

Betriebssysteme — Übersicht

Andreas Pommer
Institut f. Scientific Computing
Universität Salzburg
apommer@cosy.sbg.ac.at

2003-10-24

Zusammenfassung

Betriebssysteme sind ein nicht mehr wegdenkbarer Teil jedes modernen Computersystems. Im folgenden werden ein paar grundlegende Informationen und Konzepte über Betriebssysteme vermittelt.

Übersicht I

Betriebssystem, BS = operating system, OS

Ein Betriebssystem stellt den Anwendungsprogrammen eine virtuelle Maschine mit einem standardisierten Interface zur Verfügung, die darunter liegende Hardware wird soweit als möglich abstrahiert und versteckt.

Beispiel: handelsüblicher PC: egal, ob Intel 80386 oder Dual AMD Athlon, ob 1MB (naja) oder 1GB RAM, ob mit kleiner IDE-Festplatte von IBM oder einem grossen RAID-Array mit SCSI-Platten von Fujitsu, ob mit kleinem monochromen LC-Display über onboard-Grafik-Adapter oder grossem Farbmonitor mit AGP-Grafikkarte, . . . es läuft das gleiche Betriebssystem (mit unterschiedlichen Treibern), und den allermeisten Anwendungen ist die tatsächliche Hardware-Ausstattung egal, sie funktionieren immer gleich.

Übersicht II

- Aufgaben eines Betriebssystems
- Allgemeines, . . .
- Prozesse
- Speicherverwaltung
- IO-Management
- Netzwerkdienste
- User Interface

Was ist Aufgabe eines Betriebssystems? I

Starten, Ausführen und Beenden von Programmen: Prozessverwaltung, Scheduling, Synchronisation, Kommunikation, . . .

Speicherverwaltung: RAM, virtueller Speicher, . . .

Dateiverwaltung: Dateisystem, Rechteverwaltung, . . .

Ein-/Ausgabe-Verwaltung: Schnittstellen zur Aussenwelt: Menschen, andere Maschinen, . . .

Netzwerk: Anbieten von Diensten, Nutzen von Diensten, . . .

. . .

Was ist Aufgabe eines Betriebssystems? II

allgemein: Ressourcenverwaltung, Accounting, Schutz

Ressourcenverwaltung: CPU, Speicher, Ein-/Ausgabegeräte, . . . (Betriebsmittel)

Accounting: Ressourcenverbrauch abrechnen

Sicherheit (protection, security): Schutz des Betriebssystems vor den Prozessen und Benutzern, gegenseitiger Schutz von Prozessen und Daten von Benutzern, . . .

Was ist nicht Aufgabe eines Betriebssystems?

- Anzeigen von Web-Pages
- Verschicken von E-Mails
- Erkennen von Viren
- Filmpräsentation
- Verwalten von Datenbank-Daten
- Backup/Restore (?)
- . . .

Allgemeines I

Den Begriff *Betriebssystem* kann man mehr oder weniger eng sehen:

im engeren Sinn: nur der eigentliche Kern, ohne irgendwelche Programme, mit der die Verwaltung des BS den Benutzern leichter gemacht wird. Der Kern bestimmt die Funktionalität, die Programme können daran nur soweit Einfluss nehmen, als es der Kern selber erlaubt.

im weiteren Sinn: mit diesen Programmen. Da ein BS ohne diese Programme im Allgemeinen überhaupt nicht benutzbar ist, kann die Gesamtheit Kern+Verwaltungsprogramme als eine Einheit gesehen werden.

Beispiele von Betriebssysteme, Geschichte

Die Geschichte der Betriebssysteme ist eng mit der Geschichte ihrer Computer verbunden:

- zuerst: kein Betriebssystem, direkte Programmierung der Hardware
- Stapelverarbeitungssysteme (batch processing): z.B. VAX von DEC im wissenschaftlichen Bereich, IBM-Rechner bei Problemstellungen aus der Wirtschaft
- Mainframe OS (z.B. IBM S/360)
- Einbenutzer-Systeme (CP/M, DOS)
- Mehrprogrammbetrieb (z.B. frühere Windows-Varianten)
- Mehrbenutzerbetrieb (z.B. Unix-Varianten, aktuelles Windows)

Einige Quellen im Internet zur Geschichte von Betriebssystemen

CS322: A Brief History of Computer Operating Systems :

