Systemprogrammierung (WS 16/17)

Prof. Dr. Christian Forler

Beuth Hochschule für Technik Berlin

29. März 2017

Organisatorisches

- Stellen Sie Fragen!
  - Unterricht,
  - E-Mail `mailto:cforler@beuth-hochschule.de`,
  - Büro: D 134,
  - ...
- Vorlesung: Fr. 10:00 Uhr (B 254)
- Übungen: Fr. 12:15 Uhr (D 132 L)
- Klausurtermine: TBA
- Bei der Klausur sind keine Hilfsmittel erlaubt
- Dozenten sind fehlbar

  $\Longrightarrow$ Bei Unklarheiten stellen Sie Fragen

Übung

- Erlaubt sind Gruppenarbeiten bis zu 3 Personen.
- Um die Übung zu bestehen müssen mind. 25% der Gesamtpunktzahl erreicht werden.
- Abnahme erfolgt in der Übung
  (Es gibt nur Punkte wenn Sie ihren Code auch erklären können.)
- Pflicht: Angabe externer Quellen.
- Natürlich ist Abschreiben nicht erwünscht.

Sonstiges

- Drucken Sie die Folien vor der Veranstaltung aus
- Machen Sie sich Notizen
- Ohne Mitarbeit und Bearbeitung der Übungen werden Sie nicht viel lernen
- Bitte keine Handys (Flugmodus) in der Vorlesung benutzen
- Stellen Sie Fragen
- Bereiten Sie die Vorlesung und Übungen nach
- Es gibt Bonuskarten
  - Bei Teilnahme an einer SU gibt es einen Stempel
  - Wer den Unterricht stört bekommt keinen Stempel
  - Sei n die Anzahl der SUs im Semester, dann gibt es ab $n - 1$ Stempel einen 1/3 Notenbonus auf die Klausur

Vorkenntnisse I

Bitte um Handzeichen – Wer von Ihnen hat schon . . .

- . . . mit Linux oder BSD gearbeitet?
- . . . mit einer anderen Shell als der `cmd` gearbeitet?
- . . . ein `python`-Script geschrieben?
- . . . ein `bash`-Script geschrieben?
- . . . beim Programmieren `fork()` benutzt?
- . . . `make` verwendet?
- . . . `git` verwendet?
- . . . einen Treiber geschrieben?
- . . . einen Github- oder Stackoverflow-Account?

Vorkenntnisse II

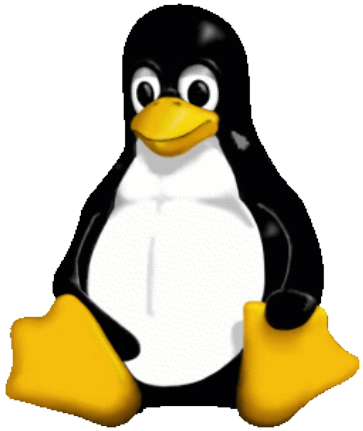
Was machen die folgenden Funktionen?

```
char *foo(char *dest, const char *src) {
   while(*dest++ = *src++);
   return dest;
}
```

```
int bar(const char *x, const char *y) {
  while( ((*x) &&  (*x == *y)) ) {
     x+=1;
     y+=1;
  }
  return (*x) - (*y);
}
```

Ziel der Vorlesung

- Nutzung elementarer Werkzeuge wie `make` und `gcc`
- Administration und Automatisierung mit Shell-Skripten
- Interaktion zwischen Programmen und dem Betriebssystem (OS)
- Tieferes Verständnis über die Funktionsweise eines OSs
- Der Fokus der Vorlesung liegt auf Unix-artigen OSen



Acknowledgments

Disclaimer

Die Folien dieser Veranstaltung sind teilweise inspiriert von Prof. Dr. Rüdiger Weis und Prof. Dr. Robert Baumgartl.

Bücher zur Vorlesung

- Moderne Betriebssysteme von Andrew S. Tanenbaum
- Linux: Das umfassende Handbuch von Michael Kofler
- Linux-Unix-Systemprogrammierung von Helmut Herold
- Shell-Programmierung von Jürgen Wolf
  `http://openbook.rheinwerk-verlag.de/shell_programmierung/`
- Linux-UNIX-Programmierung von Jürgen Wolf
  `http://openbook.rheinwerk-verlag.de/linux_unix_programmierung/`

Betriebssysteme

Es gibt:

- Windows-Familie (2.0 $\rightarrow$ 10)
- Den GNU/Linux Zoo (Debian, Suse, Ubuntu, . . .)
- BSD-Familie (FreeBSD, OpenBSD, NetBSD)
- Mac OS X

Das war's, oder?

MS-DOS, RTEMS, QNX, PikeOS, SymbianOS, PalmOS, RTAI, HP-UX, BeOS, Minix, Chorus, L4, Mach, Amoeba, OS/390, DCP, TOS, CP/M, VMS, eCos, Contiki, OS/2, Plan9, . . .

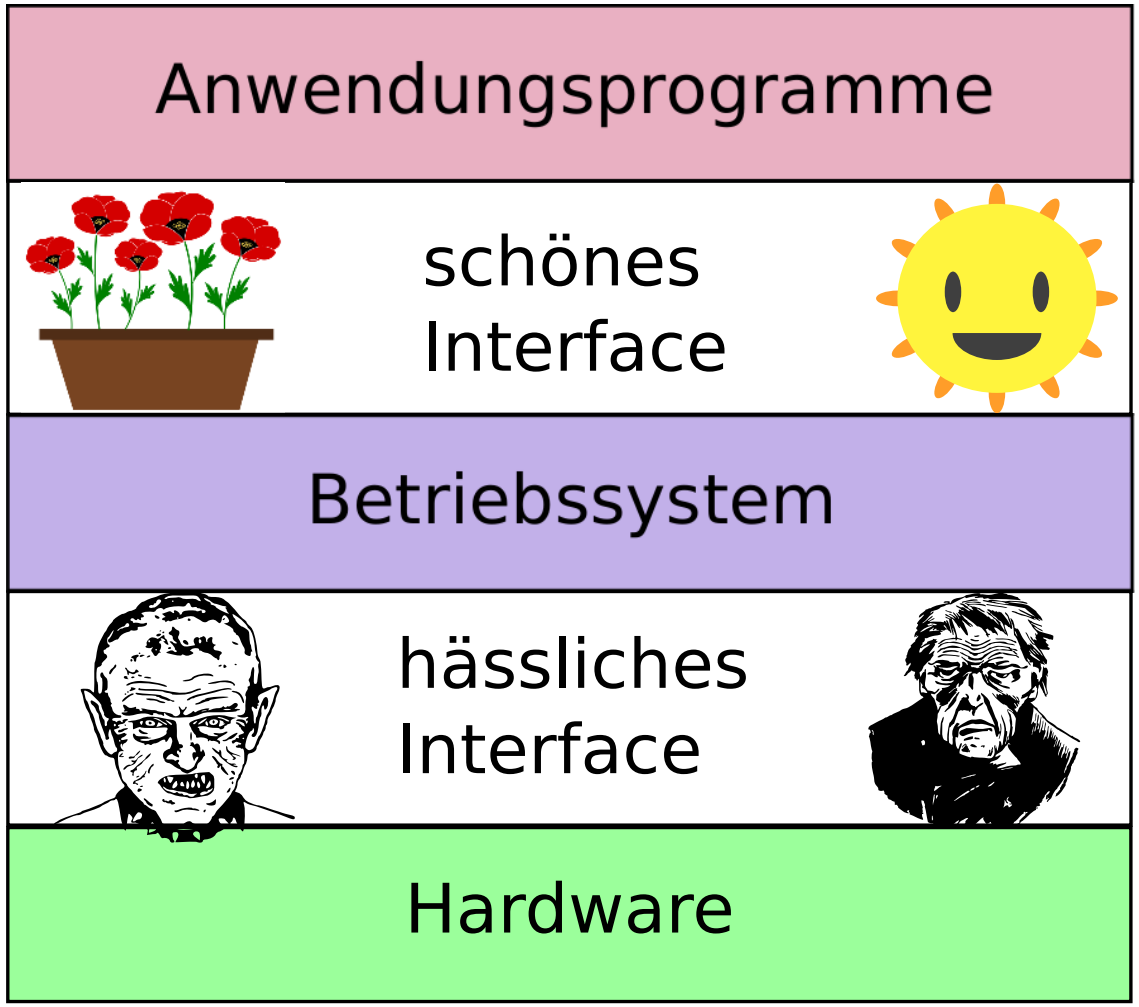
vgl. `http://de.wikipedia.org/wiki/Liste_der_Betriebssysteme`

