Moderne Betriebssysteme

5., aktualisierte Auflage

Andrew S. Tanenbaum
Herbert Bos

Pearson

EXTRAS ONLINE

die Zeit der Erzeugung, den Startblock und die exakte Dateigröße. Dateinamen, die kürzer als 8+3 Zeichen sind, werden links ausgerichtet und rechts in jedem einzelnen Feld mit Leerzeichen aufgefüllt. Das Feld *Attribute* ist neu und enthält Bits um anzuzeigen, dass die Datei nur zu lesen ist, archiviert werden muss, versteckt ist oder eine Systemdatei ist. Dateien, die nur gelesen werden dürfen, können nicht beschrieben werden. Das schützt die Daten vor unbeabsichtigter Beschädigung. Das Archiv-Bit erfüllt keine tatsächliche Betriebssystemfunktion (d.h., MS-DOS überprüft oder setzt es nicht). Die Intention dahinter ist, dass Archivierungsprogramme der Benutzerebene es bei Sicherung der Datei löschen und andere Programme es nach Veränderungen setzen. Auf diese Weise können Sicherungsprogramme durch einfache Untersuchung dieses Bits bei jeder Datei entscheiden, ob sie archiviert werden muss oder nicht. Das Hidden-Bit kann gesetzt werden, um eine Datei bei der Verzeichnisausgabe unsichtbar zu machen. Der Hauptverwendungszweck ist, Neulinge nicht mit Dateien zu verwirren, die sie nicht verstehen. Das System-Bit schließlich versteckt ebenfalls Dateien. Außerdem können Systemdateien nicht fälschlicherweise durch das *del*-Kommando gelöscht werden. Die Hauptkomponenten von MS-DOS haben dieses Bit gesetzt.

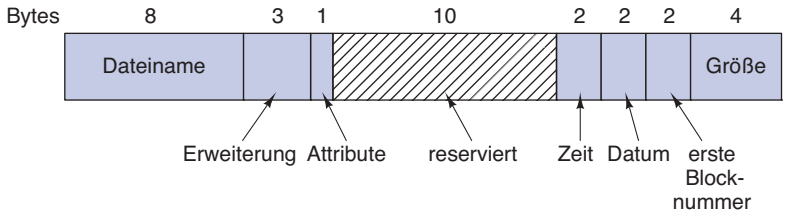

Abbildung 4.32: Der MS-DOS-Verzeichniseintrag.

Der Verzeichniseintrag enthält auch das Datum und die Zeit der Erzeugung oder letzten Modifikation. Die Zeit wird nur auf ±2 s genau angegeben, da sie in einem 2-Byte-Feld gespeichert wird, das nur 65.536 unterschiedliche Werte speichern kann (ein Tag hat 86.400 Sekunden). Das Zeitfeld wird in Sekunden (5 Bit), Minuten (6 Bit) und Stunden (5 Bit) unterteilt. Das Datum wird in Tagen gezählt und enthält drei Unterfelder: Tag (5 Bit), Monat (4 Bit) und Jahr −1980 (7 Bit). Da eine 7-Bit-Zahl für das Jahr benutzt wird und die Zeitrechnung mit 1980 beginnt, ist das größte darstellbare Jahr das Jahr 2107. Folglich hat MS-DOS ein eingebautes Jahr-2108-Problem. Um Katastrophen zu vermeiden, sollten sich die MS-DOS-Benutzer so früh wie möglich um die Jahr-2108-Fähigkeit ihres Systems kümmern. Wenn MS-DOS 32 Bit für das kombinierte Datums- und Zeitfeld verwendet hätte, so könnte jede Sekunde exakt dargestellt werden und die Katastrophe hätte auf das Jahr 2116 verschoben werden können.

MS-DOS speichert die Dateigröße als eine 32-Bit-Zahl, somit kann eine Datei theoretisch 4 GB groß sein. Andere Vorgaben (weiter unten beschrieben) beschränken die maximale Dateigröße auf 2 GB oder weniger. Ein überraschend großer Teil des Eintrags (10 Byte) ist ungenutzt.

MS-DOS hält die Informationen über die Dateiblöcke im Speicher in einer Datei-Allokationstabelle. Der Verzeichniseintrag enthält die Nummer des ersten Dateiblocks.

Diese Nummer wird dazu benutzt, um eine FAT aus 64.000 Einträgen im Speicher zu indizieren. Durch Verfolgen der Kette können alle Blöcke gefunden werden. Die Funktionsweise der FAT ist in *Abbildung 4.14* illustriert.

