Techn.Inf. II Zusammenfassung

Andreas Biri, D-ITET 30.07.14

1. Betriebssysteme

OS: Mittler zw. User & Hardware, verwaltet Ressourcen, abstrahiert Hardware & bietet Umgebung für Programme

Dateisystem
- Datei ist eine Abfolge von Bytes ohne Struktur
- gibt auch strukturierte Dateien
- Statusinfo: Name d. Datei, Länge, Zugriffsarten, Sektoren

Verkette Liste von Sektoren
- Verzeichniseintrag enthält ersten Sektor
- In einer Tabelle ist der Folgesektor verzeichnet
- Spezielle Markierungen für Dateiende / frei / corrupted

Kontextwechsel
Man merkt sich, „wo man gerade war" vor Unterbrechung
Nach Behandlung d. Interrupts Zustand wiederherstellen

Stack: Speicher für Eigenschaften v. Programmen (SP+FP)
Heap: Speicher für dynamische Variablen (*malloc, new*)

Virtueller Speicher
jeder Prozess sieht komplett freuen Speicher (virtueller Adressraum, kann durch Auslagern sogar grösser als physikalisch vorhandener Speicher sein; von OS verwaltet)

Quantities of bytes					
SI prefixes (decimal)			IEC prefixes (binary)		
Legacy use (often with KB for kB)					
Value	Name		Value	Name	
$1000^1 = 10^3$	kilobyte	(kB)	$1024^1 = 2^{10} = 1.024 \cdot 10^3$	kibibyte	(KiB)
$1000^2 = 10^6$	megabyte	(MB)	$1024^2 = 2^{20} \approx 1.049 \cdot 10^6$	mebibyte	(MiB)
$1000^3 = 10^9$	gigabyte	(GB)	$1024^3 = 2^{30} \approx 1.074 \cdot 10^9$	gibibyte	(GiB)
$1000^4 = 10^{12}$	terabyte	(TB)	$1024^4 = 2^{40} \approx 1.100 \cdot 10^{12}$	tebibyte	(TiB)
$1000^5 = 10^{15}$	petabyte	(PB)	$1024^5 = 2^{50} \approx 1.126 \cdot 10^{15}$	pebibyte	(PiB)
$1000^6 = 10^{18}$	exabyte	(EB)	$1024^6 = 2^{60} \approx 1.153 \cdot 10^{18}$	exbibyte	(EiB)
$1000^7 = 10^{21}$	zettabyte	(ZB)	$1024^7 = 2^{70} \approx 1.181 \cdot 10^{21}$	zebibyte	(ZiB)
$1000^8 = 10^{24}$	yottabyte	(YB)	$1024^8 = 2^{80} \approx 1.209 \cdot 10^{24}$	yobibyte	(YiB)

2. Prozesse und Threads

Gleichzeitig: Prozesse überlappen sich in einer Stufe

Multitasking: Scheinbare simultane Ausführung mehrerer Aufgaben (quasi parallel durch Wechsel)

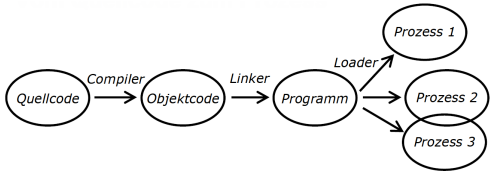
Echtzeitsystem: Optimierung auf geforderte Reaktionszeit

Prozesssystem: mehrere gleichzeitige und kooperierende Prozesse (bedingt Kommunikation)

FIFO: Verarbeitung kontinuierlich eintreffender Daten
Zwei unabhängige Prozesse für Einlesen & Verarbeiten, „First-In First-Out"-Zwischenspeicher dient z. Entkoppeln

Benötige drei Variablen: *front, rear, length* (2: voll/leer?)
$\lambda_w : Schreibrate$, $\lambda_r : Leserate$
$Durchschnittliche\ Länge$: λ_w / ($\lambda_r - \lambda_w$)

LIFO: „Last-In First-Out" beschreibt Stack / Kellerspeicher



Programm: Menge von Instruktionen für einen Computer, um spezifische Aufgabe auszuführen; beschreibt Prozess

Prozess: Programm in Ausführung / Instanz d. Programms; benötigt Ressourcen (CPU, Speicher, I/O, Dateien)
Hat Zustand (Variablen, Register, Programmzähler etc.)

Scheduler: jeder Prozess hat ganze Hardware „für sich", Scheduler teilt die CPU den Prozessen zu

2.2 Prozesse und deren Verwaltung

Aufgaben des Betriebssystems
- Erzeugen und Entfernen von Prozessen (Ressourcen)
- Anhalten und Fortsetzen (Scheduler)
- Mechanismen für Synchronisation & Kommunikation
- Erkennung & Behebung von Fehlersituationen

Prozess im Arbeitsspeicher

Stack: Linear gestapelter Aktivierungsrahmen (stack frames) von Unterprogrammen

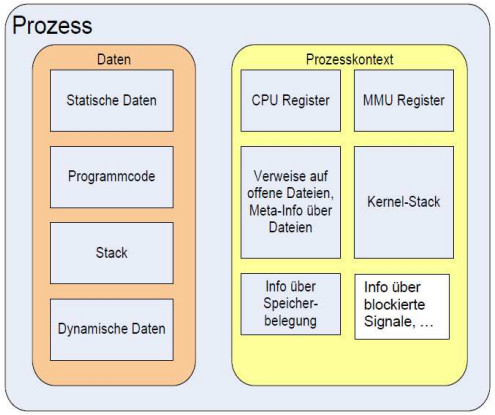
Heap: Dynamischer Speicher (zur Laufzeit alloziert)

Data: Zur Compilezeit bekannter Sp., zB. globale Variablen

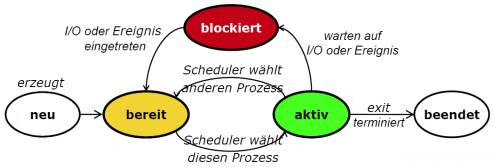
Text: Ausführbare Instruktionen (code)

Aufbau eines Stack Frames (Aktivierungsrahmen)

Stackpointer (SP): markiert das Ende d. aktuellen Frames
Framepointer (FP): zeigt auf das aktuelle Frame, erlaubt Zugriff auf Variablen & Parameter



Lebenszyklus eines Prozesses

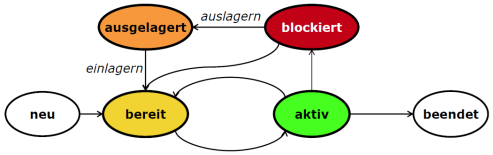


Neu: Für Prozess notwendigen Datenstrukturen wurden erstellt und im OS vermerkt, zugehöriges Speicherbild ist im Speicher oder wird bei Bedarf aus Datei geladen

Blockiert: Prozess wartet auf Nachricht / Zeitsignal / I/O

Terminierung: Ressourcen sind wieder freigegeben
- freiwillig: Normales Ende, Fehlerausgang
- unfreiwillig: Fataler Fehler, Abbruch d. anderen Prozess

Auslagern: inaktiver Prozess wird nicht mehr für Scheduling bestimmt, bestimmte Betriebsmittel (Hauptspeicher) werden auf externe Speicher ausgelagert



User-Mode vs. Kernel-Mode
Privilegierte Instruktionen nur im Kernel-Mode ausführbar

Process Control Block (PCB)
Datenstruktur für Verwaltung, beschreibt Prozesskontext

