

8. Externspeicherverwaltung

Speicherzuordnungsstrukturen

- ☐ Allgemeine Aufgaben
- ☐ Realisierung eines Dateikonzeptes
- ☐ Verfahren der Blockzuordnung
 - statisch
 - dynamische Extent-Zuordnung
 - dynamische Block-Zuordnung
- ☐ Lesen und Schreiben von Blöcken
 - Maßnahmen zur Fehlertoleranz (z.B. stabiles Schreiben)
 - Nutzung von Speicherredundanz (z.B. Spiegelplatten)

Seitenzuordnungsstrategien

- ☐ Segmentkonzept
- ☐ indirekte Einbringstrategien
- ☐ Bewertung

310.

Aufgaben der Externspeicherverwaltung

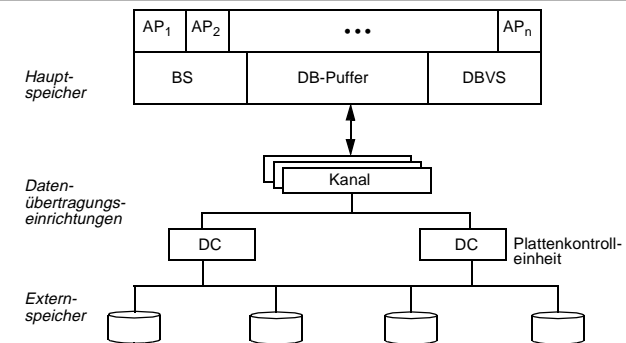
- ☐ Verwaltung externer Speichermedien (Magnetplattenspeicher, CD-ROM, Magnetbänder)
- ☐ Verbergen von Geräteeigenschaften
- ☐ Abbildung von physischen Blöcken
- ☐ Kontrolle des Datentransports von/zum Hauptspeicher
- ☐ Realisierung einer mehrstufigen Speicherhierarchie
- ☐ Fehlertoleranzmaßnahmen (z. B. um defekte Blöcke zu "reparieren")

Vorteile eines Dateikonzeptes?

- ☐ dynamisches Aktivieren und Deaktivieren der Dateien
- ☐ Adressierung der Blöcke in einer Datei erfordert nur wenige Bytes
- ☐ explizite Unterscheidung zwischen einem Hauptspeicherzugriff und einem Externspeicherzugriff.
 - wie groß ist der Unterschied zwischen Hauptspeicher und Magnetplattenspeicher?

312.

Aufbau einer zweistufigen Speicherhierarchie



311.

8.1 Dateikonzept

- ☐ Dateien repräsentieren einen "langen Byte-Vektor" bzw. eine Folge von Blöcken.
- ☐ Dateien können i.a. nur am Ende erweitert und verkleinert werden.
- ☐ Dateien werden auf externen Speichermedien (z. B. Plattenspeicher) verwaltet.
- ☐ Das Dateisystem verwaltet Dateien und bietet dem Benutzer den ADT Datei und entsprechende Operationen auf dem ADT an.
- ☐ Dateisystem
 - erlaubt eine Reihe von Operationen (vereinfachte Definition)
 - ist als lokales Dateisystem oder als eigenständiger Datei-Server realisiert
 - verwaltet typischerweise einen hierarchischen Namensraum
- ☐ Operationen auf Dateien
 - Anlegen/Löschen

```

enum STATUS = {OK, FAILED}; /*Ergebnisanzeige der Dateioperation*/

typedef char * filename;
STATUS create (filename);
STATUS delete (filename);
      
```

Welche Parameter sind für **create** erforderlich?

313.

- Verarbeiten

```
typedef struct { } FILE; /*Standard-Def. bei UNIX*/
STATUS open (filename, * FILE);
STATUS close (* FILE);
```

Welche Aktionen sind bei **open** erforderlich?

- Lesen/Schreiben

```
STATUS read (FILE, file_address, memory_address, length);
STATUS write (FILE, file_address, memory_address, length);
```

Operationen beziehen sich auf Dateikontrollblock, der für die Anwendung angelegt wurde (FILE)

- weitere Operationen (insbesondere wichtig für DBS):

Operationen auf Blöcken fester Größe:

- readBlock(FILE, file_address, memory_address)
- writeBlock(FILE, file_address, memory_address)

Operationen auf Mengen von Blöcken, z.B.

- vreadBlock(FILE, file_address, memory_address[], nblocks)
- multiBlockRead(FILE, file_address[], memory_address[], nblocks)

314.

- Freispeicherverwaltung für Externspeicher

- formatierte Bitlisten, hierarchische Struktur

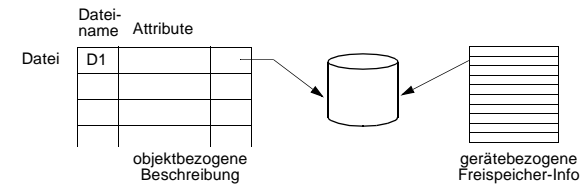
- Anlegen/Reservieren von Speicherbereichen

- Erstzuweisung (bei der Erzeugung einer Datei)
- Erweitern (bei einer Erweiterung der Datei)

316.

