

AllgemeinesGründe für die Verbreitung von Computern:

- Hardware ist preiswert und leistungsfähig
- Betriebssystem verwaltet Hardware
↳ Unabhängigkeit zwischen Hard- und Software
- Vernetzen von Computern möglich

Nutzergruppen von Betriebssystemen und Rechnernetzen

- reiner Anwender (Benutzeroberfläche)
- Anwendungsprogrammierer (Programmierschnittstelle)
- Systemadministrator (Superuser)
- Systementwickler (Weiterentwicklung Betriebssystem und Systemsoftware)

Binärdarstellung

	IEC-Prefix-Notation
8 Bit = 2^3 Bit = 1 Byte	1 Byte
2^{10} Byte = 1 KB	1 KiByte
2^{10} KB = 1 MB	1 MiByte
2^{10} MB = 1 GB	1 GiByte
2^{10} GB = 1 TB	

Konzepte der Informatik

- Abstraktion: auf höherer Ebene keine Details notwendig; Konzentration auf das Wesentliche
- Kapselung / Geheimnisprinzip: auf höherer Ebene kein Detail zulässig
 - ↳ Implementierung leichter zu ändern
 - ↳ Volle Kontrolle über die Funktionsweise
- Schichtenaufbau: jede Schicht leistet Dienste für die nächsthöhere Schicht und nutzt nächtniedrigste Schicht

Rechnerarchitektur und Systemtypen

Ursprüngliche Rechner: Daten im Speicher, Rechner wurden durch physische Veränderung der Hardware programmiert

Von-Neumann-Rechner: Programme werden wie Daten im Speicher hinterlegt

↳ Vorstellung des Rechners als universelle Maschine zum Ausführen von Programmen

Bestandteile: - Prozessor (CPU) - Hauptspeicher (main memory) - Ein-/Ausgabegeräte (IO-Devices)

Systemtypen:

- Einprozessoren
- Parallelrechner: mehrere Prozessoren teilen sich einen gemeinsamen Hauptspeicher
 - Vorteile: - ggf. Effizienzsteigerung bei gut parallelisierbaren Problemen
 - Fehlertoleranz und Ausfall Sicherheit
 - Kostenersparnis durch mehrere CPUs in einem Gehäuse
- Verteilte Systeme: mehrere Prozessoren mit jeweils eigenem Hauptspeicher
 - ↳ Kommunikation nur über Nachrichtenverband
- Realzeitsysteme: Ereignisse müssen innerhalb einer vorgegebenen Zeit berechnet werden.

Hardware und GeräteProzessor (CPU)

- führt alle eigentlichen Berechnungen im Rechenwerk (ALU / arithmetic-logic-unit) aus
- greift auf Programme und Daten im Hauptspeicher zu
- Prozessor arbeitet getaktet (1 Takt = 1 Elementaroperation, Taktfrequenz = Anzahl Takte pro Sekunde, 1 GHz = 1 Takt pro Nanosekunde)

Prozessorregister

- Befehlszähler (Program Counter): Speicheradresse des nächsten Befehls
- Befehlsregister (Instruction register): aktueller Befehl
- Speicheradressregister (memory address register): Adresse der als nächstes zu lesenden / schreibenden Speicherzelle
- AKKumulator: Datenregister für Zwischenergebnisse

Arbeit

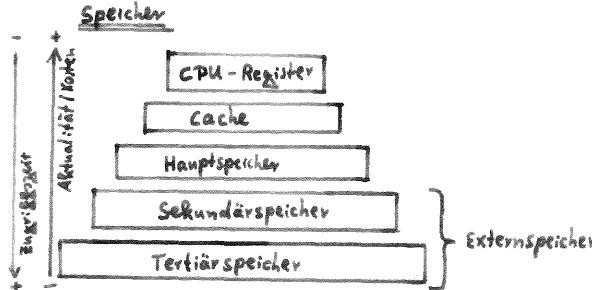
- Prozessor arbeitet Befehle in Zyklen ab

1.) Holphase (Fetch stage)

- Adresse des nächsten Befehls wird vom Befehlszähler in den Speicheradressregister übertragen
- Die Adresse wird auf den Adressbus geschrieben
- Die Daten werden vom Hauptspeicher über den Datenbus übertragen und in das Befehlsregister geschrieben sowie interpretiert

2.) Ausführungsphase (Execution stage)

- Befehl wird ausgeführt, ggf. werden weitere Daten / Adressen geholt
- Befehlszähler wird inkrementiert bzw. bei einem Sprungbefehl auf einen Wert gesetzt



Hauptspeicher / Primärspeicher

- Folge gleich großer, einzeln adressierbarer Speicherzellen → wegen wahlfreiem Zugriff auch als RAM bezeichnet
- Hauptspeicher enthält Daten und Programme (nicht-persistent)
- Wort := Inhalt einer Speicherzelle
- Speicherzugriff := Lesen oder Schreiben eines Datums in die Zelle
- Zugriffszeit := Zeit zwischen Aufrufen einer Adresse auf den Adressbus und Ankunft der Daten über Datenbus

Cache

- Kleiner, aber schneller als Hauptspeicher
- Enthält häufig benötigte Daten im Kopie
- Werden Daten benötigt, so wird erst im Cache nachgekaut

Aufgabe des Betriebssystems

- 1) Cache-Management: welche Daten sollen bei vollem Cache überschrieben werden?
- 2) Cache-Konsistenz: Konsistenz zwischen Cache und Hauptspeicher sicherstellen bzw. Fehler durch Inkonsistenz abfangen

Sekundärspeicher

- ermöglicht persistente Speicherung von Programmen und Daten
- Fasst alle Daten, da Hauptspeicher zu teuer
- nicht zur Langzeitarchivierung geeignet, da zu teuer, im PC verbaut und als Flächspeicher begrenzt lösbar
- längere Zugriffzeit als Primärspeicher

↳ Fest- / Magnetplatte

- beidseitig magnetisierte, rotierende Platte
- Oberfläche ist in kreisförmige Spuren gleicher Breite unterteilt
- Spuren setzen sich aus Sektoren zusammen
- wahlfreier Zugriff auf Sektoren über beidseitigen Schreib-/Lesekopf
- mehrere Platten übereinander bilden einen Zylinder

Zugriffszeit

- | | |
|--|--|
| <ul style="list-style-type: none"> - Positionierungszeit: Kopf auf Spur setzen - Latenzzeit: Zeit, bis Sektor vorbeiläuft - Übertragungszeit: Daten in Puffer schreiben | Suchzeit, kann verringert werden durch: <ol style="list-style-type: none"> a) zusammengehörige Informationen in benachbarten Sektoren oder Zylindern speichern b) Anträge günstig bearbeiten (disk scheduling) |
|--|--|

Strategien zum disk scheduling

- FCFS (first-come first-served): Anträge werden nach Eingangsreihenfolge bearbeitet
- SSTF (shortest seek-time first): Antrag mit dem nächstgelegenen Sektor zum Kopf wird bearbeitet
- SCAN: Kopf läuft zyklisch über alle Spuren

Block := Nutzinformation eines Sektors (Daten, Sektornummer, Zylindernummer, Hilfsdaten Fehlerkorrektur)

↳ z.B. Parity Bit := Wertsumme bestimmter Datenbits zum Prüfen

↳ RAM-Disk (Spezialfall)

- reservierter Teil des Hauptspeichers
- kann wie Magnetplatte genutzt werden
- Vorteil: geringe Zugriffszeiten
- Nachteil: Daten werden nicht persistent gespeichert

↳ Flashspeicher (solid state disk; SSD)

- elektrischer EEPROM-Speicher (electrically erasable programmable read only memory)
- geringe Zugriffszeiten als bei Magnetplatte
- besteht aus kleinen Blöcken à 128 KByte oder 256 KByte
- kleine beweglichen Teile → widerstandsfähig gegen Erschütterungen
- Löschoperationen können nur blockweise ausgeführt werden (→ alle Bits auf 1 setzen)
- Block verträgt nur begrenzt Anzahl an Löschoperationen (wear-out) → Organisationsstruktur notwendig
- Nachteil: Seiten, erst wieder nach Löschung eines Blocks beschreibbar

↳ Variante 1 NOR: Block besteht aus einzeln les- und schreibbaren Daten-Bits

↳ Variante 2 NAND: Block besteht aus Seiten mit Daten + Verwaltungsinformation; Lesen und Schreiben nur zeitweise möglich

Tertiärspeicher

- günstig und leicht vom Rechner trennbar

- ↳ CD (compact disk) / DVD (digital versatile disk)
 - optische Datenspeicherung, wird mit Laser abgetastet
 - ↳ CD/DVD - ROM: nur lesbar
 - ↳ CD/DVD - R: einmal beschreibbar
 - ↳ CD/DVD - RW: mehrfach beschreibbar

↳ Disketten

- Einziger Festplattenlaufwerk
- längere Zugriffzeit, kleineres Datenvolumen
- wahlgreier Zugriff

↳ Magnetbänder

- größere Kapazität als Magnetplatte, da Oberfläche größer
- Aufzeichnung von Daten in variable Blöcke
- nur sequentieller Zugriff möglich (vor-/zurückspulen)
- geeignet zur Langzeitarchivierung

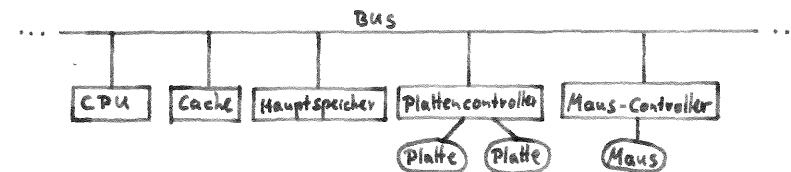
↳ externe Festplatten

↳ Flash-basierte Speicher

- MSB-Stick
- Speicherkarten

Arbeit des RechnersKommunikation mit GerätenBUS

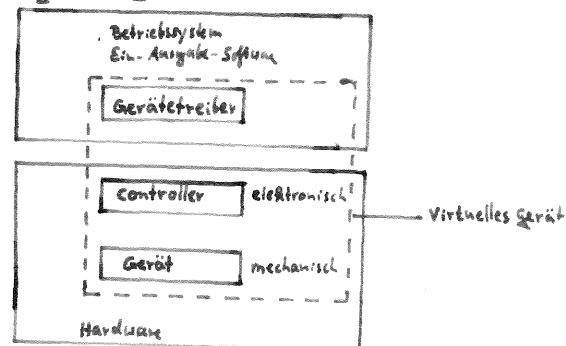
- Bus := Bündel von Leitungen mit Protokoll
- verbindet viele Geräte mit einander (CPU, HS, IO)
- zu jedem Zeitpunkt wird nur eine Nachricht gesandt

Controller

- elektronische Hardware
- steuert das jeweilige Gerät → Belastung der CPU wird verringert
- arbeitet parallel zur CPU

Controllerregister

- Datenausgangsregister (data-out): Daten vom Treiber
- Dateneingangsregister (data-in): Daten für Treiber
- Statusregister
- Kontrollregister: Befehle vom Treiber

Unterbrechungen

- Prozessor besitzt einen über den Bus ansprechbaren Unterbrechungseingang
- Unterbrechungen (interrupts) können von der Hardware und Software ausgelöst werden
- Abstrakt: Unterbrechung ist ein Mechanismus zum Aktivieren des Betriebssystems
- Während dem Bearbeiten einer Unterbrechung kann der Unterbrechungseingang temporär deaktiviert werden
 - ↳ Alternativ können Unterbrechungen priorisiert werden

Hardware-Unterbrechungen

- wird von externen Geräten ausgelöst
- kann nicht reproduziert werden
- Beispiel: Controller will CPU über ausgeführten Leseauftrag informieren
- Vorgehen bei Unterbrechung:
 - 1.) Wert des Befehlszählregisters in systemeigenen Hauptspeicherbereich schreiben
 - 2.) allgemeine Unterbrechungsprotokoll (interrupt handler) laden
 - 3.) feststellen, welches Gerät die Unterbrechung ausgelöst hat (Abfrage aller Geräte oder Unterbrechung-Controller befragt)
 - 4.) Mit Gerätenummer aus dem betriebssystem-internen Unterbrechungstext oder Startadresse der gerätespezifischen Unterbrechungsroutine laden und ausführen
 - 5.) Betriebssystem interne Unterbrechungsliste des Geräts

[5.] Ggf. Registerinhalte im systemeigenen Speicher sichern, falls längere Berechnungen nötig]

- 6.) Befehlszählregister + ggf. Registerinhalte vor Unterbrechung laden
- 7.) return from interrupt

Software - Unterbrechungen (traps)

- wird von Software ausgelöst
- kann reproduziert werden
- Fall 1: Ausnahmen (exceptions)
 - ↳ Division durch null
 - ↳ Fehler bei Speicherzugriff

- Fall 2: Systemaufruf (system call)

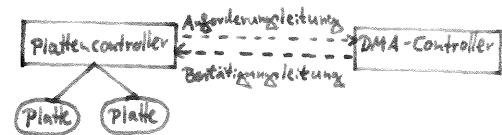
- ↳ Programm nutzt Betriebssystem über Programmierschnittstelle
- ↳ Funktion aufrufen: z.B. Parameter in CPU-Register oder als Argumente übergeben
- ↳ trap-instruction
- ↳ vorgehen analog zur Hardware-Unterbrechung

Direkter Speicherzugriff (DMA)

- DMA-Controller kann eigenständig (also ohne CPU) Daten über den Bus in den Hauptspeicher schreiben

- Vorgehen:

- 1.) Gerätetreiber teilt
 - ↳ Gerätetreiber Nummer des zulesenden Blocks mit
 - ↳ DMA-Controller Angangsadresse des Hauptspeicherbereichs mit
- 2.) Gerätetreiber liest Wort in Ausgangsregister und verständigt DMA-Controller über Anforderungsleitung
- 3.) DMA-Controller
 - ↳ Schreibt Adressen auf Bus und gibt Übertragung durch
 - ↳ informiert über Bestätigung leitung den Gerätetreiber
 - ↳ inkrementiert Adressregister
 - ↳ löst Unterbrechung aus, falls alle Wörter des Blocks übertragen wurden



- Vorteil: Im Vergleich zur unterbrechungsgenerierten Ein-/Ausgabe (Controller reicht Wort in Ausgangsregister und löst Unterbrechung aus, Gerätetreiber veranlasst Übertragung) wird der Gebrauch der CPU für Übertragungsdienste minimiert.

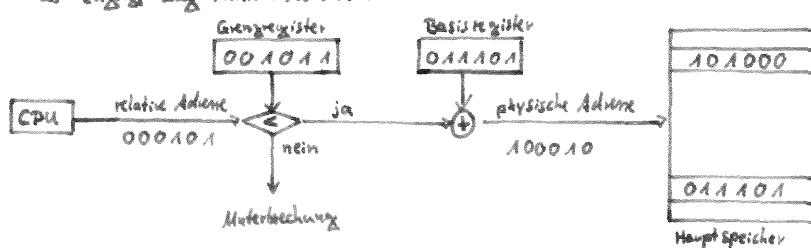
Adressraum

- Adressraum eines Programms := vom Programm durch das Betriebssystem zugewiesener Hauptspeicherbereich
- enthält sowohl das Programm selbst als auch die Daten
- Programm darf nur auf seinen Adressraum zugreifen
- Problem: Beim Compilieren steht der Adressraum zur Laufzeit nicht fest —> Vergabe relativer Adressen

↳ Adressraum := $\{ \begin{array}{l} \text{Basisregister (base register)} : \text{niedrigste Adresse des Adressraums} \\ \text{Grenzregister (limit register)} : \text{Länge des Adressraums} \end{array} \}$

↳ Register können nur von privilegierten Maschinenbefehlen verändert werden

↳ Zugriff auf relative Adresse:



- Vorteil: Adressräume können im Hauptspeicher verschoben werden (relokierbar)

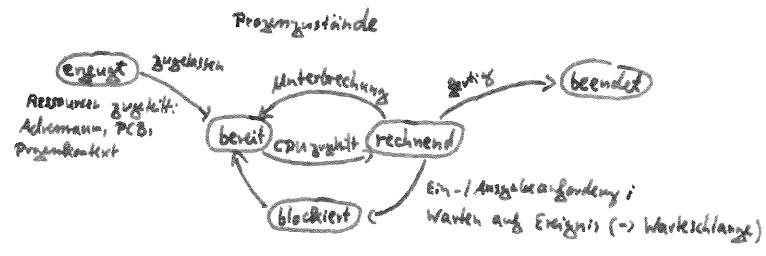
SpeicherSchutz

- Anwendungsprogramme könnten andere Programme, imbetriebenes Betriebssystem überschreiben, löschen oder verändern
 - ↳ Zugriff auf Haupts- und Sekundärspeicher sowie auf Geräte einschränken
- Prozessor unterscheidet durch besonderes Bit zwischen dem Systemmodus (system mode) und Benutzermodus (user mode)
 - ↳ alle Anwendungsprogramme laufen im Benutzermodus
 - ↳ Betriebssystem läuft im Systemmodus
 - ↳ Kann privilegierte Maschinenbefehle ausführen (imbesondere Umgebungsbefehl System- / Benutzermodus)
- Moduswechsel:
 - ↳ Bei einer Unterbrechung schaltet der Prozessor in den Systemmodus und startet die Unterbrechungsroutine (Teil des Betriebssystems)
 - ↳ Vor der Rückkehr kehrt der Prozessor zurück in den Benutzermodus

Prozesse

- Prozess := in Ausführung befindliches Programm + Prozesskontext
- Prozesskontext wird in Prozesskontrollblock zusammengefasst
 - ↳ Registerinhalt, insbesondere Befehlszähler
 - ↳ Adressraumgrenzen
 - ↳ Prognummern
 - ↳ Prioritäten
 - ↳ System- / Benutzermodus
- Bei Einprozessorsystemen ist zu jedem Zeitpunkt nur ein Prozess rechnend

KE 1



Scheduling - Strategien

- Scheduling := Entscheidung, welcher Prozess als nächster rechnen darf
- wird vom Scheduler übernommen

a.) nicht präemptive Systeme (Prozess gibt CPU freiwillig ab)

- FCFS (First-come, first-served)
 - ↳ leichte Implementierung durch Warteschlungen
 - ↳ Nachteil: Rechenzeit-intensive Prozesse halten nachfolgende Prozesse auf
- SJF (shortest job first)
 - ↳ bereite Prozesse werden nach aufsteigendem Rechenzeitbedarf ausgeführt
 - ↳ Voraussetzung: Rechenzeit ergebnismäßig gut vorhersagbar
 - ↳ geeignet für Stapel- / Batchbetrieb

b.) präemptive Systeme (Prozess wird CPU entzogen)

- Round Robin
 - ↳ jeder bereite Prozess bekommt vom Scheduler eine Zeitscheibe derselben Dicke
 - ↳ Prozesse werden reihum bedient → alle Prozesse werden gleich behandelt
- Round Robin mit Prioritäten
 - ↳ Dicke der Zeitscheibe ist abhängig von der Priorität des Prozesses oder der bereits verbrauchten Rechenzeit
- dynamische Prozesspriorisierung
 - ↳ Priorität dynamisch an die Bedürfnisse des Prozesses anpassen
 - ↳ z.B. länger, aber häufigere Zeitscheiben für interaktive Prozesse mit vielen I/O-Operationen

Technische Details:

- Timer speichert Dicke der Zeitscheibe in Register
- Timer (oft eigener Chip) ist an CPU-Takt ausgetimed
- Timer dokumentiert Zeitscheibe und löst nach Ablauf die Zeit eine Unterbrechung des CPUs aus
- Umschalten zwischen den Prozessen (Kontextwechsel) durch Dispatcher

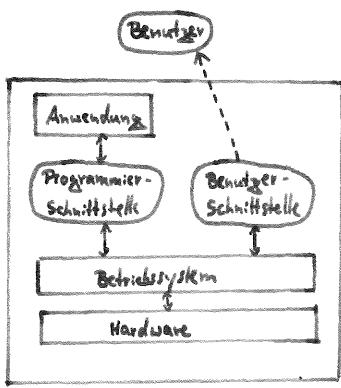
Zusammenfassung: Gerätzugriff

- Beispiel: Programm lädt Systemaufruf zum Lesen eines Datensatzes aus einer Datei aus
 - ↳ Softwareunterbrechung: Kontext retten, Ein-/Ausgabeteil des Betriebssystems aufrufen
 - ↳ Prüfe, ob Daten schon im Cache stehen. Falls ja, kopiere Datensatz in Adressraum des Prozesses. Nach Unterbrechung Prozess fortführen - Ende
 - ↳ Plattenzugriff erforderlich: Prozess blockieren, Leseaufruf an Warteschlange anhängen, Gerätetreiber informieren
 - ↳ Gerätetreiber entnimmt Auftrag aus Warteschlange, reserviert im Systemeigenen Speicherbereich Platz, schickt Leseaufrug an Controller, warte
 - ↳ Gerätetreiber bestimmt physische Adresse, führt Leseaufrug aus, überträgt mit DMA-Controller Daten in den systemeigenen Speicher. Anschließend Unterbrechung
 - ↳ Startet mit Unterbrechungsvektor die Unterbrechungsroutine des Plattenlaufwerks, mache Gerätetreiber bereit und Reise von Unterbrechung zurück
 - ↳ Gerätetreiber prüft in Warteschlange, von welchem Prozess der Leseaufrug stammt. Informiert Ein-/Ausgabeteil des Betriebssystems
 - ↳ Ein-/Ausgabeteil kopiert die Daten aus dem Systemspeicher in den Adressraum des Prozesses und macht Prozess wieder bereit
 - ↳ Programm setzt Arbeit hinter Systemaufruf fort

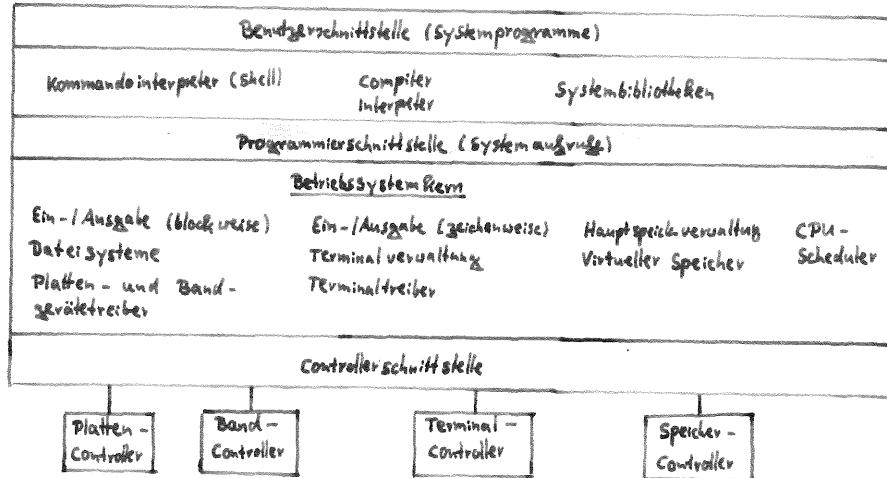
Betriebssystem

- verwalten und steuert die Hardware → Ausführung von Anwendungen ermöglichen
- liegt als eigene Schicht zwischen Hardware und Anwendungprogrammen → Unabhängigkeit von Hard- und Software
- Anwendungen und Benutzer kommunizieren nur über das Betriebssystem mit der Hardware
- Eine Schicht bietet Dienste für nächsthöhere Schicht als Schnittstelle (= Menge von Operationen) an