<http://www.cs.gordon.edu/courses/cs322/lectures/history.html>

The Creation of the UNIX Operating System :

<http://www.bell-labs.com/history/unix/>

8-Bit Operating Systems :

<http://www.armory.com/~spectre/tech.html>

History of operating systems :

<http://www.osdata.com/kind/history.htm>

Unix History :

<http://www.levenez.com/unix/>

Windows History :

<http://www.levenez.com/windows/>

Windows :

<http://members.fortunecity.com/pcmuseum/windows.htm>

Apple History :

<http://www.apple-history.com/noframes/>

Betriebssystem im engen Sinn: Kern/kernel

Es gibt 2 grundlegende Varianten von Kernen:

grosser, monolithischer Kernel: traditionelle Variante, die ganze Funktionalität wird durch eine grosse Einheit zur Verfügung gestellt. z.B. Windows, Linux, . . .

Microkernel-Architektur: kleine, stark gekapselte Einheiten mit stark beschränkter Funktionalität, die Einheiten sind leicht austauschbar. Wird als moderner angesehen, hat allerdings durch die Architektur (Message-Passing → Kontextwechsel) Performance-Probleme und dadurch Akzeptanzprobleme. z.B. Mach

Eine historische Anekdote zu diesem Thema: Diskussion zwischen A.Tanenbaum und L.Torvalds, z.B. unter folgender Adresse:

<http://www.oreilly.com/catalog/opensources/book/appa.html>

Prozesse

- Programm während der Ausführung
- “Arbeitseinheit” der meisten Betriebssysteme
- gesamte Zustandsinformation der Betriebsmittel eines Programms
- kann mehrere Ablauffäden (*threads*, *Coroutinen*) enthalten
- kann neue Prozesse erzeugen

Prozessverwaltung — Übersicht

- Prozesszustände
- Prozesskontext
- Scheduling
- Interprozesskommunikation
- Synchronisation

Prozess-Zustände

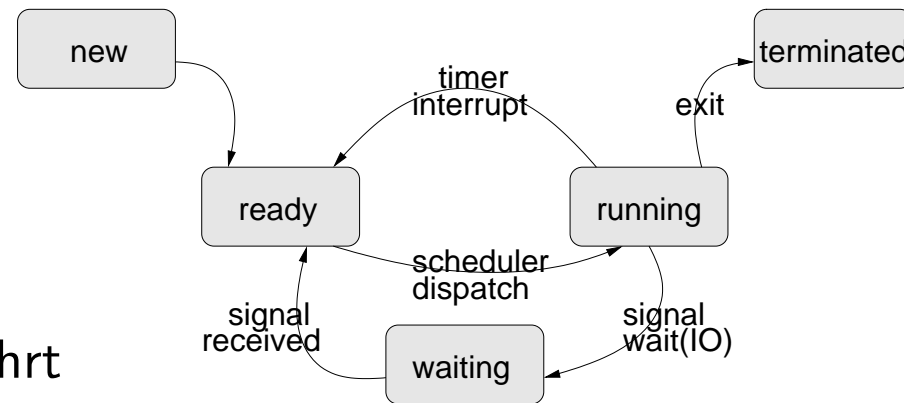
new: Prozess wird angelegt

ready: warten auf CPU

running: Prozess wird ausgeführt

waiting: Prozess wartet auf ein Ereignis
(meist IO)

