HOCHSCHULE FULDA
UNIVERSITY OF APPLIED SCIENCES

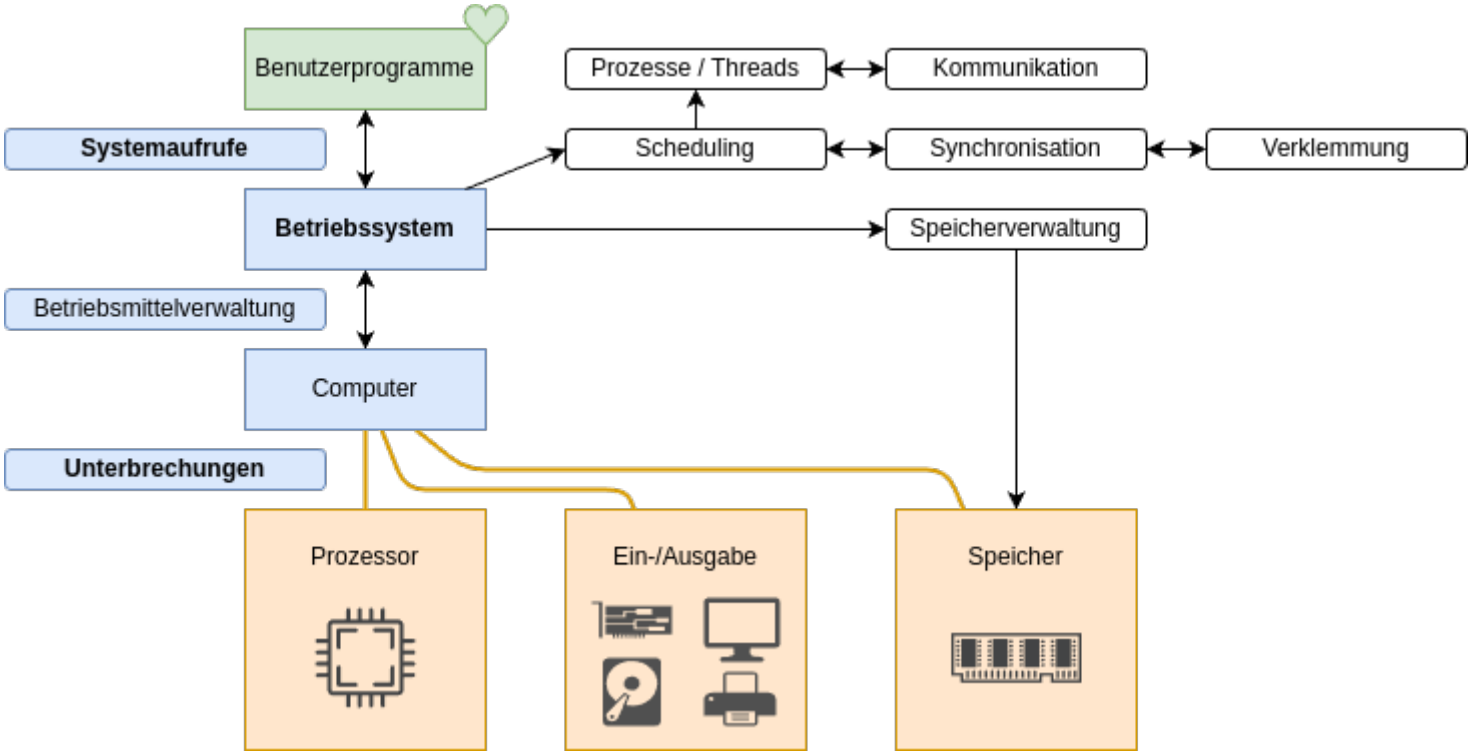
Betriebssysteme

Stage 3 – Betriebssystemkonzepte

Was bisher geschah ...

- Geschichte der Rechner und Betriebssysteme
  - Vom Minimalrechner zu “modernen” PCs
- Von-Neumann-Architektur
  - CPU (Rechen- und Steuerwerk), Speicher, I/O-Geräte
  - Verbunden über Bus-Systeme (Steuer-, Adress-, Datenbus)
- Mehrprogrammbetrieb
  - Scheinbar parallele Ausführung (Multitasking, Multithreading)
  - Echt parallele Ausführung (Multiprozessor, Multicore)
- Das Betriebssystem
  - Definition, Aufgaben, Betriebsarten

Übersicht


Benutzerprogramme
Systemaufrufe
Betriebssystem
Betriebsmittelverwaltung
Computer
Unterbrechungen
Prozesse / Threads
Kommunikation
Scheduling
Synchronisation
Verklemmung
Speicherverwaltung
Prozessor
Ein-/Ausgabe
Speicher

Themen und Lernziele

- Grundlegende Konzepte
  - Bootprozess
    (Wie startet das Betriebssystem?)
  - Sicherheitskonzept
    (Wie bekommt man das eigentlich sicher?)
  - Systemaufrufe (*system calls*)
    (Wie können Benutzerprogramme privilegierte Aktionen ausführen?
  - Unterbrechungen (Interrupts)
    (Wie kommuniziert die Hardware mit dem Betriebssystem?)

- Aufbau von Betriebssystemen

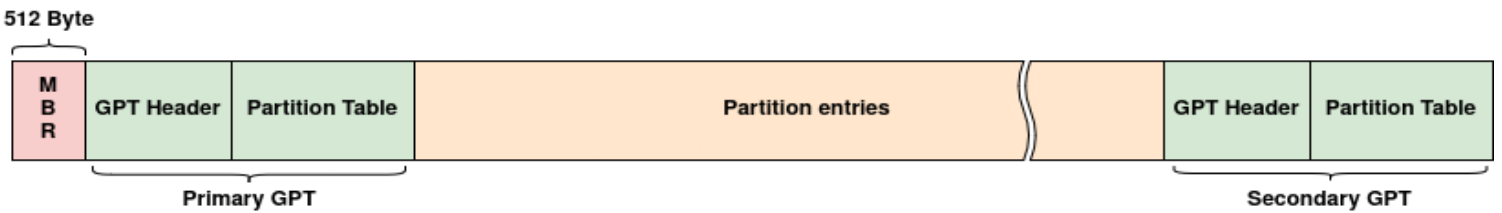
- Betriebssystemstrukturen
  - Monolithische Kerne (und Modulkonzept)
  - Mikrokerne (Spezialform Client-Server)
  - Mischform “Hybridkern” (Windows NT)
  - Virtualisierung

Bootprozess

(x86-Architektur)

1. Initialisierung der Hardware

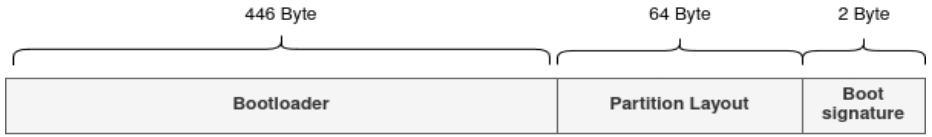
- Prozessor startet im Real Mode
  Verhalten wie 8086 CPU: Keine Konzept von Privilegstufen, Zugriff auf den gesamten adressierbaren Speicher (1 MB)
- Viele CPU-Register besitzen definierte Startwerte (reset vector)
  Der Programmzähler (EIP) ist auf den Wert 0xFFFFFFF0 initialisiert
- Start der Programmausführung (BIOS/UEFI)
  Erste durch die CPU ausgeführte Anweisung ist ein JMP an die Stelle im Speicher, wo der Programmcode des BIOS steht
- Power-On-Self-Test (POST)
  Programmcode des BIOS führt einen Power-On-Self-Test (POST) durch, detektiert und konfiguriert angeschlossene Geräte
- Suche nach einem Boot-Device
  Suche nach Master Boot Record (MBR; Boot Signatur “0x55, 0xAA” in Sektor 0, Offset 510 und 511) auf bootfähigen Geräten



