Dr. Ulrich Margull
Kirchweg 6 * 93055 Regensburg
Tel. (+49) 941 78956-8 Fax –9
www.margull.de

9 Dateisysteme

Permanenter Speicher (Secondary Storage) ist definiert durch:
- Die Daten sind über das Prozessende hinaus verfügbar
- Daten können gelesen, geändert und neu geschrieben werden
- Es können je nach Anforderung beliebige Daten von verschiedenen Stellen des Mediums gelesen werden (Random Access).

Typischerweiser permanenter Speicher ist die Festplatte, aber auch Bandlaufwerke, oder NVRAM (non-volatile RAM, e.g. Flash oder EEPROM).

Das Betriebssystem verwendet *Dateisysteme:*
- um die Daten in Dateien zu speichern
- um die Dateien wiederzufinden
- und um die Daten wieder aus den Dateien zu lesen.

Ein Dateisystem hat dabei zwei Seiten:
- Zum einen eine Schnittstelle zum Benutzer. Diese definiert, wie Dateien benannt werden, welche Attribute es gibt, die Operationen, die mit Dateien erlaubt sind, und wie die Dateien angeordnet sind (Verzeichnisstruktur). Die Benutzerschnittstelle soll einfach sein, jedoch sollen alle notwendigen Operationen enthalten.
- Zum anderen eine Implementierung, die die Dateien auf das physikalische Medium abbildet. Sie muss entsprechende Datenstrukturen verwalten und Algorithmen bereitstellen, um effiziente Zugriffe zu ermöglichen.

Dateisysteme sind in der Regel block-orientiert: die Daten werden in Blöcken gespeichert. Typische Blockgrößen sind z.B. 512 Bytes, 1 KB, 4KB oder 8KB.

9.1 Dateien

9.1.1 Dateinamen

- MSDOS 8.3-Notation:
  - Dateiname 8 Zeichen
  - Dateityp (Extension) 3 Zeichen
  - Case-Insensitive: MeinText.txt, MEINTEXT.TXT und meintext.txt bezeichnen alle dieselbe Datei
  - ASCII Zeichensatz, nicht alle Zeichen erlaubt
- Andere Dateisysteme:
  - Längere Dateinamen möglich, z.B. 255 oder 65535 Zeichen
  - Andere Zeichensätze, z.B. Windows NT/2K/XP: Unicode
- Dateityp wird durch die Dateiendung definiert (MSDOS/Windows) oder durch eine *magic number*, d.h. durch eine Kennung am Anfang der Datei. Anhand des Dateityps kann das Betriebssystem verschiedene Unterscheidungen treffen, z.B. ob die Datei ausführbar ist (.exe. oder .txt?), oder welches Programm zum Öffnen verwendet werden kann

Neben regulären Dateien verwendet das Betriebssystem Verzeichnisdateien, in denen die Verzeichnisstruktur gespeichert wird. Weiterhin gibt es so spezielle Dateien, über die z.B. in

Dr. Ulrich Margull
Kirchweg 6 * 93055 Regensburg
Tel. (+49) 941 78956-8 Fax –9
www.margull.de

Unix Geräte eingebunden werden (Verzeichnis /dev). Unter Windows entspricht dies den speziellen Laufwerken "COM1:", "LPT3:" usw.

9.1.2 Dateistruktur und -zugriff

Eine Datei kann verschieden strukturiert werden:
1. Ein Feld von einzelnen Bytes
2. Ein Feld von einzelnen Blöcken
3. Baumartig nach einem Schlüssel (Key)
4. Bestehend aus mehreren Streams, d.h. mehreren Dateien (z.B. MacOS: Programm und Resourcen-Stream)

V Bild

Einfachste Art des Zugriffs ist sequentiell, d.h. die Datei kann nur von Anfang bis Ende Byte-für-Byte (oder Block-für-Block) gelesen werden. Freier Zugriff ist mittels "random access" möglich, wo das Programm an beliebiger Stelle positionieren und einlesen kann.

9.1.3 Datei-Attribute

Zu jeder Datei sind eine Reihe von Attributen möglich:

Tabelle 6: Mögliche Dateiattribute

Attribut	*Bedeutung*
Zugriffsrechte	Wer darf wie auf die Datei zugreifen? Z.B. Lese-, Schreib-, Ausführungsrechte. Hier sind sehr komplexe Rechtesysteme möglich
Erzeuger	Wer hat die Datei angelegt
Besitzer	Wem gehört die Datei?
Read-Only Flag	Nur lesender Zugriff möglich
Hidden Flag	Die Datei ist versteckt und wird im Listing nicht angezeigt
System Flag	Es handelt sich um eine System-Datei
Archiv Flag	Die Datei wurde seit dem letzten Archivieren modifiziert und muss neu archiviert werden
Zeitpunkt der letzten Änderung	
Zeitpunkt des letzten Zugriffs	
Größe	Dateigröße
Maximale erlaubte Größe	

9.1.4 Dateioperationen

Typische Dateioperationen sind:

Dr. Ulrich Margull
Kirchweg 6 * 93055 Regensburg
Tel. (+49) 941 78956-8 Fax –9
www.margull.de

Operation	*Beschreibung*	*Unix*	*Win32*
Erzeugen der Datei	eine Datei wird im Dateisystem angelegt	**creat()**	**CreateFile()**
Öffnen der Datei	Beim Öffnen einer Datei muss der Name sowie der gewünschte Mode angegeben werden. Die Datei muss bereits existieren.	**open()**	**CreateFile()**[6]
Löschen der Datei	Die Datei wird gelöscht.	**remove()**	**DeleteFile()**
Schließen der Datei	Die Datei wird geschlossen, alle gecachten Daten werden auf Festplatte geschrieben	**close()**	**CloseHandle()**
Lesen aus der Datei	Es wird eine bestimmte Anzahl von Zeichen (oder Blöcken) aus der Datei gelesen	**read()**	**ReadFile()**
Schreiben in eine Datei	Es wird eine bestimmte Anzahl von Zeichen (oder Blöcken) aus der Datei geschrieben	**write()**	**WriteFile()**
Positionieren in der Datei	Der Dateizeiger wird an eine Stelle in der Datei posititioniert (random access)	**lseek()**	**SetFilePointer()**
Lesen der Attribute, Ändern der Attribute		**stat(), chmod(), chown(), readlink()**	**GetFileAttributes(), SetFileAttributes()**
Umbenennen	Umbenennen einer Datei; oftmals auch verschieben einer Datei	**rename()**	**MoveFile()**

Eine vollständige Liste aller Win32-Dateifunktionen findet sich weiter unten.

9.1.5 Memory-Mapped Files

Zugriff auf eine Datei über die Zugriffsoperationen ist umständlich und aufwändig. Eigentliches Ziel ist es, die Daten aus der Datei in den Hauptspeicher zu transportieren, um sie dort zu bearbeiten. In modernen Betriebssystemen gibt es bereits einen Mechanismus, der sehr effektiv Dateien von und zum Hauptspeicher bringen kann: das virtuelle Speichermanagement.
Bei Memory-Mapped Files wird das Speichermanagement angewiesen, bestimmte Seiten im Hauptspeicher auf die Datei abzubilden. Alle Änderungen in diesen Seiten werden wieder in die Datei zurückgeschrieben.
Dabei gelten folgende Eigenschaften:

- Große Dateien (z.B. 6 GB Video) können nicht komplett in den Speicher eingeblendet werden. Dann müssen jeweils Teile davon eingeblendet werden.

[6]Die Win32 Funktion OpenFile() gibt es nur aus Kompatibilitätsgründen und sollte nicht verwendet werden.

Dr. Ulrich Margull
Kirchweg 6 * 93055 Regensburg
Tel. (+49) 941 78956-8 Fax –9
www.margull.de

- Oftmals können nicht beliebige Stücke eingeblendet werden, sondern nur vielfache der Seitengröße. Unter Win32 z.B. können nur 64kB-Blöcke bearbeitet werden.
- Ein Problem ist das Dateiende: Da das virtuelle Speichermanagement immer nur ganze Seiten betrachtet, weiss es nicht, wieviele Bytes in der letzten Seite zur Datei gehören. Deshalb muss das Programm typischerweise die Dateigröße angeben; diese kann auch bei geöffneten Dateien geändert werden.