Vereinfachtes Schichtenmodell



Struktur UNIX



- Betriebssystem ist selbst Software
- Start des Rechners (booting):
 - ↳ Mrlader (bootloader) ist in einem speziellen, persistenten Teil des Hauptspeichers hinterlegt (Firmware / Bios)
 - ↳ Mrlader lädt eigentlichen Lader aus dem Sekundär - in den Hauptspeicher
 - ↳ Lader lädt die wesentlichen Teile des Betriebssystems aus dem Sekundär - in den Hauptspeicher

Programmierschnittstelle

- dient der Kommunikation der Programme mit dem Betriebssystem

Funktionen

- wait event
- Signal event
- halt
- abort
- dump
- load

Dateien

- execute
- fork
- set attributes
- terminate

Informationen

- create
- delete
- open
- close
- read
- write

Kommunikation

- a.) Nachrichtenverkehr (message passing)
 - Verbindungsorientierter Daten austausch (analog Briefverkehr)
 - send
 - receive
 - Verbindungsorientierte Daten austausch (analog Telefon)
 - open
 - close
 - write
 - read
 - wait
 - accept
 - b.) gemeinsamer Speicherbereich (shared memory)
 - Vorteil: höhere Effizienz durch schnelle CPU-Befehle
 - Nachteil: vermischte Arbeitsspuren
- Synchronisationsproblematik

Benutzerschnittstelle

- dient der Kommunikation des Benutzers mit dem Rechner
- Systemprogramme sind in Benutzeroberfläche integriert

Programme

- editor
- compiler
- interpreter
- Binder (Linker)
- Lander
- Debugger

Dateien / Verzeichnisse

- anlegen
- kopieren
- umbenennen
- löschen
- drucken
- Zugriffsberechtigungen

Information

- Zeit
- Datum
- freier Speicherplatz
- CPU-Auslastung
- Nutzerlink

Kommunikation (zwischen Benutzern)

- email
- SFTP
- remote login

Hauptspeicherverwaltung

- Betriebssystem vergibt einen physischen Adressraum an Prozesse
- Prozesse verwenden logische Adressen \rightarrow Speicherdichte, Relokierbarkeit
- Betriebssystem bildet mit der MMU (memory management unit) logische in physische Adressen ab
- bei voller Hauptspeicher entscheidet ein Langzeit-Schedule, welche Adressräume bereiter/blockierter Prozesse in den Externspeicher ausgelagert werden ("swapping")
- interne Fragmentierung: nicht belegter Speicher einer zugeordneten Bereiche
- externe Fragmentierung: Bereich zwischen Adressräumen; entsteht durch ständig wechselnde Prozessanzahl

Zusammenhängende Hauptspeicherzuweisungen

- Variante 1: Hauptspeicher wird in fest zusammenhängende Bereiche unterschiedlicher Größe aufgeteilt
 ↳ jeder Prozess erhält Bereich, in dem er mindestens auch passt
- Variante 2: jeder Prozess erhält zusammenhängenden Bereich der angeforderten Größe
- Vorteil: Abbildung von logischen in physische Adressen mit Basis- und Grenzregister einfach zu realisieren
- Nachteil: externe Fragmentierung \rightarrow ineffizient, falls keiner der freien Bereiche groß genug für einen Prozess ist
 ↳ Kompaktifizierung (sporadisches Zusammenstoßen der Adressräumen) ist sehr aufwändig

Nicht-zusammenhängende Hauptspeicherzuweisungen

- Variante 1: Segmentierung / Segmentation
 - ↳ unterteilt Hauptspeicher in ggf. unterschiedlich lange, zusammenhängende Segmente
 - ↳ Prozess bekommt als Adressraum mehrere Segmente
 - ↳ Adressierung über Segmentnummer; innerhalb des Segments über Basis- und Grenzregister
- Variante 2: Paging
 - ↳ Hauptspeicher wird in viele gleichgroße Stücke (Seitenrahmen / frames) aufgeteilt
 - ↳ logischer Speicher (Speicher vom Benutzer- / Programmierericht) wird in gleichgroße Stücke (Seiten / pages) aufgeteilt
 - ↳ Prozess erhält erforderliche Anzahl von Seiten
 - ↳ eine Seite passt genau in einen Seitenrahmen
 - ↳ logische Adresse / Seite wird durch Seitentabelle auf physische Adresse / Seitenrahmen abgebildet
 - ↳ logische Adresse besteht aus Seitennummer und Offset
 - ↳ Seitennummer wird mit Seitentabelle durch zugeordneten Seitenrahmen erzeugt und bildet mit Offset die physische Adresse
 - ↳ Offset ist Position eines Wortes innerhalb einer Seite bzw. Seitenrahmennummer
 - ↳ Betriebssystem legt Seitengröße unabhängig von einzelnen Prozessen fest und konfiguriert die MMU
 - ↳ Seitentabelle ist Teil des Prozesskontexts
 - ↳ einige Einträge werden im schnellen Speicher gehalten
- Vorteil: keine externe Fragmentierung
- Nachteil: interne Fragmentierung nicht zu vermeiden

Virtueller Hauptspeicher (virtual memory)

- physischer Speicher wird in Seitenrahmen unterteilt
- Es werden nur gerade benötigte Informationen im Hauptspeicher gehalten
- ein Prozess wird rechnend gemacht, auch wenn nicht alle Seiten in einem Seitenrahmen stehen
- in der Seitentabelle werden die vorhandenen Seiten mit einem present-bit markiert
- beim Zugriff auf eine Seite ohne Seitenrahmen löst die MMU eine Software-Unterbrechung aus (Seitenteller)
 - ↳ Seitenrahmeneingabe (demand paging)

- Strategien zum Ausfall von ausgelagerten Seiten

- ↳ optimale Strategie: Seite auslagern, die am weitesten in der Zukunft benötigt wird
 - ↳ Information meist nicht verfügbar
- ↳ LRU (least recently used): lösigt am längsten ungenutzte Seite aus
- ↳ legt vorzugsweise seit dem Einladen ungenutzte Seite aus
 - ↳ Empf. Schreiboperation
 - ↳ verdeckte Änderungsinformation über ein dirty-bit in der Seitentabelle

Exkurs Hauptspeicherverwaltung Linux

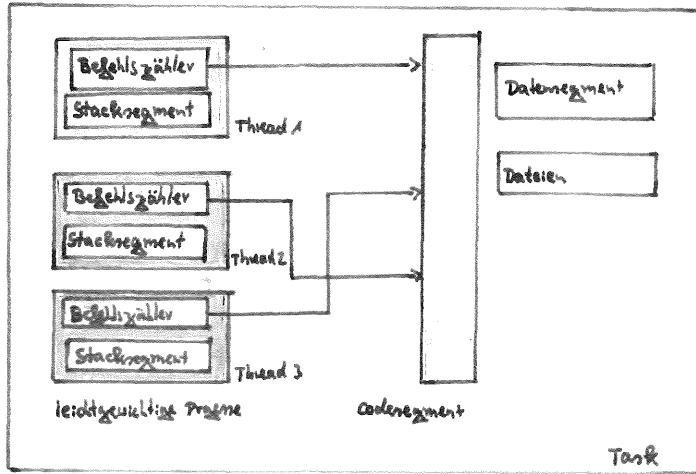
KE2

- Seitenallokator stellt Bereiche aufeinander folgender Seitenrahmen bereit
- Buddy - Strategie:
 - ↳ zusammenhängende Bereiche à 2^x Seiten, die entweder frei oder belegt sind
 - ↳ Prog. wird das kleinste freie passende Stück zugeteilt
 - ↳ Falls Stück trotzdem doppelt so groß wie notwendig → Splitte Stück von seinem Buddy (halbieren)
 - ↳ Falls Stück und Buddy beide wieder frei → verschmelzen
- Vorteil: recht einfach zu implementieren
- Nachteil: erhöhte interne Fragmentierung möglich (z.B. Prog. benötigt 33 Seiten, belegt aber im Stück 64 Seiten)

Prozesse und Prozessynchronisation

Tasks und leichtgewichtige Prozesse

- leichtgewichtiger Prog.: eigenständiger Prog., der sich Programm, Adressraum und Dateien mit anderen Prog. teilt
 - ↳ Prog. hat eigene Registerinhalte (insbesondere Befehlszähler) und eigener Stack
 - ↳ Vorteil: schneller Progr.-wechsel, da Adressraum nicht geteilt werden muss
- Task: Menge zusammengehöriger leichtgewichtiger Prozesse



- Variante 1: Kernel - Threads

- werden im Betriebssystem Kern realisiert
 - ↳ Schwer- und leichtgewichtige Prozesse werden gleich behandelt
 - ↳ Progr.-wechsel und Scheduling erfolgt im Kern
- Variante 2: Benutzer - Threads
 - leichtgewichtige Prozesse werden innerhalb eines schwergewichtigen Prog. realisiert
 - System ruft auf einen leichtgewichtigen Prog. wird durch Aufruf einer Bibliotheksfunktion erreicht
 - ↳ Falls Suspendierung des leichtgewichtigen Prog. notwendig ist, vertauscht Prog. die Registerinhalte durch die Inhalte eines anderen bereits leichtgewichtigen Prog.
 - Betriebssystem weiß nichts von dem Task
 - Vorteile: → Scheduling vom Benutzer steuerbar
 - sehr kurze Umschaltzeiten zwischen leichtgewichtigen Prozessen
 - Nachteile: blockierender Systemaufruf blockiert gesamten Task

Prozessynchronisation

- Prog. können gemeinsame Ressourcen nutzen (z.B. gemeinsame Hauptspeicherbereiche zum Nachvorteil austausch)
- ↳ Zugriff auf Ressource muss reguliert werden
- ↳ Programmabschnitte, die auf Ressourcen zugreifen, auf die auch andere Prog. zugreifen können, heißen kritischer Abschnitt bzw. atomar / unteilbar
- ↳ Prozessynchronisation garantiert Threads exklusiven Zugriff auf kritische Abschnitte

- Variante 1: Synchronisationsvariable

- beim Betreten des kritischen Bereich wird die genannte Synchronisationsvariable genutzt, beim Verlassen zurückgelegt.
- ist der kritische Abschnitt belegt, so warten andere Prog. geschäftig
- ↳ Nachteile:
 - CPU-Zeit wird verschwendet
 - potentieller Deadlock: wird ein Thread im kritischen Abschnitt beendet, so bleibt der Abschnitt für alle anderen Prog. blockiert

- Variante 2: Semaphore (Dijkstra)

- lädt Probleme, wenn mehrere Prog./Threads gleichzeitig n vorhandene Betriebsmittel belegen wollen
- Semaphore ist ein abstrakter Datentyp (S Zählvariable für Anzahl freier Betriebsmittel,
 - W Meng aller auf Betriebsmittel wartende Prog.,
 - Operationen down/up zum in- und abdecrementieren von S
- Semaphore - Operationen werden im Betriebssystem implementiert
 - ↳ Unterbrechungseingang der CPU kann für atomare Operationen deaktiviert werden

Dateisysteme

- Datei := Folge von Datensätzen mit zusammengehöriger Information
- Datei wird durch Dateinamen für Nutzer kenntlich
- Dateien werden in Verzeichnissen zusammengefasst
- Benutzer befindet sich zu jedem Zeitpunkt in einem Arbeitsverzeichnis

Objektinformation:

1 drwxrwxr--	2 mueller	bteam	1024	Jul 27 16:43	archiv
↑ Angall Zugriffsberechtigungen	↑ Besitzer	↑ Arbeits- gruppe des Besitzers	↑ Objektgröße in Bytes	↑ letzter schreibender Einzugriff	↑ Objektname

Zugriffsberechtigung

- Systembenutzer werden in Klassen eingeteilt
 - user (Objektbesitzer)
 - group (Arbeitsgruppenmitglieder)
 - other (sonstige Systembenutzer)
- Zugriffssarten werden in Klassen eingeteilt
 - read
 - write (schreiben, überschreiben, löschen)
 - execute (bei Verzeichnis kann es als Heimverzeichnis genutzt werden.)

d	<u>rwx</u>	<u>rwx</u>	<u>r--</u>
↑ directory	↑ Rechte Präfix: bei Datei:	↑ Rechte user	↑ Rechte group

Unix - Kommandos

- pwd (print working directory)
- cd (change directory)
- ls (liste Objekte auf)
- chmod (change mode - Rechte ändern)
- touch (neue Datei mit 0 Byte Länge anlegen)
- mkdir (neues Verzeichnis anlegen)
- rm (Datei löschen)
- rmdir (Verzeichnis löschen)
- cp (Datei Kopieren)

Interne Dateisystemstruktur

- Datei := Folge gleich großer Blöcke

Variante 1: Speicherung in aufeinander folgenden Blöcken

- ↳ Vorteile: minimale Zugriffszeit bei Magnetplatte, leichte Berechnung der Blocknummer bei wahlfreiem Zugriff
- ↳ Nachteil: externe Fragmentierung (Zusammenführen der Blöcke sehr aufwendig)

Variante 2: Speicherung in nicht aufeinander folgenden Blöckennativ: verketzte Liste

- verketzte Datenblöcke zu einer Liste
- ↳ Datenname enthält physische Adresse des ersten Blocks
- ↳ Blockende enthält physische Adresse des nächsten Blocks
- Vorteil: nicht zusammenhängende Speicherung wird möglich, externe Fragmentierung wird vermieden
- Nachteil: kein wahlfreier Zugriff

File-Allocation Table / FAT (MS-DOS)

- verketzte die physische Blockadressen → FAT := array [0..MaxBlockNr-1] of BlockNr (enthält alle Blocknummern der Magnetplatte)
 - ↳ ggf. existiert für Teilsystem eine eigene FAT
- Datenname enthält physische Adresse i von Dateiblock 0
- FAT[i] enthält physische Adresse j von Dateiblock 1
- FAT[j] enthält physische Adresse von Dateiblock 2 ...
- letzte Dateiblock L gilt FAT[L] = end of file (eof)
- FAT wird in konsekutiven Blöcken im Externspeicher abgespeichert (sollte zur Laufzeit in den Hauptspeicher passen. → Dateiblock i durch i Hauptspeicherzugriffe erreichbar)

inode (Unix)

- Für jede Datei / Verzeichnis existiert ein inode
- inode := (Attribute, physische Adressen der ersten 12 Datenblöcke, einfacher Index [physische Adresse vom Block mit Adresse der nächsten logischen Datei-Nachk.], zweiter Index [physische Adresse eines freien Indexblocks, der noch einfache Indexblöcke enthlt], drei Zeiger (Index))
- Vorteil:
 - alle wesentlichen Informationen auf kontinuierl. Raum
 - nur vorhandene Objekte belegen Speicherplatz
 - Zugriff auf ersten Datenblock sehr effizient
- Nachteil:
 - keine Information über freie Speicherblöcke (→ Blockübersicht notwendig)

Allgemein

- Rechnernetze := über Kommunikationsleitungen (geführt oder ungeführte physische Medien) miteinander verbundene Rechner (Hosts / Endsysteme)
 - ↳ Anwendungsprogramme, die auf unterschiedlichen Hosts laufen, können mittels Nachrichtenwechsel über das Netzwerk miteinander kommunizieren
 - ↳ Anwendungen werden verteilte Anwendungen oder Netzwerkapplikationen genannt
- Endsysteme können über Transitsysteme (z.B. Router) mit einander verbunden werden
- Rechnernetze unterschiedliche Konfiguration / Technologie können miteinander verbunden werden
- Rechnernetze werden im Schichtmodell realisiert
- Route / Pfad := Weg der Information vom Quell- zum Zielhost
- Übertragungsrate / Bandbreite := Kenngröße der Leitung (übertragene Bits pro Sekunde). Wird im Dualsystem angegeben: $1 \text{ Kbps} = 10^3 \text{ bps}$, $1 \text{ Mbps} = 10^6 \text{ bps}$

Gründe für das Vernetzen von RechnernBetriebsmittel - oder Funktionsverbund (resource sharing)

- Betriebsmittel (Programme, Daten, Geräte) sollen allen Hosts des Netzwerks zugänglich sein
- bei Fokus auf Datenzugriff spricht man auch von einem Datenverbund (data sharing)

Last - oder Leistungsverbund

- Rechenaufwand gleichmäßig auf mehrere Rechner verteilt werden
- Mehrere Rechner sollen eine einzige umfangreiche Aufgabe lösen

Wartungsverbund

- zentrale Wartung und Störungsbekämpfung für viele räumlich verteilte Rechner

Computerunterstützte Gruppenarbeit (CSCW)

- Kommunikation und Kooperation menschlicher Benutzer unterstützen

Netzwerkarten

- Lokale Netze (Local Area Network / LAN): wenige Kilometer Ausdehnung
- Regionale Netze (Metropolitan Area Network / MAN): ca. 100 Km Ausdehnung
- Weltverkehrsnetze (Wide Area Network / WAN): Länder/Kontinente; im Verbund global

Netzwerkkomponenten

- Netzwerktopologie := Netzwerkkomponenten und ihre Verbindungen
- Netzwerkkern := Menge der verbundenen Router ohne Endsysteme
 - ↳ Ansätze zum Aufbau des Netzwerkkerns: Leistungsermittlung oder Paketvermittlung (Siehe 7^o Kommunikation)

Netzwerkkomponenten:↳ Endsysteme (Hosts)↳ Transitsysteme

- ↳ Switches / Bridges

- ↳ Gateways: verbinden Netze, die verschiedene Routing-Protokolle nutzen

- ↳ Router:

- nimmt in einem Netzwerk Informationen an und sendet sie über Ausgangsleitungen weiter

- Speicherung: Router kann Paket ent vereinen, wenn gesamtes Paket empfangen wurde ("Store-and-Forward")

- ↳ Paket wird verworfen, wenn Warteschlange des Routers voll ist

- ↳ Routerleistung wird oft als Wahrscheinlichkeit eines Paketverlusts angegeben

KommunikationVermittlungsarten↳ Leistungsermittlung

- für jede Kommunikation wird eine persistente Verbindung aufgebaut und auf dem Pfad zwischen Sender und Empfänger die Ressourcen für die Sitzgabare reserviert

↳ Paketvermittlung

- jede Nachricht verwendet Ressourcen nach Bedarf → Warteschlangen möglich
- Quellhost sendet Nachrichten als Paketfolge
- erlaubt gleichzeitige Nutzung von (Teil-)Pfaden durch mehrere Endsysteme
- Vorteile:
 - Zwischenstationen arbeiten parallel an der Übertragung
 - bei Übertragungsfehlern müssen nur die betroffenen Pakete erneut versandt werden
 - ⇒ höherer Durchsatz
- Nachteil: - höherer Header-Overhead

↳ Paketvermittlungsklasse 1: Datagramm - Netzwerk

- Pakete werden anhand von Hostzieladressen weitergeleitet
- Pakete fließen vom Sender-Host durch Router zum Empfänger-Host
- Router praktizieren Spezervermittlung (siehe \rightarrow Netzwerkkomponenten, Transitsysteme)

↳ Paketvermittlungsklasse 2: VC-Netzwerke („Virtual channels“)

- Pakete werden anhand virtueller Kanalnummern weitergeleitet

Verbindungsarten

↳ nicht-persistente Verbindungen

- Verbindung wird Lieferung der Nachricht sofort wieder abgebaut

↳ persistente Verbindung

- Sender bindt Verbindung nach Wunsch ab

↳ mit Pipelining:

- Sender kann Nachrichten an Empfänger senden ohne auf Antworten zu warten
- bessere Auslastung und schneller Zugriff auf Antworten

↳ ohne Pipelining:

- Sender wartet vor dem Senden weiterer Nachrichten auf Antwort des Empfängers

Kommunikationsverzögerungen

↳ Verarbeitungsverzögerung

- Feststellen, wohin Paket weitergeleitet werden soll
- z.B. Entdecken von Bitfehlern

↳ Warteschlangenverzögerung

- Wartezeit des Pakets in den Warteschlangen der Routers

↳ Übertragungsverzögerung

- Einstellen aller Paketbits im Übertragungsmedium (Paketbits / Übertragungsrate Kommunikationschnittstelle zum physischen Medium in bps)

↳ Ausbreitungsverzögerung

- Ausbreitungszeit eines Bits von Router A nach Router B nach Einstellen im Übertragungsmedium ($d(A,B) / \text{Ausbreitungsgeschwindigkeit}$)

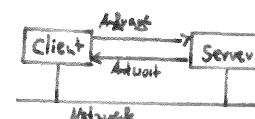
\Rightarrow Gesamtverzögerung für die Paketübertragung von Router A nach Router B

\Rightarrow Ende-zu-Ende-Verzögerung des Pakets = Summe Seriativverzögerungen aller Router des Pfades

Kommunikationsarchitektur für Anwendungen

↳ Client-Server-Modell

- Programme werden nach Server- und Clientprogrammen unterschieden
- ↳ Serverprogramm: implementiert einen Dienst und bietet ihn Client-Programmen an
- ↳ Clientprogramm: fordert Dienst vom Server-Programm an und wartet auf Antwort
- Client ist üblicherweise im Anwender eingeschlossen



↳ Peer-to-Peer-Modell

- Kommunikationsteilnehmer können Dienste anfordern und anbieten
- ↳ Reine zweiseitige Trennung von Server und Client

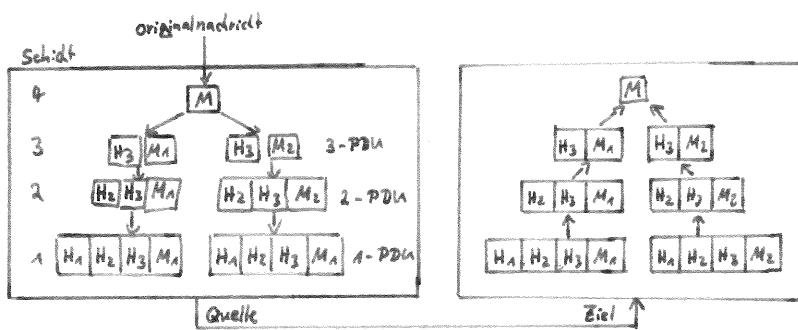
Protokolle und Protokollsichten

- Protokoll := Menge von Regeln bzgl. Konventionen für die Kommunikation zwischen Parteien. Definiert werden

- Nachrichtentypen - Syntax - Semantik - Regeln zum Zeitpunkt, Art und Realität von Nachrichtenempfang bzgl. - versand

geschichtete Protokollarchitektur

- Kommunikation zwischen Sender und Empfänger wird in n Schichten realisiert
- Jeder Schicht werden Schicht-n-Protokolle zugeordnet, alle Protokolle bilden den Protokollstapel
- Quell- und Zielhost kommunizieren auf Schicht n über Schicht-n-Nachrichten, deren Inhalt und Format durch das Schicht-n-Protokoll definiert wird
- ↳ Schicht-n-Nachrichten = Protokolldateneinheiten der Schicht n" bzw. n - PDUs (Protocol Data Units)
- ↳ physische Übertragung geschieht nur in der untersten Schicht
- Schicht n bereikt empfangene Daten für obere Schicht (bzw. erhalten Daten für die untere Schicht auf)
- ↳ Hinzufügen von Headern/Tailern (enthalten Zählinformationen) zu n - PDUs bzw. Entfernen
- ↳ Daten in Pakete zusammenfassen bzw. in kleinere Pakete aufteilen
- ↳ Nachrichten zerlegen (disassemblieren) oder zusammenfügen (reassemblieren)