Das FAT-Dateisystem gibt es in drei Versionen: FAT-12, FAT-16 und FAT-32 – abhängig von der Anzahl der Bits, die für eine Plattenadresse verwendet werden. Eigentlich ist FAT-32 so etwas wie eine Fehlbenennung, da nur die niederwertigen 28 Bit der Plattenadresse genutzt werden. Es sollte eigentlich FAT-28 heißen, doch Zweierpotenzen klingen viel hübscher.

Eine weitere Variante des FAT-Dateisystems ist exFAT, das von Microsoft für große Wechselmedien eingeführt wurde. Apple hat exFAT lizenziert, damit gibt es ein modernes Dateisystem, das zur Dateiübertragung zwischen Windows- und MacOS-Rechnern in beide Richtungen verwendet werden kann. Da exFAT proprietär ist und Microsoft die Spezifikation nicht veröffentlicht hat, werden wir es hier nicht weiter betrachten.

Für alle FATs können die Plattenblöcke auf ein Vielfaches von 512 Byte (für jede Partition auch unterschiedlich) gesetzt werden, wobei die Menge der erlaubten Blockgrößen (bei Microsoft **Clustergrößen** (*cluster size*) genannt) für jede Variante unterschiedlich ist. Die erste Version von MS-DOS benutzte FAT-12 mit 512-Byte-Blöcken und erlaubte daher eine maximale Partitionsgröße von $2^{12} \times 512$ Byte (eigentlich nur 4086×512 Byte, da zehn der Plattenadressen für spezielle Markierungen wie Dateiende, fehlerhafter Block etc. verwendet wurden). Mit diesen Parametern war die maximale Partitionsgröße ca. 2 MB und die Größe der FAT im Speicher betrug 4.096 Einträge mit 2 Byte pro Eintrag. 12-Bit-Tabelleneinträge wären zu langsam gewesen.

Dieses System funktionierte für Disketten recht gut, doch als die Festplatten aufkamen, wurde es zum Problem. Microsoft löste dieses Problem, indem es zusätzliche Blockgrößen von 1 KB, 2 KB und 4 KB zuließ. Diese Änderung erhielt die Struktur und die Größe der FAT-12-Tabelle, erlaubte aber Partitionsgrößen von bis zu 16 MB.

Da MS-DOS vier Partitionen pro Laufwerk unterstützte, konnte das neue FAT-12-Dateisystem mit Platten bis 64 MB umgehen. Doch damit war die Grenze von FAT-12 erreicht und es musste etwas geschehen. Also wurde FAT-16 mit 16-Bit-Zeigern und zusätzlichen Blockgrößen von 8 KB, 16 KB und 32 KB eingeführt. (32.768 ist die größte Zweierpotenz, die in 16 Bit dargestellt werden kann.) Die FAT-16-Tabelle beanspruchte nun die ganze Zeit über 128 KB Arbeitsspeicher, doch wurde sie aufgrund des verfügbaren größeren Speichers weithin verwendet und ersetzte schnell das FAT-12-Dateisystem. Die größte Plattenpartition, die von FAT-16 unterstützt wird, ist 2 GB (64.000 Einträge, jeder mit einer Länge von 32 KB) und die größte unterstützte Platte ist 8 GB groß, nämlich vier Partitionen von der Größe 2 GB. Eine ganze Weile war dies ausreichend.

Doch nicht für immer. Für Geschäftsbriefe stellt dieses Limit kein Problem dar, aber im Fall der Speicherung von digitalen Videos mit dem DV-Standard enthalten 2 GB gerade einmal etwas mehr als 9 Minuten Video. Als Konsequenz der Tatsache, dass eine PC-Festplatte nur vier Partitionen unterstützt, dauert das längste Video auf einer

Platte ungefähr 38 Minuten, egal wie groß die Platte ist. Dieses Limit hat zur Folge, dass das längste Video, das während des laufenden Betriebs bearbeitet werden kann, weniger als 19 Minuten dauert, da sowohl Eingabe- als auch Ausgabedateien benötigt werden.

Mit der zweiten Version von Windows 95 wurde das FAT-32-Dateisystem mit seinen 28-Bit-Plattenadressen eingeführt und die Version von MS-DOS, auf der Windows 95 aufsetzte, wurde angepasst, um FAT-32 zu unterstützen. Bei diesem System konnten die Partitionen theoretisch $2^{28} \times 2^{15}$ Byte groß sein, doch sind sie tatsächlich auf 2 TB (2.048 GB) beschränkt. Das System verfolgt die Partitionsgrößen nämlich intern in 512 Byte großen Sektoren über eine 32-Bit-Nummer und $2^9 \times 2^{32}$ ergibt 2 TB. Die maximale Partitionsgröße für die verschiedenen Blockgrößen und alle drei FAT-Typen ist in ▶ *Abbildung 4.33* dargestellt.