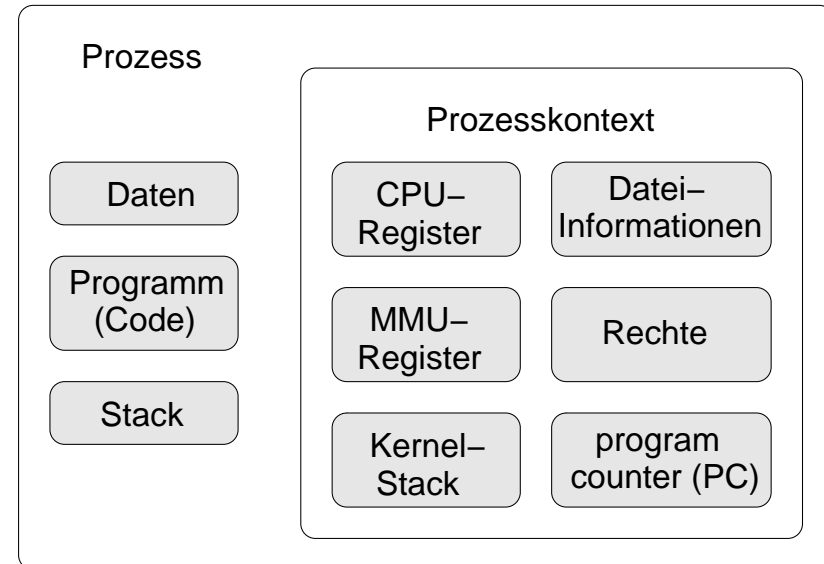
terminated: Prozess wurde beendet



Prozess-Kontext

Jeder Prozess hat seinen eigenen Kontext. Threads des gleichen Prozesses teilen sich einen Kontext. So ein Kontext beinhaltet alle Zustandsinformationen, die zur Fortführung des jeweiligen Prozesses notwendig sind.

Diese Informationen werden bei *Kontextwechsel* (engl. context switch), d.h. wenn der Prozess vom Zustand "running" in einen anderen wechselt, gespeichert. Bei der Rückkehr zu "running" werden die Informationen wieder geladen.



Scheduling I

Ziel des Scheduling ist eine optimale Auslastung des Gesamtsystems. Doch was ist optimal? Es gibt mehrere Ansätze, die sich teilweise widersprechen:

- maximale Auslastung der CPU
- maximaler Durchsatz (Jobs pro Zeiteinheit)
- faire Behandlung — alle Jobs, die gleich sind, auch gleich behandeln
- minimale Ausführungszeit: möglichst kurze Zeit zwischen Job-Beginn und Ende (inkl. Wartezeiten)
- minimale Wartezeit: Zeit, die der Prozess im Zustand “ready” verbringt.
- minimale Antwortzeit: wichtig für interaktive Systeme

Scheduling II — preemptive vs. non-preemptive

non-preemptive Scheduling (Prozess gibt freiwillig die Kontrolle ab):

First Come, First Served FCFS

Shortest Job First SJF, minimiert mittlere Wartezeit

Priority Scheduling PS

preemptive Scheduling (Prozess kann von BS die Kontrolle entzogen werden): in allen modernen BS zu finden. Einteilung der Ressource CPU in Zeitscheiben:

Round Robin Scheduling RR: FCFS mit Zeitscheiben

Prioritätsbasiertes RR Strategien zum Einfügen der Prozesse in die Warteschlange

Interprozesskommunikation

Kommunikation ist notwendig zur Koordination von Aufgaben. Interprozesskommunikation ist auf verschiedene Arten möglich:

Signal: rudimentäre Informationen über gewisse Ereignisse. asynchrone Kommunikation, Bearbeitung durch Exception-Handler z.B. Unix: SIGTERM Beenden des Prozesses, SIGFPE numerischer Fehler, SIGSEGV Benutzungsversuch einer nicht verfügbaren Speicheradresse, . . .

Pipes: unidirektional (oft Unix), bidirektional (Windows). Zwischen Prozessen, die in einer Eltern/Kind-Beziehung stehen

Nachrichten: allgemeinste Form

Kommunikationsarten: unicast, multicast, broadcast

Prozess-Synchronisation

Wenn mehrere Prozesse an zusammengehörenden Aspekten des gleichen Problems arbeiten, müssen sie miteinander kommunizieren und ihre Arbeit gegenseitig abstimmen — synchronisieren.

Kritischer Abschnitt — immer nur ein Prozess darf zur gleichen Zeit auf eine bestimmte Ressource zugreifen oder eine Aufgabe ausführen.

siehe Arbeit von z.B. Dijkstra: Lösung mit Signalen, Semaphoren und atomaren Aktionen.

z.B. Erzeuger-Verbraucher-Problem.

Problem: Verklemmungen (deadlocks)

grösseres Problem: verteilte Anwendungen (distributed systems)

Speicherverwaltung

Es gibt drei verschiedene Bereiche:

Massenspeicher: Dateiverwaltung, virtueller Speicher

Hauptspeicher: Zuteilung von RAM an verschiedene Prozesse

Anwendungsprogramm: Organisation des einem Prozess zugeteilten RAMs. z.B. garbage collection. Üblicherweise nicht Aufgabe des Betriebssystems.

