Mutual Exclusion und Synchronisation

Florin Hillebrand 0925917

Notes:
- Threads section is incomplete
- Original pages have been cleaned with gimp
- Index added with ghostscript

- disziplinierter Zugriff
- geordnete Abarbeitung

∧

- Fehlverhalten
- Inkonsistenz
- Andere Ergebnisse durch andere Abarbeitungsfolge

Mutual Exclusion

- Wechselseitiger Ausschluss
- Ziel: Konsistenz d. Daten

Condition Synchronisation

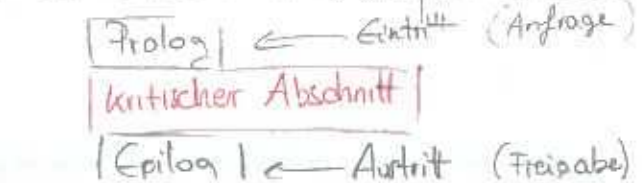
- Warten auf Eintreten von Bedingung
- Bedingungssynchronisation
- Ziel: Bestimmte Abarbeitungsfolge

Mut. Excl	Cond Synch	Ziel
–	–	unabhängige Aktionen
–	+	vorgegebene Abfolge
+	–	Konsistenz
+	+	Konsistenz u. Abfolge

Kritischer Abschnitt

- Ein Prozess befindet sich im kritischen Abschnitt, wenn er auf gemeinsame Ressourcen zugreift.

- Wechselseitiger Ausschluss: Nur ein Prozess in krit. Abschnitt erlaubt
- Eintritt geregelt



Anforderungen für Lösung f krit. Abschnitt

- Mutual Exclusion
- Progress: Wenn kein P. in krit. Abschnitt dann darf Entscheidung über Nächsten P. nicht ewig verzögert werden.
- Bounded Waiting: Nach Request muss Anzahl anderer Prozesse f krit. Abschnitt geregelt sein

Lösungen

- Software (Annahme Atomizität von Speicher OPs)
- Hardware
- Höhere Konstrukte: Semaphore, Monitor, Message Passing

Software-Lösungen

- Atomares Lesen u. Schreiben auf Speicher
- Synchronisation über globale Variable
- Busy Waiting
- Bsp:
  - Dekker Algorithmus 1, 2, 3, 4
  - Peterson Algorithmus
  - Bakery Algorithmus

Hardware-Lösungen

- Disable Interrupt: Uniprozessor, Dispatching?, Rückgabe?
- Maschinenbefehle: Test and Set, Exchange (Swap)

Test and Set

- Hardware Befehl der zwei Aktionen atomar ausführt

```
bool test_set (int i) {
    if (i == 0){
        i - 1,
        return true;
    }else
        return false;
}
```

} Atomar, kann nicht unterbrochen werden

Exchange

- Tauscht Variable atomar aus
- erfordert Busy Waiting
- Starvation möglich

Semaphore (Software u. Hardware Lösung)

- kein Busy Waiting
- wait, signal auf Integer Variable
- Blockieren in OS blocked Queue
- Mutual Exclusion durch init(S, 1); {Binary}
  P V
- wait, signal - atomar mittels Test-Set
- 'count' Prozesse dürfen krit. Abschnitt betreten
- Bedingungssynchronisation

Semaphore

```
wait (S);
| kritischer Abschnitt |
signal (S);
```

```
typedef struct sem {
    int count;
    queue waiting;
}
```

Beispiele für Semaphoren

Producer and Consumer

- Begrenzter / Unbegrenzter Puffer
- Mutual Exclusion: Nur ein Prozess darf auf Puffer zugreifen
- Bedingungssync.: Konsument darf nur lesen wenn neues Element vorhanden ist.

P-C: Implementierung

nitialisierung: init (S, 1); init(N, 0); in := out := 0;

```
append (v):
  b[in] := v;
  in := in + 1;

take ():
  w := b[out];
  out := out + 1;
  return w;
```

Producer:

```
loop
   produce (v);
   P (S);
   append (v);
   V (S);
   V (N);
end loop
```

Consumer:

```
loop
   P (N);
   P (S);
   w := take ();
   V (S);
   consume (w)
end loop
```

P-C Ringpuffer-Implementierung

Initialisierung: init (S, 1); init (N, 0); init (E, K);
in := out := 0;

```
append (v):
  b[in] := v;
  in := (in + 1)
        mod K;

take ():
  w := b[out];
  out := (out + 1)
         mod K;
  return w;
```

Producer:

```
loop
   produce (v);
   P (E);
   P (S);
   append (v);
   V (S);
   V (N);
end loop
```

Consumer:

```
loop
   P (N);
   P (S);
   w := take ();
   V (S);
   V (E);
   consume (w)
end loop
```

- Reihenfolge von V-Operationen ist beliebig
- Reihenfolge von P-Operationen ist wichtig

Reader - Writer

- Lese und Schreibprozesse die auf gemeinsame Ressource zugreifen
- Beliebig viele Leser dürfen parallel lesen
- Schreiber benötigt exklusiven Zugriff

Reader-Writer Problem

```
init (x, 1); init (y, 1); init (z, 1); init (wsem, 1); init (rsem, 1);
rc := 0; wc := 0;

Reader:
loop
   P (x);
   rc := rc + 1;
   if rc = 1 then P (wsem);
   V (x);
      read;
   P (x);
   rc := rc - 1;
   if rc = 0 then V (wsem);
   V (x);
end loop
```

```
Writer:
loop
   P (wsem);
      write;
   V (wsem);
end loop
```

Leser haben Priorität

Reader-Writer Problem

```
Reader:
loop
   P (z);
   P (rsem);
   P (x);
   rc := rc + 1;
   if rc = 1 then P (wsem);
   V (x);
   V (rsem);
   V (z);
      read;
   P (x);
   rc := rc - 1;
   if rc = 0 then V (wsem);
   V (x);
end loop
```

```
Writer:
loop
   P (y);
   wc := wc + 1;
   if wc = 1 then P (rsem);
   V (y);
   P (wsem);
      write;
   V (wsem);
   P (y);
   wc := wc - 1;
   if wc = 0 then V (rsem);
   V (y);
end loop
```

Schreiber haben Priorität

Dining Philosophers Problem

- Deadlock bei ersten Versuch mit normalen Semaphoren

⇒ MP and_M V für mehrere Semaphoren

(and_M V: atomare)

Spinlocks

- Mehrwertige Semaphore mit Busy Waiting
- Sicherung kurzer kritischer Abschnitte auf Multi-Cores
- CPU overhead aber kein Prozess switch

Eventcounts

- zählt Ereignisse, Anfangswert = 0
- advance(E), await(E, val) blockiert P, bis E ≥ val

Sequencer

- ticket(S), liefert inkrementierten Int, atomar

Mutex mit S u. E

```
sequencer S;
eventcount E;

await(E, ticket(S));
critical sec.
advance(E);
```

E und S für Producer Consumer

Producer:
loop
produce (v);
t := ticket (Pticket);
-- nur ein Producer
await (In, t);
-- schreibbares Element
await (Out, t-K+1);
b[t mod K] := v;
-- Element geschrieben
-- nächster Producer
advance (In);
end loop

Consumer:
loop
u := ticket (Cticket);
-- nur ein Consumer
await (Out, u);
-- lesbare Daten
await (In, u+1);
w := b[u mod K];
-- Daten gelesen
-- nächster Consumer
advance (Out);
consume (w)
end loop

- E mit S wenn neue Prozesse aufsynchronisieren müssen
- E ohne S ⇒ Bedingungssynch, vordefiniertes zyklisches Muster, Ablaufsteuerung durch State Machine

Monitor

- Monitor sorgt für Mutex
- Gemeinsamer Speicher im Monitor
- Condition Variable
- Für verteiltes (Netzwerk) Synchronisieren

Condition Variables

- cwait(c), csignal(c)
- Wenn mehrere cwait(c), wähle einen aus.
- wenn keine — ", nichts tun ⇒ nicht gespeichert

