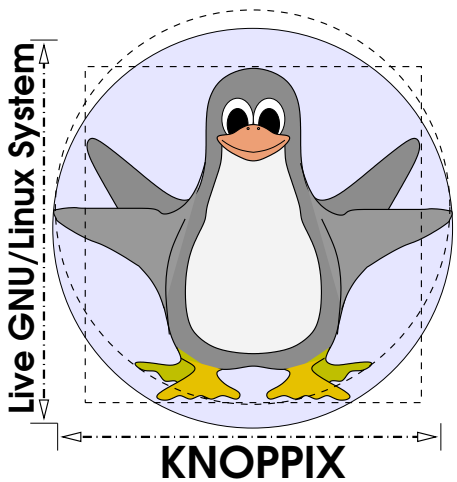
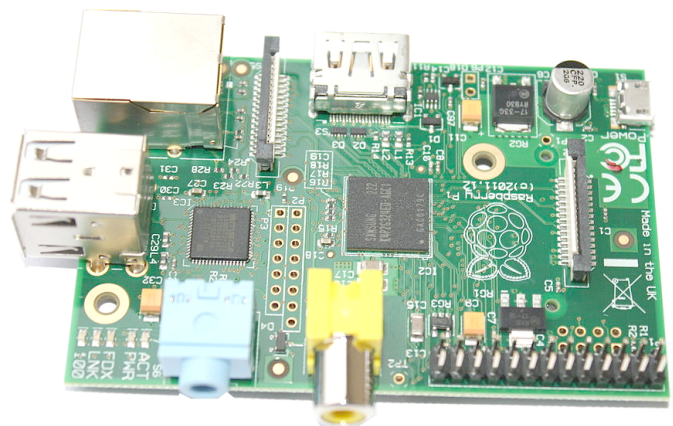
Betriebssysteme

Prof. Dipl.-Ing. Klaus Knopper





Vorlesung an der DHBW Karlsruhe im Sommersemester 2018

Organisatorisches

☞ Vorlesung mit Übungen Betriebssysteme WWI17B2 jeweils Montags in A467 Hörsaal Wirtschaft/WI

Mo. 05.03.2018 ab 10:30 Uhr	Organisatorisches, Einführung, Ausgabe Arbeitsmaterial
Mo. 12.03.2018 ab 12:15 Uhr	Vorlesung, Übungen
Mo. 19.03.2018 ab 10:00 Uhr	Vorlesung, Übungen
Mo. 26.03.2018 ab 13:00 Uhr	Vorlesung, Übungen
Mi. 28.05.2018 10:00-11:30 Uhr	Betriebssysteme-Klausur

☞ **http://knopper.net/bs/**

(später moodle)

Kursziel

- Grundsätzlichen Aufbau von Betriebssystemen in Theorie und Praxis kennen und verstehen,
- grundlegende Konzepte von Multitasking, Multiuser-Betrieb und Hardware-Unterstützung / Resource-Sharing erklären können,
- Sicherheitsfragen und Risiken des Ubiquitous und Mobile Computing auf Betriebssystemebene analysieren,
- mit heterogenen Betriebssystemumgebungen und Virtualisierung arbeiten, Kompatibilitätsprobleme erkennen und lösen.

Themen (Top-Down)

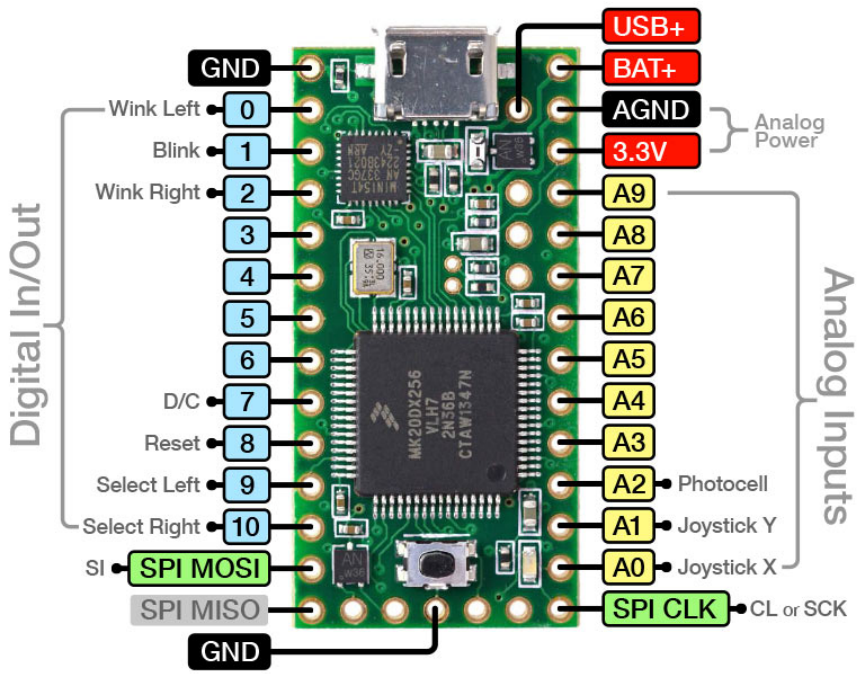
- ☞ Übersicht Betriebssysteme und Anwendungen, Unterschiede in Aufbau und Einsatz, Lizenzen, Distributionen,
- ☞ GNU/Linux als OSS-Lernsystem für die Vorlesung, Tracing und Analyse des Bootvorgangs,
- ☞ User Interface(s),
- ☞ Dateisystem: VFS, reale Implementierungen,
- ☞ Multitasking: Scheduler, Interrupts, Speicherverwaltung (VM), Prozessverwaltung (Timesharing),
- ☞ Multiuser: Benutzerverwaltung, Rechtesystem,
- ☞ Hardware-Unterstützung: Kernel und Module vs. „Treiber" - Konzept,
- ☞ Kompatibilität, API-Emulation, Virtualisierung, Softwareentwicklung.
- ☞ Sicherheits-Aspekte von Betriebssystemen, „Schadsoftware" und forensische Analyse bei Kompromittierung oder Datenverlust.

Zur Benutzung der Folien

- Foliensätze werden nach Bedarf erstellt, und können sich bis zum Ende der Veranstaltung noch ändern. Daher bitte Vorsicht beim Ausdrucken.
- Verweise auf Handouts oder sinnvolle ☞ Sekundärliteratur sind entsprechend gekennzeichnet und i.d.R. direkt anklickbar.
- Prüfungsrelevant sind grundsätzlich alle in der Vorlesung behandelten Themen.

Wann braucht man (k)ein Betriebssystem? *

Beispiel 1 „Hut mit Augen„: „Teensy“ Mikrocontroller (Nano-☞ Arduino-Board), 2 TFT-Displays, Stromversorgung, C-Programm (Sketch)

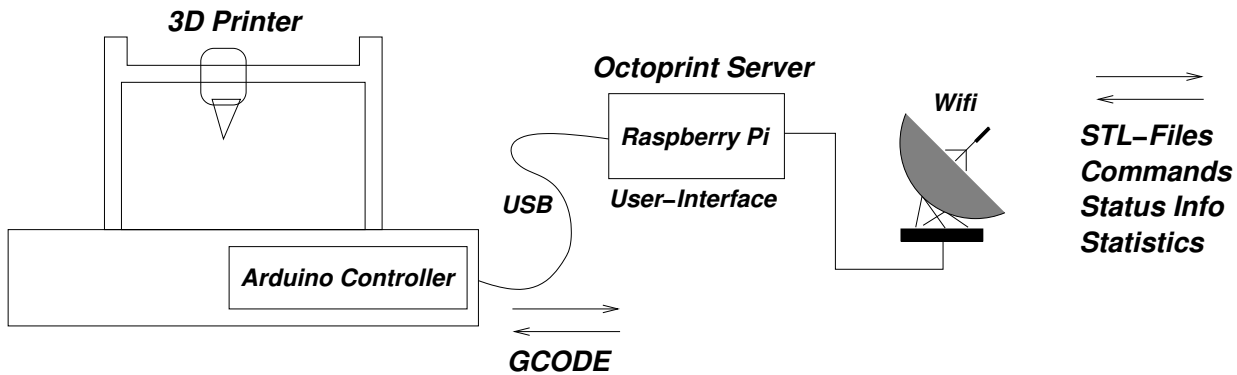


Ein auf dem Microcontroller ständig laufendes Programm steuert (ohne Betriebssystem) zwei TFT-Displays an.

Wann braucht man (k)ein Betriebssystem? *

Beispiel 2 „Motorsteuerung 3D Drucker“ , ebenfalls mit Mikrocontroller (Demo)

Kombination: Steuerung eines 3D-Druckers



☞ Octoprint kommuniziert mit dem Drucker über primitive ☞ G-CODE Steuerbefehle, bietet dem Benutzer aber über einen unter Linux/Raspi laufenden Webserver eine komfortable Oberfläche.

Warum ausgerechnet Linux als Lern-OS?

- Funktionsweise und „Interna“ gut dokumentiert,
- technische Konzepte auch auf andere Betriebssysteme übertragbar,
- Open Source (Folie ☞ 39 ff.):
  - Bauplan / Quelltext aller essentiellen Komponenten ist offengelegt,
  - kostenlose Nutzung,
  - Kopie/Modifikation/Verbreitung erlaubt.
- Dominierendes Betriebssystem im Bereich Internet of Things (IoT) und Mobile Computing (☞ (24)),

Ein wenig Geschichte (1)

um 1970: Ken Thompson (Bell Labs): Betriebssystem für Großrechner ☞ MULTICS, nach dessen Scheitern Entwicklung von ☞ Unix

Ende der 70er: University of California in Berkeley (UCB): ☞ „Berkeley System Distribution“ (BSD), Ansätze zur Netzwerkfähigkeit und virtueller Speicher.

70er-80er: Wettbewerb zwischen den beiden Hauptderivaten (commercial) System V ☞ Unix (AT&T-Zweig) und der freien Berkeley-Version BSD. (Mainframes)

Ende 80er: ☞ MS-DOS, ☞ Windows, OS/2, div. andere proprietäre Betriebssysteme für Einzelplatzrechner.

Ein wenig Geschichte (2)

1984 Richard Stallman gründet die ☞ Free Software Foundation, eine Gesellschaft, die freie Software (mit offenen Quelltexten) fördert, und mit Hilfe einer speziellen Lizenz, der GNU General Public License, die Offenheit und freie Verteilbarkeit der Software garantiert. Langfristiges Ziel ist es, ein Betriebssystem und eine Suite von Anwendungen zur Verfügung zu stellen, die vollständig frei sind von proprietärem oder nutzungslizenzpflichtigem Material.

1988: ☞ **POSIX** 1003.1 wird verabschiedet, ein Standard, der die Mindestanforderungen der Unix-basierten Lager vereint. Fast alle modernen Unices sind **POSIX**-konform.

Ein wenig Geschichte (3)

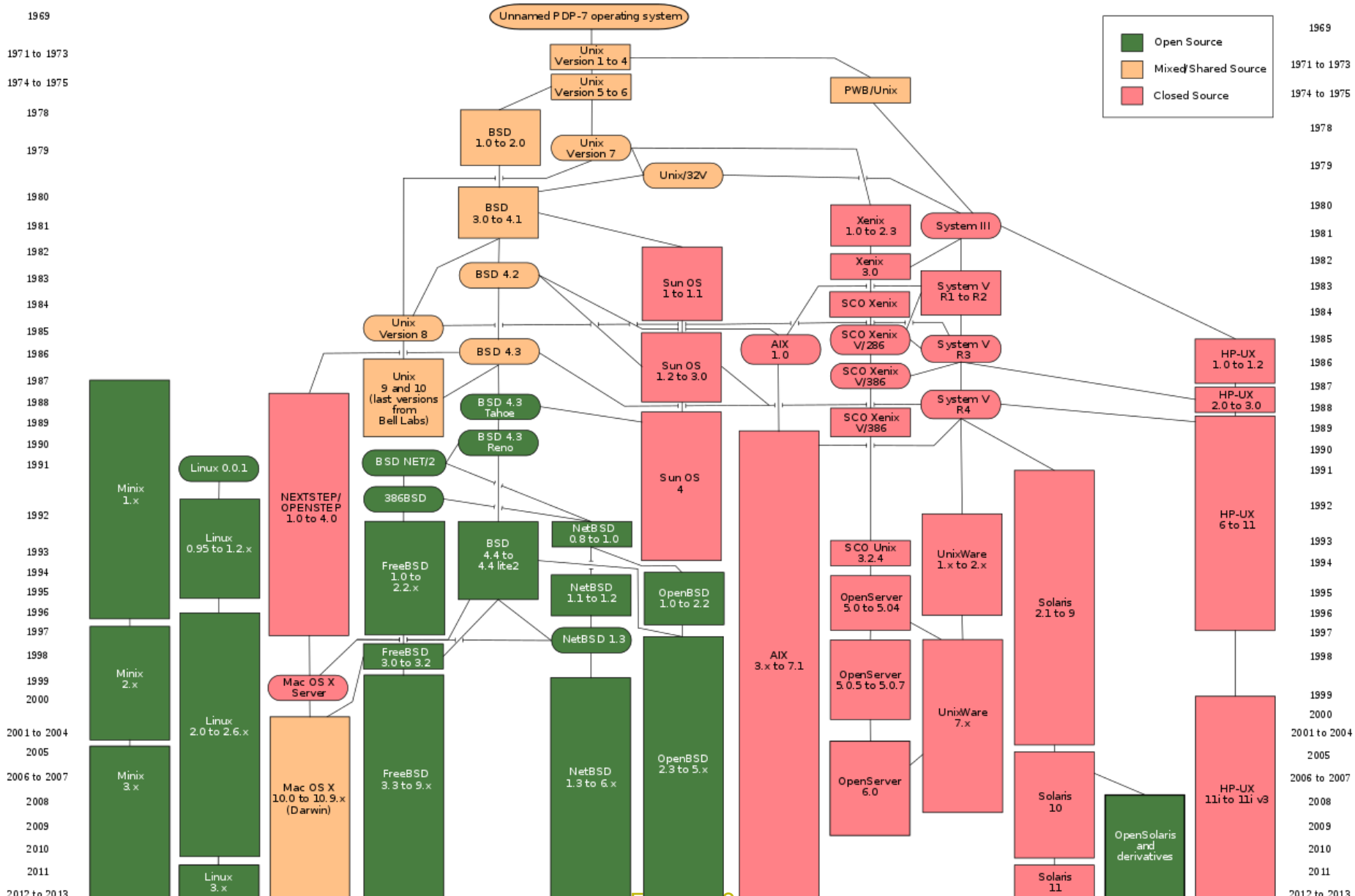
1993 Der finnische Student Linus Torvalds schreibt eine virtuelle Speicherverwaltung für i386-basierte Rechner. Er entscheidet sich dafür, den Quelltext seiner Arbeit zu veröffentlichen, was eine zuvor selten gekannte Kooperation zwischen Entwicklern über das Internet weltweit auslöst und schafft damit die Grundlage für das heute populärste freie Unix-artige Betriebssystem ☞ **Linux** für kostengünstige Desktop-PCs und andere. Auf Anwendungsebene wird die bereits für andere Unix-Systeme vorhandene GNU-Software portiert und verwendet, so dass nach kurzer Entwicklungszeit ein vollständiges Set an Anwendersoftware inclusive Entwicklungsumgebungen zur Verfügung steht.

Ein wenig Geschichte (4)

1995- Das ☞ Linux-Betriebssystem und auf Unix-Betriebssystemen basierende Anwendungen wie SAMBA und Apache verbreiten sich vor allem als preisgünstige Server-Systeme, zunächst nur als Geheimtipp unter Technikern, heute als Mainstream-Software im regulären Ersatz als File, Print- und Informationsserver sowie Gateways in heterogenen Netzwerkumgebungen.

1998-heute Neben dem zunehmenden Einsatz als Desktop/Client-System mit KDE oder GNOME hält Linux auch Einzug als embedded Betriebssystem in Handhelds, Tablets, Kameras, MP3-Spielern und anderen Geräten der Unterhaltungs- und Kommunikationsindustrie.

Unix-Stammbaum*



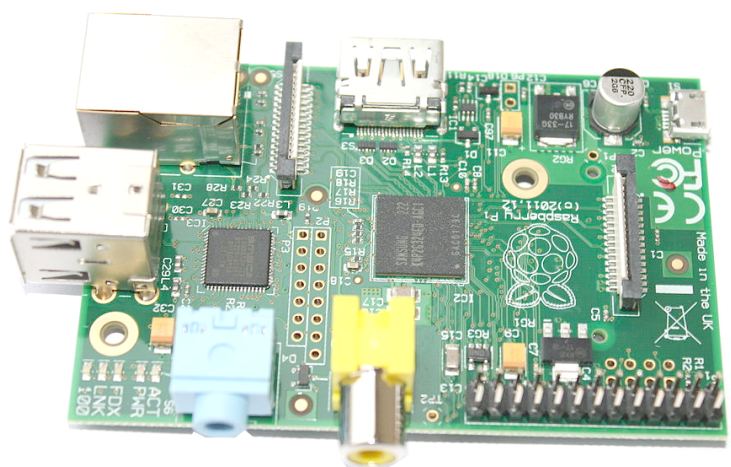
Nicht-PCs und embedded Systeme

Durch seine Flexibilität und die freizügige Lizenz ist GNU/Linux eine beliebte Systemsoftware für neu entwickelte Hardware, wenn nicht sogar in vielen Fällen die einzige Möglichkeit.

- ☞ RaspberryPi, ☞ Arduino Microboards,
- Netzwerkkomponenten: Accesspoints (☞ OpenWRT), Managed Switches, ...
- Unterhaltungselektronik: Receiver, SmartTVs, ...
- Steuerelektronik: autonome Fahrzeuge, Flugdrohnen, ...

Beispiel RaspberryPi

- ARM-Architektur, Board kaum größer als eine EC-Karte,
- Energiebedarf $<$ 3W, Versorgung über USB,
- Netzwerk-, USB-, HDMI-, Composite-, Audio-Anschluss, SD-Karteneinschub für OS,
- Betriebssystem: Raspbian (Debian), BSD, Plan9, RiscOS,
- Anwendung als: Steuerungssystem, Streaming Server, Multimedia (☞ XBMC), „normaler Desktop-PC“ (mit gewissen Performance-Einschränkungen).