2. Bootloader

Master Boot Record (MBR) einlesen

Einlesen des ersten Sektor (512 Byte) des gefundenen Bootdevice an eine fest definierte Position im Hauptspeicher



Master Boot Record (MBR)

Bootloader ausführen

Es handelt sich gerade um genug Code um die Boot-Partition auszuwählen und den dortigen boot sector (erster Sektor in der Partition) zu laden, mit dessen Hilfe weiterer Bootloader-Programmcode von der Festplatte geladen wird (z.B. GRUB Stage 2, C:\NTLDR)

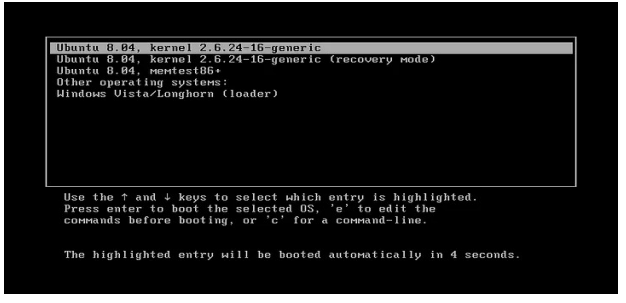
Ausführen des Stage 2 Bootloader

Laden der Bootloaderkonfiguration (z.B. grub.cfg, boot.ini) und ggf. anzeigen eines “Boot-Menu” (z.B. versch. Kernelversionen, “Abgesicherter Modus”)

Hier ggf. auch Aktivierung des Protected Mode der CPU
→ Dadurch Nutzung von Privilege Levels, Zugriff auf den gesamten adressierbaren Speicher, Speicherschutz, usw.

```
Absolute sector 0 (cylinder 0, head 0, sector 1)
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0000  EB 48 90 00 00 00 00 00 00 00 00 00 00 00 00 00  .H..............
0010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 03 02  ................
0040  80 00 00 80 DF 0A 93 01 00 08 FA EA 50 7C 00 00  ............P|..
0050  31 C0 8E D8 8E D0 BC 00 20 FB A0 40 7C 3C FF 74  1....... ..@|<.t
0060  02 88 C2 52 BE 76 7D E8 34 01 F6 C2 80 74 54 B4  ...R.v}.4....tT.
0070  41 BB AA 55 CD 13 5A 52 72 49 81 FB 55 AA 75 43  A..U..ZRrI..U.uC
0080  A0 41 7C 84 C0 75 05 83 E1 01 74 37 66 8B 4C 10  .A|..u....t7f.L.
0090  BE 05 7C C6 44 FF 01 66 8B 1E 44 7C C7 04 10 00  ..|.D..f..D|....
00A0  C7 44 02 01 00 66 89 5C 08 C7 44 06 00 70 66 31  .D...f.\..D..pf1
00B0  C0 89 44 04 66 89 44 0C B4 42 CD 13 72 05 BB 00  ..D.f.D..B..r...
00C0  70 EB 7D B4 08 CD 13 73 0A F6 C2 80 0F 84 F3 00  p.}....s........
00D0  E9 8D 00 BE 05 7C C6 44 FF 00 66 31 C0 88 F0 40  .....|.D..f1...@
00E0  66 89 44 04 31 D2 88 CA C1 E2 02 88 E8 88 F4 40  f.D.1..........@
00F0  89 44 08 31 C0 88 D0 C0 E8 02 66 89 04 66 A1 44  .D.1......f..f.D
0100  7C 66 31 D2 66 F7 34 88 54 0A 66 31 D2 66 F7 74  |f1.f.4.T.f1.f.t
0110  04 88 54 0B 89 44 0C 3B 44 08 7D 3C 8A 54 0D C0  ..T..D.;D.}<.T..
0120  E2 06 8A 4C 0A FE C1 08 D1 8A 6C 0C 5A 8A 74 0B  ...L......l.Z.t.
0130  BB 00 70 8E C3 31 DB B8 01 02 CD 13 72 2A 8C C3  ..p..1......r*..
0140  8E 06 48 7C 60 1E B9 00 01 8E DB 31 F6 31 FF FC  ..H|`......1.1..
0150  F3 A5 1F 61 FF 26 42 7C BE 7C 7D E8 40 00 EB 0E  ...a.&B|.|}.@...
0160  BE 81 7D E8 38 00 EB 06 BE 8B 7D E8 30 00 BE 90  ..}.8.....}.0...
0170  7D E8 2A 00 EB FE 47 52 55 42 20 00 47 65 6F 6D  }.*...GRUB .Geom
0180  00 48 61 72 64 20 44 69 73 6B 00 52 65 61 64 00  .Hard Disk.Read.
0190  20 45 72 72 6F 72 00 BB 01 00 B4 0E CD 10 AC 3C   Error........<
01A0  00 75 F4 C3 00 00 00 00 00 00 00 00 00 00 00 00  .u..............
01B0  00 00 00 00 00 00 00 00 A8 E1 A8 E1 00 00 80 01  ................
01C0  01 00 07 FE FF 6D 3F 00 00 00 AF 39 D7 00 00 00  .....m?....9....
01D0  C1 6E 0C FE FF FF EE 39 D7 00 BD 86 BB 00 00 FE  .n.....9........
01E0  FF FF 83 FE FF FF AB C0 92 01 CD 2F 03 00 00 FE  ........../....
01F0  FF FF 0F FE FF FF 78 F0 95 01 83 AF CC 00 55 AA  ......x.......U.
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
```

GRUB MBR example using version 0.92/0.93 code.



Beispiel: Boot-Menu

3. Initialisierung des Betriebssystemkerns

Laden des Betriebssystemkerns

Der Betriebssystemkern (z.B. /boot/vmlinuz-6.0.2.amd64, C:\Windows\System32\ntoskrnl.exe) wird von der Festplatte in den Hauptspeicher geladen bzw. entpackt

(Hinweis: Bereits zu dieser Zeit müssen Treiber bereitgestellt werden um z.B. von einem Dateisystem zu lesen, Dateien zu entpacken, usw.)