ISO/OSI - Referenzmodell

- Standardisierungsrahmen zur Kommunikation zwischen Computersystemen → Framework zur Implementierung von Protokollen
- definiert sieben Schichten

Schicht 1: Bitübertragung (physical layer)

physischer Übertragungskanal für beliebige Bitfolgen

Schicht 2: Sicherungsrichtschicht (data link layer)

weitgehend sichere Übertragungskanäle für Datenblöcke mit Fehlererkennung und -korrektur

Schicht 3: Vermittlungsschicht (network layer)

logische Übertragungskanäle zwischen Endsystemen

Schicht 4: Transportschicht (transport layer)

logische Übertragungskanäle zwischen Anwendungsprogrammen auf den Endsystemen

Schicht 5: Kommunikations- oder Sitzungsrichtschicht (session layer)

Dialogkontrolle und -synchronisation für Anwendungsprogramme

Schicht 6: Darstellungsrichtschicht (presentation layer)

Darstellung von Daten in unterschiedlichen Repräsentationen (z.B. verschiedene Zeichenkodierungen)

Schicht 7: Anwendungsrichtschicht (application layer)

implementiert Anwendungsfunktionalität

Internet

- Internet := Verbund von privaten und öffentlichen Netzen zu einem einzigen logischen Netz mittels dem IP-Protokoll („Netzwerk aus Netzwerken“)
- hierarchische Topologie des Internets (Top-Down):
 - ↳ nationale und internationale Internet-Service-Provider (ISPs); sind miteinander verbunden
 - ↳ regionale ISPs; sind mit nationalen und internationalen ISPs verbunden
 - ↳ lokale ISPs; sind mit regionalen ISPs verbunden
 - ↳ Endsysteme; sind über Zugangsnetzwerke an lokale ISPs angeschlossen
- Basis des Internets: Internet-Standardisierungsdokumente (RFC = „Request for Comments“) der IETF („Internet Engineering Taskforce“)

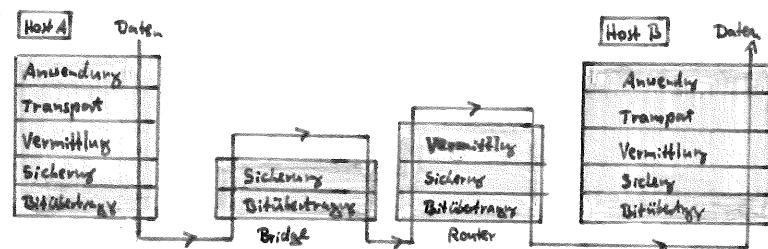
Internet - Schichtenmodell

Schicht	Layer	PDU	Protokolle (Beispiele)	Kommunikationspartner	Anmerkungen
5	Anwendung	Application	Nachricht	HTTP, SMTP, IMAP, FTP	Anwendungen
4	Transport	Transport	Segment	TCP, UDP	Programme
3	Vermittlung	Network	Datagramm	IP, Routing-Protokolle	Hosts
2	Sicherung	Data Link	Rahmen	Ethernet, PPP	Nachbarhost
1	Bitübertragung	Physical	1-PDU		Überträgt Bits der 1-PDU von einer Route zum nächsten

- Schicht 5 heißt auch „Verarbeitungsschicht“
- Schicht 3 heißt auch „IP-Schicht“ oder „Internet-Schicht“
 - ↳ IP definiert Format eines IP-Datagramms
 - ↳ Routing-Protokole bestimmen die Route der Datagramme zwischen Quell- und Zielhost; innerhalb eines Netzwerks beliebiges Routing-Protokoll
- Schicht 1 und 2 wird auch „Netzwerkschnittstellenrichtschicht“ oder „Host-an-Netz-Schicht“ genannt

- Netzwerkkomponenten:

- ↳ Hosts implementieren alle Schichten
- ↳ Routen implementieren Schichten 1-3
- ↳ Switches/Bridges implementieren Schichten 1-2
- ⇒ Ziel: Komplexität an Netzwerkgerüsten verlagern

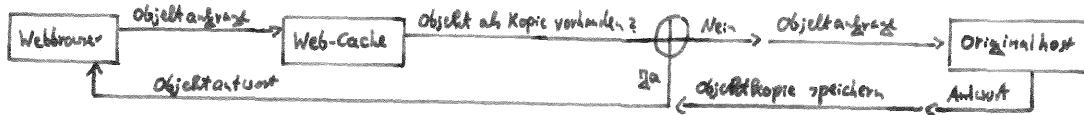


World Wide Web (WWW)

- WWW gebräuchlichste Netzwerkverwendung: ermöglicht den Zugriff auf Dokumente von anderen Endsystemen
- WWW != Internet → WWW kann auf anderem Netzwerk betrieben werden. WWW nur eine Anwendung von vielen im Internet
- Komponenten des WWW
 - ↳ Benutzeroberfläche: Webbrowser
 - ↳ HTML (HyperText Markup Language): Standarddokumentenformat zum Anzeigen im Browser
 - ↳ Web-Server: halten Dokumente bereit
 - ↳ HTTP (HyperText Transfer Protocol): definiert Nachrichtentypen und Weiterleitungswarten zwischen Web-Browsern und Web-Servern
- Objekte im WWW (Webseiten, Bilder,...) haben ein eindeutig URL (Uniform Resource Locator)
 - ↳ besteht aus Protokoll, DNS-Namen / IP-Adresse des Rechners und lokal eindeutigen Seitennamen
 - ↳ Namensschema: < protocol id > : < protocol specific address >

Web-Caches

- Web-Cache := Webserver, der Kopien der zuletzt angeforderten Objekte speichert
 - ↳ Web-Cache fungiert als Proxy-Server (Vermittlungsrechner, nimmt Anfragen an, fordert Antworten an und leitet diese weiter)
 - ↳ Web-Cache speichert keine Client-Information
 - ↳ Web-Cache operiert als Server und als Client



- Vorteile einer Web-Cache
 - ↳ geringere Reaktionszeit für Client-Anfragen (z.B. Hochgeschwindigkeitsleitung zwischen Cache und Host)
 - ↳ geringerer Internetausgangsverkehr für Organisationen
 - ↳ Anonymisierung der Hosts durch vertrauenswürdigen Proxy-Server

Domain Name System (DNS)

- Problem: Nutzer verwenden mnemonische Hostbezeichner (z.B. www.gernunihagen.de), Router verwenden IP-Adressen zum Weiterleiten von Nachrichten
- DNS bildet mnemonische Hostnamen in IP-Adressen ab
- DNS ist ein Protokoll der Anwendungsschicht und wird von anderen Protokollen der Anwendungsschicht zum Ermitteln von IP-Adressen für Benutzereingaben (Hostnamen) verwendet
- DNS implementiert eine hierarchische, verteilte Datenbank:

Iterative Namenauflösung:
 antworten auf Anfragen mit Verweis auf anderen/niedrigeren Server, an diesen ist Anfrage erneut zu stellen

Root-Name-Server
 ↳ Rennen mindestens alle TOP-Level-Name Server

Top-Level-Name-Server
 ↳ sind für Domains wie z.B. .de zuständig
 ↳ Rennen innerhalb ihrer Domains die autoritativen Name-Server

Rekursive Namenauflösung:
 übernehmen Hostanfragen und liefern endgültige Antwort zurück

Authoritative Name-Server
 ↳ Rann alle Anfragen zu einer bestimmten Domain beantwortet
 ↳ wird vom jeweiligen Betreiber einer Second-Level Domain (z.B. www.gernunihagen.de) zur Verfügung gestellt

lokaler Name-Server
 ↳ verarbeitet DNS-Anfragen von Clients zu erledigen und Antworten zu senden
 ↳ kann zeitgleich als authoritative Name-Server fungieren

- Alle Server arbeiten zur Entlastung mit einem Cache

Anwendungsschicht

- Netzwerkapplikationen sind von Protokollen der Anwendungsschicht zu unterscheiden
- Architekturen für Netzwerkapplikationen: Client-Server oder Peer-to-Peer

Anwendung	Protokoll
World Wide Web	HTTP
E-Mail	SMTP, IMAP, POP3
File Transfer	FTP
News	NNTP
Remote Login	Telnet
Streaming Multimedia	HTTP, RTP, RTMP
Internet - Telefonie	SIP, RTP, proprietär (z.B. Skype)

Häufig benutzte verteilte Anwendungen mit zugehörigen Anwendungsschichtprotokollen

Protokolle

HTTP (HyperText Transfer Protocol)

- Zustandsloses Datenübertragungsprotokoll der Anwendungsschicht
- nutzt TCP auf der Transportschicht
- kann mit nicht-persistenten und ab Version 1.1 auch mit persistenten Verbindungen arbeiten
- definiert HTTP-Auftragsnachricht und HTTP-Antwortnachricht

↳ Auftragsnachricht

GET < Dateiname > < HTTP-Version > } Auftragszeile
 Host: < Webserver > oder < Host-ID >
 Connection: < open, close >
 User-Agent: < Browser, Version >

} Headerzeilen

↳ Antwortnachricht

< HTTP-Version > < Statuscode > < Phrase > } Statuszeile
 Connection: < open, close >
 Date:
 Server:
 Last-Modified:
 Content-Length:
 < Entity Body >

} Headerzeile

} enthält die Objektdatei

FTP (File Transfer Protocol)

- Benutzer können Dateien von oder zu einem entfernten Host übertragen
- FTP nutzt zur Kommunikation zwischen Client und Server zwei TCP-Verbindungen

↳ Steuerverbindung

- bleibt für die Sitzungsdauer bestehen
- Navigation durch Verzeichnisstruktur
- Dateitransfer anstreben

↳ FTP sendet Steuerinformation „out-of-Band“

↳ Datenverbindung

- wird nur für den Dateitransfer aufgebaut und anschließend wieder abgebrochen

- Anzahl der Sitzungen ist limitiert, da FTP Zustand aller Sitzungen verwalten muss

↳ FTP ist ein Protokoll mit Zustand

E-Mail

- ursprünglich nur SMTP (Simple Mail Transfer Protocol)

↳ einfaches Protokoll zum Mailen und zwischen Mail-Servern mittels TCP mit Wert auf Zuverlässigkeit bei Anfall von Leistungen / Hosts

↳ Einschränkung: Nachrichten bestehen nur aus 7-Bit-ASCII-Zeichen

↳ Zum Darstellen von Sonderzeichen wurde Kodierungstandard MIME (Multipurpose Internet Mail Extensions) eingeführt

↳ SMTP ist Push-Protokoll: Mails werden unabhängig von empfangenden Server geschickt

- Mail-Server üblicherweise von ISPs betrieben

↳ gängiger Prozess MTA (Message Transfer Agent)

↳ nimmt Mails entgegen, speichert in lokale Mailboxen oder leitet per SMTP an andere Web-Server weiter

↳ Server agiert i.d.R. gleichzeitig als Client und als Server

- weitere Protokolle

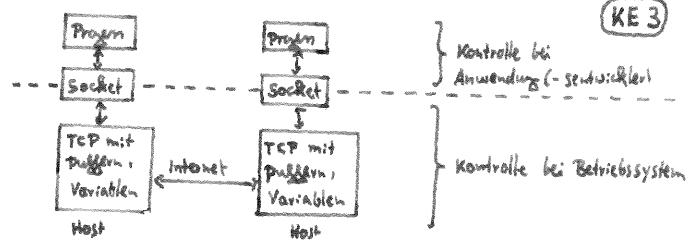
↳ POP 3 (Post Office Protocol Version 3): Holt Mails vom Server auf eigenen Rechnern

↳ IMAP (Internet Message Access Protocol): zentrale Mailverwaltung

- Nutzer verwenden i.d.R. ein mächtiges Programm (MUA / Mail User Agent)

Schnittstelle zur Transportschicht

- Anwendungssektor nutzt Dienste der darunterliegenden Schicht (Transportschicht)
- Socket dient als Zugangspunkt zum Protokoll der Transportschicht
- Schnittstelle wird auch als API (Application Programming Interface) bezeichnet



Anforderungen des Anwendungssektors an die Transportschicht

- zuverlässige Übertragung (→ Datenverlust)
- Bandbreite / Übertragungsrate
 - ↳ breitband sensiv: geringe Bandbreite führt zu Problemen
 - ↳ elastisch: geringe Bandbreite innerhalb einer Toleranz akzeptabel
- Zeitsensibilität: akzeptable Ende-zu-Ende - Verzögerung

Anwendungen	Datenverlust	Bandbreite	Zeitsensibel
Filetransfer	Rein Verlust	elastisch	nein
E-Mail	Rein Verlust	elastisch	nein
Web-Transfer	Rein Verlust	elastisch	nein
Echtzeitaudio	verlusttolerant	wenig Kbps - 1Mbps	einige 100 ms
Echtzeitvideo	verlusttolerant	10 Kbps - 5 Mbps	einige 100 ms
Kommunikative Audio/Video	verlusttolerant	mit Echtzeitaudio und Echtzeitvideo	wenige Sekunden

Transportschicht

Allgemein

- Transportschicht ermöglicht die logische Kommunikation zwischen Anwendungsprogrammen auf unterschiedlichen Endsystemen
 - ↳ erweitert Host-zu-Host Übertragung zur Programm-zu-Programm-Übertragung
- Transportprotokolle nutzen Dienste der Vermittlungsschicht
- Transportprotokolle müssen Nachrichten verschiedener Progms über denselben Host senden und empfangen; da Vermittlungsschicht nur die Host-zu-Host-Kommunikation realisiert
 - ↳ Multiplexen: Kontrollinformationen zu Nachrichten der Anwendungssektoren hinzufügen
 - ↳ Demultiplexen: Nachrichtenstrom der Vermittlungsschicht auf Empfängerprogs verteilen
- Transportprotokolle werden auf Endsystemen implementiert, nicht in Routern
- Nachrichten auf der Transportschicht werden als Segmente bezeichnet

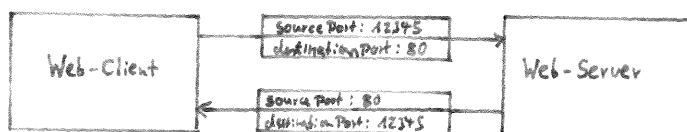
Programmnummierung

- Portnummer
 - ↳ besteht aus 16 Bit → jeder Host besitzt 65.536 Ports für potentiell ansprechbare Progms
 - ↳ Portnummern 0 - 1023 sind für bekannte Anwendungsprotokolle reserviert
- IP-Adresse
 - ↳ ist 32 Bit lang und wird als 4 Byte interpretiert, z.B. 132.176.71.2
 - ↳ IP-Adressen sind global eindeutig
 - ↳ werden in Netzwerk- und Host-Teil unterteilt

- Netzwerkverbindungen müssen Transportschicht Portnummern des empfangenden Progms auf dem Zielhost und Name/Adresse des Hostrechners mitteilen

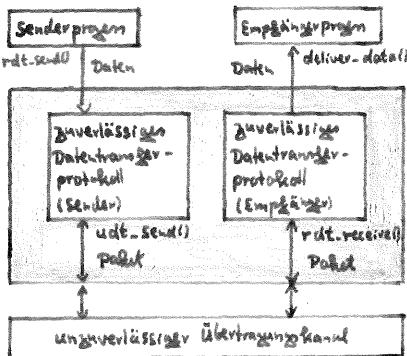
Multiplexen und Demultiplexen

- Segmente senden:
 - ↳ Multiplexen: fügt Segmentheader mit SourcePort und DestinationPort zum Segment hinz.
 - ↳ gebe Segment zur Versendung an die Vermittlungsschicht weiter
- Segment empfangen
 - ↳ Demultiplexen: stelle Segment mithilfe des Segmentheaders dem Empfängerprog. zu



Zuverlässige Datenübertragung

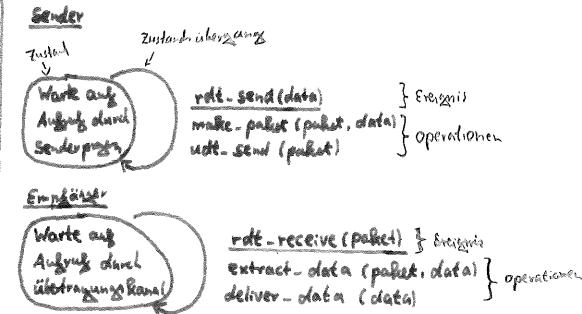
- ein zuverlässiger Datenübertragungskanal soll Daten
 - ↳ unverändert, ↳ vollständig und ↳ in der Versendungsreihenfolge zum Empfänger übertragen
- Transportschicht soll ein zuverlässiges Datenübertragungsprotokoll auf Basis der potentiell unzuverlässigen darunterliegenden Schichten realisieren
- bei unzuverlässigen Übertragungskanälen können generell folgende Probleme
 - ↳ Bitfehler: einzelne Bits einer Nachricht werden fehlerhaft übertragen
 - ↳ Paket-/Nachrichtenverlust
 - ↳ Nachrichtenübergütigung: Empfangsdatenrate des Empfängers niedriger als Sendekapazität des Senders



$rdt :=$ reliable data transfer
 $udt :=$ unreliable data transfer

Implementierung eines zuverlässigen Datenübertragungsdienstes

einfaches Datenübertragungsprotokoll als Zustandsautomat

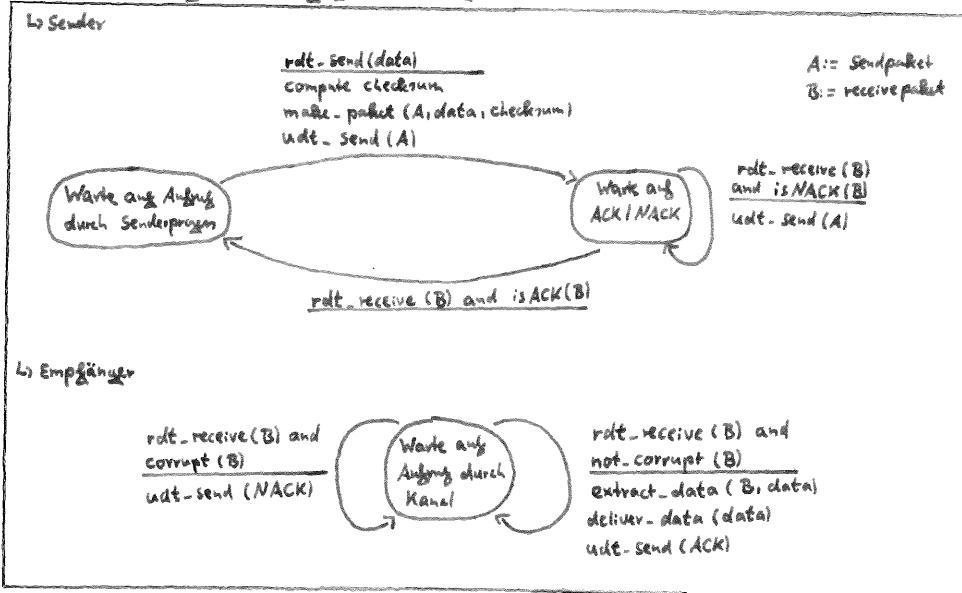


Behandlung von Bitfehlern

- allgemein: Empfänger prüft Nachricht und gibt dem Sender eine Rückmeldung; Sender wiederholt Nachricht bei gemeldeten Bitfehlern
 - ↳ Acknowledgement / ACK: positive Bestätigung, kein Bitfehler
 - ↳ negative Acknowledgement / NACK: negative Bestätigung, Bitfehler
- Literatur: ARQ-Protokolle („Automatic Repeat reQuest“) := auf Wiederholung basierende zuverlässige Datenübertragungsprotokolle
- grundsätzliche Fähigkeiten zur Bitfehlerbehandlung eines ARQ-Protokolls
 - ↳ Bitfehler erkennen: zusätzliche Informationen wie z.B. Prüfsumme übertragen und prüfen
 - ↳ Rückmeldung an Sender
 - ↳ Nachricht wiederholen

STOP-AND-WAIT-Protokoll

- Annahme: verlustfreier Übertragungskanal mit Bitfehlern



- Problem: Kollision mit Bitfehlern in den Bestätigungen.