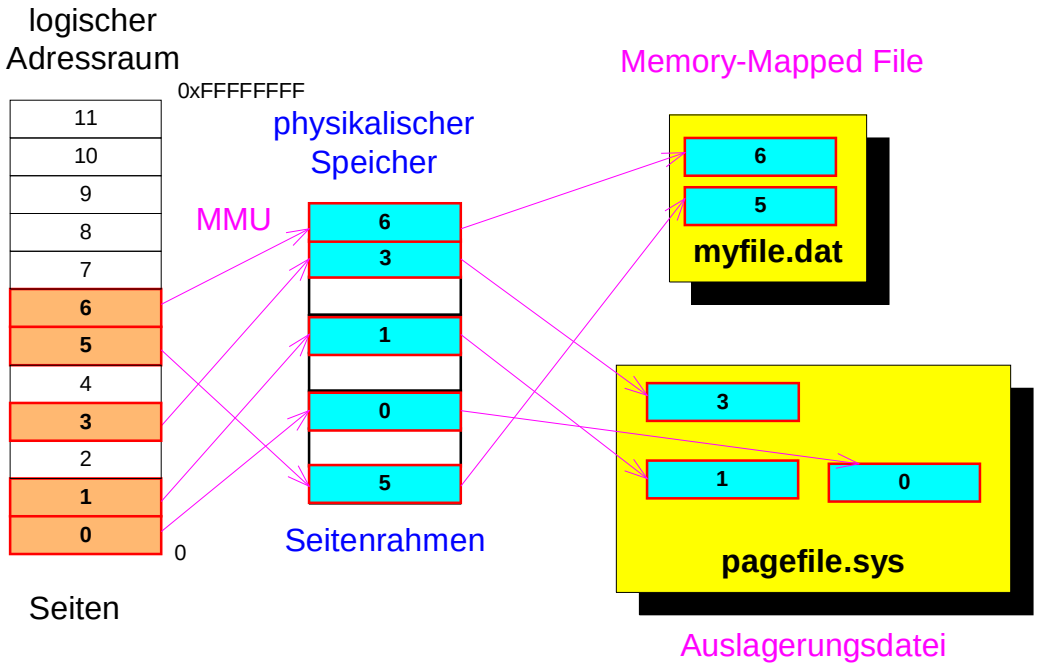


Abbildung 99: Virtueller Speicher und Memory-Mapped Files

9.2 Verzeichnisse

9.2.1 Verzeichnisse mit einer Ebene

Alle Dateien befinden sich in einem Verzeichnis, dem sog. **root**-Verzeichnis.
Beispiel: CPM-Betriebssystem, erste MS-DOS Version

9.2.2 Verzeichnisse mit zwei Ebenen

Im root-Verzeichnis befinden sich eine Reihe von Unterverzeichnissen, in denen die eigentlichen Dateien liegen.

9.2.3 Hierarchische Verzeichnisstrukturen

Es sind beliebige Verzeichnisstrukturen möglich. Jedes Verzeichnis kann beliebig viele Dateien und Unterverzeichnisse besitzen (Baumförmige Struktur).

9.2.4 Dateipfade

Um Dateien in einem Verzeichnisbaum zu finden, muss der gesamte Pfad (beginnend vom root-Verzeichnis) angegeben werden. Alternativ können auch relative Pfade angegeben

Dr. Ulrich Margull
Kirchweg 6 * 93055 Regensburg
Tel. (+49) 941 78956-8 Fax –9
www.margull.de

werden. Zusätzlich hat jedes Verzeichnis die zwei Spezialverzeichnisse "." (das Verzeichnis selbst) und ".." (das übergeordnete Verzeichnis).

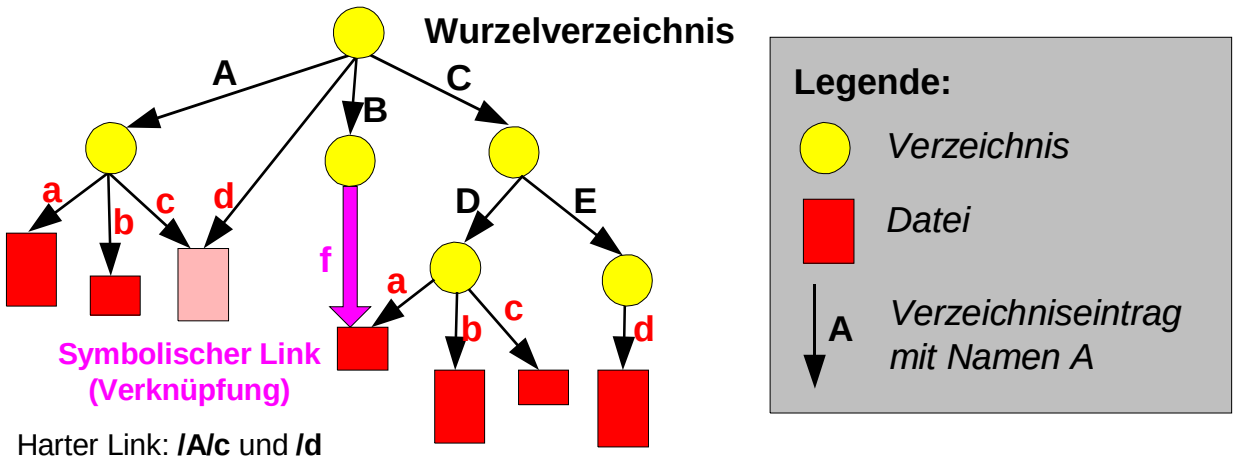
9.2.5 Verzeichnis-Operationen

Typische Operationen im Verzeichnisbaum sind:
- Erzeugen eines Verzeichnisses
- Löschen eines Verzeichnisses
- Verzeichnis lesen (Öffnen, Lesen, Schließen)
- Umbenennen
- Verlinken. Mit einem Link kann innerhalb des Dateisystems eine Datei mehrere Namen besitzen.

9.2.6 Links

In Unix-Dateisystemen können Links gesetzt werden. Durch ein Link wird ein Dateisystem-Objekt (Datei oder Verzeichnis) über einen anderen Pfad und Namen angesprochen; das Objekt hat dann zwei Namen im Dateisystem.
Dabei gibt es zwei Arten:
- Symbolische Links entsprechen den Verknüpfungen unter Windows. Ein symbolischer Link deutet jeweils auf die entsprechende Zieldatei oder das Zielverzeichnis. Wird das Ziel gelöscht, so zeigt der Link ins Leere.
  Symbolische Links sind manchmal verwirrend.
- Harte Links sind eine Eigenschaft des Dateisystems: damit können mehrere Namen und Pfade für eine Datei vergeben werden. Bei harten Links gibt es keine Unterscheidung zwischen der Datei (Originaldatei) und dem Link: der harte Link ist ein weiterer Name für die Datei und gleichbedeutend mit dem Orignalnamen.
  Beispiel: Legt man im Verzeichnis test1 eine Datei x.txt an, und zusätzlich ein harten Link y.txt, der auf x.txt zeigt, so kann die Datei sowohl unter ./test1/x.txt als auch unter ./y.txt angesprochen werden. Wird einer der beiden Einträge entfernt, so bleibt die Datei bestehen und ist immer noch unter dem anderen zu erreichen. Erst wenn der letzte Eintrag gelöscht wird, werden auch die Daten gelöscht.
  Harte Links sind oft verwirrend.

Dr. Ulrich Margull
Kirchweg 6 * 93055 Regensburg
Tel. (+49) 941 78956-8 Fax –9
www.margull.de



Harter Link: **/A/c** und **/d** bezeichnen dieselbe Datei

Abbildung 100: Links

9.2.7 Mount-Punkte

Unter Windows hat jede Partition, jedes Netzlaufwerk und jedes Dateisystem einen eigenen Laufwerksbuchstaben.
In der Unix-Welt gibt es keine Laufwerksbuchstaben: im Dateisystem können durch Mount-Punkte andere Dateisysteme eingefügt werden.
Mount-Punkte werden in die System-Datei *fstab* eingetragen und durch das Mount-Programm durchgeführt.

