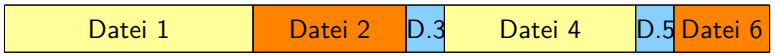
Kontinuierliche Allokation



Jeder Datei wird eine Menge zusammenhängender Blöcke zugeordnet (beim **Anlegen reserviert**).

Vorteile:

- Einfach zu implementieren (nur die Adresse des ersten Blocks ist zu speichern).
- Sehr gute Performance beim Lesen und Schreiben der Datei (minimale Kopfbewegungen).

Nachteile:

- Maximalgröße der Datei muss zum Erzeugungszeitpunkt bekannt sein, Wachsen (Append) ist nicht möglich
- Externe Fragmentierung durch Löschen / Überschreiben
- Verdichtung extrem aufwändig / langwierig

Einsatzgebiete:

- Echtzeit-Anwendungen (zusammenhängende Dateien (*contiguous files*) für kalkulierbare Zugriffszeiten)
- Write-Once Dateisysteme (CD/DVD, Logs, Backups, Versionierung).

Notizen

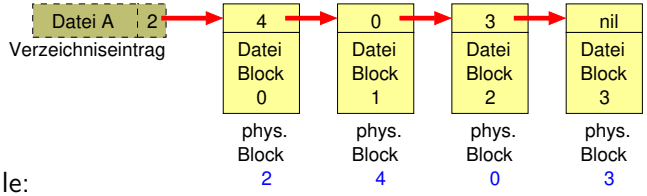
Allokation mittels verketteter Liste

Idee: Speicherblöcke einer Datei werden durch Verweise miteinander verkettet.

Jeder Block hat einen **Verweis auf Nachfolger-Block**

Verweis z.B. direkt am Beginn jedes Speicherblocks

Verzeichniseintrag verweist auf ersten Block der Datei



Vorteile:

- Keine Externe Fragmentierung

Nachteile:

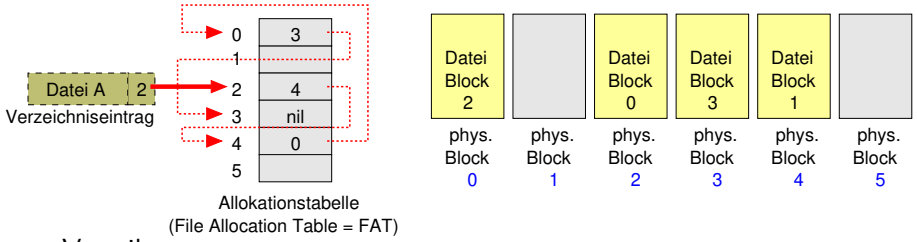
- Wahlfreier Zugriff ist seeehr langsam.
- Es steht nicht der gesamte Datenblock für Daten zur Verfügung.

Notizen

Allokation mittels verketteter Liste und Index

Hochschule RheinMain

Speicherung der Verkettungsinformation in einer separaten Tabelle (Index oder **F**ile **A**llocation **T**able = FAT).



Vorteile:

- Gesamter Datenblock steht für Daten zur Verfügung.
- Akzeptable Performance bei direktem Zugriff, da der Index im Arbeitsspeicher gehalten werden kann.

Nachteile:

- Die gesamte Tabelle muss im Arbeitsspeicher gehalten werden. Kann bei großer Platte sehr speicherplatzaufwändig sein.

Beispiel: MS-DOS FAT File System

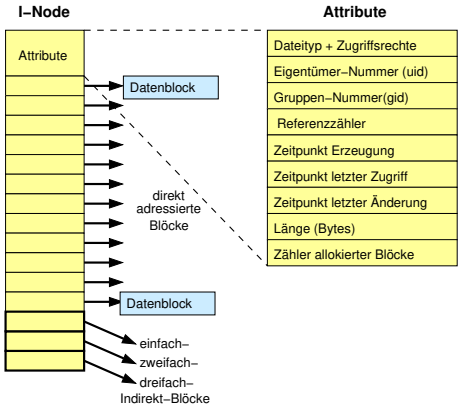
Notizen

Allokation mittels Index Nodes (1)

Hochschule RheinMain

Definition:I-Node

Ein **I-Node** (UNIX: *inode*) oder Index Node ist ein Dateikontrollblock



enthält neben Attributen der Datei eine Tabelle mit Adressen von zugeordneten Plattenblöcken.