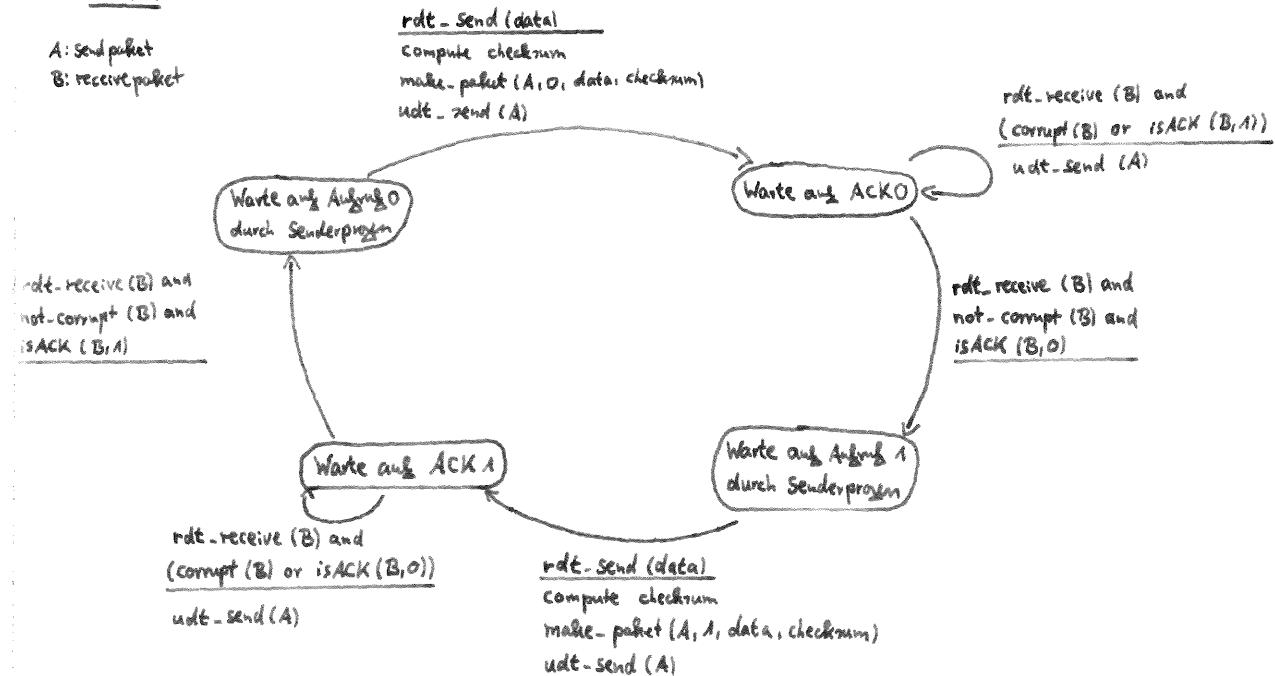
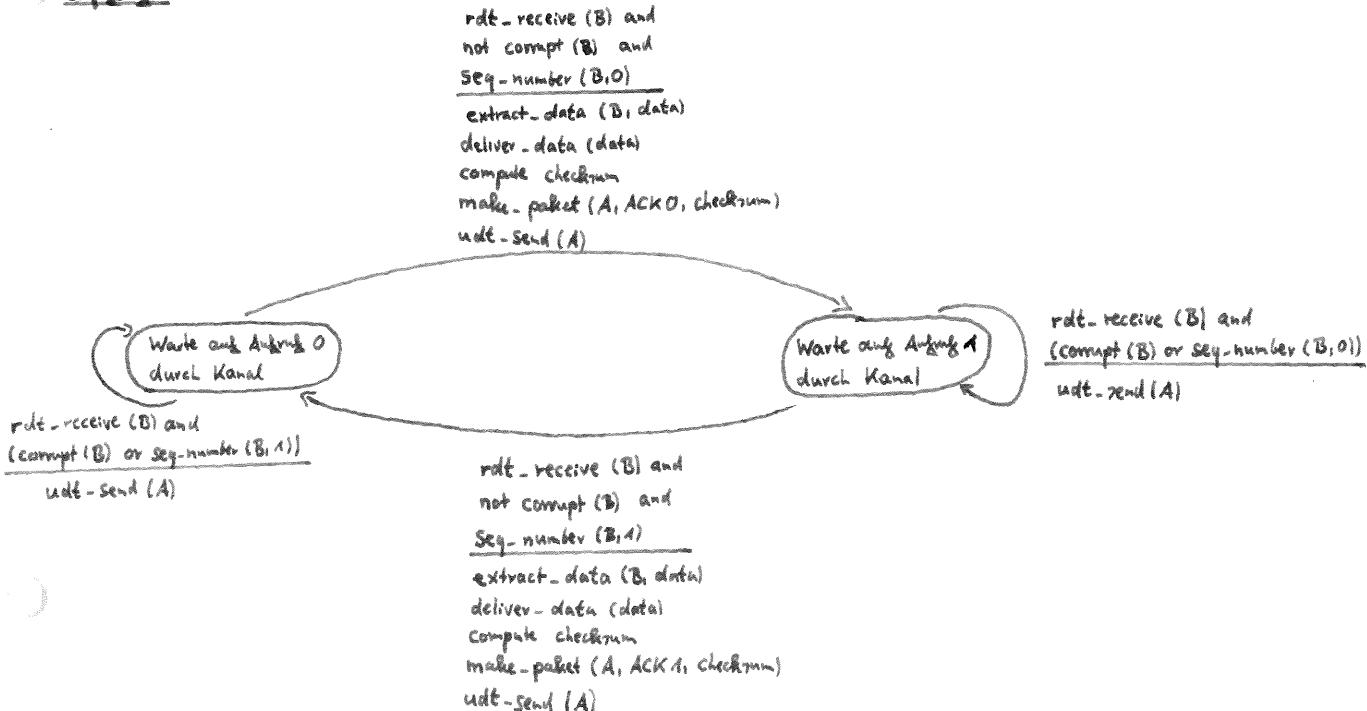
↳ Lösungsansatz 1: Bitfehlererkennung und -belebung durch Hinzufügen einer ausreichend langen Prüfsumme
 ↳ Problemlösung für verlustfreien Kanal

↳ Lösungsansatz 2:

- Sender wiederholt beim Empfang einer verzögerten Bestätigung das aktuelle Paket
- Empfänger bekommt ggf. Duplikate im Nachrichtenstrom
 - ↳ Sender nummeriert alle Pakete (Sequenznummer) → Empfänger kann Duplikate erkennen

Sender

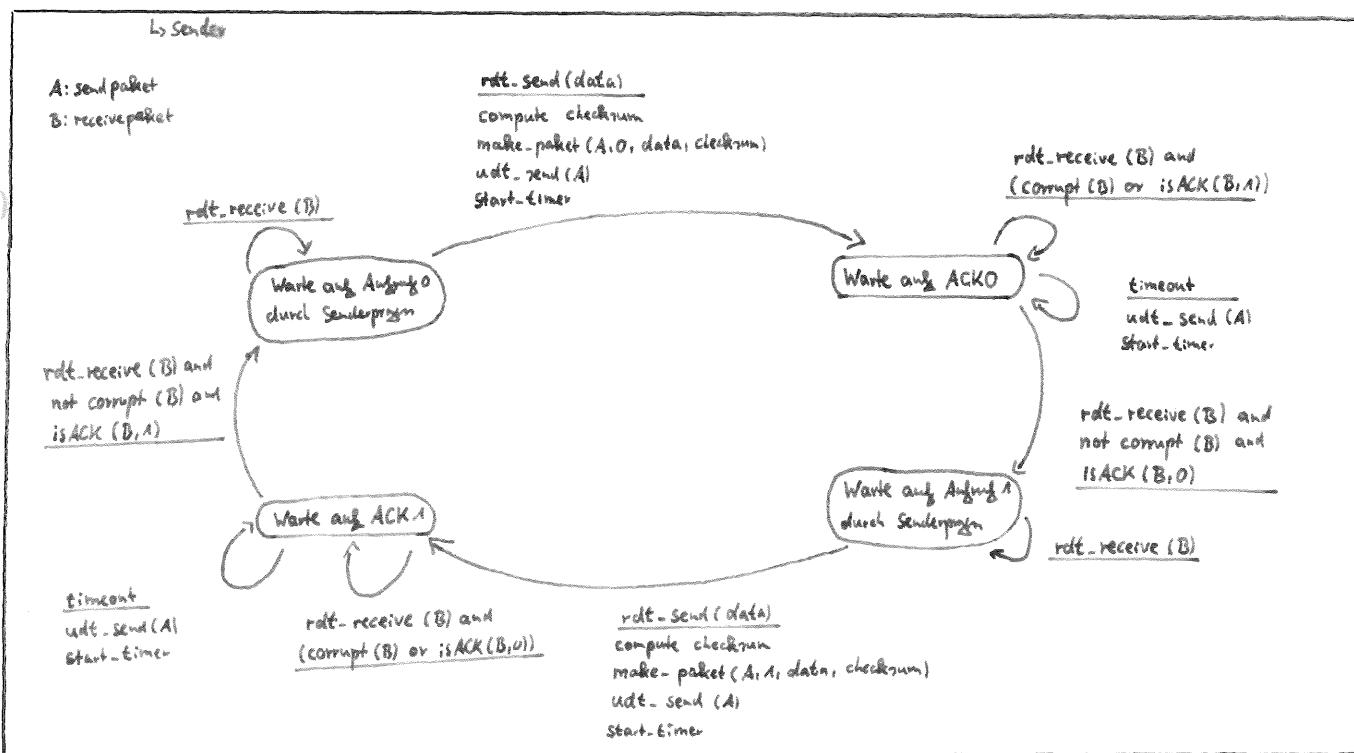
A: send-paket
B: receive-paket

Empfänger

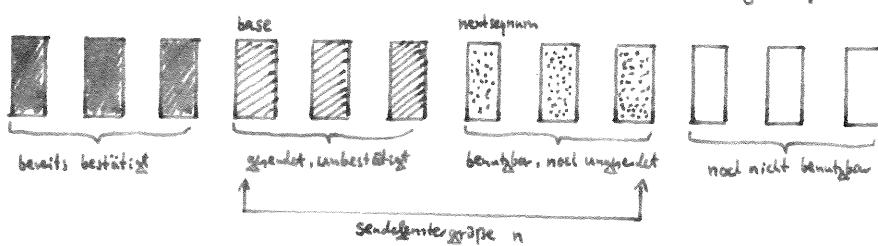
STOP - AND - WAIT - Protokoll mit Sequenznummern zwischen 0 und 1 ohne Paketverlust zur Behandlung von Bitfehlern

Behandlung von Paketverlusten

- Paketverluste können mithilfe von Prüfziffern, Sequenznummern, Benachrichtigungen (ACK) und Wiederholungen behandelt werden
- benötigte Fähigkeiten
 - ↳ Paketverlust erkennen
 - ↳ Paketverlust behandeln
- allgemein: Sender wartet einen gelegten Zeitraum auf Bestätigung des Empfängers und sendet nach Ablauf des letzten Paket erneut
 - ↳ Theorie: Wartezeit = Mindestzeit des Pakets vom Sender zum Empfänger und zurück inkl. Puffer auf Transitsystemen und Verarbeitung auf Hosts
 - ↳ Praxis: angemessene Zeitspanne wird gewählt, da Worst-case-Vergößerung nicht abschätzbar (Timeout-Intervall)
- Sender benötigt Zeitgeber (Countdown-timer)
 - ↳ paketweise zeitbar ↳ nach Ablauf Paket wiederholen ↳ bei Bestätigungseingang Zeitgeber stoppen
- Behandlung von Duplikaten
 - ↳ Empfänger schreibt Sequenznummern des jeweiligen Pakets in Bestätigungsfeld des ACK-Pakets
 - ↳ Duplikate werden verworfen

Alternating-Bit-Protokoll

- Alternating-Bit-Protokoll ist ein STOP-AND-WAIT-Protokoll
- Praxis: Einsatz von Protokollen, die mehrere Pakete vor Bestätigungseingang senden
 - ↳ Fenstergröße := Anzahl der sendbaren Pakete
 - ↳ Pakete werden nummeriert
 - ↳ Sendefenster := enthält alle Paketnummern, die aktuell gesendet werden dürfen
 - ↳ Empfänger bestätigt Paketfolge mit Nummer des letzten (korrekt empfangenen) Pakets (Kumulative Bestätigung)
 - ↳ „Fenster-Protokolle“
 - ↳ Einzigmöglichkeit des Empfängers
 - ↳ Sende zum Wiederholen des fehlerhaften Pakets und aller $n-1$ Folgepakete auf Kunden („go-back-n“ / „Sliding Window-Protokoll“)
 - ↳ Sende zum Wiederholen des fehlerhaften Pakets auf Kunden („selective-reject/repeat“)



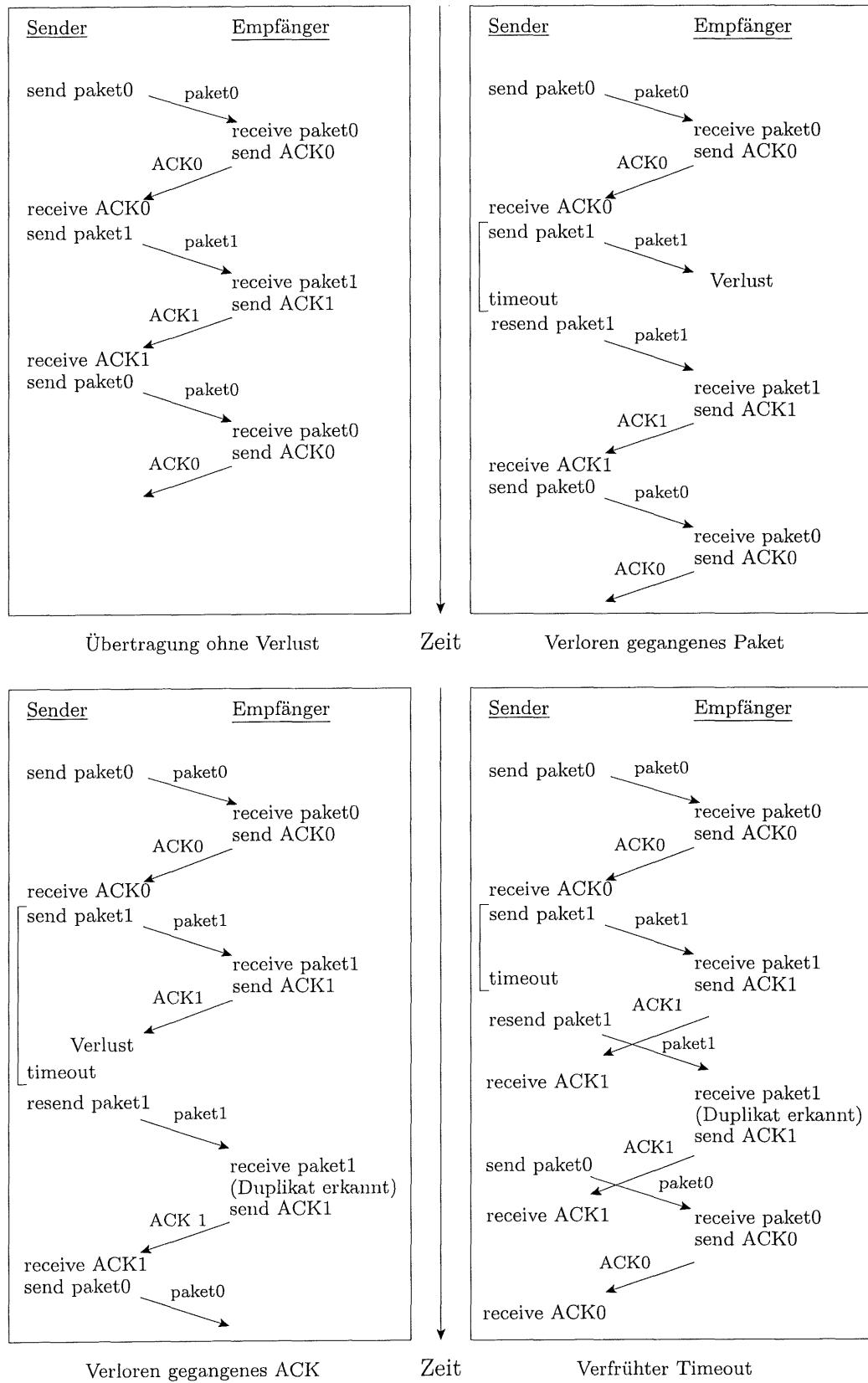
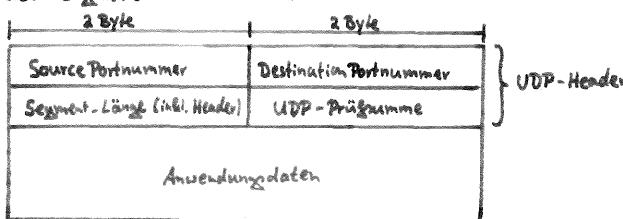


Abbildung 3.24: Typische Situationen beim Alternating-Bit-Protokoll.

UDP („User Datagram Protocol“)

- UDP ist ein Protokoll der Transportschicht und vermittelt eine Nachricht
- UDP ist ein verbindungsloses Protokoll → Prog kann UDP-Segmente ohne vorherigen Verbindungsaufbau zum Empfängerprog senden
- Nachrichten können verloren gehen
- Reihenfolge des Nachrichteneingangs ist nicht garantiert } UDP ist ein unzuverlässiger Transportdienst
- Keine Aussage über Übertragungsverzögerung
- Einsatzgebiete
 - ↳ einmalige Abfragen
 - ↳ Anwendungen in Client/Server-Umgebungen mit Fokus Schnelligkeit > Genauigkeit
 - ↳ Beispiel: DNS Abfragen
- UDP implementiert eine simple Fehlererkennung
 - ↳ Prüfsumme im Header: 1-er Komplement der Summe der 1-er Komplemente aller 16 Bit-Wörter des Segments
 - ↳ Bei Fehler wird das Segment verworfen oder mit Warnings weiterrchickt

- UDP-Segmentstruktur mit 8-Byte-HeaderTCP („Transmission Control Protocol“)Allgemein

- TCP ist ein verbindungsorientiertes Protokoll → Sender muss vor der Übertragung eine Verbindung mit dem Empfänger aufbauen.
- TCP realisiert zuverlässigen Datentransfer zwischen zwei Anwendungspingen.
- TCP vermittelt Datenstrom → Anwendungspingen müssen Beginn und Ende von Nachrichten codieren
- TCP unterstützt Vollduplex-Datenübertragung (full-duplex): Sender und Empfänger können zeitgleich Nachrichten über selbe Verbindung senden

Verbindungsmanagement

- Client-Prog und Server-Prog werden mit Socket verbunden
 - ↳ Client-Prog gibt IP-Adresse und Portnummer des Server-Progs an und TCP baut Verbindung auf

Verbindungsauflauf

- „Drei-Weg-Handschlag“ (Three-Way handshake)

↳ Schritt 1:

- Client-TCP sendet SYN-Segment an Server-TCP („Synchronize sequence numbers“)
 - ↳ enthält keine Anwendungsdaten
 - ↳ spezielles Header-Feld: SYN-Bit = 1
 - ↳ Sequenznummer-Feld enthält vom Client-TCP gewählte initiale Sequenznummer („client_isn“)

↳ Schritt 2:

- Server-TCP erzeugt Verbindung und weist Puffer und Variablen zu
- Server-TCP sendet SYNACK-Segment an Client-TCP („Synchronize sequence numbers acknowledge“)
 - ↳ enthält keine Anwendungsdaten
 - ↳ Header-Feld: SYN-Bit = 1
 - ↳ Acknowledgment-Feld = client_isn + 1
 - ↳ Sequenznummer-Feld enthält vom Server-TCP gewählte initiale Sequenznummer („server_isn“)

↳ Schritt 3:

- Client-TCP erzeugt Verbindung und weist Puffer und Variablen zu
- Client-TCP sendet Bestätigungsmitteilung
 - ↳ SYN-Bit = 0
 - ↳ Acknowledgment-Feld = server_isn + 1

↳ Verbindung etabliert, Daten können übertragen werden

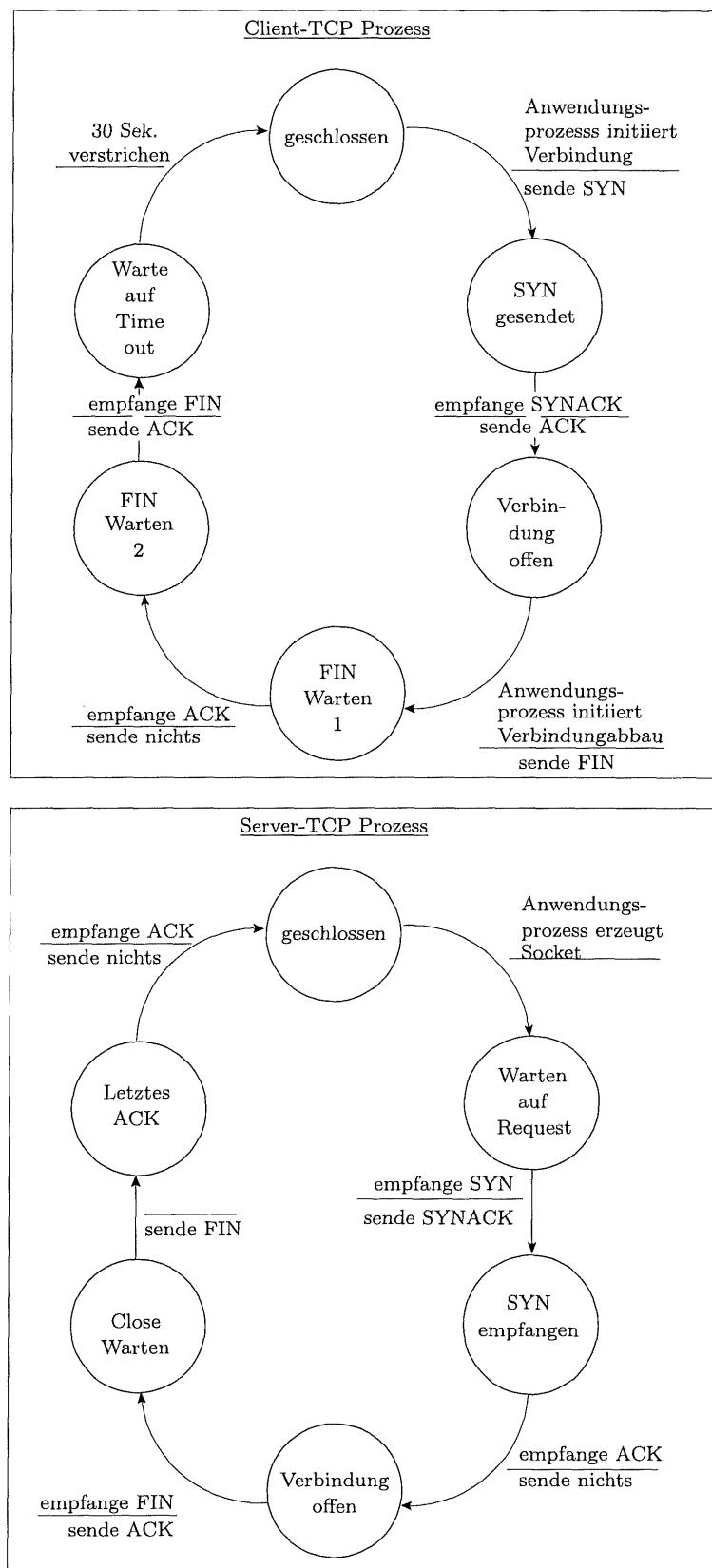


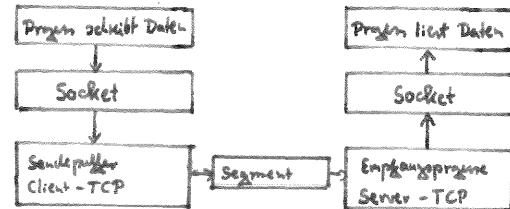
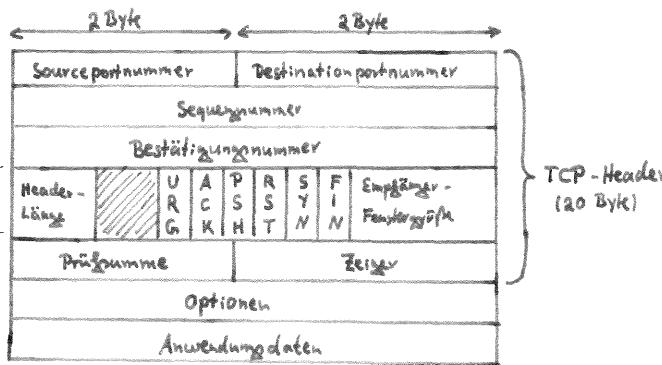
Abbildung 3.27: Zustandsübergänge von Client-TCP und Server-TCP.

Verbindungsabbau

- Client sendet ein close Kommando an Client - TCP
- ↳ Schritt 1:
 - Client - TCP sendet FIN-Segment ("finish") an Server - TCP
 - ↳ enthält keine Anwendungsdaten
 - ↳ Header: FIN-Bit = 1
- ↳ Schritt 2:
 - Server - TCP sendet ACK-Segment an Client - TCP
- ↳ Schritt 3:
 - Server - TCP sendet FIN-Segment an Client - TCP
 - ↳ Header: FIN-Bit = 1
- ↳ Schritt 4:
 - Client - TCP sendet ACK-Segment an Server - TCP
 - nach Wartezeit wird Verbindung gelöscht und Ressourcen wieder freigeben

Datentransfer

- Client - Anwendungsprogramm schreibt Daten in Socket
- Client - TCP leert Daten in Sendepuffer der Verbindung weiter
- Daten werden in Segment zusammenge stellt
- IP - Protokoll überträgt TCP - Segment als IP - Datagramm und liefert Daten an Server - TCP
- Server - TCP speichert Daten im Empfangspuffer
- Server - Anwendungsprogramm liest Daten aus Empfangspuffer



- Maximal übertragbare Bytes pro Segment ist implementationsabhängig
- ↳ sollte entsprechen zur Datagrammgröße des Vermittlungsschicht gewählt werden
- ↳ maximal 64 KB mit IP Version 4

Sequenz- und Bestätigungsnr.

