Grundlagen der Betriebssysteme

Praktische Einführung mit Virtualisierung

Stefan Bosse

Universität Koblenz - FB Informatik

Einführung und Überblick

Was sind Betriebssysteme?

Einführung und Überblick

Was sind Betriebssysteme?

Was sind Virtuelle Maschinen?

Einführung und Überblick

Was sind Betriebssysteme?

Was sind Virtuelle Maschinen?

Wie sind Betriebssysteme aufgebaut, wie programmiert man sie, wie analysiert man Fehler?

Betriebsysteme

Was ist ein Betriebssystem?

Betriebsysteme

Was ist ein Betriebssystem?

Das Betriebssystem soll den Anwendungsprogrammierer bzw. den Anwender von Details der Rechnerhardware entlasten. Modern strukturierte Betriebssysteme kapseln den Zugriff auf die Betriebsmittel. Das Betriebssystem stellt somit eine virtuelle Maschine über der Hardware bereit.

Das Betriebssystem ist ein Programm mit Besonderheiten die in diesem Kurs untersucht werden. **Auch praktisch!**

Betriebsysteme

- Die geräteunabhängige Ein-/Ausgabe war eine der wichtigsten Errungenschaften bei der erstmaligen Einführung von Betriebssystemen.
- Früher war es notwendig, dass Applikationen die Eigenheiten der angeschlossenen Peripheriegeräte im Detail kennen mussten.

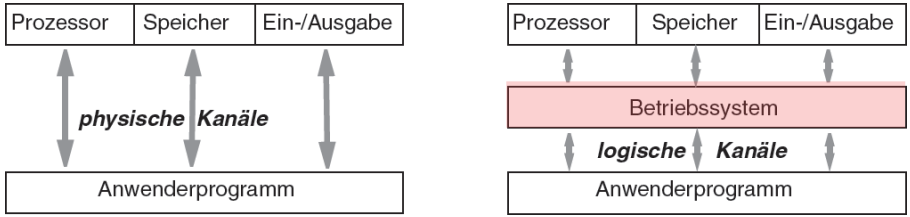


Abb. 1. Ein-/Ausgabe ohne und mit Betriebssystem

Lernziele

1. Sie erklären den Zweck und die Rolle eines modernen Betriebssystems.
2. Sie erkennen wie Rechnerressourcen durch Applikationen genutzt werden, wenn sie das Betriebssystem verwaltet.
3. Sie beschreiben die Funktionen eines aktuellen Betriebssystems in Bezug auf Benutzbarkeit, Effizienz und Entwicklungsfähigkeit.
4. Sie können die Vorteile abstrakter Schichten und ihrer Schnittstellen in hierarchisch gestalteten Architekturen erklären.
5. Sie analysieren die Kompromisse beim Entwurf eines Betriebssystems.
6. Sie erläutern die Architektureigenschaften monolithischer, geschichteter, modularer und Mikrokernsysteme.
7. Sie stellen netzwerkfähige, Client/Server- und verteilte Betriebssysteme einander gegenüber und vergleichen diese.
8. Sie erlernen die C Programmiersprache - Die Basis fast aller betriebssysteme.

Literatur

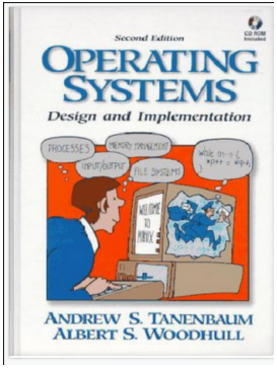
Grundkurs Betriebssysteme
Peter Mandel
Springer Verlag

Betriebssysteme
Eduard Glatz
dpunkt Verlag

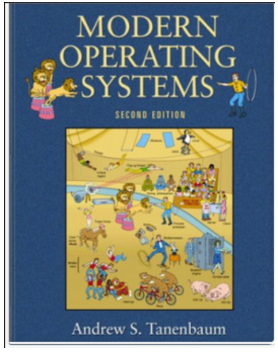
Betriebessysteme, kompakt
Christian Baun
Springer Verlag

Literatur

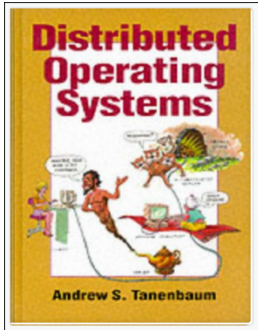
Operating Systems - Design And Implementation
Andrew S. Tanenbaum
Prentice Hall



Modern Operating Systems
Andrew S. Tanenbaum
Prentice Hall



Distributed Operating Systems
Andrew S. Tanenbaum
Prentice Hall, 1996



Taxonomie der Betriebssysteme

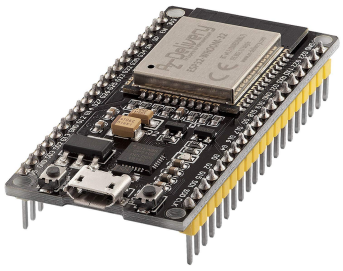
Klassen:

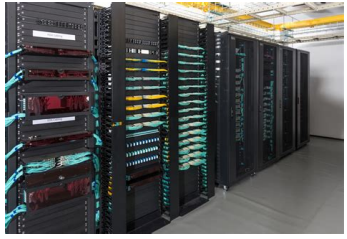
1. IO Loop: Kein Betriebssystem, eine einzige Endlosschleife und endless pain
2. Mikrokernel und Dienstprogramme
3. Monolithischer Kernel (eingebettet)
4. Monolithischer Kernel und Dienstprogramme
5. Verteilte Betriebssysteme (inkl. Cloud)
6. Echtzeitfähige Betriebssysteme
7. Generische versa applikationsspezifische (domänenspezifische) BS

Betriebssysteme und Rechnerarchitektur

Betriebssysteme sind eng verknüpft mit Rechnerarchitekturen!

Eingebettetes System	Server	Notebook





Programmierung von Betriebssystemen

Randbedingungen:

1. Rechnerarchitektur
2. Speicherarchitektur und Speichermodell
3. Nutzerinteraktion
4. Vernetzung und Netzwerkmodell
5. Aufgaben
6. Zeitmodell (Echtzeit)

Wir werden ein minimales Betriebssystem (*basekernel*) in C programmieren. C?

Programmierung von Betriebssystemen

Die Programmierung, der Test und die Fehlersuche in Betriebssystemen ist wenig komfortabel und methodisch getrieben wie die Informatik es im klassischen Softwareentwurf lehrt und ggfs. praktiziert!

Programmierung von Betriebssystemen

Die Programmierung, der Test und die Fehlersuche in Betriebssystemen ist wenig komfortabel und methodisch getrieben wie die Informatik es im klassischen Softwareentwurf lehrt und ggfs. praktiziert!

Welcher Debugger (Diagnosewerkzeug zur Fehlersuche) wird am häufigsten in der Praxis bei der Fehlersuche eingesetzt ?

Programmierung von Betriebssystemen

Die Programmierung, der Test und die Fehlersuche in Betriebssystemen ist wenig komfortabel und methodisch getrieben wie die Informatik es im klassischen Softwareentwurf lehrt und ggfs. praktiziert!

Welcher Debugger (Diagnosewerkzeug zur Fehlersuche) wird am häufigsten in der Praxis bei der Fehlersuche eingesetzt ?

Fehlersuche in Betriebssystemen ist wie Detektivarbeit (Sherlock Holmes, Hercule Poirot)!

Basekernel

https://github.com/dthain/basekernel

Prof. Douglas Thain (University of Notre Dame) et al.

1. Minimaler Betriebssystemkernel (monolithisch)
2. Betriebsprogramme (Shell Interpreter, inklusive minimaler grafischer Benutzeroberfläche)

Statistik der Programmiersprachen:

C	Assembler	Sonstiges
96.3%	3.0%	0.7%
Kernel: ≈13000 Zeilen	Kernel: ≈710 Zeilen	-
Library: ≈7200 Zeilen	-	-
User: ≈1500 Zeilen	-	-

Basekernel

1. Basekernel ist ein einfacher Betriebssystemkern für Forschung, Lehre und Spaß.

2. Der Basiskernel ist kein vollständiges Betriebssystem, aber er ist ein Ausgangspunkt für diejenigen, die neuen Betriebssystemcode studieren und entwickeln möchten.