Message Passing

- IPC
- send (dest, msg), receive (source, msg)
- Msg. ist Atomare Datenstruktur

- Synchronisation
- Datenverwaltung
- Addressierung 1:1

- Ereignisnachricht
  - send : Queue (block./non-block.)
  - receive : Queue
- Zustandsnachricht
  - send: non-blocking
  - receive: Überschreiben alter Werte

Deadlock

- Permanentes blockieren einer Menge von Prozessen
- Zyklischer Ressourcenkonflikt
- keine universelle Lösung

4 Bedingungen

notwendig u. hinreichend

1. MUTUAL EXCLUSION
2. HOLD AND WAIT ; P. kann Res. halten und auf neue warten
3. NO PREEMPTION ; Res. werden P. nicht weggenommen
4. CIRCULAR WAIT

Gilt 1,2,3 => kann Circular Wait nicht auflösen => Deadlock

Behandlung

- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection

Deadlock Prevention

- Indirect Deadlock Prevention
  - Verhindern einer von 1 bis 3 ⇒
    1 ~~Mutex~~ immer gebraucht
    2 ~~Hold-n-Wait~~ Alle Res. auf einmal behandeln
    3 No Preemption

- Direct Deadlock Prevention
  - Verhindern von Circular Wait

$$O(R_0) < O(R_1) < O(R_0)$$

  - möglich ⇒ verhindert DL
  - kann paralleles Arbeiten ⇒ Performance ↓

Deadlock Avoidance

- 1, 2, 3 erlaubt
- Process Initial Denial

  Prozess wird nicht gestartet wenn nicht alle Res verfügbar sind
- Ressource Allocation Denial
  - Ressourcenbedarf von Prozess muss bekannt sein

⇒ Banker's Algorithm (Siehe Blatt)

Deadlock Detection

Ressourcenanforderung wird immer gewährt sofern vorhanden

- DL detection bei jeder Res Anf ⇒ Aufwendig
  - Markiere alle P_i mit $A_{ik} = 0$
  - Gibts Q_i mit $Q_{ik} \leq V_i$
    ⇒ $V_i += Q_{ik}$ und markiere
    else ⇒ Deadlock

Bankers Algorithm

n Prozesse (k)
m Ressourcen (i)

$$Ressource = (R_1, R_2, R_3, \dots, R_m)$$

$$Available = (V_1, V_2, V_3, \dots, V_m)$$

$$Claim = \begin{pmatrix} C_{11}, C_{12}, \dots, C_{1m} \\ \\ C_{n1}, C_{n2}, \dots, C_{nm} \end{pmatrix}$$

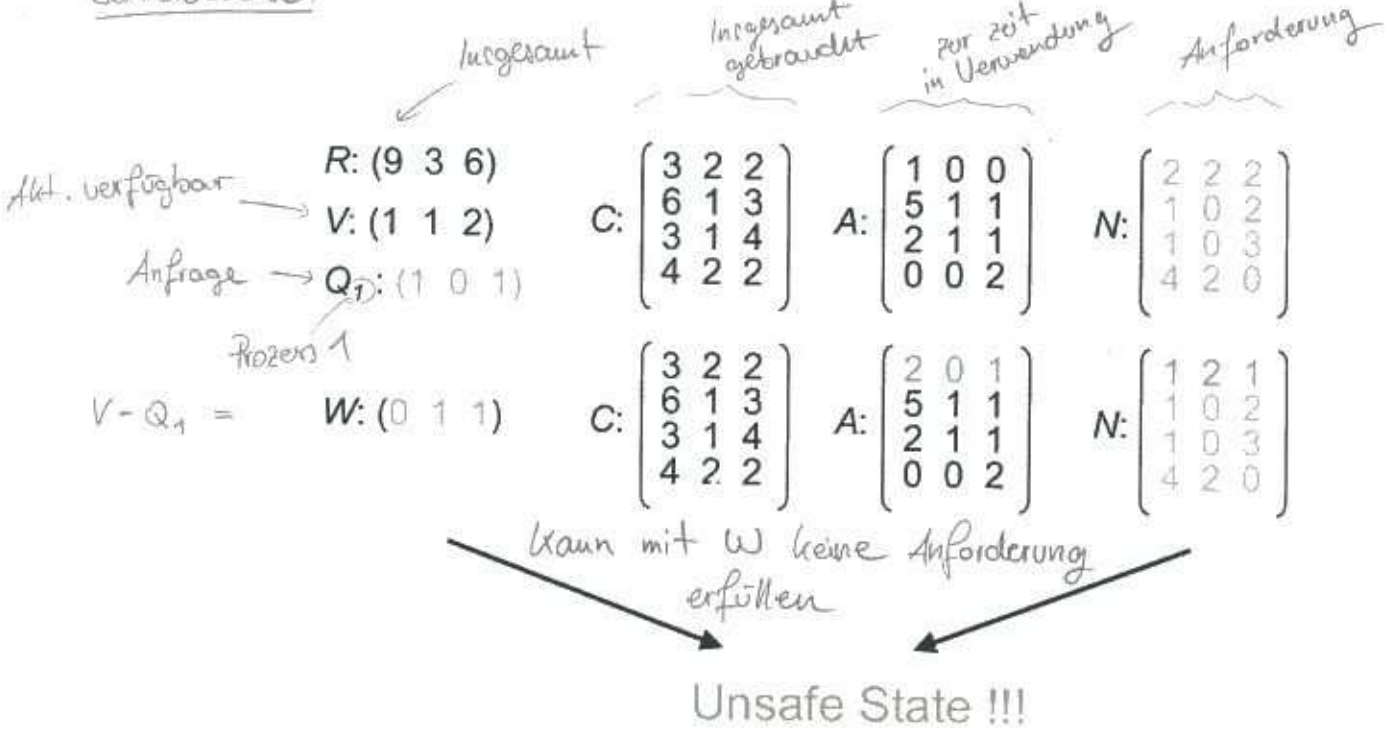
$$Allocation = \begin{pmatrix} A_{11}, A_{12}, \dots, A_{1m} \\ \\ A_{n1}, A_{n2}, \dots, A_{nm} \end{pmatrix}$$

$$Required = \begin{pmatrix} N_{11}, N_{12}, \dots, N_{1m} \\ N_{n1}, N_{n2}, \dots, N_{nm} \end{pmatrix} = Q_k \qquad N_{ki} = C_{ki} - A_{ki}$$

V ... Work vektor $, w_1, w_2, w_3$ ≙ Aktuell verfügbare Ressourcen

$$V_{i+1} = V_i - Q_{ki}$$

Schreibweise:



Deadlock Recovery (mit DL Detection)

- Auflösen des Deadlock durch:
  - Abbrechen aller betroffenen Prozesse
  - Kollback bis zu def Zustand
  - Abbrechen von P. bis DL gelöst (⇒ Detection aufrufen)
  - Ressourcen werden schrittweise entzogen bis kein DL. Rollback einzelner P.