Aufgaben eines Betriebssystems

- **Frage:** Was sind die Aufgaben eines Betriebssystems?
- Die Hauptaufgabe eines Betriebssystems ist die
  1. Abstraktion und die
  2. Verwaltung

  der Systemressourcen/Betriebsmittel/Hardware wie beispielsweise Festplatten, Arbeitsspeicher und Drucker

Abstraktion von Betriebsmitteln



Betriebssysteme verwandeln hässliche Hardwareinterfaces in schöne Softwareinterfaces

Verwaltung von Systemressourcen

Quelle: `https://s3.amazonaws.com/aphs.worldnomads.com/safetyhub/12392/chinese_traffic.jpg`

Betriebssysteme verwalten Systemressourcen um Multiplexing zu ermöglichen.

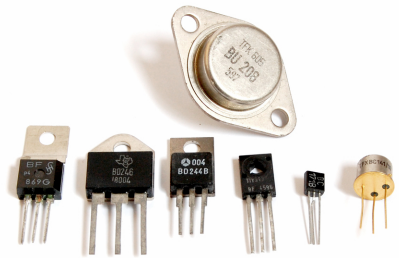
Geschichte der Betriebssysteme I

Andrew Tanenbaum unterscheidet 4 Epochen

1. 1945–1955 Elektronenröhren (Keine Betriebssysteme)

Quelle: `https://de.wikipedia.org/wiki/Elektronenröhre`

2. 1955–1965 – Transistoren

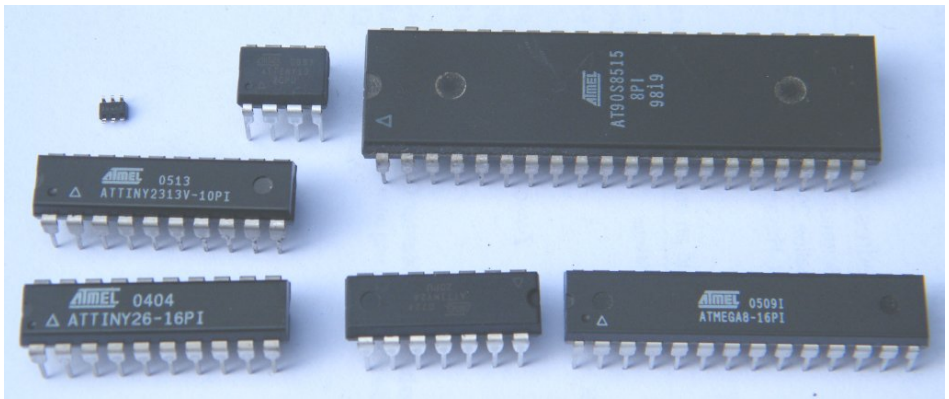


Quelle: `https://de.wikipedia.org/wiki/Transistor`

- Mainframes – kommerzielle Computer
- Batchsysteme (Ziel: maximale Auslastung)

Geschichte der Betriebssysteme II

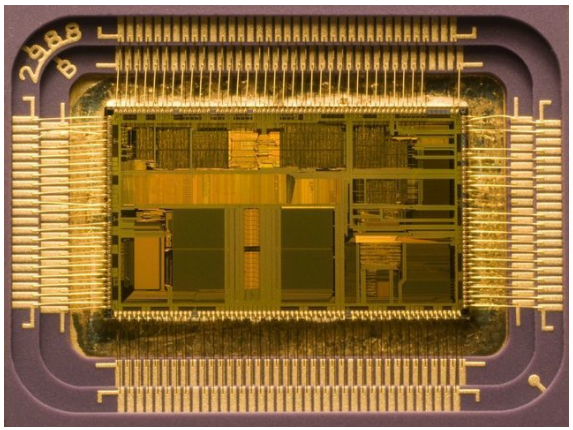
3. 1965–1976 – integrierte Schaltkreise
   - IBM System/360 $\rightarrow$ **OS360**
   - Multiprogamming (Kontextwechsel bei Zugriff I/O-Geräte)
   - Spooling (Puffer für zu bearbeitende Aufträge)
   - Timesharing (Multiuserkonzept)
   - MULTICS (Process abstraction, Virtual Memory, Dynamic Linking, Multiprozessor Support, Hierachical File System, Multiuser Support, Mulit-Level Security . . .)
   - Unix (portabel, offen, kollaborativ entwickelt)



Quelle: `http://www.avr-asm-tutorial.net/avr_de/absolute_beginner/absolut_beginner.html`