Ursprung Beispiel: BSD UNIX Fast File System (ufs)

Notizen

Allokation mittels Index Nodes (2)

Hochschule RheinMain

Logische Blocknummern einer Datei ...

...werden fortlaufend vergeben,
beginnend bei den direkt adressierten Blöcken.

Einige wenige (~12) Blockadressen sind im Inode selbst gespeichet

→ Nach Öffnen einer Datei und damit verbundenem Einlagern des Inodes in den Hauptspeicher stehen diese Adressen sofort zur Verfügung.
→ Schneller Zugriff bei kleinen Dateien.

Für größere bis sehr große Dateien werden nach und nach einfach, zweifach und dreifach indirekte Blöcke zur Speicherung verwendet.

→ Zugriffsgeschwindigkeit sinkt bei größeren Dateien.

Beispiel: Linux ext2, ext3

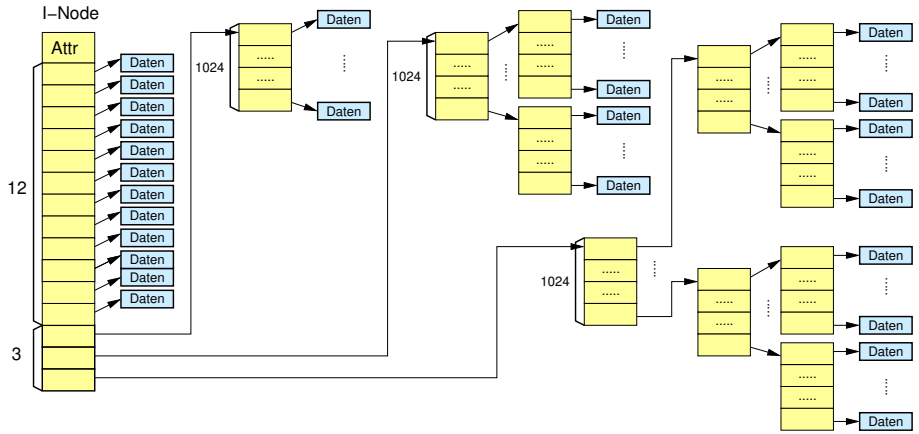
Blockgröße: 4KB
12 direkt adressierte Blöcke → 48KB
Es sind 1-fach, 2-fach und 3-fach indirekte Blöcke vorgesehen
Indirekte Blöcke (4KB) speichern bis zu 1024 ($=2^{10}$) weitere Verweise
⇒ Maximale Dateigröße:
$(12 + 2^{10} + 2^{30} + 2^{30}) \cdot 4KB = 48KB + 4MB + 4GB + 4TB \approx 4TB$

Notizen

Allokation mittels Index Nodes (3)

Hochschule RheinMain

Nutzung von Indirekt-Blöcken



Notizen

Implementierung von Verzeichnissen

Hochschule RheinMain

Hauptaufgabe des Verzeichnissystems:
Abbildung der Zeichenketten-Namen von Dateien in Informationen zur Lokalisierung der zugeordneten Plattenblöcke.

Bei Pfadnamen werden die Teilnamen zwischen Separatoren schrittweise über eine Folge von Verzeichnissen umgewandelt.

Verzeichniseintrag liefert bei gegebenem Namen (Teilnamen) die Information zum Auffinden der Plattenblöcke:

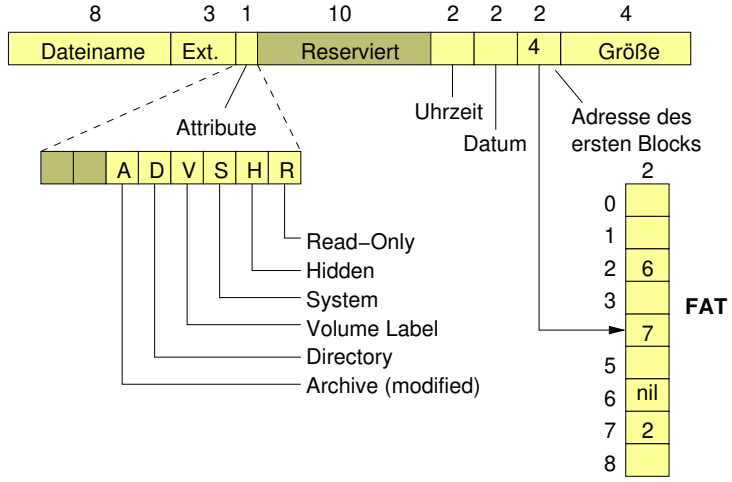
- bei kontinuierlicher Allokation: die Plattenadresse der gesamten Datei oder des Unterverzeichnisses.
- bei Allokation mit verketteter Liste mit und ohne Index: die Plattenadresse des ersten Blocks der Datei oder des Unterverzeichnisses.
- bei Allokation mit Index Nodes: die Nummer des Inodes der Datei oder des Unterverzeichnisses.

Notizen

Beispiel: MS-DOS

Hochschule RheinMain

(s.o.) Hierarchisches Verzeichnissystem, Allokation von Plattenblöcken mittels verketteter Liste und Index.
Verzeichniseintrag:



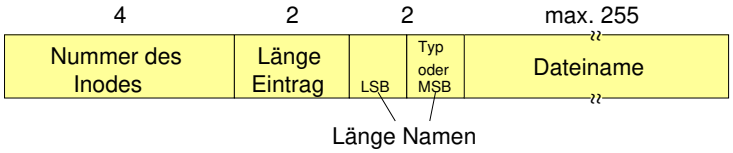
Notizen

Beispiel: UNIX (1)

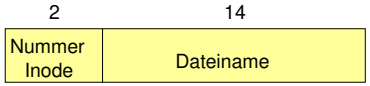
Hochschule RheinMain

(s.o.) Hierarchisches Verzeichnissystem, Allokation von Plattenblöcken mittels Index Nodes (inodes).

Verzeichniseintrag Linux (ext2:)



Verzeichniseintrag klassisches UNIX System V (s5):

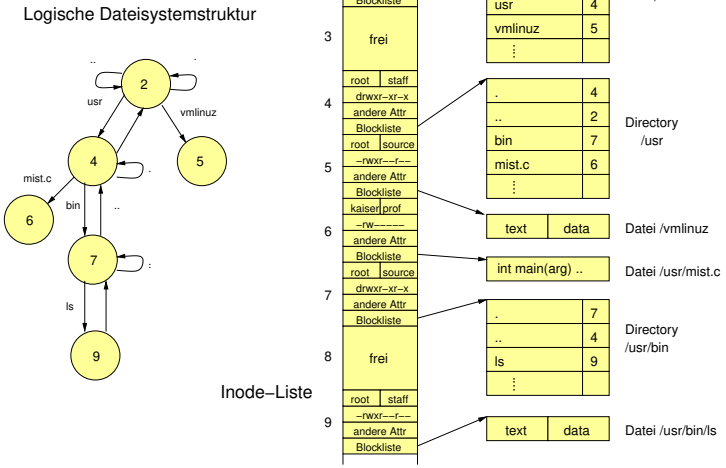


Notizen

Beispiel: UNIX (2)

Hochschule RheinMain

Prinzip der Umsetzung eines Pfadnamens



Notizen

Dateisystemstruktur

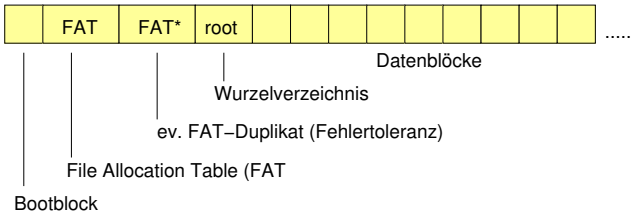
Hochschule RheinMain

Die Struktur eines Dateisystems wird beim Erzeugen auf die Blockmenge eines logischen Laufwerks (Partition) aufgeprägt.