- **Beispiel Linux: Ausführen von start_kernel()**
  Initialisierung des Betriebssystemkern

- **Vorbereiten von Datenstrukturen**
  Eine Reihe globaler Datenstrukturen (z.B. Global Descriptor Table (GDT), Local Descriptor Table (LDT), Interrupt Descriptor Table (IDT)) muss vorbereitet und mit validen Daten gefüllt werden. Zudem müssen CPU-Register mit den Adressen dieser Descriptor Tabellen initialisiert werden

- **Bereitstellen von Service Routinen**
  Diverse Service Routinen (z.B. für Interrupts, Systemaufrufe, ...) werden an definierten Positionen im Hauptspeicher initialisiert (diese können - z.B. im Falle von Interrupts - durch den IDT gefunden werden)

- **Initialisierung von Hardware und Treibern**
  Das Betriebssystem erkennt Hardware, lädt entsprechende Treiber und stellt das Root-Dateisystem bereit

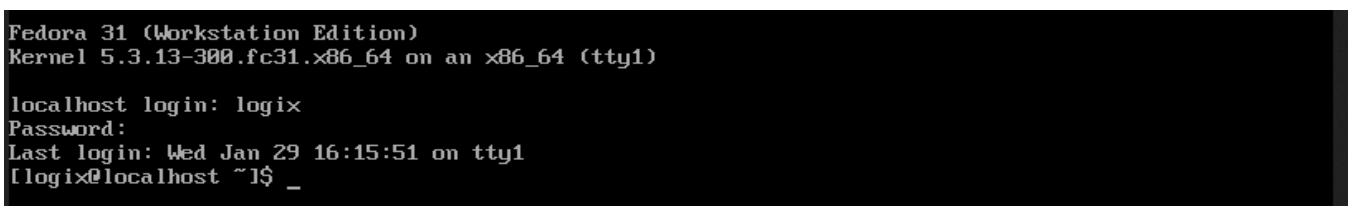
4. Initialisierung des Systems

- **Ausführen der zentralen Systemmanagement-Komponente**
  Linux-Systeme verwenden i.d.R. init bzw. systemd als Systemmanagement-Komponente. Diese wird durch den Linux-Kernel gestartet und erhält die Prozess-ID 1. Sie erhält nun die Kontrolle über den weiteren Systemstart.

  - **Konfiguration des Systems**
    Setzen des Hostnamen, Einstellen von Kernel-Parametern, Einhängen von Dateisystemen, Laden zusätzlicher Treiber

  - **Starten von Systemdiensten**
    Zum Beispiel Dienste zur Zeitsynchronisation, IP-Adresskonfiguration, Logging, Event-Management, Bluetooth, usw.

  - **Bereitstellen eines virtuellen Terminal zur Nutzerinteraktion**
    Dies kann ein textbasiertes Terminal, ein grafisches Benutzerinterface, oder beides sein



Sicherheitskonzept

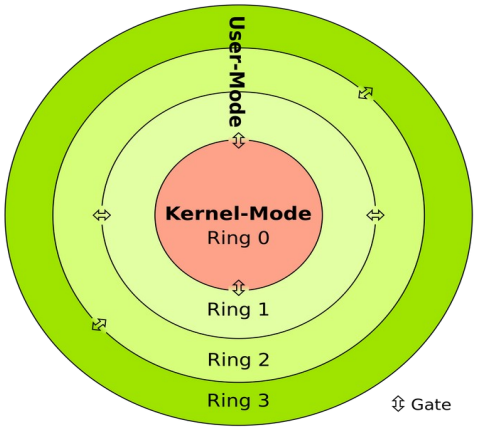
Rückblick: Von-Neuman-Architektur

- Die CPU arbeitet kontinuierlich Befehle ab
  - Diese befinden sich „aufeinanderfolgend“ im Hauptspeicher

- Wie kann sichergestellt werden, dass Benutzerprogramme nicht
  - auf „fremden“ Speicher zugreifen
  - endlos weiterlaufen

- Benötigt: Grundlegendes Sicherheitskonzept
  - Unterscheidung zwischen privilegiertem Betriebssystem und unprivilegierten Benutzerprogrammen

Sicherheitskonzept



Bildquelle: Wikipedia

- CPUs bieten Privilegierungsstufen
  - Beschränkung von Rechten (z.B. ausführbare Instruktionen)
  - Wechsel zwischen Privilegienstufen mittels sog. Gates (z.B. Call-Gates, Interrupt-Gates)

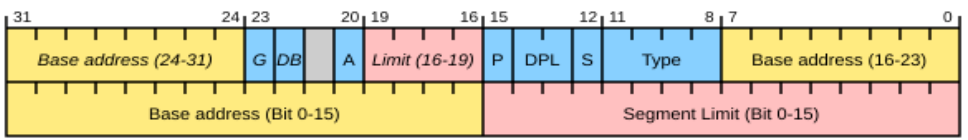
- Beispiel: x86-Architektur
  - Ring 0: Kernel-Modus (Höchste Privilegstufe)
    Programmcode dieser Stufe darf alle verfügbaren CPU-Anweisungen ausführen (z.B. Deaktivieren von Interrupts, Verändern von Kontrollregistern), auf den gesamten Speicher und auf alle I/O-Geräte zugreifen
  - *Ring 1+2: Treiber-Modus*
    In Windows und Linux nicht verwendet
    In OS/2 für wurde Ring 2 für Grafiktreiber genutzt
  - Ring 3: Benutzer-Modus (Niedrigste Privilegstufe)
    Programmcode dieser Stufe kann nur eingeschränkt CPU-Anweisungen ausführen und muss den Zugriff auf Speicher und I/O-Geräte über das Betriebssystem “beantragen”. Dies geschieht durch Nutzung von Systemaufrufen (*system-call*, TRAP)

- Beispiel RISC-V-Architektur: machine-, hypervisor-, supervisor- und user-mode
- Beispiel ARM-Architektur: Exception Level 0 – 3

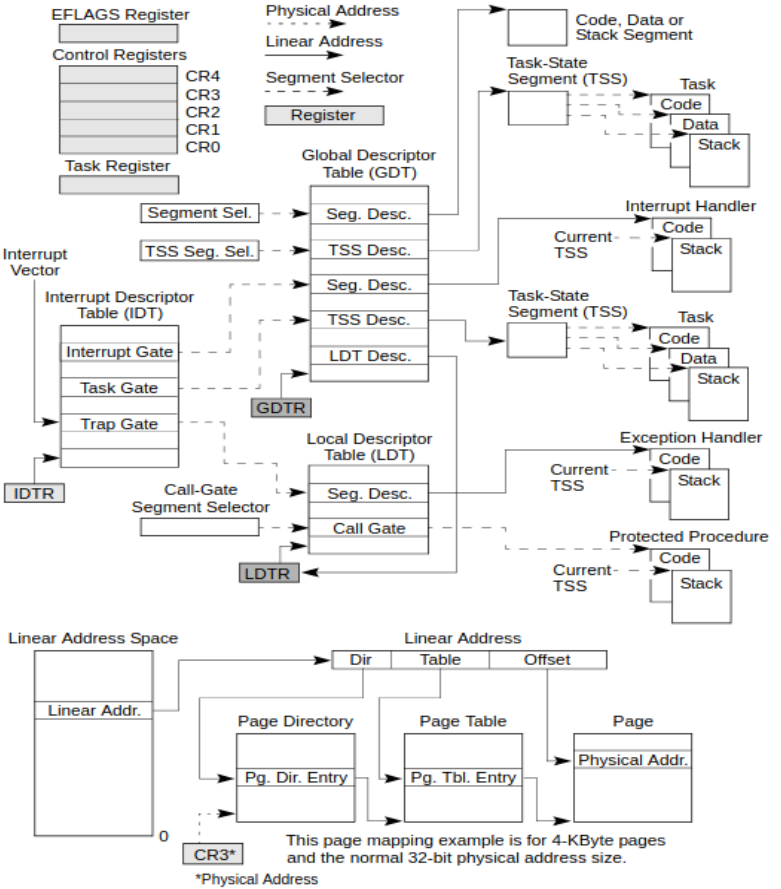
Sicherheitskonzept: Control Registers

- Diverse CPU Register definieren Privilegien
  Umschaltung zw. Real- und Protected Mode, Aktivierung von Speicherschutz, usw. durch Veränderung der entsprechenden CPU-Register.
  Diese Veränderung ist nur im Privilegierten Modus möglich.