Blockgröße	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		1.024 MB	2 TB
32 KB		2.048 MB	2 TB

Abbildung 4.33: Die maximalen Partitionsgrößen für verschiedene Blockgrößen, die leeren Felder zeigen verbotene Kombinationen an.

Neben der Unterstützung größerer Platten hat das FAT-32-Dateisystem noch zwei weitere Vorteile gegenüber FAT-16. Erstens kann eine 8-GB-Platte, die FAT-32 verwendet, mit nur einer Partition betrieben werden. Mit FAT-16 muss sie vier Partitionen haben, die unter Windows dem Benutzer als die logischen Laufwerke *C:*, *D:*, *E:* und *F:* erscheinen. Es liegt nun am Anwender zu entscheiden, welche Dateien er auf welchem Laufwerk haben möchte, und darüber informiert zu bleiben, was wo ist.

Der andere Vorteil von FAT-32 gegenüber FAT-16 ist, dass für eine gegebene Partitionsgröße eine kleinere Blockgröße verwendet werden kann. Beispielsweise muss FAT-16 für eine 2-GB-Partition 32-KB-Blöcke verwenden, da andernfalls mit 64.000 verfügbaren Plattenadressen nicht die gesamte Partition abgedeckt werden könnte. Im Gegensatz dazu kann FAT-32 zum Beispiel 4-KB-Blöcke für eine 2-GB-Partition verwenden. Der Vorteil kleinerer Blöcke liegt darin, dass die meisten Dateien viel kleiner als 32 KB sind. Wenn die Blockgröße 32 KB beträgt, so belegt eine 10 Byte große Datei 32 KB des Plattenplatzes. Falls die durchschnittliche Dateigröße zum Beispiel 8 KB ist, dann werden mit 32-KB-Blöcken drei Viertel der gesamten Platte verschwendet –

kein besonders effizienter Weg, die Platte auszunutzen. Bei einer 8-KB-Datei und 4-KB-Blöcken wird kein Plattenplatz verschwendet, doch der Preis dafür ist, dass mehr RAM durch die FAT verbraucht wird. Bei einer Blockgröße von 4 KB gibt es bei einer 2-GB-Platte 512.000 Blöcke, also muss die FAT 512.000 Einträge im Speicher enthalten (damit belegt sie 2 MB RAM).

MS-DOS benutzt die FAT, um die freien Plattenblöcke zu verfolgen. Jeder Block, der zurzeit nicht belegt ist, wird mit einem bestimmten Code markiert. Wenn MS-DOS einen neuen Block benötigt, so sucht es die FAT nach einem Eintrag ab, der diesen Code enthält. Eine Bitmap oder eine Freibereichsliste wird folglich nicht benötigt.

4.5.2 Das UNIX-V7-Dateisystem

Sogar die frühen Versionen von UNIX hatten ein recht ausgefeiltes Mehrbenutzer-Dateisystem, da UNIX von MULTICS abgeleitet worden war. Im Folgenden werden wir das V7-Dateisystem besprechen, das Dateisystem für die PDP-11, die UNIX berühmt machte. Eine moderne Version des UNIX-Dateisystems für Linux behandeln wir in *Kapitel 10*.

Das Dateisystem hat die Form eines Baums, der mit dem Wurzelverzeichnis beginnt. Zusätzlich gibt es Links, die einen gerichteten azyklischen Graph entstehen lassen. Dateinamen können bis zu 14 Zeichen lang sein und jedes ASCII-Zeichen enthalten, bis auf / (da dies der Separator zwischen den Komponenten eines Pfads ist) und NUL (da dies zum Auffüllen der Namen, die kürzer als 14 Zeichen sind, verwendet wird). NUL hat den numerischen Wert 0.

Ein UNIX-Verzeichniseintrag enthält einen Eintrag für jede Datei in diesem Verzeichnis. Jeder einzelne Eintrag ist äußerst simpel, da UNIX das I-Node-Schema aus *Abbildung 4.15* benutzt. Ein Verzeichniseintrag enthält nur zwei Felder: den Dateinamen (14 Byte) und die Nummer des I-Node für diese Datei (2 Byte), siehe ▶ *Abbildung 4.34*. Diese Parameter begrenzen die Anzahl der Dateien pro Dateisystem auf 64.000.