Dienstprogramme zum Erzeugen:

- MS-DIS: `format`[5]
- UNIX: `mkfs`, (`newfs`)

Beispiel: MS-DOS



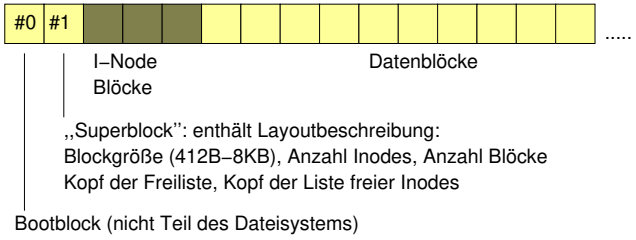
[5] nicht zu verwechseln mit dem Formatieren eines Mediums, in MS-DOS low-level Formatierung genannt

Notizen

Dateisystemstruktur (2)

Hochschule RheinMain

Beispiel: Klassisches UNIX System V (s5)



Notizen

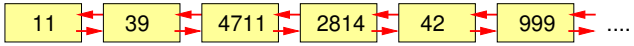
Verwaltung freier Blöcke

Hochschule RheinMain

Angewendete Methoden:

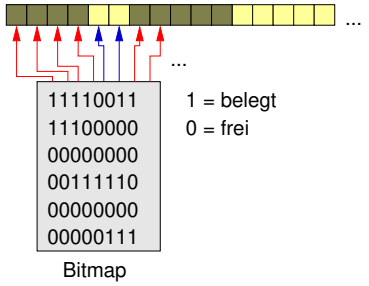
Verkettete Liste.

- Vorteil: Größere freie Bereiche einfacher erkennbar
- Beispiele: MS-DOS FAT, UNIX System V



Bitmap

- Vorteil: Größere freie Bereiche einfacher erkennbar.
- Beispiel: Linux ext2



Notizen

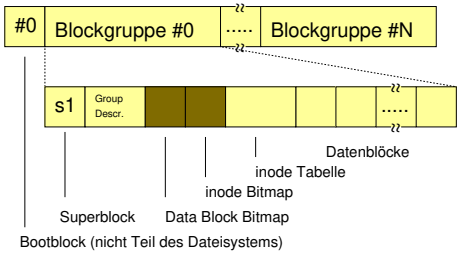
Dateisystemstruktur (3)

Hochschule RheinMain

Beispiel: Linux ext2: Besonderheiten

Einführung sogenannter **Bockgruppen**, d.h. Mengen von aufeinander folgenden Blöcken innerhalb eines Dateisystems mit jeweils eigenen Verwaltungsstrukturen.

I-Node und zugehörige Datenblöcke sollen möglichst dicht beisammen bleiben (Performance).



Redundante Kopien s1, .. des Superblocks in jeder Blockgruppe an verschiedenen Stellen (Kopfpositionen) zur Verbesserung der Verfügbarkeit der Layoutinformation auch bei Plattenfehlern.

Notizen

Dateisystemstruktur (4)

Hochschule RheinMain

Weitere Ansätze (hier nicht im Detail besprochen)

Log-basierte Dateisysteme:

- Für Halbleiter-Speichermedien konzipiert (keine seek-Zeiten)
- Schreibaufträge puffern, in regelmäßigen Zeitabständen als ganzes Segment schreiben.
- Position der inodes über in-Memory Map verwalten.
- Platte wird als „Ringpuffer“ betrieben.
- „Cleaner“ Thread gibt regelmäßig unbenutzte Blöcke frei.

Journaling-Dateisysteme:

- Erst: Geplante Operationen (idempotent) in Log schreiben
- Dann: Operationen ausführen
- Bei Absturz: Geplante, aber nicht mehr zur Ausführung gekommene Operationen nachholen.

Notizen

Quotas (Plattenplatz-Kontingentierung)

Hochschule RheinMain

Ziel:
Vermeidung der Monopolisierung von Plattenplatz durch einzelne Benutzer in Mehrbenutzersystemen.

Systemadministrator kann jedem Benutzer **Schranken** (Quotas) zuordnen für

- maximale Anzahl von eigenen **Dateien** und
- Anzahl der von diesen benutzten **Plattenblöcke**

Betriebssystem stellt sicher, dass diese Schranken nicht überschritten werden.

Beispiel: UNIX Quotas

Je Benutzer und Dateisystem werden verwaltet:

- Weiche Schranke (Soft Limit) für die Anzahl der benutzten Blöcke (kurzfristige Überschreitung möglich).
- Harte Schranke (Hard Limit) für die Anzahl der benutzten Blöcke (kann nicht überschritten werden).
- Anzahl der aktuell insgesamt zugeordneten Blöcke.
- Restanzahl von Warnungen. Diese werden bei Überschreitung des Soft Limits beim Login beschränkt oft wiederholt, danach ist kein Login mehr möglich.
- Gleiche Information für die Anzahl der benutzten Dateien (Inodes).

