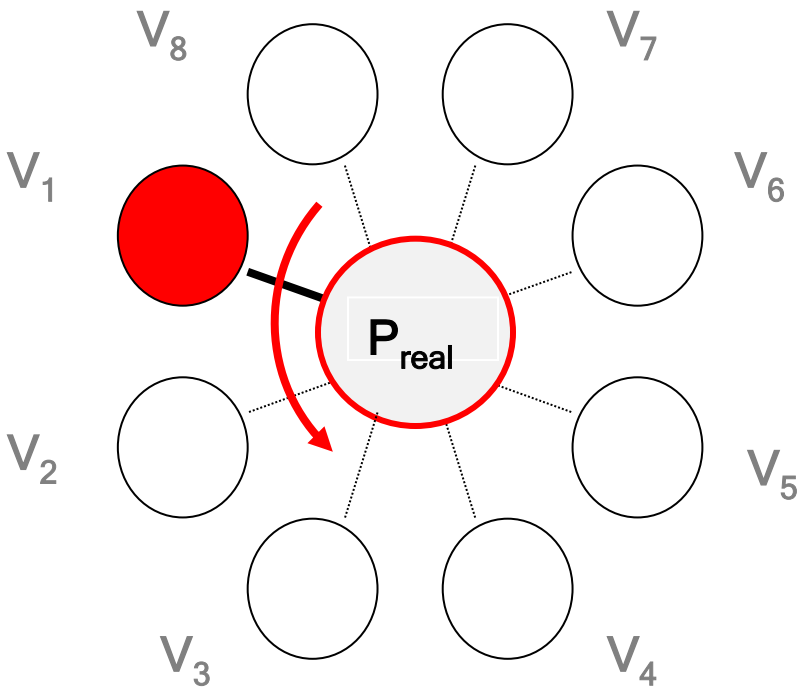
bzw.
Quasiparallelität, falls nur ein realer Prozessor verfügbar ist

- Einem virtuellen Prozessor wird realer Prozessor zugeordnet.
- Betriebssystem organisiert regelmäßige Prozeßumschaltungen, damit alle Prozesse ablaufen können.

Beispiel

Betriebssystem organisiert für 8 Prozesse
eine zeitzyklische Prozeßumschaltung (Zeitscheibe)

Ergebnis: Jeder Prozeß hat einen virtuellen Prozessor
mit mindestens 1/8 der Leistung des realen Prozessors.



Prozeß

BS kann Prozesse erzeugen, abbrechen, anhalten, fortsetzen,
ein Prozeß wird erzeugt (initiiert bzw. kreiert) durch