- Beispiel für solche Register
  - Control Registers (CR0-8)
    (Beispiel: Protection Mode Enable (CR0, Bit 0))
  - Protected Mode Registers
    (Global-, Local-, Interrupt Descriptor Table (GDT, LDT, IDT))
  - Segment Selector
    (Gibt die Art eines Speichersegments an (Code, Data, Stack, ...)



Format eines *segment descriptor* im GDT
(Bild: Wikipedia)



IA-32 System-Level Registers and Data Structures
(Bild: Intel® 64 and IA-32 Architectures Software Developer's Manual)

Systemaufrufe (*system calls*)

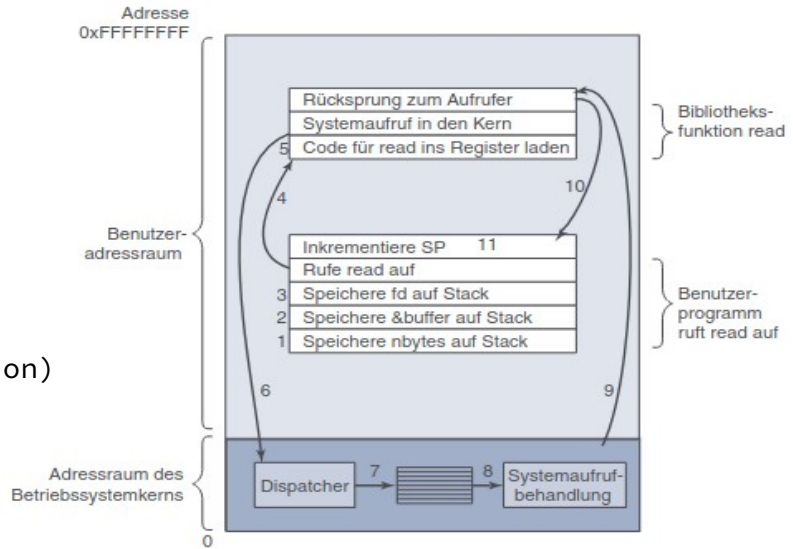
Systemaufrufe (Tanenbaum, Kapitel 1.6)

- Mit Hilfe eines Systemaufruf (system call) ist es Programmen (im Benutzer-Modus) möglich, Funktionen bzw. Dienste des Betriebssystems (die im Kernel-Modus ausgeführt werden) in Anspruch zu nehmen

- Implementierung per Software Interrupt (TRAP)
  (Siehe auf x86 auch: SYSENTER/SYSEXIT (Intel), SYSCALL/SYSRET (AMD))

- Beispiel: read (int count = read (fd, buffer, nbytes);)
  - 1-3: Benutzerprogramm legt Parameter auf Stack
  - 4: Sprung in Bibliotheksfunktion “read”
  - 5: Laden der Nummer des Systemaufrufs (read) in CPU-Register
  - 6: Ausführen des SYSENTER-Befehls (Wechsel in den Kernel-Modus)
  - 7: Lokalisieren des Prozedur-Code für den Systemaufruf
  - 8: Ausführen der Systemfunktion
  - 9: SYSEXIT-Befehl (Rückgabe der Kontrolle an die Bibliotheksfunktion)
  - 10+11: Rücksprung in das Benutzerprogramm und aufräumen des Stack

- Der Kern selbst ist dabei passiv



Bildquelle: [Tanenbaum]

Beispiel: Systemaufrufe

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing, or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

Directory- and file-system management

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

Miscellaneous

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970

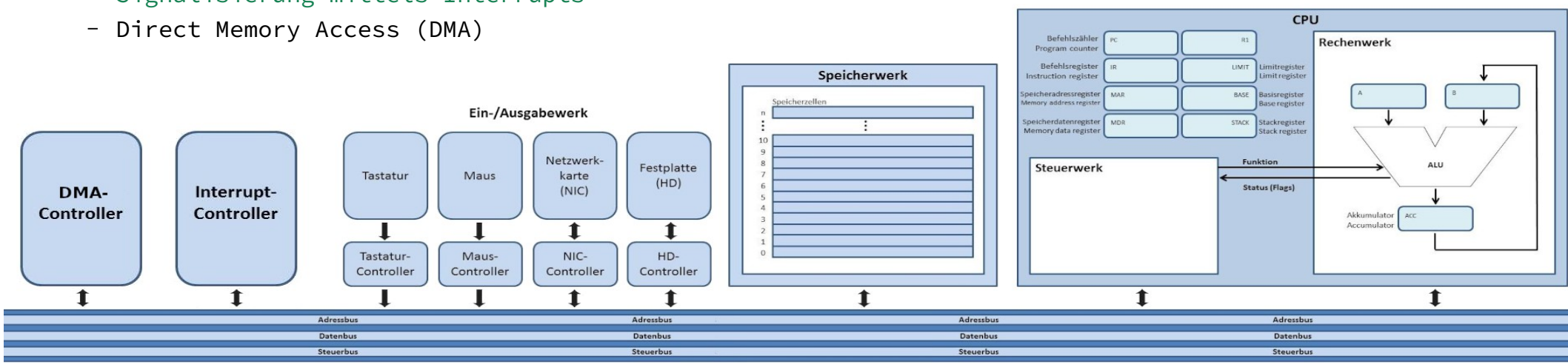
UNIX	Win32	Beschreibung
fork	CreateProcess	Erzeugen eines neuen Prozesses
waitpid	WaitForSingleObject	Warten auf das Ende eines Prozesses
execve	(nicht vorhanden)	CreateProcess = fork + execve
exit	ExitProcess	Ausführung beenden
open	CreateFile	Erzeugen einer Datei oder Öffnen einer existierenden Datei
close	CloseHandle	Datei schließen
read	ReadFile	Daten aus einer Datei lesen
write	WriteFile	Daten in eine Datei schreiben
lseek	SetFilePointer	Dateizeiger bewegen
stat	GetFileAttributesEx	Dateiattribute erfragen
mkdir	CreateDirectory	Erzeugen eines neuen Verzeichnisses
rmdir	RemoveDirectory	Löschen eines leeren Verzeichnisses
link	(nicht vorhanden)	Win32 unterstützt keine Links
unlink	DeleteFile	Löschen einer existierenden Datei
mount	(nicht vorhanden)	Win32 unterstützt kein Einhängen
umount	(nicht vorhanden)	Win32 unterstützt kein Einhängen
chdir	SetCurrentDirectory	Ändern des aktuellen Arbeitsverzeichnisses
chmod	(nicht vorhanden)	Win32 unterstützt Security nicht (NT schon)
kill	(nicht vorhanden)	Win32 unterstützt keine Signale
time	GetLocalTime	Aktuelle Zeit erfragen

Unterbrechungen (*interrupts*)

Eingabe / Ausgabe: Ablauf

- Beispiel “Daten einlesen” (Schritt 8 im Systemaufruf-Beispiel)
  - *Ein Prozess fordert das Betriebssystem per Systemaufruf auf, Daten von einer Festplatte zu lesen*
  - Die CPU sendet dem Festplattencontroller einen I/O-Befehl
  - Der Controller lokalisiert die Daten auf der Festplatte
  - Der Festplattencontroller löst einen Interrupt aus
  - Die Daten werden durch die CPU in den Hauptspeicher geschrieben
  - Der Prozess kann nun auf die angeforderten Daten zugreifen

- Arten der Durchführung von Ein/Ausgabe (I/O)
  - Aktives Warten auf Beendigung (Polling)
  - Signalisierung mittels Interrupts
  - Direct Memory Access (DMA)



Unterbrechungen (Interrupts) (Tanenbaum, Kapitel 5.1.5)