Link (Verkettung mit andern PCB), Prozesszustand, Prozess-identifikation, Programmzähler, CPU-Register, Speicherbild, Offene Files, Verrechnungsinfo, I/O-Zustand

Teils auslagerbar: Prozessorzustand, Systemaufruf, HS-Inhalt
Nicht auslagerbar: Scheduling-Paras, Prozesszustand, Signale

Umschaltung: Zustand des laufenden Prozesses wird in PCB gespeichert, PCB des anderen Prozesses wird geholt

Verkettete Listen zur Verwaltung von PCB:
ready queue sowie „I/O-Device Queues" für jedes I/O-Gerät

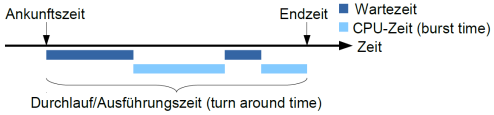
Prozessvergabelung fork(): Prozess gabelt sich auf in zwei Kontrollflüsse mit gleichem Code (PCB wird an Kind vererbt)
wait() : blockiert, bis Kindprozess seinen Status ändert
falls Parent vorher terminiert, wird Kind zum Zombie

Init-Prozess: erster, startet alle andern Prozesse, Prozess-ID 1

2.3 Prozess-Scheduling

Langzeit-Scheduling: Welche Prozesse sollen wann im Speicher bzw. ausgelagert werden? Kriterien:
- Optimierung d. Gesamtdurchsatzes, Priority, Wunsch

Kurzzeit-Scheduling: Zuteilung der CPU zu den aktuell im Speicher befindlichen Prozessen; Ziel:
- Durchsatz (Verarbeitungsaufgaben / Zeiteinheit)
- Möglichst hohe CPU-Auslastung
- Fairness: Gleichbehandlung aller gleichartigen Prozesse
- Durchlaufzeit der Prozesse ($t_{creation} - t_{termination}$)
- Wartezeit von ausführbereiten Prozessen
- Antwortzeit / Reaktionszeit auf Ereignisse



Ankunftszeit (arrival time): Zeitpunkt, ab dem Prozess bereit zur Ausführung ist

CPU-Zeit (burst-time): Benötigte CPU-Zeit für Ausführung

Durchlaufzeit: Zeitdauer, bis Prozess vollständig ausgeführt

Wartezeit (waiting time): *Kumulierte* Zeitdauer, in der der Prozess zwar bereit ist, aber nicht ausgeführt wird

Reaktionszeit/Antwortzeit (response time): Zweit zwischen Eingabe und Übergabe der Antwortdaten an Ausgabe

Zeitquantum (time quantum): maximaler CPU-Zeit-Slot

Scheduler-Aktivierung: wird aktiviert, falls
- laufender Prozess sich selbst blockiert (I/O, System-Call)
- laufender Prozess in den Zustand „bereit“ versetzt wird
- blockierter Prozess in den Zustand „bereit“ versetzt wird
- ein Prozess terminiert oder freiwillig in „bereit“ übergeht

I/O intensiv (I/O-bound): wenige CPU-Bursts, viele I/O
CPU intensiv (CPU-bound): viele CPU-Bursts, wenige I/O

Scheduling-Methoden

Non-preemptive: „Prozess läuft so lange, wie er will“
- nur I/O, Terminierung & freiwillige Aufgabe beenden

Preemptive: „Jeder Prozess läuft nur so lange, wie er darf“
- Neuzuteilung kann auch durch Interrupt & I/O geschehen

First-Come, First-Served (FCFS): non-preemptive
Prozesse werden nach der Reihenfolge ihres Eintreffens im Zustand „bereit“ in eine Warteschlange eingereiht

Shortest Job First (SJF): non-preemptive & preemptive
Der Prozess mit der kürzesten (erwarteten) CPU-Belegung wird zuerst ausgeführt; optimiert mittlere Wartezeit

Modell für Voraussage der erwarteten CPU-Belegung:
gewichteter exponentieller Mittelwert: $T_{n+1} = \alpha\, t_n + (1-\alpha) T_n$
T_n : $letzter\ Schätzwert$ T_0 : $Konstante$
t_n : $aktuell\ gemessener\ Wert$ α : $Gewichtung$

Scheduling nach Prioritäten
- Warteschlange wird nach Prozesspriorität bewirtschaftet
- Alternativ: Pro Priorität eine Schlange, wähle aus höchster nicht-leerer Schlange per Round Robin
- kann preemtive und non-preemptive sen (siehe SJF)
- Problem: **Verhungern / Starvation** (kommt nie dran)
  Lösung: **Altern / Aging** (erhöhe Priorität mit Warten)
- Problem: **Priority Inversion** – niedriger Prozess blockiert Ressource, die höherer Prozess benötigt und darum wartet

Round Robin (RR): FCFS mit time-slicing im Intervall T
Regelmässige Neuzuteilung , Prozess wird hinten angehängt
- T gross → non-preemptive FCFS (da nie unterbrochen)
- T klein → simuliert Prozessor mit Leistung $1/n$

Multi-Level Scheduling: zB. für Vorder/Hintergrundprozesse
ML Queue Scheduling: Prozesse werden statisch in Klassen eingeteilt, welche separat behandelt werden (Priorität)
ML *Feedback* Q S: dynamische Einteilung entsp. Verhalten

Real-Time Scheduling: für Echtzeit-Anwendungen
Hard Real-Time: Deadline-Scheduling (zB. *earliest DL first*)
Soft Real-Time: Prioritätsbasiertes Scheduling mit höchster Priorität für Real-Time-Anwendungen

2.4 Synchronisation

Gemeinsames Betriebsmittel: Betriebsmittel, auf das mehrere Prozesse gleichzeitig zugreifen können
Kritischer Bereich: Abschnitt eines Prozesses, in welchem dieser auf ein gemeinsames Betriebsmittel zugreift
Verklemmung (Deadlock): Prozesse behindern sich gegenseitig beim Eintreten in einen kritischen Bereich
Race Condition: Ergebnis ist nicht deterministisch, hängt v. zeitlichem Verlauf / Scheduling von Einzeloperationen ab

Anforderung an die Behandlung kritischer Bereiche
- **Mutual exclusion:** zwei Prozesse dürfen nicht gleichzeitig in kritischen Bereichen bzw. desselben Betriebsmittel sein
- **Fairness condition:** Jeder Prozess, der einen kritischen Bereich betreten will, muss dies auch irgendwann mal können
- Prozess darf ausserhalb eines kritischen Abschnitts einen andern Prozess *nicht* blockieren
- Es dürfen **keine Annahmen** über die Abarbeitungsgeschwindigkeit oder das Scheduling gemacht werden

Spinlocks (Busy Waiting): Prozess wartet nicht bis Aufruf, sondern testet agressiv die Lock-Variable, um in kritischen Bereich zu gelangen → 100% CPU-Last
Problem: Priority Inversion bei Lock-Test von High-Prio.