Realisierung eines Dateikonzpts

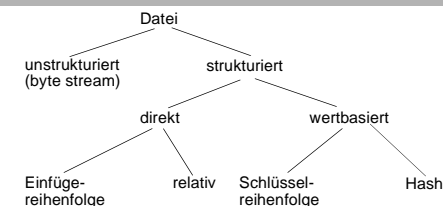
- Katalog für alle Dateien (Urdatei)
 - feste Position
 - effiziente Implementierung/Verarbeitung
- Beispiel: Katalog



- objektbezogene Beschreibung durch Dateideskriptor
 - Dateiname, OwnerID,...
 - Zugriffskontrollliste
 - Zeitinformation über Erzeugung, letzter Zugriff, letzte Archivierung,...
 - Dateigröße, Externspeicherzuordnung,...

315.

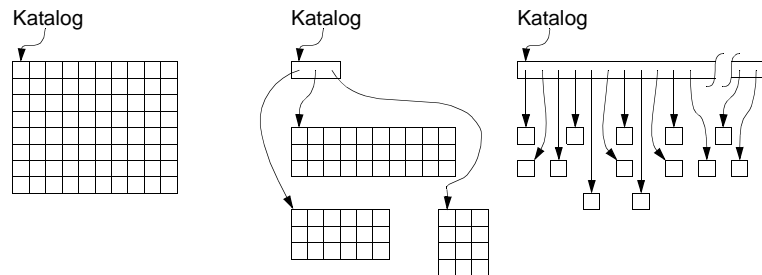
Dateiorganisationsformen



- Datei in Einfügereihenfolge (entry-sequenced): Einfügen an Dateiende, Satzadresse ist RBA (relative Byteadresse), sequentieller Zugriff oder direkter Zugriff über RBA
- relative Datei: Organisationsform ist ARRAY OF RECORDS
- wertbasierte Dateien erlauben Zugriff über Satzschlüssel
- Datei in Schlüsselreihenfolge (key-sequenced): Speicherung der Sätze in Schlüsselreihenfolge, direkter Zugriff über Schlüssel und sortiert-sequentieller Zugriff
- Hash-Datei: direkter Zugriff über Schlüsseltransformation.

317.

Blockzuordnungsverfahren



Statische Datei-Zuordnung

- direkte Adressierung
- minimale Zugriffskosten
- keine Flexibilität

Dynamische Extent-Zuordnung

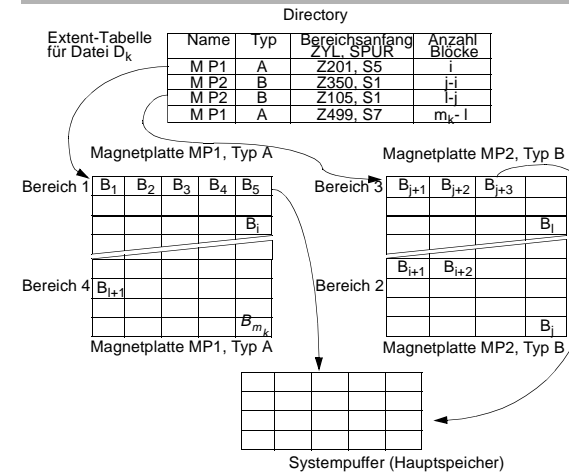
- Adressierung über eine kleine Tabelle
- geringe Zugriffskosten
- mäßige Flexibilität

Dynamische Block-Zuordnung

- Adressierung über eine große Tabelle
- hohe Zugriffskosten
- maximale Flexibilität

318.

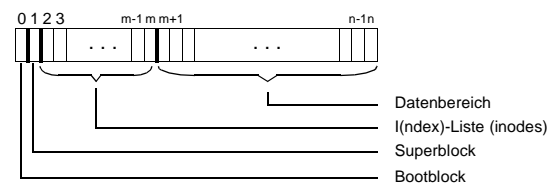
Blockzuordnung über Extent-Tabellen



319.