3. Basekernel kann in einer Intel PC-kompatiblen virtuellen Maschine im geschützten 32-Bit-Modus gestartet werden, mit Unterstützung für VESA-Framebuffer-Grafiken, ATA-Festplatten, optische ATAPI-Geräte, Prozessverwaltung, Speicherschutz, einfache Grafiken und grundlegendes Dateisystem.

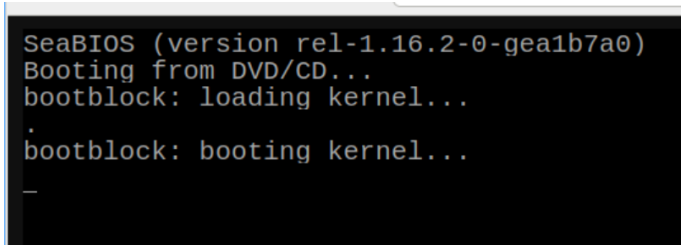
4. Es können einfach neue Programme auf Benutzerebene geschrieben und das System erweitert werden.

Basekernel: Das erste Experiment geht schief

Das Betriebssystem wird automatisch mittels des *make* Programms erzeugt (Dauer: 2 Sekunden!):

1. Der Kernel
2. Die Betriebsprogramme
3. Alles zusammen in eine ISO Datei gepackt.

Der erste Versuch das Betriebssystem (`baserkernel.iso`) in der *vm86* zu starten führt zu der Ausgabe (und 100% CPU Last):



Basekernel: Das erste Experiment geht schief. Was war passiert?

1. Ein Betriebssystem braucht selber ein kleines minimales Betriebssystem um (von einem Speichermedium) geladen und gestartet zu werden.
   - Hier wird der Betriebssystemkernel von einem Bootloader geladen
2. Der Bootloader wird selber wieder von einem Programm geladen
   - Hier wird der Bootloader vom im Rechner eingebauten Basic Input Output System (BIOS) geladen

Wir sehen an der Terminalausgabe dass der Bootloader geladen wurde, dieser das Kernelprogramm geladen hat, und es schließlich gestartet hat.

Dann war Schluss...

Basekernel: Das erste Experiment geht schief. Was war passiert?

Okay, in den Programmcode des Bootloaders schauen und her mit dem *printf* Debugger ...

- Programmiersprache des Bootloaders: Maschinensprache (Assembler)! OMG
- Printf? Gibt es nicht.
- Es gibt nur die Möglichkeit mit einer Reihe von Maschinenbefehlen des Mikroprozessors via BIOS Funktionen Textzeichen auf den Bildschirm zu setzen!

Basekernel: Das erste Experiment geht schief. Was tun?

```
bios_putstring:                # routine to print an entire string
    mov  (%si) , %al
    cmp  $0    , %al
    jz   bios_putstring_done
    call bios_putchar
    inc  %si
    jmp  bios_putstring
bios_putstring_done:
    ret
bios_putchar:                # routine to print a single char
    push %ax
    push %bx
    mov  $14   ,  %ah
    mov  $1    ,  %bl
    int  $0x10
    pop  %bx
    pop  %ax
    ret
bootmsg:
    .asciz "\r\nbootblock: booting kernel...\r\n"
****
    mov $(bootmsg),%si        # print boot message
    call bios_putstring
```

Code 1. Ausschnitt Minimalimplementierung "printf" in Assembler (`bootblock.S`)

Basekernel: Das erste Experiment geht schief. Was tun?

Wir fügen Checkpoint Ausgaben in den Bootloader (`bootblock.S`) ein um zu sehen wie weit wir kommen und später im Anfang des Betriebssystemkerns (`kernelcore.S`)

```
checkpoint_msg_a:
    .asciz "Checkpoint BL.1\r\n"
checkpoint_msg_b:
    .asciz "Checkpoint BL.2\r\n"
checkpoint_msg_c:
    .asciz "Checkpoint BL.3\r\n"

loadsector:
    mov  $(checkpoint_msg_a),%si        # Checkpoint A
    call bios_putstring

    mov  $1, %al            # load 1 sector
    mov  $0x02, %ah        # load command
    int  $0x13            # execute load
```

Code 2. Checkpoint Ausgaben in `bootblock.S` Bootloader

Basekernel: Das erste Experiment geht schief. Was tun?

```
.code16
.text
.global _start
_start:

.org KERNEL_SIZE_OFFSET
.global kernel_size
kernel_size:
    .long    _end-_start


bios_putstring:                # routine to print an entire string
***
bios_putchar:                # routine to print a single char
***
```

Code 3. Checkpoint Ausgaben in `kernelcore.S` Kernelstart

1. Kernel wieder kompilieren, ISO bauen
2. Kernel in VM starten
3. Ergebnis: Checkpoints `BL.1`, `BL.2`, `BL.3` werden erreicht, `KS.1` nicht!
4. Der Kernel wird überhaupt nicht gestartet (oder?)! Was ist falsch hier? Wir sind im Nirvana.

Basekernel: Das erste Experiment geht schief. Was tun?

5. Was passiert im Bootloader?
   - Er lädt den Kernel blockweise, ein Block sind hier 512 bytes, das Kernelimage (`basekernel.img`) ist aber rund 1 MB groß, d.h. mindestens 250 Blöcke müssten von dem virtuellen CDROM Laufwerk geladen werden.
   - Tatsächlich wird nur ein Block geladen!
6. Wieso? Woher weiß der Bootloader wie viele Blöcke er laden soll? Das steht am Anfang vom Kernelimage:

```
.org KERNEL_SIZE_OFFSET
.global kernel_size
kernel_size:
    .long _end-_start
```

7. Ist vielleicht die Angabe im Kernelimage falsch? Schauen wir mit einem Hex-Editor hinein:

Basekernel: Das erste Experiment geht schief. Was tun?

```
00000000   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
00000010   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
00000020   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
00000030   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
00000040   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
00000050   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
00000060   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
00000070   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
00000080   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
00000090   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
000000A0   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
000000B0   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
000000C0   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
000000D0   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
000000E0   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
```

Code 4. Hex Editor Dump vom Beginn des Kernelimages. Lauter Nullen!?

8. Nur Nullen im Image - bis - halt stopp, da war doch etwas mit dem Kernelimage und dessen seltsame virtuelle Größe!

9. Wir schauen uns nun die Symboltabelle, d.h. die Adresstabelle des Kernels an ...

Basekernel: Das erste Experiment geht schief. Was tun?

```
>> objdump -t kernel.elf
kernel.elf:     file format elf32-i386

SYMBOL TABLE:
080480f4 l    d  .note.gnu.property     00000000 .note.gnu.property
00010000 l    d  .text                  00000000 .text
0001e000 l    d  .rodata                00000000 .rodata
0001f1f0 l    d  .eh_frame              00000000 .eh_frame
00024000 l    d  .data                  00000000 .data
00024b80 l    d  .bss                   00000000 .bss
00000000 l    d  .comment               00000000 .comment
```

Code 5. Auszug von `objdump -t kernel.elf` (Symbole und ihre Adressen)

Basekernel: Das erste Experiment geht schief. Was tun?

```
>> objdump -t kernel.elf
kernel.elf:     file format elf32-i386

SYMBOL TABLE:
080480f4 l    d  .note.gnu.property     00000000 .note.gnu.property
00010000 l    d  .text                  00000000 .text
0001e000 l    d  .rodata                00000000 .rodata
0001f1f0 l    d  .eh_frame              00000000 .eh_frame
00024000 l    d  .data                  00000000 .data
00024b80 l    d  .bss                   00000000 .bss
00000000 l    d  .comment               00000000 .comment
```

Code 6. Auszug von `objdump -t kernel.elf` (Symbole und ihre Adressen)