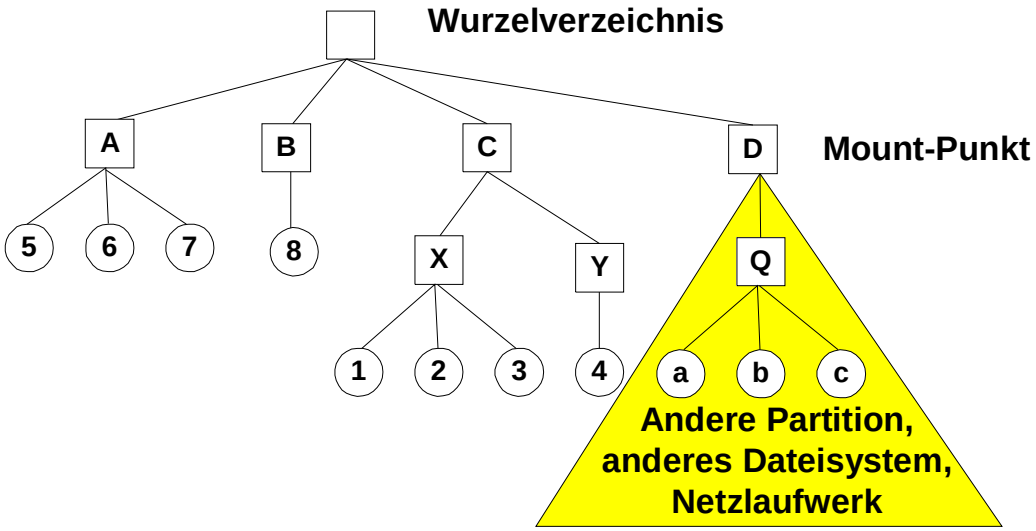


Abbildung 101: Mount-Punkt im Verzeichnis

Dr. Ulrich Margull
Kirchweg 6 * 93055 Regensburg
Tel. (+49) 941 78956-8 Fax –9
www.margull.de

9.3 Implementierung von Dateisystemen

9.3.1 Partitionierung von Festplatten

Typischerweise werden Dateisysteme auf Festplatten installiert. Eine Festplatte kann nun mehrere *Partitionen* unterstützen; in jeder Partition kann ein eigenes Dateisystem installiert werden. Der **MBR** enthält Informationen über die Partitionsverteilung auf der Festplatte.
Er befindet sich typischerweise im ersten Block der Festplatte (Block 0).
Eine Partition ist im MBR als aktiv gekennzeichnet; beim Booten des Rechners liest das BIOS den MBR aus, sucht die aktive Partition, lädt von dort den **boot block** und führt ihn aus. Der boot block der Partition enthält den Start-Code für das Betriebssystem.

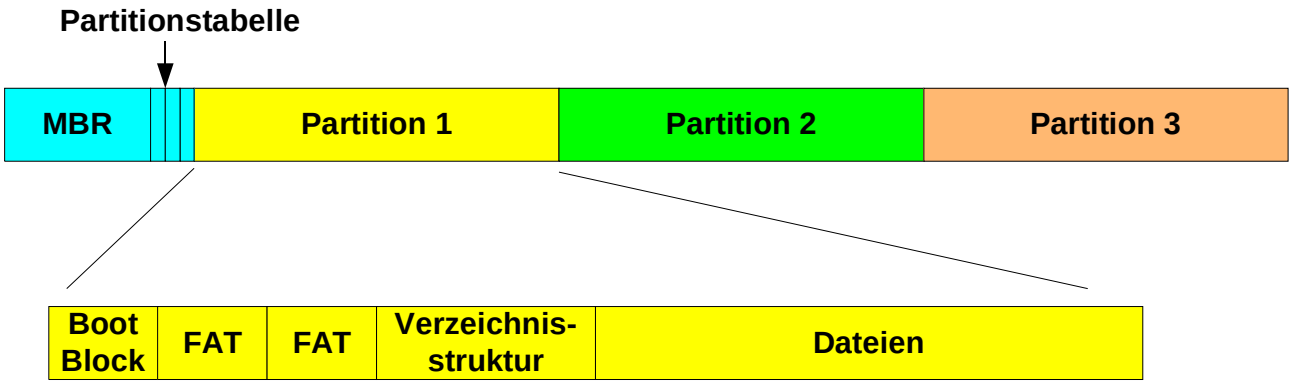


Abbildung 102: Partitionen

9.3.2 Implementierung von Dateien

Die Daten eines Programmes werden in den Dateien in Form von Blöcken gespeichert. Eine Datei besteht somit aus einer geordneten Menge vom N Blöcken $\{ B_0, B_1, \ldots, B_{N-1} \}$.
Der Zugriff erfolgt entweder sequentiell, d.h. die Blöcke werden nacheinander in ihrer Reihenfolge eingelesen, oder als wahlfreier Zugriff (random access), d.h. es werden beliebige Blöcke nacheinander gelesen.
Die Speicherung kann wie folgt geschehen:

9.3.2.1 Kontinuierliche Belegung

Jede Datei besteht aus einer Folge von Blöcken; dabei müssen alle Blöcke kontinuierlich angeordnet sein. Ist eine Datei also 10 Blöcke groß und beginnt bei Block 4711, dann belegt sie die Blöcke 4711 – 4720.
Die *Vorteile* liegen vor allem in der Einfachheit:

- Einfache Verzeichnisstruktur: die Datei ist durch Anfangsblock und Größe eindeutig beschrieben
- Zugriff auf die Datei ist einfach: beim sequentiellen Zugriff werden die Blöcke nacheinander, beginnend beim ersten Block b, gelesen. Bei wahlfreiem Zugriff (random access) auf Block i wird entsprechend der Block $b+i$ gelesen.

Die kontinuierliche Belegung hat jedoch schwerwiegende Nachteile:

- Wird eine neue Datei angelegt, so muss ein entsprechend großer Platz auf der Festplatte gefunden werden. Die Probleme sind dieselben wie beim Swapping (siehe Abschnitt 7.4):
    - Verwaltung der freien Bereiche (Bitmaps oder verlinkte Liste)

Dr. Ulrich Margull
Kirchweg 6 * 93055 Regensburg
Tel. (+49) 941 78956-8 Fax –9
www.margull.de

  - Externe Fragmentierung tritt auf; dem kann nur durch gelegentliche Kompaktierung begegnet werden, was jedoch sehr zeitaufwändig ist.
  - Passender Algorithmus für die Verteilung ist nötig (Best Fit, First Fit, Worst Fit)
- Ein schwerwiegender Nachteil besteht darin, dass bei kontinuierlicher Belegung die maximale Größe zu Beginn bekannt sein muss. Beginnt der Benutzer z.B. ein Word-Dokument zu schreiben, so könnte das System 100 Blöcke zu 512 Bytes reservieren; schreibt der Benutzer jedoch mehr als 50kB, so reicht der Platz nicht aus. Befindet sich hinter dem Bereich bereits eine neue Datei, so kann die Datei nicht vergrößert werden, sondern sie muss komplett in einen größeren freien Bereich verschoben werden. Das ist sehr aufwändig ist.

Wegen dieser Nachteile wird die kontinuierliche Belegung heutzutage nicht mehr verwendet.

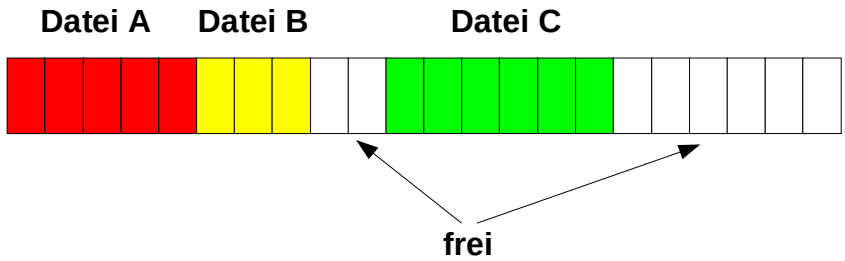


Abbildung 103: Kontinuierliche Belegung

9.3.2.2 Verlinkte Liste

Eine andere Methode besteht darin, alle Blöcke einer Datei miteinander zu verlinken. Der Verzeichniseintrag zeigt dann auf den ersten Block der Datei; dieser zeigt auf den zweiten Block, der wiederum auf den dritten, usw.

Vorteile:

- Die Datei kann beliebig wachsen: bei Bedarf kann ein beliebiger Block auf der Festplatte hinten an die Datei angehängt werden, und es müssen nur die entsprechenden Links angepasst werden.
- Einfacher sequentieller Zugriff: durch folgen der verlinkten Liste können die Blöcke nacheinander gelesen werden.

Nachteile:

- Wahlfreier Zugriff ist nicht möglich. Soll der *i*-te Block der Datei gelesen werden, so muss die verlinkte Liste vom Beginn bis zum *i*-ten Block verfolgt werden
- In jedem Block muss ein Zeiger auf den nächsten Block gespeichert werden. Beträgt die Blockgröße 512 Bytes, und werden 4 Bytes für den Zeiger benötigt, so wird insgesamt 0,78 % des verfügbaren Platzes für Zeigerinformationen benötigt.
  Cluster: jeweils mehrere Blöcke werden zu einem Cluster zusammengefasst: weniger Platz für Zeiger, jedoch höhere interne Fragmentierung.
- Sind die Blöcke einer Datei weit auf der Festplatte verstreut, wird der sequentielle Zugriff durch die vielen Kopf-Bewegungen verlangsamt.
- Zuverlässigkeit: tritt mitten in der Datei ein Fehler auf und ist der Zeiger auf den nächsten Block fehlerhaft, so ist der Rest der Datei verloren.