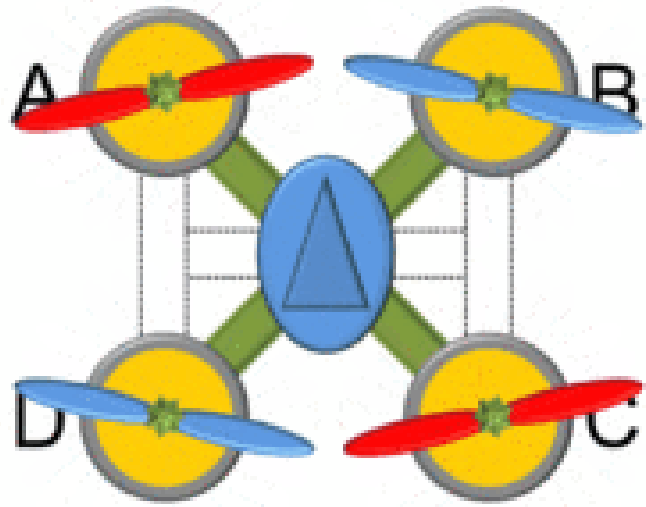


Beispiel Haushaltsroboter

Autonomer Staubsauger

Beispiel Flugdrohne

Zur Steuerung der 4 Propeller eines ☞ Quadrokopters und Anbindung an eine Steuerungskonsole (Computer, Smartphone oder RC-Steuerung per Funk) wird eine „leichtgewichtige" Lösung benötigt, die auf sensorisch erfasste Positionsveränderungen in ☞ Echtzeit reagieren kann. Hierdurch wird das Gerät auch ohne Interaktion stabil in der Luft positioniert und kann auf Anweisungen reagieren.



Demo mit ☞ AR.DRONE

Eigenschaften von Unix

☞ Unix in Wikipedia

- Mehrere Aufgaben gleichzeitig (**Multitasking**)
- Mehrbenutzerfähig (**Multiuser**)
- Auf vielen Hardware-Plattformen lauffähig (**portabel**)
- Effiziente, transparente Ausnutzung der Resourcen
- **hierarchisches Dateisystem**
- Stabilität durch eigenen Speicherbereich für jedes Programm (**Virtual Memory**, Speicherschutz)
- strikte Trennung zwischen Betriebssystem („**Kernel**", Systemdienste) und Anwendersoftware (Desktop-, Server-Suiten)

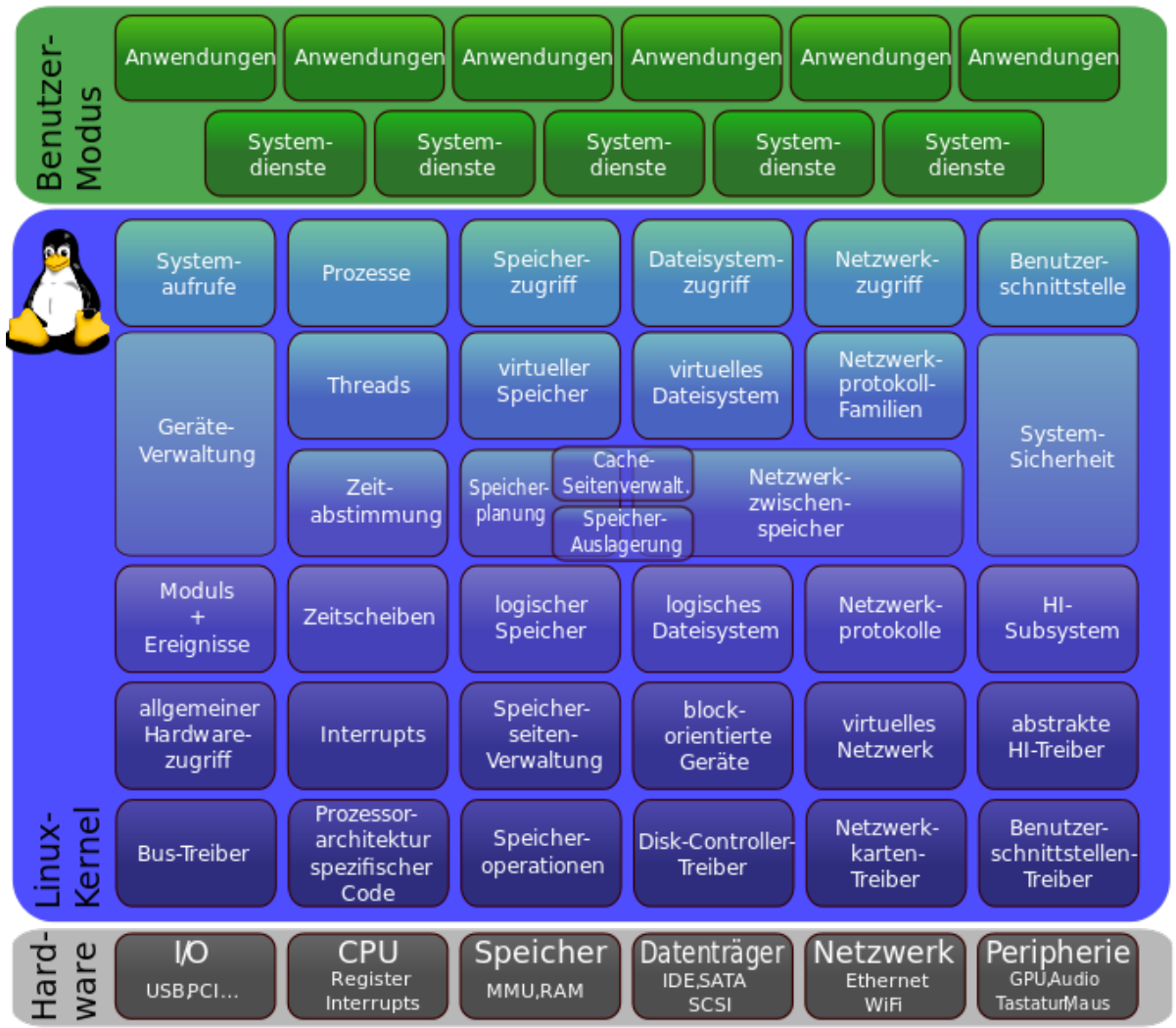
Auch andere Betriebssysteme haben diese Design-Vorgaben übernommen, in unterschiedlicher Ausprägung.

Verschiedene Unix-Betriebssysteme

Name/TM	Hersteller
☞ SunOS/Solaris	Oracle (ehem. SUN Microsystems)
☞ HPUX	Hewlett Packard
☞ Aix	IBM
☞ Sinix	Siemens/Nixdorf
☞ Ultrix/DEC Unix(OSF/1)	Digital Equipment
☞ „Services for Unix“ (Interix)	Microsoft
☞ Mac OSX / iOS	Apple
☞ Berkeley Software Distribution (BSD)	Community (Entwickler)
☞ Linux	Community (Entwickler)
☞ Android	Google

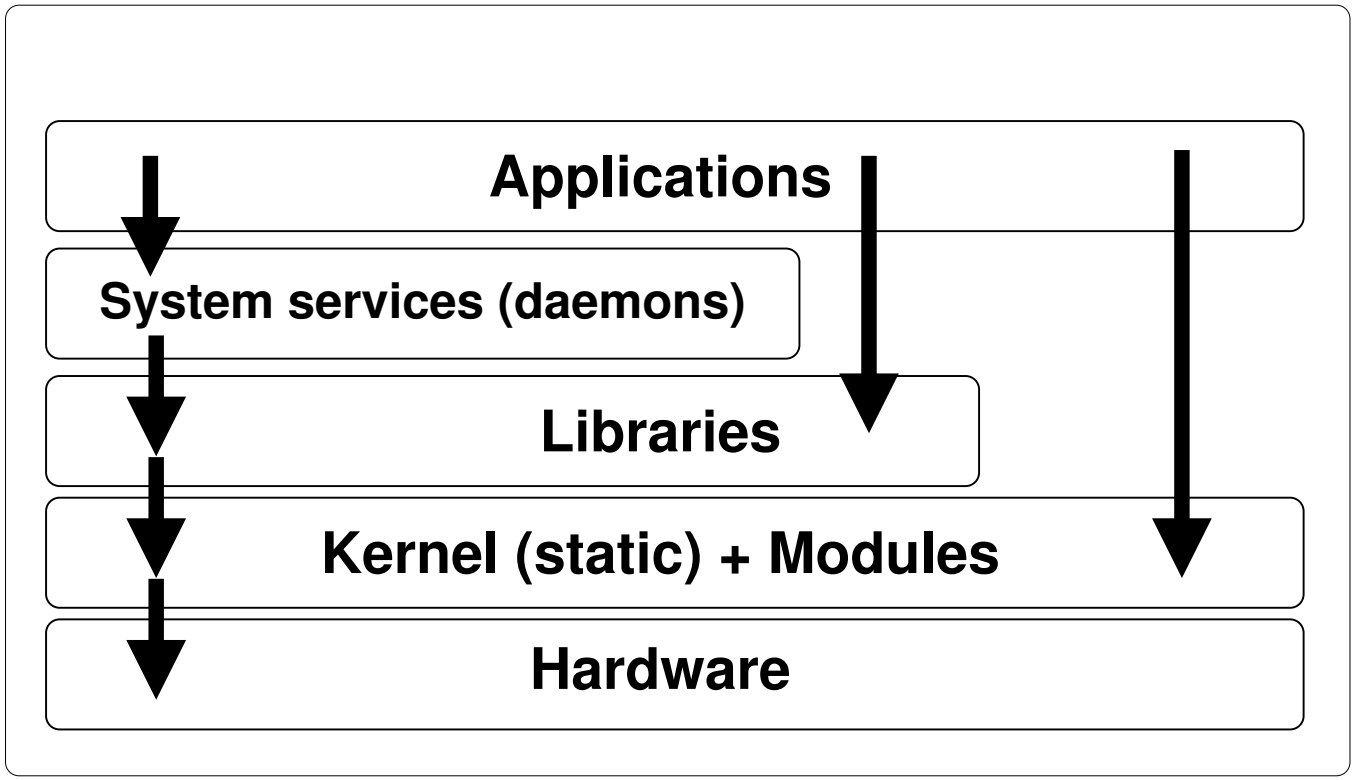
...

Technik: Aufbau und Eigenschaften von Linux *



Linux-Architektur

Vereinfachtes Schema



Einwurf: Die Softwarekrise

- Die **Komplexität** von Betriebssystemen und Anwendungen und damit auch die **Anfälligkeit für Fehler** steigt ständig, gleichzeit versprechen die Software-Anbieter, dass die Programme immer **„einfacher“ bedienbar** werden.
- Früher: Anwender, Programmierer, Systemadministrator in einer Person ☞ Fähigkeit, das System selbst zu kontrollieren und erweitern zu können, Programem möglichst effizient gestalten und nutzen.
- Heute: Der Anwender möchte sich mit der Technik nicht auseinandersetzen, aber trotzdem alle Möglichkeiten effizient nutzen. ☞ Kontrolle über das System, Verantwortung für Wartung und Aktualisierung wird an den Softwarehersteller übertragen.
- Heute: Auch für Systemprogrammierer und Administratoren wird es immer schwieriger, die Übersicht zu behalten und die Software in jedem Detail zu verstehen, Fehler zu erkennen.

Konsequenzen daraus?

- **Verlust von Kontrolle und Verständnis für technische Zusammenhänge**, fast bedingungsloses Vertrauen gegenüber Hard- und Softwareherstellern, **Missbrauchsmöglichkeiten** (☞ Spionage, ☞ Malware, Fernsteuerung / ☞ Botnetze, ☞ Identitätsdiebstahl).
- Schere zwischen ☞ „Geeks“ und ☞ „DAUs“ wird stetig größer?
- Lösungsansätze?

Anteil Linux-Systeme auf Smartphones

Period	Android	iOS	Windows Phone	Others
2016Q1	83.4%	15.4%	0.8%	0.4%
2016Q2	87.6%	11.7%	0.4%	0.3%
2016Q3	86.8%	12.5%	0.3%	0.4%
2016Q4	81.4%	18.2%	0.2%	0.2%
2017Q1	85.0%	14.7%	0.1%	0.1%

Quelle: ☞ IDC

OSS als Chance?

- Das Open Source Prinzip erlaubt es, die Software legal zu analysieren, zu verändern und weiterzuentwickeln.
- Ein Großteil der am häufigsten genutzen Software ist Open Source.
- Open Source erlaubt es, nach dem Baukasten-Prinzip ein auf den Anwender optimiertes System aufzubauen, ohne künstliche Einschränkungen.

Open Source für Linux und Windows

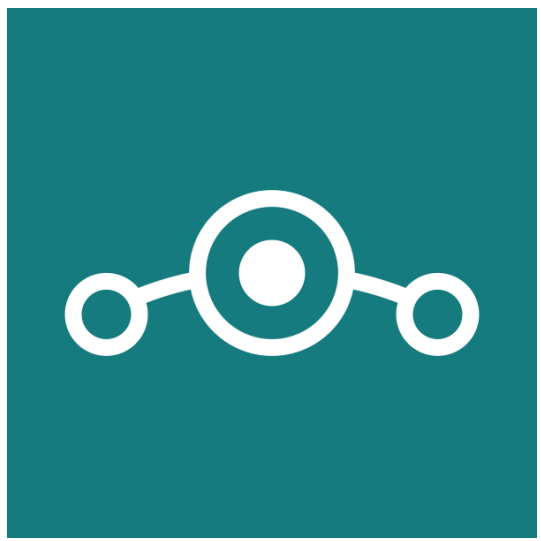
- ☞ Cygwin Unix-Shell (Windows),
- ☞ Firefox Browser,
- ☞ OpenOffice/LibreOffice (ISO-Standard) Office-Suite,
- Web-Programmiersprachen PHP, JavaScript, Java, ...
- ☞ eclipse Entwicklungsumgebung,
- ☞ Apache Webserver und ☞ SAMBA als führende Internet-/Intranet-Serverdienste.

Proprietäres vs. Open Source Android

- Google entwickelt gemäß der GNU GENERAL PUBLIC LICENSE, unter der der Linux-Kernel und viele Linux-Komponenten stehen, Android zunächst auf Open Source Basis. Jeder kann sich das ☞ Android SDK kopieren und damit Betriebssysteme und Anwendungen entwickeln.
- Hardware-Hersteller passen Android auf ihre speziellen Plattformen an, erweitern das System jedoch oft mit *proprietären Komponenten* (z.B. Anwendungen für spezielle Eingabegeräte wie Stift, Spracheingabe oder Gesten) und verhindern durch technische Maßnahmen, dass der Anwender Zugriff auf Betriebssystememebene erhält. *Gefahr durch potenzielle Überwachung und „Kundenbindung"*.
- Freie Derivate, die sich an der Open Source-Basis von Android orientieren, ermöglichen die Rückgewinnung der Kontrolle und unabhängige Unstallations- und Updatemöglichkeiten („Rooten"). Laut Aussage der Hardwarehersteller bedeutet dies jedoch den Verlust der Garantie auf das Gerät, da eine herstellerseitige Remote-Wartung unterbunden wird. (Legalität?)

„Mod“ oder „Back to the Roots“

☞ LineageOS als beliebtestes Android-Derivat / Alternative zur herstellereigenen, proprietären Firmware.

☞ 

Distribution (1)

Eine Distribution fasst bekannte Software als ein in sich stimmiges „Produkt“ zusammen, das als Zusammenstellung von Betriebssystem und ausgewählter Anwendersoftware verteilt wird. Der Fokus liegt dabei auf bestimmten Zielgruppen.

Distribution (2)

Ubuntu gilt als einsteigerfreundlich und soll in der Installation und Wartung (Administration) besonders einfach zu bedienen sein.

Fedora wendet sich v.a. an Kunden aus dem unternehmerischen Bereich, die stabile Server-Software mit Zertifizierung für Standardsoftware, u.U. auch proprietäre, wünschen.

OpenSuSE war ursprünglich speziell auf Anwender im deutschsprachigen Raum spezialisiert, seit dem Kauf durch Novell jedoch auch eher am internationalen Markt ausgerichtet.

Debian ist die größte vollständig Community-basierte Distribution, setzt höheres technisches Verständnis voraus, und ist die Basis für viele kommerzielle Distributionen wie Ubuntu.

Knoppix ist eine auf den Betrieb als autokonfigurierendes Live-System (direkter Start von Wechselmedien oder übers Netz) spezialisierte Variante von Debian.

Knoppix

☞ KNOPPIX ist eine komplett von CD, DVD oder übers Netzwerk lauffähige Zusammenstellung von GNU/Linux-Software mit automatischer Hardwareerkennung und Unterstützung für viele Grafikkarten, Soundkarten, USB-Geräte und sonstige Peripherie.

KNOPPIX kann als produktives Linux-System für den Desktop, Schulungs-CD, Rescue-System oder als Plattform für kommerzielle Software-Produktdemos angepasst und eingesetzt werden. Es ist keinerlei Installation auf Festplatte notwendig. Auf der CD können durch transparente Dekompression bis zu 2 Gigabyte an lauffähiger Software installiert sein (in der DVD „Maxi" Edition über 10 Gigabytes).

Betriebssysteme vs. Anwendungen

1. Systemsoftware
   Hierzu gehören das Betriebssystem (inkl. „Treiber“ bzw. Kernel und -module) des Computers sowie alle Systemdienste und -programme, die dafür sorgen, dass die Hardware-Resourcen des Computers im laufenden Betrieb nutzbar und sicher sind.

2. Anwendersoftware
   Hierzu gehören die Programme, mit denen der Computer-Nutzer direkt arbeitet. Unter Unix zählt neben den Anwendungen (Office-, Datenverarbeitende Programme, Spiele, Internet-Nutzungssoftware) auch der graphische Desktop zur Anwendersoftware, und kann durch den Anwender beliebig ausgetauscht und verändert werden.

Startvorgang von Computersystemen

Der Bootvorgang beginnt mit dem Einschalten des Rechners, und endet mit dem Produktivbetrieb bzw. dem Zugang zur interaktiven Oberfläche.
Abhängig von der Rechner-Architektur verläuft der Startvorgang unterschiedlich. Wir beobachten den Verlauf auf einem üblichen Desktop-PC oder Notebook.
☞ Handout „Bootvorgang“

Bootvorgang

1. Strom einschalten
2. BIOS aktiviert die im Rechner eingebaute Hardware und sucht nach einem „bootfähigen Gerät“ , Reihenfolge nach Einstellung in der BIOS- Konfiguration
3. Bootlader startet vom bootfähigen Gerät (Festplatte, DVD, USB-Stick, ...) und lädt den Betriebssystem-Kern in den Hauptspeicher.
4. Betriebssystem-Kern muss selbstständig die Hardware lauffähig machen (Treiber unter Windows bzw. „Kernel-Module“ unter Linux).
5. Linux: Das erste Programm „init“ startet, und „fährt den Rechner hoch“ , d.h. es startet einzelne Systeme wie Grafikserver, Login-Manager, Desktop, ...
6. Der Benutzer kann sich anmelden bzw. mit der interaktiven Arbeit mit dem Computer beginnen.

Kompatibilität

Jedes Betriebssystem und jede Version davon stellt eine Laufzeitumgebung für Anwenderprogramme zur Verfügung. Aufgrund der unterschiedlichen Schnittstellen (APIs) sind diese leider in den meisten Fällen untereinander inkompatibel, d.h. grundsätzlich:

- Windows-Programme laufen nicht unter Linux,
- Linux-Programme laufen nicht unter Windows,
- Programme für neuere Windows-Versionen laufen nicht mit älteren Versionen zusammen,
- Programme für neuere Linux-Versionen laufen nicht mit älteren Versionen zusammen,
- Programme für eine Prozessorarchitektur laufen nicht auf einer anderen.