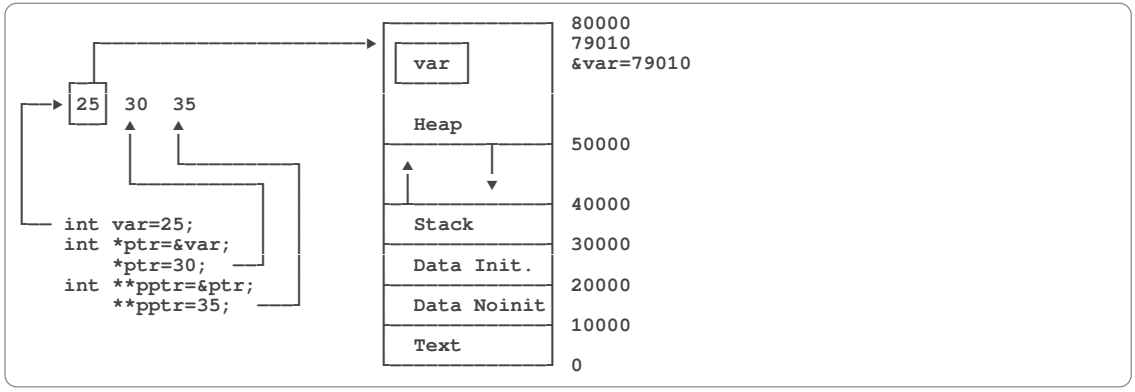
10. Das vom Linker automatisch eingefügte und nicht benötigte Symbol `.note.gnu.property` wurde mit in das Image übernommen, hatte aber unabänderlich eine völlig andere Adresse (0x8000000) als die Kernelsymbole (0x10000). Das führte zu einem großen "Speicherloch" im Image und einem völlig falschen Aufbau mit Nullen gefüllt!
11. Lösung 1: `objcopy -R .note.gnu.property`
12. Lösung 2: Das Kleingedruckte lesen und den Hinweis beachten vorher einen Crosscompiler zu bauen und zu benutzen (der diesen Mist nicht macht).

Programmiersprache C

Warum C?

1. Wir haben an der beispielhaften Fehlersuche am häufigsten Speicher und Speicheradressen gesehen.
2. Die Programmiersprache bildet das lineare Speichermodell direkt ab (fast alles sind Speicheradressen)
3. Aber trotzdem sind C Programme (Quelltext) nicht auf einen Rechner oder eine Rechnerarchitektur beschränkt
4. Portabilität der Programme!
5. Besser schreib- und lesbar als Assembler (nicht portabel)
6. C ist eine Hochsprache mit statischer Typisierung, aber expliziten Speichermanagement (durch den Programmierer) und direkter Sichtbarkeit des Speichers und Speicheradressenoperationen (Pointer)
7. Aber trotzdem ist maschinennahe Programmierung (also die Programmierung der Maschine) möglich, gerade auch Gerätetreiber und Ein-und Ausgabe

Programmiersprache C



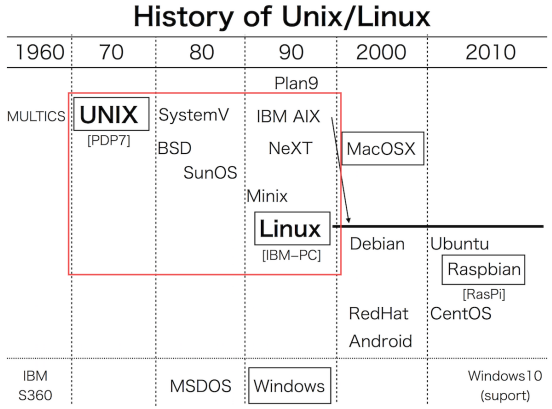
Code 7. Speicherlayout in C / Pointervariablen

Programmiersprache C

C History

1960	ALGOL	International Group
1967	BCPL	Martin Richards
1970	B	Ken Thompson
1972	Traditional C	Dennis Ritchie
1978	K&R C	Kernighan and Ritchie
1983	ANSI C	ANSI Committee
1990	ANSI / ISO C	ISO Committee
1999	C99	Standardization Committee

UNIX History



Programmiersprache C: C2JS

- Statisch kompiliert, nicht inkrementell (wie JavaScript, Python)
- Aber mit dem C2JS Transpiler ...

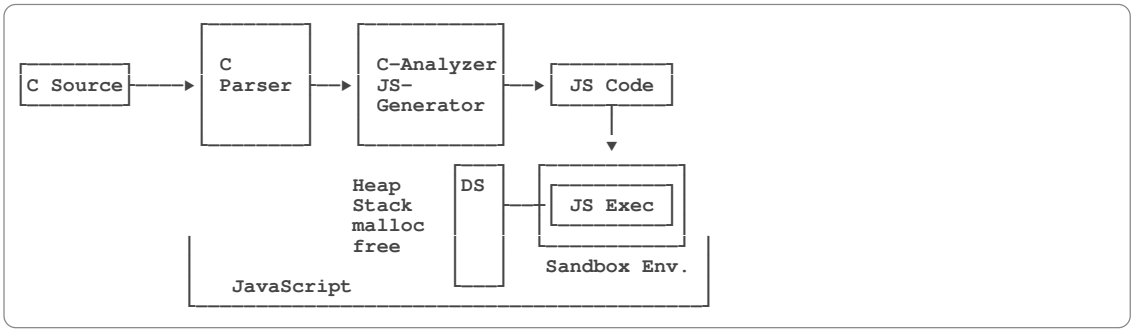
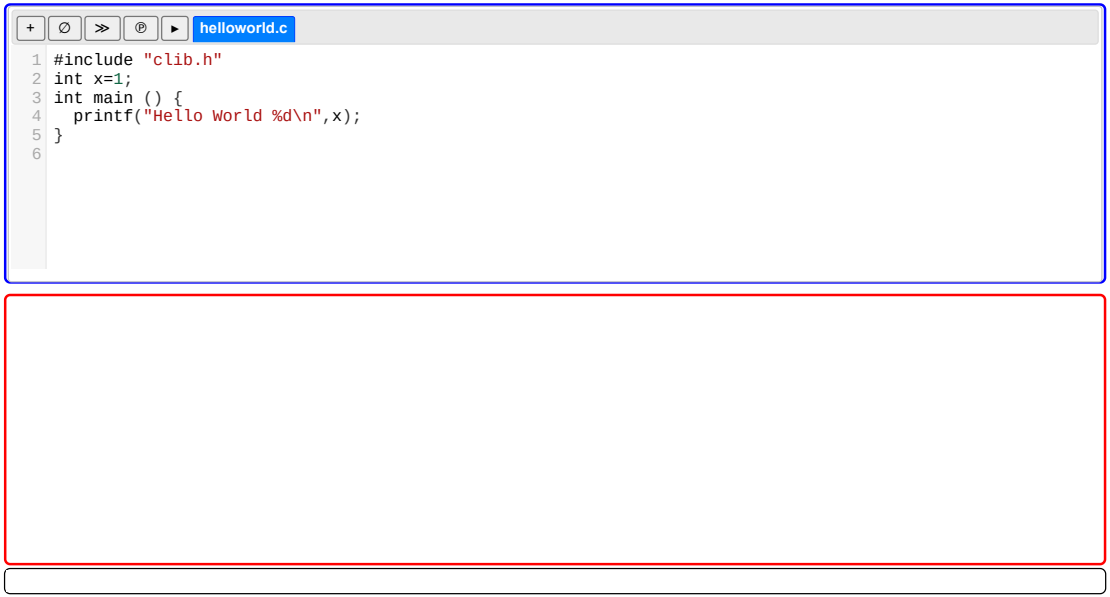


Abb. 2. C2JS Architektur

Programmiersprache C: Live Programming mit C2JS



Programmiersprache C: Live Programming mit C2JS

```
#include "clib.h"
int x=1;
int main () {
  printf("Hello World %d\n",x);
}
```

```
Object.assign(CS,{"-1":"free","-2":"malloc","-3":"memcpy","-4":"printf","-5":"main"});
Object.assign(CSI,{"free":-1,"malloc":-2,"memcpy":-3,"printf":-4,"main":-5});
var x=x||__heap_allocate(1,4,"int");
DS.writeInt32(1,x);
function main() {
  var __sp=__stack_pointer();
  printf("Hello World %d\n",DS.readInt32(x));
  __stack_pointer(__sp);
};
main(argc,argv)
```

Bsp. 1. (Oben) C Code (Unten) Transpillierte JS Code

Betriebssysteme - Eine Einführung

1. Alle Einheiten die für einen Rechnerbetrieb nötig sind, sind **Ressourcen** oder **Betriebsmittel**
2. Ein Betriebssystem ist die **Gesamtheit der Programmteile**, die die Benutzung von Betriebsmitteln steuern und verwalten.
3. Das Betriebssystem ist die Software (Programmteile), die für den Betrieb eines Rechners anwendungsunabhängig (unterstützend) notwendig ist.