- Unterbrechung der Programmausführungum auf (zeitkritische) Ereignisse zu reagieren
  - Asynchron → Keine deterministischer Unterbrechungszeitpunkt

- Begriffe
  - Interrupt: Eine kurzzeitige Unterbrechung der regulären Programmausführung
  - Interrupt request (IRQ): Die Unterbrechungsanforderung bezeichnet das auslösende Ereignis
  - Interrupt service routing (ISR): Eine Unterbrechungsroutine wird in Folge der Unterbrechung ausgeführt
  - Interrupt Descriptor Table (IDT): Eine Datenstruktur im Speicher, anhand derer für einen aufgetretenen IRQ die richtige ISR bestimmt wird

- Interrupts stellen ein essentielles Konzept zur Realisierung von *präemptivem Multitasking* dar

IRQ	Verwendung
0	System-Taktgeber
1	Tastatur
2	Kaskadiert zu IRQ 9 (für 8–15)
3	COM 2, 4, 6, 8 (EIA-232/RS-232)
4	COM 1, 3, 5, 7
5	LPT 2 (IEEE 1284) oder Soundkarte
6	Diskettenlaufwerk (Floppy)
7	LPT 1
8	Echtzeituhr (RTC)
9	Zu IRQ 2 umgeleitet (aber auch VGA und NIC, IRQ 16–23)
10	Frei ggf. PCI-Bus
11	Frei ggf. Adaptec-SCSI
12	PS/2 (Maus, andere Eingabegeräte)
13	Mathematischer Coprozessor (FPU)
14	Primärer IDE oder ATA
15	Sekundärer IDE oder ATA

Beispiele für Interruptanforderungen (Tabelle: Wikipedia)

Unterbrechungen: Funktionsweise

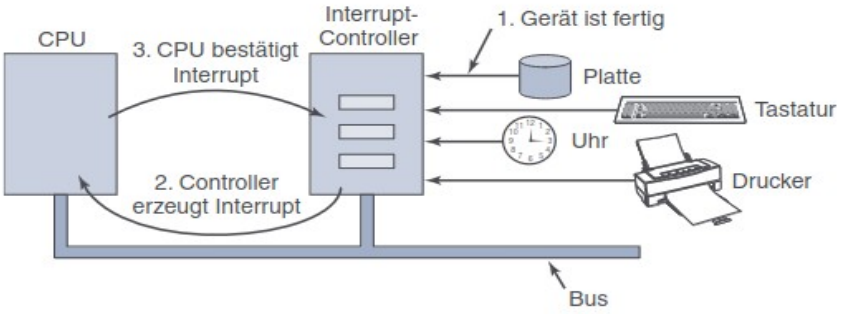
- Funktionsweise am Beispiel Disk-I/O

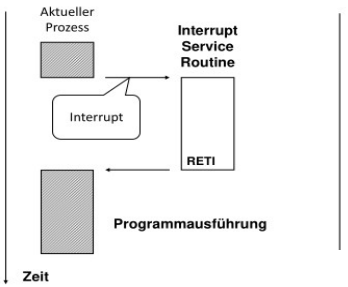
  - 1. Gerät ist fertig
    Die angeforderten Daten stehen im Cache des Festplatten-Controller zur Verfügung. Dies signalisiert der Festplatten-Controller gegenüber dem Interrupt-Controller.

  - 2. Controller erzeugt Interrupt
    Der Interrupt-Controller empfängt die Anforderung vom Festplatten-Controller und signalisiert einen Interrupt Request (IRQ) an die CPU

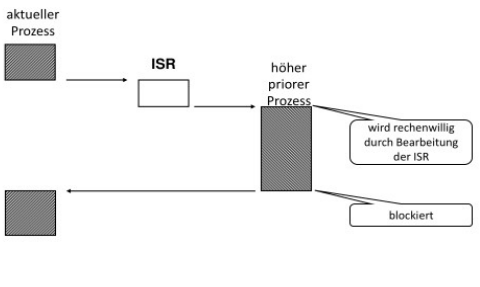
  - Kontextwechsel in die Interrupt Service Routine (ISR)
    Die Speicheradresse der ISR wird aus der Interrupt Nummer mit Hilfe des Interrupt Descriptor Table (IDT) bestimmt.
    Die ISR wird ausgeführt und anschließend muss der Kontext des zuvor aktiven Prozesses wiederhergestellt werden.

  - 3. CPU bestätigt Interrupt
    Die CPU signalisiert dem Interrupt-Controller, dass der Interrupt Request fertig bearbeitet wurde



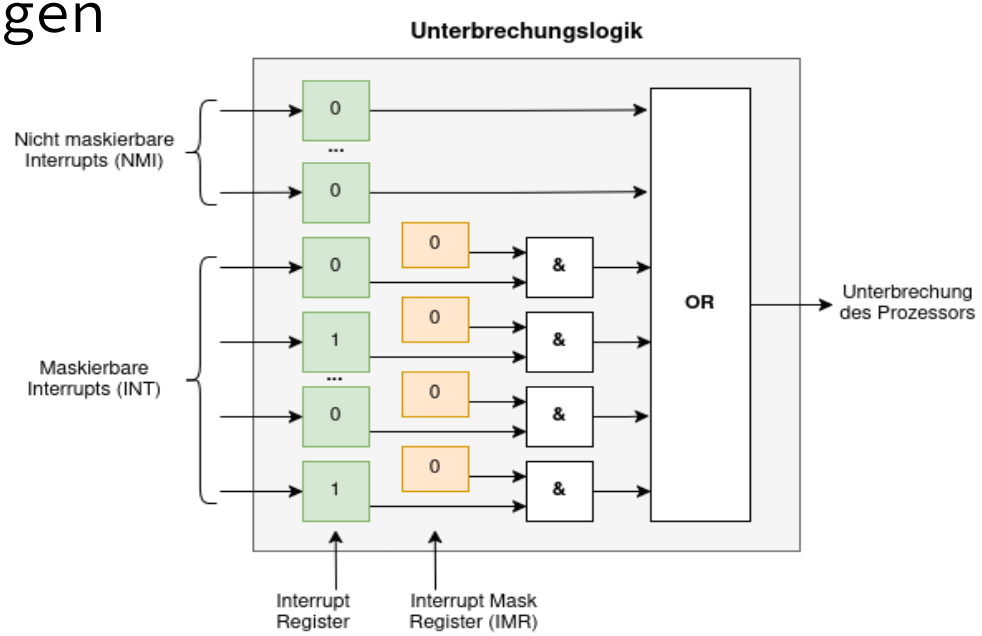


Unterbrechung und Rückkehr zum vorherigen Prozess



Unterbrechung und Wechsel zum anderem Prozess

Unterbrechungen: Unterscheidungen



- Maskieren von Interrupts
  - Nicht-maskierbare Interrupts
    Diese Interrupts können nicht ignoriert werden (Hardware-Fehler, Watchdog timer, usw.).
    Nutzen i.d.R. den NMI-Pin der CPU
  - Maskierbare Interrupts
    Können mit Hilfe des *Interrupt Mask Register* (IMR) ignoriert werden. Den versch. maskierbaren Interrupts ist jeweils ein entsprechendes Masken-Bit zugeordnet.
    Nutzen i.d.R. den INT-Pin der CPU
- Auslöser
  - Hardware Interrupts
    Werden durch Geräte (z.B. Tastatur, Festplatte, Drucker) ausgelöst und führen zu einer Reaktion des Betriebssystems auf eine Geräteanforderung
  - Software Interrupts
    Werden durch Software (den aktuell laufenden Prozess) ausgelöst und können zum Beispiel zur Realisierung von Systemaufrufen genutzt werden
- Unterscheidung auch nach Präzise vs. Unpräzise
  (vgl. CPU-Pipeline, Superskalare CPU, usw.)