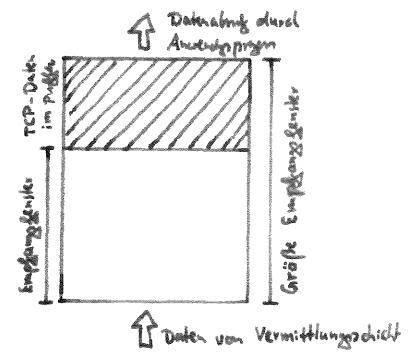
- jedes Byte im Bytestrom wird nummeriert → Bytestromnummer
- ab Sequenznummer eines Segments wird die Bytestromnummer des unteren Bytes des Segments verwendet
- TCP verwendet Bestätigungsnr. (Acknowledgment) zum Signallieren des nächsten erwarteten Bytes
 - ↳ kumulative Bestätigung der korrekt empfangenen Bytes
 - ↳ Daten nach folgendem Byte werden implementation abhängig gepuffert oder verworfen

Fehlerkorrektur

- TCP gibt jedem Segment eine eigene Prüfsumme hinz
- verfälschte Daten werden erneut übertragen
- Algorithmus
 - ↳ Prüfsummen bilden: 1-er-Komplement der Summe aller 16-Bit-Wörter des Segments;
 - ↳ Prüfsumme kontrollieren: Summe aller 16-Bit-Wörter inklusive Header

Flusskontrolle

- Empfangene Daten werden in den Empfangspuffer geschrieben und vom Anwendungsprog später geladen
- Falls Senderate > Leserate kann es zu einem Puffervöllung kommen
 - ↳ Flusskontrolldienst (Flow Control Service) ermöglicht dem Empfänger die Senderate des Senders zu verringern
 - ↳ TCP - Prog besitzt Variable namen „Empfangsfenster“ (receive window)
 - ↳ zeigt freien Pufferpunkt beim Empfänger
 - ↳ Empfänger teilt Sender in jedem Segment im Window - Feld des Headers die Fenstergöße mit
 - ↳ Fall: momentan kein Nachfrager am Sender ansteht, muss ein leeres Segment gesendet werden
- weitere Anwendung zur Überlastkontrolle (Congestion Control)



Vergleich UDP - TCP

(KE3)

UDP

- unverlässige Datenübertragung

- Vorteile:

- ↳ verbindungslos
- ↳ einfache Implementierung ohne Speicherung der Verbindungsgenügender \rightarrow Serverprozesse können deutlich mehr Clients bedienen
- ↳ geringer administrativer Aufwand: Header nur 8 Byte
- ↳ Anwendungsprozesse können soviel Daten pro Zeiteinheit senden wie sie generieren und empfangen können

- Anwendungsbereiche:

↳ Remote File service (NFS: Network File System)

↳ Streaming / Multimedia Anwendungen

↳ Internet Telefonie

↳ Network Management Anwendungen (SNMP: Simple Network Management Protocol)

↳ Routing Protocol (RIP: Routing Internet Protocol): versucht periodisch neue Routing-Tabelle, die in der Internetvermittlungsschicht zur Paketflussteuerung eingesetzt wird

↳ DNS

TCP

- zuverlässige Datenübertragung

- Vorteile:

- ↳ zuverlässig
- ↳ bidirektional
- ↳ Flusskontrolle
- ↳ entlastet Anwendungsprogrammierer von der Behandlung von Übertragungsfehlern

- Nachteile:

- ↳ verbindungsorientiert \rightarrow initiale Verzögerung bei Datenübertragung
- ↳ weniger Clients als MDP bedienbar, da Verbindungsstab geplaudert werden muss
- ↳ Flusskontrolle limitiert gleichzeitig Nachfrage

- Anwendungsbereiche:

↳ E-Mail (SMTP)

↳ Remote Terminal Access (Telnet)

↳ WWW (HTTP)

↳ File Transfer (FTP)

(23)

Vermittlungsricht

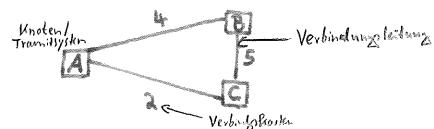
Aufgaben

- Vermittlungsricht (data link layer) transportiert Transportschichtsegmente als Datagramme zwischen Quell- und Zielhost (z.B. durch Transitsysteme)
- Teilaufgaben
 - ↳ Ermittlung eines geeigneten Pfades vom Quell- zum Zielhost („Routing“)
 - ↳ Paketvermittlung in Transitsystemen
 - ↳ Call-Setup
 - ↳ VC-Netzwerk: Konfiguration aller Transitsysteme

Routing-Algorithmen

Modell Netzwerk als Graph

- Netzwerk wird als gewichteter Graph modelliert (Menge von Knoten, die durch Kanten verbunden sind, denen eine Zahl zugeordnet ist)
- Knoten := Transitsysteme mit Routing-Entscheidung
- Kanten := Kommunikationsleitungen zwischen Transitsystemen
- Gewichte := Kosten der Verbindungsnutzung
- ⇒ Pfade können Gesamtkosten zugeordnet werden



Aufgabe und Merkmale von Routing-Algorithmen

- Aufgabe: Wegauswahlverfahren: Ermittlung des günstigsten Pfades zwischen Quell- und Zielknoten
- Merkmal: Global und Dezentral
 - ↳ globaler Routing-Algorithmus
 - ↳ besitzt vollständiges Wissen über gesamtes Netzwerk → „Link-State“ / globale Zustandinformation
 - ↳ Berechnungen können zentral oder verteilt ausgeführt werden
 - ↳ dezentraler Routing-Algorithmus
 - ↳ Knoten kennen nur Informationen der direkten Verbindungen, nicht des gesamten Netzwerks
 - ↳ billiger Pfad wird durch Informationsaustausch mit den Nachbarn iterativ verteilt berechnet (Beispiel: Distanzvektoralgorithmus)
- Merkmal: Statisch und Dynamisch
 - ↳ statischer Routing-Algorithmus
 - ↳ Routing-Entscheidungen werden manuell geändert (z.B. Routing-Tabelle im Router editieren)
 - ↳ dynamischer Routing-Algorithmus
 - ↳ Routing-Entscheidungen werden automatisch (periodisch oder veränderungsbedingt) an Netzwerksituation angepasst

Notation

Im Folgenden gilt nachfolgende Notation. Seien i, j Knoten eines Netzwerks.

$c(i,j) \geq 0$ Kosten der direkten Verbindung von Knoten i zu Knoten j .
Es gilt Symmetrie $c(i,j) = c(j,i)$ und $c(i,i) = \infty$, falls keine direkte Verbindung existiert.

d_i Kosten des Pfades vom Quellknoten zum Knoten i , der in dieser Iteration am billigen ist
 p_i Vorgängerknoten von Knoten i auf aktuell günstigsten Pfad vom Quellknoten zu Zielknoten i
 S Menge der Knoten, deren billigste Pfade zum Quellknoten bereits bekannt sind

Globaler Link-State-Algorithmus

- Voraussetzung: alle Netzwerkknoten verfügen über alle Informationen
 - ↳ Praxis: jeder Knoten sendet Kosten seiner angrenzenden Verbindungen an alle anderen Knoten („State-Broadcast-Message“)
 - ↳ Senden von $O(nL)$ Nachrichten im Netzwerk mit n Knoten und L Verbindungen notwendig, damit jede Knoten die gesamten Graphenkosten kennt
- jeder Knoten berechnet mit Dijkstras Algorithmus jeweils die billigsten Pfade zu allen anderen Netzwerkknoten
 - ↳ Berechnung für k Zielknoten erfolgt in k Iterationen
 - ↳ Zeitkomplexität $O(n^2)$ für Pfadberechnung zu allen $n-1$ Zielknoten
 - ↳ Nach Terminierung kennt jeder Knoten seinen Vorgänger auf dem billigsten Pfad zum Quellknoten → Pfad rekonstruierbar
 - ↳ Berechnung muss von außen angestoßen werden

Dijkstras iterativer Algorithmus

Initialisierung

$S = \{Q\}$ (Q Quellknoten)
 Für alle Nebenknoten i
 begin
 Falls i direkter Nachbar von Q
 dann $d_i = c(Q, i)$; $P_i = Q$
 sonst $d_i = \infty$.
 end

Iteration

Wiederhole bis alle Netzwerkknoten in S sind

begin

 Finde ein $i \notin S$ mit $d_i = \min_{j \in S} \{d_j\}$

 Füge i zu S hinzu;

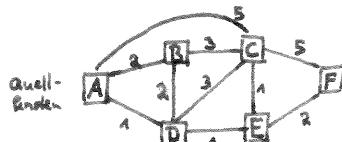
 Update d_j für alle direkten Nachbarn $j \notin S$ von i , falls über j billiger zu erreichen

 Falls $d_i + c(i, j) < d_j$

 dann $d_j = d_i + c(i, j)$; $P_j = i$,

end

Beispiel



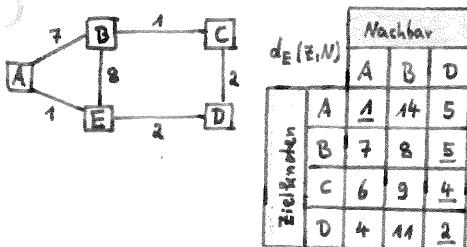
Rekonstruktion billiger Pfade

- $A \rightarrow B: 2, A \rightarrow B$
- $A \rightarrow C: 3, A \rightarrow D \rightarrow E \rightarrow C$
- $A \rightarrow D: 1, A \rightarrow D$
- $A \rightarrow E: 2, A \rightarrow D \rightarrow E$
- $A \rightarrow F: 4, A \rightarrow D \rightarrow E \rightarrow F$

Schritt	S	d_B, P_B	d_C, P_C	d_D, P_D	d_E, P_E	d_F, P_F
Init	A	2, A	5, A	1, A	∞	∞
1	A, D	2, A	4, D		2, D	∞
2	A, D, E	2, A	3, E			4, E
3	A, D, E, B		3, E			4, E
4	A, D, E, B, C					4, E
5	A, D, E, B, C, F					

Dezentraler Distanzvektor - Algorithmus

- iteratives, asynchrones Verfahren zur Berechnung der billigsten Pfade
- jeder Knoten tauscht Informationen mit seinen direkten Nachbarn aus \Rightarrow jeder Knoten soll günstigste Pfadkosten jedes Nachbarn zu jedem Zielknoten kennen
 - \hookrightarrow Nachrichten tauschen Nachbarn aufzulisten am (z.B. Änderung des günstigsten Pfades)
 - \hookrightarrow Knoten müssen nicht innerhalb einer bestimmten Zeit reagieren
- grundlegende Datenstruktur: Distanztabelle pro Nebenknoten (wird in Knoten gespeichert)
 - \hookrightarrow gibt minimale Pfadkosten vom aktuellen Knoten über Nachbarknoten zu Zielknoten an
 - \hookrightarrow jeder Knoten erstellt aus Distanztabelle seine Routing-Tabelle (anhand der Zeilenminima)
 - \hookrightarrow Knoten ändert Distanztabelle bei Kostenveränderung einer von ihm abgehenden Verbindung oder bei Update eines direkten Nachbarknoten
 - \hookrightarrow Knoten informiert alle direkten Nachbarn, falls er einen neuen billigen Pfad berechnet



Distanztabelle für Knoten E

- Zelle $d_E(Z, N) :=$ Kosten des Pfades von E nach Ziellknoten Z über direkten Nachbarn N
- $d_E(Z, N) =$ Kosten direkte Verbindung von E nach N + billige Pfadkosten von N nach Z
- $d_E(Z, N) = C(E, N) + \min_W \{d_N(Z, W)\}$ mit $W = \{\text{Nachbarknoten von } N\}$

Welt-Knoten E nur durch Update der Nachbarknoten bekannt

- Daten austausch zwischen direkten Nachbarn über Bellman-Ford-Algorithmus
 - \hookrightarrow wird an jedem Knoten ausgeführt

Bellman - Ford - Algorithmen↳ Initialisierung

Für alle Nachbarknoten V:

begin

$d_x(\ast, V) = \infty$ (\ast := für alle Zeilen)

$d_x(V, V) = c(x, V)$

end

Für alle Zielknoten Z:

begin

Sende $\min_w \{d_x(Z, w)\}$ an jeden direkten Nachbarn

end

↳ Iteration

Wiederhole für immer:

begin

Warte auf Koständerung der direkten Nachbarn V oder auf Update der Verbindungs kosten von V

Falls $c(x, V)$ sich um Δ geändert hat

begin

Für alle Zielknoten Z:

begin

$d(Z, V) = d(Z, V) + \Delta$

end

end

Sonst, falls Update von Nachbar V bzgl. Zielknoten Z empfangen

/* V hat neuer Wert = $\min_w \{d_V(Z, w)\}$ gesetzt */

begin

$d_x(Z, V) = c(x, V) + \text{neuer Wert}$

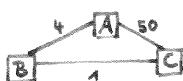
end

Falls neu $\min_w \{d_x(Z, w)\}$ für Zielknoten Z existiert

begin

Sende neuen minimalen Kostenwert $\min_w \{d_x(Z, w)\}$ für Z an alle direkten Nachbarn

end

Beispiel

nach
Initialisierung

	B	C
B	4	<u>∞</u>
C	<u>∞</u>	50

	A	C
A	4	<u>∞</u>
C	<u>∞</u>	1

	A	B
A	50	<u>∞</u>
B	<u>∞</u>	1

erste
Iteration

	B	C
B	4	51
C	5	50

	A	C
A	4	51
C	54	1

	A	B
A	50	5
B	54	1

zweite
Iteration

	B	C
B	4	51
C	5	50

	A	C
A	4	6
C	9	1

	A	B
A	50	5
B	54	1

↳ Problem Routingschleifen

- bei Kostenerhöhung einer Verbindung benötigen Knoten x Iterationen, um die globale Netzwerksituation in Routing-Tabelle dargestellt

↳ beide Knoten können annehmen, dass ein gemeinsamer Zielknoten über den jeweils anderen Knoten am günstigsten zu erreichen ist

⇒ Pakete werden endlos zwischen beiden Knoten hin- und hergeschickt, bis globale Netzwerksituation nach x Iterationen korrekt dargestellt ist

↳ Count-to-Infinity-Problem

- Kostenverhöungen werden nur langsam durch Netzwerk propagiert

↳ während der x Iterationen zur Berechnung des nun kostengünstigsten Pfades wird der Verkehr noch über die schlechteste Verbindung abgewichen

↳ Lösung: Routing-Schleife: Poisoned-Reverse-Strategie

- Wenn ein Knoten Pakete über Nachbarknoten an einen Zielpunkt sendet, obwohl er eine direkte Verbindung zum Zielpunkt besitzt, so verhindert er diese gegenüber dem Nachbarknoten

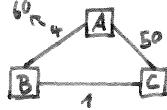
⇒ Nachbarknoten wird kein Pakete über Knoten an Zielpunkt senden

- Nachteile

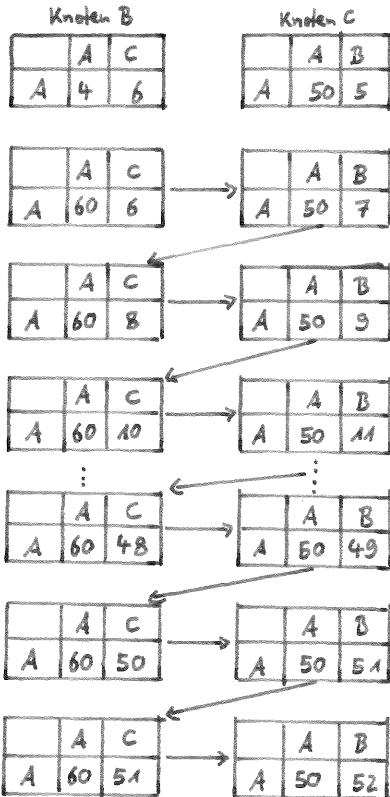
↳ Schleifen zwischen mehr als zwei Knoten werden nicht beseitigt

↳ Strategie löst nicht das Count-to-Infinity-Problem

Entstehung einer Routing-Schleife

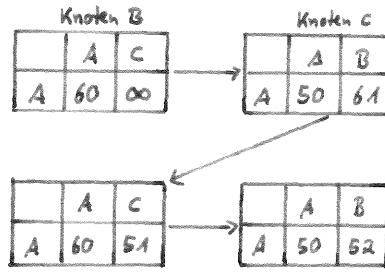


Distanztabelle



Poisoned - Reverse - Strategie

Distanztabelle



Vergleich Link-State - vs. Distanzvektor - Algorithmus

Komplexität bzgl. Nachrichtenzahl

- Link-State: Verwendung von $O(nL)$ Nachrichten im Netzwerk mit n Knoten und L Verbindungen
 - ↳ Nachrichtenverbreitung durch Fluten-Methode
 - ↳ Nachrichtenpaket mit IP-Adresse des Knotens, Verbindungskosten zu direkten Nachbarn und Sequenznummer an Nachbarn senden
 - ↳ Empfänger:
 - Speichert Paket, falls noch kein Paket vom Sender erhalten oder
 - speichert Paket, falls größere Sequenznummer
 - leitet Paket an alle Nachbarn außer Sender weiter
- Distanzvektor: Nachrichten müssen nur an direkte Nachbarn gesendet werden, wenn neuen billigsten Pfad berechnet wurde

Konvergenzgeschwindigkeit (Geschwindigkeit zum Erhalt einer stabilen Lösung)

- Link-State: Reine Konvergenzphase
- Distanzvektor:
 - Während der Konvergenzphase sind Routing-Tabellen nicht stabil, Updates werden oft mehrfach nötig
 - ↳ Dieser der Tabellenveränderung nicht abelbar
 - ↳ Routing-Schleifen und count-to-infinity-Problem
 - Algorithmus reagiert schnell auf gute und langsam auf schlechte Verbindungen

Robustheit (Knotenauffall / Fehler werden von Nachbarn bemerkt)

- Link-State: jeder Knoten verbreitet Nachricht im Netz → Fehlerhafte Knoten werden nicht mehr im Netz aufgeführt → gewisse Robustheit des verteilten Systems
- Distanzvektor:
 - Nachbarn des ausgefallenen Knotens aktualisieren Routing-Tabellen und informieren Nachbarn
 - Routing-Schleifen können entstehen → Netzwerkgesamtunstabil möglich

⇒ Link State ist besser als Distanzvektor-Algorithmus, ist aber nicht für große Netzwerke verwendbar, da jede Knoten den gesamten Netzwerkgraphen kennen muss

Hierarchischer Routing

KE4

Praxisprobleme

- GröÙe reale Netzwerke (z.B. Internet) → Berechnen, Speichern und Weiterleiten von Routing-Tabelle-Informationen nicht tragbar
- Autonomie der Routing-Algorithmenwahl von den netzbetreibenden Organisationen

Lösung

- Autonome Systeme (AS) bilden Mengen von Transit systemen mit selben Routing-Algorithmen zusammen (Intra-AS-Routing)
- Gateway-Router verbinden autonome Systeme, in dem sie Pakete an Zielknoten außerhalb des AS weiterleiten (Inter-AS-Routing)

Vorteile

- Komplexität reduziert
 - ↳ AS muss nur Größe der eigenen homogenen Netzwerke bewältigen
 - ↳ Gateways müssen nur mit der Anzahl der verbundenen AS fertig werden
 - Betreiber AS kann Routing-Protokoll seines Systems festlegen
 - Gateway führt Intra- und Inter-AS-Routing-Protokoll an
- Knoten $\xrightarrow[\text{(Intra)}]{\text{Paket}} \xrightarrow[\text{(Inter)}]{\text{Gateway}} \xrightarrow[\text{(Inter)}]{\text{Gateway}} \xrightarrow{\text{Zielknoten}}$

Beispiel Internet

Allgemein

- Vermittlungsschicht im Internet verwendet aktuell das IP-Protokoll
- IP-Protokoll bietet einen verbindunglosen Best-Effort-Dienst zum Übertragen von Datagrammen
 - ↳ Paketverluste möglich ↳ Ankunftsreihenfolge kann vom Sendereiterfolg abweichen ↳ Keine Bandbreitengarantie
 - ↳ Keine Aussage zur Übertragungsduer ↳ Keine Weitengale von Überlastungsinformationen.
- Transportschicht-Segmente werden in ein oder mehrere IP-Datagramme mit Sender- und Empfängerinformationen verpackt
- IP-Datagramm
 - ↳ IP-Adresse Sender und Empfänger
 - ↳ Datengeld (üblicherweise TCP- oder UDP-Segment)
 - ↳ Längenfeld: Anzahl tatsächlich genutzter Bytes
- verwendete Routing-Algorithmen
 - ↳ dynamischer globaler Link-State-Algorithmus
 - ↳ dynamischer dezentraler Distanzvektor-Algorithmus
- Komponenten der Vermittlungsschicht
 - ↳ Internetprotokoll Version IPv4
 - definiert Adressierung, Datagramm-Felder und Aktionen von Routern / Endgeräten auf Datagrammen
 - ↳ Pfadbestimmungskomponente
 - ↳ Protokoll ICMP (Internet Control Message Protocol)
 - Datagrammfehler anzeigen und Vermittlungsschichtinformationen abfragen

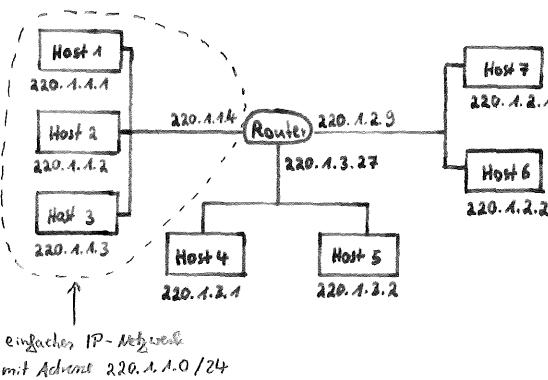
Adressierung in IPv4

- Schnittstelle (Interface) := Grenz zwischen Host / Router und der physischen Netzwerkverbindung
- Jeder Interface benötigt eine global eindeutige IP-Adresse
- IP-Adressen sind 4 Byte lang und bestehen aus Netzwerkpräfix und Interfaceid
 - ↳ Problem: feste Länge des Netzwerks zu unflexibel bzgl. der Anzahl der Hosts im Netzwerk
 - ↳ Lösung: CIDR-Standard (Classless Interdomain Routing Standard)
 - ↳ Adresse beliebig aufteilbar Notation: a.b.c.d/X (erste x von 32 Bit beschreiben Netzwerkpräfix)
 - ↳ /X wird auch als Netzwerkmaske (subnet mask) bezeichnet
- Interfaceid wird entweder manuell vom Systemadministrator oder durch DHCP-Server (Domain Host Configuration Protocol) dynamisch zugewiesen
- Netzwerkpräfix wird Administrator von Internet-Provider zur Verfügung gestellt

- IP-Adressräumen in IPv4

Host - Adressbereich		
A	0 Netzwerk (7)	Host (24)
		4.0.0.0 bis 127.255.255.255
B	10 Netzwerk (10)	Host (16)
		128.0.0.0 bis 191.255.255.255
C	110 Netzwerk (20)	Host (8)
		192.0.0.0 bis 223.255.255.255
D	1110 Multicast - Adresse (28)	
		224.0.0.0 bis 239.255.255.255
E	11110 Reserviert für spätere Nutzung (27)	
		240.0.0.0 bis 247.255.255.255
32 Bit		