Dr. Ulrich Margull
Kirchweg 6 * 93055 Regensburg
Tel. (+49) 941 78956-8 Fax –9
www.margull.de

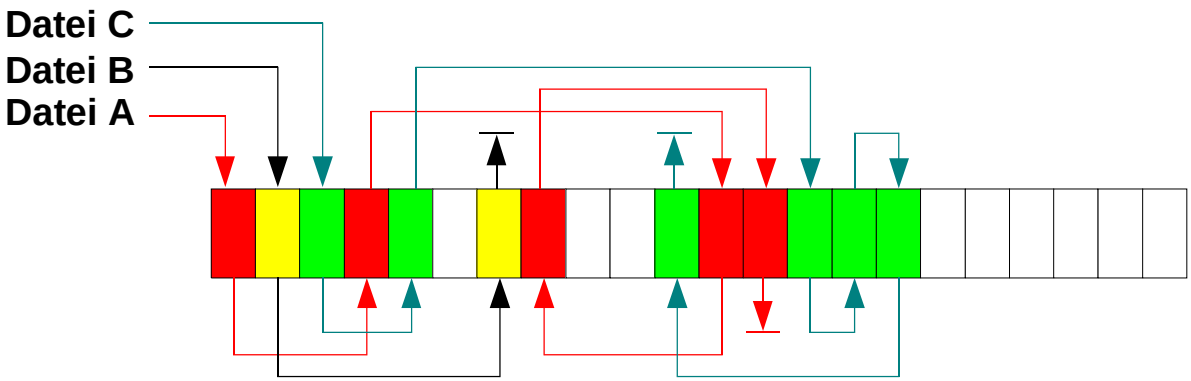


Abbildung 104: Verlinkte Liste

9.3.2.3 FAT

Die Nachteile der verlinkten Liste können durch eine File-Allocation-Table (FAT) ausgeglichen werden. Die FAT speichert dabei die Verlinkung der einzelnen Dateiblöcke. Ein Verzeichniseintrag für eine Datei enthält einen Zeiger auf den ersten Block der Datei. Mit diesem Zeiger kann aus der FAT der jeweils nächste Block ermittelt werden. Nicht benutze Blöcke werden mit 0 gekennzeichnet (siehe Graphik).
Um einen schnellen wahlfreien Zugriff zu ermöglichen, wird die gesamte FAT im Hauptspeicher gehalten. So kann beim Zugriff auf den *i*-ten Block einer Datei schnell die verlinkte Liste durchsucht und der entsprechende Block gefunden werden. Anschließend wird der gefundene Block von Festplatte geladen.

Eigenschaften:

- Dieses Verfahren wird bei MS-DOS und OS/2 verwendet.
- Die FAT wird typischerweise am Beginn der Partition gespeichert.
- Zuverlässigkeit: tritt in der FAT ein Fehler auf, so sind die betroffenen Dateien unter Umständen komplett verloren. Deshalb ist die FAT immer zweimal gespeichert; alle Änderungen werden erst in der einen, dann in der anderen durchgeführt. Jede FAT ist dabei durch eine Prüfsumme geschützt; tritt ein Fehler auf, so wird dies an der falschen Prüfsumme erkannt, und stattdessen kann die andere, korrekte FAT verwendet werden.
- Größe der FAT: die FAT muss für jeden Block einen Zeiger bereitstellen. Eine 8 GB Festplatte besitzt 16 Millionen Blöcke zu 512 Bytes; werden 4 Bytes / Zeiger benötigt, so ist die FAT 64 MB groß! Durch Cluster-Bildung kann dies reduziert werden; werden z.B. 8 kB Cluster eingesetzt, so ist die FAT nur noch 4 MB groß. Da sie jedoch im Hauptspeicher gehalten werden muss, ist das immer noch viel. Für große Festplatten ist das Verfahren also nicht besonders gut geeignet.

Dr. Ulrich Margull
Kirchweg 6 * 93055 Regensburg
Tel. (+49) 941 78956-8 Fax –9
www.margull.de

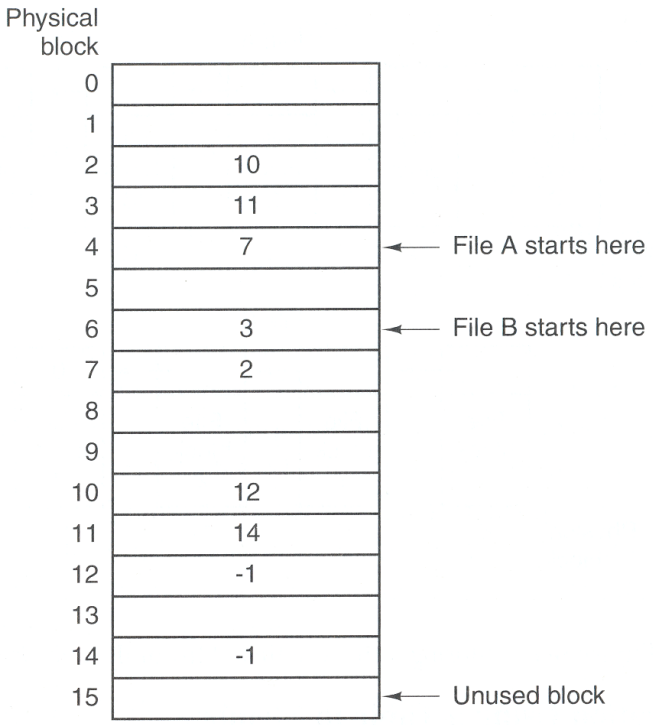


Abbildung 105: FAT (aus [1])

9.3.2.4 Indizierte Belegung (I-Nodes)

Anstelle die Zeiger jeweils in den Blöcken zu speichern, werden bei der indizierten Belegung alle Blockzeiger einer Datei zentral in einem *I-Node* (Index-Knoten) gespeichert. Wird ein bestimmter Block der Datei benötigt, so kann dieser direkt aus dem I-Node ermittelt werden. Aus Effizienzgründen ist ein I-Node oftmals genau einen Block groß. Wie können nun große Dateien, die mehr als einen I-Nodes benötigen, verwaltet werden?

- Alle I-Nodes werden in einer verlinkten Liste gespeichert. Der letzte Zeiger eines I-Nodes zeigt dann auf den nächsten I-Node der Datei.
- Alternativ können auch zwei Hierarchieebenen verwendet werden: der erste I-Node einer Datei zeigt jeweils auf Sub-I-Nodes; diese wiederum zeigen auf die Blöcke der Datei. Auch mehr als zwei Hierarchieebenen sind möglich.
- Auch Kombinationen sind möglich: Besitzt ein Verzeichniseintrag z.B. Platz für 15 Zeiger, so können diese wie folgt verwendet werden:
    - Die ersten 12 Zeiger zeigen auf direkt auf die Blöcke der Datei. Bei einer Blockgröße von 4kB können so 48 kB an Daten verwaltet werden; somit wird für kleine Dateien kein weiterer I-Node benötigt.
    - Der 13te Zeiger zeigt auf einen I-Node, der direkt die Blöcke der Datei adressiert.
    - Der 14te Zeiger zeigt auf einen doppelt-indirekten I-Node: dieser I-Node zeigt jeweils auf weitere I-Nodes, die die Blockadressen enthalten.
    - Der 15te Zeiger enthält eine dreifache Indirektion: er zeigt auf I-Nodes, die jeweils auf weitere I-Nodes zeigen, die wiederum auf weitere I-Nodes zeigen. Dadurch können auch sehr große Dateien komplett verwaltet werden.