Notizen

Wahl der Blockgröße

Hochschule RheinMain

Fast alle Dateisysteme bilden Dateien aus Blöcken fester Länge
(Block umfasst zusammenhängende Folge von Sektoren.)
Problem: Welches ist die optimale Blockgröße?

Kandidaten aufgrund der Plattenorganisation sind:

- Sektor (512 B - 4KB)
- Spur (z.B. 256 KB)

Untersuchungen an Dateisystemen in UNIX-Systemen zeigen:

- Die meisten Dateien sind klein ($<$ 10 KB) aber mit wachsender Tendenz.
- In Hochschul-Umgebung im Mittel 1 KB [Tanenbaum]

Eine große Allokationseinheit (z.B. Spur) verschwendet daher zuviel Platz.

Beispielrechnung: Verschwendeter Platz

(Daten basieren auf realen Dateien, Quelle [Leffler et al].)

Gesamt [MB]	Overhead [%]	Organisation
775.2	0.0	nur Daten, byte-variabel lange Segmente
807.8	4.2	nur Daten, Blockgröße 512 B, int. Fragmentierung
828.7	6.9	Daten und Inodes, UNIX System V, Blockgröße 512 B
866.5	11.8	Daten und Inodes, UNIX System V, Blockgröße 1 KB
948.5	22.4	Daten und Inodes, UNIX System V, Blockgröße 2 KB
1128.3	45.6	Daten und Inodes, UNIX System V, Blockgröße 4 KB

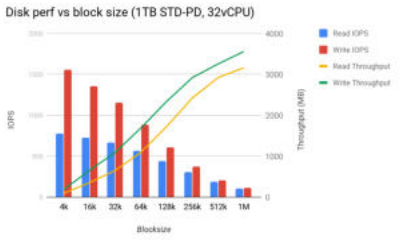
Notizen

Wahl der Blockgröße (2)

Hochschule RheinMain

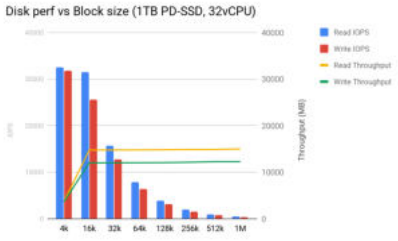
Eine kleine Allokationseinheit (z.B. Sektor) führt zu schlechter zeitlicher Performance (viele Blöcke = viele Kopfbewegungen).

Mechanische Festplatte



https://medium.com/@duhroach/the-impact-of-blocksize-on-persistent-disk-performance-7e50a85b2647

zum Vergleich: SSD
N.B.: andere Skalierung



https://medium.com/@duhroach/the-impact-of-blocksize-on-persistent-disk-performance-7e50a85b2647

Notizen

Wahl der Blockgröße (3)

Hochschule RheinMain

Kompromiss:

Wahl einer mittleren Blockgröße, z.B. 4 KB oder 8 KB.
Bei Sektorgröße 512 B entspricht ein Block von 4 KB Größe dann 8 aufeinanderfolgenden Sektoren.
Für Lesen oder Schreiben eines Blockes wird die entsprechende Folge von Sektoren als Einheit gelesen oder geschrieben.
Ab ca. 2010 für PC-Systeme und Notebooks vermehrt Festplatten mit 4 KB-Sektorgröße (*Advanced Format*)

Ca. 9% Kapazitätsgewinn (da weniger Gaps)
Kompatibilitätsprobleme (Lösung durch Emulation) können Performance-Nachteile haben

Notizen

Wahl der Blockgröße (3)

Hochschule RheinMain

Tricks ...

Beispiel: BSD UNIX Fast File System (heute ähnlich auch bei ext2)

Einführung **zweier** Blockgrößen, genannt **Block** und **Fragment** als Teil eines Blocks (typ. heute 8 KB / 1 KB).

Eine Datei besteht aus ganzen Blöcken (falls nötig) sowie ein oder mehreren Fragmenten am Ende der Datei.