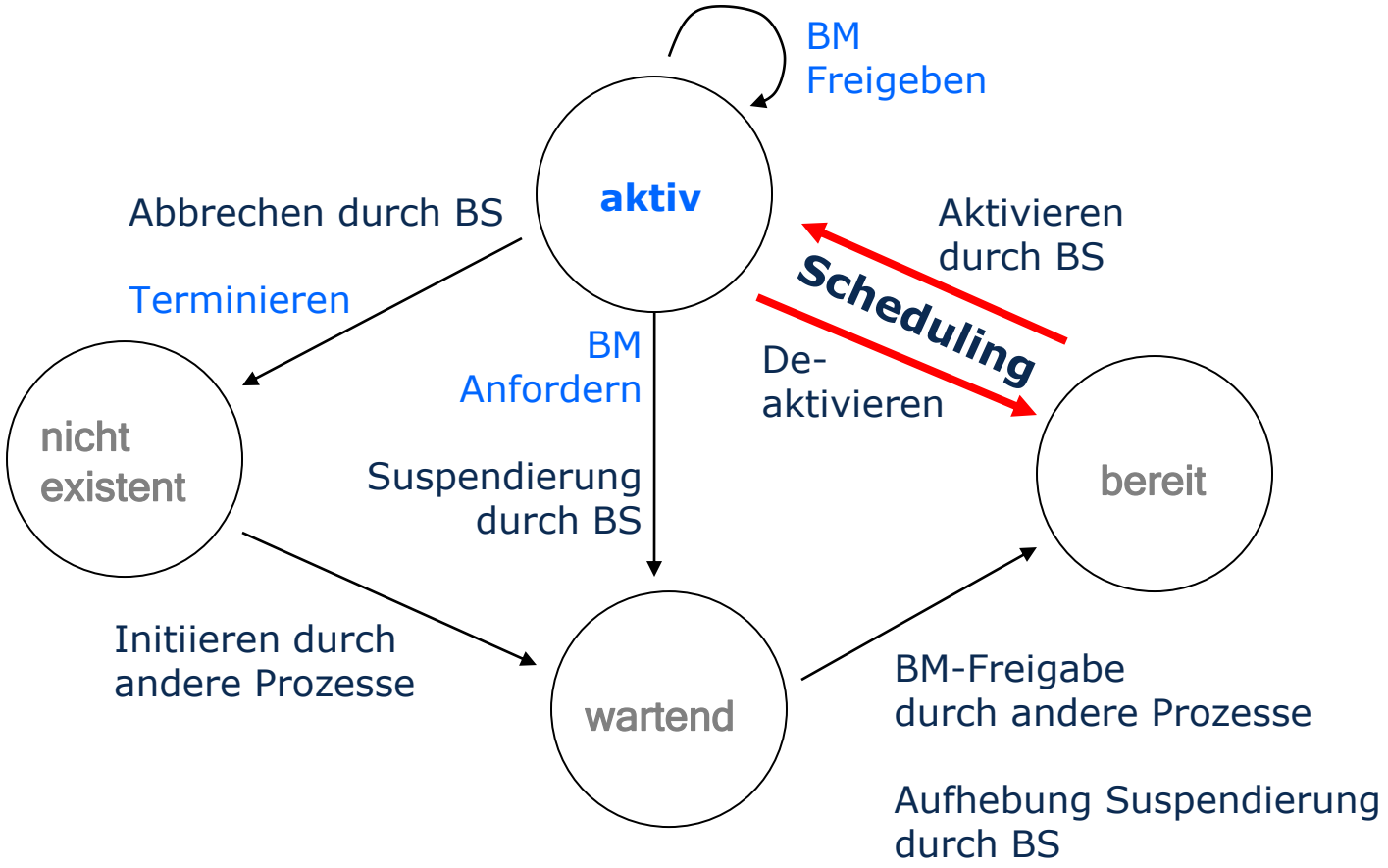
- Bereitstellung eines virtuellen Prozessors,
  des erforderlichen Hauptspeichers
  und evtl. weiteren Ressourcen
- Laden des Programmkodes und
  Bereitstellung von Informationen in Prozeßumgebung
  (z.B. Schalter, Dateinamen)
- durch Anlegen eines Steuerblockes zur Prozeßverwaltung

Prozesse

- arbeiten Programmschritte ab (nichtprivilegiert)
- können BS-Dienste nutzen, Betriebsmittel anfordern, besitzen, freigeben
- können jederzeit durch BS unterbrochen werden (Entzug BM Prozessor)
- können mit Hilfe von BS-Diensten mit anderen Prozessen kommunizieren
- können weitere Prozesse erzeugen (kreieren)
- müssen nach Erfüllung ihrer Aufgaben terminieren (Existenz beenden)

Prozeßzustände

Prozesse befinden sich aus BS-Sicht immer in einem der Zustände, zwischen denen sie während ihrer Existenz wechseln.



Prozeßzustandsübergänge

Prozeßerzeugung → „wartend“

Zuteilung aller BM (außer Prozessor) → „bereit“

Prozeßumschaltung (Auswahl eines aktiven Prozesses)
Scheduling „bereit“ ← → „aktiv“

aktiver Prozeß fordert erfolglos Betriebsmittel an
BM sind z.B. Geräten, Meldungen (z.B. „E/A-Ende“), ….
→ „wartend“

Freigabe Betriebsmittel durch Prozesse bzw. externe Ereignisse
→ „bereit“

BS kann BM entziehen (Suspendierung)
z.B. Wechsel aktives Fenster – Tastatur/Maus-Entzug
→ „wartend“

Prozeß beendet sich selbst
bzw. BS beendet Prozeß → „nicht existent“

Schedulingalgorithmen

Prozeßumschaltungsalgorithmen bestimmen wesentlich Systemeffizienz

Aktivierung des Prozesses mit der höchsten Priorität
Jedem Prozeß wird eine Priorität zugeordnet (nicht trivial!).

- E/A-intensive Prozesse → hohe Priorität
- rechenintensive Prozesse → niedrige Priorität
  andernfalls im Extremfall keine Prozeßumschaltung
- Prioritäten bestimmen Echtzeiteigenschaften!