Def. 1. Formale Definition Betriebssystem

- Betriebssysteme sind also Programme und Software. Betriebssystementwicklung ist Softwareentwurf.
- Im vorherigen Praxisbeispiel wurden schon einige Unterschiede zum klassischen Softwarenentwurf deutlich:
  - Zugriff auf Ein- und Ausgabegeräte (Betriebsmittel)
  - Direkte Maschinenprogrammierung
  - Speichermodell und Rechnerarchitektur

Betriebssysteme - Eine Einführung

	Zentrale Ressourcen	Periphere Ressourcen
Aktive Ressourcen	Prozessor(en)	Kommunikationseinheiten: 1. Endgeräte (Tastaturen, Drucker, Anzeigen, Zeigegeräte etc.) 2. Netzwerk (entfernt, lokal) etc.
Passive Ressourcen	Hauptspeicher	Speichereinheiten: 1. Festkörper SDD 2. Platten HDD 3. Bänder Tape 4. CD-ROM/DVD etc.

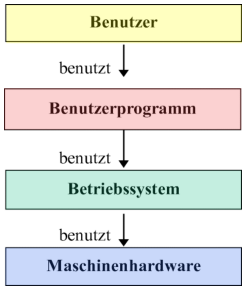
Tab. 1. Ressourcenklassen

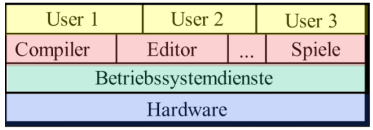
Betriebssysteme - Eine Einführung

Es gibt nicht das Betriebssystem schlechthin, sondern nur eine den Forderungen der Anwenderprogramme entsprechende Unterstützung, die von der benutzerdefinierten Konfiguration abhängig ist und sich im Laufe der Zeit stark gewandelt hat. Gehörte früher nur die Prozessor-, Speicher- und Ein-/Ausgabeverwaltung zum Betriebssystem, so werden heute auch eine grafische Benutzeroberfläche mit verschiedenen Schriftarten und -größen (Fonts) sowie Netzwerkfunktionen verlangt.

Betriebssysteme - Das Schichtenmodell (1)

[978-3-96088-839-0]





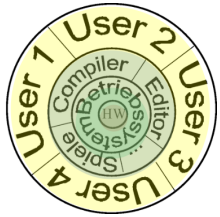


Abb. 3. Das Nutzerschichtenmodell von Betriebssystemen

Betriebssysteme - Programme

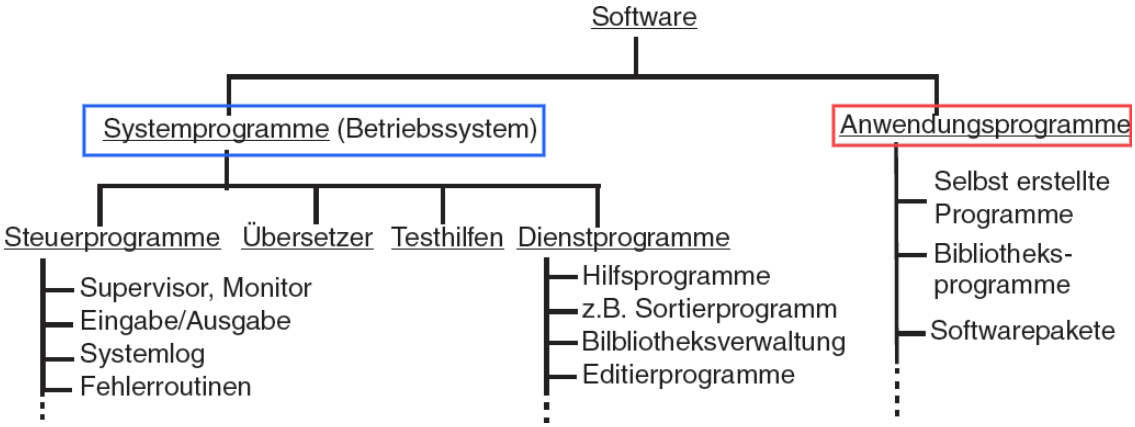


Abb. 4. Softwaregliederung

Betriebssysteme - Programme

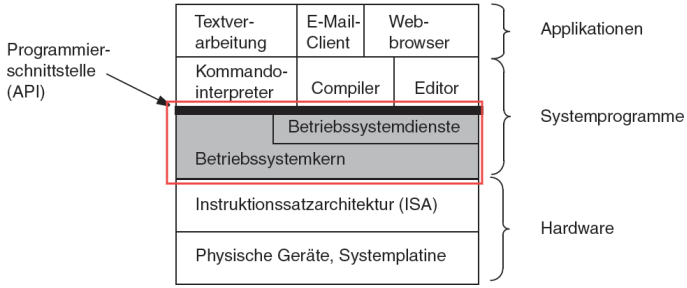
Die eigentlichen **Steuerprogramme** sind für folgende Funktionen zuständig:

- Steuerung aller Computerfunktionen und Koordination der verschiedenen zu aktivierenden Programme.
- Steuerung der Ein-/Ausgabeoperationen für die Anwendungsprogramme.
- Überwachung und Registrierung der auf dem Computersystem ablaufenden Aktivitäten.
- Ermittlung und Korrektur von Systemfehlern.

Auffallend bei dieser Definition ist die Einbeziehung von Übersetzern (Compiler, Binder), Testhilfen und Dienstprogrammen.

Für klassische Betriebssysteme (z.B. Unix und GNU-Tools) trifft dies vollumfänglich zu, während moderne Betriebssysteme oft die Bereitstellung von Übersetzungstools irgendwelchen Drittherstellern überlassen bzw. diese als separate Applikation ausliefern (z.B. Windows und Visual Studio).

Einordnung im Computersystem



[bsdp]

Abb. 5. In einem Rechner stellt das Betriebssystem eine Softwareschicht dar, die zwischen den Benutzerapplikationen einerseits und der Rechnerhardware andererseits liegt.

- Programme zur Softwareentwicklung, wie Editoren und Compiler, können dazugehören.
- Häufig wird nur der Betriebssystemkern als Betriebssystem bezeichnet, während der Begriff Systemprogramme für das Gesamtpaket inklusive der Programmentwicklungswerkzeuge benutzt wird.

Abstraktion von Hardwareelementen

- Prozessor
- Arbeitsspeicher (main memory)
- Massenspeicher (mass storage), z.B. Festplatten, CD-ROM, DVD
- Benutzerschnittstelle (user interface)
- Kommunikations- und andere Peripheriegeräte (LAN, WLAN usw.)

Die Betriebssystemtheorie beruht damit auf den Prinzipien der Computertechnik.
Computertechnik befasst sich mit:

- Rechner-Grundmodellen (Von-Neumann-, Harvard-Architektur)
- Funktionsweise des Prozessors (Instruktionssatz, Registeraufbau)
- Speichern und ihren Realisierungen (Primär- und Sekundärspeicher)
- Peripheriegeräten (Tastatur, Bildschirm, Schnittstellenbausteine usw.)

Betriebssystemarten

Wir unterscheiden klassisch:

Stapelverarbeitung (batch processing)

Typisches Merkmal ist, dass Programme angestoßen werden, aber ansonsten keine nennenswerte Benutzerinteraktion stattfindet. Die auszuführenden Befehle sind stattdessen in einer Stapeldatei abgelegt, deren Inhalt fortlaufend interpretiert wird. Klassische Großrechnerbetriebssysteme werden auf diese Art und Weise genutzt, z.B. zur Ausführung von Buchhaltungsprogrammen über Nacht.

Time-Sharing-Betrieb

Die zur Verfügung stehende Rechenleistung wird in Form von Zeitscheiben (time slices, time shares) auf die einzelnen Benutzer aufgeteilt mit dem Ziel, dass jeder Benutzer scheinbar den Rechner für sich alleine zur Verfügung hat. Historisch gesehen sind Time-Sharing-Systeme die Nachfolger bzw. Ergänzung der Batch-Systeme mit der Neuerung, dass sie Benutzer interaktiv arbeiten lassen (Dialogbetrieb).