Dr. Ulrich Margull
Kirchweg 6 * 93055 Regensburg
Tel. (+49) 941 78956-8 Fax –9
www.margull.de

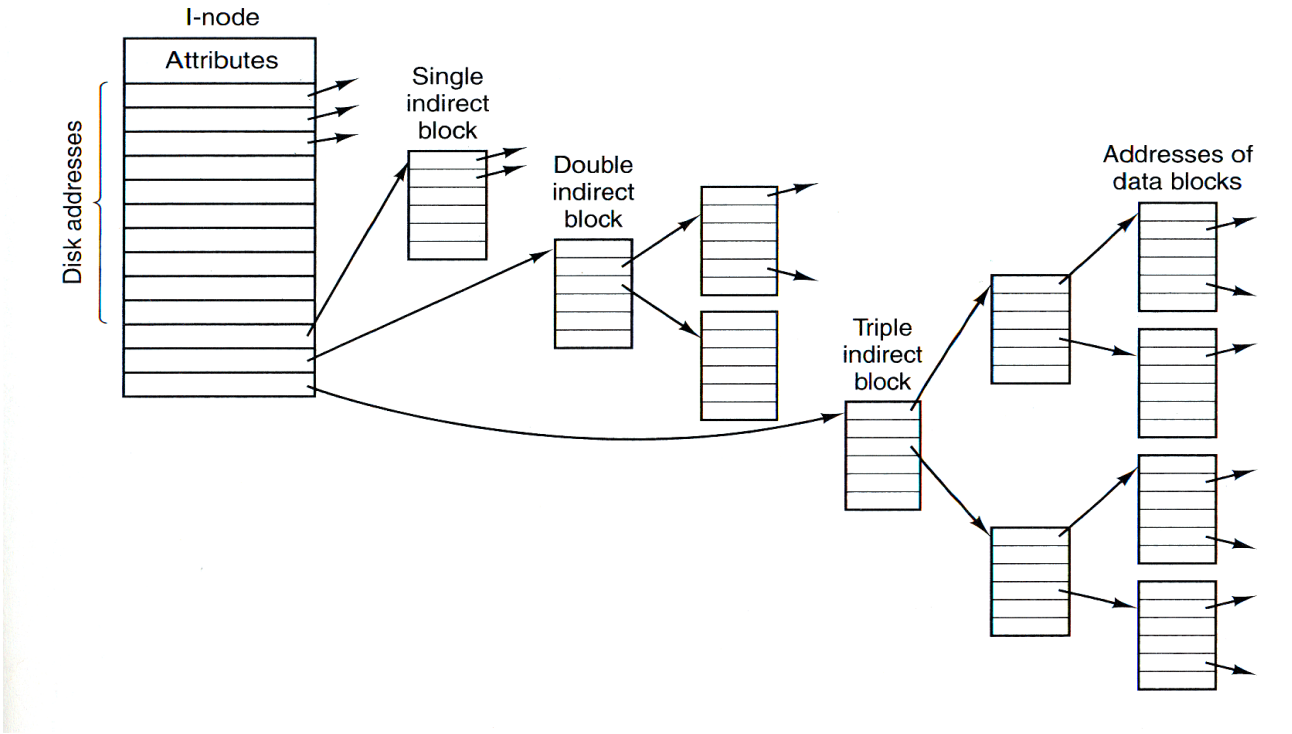
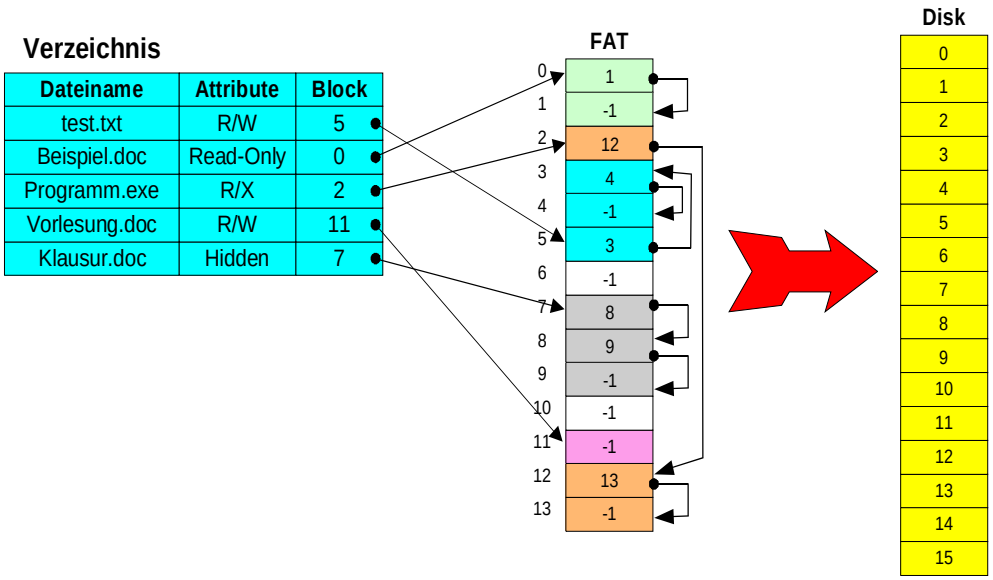


Abbildung 106: I-Node (direkt sowie einfache, zweifache und dreifache Indirektion) (aus [1])

9.3.3 Implementierung von Verzeichnissen

9.3.3.1 Lineare Liste

Das Verzeichnis wird als lineares Feld gespeichert; jeder Eintrag bezeichnet eine Datei und enthält zumindest den Dateinamen und einen Zeiger auf die Daten.



Vorteile:

- Einfache Implementierung

Dr. Ulrich Margull
Kirchweg 6 * 93055 Regensburg
Tel. (+49) 941 78956-8 Fax –9
www.margull.de

Nachteile:
- Lineare Suche in der Liste ist nötig -> schlechte Performance
- Alternativ kann das Verzeichnis immer sortiert sein, dann ist jedoch Einfügen oder Löschen von Dateien aufwändig
- Eine andere Möglichkeit sind binäre Bäume, die immer sortiert sind.

Im MS-DOS wurde ein einfaches Feld verwendet:
- max. 255 Einträge in einem Verzeichnis möglich
- Löschen einer Datei -> im Verzeichnis wird der erste Buchstabe gelöscht,

Löschen von Dateien

Beim Löschen einer Datei wird im Verzeichnis der erste Buchstabe des Dateinamens gelöscht, damit ist dieser Verzeichniseintrag frei und kann beim Anlegen einer neuen Datei wiederverwendet werden. Die Blöcke der Datei werden in die Liste der freien, verfügbaren Blöcke aufgenommen; die FAT-Einträge selbst bleiben erhalten, bis sie durch Überschreiben einer anderen Datei gelöscht werden.

Wurde eine Datei unter MS-DOS gelöscht, so kann sie unmittelbar danach wieder komplett hergestellt werden. Dazu gibt es Spezialprogramme wie z.B. undelete, die die Verzeichnisliste nach gelöschten Dateien durchsuchen (erste Buchstabe gelöscht), und eine Wiederherstellung versuchen. Wurden die Blöcke der gelöschten Datei noch nicht durch eine andere Datei belegt, so kann die Datei wiederhergestellt werden; nur der erste Buchstabe des Dateinamens muss vom Benutzer eingegeben werden. Ist jedoch einige Zeit vergangen, so ist die Wahrscheinlichkeit groß, dass Blöcke durch andere Dateien überschrieben wurden; in diesem Fall sind die Daten nicht mehr vorhanden.

Alte Festplatten stellen ein Sicherheitsrisiko dar; werden die Festplatten ohne spezielle Maßnahmen entsorgt (z.B. in den Müll geschmissen), so können die Daten meistens wieder hergestellt werden. Auch löschen oder neu formatieren reicht nicht aus.

Um Daten auf einer Festplatte sicher zu löschen, müssen spezielle Löschprogramme verwendet werden. Diese überschreiben jeden Block der Datei mehrfach (3-27 mal) mit wechselnden Folgen von Nullen und Einsen; dadurch wird zuverlässig die Magnetisierung einzelner Bits gelöscht.

Eine Alternative ist die physikalische Zerstörung der Festplatte: schreddern (in viele, kleine Teile), Plattenoberfläche stark zerkratzen oder anderweitig zerstören, einschmelzen.

9.3.3.2 Hash-Tabelle

Aus dem Dateinamen wird ein Hash-Wert gebildet, über den der Eintrag einem Feld ausgelesen werden kann.