Aktivierung nach Zeitscheiben

- Zeitüberwachung des aktiven Prozesses (HW-unterstützt)
- Suspendierung nach Ablauf einer Zeitscheibe
- Zeitscheibengröße beeinflußt Prozeßpriorität

Mischstrategien
z.B. variable Zeitscheibengröße je nach Rechenintensivität

Scheduling

Vergabealgorithmus für BM
im Falle konkurrierender Anforderungen von exklusiven BM

Strategien

- FIFO (First In First Out)
  Bedienung der Anforderungen in der Reihenfolge des Eingangs
  meistverwendete Strategie
- LIFO (Last In First Out)
  zuerst Bedienung der letzten Anforderung
- Vergabe nach Prioritäten
  z.B. bei zeitkritischen Vorgängen
  Prioritätenvergabe ist nicht trivial
- stochastische Vergabe
- Mischstrategien

Prozeßzustandsblock PCB

für alle Prozesse Prozeßzustandsblock angelegt mit aktuellem Zustand

PCB muß alle Informationen enthalten,
die eine Fortsetzung unterbrochener Prozesse gestatten.

PCB-Parameter u.a.	**Bemerkung**
Name des Prozesses	zur Identifikation
Globalzustand des Prozesses	aktiv/bereit/wartend
Priorität	Besonders wichtig für Echtzeitsysteme
Betriebsmittelanforderungen	Verwaltung aller Prozeß-BM, bis auf Prozessor
Betriebsmittelzuteilungen	
Parameter für entzogene Betriebsmittel (Fortsetzungsparameter für nichtaktive Prozesse)	z.B. Stand des Befehlsadreßregisters bei Prozeßunterbrechung Registerinhalte des Prozessors Bildschirmfensterinhalte ausgelagerte Programmkomponenten Informationen über externe Speicher

Aktivierung Scheduler

Scheduler muß für Prozeßumschaltungen selbst Prozessor besitzen.

Scheduleraktivierung durch folgende Ereignisse möglich:

- **aktueller Prozeß ruft Betriebssystemdienst auf**
  nach Prozedurende normalerweise kein Prozeßwechsel
  Trick:
  Erledigung von Scheduleraufgaben innerhalb BS-Routine, d.h.
  aktueller Prozeß wird in Bereitzustand versetzt (PCB-Eintrag),
  dann die Dienstaufgabe erfüllt
  abschließend Aktivierung eines (evtl. anderen) Prozesses
  aus der Menge der „bereiten" Prozesse

- **externe Ereignisse aktivieren Unterbrechungsbehandlung**
  durch Betriebssystem
  UBR analysiert neben UB-Behandlung auch den PCB
  und realisiert evtl. vor UB-Ende eine Prozeßumschaltung

Wechselwirkung zwischen Prozessen

unabhängige Prozesse

- Ressourcenkonkurrenz um Prozessorzeit
- Konflikte beim Zugriff auf exklusive Betriebsmittel
- Programmierfehler → evtl. Auswirkungen auf andere Prozesse

abhängige Prozesse

- Erzeugung von (Kind-)Prozesse , dabei entstehen Hierarchien
  Prozeßterminierung → alle Prozesse (Väter, Söhne, ...) beenden

- zeitliche parallele Arbeit an einer Aufgabe → Synchronisation

- gemeinsame Nutzung geteilt nutzbare BM
  evtl. Zeitbeeinflussungen und Datenkonsistenzprobleme

- Prozeßkommunikation bzw. Nachrichtenaustausch

Isolation von Prozessen

Probleme bei unbeschränktem BM-Zugriff der Anwenderprozesse,
evtl. unerwünschte Störungen bei fehlerhaftem Prozeßlauf

- z.B. Verfälschen von HS-Bereichen anderer Prozesse
  Fehlersuche schwierig, falls Ursache nicht im abstürzenden Prozeß

- **Isolation der Prozesse bzgl. HS-Zugriff**, …
  (BS privilegiert, AP niederpriorisiert)
  - Prozesse können nur eigenen HS-Bereich nutzen
  - sonstige Zugriffe nur über Prozeßkommunikation des BS
    zeitaufwendig, weil keine Referenzparameter möglich

→ **Threads** leichtgewichtige Prozesse mit reduziertem HS-Schutz
Übertragung von Zeigern („call by reference") möglich

- effizient, aber nur nach sehr guter Testphase sinnvoll
- Thread-Kommunikation in verteilten Systemen nicht möglich

Kritische Abschnitte

Prozeßumschaltungen dürfen nur erfolgen,
wenn unterbrochener Prozeß später korrekt fortgesetzt werden kann.