Semaphoren
- ganzzahlige, nichtnegative Variable (minimal 0)
- down*():* wait, lock, dec ; *up():* signal, unlock, inc
- **down()** : testet, ob Semaphore positiv ist; wenn ja, wird dekrementiert; wenn nein, wird Prozess schlafen gelegt
- **up()** : inkrementiert Semaphore; falls Prozesse an diesem Semaphor schlafen, wird der erste geweckt (FCFS)

- **Binary Semaphore** (0 / 1): für gegenseitigen Ausschluss
- **Counting Semaphore** (Value 0 bis n): Elemente zählen

Weitere Synchronisationsmöglichkeiten
- **Semaphore:** haben keine Besitzer (von allen freigebbar)
- **Locks & Conditions:** haben einen Besitzer (kann freigeben)
- **Monitor:** „Klasse“, Daten nur über Betreten zugreifbar
- **Barrier:** blockiert Prozesse, bis gewisse Anzahl wartet

2.5 Klassische Probleme der Synchronisation

Ringpuffer (Producer / Consumer): 2 Semaphoren zur Verwaltung voller und leerer Slots (n_{full} / n_{empty})

Reader & Writers
- es darf immer nur ein Prozess Daten ändern
- es dürfen beliebig viele Prozesse Daten lesen
- Während Änderung darf niemand auf Daten zugreifen

Dining Philosophers
5 Philosphen & 5 Gabeln, brauchen 2 Gabeln, um zu essen
- Lösung 1: Warte zufällige Zeitspanne, nicht deterministisch
- Lösung 2: neuer Zustand *„hungrig“*, versucht Gabeln akquirieren; Zustandsübergänge d. Semaphore geschützt

Barbershop: Sequentieller Server, verarbeitet Kunden
Barbier = Server-Prozess, Stühle = Auftragsschlange (FIFO)

Probleme der Synchronisation

Verklemmung (deadlock): Prozesse sperren sich gegenseitig; alle warten für immer!

Verhungern (starvation): ein Prozess kommt nie dran

Bedingungen & Vermeidung von Deadlocks
Deadlock kann auftreten, falls alle 4 Regeln erfüllt sind:
- **Mutual exclusion:** jede Ressource ist nur einfach belegbar
- **Hold & Wait Condition:** Prozesse, die bereits Ressourcen besitzen, verlangen weitere (akquirieren nicht atomar)
- **No Preemption:** OS kann belegte Ressource nicht wegnehmen, nur Prozess kann sie freiw. zurückgeben
- **Zyklische Wartebedingung:** zyklische Kette von Prozessen, die auf Ressource warten, die Nachfolger bereits belegt

Vermeidung: Ressourcen werden systemweit identifiziert
Prozess muss Ress. immer num. aufsteigend akquirieren

Lösen: Zuerst Deadlock entdecken und danach auflösen:
- *Inform the operator and let him deal with the problem*
- Beendigung des Prozess (gegen *Circular Wait*)
- Entzug der Ressource (gegen *No Preemption*)

2.6 Interprozesskommunikation

- **Message passing:** Austausch von Nachrichten
  Synchrone Komm. : direkt zwischen Prozessen
  Asynchrone Komm. : über Briefkasten/Mailbox
  (kann über Kanal mit Speicherkapazität geschehen)
- **Signale** des Betriebssystem (*Signalling*)
- **shared memory:** Verwendung v. gemeinsamem Speicher
- **Rendez-vous:** Konzept von ADA für eingebettete Systeme

2.7 Threads

- Prozesse teilen sich die Ressourcen des Computers
- sind unabhängig voneinander, Koordination durch OS

Thread: Vereinfachte Form eines Prozesses, benutzt die der Anwendung zugeordneten Ressourcen

Prozess: eine laufende Anwendung, erhält Ressourcen v. OS enthält einen oder mehrere Threads, verwaltet diese selbstständig (Scheduling, teilt Ressourcen zu)

Pro Prozess vorhanden	**Pro Thread vorhanden**
Adressraum	Programmzähler
Globale Variablen	Register
Geöffnete Dateien	Stapel /Stack
Kinderprozesse	Zustand
Singale und Signalhandler	
Buchhaltungsinformationen (*accounting information*)	

User Space Thread Scheduling: Kernel wählt Prozess hat keine Kontrolle über Threads (Prozess hat)

Kernel Space Thread Scheduling: Kernel wählt Thread Kernel hat auch „Thread table” und somit gesamte Macht

CPU Threads: Moderne CPUs hat Threads auf HW-Ebene
- *Block multithreading*: ereignisgesteuert (non-preemptive)
- *Interleaved multithreading*: wie Round Robin
- *Simultaneous multithreading*: echte Parallelität

3. Speicherverwaltung

Mechanismus	Funktionsweise
(linear)	Statisches Speicherbild
Swapping	Ein- und Auslagern von ganzen Prozessen in/aus dem Hauptspeicher.
Relocation (Relokation)	Dynamisches Binden von Speicheradressen mittels eines Offsets.
Paging (Seitenverwaltung)	Dynamisches Ein- und Auslagern von Speicherblöcken konstanter Grösse (Seiten/Pages).
Virtual Memory (Virtueller Speicher)	Dynamisches Binden von Speicheradressen zur Laufzeit.
Segmentation	Dynamisches Ein- und Auslagern von logisch zusammengehörigen Speicherbereichen.

Aufgaben OS: (De-)Allokation von Speicher; Verwaltung der Belegung; Entscheidung, was wann im Speicher ist

Tradeoff: Zugriffsgeschw. , Grösse / Menge, Kosten, Erhaltungszeit (flüchtig/permanent), Transportierbarkeit

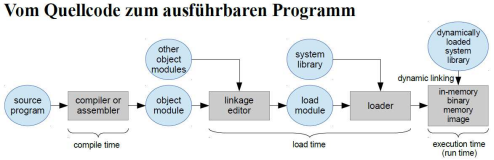
Caching / Puffering: Entkopplung der Stufen

Daten (Variabeln) eines Prozesses
- **statisch alloziert:** globale Variablen
- **automatisch alloziert:** auf dem Stack für lokale Variablen
- **dynamisch/explizit alloziert:** Heap, je nach Bedarf

Adressbinding: kann geschehen zur
- *Compile-Zeit*: absolute Adressierung, keine Relokation
- *Lade-Zeit:* Compiler generiert relozierbaren Code
- *Laufzeit / während Ausführung*: Hardware benötigt

Dynamisches Laden: Code wird bei Aufruf reingeladen
Dynamic Linking: Bibliothek zur Laufzeit eingebunden
Shared Libraries: von mehreren Prozessen zugreifbar

Vom Quellcode zum ausführbaren Programm



3.2 Swapping / Relokation / Adressarten

Monoprogrammierung: 1 Prozess im Speicher
Multiprogrammierung: mehrere Prozesse (Relokation)

Statische Speicher-V: Prozesse werden einmal geladen
Dynamische SV: Prozesse können mehrmals geladen & entfernt werden (während Laufzeit → Swapping)

Verwaltungsgrösse
- **Monolithisch:** Prozess als ganze Einheit verwaltet
- **Segmentation:** in funktionale Segmente aufteilen
- **Paging:** in Seiten mit konstanter Länge aufteilen

Swapping: auslagern = *swap out*, einlagern = *swap in*
Entscheid über Änderung durch Langzeitscheduler oder Speicherverwaltung (Krit: Zustand, Priorität, Auslastung)
Problem: Prozess kann nach Laden an anderem Ort sein → alle Daten (Var., Sprungaddr.) relativ z. Frame Pointer

Relokation: mithilfe Memory Managment Unit (MMU)
virtuelle Adresse → Relokation → physikalische Adresse
Statisch zur Compile oder Ladezeit durch virtuellen Sp.
Dynamisch zur Laufzeit durch Mapping auf Speicher
(falls über eigenen Adressraum hinaus → Fehler)

Virtueller Adressraum: für jeden Prozess verschieden
Relokationsregister enthält die Verschiebungsdaten
(*Basisreg.* : Adresse 0 des virtuellen Speichers
Grenzreg.: enthält Grösse des Bereichs)

MMU: Teil d. Prozessors, Speicher sieht absolute Addr.
Tracing mode: Verifikation jedes Speicherzugriffs mit ID