Hierfür gibt es einige Lösungsansätze, die das „Unmögliche" möglich machen. ☞ Handout „Windows-Programme-unter-Linux-und-umgekehrt". Mit dem Thema „Virtualisierung" werden wir uns später noch näher beschäftigen.

Anwenderprogramme (Beispiele) unter OSS

1. Textverarbeitung, Tabellenkalkulation, Präsentation, Zeichnungen, Datenbank-Anbindung: OpenOffice
2. Grafikbearbeitung: Gnu Image Manipulation Program (GIMP)
3. WWW: Apache WWW Server, Mozilla Firefox Browser
4. E-Mail/Groupware: Evolution
5. Multiprotokoll Chat: Pidgin
6. Videokonferenz: Ekiga

Rechtliche Aspekte von Software / Lizenzen

- Urheberrecht
- Überlassungsmodelle (Lizenzen)
  - Verkauf (selten)
  - Nutzung / Miete (entgeltlich oder unentgeltlich)
  - Open Source / Freie Software (weitgehende Übertragung der Verwertungsrechte auf den Lizenznehmer)
- Patente (?)

Proprietäre Software

- Der Empfänger erwirbt mit dem Kauf eine eingeschränkte, i.d.R. nicht übertragbare *Nutzungs*lizenz.
- Der Empfänger darf die Software nicht analysieren („disassemble"-Ausschlussklausel).
- Der Empfänger darf die Software nicht verändern.
- Der Empfänger darf die Software nicht weitergeben oder weiterverkaufen.

Diese Restriktionen werden im Softwarebereich so breit akzeptiert, dass man fast schon von einem „traditionellen" Modell sprechen kann.

„Open Source“

- Open Source stellt Software als Resource/Pool zur Verfügung.
- Open Source sichert dem Anwender (Benutzer und Programmierer) bestimmte Freiheiten.
- Open Source stellt eine Basis (Lizenz) für eine Zusammenarbeit von Gruppen (oder Firmen) zur Verfügung.

Was ist Freie Software/Open-Source?

- Open-Source (engl. = offene Quelle)
- Freie Software (FSF, 1984) ist Teilmenge von Open-Source-Software.
- Open-Source ist kein Produkt, sondern
- eine *Methode*, um Software zu entwicklen.
- Open-Source-Definition lt. OSI.
- „Frei" steht für **Frei**heit (ff.), nicht für „kosten**frei**" !

Die GNU General Public License

gibt den *Empfängern* der Software das Recht, ohne Nutzungsgebühren

- die Software für alle Zwecke einzusetzen,
- die Software (mit Hilfe der Quelltexte) zu analysieren,
- die Software (mit Hilfe der Quelltexte) zu modifizieren,
- die Software in beliebiger Anzahl zu kopieren,
- die Software im Original oder in einer modifizierten Version weiterzugeben oder zu verkaufen, auch kommerziell, wobei die neuen Empfänger der Software diese ebenfalls unter den Konditionen der ☞ GPL erhalten.

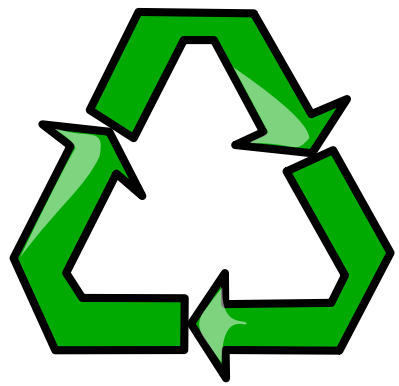
`http://www.gnu.org/`

Die GNU General Public License

- zwingt NICHT zur Veröffentlichung/Herausgabe von Programm oder Quellcode,
- zwingt NICHT zur Offenlegung ALLER Software oder Geschäftsgeheimnisse,
- verbietet NICHT die kommerzielle Nutzung oder den Verkauf der Software,
- verbietet NICHT die parallele Nutzung, oder lose Kopplung mit proprietärer Software.

Die GNU General Public License

Aber: Alle EMPFÄNGER der Software erhalten mit der GPL die gleichen Rechte an der Software, die die Mitentwickler, Distributoren und Reseller ursprünglich hatten (und weiterhin behalten).



Wer legt die Lizenz fest?

Der Urheber.

Für wen gilt eine Lizenz?

Eine Lizenz gilt für die in der Lizenz angegebenen Personenkreise (sofern nach landesspezifischen Gesetzen zulässig).
Beispiel: Die GNU GENERAL PUBLIC LICENSE gilt für

- alle legalen **EMPFÄNGER der Software**, die
- **die Lizenz AKZEPTIERT** haben.

„Wer liest schon Lizenzen?“

- Zumindest in Deutschland bedeutet das FEHLEN eines gültigen Lizenzvertrages, dass die Software NICHT ERWORBEN und NICHT EINGESETZT werden darf.
- In Deutschland gibt es seit der letzten Änderung des Urheberrechtes keine generelle Lizenz-Befreiung mehr.
- Wurde die Lizenz nicht gelesen, oder „nicht verstanden“ (weil z.B. nicht in der Landessprache des Empfängers vorhanden), so ist die rechtliche Bindung, und daraus resultierend, die Nutzungsmöglichkeit der Software, formal nicht gegeben.

Auch als „Freeware“ deklarierte Software ist hier keine Ausnahme. Wenn keine Lizenz beiliegt, die eine bestimmte Nutzungsart ausdrücklich ERLAUBT, gilt sie als VERBOTEN.

„Kopierschutz“ (1)

- Soll die nicht vom Rechteverwerter genehmigte Vervielfältigung unterbinden,
- ist de facto technisch überhaupt nicht realisierbar,*)
- ein „wirksamer“ Kopierschutz (juristisch genügt die Angabe auf der Packung, technisch kann der Kopierschutz absolut wirkungslos sein) darf nach der Urheberrechtsnovelle von 2003 nicht mehr umgangen werden, auch nicht zum Anfertigen einer Kopie für den Privatgebrauch,

...

*) Alles, was audiovisuell wahrgenommen werden kann, kann auch kopiert werden, notfalls über die „analoge Lücke“.

„Kopierschutz“ (2)

- dennoch bleibt das Umgehen eines Kopierschutzes zur ausschließlichen Eigennutzung nach §108b UrhG aber straffrei,
- und laut §69a Abs. 5 UrhG ist das Umgehen einer Kopiersperre speziell bei Computerprogrammen auch nicht in jedem Falle ein Strafdelikt (VORSICHT!).

Fragwürdige Lizenzklauseln

- Genereller „Haftungsausschluss“,
- Eigentumsvorbehalt,
- Konkurrenzausschluss,
- Abtreten von gesetzlich garantierten Grundrechten.

Autor/Distributor haften...

- für „Geschenke“ nur bei GROBER FAHRLÄSSIGKEIT,
- für „Verkäufe“ bei allen vom Verkäufer/Hersteller verschuldeten Fehlern.

GPL-Verträglichkeit

- **GPL** erlaubt die Integration proprietärer Software auf dem gleichen Datenträger, solange die nicht-GPL-Komponenten wieder separierbar sind (Beispiel: KNOPPIX-CD, versch. Linux-Distributionen).
- **BSD**-Lizenz erlaubt die Integration von Code in proprietäre Programme ohne Offenlegungspflicht. Es muss lediglich darauf hingewiesen werden, dass die Software BSD-Komponenten enthält (Beispiel: TCP/IP-Stack im Windows-Betriebssystem).
- Die Programm-Urheber können für ihr Werk auch eine Auswahl verschiedener Lizenzen „zum Aussuchen" anbieten (**Dual Licensing**).

Tabelle: Lizenzmodelle und Rechte

	Nutzung kostenpflichtig	Nutzung kostenlos	frei kopierbar	zeitlich unbegrenzt nutzbar	Quelltext wird mitgeliefert	Modifikation erlaubt	Einbau in prop. Produkte erlaubt	Derivate mit anderen Lizenzen mögl.
proprietäre Software	✔							
Shareware		✔	✔					
Freeware		✔	✔	✔				
GPL		✔	✔	✔	✔	✔		
LGPL		✔	✔	✔	✔	✔	✔	
BSD		✔	✔	✔	✔	✔	✔	✔

Geld verdienen mit Open Source

Da das Einkassieren von „Nutzungslizenzgebühren“ unter Open Source nicht zulässig ist, und die Verbreitung (Kopie, Weiterbearbeitung etc.) auch nicht eingeschränkt werden kann, ist das Geschäftsmodell bei Open Source:

- Nicht die Software selbst, sondern eine Dienstleistung als Produkt anbieten (Support, Wartung, Anpassung),
- nicht „Software von der Stange“ verkaufen, sondern Software im Auftrag entwickeln bzw. auf Kundenbedürfnisse individuell anpassen (Baukasten-Prinzip).

☞ Der Großteil des Umsatzes der bekannten „Software-Riesen“ baut auf diesem Konzept auf, wobei der Anteil an eingesetzter Open Source Software aber unterschiedlich hoch ist.

Transfer der OSS-Idee

- Verschärfung des Urheberrechtes zugunsten der Rechteverwerter-Industrie führt zu Ablehnung durch viele Kreative.
- Schaffung von rechtlichen Grundlagen zur Eigenvermarktung und Eigenverlag von Kunstwerken durch die Künstler ohne Exklusivvertrag mit einer Verwertungsgesellschaft.
- „Lizenz-Baukasten" für verschiedene Empfängerkreise und Verwertungszwecke.

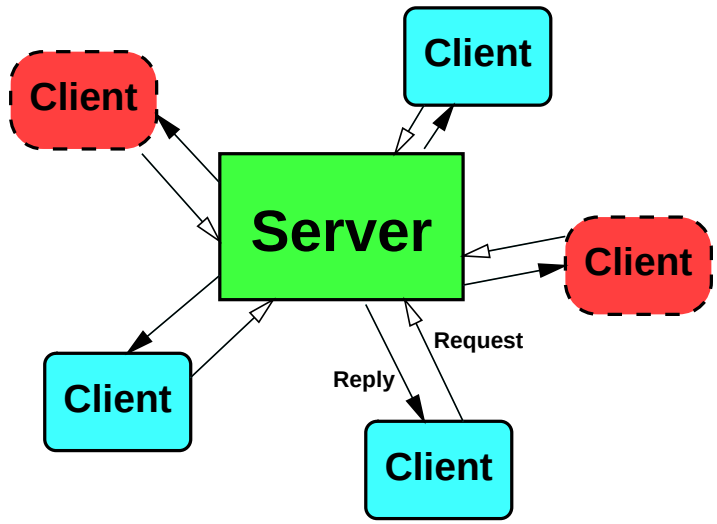
Beispiel für prof. Animationsfilme unter ☞ Creative Commons Lizenz: ☞ „Elephants Dream", ☞ Big Buck Bunny, ☞ Sintel, ☞ Tears of Steel (neu).

Empfohlene Literatur zum Internetrecht

Professor Dr. Thomas Hoeren, Institut für Informations-, Telekommunikations- und Medienrecht an der Universität Münster, Kompendium zum Internetrecht (PDF)

Client/Server-Prinzip

Die meisten Dienste unter Unix basieren auf dem Client/Server-Prinzip.



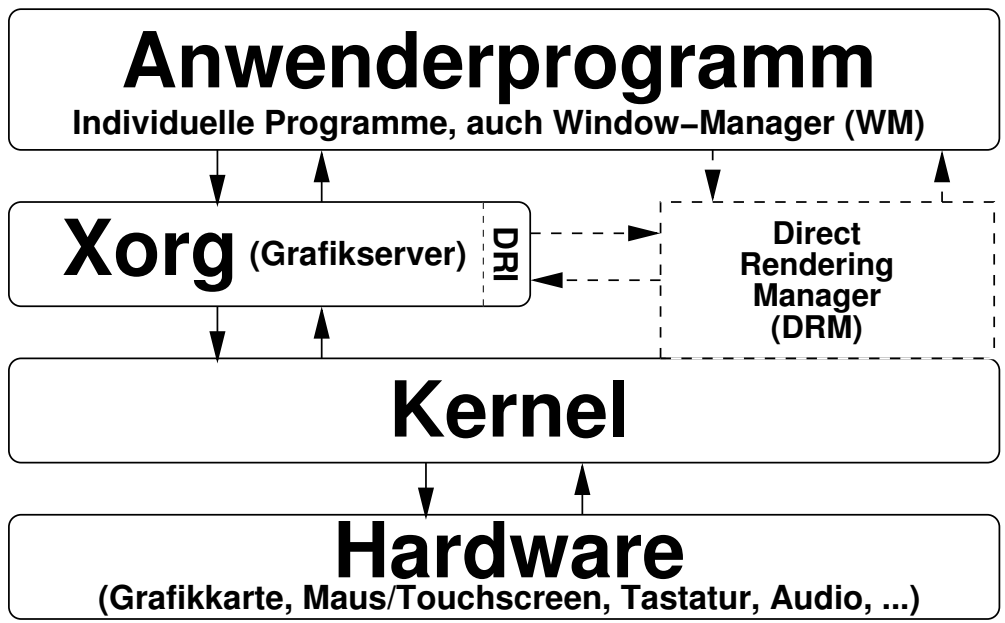
- Client: Dienstanforderer (z.B. Browser)
- Server: Diensterbringer (z.B. WWW-Server)
- Abwicklungsprotokoll, das beide Partner verstehen, z.B. **`http`**.

Die Benutzeroberfläche (User Interface)

- Wird meist mit einer graphischen Mensch-Maschine Schnittstelle in Verbindung gebracht (Graphical User Interface bzw. GUI).
- Soll eine intuitive interaktive Bedienung von Anwenderprogrammen ermöglichen.
- Für Menschen mit Behinderung oder Anwendungen, bei denen die Aufmerksamkeit des Benutzers für andere Dinge notwendig ist, wird auf eine graphische Benutzerführung verzichtet und alternative Konzepte realisiert, wie Steuerung durch Gesten oder Hand/Kopfbewegugen, Ausgabe über Sprache oder haptisch (z.B. Vibration oder Braille-Zeichen).
- Theoretisch kann jedes Programm seine eigene Benutzerschnittstelle implementieren, in der Praxis wird jedoch von vielen Betriebssystem-Herstellern mehr oder weniger großer Wert auf ein konsistentes Look & Feel gelegt (Beispiel MAC/OSX).

GUI bei Windows vs. Unix

Währed bei Windows die graphische Oberfläche integraler Bestandteil des Betriebssystems ist, ist bei Linux die globale Bedienoberfläche ein „Anwenderprogramm“ und beliebig austauschbar.



Beispiel Compiz

Compiz ist ein Compositing Windowmanager, der Plugin-basiert alle graphischen Ausgaben mit Rendering-Funktionen der Grafikkarte auf dem Desktop (der unabhängig davon gewählt werden kann) verwaltet.


```
compiz --replace &
```

Desktop-Systeme

Während der (jeweils einzige) Hersteller bei Windows und Mac bei allen vom ihm selbst hergestellten Anwendungen ein konsistentes Look & Feel vorgibt, gibt es unter Linux einen „Wettbewerb“ zwischen den großen graphischen Anwendungssystemen.

- KDE (umfangreiche Anwendungen, die nahtlos zusammenarbeiten),
- GNOME (ebenso),
- XFCE (besonders schlank, dennoch viele Desktop-Funktionen, kleinere Anwendungen),
- LXDE (besteht nur aus Dateimanager **`pcmanfm`**, Panel **`lxpanel`**, Session-Manager **`lxsession`**, kein eigener WM, startet sehr schnell),
- Enlightenment, Ratpoison, Larswm, Openbox ... (sehr individuell).

Anwendungen eines bestimmten Desktop-Systems lassen sich auf jedem der anderen Desktop-Systeme starten und sind nicht an einen bestimmten Windowmanager gebunden, aber: Uneinheitliches Aussehen, unterschiedliche Anwendungen für den gleichen Zweck, z.B. Browser, Dateimanager...

Embedded Devices

- Die **überwiegende Mehrheit** der Geräte im Internet of Things (IoT)-Bereich kommt völlig ohne Benotzerinterface aus und kommuniziert wireless mit anderen Geräten oder Cloud-Diensten.
- Entwickler erhalten interaktiven Zugriff auf die Geräte mittels einer einfachen seriellen Schnittstelle, die bei Bedarf „angedockt" wird (vergl. Handout „Zugang zum Raspberry Pi" mit seriellem Kabel).
- Da der Zugang über serielle Schnittstellen extrem einfach aufgebaut ist, wird hierbei i.d.R. ein rein Text-/Kommandobasiertes Interface realisiert.

UI für Fortgeschrittene: Die Shell „Kommandozeile“

Die Shell ist, neben einer Programmiersprache mit einer Vielzahl von Funktionen, ein „Starter“ für Programme, deren Aufruf üblicherweise per Tastatur eingegeben wird.

Hierbei steuern Optionen die Arbeitsweise der Programme. Unabhängig von ihrem Start über die Shell können Programme graphische oder textuelle („Konsolen“-) Ausgaben produzieren.

Für fortgeschrittene Administratoren und „Power-User“ ist die Shell nicht nur im IoT-Entwickler-Bereich der direkte und unmissverständliche Weg, mit Betriebssystem und Systemsoftware zu kommunizieren, da viele betrisbssysteminterne Funktionen nicht in den „einfach“ gehaltenen graphischen Oberflächen abgebiltet werden bzw. verfügbar sind.

Aufrufkonvention:

```
Kommandoname Optionen... Argumente...
```

Standard Unix-Systemtools

- Viele, kleine Programme für jeweils nur eine Aufgabe,
- extrem kurze, „selbsterklärende“ Kommandonamen,
- leichte Kombinationsmöglichkeit dieser kleinen Programme mit Eingabe- und Ausgabeumleitung („Pipes“).

Kommando-Syntax

In der Shell eingegebene Kommandos haben im Allgemeinen das folgende Format:

```
Programmname Optionen Argumente Umleitung
```

Die Umlenkung von Ein- und Ausgabe funktioniert, anders als bei DOS und Windows, bei denen häufig Temporärdateien geschrieben werden, auf direktem Weg auch zwischen Programmen.

> Datei	Umlenkung der Ausgabe in Datei
< Datei	Umlenkung der Eingabe von Datei
| Kommando	Umlenkung der Ausgabe in die Eingabe eines anderen Programms.

Beispiel: Sortierte, seitenweise Ausgabe der aufsummierten Verzeichnisinhalte in Kilobytes

```
du -sk /var/* | sort -rn | less
```

Einwurf: Dateisystem (Übersicht)

Um identifizierbare Objekte (Multimedia-Dateien, Dokumente, Programme, ...) auf Datenträgern schreiben und wieder von ihnen lesen zu können, werden diese bei fast allen Betriebssystemen in einem Ordnungssystem mit hierarchisch angeordneten Verzeichnissen abgelegt.