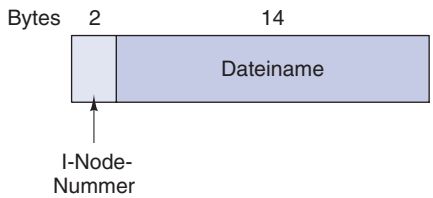

Abbildung 4.34: UNIX-V7-Dateieintrag.

Wie der I-Node aus *Abbildung 4.15* enthält auch der UNIX-I-Node einige Attribute. Die Attribute umfassen die Dateigröße, drei Zeiten (Erzeugung, letzter Zugriff und letzte Veränderung), Eigentümer, Gruppe, Schutzinformationen und einen Zähler für die Anzahl der Verzeichniseinträge, die auf den I-Node zeigen. Das letzte Feld wird zur Verwaltung der Links benötigt: Jedes Mal, wenn ein neuer Link auf einen I-Node erzeugt wird, erhöht sich der Zähler im I-Node. Wenn ein Link entfernt wird, so wird

der Zähler dekrementiert. Erreicht der Zähler 0, dann wird der I-Node zurückgefordert und der entsprechende Plattenblock wieder in die Freibereichsliste eingehängt.

Die Verfolgung der freien Plattenblöcke läuft unter Verwendung einer verallgemeinerten Version der *Abbildung 4.15* ab, um sehr große Dateien verwalten zu können. Die ersten zehn Plattenadressen werden im I-Node selbst gespeichert. Für kleine Dateien befinden sich alle notwendigen Informationen also im I-Node und werden von der Platte in den Arbeitsspeicher geladen, wenn die Datei geöffnet wird. Für etwas größere Dateien ist eine der Adressen im I-Node die Adresse eines Plattenblocks, der **einfach indirekter Block** (*single indirect block*) genannt wird. Dieser Block enthält zusätzliche Plattenadressen. Wenn dies immer noch nicht ausreicht, dann enthält eine weitere Adresse im I-Node, **doppelt indirekter Block** (*double indirect block*) genannt, die Adresse eines Blocks, der eine Liste von einfach indirekten Blöcken enthält. Jeder dieser einfach indirekten Blöcke zeigt auf ein paar hundert Datenblöcke. Sollte das immer noch nicht ausreichen, so kann auch ein **dreifach indirekter Block** (*triple indirect block*) verwendet werden. Das vollständige Bild ist in ▶ *Abbildung 4.35* zu sehen.

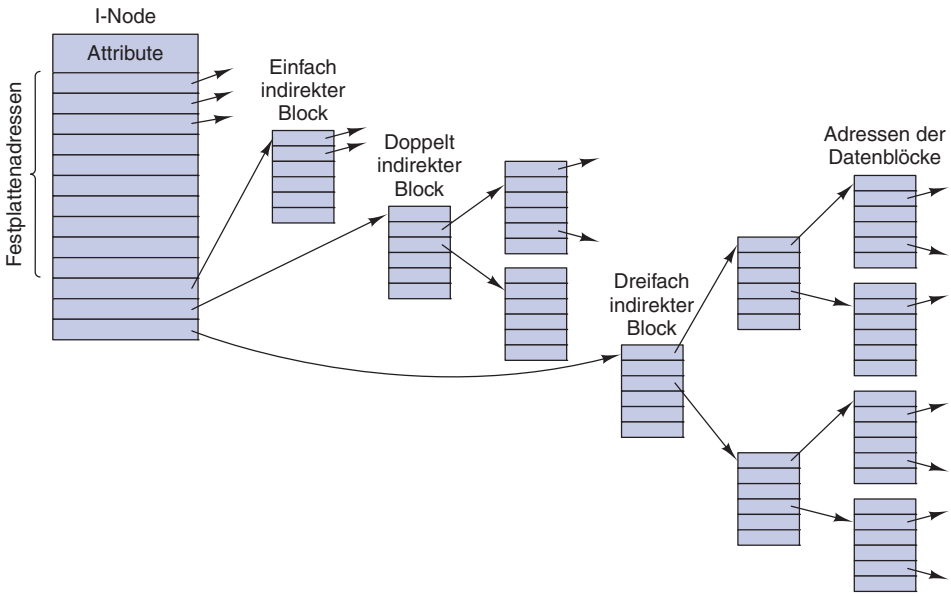

Abbildung 4.35: I-Node unter UNIX.