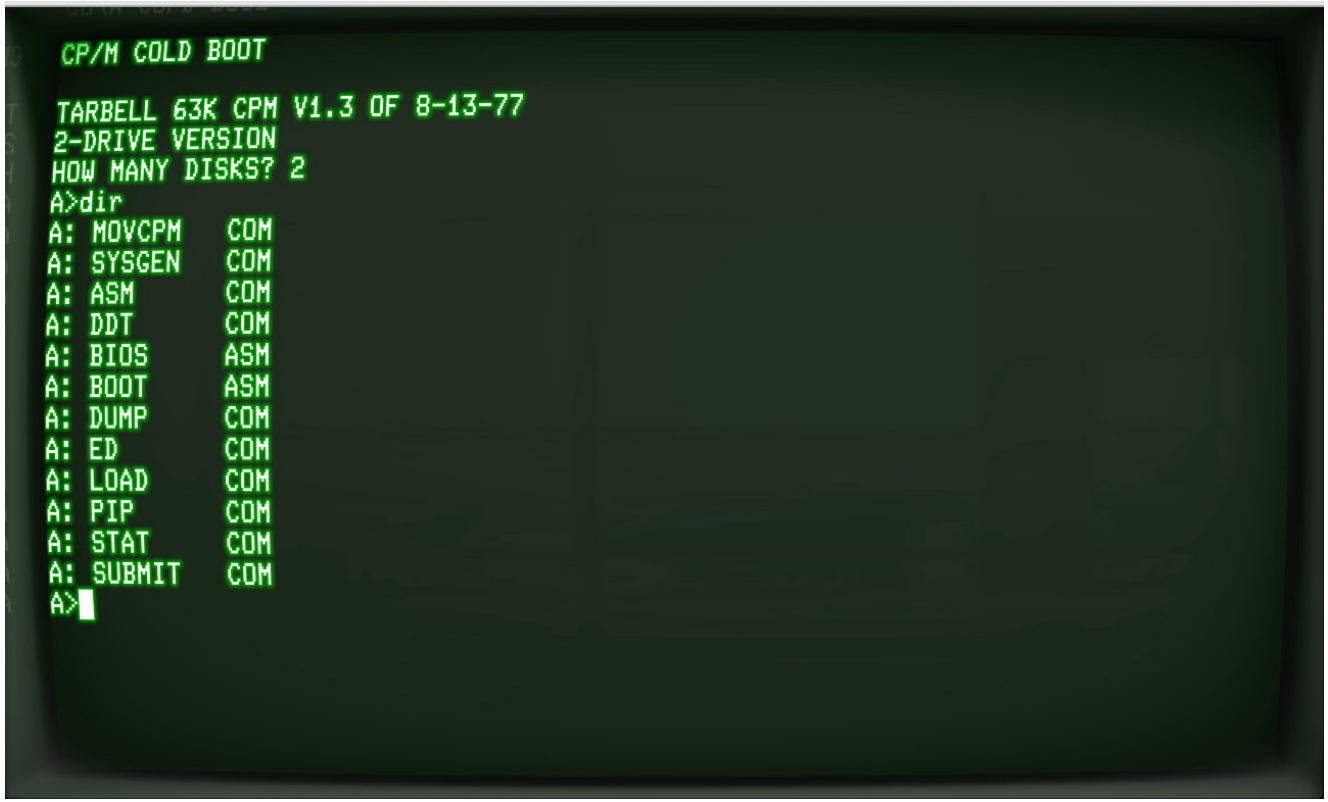
Geschichte der Betriebssysteme II

4. 1976–heute – der Micro- und Personalcomputer (PC)
   - Start der Homecomputer Ära (Apple II, C64, Amiga, Atari ST, . . .)
   - 1976: Prozessor Zilog Z80 → **CP/M**
   - 1977: PDP-11 → **BSD**
   - 1980: **QDOS** → **MS-DOS**
   - 1984: Apple Macintosh → **MacOS** (GUI)
   - 1985: Atari ST → **TOS**
   - 1992: GNU/Linux
   - 1995: Windows 95



Quelle: `https://de.wikipedia.org/wiki/Integrierter_Schaltkreis`

CP/M 1.3



Quelle: `http://www.computerhistory.org/atchm/early-digital-research-cpm-source-code/`