8.2 Das Datei-System in UNIX

□ Org. von Plattenspeicher:

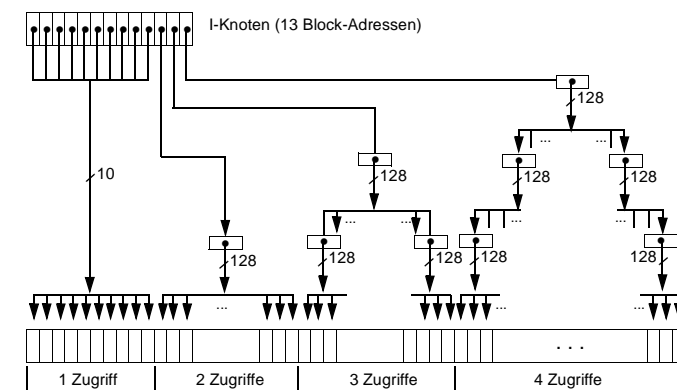


□ Directory-Einträge ordnen dem Datei-Namen einen Index i ($1 \leq i \leq m-1$) zu. Jeder Knoten der I-Liste enthält folgende Informationen:

- Identifikation des Datei-Eigentümers
- Schutzbits
- Adressen von 13 physischen Blöcken
- Größe der Datei in Bytes
- Zeitpunkt der Erstellung, letzten Referenz, letzten Modifikation
- Anzahl der Verweise auf die Datei
- Art der Datei

320.

Speicherzuordnung im UNIX-Dateisystem



Die Dateiblocke der untersten Ebene sind nur logisch zusammenhängend

321.

8.3 Atomares Schreiben eines Blocks

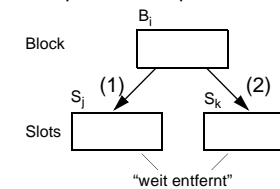
- ❑ Wunsch bei der Ausführung von writeBlock:
 - Speicherplatz auf der Platte hat den neuen Wert zugewiesen bekommen.
 - ❑ Einfaches Schreiben
 - Schreiben des Blocks als atomare Aktion (ganz oder überhaupt nicht) kann nicht garantiert werden.
 - Schreiben kann durch transiente und dauerhafte Fehler zu falschen Resultaten führen:
 - z. B. Crash des Systems wenn der erste Teil des Blocks geschrieben ist.
 - z. B. defekter Block, der von der Plattenkontrolleinheit nicht als defekt erkannt wurde.
- Schreibfehler im Katalog sind besonders kritisch (?)
- ❑ UNIX bietet nur einfaches Schreiben an
 - fsck prüft das Dateisystem auf Konsistenz (wenn mal was schief geht)

322.

- Annahme: ein Block wird nicht in einen falschen Slot geschrieben, sonst ist read-after-write erforderlich.
 - Lesen von B_i erfolgt erst von Slot S_j . Falls es erfolgreich ist, wird angenommen, daß es sich um die jüngste gültige Version von B_i handelt.
 - Falls das Lesen von Slot S_j scheitert, wird Slot S_k gelesen.
 - Da ein Systemausfall nur einen Schreibvorgang unterbrechen kann, ist bei Wiederanlauf stets eine Version des Blockes verfügbar.
- ⇒ Schreiben ist somit gesichert gegen dauerhafte Fehler.
Der Fall, beide Versionen sind nicht lesbar, gilt als unerwartet.
- ❑ Schreiben mit Logging (logged write)
 - Nach dem Lesen wird ein Block B_i mit seinem alten Inhalt auf einen sicheren Platz L geschrieben.
 - Nach Aktualisierung wird B_i mit einem einfachen Schreiben auf seinen alten Platz zurückgeschrieben (ggf. read-after-write).
 - Falls das Schreiben erfolgreich war, wird die Kopie von B_i auf L nicht mehr gebraucht.

324.

- ❑ Sicheres Schreiben
 - nach einem einfachen Schreiben wird der Block sofort wieder gelesen und mit dem Originalblock verglichen.
 - Wiederholung dieser Operationsfolge, bis der Block erfolgreich geschrieben ist.
 - Schreiben ist gesichert gegen transiente Fehler: dauerhafte Fehler können jedoch zu falschen Resultaten führen.
- ❑ Stabiles Schreiben (duplexed write)
 - Jeder Block hat eine Versionsnummer, die bei jedem stabilen Schreiben erhöht wird.
 - Jeder Block wird in festgelegter Reihenfolge in zwei verschiedene Slots S_j und S_k geschrieben
 - Prinzip: Stabiler Speicher



- einfaches Schreiben: (1) dann (2) (synchron)
- Atomizität für einen Block
- verschiedene Platten, Kontroller, E/A-Pfade

323.

Zusammenfassung

- ❑ Speicherzuordnungsstrukturen erfordern effizientes Dateikonzept
 - viele Dateien variieren in der Größe
 - Wachstum und Schrumpfung erforderlich
 - permanente und temporäre Dateien
- ❑ empfohlene Datei-Eigenschaften:
 - direkter und sequentieller Blockzugriff
 - Blockgröße pro Datei definierbar
 - Blockzuordnung über dynamische Extents
 - ⇒ Blockzuordnungsverfahren von UNIX untauglich für große Dateien
- ❑ Schreiboperationen sollen atomar ablaufen (z. B. durch Schreiben der "before images" in Logging-Dateien).

325.