Wenn eine Datei geöffnet wird, dann muss das Dateisystem den gegebenen Dateinamen annehmen und dessen Plattenblöcke finden. Als Beispiel wollen wir betrachten, wie nach dem Pfadnamen */usr/ast/mbox* gesucht wird. Obwohl wir hier UNIX als Beispiel nehmen, so ist doch der Algorithmus grundsätzlich der gleiche für alle hierarchischen Verzeichnissysteme. Zuerst lokalisiert das Dateisystem das Wurzelverzeichnis. In UNIX ist dessen I-Node an einer festen Position auf der Platte. Von diesem I-Node aus wird das Wurzelverzeichnis gesucht, das überall auf der Platte sein kann; hier nehmen wir an, es sei in Block 1.

Danach wird das Wurzelverzeichnis gelesen und dort die erste Komponente des Pfads gesucht, hier *usr*, um die I-Node-Nummer des Eintrags */usr* zu finden. Die Lokalisierung eines I-Node aus seiner Nummer ist unkompliziert, denn jeder I-Node hat eine feste Position auf der Platte. Von diesem I-Node aus wird das Verzeichnis für */usr* gefunden und die nächste Komponente, *ast*, darin gesucht. Wenn der Eintrag für *ast* gefunden ist, dann hat das System den I-Node für */usr/ast*. Wiederum von diesem I-Node aus kann das Verzeichnis selbst gefunden und nach *mbox* gesucht werden. Der I-Node für diese Datei wird dann in den Speicher gelesen und dort aufbewahrt, bis die Datei geschlossen wird. Die Suche ist in ▶ *Abbildung 4.36* dargestellt.

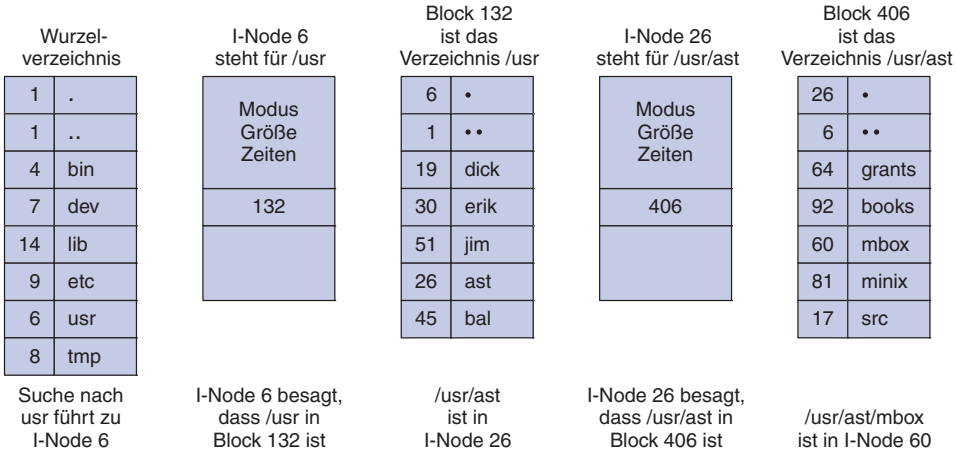

Abbildung 4.36: Schritte, um */usr/ast/mbox* zu finden.

Relative Pfadnamen werden nach dem gleichen Verfahren gesucht, nur beginnt hierbei die Suche im Arbeitsverzeichnis und nicht im Wurzelverzeichnis. Jedes Verzeichnis hat die Einträge . und .., die angelegt werden, wenn das Verzeichnis erzeugt wird. Der Eintrag . hat die I-Node-Nummer für das aktuelle Verzeichnis und der Eintrag .. enthält die I-Node-Nummer für das Vorgängerverzeichnis. Folglich sucht eine Prozedur, die *../chris/prog.c* finden soll, einfach nach .. im Arbeitsverzeichnis. Sie findet die I-Node-Nummer des Vorgängerverzeichnisses und sucht dieses nach *chris* ab. Es wird kein spezieller Mechanismus für den Umgang mit diesem Namen benötigt. So weit es das Verzeichnissystem betrifft, sind diese Namen wie jeder andere Name lediglich ASCII-Zeichenketten. Der einzige Trick hier ist, dass .. im Wurzelverzeichnis auf sich selbst zeigt.

4.6 Forschung zu Dateisystemen

Dateisysteme inspirierten die Forschung immer mehr als die übrigen Gebiete der Betriebssysteme und das ist auch heute noch so. Ganze Konferenzen wie FAST, MSST und NAS widmen sich fast ausschließlich den Datei- und Speichersystemen.

Speziell die Zuverlässigkeit von Speicher- und Dateisystemen ist Gegenstand zahlreicher Forschungsarbeiten. Eine wirksame Methode zur Gewährleistung der Zuverlässigkeit ist der formale Beweis der Sicherheit Ihres Systems selbst bei katastrophalen Ereignissen wie Abstürzen (Chen et al., 2017). Angesichts der zunehmenden Beliebtheit von SSDs als primäres Speichermedium ist es auch interessant zu untersuchen, wie gut sie sich in großen Unternehmensspeichersystemen bewähren (Maneas et al., 2020).