3.3 Speicherallokation & Fragmentierung

Buchführung über die Belegung des Speichers
- **Belegungstabelle:** Tabelleneinheit (1 Bit) gibt Zustand einer Speichereinheit (zB. 32Bit-Wort) an
- **Verkettete Liste:** traversiere alle Speicherblöcke, speichert zu jedem Block auch dessen Grösse
- **Hash-Tabelle**
- *eigene Liste für freien Speicher*: traversiere diese Liste und suche/akquiriere nach folgenden Prinzipien

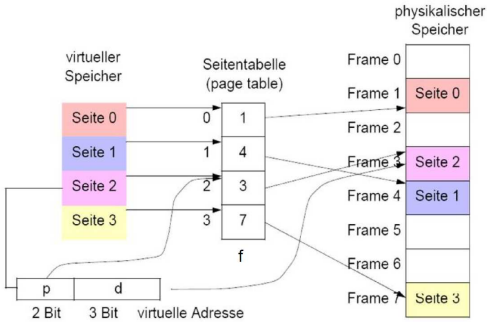
Algorithmen für die Speicherzuteilung
- **First-fit:** erster passender Bereich wird genommen
  → Schnell; Achtung Reststücke, immer schwieriger
- **Next-fit:** First-fit + Allokationszeiger wird mitgeführt
- **Best-fit:** am besten passende Bereich wird gewählt
  → muss ganze Liste trav., viele kleine freie Blöcke
- **Worst-fit:** der grösste Bereich wird verwendet
  → muss ganze Liste trav., grosse Blöcke schnell weg
- **Quick-fit:** eine Liste pro Anforderungsgrösse/Typ
  → gut, wenn Kenntnisse über typische Speichergrösse
- **Buddy:** Speicherbereiche ausschliesslich der Längen 2^k
  Speicher wird rekursiv in je zwei Buddies unterteilt
  Aufwand: $O(\log N)$, N: Anzahl Speicherblöcke

Fragmentierung
- **extern:** nicht-zugeteilte Speicherbereiche nicht genutzt, da nicht genügend *zusammenhängender Speicher*
- **intern:** nicht genutzter Speicher *innerhalb* des Bereichs
- **„Garbage collection“:** Periodisches Verdichten d. Speichers, gegen externe Fragmentierung; teuer od. unmöglich, falls Speicher absolut adressiert (statisch)

3.4 Paging

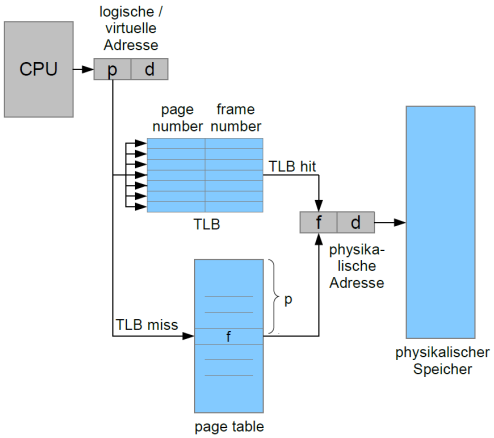
Base/limit-Technik (vorher): anfällig für Fragmentierung

Paging (Kachel-Verwaltung): löst *externe* Fragmentierung
- Aufteilung d. physikalischen Speichers in Seitenrahmen (page frames); typisch 2er-Potenz zw. 512 und 8192
- virtueller Speicher in Seiten d. gleichen Länge aufteilen
- OS führt Buch über Zuteilung aller Seitenrahmen
- virtuelle Adresse (p,d) = (Seitennummer, Displacement)
- physikal. Adresse (f,d) = (Framenummer, Displacement)
- Adressübersetzung mittels Seitentabelle & Hardware



Page Table Base Register (PTBR): in Prozesskontext
- Zeigt auf die Seitentabelle des Prozesses; teuer → TLB
- Pro Zugriff zwei Speicherzugriffe + Adressberechnung

Translation Lookaside Buffer (TLB)
- Cache für Seitentabellen-Einträge
- Verkürzt d. Laden der Seitentabellen für jeden Prozess



p: Seitentabellen-Index d: Seitenoffset (byte-genau)

Funktionen des Pagings
- löst externe, aber nicht interne Fragmentierung
- Form d. Adressbindung zur Laufzeit mit virtuellen Addr
- Jedes Objekt mit einem eigenen virtuellen Adressraum (Prozess, Datenstruktur) hat eigene Seitentabelle
- Speicherschutz durch Flagbit pro Seite (r / w / e)
- kann Code für mehrere Prozesse verwenden (falls Code reentrant und evtl. positionsunabhängig)

Hierarchische Seitentabelle: mehrstufige Seitentabellen
- Seitentabelle wird in Teile d. Länge der Seite aufgeteilt
- *Outer page table* hat Eintrag für jede Seite der *Inner page table,* welche einen Eintrag pro Seite hat

Hashtabelle: Seitennummer d. virtuellen Adresse als Schlüssel für die Speicherung von Einträgen
$Hash = Seitennr. mod\ Grösse_{Hashtabelle}$
- Kollisionen: verkettete Listen pro Hash durchsuchen

Invertierte Seitentabelle
- Tabelle mit Eintrag pro Seitenrahmen (frame)
- Eintrag enthält Seriennummer und Prozess-ID

Struktur eines Seiteneintrages

Name	Bits	Bedeutung
Caching disabled	1	ext. Speicher verwenden, z.B. Register eines Gerätes (protection bit)
Referenced	1	1 = Seite wird verwendet (referenced bit)
Modified	1	1 = Seite wurde modifiziert (dirty bit)
Protection	3	Lesen, Schreiben, Ausführen (rwx bits)
Present/absent	1	1 = ist im Speicher geladen (valid bit)
Page frame no.	20	

3.5 Demand Paging

Virtueller Speicher (Virtual Memory, VM)
- kann Prozess ausführen, der nicht vollständig in Speicher (oder sogar mehr Speicherplatz braucht, als physikalischer Speicher vorhanden ist!)
- Abschnitt d. Speicherbild wird erst bei Bedarf geladen
- Falls nicht genügend Platz vorhanden, muss zuerst ein Ausschnitt auf die Harddisk ausgelagert werden
- **Locality of reference:** Speicherzugriff lokal beschränkt während Zeitintervall nur wenige Pages betroffen einige Codeabschnitte werden gar nie benötigt!

Potentielle Vorteile von VM
- *Grösse von Programmen* nicht mehr durch verfügbaren Speicherplatz (stark) eingeschränkt
- *Mehr Parallelität*: es können mehr Prozesse gleichzeitig laufen, da pro Prozess weniger Platz weg
- *Weniger I/O-Operationen*, da kleineres Auslagern

Ablauf
1. Zugriff auf Speicher, Seitentabelle konsultieren
2. Falls nicht in Speicher: Trap, Unterbruch d Instruktion
3. OS lokalisiert Seite, lagert Seite ein und aktualisiert ST

Performanz von Demand Paging

Page Fault Rate: $0 \le p \le 1$, $p = 0$: *nie*, $p = 1$: *immer*

Effective Acces Time (EAT): mit TBL noch weitere Stufe

$$EAT = (1-p) * t_{mem,acc} + p * (t_{fault} + t_{swap,out} + t_{swap,in} + t_{restart})$$

Hardware: TLB, Übersetzung virtuell/physikalische Addr. Unterbruch & Fortsetzung d. Instruktion
Software: OS muss Datenstrukturen verwalten, Swaps