Während Windows aus historischen Gründen allen Datenträgern „Laufwerksbuchstaben", und jedem Laufwerk eine individuelle Verzeichnisstruktur gibt, ist unter Unix nur ein einziger Verzeichnisbaum vorhanden, beginnend mit dem Wurzelverzeichnis **`/`**. Alle Datenträger, auch „Netzlaufwerke", werden vom Administrator oder einem dafür ausgelegten Dateimanager in einen frei wählbaren Unterordner dieses Verzeichnisbaums „montiert" (**`mount`**).

☞ Handout „Datentraegerverwaltung"

Navigieren im Dateisystem mit der Shell

`pwd`	Ausgabe aktuelles Arbeitsverzeichnis
`cd Verzeichnisname`	Wechsel des aktuellen Verzeichnisses
`cat Datei...`	Inhalt von Dateien lesen/ausgeben
`ls -l [wildcards]`	Ausführliches Auflisten von Dateien*)
`mkdir [-p] Verz.`	Lege (Mit Unterverz.) Verzeichnis an.
`cp [-a] Alt Neu`	Kopiere (Klone Alles) von Alt nach Neu
`mv Alt Neu`	Benenne Alt nach Neu um
`rm [-rf] wildcard`	Lösche unwiderruflich (rekursiv forciert) †)

---

*) Die angezeigten Dateirechte werden noch ausführlich besprochen.

†) Tipp: Überlegen Sie zweimal, bevor Sie mit der Option **-rf** etwas löschen, gerade, wenn Sie momentan Administratorstatus haben!

Dateisystem-Informationen in der Shell

`df`	Ausgabe der eingebundenen Dateisysteme mit „Füllstand“
`mount`	Ausgabe der eingebundenen Dateisysteme mit Optionen
`du -s[km] [wildcards]`	Aufsummieren der Inhalte (in Kilobytes/Megabytes)

Unter Linux sind Detailinformationen über Dateisysteme auch im virtuellen **`/sys`** und **`/proc`** Dateisystem abrufbar.

`cat /proc/filesystems`	Ausgabe der bekannten Dateisystem-Typen
`cat /proc/partitions`	Ausgabe der verfügbaren Partitionen mit Größen in kB
`cat /proc/mounts`	Ausgabe der eingebundenen Dateisysteme mit Optionen

Linux Benutzer und Administrator

Merkregel: Benutzen Sie den Administratoraccount (User-ID 0) **ausschließlich** zur systemweiten Installation von Programmpaketen und für Konfigurationsarbeiten, **niemals** jedoch zum regulären Arbeiten![a]

Zum regulären Arbeiten in einer komfortablen Umgebung (z.B. graphische Benutzeroberfläche) ist ein normaler Benutzer-Account vollkommen ausreichend, und in vieler Hinsicht im Arbeitskomfort dem Administratoraccount sogar überlegen.

---

[a] Warum?

Wechsel des Benutzerstatus

Mit dem Kommando **su** wechseln Sie in der aktiven Shell zum Status des Systemadministrators. Hierbei werden Sie normalerweise nach dem Passwort des Administrators gefragt, das Sie (unsichtbar) eingeben müssen.

Bei Erfolg verändert sich Ihr Eingabeprompt, und Sie haben in dieser Shell alle Rechte des Administratoraccounts (welcher auf den meisten Unix-Systemen die Benutzerkennung **root** hat).

Vorsicht: ab diesem Zeitpunkt können falsch eingegebene Kommandos, die als normaler Benutzer harmlos sind, in dieser Shell zerstörerische Wirkung auf Ihr System haben!

Mit **`exit`** können Sie die root-Shell wieder verlassen (Kontrolle mit **`id`**).

sudo und **id**

Mit dem Kommando **id** können Sie Ihren momentanen Status, Kennung und Gruppenzugehörigkeit feststellen.

Mit Benutzer- und Gruppenrechten auf Dateisystemebene werden wir uns noch ausführlich im Abschnitt „Dateirechte“ beschäftigen.

Das Kommando **sudo** erlaubt, gesteuert durch die Konfigurationsdatei **/etc/sudoers**, das Ausführen eines einzelnen Kommandos mit anderer Benutzerkennung, ohne die Benutzerkennung in dieser Shell dauerhaft zu wechseln. (Beispiel: **sudo id** führt das Kommando **id** als root aus.)

Intermezzo: Text-Dateien unter Unix bearbeiten

Da die meisten systemweiten Konfigurationsdateien unter Linux/Unix einfache Textdateien im Verzeichnis **`/etc`** sind, lohnt es sich, an dieser Stelle auf Texteditoren einzugehen.

`nano`	Sehr primitiver Kommandozeilen-Texteditor
`xedit`	Sehr primitiver graphischer Texteditor
`leafpad`	Graphischer Texteditor im LXDE-Desktop
`emacs`	„Programmier"-Texteditor im Textmodus
`vi`	Kommando-orientierter Texteditor im Textmodus
`soffice`	LibreOffice Textverarbeitung, für einfache Textdateien meist „Overkill".

vi – Seitenorientierter Texteditor

- Kompakter, schneller Texteditor,
- gehört zum Standard-Equipment auf jedem Unix-System,
- kennt *Kommandomodus* (Befehlseingabe) und *Insert*-Modus (Texteingabe),

Vorsicht: Direkt nach dem Start befindet sich der vi im *Kommandomodus*, d.h. jede Tastatureingabe wird als **Kommando** interpretiert, nicht als Eingabetext!

vi – Kommandomodus

<**Escape**>	Rückkehr vom Insert- in den Kommandomodus
i	Wechsel in den **I**nsert-Modus (Direkteingabe)
o	**O**pen: Neue Zeile anfügen ☞ Insert-Modus
dd	**D**elete: Aktuelle Zeile löschen
p	**P**aste: (Gelöschten) Text einfügen
x	Zeichen, auf dem der Cursor steht, löschen
:r Datei	**R**ead: Datei ab Cursor einfügen
:w	**W**rite: Datei speichern
:q	**Q**uit: `vi` beenden
:wq	**W**rite and **Q**uit: Speichern und Beenden
:q!	Beenden ohne Speichern

:%s/alt/neu/gc Interaktiv `alt` durch `neu` ersetzen

Fast alle Kommandos lassen sich gruppieren oder mit einer vorangestellten Zahl mehrfach ausführen.

Quick-Admin-Edit

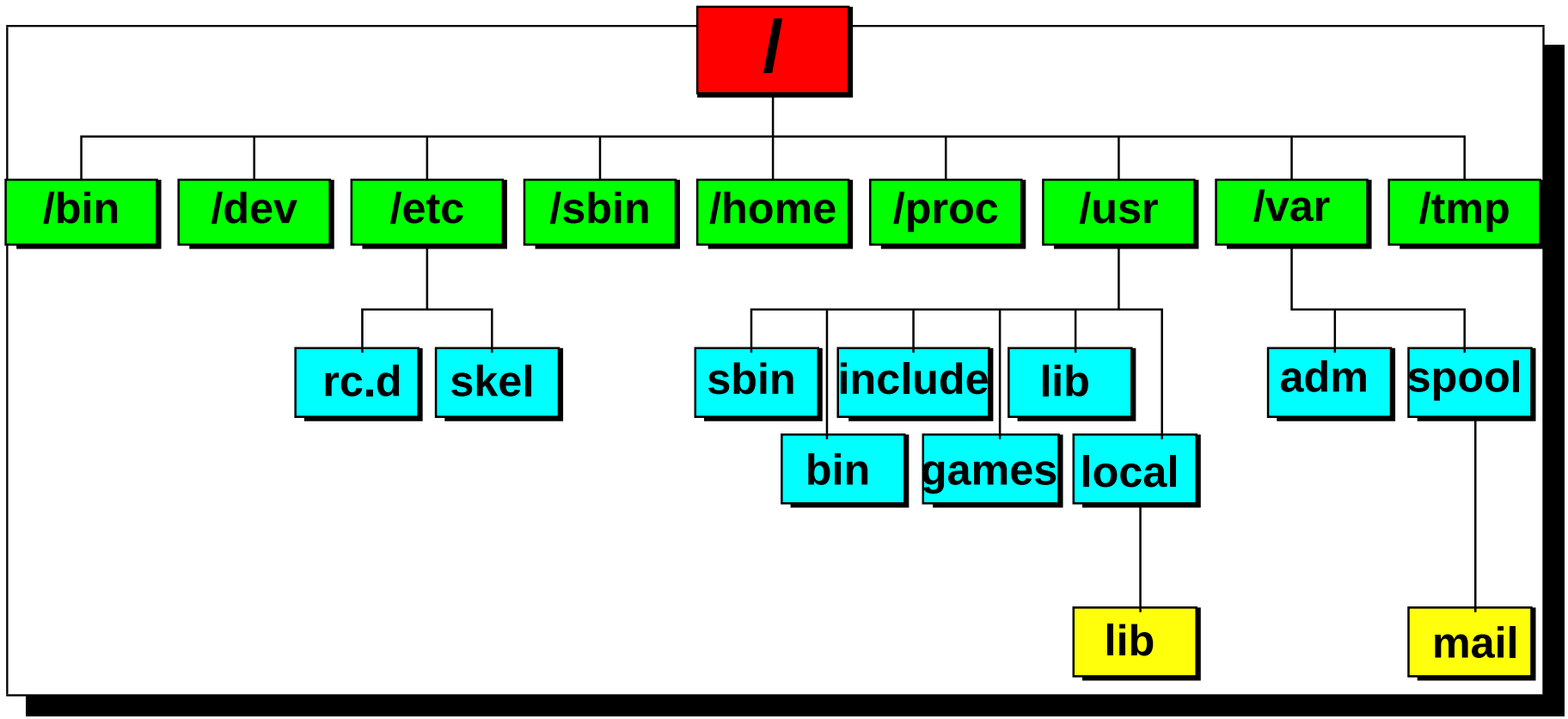
Um systemweite Konfigurationsdateien zu bearbeiten, kann dem Editor-Aufruf ein **sudo** vorangestellt werden, so dass der Editor unter **root**-Kennung läuft und somit Schreibrechte auf die jeweilige Datei hat.

```
sudo vi /etc/sudoers
```

oder

```
sudo leafpad /etc/sudoers
```

Eine Reise durch das Unix-Dateisystem



„Everything is a file“

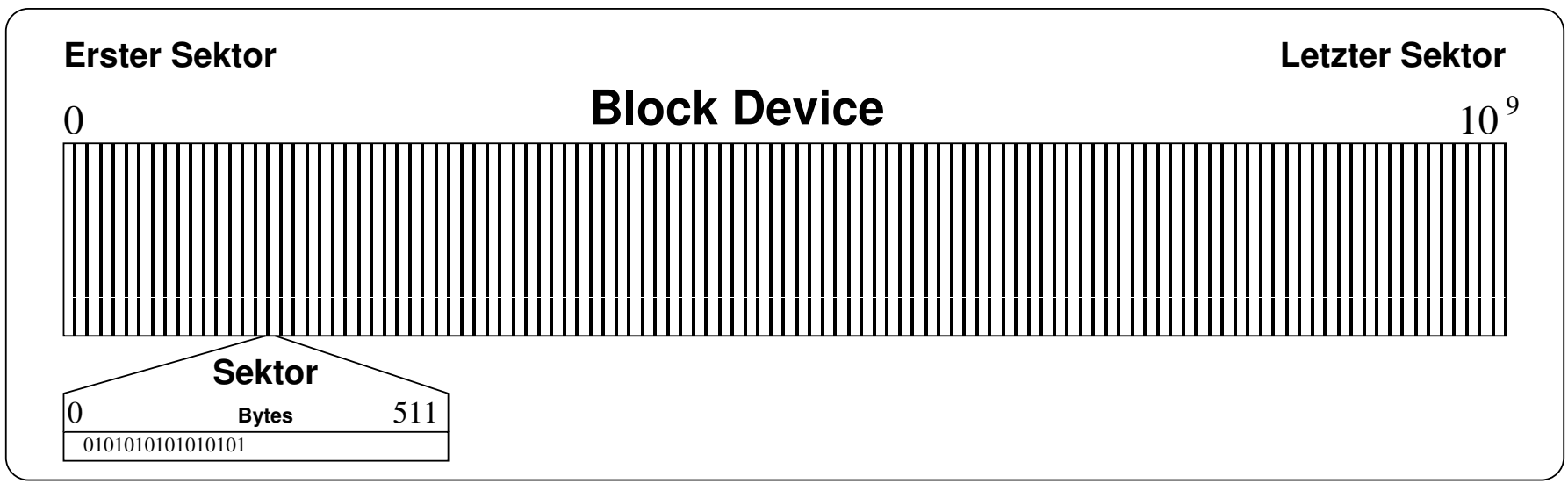
- Gewöhnliche Dateien (Daten, Texte, Konfigurationsdateien...)
- Verweise ☞ „Links“ (hard, soft)
- Spezialdateien: **Geräte** (meist in **/dev**), named Pipes, Unix-Domainsockets, ...

Beispiel: Kopieren einer Daten-DVD in eine Image-Datei

```
$ cp /dev/sr0 dvd.iso
```

Was ist ein „Block-Device“?

- Prinzipiell ein Bereich fester Größe, auf dem Daten untergebracht sind,
- keinerlei sichtbare „Struktur“, abgesehen von der Unterteilung in Sektoren (physikalisch) und „Blöcke“ (logisch) konstanter Größe,
- kann z.B. eine Datei (loopback-File), eine Partition (partition), oder ein Datenträger (disk) sein.



Block Devices

Beispiel (Festplatte an SATA-Controller):

```
knopper@Koffer:~$ ls -l /dev/sda*

brw-rw---- 1 root disk 3, 0 2006-03-25 14:31 /dev/sda
brw-rw---- 1 root disk 3, 1 2006-03-25 14:31 /dev/sda1
brw-rw---- 1 root disk 3, 2 2006-03-25 14:31 /dev/sda2
```

`fdisk` / `gdisk` – Partitionieren

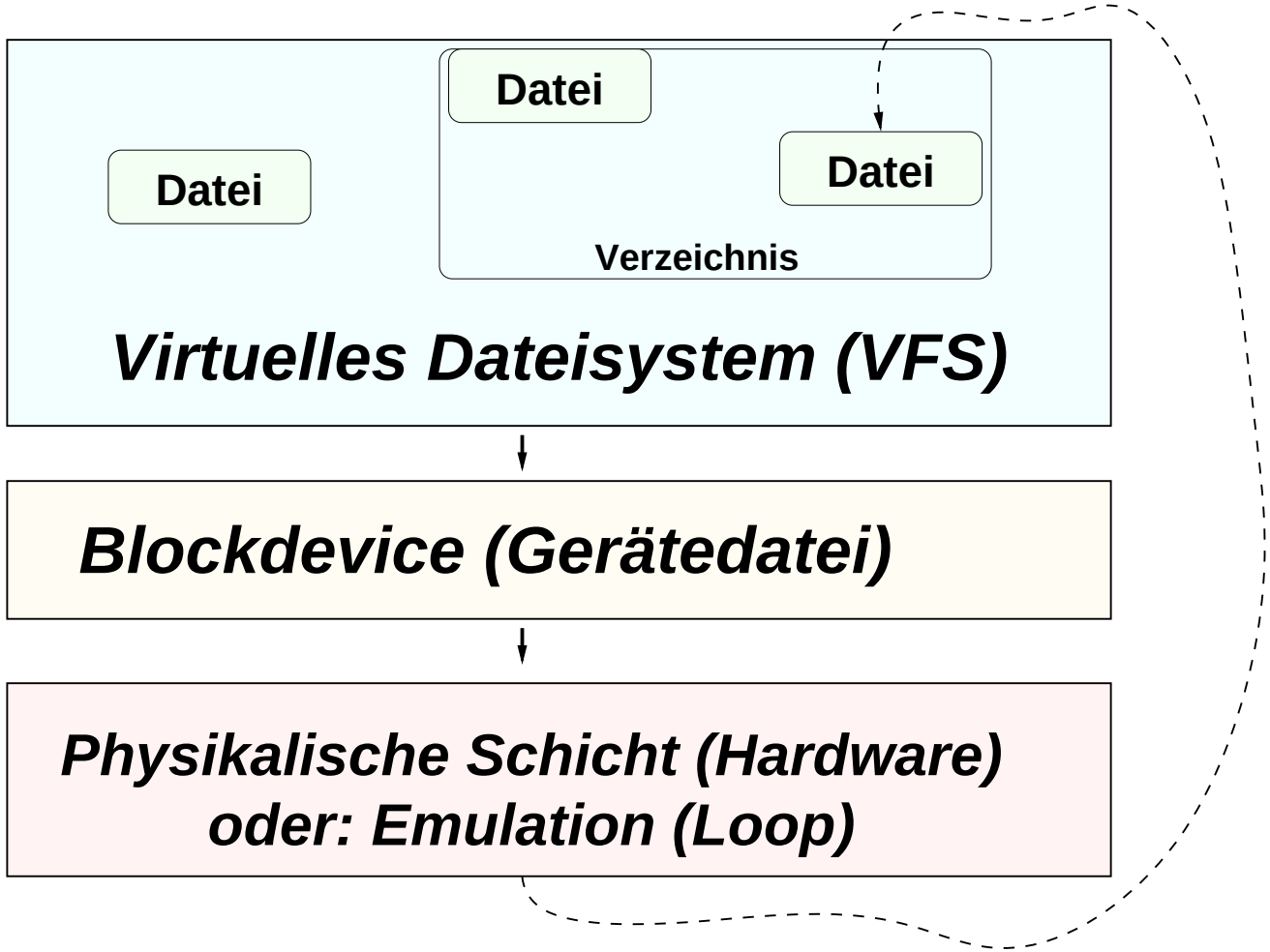
Festplatten werden als **Block Devices** im `/dev`-Verzeichnis angelegt (s.a. Handout „Datentraegerverwaltung„), z.B. `/dev/sda`. Um Bereiche auf dem Datenträger flexibel verschiedenen Aufgaben zuordnen zu können, werden Festplatten meist durch **Partitionierung** unterteilt:

`fdisk /dev/sda` ☞ MBR-Partitionierung

`gdisk /dev/sda` ☞ GUID-Partitionierung

`gparted` – graphisches Werkzeug zur Partitionierung

Vom Blockdevice zum Dateisystem



mount

Syntax: **mount -t dateisystemtyp** \
-o optionen,... \
blockdevice \
zielverzeichnis

Aufgabe: Abbilden der „unstrukturierten“ Daten eines Block Device in eine Verzeichnisstruktur.

Hierbei sind die Optionen und ihre Wirkung hochgradig Dateisystemtyp-spezifisch!