Auswahl von P mit:

- wenig CPU Time
- wenigste bisher belegten Res. (billigste)
- geringsten Fortschritt

Integrierte DL Strategie

- Aufteilen der Res. in Klassen
  - z.B. - Swappable Space
    - Prozessorres (Avoidance)
    - Hauptspeicher (Prevention)
- Circular Wait zwischen Klassen verhindern
- Unterschiedliche Strategien innerhalb Klassen

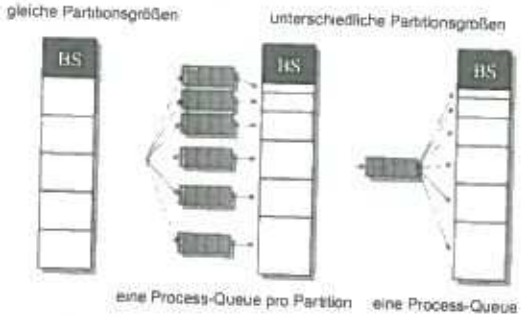
Memory Management

- Partitioning · aufteilung auf Prozesse
- Relocation Position v. Daten im Speicher, Virtual Memory
- Protection Schutz
- Sharing
- Performance

Partionierung

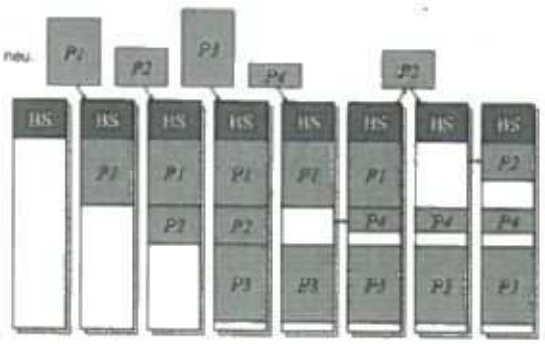
Fixed Partitioning

- Program size ≤ Part. size
- Auslagern wenn alle belegt
- Wenn Prog. size > Part size
  ⇒ Overlays



Dynamic Partitioning

- Variable Größe
- Löcher zwischen Partitionen → Compaction (aufwendig)
- Placement Strategies
  - Best fit
  - First fit
  - Next fit

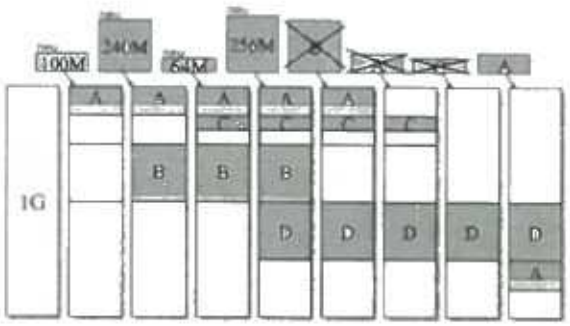


Fragmentierung

- Interne Fragmentierung : innerhalb Partition
  z.B. Fixe Part.
- Externe Fragmentierung : außerhalb Partition
  z.B Dyn Part.

Buddy System

- Nachteile von Fixed u. Dyn verringern
- Zerlegung in Buddies 2^k, mi $Val \leq k \leq maxVal$
- Schritte:
  - Lege 2^{maxVal} block an
  - Anforderung S: S auf 2^k aufrunden
  - existiert Block für S?
    - => Nein: Teile nächstgrößeren in zwei
    - wiederhole
  - Wenn Block frei gegeben wird kombiniere ursprüngliche Blöcke.



Relocation

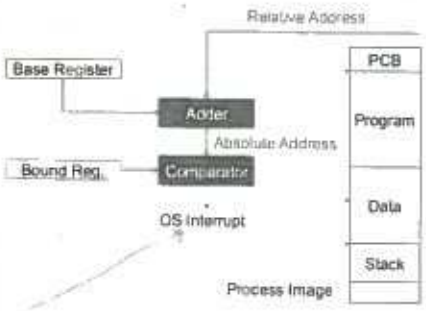
- Referenzen auf Speicher von Programme können sich ändern
  => Logische vs. Physikalische Adresse

Speicheraddressierung

- Physikalische Adresse : im Hauptspeicher
- Logische Adresse : Referenz aus Programm
- Relative Adresse : relativ zu logischer Adresse

Einfache Adressierung

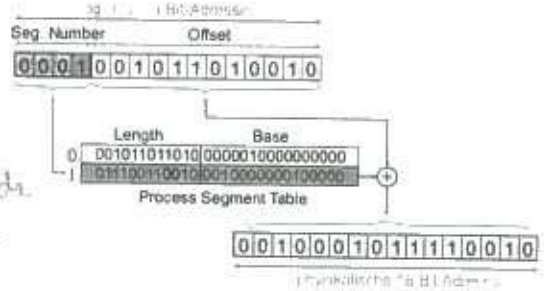
- Adrenübersetzung durch Hardware
- Base Register (Anfang)
- Bound Register (Ende)



Base Register + logische Adr. < Bound Register ansonsten Interrupt

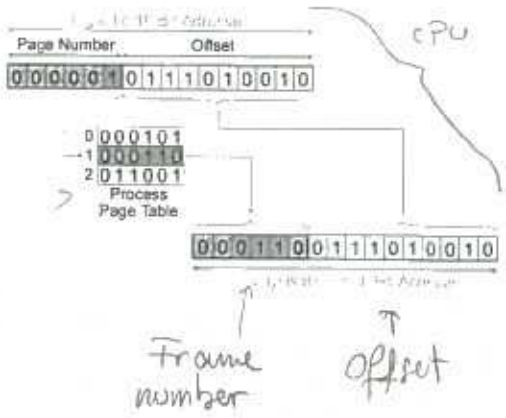
⇒ Protection

Segmentierung



- Unterteilung von Programmen in versch. Blöcke
- Segmente: Code / Daten
- Segmente vom Programm bel. verteilt
- keine interne Frag, → externe Frag.
- Segmenttabelle pro Prozess. (each segment (start, Länge)

Paging

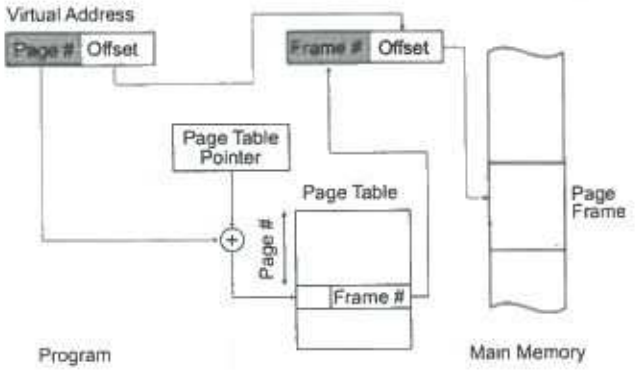


- Unterteilung des Hauptspeichers in Frames gleicher größe
- Prozesse werden in Pages gleicher Größe aufgeteilt
- Page Table pro Prozess
- Free Frame List im OS
- Einfache Adrenübersetzung, keine Addition

Virtual Memory

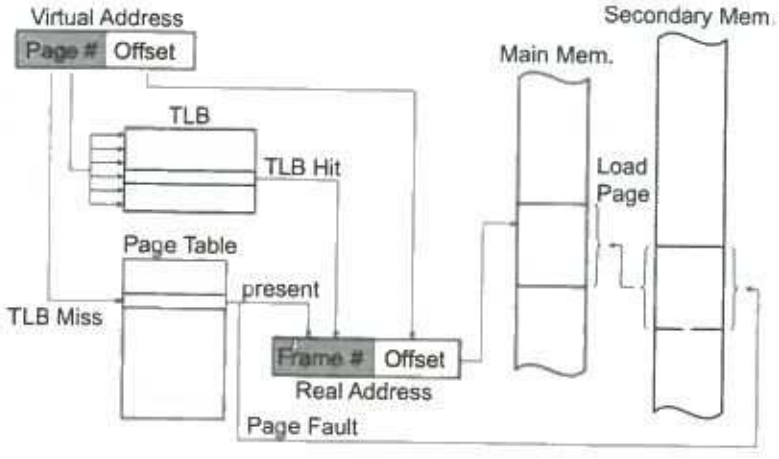
- Logische Adressen
- Seitentabelle → MMC HW unterstützen Übersetzung
- Resident Set Teile eines P. die sich im Hauptsp. befinden
- Page Fault : Address befindet sich nicht im Hauptsp.
- Thrashing : Dauernden aus- und einlagern von Frames
  ⇒ Häufige Page Faults

Paging



- Pro P. eine Page Table
- Tabelleneintrage pro Page:
  - Present Bit (Hauptsp.?)
  - Frame Number
  - Modified Bit
  - Control Bits: (r/w, Kernel/User)