Ist dies nicht garantiert, liegt ein kritischer Abschnitt vor,
z.B. bei „gleichzeitiger" Nutzung von Variablen

- Lesen - keine Probleme
- Ändern - **evtl.** Konsistenzproblem

Zeitpunkt	Thread A	gemeinsame Variable x	Thread B
0		x = 1	
1	y := x + 1		
2		x = 8	x := x + 7
3	x := y	x = 2	

Ergebnis abhängig vom Zeitverlauf(!), richtig wäre x = 9
Konsistenzverletzung trotz richtiger Programmierung(!) der Threads

Kritische Abschnitte

Sperrungen

BS muß Dienste bereitstellen,
mit deren Hilfe in kritischen Abschnitten eine Prozeßumschaltung verhindert werden kann.

z.B. Zugriffsschutz über Semaphore

Zeitpunkt	Thread A	gemeinsame Variable x	Thread B	Bemerkungen
0	x_request	x = 1		Sperrung durch A
1	y := x + 1			
2			x_request	Zugriff gesperrt für B Wartezustand
3	x := y	x = 2		
4	x_release			Freigabe durch A
5		x = 9	x := x + 7	B darf zugreifen
6			x_release	Freigabe durch B

Ergebnis unabhängig vom Zeitverlauf(!) immer richtig

Reentrante Prozeduren

Betriebsmittel „gemeinsam genutzte Prozeduren“
in geteilt genutzten Hauptspeicherbereichen

- Kode nur einmal im HS → HS-Einsparung
- evtl. Konsistenzverletzungen
  durch Nutzung gemeinsamer (prozedurinterner) Variablen

  Verhinderung

  - Prozeduren als kritische Abschnitte schützen

  - **Reentrante Programmierung** der Prozeduren
    (Wiedereintrittsfähigkeit, z.B. beim PC BIOS)

    dabei Variablen nicht in gemeinsamen HS-Bereichen,
    sondern ausschließlich in **prozeßeigenen Speichern**
    (z.B. Register, Stack, …)

Prozeßverklemmungen

Warten auf Ereignis, das nicht eintreten kann

Ursachen
- Programmfehler oder
- Zuteilungskonflikt bei exklusiven Betriebsmitteln

z.B. Zuteilungskonflikt

Zeit	Prozeß A	Prozeß B	Bemerkungen
	. . .	. . .	
	x_request	. . .	A erhält BM x
	. . .	y_request	B erhält BM y
	y_request	. . .	y-Zugriff gesperrt für A
	W	. . .	Wartezustand
	W	x_request	x-Zugriff gesperrt für B
	W	W	Wartezustand / Verklemmung
	. . .	. . .	
	y_release	x_release	
	x_release	y_release	

Auflösung Abbrechen eines beteiligten Prozesses

Vermeidung bereits bei Prozeßstart alle BM zuteilen

Prozesse unter Unix

baumartige Prozeßstruktur, jeder Prozeß mit Prozeß-ID PID

- PID 0 swapper wird bei Systemstart erzeugt
- PID 1 *init* Urvater aller anderen Prozesse

Dämonprozesse laufen im Hintergrund als Serverprozesse
Nutzerprozesse mit Dialogroutine *shell*, Kindprozesse bilden *session*

fork erzeugt Sohnprozeß, zunächst Kopie Vaterprozeß,
Sohn erbt geöffnete Dateien und Prozeßumgebung ,
(Prozeßumgebungsvariable für Informationsaustausch)
liefert PID des Sohnes
Vaterprozeß kann asynchron weiterarbeiten

wait Vater wartet auf Beendigung des Sohnprozesses

exec neuer Prozeß lädt Programmkode, danach Prozeßarbeit

exit Prozeßende, evtl. Informationsübermittlung an Vater

Unix, weitere Systemdienste

- Thread-Unterstützung, reentranter Programmkode
  (Shared Memory, Shared Library)

- Prozeßsynchronisation (Nutzung von Semaphoren)
  Signalaustausch zwischen Prozessen → (Softwareunterbrechung)
  SIGKILL für Prozeßabbruch beim Zielprozeß
  SIGSEGV für Prozeßabbruch bei verbotenen Speicherzugriffen
  SIGUSR1 für Prozeßsynchronisation
  …
- Prozeßkommunikation (mehrere Möglichkeiten)