Unterstützte Dateisysteme (1)

Native (blockdevice-basiert)

ext2	Standard
ext3/ext4	ext2 mit Journal-File
reiserfs	Journaling, b-tree
jfs	Journaling, b-tree (IBM)
xfs	Journaling, b-tree (SGI)
btrfs	Journaling, b-tree (neu)
iso9660	ISO/Rockridge/Joliet/zISO
minix	heute eher akademisches FS

Unterstützte Dateisysteme (2)

	RAM/Flash/Packet
jffs2	Journaling Flash Filesystem
romfs	ROM-FS
cramfs	komprimiertes ROM-FS
ramfs	experimentelles Ramdisk-FS
tmpfs	skalierendes Ram-FS
udf	DVD/CDRW-Packetmode FS (u.U. RW)

Unterstützte Dateisysteme (3)

	Compatibility
adfs	Acorn (Archimedes) Disc Filing System
affs	Commodore Amiga Fast File System
hfs	"Hierarchical File System" (MAC)
bfs	"Boot File System" (SCO Unix)
efs	Altes IRIX (SGI) Dateisystem
vxfs	VERITAS VxFS (SCO UnixWare)
hpfs	"High Performance FS" (OS/2)
qnx	QNX FS (vorwiegend RO)
sysv	XENIX/SCO/Coherent+PDP11
ufs	BSD/SunOS/NeXTstep FS
vfat	Altes und neues MSDOS/FAT* Format
ntfs(-3g)	NTFS ro (rw)

Unterstützte Dateisysteme (4)

	Virtuelle
proc	virt. Kernel-Filesystem (wichtig!)
sysfs	virt. Kernel-Filesystem (wichtig!)
devpts	virt. Terminals
usbfs	USB-spezifisch (veraltet)
capifs	ISDN-Subsystem

Unterstützte Dateisysteme (5)

Network

nfs	Network Filesystem v2/3/4 (Client+Server)
cifs	Windows File Server / SAMBA (Client)
sshfs	Verschlüsselte Übertragung per SSH

`mount` – Einbinden von Dateisystemen

`mount` aktiviert die Kernel-Schnittstelle, die über das **Dateisystem** die auf Partitionen vorhandenen Rohdaten in Ordner, Verzeichnisse und Dateien abbildet. Im Zielverzeichnis werden die Daten in einer hierarchisch-logischen Struktur sichtbar.

```
$ mount -t ext2 /dev/sda2 /media/sda2

$ mount -t iso9660 -o ro /dev/sr0 /media/sr0

$ mount /media/sr0

$ mount pizza:/media/sr0 /hosts/pizza/cdrom
```

/etc/fstab

In der Konfigurationsdatei **/etc/fstab** kann eine Zuordnung zwischen Partition und Verwendungszweck konfiguriert werden, die mit **mount -a** automatisch beim Systemstart aktiviert wird.

```
# <device>    <mountpoint>   <type>  <options>         <dump> <fsck>

/dev/sda2     /              ext2    defaults          1      1
/dev/sda3     /usr           ext2    defaults          1      2
/dev/sda5     /home          ext2    defaults          1      3
/dev/sda6     none           swap    defaults          0      0
/dev/sr0      /media/sr0     iso9660 user,ro,noauto    0      0
none          /proc          proc    defaults  0       0
```

umount Verzeichnis

Mit **umount Verzeichnis** wird eine Zuordnung zwischen der Verzeichnisstruktur-Abbildung und dem angesprochenen Datenträger wieder gelöst. **Dies funktioniert nur, wenn nicht mehr auf den Datenträger zugegriffen wird!**

Mit der „Lazy Umount"-Option **-l** wird die Zuordnung automatisch dann gelöst, wenn alle Zugriffe beendet sind. Neue Zugriffe werden bereits auf das alte Verzeichnis (vor **mount**) umgeleitet, damit keine weiteren Blockaden entstehen.

Beispiel: **umount [-l] /media/sr0**

hängt ein eingebundenes DVD-Rom aus. Erst dann kann der Datenträger wieder entfernt bzw. aus dem Laufwerk genommen werden.

chown – Setzen des Dateibesitzers

```
chown [Optionen] Benutzer Datei(en)...
```

`chown` ändert das Besitzer-Attribut von Dateien und Verzeichnissen. Der `chown`-Befehl kann auf POSIX-konformen Unix-Systemen nur vom Systemadministrator ausgeführt werden.
Der ursprüngliche Besitzer der Datei verliert mit sofortiger Wirkung die Besitzer-Rechte an dieser Datei und kann nur noch aufgrund gesetzter Gruppen- oder globaler Rechte auf die Datei oder das Verzeichnis zugreifen.

```
chown -R demo /home/demo
```

Mit der Option `-R` kann rekursiv das Besitzerattribut ganzer Verzeichnisbäume geändert werden.

`chgrp` – Ändern der Gruppenzugehörigkeit

`chgrp [Optionen] Gruppe Dateien...`

`chgrp` ändert die Unix-Gruppe von Dateien und Verzeichnissen. Der Befehl kann vom Besitzer einer Datei ausgeführt werden, wenn er selbst Mitglied der angegebenen Unix-Gruppe ist (POSIX).

```
$ ls -l helloworld.c
-rw-r--r-- 1 knopper users     29 Aug 5 22:39 helloworld.c
$ groups
users developer
$ chgrp developer helloworld.c
$ ls -l helloworld.c
-rw-r--r-- 1 knopper developer 29 Aug 5 22:39 helloworld.c
```

`chmod` – Ändern von Rechten

```
chmod [Optionen] Änderungen Dateien
```

`chmod` ändert die Zugriffsrechte von Dateien und Verzeichnissen. Man kann die **Rechte**

r = read lesen	w = write schreiben	x = execute ausführen	s = suid set ID

an bestimmte **Personenkreise** vergeben

u = user Besitzer	g = group Gruppe	o = others Andere

Mit der Option `-R` werden die Änderungen auch für Unterverzeichnisse durchgeführt.

Beispiele zu **chmod**

```
$ ls -l
total 11
-rw-r--r--  1 knopper  users  7185 Nov 20 23:17 auswertung.sh
-rw-------  1 knopper  users   938 Nov 20 23:17 juli.dat
-rw-------  1 knopper  users   469 Nov 20 23:17 juni.dat
-rw-------  1 knopper  users    54 Nov 20 23:17 mai.dat

$ chmod u+x auswertung.sh
```

Das Script `auswertung.sh` wird zum Ausführen freigegeben.

Beispiele zu **chmod**

```
$ chmod og+r *.dat
$ ls -l
total 11
-rwxr--r--  1 knopper  users  7185 Nov 20 23:17 auswertung.sh
-rw-r--r--  1 knopper  users   938 Nov 20 23:17 juli.dat
-rw-r--r--  1 knopper  users   469 Nov 20 23:17 juni.dat
-rw-r--r--  1 knopper  users    54 Nov 20 23:17 mai.dat
```

Alle dürfen ab jetzt die „`.dat`“-Dateien lesen.

Spezielle Dateiattribute

Neben den Standard-Rechten Lesen, Schreiben und Ausführen existieren noch weitere Dateiattribute, die vom Besitzer einer Datei oder vom Systemadministrator gesetzt werden können.

```
$ chmod u+s /usr/bin/cdrecord
$ ls -l /usr/bin/cdrecord
-rwsr-xr-x 1 root root 13956 May 10 17:31 /usr/bin/cdrecord
```

Durch das Setzen des s-Attributes ("s-Bit") für den Besitzer bzw. die Gruppe einer Datei wird beim Ausführen der Datei die Benutzerkennung bzw. die Gruppe des neuen Prozesses auf den Besitzer bzw. die Gruppe der Datei gesetzt. (Nur so können übrigens Programme wie **`su`** und **`sudo`** funktionieren.)

Hinweis: Dateirechte und Sicherheit

Unter Unix steht und fällt die lokale Sicherheit und Stabilität mit der angemessenen Rechtevergabe für Systemdateien. Unix ist sehr robust gegen Viren und Schadsoftware, da nur der Administrator Schreibrechte auf Systemdateien hat.

Andererseits wird der Zugriff auf Hardware durch spezielle Benutzergruppen streng reglementiert, daher ist auf manchen Systemen für unprivilegierte Anwender kein Zugriff auf Soundkarte oder Festplatten, außer bereits für Benutzer freigegebene eingebundene Datenträger, möglich.

Dateitypen unter Unix/Linux

In diesem Beispiel wurde ein Verzeichnis namens **`test`** angelegt, in dem sich verschiedene Dateiarten befinden.

```
knopper@Koffer:~$ ls -l test/
insgesamt 8
brw-rw---- 1 root    floppy 2,  0 2006-04-04 12:06 blockdevice
crw-rw---- 1 root    root  10,  1 2006-04-04 12:06 chardevice
-rw-r--r-- 2 knopper users      5 2006-04-04 13:14 datei.txt
drwxr-xr-x 2 knopper users     48 2006-04-04 13:17 directory
prw-r--r-- 1 knopper users      0 2006-04-04 13:16 fifo
-rw-r--r-- 2 knopper users      5 2006-04-04 13:14 hardlink
srwxrwxrwx 1 knopper users      0 2006-04-04 10:06 socket
lrwxrwxrwx 1 knopper users      9 2006-04-04 13:14 symlink
                                                   -> datei.txt
```

Der erste von **`ls -l`** angezeigte Buchstabe in den Dateirechten kennzeichnet die Art der Datei.

Einfache Dateien

Einfache Dateien können Dokumente, Programme, Bibliotheken oder Daten jedweder Art sein.

```
-rw-r--r-- 2 knopper users     5 2006-04-04 13:14 datei.txt
```

Unter Windows werden bestimmten Dateiendungen Dateitypen und somit Programme zum „Öffnen“ zugeordnet, während unter Unix zwar einige Dateiendungen üblich, aber nicht zwingend erforderlich sind.

Z.B. werden unter Unix ausführbare Programme aufgrund ihres internen Aufbaus und am gesetzten „Ausführen“-Dateirecht erkannt, während sie unter Windows die Endung **`.exe`** besitzen müssen.

„Versteckte“ Dateien

Unter Unix ist es Konvention, d.h. in vielen Dateimanagern und auch im **`ls`**-Kommando „eingebaut“, dass Dateien mit einem führenden „**`.`**“ standardmäßig nicht im Verzeichnislisting auftauchen (sog. „versteckte“ Dateien). Dies wird speziell für Konfigurationsdateien und -verzeichnisse im Benutzer-Heimverzeichnis verwendet, die ansonsten das Dateilisting um hunderte Einträge verlängern würden.

Unter DOS und Windows gibt es hingegen Dateiattribute, die Dateien als „hidden“ kennzeichnen, und die mit dem Kommando **`attrib.exe`** verwaltet werden. Speziell unter NTFS ist **`ADS`** (Alternate Data Streams) ein Feature, das es erlaubt, Dateien unnerhalb anderer Dateien zu „verstecken“, was wahrscheinlich im Design für Konfigurationsdateien, Bibliotheken und andere Zugehörigkeiten gedacht ist, die zusammen mit einem Programm oder einer Multimedia-Datei „mitgeliefert“ werden sollen. Leider bietet dieses Feature Angriffspunkte für Schadsoftware, die so z.B. Code innerhalb anderer Dateien verstecken können.

Verzeichnisse

Verzeichnisse und Unterverzeichnisse sind ein Ordnungsmittel, um Dateien zu kategorisieren und leichter wieder auffindbar zu machen.

```
drwxr-xr-x 2 knopper users    48 2006-04-04 13:17 directory
```

Symlinks

Symbolische Links, also „Namensverknüpfungen“ , sind Zeiger auf Datei- oder Verzeichnisnamen, die den Zugriff auf die entsprechenden Daten unter einem anderen Namen ermöglichen. Sie werden mit dem Kommando

`ln -s datei verknüpfung`

angelegt. Der symbolische Link und sein Ziel ist bei **`ls -l`** deutlich identifizierbar.

```
-rw-r--r-- 2 knopper users 5 2006-04-04 13:14 datei.txt
lrwxrwxrwx 1 knopper users 9 2006-04-04 13:14 symlink -> datei.txt
```

Achtung: Wird die Originaldatei gelöscht, so ist der Inhalt auch nicht mehr über den symbolischen Link verfügbar, obwohl dieser vorhanden bleibt.

Hardlinks

Ähnlich wie beim Symbolischen Link auf den NAMEN einer Datei verwiesen wird, wird beim Hardlinks auf den Datei-INHALT verwiesen. D.h. es wird mit

`ln datei hardlink`

eine neue Datei angelegt, deren Inhalt aber mit dem der ersten Datei immer identisch ist. Aus dem Verzeichnislisting ist die Tatsache, dass es sich um einen Hardlink handelt, allerdings nicht ohne weiteres erkennbar.

```
-rw-r--r-- 2 knopper users     5 2006-04-04 13:14 datei.txt
-rw-r--r-- 2 knopper users     5 2006-04-04 13:14 hardlink
```

Wird eine der beiden Dateien gelöscht, so ist der Inhalt nach wie vor unter dem anderen Dateinamen verfügbar. Originaldatei und Hardlink sind also "gleichberechtigt".

Character und Block Devices

Character Devices erlauben sequentielles, zeichenweises Schreiben und Lesen von Daten auf die damit verbundenen Geräte (vergl. voriges Kapitel über Dateisysteme und Blockdevices).
Block Devices erlauben wahlfreien Zugriff auf beliebige „Blöcke“ eines Gerätes, ohne dass ein „Vor-“ oder „Zurückspulen“ nötig ist.

```
brw-rw---- 1 root    floppy 2,  0 2006-04-04 12:06 blockdevice
crw-rw---- 1 root    root  10,  1 2006-04-04 12:06 chardevice
```

Block Devices bilden üblicherweise Festplatten ab, während Char Devices Mäuse oder Bandlaufwerke und Modems abbilden.

Fifos und Sockets

Während Sockets mit Netzwerkoperationen (z.B. Kommunikation mit dem X-Server) verknüpft sind, sind Fifos ein Mittel, um die Ein- und Ausgabe von Programmen miteinander zu verknüpfen, ähnlich wie mit dem Pipe-Symbol | .

```
prw-r--r-- 1 knopper users     0 2006-04-04 13:16 fifo
srwxrwxrwx 1 knopper users     0 2006-04-04 10:06 socket
```

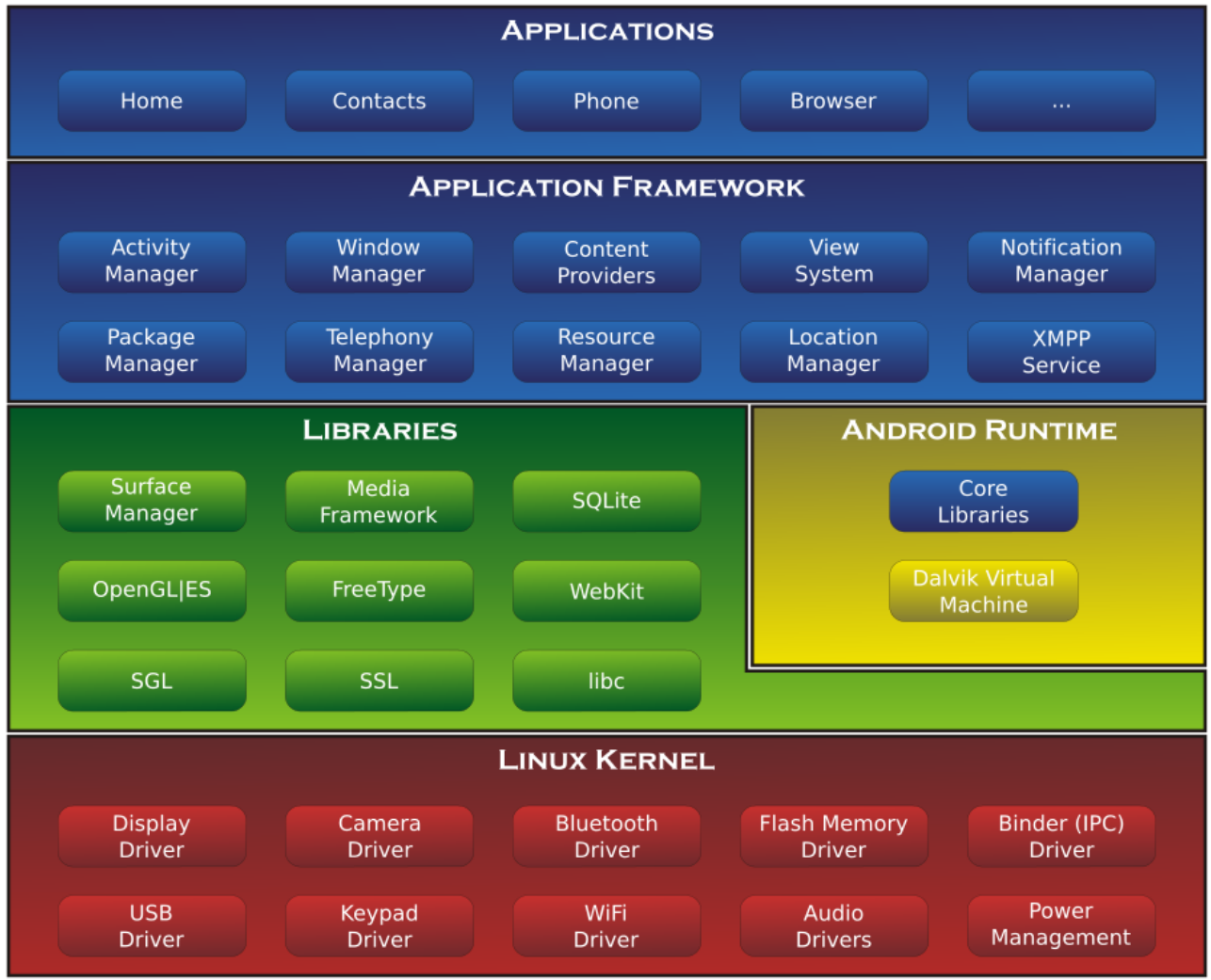
Der Linux-Kernel

Kernel = Schnittstelle zwischen Hardware und Anwendersoftware

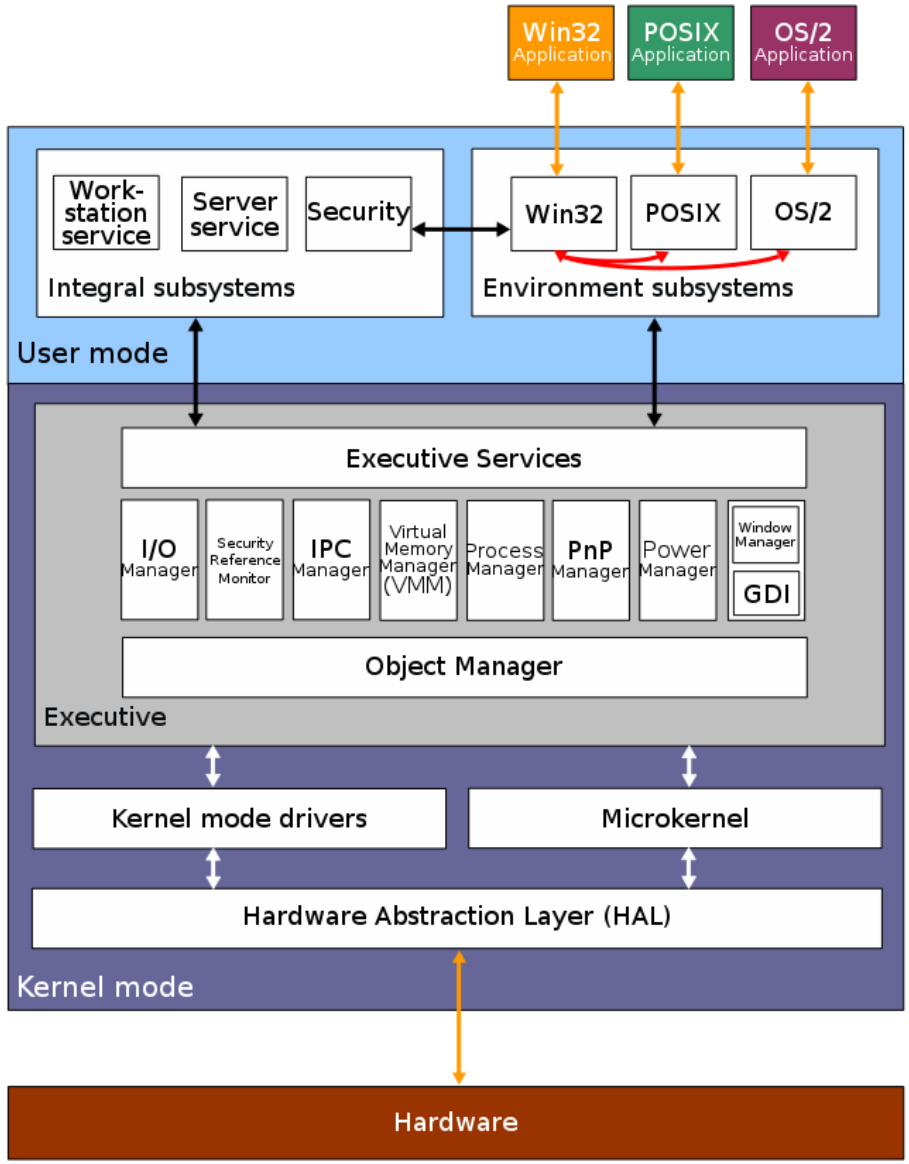
- Open Source Projekt mit mehreren Tausend Entwicklern, Projektleitung der Mainline durch Linus Torvalds,
- Versionsnummer Zusammensetzung: *Hauptversion(4).Major_Release.Minor_Release* (neues Major-Release etwa alle 4-6 Wochen)
- „Statischer" Teil: Enthält die zum Start von Bootmedium notwendigen Komponenten
- „Dynamischer" Teil: Module mit Hardwareunterstützung für Grafik, Sound, Netzwerk, Peripheriegeräte
- Nicht Teil des Kernels, aber oft angehängt: Ramdisk (Mini-System) mit Startup-Code