Speicherverwaltung — Übersicht

- Speicherhierarchie
- Fragmentierung
- Virtueller Speicher
- Swapping, Paging
- Dateiverwaltung
- Rechte, Dateisystem

Die Speicherhierarchie

Daten, die erst vor kurzem benutzt wurden, befinden sich an der Spitze der Pyramide. Daten, die schon lange unbenutzt sind, wandern nach unten. Die im folgenden angegebenen Zahlen sind nicht exakt, sondern sollen die Größenordnungen veranschaulichen. Je kürzer die Zugriffszeit, desto teurer der Speicher, und daher auch eine kleinere Kapazität.

Zugriffszeit		Speicherkapazität
<1 ns	CPU-Register	<1 kB
2 ns	Cache	1 MB
10 ns	Hauptspeicher (RAM)	128–1024 MB
10 ms	Festplatte	20–200 GB
100 s	Bandspeicher (DAT, DLT,...)	100–... GB

Fragmentierung (Verschnitt)

Freier Speicher wird meist als Liste von freien Blöcken verwaltet. Welchen Block gibt man her, wenn eine Anfrage nach Speicher kommt?

Oft benutzte Zuteilungs-Strategien: first fit, best fit, worst fit.

externe Fragmentierung: es wäre in Summe genügend freier Speicher vorhanden, um eine Zuteilung durchzuführen, aber nicht in einem zusammenhängenden Block: Freier Speicher *zwischen* belegtem Speicher.

interne Fragmentierung: Speicher wird üblicherweise nicht auf Byte-Grenzen genau verwaltet, sondern in Blöcken gewisser Grösse: Ungenutzter Speicher *innerhalb* eines belegten Blockes.

Virtueller Speicher

Das Konzept des *virtuellen Speichers* wurde aus mehreren Gründen eingeführt: Benutzung von mehr Speicher wird möglich als RAM vorhanden ist (Swapping, Paging), Schutz der Prozesse, Lösung des Fragmentierungsproblem durch Relokation und Einführen von Speicherseiten (pages).

Logische Adressen sind Adressen innerhalb eines Prozesses

Physikalische Adressen sind die “tatsächlichen” Adressen im RAM.

Zur Umsetzung von logischen in physikalische Adressen ist die *Memory Management Unit* (MMU) zuständig, welche in die CPU integriert ist.

Die Umsetzung erfolgt transparent, die Anwendungsprogramme und auch weite Teile des kernels merken nichts davon.

Swapping

Der gerade ausgeführte Programmcode eines Prozesses muss sich im Hauptspeicher befinden.

Es kann aber sein, dass der Prozess ausgelagert wurde, weil andere Prozesse den Hauptspeicher dringender brauchten.

Dieses Aus- und Einlagern der gesamten Prozesse wird als *swapping* bezeichnet.

Dieses Konzept ist heute nur noch selten in Benutzung, aktuelle BS benutzen die Paging-Technik.

Paging I

Ein/Auslagern von *variabel grossen* Blöcken: es gibt ein Problem mit Fragmentierung

Paging ist ein Mechanismus zur Speicherverwaltung, bei dem der *physikalische* Adressraum eines Prozesses nicht mehr zusammenhängend sein muss.

keine externe Fragmentierung mehr (interne gibt es natürlich weiterhin)

wird in den meisten Betriebssystemen verwendet

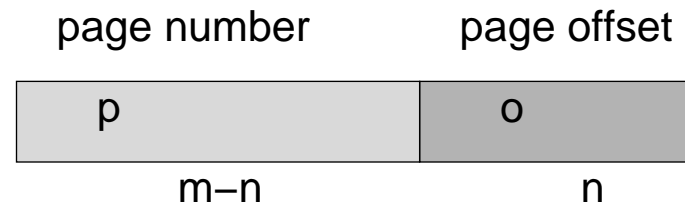
Paging II

der physikalische Speicher ist in gleich grosse Blöcke unterteilt, den *Rahmen* oder *frames*.

der logische Speicher ist auch in Blöcke der selben Grösse unterteilt, den *Seiten* oder *pages*.

Das gleiche gilt für den Auslagerungsspeicher (backing store, traditionell auch swap-space bezeichnet)

Eine von der CPU generierte (logische) Adresse ist zweigeteilt in die Seitennummer (*page number*, p , insgesamt = 2^{m-n} Seiten), und den Offset innerhalb der Seite (*page offset*, d , Seitengrösse 2^n):



Die Seitennummer ist ein Index in die *Seitentabelle*, *page table*, welche die Startadresse jeden Rahmens des physikalischen Speichers enthält.