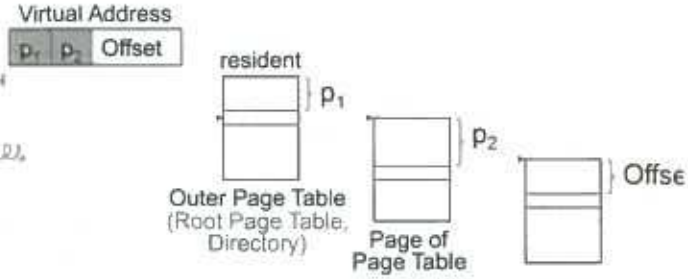
Translation Lookaside Buffer



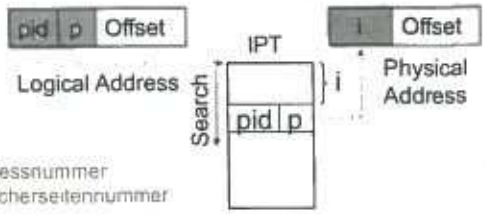
- Page Table liegt im Hauptsp.
- Cache für Page Table Einträge
  - Letzte verwendete Pages
  - Assoziativer Zugriff
  - Context switch ⇒ TLB löschen
    ≙ Process switch

Multilevel Page table

- Große Prozesse haben große Page Table
  ⇒ Page Table braucht mehrere Pages
  ⇒ Multilevel Page Table



Inverted Page Table



pid Prozessnummer
p Speicherseitennummer

- Durch Multi Level PT werden Speicherzugriffe sehr häufig.

⇒ Eine IPT per System
- jeder Eintrag zeigt auf Frame
- PID und Page number

- Suche dauert lange
  - Hashing

Fetch Policy

- Demand Paging : Wenn Adresse gebraucht wird ⇒ Viele Page Faults
- Preparing : Lädt Seiten im voraus, Lokalitätsprinzip, Lädt nicht verwendete Seiten

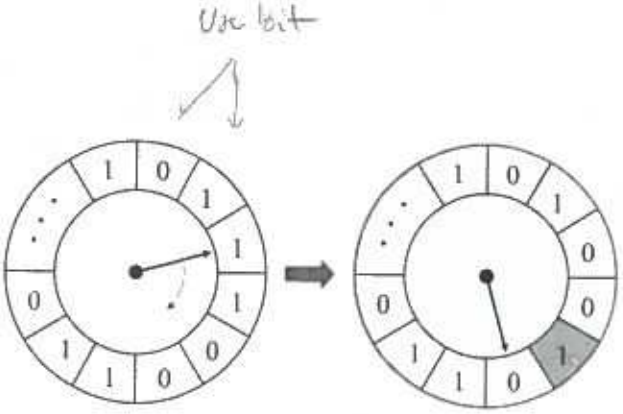
Replacement Policy (welche Seite wird ersetzt)

- Lokal : innerhalb Prozess Seiten
- Global : innerhalb System

- **OPT** (optimale Strategie)
  - wenigste Page Faults
  - ersetzt Page deren Zugriff am weitesten in Zukunft
  - keine Reale Strategie
- **LRU** (Least Recently Used)
  - Lokalitätsprinzip: Wird in Zukunft nicht benutzt
  - Page Faults nicht viel höher als OPT
  - Implementierung schwer: Zeiten speichern, suchen
- **FIFO**
  - Älteste wird ersetzt
  - Nachteil: Wenn Seite oft verwendet wird und älteste ist.
  - Implementierung einfach

Clock Policy

- Ring Puffer für Pages
- Strategie:
  - Bei PF ⇒ Zeiger 1 weiter
  - Use bit pro Frame
    1 ... wenn gebraucht
  - Bei Page Fault: Suche erste Page mit Use Bit = 0 und ref ⇒ Use = 1
    ↳ Wenn Use Bit == 1 ⇒ Use Bit = 0



- kaum mehr Page Faults als LRU
- Mehr Frames pro Prozess

Größe des Resident Set

- Wieviele Pages pro Prozess
  - Viele Pages ⇒ Einschränkung d Parallelität
  - Wenige Pages ⇒ Viele Page Faults
  - Fixed Allocation
  - Variable Allocation

Working Set Strategie

- Variable Allocation
- Working Set $W(D,t)$ = Menge der Seiten des Prozesses, die in den letzte D Zeiteinheiten referenziert wurden (zum Zeitpunkt t)
- $W(D,t)$ beschreibt Lokalität
- Wächst am Anfang schnell
- Stabilisiert sich mit der Zeit (Lokalitätsq.)
- Wächst wenn sich Lokalität ändert
- Pages die nicht im $W(D,t)$ ⇒ Löschen
- Wenn nicht genug Frames ⇒ suspend Proces, Platz für Anderen machen

Probleme beim WS:

- Mitloggen von Referenzen
- Ordnen d Referenzen
- Optimales D nicht bekannt

=> Beobachtete Anzahl Page Faults pro Zeitintervall

Protection

- kontinuierlicher Adressraum für Prozess
- Adressen werden mit Page Table abgebildet oder sind protected
- Bei Zugriff auf Adresse
  - Adresse physisch vorhanden
  - Page Fault
    - wenn nicht in fak. Speicher => Segmentation Fault

Protection Keys

Variante 1:

- Pro phys Frame 1 Prot. Key
- Jeder Prozess hat Prot. Key
- Bei Zugriff müssen beide übereinstimmen

Variante 2:

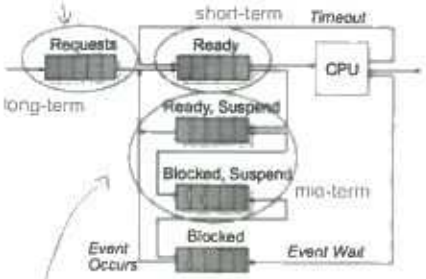
- Pro TLB Eintrag 1 Key
- Jeder Prozess hat Menge von Prot Keys
- Bei Zugriff: Prozess Keys = TLB Key
- Fehler wenn kein Key passt.

Scheduling

- Bestimmt die Abarbeitungsreihenfolge der Prozesse
- Optimierungsziele:
  - Durchsatz
  - Prozessorauslastung
  - Fairness
  - Response Time
  - Deadlines ?

- Kreierung von P.
- Mix CPU / I/O intensive?
- best Parallelität

Bestimmt nächsten Prozess



- Ein/Auslagern von Prozess

- Short-Term Scheduling
  - Entweder Scheduler oder Dispatcher bei:
    - System Calls und Traps
    - I/O Interrupt, Signale
    - Uhr-Interrupt

Prioritäten

- Führt zu Starvation

Kriterien	User-Oriented	System-Oriented
Performance	Response Time Turnaround Time Deadlines	Throughput Processor Utilization
Other	Predictability	Fairness Resource Balance

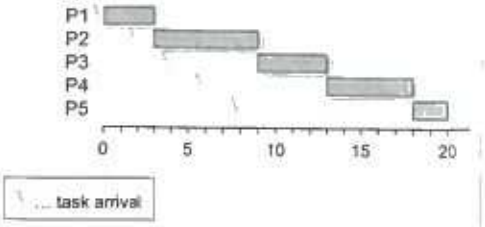
Strategien

- Selection function (Auswahl des Nächsten P.)
  - bisherige Verweildauer
  - bisherige / gesamte Exe. Time
  - Deadline
- Decision Mode
  - Non-Preemptive (Keine Unterbrechung von BS)
  - Preemptive (Unterbrechung von BS)

First Come First Serve (FCFS)

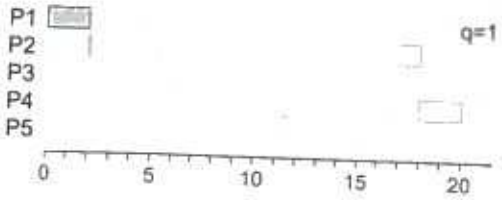
Prozess	Arrival Time	Service Time
P1	0	3
P2	2	6
P3	4	4
P4	6	5
P5	8	2

- Select.: P. der am längsten in Ready Queue
- Decision: Non Preemptive



- begünstigt:
  - Lange Prozesse
  - CPU-Intensive Prozesse
- schlechte Auslastung von CPU und I/O