Android Systemarchitektur

Der Linux-Kernel ist auch Teil des Android-Laufzeitsystems, das auf anwendungsorientierte mobile Geräte optimiert ist.



Vergleich: Windows Architektur (NT Kernel, W2K)



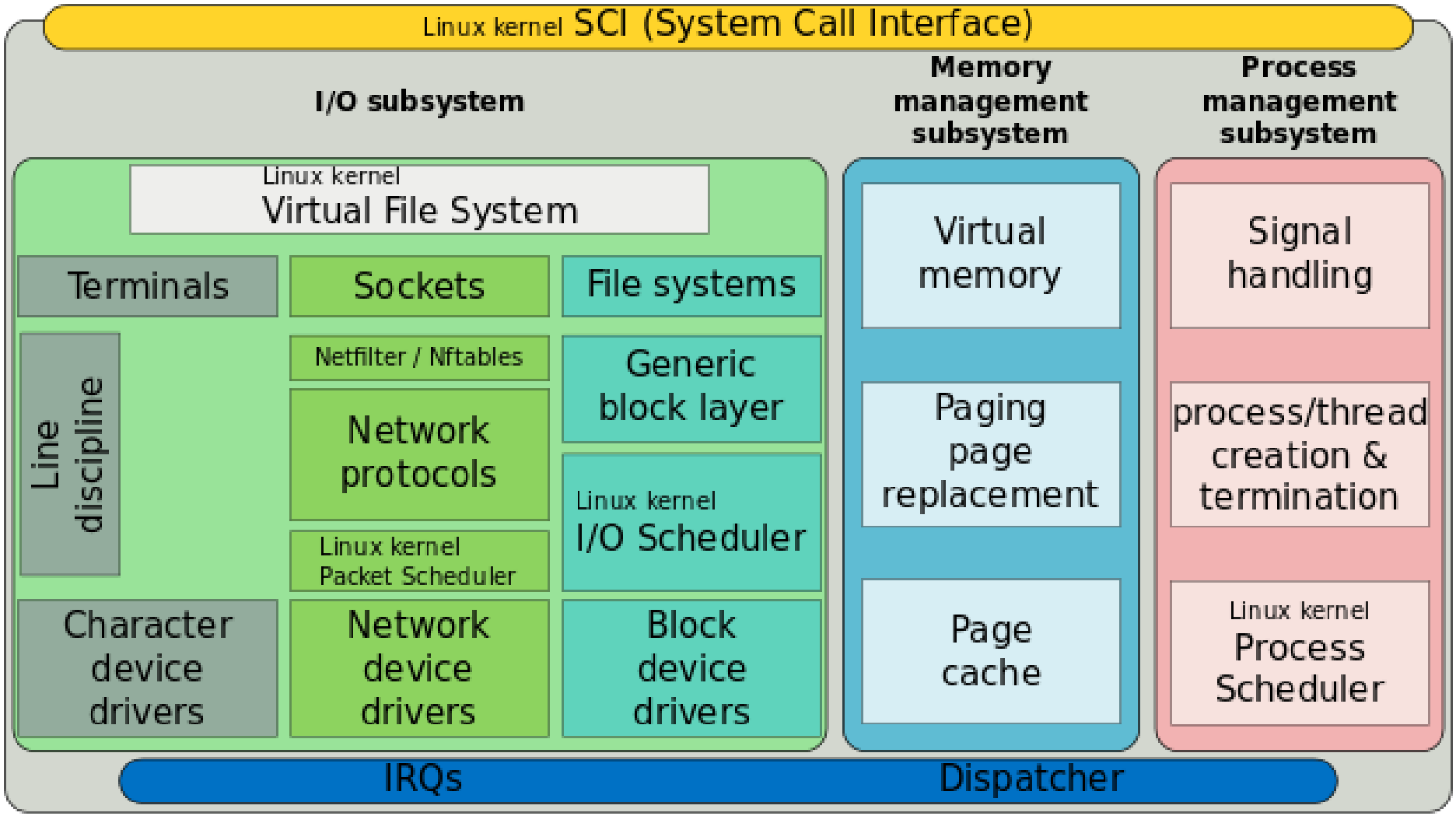
„Shared Code“-Konzept: Bibliotheken

Ein Großteil des Codes, der für die Anwendersoftware verwendet wird, ist nicht in der Software selbst, sondern in Bibliotheken untergebracht, die zwischen verschiedenen Anwendungen geteilt werden (auch gemeinsam genutzter Code dauerhaft im Hauptspeicher ☞ Effizienz).

`ldd /usr/bin/lxterminal`

```
linux-gate.so.1 (0xf773d000)
/lib/ld-linux.so.2 (0x56624000)
libdl.so.2 => /lib/i386-linux-gnu/libdl.so.2 (0xf6247000)
libm.so.6 => /lib/i386-linux-gnu/libm.so.6 (0xf65c7000)
libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xf66b7000)
libselinux.so.1 => /lib/i386-linux-gnu/libselinux.so.1 (0xf63e7000)
libglib-2.0.so.0 => /lib/i386-linux-gnu/libglib-2.0.so.0 (0xf75bf000)
libX11.so.6 => /usr/lib/i386-linux-gnu/libX11.so.6 (0xf68a7000)
libXinerama.so.1 => /usr/lib/i386-linux-gnu/libXinerama.so.1 (0xf65af000)
[...]
```

Kernel, Devices, Scheduler



„Timesharing“ - Scheduler

Um mehrere Programme und Systemdienste, sog. „Prozesse“, quasi-parallel ablaufen zu lassen, obwohl nur eine oder wenige reale Recheneinheiten (CPUs) vorhanden sind, wird das ☞ **Multitasking**-Konzept eingesetzt.

Der **Prozess-Scheduler** im Kernel verwaltet quasi-parallel (oder tatsächlich parallel bei Multiprozessor-Systemen) ablaufende Programme im Multitasking-Betrieb in Form einer ringförmigen Warteschlange (zyklische Verteilung der Rechenzeit).

Arten des Multitasking

Kooperatives Multitasking

Ein Prozess, der die CPU besitzt, muss nach Verrichtung einer Teilaufgabe die CPU exlizit wieder freigeben, damit der nächste Prozess bearbeitet werden kann.

Präemptives Multitasking

Ein Prozess, der die CPU besitzt, wird dann, wenn er nicht von selbst die CPU freigibt, vom Scheduler nach einer festgelegten Zeit angehalten. Der nächsten Prozess in der Wartenschlange bekommt daraufhin die CPU zugeteilt.

Präemptibles Multitasking

Wie Präemptives Multitasking, jedoch können Prozesse mit höhere Priorität den Scheduler anweisen, Prozesse mit niedriger Priorität zu ihren Gunsten zu unterbrechen, wenn ein vorher bestimmtes Ereignis auftritt (wird von OS/2 und Linux ab Kernel 2.6 unterstützt).

Präemptives und Präemptibles Multitasking setzen zwingend einen Zeitgeber voraus, der dem Scheduler die notwendige Rechenzeit zuteilt, um die Prozesswarteschlangen verwalten zu können.

Resource-Sharing

Neben der CPU wird unter Unix auch der Datentransport von und zu Datenträgern und über Netzwerke durch einen **Scheduler** verwaltet, in diesem Fall ein sog. **IO-Scheduler**, der Warteschlangen (engl. queue) verwaltet. Für diesen können Strategien festgelegt werden, nach denen je nach Anwendungsfall besonders performant ein „flüssiges" Systeverhalten eingestellt werden kann. Durch Steuerungsdateien im **`/sys`**-Dateisystem unter Linux kann das Verhalten des IO-Schedulers für verschiedene Geräte individuell vom Administrator eingestellt werden.

```
cat /sys/block/sdb/queue/scheduler
 noop deadline [cfq]
echo deadline > /sys/block/sdb/queue/scheduler
```

Prozess- und Jobverwaltung

Sowohl dem Benutzer, als auch dem Administrator stehen die Befehle **`ps`** und **`kill`** zur Verfügung, um den Prozess-Status zu erfragen und einem Prozess, der durch eine im System eindeutige Nummer, die **Prozess-ID** gekennzeichnet ist, **Signale** zu schicken. Insofern ist der Name **`kill`**, der lediglich ein Signal verschickt, etwas missverständlich, da er keineswegs in jedem Fall den angegebenen Prozess beendet.

`ps` – Prozessinformationen anzeigen

`ps` `[Optionen]`

`ps` zeigt die Liste der laufenden Prozesse (=Programme) an. Das Kommando ist insbesondere im Zusammenspiel mit `kill` sehr praktisch, um die Prozess-ID „amoklaufender" Programme zu erfahren und diese „gewaltsam" zu beenden.

```
$ ps auxwww
USER       PID %CPU %MEM  SIZE   RSS TTY STAT START   TIME COMMAND
root         1  0.0  0.0   796   128  ?  S    Jun 12   0:58 init
root         2  0.0  0.0     0     0  ?  SW   Jun 12   3:45 (kflushd)
root         3  0.0  0.0     0     0  ?  SW   Jun 12   0:00 (kpiod)
root         4  0.0  0.0     0     0  ?  SW   Jun 12   6:53 (kswapd)
root       296  0.2  0.0   820   188  ?  S    Jun 12 101:09 syslogd
knopper   4832  0.1  0.8  3044  2148  pf S     23:09   0:30 gv unixkurs.ps
knopper   4936  0.0  0.3  1436   996  q3 S     23:15   0:01 vi unixkurs.tex
```

`kill` – Signal an Prozesse schicken

`kill` `[Signal] Prozeßnummer`

`kill` versendet Signale an einen laufenden Prozess. Wenn `kill` ohne die Signal-Option ausgeführt wird, wird der angeführte Prozeß mit dem Signal `TERM` beendet. Die Signal-Option `-HUP` (Hang Up) gibt dem Programm die Möglichkeit, sich „sauber" zu beenden, und dient bei einigen Systemprozessen (daemons) dazu, Konfigurationsdateien neu einzulesen. Bei hartnäckigeren Fällen hilft das Signal `-KILL`, gegen das sich kein Prozess wehren kann.

$ kill 1234	beendet den Prozeß 1234
$ kill –KILL 1234	beendet den Prozeß, falls der obere Versuch erfolglos bleibt.

Übung

1. Starten Sie das Programm
   `xclock -update 1 &`
   in der Shell als *Hintergrundprozess*.
2. Kontrollieren Sie mit dem Shell-internen Kommando **`jobs`** Ihre Hintergrundprozesse.
3. Finden Sie die Prozess-ID des laufenden `xclock` mit Hilfe von **`ps aux | grep xclock`**.
4. Halten Sie den Prozess kurzfristig an, indem Sie ihm mit **`kill -STOP Prozess-ID`** ein STOP-Signal schicken. Beobachten Sie die Reaktion des Programms.
5. Lassen Sie den Prozess weiterlaufen, indem Sie ihm das Signal **`CONT`** schicken.
6. Terminieren Sie den Prozess mit dem Signal **`TERM`**.

Interrupts

Ein Timer-Interrupt wird zur Steuerung des Schedulers („Zeitscheiben“ - Slices) verwendet, um auch „Dauerläufer“ regelmäßig zugunsten anderer Prozesse zu unterbrechen.

Ausgabe der aktiven Interrupts mit Zuordnungen:
`cat /proc/interrupts`

Im Echtzeit-Betrieb („Realtime“) wird die Steuerung so modifiziert, dass bestimmte Prozesse zu definierten Zeiten antworten und entsprechende Priorität vor allen anderen Prozessen haben, was aber nicht zwangsläufig heißt, dass sie dadurch „schneller“ laufen. Beispiel: Steuerungsanlagen.

Bei hardwarenaher Programmierung werden Interrupts von Geräten ausgelöst, z.B. wenn ein Netzwerk-Paket empfangen wurde, so dass die „Treiber“ bzw. Kernel-Module diese Ereignisse quasi-parallel zum interaktiven Betrieb abarbeiten können.

Input/Output

Zur Verwaltung von Ein- und Ausgabedaten werden ebenfalls Warteschlangen verwendet, teilweise eine oder mehrere pro Gerät. Bei ereignisgesteuerter Verarbeitung werden diese Warteschlangen ebenfalls durch Interrupts gesteuert.

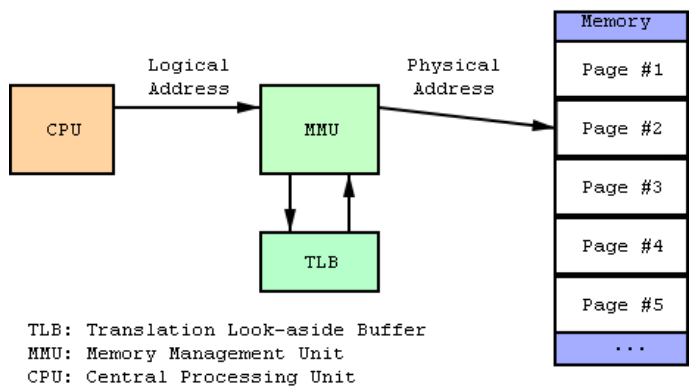
Teilweise versucht Linux, akkumulierte Daten „am Stück“ zu verarbeiten, beispielsweise Segmente zu größeren Datenblöcke zusammenzufassen und auf Harddisk zu schreiben, um den Durchsatz und die Effizienz zu erhöhen.

Bei blockorientierten Datenträgern (Harddisks, Flash-Medien, CD- und DVD-Roms) werden einmal gelesene Daten im Hauptspeicher gehalten, bis dieser für wichtigere Daten benötigt wird. Dadurch fühlt sich das System bei genügend Hauptspeicher „schneller“ an, je öfter die gleichen Daten gelesen werden sollen.

Speicherverwaltung und Virtual Memory (VM)

Moderne Betriebssysteme unterstützen über die Memory Managing Unit (MMU) der Hardware:

- Speicherschutz (Memory Protection),
- Virtuelle Adressierung für jeden Prozess,
- temporäre Auslagerung von Daten aus dem Hauptspeicher (RAM), die längere Zeit nicht benutzt wurden (☞ NRU = „Not Recently Used“) auf Datenträger mittels ☞ Swapping und ☞ Paging.



Speicherschutz und Virtuelle Adressen

Jedem Prozess werden reale Speicherbereiche zugewiesen, die jedoch mit einer Übersetzungstabelle prozessintern in einem „virtuellen Speicher“ abgebildet werden, der durchgängig meist von Speicheradresse 0 bis zum Ende des adressierbaren Speichers reicht, wobei sich jedoch nur tatsächlich gelesene oder geschriebene Speicherzellen im realen Speicherverbrauch niederschlagen.

Da jeder Prozess nur den Speicherbereicht „sieht“, der ihm durch das Memory-Mapping zugewiesen wurde, können Prozesse sich nicht gegenseitig Speicherbereiche überschreiben oder auch nur auslesen. Eine Ausnahme hiervon sind durch spezielle Mechanismen eingeblendete Bereiche, das sogenannte „Shared Memory“.

Es ist auch üblich, dass z.B. Code von geladenen Bibliotheken read-only von mehreren Prozessen an unterschiedlichen virtuellen Speicheradressen eingeblendet und verwendet wird, was die Speichereffizienz erhöht. Virtuelle Maschinen nutzen diesen Mechanismus, um mit ihren zur Verfügung gestellten Mechanismen mehrere Prozesse gleichzeitig zu bedienen, ohne sich für jeden Prozess zu „klonen“.

Speicherauslagerung

Mitunter sind komplexe Programmpakete sehr „speicherhungrig“. Steht weniger Speicher zur Verfügung als vom Programm angefordert, so kann das Betriebssystem das Programm beenden (unter Linux wird ein „Segmentation Fault“-Signal an das Programm geschickt), oder bei eingerichtetem Auslagerungsbereich

1. gerade nicht benutzen Code und Daten aus dem RAM-Speicher auf einen Datenträger auslagern und
2. den so freigewordenen RAM-Speicher dem anfordernden Programm zuweisen.

Swap und Paging

Die Summe aus realem RAM und Auslagerungsbereich wird als virtueller Speicher (Virtual Memory, VM, nicht zu verwechseln mit „Virtueller Maschine“!) bezeichnet. Für das Programm selbst ist der Übergang von „echtem“ zu „ausgelagertem“ Hauptspeicher nicht sichtbar, es wird während dieses Vorgangs angehalten.