  - Pipeline - Nachrichtenaustausch über unidirektionale Puffer
    (Sender schreibt in Pufferdatei, Empfänger liest)
  - Filter - spezielle Pipelines
    Standardausgabe Prozeß 1 → Standardeingabe Prozeß2
  - Berkeley-Sockets
    (später weiterentwickelt für Internet-Kommunikation, s.Kap. 6)

Automatisierte Kommandoabarbeitung

Stapelverarbeitung

- häufig Eingabe immer gleicher Kommandofolgen
  → Routine, hoher Bedienaufwand
- Programmierung von Kommandodateien
  (**Shellscripte**, Batchfiles)
- Betriebssystem liest Kommandos nacheinander aus Datei

Problem

- menschlicher Bediener handelt intuitiv (meist) richtig,
  (z.B. bei Anzeige von Übersetzungsfehlern wird
  Abarbeitung nicht gestartet, dafür Quellprogramm überarbeitet)
- Kommandodateien müssen Steueranweisungen enthalten
  für alternative Abläufe

Vorteil

- kurze Sitzungszeiten (weniger Fehlkommandos, schnelles Einlesen)
- zeitversetzte Arbeit möglich (nachts ohne Bediener, …)

Mehrnutzersysteme

Computertechnik

- anfänglich ökonomischen Gründe für Mehrnutzersysteme
  später Bedarf in Rechnernetzen

Einfache Lösung: Nutzer arbeiten nacheinander

- evtl. irrtümliche Verfälschung der Daten anderer Nutzer
- oder gar vorsätzliche Datenschutzverletzungen
- Veränderungen der Nutzeroberfläche u.a. Systemeinstellungen
- aufwendige Verwaltungsarbeiten an,
  z.B. Nutzerzulassung, Nutzereinlass,
  Rechenzeitplanung und -abrechnung

Besser: Betriebssysteme mit Nutzerverwaltung

- Rechteverwaltung
- Unterstützung zeitparalleler Arbeit

Nutzeradministration

Einrichten des Betriebssystems → Einrichtung eines privilegierten Nutzers

Administrator, Superuser

- uneingeschränkte Nutzerrechte
- hat Nutzerverwaltungsaufgaben
  - Anmeldung neuer Nutzer beim Betriebssystem
  - Zuordnung von Nutzerrechten zu den Nutzern
  - Abmeldung von Nutzern

Nutzerrechte erlauben bzw. beschränken

- Datenzugriff,
- Gerätezugriff
- Nutzungsumfang und -intensität.

Betriebssystem hat abzusichern

- Identitätsprüfung bei Sitzungsbeginn, evtl. Abweisung
- Verhinderung der Überschreitung von Nutzerrechten
- Abrechnung erbrachter Dienstleistungen

Praktische Nutzeradministration

Windows,über Systemsteuerung

Unix, i.a. Nutzung der Bedienkonsole

Die wichtigsten Nutzeradministrationskommandos sind

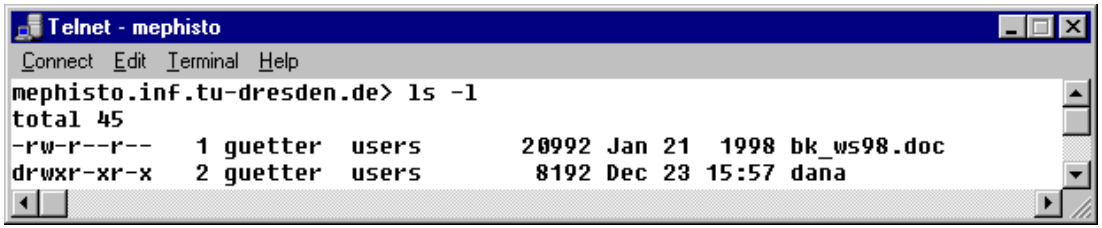
- login Anmelden eines neuen Nutzers
  initiales Paßwort, Gruppenzuordnung, Rechte,
  eigenes Nutzerverzeichnis
- su Erweitern der Nutzerzugriffsrechte
- newgrp Ändern Gruppennummers eines Nutzers
- chgrp Wechsel der Gruppenzugehörigkeit
  von Dateien oder Verzeichnissen
- chown Wechsel der Besitzerzugehörigkeit
  von Dateien oder Verzeichnissen
- sa Systemabrechnung
- ac Nutzerprotokollierung/-Abrechnung