→ Transfer großer Dateien wird effizient.

→ Speicherplatz für kleine Dateien wird gut genutzt.
Empirisch wurde ein ähnlicher Overhead für ein 4KB/1KB BSD File System beobachtet wie für das 1 KB System V File System.

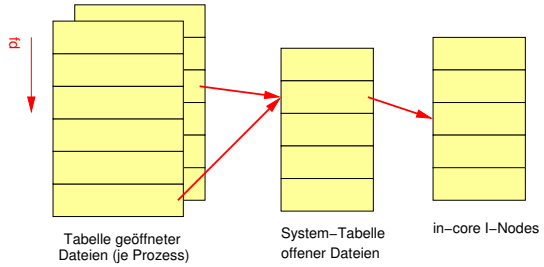
Notizen

Repräsentierung im Hauptspeicher

Hochschule RheinMain

Bisher wurde die Repräsentierung von Dateien und Verzeichnissen auf dem Hintergrundspeicher betrachtet.

Geöffnete Dateien besitzen eine Repräsentierung im Arbeitsspeicher



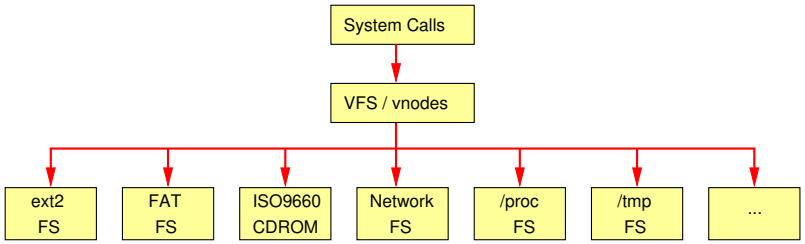
N.B.: Die System-Tabelle enthält insbesondere die Datei-Position jeder geöffneten Datei. Da geöffnete Dateien bei fork() vererbt werden, ist die einmalige Verarbeitung des Dateiinhalts durch Eltern- und Kindprozess oder durch mehrere Kindprozesse möglich. Erneutes Öffnen derselben Datei resultiert dagegen in einem neuen Eintrag mit unabhängigem Positionszeiger.

Notizen

Virtuelles Dateisystem

Hochschule RheinMain

Innerhalb des BS-Kerns wurde neue Schnittstelle des Dateisystems eingezogen, das sogenannte vnode-Interface (virtual inode) des VFS (Virtual File System).
Das vnode-Interface umfasst generische Operationen zum Umgang mit Dateien und Verzeichnissen bzw. Dateisystemen als ganzes.
Die virtuelle Schnittstelle wird für jeden Dateisystemtyp implementiert.
Das Interface ist auch auf Pseudo-Dateisysteme anwendbar, wie etwa das `/proc`-Prozessdateisystem (jedem aktiven Adressraum entspricht eine Datei) oder RAM-Disk.



Notizen

Zuverlässigkeit des Dateisystems

Hochschule RheinMain

Hauptgesichtspunkte:

Behandlung fehlerhafter Blöcke.

Dateisystemkonsistenz.

Erzeugen und Verwalten von Backups.

Notizen

Behandlung fehlerhafter Blöcke

Hochschule RheinMain

Sowohl Festplatten als auch Flash-Speicher haben i.d.R. von Anfang an **fehlerhafte Blöcke** (z.B. aufgrund ungleichmäßiger Magnetisierung der Oberflächen, Toleranzen in der Chip-Fertigung, etc).
Bei Festplatten werden fehlerhafte Sektoren beim Formatieren des Mediums (Aufbau der Sektoren) festgestellt. (Im PC-Umfeld wird das Formatieren auch low-level-Formatierung genannt).
Bei Flash-Speichern kommen aufgrund des Verschleißes im lfd. Betrieb weitere defekte Blöcke hinzu.
Verzeichnis der fehlerhaften Sektoren wird **Media Defect List** genannt.
Hardware-Lösung (heute üblich):

- Media Defect List wird vom Gerätecontroller selbst geführt.
- Jedem defekten Block wird ein Ersatzblock (i.d.R. aus einem dafür reservierten Bereich) zugeordnet, der statt des defekten Blocks benutzt wird.
- Bei Festplatten Problem: evtl. unvorhersehbare Kopfbewegungen.
- Bei Flash-Speichern kann der reservierte Bereich irgendwann erschöpft sein.