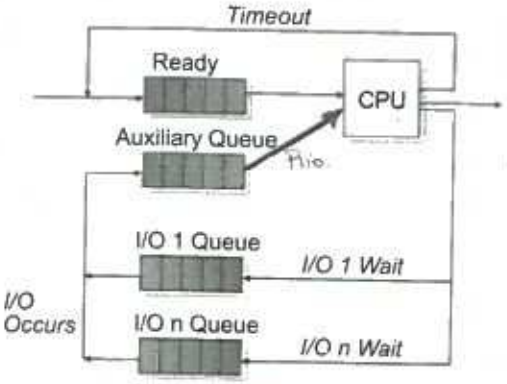
Round Robin



- Zeitscheibenlänge:
  - länger als Clock Interrupt
- benachteiligt I/O

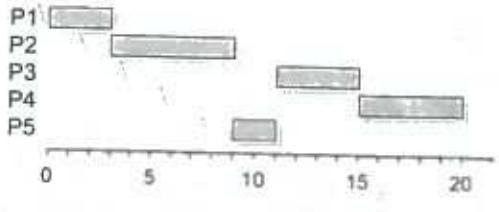
- Select.: P. der am längsten in Ready Queue
- Decision: Preemptive (Zeitscheiben, Clock-Interrupt)

Virtual Round Robin



- Round Robin mit zusätzlicher Auxiliary Queue mit höherer Priorität als Ready Queue

Shortest Procen Next



- Select: P. mit kürzesten CPU-Burst
- Decision: Non-Preemptive
- bessere Response T aber mehr varial
- Verzögerung langer Prozesse
- Probleme: Schätzung d Burst, Starvation

nicht für interaktive Aktionen

Shortest Remaining Time

- Select: P. mit kürzesten CPU-Burst
- Deasion: Preemptive

- kürzere Prozesse werden fair behandelt
- nicht so viele Interrupts wie bei RR
- Protokollieren der Service time
- Starvation möglich

Highest Response Ratio Next

- Select: Response Ratio = $RR = \frac{w+s}{s}$

  w ... bisher Wartezeit
  s ... geschätzte Service time

  P. mit größtem Response Ratio
- Deasion: Non-Preemptive
- kürzere P. werden fair behandelt
- keine Starvation
- Schätzung der Service Time notwendig

Feedback Scheduling

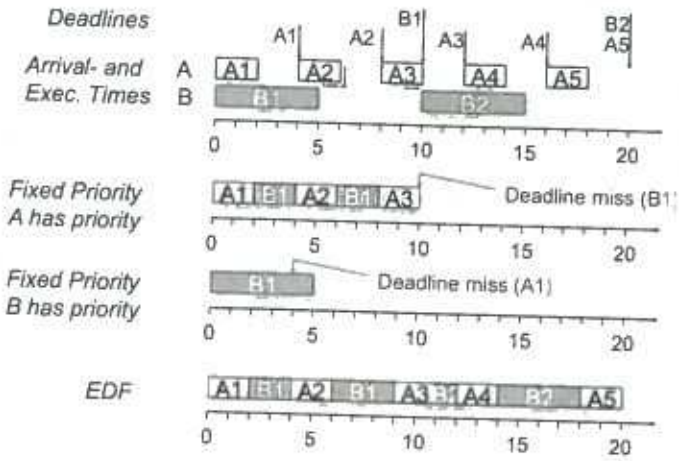
- Select: Desto mehr (bisherige) CPU Time ⇒ kleinere Priorität, Prio = Prio. eigene Prio. Queue
- Deasion: Preemptive
- Starvation möglich ⇒ Prio ++ nach gewisser Zeit.

Real-Time Scheduling

- Deadlines
- Hard vs Soft real time
- Periodische vs Aperiodische Tasks
- Scheduling: Preemptive, stat/dyn Prioritäten
- Schedulability Test zeigt machbarkeit (Deadline)

① Earliest Deadline First

- Select: Task mit frühester Deadline → Minimiert Deadline misses
- Decision: Preemptive



② Rate Monotonic Scheduling

- Selection: kürzeste Periode ⇒ höchste Prio
- Decision: Preemptive
- Periode T
- Deadline D = T
- Execution Time C ≤ D
- Tasks sind unabhängig

Schedulability Test

- Können Deadlines eingehalten werden?
- Notwendige Bedingung

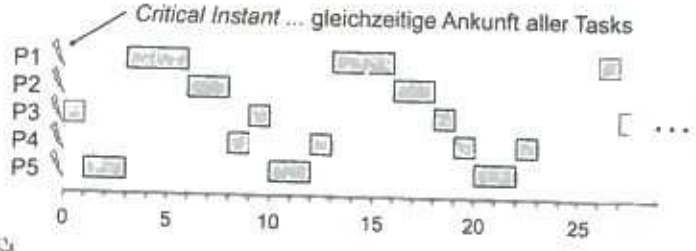
$$\sum \frac{C_i}{T_i} \leq 1$$

- Hinreichende Bedingung

$$\sum \frac{C_i}{T_i} \leq n\left(2^{\frac{1}{n}} - 1\right) \approx 0{,}69 \quad \gg n$$

- Tatsächliche erreichbare CPU-Auslastung bei RMS ist in Regel größer als $n(2^{\frac{1}{n}} - 1)$

Prozess	Laufzeit	Periode
P1	3	13
P2	2	15
P3	1	9
P4	2	16
P5	2	10



Input/Output and Disc Scheduling

- I/O Geräte
- Anforderungen an I/O
- BS:
  - Pufferung
  - Disk Scheduling
  - Disk Cache

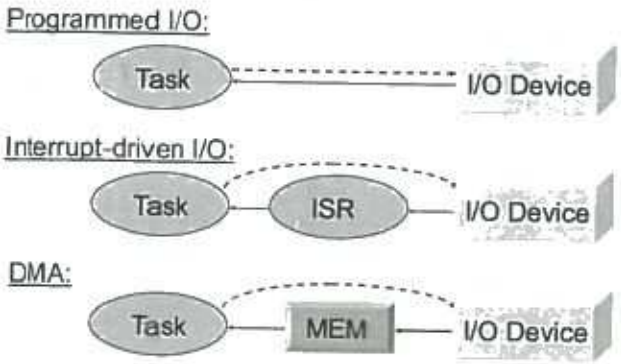
Herausforderung:

- Vielzahl an Geräten
- Mensch - Interface (Tastatur, ...)
- Maschinen I/f (Laufwerk, Sensor, ...)
- Kommunikation (Netzwerk, ...)

Merkmale:

- Datenrate
- Anwendung
- Ansteuerung (Polling, Interrupt, ...)
- Transfer einheit (Zeichen, Blöcke)
- Daten Repräsentation (Byte Order, Parity)
- Fehlerbehandlung

Funktionen



Programmed:

- Synchron, Busy Waiting

Interrupt:

- Asynchron

DMA:

- Asynchron, I/O-Gerät schreibt Daten direkt in Speicher

DMA:

- Übernimmt Bus
- Transfer: Blockweise, Cycle Stealing
- 1 DMA Modul → N I/O-Geräte

I/O-Channel:

- Erweiterung zu DMA
- Eigener I/O-Prozessor, kontrolliert Gerät direkt mit I/O-Befehle
- I/O-Befehle in Hauptsp.
- Typen:
  - Selector Channel : Mehrere Geräte, immer 1 aktiv
  - Multiplexer Channel: Mehrere gleichzeitig

Kriterien

- Effizienz : I/O oft Bottleneck
- Feexibilität: Einfachheit, Ersetzbarkeit, Iface: open, close, write, read

Struktur:

- Einheitliche Interface
- Logical I/O : open, close, read, write ≙ Operationen
- Device I/O : Operationen ⇒ I/O-Kommandos, Puffering
- Scheduling and control : Queuing, Interrupt Handling

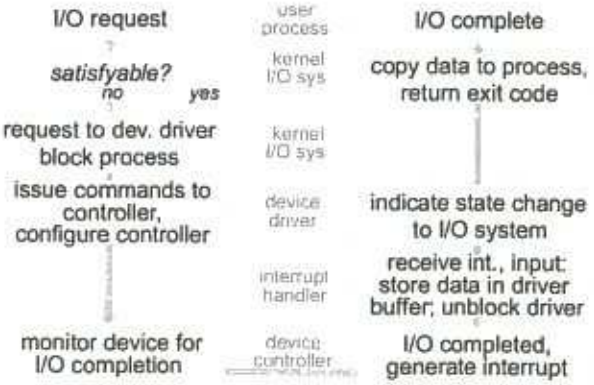
File System Geräte:

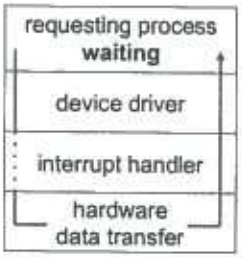
- Logical I/O:
  - Directory Management (Dateien, Ordner, add, delete, ...
  - File System (logische Struktur v. Daten, open, close, read, write, ...)
  - Physical Organization (Spuren, Sektoren, ...)