Betriebssystemarten

Echtzeitbetrieb

Die Rechenleistung wird auf mehrere Benutzer oder zumindest Prozesse aufgeteilt, wobei zeitliche Randbedingungen beachtet werden. Oft sind Echtzeitsysteme reaktive Systeme, indem sie auf gewisse Signale aus der Umgebung (Interrupts, Meldungen) möglichst rasch reagieren.

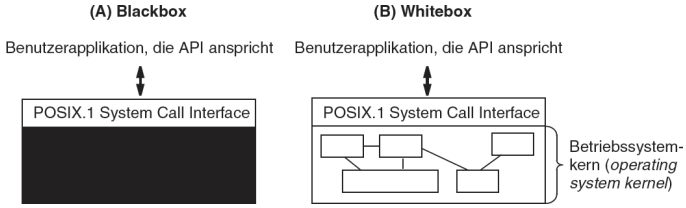
Weitere Unterteilung (modern) von Betriebssystemen nach unterstützter **Rechnerstruktur**:

1. **Einprozessorsysteme**
2. **Multiprozessorsysteme**
3. **Verteiltes System**

Betriebssystemmodell

Es gibt zwei Arten von Betriebssystemmodellen:

1. Blackbox
2. Whitebox



[bsdp]

Abb. 6. Black- und Whitebox-Betrachtung (Beispiel: Unix)

- Solange es lediglich um die Systemprogrammierung geht, ist eine Blackbox-Betrachtung des Betriebssystems ausreichend.
  - Nach außen ist damit nur die Programmierschnittstelle sichtbar, jedoch nicht das Systeminnere
  - Dies entspricht einem klassischen Ideal des Software Engineering, das aussagt, dass die Schnittstelle das Maß aller Dinge ist und die Implementierung dahinter beliebig austauschbar sein soll.
- Aber: Hier ist interessant, unter die »Motorraumhaube« eines Betriebssystems zu gucken (Whitebox).

Entwurf von Betriebssystemen

Anforderungen

- Fehlerfreiheit des Codes: z.B. durch minimale Komplexität des Quellcodes
- Einfache Operationen (auf API und für alle Schnittstellen)
- Erweiterbarkeit (extensibility)
- Skalierbarkeit (scalability)
- Orthogonalität: Operationen wirken gleich auf verschiedenartigen Objekten
- Robuste Betriebsumgebung (»crash-proof«, »reliable«)
- Einhaltung der Sicherheitsziele (mehrere Anforderungsstufen denkbar)
- Portabilität (Unterstützung verschiedenartiger Plattformen)
- Echtzeitfähigkeit (z.B. für Multimedia-Anwendungen)
- Effizienz (schnelle Diensterbringung, minimaler Ressourcenbedarf)
- Weiterentwickelbarkeit: Trennung von Strategie (policy) und Mechanismus (mechanism)

Betriebsmodus

Betriebssystemeprogramme sind wie Benutzeprogramme Programme aber mit verschiedenen Rechten und Rollen.

Diese Rechte und Rollen benötigen Hardwareunterstützung

- **Mikrokontroller** sind einfache Mikroprozessoren (eingebettete Systeme) und bieten kaum oder gar keine Unterstützung für ein Betriebssystem. Sie bündeln Prozessor, Speicher, und Peripherie 8auf einem Chip)
  - Beispiele sind ESP32, ARM Cortex M0 STM32, Atmel ATMega (aktuell).
- **Einfache Universalmikroprozessoren** bieten nicht viel mehr, nur dass hier keine Peripheriegeräte integriert sind.
  - Beispiels sind Motorolla 6800, 68000, Intel 8086 (ca. 1990)
- **Moderne leistungsfähige Universalmikroprozessoren** bieten verscheiden Mechanismen um Betriebssysteme zu unterstützen.
  - Beispiele sind ARM v6/7/8, Intel 80386/Pentium usw., Sun Microsystems UltraSPARC usw.
  - MMU (Memory Management Unit) und Mechanis-men für einen privilegierten Betriebsmodus für die Systemsoftware (Privilegiensystem).

Betriebsmodus und Schutz

- Bei Universalmikroprozessoren werden durch das Privilegiensystem kritische Operationen und Zugriffe geschützt, damit ein Programmierfehler in einem Anwendungsprogramm nicht das ganze Computersystem beeinträchtigt.
- Insbesondere bei Multitasking-Anwendungen und Multiuser-Betrieb (mehrere gleichzeitige Benutzer) ist ein solcher Schutz erforderlich!
- So wird in den meisten Betriebssystemen der Zugriff auf Hardwareteile mittels dieser Schutzfunktionen dem normalen Anwender (bzw. Benutzerapplikationen) verwehrt.
  - Dazu dienen unterschiedliche CPU-Betriebsarten

Minimales Zweiebenen Modussystem:

1. **Kernmodus** (kernel mode, supervisor mode) mit »allen Rechten« für Betriebssystemcode
2. **Benutzermodus** (user mode) mit »eingeschränkten Rechten« für Applikationscode

Betriebsmodus und Schutz

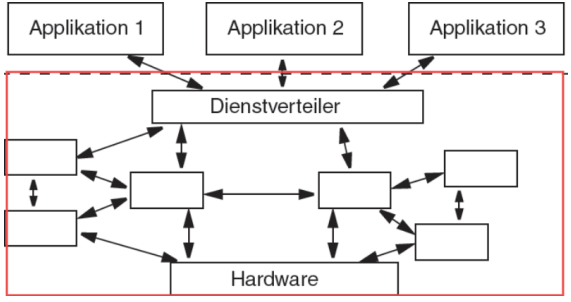
	Benutzermodus	Kernmodus
Ausführbare Maschinenbefehle	Begrenzte Auswahl	Alle
Hardwarezugriff	Nein bzw. nur mithilfe des Betriebssystems	Ja, Vollzugriff
Zugriff auf Systemcode bzw. Daten	Keiner bzw. nur lesend	Exklusiv

Tab. 2. Vergleich zwischen Benutzer- und Kernmodus

Im Idealfall werden Schutzmechanismen auch für die Abschottung verschiedener Systemteile untereinander eingesetzt. Die Grenze des Sinnvollen ist allerdings darin zu sehen, dass eine teilweise lahmgelegte Systemsoftware aus Sicht des Anwenders oft nicht besser ist als ein Totalabsturz.

Monolithische Systeme

Die Struktur dieser Systeme besteht darin, dass sie keine oder nur eine unklare Struktur haben.



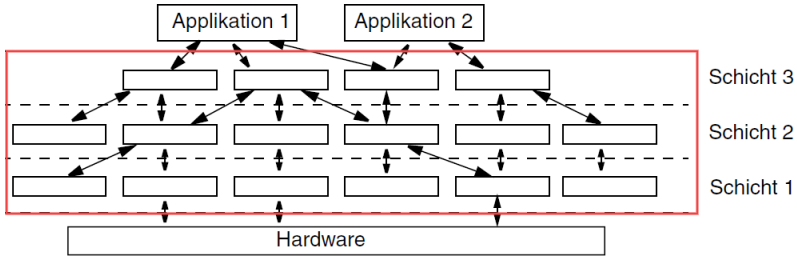
[bsdp]

Abb. 7. Beispiel für eine monolithische Betriebssystemstruktur

- Meist handelt es sich um evolutionär gewachsene Betriebssysteme, bei denen es anfänglich unwichtig war, einzelne Teilfunktionen klar mit Schnittstellen voneinander abzugrenzen.

Schichtsysteme

Bei dieser Strukturierungsform sind die Betriebssystemfunktionen in viele Teilfunktionen gegliedert, die hierarchisch auf mehrere Schichten verteilt sind.



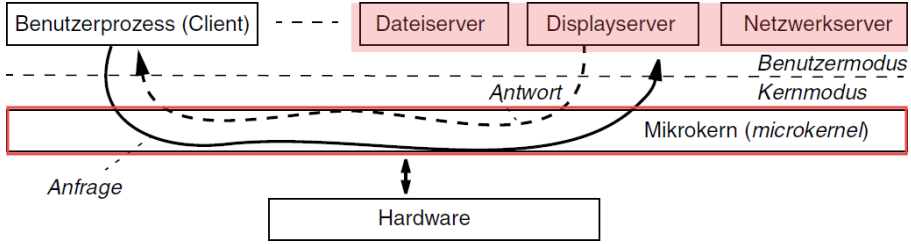
[bsdp]

Abb. 8. Beispiel einer geschichteten Struktur

- Die Funktionen einer höheren Schicht bauen strikt nur auf Funktionen einer tieferen Schicht auf.
- Die erste Schicht könnte eine Hardware-Abstraktionsschicht sein.
- Die Festlegung der Schichtenstruktur folgt keiner Standardstruktur.