OS/360 auf dem System/360 Model 91

Quelle: `https://en.wikipedia.org/wiki/IBM_System/360`

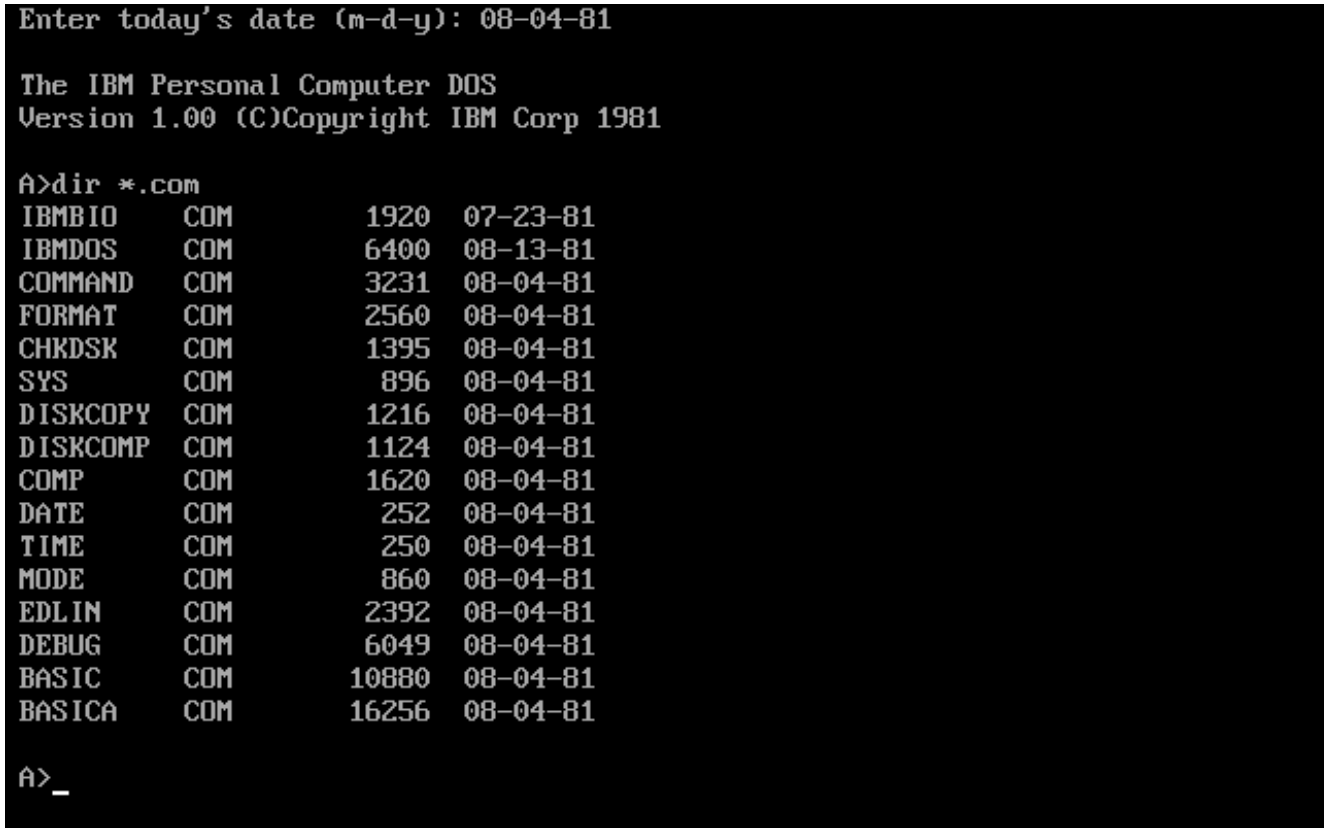
MULTICS auf dem IBM 704

Quelle: `http://rust-class.org/tag/multics.html`

Unix auf dem PDP-11 Minicomputer

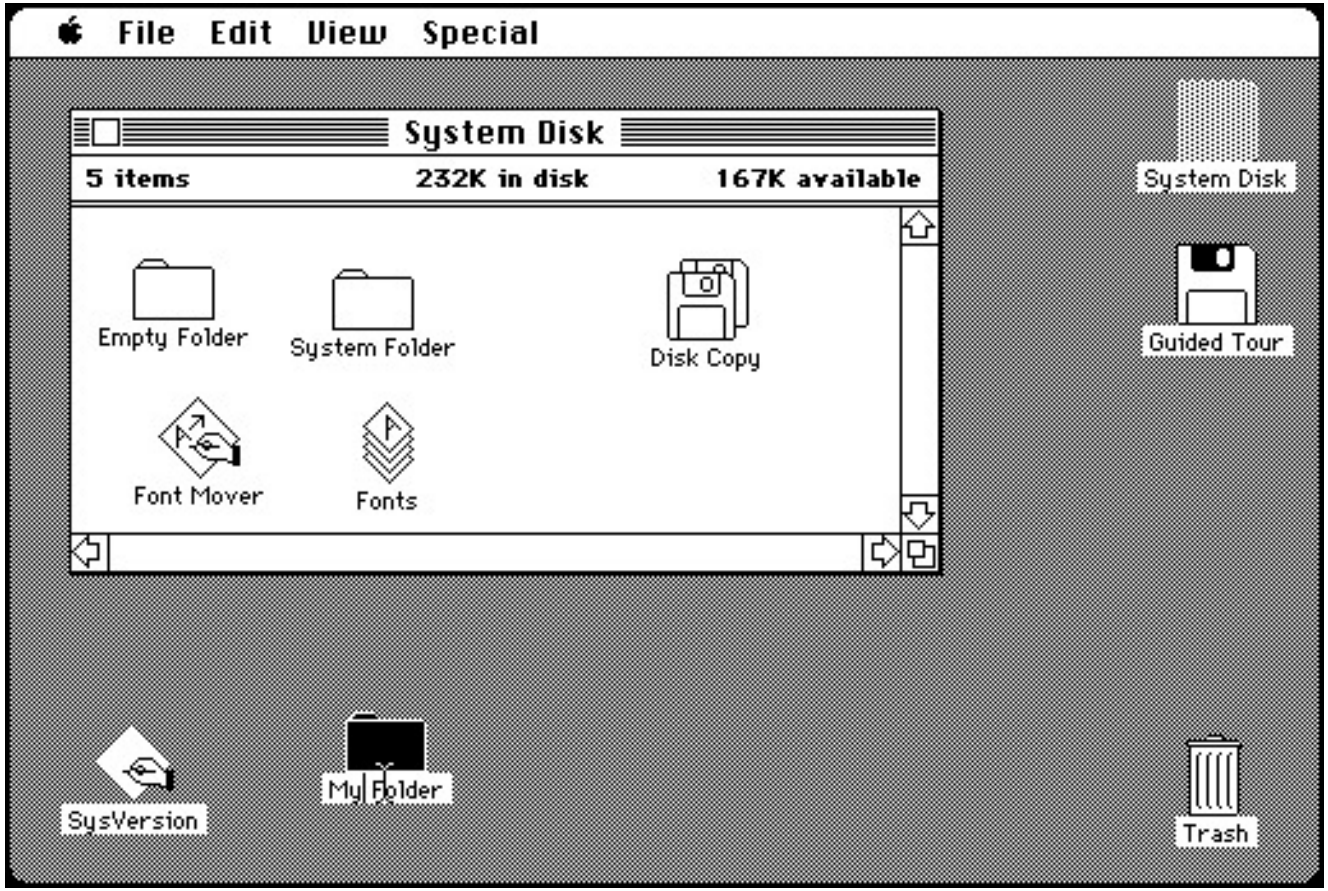
Quelle: `http://www.theregister.co.uk/2015/04/14/the_appeal_of_appliances/`

MS-DOS



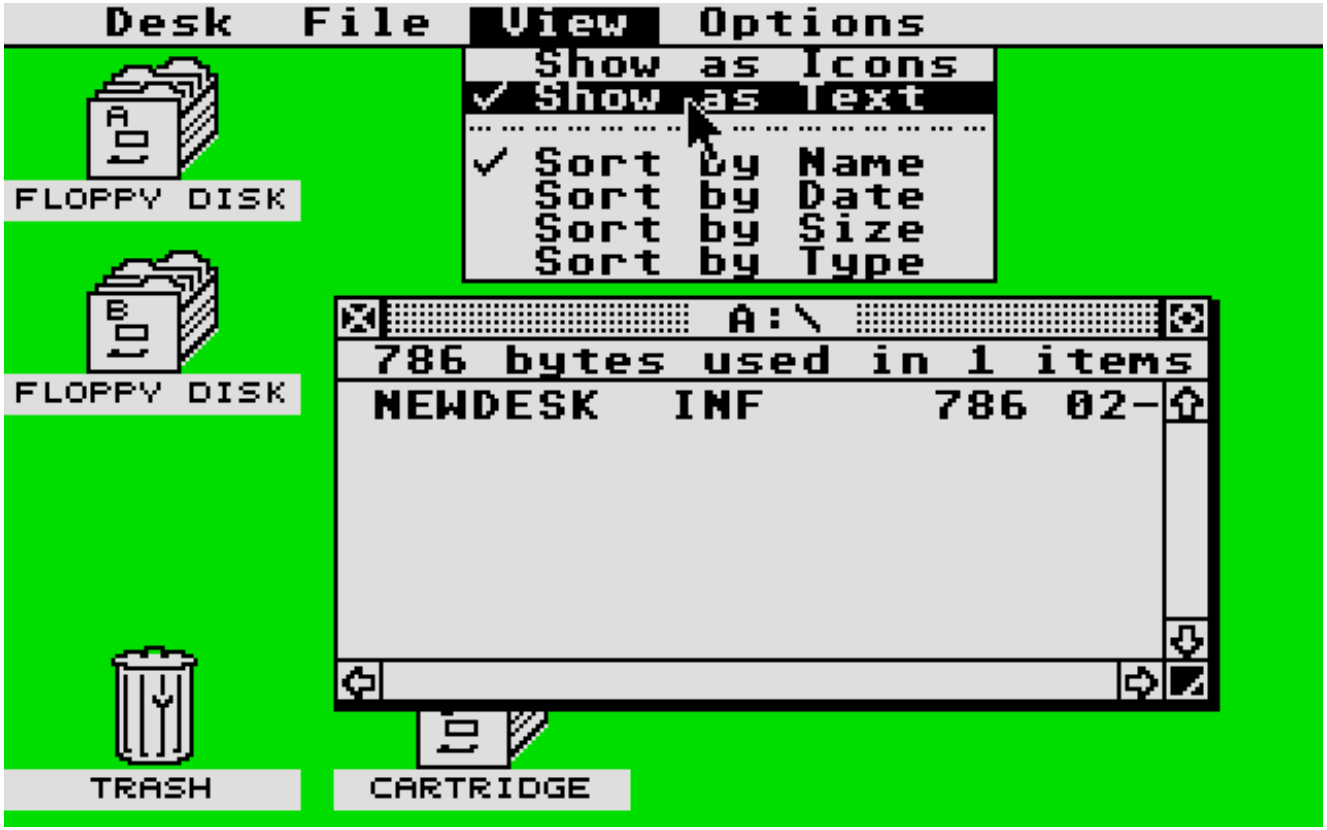
Quelle: `http://www.winhistory.de/more/msdos.htm`

MacOS 1



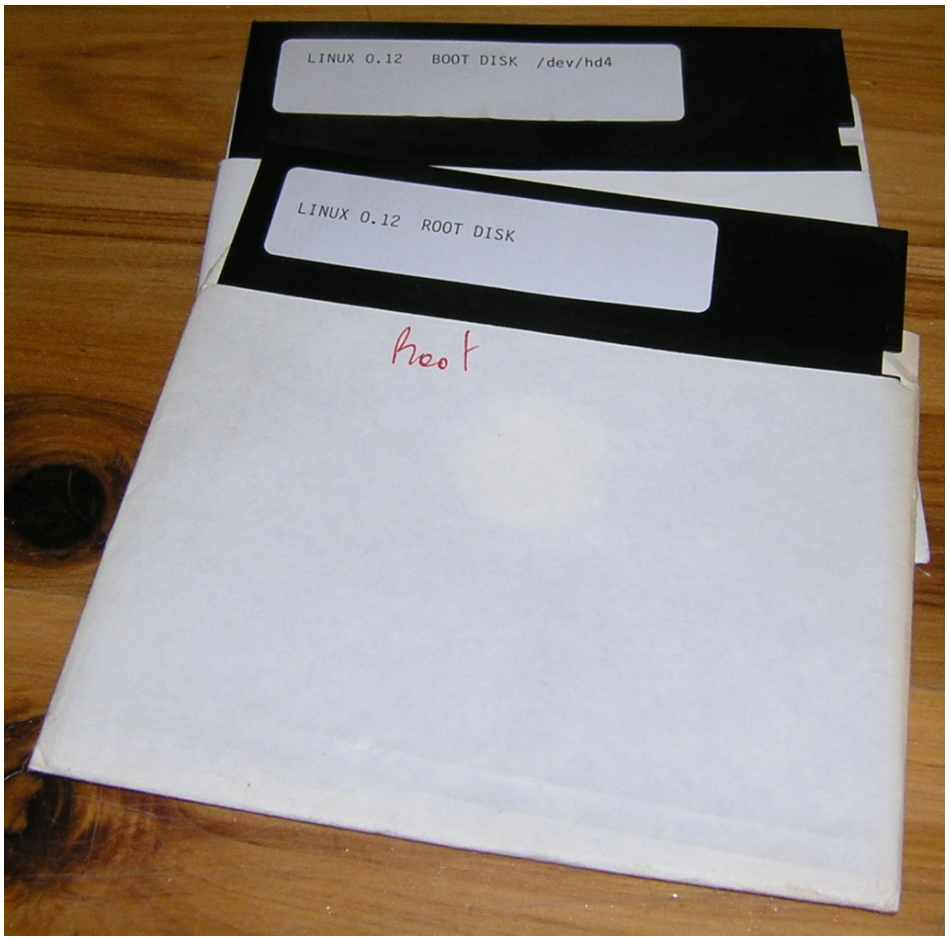
Quelle: `http://lowendmac.com/wale/07/the-roots-of-the-mac-os.html`