Wie wir in diesem Kapitel gesehen haben, sind Dateisysteme komplexe Gebilde und die Entwicklung neuer Systeme ist nicht einfach. Viele Betriebssysteme ermöglichen das Erzeugen von Dateisystemen im Benutzerraum, z.B. FUSE (Filesystem in Userspace) unter Linux, aber die Leistung ist im Allgemeinen viel geringer. Mit der Markteinführung neuer Speichermedien wie Low-Level-SSDs wächst der Bedarf an modernen Speichersystemen; die Forschungsanstrengungen müssen intensiviert werden, um schnell leistungsstarke Dateisysteme zu entwickeln (Miller et al., 2021). Tatsächlich wird ein Großteil der Forschung zu Dateisystemen von den Fortschritten in der Speichertechnologie angetrieben. Wie können wir zum Beispiel effiziente Dateisysteme für die neuen persistenten Speicher entwickeln (Chen et al., 2021; Neal, 2021)? Oder wie können wir die Überprüfung von Dateisystemen beschleunigen (Domingo, 2021)? Sogar die Fragmentierung wirft für Festplatten und SSDs unterschiedliche Probleme auf und erfordert unterschiedliche Ansätze (Kesavan, 2019).

Die Speicherung zunehmender Datenmengen auf demselben Dateisystem ist eine Herausforderung, insbesondere bei mobilen Geräten, was zur Entwicklung neuer Methoden führt, um die Daten zu komprimieren, ohne das System zu sehr zu verlangsamen, zum Beispiel durch Berücksichtigung der Zugriffsmuster von Dateien (Ji et al., 2021). Wir haben gesehen, dass als Alternative zur datei- oder blockweisen Komprimierung einige Dateisysteme heute die Deduplizierung über das gesamte System unterstützen, um zu verhindern, dass dieselben Daten doppelt gespeichert werden. Leider führt die Deduplizierung zu einer schlechten Datenlokalisierung und der Versuch, eine gute Deduplizierung ohne Leistungseinbußen aufgrund mangelnder Lokalisierung zu erreichen, ist schwierig (Zou, 2021). Wurde die Datendeduplizierung über den ganzen Speicherbereich ausgeführt, ist es natürlich viel schwieriger abzuschätzen, wie viel Speicherplatz noch übrig ist oder übrig bleiben wird, wenn wir eine bestimmte Datei löschen (Harnik, 2019).

ZUSAMMENFASSUNG

Von außen betrachtet ist ein Dateisystem eine Ansammlung von Dateien und Verzeichnissen sowie den Operationen darauf. Dateien können gelesen und beschrieben werden, Verzeichnisse können erzeugt und gelöscht werden und Dateien können schließlich noch von Verzeichnis zu Verzeichnis verschoben werden. Die meisten modernen Dateisysteme unterstützen ein hierarchisches Verzeichnissystem, in denen die Verzeichnisse Unterverzeichnisse haben können und diese wiederum weitere Unterverzeichnisse ad infinitum enthalten.

Von innen betrachtet sieht ein Dateisystem etwas anders aus. Die Dateisystementwickler müssen sich darum kümmern, wie Speicherplatz belegt wird und wie das System verfolgt, welche Blöcke zu welcher Datei gehören. Die Möglichkeiten umfassen zusammenhängende Dateien, verkettete Listen, Datei-Allokationstabellen und I-Nodes. Die verschiedenen Systeme haben unterschiedliche Verzeichnisstrukturen. Attribute können in den Verzeichnissen oder irgendwo anders (z.B. in einem I-Node) abgelegt werden. Plattenplatz kann über Freibereichslisten oder Bitmaps verwaltet werden. Die Zuverlässigkeit von Dateisystemen wird durch inkrementelle Sicherungen erweitert und durch Programme verstärkt, die beschädigte Dateisysteme wiederherstellen können. Die Performanz des Dateisystems ist wichtig und kann auf verschiedene Art und Weise verbessert werden. Dies kann durch Caching, vorausschauendes Lesen und bedachtes Platzieren der Dateiblöcke nahe zueinander geschehen. Log-basierte Dateisysteme verbessern die Performanz zusätzlich durch das Schreiben großer Einheiten.