Software-Lösung (heute eher unüblich):

- Es wird eine Datei konstruiert, die nie gelesen oder geschrieben wird und der alle defekten Blöcke zugeordnet werden.

Notizen

Konsistenz des Dateisystems

Hochschule RheinMain

Definition: Dateisystem-Konsistenz

Konsistenz eines Dateisystems meint Korrektheit der inneren Struktur des Dateisystems.

d.h. aller mit der Aufprägung der Dateisystemstruktur auf die Blockmenge verbundenen Informationen (Meta-Information des Dateisystems, UNIX: z.B. Superblock, Freiliste oder Bitmap).

Beispiel einer Konsistenzregel:

- Jeder Block ist entweder Bestandteil genau einer Datei oder eines Verzeichnisses, oder er ist genau einmal als freier Block bekannt.

Verletzung der Konsistenz:

- I.d.R. durch Systemzusammenbruch (z.B. aufgrund eines Stromausfalls) vor Abspeicherung aller modifizierten Blöcke eines Dateisystems.

Überprüfung der Konsistenz:

- Betriebssysteme besitzen Hilfsprogramme zur Überprüfung und evtl. Wiederherstellung der Konsistenz bei eventuell auftretendem Datenverlust.

Notizen

Beispiel: UNIX

Hochschule RheinMain

UNIX war traditionell schwach in Bezug auf Sicherstellung der Konsistenz von Dateisystemen:

Dateisystem (z.B. ufs) wird **nicht in atomaren Schritten** von einem konsistenten Zustand in einen neuen konsistenten Zustand überführt. Eine solche Veränderung verlangt i.d.R. mehrere Schreibzugriffe.

Modifizierte Datenblöcke bleiben im Pufferspeicher (Block Buffer Cache) und werden durch einen Dämonprozess spätestens nach 30 sec zurückgeschrieben.

Modifizierte Blöcke mit Meta-Informationen werden zur Verringerung der Gefahr der Inkonsistenz sofort zurückgeschrieben.

System Call `sync` existiert zur Einleitung eines sofortigen Zurückschreibens aller veränderten Blöcke (*forced write*).

Notizen

Beispiel: UNIX (2)

Hochschule RheinMain

Durch Einführung von Journaling-Dateisystemen[6] hat sich die Situation deutlich gebessert.

- Meta-Informationen des Dateisystems werden vorab per Write-Ahead-Logging (analog Datenbanken) gespeichert.
- Konsistenz ist gewährleistet, z.B. bei Systemzusammenbruch.
- Dennoch können plötzliche Stromausfälle während laufender physischer Schreibvorgänge Schäden verursachen.
- Gilt für Festplatten wie für Flash-Speicher.
- Deshalb: Medien vor dem Entfernen immer erst „unmounten“.

Zur Verbesserung der Verfügbarkeit von Daten im Falle von Plattenfehlern werden z.B. eingesetzt:

- Spiegelplattenbetrieb (RAID1 - Disk mirroring, vgl. 10.3.2)
- Paritätsinformation speichern (RAID5, vgl. 10.3.2)

[6] `ext3`, `ext4`, ReiserFS, Btrfs, XFS, ..., vgl 10.3.5

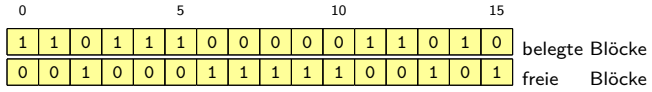
Notizen

Überprüfung und Reparatur

Hochschule RheinMain

Dienstprogramme zur Überprüfung / Reparatur der Konsistenz eines **nicht in Benutzung** befindlichen Dateisystems **bei möglichem Datenverlust**: `fsck`, z.T. auch `ncheck`
`fsck` (*file system check*) führt Konsistenzüberprüfungen durch:
Blocküberprüfung:

- zwei Tabellen mit jeweils einem Zähler je Block
- anfangs alle Zähler mit 0 initialisiert



- erste Tabelle: wie oft tritt jeder **Block in einer Datei** auf?
  - ★ alle inodes lesen.
  - ★ für jeden verwendeten Block Zähler in erster Tabelle inkrementieren.
- zweite Tabelle: **freie Blöcke**
  - ★ Für Blöcke in der Liste / Bitmap der freien Blöcke Zähler in zweiter Tabelle inkrementieren.