Atari TOS 1



Quelle: `http://toastytech.com/guis/tos.html`

GNU/Linux 0.12

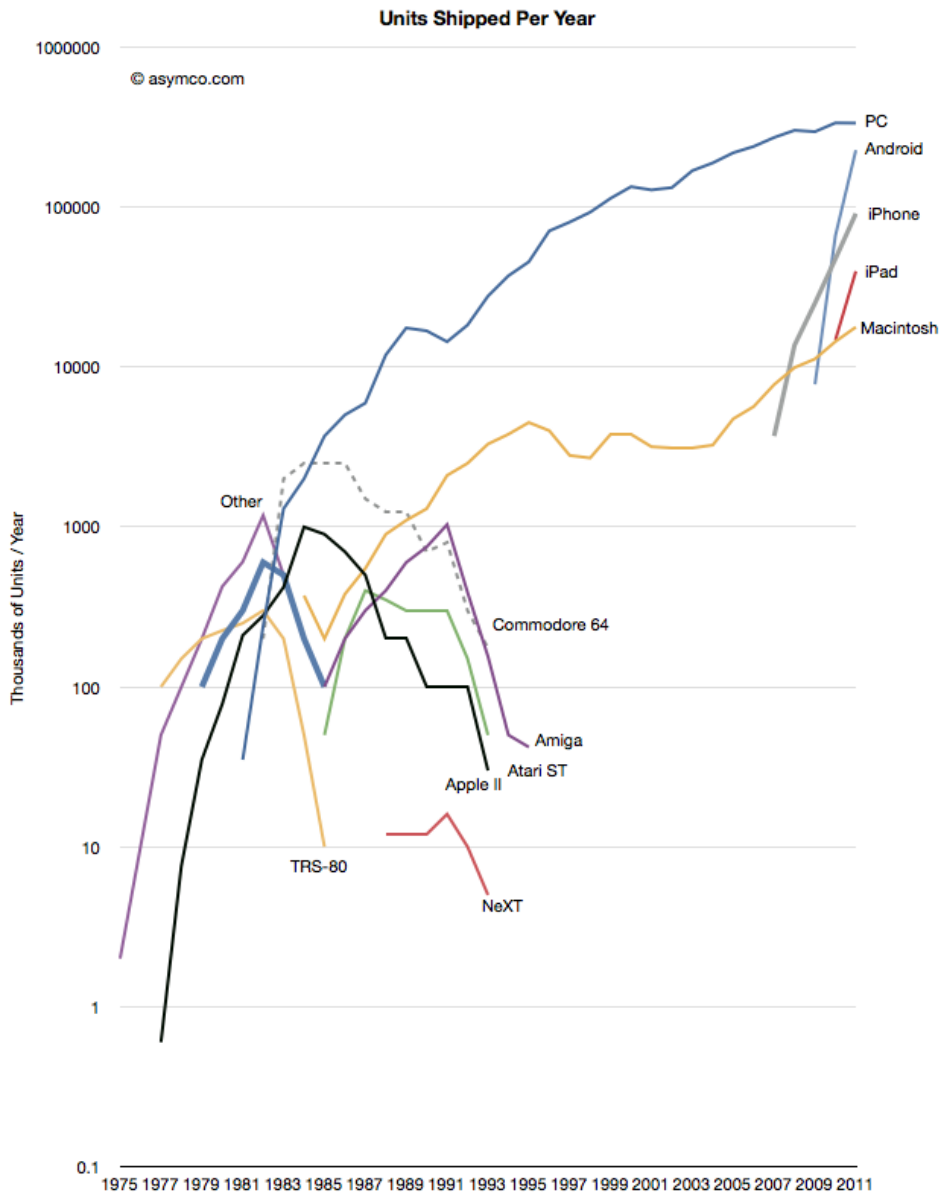


Quelle: `https://en.wikipedia.org/wiki/History_of_Linux`

Windows 95

Quelle: `https://en.wikipedia.org/wiki/Windows_95`

Plattformen fürs “Personal Computing”



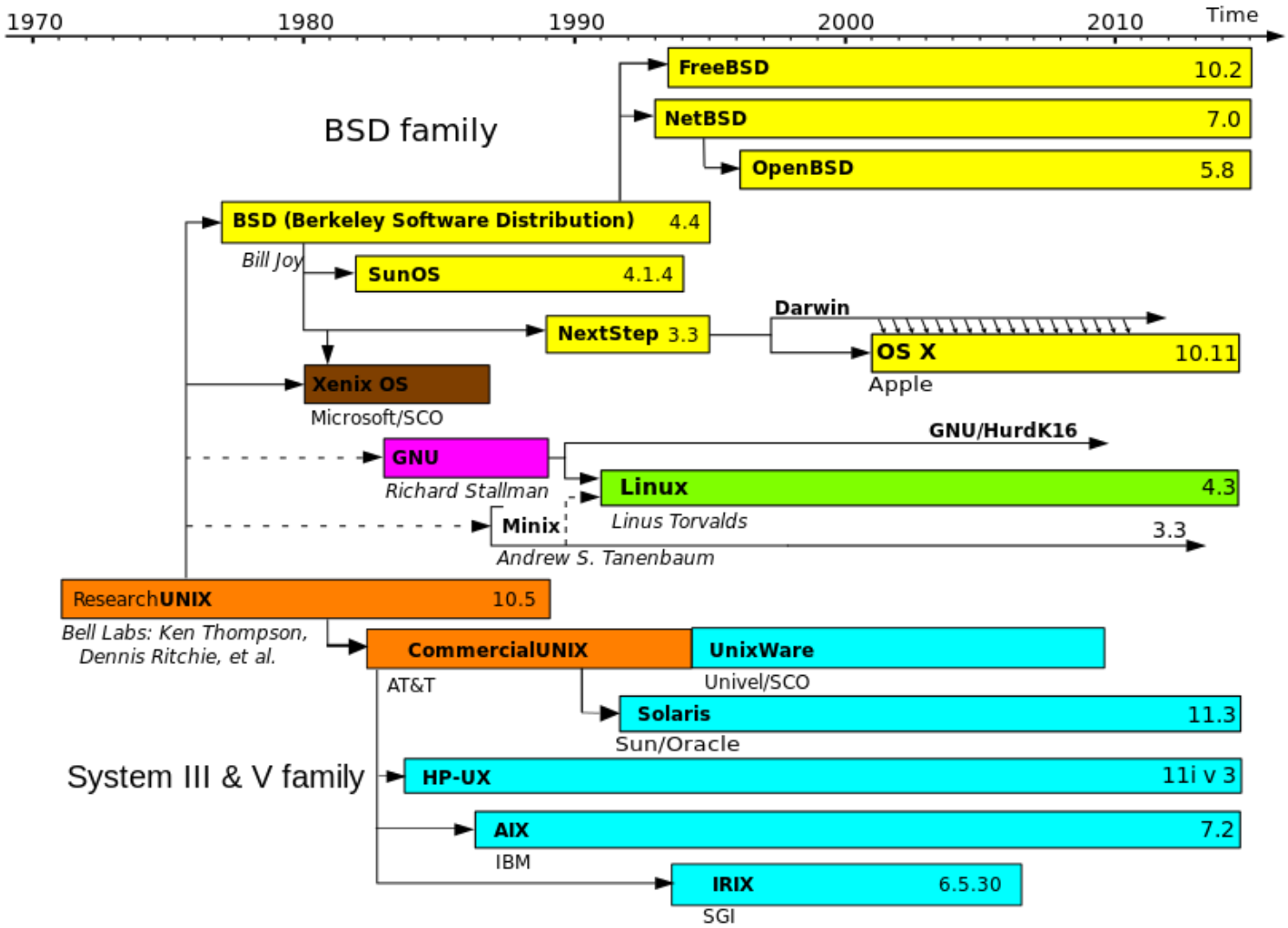
Welches Betriebssystem wird eingebettet eingesetzt?