9.3.4 Freispeicherverwaltung

Wenn eine neue Datei angelegt oder eine bestehende erweitert wird, so wird dazu freier Speicher benötigt. Der freie Speicher auf einer Festplatte muss also verwaltet werden:
- Bei Bedarf (Anlegen einer neuen Datei, Erweiterung einer bestehenden) werden freie Blöcke benötigt

Dr. Ulrich Margull
Kirchweg 6 * 93055 Regensburg
Tel. (+49) 941 78956-8 Fax –9
www.margull.de

- Gegebenfalls werden Blöcke frei (z.B. Löschen einer Datei), die anderen Dateien zur Verfügung stehen

Die verschiedenen Algorithmen dazu haben wir bereits bei der Freispeicherverwaltung des Hauptspeichers besprochen (siehe Abschnitt 7.4.4Verwaltung von freien Speicherbereichen). Die wichtigsten Verfahren sind:

- Freier Speicher in Bitmaps: Jedes Bit entspricht einem Block; eine Null bedeuted, der Block ist frei, andernfalls ist er belegt
- Verlinkte Liste des freien Blöcke

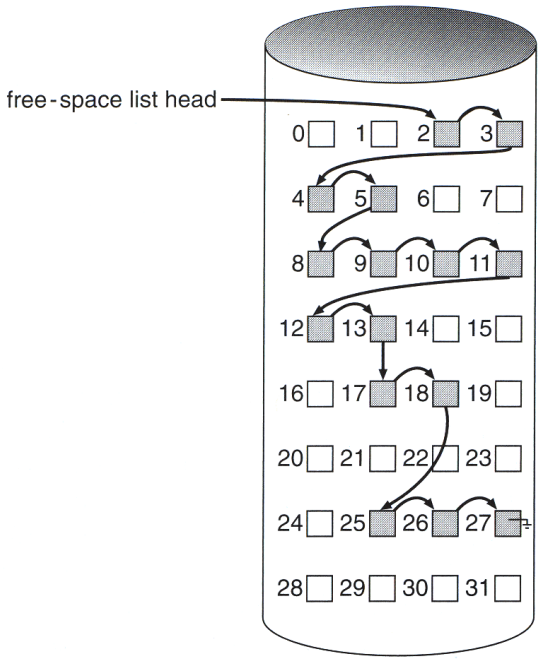


Abbildung 108: Liste freier Blöcke

9.3.5 Implementierung von Links (Verknüpfungen)

9.3.5.1 Harter Link

Harte Links sind Bestandteil des Unix-Dateisystems. Dabei können mehrere Verzeichniseinträge auf dieselbe Datei (Inode) verweisen. Zusätzlich zu den bisherigen Eigenschaften besitzt ein Inode einen Referenzzähler, der zählt, wieviele Verzeichniseinträge auf diese Datei verweisen:

- Wird ein harter Link auf eine Datei angelegt (mittels *ln* Befehl), so wird ein neuer Verzeichniseintrag angelegt, der auf denselben Inode verweist wie der Orginalname. Der Referenzzähler im Inode wird um eins hochgezählt. Für das Dateisystem sind alle Verzeichniseinträge, die auf denselben Inode verweisen, gleichberechtigt.
- Wird eine Datei gelöscht, so wird zunächst der Referenzzähler des Inodes um eins erniedrigt. Ist der Zähler größer als Null, so wird nur der Verzeichniseintrag gelöscht; die Datei selbst bleibt bestehen und ist unter anderem Namen erreichbar. Erst wenn der letzte

Dr. Ulrich Margull
Kirchweg 6 * 93055 Regensburg
Tel. (+49) 941 78956-8 Fax –9
www.margull.de

Verzeichniseintrag der Datei gelöscht ist (Referenzzähler gleich Null), so wird die Datei selbst gelöscht.

Beispiel:

- Die Datei in der folgenden Graphik ist unter zwei Verzeichniseinträgen zu erreichen: */Verzeichnis1/Name1* und */Verzeichnis1/Verzeichnis2/Name2*.
- Wird z.B. */Verzeichnis1/Name1* gelöscht, so wird der Referenzzähler im Inode um eins erniedrigt; die Datei bleibt bestehen.
- Erst wenn auch */Verzeichnis1/Verzeichnis2/Name2* entferntwird, ist der Referenzzähler Null und die Datei wird gelöscht.

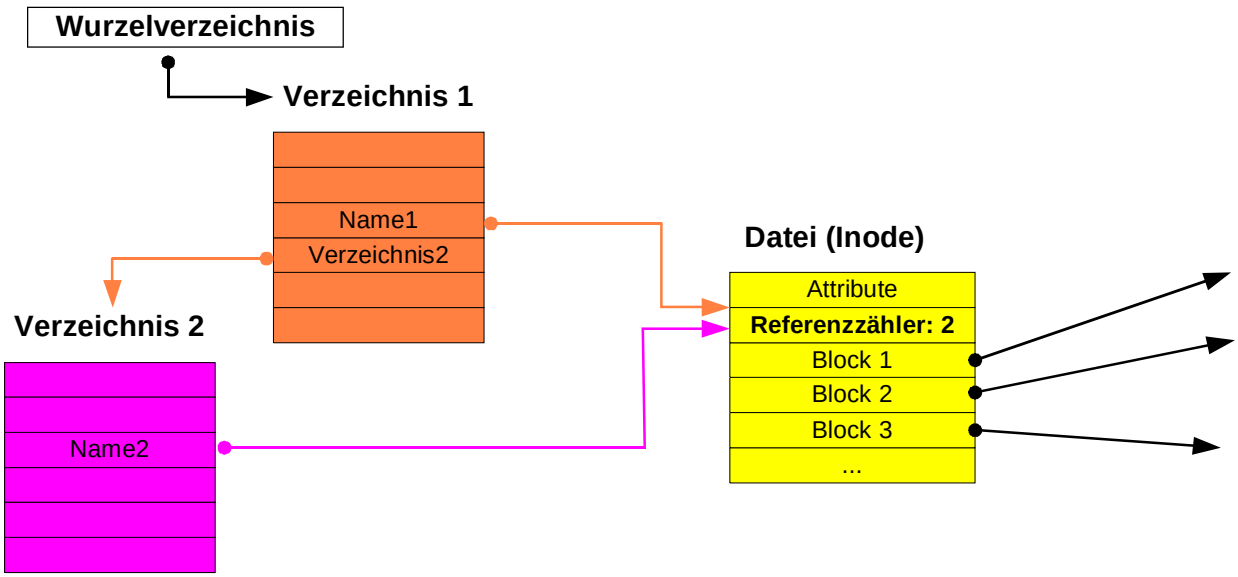


Abbildung 109: Implementierung eines harten Link

9.3.5.2 Verknüpfung (Symbolischer Link)

Eine Verknüpfung (symbolischer Link) enthält im wesentlichen den Pfad zu der Orginaldatei. Wird die Verknüpfung geöffnet, so wird zunächst der Pfad zur Orginaldatei eingelesen; anschließend wird diese geöffnet.

Eigenschaften:

- Wird die Orginaldatei gelöscht, so zeigt die Verknüpfung ins Leere; beim Öffnen der Verknüpfung erfolgt eine Fehlermeldung.
- Eine Verknüpfung kann über mehrere Dateisysteme gehen (z.B. in ein Netzwerkverzeichnis); ein harter Link muss immer im selben Dateisystem erfolgen.

9.3.6 Nicht-Funktionale Aspekte: Performance, Zuverlässigkeit, Effizienz

Dateisysteme sind ein wichtiger und oft benutzter Teil des Betriebssystems. Die Schnelligkeit von Dateisystemen ist deshalb für den gesamten Betrieb des Rechners von Bedeutung. Dazu gehören effektive Verwaltung der Verzeichnisse sowie der Blöcke einer Datei, ebenso wie effektive Verwaltung der freien Blöcke. Oftmals werden Daten im Hauptspeicher gehalten, um schnellen Zugriff zu ermöglichen.

Dr. Ulrich Margull
Kirchweg 6 * 93055 Regensburg
Tel. (+49) 941 78956-8 Fax –9
www.margull.de

Die Daten im Hauptspeicher sind jedoch nicht permanent; beim ungeplanten Abschalten des Rechners (z.B. bei Stromausfall oder Absturz des OS) gehen diese verloren. Dabei besteht die Gefahr, das die Verwaltungsstrukturen auf der Festplatte inkonsistent sind:

- Stimmt z.B. die FAT-Tabelle auf der Festplatte nicht mehr, so können einzelne Blöcke einer Datei verloren gehen
- in der Freiblockverwaltung können Blöcke fehlen; diese Blöcke sind weder frei verfügbar noch gehören sie einer Datei an
- Die Verzeichnisstruktur kann Fehler enthalten, ganze Verzeichnisse können verloren gehen

Gegen all diese Fehler muss ein Dateisystem abgesichert sein. Ältere Dateisysteme benötigten extra Korrekturprogramme, die im Fehlerfall die gesamte Partition untersucht haben und deren Konsistenz wiederhergestellt haben. Beim Systemstart wurde das Dateisystem überprüft und bei einem Fehler wurde eine Korrektur durchgeführt; dies dauerte teilweise mehrere Stunden. Als Ergebnis erhielt man ein konsistentes Dateisystem, und oftmals eine Reihe von Dateien, die Blöcke enthielten die keiner Datei zugeordnet wurden (utner Unix z.B. in einem lost+found Ordner).
Zudem muss ein Dateisystem performant sein; die Geschwindigkeit des gesamten Systems wird durch das Dateisystem bestimmt.