Blocking vs Non-Blocking

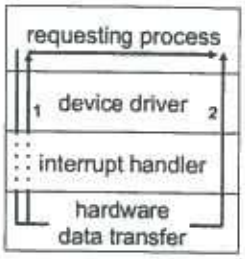
- Blocking: Running → Wait
- Non-Blocking: Fkt. returnt gleich

Synchron vs Asynchron





Synchronous I/O (blocking or non-blocking)



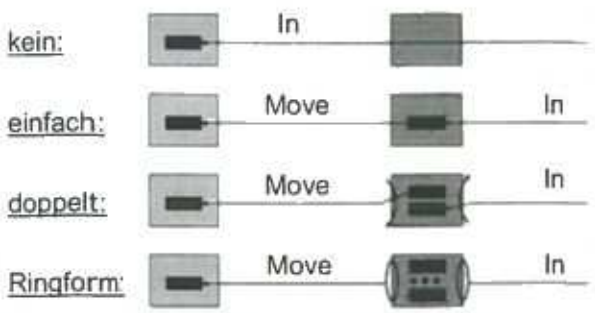
Asynchronous I/O

Parallelität und IS

- Prozess wird durch warten auf I/O verzögert
- Mid-Term Scheduling (Swapping)
  - Daten können nicht ausgelagert werden; Inkonsistenz
  - Deadlock möglich

Puffern

- Zusammenfassen von Operationen
- Entkopplung BS – Prozess, I/O – Swapping
- Langzeit Perf. < Device Perf.
- Kosten: Speicher, Management



Disk I/O -

Scheduling:

- Arm (Seektime T_S)
- Rotational Delay T_{RD}, bis Anfang v. Sektor
- Transfer Time T_{TF}, übertragen der Daten
- Average Access Time: T_A,

b ... Anzahl bytes, Size
N ... Anzahl Bytes pro Spur
r ... U/sec

$$T_{RD} = \frac{1}{2 \cdot r} \quad T_{TF} = \frac{b}{r \cdot N}, \quad T_A = T_S + T_{RD} + T_{TF}$$

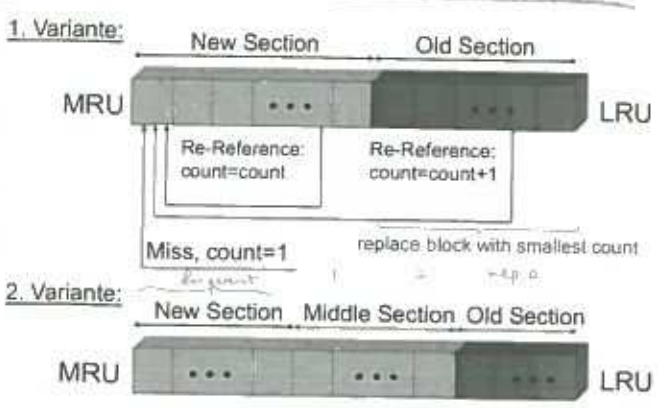
Strategien

- I/O - Operationen umordnen
- Priority
- FIFO
- LIFO
- Shortest Service Time First (SSTF)
- SCAN
- RAID

Cache:

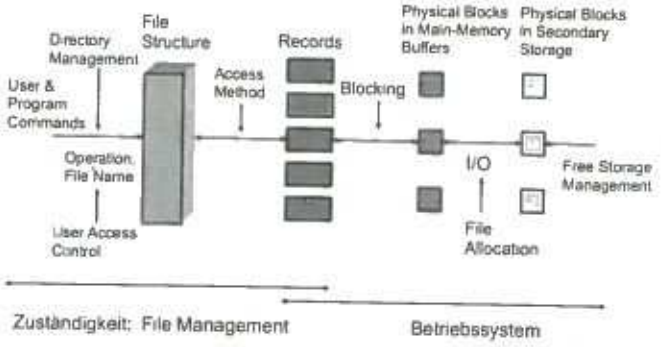
- Lokalität
- Austausch von Sektoren
  - LRU
  - LFU
  - Frequency Based Replacement

Frequency based Replacement:



File - Management

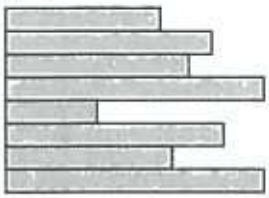
- Persistenz nach Prozess beenden
- Sharing



Data Organization

- Zugriffszeit
- Aktualisierbarkeit
- Platzverbrauch
- Wartung
- Zuverlässigkeit

Pile:

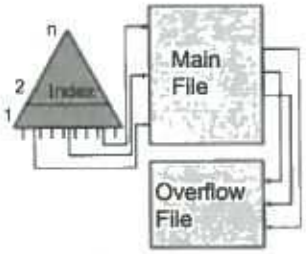


- Records variabler Länge
- Variable Menge an Feldern
- Chronologische Reihenfolge

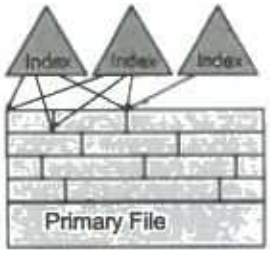
Sequential File:

- Records fixer Länge
- Fixe Menge an Feldern mit fixer Reihenfolge
- Reihenfolge durch Key

Indexed Sequential File:

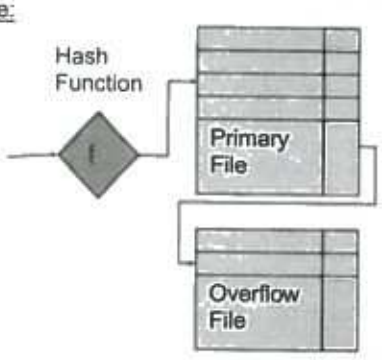


Indexed File:



- Index für direkten Zugriff
- Alle möglichen Suchfelder

Hash File:



- Keine sequentielle Reihenfolge

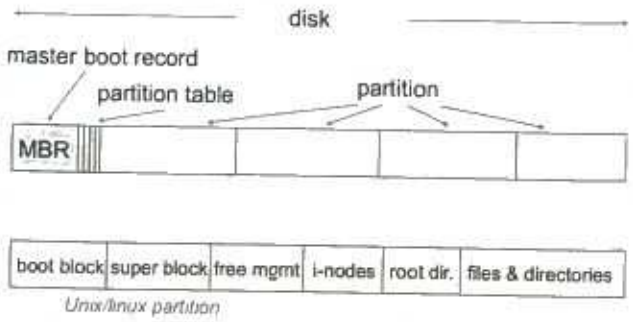
File Types:

- Regular Files: ASCII, Binary
- Directories
- Special Char. Files
- Special Block Files

Attributes:

- Protection, Password, Creator, Owner
- Read only, hidden, archive
- Record length
- Creation / Access / Modified time
- Size