Quelle: `http://www.embedded.com/collections/4402460/Embedded-Market-Surveys`

Unix

- Unix ist Familie von Betriebssystemen
- Unixoide Systeme: AIX, *BSD, Darwin (Mac OS X), HP-UX, GNU/Linux, Minix, Solaris, ...
- Microsoft hatte auch schon ein Unix entwickelt: XENIX (es ist aber schon lange tot)
- Name ist ein Wortspiel aus dem Vorgänger Multics und unique
- Unix wurde zusammen mit der Programmiersprache C in den 70er Jahren entwickelt
- Portable Operating System Interface (POSIX) ist ein Unix API-Standard für Anwendungsprogramme
- Beliebt vor allem im Serverbereich (aber nicht nur!)

Entwicklung von Unix und unixoiden Systemen



Quelle: `https://de.wikipedia.org/wiki/Unixoides_System`

Die Unix-Philosophie von Mike Gancarz

1994 veröffentlichte Gancarz die Folgenden 9 Grundsätze

1. Klein ist schön
2. Jedes Programm soll genau eine Sache gut machen
3. Erzeuge so bald wie möglich einen funktionierenden Prototyp
4. Portabilität geht über Effizienz
5. Speichere (numerische) Daten in einfachen Textdateien
6. Nutze die Hebelwirkung der Software zu deinem Vorteil
7. Verwende Shell-Skripte, um die Hebelwirkung und die Portierbarkeit zu verbessern
8. Vermeide Benutzeroberflächen, die den Benutzer fesseln
9. Gestalte jedes Programm als Filter (`stdin`, `stdout`, `stderr`)

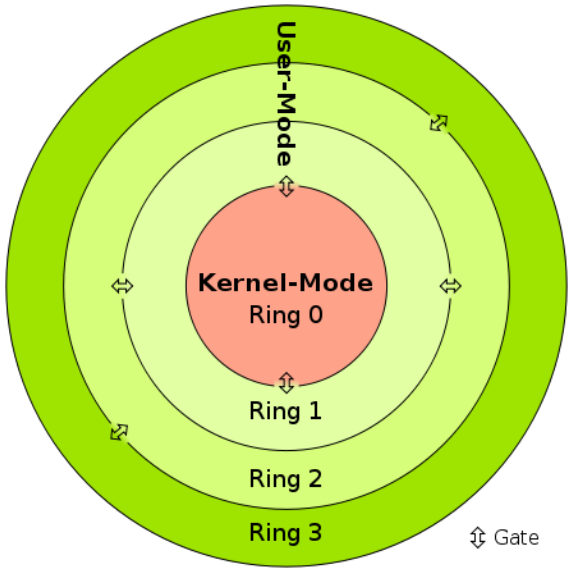
Interaktion mit dem Betriebssystem

Paradigmen:

- Textorientiert (Konsole, Shell, Eingabeaufforderung)
- Graphical User Interface (Windows, KDE, iOS)

- Was ist besser? Kommt darauf an.
  - Zielgruppe
  - Environment
  - Automatisierbarkeit
  - Persönliche Präferenzen

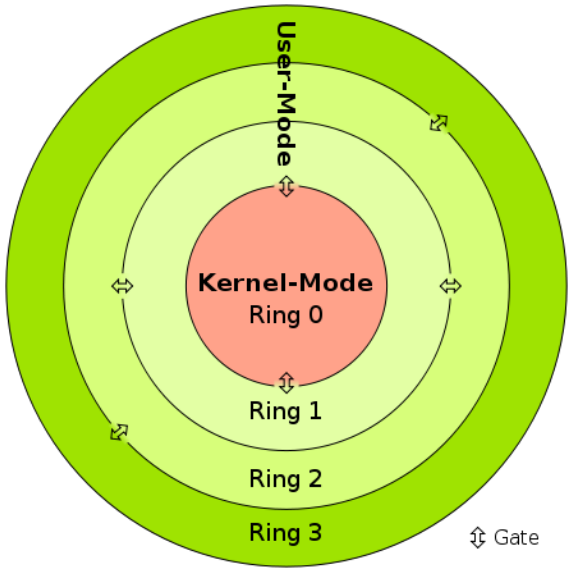
CPU-Ringe I



Quelle: `https://de.wikipedia.org/wiki/Ring_(CPU)`

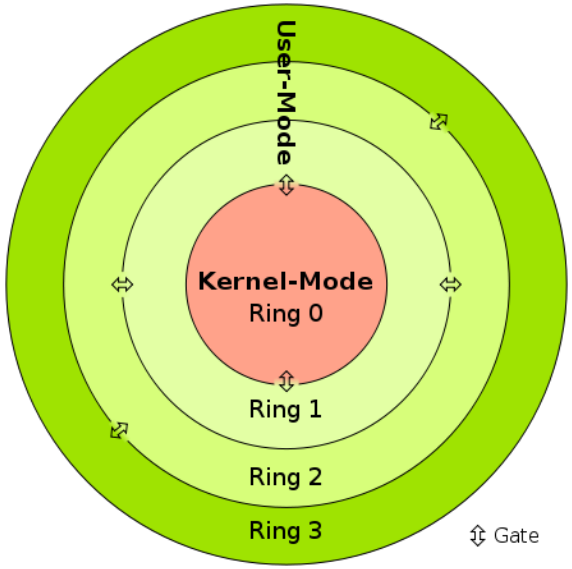
- x86-CPUs haben 4 Privilegierungsstufen: Ring 0 - Ring 3
- Höchste Privilegierungsstufen: Ring 0 (*supervisor mode*)
- Niedrigste Privilegierungsstufen: Ring 3
- Assembler-Instruktionen benötigen Privilegien
  - in Ring 0 können alle Instruktionen ausgeführt werden
  - in Ring 3 können die wenigsten Instruktionen ausgeführt werden

CPU-Ring (x86-Systeme) II



- Ring 0 und 1 sind für den Kernel vorgesehen (kernel mode)
- Ring 2 und 3 sind für den Userspace vorgesehen (user mode)
- Häufig werden nur Ring 0 und 3 benutzt (Windows, Linux, . . .)
- Speicher wird in kernel space und `user space` aufgeteilt