- Beispiel Interface - Adressen für Mitglieder aus drei einfachen Netzwerken



(KE4)

Routing von IP-Datagrammen

- 1.) IP-Protokoll des Quellhosts sucht in Routing-Tabelle nach einem Eintrag mit der Netzwerkkennung der IP-Adresse des Zielhosts
- 2.) IP-Datagramm an Sicherungsschicht zum Transport zum Zielhost übergaben
- 3.) Falls Zielhost ein Router (Transitsystem) zum eigentlichen Zielhost in einem anderen Netzwerk ist, sucht in Routing-Tabelle des Routers die nächste Pfadstation und leite IP-Datagramm weiter

Fragmentierung und Reassembly von IP-Datagrammen in IPv4

- Sicherungsprotokolle zum Datagrammtranspat über physische Leistungen unterscheiden sich durch maximale Paketgröße (MTU, maximum transfer unit)
 - ↳ meist 576 Byte Dateninhalt, Ethernet 1.500 Byte Dateninhalt
 - ↳ Problem: IP-Datagramme werden über Pfade mit unterschiedlichen MTUs gesplittet werden
- Lösung: Fragmentierung von IP-Datagrammen auf kleinste gemeinsame MTU und Reassembly nach Ankunft durch Endsysteme
- vorendete Header-Felder für die Fragmente
 - ↳ Identifikation (ID): Fragment gehört die ID seines IP-Datagramms
 - ↳ Flag: Endfragment Flag = 0, alle restlichen Fragmente Flag = 1
 - ↳ Fragmentation Offset: gibt Reihenfolge des Fragments im IP-Datagramm an
- bei: Fragmentverlust wird gesamtes Datagramm verloren
- Nachteile
 - ↳ Fragmentierung verlangt Verarbeitung im Router
 - ↳ Reassembly verlangt Verarbeitung auf dem Empfänger

Intra-AS-Routing

- Routiert Routing-Tabellen in allen Routern des autonomen Systems
- Protokolle Praktis:
 - ↳ OSPF (Open Shortest Path First)
 - ↳ EIGRP (Enhanced Interior Gateway Routing Protocol)
 - ↳ RIP (Routing Information Protocol)
- RIP
 - ↳ nutzt Distanzvektor-Algorithmus
 - ↳ Kosten (Anzahl der Hops) auf maximal 15 beschränkt
 - ↳ Nachbarknöte tauschen alle 30 Sekunden Routing-Informationen über RIP Response Nachrichten aus
 - ↳ Routen ohne Update für mehr als 180 Sekunden gilt als unerreichbar → Nachbar modifiziert ihre lokale Routing-Tabelle und sendet Update
 - ↳ Route können über RIP Nachbarn zu Verbindungskosten begrenzt
- (↳ RIP kommuniziert mit UDP-Datagrammen über Standardport 520 ⇒ RIP ist Protokoll der Anwendungsschicht, da es ein Trampslichtprotokoll ist → Realisierung einer Vermittlungsfunktion nutzt)

Inter-AS-Routing

- administrative Domänen (Domains) := autonome Systeme im Internet
- BGP4 (Border Gateway Protocol Version 4) De-Facto - Standard
 - ↳ Pfadvektor-Protokoll: basiert auf Distanzvektor-Algorithmus, aber in Routen werden Reihenfolge der AS vom Quell- zum Ziel-AS hinzugefügt, keine Kosten!
 - ↳ betrachtet Internet als Graph nummerierter AS, AS kann Pfad über andere AS zum Ziel-AS kennen
 - ↳ Auswahl von mehreren möglichen Pfaden ist Richtlinienentscheidig (policy - decision) des Domänenadministrators
- Propagierung von BGP-Informationen durch BGP-Nachtausstausch mit direkten Nachbarn
 - ↳ Open-Nachricht: Authentifizierung und Informationsabgleich mit Nachbarn
 - ↳ Update-Nachricht: Pfade an Nachbarn melden oder zurückziehen
 - ↳ Keepalive-Nachricht: Akzeptierte Verbindungsauflauf mit Open-Nachricht oder Lebenszeiten
 - ↳ Notification-Nachricht: signalisiert Fehler bei vorheriger Nachricht oder beabsichtigten Verbindungen

ICMP - Protokoll (Internet Control Message Protocol)

KE 4

- Hosts, Router und Gateways im Internet tauschen Vermittlungsrouteninformationen mittels ICMP-Nachrichten aus
 - ↳ wird oft zur Kommunikation von Fehlermeldungen eingesetzt (z.B. Router kennt keinen Pfad zum Zielhost)
- nutzt IP-Diagramme zur Kommunikation
- Nachrichten werden über Typ und Code-Feld spezifiziert
 - ↳ enthält erste 8 Byte des betreffenden IP-Datagramm

ICMP-Typ	Code	Bedeutung
0	0	Echo - Antwort auf Ping-Nachricht
3	0	Zielnetzwerke unerreichbar
3	1	Zielhost unerreichbar
3	2	Zielprotokoll unerreichbar
3	3	Zieldport unerreichbar
3	6	Zielnetzwerk unbekannt
3	7	Zielhost unbekannt
4	0	Quelle reduzieren (zur Überlastkontrolle)
8	0	Echo-Anforderung (Ping-Nachricht)
9	0	Router - Bekanntmachung
10	0	Router - Entdeckung
11	0	Time-To-Live (TTL) abspielen
12	0	Fehlerkoffer IP-Header

- Verwendungsbereiche ICMP

↳ Programm ping

↳ Programm traceroute

↳ ermittelt Pfad der IP-Datagramme zu einem Zielhost

↳ sendet IP-Datagramme mit jeweils um eins inkrementiertem TTL-Feld an Zielhost

↳ Router decremente TTL-Feld um eins

↳ TTL = 0 => IP-Datagramm verloren und Warning mit IP-Adresse des Routens und Zeitstempel am Sender senden

Neue Entwicklungen in IPv6

- IPv6-Adressen sind 128 Bit lang → größter Adressraum für rapido Wachstum des Internets
- Anycast-Adressen als Ergänzung zu Unicast- und Multicast-Adressen
 - ↳ Datagramm wird an irgendeinen Host einer Anycast-Gruppe weitergeleitet
- Header ist 40 Byte lang zur effizienteren Verarbeitung
 - ↳ Prüfsummen-Feld entfernt, da Trägerprotokoll und viele Sicherungsstufen bereits Fehlererkennung und - Korrektur betreiben
 - ↳ Berechnen & Testen der Prüfsumme in jedem Knoten entfällt
 - ↳ Fragmentierung und Reassembly entfernt
- Übergangsmöglichkeiten von IPv4 zu IPv6
 - ↳ Stichwort (nicht realistisch)
 - ↳ Dual-Stack-Ansatz
 - ↳ jeder IPv6-Knoten implementiert auch IPv4
 - ↳ Knoten stellen versendete IP-Version durch DNS fest
 - ↳ Datagramme können Vorteile von IPv6 nicht nutzen, wenn mindestens ein IPv4-Knoten auf dem Pfad liegt
 - ↳ Tunneling-Ansatz
 - ↳ IPv6-Knoten verpackt IPv6-Verkehr bei Übertragung über IPv4-Knoten in IPv4-Datagramme einpacken

Sicherungsschicht

KE4

- Sicherungsschicht (data link layer) überträgt Datagramme von einem Knoten über eine einzelne Verbindungsleitung an einen direkten Nachbarknoten
- Framing: Datagramme werden in Rahmen (frames) gekapselt
- Sicherungsschicht - Protokolle werden i.d.R. für bestimmte Kommunikationsleitungen implementiert
- meist verwendete Vermittlungsform: Busstruktur
- meist verwendetes Protokoll: Ethernet

Dienste der Sicherungsschicht

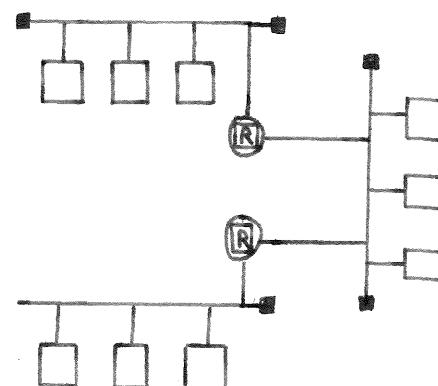
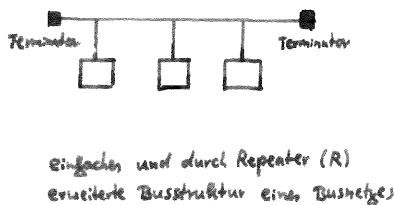
- möglichst zuverlässige Weiterleitung von Frames
- Techniken zum Bestätigen und Neuübertragung
- Flunkkontrolle
- Voll duplex / Halbduplex - Dienst
 - ↳ Voll duplex := beide Knoten können gleichzeitig Pakete übertragen
 - ↳ Halbduplex := Knoten können nicht gleichzeitig senden

- Unterschiede der angebotenen Dienste je nach Leitungstypen
 - ↳ Störungsempfindliche Leitungen → Fehlerbehandlung
 - ↳ Mehrfachzugriffproblem bei Broadcast - Leitung mit mehreren Knoten
 - ↳ Konkurrenzgriffprotokolle
 - ↳ gemeinsame Nutzung Broadcast - Kanal durch mehrere Knoten
 - ↳ Address Resolution Protocol (ARP) bildet Vermittlungsroutenliste auf Sicherungsschichtebene ab

Vermittlungsformen

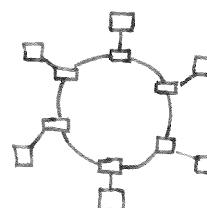
Busstruktur

- Bus := allen angeschlossenen Knoten zugängliches Übertragungsmedium
- Im Busnetz hören alle Knoten die gesendeten Signale mit (Broadcast - Netz / Carrier Sense - Netz)
- Zugangsprotokoll für Knotenzugriff auf Bus und Kollisionsverhalten
- Terminator absorbiert Restenergie der Signale
- Repeater verstärkt Signale und leitet sie an angeschlossene Segmente weiter (Ethernet maximal 4 Repeater → 2.500 Meter Kabellänge)



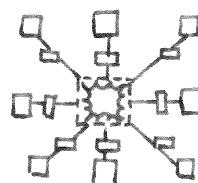
Ringnetz

- Datenübertragung zwischen zwei Knoten über gerichtete Leitungen
- Protokolle: Token-Ring, FDDI
- Knoten entscheiden, ob Nachricht
 - ↳ unverändert weitergegeben (Broadcast)
 - ↳ verändert weitergegeben oder
 - ↳ vom Ring genommen wird
- Nachteil: Verzögerung bei jedem Knoten und Kommunikation bei Knotenausfall eingeschränkt



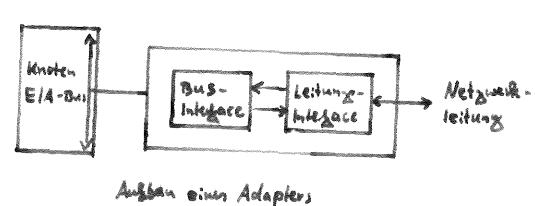
sternförmiges Netz

- alle Knoten sind mit einem zentralen HUB verbunden
- Vorteil: Kommunikation bei Knotenausfall möglich und leichtes Hinzufügen neuer Knoten



Adapter

- Sicherungsschicht - Protokolle werden in Adapters (Network Interface Card / NIC) realisiert
- Adapter gehören zum physischen Gerät des Knotens (Elternknoten) und verbinden Host mit Netzwerkleitung
- Aufbau eines Adapters
 - ↳ Komponente 1: Bus - Interface (Host - Bus - Schnittstelle)
 - ↳ Komponente 2: Leistung - Interface (Leitungsschnittstelle) zu Netzwerkleitung
- Adapter als „halbautonome Einheit“
 - ↳ Entscheidungen der Sicherungsschicht - Protokolls werden in der Leitungsschnittstelle (oft in Hardware) implementiert



Bitübertragungsschicht

KE4

Allgemein

- Bitübertragungsschicht stellt physische Übertragungskanäle zum übertragen beliebiger Bitfolgen zur Verfügung
 - ↳ Informationen auf physischen Medium darstellen und beim Empfänger rekonstruieren
- Qualität des Mediums bestimmt Protokollentscheidung
 - ↳ Kupferkabel (elektrische Signale) raumbehaftet und störungsempfindlich
 - ↳ Glasfaserkabel (optische Signale) relativ sicher
- maximale Übertragungsgeschwindigkeit im bit/s bestimmt Leistungsfähigkeit
- Übertragungsverzögerung kann zu Kollisionen führen
- Einsatz von Repeatern gegen Signalabschwächung
- Daten liegen digital (Texte/Bilder) oder analog (Audio/Video) vor
 - ↳ Abbildung digitale Daten \leftrightarrow analoge Signale und analoge Daten \leftrightarrow digitale Signale notwendig

Abbildung digitale Daten \leftrightarrow analoge Signale

- Trägerstrom - Übertragungsverfahren stellt digitale Daten in analogen Signalen dar
- Modems (Modulator/Demodulator) wandeln digitale in analoge Signale um und gewinnt aus analogen Signalen digitale Information
- während der Übertragung werden Signale gestützt oder gestört
- Modellierungarten

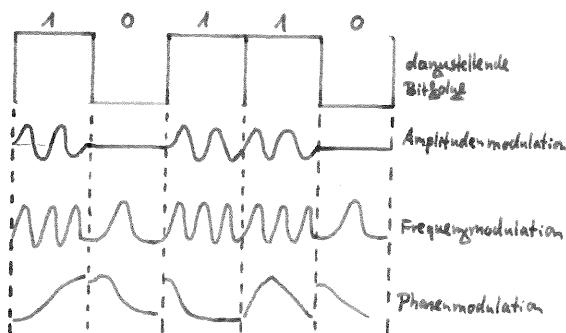
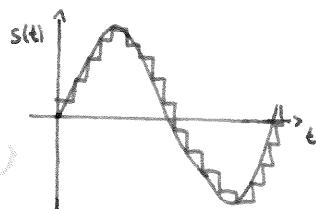


Abbildung analoge Daten \leftrightarrow digitale Signale

- Codec (Codierer / Decodierer) wandelt analoge in digitale Signale um und gewinnt aus digitalen Signale analoge Information
- bekanntes Verfahren PCM (Pulse Code Modulation): reellen Werte des analogen Signals werden als gemusterte Zahlen digital übertragen



Synchronisation

- Sender und Empfänger synchronisieren sich und legen Schrittdauer und Zeichenlänge fest
 - ↳ asynchroner Start-/Stopbetrieb

- Beginn und Ende einer Teilübertragung werden durch besondere Signalfolgen gekennzeichnet

synchrone Datenübertragung

- Takt wird vom Netz geliefert oder Empfänger synchronisiert sich mit dem empfangenen Signal des Senders

- ↳ Datenstrom wird in Blöcke fester Länge eingeteilt, die mit Synchronisationszeichen beginnen

- ↳ Vorteil: Mittlere Zeichenübertragung durch fehlende Start/Stop-Schritte größer als bei asynchroner Betriebsweise

Grundbegriffe

- Datenbankschema := enthält Informationseinheiten und ihre Beziehungen
- Redundanz := mehrfacher Vorhalten gleicher Information
- Konsistenz := logische Übereinstimmung von Daten
- Integrität := Vollständigkeit und Korrektheit der Daten gemessen an der Realwelt
- Datenunabhängigkeit := Änderung von Datendefinitionen gilt nicht zu Änderungsbedarf in Anwendungsprogrammen
- Informationssystem := Gesamtheit aller Instanzen und Prozesse, die Unternehmensdaten aufnehmen, verarbeiten und weitergeben

Datensysteme (File systems)

- Konventionelle Datenverwaltung vor dem Einsatz von Datenbanksystemen
- Daten werden in Säulen nach deklarierten Satztypen in benannten Dateien verwaltet
- Datensysteme übernehmen Zugriff auf Sätze einer Tabelle mithilfe eines Parameters, z.B. den Schlüssel
- Dateneorganisation für Sätze: sequentiell, index-sequentiell oder direkte Organisation (z.B. Hash-Vorfahren)
- jedes Anwendungsprogramm definiert und verwaltet seine eigenen Datenbestände in Dateien

Probleme

- Redundanz: Speicherverschwendug; erhöhte Verarbeitungskosten
- Inkonsistenzen durch Redundanz möglich und wahrscheinlich
- enge Kopplung zwischen Programm und Daten: Änderung des Dateiaufbaus oder -organisation macht Anpassungen in allen verwendeten Programmen notwendig
- Inflexibilität beim Entwickeln neuer Anwendungen und Ad-Hoc-Auswertungen von Daten

Datenbanken

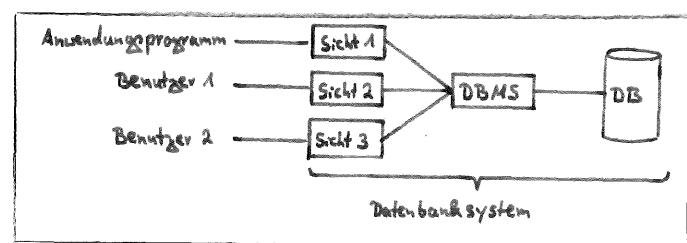
- „Definition“ Datenbank: integrierte Ansammlung von Daten, die allen Benutzern eines Anwendungsbereichs als gemeinsame Basis aktueller Information dient
- Daten werden als eigenständigen Betriebsmittel gesehen und den natürlichen Zusammenhängen entspricht als logische Informationseinheiten strukturiert
 - ↳ Trennung von Anwendungsprogrammen und Daten
- Daten werden einmal definiert, zentral verwaltet und von allen Anwendungsprogrammen benutzt
- Anwendungsprogramme erhalten nur die benötigten Datenelemente

Vorteile

- gemeinsame Datenbasis für alle Anwendungen
 - ↳ Konsistenzprobleme entfallen durch Vermeiden oder zentrale Kontrolle von Redundanz
 - ↳ einfache Anwendungsprogrammierung
- Datenunabhängigkeit
- leichtere und flexible Entwicklung von Anwendungsprogrammen
- Ad-Hoc-Auswertungen möglich
- Integritätsprüfungen und Mechanismen zur Wiederherstellung eines konkreten Datenbankstandards nach Fällen

Datenbanksystem (DBS)

- Datenbanksystem (DBS) := Datenbank + Datenbanksoftware (Database Management System / DBMS)
- Datenbank Management System (DBMS) kontrolliert Datenbank und isoliert Datenbank von Anwendungsprogrammen
- Architektur Datenbanksystem:
 - logische Gesamtansicht
 - beschreibt Informationseinheiten und ihre Beziehungen
 - interne Sicht
 - physische Speicherorganisation für effizienten Zugriff
 - externe Sicht
 - benutzerspezifische Darstellung der Daten
- Sichten werden in Modellen implementiert
- Abbildung zwischen den Sichten durch Transformationen



Konzeptuellen Modell

- realisiert die logische Gesamtansicht
 - ↳ beschreibt logische Gesamtansicht + Integritätsbedingungen für Datenmanipulation + evtl. Operationen auf den Daten
 - ↳ Konzeptuelles Modell wird mit Datenbeschreibungssprache (data definition language / DDL) im Konzeptuellen Schema
- Vorteile
 - ↳ stabiler Bezugspunkt zur Realwelt → erlaubt Änderungen der internen und externen Schicht ohne Anpassungen bestehende Programme
 - ↳ einheitliche Dokumentation wesentlicher Unternehmensaspekte
 - ↳ zentrale Stelle zur Gebrauchskontrolle der Daten
 - ↳ Voraussetzung für die Datenunabhängigkeit

internes Modell

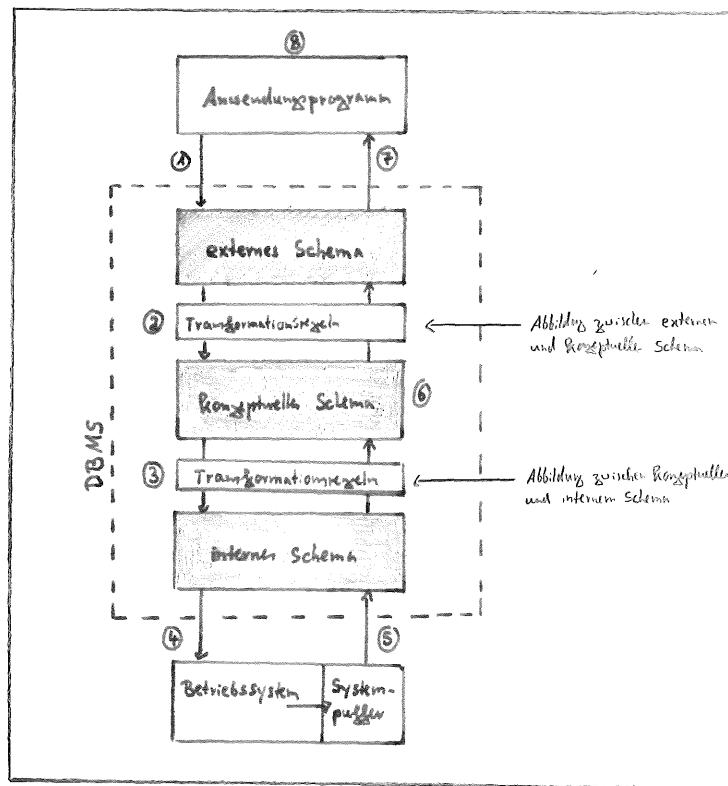
- beschreibt Speicherorganisation und Zugriffsmöglichkeiten auf Daten
- physische Speicherorganisation bzgl. internen Modell wird im internen Schema beschrieben
- Festlegungen zu:
 - ↳ Repräsentation von Attributwerten
 - ↳ Satzgruppen
 - ↳ Zugriffsmethoden
 - ↳ günstige Zugriffsfolge (Indizes, Verkettungen etc.)
- Datenbankadministrator benötigt zur Einrichtung statistische Informationen

externes Modell

- jeder Benutzer erhält eigene Sicht (view) auf die Daten
- Objekt- und Beziehungstypen externer Modelle müssen nur inhaltlich im Konzeptuellen Modell enthalten sein

Datenunabhängigkeit

- Änderungen eines Schemas können durch Anpassung der Transformationssregeln von anderen Schichten ferngestellt werden
 - ↳ physische Datenunabhängigkeit: Änderung von Änderungen des internen Schemas isoliert
 - ↳ logische Datenunabhängigkeit: Änderung von Änderungen des Konzeptuellen Schemas isoliert
 - ↳ statische Datenunabhängigkeit: beim Binden zur Abfragezeit muss Änderung neu übersetzt (compiled) werden
 - ↳ dynamische Datenunabhängigkeit: beim Binden zur Zugriffzeit (Interpretation) entfällt Neuübersetzung → verhindert hohe Kosten



Datenbankmanagementsystem (DBMS)

- übernimmt Zugriffe auf Datenbank mithilfe des Betriebssystems
- bildet die einzelnen Schichten über Transformationssregeln ab
- Aufgaben:

1. Objekte lösen

- Anfragen werden in DatenmanipulationsSprache (data manipulation language / DML) in den Begriffen des externen Schemas entzogen
- ↳ Zugriff auf externes Objekt Raum zum Zugriff auf mehrere Konzeptuelle und interne Objekte führen
- ↳ Vorgehen:
 - ① DBMS empfängt Lesebefehl des Anwenders und holt Definitionen an externem Schema
 - ② DBMS ermittelt mit Transformationssregeln benötigte Konzeptuelle Objekte / Beziehungen
 - ③ DBMS ermittelt mit Transformationssregeln zu lösende physische Objekte / Zugriffsfolge
 - ④ DBMS übergibt Betriebssystem Nummern der zu lösenden Speicherblöcke
 - ⑤ Betriebssystem übergibt Blöcke im Systempuffer an DBMS
 - [Gepl. prüfen ob gewünschte Daten gefunden wurden]
 - ⑥ DBMS stellt mit Transformationssregeln aus den physischen Sätzen das verlangte externe Objekt zusammen
 - ⑦ DBMS übergibt externes Objekt an Anwender
 - ⑧ Anwender verarbeitet übergebene Daten

2. Datendefinition

- Datendefinitionen in DatendefinitionsSprache (data definition language / DDL) interpretieren
- aus Quellform (source) eine interne Darstellung erzeugen

3. Integrität der Datenbank schützen

4. Datensicherung (Recovery)

5. Koordination parallel arbeitender Benutzer

6. Datenschnitt

- Berechtigungsstrukturen zum Lesen und Manipulieren von Daten

- Weitere Komponenten DBMS

KE 1

↳ Tools

- Abfragezeile
- Report - Writer
- Spreadsheets
- Tools für Datenbankentwurf
- 4GL - Entwicklungsumgebung

↳ Utilities

- Laderoutinen für initiale Laden der Datenbank
- statische Routinen
- Routinen zur Fehleranalyse
- Routinen zur Dataorganisation auf Speicher
- Kopier- und Archivierungs routinen

↳ Data Dictionary und Repository

- DBMS speichert Meta- und Verwaltungsdaten der Datenbank im Data Dictionary
- Data Dictionary ist selbst eine Datenbank und stellt Benötigte Schema-Informationen zur Verfügung
- Inhalte:
 - Datenbeschreibung & Beziehungen
 - Programmbeschreibung (Transaktionen)
- Konsistenzbedingungen
- Zugriffsberechtigungen

Modellierung der Konzeptuellen Ebene

- Realwelt wird zunächst als semantisches Datenmodell beschrieben
- Konzeptuelles Modell wird aus semantischem Modell abgeleitet und mit DDL als Konzeptuelle Schema realisiert

Entity-Relationship-Modell (ER-Modell)

- Basiskonstrukte:

Attribute

- besteht aus Namen und Wertebereich
- Symbol: Kreis

Entität (entity)

- einzelnes Objekt, besteht aus Attributen

Entitätstyp (entity-type)

- Menge und Strukturbeschreibung gleichartiger Entitäten
- Symbol: Rechteck

Beziehungen und Beziehungstypen

- Entitäten können in unrichteten Beziehungen zueinander stehen
- Beziehungen können zusätzliche Attribute enthalten
- $B(E_1, E_2, \dots, E_k)$ bezeichnet die Menge aller Beziehungen B zwischen Entitäten
- Komplexität von Beziehungstypen

↳ 1:1 Beziehung: Jede Entität $e \in E_1$ ist höchstens einer Entität $e' \in E_2$ zugeordnet und umgekehrt

↳ n:1 Beziehung von E_1 nach E_2 : Jede Entität $e \in E_1$ steht in Beziehung zu keinem, einem oder mehreren Entitäten $e' \in E_2$.
Jede Entität $e \in E_1$ steht in Beziehung zu höchstens einer Entität $e' \in E_2$.

↳ n:m Beziehung: Jede Entität $e \in E_1$ kann mehreren Entitäten $e' \in E_2$ zugeordnet sein und
jede Entität $e' \in E_2$ kann mehreren Entitäten $e \in E_1$ zugeordnet sein

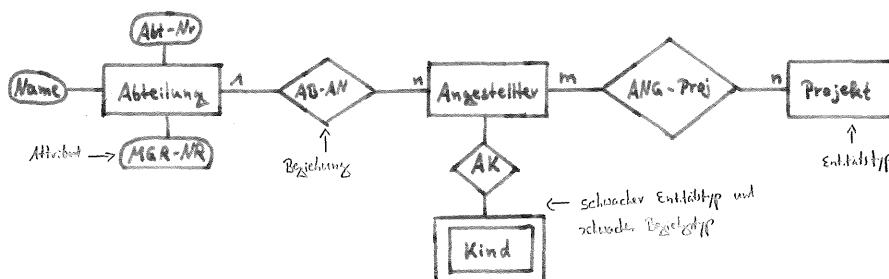
Schwache Entitätstypen

- Entität kann nur über Beziehungen identifiziert werden,
nicht allein durch eigene Attribute
- Symbol: doppelt umrandetes Rechteck

Schlüssel

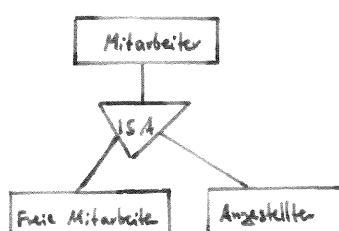
- Menge von Attributen, die Entitäten des Entitätstyps eindeutig identifizieren
- existiert mehrere Schlüssel, wird einer ausgewählt und als Primärschlüssel bezeichnet

- Beispiel



Erweiterungen ER-Modell

- Spezialisierung und Generalisierung (is-a-Relationship)



relationaler Datenmodell

KE2

- Realweltdaten werden als Sammlung von Relationen dargestellt

Definition: n-stellige Relation

Seien D_1, D_2, \dots, D_n nichtleere Mengen. Wir nennen $R \subseteq D_1 \times D_2 \times \dots \times D_n$ eine n -stellige Relation über den Mengen. Wir bezeichnen n als den Grad der Relation, $r = (d_1, d_2, \dots, d_n) \in R$ als ein n -Tupel der Relation R und d_i als i -te Komponente des Tupels.

Definition: Relationenschema

Ein Relationenschema $R(A_1, \dots, A_n)$ beschreibt mit paarweise verschiedenen Attributnamen A_i und zugehörigen Wertebereichen $\text{dom}(A_i)$ die Eigenschaften einer Relation. In einer Datenbank existiert zu jedem Zeitpunkt zu einem Relationenschema genau eine Relation.

Definition: erste Normalform (1.NF)

Alle Attribute einer Relation in erster Normalform sind elementar bzw. atomar. Können also nicht weiter aufgeteilt werden.

Definition: Schlüssel

Sei R eine Relation. Wir bezeichnen eine Menge von Attributen von R als Schlüssel von R , wenn die Attribute die Tupel der Relation eindeutig identifizieren und keine Teilmenge der Attributmenge dies ebenfalls leistet. Wir bezeichnen die Menge aller Schlüsselelemente einer Relation als Schlüsselkandidaten (candidate keys). Beim Schemaentwurf wird dann ein Primärschlüssel spezifiziert.

Modellierung im relationalen Datenmodell

- Schemata der Datenbank stellen die Gesamtheit der Relationenschemata, der Beziehungen von Relationen und die Menge der Integritätsbedingungen dar
↳ Entitytypen und Beziehungen werden als Relationen modelliert
- m:n-Beziehung wird als eigenständige Relation dargestellt
- 1:1- und 1:n-Beziehungen werden durch Hinzufügen des Schlüssels zur Relation auf die eindeutige Beziehbarkeit modelliert.

Datenmanipulationsprache (DML) für relationale Datenbanken

Relationenalgebra

- gewünschte Relation wird durch Operationsschritte auf anderen Relationen beschrieben
- Algebra := System nichtleerer Mengen mit darauf definierten Operationen

Machfolgender Operationsatz ist minimal und relational vollständig

↳ minimal := keine Operation kann durch andere Operationen dargestellt werden

↳ relational vollständig := Relationenalgebra kann mit operatoren die zellen Abfragen wie der Relationalkalkül ausdrücken

Mengenorientierte Operationen

Seien $R(R_1, \dots, R_n)$ und $S(S_1, \dots, S_n)$ Relationen.

Definition: Vereinigung verträglich

R und S sind Vereinigung verträglich, wenn $\{R_1, \dots, R_n\} = \{S_1, \dots, S_n\}$ gilt, also $n = m$ gilt und eine Permutation $T: \{1, \dots, n\} \rightarrow \{1, \dots, n\}$ existiert, so dass für alle $i \in \{1, \dots, n\}$ der Name und Wertebereich R_i und $S_{T(i)}$ gleich.

Definition: Vereinigung von R und S

Seien R und S Vereinigung verträglich.

$V = R \cup S$ ist die Vereinigung von R und S , wenn gilt $\Leftrightarrow V$ hat das alphabetisch geordnete Schema $(R_{1,1}, \dots, R_{1,n}, R_{2,1}, \dots, R_{2,m})$ mit $\{R_{1,1}, \dots, R_{1,n}\} = \{R_1, \dots, R_n\}$ und $(R_{j,1}, \dots, R_{j,n}) = (S_1, \dots, S_n)$ und $v = (a_{1,1}, \dots, a_{1,n}) \in V \Leftrightarrow (a_{1,1}, \dots, a_{1,n}) \in R \vee (a_{1,1}, \dots, a_{1,n}) \in S$.

Definition: Differenz von R und S

Seien R und S Vereinigung verträglich.

$D = R \setminus S$ ist die Differenz von R und S , wenn gilt: D hat das alphabetisch geordnete Schema $(R_{1,1}, \dots, R_{1,n})$ mit

$\{R_{1,1}, \dots, R_{1,n}\} = \{R_1, \dots, R_n\}$ und $(R_{j,1}, \dots, R_{j,n}) = (S_1, \dots, S_n)$ und

$d = (a_{1,1}, \dots, a_{1,n}) \in D \Leftrightarrow (a_{1,1}, \dots, a_{1,n}) \in R \text{ und } (a_{1,1}, \dots, a_{1,n}) \notin S$.

Definition: Kartesisches Produkt

Sei $\{R_1, \dots, R_n\} \cap \{S_1, \dots, S_m\} = \emptyset$.

$K = R \times S$ ist Kartesisches Produkt von R und S , wenn gilt: K hat das alphabetisch geordnete Schema $(K_{1,1}, \dots, K_{1,n+m})$ mit

$\{K_{1,1}, \dots, K_{1,n+m}\} = \{R_1, \dots, R_n\} \cup \{S_1, \dots, S_m\}$ und sei

$(K_{1,1}, \dots, K_{1,n}) = (R_1, \dots, R_n)$ und $(K_{j,1}, \dots, K_{j,m}) = (S_1, \dots, S_m)$ und

$k = (a_{1,1}, \dots, a_{1,n+m}) \in K \Leftrightarrow (a_{1,1}, \dots, a_{1,n}) \in R \text{ und } (a_{1,n+1}, \dots, a_{1,n+m}) \in S$.

relationenmanipulierende Operationen

Definition: Projektion (entfernt Attribute aus Relation)

Sei $\{R_1, \dots, R_k\} \cup \{R_{j_1}, \dots, R_{j_{n-k}}\} = \{R_1, \dots, R_n\}$ mit $\{R_1, \dots, R_k\} \cap \{R_{j_1}, \dots, R_{j_{n-k}}\} = \emptyset$.

$P = \Pi_{R_1, \dots, R_k}(R)$ ist die Projektion von R auf R_1, \dots, R_k , wenn P das Schema (R_1, \dots, R_k) hat.

Es ist $p = (a_1, \dots, a_k) \in P$, falls $\exists \{a_{j_1}, \dots, a_{j_{n-k}}\} \in \text{dom}(R_{j_1}) \times \dots \times \text{dom}(R_{j_{n-k}})$ mit $(a_1, \dots, a_n) \in R$.
(Die Projektion enthält keine Duplikate)

Definition: Selektion (entfernt Tupel anhand von Selektionsbedingung aus Relation)

Sei B eine Bedingung, in der nur Attribute von R als freie Variablen vorkommen und sei $B(t)$ der Wahrheitswert von B für ein Tupel $t \in R$.

$S = \sigma_B(R)$ ist die Selektion von R bzgl. B , wenn gilt: S hat das Schema (R_1, \dots, R_n) und $s = (a_1, \dots, a_n) \in S$, falls $S(s)$ und $B(s)$.

Definition: Umbenennung (Attribute einer Relation enthalten neue Bezeichnungen)

Seien (R'_1, \dots, R'_n) n Attribute mit $\text{dom}(R'_i) = \text{dom}(R_i)$ für alle $i \in \{1, \dots, n\}$.

$M = \rho_{R'_1, \dots, R'_n}(R)$ ist die Umbenennung der Attribute von R nach R'_1, \dots, R'_n , wenn gilt:

M hat das Schema (R'_1, \dots, R'_n) und $u \in M$, falls $u \in R$.

Alternative Schreibweise für kleine Attributanzahl: $M = \rho_{R'_1 \leftarrow R_1, \dots, R'_k \leftarrow R_k}(R)$

zusätzliche Operationen

Definition: Verbund / Join (Kombiniert Tupel zweier Relationen bzgl. einer Felder beider Relationen)

Sei $\{R_1, \dots, R_n\} \cap \{S_1, \dots, S_m\} = \emptyset$, $i \in \{1, \dots, n\}$ und $j \in \{1, \dots, m\}$ mit $\text{dom}(R_i) = \text{dom}(S_j)$.

Sei Θ ein zweistelliger Operator auf $\text{dom}(R_i)$, der einen Wahrheitswert liefert. Wir definieren

$V = R \bowtie_{R_i \Theta S_j} S$ ist der Verbund / Join von R und S bzgl. der Verbundbedingung $R_i \Theta S_j$ und

es gilt $V = \sigma_{R_i \Theta S_j}(RXS)$.

Definition: natürlicher Verbund / natural join (Kombiniert Tupel zweier Relationen mithilfe von Attributen mit gleichem Namen und Wertebereich)

Seien $(R_1, \dots, R_k) = (S_1, \dots, S_k)$ ($k > 0$) gleichnamige Attribute von R und S und seien

$\{S_{i_1}, \dots, S_{i_{m-k}}\} = \{S_1, \dots, S_m\} \setminus \{S_1, \dots, S_k\}$ exklusive Attribute von S .

$V = R \bowtie S$ ist ein natürlicher Verbund / natural join von R und S , wenn gilt:

$V = \Pi_{R_1, \dots, R_k, S_{i_1}, \dots, S_{i_{m-k}}} ($

$\sigma_{R_i = S_i} \wedge \dots \wedge R_k = S_k ($

$P(S_{i_1} \in S_{j_1}, \dots, S_{i_k} \in S_{j_k}) (S \times R))$

Relationalkalkül

- gewünschte Relation wird deskriptiv spezifiziert

↳ Es werden Prädikate (Bedingungen) angegeben, welche die Tupel der Relation erfüllen müssen

↳ Es werden keine Aussagen über die auszuführenden Operationen getätigt

⇒ Relationalkalkül ist eine formale Sprache zur Definition einer neuen Relation in Begriffen bestehender Relationen

- Relationalkalkül wird als Maßstab für die Mächtigkeit der Relationenalgebra verwendet

- Arten von Relationalkalkülen

↳ Wert-orientierter Relationalkalkül (Variablen bezeichnen Tupelkomponenten)

↳ Tupel-orientierter Relationalkalkül (Variablen bezeichnen Tupel)

- Aufbau einer Abfrage

$\{t \mid q\}$ mit $t :=$ Relationschema der gewünschten Relation und $q :=$ Prädikat, das alle Tupel der Relation erfüllen müssen

- Prädikat ist ein logischer Ausdruck aus Attributnamen, Konstanten, Vergleichsoperatoren, booleschen Operatoren, Quantoren und Tupelvariablen

- Tupelvariablen werden durch Quantor gebunden, ansonsten sind sie free

- jeder Relationenname in q , der nicht in t vorkommt muss durch eine gebundene Tupelvariable explizit definiert werden

- Beispiel:

RANGE ANGEST X ← explizit definierte Tupelvariable X

{(ANGEST.NAME) | ANGEST.Beruf = 'Programmierer'}

implizit definierte Tupelvariable ANGEST

$\exists X (X.Beruf = 'Programmierer' \wedge X.Gehalt \geq ANGEST.Gehalt)$

Erweiterungen zur Relationenalgebra / -Rekalkül

- Tupelsortierung
- arithmetische Operationen
- Aggregationsfunktionen (Anzahl Tupel, Durchschnitt, Summe, Maximum, Minimum)
- Update: Manipulation von Relationen

SQL (Structured Query Language)

- SQL basiert auf relationalen Modell, weicht aber in einigen Aspekten davon ab:

- ↳ Tabellen können Duplikate enthalten
- ↳ Aggregationsfunktionen werden angeboten
- ↳ NULL-Wert: jedes Attribut kann prinzipiell den Wert Null (= noch nicht festgelegt annehmen)
- ⇒ ternäre Logik mit Wahrheitswerten WAHR, FALSCH und UNBEKANNT

relationaler Modell	SQL
Relation	Tabelle (table)
Tupel	Zeile (row)
Attribut	Spalte (column)

DatenmanipulationsSprache (DML)

- Grundstrukturur SQL-Abfrage:

```
SELECT A1, ..., An
  FROM R1, ..., Rn
 [WHERE]
 [GROUP BY]
 [HAVING]
 [ORDER BY]
```

Unterabfragen (Subqueries)

- SQL-Abfragen (innere Abfrage) können in eine andere SQL-Abfrage (äußere Abfrage) geschachtelt werden
- Schachtelungstiefe theoretisch beliebig
- Korrelierte Subquery: Tabellen- und Spalten der äußeren Abfrage sind in der inneren Abfrage sichtbar und werden verwendet
- Arten von Subqueries
 - ↳ Skalare Subqueries
 - ↳ Zeilen-Subqueries
 - ↳ Tabellen-Subqueries

Mengenoperationen

- Schlüsselwort ALL angeben, um Duplikate beizubehalten
- Differenz: EXCEPT / MINUS } von vielen DBMS nicht unterstützt
- Durchschnitt: INTERSECT }
- Vereinigung: UNION

UPDATE

- Angabe der Daten als Zeile oder Zeilen-Multikabfrage
- WHERE-Klausel spezifiziert zu ändernde Datensätze

```
UPDATE <Tabellename>
SET <Attribut> = <Wert>
WHERE <Bedingung>
```

DELETE

- WHERE-Klausel spezifiziert zu lösende Datensätze

```
DELETE FROM <Tabellename>
WHERE <Bedingung>
```

INSERT

- Angabe der Daten als Zeile oder Zeilen-Multikabfrage

```
INSERT INTO <Tabellename>
```

```
VALUES (x1, ..., xn),
```

↳ Nachteil: Werte werden in definierter Tabellenreihenfolge eingetragen
⇒ Statement bei jeder Tabelleänderung anpassen

```
INSERT INTO <Tabellename> (Attribut1, ..., Attributn)
VALUES (x1, ..., xn)
```

Select

- Schlüsselwort DISTINCT eliminiert Duplikate
- Angabe aller Attribute mit Asterisk *
- Manipulation von Attributen möglich durch arithmetische oder Zeichenkettenoperatoren
- Mindesteinstellung von Attributen mit Schlüsselwort AS

FROM

- Bildung des Kartesischen Produkts durch Kommaseparierte Auflistung von Relationen: R1,...,Rn
- Bildung des Verbands
 - ↳ Inner Join: R1 inner join R2 on R1.X = R2.Y
 - ↳ Natural Join: R1 natural join R2 Using X
 - ↳ Outer Join: R1 [Left/Right/Full] outer join R2 on R1.X = R2.Y
 - ↳ Vorteil: enthalten auch Zeilen, die nicht die join-Bedingung entsprechen
 - ↳ Nachteil:
 - Syntax nicht standardisiert
 - hech Null-Constraints aus
 - nicht ermittelbar, ob Null-Wert am Outer Join resultiert
 - schlechten Performance als Inner Join
 - Verkettung kann zu Kognitiver Überlastung führen

WHERE

- Tupelselektion mit logischen, arithmetischen und Vergleichsoperatoren sowie Eigenschaftsoperatoren auf Einzelwerten (Substring, Like, Between)
- Schlüsselwörter zur Behandlung von Unterabfragen innerhalb der Where-Klausel
 - ↳ When EXISTS (Subquery): prüft, ob Subquery überprüft Werte enthält
 - ↳ When X > ALL / ANY (Subquery) bzu. When X in (Subquery)
 - ↳ erwartet immer einzige Liste

Group By

- Group-By-Klausel gibt ein oder mehrere Attribute an
- Tabelle wird bezüglich der Attribute partitioniert (in disjunkte Teilmengen zerlegt)
- Erlaubt das Anwenden von Aggregationsfunktionen in Select-Klausel auf Teilmengen

HAVING

- Schränkt die Betrachtung der durch Group-By-Klausel generierten Teilmengen ein

ORDER BY

- gibt ein oder mehrere Attribute an, nach denen die Tabelle jeweils aufsteigend (ASC) oder absteigend (DESC) geordnet wird

Aggregationsfunktionen (set functions)

- Erweiterung der relationalen Modelle
- SUM, AVG, COUNT, MAX, MIN
- EVERY: anwendbar auf Boolean-Spalten. Liegt False, wenn mindestens ein Wert False ist
- ANY/SOME: anwendbar auf Boolean-Spalten. Liegt True, wenn mindestens ein Wert True ist

Datendefinitionssprache (DDL)

CREATE TABLE

- Einschränkungsoptionen für Attributwerte: NOT NULL und UNIQUE
- Fremdschlüssel-Attribute dürfen nur Werte annehmen, die in der referenzierten Tabelle als Schlüssel vorkommen ("referentielle Integrität")
- ~~z.B.~~: Röhnen auch Parameter für das interne Repräsentationsschema angegeben werden

CREATE TABLE <Tabellename> (
 <Attribut1> <Wertebereich1> [Einschränkung],
 ...
 <Attribut n> <Wertebereich n> [Einschränkung],

{ [CONSTRAINT <Constraintname> PRIMARY KEY (<Attributliste>)],
 [CONSTRAINT <Constraintname> FOREIGN KEY (<Attribut x>) REFERENCES <Tabelle>, <Attribut>] };

DROP TABLE

DROP TABLE <Tabellename>;

CREATE VIEW

- Views sind von den Basis-tabellen abgeleitete, nicht physisch gespeicherte Tabellen, die eine spezielle Darstellung von Daten für Benutzer liefern
- Inhalte der Views werden zum Zugriffspunkt aus Basis-tabellen ermittelt
- Views können ebenfalls geschachtelt werden
- Benutzer (meist Anwendungsprogramme) können mit Ausnahme von Datenänderungen die gleichen Operationen wie auf Tabellen ausführen

CREATE VIEW <Viewname> AS
 <Select-Statement>

prozedurale bzw. imperativer Erweiterungen

- mögliche Erweiterungen von kommerziellen SQL-Dialekt-en durch prozedurale Sprachelemente
 ↳ Keine Einheitlichkeit im Aufbau und Syntax (Beispiel/diskussion anhand PL/SQL von Oracle)

Prozeduren und Funktionen

- bieten Benutzer individuell angepasste Operationen
- Vorteile:
 - + Röhnen mit Parametern, wieder verwendbar
 - + werden datenbank-spezifisch definiert
 - ↳ DBMS kann vorarbeiten und Zugriffspfade ermitteln
 - ↳ Anwendungen müssen nur Aufruf über den Nutzenreiter übernehmen
 - ↳ Reduktion von Übertragungszeit und Bandbreite
 - + Interaktion mit Datenbank auf Prozeduraufgabe einschränkbar

```
CREATE <PROCEDURE/FUNCTION> <Name> (<Parameterliste>)
[RETURN <Typ des Rückgabewerts>]
IS
BEGIN
  <Anweisungen>
END;
```

Trigger

- Anwendungszweck, die nicht explizit, sondern durch das Eintreten eines Ereignisses ausgelöst wird (z.B. Änderung von Datenbeständen)
- ~~z.B.~~ angeben, ob Trigger pro Operation oder pro geänderte Zeile ausgeführt wird

CREATE TRIGGER <Triggername> [BEFORE / AFTER]

[INSERT / UPDATE / DELETE] ON <Tabellename>

[FOR EACH ROW]

BEGIN

<Anweisungen>

END;

Cursor

- Hilfsmittel der imperativen Programmierung (tupel-orientiert) zum Zugriff auf SQL-Ergebnisse (mengenorientiert)

CURSOR <cursorname> IS <Select-Anweisung>; ← cursor definieren

OPEN <cursorname>; ← cursor öffnen

FETCH <cursorname> INTO <Variable>; ← Zugriff auf aktuelles Tupel

WHILE <cursorname> %FOUND

LOOP

<Anweisungen>

END LOOP;

Konsistenz, Transaktion und Recovery

- Konsistenz := Korrektheit und Vollständigkeit der Informationen

↳ mögliche Ursachen für Inkonsistenzen:

- ↳ Fehler in Anwendung ↳ Eingabefehler (durch Integritätsbedingungen vermeidbar)
- ↳ Falsche Behandlung paralleler Anwendungsabläufe
- ↳ Absturz des DBMS oder Betriebssystems
- ↳ Ausfall des Mainmemory
- ↳ Falsche Schemadefinition

- Aufgaben des DBMS

- ↳ Sicherstellen, dass parallel ausgeführte Aktionen keine Fehler verursachen (Transaktionskonzept)
- ↳ Datenbank nach dem Auftreten von Fehlern wieder in konsistentem Zustand versetzen (Wiederherstellung / Recovery)

Transaktionen und Transactionsmanagement

- Transaktion := Befehlspaket, die entweder vollständig und konsistent oder überhaupt nicht ausgeführt wird. Alle vor dem Auftreten eines Fehlers ausgeführten Befehle der Transaktion werden rückgängig.