Realer Speicher (RAM)	Auslagerungsbereich/e (SWAP)

Ähnlich wie bei Festplattensektoren wird Hauptspeicher bei modernen Betriebssystemen in Blöcke fester Größe, sog. „Pages“ (Speicherseite) organisiert und von der MMU entsprechend „seitenweise“ gelesen und geschrieben.
In dem Moment, in dem ein Programm auf eine ausgelagerte Speicherseite zugreift, wird ein „Page Fault“ ausgelöst, der die MMU veranlasst, die ausgelagerte Speicherseite von einer Festplattenpartition (Swap-Partition) oder aus einer Swap-Datei wieder ins RAM zu laden, bevor das Programm darauf zugreift.

Einrichten von Swap unter Linux

Unter Linux kann eine Partition zur Swap-Partition erklärt und entsprechend mit einer Paging-Signatur „formatiert“ werden:

```
# Setze Partitionstyp von /dev/sdb2 auf 82 (swap)
sfdisk --change-id /dev/sdb 2 82

# Richte Swap auf /dev/sdb2 ein
mkswap /dev/sdb2

# Aktiviere Swap auf /dev/sdb2
swapon /dev/sdb2
```

Mit einem Eintrag `/dev/sdb2 none swap defaults 0 0` in **/etc/fstab** wird die eingerichtete Swap-Partition beim Systemstart automatisch mit **swapon -a** eingebunden.

Statt **/dev/sdb2** in diesem Beispiel könnte auch eine Datei, z.B. **/swap.img** verwendet werden.

Speicher-Monitoring

In der Ausgabe des Prozessliste mit **`ps aux`** wird der virtuelle (angeforderte) und tatsächlich belegte (residente) Speicher für jeden Prozess angezeigt.

Ausgabe von **`free`**:

```
                 total       used       free     shared    buffers     cached
Mem:           2063844    1863828     200016          0      38008     224368
-/+ buffers/cache:      1601452     462392
Swap:          5564124        112    5564012
```

Der gekennzeichnete Werte aus der Zeile „abzüglich Puffer/Cache“ gibt den tatsächlich durch laufende Programme belegten Speicher an.

Eine detailierte Auflistung der Speichernutzung erhält man unter Linux mit **`cat /proc/meminfo`**.

Netzwerk

Die meisten Betriebssysteme sind netzwerkfähig und unterstützen standardmäßig das TCP/IP-Protokoll.

Hierzu müssen einerseits die für jedes Netzwerkgerät (LAN, WLAN) passenden „Treiber“ (Kernel-Module) geladen werden, andererseits sind die Netzwerkprotokolle zur Datenübertragung in einer mittleren Schicht erforderlich.

```
$ lsmod
...
bluetooth      247796  3 ath3k,btusb
ath9k           75729  0
ath9k_common    18158  1 ath9k
ath9k_hw       386299  2 ath9k_common,ath9k
ath             16912  3 ath9k_common,ath9k,ath9k_hw
cfg80211       364859  4 ath,ath9k_common,ath9k
ipv6            63434  0
```

Beispiel: Ethernet-Parameter einstellen

Bringt viele Administratoren MAC-basierter Firewalls zur Verzweiflung:

```
ifconfig eth0 down
ifconfig eth0 hw ether 00:04:23:44:22:11
```

Hiermit wird die „Hardware-Adresse“ von Netzwerkkarten eingestellt.

`ifconfig` – IP-Adresse und Netzmaske

```
ifconfig eth0 192.168.0.1
         netmask 255.255.255.0
         broadcast 192.168.0.255

ifconfig eth0
```

```
eth0  Link encap:10Mbps Ethernet  HWaddr 00:00:C0:68:FB:29
      inet addr:192.168.0.1 Bcast:192.168.0.255 Mask:255.255.255.0
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
      RX packets:0 errors:0 dropped:0 overruns:0
      TX packets:0 errors:0 dropped:0 overruns:0
      Interrupt:5 Base address:0x310 Memory:ca000-cc000
```

route – Netzwerkrouten und Gateway(s)

route add -net 192.168.1.0
netmask 255.255.255.0 dev eth0

Setzt eine Route zum Netzwerk 192.168.1.0 auf die gleiche Netzwerkkarte wie vorher 192.168.0.0. Es muß allerdings ggf. vorher eine zweite lokale IP-Adresse auf dem Interface **eth0:1** gesetzt werden, die diesem Netz entspricht.

route add default gw 192.168.0.254

Setzt das „Tor zur Welt" über den Rechner mit der IP-Adresse 192.168.0.254.

`/etc/resolv.conf` – Nameserver

In der Datei **`/etc/resolv.conf`** werden mit dem vorangestellten Schlüsselwort **`nameserver`** die IP-Adressen der Nameserver angegeben, die befragt werden sollen, wenn der Rechner versucht, einen DNS-Namen aufzulösen (DNS = „Domain Name System“ oder „Service“). Fehlt dieser Eintrag, so kann lediglich über die numerische Adresse, nicht aber über Rechnernamen, auf andere Rechner im Internet zugegriffen werden.
Dynamische Dienste wie **DHCP** (**`pump`**, **`dhclient`**, **`dhcpcd`**) oder **PPP** setzen bei erfolgreichem Verbindungsaufbau automatisch einen gültigen Nameserver in **`/etc/resolv.conf`** ein. Ein Programm, mit der überprüft werden kann, ob der Nameserver korrekt arbeitet, ist **`nslookup`** (interaktiv) oder **`host rechnername`** (nicht-interaktiv).

Übung

1. Setzen Sie mit **`ifconfig`** eine zweite IP-Adresse auf das virtuelle Netzwerkkarten-Interface **`eth0:1`**, welche eine um 1 erhöhte Netzwerkadresse (nicht Host-Adresse!) besitzen soll.
2. Sehen Sie mit **`route`** nach, ob der Kernel automatisch eine Netzwerkroute für dieses Interface angelegt hat.
3. Versuchen Sie mit **`ping`**, das zweite, virtuelle Interface Ihres Tischnachbarn "anzupingen".
4. Sehen Sie mit dem Kommando **`traceroute IP-Adresse`** nach, welchen Weg Ihre IP-Pakete nehmen, wenn Sie einen bestimmten Rechner außerhalb des lokalen Netzes zu erreichen versuchen.

Eigenschaften eines TCP/IP-Paketes

- SOURCE (Herkunfts-) **Adresse**,
- DESTINATION (Ziel-) **Adresse**,
- SOURCE (Herkunfts-) **Port**,
- DESTINATION (Ziel-) **Port**,
- Protokolltyp (**TCP** oder **UDP**).

Der SOURCE-Port auf einem Server kennzeichnet i.d.R. den angesprochenen **Dienst** (s.a. **`/etc/services`**).

Netzdienste starten

- Einen Server-Dienst starten, der sich (gemäß seiner Einstellungen) auf einen bestimmten **Port** bindet, oder
- Mit Hilfe des Internet-Metadämons **`inetd`** einen Dienst oder ein Programm mit einem wählbaren **Port** verbinden (Konfigurationsdatei **`/etc/inetd.conf`**).

Welche Dienste laufen auf meinem Rechner?

```
sudo netstat -tulpen
```

Oder scannen:

```
sudo nmap -O -P0 -sT -sU localhost
```

Wichtige Systeminformationen auslesen (Linux)

Kommando	Info
`cat /proc/interrupts`	Interrupts und Timer
`cat /proc/cpuinfo`	CPU und Kerne / Threads
`free`	Arbeits-Speicher (kompakt)
`cat /proc/meminfo`	Arbeits-Speicher (ausführlich)
`cat /proc/swaps`	Auslagerungs-Speicher (ausführlich)
`df`	Füllstand gemounteter Partitionen
`mount`	Eingebundene Partitionen (mit FS-Opt.)
`ps [auxw]`	Prozesse anzeigen
`top`	Prozesse mit Aktualisierung
`lspci [-v]`	Geräte (PCI(e)) anzeigen
`lspci [-k]`	Kernel-Module für (PCI(e)) Geräte
`lsusb [-v]`	Geräte (USB) anzeigen
`lsmod`	Geladene Module (Treiber) anzeigen
`ifconfig wlan0`	IP-Konfiguration für wlan0 (LAN)
`route [-n]`	Netzwerkrouten für alle Geräte
`cat /etc/resolv.conf`	Nameserver anzeigen

Standard für Betriebssysteme - POSIX

Ursprünglich aus der ☞ Unix-Ecke kommend, wurde 1988 ein Standard für den Aufbau und Eigenschaften verschiedener Betriebssysteme aufgestellt, ☞ POSIX, IEEE 1003.1-1988, auch oft einfach als IEEE 1003 bzw. internationaler Standard ISO/IEC 9945 in der Literatur aufgeführt. Der Standard wurde seitdem mehrmals präzisiert und erweitert.

Neben den meisten Unix-Systemen, zu denen auch Linux gehört, streben auch andere Betriebssysteme POSIX-Kompatibilität an. Windows enthält seit Version „Windows 2000" das ☞ POSIX Subsystem, das POSIX-kompatible Services zur Verfügung stellt und teilweise auch das Portieren von Unix-Anwendungen auf Windows ermöglicht. Eine andere Variante eines POSIX-Layers für Windows ist ☞ Cygwin.

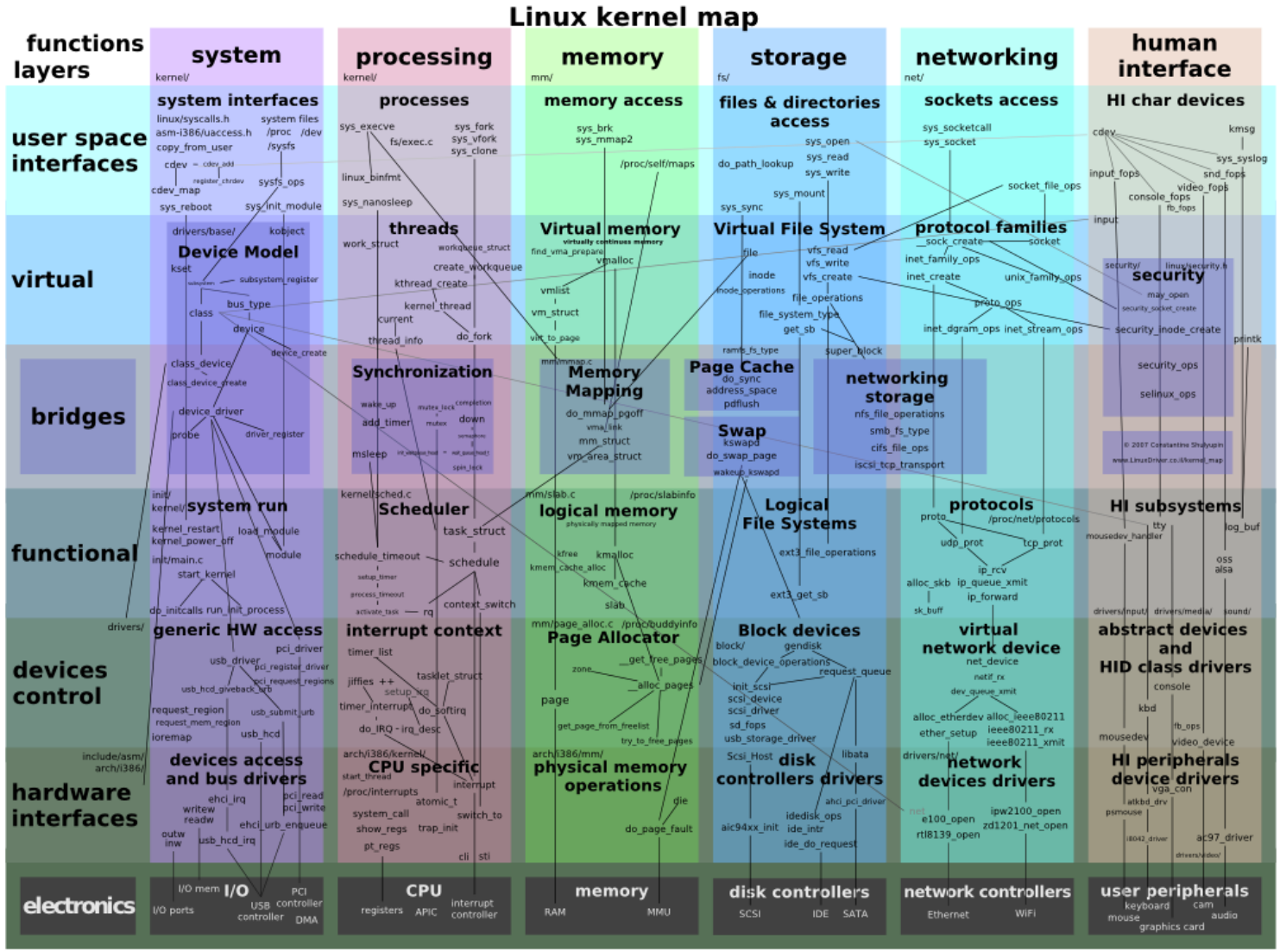
Das „Portable Operating System Interface" spezifiziert API (Application Programmers Interface), Kommandos und Utilities, die in kompatiblen Betriebssystemen vorhanden sein sollen.

Kernel

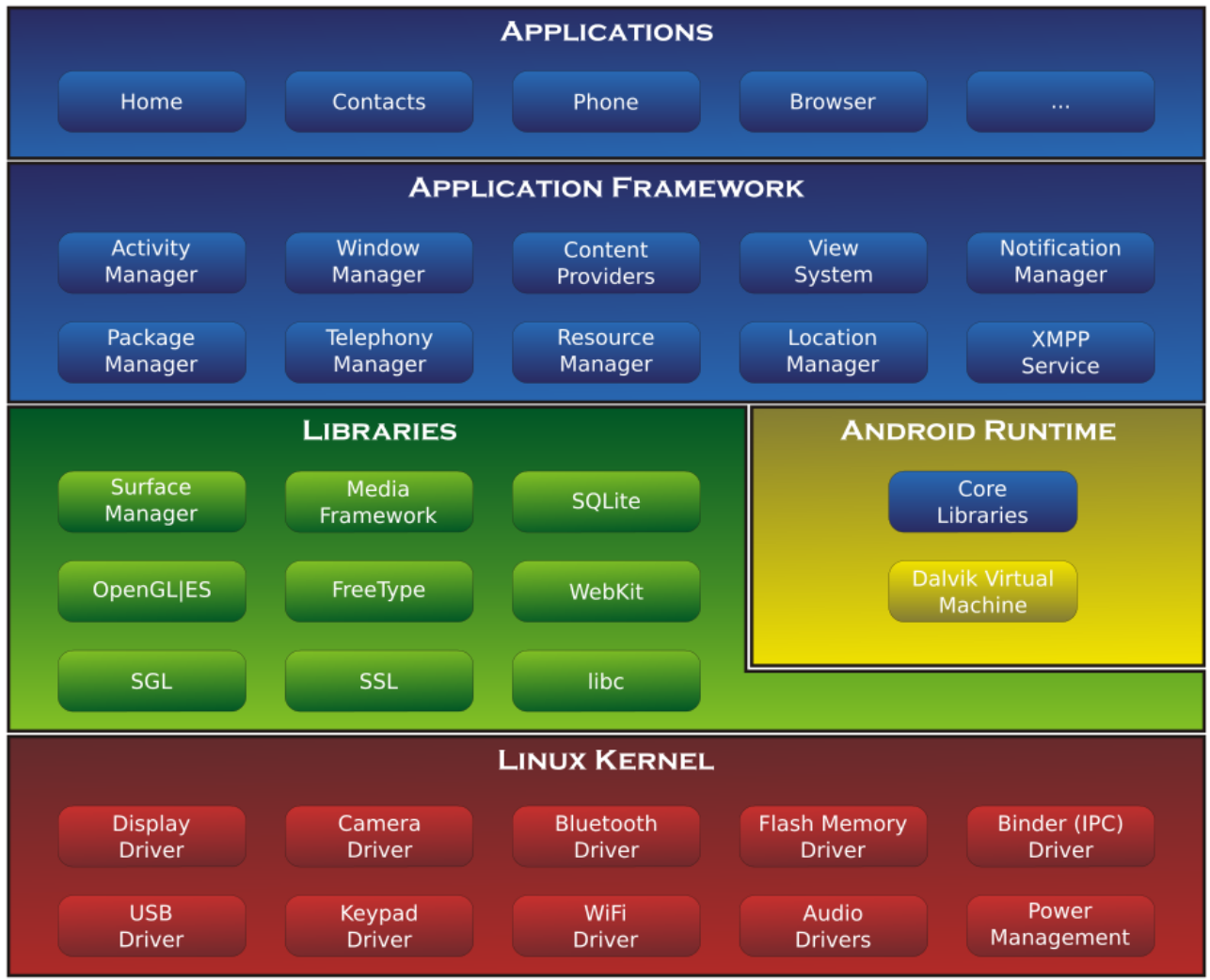
Nachdem wir uns auf der Anwendungs- und Administrationsebene mit dem Betriebssystem beschäftigt haben, beschäftigen wir uns nun mit den „Interna“ eines Betriebssystems, dem Systemkern mit den „Treibern“.

Der Kernel stellt über einen statischen sowie einen dynamischen Teil (Module) eine Schnittstelle zum Zugriff auf alle Hardware und Systemresourcen zur Verfügung.

Linux Kernel Architektur



Beispiel Android System Architektur



Exkurs Android: Zugriff auf OS-Ebene

1. Aktivieren der Entwickler-Optionen: „Einstellungen“ ☞ „Über das Tablet“ ☞ 7 Mal auf „Build-Version“ klicken ☞ „Entwickleroptionen“ sind ab sofort in den Einstellungen verfügbar. Dort aktivieren „ADB über USB“.