CPU-Ring (x86-Systeme) III



- Direkter Hardwarezugriff (I/O) geht nur in Ring 0
- Speichervirtualisierung verhindert direkten Speicherzugriff
- Prozess in Ring 3 kann keine anderen Prozesse beeinflussen
- Durch Gates können unprivilegierte Prozesse Kernelfunktionen aufrufen (Beispiel: `read(fd, buf, buflen))`

Hypervisor Mode

- Moderne x86 Prozessoren haben einen *Hypervisor Mode*
- Der Hypervisor Mode wird auch Ring -1 gennant.
- AMD Codename ist "Pacifica" bzw. AMD Virtualization
- CPU-Flag: `svm`
- Intel Codename "Vanderpool" bzw. VT-x
- CPU-Flag: `vmx`
- Haupteinsatzgebiete: Servervirtualisierung

System Management Mode (SMM)

- Moderne x86 Prozessoren haben einen System Management Mode (Ring-2)
- Code kann den das Betriebsystem / Hypervisor transparent unterbrechen
- Anwendungen
  - Behandlung von Hardwarefehlern
  - Notabschaltung
  - Power Management
  - Emulation von Hardware
- Probleme
  - Sicherheit (Rootkits)
  - Echtzeitfähigkeit (OS wird unterbrochen)

ARM-Architektur

ARM v7 Prozessoren verfügen über drei Ringe

1. Hypervisor Level
2. Operating System Level
3. Application Level

Aufgaben eines Kernels

- Prozess-Steuerung (Process Scheduling)
  - Verwaltung der System-Ressource *CPU*
  - Logik wann welcher Prozess Zugriff auf die CPU erhält
- Speicherverwaltung
  - Verwaltung der System-Ressource *Arbeitsspeicher*
  - Prozessisolation (Rechteverwaltung)
- Bereitstellung eines Filesystems
  - Verwaltung der System-Ressource *Massenspeicher*
  - Erstellen, Löschen, Lesen, Schreiben, . . . von Dateien
- Hardwarezugriffe (Monitor, Tastatur, Maus, . . .)
- Netzwerk (Senden und Empfangen von Netzwerkpaketen)
- Bereitstellung einer API (Systemcalls) um Prozessen die Nutzung von Systemressourcen zu ermöglichen

Modellierung und Strukturierung von OSen

Problem

Betriebssysteme gehören zu den komplexesten Softwaresystemen!
$\Longrightarrow$ OSe sind durch Lesen des Programmcodes kaum zu verstehen.

Lösung

Reduktion der Kommunikationsbeziehungen zwischen Komponenten

Modell 1: Monolithisches System

- Ein einziges Programm im kernel mode (Ring 0)
- Betriebsssystem besteht aus einem großen binären Programm
- Alle Routinen und Funktion können sich gegenseitig aufrufen
- Oftmals können Treiber (Module) zur Laufzeit nachgeladen werden
- Andrew Tanenbaum: „The Big Mess “
- Typisch für "historisch gewachsene" Systeme
- Keine *Datenkapselung*
- Sehr effizient!

Monolithisches System – Beispiel Linux

Various layers within Linux, also showing separation between the userland and kernel space

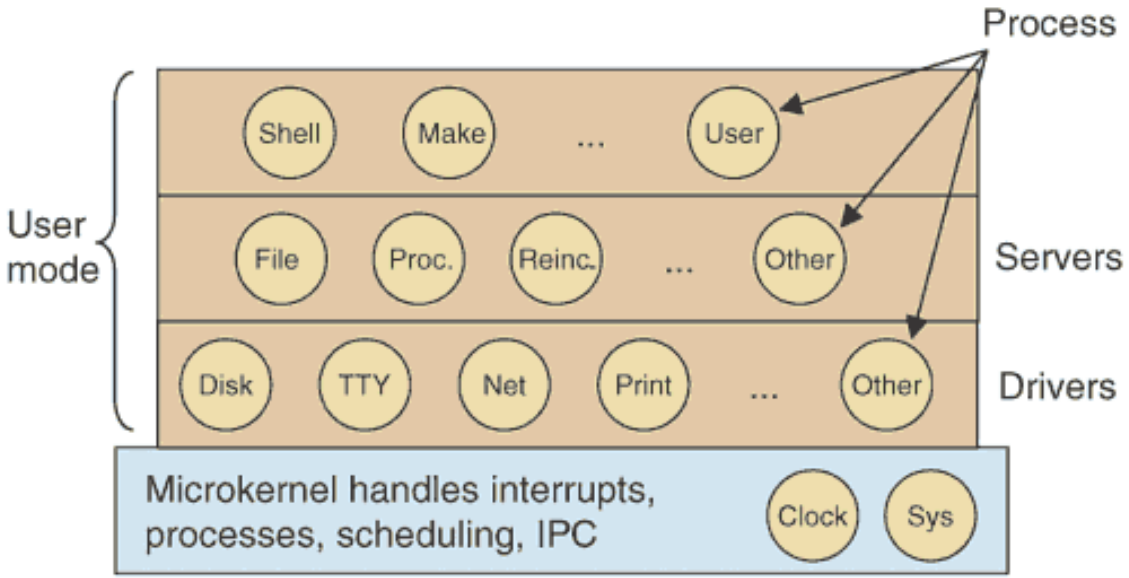
<table>
<tr><td rowspan="3">User mode</td><td>User applications</td><td colspan="5">For example, bash, LibreOffice, Apache OpenOffice, Blender, 0 A.D., Mozilla Firefox, etc.</td></tr>
<tr><td>Low-level system components:</td><td>System daemons:
systemd, runit, logind, networkd, soundd, ...</td><td>Windowing system:
X11, Wayland, Mir, SurfaceFlinger (Android)</td><td colspan="2">Other libraries:
GTK+, Qt, EFL, SDL, SFML, FLTK, GNUstep, etc.</td><td>Graphics:
Mesa, AMD Catalyst, ...</td></tr>
<tr><td>C standard library</td><td colspan="5"><code>open()</code>, <code>exec()</code>, <code>sbrk()</code>, <code>socket()</code>, <code>fopen()</code>, <code>calloc()</code>, ... (up to 2000 subroutines)
glibc aims to be POSIX/SUS-compatible, uClibc targets embedded systems, bionic written for Android, etc.</td></tr>
<tr><td rowspan="3">Kernel mode</td><td rowspan="3">Linux kernel</td><td colspan="5"><code>stat</code>, <code>splice</code>, <code>dup</code>, <code>read</code>, <code>open</code>, <code>ioctl</code>, <code>write</code>, <code>mmap</code>, <code>close</code>, <code>exit</code>, etc. (about 380 system calls)
The Linux kernel System Call Interface (SCI, aims to be POSIX/SUS-compatible)</td></tr>
<tr><td>Process scheduling subsystem</td><td>IPC subsystem</td><td>Memory management subsystem</td><td>Virtual files subsystem</td><td>Network subsystem</td></tr>
<tr><td colspan="5">Other components: ALSA, DRI, evdev, LVM, device mapper, Linux Network Scheduler, Netfilter
Linux Security Modules: SELinux, TOMOYO, AppArmor, Smack</td></tr>
<tr><td colspan="7">Hardware (CPU, main memory, data storage devices, etc.)</td></tr>
</table>

Quelle: `https://en.wikipedia.org/wiki/Linux_kernel`

Modell 2: Microkernel

- Schlanker Kernel verfügt nur über grundlegende Funktionen
  - Speicherverwaltung
  - Prozessverwaltung
  - Interprozesskommunikation
  - Elementare I/O-Funktionen
  - Interrupt Handling
- Microkernel besteht aus mehreren austauschbaren Komponenten
- Gerätetreiber laufen im Userspace
- Weitere Funktionalität durch Userspace-Programmbibliotheken
- Viele Kontextwechsel (Umschalten zwischen Prozessen)