Mikrokernelsysteme

Nur die allerzentralsten Funktionen sind in einem (kleinen) Kernteil zusammengefasst, alle übrigen Funktionen sind als Serverdienste separat realisiert. Nachrichtenbasierte Kommunikation!



[bsdp]

Abb. 9. Mikrokern nach dem Client/Server-Prinzip

- Serverdienste können z.B. Dateidienste, Verzeichnisdienste sein

Mikrokernelsysteme

Der Mikrokern enthält lediglich die vier Basisdienste:

1. Nachrichtenübermittlung (message passing)
2. Speicherverwaltung (virtual memory)
3. Prozessorverwaltung (scheduling)
4. Gerätetreiber (device drivers).

- Da ein Großteil des Betriebssystemcodes auf die Benutzerebene (Benutzermodus) verschoben wird, sind überschaubare zentrale Teile des Systems durch den Kernmodus gegen fehlerhafte Manipulationen geschützt.
- Ebenso ist es nur dem eigentlichen Kern erlaubt, auf die Hardware zuzugreifen.
- Beansprucht ein Benutzerprozess einen Systemdienst, so wird die Anforderung als Nachricht durch den Mikrokern an den zuständigen Serverprozess weitergeleitet.
- Entsprechend transportiert der Mikrokern auch die Antwort an den anfordernden Prozess zurück

Multiprozessorsysteme

Eine weitere Abstraktion und Schnittstellen sollten Betriebssysteme bei Multiprozessorrechnern anbieten.

- Das Verarbeiten von Prozessen auf mehreren Prozessoren ist im Detail nicht so einfach und birgt eine Reihe von Problemen, die typische für Multiprozessverareitung sind.

Multiprozessorsysteme haben durch die Einführung von Multicore-CPUs eine große Verbreitung erfahren. Dabei teilen sich typischerweise alle Rechenkerne die Peripherie und den Hauptspeicher, weswegen man sie Shared-Memory-Multiprozessoren nennt.

Multiprozessorsysteme

Drei Betriebssystemtypen sind für diese Rechner denkbar:

1. **Monosystem**: Jede CPU hat ihr eigenes Betriebssystem: Der Speicher wird in Partitionen (pro CPU/Betriebssystem) aufgeteilt.
2. **Asymmetrische Multiprozessoren** (asymmetric multiprocessing, AMP): Das Betriebssystem läuft nur auf einer einzigen CPU (=Master), die Anwendungsausführung nutzt alle restlichen Prozessoren (=Slaves).
3. **Symmetrische Multiprozessoren** (symmetric multiprocessing, SMP): Nur eine einzige Kopie des Betriebssystems liegt im Speicher. Diese ist von jeder CPU ausführbar.

Monosystem

- Am einfachsten ist die Lösung, dass jede CPU ihr eigenes Betriebssystem hat und der Speicher partitionsweise den einzelnen CPUs zugeteilt wird, womit die einzelnen Prozessoren unabhängig voneinander arbeiten.
  - Der Code des Betriebssystems ist nur einmal im Speicher abgelegt, da unveränderlich.

Infolge des fehlenden Lastausgleichs und der fixen Speicheraufteilung **skaliert diese Lösung schlecht** und hat deswegen keine nennenswerte Verbreitung.

[bsdp]

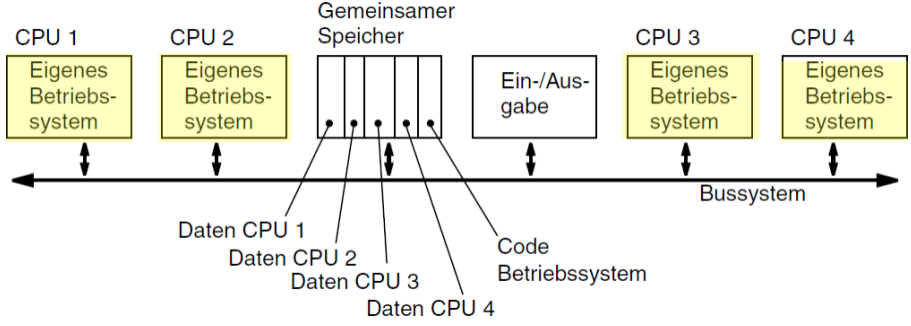


Abb. 10. Jede CPU hat ein eigenes Betriebssystem.

Asymmetrische Multiprozessoren

- Die Variante der asymmetrischen Multiprozessoren weist alle Betriebssystemaktivität einer bestimmten CPU zu, die damit der Chef (Master) wird.
- Die Anwenderprozesse laufen auf den restlichen CPUs, die man Slaves nennt .

Vorteilhaft ist die Möglichkeit der flexiblen Zuteilung ablaufwilliger Prozesse an die einzelnen Slaves. Der wesentliche Nachteil ist aber der Flaschenhals, der durch den Master entsteht, da alle Systemaufrufe nur von dieser CPU bearbeitet werden.

Asymmetrische Multiprozessoren

- Es entstehen infolge des Flaschenhalses (Master) bereits bei einem System mit 8 Prozessoren signifikante Wartezeiten.
- Es handelt sich also um eine Lösung, die nur für eine kleine Prozessoranzahl sinnvoll ist.
- **Die Skalierung ist also mäßig**, die Auslastung typisch (N-1)/N% (gewünscht ist 100%. bei N=4 dann nur 75%)

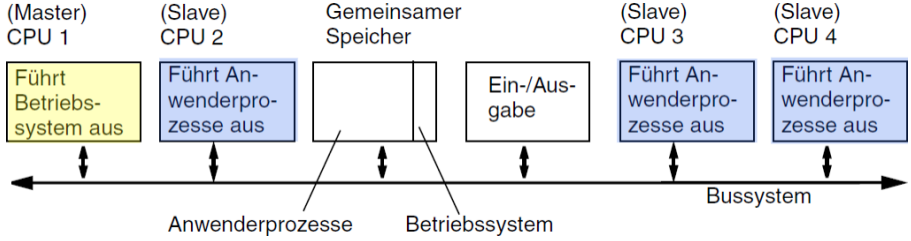


Abb. 11. Asymmetrisches (Master/Slave-)Multiprozessorsystem

Symmetrische Multiprozessoren

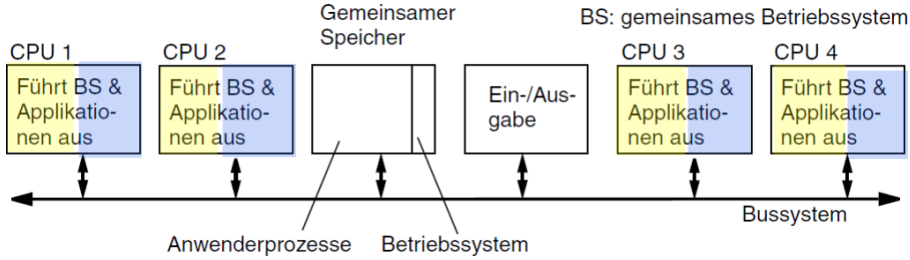
- Bei symmetrischen Multiprozessoren ist das Betriebssystem genau einmal im Speicher, und zwar sowohl der Code als auch die Daten
- Systemaufrufe können von allen CPUs ausgeführt werden. Da die Daten des Betriebssystems für alle gemeinsam zugreifbar sind, entfällt damit der Flaschenhals der Master-CPU.

Allerdings stellt sich damit auch das Problem des koordinierten Zugriffs auf die Systemdaten, um Dateninkonsistenzen zu vermeiden (kritische Bereiche).

- Der einfachste Weg wäre der, dass zu jedem Zeitpunkt nur eine einzige CPU einen Systemaufruf ausführen darf (Spinlock).
  - Damit wäre das Flaschenhalsproblem aber in einer neuen Form wieder vorhanden und die Leistung limitiert.
- In der Praxis genügt es aber, wenn die einzelnen Systemtabellen separat abgesichert werden.
  - Systemaufrufe auf unterschiedlichen Tabellen können dann parallel stattfinden.