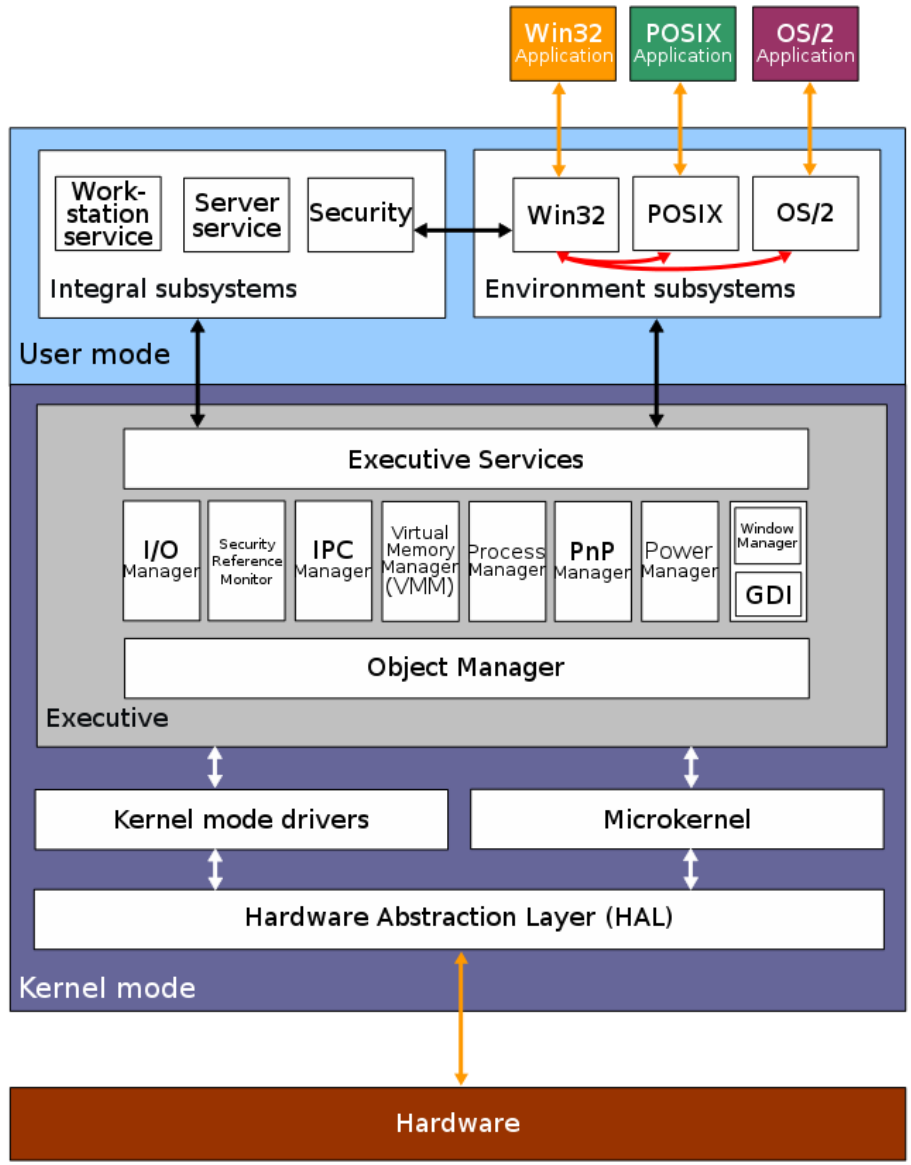
2. Einloggen per seriellem USB-Kabel: **`adb shell`** auf dem Host.

3. Ausprobieren: **`df`**, **`ps w`**, **`cat /proc/cupinfo`**, **`free`** ...

4. Root-Zugang (nur bei gerootetem Smartphone): **`adb root`** (Host) oder **`su`** (Android),

5. Wichtige Android-Verzeichnisse: **`/system`** (read-only Basis-System), **`/data`** (Programmdaten und eigene Programme), **`/cache`** (VM-Temporärdaten), **`/sdcard`** (interne oder externe SD-Karte), **`/extsd`** (Externe SD-Karte).

Beispiel Windows Architektur (NT Kernel, Windows2K)



Linux Kernel Verhalten

Einige Parameter lassen sich zur Start- und Laufzeit mit Hilfe von Kernel-Optionen einstellen, die per Bootloader übergeben werden, andere (wie die Auswahl von CPU-Architektur und Treibern) können hingegen nur zur Übersetzungszeit (Compile-Vorgang) eingestellt werden.

```
vmlinuz mem=2000M acpi=off nolapic_timer
```

Kernel bauen

1. Kernel-Archiv herunterladen von www.kernel.org,
2. Auspacken: **`tar jxvf linux-3.8.6.tar.bz`**,
3. Ins Kernel-Verzeichnis wechseln und menübasierte Konfiguration starten: **`cd linux-3.8.6`**, **`make menuconfig`** (Textkonsole), **`make xconfig`** (graphisches Menü)
4. Kernel und Module übersetzen: **`make`**

Intel/AMD vs. ARM Architektur

- Grundsätzlich lässt sich ein Linux-Kernel für fast sämtliche existierende Hardware (CPU/Board/Peripherie) bauen.
- Intel/AMD: PCI(e) unterstützt eine Hardwareerkennung, d.h es kann der richtige Treiber mit den richtigen Registern/Adressen bei Bedarf aktiviert werden.
- ARM-basierte Boards haben i.d.R. keinen PCI-Bus, sondern steuern direkt über GPIO-Schnittstellen die Hardware an. Eine Hardware-„Erkennung" ist hier nich möglich, erst hinter z.B. USB-Schnittstellen, die ein entsprechendes Protokoll besitzen

☞ Für ARM-basierte Systeme muss in fast jedem Fall ein auf genau dieses System optimierter und konfigurierter Kernel gebaut werden, sonst startet das Gerät erst gar nicht.
☞ S.a. Cyanogenmod für verschiedene Tablets/Smartphones.

Kernel installieren

Mit **`make modules_install`** werden die Module nach **`/lib/modules/`*`kernel-version`*** kopiert. Der statische Teil des Kernels aus dem Kernel Source-Verzeichnis unter **`arch/i386/boot/bzImage`** muss dem Bootloader in der jeweiligen Konfigurationsdatei (**`/etc/lilo.conf`** oder **`/boot/grub/menu.lst`**) bekannt gemacht werden, damit das Betriebssystem starten kann.

Bei den meisten Distributionen sind die statischen Kernel-Komponenten im Verzeichnis **`/boot`** zu finden.

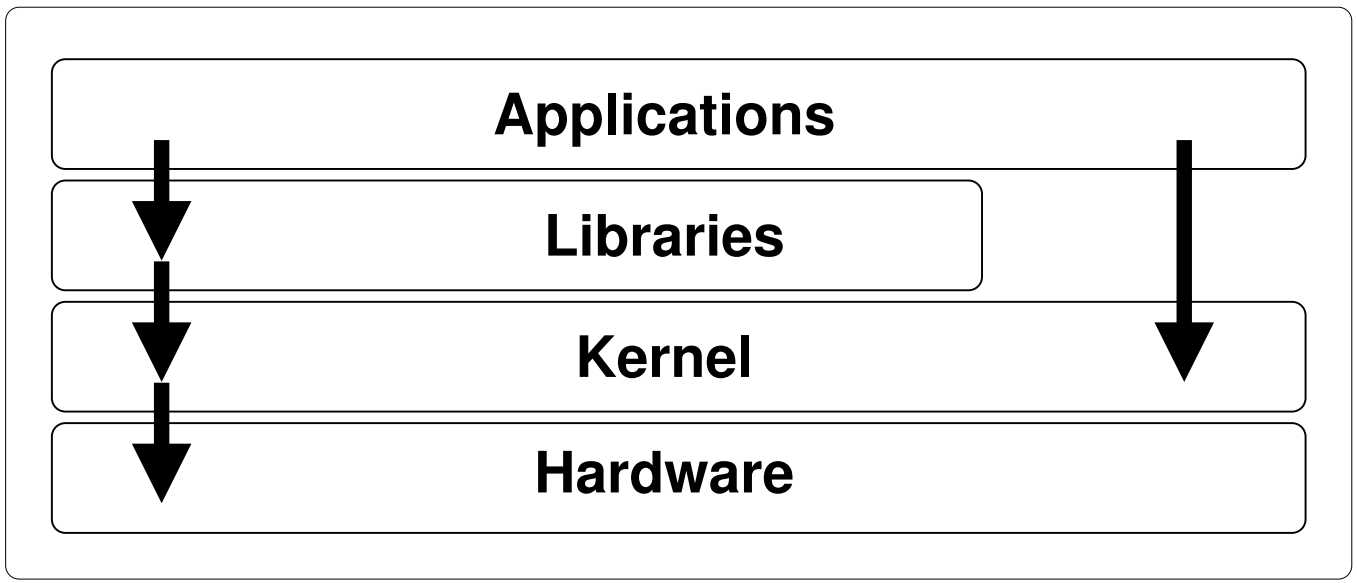
„Userspace-Treiber“

- Werden NICHT mit dem Kernel zusammen compiliert
- sind im Prinzip Anwenderprogramme, die
- die SChnittstellen der Kernel-Treiber nur NUTZEN.

Beispiel: Touchscreen-Treiber, Serial Devices (z.B. Myo).

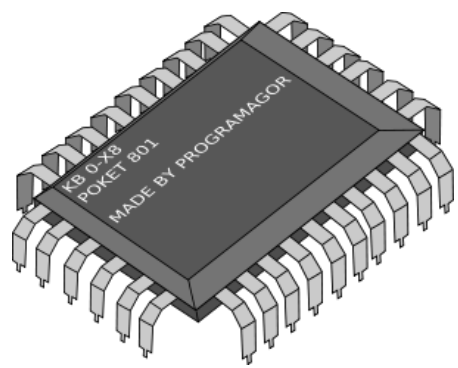
Layer-Darstellung

Der Kernel bildet die Schnittstelle zwischen Hardware und Anwendersoftware, wobei dynamisch gebundene Programme System- und Programmbibliotheken zum Zugriff auf die Kernel-API verwenden.



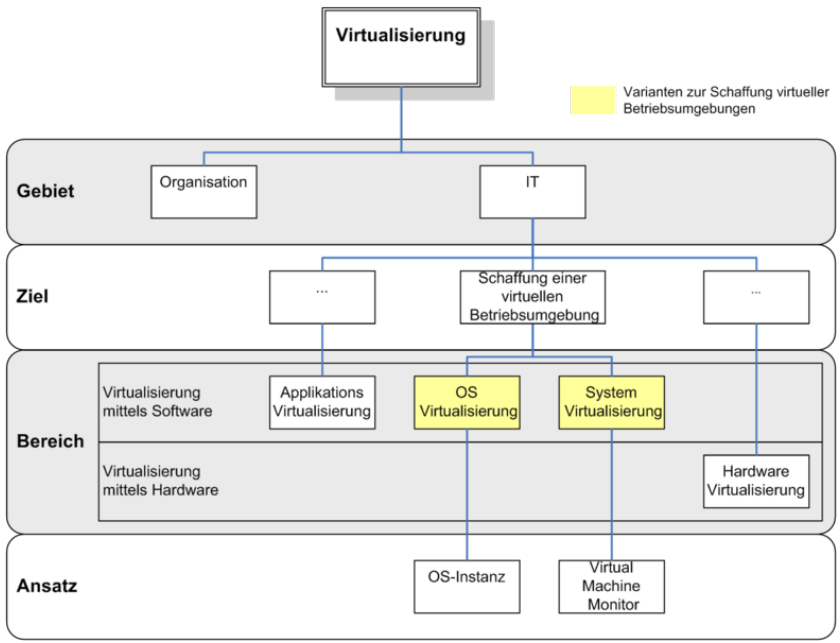
Peripherie-Virtualisierung

Durch das **Verhindern des direkten Zugriffs** von Anwenderprogrammen auf die Hardware und die **Abbildung** von Hardware in Form von „Gerätedateien“ im Verzeichnis **`/dev`** findet bereits eine „Virtualisierung“ statt - nämlich eine Modellierung realer Hardware auf einer API, die Programme verwenden können, um mit Hilfe von Dateizugriffen und **`Systemaufrufen`** (system calls, **io controls**) auf die vom Kernel geschützte Hardware zuzugreifen.



OS-Virtualisierung

Bei der Virtualisierung wird die Hardwareschicht zum größten Teil durch eine „Simulation“ ersetzt. Je nach Virtualisierungsgrad werden Anfragen an einzelne Resourcen wie CPU oder Datenträger des „Gast“-Betriebssystems vom Hypervisor (Virtual Machine Monitor VMM) direkt oder indirekt an das „Host“-Betriebssystem durchgereicht.



Virtualisierungs-Arten

Hardware-Emulation

Die komplette Hardware des „Gastes“ wird simuliert, selbst CPUs mit anderem Befehlssatz.
Beispiele: Bochs, MS Virtual PC (PPC Version), QEMU.

Hardware-Virtualisierung

Teile der Hardware werden virtualisiert (unter anderem Namen, verwaltet vom „Host“) an den „Gast“ weitergereicht, z.B. werden die meisten Instruktionen nativ auf der CPU ausgeführt und konfigurierbare Resourcen wie Festplattenpartitionen an das Gast-Betriebssystem „durchgereicht“.
Beispiele: VMware, Microsoft Virtual PC (x86), KVM, Xen, VirtualBox

Paravirtualisierung

Es wird eine neue Instanz eines Betriebssystems gestartet, jedoch ohne Simulation der Hardware. Entweder muss eine andere Instanz eine Hardwareschicht zur Verfügung stellen, oder das „Gast“-Betriebssystem muss mit einer virtualisierenden Verwaltungsschicht darauf vorbereitet werden, als separate Instanz auf der gleichen Hardware zu laufen.
Beispiele: VMware ESX 3.5, KVM, Xen 3.0, PikeOS, Iguest

Virtualisierung OS-Images

Festplatteninstallationen und Speicherabbilder werden häufig in Images abgelegt, die optional komprimiert sein können, um realen Speicherplatz einzusparen. Auch sog. „sparse files" - Dateien mit „Löchern" (noch nicht belegtem Platz, der erst beim Zugriff expandiert) sind möglich, wenn vom Host-Dateisystem unterstützt.

QEMU besitzt ein umfangreiches Konvertierungs-Tool, das virtuelle Images zwischen verschiedenen Virtualisierungs-Engines umwandeln kann.

Übung

1. Stellen Sie fest, ob Ihr Computer „Hardware-Virtualisierung“ unterstützt:
   `grep -E '^flags.*(vmx|svm)' /proc/cpuinfo`
2. Falls ja, laden Sie das kvm-Modul für Intel oder AMD:
   `sudo modprobe kvm_intel` bzw.
   `sudo modprobe kvm_amd`
3. Starten Sie KNOPPIX in der Virtualisierung neu:
   `kvm -m 1024 -boot d -cdrom /dev/sr0`
   bzw.
   `qemu-system-x86_64 -m 1024 -boot d -cdrom /dev/sr0`
   bzw. für USB-Stick:
   `qemu-system-x86_64 -m 1024 -boot c -hda /dev/sdb`
   (ACHTUNG: hier unbedingt bei den Knoppix-Bootoptionen „`knoppix64` **`noimage`**“ angeben!!!)

Ausblick: Software Engineering

Hinweis: Nicht prüfungsrelevant!

Kernprozesse:

1. Planung
2. Analyse
3. Entwurf
4. Programmierung
5. Tests

Unterstützungsprozesse:

6. Projektmanagement
7. Qualitätsmanagement
8. Konfigurationsmanagement
9. Dokumentation

Aufbau von OSS-Projekten

- (Fast) alle OSS-Projekte haben einen koordinierenden Projektleiter,
- Kommunikationsmotor ist das Internet bzw. Client/Server-Entwicklungplattformen,
- Zusammenarbeit ist i.d.R. hierarchisch organisiert.

Organisatorisch ist die Entwicklung bei OSS kaum anders als bei proprietärer Software, bis auf die unkompliziertere Verwaltung der Rechte am „geistigen Eigentum“ und der meist höhere Grad an Dezentralität.

Plattformen/Tools (1)

- Quelltexte-Versions/Revisionsverwaltungen
  - rcs (lokal)
  - cvs, svn (Zentrales Remote-Archiv)
  - git (Kernel, P2P)

Zusammenarbeit mehrerer Entwickler am gleichen Quelltext mit Unterstützung von Fork, Branch und Merge.

Plattformen/Tools (2)

- Paketierung von Binaries und Quelltexten
  - Klassisch, mit selbst übersetzten Quelltexten **`*.tar.gz`** mit **`configure`**, **`make`**, **`make install`**.
  - Selbstextrahierende/-installierende **`Shell-Archive`**.
  - Paketbasiert **`rpm`** (RedHat/Fedora/SuSE) zum Software-Installationsmanagement.
  - Paketbasiert **`deb`** (Debian und Derivate wie Ubuntu) zum Software-Installationsmanagement.

Plattformen/Tools (3)

- **IDE** (☞ Integrated Development Environment) und **CASE** (☞ Computer Aided Software Engineering):
  - **`kdevelop`**, **`monodevelop`**,
  - **`eclipse`**,
  - **`argouml`** (UML-Diagramme $\leftrightarrow$ Quelltexte),
  - **`xwpe`**, **`ddd`**, ...

Plattformen/Tools (4)

- Klassiker:
  - **make**,
  - **autoconf**,
  - **configure**.
- Exoten:
  - **jam**,
  - **smake**,

Plattformen/Tools (5)

- GNU Compiler Collection (GCC) mit
  - C,
  - C++,
  - Assembler,
  - Pascal,
  - Fortran,
  - Java (OSS),
  - ...

Plattformen/Tools (6)

- Verschiedene
  - JAVA (Sun),
  - C#/Mono,
  - div. Lisp-Dialekte,
  - **`sed`**, **`awk`**,
  - PHP, Perl, Python,
  - ☞ „Esoterische"-Programmiersprachen wie *white-space*, *brainfork*.

Plattformen/Tools (7)

Öffentliche Repositories als „Anlaufstelle“ für die Projektdaten, Beispiele:

- **`http://www.`*`projektname`*`.org/`**
- **`http://`*`projektname`*`.sourceforge.net/`**
- **`http://`*`projektname`*`.alioth.debian.org/`**
- **`http://github.com/`*`projektname`*`/`**

Exkurs: OS-Security, Recovery, Forensik

Hinweis: Nicht prüfungsrelevant!

- Um an sensitive Daten auf dem gleichen oder einem anderen Rechner zu gelangen, nutzen Angreifer Systemschwächen von Betriebssystem und Diensten aus.
- Fehlbedienung oder „Denial of Service“ Attacken können zu Datenverlusten führen.
- In der Computerforensik wird das System zunächst read-only analysiert, um entweder später Daten oder das System wiederherzustellen oder Rückschlüsse auf Angriffspunkte (Einfallstore) oder Probleme ziehen zu können.

Forensik-Werkzeuge: ☞ Intrusion Detection Systems, testdisk, photorec, foremost, hexedit.

Mit sog. „Virenscannern“ soll bekannte (i.d.R. Windows-) Schadsoftware vor der Ausführung erkannt werden. Unter Unix sind eher IDS bzw. Prüfsummenchecks sowie signierte Softwarepakete üblich, um Installation und Ausführung von Schadsoftware zu verhindern, wobei das Unix-Rechtesystem bereits die Modifikation von Systemsoftware durch unprivilegierte Anwender verhindert.

Ende der Vorlesung

Eingrenzung der klausurrelevanten Teile:

- Alle im Unterricht verwendeten Folien, wenn sie nicht explizit als *nicht prüfungsrelevant* deklariert wurden.
- Besprochene Handouts.
- Gemeinsam durchgeführte Übungen und Beobachtungen am Rechner.

Grundsätzlich sollen die verwendeten Begriffe beherrscht und mit eigenen Worten bzw. mit den dargestellten Stichworten erklärt werden können. Ein „wörtliches Auswendiglernen“ von Folientexten ist hingegen kaum sinnvoll.