Paging III — Demand Paging ↔ Prepaging

Mit Paging wird der Hauptspeicher für jeden Prozess zu einem grossen zusammenhängenden Block abstrahiert, es sind auch Prozesse möglich, die nicht komplett im Hauptspeicher sind (z.B. auch solche, die grösser sind)

Die Swapping-Einheit war ein Prozess, die Paging-Einheit ist eine Seite, ein Prozess besteht aus mehreren Seiten

Wird ein auf die CPU wartender Prozess aktiviert, so kann man die dazu gehörenden Seiten vom Auslagerungsspeicher laden: *Prepaging*

man kann auch warten, bis der Prozess versucht, auf eine derzeit nicht im Hauptspeicher befindliche Seite zuzugreifen, und sie erst dann einzulagern: *Demand Paging*

Dateiverwaltung

Daten, die auf Festplatten abgespeichert werden, werden in Einheiten mit dem Namen *Datei* (engl. file) verwaltet, Dateien werden in *Verzeichnissen* (engl. directories) zusammengefasst.

Die Struktur einer Datei, d.h. ihr Inhalt, interessiert ein Betriebssystem nicht, dafür sind die Anwendungsprogramme zuständig.

Durch die rekursive Struktur von Verzeichnissen ergibt sich eine *hierarchische Anordnung*, d.h. eine Baumstruktur.

Manchmal ist eine allgemeinere Struktur erlaubt, und zwar azyklische Graphen: z.B. bei Unix mit Hilfe von *links*.

Zugriffskontrolle

- Ein Prozess kann mit (Dateisystem-)Objekten verschiedene Aktionen durchführen: z.B. read, write, execute, append, delete, list, Das Besitzverhältnis kann z.B. so kategorisiert werden: Besitzer, Gruppe, Rest (Unix).
- Jeder Datei kann man nun für jede Benutzerkategorie gewisse Rechte vergeben, z.B. (Unix-Modell mit r,w,x):

```
drwxr-sr-x    2 andreas  wavelab    4096 Oct 21 22:28 .
-rw-r-----  1 andreas  wavelab   99031 Oct 20 18:06 uebersicht.pdf
-rw-r-----  1 andreas  wavelab   16862 Oct 21 22:28 uebersicht.tex
```

- das ist oft eine zu grobe Einteilung. Abhilfe:
- *Access Control Lists (ACL)*: Detaillierte Vergabe von Rechten an einzelne Benutzer und/oder Gruppen.

Filesystem (FS) und Meta-Informationen

- Informationen über das Dateisystem selber werden in speziellen Datenstrukturen auf den Datenträgern selber gespeichert: *superblock* (Unix) oder *Master File Table* (Windows NTFS).
- Informationen über die einzelnen Dateien werden in *File Control Blocks* (*FCB*) (Unix: *inode*) gespeichert, das sind: Dateiname, Datum (access, modify, change), Besitzer+Gruppe, Rechte, Grösse, Position am Datenträger, . . .
- Diese Informationen können auf verschiedene Weise organisiert sein: z.B. als Listen innerhalb eines Verzeichnisses (z.B. ext2, FAT), oder Baumartig (z.B. mittels B-Bäumen: reiserfs, NTFS, HPFS).
- Zur Wahrung der Konsistenz der Metainformationen des Dateisystems bei Problemen können manche FS die Schreibaktionen transaktionsorientiert durchführen: *journaling*, *logfile*

IO-Verwaltung

Ein Computer, der nur Daten berechnet, aber nie mit der Aussenwelt kommuniziert, d.h. nie Daten einliest oder ausgibt, kann eigentlich nicht existieren.

Aktuelle Computer bieten eine Vielzahl an Möglichkeiten an:

- zur Kommunikation mit Menschen: Grafikkarte/Monitor, Tastatur, Maus, Mikro+Lautsprecher+AD/DA-Wandler für Aufnahme/Wiedergabe von Text und Musik, . . .
- zur Kommunikation mit Maschinen: Datenübertragung über verschiedene Medien (Kupfer/Glasfaser-Kabel, Funk, mit parallelen und seriellen Protokollen), Datenspeicherung auf verschiedenen Medien (mit wahlfreiem Zugriff — Plattenspeicher, oder sequentiellm Zugriff — Bandspeicher), . . .