Copy-on-Write: ermöglicht gemeinsame Benützung derselben Seite; nur bei Änderung wird dupliziert

Memory-Mapped Files: ermöglicht Zugriff auf Dokumente analog zu Speicher durch Abbildung auf Seite
Mehrere Prozesse können gleiches Dokument abbilden

Seitenersetzungsstrategien: nur **„dirty“** Seiten ersetzen
- **FIFO:** älteste Seite wird ersetzt, oft ineffizient
- **Least recently used (LRU):** am längsten unbenutzt
- **Least frequently used(LFU):** am wenigsten benützt
- **Second chance:** setzt Referenzbit auf 0, *last chance*

Prepaging: Vorausladen oft benützter Pages beim Start
Globale/Lokale Ersetzung: Prozess wählt v. allen/v. sich
Beladys's Anomalie: trotz mehr Platz mehr Faults (per Zufall)

3.6 Thrashing

- Bei Prozessen mit viel I/O-Operationen kann die CPU-Auslastung durch grösserer Parallelität erhöht werden
- Ab gewissen Grad nimmt die Auslastung jedoch ab

Thrashing: zu häufiges Ein- & Auslagern von Pages

Todes-Spirale

→ Prozess hat nicht „genügend" gültige Seiten
→ Tiefe CPU-Auslastung, OS will Parallelität erhöhen
→ weiterer Prozess führt zu mehr Seitenswaps usw.

Lokalität: benötigte Grösse, um arbeiten zu können
- auch **Working Set**: soll in der TLB Platz haben

$\sum Grösser\ d.\ Lokalität > Verfügbarer\ Speicher \rightarrow \mathbf{Thrashing}$

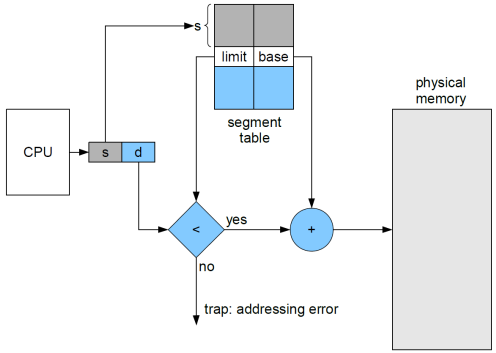
Lösung: bei Thrashing wieder einen Prozess ausgelagert

Festlegung für die Rate von Seitenfehler
- falls zu tief, werden dem Prozess Seiten entzogen
- falls zu hoch, bekommt er mehr zur Verfügung

3.7 Segmentierter Speicher

Verschiedene Segmente eines Programms:
- Hauptprogramm, Module, Unterprogramm
- Statische Variablen, Stack, Heap
- gemeinsam mit anderen Progr. verwendeter Bereich

Segmenttabelle
- **Segment-table base register (STBR)**
- **Segment-table length register (STLR)**



- Ermöglicht dynamische Relokation (zur Laufzeit)
- Gemeinsame Segmente mehrerer Prozesse möglich
- Speicherschutzinformation mit jedem Eintrag (r / w)

4. Input / Output

- I/O-Geräte und CPU können gleichzeitig arbeiten
- **Device controller** steuert einen bestimmten Gerätetyp
  Zugriff auf I/O-Gerät über diese; haben lokalen Puffer
- I/O wir durch BS-Steuerbefehl angestossen
- Controller teils HW (IC, Adapter), teils SW (Treiber)

Interrupts / Programmunterbrechungen
- wird bei Abschluss einer I/O-Operation von Controller gesendet, sodass OS wieder weiterfahren kann
- benützt, um dem OS die Kontrolle zurückzugeben
- wird durch Interrupt Handler (ISR) bearbeitet
- Trap: durch Software generierter Interrupt
- auch für Fehlerbehandlung verwendet (DivisionByZero)

Polling: aktives Warten (*busy waiting*) $\rightarrow CPU$

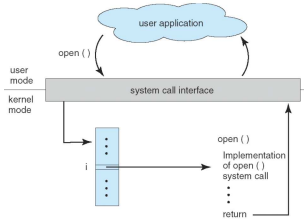
Programmed I/O: Kopieren von Daten durch CPU
- benötigt viel Rechenzeit, CPU ist verschwendet

Direct Memory Access (**DMA**)
- Spezialisierte Hardware für das Kopieren von Daten
- kopiert direkt zwischen I/O-Gerät und Speicher
- entlastet CPU, benötigt keine Rechenzeit
- behandelt rein physische Adresse, auch mit Paging

1. Gerätetreiber wird gesagt, er muss Daten zum Buffer mit Adresse X liefern
2. Gerätetreiber gibt Auftrag weiter an Disk Controller
3. Disk Controller initialisiert DMA-Transfer
4. Disk Controller sendet jedes Byte zu DMA Controller
5. DMA Controller schreibt Bytes in Buffer X, erhöht Speicheradresse und verringert C bis $C = 0$
6. Bei $C = 0$ erzeugt DMA Controller Interrupt, um der CPU den Abschluss der Transaktion zu signalisieren

4.2 I/O-Unterstützung des Betriebssystems

- OS versteckt Eigenheiten bestimmter I/O-Geräte
- Speicherverwaltung (Zwischenspeicher, Caching)
- Bereitstellung einer generischen Schnittstelle für I/O-Geräte (I/O-Treiber, Treiberschnittstelle), API:



Methoden zur Parameterübergabe bei Systemaufrufen

Register: einfach & effizient, aber beschränkte Anzahl

Stack: Parameter werden wie bei Funktionsaufruf von Aufrufer auf Stack gelegt und v. Aufgerufenem gelesen

Heap: Allokation v. dediziertem Speicher, Übergabe von Zeiger auf den Speicherblock mittels Register

Charakteristiken von Systemaufrufen
- **Blockierend:** Systemaufruf blockiert bis Ergebnis da
- **Nicht Blockierend:** sofort mit Statusmeldung zurück
- **Synchron** (Handschlag): Sender wartet auf Bestätigung
- **Asynchron** (Briefkasten): Senden und Empfangen unabhängig voneinander mittels Puffer

	Benutzerprozess ↔ I/O Gerät	
	Synchron	**Asynchron**
Blockierend	Der Prozess übergibt die zu senden Daten an den Kern. Der Prozess blockiert, bis die Übertragung der Daten zum I/O Gerät erfolgreich abgeschlossen wurde.	Der Prozess übergibt die zu senden Daten an den Kern. Der Prozess blockiert, bis die Daten in den Kern kopiert wurden. Anschliessend kommt der Systemaufruf zurück.
Nicht Blockierend	—	Der Prozess informiert den Kern, wo die zu senden Daten sind. Der Kern holt diese Daten selber ab. Der Prozess blockiert nicht.

