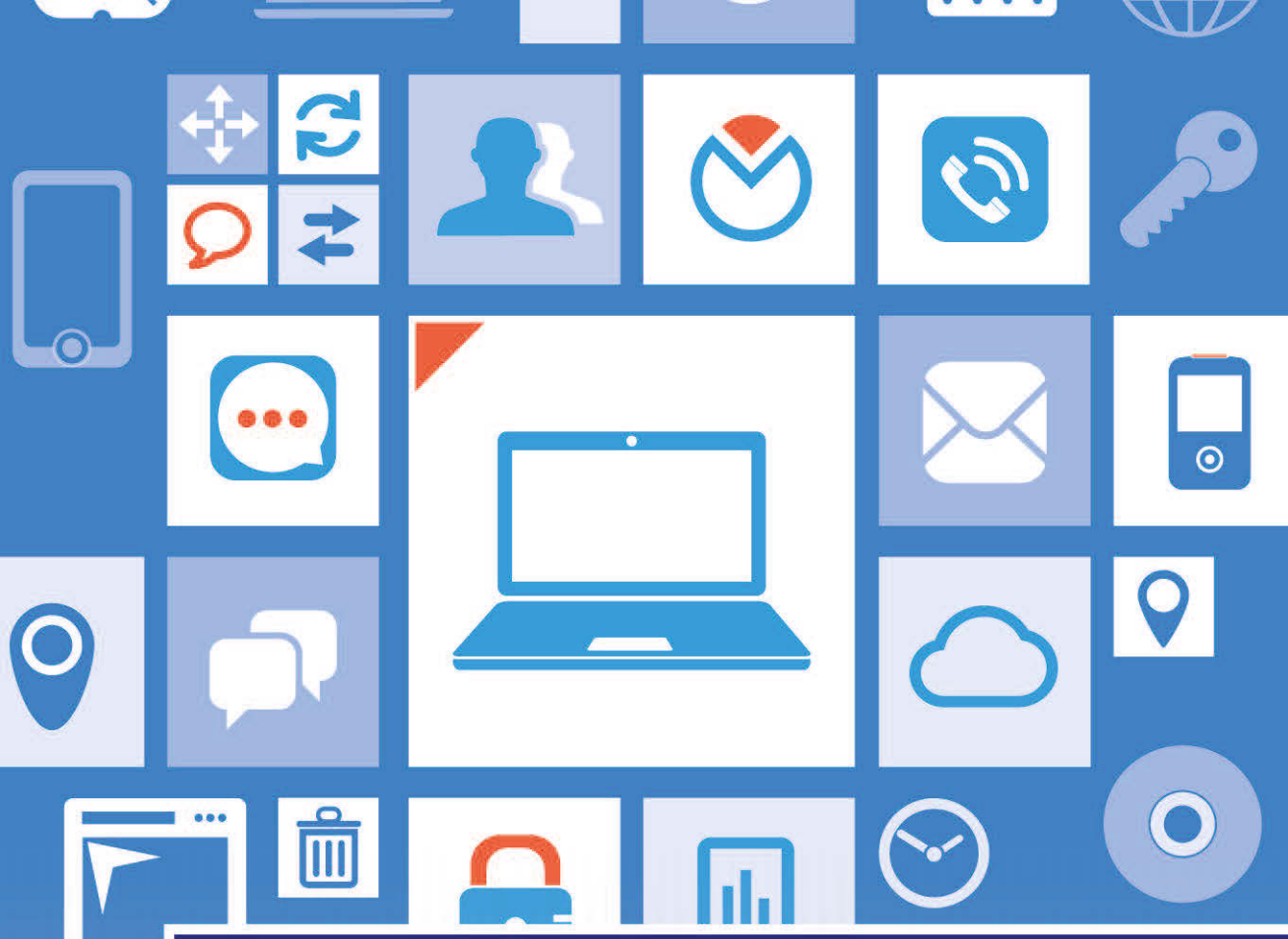

Moderne Betriebssysteme

4., aktualisierte Auflage

Andrew S. Tanenbaum
Herbert Bos

EXTRAS ONLINE

ALWAYS LEARNING

PEARSON

Methode nicht. Im Gegenteil, sie schadet dann sogar, da die Bandbreite der Platte durch das Lesen unnützer Blöcke und das Entfernen potenziell nützlicher Blöcke geringer wird. (Möglicherweise wird die Bandbreite sogar noch geringer, wenn die veränderten Blöcke auf die Platte zurückgeschrieben werden.) Um festzustellen, ob es den Aufwand wert ist, vorausschauend zu lesen, kann das Dateisystem die Zugriffsmuster jeder geöffneten Datei beobachten. Zum Beispiel könnte jeder Datei ein Bit zugeordnet werden, das darüber Auskunft gibt, ob sie sich im „sequenziellen Zugriffsmodus“ oder im „wahlfreien Zugriffsmodus“ befindet. Nach dem Motto „Im Zweifel für den Angeklagten“ wird die Datei anfänglich in den sequenziellen Modus versetzt. Wann immer jedoch ein Plattenzugriff erfolgt, wird das Bit zurückgesetzt. Beginnen nun wieder sequenzielle Zugriffe, wird das Bit erneut gesetzt. Mit dieser Methode kann das Dateisystem vernünftige Annahmen darüber machen, ob vorausschauendes Lesen günstig ist oder nicht. Rät es hin und wieder falsch, ist das nicht gleich eine Katastrophe, es wird nur etwas von der Bandbreite der Platte verschwendet.

Reduzierung der Bewegung des Plattenarms

Caching und vorausschauendes Lesen sind nicht die einzigen Möglichkeiten zur Erhöhung der Performanz eines Dateisystems. Eine weitere wichtige Technik besteht darin, die Anzahl der Plattenarmbewegungen zu vermindern, indem man Blöcke, auf die wahrscheinlich der Reihe nach zugegriffen wird, nahe nebeneinander, vorzugsweise auf demselben Zylinder platziert. Wenn die Ausgabedatei geschrieben wird, so muss das Dateisystem die Blöcke bei Bedarf einen nach dem anderen