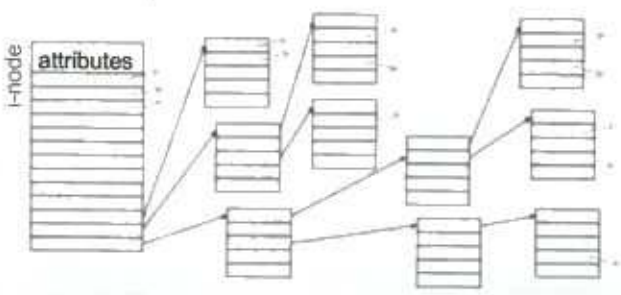
Layout



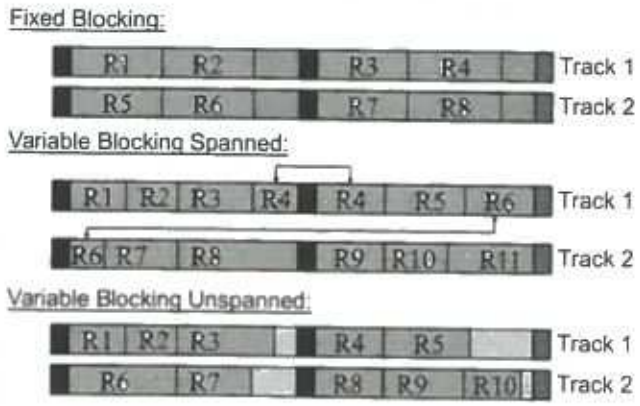
- MBR: Boot code, Partition Table

Datei-Implementierung

- Cautiguos Allocation:
  - 1 Datei ≙ 1 Block
  - Vorteil: gute Perf. beim Lesen
  - Nachteil: Vergrößerung v. Datei ⇒ externe Fragmentierung
- Chained Allocation
  - 1 Datei ≙ N-Blöcke, mit Zeiger verkettet
  - Nachteil: Keine Lokalität, Random Access langsam, viel Platzbedarf
- Indexed Allocation
  - 1 Datei ≙ N-Blöcke, mit Zeiger im Speicher (File Allocation Table)
  - Vorteil: Schnell
  - Nachteil: Viel Platz im Speicher
- I-Nodes
  - 1 Datei ≙ 1 Datenstruktur ≙ N-Blöcke
  - Vorteil: I-Node in Speicher laden wenn nötig



Block-Verwaltung



⇒ Verschnitt

⇒ kein Verschnitt

⇒ Verschnitt

Directories

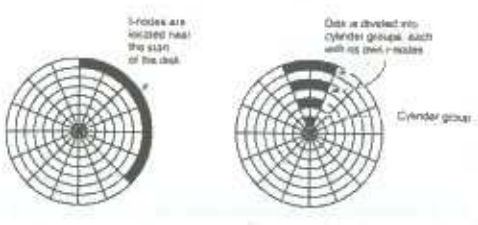
- Suchen:
  1) Root Verzeichnis
  2) Pfad interpretieren
- Root Verzeichnis
  - Unix: Start von I-Nodes in super block
  - Win: MFT
- Datei Attribute:
  - In Directories
  - In I-Nodes

Freie Blöcke

- Chained Free Portions
  - Zeiger + Länge auf zusammenhängende Block
- Bit Tables
  - 1 Block ≙ 1 Bit im Vektor
  - Geringer Platzbedarf, guter Überblick
  - Indexing: 1 Freier Block ≙ 1 spezielle Datei

Performance

- Disk Cache
- Block Read Ahead
- I-nodes anordnen



Security

- Confidentiality (Geheimhaltung) ⇒ Exposure
- Integrity (Sollzustand) ⇒ Modification
- Availability → DoS
- Authenticity (Korrektheit d Identität)
- Accountability (Protokoll)

Typen:

- Passiv: Monitoring, Abhören
- Aktiv: Manipulation
- DoS: Service Unterbrechen
- Exposure: Unautorisierter Leser
- Modification. Datenintegrität, Man in Middle

Malware:

- Virus. Infiziert Programme ⇒ DoS, key logger
- Worm: Repliziert sich selbst
- Trojan Horse: Gewünschtes Programm mit Malware
- Logic Bomb: Startet bei Bedingung
- Trapdoor: Geheimer Einstiegspunkt
- Port Scan
- DoS + Distributed DoS

Methoden

- Auslesen von Speicher
- System calls
- Modification von BS
- Login form
- Social Hacking

Buffer Overflow

- Über Bufferende schreiben
- Return-Adresse ändern ⇒ Exploit Code
- Exploit Code schreiben

Design Principles

- Open Design
- Default: keine Berechtigung
- Überprüfung gegenwärtiger Berechtigung
- ...

Passwörter

- Durchprobieren
- Wörterbücher
- Typische Ersetzungen (i→1, e→3)

⇒ Maßnahmen:

- One Time Passwort (TAN)
- Zeitablauf
- Zeitverzögerung bei Login
- ...

Authentication

- Ist Benutzer der richtige
  - Schlüssel: Chipcard
  - Fingerabdruck
  - Passwort, Frage

Protection

- Kontrollierter Zugriff auf Dateien und Programme

Objekte: CPU, Speicher, Dateien

- Unix: Domain Paar

  Domain vs Prozess (UID, GID)

- Access Matrix:

  Domain × Object ↦ Right

- Dynamic Protection

  Domain × (Object, Domain) ↦ Right

Access Control Lists

- Rechte in Objekte gespeichert

Capability Lists

- Capability pro Prozess gespeichert
- Tickets, weitergabe, vererbung

Lock-key System

- Jedes Object hat mehrere Locks
- Domain hat Keys
- Prozess authorisiert wenn Key in Lock passt.

Intrusion Detection

- Threshold detection (Login versuche)
- Typische Zugriffsmuster
- Anomaly Detection
- Audits (Logs)

Netzwerke

Distributed System

- Resource Sharing (Files, Drucker, ...)
- Load Sharing
- Zuverlässigkeit (Redundanz)
- Kommunikation

Dist Operating System

- Network Operating System
  - Für Benutzer sichtbar
  - Spezielles User-Interface
  - Remote Login
- Distributed OS
  - Für Benutzer nicht sichtbar
  - Data Migration
  - Computation Migration (RPC, ...)
  - Procen Migration
    - Load Balancing
    - Speedup
    - Hardware preference
    - ...

Network Structure

LAN:

- kleine geo. Ausdehnung (~10km)
- Multi Access Bus; Ring, Stern
- Speed: 10Mbit/s - 1Gbit/s
- Broadcast: schnell, billig
- Knoten: PCs, Workstation, Server, ...

WAN:

- große geo. Ausdehnung
- Speed: 1Mbit/s - 45Mbit/sec
- Broadcast: mehrere Nachrichten

Topologie

- Installationskosten
- Kommunikationskosten
- Zuverlässigkeit (Ausfall v. Knoten?)

Routing

- Info über Alternativen, Speed, Kosten

Strategien:

- Fixed Routing
- Virtual Routing: Pro Session
- Dynamic Routing: Pfad wird dyn bestimmt
  ⇒ Out of Order möglich

DNS:

- Name Resolution
  Name ⇒ IP
- DNS-Lookup: (mail.tuwien.ac.at)
  - ac.at → at
  - tuwien.ac.at → ac.at
  - mail.tuwien.ac.at ⇒ tuwien.ac.at
  - ⇒ 128.130.35.36

Connection Strategy

- Circuit switching : Permanente phys Verbindung
- Message switching : Temp. Verbindung für Nachricht
- Packet switching : Nachricht => Pakete mit versch. Pfade