Unix - Zugriffsrechte

Nach erfolgreichem Einloggen kann der Nutzer

- unbegrenzt auf seine Dateien zugreifen,
  auf andere Nutzerverzeichnisse nur gemäß seiner Rechte.
- den Zugriff anderer Nutzer auf seine Daten steuern

Kommando **ls** *-l* Anzeige Zugriffsrechte im Verzeichnis



Zugriffsrechtsangaben

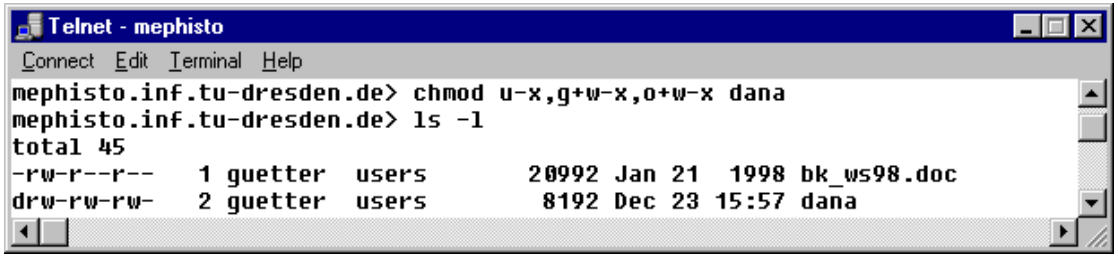
- 1. Zeichen ("-" oder "d") Anzeige Datei oder Verzeichnis
- danach Rechte von Nutzer, Nutzergruppe und Anderen,
  "rwx" (Recht Lesen, Schreiben und Programmausführung),
  bei Nichterteilung "-„
- weiterhin Anzeige Nutzername und Gruppenname

Unix - Zugriffsrechte

Kommando **chmod** Einstellen Zugriffsrechte im Verzeichnis

Beispielkommando in Abbildung bewirkt,
daß der Nutzer

- sich selbst das Ausführungsrecht entzieht
- der Nutzergruppe und den Anderen das Schreibrecht erteilt
- der Nutzergruppe und den Anderen das Ausführungsrecht entzieht



Betriebssystem

- führt vor jedem Dateizugriff eine Berechtigungsprüfung aus
- und verweigert ggf. den Zugriff

Unix - Nutzerkommunikation

Unterstützung der

- direkten Kommunikation zwischen angemeldeten Nutzern

  Kommando: **write** *guetter ttyp2*
  Hallo !
  ^C
  bewirkt Ausschrift „Hallo !“ am Terminal *ttyp2* des Nutzers *guetter*

- Kommunikation über BS-Mailboxen (Vorbild für Internet-Mailservice)

  Kommando: **mail** *guetter*
  Subject: *Test*
  Hallo !
  ^D
  Cc:
  bewirkt Abspeichern Botschaft in Mailbox des Nutzers *guetter*
  Nutzer *guetter* liest Botschaft durch Aufruf des Kommandos **mail**.

Sicherheit

Außerordentlich wichtig !

Da heute die meisten Computer in Rechnernetze integriert sind, haben sehr viele Menschen die technische Zugriffsmöglichkeit und könnten Fehler beim Datenschutz ausnutzen.

Folgende **Vorsichtsmaßnahmen** sollten generell ergriffen werden

- Vermeidung von Administrationsfehlern
  Nutzungsberechtigungen restriktiv vergeben
- regelmäßige Prüfung auf Virenbefall
- Wahl komplexer Passworte
  regelmäßiges Wechseln von Passworten
- Vermeiden von Fehlern beim Setzen von Dateizugriffsrechten

Portabilität von Anwendungsprogrammen

Sind Anwendungsprogramme auf unterschiedlichen Systemen lauffähig?

ja, wenn Systemübereinstimmung von Hardware, Betriebssystem, …

? bei Nichtübereinstimmung

Kompatibilität von

Binärkode — Maschinenkodeprogramme sollen portierbar sein
nur bei Rechnerfamilien möglich
z.B. Wintel, IBM Familie 360/370/390

Quellkode — *Portierbar durch Übersetzung auf neuem Zielsystem*
z.B. C-Programm entwickelt an IBM Workstation/AIX
→ PC/Linux

interpretative Abarbeitung des Quellkodes
z.B. Scriptsprachen (UNIX-Shellscripte, perl, php, …)