belegen. Wenn die freien Blöcke in einer Bitmap verwaltet werden und die gesamte Bitmap im Arbeitsspeicher ist, dann ist es nicht allzu schwer, einen freien Block so zu wählen, dass dieser so nah wie möglich zum vorhergehenden Block liegt. Bei einer Freibereichsliste, die zum Teil auf der Platte steht, ist es viel schwerer, nahe zusammenliegende Blöcke zu finden.

Doch auch mit einer Freibereichsliste kann man Blöcke gruppieren. Der Trick dabei ist, den Speicherplatz nicht in Blöcken, sondern in Gruppen von aufeinanderfolgenden Blöcken zu verwalten. Wenn alle Sektoren aus 512 Byte bestehen, so könnte das System 1-KB-Blöcke (zwei Sektoren) verwenden, den Speicherplatz jedoch in Einheiten von zwei Blöcken (vier Sektoren) belegen. Das ist nicht dasselbe, wie 2-KB-Blöcke zu verwenden, da der Cache immer noch 1 KB große Blöcke benutzt und der Datentransfer auch noch 1 KB beträgt. Das sequenzielle Lesen einer Datei auf einem ansonsten untätigen System würde die Anzahl der Plattenzugriffe um den Faktor 2 vermindern – eine beträchtliche Verbesserung der Performanz. Eine Abwandlung dieser Optimierung ist die Einbeziehung der rotierenden Positionierung. Wenn Blöcke belegt werden, dann versucht das System, die zusammenhängenden Blöcke einer Datei auf demselben Zylinder abzulegen.

Ein anderer Engpass bezüglich der Performanz entsteht in Systemen, die I-Nodes oder etwas Ähnliches benutzen, denn der Lesezugriff selbst auf eine kleine Datei erfordert zwei Plattenzugriffe: einen auf den I-Node und einen auf den Block. Die übliche Position der I-Nodes ist in ▶ *Abbildung 4.29a* dargestellt. Hier liegen alle I-Nodes in der

Nähe des Plattenanfangs, somit beträgt die durchschnittliche Entfernung zwischen dem I-Node und seinem Block die Hälfte der Zylinderanzahl, was lange Zugriffszeiten zur Folge hat.