Interrupts Anzeigen:

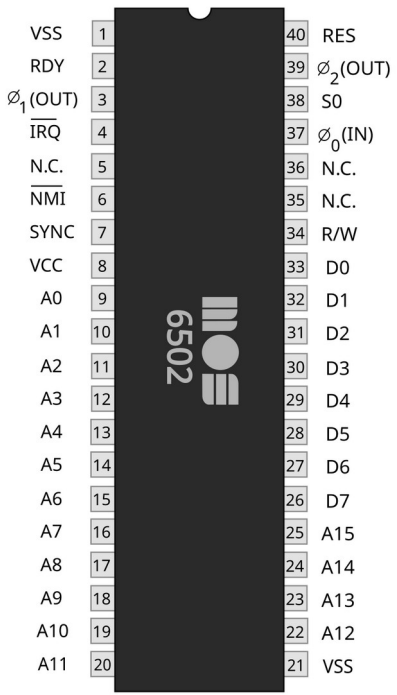
Linux: cat /proc/interrupts
Windows: msinfo32.exe → Hardwareressourcen → IRQs

Unterbrechungen

- Erste CPU mit Interrupt-Funktionalität schon in den 50er Jahren

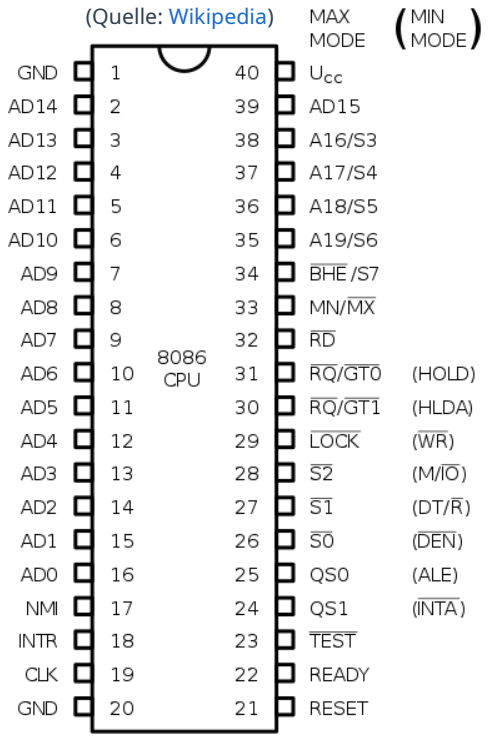
MOS 6502 (1975)

(Quelle: Wikipedia)



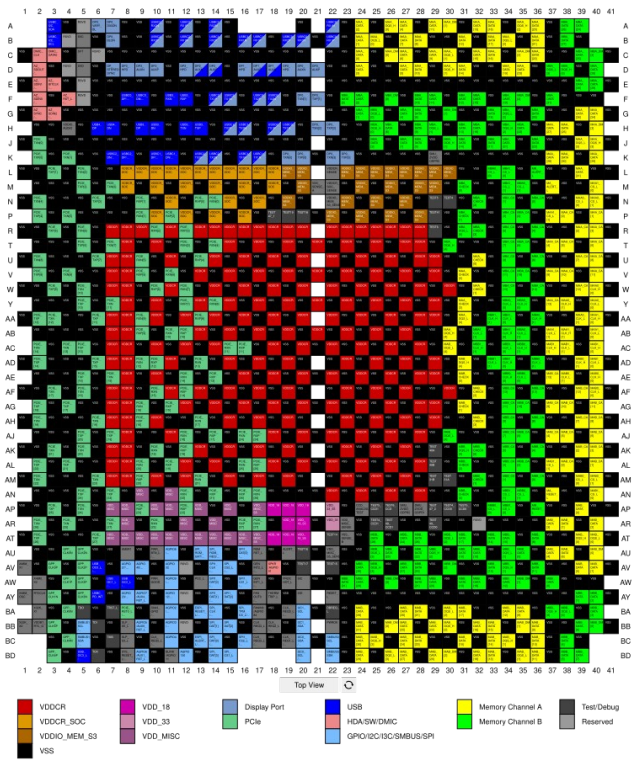
Intel 8086 (1978)

(Quelle: Wikipedia)



AMD AM5 Sockel (2022)

(Quelle: Wikipedia)



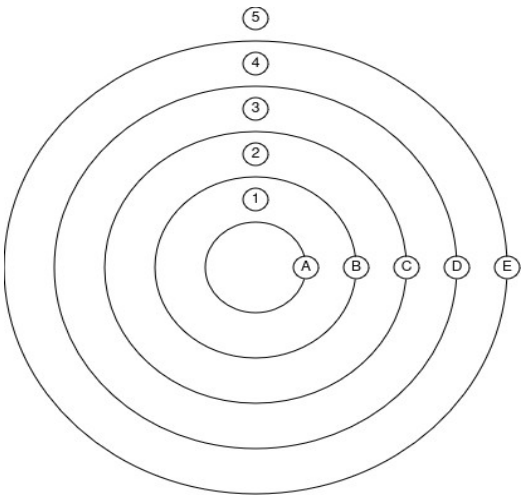
Betriebssystemaufbau

Schichtenmodell

- Betriebssysteme werden als sich umschließende Schichten strukturiert
  - Nach außen werden immer abstraktere Funktionen bereitgestellt
  - Es kommen i.d.R. drei oder mehr Schichten zum Einsatz

- Minimalaufbau
  - Die Innerste Schicht enthält die hardwarespezifischen Teile des Betriebssystems
  - Die Mittlere Schicht enthält Bibliotheken und Schnittstellen
  - Die Äußere Schicht enthält die Anwendungsprogramme und Benutzerschnittstelle

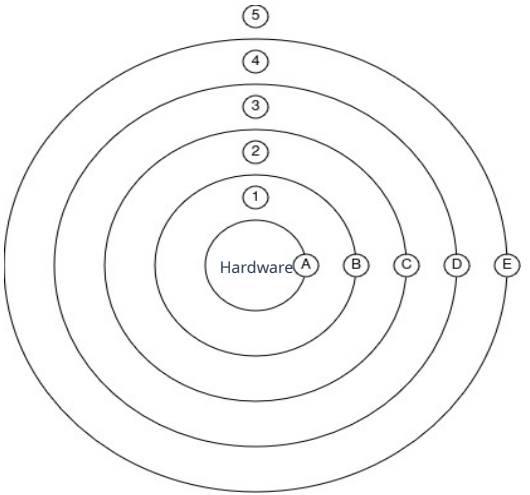
- Funktionsweise
  - Schichten kommunizieren jeweils mit den benachbarten Schichten, indem sie Funktionen der nächst inneren Schicht aufrufen
  - Gleichzeitig stellen die Schichten jeweils Funktionen für die jeweils nächst äußere Schicht zur Verfügung
  - Angebotene Funktionen und einzuhaltende Regeln einer Schicht nennt man Protokoll