⇒ Ein-/Ausgabe-Verwaltung (IO)

Grundsätzlicher Aufbau

user mode	Benutzerprozess
kernel mode	kernel-Verteiler
	Auftragsverwaltung
	Pufferung
	Treiber
	Controller
	Gerät

Idealisierte, grundsätzliche Struktur einer IO-Verwaltung.

Manche Schichten oder Schichtenkombinationen können auch mehrfach vorkommen.

Üblicherweise kommuniziert jede Schicht nur mit ihren direkten Nachbarn, stellt Anfragen an sie, und beantwortet andere Anfragen.

Interface zu den Anwendungsprogrammen

Betriebssysteme bieten den Anwendungsprogrammen verschiedene Varianten an, um mit der Aussenwelt kommunizieren zu können:

Zeichen- vs. block-orientiert: z.B. Tastatur vs. Festplatte

Sequentieller vs. wahlfreier Zugriff: fixe Reihenfolge der Daten? (Band vs. Festplatte)

synchrone vs. asynchrone Übertragung: warten auf Antwort?

weilers:

gemeinsam vs. exklusiv genutzte Kanäle: z.B. mehrere Prozesse können auf eine Festplatte schreiben, aber nicht auf einen Drucker

Geschwindigkeit: abhängig vom benutzten Gerät, und teilweise vom Kommunikationspartner (z.B. serielle Schnittstelle)

read/write, read only, write only: z.B. Festplatte vs. Maus vs. Drucker

Problematiken im Umfeld von IO

IO scheduling: Werden Festplattenzugriffe umsortiert, damit die Kopfbewegungen minimal sind? Gibt man gewissen IP-Paketen einen Vorzug?

buffering/caching: Puffert man Aktionen, kann man die Daten zu grösseren Einheiten zusammenfassen (grösserer Durchsatz, aber auch höhere Latenz). Caching verbraucht zusätzliche (teure) Ressourcen (meist RAM) — Abwägung.

spooling, Reservierung: In welcher Reihung werden die Aufträge abgearbeitet, wie viele können in der Warteschlange sein, bevor weitere abgewiesen werden, Leser ↔ Schreiber, . . .

Fehlerbehandlung: Welche Fehler können auftreten, welche kann man selber beheben, welche erfordern Eingriff von aussen?

Hardware: Polling & Interrupts

Die Übertragung von Daten von oder zu Geräten kann relativ aufwändig sein, prinzipiell wird die Methode das *Handshaking* benutzt. Diese lässt sich auf zwei Arten realisieren:

Polling: in einem *Status-Register* auf gewisse Werte testen, bevor in ein anderes Register Kommandos (oder Parameter) geschrieben werden. Wenn man auf ein bestimmtes Ereignis wartet, muss dieses Status-Register immer wieder abgefragt werden: CPU-intensiv, aber sehr reaktionsschnell

Interrupts: der CPU wird signalisiert, dass ein gewisses Ereignis eingetreten ist, der Interrupt-Handler des OS wird aufgerufen und kümmert sich um die Bearbeitung. Speichern des aktuellen CPU-Kontextes!