Virtuelle Maschinen

vorangetrieben durch **IBM**, um auf einem Wirtsrechner koexistente Arbeit mehrerer (IBM-)Betriebssystem zu erreichen

Plattform mit Mikrokern stellt virtuelle Rechner (VM) zur Verfügung

VM (virtuelle) Abstraktion eines IBM-Rechners mit eigenem

- virtuellem Prozessor
- virtuellem Hauptspeicher
- virtuellem Peripheriegeräten

Jede virtuelle Maschine kann eigenes BS laden

VMware

vergleichbare Lösung zur Bereitstellung virtueller Maschinen auf Rechnern mit x86-Architektur

Nutzung z.B. Reduktion der Anzahl der Serverrechner in Rechenzentren

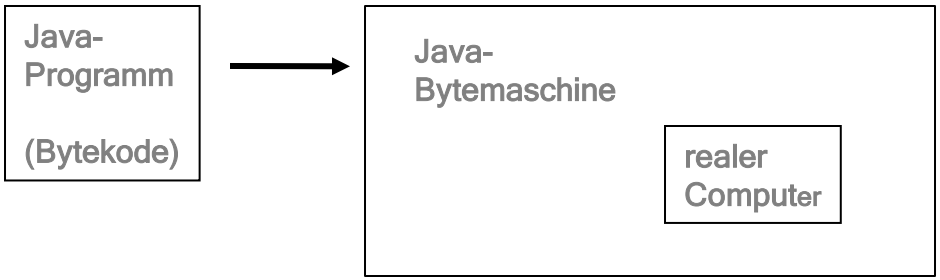
JAVA

Interpretative Abarbeitung von Quellkode

- hohe Portabilität, aber
- hoher Leistungsverlust gegenüber Abarbeitung im Maschinenkode

JAVA – Ansatz (nicht zu verwechseln mit Javascript)

- Quellprogramm in Sprache Java (objektorientiert, ähnlich C++)
- Übersetzung in Java-Bytekode
  (kompakter/effizienter als Quellkode, nicht maschinenabhängig)
- Interpretative Abarbeitung in virtueller Java-Bytemaschine
  (existiert für praktisch alle Plattformen)



Betriebssystem – Client/Server Architektur

(Mikro)-Kern

- nur unbedingt im privilegierter Status erforderliche Komponenten
- Hauptspeicherverwaltung
- Prozeßverwaltung mit Prozeßkommunikationsunterstützung
- elementare Hardwarebehandlungsroutinen

Betriebssystem- und **Anwendungsprozesse**

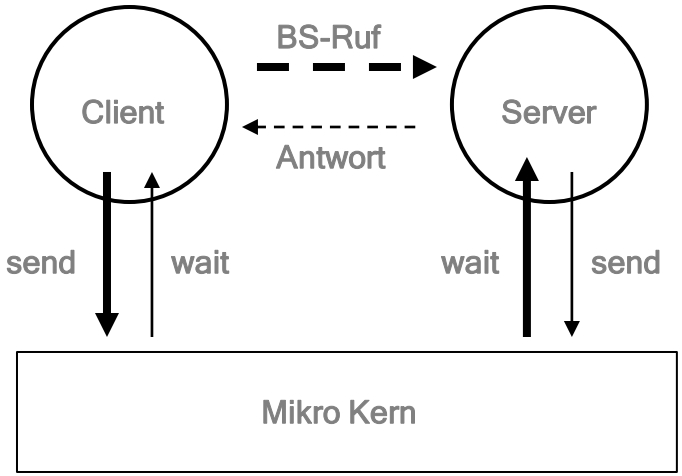
- gesteuert vom Kern, unprivilegierter Status

Betriebssystem

- stabil
  fehlerhafte Anwendungsprozesse können Kern nicht zerstören
- erleichterte Weiterentwicklung des Kerns (Testung, …)
  fehlerhafte Betriebssystemprozesse
  führen nicht zur Zerstörung des Kerns

Ablauf Betriebssystemruf

1. Anwendungsprozesse (Client) kodiert BS-Ruf in Nachricht,
2. sendet über Mikrokern an Betriebssystemprozeß
3. Betriebssystemprozeß empfängt Nachricht, führt BS-Ruf aus und
4. schickt Antwortnachricht zurück



Laufzeitverschlechterung gegenüber direktem Aufruf
getrennte Speicherräume Client/Server → Referenzparameter unmöglich