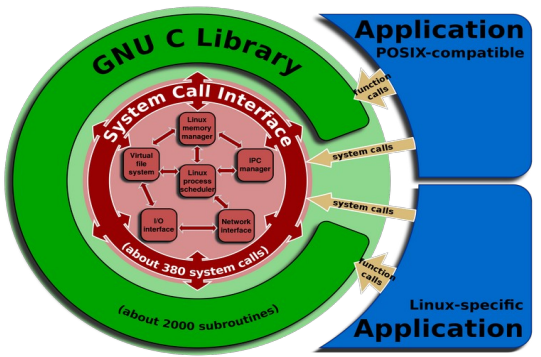
Schichtenmodell: Beispiel Linux

- Schichten
  - 1) Kernel (Maschinenabhängiger Teil)
  - 2) Kernel (Maschinenunabhängiger Teil)
  - 3) Standardbibliothek (glibc)
  - 4) Shell, Anwendungen
  - 5) Benutzer:in

- Schnittstellen
  - A) Hardware-Schnittstelle
  - B) “Virtuelle” Hardware-Schnittstelle im Kernel
  - C) Systemaufruf-Schnittstelle (system call interface)
  - D) Bibliothek-Schnittstelle
  - E) Benutzer-Schnittstelle



Beispiel: Schichten UNIX/Linux



GNU C Interface (Lary Ewing, Wikipedia)

Big Picture

- Im Bild
  - Schichtenmodell
  - Sicherheitskonzept (Kernel Mode /User Mode)

User mode	**User applications**	*bash*, *LibreOffice*, *GIMP*, *Blender*, *0 A.D.*, *Mozilla Firefox*, ...				
	System components	**init daemon**: *OpenRC*, *runit*, *systemd*...	**System daemons**: *polkitd*, *smbd*, *sshd*, *udevd*...	**Windowing system**: *X11*, *Wayland*, *SurfaceFlinger* (*Android*)	**Graphics**: *Mesa*, *AMD Catalyst*, ...	**Other libraries:** *GTK*, *Qt*, *EFL*, *SDL*, *SFML*, *FLTK*, *GNUstep*, ...
	C standard library	`fopen`, `execv`, `malloc`, `memcpy`, `localtime`, `pthread_create`... (up to 2000 subroutines) *glibc* aims to be fast, *musl* aims to be lightweight, *uClibc* targets embedded systems, *bionic* was written for Android, etc. All aim to be POSIX/SUS-compatible.				
Kernel mode	**Linux kernel**	`stat`, `splice`, `dup`, `read`, `open`, `ioctl`, `write`, `mmap`, `close`, `exit`, etc. (about 380 system calls) The Linux kernel System Call Interface (SCI), aims to be POSIX/SUS-compatible[3]				
		Process scheduling subsystem	IPC subsystem	Memory management subsystem	Virtual files subsystem	Networking subsystem
		Other components: ALSA, DRI, evdev, klibc, LVM, device mapper, Linux Network Scheduler, Netfilter Linux Security Modules: *SELinux*, *TOMOYO*, *AppArmor*, *Smack*				
Hardware (CPU, main memory, data storage devices, etc.)						

Betriebssystemstrukturen

Betriebssystemstrukturen (Tanenbaum, Kapitel 1.7)

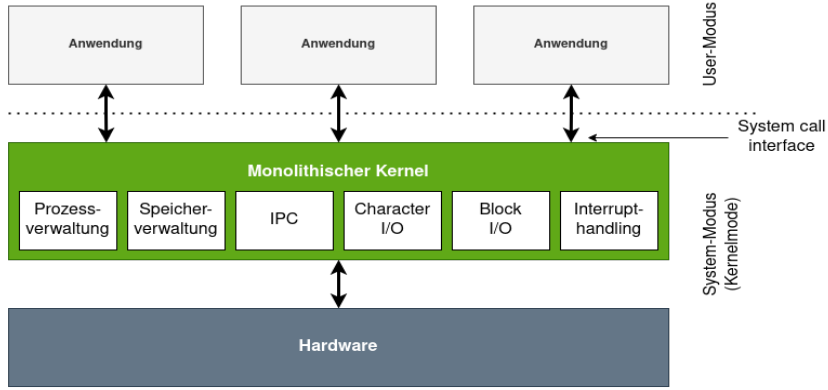
- Makrokern (Monolithischer Betriebssystemkern)
  - Verallgemeinerung “geschichtetes System”, z.B. THE System (Dijkstra, 1968)
  - Sonderfall “Dynamischer” Betriebssystemkern (→ Modulsystem)

- Mikrokern (Client-Server-Modell)
  - Sonderfall ”Verteilte Systeme”
  - Sonderfall “Hybridkern”

- Virtualisierung
  - Typ-1 Hypervisor
  - Typ-2 Hypervisor
  - Paravirtualisierung

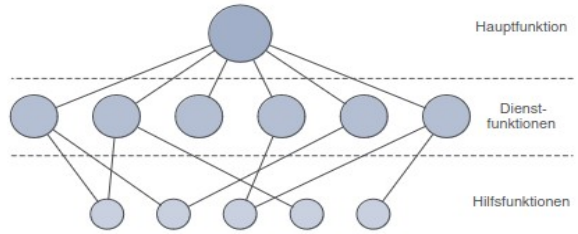
Strukturen: Makrokern (Monolithische Systeme)

- Sämtliche Prozeduren des Betriebssystems werden zu einer einzigen ausführbaren Datei verknüpft
  - Dieser “Kern” enthält alle traditionellen Funktionen (Prozessverwaltung, Hauptspeicherverwaltung, Dateiverwaltung, Netzwerkdienste, …), sowie alle Gerätetreiber, die eventuell benötigt werden
  - Alle Funktionen werden im selben Adressraum (*kernel space*) erbracht



- Problem: Erweiterung erfordert Neuübersetzung
  - Lösung: Modulkonzept
    (Nachladen von Funktionalität in den Adressraum des Kernel)

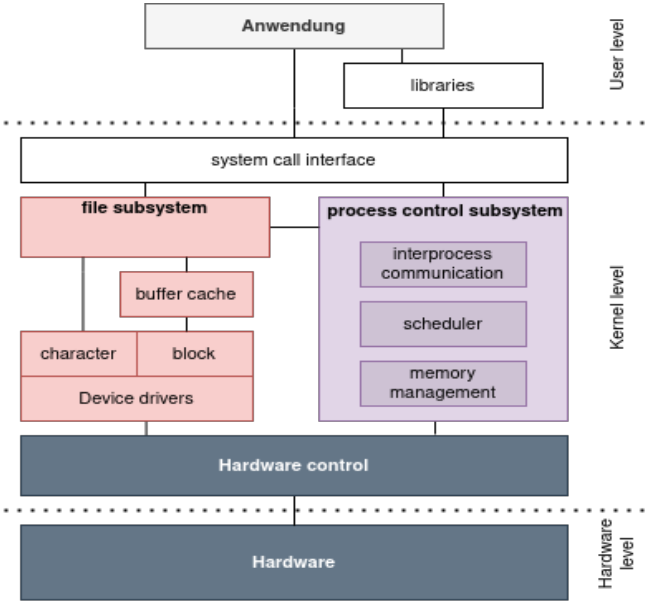
- Bild: Innere Struktur eines monolithischen Kernel
  Der Kern besteht aus einer Menge passiver Prozeduren, daher nennt man solche Betriebssysteme Prozedurorientiert.