Konsistenz: Für jeden Block muss der Zählerstand aus Tab 1 und Tab 2 zusammen „1“ sein.

Notizen

Überprüfung und Reparatur (2)

Hochschule RheinMain

Fehlender Block: Block 4 ist weder belegt noch frei?

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	1	0	1	0	1	0	0	0	0	0	1	1	0	1	0	belegte Blöcke
0	0	1	0	0	0	1	1	1	1	1	0	0	1	0	1	freie Blöcke

→ Maßnahme: Block zu freien Blöcken hinzunehmen

Doppelter Block in Freiliste (Block 8)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	1	0	1	1	1	0	0	0	0	0	1	1	0	1	0	belegte Blöcke
0	0	1	0	0	0	1	1	2	1	1	0	0	1	0	1	freie Blöcke

→ Maßnahme: Freiliste neu aufbauen

Doppelter belegter Block (Block 11)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	1	0	1	1	1	0	0	0	0	0	2	1	0	1	0	belegte Blöcke
0	0	1	0	0	0	1	1	1	1	1	0	0	1	0	1	freie Blöcke

→ Maßnahme: Block kopieren, Kopie-Block in eine der beiden betroffenen Dateien statt Block 11 einbauen

Notizen

Überprüfung und Reparatur (3)

Hochschule RheinMain

Darüber hinaus **Überprüfung der Verzeichniseinträge**:

- Vergleiche Anzahl aller Verzeichniseinträge mit Verweis auf einen inode mit dem darin gespeicherten Referenzzähler.

→ Maßnahme: ggf. inode-Referenzzähler der durch Zählung festgestellten Zahl von Referenzen anpassen.
(N.B.: Es kann „beliebig viele“ (> 0) Referenzen auf einen inode geben (aufgrund harter Links!))

Problem: Für große Platten kann ein `fsck`-Lauf sehr lange (Stunden!) dauern. In dieser Zeit ist das System möglicherweise nicht verfügbar (Kosten!)

Notizen

Performance des Dateisystems

Hochschule RheinMain

Maßnahmen zur Performance-Steigerung:

Blockgruppen:

- Ziel: Vermeiden von weiten Kopfbewegungen.
- Beispiel: Linux ext2 File System (vgl. 10.3.5).
- Kein Effekt (auch kein negativer) bei Flash-Speicher

Block Buffer Cache.

- Ziel: Reduzierung der Anzahl der Plattenzugriffe.
- Ein Teil des Arbeitsspeichers, **(Block) Buffer Cache** genannt, wird als Cache für Dateisystemblöcke organisiert.
- Typische Hitrate: 85%
- Modifizierter LRU-Algorithmus zur Auswahl zu verdrängender Blöcke unter Berücksichtigung der Forderungen zur Verringerung der Gefahr von Inkonsistenz
- Blöcke verbleiben im Cache, auch wenn die entsprechenden Dateien geschlossen sind.

Notizen

Performance des Dateisystems (2)

Hochschule RheinMain

Dateinamens-Cache:

- Ziel: Verringerung der Anzahl der Schritte bei der Abbildung von Datei-Pfadnamen auf Blockadressen.
- Beispiel: BSD UNIX namei-Cache:
  - `namei()`-Routine
    zur Umsetzung von Pfadnamen auf inode-Nummern.
    benötigte vor Einführung von Caching ca. 25% der Zeit eines Prozesses im BS-Kern, auf <10% gesenkt.
  - Cache der *n* letzten übersetzten Teilnamen:
    Typische Hitrate: 70-80%.
    Höchste Bedeutung für Gesamtperformance.
  - Cache des letzten benutzten Directory-Offsets:
    Wenn ein Name im selben Verzeichnis gesucht wird, beginnt die Suche am gespeicherten offset und nicht am Anfang des Verzeichnisses (sequentielles Lesen eines Verzeichnisses ist häufig, z.B. durch das ls-Kommando).
    Typische Hitrate: 5-15 %.
  - Gesamthitrate von ca. 85% wird erreicht.

Notizen