9.4 Protokollierende Dateisysteme (Journaling File Systems)

Alle Änderungen an den Metadaten (Verwaltungsdaten des File-Systems) werden zunächst in eine Log-Datei geschrieben. Die Log-Datei ist transaktionsorientiert, d.h. alle zusammengehörenden Aktionen werden als eine Transaktion behandlet. Wird z.B. eine neue Datei angelegt, so umfasst das einen Eintrag ins Verzeichnis, belegen der benötigten Blöcke, etc. Wurden alle Aktionen einer Transaktion ins Log geschrieben, so gilt die Transaktion als committed. Anschließend werden im Hintergrund die entsprechenden Änderungen an den Verwaltungsstrukturen durchgeführt. Wurden alle Aktionen erfolgreich durchgeführt, so wird die Transaktion aus dem Log entfernt.
Stürzt das System ab, so sind Null oder mehr Transaktionen im Log vorhanden. Keine dieser Transaktionen wurde erfolgreich beendet; sie werden alle neu aufgesetzt und beendet. Transaktionen, die nicht committed wurden (sie wurden noch nicht komplett ins Logfile geschrieben), werden nicht berücksichtigt. Im Fehlerfall ist die Konsistenz des Dateisystems sehr schnell wiederhergestellt. Zusätzlich bieten JFS auch im Normalbetrieb Performance-Vorteile.
Viele moderne Dateisysteme verwenden diesen Ansatz, z.B. NTFS auf Windows-Rechnern, UFS auf Solaris-Systemen, ReiserFS u.a. bei Linux-Systemen.
Beispiel für eine dateiorientierte Transaktion: eine Datei wächst, d.h. sie wird um einen Block erweitert.
Nötig sind die folgenden Aktionen:
1. Freien Block im Dateisystem suchen
2. Freien Block als belegt markieren
3. Den neuen Block in die Verlinkung des Inodes aufnehmen
4. Größe der Datei anpassen.

Die Schritte 2-4 werden zu einer Transaktion zusammengefasst, da sie zusammen ausgeführt werden müssen. Wird die Transaktion nicht komplett durchgeführt, so ist das Dateisystem inkonsistent: bricht man z.B. nach Schritt 2 ab, so wurde 1 Block als belegt markiert, der jedoch von keiner Datei benutzt wird.
Schritt 1 ist nicht Teil der Transaktion, da nur lesender Zugriff nötig ist.

Dr. Ulrich Margull
Kirchweg 6 * 93055 Regensburg
Tel. (+49) 941 78956-8 Fax –9
www.margull.de

9.5 Netzwerk-Dateisysteme: NFS

Das verbreitetste Netzwerk-Dateisystem ist NFS (Network File System) und wurde ursprünglich von SUN entwickelt. Es erlaubt, Laufwerke über das Netzwerk anzusprechen. Der Zugriff erfolgt dabei über RPC (remote procedure calls):

- auf dem Client findet ein Zugriff (Lesen/Schreiben/Öffnen etc.) auf ein Netzlaufwerk statt. Der Client nimmt die Anfrage entgegen und leitet sie via RPC zum Server weiter.
- Der Server führt den Zugriff auf dem echten Dateisystem aus und schickt die Antwort an den Client zurück.
- Das NFS-Laufwerk wird über einen Mount-Punkt ins Dateisystem gehängt; der ganze Vorgang ist für den Benutzer transparent.

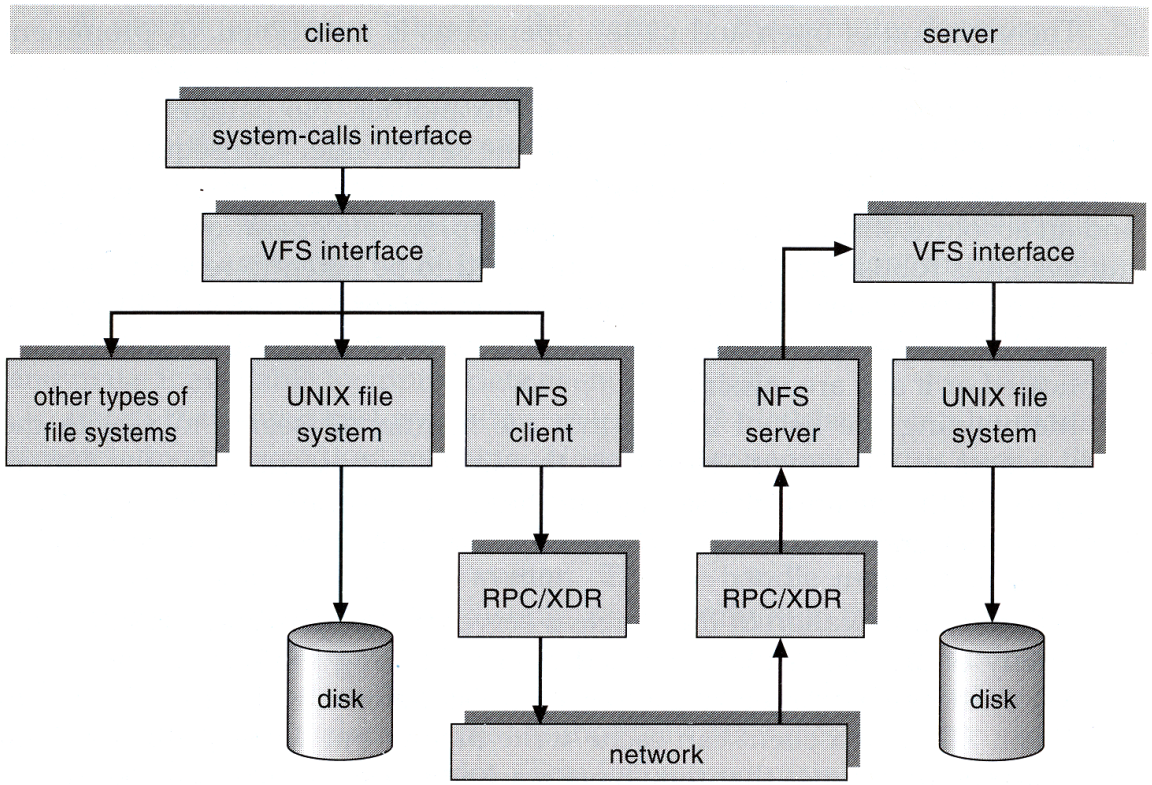


Abbildung 110: NFS Systemarchitektur

9.6 Beispiele

9.6.1 Win32-Schnittstelle (API)

In der folgenden Tabelle sind alle Dateioperationen der Win32 Schnittstelle aufgeführt:

Function	*Description*
AreFileApisANSI	Determines whether the file I/O functions are using the ANSI or OEM character set code page.
Cancello	Cancels all pending I/O operations that were issued by the calling thread for the specified file handle.

Dr. Ulrich Margull
Kirchweg 6 * 93055 Regensburg
Tel. (+49) 941 78956-8 Fax –9
www.margull.de

Function	*Description*
CloseHandle	Closes an open object handle.
CopyFile	Copies an existing file to a new file.
CopyFileEx	Copies an existing file to a new file.
CopyProgressRoutine	An application-defined callback function used with **CopyFileEx** and **MoveFileWithProgress**.
CreateDirectory	Creates a new directory.
CreateDirectoryEx	Creates a new directory with the attributes of a specified template directory.
CreateFile	Creates or opens a file object.
CreateIoCompletionPort	Creates and I/O completion port or associates an instance of an opened file with a newly created or an existing I/O completion port.
DefineDosDevice	Defines, redefines, or deletes MS-DOS device names.
DeleteFile	Deletes an existing file.
FileIOCompletionRoutine	An application-defined callback function used with **ReadFileEx** and **WriteFileEx**.
FindClose	Closes the specified search handle.
FindCloseChangeNotification	Stops change notification handle monitoring.
FindFirstChangeNotification	Creates a change notification handle.
FindFirstFile	Searches a directory for a file whose name matches the specified file name.
FindFirstFileEx	Searches a directory for a file whose name and attributes match those specified.
FindNextChangeNotification	Requests that the operating system signal a change notification handle the next time it detects an appropriate change.
FindNextFile	Continues a file search.
FlushFileBuffers	Clears the buffers for the specified file and causes all buffered data to be written to the file.
GetBinaryType	Determines whether a file is executable.
GetCurrentDirectory	Retrieves the current directory for the current process.
GetDiskFreeSpace	Retrieves information about the specified disk, including the amount of free space on the disk.
GetDiskFreeSpaceEx	Retrieves information about the specified disk, including the amount of free space on the disk.
GetDriveType	Determines whether a disk drive is a removable, fixed, CD-ROM, RAM disk, or network drive.
GetFileAttributes	Retrieves attributes for a specified file or directory.
GetFileAttributesEx	Retrieves attributes for a specified file or directory.
GetFileInformationByHandle	Retrieves file information for a specified file.
GetFileSize	Retrieves the size of a specified file.
GetFileSizeEx	Retrieves the size of a specified file.
GetFileType	Retrieves the file type of the specified file.
GetFullPathName	Retrieves the full path and file name of a specified file.