Beispiel: Monolithische Systeme: UNIX, Linux, Windows

- Der UNIX Betriebssystemkern ist monolithisch und proprietär
  - Benutzerprogramme nutzen OS-Dienste direkt oder durch Bibliotheken (System-Call-Schnittstelle)
  - Jedes Objekt im System lässt sich als (virtuelle) Datei darstellen, jede Datei ist einfach ein unformatierter Strom von Bytes

- Auch Linux wurde als Monolithischer Kernel in Anlehnung an UNIX / POSIX entwickelt
  (LINUX is obsolete, A. Tannenbaum, 1992 (Link))
  - Der „eigentliche“ Kern enthält nur häufig genutzte Funktionen
  - Weitere Funktionalitäten sind als Module konzipiert und werden bei Bedarf hinzu geladen bzw. entfernt (z.B. Gerätetreiber, Dateisysteme, Netzwerkdienste)



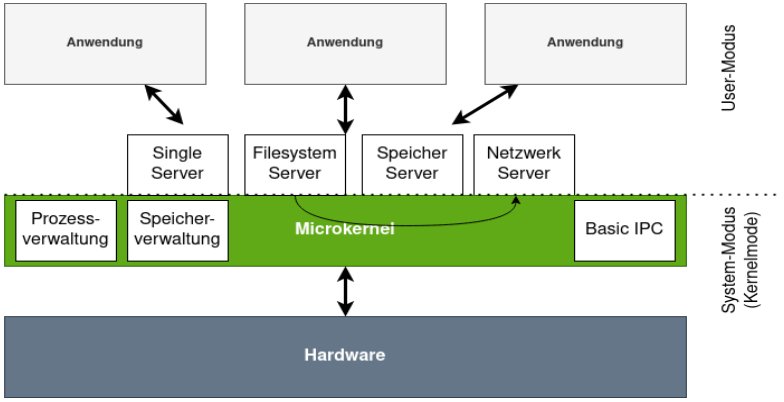
- Der Microsoft Windows NT Kernel wird häufig als “Hybrid-Kernel” bezeichnet
  Es befinden sich viele Funktionen (vor allem aus Performance-Gründen) noch direkt im Kernel, während einige Funktionen in den Benutzermodus ausgelagert werden. Es entsteht ein Kompromiss aus Makro- und Micro-Kernel

Strukturen: Monolithische Systeme: Bewertung

- **Isolation**
  - ?
- **Interaktionsmechanismen**
  - ?
- **Interruptbehandlung**
  - Behandlung durch IRQ-Handler des Kernel, direkter Aufruf der ISR
- **Erweiterbarkeit**
  - ?
- **Robustheit**
  - ?
- **Leistung**
  - Hoch, da wenig Kooperation notwendig ist und ein gemeinsamer Adressraum verwendet wird

Strukturen: Microkern-Systeme

- Aufspalten des Betriebssystems in kleinere Teile
  - Nur der Microkern selbst läuft im priviligierten Kern-Modus, während der Großteil der Funktionalität als gewöhnlicher Benutzerprozess, ausgeführt wird
  - Kommunikation der verschiedenen Komponenten des Mikrokern per *Inter Process Communication (IPC)*

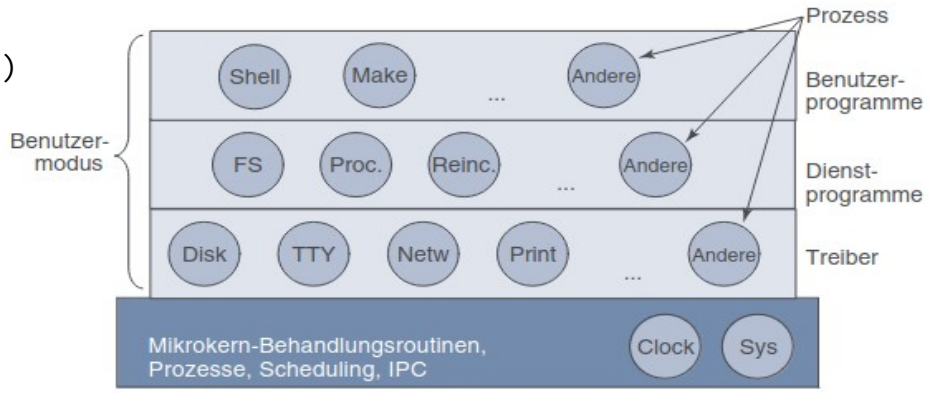


- Problem: Komplexes Zusammenspiel
  Die einzelnen Teilkomponenten des Betriebssystems (Speicherverwaltung, Netzerk, usw.) arbeiten unabhängig und kommunizieren miteinander. Dabei spielen die Effizienz und das Timing der Kommunikation eine wichtige Rolle.

- Vorteil: Hohe Ausfallsicherheit und Robustheit
  Alle arbeiten mit den minimalen Berechtigungen. Nur die nötigsten Komponenten laufen mit erhöhten Rechten im Kernel-Adressraum. Fehlerhafte Komponenten können neu gestartet werden.

Beispiel: Mikrokern-Systeme: MINIX-3 (und andere)

- MINIX-3 Mikrokernel
  - Kern selbst besteht aus ca. 12.000 Zeilen C-Code, 1.400 Zeilen Assembler-Code und stellt etwa 35 Kernel Systemaufrufe bereit (Komponente Sys in der Abbildung)
  - Alle weiteren Komponenten laufen im Benutzermodus in mehreren Schichten
  - Reincarnation-Server zum autonomen Ersetzen fehlerhafter Komponenten (durch Neustart)
  - Komplexe, mehrstufige Kontrollmechanismen um die Befugnisse von Prozessen zu begrenzen



Minix Mikrokernel (Tanenbaum, Abb. 1.26)

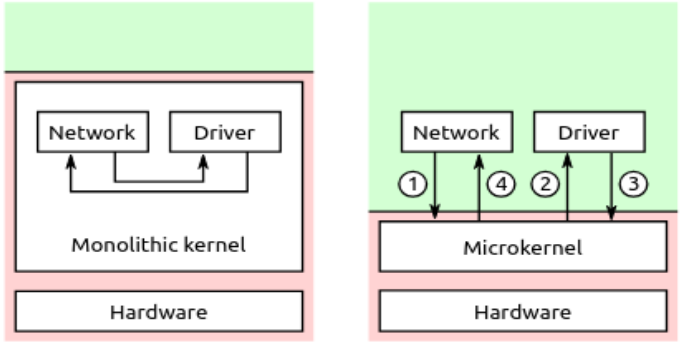
- Weitere Beispiele
  - Redox ist ein in Rust geschriebener Mikrokern (RedoxOS)
  - VMware ESXi basiert auf einem proprietären Mikrokern
  - Einige weitere sind QNX, Minix, L4, L4/Fiasco (TU Dresden), seL4 (formal verifiziert)

Strukturen: Mikrokern-Systeme: Bewertung

- **Isolation**
  - ?
- **Interaktionsmechanismen**
  - ?
- **Interruptbehandlung**
  - Interrupts werden in IPC-Nachrichten umgewandelt und zugestellt
- **Erweiterbarkeit**
  - ?
- **Robustheit**
  - ?
- **Leistung**
  - Stark abhängig von der IPC-Performance

Zusammenfassung

- Bootprozess und Sicherheitskonzept
- Interaktion mit dem Betriebssystem
  - Systemaufrufe (*system calls*)
  - Unterbrechungen (*interrupts*)
- Aufbau von Betriebssystemen
- Betriebssystemstrukturen
  - Monolithische Kerne (und Modulkonzept)
  - Mikrokerne (Spezialform Client-Server)
  - Mischform “Hybridkern” (Windows NT)
  - Virtualisierung
- Übung: Erstes C-Programm, Compiler, Systemaufruf



Makrokern vs. Mikrokern
(Bild: Nils Asmussen, TU Dresden)