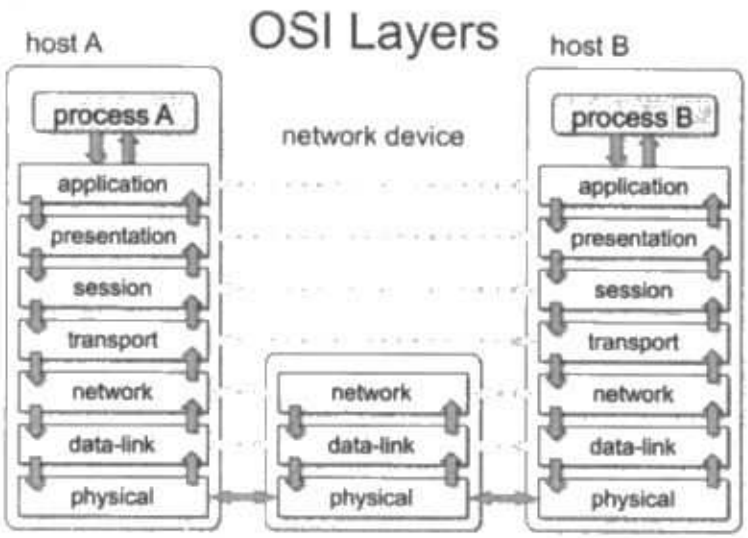
LAN-Contention

- Collision Detection
- CSMA/CD : Carrier Sense Multiple Access / Collision Detection
  - Senden wenn frei
  - Bei Kollision abbrechen und nach random Zeit wieder probieren
  - Hochlast => Viele Kollisionen
  - Ethernet
- Token passing:
  - Senden wenn Knoten Token hat
  - Token wird in Kreis weitergegeben
  - Ringstruktur : Logisch oder Physikalisch
  - Token generierung bzw. regenerierung (bei Fehler)

Kommunikationsprotokoll

- Satz von Regeln und Konventionen für Kommunikation von Daten
- Syntax, Semantic und Synchronisation
- Eigenschaften:
  - Identification d. phy. Schnittstelle
  - Verbindungs-aufbau/-abbau
  - Botschaften (Start, Ende)
  - Flow Control, Error Handling

- Verschiedene Schichten um Komplexität zu beherrschen
- Layer n bietet Interface zu Layer n+1 mit Service Access Point
- Komm Partner immer gleiche Schicht von anderen Knoten



① physical: Mechanische und elektrische Codierung von Bit Streams

② data-link: Gesicherte Übertragung von Frames. (Flow Control, Error Handling)

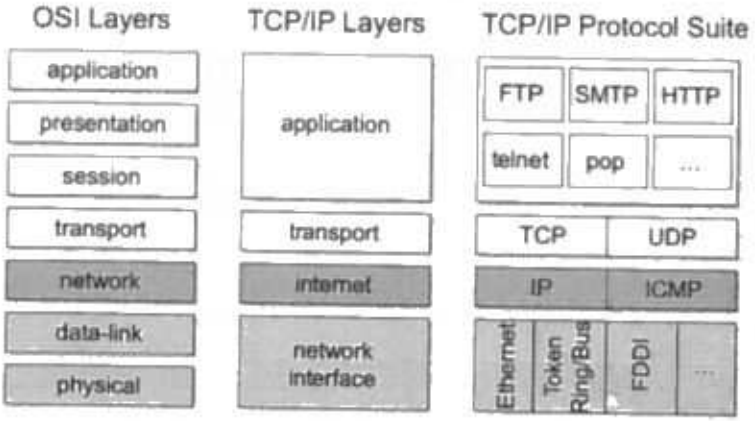
③ network: Verbindung und Routing von Pakete (Adressierung, IP, ...)

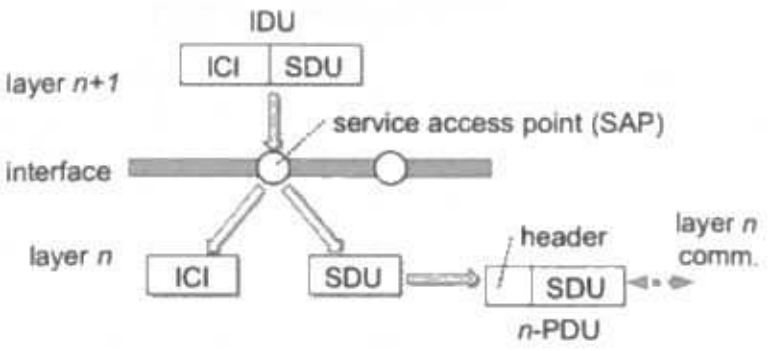
④ transport: Partitionierung von Nachrichten, Ordnen (TCP, UDP, ..)

⑤ session: Verbindung zwischen Prozessen, Remote Login, (FTP, SSH, ..)

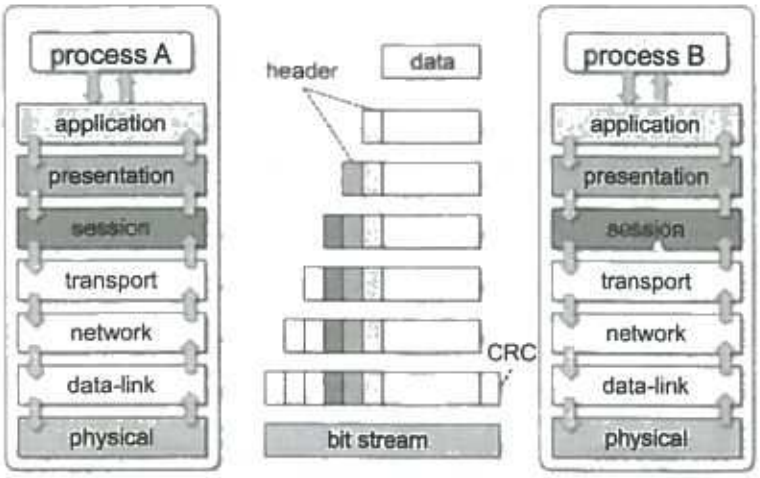
⑥ presentation: Auflösung versch. Formate, Syntax u Semantik (FTP, SSH, ..)

⑦ application: Interaktion mit Programmen (FTP, SSH, ..)





OSI Communication



Distributed File Systems

- Verteilte Speichereinheiten
- Globale Namensstruktur für alle Dateien
- Remote Service Mechanism
- Cachen von Teilen:
  - Consistency Problem
  - Disk vs Main Memory
  - Write trough vs. Delayed
- Stateful vs Stateless File Service (Performance!)
  - Stateful: Client öffnet File, Server behält seek pos usw. (Recovery?)
  - Stateless: Jeder Request enthält seek pos.
- Replikation
- Availability

Distributed Mutual Exclusion

- Zentralisiert mit Server
  - Prozess sendet Request zu Koordinator
  - K. sendet Reply an P. (sobald k.A. frei)
  - P. sendet Release an K. sobald k.A. fertig)
- Verteilt
  - Broadcast von Request mit timestamp zu P. (mehrere)
  - P. sendet wenn selbst kein Halten ⇒ ok
  - P. wartet auf Reply von Allen Prozessen
  - Nachteil: Jeder muss jeden kennen, Fehler von einen ⇒ Blockiert Protokoll
- Token
  - Token sig. krit. A.
  - Besitzer arbeitet und gibt Token weiter
  - Nachteil: Tokenverlust

Betriebssysteme V1 25.02

Threads

PROCESS
- Control Block
- Process Information
- State Information
- Kernel Information
- Dispatching Information

THREAD
- Thread Control Block
- per Space
- Kernel Space

Threads teilen mehrere Ressourcen, aber unterschiedliche Dispatching Information

- Vorteile
  - Erzeugen von Threads ist deutlich schneller als bei Prozessen
  - Kommunikation einfacher
- Probleme
  - Synchronisierung
- Zustände
  - Ready
  - Blocked
  - Running

Bei Thread gibt keine Spende => selbst essarcen

- Implementierungen
  - im User Space (User Level Thr.)
  - im Kernel Space
- User Level Thread (ULT)
  - kein User Mode Switch
  - Kernel weiß nichts von Threads
  - Blockierende Thread blockt Prozess (somit alle Threads im Prozess)
  - Verteilen auf mehrere Cores nicht möglich
- Kernel Level Threads (KLT)
  - User Mode Switching (x Mode Switch)
  - Einzelne Threads können blockieren
- Kombination ULT/KLT