Dr. Ulrich Margull
Kirchweg 6 * 93055 Regensburg
Tel. (+49) 941 78956-8 Fax –9
www.margull.de

Function	*Description*
GetLogicalDrives	Returns a bitmask representing the currently available disk drives.
GetLogicalDriveStrings	Fills a buffer with strings that specify valid drives in the system.
GetLongPathName	Converts the specified path to its long form.
GetQueuedCompletionStatus	Attempts to dequeue an I/O completion packet from a specified I/O completion port.
GetShortPathName	Retrieves the short path form of a specified input path.
GetTempFileName	Creates a name for a temporary file.
GetTempPath	Retrieves the path of the directory designated for temporary files.
Int32x32To64	Multiplies two signed 32-bit integers.
Int64ShllMod32	Performs a left logical shift operation on an unsigned 64-bit integer value.
Int64ShraMod32	Performs a right arithmetic shift operation on a signed 64-bit integer value.
Int64ShrlMod32	Performs a right logical shift operation on an unsigned 64-bit integer value.
LockFile	Locks a region in an open file.
LockFileEx	Locks a region in an open file for shared or exclusive access.
MoveFile	Moves an existing file or a directory.
MoveFileEx	Moves an existing file or a directory.
MoveFileWithProgress	Moves a file or directory.
MulDiv	Multiplies two 32-bit values and then divides the 64-bit result by a third 32-bit value.
PostQueuedCompletionStatus	Posts an I/O completion packet to an I/O completion port.
QueryDosDevice	Retrieves information about MS-DOS device names.
ReadDirectoryChangesW	Retrieves information describing the changes occurring within a directory.
ReadFile	Reads data from a file, starting at the specified position.
ReadFileEx	Reads data from a file asynchronously.
ReadFileScatter	Reads data from a file and stores the data into a set of buffers.
RemoveDirectory	Deletes an existing empty directory.
ReplaceFile	Replaces one file with another file.
SearchPath	Searches for the specified file.
SetCurrentDirectory	Changes the current directory for the current process.
SetEndOfFile	Moves the end-of-file position for the specified file.
SetFileApisToANSI	Causes the file I/O functions to use the ANSI character set code page.
SetFileApisToOEM	Causes the file I/O functions to use the OEM character set code page.
SetFileAttributes	Sets a file's attributes.
SetFilePointer	Moves the file pointer of an open file.

Dr. Ulrich Margull
Kirchweg 6 * 93055 Regensburg
Tel. (+49) 941 78956-8 Fax –9
www.margull.de

Function	*Description*
SetFilePointerEx	Moves the file pointer of an open file.
SetFileSecurity	Sets the security of a file or directory object.
SetVolumeLabel	Sets the label of a file system volume.
UInt32x32To64	Multiplies two unsigned 32-bit integers, returning an unsigned 64-bit integer result.
UnlockFile	Unlocks a previously locked region in an open file.
UnlockFileEx	Unlocks a previously locked region in an open file.
WriteFile	Writes data to a file.
WriteFileEx	Writes data to a file asynchronously.
WriteFileGather	Gathers data from a set of buffers and writes the data to a file.

9.6.2 MS-DOS Dateisystem: FAT

Eigenschaften:

- 8 + 3 Namenskonvention (nur Großbuchstaben)
- Die erste Version (MS-DOS 1.0) hatte nur ein Verzeichnis, in dem alle Dateien lagen; die nächsten Versionen erlaubten dann hierarchische Verzeichnisstrukturen.
- Links sind nicht erlaubt
- Kein Rechtesystem; jeder Benutzer hat Zugriff auf alle Dateien
- MS-DOS Verzeichniseintrag ist 32 Bytes groß und wie folgt aufgebaut:

Bytes	*8*	*3*	*1*	*10*	*2*	*2*	*2*	*4*
	Name	Ext	Attr.	Reserviert	Zeit	Datum	Erster Block	Größe

Die einzelnen Einträge sind:

- Name und Typ (Extension) der Datei
- Attribute: R = Read-Only, H = Hidden (versteckt), A = muss archiviert werden, S = Systemdatei
- 10 Bytes sind reserviert (unbenutzt)
- Zeit und Datum, an dem die Datei erzeugt wurde: die Zeit wird dabei in Einheiten von 2 Sekunden angegeben (1Tag ist 86400 Sekunden), das Datum besteht aus 5 Bits für den Tag, 4 Bits für den Monat und 7 Bits für das Jahr (0 entspricht 1980; somit reicht das Datum bis 2107; *Achtung! Jahr-2108-Problem mit MS-DOS!!*)
- Der erste Block bezeichnet den ersten Block der Datei in der FAT; von dort können jeweils die nächsten Blöcke der Datei gefunden werden
- Größe der Datei: dadurch kann unter anderem im letzten Block das Ende der Datei ermittelt werden.

- MS-DOS verwendet eine FAT für die Zuordnung der Beschreibung der Blöcke einer Datei:
  - Zunächst FAT-12 für Floppies: bei 12 Bit = 4096 Blöcke und 512 Bytes / Block können 2 MB addressiert werden
  - Später FAT-16 mit 16-Bit Adressen. Diese Adresse adressiert einen Cluster d.h. mehrere Blöcke auf einmal; dadurch können je nach Cluster-Größe unterschiedliche Datenmengen adressiert werden:

Dr. Ulrich Margull
Kirchweg 6 * 93055 Regensburg
Tel. (+49) 941 78956-8 Fax –9
www.margull.de

Tabelle 7: Maximale Partitionsgrößen bei MS-DOS

Block Größe	*FAT-12*	*FAT-16*	*FAT-32*
0,5 kB	2 MB		
1 kB	4 MB		
2 kB	8 MB	128 MB	
4 kB	16 MB	256 MB	1 TB
8 kB		512 MB	2 TB
16 kB		1024 MB	2 TB
32 kB		2048 MB	2 TB

Bei größerer Clustergröße nimmt jedoch der interne Verschnitt der Dateien zu; beträgt die Clustergröße z.B. 32 kB, so belegt jede Datei mindestens 32 kB, auch wenn sie nur einige Bytes enthält.

9.6.3 NTFS

Hier ein Überblick über eine Eigenschaften von NTFS (für einen vollständigen Überblick, siehe [1]):

- NTFS hat eine feste Blockgröße, von 512 Bytes bis 64 kB (meistens wird 4 kB verwendet). Zur Adressierung eines Blocks wird eine 64-Bit Zahl verwendet.
- Hauptstruktur jeder NTFS-Partition ist die MFT (Master File Table), eine lineare Liste von 1 kB-großen Einträgen. Jeder Eintrag bezeichnet eine Datei oder ein Verzeichnis, und enthält den Dateinamen, alle Attribute, und die verwendeten Blöcke. Große Dateien können mehr als einen Eintrag besitzen.
- Freie MFT-Einträge werden über eine Bitmap verwaltet.
- Die MFT ist selbst eine Datei und kann irgendwo in dem Dateisystem vorhanden sein. Der Boot-Block enthält einen Verweis auf die MFT.
- Dateien haben sowohl einen 8+3 Namen (für MS-DOS Kompadibilität) als auch einen längeren Namen (in Unicode-Zeichen).
- Attribute von NTFS-Dateien sind unter anderem: Standardinformationen (Flag bits, Zugriffszeiten, etc.), Dateiname, Sicherheitsattribute, Verweis auf weitere Attribute (falls nötig)
- Kurze Dateien (einige hundert Bytes) werden direkt in der MFT gespeichert, anstelle von Verweisen auf die benötigten Blöcke (reduziert Speicherbedarf für kleine Dateien)
- NTFS versucht, Dateien immer möglichst kontinuierlich zu speichern. Falls das gelingt, reicht ein Eintrag auf den ersten Block, und die Anzahl der folgenden Blöcke; gelingt dies nicht, so werden weitere Einträge verwendet, für jeden kontinuierlichen Bereich einen.
- NTFS bietet Komprimierung und Verschlüsselung für die Dateien
- NTFS ermöglicht mehrere Datenströme (Streams) in einer Datei: die Datei enthält mehrere Datenströme, z.B. Video- und Audiodaten in unterschiedlichen Strömen. Wurde ursprünglich implementiert, um MAC-Dateien in NTFS zu speichern: diese bestehen aus zwei Strömen, den eigentlichen Daten und Metadaten.