Symmetrische Multiprozessoren

- Da viele Systemtabellen in verschiedener Beziehung voneinander abhängen, ist die Realisierung eines derartigen Betriebssystems sehr anspruchsvoll (z.B. Deadlock-Problematik).



[bsdp]

Abb. 12. Symmetrisches Multiprozessor-(SMP-)System

Verteilte Betriebssysteme

Parallels System
Starke Kopplung der Prozessoren, geteilter Speicher

Verteiltes System
Lose Kopplung der Prozessoren, privater Speicher

- Verteilte Betriebssysteme nutzen eine Menge von über ein Netzwerk verbundenen Rechnern zur Lösung größerer Aufgaben und um eine gemeinsame Rechenplattform in größerem Rahmen zu realisieren.
- Idealerweise wird das Betriebssystem so realisiert, dass Ortstransparenz herrscht.
  - Dies bedeutet, dass der Benutzer nur ein einziges System sieht (Single System Image, SSI), egal an welchem der teilnehmenden Rechner er sich momentan angemeldet hat.

Verteilte Betriebssysteme

- Solche Betriebssysteme realisieren die rechnerübergreifende Kommunikation systemintern und unterstützen Mechanismen zum Lastausgleich zwischen den einzelnen Rechnern wie auch Ausfallredundanz.

Clustersysteme hingegen unterstützen die Ortstransparenz nur partiell oder gar nicht, womit sie partiell Fähigkeiten verteilter Betriebssysteme besitzen.

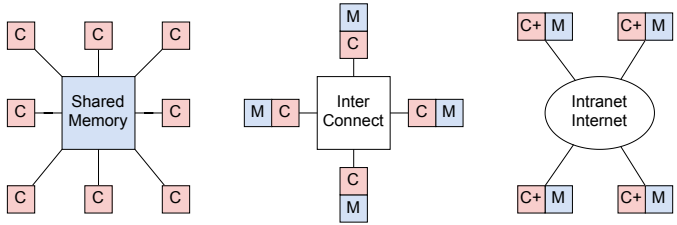


Abb. 13. Von parallelen zu verteilten Betriebssystemen und Rechnern

Netzwerk Betriebssysteme

- Nicht ganz ein verteiltes System
  - Es gibt lokalisierte und isolierte Betriebssysteme
  - Eine Software Applikation oder Mittelebenensoftware verbindet die Rechner

[https://www.slideserve.com/loc/distributed-operating-systems]

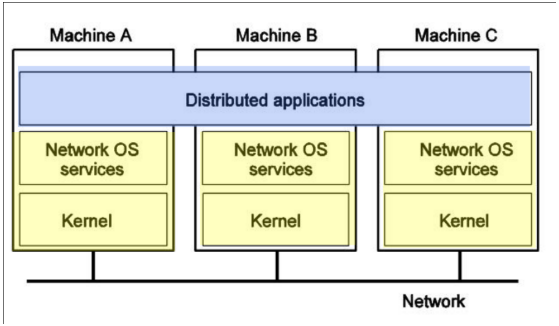


Abb. 14. NOS: Nur die Applikation ist verteilt

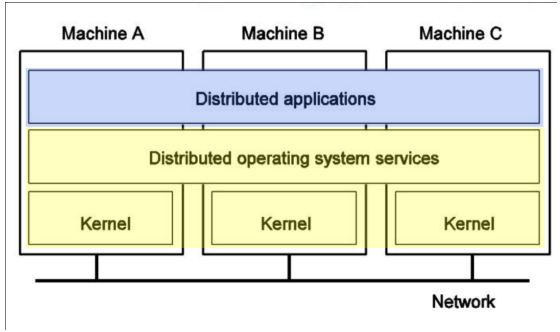


Abb. 15. DOS: Betriebssystem und Applikation sind verteilt

Das WWW als Betriebssystem?

Durch die Einführung von "Middleware" Software können heterogene Betriebssystemlandschaften vernetzt werden und quasi verteilte Applikationen ermöglichen.

[https://www.slideserve.com/loc/distributed-operating-systems]

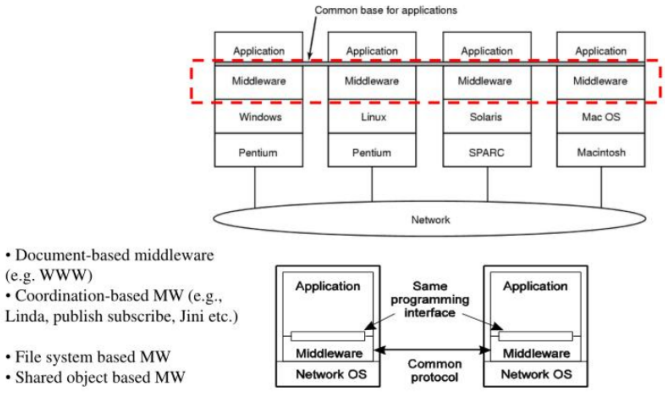


Abb. 16. WWW als verteilte Anwendung durch HTTP und Web Browser (+ JavaScript + REST APIs und Web Server)

Taxonomie der Betriebssysteme

[bspm]

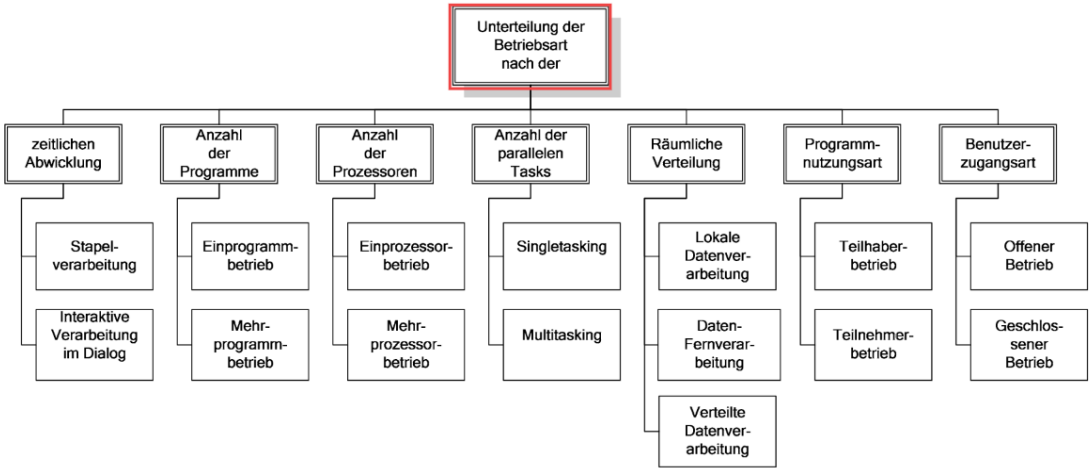


Abb. 17. Unterteilung nach Betriebsarten ergibt eine Terminologie

Single- und Multitasking

Die Begriffe Singletasking und Multitasking sind eng verwandt mit den Begriffen Einprogramm- und Mehrprogrammbetrieb.

Singletasking

Im Singletasking-Betrieb ist nur ein Programm aktiv, das sämtliche Betriebsmittel des Systems nutzen kann. Die alten PC-Betriebssysteme wie MS-DOS unterstützen z.B. nur Singletasking.

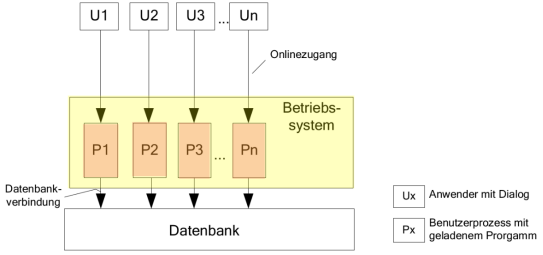
Multitasking

Im Multitasking-Betrieb können mehrere Programme nebenläufig ausgeführt werden. Die erforderlichen Betriebsmittel werden nach verschiedenen Strategien (Prioritäten, Zeitscheibenverfahren) zuge-teilt. Die Zuordnung des Prozessors nach Zeitintervallen an die nebenläufigen Programme wird als Timesharing bezeichnet.