Netzwerkdienste

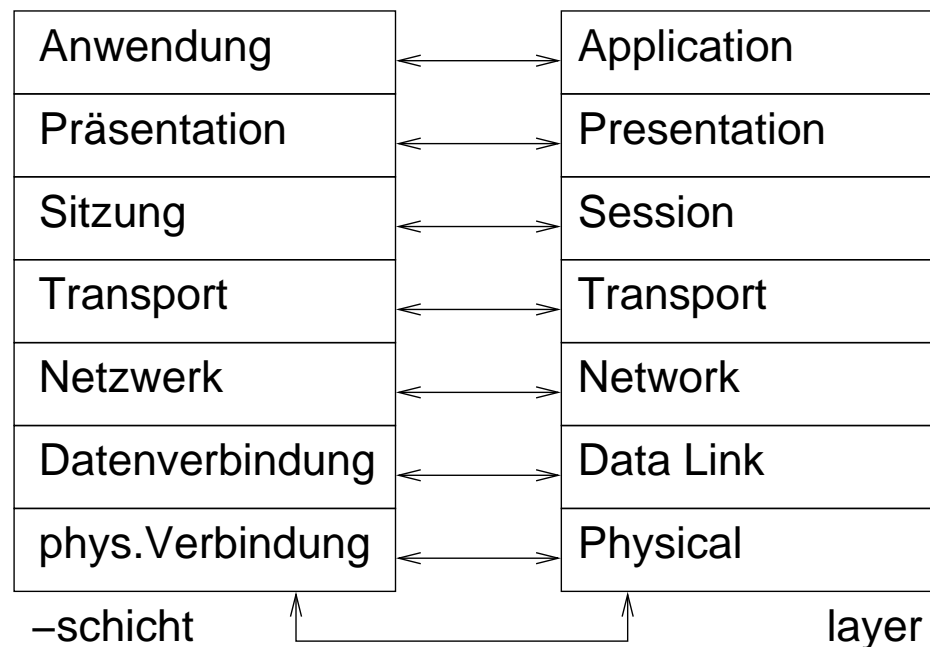
Moderne Computer existieren nur selten für sich alleine, zusätzliche Funktionalität erlangen sie, bzw. bieten sie durch Netzwerkdienste an.

Ein paar wichtige Kategorien sind:

- file sharing: Dokumente und Daten können gemeinsam erstellt und genutzt werden (z.B. NFS, Windows Laufwerksfreigaben, Web Server, . . .)
- message transfer: Übermittlung elektronischer Nachrichten zur Koordination und Kommunikation (z.B. email, IRC, IM, . . .)
- printer sharing: Ausdrücke auf Druckern, unabhängig von deren Aufstellungsort.
- job management: Verteilung von Teilen eines Rechenjobs auf Rechner, die anderweitig unbenutzt sind.

Protokolle

Oft ist es für die Benutzer nicht erkennbar (d.h.transparent), ob ein Dienst lokal oder von einem anderen Rechner zur Verfügung gestellt wird (*verteiltes Computersystem*). Für eine korrekte Interaktion sind festgelegte, d.h. standardisierte Protokolle notwendig. Das ISO-OSI 7-Schichten Modell:



Bedeutung der einzelnen Schichten?

Beispiele für Protokolle?

Vor-/Nachteile von Netzwerken

Benutzt man Netzwerke, muss nicht mehr jeder Rechner alle Dienste erbringen, sondern man kann einzelnen Rechnern Spezialaufgaben zuweisen. Je nach dem genauen Konzept sind dabei die folgenden, sowie andere Punkte zu bedenken:

- Zentrale Datenhaltung: Aktualität (keine lokalen, alten Versionen), längere Zugriffszeit, sowie höhere Datensicherheit :-)
- Gemeinsame Benutzung von Ressourcen: höhere Auslastung (z.B. Abteilungsdrucker), Spezialgeräte werden möglich, . . .
- Wartungsaufwand bei Clients ist geringer (v.a. in der Masse, daher auch billigere Hardware möglich) ↔ Server
- soziale Aspekte: je nach Organisation und persönlicher Einstellung der Betroffenen (Unterstützung der Arbeit ↔ Bevormundung)

UI — User Interface

Gehört ein Benutzer-Interface zu den Aufgaben eines OS? Wenn ja, wie weit?

Es gibt unterschiedliche Ansätze:

Unix: nur stark limitiert: Zeichen auf Bildschirm bringen, das X-Window System ist eigenständig, . . . X11 gehört also nicht zum Kern, kann es trotzdem als zum OS gehörig angesehen werden?

Windows: gesamte Fensterverwaltung ist Teil des OS

Mac: MacOS X hat Unix-Kern, GUI ist getrennt — wie war das früher?

Zusammenfassung

Ein Betriebssystem übernimmt viele Aufgaben, die an einen Computer gestellt werden, aber nicht alle.

Es sind Aufgaben, die automatisiert ablaufen. Es sind oft Aufgaben, die Benutzerprozess nicht ausführen kann oder darf.

“Ressourcen-Management”

Danksagungen, Literatur

Diese Folien basieren auf:

Folien von Wolfram Stering

R.Brause “Betriebssysteme – Grundlagen und Konzepte”, Springer Verlag, 2.Auflage, 2001, ISBN 3-540-67598-1.

Silberschatz, Galvin, Gagne “Operating System Concepts”, John Wiley and Sons, 6th edition, 2002, ISBN 0-471-41743-2.