I/O-Multiplexing: warte auf neue Daten bei mehreren Filedeskriptoren (mittels *select*())
Signal-driven I/O: OS gibt Prozess Signal b. neuen Daten
Asynchronous I/O: schreibt Daten an gegebene Speicherstelle und gibt Signal an OS

Schichten im Kern
user mode: Benutzerprozess
kernel mode: *kernel*-Verteiler → Auftragsverwaltung → Pufferung → Treiber → Controller → Gerät

Gerätetreiber: plattformunabhängige Ablaufumgebung
- Standartschnittstellen für Treiber, zB. UDI (Unix)

Stream-System: Einschalten von Filterschichten zw. Layers

Geräteschnittstellen
- Register der I/O-Geräte (Status-, Controll, Data-in/out-)
- Steuerung über **Memory Mapped I/O** (schreibe in Register)
- Behandlung der Geräteregister (Funkt., Paras) wie Speicher

Interrupt-driven I/O: CPU startet I/O, I/O-Controller sendet nach Beendigung/Fehler Interrupt, CPU verarbeitet diesen

Caching: schneller Speicher hält Kopie der Daten
Spooling: Zwischenspeicher bei Ausgabedaten für Geräte

4.3 Gerätemodelle & Schnittstellen

Zugriffsarten
- *Wahlfreier Zugriff* (random access): RAM, Harddisk, CD
- *Sequentieller Zugriff*: Terminals, Magnetbänder

Blockorientiertes Gerät: zB. HDD, DVD
- nützt örtliche Lokalität; typ. Befehle: *read, write, seek*
- direkter Zugriff od. Dateisystem; memory maped I/O möglich

Zeichenorientiertes Gerät: zB. Tastatur, Maus, USB
- typ. Befehle: *get, put* ; Bibliotheken f. Zeichenorientierung

Netzwerkgerät: typ. Befehle: *select, poll*; arbeitet m. sockets

Virtuelles Gerät: zB. Zufallszahlgenerator, Null-Device

Eigenschaften von I/O-Geräten
- *data-transfer mode*: character (terminal) / block (disk)
- *access method*: sequential (modem) / random (CD)
- *transfer schedule*: synchron (tape) / asynchron (keyboard)
- *sharing*: dedicated (tape) / sharable (keyboard)
- *device speed:* latency, seek time, transfer rate
- *I/O direction*: read only, write only, read-write

Plug-and-Play (PnP): Erfassen aller Geräte, Auslegung der Listen für Adressen & Interrupts unter Berücksichtigung v. *legacy devices*, Einstellen der Geräte ensprechend Liste

BIOS (Basic Input-Output System): Erstellt Liste für OS
PCI-Bus (Peripheral Component Interface): $64\ bit, 66\ MHz$
- Bus passt sich langsamstem Berät an, taktet Rest runter

4.4 Festplatten

Zylindergruppe: fasst Spuren mehrerer Platten zusammen
- kann parallel Spuren desselben Blocks lesen

Will: schnellen Zugriff, hohe Kapazität, einfache Verwaltung

Zoning: Einteilung der Platte in ca. 20-40 Zonen
- dient zur effizienten Ausnützung der Speicherdichte
- jeweils konstante Sektorenanzahl pro Zone (aussen mehr)
- jede Zone hat eigene Transfergeschw. (Sektoren/Zeiteinh.)

Zugriffsoptimierung: Hersteller optimiert intern Controller (Anpassung an Modell), nach aussen einheitliches virt. Gerät

Massenspeicher: Redundanz, Zugriffsgeschw., Grösse (Kap./Vol.)
- **NAS:** Network-Attached Storage (Speicher über Netzwerk)
- **SAN:** Storage Area Network (Verbund zu grossem Speicher)
- **RAID:** Redundant Arrays of Inexpensive Disks

4.5 Festplatten Scheduling

- Ziel: Optimierung der Suchzeit und des Durchsatzes

Zugriffszeit: *Suchzeit + Rotationsverzögerung*
- **Suchzeit:** Zeit, bis Lesekopf über gewünschter Spur
- **Rotationszeit:** bis Platte zu gewünschtem Sektor gedreht

Mittlere Suchzeit: t_s
Mittl. Rotationsver.: t_D , $max: t_R = 2 * t_D$
Transferzeit: t_T *für* k *Bytes bei Spurgr.* m *Bytes*
Datentransferrate: k/t_T

Gesamtzeit: $T = t_S + \frac{t_R}{2} + \frac{k}{m}\ t_R$

Zugriffsmethoden
- **FCFS** (*First Come, First Served*): fair, aber ineffizient
- **SSTF** (*Shortest Seek Time First*): wählt kürzeste Suchzeit aufgrund aktueller Position; Starvation (Verhungern) !
- **SCAN** (*Elevator Algorithm*): von einem Ende zum Andern; danach wieder zurück (hin- und her- schweifend)
- **C-SCAN**: von einem Ende zum Andern, geht dann direkt wieder an den Anfang (da dort am meisten Aufträge warten)
- **C-LOOK**: so weit nach aussen, bis keine Anfragen, dann geht er wieder zur innersten („intelligenter" C-SCAN)

Im Normalfall: SSTF (aber Starvation) oder C-LOOK

4.6 RAID (Redundant Arrays of Inexpensive Disks)

- Hot Spare: eingebaute leere Disk, bei Ausfall sofort Ersatz

RAID 0: *keine Redundanz*; Paralleles Lesen & Schreiben
RAID 1: *Mirroring*; Duplizierung, Datensicherung & Lesen
RAID 2: *Error Decting and Correcting Code*
RAID 3: *Bit-Interleaving Parity*; Paritätsbit auf Zusatzplatte
RAID 4: *Block-Interleaving Parity*;
RAID 5: *Distributed Block-Interleaving Parity*; Paritätsbit über alle Platten verteilt; bei Schreiben immer 2 Befehle!
RAID 6: *P+Q Redundancy*; Doppelte Ausfallsicherheit durch zwei Paritätsbit über mehrere Daten-Zeilen

RAID(n,m) / RAID n+m
- n: Gesamtzahl der Festplatten
- m: Grad des RAIDs (Anzahl Parities)

Lesegeschwindigkeit: n * Lesegeschw. d. Einzelplatte
Schreibgeschw.: (n-m) * Schreibgeschw. d. Einzelplatte
Kapazität: (n-m) * Kapazität d. Einzelplatte

RAID(n,0) = RAID 0; RAID(n,1) = RAID 5; RAID(n,2) = RAID 6

4.7 SSD (solid state disk)

- low power consumption, no failures from moving parts
- Schreiben: eine Seite pro Zeit; bei Löschen ganzer Block

5. Dateisysteme

5.1 Grundlagen & 5.2 Zugriffsstruktur

OS ermöglicht Abstraktion von physischen Eigenschaften zur virtuellen Einheit der Speicherung (Datei)

Dateisystem: Liste von Dateien auf Massenspeicher
zB. Nummer,Name, Datum, Länge → relationale DB

Hierarchischer Namensraum: Verzeichnisbaum
- Knoten = Ordner, gleiche Dateinamen möglich
- zusätzliche Vernetzung (Links/Verweise) → azyklischer Graph

Verweise auf Dateien/Verzeichnisse
- Relativer Pfadname: gute Portabilität ↔ absoluter Pfad

Soft Link: Verweist auf den Pfad der Datei/Verzeichnisses
- textueller Verweise, partitionsübergreifend

Hard Link: Direkter Verweis auf Datei / eindeutige Referenz
- nur innerhalb gleicher Partition, nicht auf Verzeichnisse

File Control Block (FCB): beinhaltet Dateiattribute & Security
- Name, Typ, Dateilänge, Datum, Ort, Tag, Erzeuger, Rechte

Zugriffsrechte: owner / group / others
Datei: Lesen *r*, Schreiben/Löschen *w*, Ausführen *x*
Verzeichnis: Liste lesen *r*, Datei anlegen *w*, Durchsuchen *x*

File Descriptor (fd): Verweis auf offene Datei, pos. Integer
- fd=0 : stdin , fd=1 : stdout , fd=2: stderr (Standart Error)

Zugriffsstrukturen
- sequentielle Dateien: Magnetbänder, Lochstreifen
- wahlfreie Daten: Festplatten, CD; allgemein einsetzbar