Beispiele für die verschiedenen Dateisysteme sind ISO 9660, MS-DOS und UNIX. Sie unterscheiden sich in vielerlei Hinsicht, beispielsweise wie sie die Zugehörigkeit der Blöcke zu einer Datei verfolgen, in ihrer Verzeichnisstruktur und in der Verwaltung des freien Plattenplatzes.

Übungen

1. Wenn ein Benutzer unter Windows eine Datei, die im Windows Explorer gelistet ist, mit einem Doppelklick ausführt, so wird ein Programm gestartet, dem die Datei als Parameter übergeben wird. Geben Sie zwei verschiedene Möglichkeiten an, wie das Betriebssystem Kenntnis darüber erlangen könnte, welches Programm gestartet werden soll.

2. In den frühen UNIX-Systemen begannen ausführbare Dateien (*a.out*-Dateien) mit einer sehr speziellen magischen Zahl, die nicht zufällig gewählt war. Diesen Dateien ging ein Header voran, gefolgt von den Text- und Datensegmenten. Warum wurde Ihrer Meinung nach eine spezielle Zahl für die ausführbaren Dateien verwendet, wohingegen andere Dateitypen eine mehr oder weniger zufällige magische Zahl als erstes Wort enthielten?

3. In *Abbildung 4.5* ist eines der Attribute die Länge des Datensatzes. Warum ist dieses Attribut für das Betriebssystem interessant?

4. Ist der `open`-Systemaufruf für UNIX absolut notwendig? Was wären die Konsequenzen, wenn er nicht vorhanden wäre?

5. Systeme, die sequenzielle Dateien unterstützen, haben immer auch eine Operation, um diese zurückzuspulen. Benötigen Systeme, die Dateien mit wahlfreiem Zugriff unterstützen, dies auch?

6. Einige Betriebssysteme stellen den Systemaufruf `rename` zur Verfügung, um einer Datei einen neuen Namen zu geben. Gibt es irgendeinen Unterschied zwischen diesem Aufruf und dem Kopieren der Datei in eine neue Datei und anschließender Löschung der alten Datei?

7. Ein einfaches Betriebssystem unterstützt nur ein Verzeichnis, dieses darf aber beliebig viele Dateien mit beliebig langen Dateinamen haben. Kann etwas Ähnliches wie ein hierarchisches Dateisystem simuliert werden? Wie?

8. Unter UNIX und Windows erfolgt der wahlfreie Zugriff durch einen speziellen Systemaufruf, der den Zeiger für die „aktuelle Position“ in der Datei auf ein anderes Byte der Datei bewegt. Schlagen Sie einen anderen Weg vor, wie wahlfreier Zugriff ohne diesen Systemaufruf realisiert werden kann.

9. Betrachten Sie den Verzeichnisbaum der *Abbildung 4.9*. Falls */usr/jim* das Arbeitsverzeichnis ist, was ist dann der absolute Pfadname für die Datei, deren relativer Pfadname *../ast/x* lautet?

10. Zusammenhängende Belegung durch Dateien führt, wie im Text erwähnt, zur Fragmentierung der Platte, da im letzten Plattenblock bei denjenigen Dateien Platz verschwendet wird, deren Länge kein ganzzahliges Vielfaches der Blockgröße ist. Ist dies interne oder externe Fragmentierung? Bilden Sie eine Analogie zu einem Konzept, das in einem früheren Kapitel besprochen wurde.

11. Angenommen, eine Dateisystemprüfung ergibt, dass ein Block zwei verschiedenen Dateien zugewiesen wurde, */home/hjb/dadjokes.txt* und */etc/motd*. Beides sind Textdateien. Die Dateisystemprüfung dupliziert die Daten des Blocks und ordnet /etc/motd den neuen Block zu. Beantworten Sie die folgenden Fragen.

i. Unter welchen realistischen Umständen können die Daten der beiden Dateien noch korrekt und konsistent mit ihrem ursprünglichen Inhalt bleiben?

ii. Wie kann der Benutzer untersuchen, ob die Dateien beschädigt wurden?

iii. Wenn die Daten einer oder beider Dateien beschädigt worden sind, welche Mechanismen könnten es dem Benutzer ermöglichen, die Daten wiederherzustellen?

12. Eine Möglichkeit, zusammenhängende Belegung zu verwenden, ohne Lücken entstehen zu lassen, ist, die Platte jedes Mal zu verdichten, wenn eine Datei entfernt wurde. Da alle Dateien zusammenhängend sind, benötigt das Kopieren einer Datei einen Plattenzugriff und eine rotationsbedingte Wartezeit zum Lesen, danach folgt die Datenübertragung mit voller Geschwindigkeit. Das Zurückschreiben der Datei erfordert denselben Aufwand. Nehmen Sie an, die Zeit für den Plattenzugriff ist 5 ms, die rotationsbedingte Wartezeit beträgt 4 ms, die Transferrate beträgt 80 MB/s und die durchschnittliche Dateigröße ist 8 KB. Wie lange dauert es dann, eine Datei in den Speicher einzulesen und sie dann an anderer Position wieder auf die Platte zu schreiben? Wie lange würde es bei diesen Werten dauern, die Hälfte einer 16-GB-Festplatte zu verdichten?