- Für eine Transaktion und ihre Eigenschaften gilt das ACID-Prinzip

Atomicity (Unteilbarkeit): Transaktion ist eine unteilbare Ausführungseinheit ("ganz oder gar nicht")

Consistency (Konsistenz): Transaktion überführt eine konsistente Datenbank in einen konsistenten Zustand

Isolation (Isolation): Transaktion arbeitet isoliert, Zwischenstücke sind nach außen nicht sichtbar

Durability (Dauerhaftigkeit): Ergebnis abgeschlossener Transaktionen sind dauerhaft und überleben nachfolgende Fehler

- Transaktionen werden implizit durch SELECT, INSERT, ALTER, CREATE, DROP, GRANT, REVOKE, OPEN, CLOSE, FETCH, UPDATE, DELETE gestartet
- Transaktionen werden explizit durch START TRANSACTION begonnen und durch COMMIT, COMMIT WORK oder COMMIT WORK AND NOJ CHAIN beendet
- Bei einem Fehler oder dem Abbruch einer Transaktion werden die bisherigen Änderungen mit ROLLBACK [WORK] rückgängig gemacht
- Durch SAVEPOINTS können Befehle innerhalb einer Transaktion mit ROLLBACK TO SAVEPOINT bis zu einem vorher definierten Punkt rückgängig gemacht werden

START TRANSACTION

READ WRITE < Transactionmodus >

ISOLATION LEVEL REPEATABLE READ

DIAGNOSTIC SIZE 10

< Code in Wortsprache und SQL >

IF < Bedingung > THEN ROLLBACK ELSE

< Code in Wortsprache und SQL >

COMMIT

START TRANSACTION

< SQL-Statement >

SAVEPOINT < Savepointname >

< SQL-Statement >

IF < Bedingung > THEN ROLLBACK TO SAVEPOINT < Savepointname >

- Transactionsmanager stellt Unterteilbarkeit und Isolation der Transaktionen sicher (i.d.R. durch das Sperren von Objekten)

↳ Benutzer erhalten immer eine konsistente Sicht auf die Datenbank, Inkonsistenzen durch Parallelität werden verhindert

Recovery und Recovery-Manager

- Recovery := Wiederherstellen des letzten konsistenten Zustands der Datenbank nach Auftreten eines Fehlers

- Recovery-Manager

↳ legt Backups der Datenbestände an

↳ protokolliert Transaktionen in Logfile und spielt alle abgeschlossenen Transaktionen nach der Wiederherstellung wieder in Datenbank ein

Kopplung von SQL an eine Programmiersprache

- SQL kann durch verschiedene Kopplungsmöglichkeiten mit einer Wortsprache verwendet werden

dynamischer SQL

- SQL-Anweisungen werden zur Laufzeit als String zusammengefasst und an DBMS weitergeleitet

embedded SQL

- SQL-Anweisungen werden direkt im Anwendungsprogramm eingebettet

↳ Precompiler muss den SQL-Kontext vorher behandeln, damit der Compiler der Wortsprache das Programm übersetzen kann

- Nachteil: Relationenschemata und Abfragen müssen vor Übersetzung des Programms bekannt sein

- Beispiele:

program X

begin

EXEC SQL BEGIN DECLARE SECTION
 VARIABLE name: string;

EXEC SQL END DECLARE SECTION

name := read;

EXEC SQL EXECUTE IMMEDIATE

 Delete from X where Name := name;

program end

Deklarative Programmvariablen a1, ..., an

EXEC SQL DECLARE C CURSOR FOR <Select-Anweisung>

EXEC SQL OPEN C

While „Cursor noch nicht am Ende“ do

 EXEC SQL Fetch C INTO : a1, ..., an

 „Verarbeiten der Werte“

END WHILE

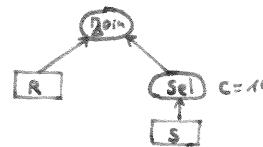
EXEC SQL CLOSE C

Abfrageoptimierung

- Abarbeitungsart einer Abfrage entscheidet über notwendigen Aufwand \rightarrow Abfrageoptimierung zeiten Query-Prozess notwendig
- Realität: Systeme bestimmen eine mit hoher Wahrscheinlichkeit gute Ausführungsstrategie
- Faktoren für Abfrageoptimierung:
 - \hookrightarrow Abfragen äquivalent umformen
 - \hookrightarrow statistische Daten beachten
 - \hookrightarrow vorhandene Zugriffspfade beachten
 - \hookrightarrow gepl. Sortierung der Tupel einer Relation beachten

- Abfragen können mithilfe eines Operatorbaums ausgedrückt werden:

\hookrightarrow Rechtecke = Relationen \hookrightarrow Ellipsen = Operationen \hookrightarrow Beispiel: $R \bowtie_{A=0}^{C=10} (S)$



Algebraische Optimierung

- möglichst günstige Ausführung von ganzen Abfragen
- heuristische Optimierungen

- 1.) Selektionen auf gleichen Operanden zu komplexen Selektionen zusammenfassen
- 2.) Selektionen möglichst früh ausführen (möglichst nah zu Blattknoten des Operatorbaums)
- 3.) Projektionen ohne notwendige Duplizitateminierung so spät wie möglich, aber nicht vor einer Selektion ausführen
- 4.) Projektionen mit notwendiger Duplizitateminierung so spät wie möglich
- 5.) Bei gemeinsamen Teilknoten des Operatorbaums prüfen, ob Speicher und Lese von Sekundärspeicher günstiger als die Berechnung ist.

physische Optimierung

- möglichst günstige Ausführung einzelner Operationen (Zugriffspfade & statistische Gruppen entscheiden)

Sekundärindex

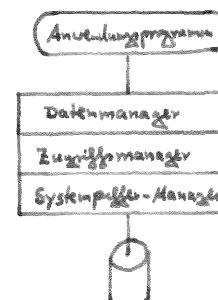
- Datenstruktur zur Zugriffsbeschleunigung auf einzelne Datensätze
- hat keinen Einfluss auf die physische Speicherung der Originaldaten
- Verzeichnis von Tupeln ($S_i p$) mit S_i = Merkmalindex und p = physische Speicherort des Datensatzes mit Merkmalsausprägung S
- bei Änderungen des Datenbestands muss i.d.R. auch der Sekundärindex angepasst werden
 - \hookrightarrow Zugriffsbeschleunigung gegen erhöhte Änderungsrate abwägen

Sortierung von Relationen nach gewähltem Attribut

- bei Verwendung effizienter Sorteralgorithmen ist die Sortierung von n Tupeln in $(\log n)$ Schritten möglich
- Sortierung erleichtert z.B. Verbindlichkeiten durch das parallele Durchlaufen der beiden Relationen

Implementierung eines relationalen Datenbanksystems

- Datenbanksysteme werden typischerweise in Schichten realisiert
- bei Datenbanksystemen zu realisieren:
 - \hookrightarrow Abbildung von Relationen auf speicherbezogene Datenstruktur
 - \hookrightarrow funktionale Komponenten anordnen
 - \hookrightarrow Abfrageoptimierer
 - \hookrightarrow Autorisierung & Integritätskontrolle
 - \hookrightarrow Recovery
 - \hookrightarrow Transaktionen



Mengenschnittstelle (Relationen, Tupel, Sichten)
Ein-Tupel-Schnittstelle (Tupel, Indizes)
Seitenschnittstelle

Softwarearchitektur eines DBMS

Systempuffer-Manager

- 负责将物理的从属系统中的物理块从外部存入到工作存储器中
- 在最近的上层中，它将工作存储器地址与块的物理地址映射
- 目标：尽可能少地将块保留在工作存储器中
- 任务：
 - 当内存满时，决定哪些块应该被替换
 - 与恢复一起，将不再需要的块写回外部存储器（“pinned page”）
 - 与恢复一起，偶尔将块写回外部存储器，即使它们在系统缓冲区中没有被使用（“forward output”）

Zugriffsmanger

- 负责物理实现的连接路径和元组表示
- 提供一个连接请求给系统缓冲区管理器
- 在最近的上层中，它通过“单元格连接”连接元组和逻辑连接路径
- 主要对象：元组和索引（定义逻辑连接到元组上的关系，从而直接访问元组）
- 操作：
 - 连接到元组上的属性值
 - 连接到元组在连接路径上的位置（“Find Next”）
 - 在连接区域中提供元组
 - 插入和删除元组
 - 修改元组属性值
 - 读取、插入和删除模式的元组条目
 - 插入和删除连接路径
 - 事务管理

Datenmanager

- bietet relationale Schnittstelle
 - ↳ Objekte: Relationen und Tupel
 - ↳ Operatoren: relationale Sprache (z.B. SQL)

- Aufgaben:
 - ↳ Anweisungen der relationalen Sprache in Aufgabe der Zugriffsmethoden abbilden
 - ↳ Zugriffsberechtigungen der Benutzer kontrollieren
 - ↳ Integritätskontrollen durchführen

Datenbank - Entwurfssystem

- Datenbanken werden i.d.R. als Teil eines größeren Informationssystems entworfen
- Datenbankentwurf wird grundsätzlich unabhängig von einem konkreten Datenmodell und einer konkreten Datenbank realisiert
- ↳ Ergebnis ist ein semantisches Datenmodell (z.B. ER-Diagramm), das anschließend in ein relationales Datenbank-Schema umgewandelt wird

Software - Entwurfssystem

- Phasen des Software - Entwurfssystems
 - Machbarkeitsanalyse (wirtschaftlich & technisch durchführbar? Kosten-Nutzen-Verhältnis?)
 - Anforderungen durch Interaktion mit künftigen Nutzern erfassen und analysieren
 - Anwendung- und Datenbanksysteme entwerfen
 - Implementierung und Testen der Anwendung- und Datenbanksysteme
 - Validierung und Abnahmetests (ergibt das System die Benutzer- und Leistungsanforderungen?)
 - Installation, Betrieb und Wartung

Datenbank - Entwurfssystem

- Enger Zusammenhang zum Software-Entwurfssystem → häufig Änderungen und Erweiterungen zu erwarten
- Ziele des Entwurfs:

- ↳ Anforderungen an Dateninhalt erfüllen
- ↳ natürliche und verständliche Strukturierung der Informationen
- ↳ Verarbeitungsanforderungen (Reaktionszeit, Verarbeitungszeit etc.) unterstützen

- Ergebnis:
 - ↳ Datenbankstrukturen als Basis für die Implementierung
- Phasen des Datenbank - Entwurfssystems

- Anforderungen erfassen und analysieren
- Konzeptionelle Datenbankentwurf
- Wahl des DBMS
- Entwurf auf Datenmodell („logischer Entwurf“) abbilden
- physischer Datenbank - Entwurf
- Datenbanksystem implementieren und tunen

Konzeptioneller Datenbankentwurf

- Ziel: erstelle ein von bestimmten Datenbanksystemen unabhängiges Datenbankschema, dargestellt durch konzeptionelle Modelliersprache (z.B. ER-Modell)
- Vorteile der DBMS-Unabhängigkeit
 - ↳ Konzeptionelle Entwurf wird nicht durch spezielle Einschränkungen einzelner Datenbanksysteme beeinflusst
 - ↳ Stabilität des konzeptionellen Entwurfs bei Änderung des eingesetzten Datenbanksystems
 - ↳ Wichtige Kommunikationsplattform zwischen Datenbank - Designern, Anwendungsentwicklern und Benutzern

Teilaufgaben des Entwurfs

- auf Basis der Interaktion mit den Fachexperten sind folgende Teilaufgaben zu leisten
 - ↳ Schema-Komponente (Entitäten, Beziehungstypen, Attribute etc.) identifizieren
 - ↳ Spezialisierung- und Generalisierungshierarchien erstellen
 - ↳ Schlüsselattribute, Beziehungskardinalitäten und Constraints definieren

Ansätze zum Entstellen komplexer Datenbankschemata

↳ Zentraler Schema-entwurf

- auf Basis der in Phase 1 gesammelten Gesamtauforderungsmenge wird ein Schema entworfen
- die Anforderungen der einzelnen Benutzergruppen und Anwendungen werden in externen Views realisiert

↳ View - Integration

- Anforderungen einzelner Anwender werden in konzeptionelle Teilschemata umgesetzt und anschließend in das globale konzeptionelle Schema integriert
 - ↳ Teilschemata können als externe Views angekoppelt werden
- Integrationsphase:

↳ Ziel: Modellierungunterschiede der Teilschemata durch Identifikation gemeinsamer Realweltkonzepte konkren zu die Integration darstellen.

Anschließend Redundanzen und unnötige Komplexität aus dem globalen Schema eliminieren

↳ mögliche Modellierungunterschiede

- Synonyme: unterschiedliche Benennung gleicher Konzepte
- Homonym: gleiche Benennung unterschiedlicher Konzepte
- Modellierung eines Konzeptes durch unterschiedliche Modellierungskontexte
- Zuordnung unterschiedlicher Wertemengen
- Definition unterschiedlicher Constraints

— parallel zum Schemaentwurf werden die Transaktionen mit Prozess-Modellierungsmethoden wie z.B. UML (Unified Modeling Language) spezifiziert

Abbildung von ER-Diagrammen in relationale Schemata

- Konzeptuelles Schema in Form eines ER-Diagramms wird mithilfe von Transformationssregeln auf Modell des ausgewählten Datenbanksystems abgebildet

Transformation starker Entitätsarten

- starker Entitätsart wird in eine nach dem Entitätsart benannte, eigene Tabelle überführt; die Schlüssel- und Nichtschlüsselattribute des Entitätsarts enthalten

Transformation schwacher Entitätsarten

- schwacher Entitätsart wird in eigene Relation überführt

- Primärschlüssel des den schwachen Entitätsarten identifizierenden Entitätsarten („Owner-Typ“) wird als Fremdschlüssel in die Tabelle eingefügt
 \hookrightarrow Primärschlüssel des schwachen Entitätsarten = Fremdschlüssel Owner + Partialschlüssel des schwachen Entitätsarten

Transformation von 1:1-Beziehungstypen

- Primärschlüssel des einen Entitätsarten wird als Fremdschlüssel des anderen Entitätsarten eingefügt

Transformation von 1:n-Beziehungstypen

- Primärschlüssel der Entitätsarten der 1-Seite wird als Fremdschlüssel in die Tabelle der Entitätsarten auf der n-Seite eingefügt

Transformation von n:m-Beziehungstypen

- n:m-Beziehung zwischen zwei Entitätsarten wird in eine eigene Relation überführt

- Primärschlüssel beider Entitätsarten werden als Fremdschlüssel in Relation eingefügt

- Primärschlüssel der Relation = Kombination der Fremdschlüssele

Transformation mehrwertiger (= zwischen mehr als zwei Entitätsarten) Beziehungstypen

- wird in eigene Relation durchgeführt

- Primärschlüssel aller beteiligten Entitätsarten werden als Fremdschlüssel in Relation eingefügt

- Primärschlüssel der Relation = Kombination der Fremdschlüssele

Transformation von Super- und Subtypenhierarchien

- es existieren mehrere Transformationsmöglichkeiten in Abhängigkeit vom späteren Zugriffsvorhaben der Datenbank

- alle Entitätsarten der Hierarchie werden in eigene Relationen überführt

- jede Relation enthält als Attribute die auf der entsprechenden Stufe definierten Eigenschaften

- Subtyp-Relationen enthalten als Fremdschlüssel den Primärschlüssel des jeweiligen Supertyps

Anomalien eines Relationenschemas

- Beispiel: Abonnement (Kunde, Adresse, Zeitschrift, Preis) mit {Kunde, Zeitschrift} als einzigen Schlüssel

Einfügen-Anomalie (Insertion Anomaly)

- Einfügen neuer Tupel unter Umständen nicht möglich

\hookrightarrow Beispiel: Aufnahme eines Kunden, der sich noch nicht endgültig für eine Zeitschrift entschieden hat ($Zeitschrift = \text{null}$ \hookrightarrow Schlüsselkonflikte)

Löschen-Anomalie (Deletion Anomaly)

- Löschen von Tupeln führt unter Umständen zum Verlust nicht unmittelbar zusammenhängender Informationen

\hookrightarrow Beispiel: Löschen des letzten Abonnements einer Zeitschrift löscht auch Information über die Existenz der Zeitschrift

Andern-Anomalie (Update Anomaly)

- Zum Ändern einer Eigenschaft sind sehr viele Änderungsoperationen nötig

\hookrightarrow Beispiel: Preisänderung einer Zeitschrift muss für alle entsprechenden Abonnement-Tupel durchgeführt werden

- Das Hauptziel der Entwurfstheorie relationaler Datenbanken ist das Erarbeiten von Schemakriterien, die Anomalien verhindern

Funktionale Abhängigkeit

Definition: Funktionelle Abhängigkeit (functional dependency) zwischen Attributen einer Relation

Sei $R(A_1, \dots, A_n)$ ein Relationenschema und seien X, Y Teilmengen von $\{A_1, \dots, A_n\}$. Wenn es keine Relation von Typ R geben kann, in der zwei Tupel denselben Wert für X, aber unterschiedliche Werte für Y haben, so nennen wir Y funktional abhängig von X und schreiben $X \rightarrow Y$.

Alternativ (unoffiziell): Für jede Relation von Typ R existiert eine Abbildung von X nach Y.

Definition: Fd-Menge

Sei $R(A_1, \dots, A_n)$ ein Relationenschema. Wir nennen die Menge aller in R vorkommenden funktionellen Abhängigkeiten Fd-Menge.

Definition: Closure von F

Sei F die Fd-Menge eines Relationenschemas R . F kann neue funktionale Abhängigkeiten implizieren und man sagt: dann $F \rightarrow Y$ impliziert, wenn $X \rightarrow Y$ in allen Relationen eines Relationenschemas gilt, in denen auch F gilt.

Wir nennen die Menge aller von F implizierten funktionellen Abhängigkeiten die Closure von F und schreiben F^+ .

Definition: voll funktional abhängig

Sei F eine Fd-Menge und sei $X \rightarrow Y \in F^+$ eine funktionale Abhängigkeit. Wir nennen Y voll funktional abhängig von X, wenn es keine echte Teilmenge $X' \subset X$ mit $X' \rightarrow Y \in F^+$ gilt.

Definition: Schlüssel

X ist Schlüssel von $\{A_1, \dots, A_n\}$, wenn $X \rightarrow \{A_1, \dots, A_n\} \in F^+$ ist und $\{A_1, \dots, A_n\}$ voll funktional abhängig von X ist.

Definition: (Nicht-)Schlüsselattribut

Wir nennen A ein Schlüsselattribut des Relationenschemas R, wenn A Element irgendeines Schlüssels von R ist.
 Andernfalls nennen wir A Nichtschlüsselattribut.

Satz: Prüfe, ob gegebene funktionale Abhängigkeit in F^+ enthalten ist

Es gilt: $X \rightarrow Y \in F^+ \Leftrightarrow Y \subseteq cl_F(X)$

Berechnung von $cl_F(X)$:

- 1) initialisiere $cl_F(X) := \{X\}$
- 2) $Z \rightarrow Z \in F$ und $Z \subseteq cl_F(X) \Rightarrow cl_F(X) := cl_F(X) \cup \{Z\}$
- 3) wiederhole Schritt 2, bis keine Attribute mehr zu $cl_F(X)$ hinzugefügt werden können

Satz: Prüfe, ob zwei FD-Mengen die gleiche Closure besitzen

Seien F und H FD-Mengen. Es gilt $F^+ \subseteq H^+ \Leftrightarrow \forall X \rightarrow Y \in F: X \rightarrow Y \in H^+$

Normalisierung

- durch Umwandlung der Schemata in die Normalformen sollen die vorgenommenen Anomalien verhindert werden
- Normalformen sind auf Basis der funktionalen Abhängigkeit definiert

Definition: erste Normalform (1NF)

- Werte des Wertebereichs jedes Attributs sind unterteilbar, d.h. sie besteht ihrerseits nicht aus Mengen oder n-Tupeln mit $n > 1$.

Definition: zweite Normalform (2NF)

- jedes Nichtschlüsselattribut ist voll funktional abhängig von jedem Schlüssel der Relation

Definition: dritte Normalform (3NF)

Eine Relationsschema R mit FD-Menge F ist in dritter Normalform, wenn gilt:

$\forall X \rightarrow A \in F^+$ mit $A \notin X: X$ enthält Schlüssel für R oder A ist Schlüsselattribut.

Umformulierung: dritte Normalform ist verletzt, wenn es eine funktionale Abhängigkeit $X \rightarrow A$ gibt, so dass X Nichtschlüssel und A Nichtschlüsselattribut ist

Definition: Boyce-Codd Normalform (BCNF)

Eine Relation R ist in Boyce-Codd Normalform, wenn gilt: $\forall X \rightarrow A \in F^+$ mit $A \notin X: X$ enthält Schlüssel für R.