Indexsequentielle Dateien: nach Index/Schlüssel geordnet

5.3 Implementierung d. DS : Dateien

- Aufteilung d. Speicherplatzes in Blöcke (frei, belegt)
- Struktur von Dateien: Zusammengehörigkeit von Blöcken

Datei als verkette Liste von Speicherblöcken
- Vorteil: Rekonstruktion möglich bei Anker-Löschung
- Nachteil: Ineffizienter wahlfreier Zugriff, nur sequentiell

Datei über zentrale indexbezogene Speicherzuweisung
- Zentraler Block von Zeigern (Indizes) in einer Liste (FAT12)

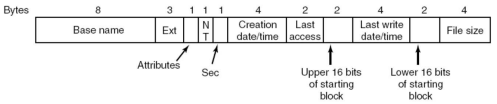
Verteilter Index: indexbezogene Speicherzuweisung
- eigene Indexliste pro Datei; muss sich Grösse anpassen

Ein- und mehrstufige Übersetzung
Baumstruktur mit 1-, 2-, 3- stufiger Tabellen zur schnellen Lokalisierung von Blöcken (schneller geladen, da nur Teil)

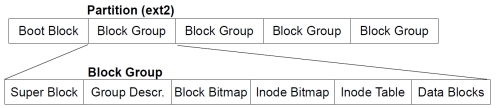
5.4 Implementierung d. Dateisystems: Verzeichnisse & Partitionen

Aufbau einer Diskette mit FAT-Format
- Boot-Block, File Allocation Table (FAT), Backup der FAT, Wurzelverzeichnis, Cluster à 1024 Bytes

FAT x : x bezeichnet die Länge der Einträge in d. FAT



Aufbau des EXT2 UNIX-Dateisystems



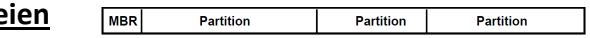
Super Block: Anzahl Inodes & Datenblöcke, Adresse des ersten (root) Inode, Anzahl freier Inodes, Blockgrösse, Blöcke / Inodes pro Gruppe, Anzahl Bytes per Inode, **Magic Signature** 0xEF53 (Beginn ALLER Superblöcke)

Block Bitmap: verwaltet die freien Datenblöcke
Inode Bitmap: verwaltet die freien Inodes
Inode Table: Speicherort der Inodes

Inode (Index-Knoten): mode (Datei oder Verzeichnis), link count, Besitzer, Dateigrösse, Zeitstempel (last access), Adressen d. ersten 10 Blöcke, 1/2/3-indirekte FAT
Link count: Anzahl Hardlinks; wenn = 0 wird gelöscht

Verzeichniseintrag im klassischen UNIX-Filesystem
- Verzeichnisse sind auch Dateien (spezielle Art)
- Tabelle mit 16 Bytes pro Eintrag (Inode + File name)

Struktur eine Festplatte



Master Boot Record (MBR): Boot-Loader, Disk-Signatur, Partitionstabelle, MBR-Signatur (0xAA55)

Grosse Blöcke → Grössere Dateien möglich, weniger Platzbedarf, bessere Performanz, schlecht bei kleinen Dateien

5.5 Virtual File System (VFS)

- ermöglicht Verwendung derselben Systemaufrufe für unterschiedliche Dateisysteme (Abstraktion , Art API)

VFS Superblock: repräsentiert ein physikalisches Dateisyst.
- Wird beim Mounten d. DS für dieses erstellt

VFS Inode: „simuliert" eine Inode des physischen Dateisyst.

VFS Superblocks u. Inodes werden im RAM alloziert (Cache)

6. Computernetzwerke

6.1 Einführung

Rechner-/Computernetzwerk: Menge autonomer Rechner, die miteinander Informationen austauschen können

Verteiltes System: Menge geographisch verteilter, autonomer und miteinander verbundener Computer, die durch Kooperation einen Dienst erbringen (als 1 Einheit)
Transparenz d. Verteiltheit: Nutzer bemerkt Aufteilung nicht

Verteilte Anwendung: Anwendung, die auf mehreren kooperierenden, geografisch verteilten Rechnern läuft und dem Benutzer einen spezifizierbaren Dienst liefert.

6.2 Verteilte Anwendungen / Systeme

Client-Server Computing
- Server(prozess) ist Anbieter von Dienstleistung, Client User.
- einfache zentrale Verwaltung, da Intelligenz im Server

Sun Netzwork File System (**NFS**)
- Protokoll für d. Zugriff auf entfernte Dateien mittels UDP/IP
- **Remote Procedure Calls**: Suche v. Dateien im Verzeichnis, Lesen v. Verzeichniseinträgen, Veränderung v. Verweisen, Zugriff Auf Dateiattribute, Lesen & Schreiben von Dateien

Domain Name System (DNS): Übersetzt Namen in Adressen

Peer-to-Peer
- alle PCs gleichberechtigt; sowohl Server als auch Client
- zB. Filesharing, Instant Messaging, Storage

6.3 Computernetzwerke

Anforderung an Computernetzwerke
- *Netzwerk-Benutzer*: zuverlässig, schnell, soll grosse Datenmengen zu möglichst kleinem Preis transportieren
- Netzwerk-Designer: Ausbaubarkeit, Wartbarkeit, Ressourcen (Kapazität, Speicher) möglichst effizient genutzt
- *Netzwerk-Betreiber*: einfach betreibbar, konfigurierbar, beobachtbar und veränderbares System, kleine Kosten

Komponenten eines Netzwerkes
- *Übertragungskanäle* (Links): Kupfer-, Koax-, fiberopt. Kabel
- aktive Komponenten: Hubs, Router, Switches, Firewalls
- Endsysteme: Workstation, Benutzer-PCs, Servers

Verbindung: Punkt-zu-Punkt vs. Mehrfachzugriff/Bus

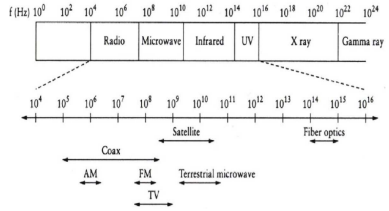
Netzwerk: zwei oder mehr Knoten/Netzwerke, durch einen oder mehrere Knoten miteinander verbunden

6.4 Netzwerke mit direkten Links

Punkt-zu-Punkt: Ein Link verbindet zwei aktive Komponenten (Rechner, Router, Switch)

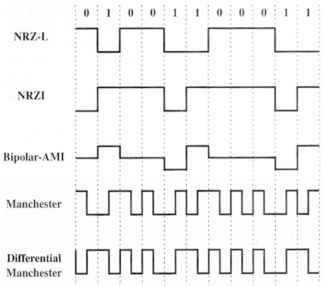
Link mit Mehrfachzugriff: Ein Link verbindet mehrere aktive Komponenten untereinander; bei Kollision zufäll. Wartezeit

Übertragungsmedien
- Paarsymmetrische Kupferkabel (twisted pair)
  - abgeschirmt (STP), ungeschirmt (UTP)
- Koaxialkabel
- Fiberoptische Kabel, Lichtwellenleiter
- Drahtlose Übertragung



Medium	Maximale Übertragungsrate	Distanz
Twisted Pair (Cat 6e/7)	10 Gbit/s	~50 m
Koaxialkabel (outdated)	100 Mbit/s	500 m
Multimode (MM) Fiber	100 Mbit/s 10 Gbit/s	2 km 300 m
Singlemode (SM) Fiber	100 Gbit/s (exp: 300 Gbit/s)	80 km
DWDM (Dense Wavelength Division Multiplexing)	160 * 40 Gbit/s	200 km
Drahtlos 2.4/5.2 GHz (Mehrfachzugriff/WLAN)	300 - 450 Mbit/s	20-70 m