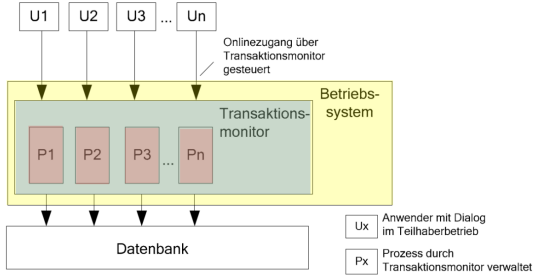
Terminal-, Teilhaber- versus Teilnehmerbetrieb

Im Hinblick auf die Programmnutzungsart unterscheidet man **Teilhaber- und Teilnehmerbetrieb**, wobei der Teilhaberbetrieb in der Regel eine Systemsoftware mit der Bezeichnung Transaktionsmonitor benötigt. Die Unterscheidung ist auf den ersten Blick nicht ganz einleuchtend und soll daher erläutert werden.

Teilnehmerbetrieb



Teilhaberbetrieb



Transaktionen und Transaktionsmonitore

Transaktion

Unter einer Transaktion wird hierbei ein atomar auszuführender Service verstanden, der entweder ganz oder gar nicht ausgeführt wird. Werden innerhalb der Ausführung einer Transaktion mehrere Operationen etwa auf einer Datenbank erforderlich, so darf dies nur in einem Stück erfolgen.

Teilnehmerbetrieb

- Im Teilnehmerbetrieb erhält jeder Anwender seinen eigenen Benutzerprozess sowie weitere Betriebsmittel vom Betriebssystem zugeteilt.
  - Der Benutzer meldet sich über einen Login-Dialog beim System an und bekommt die Betriebsmittel dediziert zugeordnet.
  - Dies ist auch heute noch die übliche Vorgehensweise in modernen Betriebssystemen. Unter Unix und Windows arbeitet man üblicher- weise nach dem Login-Vorgang im Teilnehmerbetrieb.

Teilhaberbetrieb

- Im Teilhaberbetrieb werden Prozesse und Betriebsmittel über einen **Transaktionsmonitor** zugeteilt.
  - Dies ist möglich, da die Benutzer aus Sicht des Betriebssystems die meiste Zeit „denken“, bevor eine Eingabe in das System erfolgt.
  - Die Betriebsart Teilhaberbetrieb ist ideal für dialogorientierte Programme mit vielen parallel arbeitenden Anwendern, die meistens kurze und schnelle Transaktionen ausführen, wie dies etwa bei einem Buchungssystem der Fall ist.

Terminalbetrieb

[bspm]

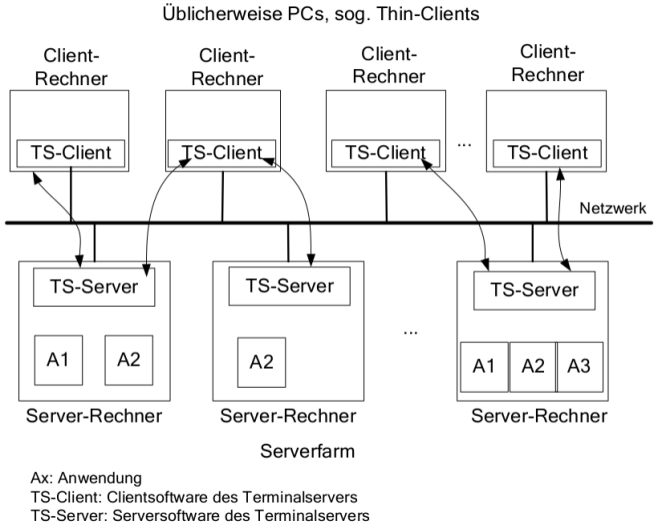


Abb. 18. Typischer Terminalservereinsatz

Terminalbetrieb

- Ein Terminalserver verwaltet Terminals bzw. Client-Arbeitsplätze.
- Sinn und Zweck von Terminalserver-Systemen ist es, die Administration verteilter Komponenten zu vereinfachen und besser kontrollieren zu können.
    - Eine zentrale Serverlandschaft, die aus großen Serverfarmen bestehen kann, bedient „dumme“ Clientrechner (sog. Thin Clients), wobei die Anwendungsprogramme vollständig in den Servern ablaufen und die Clientrechner nur noch für Zwecke der Präsentation eingesetzt werden.

Die Idee hinter Terminaldiensten ist die Zentralisierung von Betriebsmitteln, um die beteiligten Systeme leichter administrieren zu können.

Betriebssystemarchitektur: Unix

[bsdp]

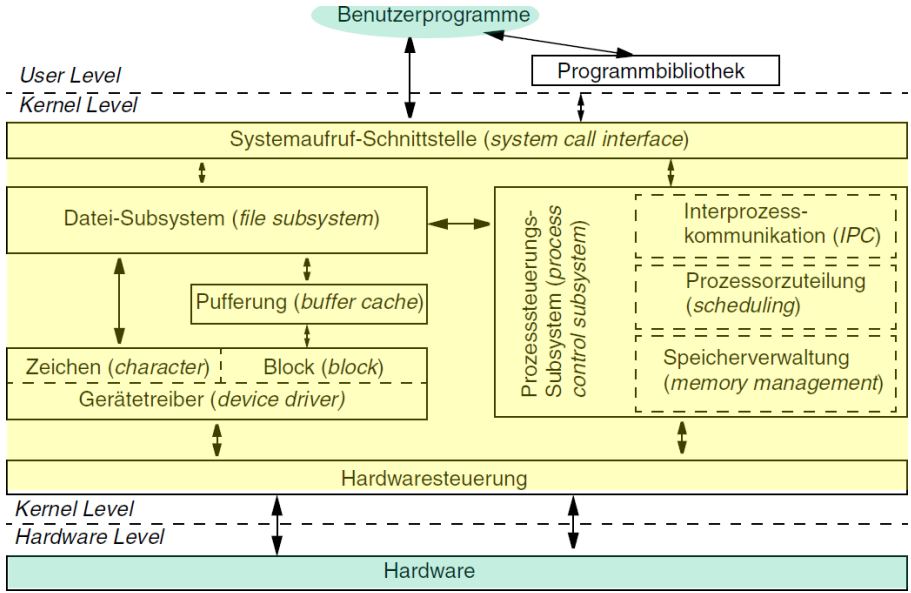


Abb. 19. Interne Struktur des Unix (Kern des System V Release 3)

Betriebssystemarchitektur: MS Windows

[bsdp]

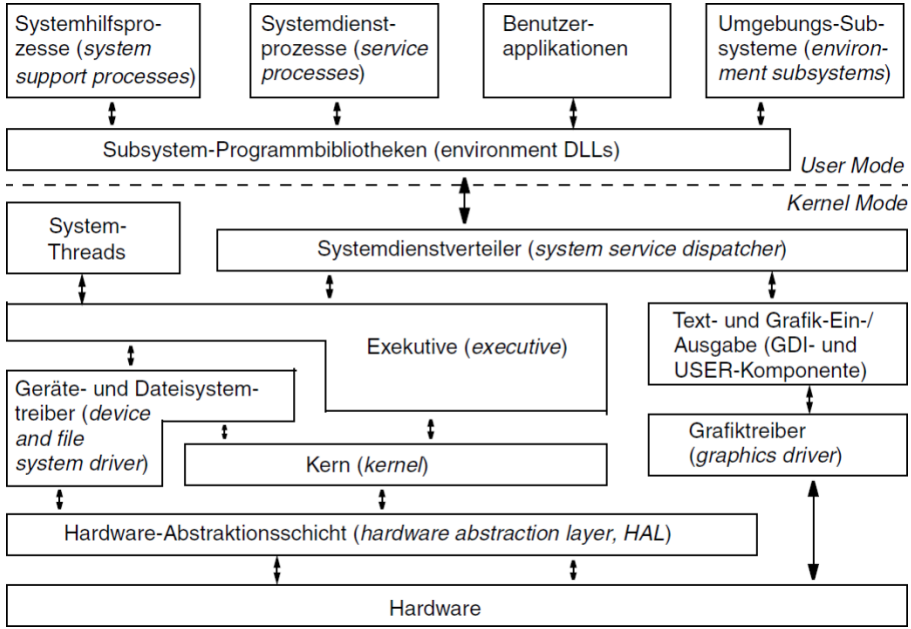


Abb. 20. Interne Struktur von Windows 10