13. MacOS bietet symbolische Links sowie Aliase. Ein Alias ähnelt einem symbolischen Link, speichert aber im Gegensatz zu diesem zusätzliche Metadaten über die Zieldatei (z.B. die I-Node-Nummer und die Dateigröße), sodass bei einer Verschiebung der Zieldatei innerhalb desselben Dateisystems der Zugriff auf den Alias zum Zugriff auf die Zieldatei führt, da das Dateisystem das ursprüngliche Ziel sucht und findet. Wie könnte dieses Verhalten vorteilhaft sein im Vergleich zu symbolischen Links? Inwiefern kann es Probleme verursachen?

14. In Anlehnung an die vorherige Frage: Wenn in früheren MacOS-Versionen nach der Verschiebung der Zieldatei eine andere Datei mit dem ursprünglichen Pfad des Ziels erstellt wird, findet der Alias immer noch die verschobene Zieldatei (und nicht die neue Datei mit demselben Pfad/Name). Wenn jedoch in Versionen von MacOS 10.2 oder höher die Zieldatei verschoben und eine andere am alten Ort erstellt wird, verbindet sich der Alias mit der neuen Datei. Hebt dies die Nachteile aus Ihrer Antwort auf die vorherige Frage auf? Schränkt es die von Ihnen erwähnten Vorteile ein?

15. Einige digitale Verbrauchergeräte müssen Daten speichern, zum Beispiel als Dateien. Nennen Sie ein modernes Gerät, das Dateien speichern muss und für das eine zusammenhängende Belegung sinnvoll wäre.

16. Betrachten Sie noch einmal den I-Node der *Abbildung 4.15*. Wenn er zehn direkte Adressen von 4 Byte Länge hätte und alle Plattenblöcke 1.024 KB lang wären, wie groß wäre dann die größtmögliche Datei?

17. Die Datensätze der Studenten eines bestimmten Seminars werden jeweils in einer Datei gespeichert. Auf die Datensätze kann zur Aktualisierung wahlfrei zugegriffen werden. Wir nehmen an, dass jeder Studentendatensatz eine festgelegte Größe hat. Welches der drei Belegungungsschemata (zusammenhängend, durch verkettete Listen, über Tabellen/Indizes) passt am besten?

18. Gegeben sei eine Datei, deren Größe während ihrer Lebensdauer zwischen 4 KB und 4 MB schwankt. Welches der Belegungsschemata (zusammenhängend, durch verkettete Listen, über Tabellen/Indizes) passt am besten?

19. Es wurde angedeutet, dass die Effizienz verbessert und Plattenplatz gespart werden kann, wenn die Daten einer kurzen Datei direkt im I-Node abgespeichert würden. Wie viele Bytes könnten im I-Node der *Abbildung 4.15* gespeichert werden?

20. Zwei Informatikstudentinnen, Caroline und Leonore, diskutieren über I-Nodes. Caroline behauptet, da Speicher so schnell und billig geworden ist, dass es einfacher und schneller sei, beim Öffnen einer Datei eine neue Kopie des I-Nodes in die I-Node-Tabelle zu laden, als die Tabelle danach zu durchsuchen. Leonore ist anderer Meinung. Wer hat Recht?

21. Nennen Sie einen Vorteil von harten Links gegenüber symbolischen Links und umgekehrt.

22. Erklären Sie, wie sich harte Links und weiche Links hinsichtlich der I-Node-Zuordnung unterscheiden.

23. Gegeben sei eine 4-TB-Platte, die 4-KB-Blöcke verwendet und die Methode der Freibereichslisten einsetzt. Wie viele Blockadressen können in einem Block gespeichert werden?

24. Über den freien Speicher kann mittels einer Freibereichsliste oder einer Bitmap gewacht werden. Plattenadressen benötigen D Bit. Geben Sie für eine Platte mit B Blöcken, von denen F frei sind, die Bedingung an, unter der die Freibereichsliste weniger Speicher verbraucht als die Bitmap. Wenn D den Wert 16 Bit hat, wie viel Prozent des Plattenplatzes muss dann frei sein?

Copyright