Leistungskodierung für digitale Signale
- *Nonreturn to Zero-Ivel (NRZ-L)* : konstant falls kein Wechsel
- *Nonreturn to Zero-Inverted (NRZI):* Pegeländerung bei 1
- *Bipolar-AMI:* 1 durch Abwechselnde +/— Impulse
- *Manchester:* 0 = fallende Flanke, 1 = steigende Flanke
- *Differential Manchester*: 1 = keine Flanke am Anfang, 0 = Flanke am Anfang der Bitzelle



Ziele der Leitungscodierung
- Gleichspannungsfreies Signal
- Möglichst hohe Datenrate, möglich kleine Fehlerrate
- Fähigkeit der Regenerierung des Takts / Steuersymbole

6.5 Netzwerke mit gemeinsamen Links

Gemeinsame Nutzung: Multiplexierung
- ***Frequenz**multiplexierung* (frequency division multiplexing)
- ***Zeit**multiplexierung* (time division multiplexing)
  - synchroner / asynchroner (statischer) Zeitmultiplex
- ***Raum**multiplex* (space division multiplexing)
- ***Code**multiplex* (code division multiplexing/spread spectrum)

Gemeinsame Nutzung: Vermittlung (Switching)
- ***Leistungs**vermittlung* (circuit switching): Leitung wird vor der Kommunikation fest reserviert für gesamte Dauer; benötigte Übertragungskapazität fix im Voraus reserviert
- ***Paket**vermittlung* (packet switching): Datenströme werden in Pakete aufgeteilt, Übertragungskapazität nach Bedarf

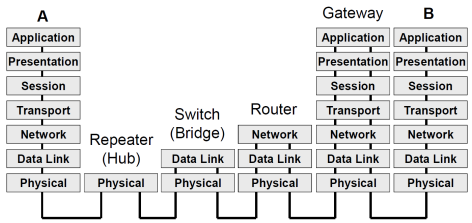
6.6 Protokoll & Referenzmodelle

Protokoll: Konvention oder Standard, welche die Verbindung, Kommunikation und den Datentransfer zwischen zwei Endpunkten kontrolliert und erlaubt.

Schicht N: Dienstanbieter
Schicht N+1: Dienstbenutzer

ISO/OSI Referenzmodell

Layer	Name	Data Container	TCP/IP	Name
7	Application	(Stream)		
6	Presentation		4	Application
5	Session			
4	Transport	Segment	3	Transport
3	Network	Datagram (Packet)	2	Internet
2	Data Link	Frame	1	Network Access (Interface)
1	Physical	Bit		



6.7 Sicherungsprotokolle

- Fehlerbehandlung (Verfälschung, Verluste, Duplikate)
- Flusssteuerung, Synchronisation, Rahmenbildung

Sicherungsfunktionen im OSI-Referenzmodell
- Hop-by-hop: Schicht 2 (Data Link)
- End-to-end: Schicht 4 (Transport)

Zeichenorientierte Rahmenbildung
- DLE & STX zu Beginn eines Rahmens (Rahmenmarke)
- DLE & ETX am Ende eines Rahmens
- **Charakter stuffing**: zusätzliches DLE zwischen DLE & ETX

Bitorientierte Rahmenbedingung
Bsp.: *High Level Data Link Control (HDLC)*
- 01111110 (Frame Delimiter) zu Beginn und Ende eines Rahmens
- nach 5 1-Bits jeweils 0 in Nutzdaten einfügen (**bit stuffing**)

Lösungsansätze
- **Vorwärtsfehlerkorrektur**: Detektion u. Korr. bei Empfänger
- **Echoverfahren**: Detektion u. Korrektur beim Sender
- **Automatic Repeat Request** (ARQ)

Automatic Repeat Request (ARQ)
- Detektion beim Empfänger, Korrektur beim Sender
- Nummerierung der Signale ACK/NAK für die Sortierung

ACK (Acknowledgment) : bei Rahmenempfang
NAK (Negative Acknowledgment): bei verfälschtem Empfang

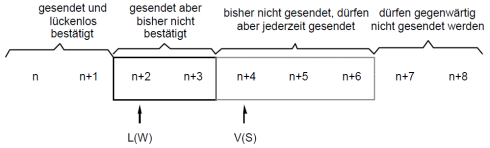
Idle Repeat Request (IRQ)
Bei Timeout: nach gewisser Zeit erneut senden

Continuous Repeat Request (CRQ)
- Mehrere Rahmen können gleichzeitig unterwegs sein
- Ziel: Bessere Auslastung des Kanals

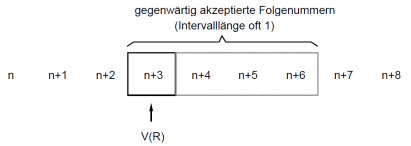
CRQ-SR (Selective Retransmission)
- Falls Lücken in Bestätigung: nochmals Übertragen
- Falls Bestätigungen ausbleibt: Timeout
- Probleme: grosser Eingangspuffer nötig
  Reihenfolge der Datenblöcke (ungeordnet)

CRQ – Go-Back-N
- ACK(N) bestätigt alle Blöcke bis und mit N
- Bei falscher Folgenummer wird ein NAK mit der Folgenummer des ausstehenden Blockes M gesendet. Dadurch werden implizit alle Blöcke bis M-1 bestätigt.
- Timeouts, falls ACK und/oder NAK ausbleiben

Sliding Window
- beim CRQ-Sender:



- beim CRQ-Empfänger (bei Go-Back_N Festergrösse 1):



Problem: Empfänger ist systematisch langsamer als Sender

7. Verschiedenes

Typische Dateifunktionen eines Dateisystems

- Eine Datei ist ein abstrakter Datentyp (ADT)
- Create File: Prüfen d. Zugriffsrechte, Name, Zugriffsart, Erstellen des Verzeichniseintrags, Inode vorbereiten
- Open File: Prüfen d. Zugriffsrechte, Puffereinrichtung Open File Table Eintrag erstellen
- Close File: Puffer + Tabellen deallozieren, OFT-Eintrag
- Delete File: Prüfen d. Zugriffsrechte, Verzeichniseintrag
- Read/Write/Append File: Puffer schreiben/lesen: DMA!
- Seek File: Index der aktuellen Position setzen
- Get/Set Attributes (u.a. Rename File)

Interrupts

1. Kontext speichern (Rücksprung-Adresse)
2. Maskieren, s.d. Interrupts mit kleinerer Prio verhindert
4. Interrupt-Vektor aufrufen & Interrupt-Handler ausführen
5. Kontext wiederherstellen und fortfahren

Anmerkungen zu Kapitel

3. Speicherverwaltung

Arbeitsspeicher: Ausführung von Programmen
- benötigt Speicherverwaltung (memory management)
- RAM : *Random Access Memory*, wahlfreier Zugriff

Dateisysteme : für (Langzeit-)Speicherung von Daten

Cache : hält Daten in schnellerem Speicher bereit

4. Input / Output

Geräteabhängige Logik, Pufferspeicher für Daten

Abschluss der E/A-Operation durch Interrupt signalisiert

Controller : Hardware (IC, Adapter) & Software (Treiber)

DMA : Direct Memory Access, übernimmt Transport

5. Dateisysteme

Funktionen des Dateisystems
- Zugang, Benutzeridentifikation & Authentisierung
- Erstellen, Lesen, Schreiben & Löschen von Daten
- Manipulation von Zugangsrechten
- Abbildung auf Sekundärspeicher
- Datensicherung