Microkernel – Beispiel Minix3



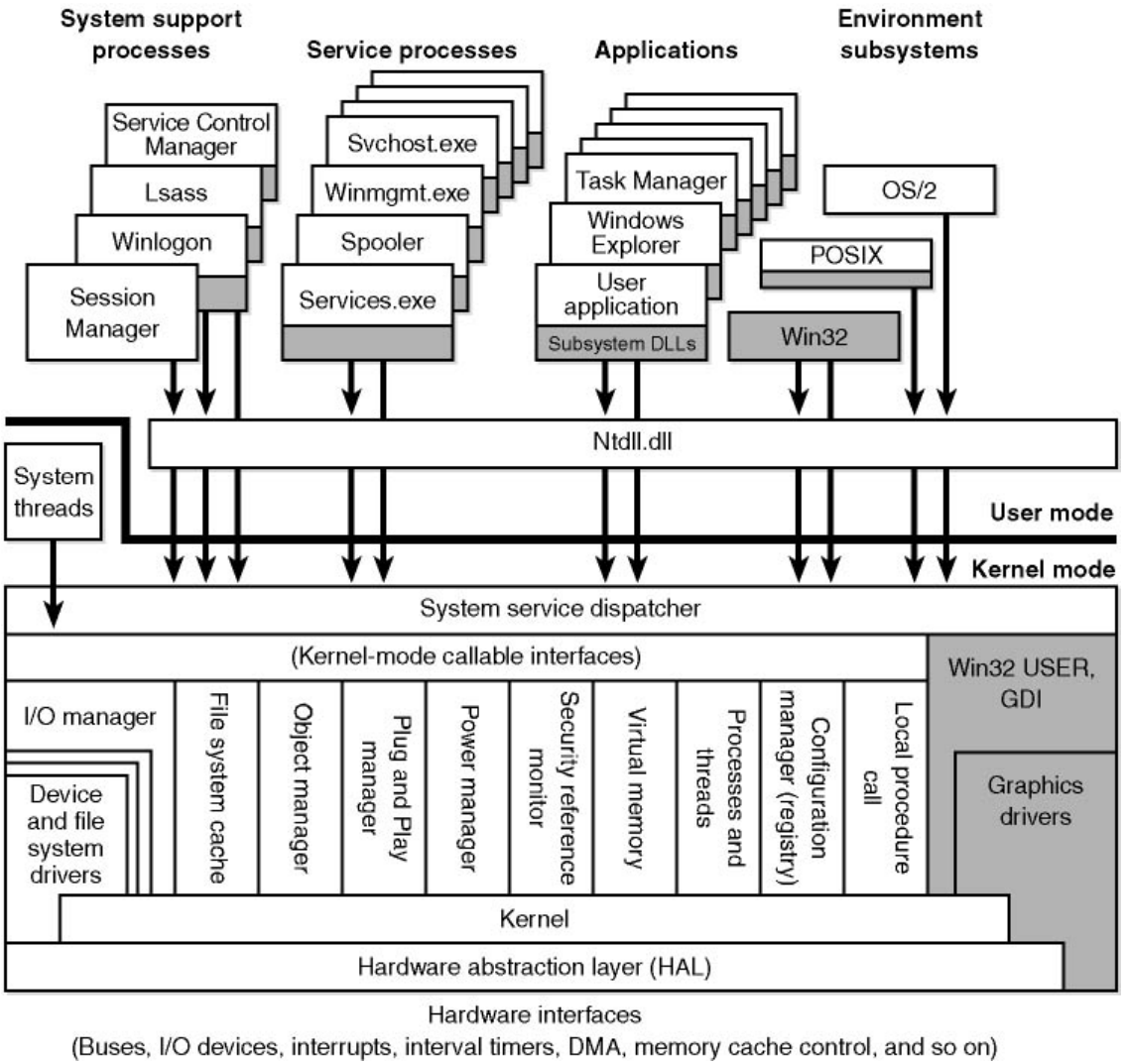
Quelle: `http://se7so.blogspot.de/2011/12/minix-3-operating-system.html`

Modell 3: Hybride Kernel

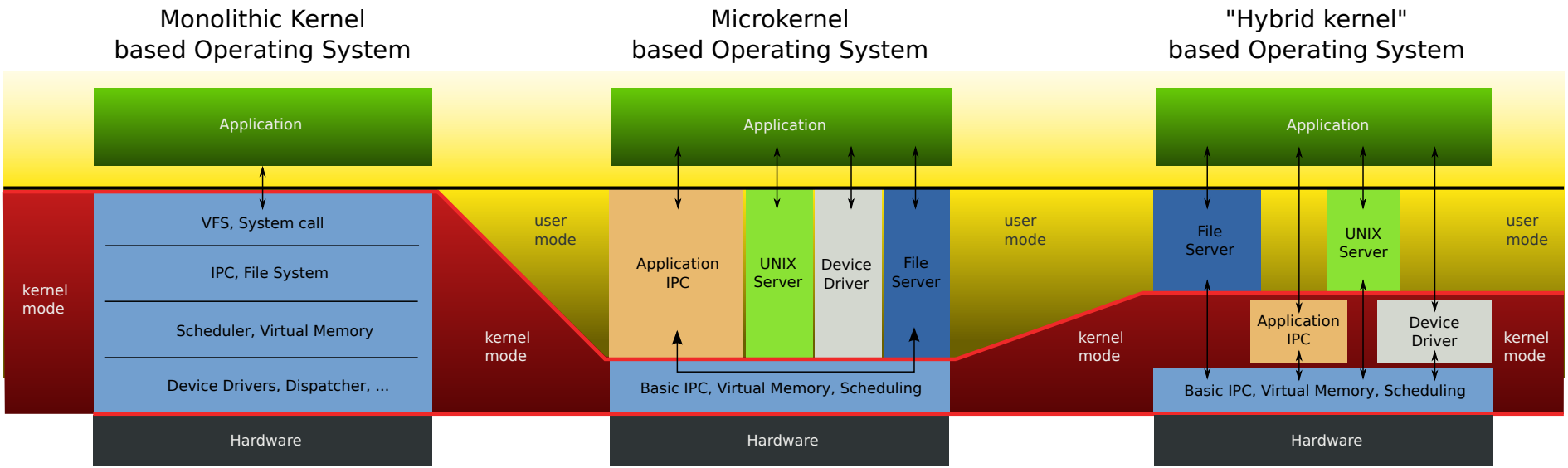
- Kompromiss zwischen Microkernel und monolithischen Kernel
- Vereinigt Vorteile des Micro- und des monolithischen Kernels
- Teile des monolithischen Kernels sind in Ring 0 integriert
- Dadurch wird die Anzahl der Kontextwechsel reduziert
- **Beispiel:** in Windows NT 4.0 läuft das Grafiksystem im Ring 0

Hybride Kernel – Beispiel Windows NT

Russinovich et al., Windows Internals, 4th edition, Seite 52



Betriebssystemmodelle – Übersicht



Quelle `http://en.wikipedia.org/wiki/Image:OS-structure.svg`

Kapitelübersicht

1. Einführung in die UNIX-Shell
2. Einführung in die Bash-Programmierung
3. Zugriffskontrolle
4. Werkzeuge zur Linux-Systementwicklung
5. C-Präprozessoranweisungen
6. Standard-I/O-Funktionen
7. Elementare-I/O-Funktionen
8. Dateiattribute, Filesysteme und Verzeichnisse