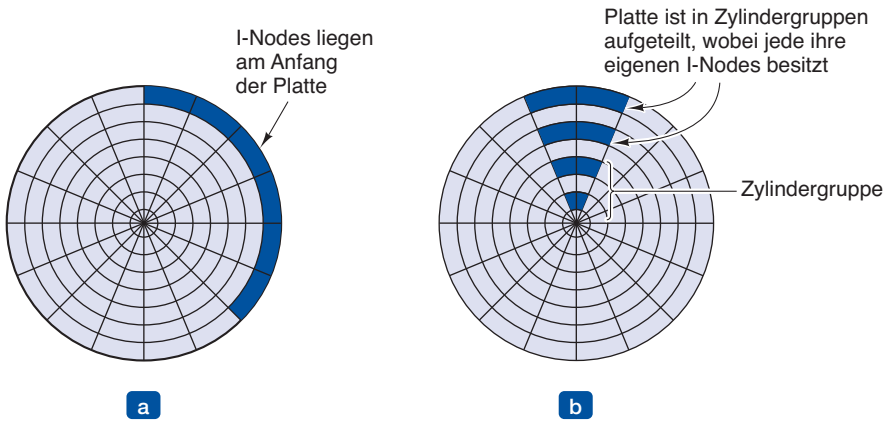

Abbildung 4.29: (a) Die Platzierung der I-Nodes am Anfang der Platte. (b) Die Platte aufgeteilt in Zylindergruppen mit jeweils eigenen Blöcken und I-Nodes.

Eine einfache Verbesserung besteht darin, die I-Nodes in der Mitte der Platte anstatt am Anfang unterzubringen, was die durchschnittliche Suchzeit zwischen dem I-Node und dem ersten Block um den Faktor zwei reduziert. Ein anderes Vorgehen ist in ▶ *Abbildung 4.29b* dargestellt. Hier wird die Platte in Zylindergruppen aufgeteilt, jede mit ihren eigenen I-Nodes, Blöcken und Freibereichslisten (McKusick et al., 1984). Wird eine Datei erzeugt, kann zwar prinzipiell jeder I-Node verwendet werden. Es wird jedoch versucht, einen Block in der gleichen Zylindergruppe zu finden, in der auch der I-Node steht. Falls keiner verfügbar ist, dann wird ein Block in einer benachbarten Gruppe verwendet.

Natürlich sind Plattenarmbewegung und Rotationszeit nur relevant, wenn die Platte bewegliche Teile besitzt. Immer mehr Rechner sind heute mit **Solid-State-Drives** (**SSD**) ausgestattet, die keinerlei solche beweglichen Elemente haben. Für diese Speichermedien, die auf der gleichen Technologie wie Flash-Speicher aufgebaut werden, sind wahlfreie Zugriffe ebenso schnell wie sequenzielle – und viele der Probleme herkömmlicher Platten verschwinden. Doch leider tauchen neue Probleme auf: zum Beispiel haben SSDs eigentümliche Eigenschaften, wenn es um das Lesen, Schreiben und Löschen geht. Insbesondere kann auf jedem Block nur eine begrenzte Anzahl von Schreibvorgängen ausgeführt werden. Es muss also besondere Sorgfalt an den Tag gelegt werden, um die Abnutzung über der Platte gleichmäßig zu verteilen.

4.4.5 Defragmentierung von Plattenspeicher

Wenn das Betriebssystem anfänglich installiert wird, sind die benötigten Programme und Dateien am Anfang der Platte nacheinander in direkter Folge abgelegt. Der gesamte freie Plattenbereich bildet eine einzige zusammenhängende Einheit, die hinter den ins-

tallierten Dateien folgt. Aber mit der Zeit werden Dateien erzeugt und gelöscht und typischerweise wird die Platte stark fragmentiert, wodurch überall Dateien und Lücken entstehen. Folglich können die Blöcke, die für die Erzeugung einer neuen Datei benötigt werden, über die ganze Platte verteilt sein, was zu einer verringerten Performanz führt.

Zur Wiederherstellung der Performanz können die Dateien hin und her geschoben werden, sodass sie wieder in aneinandergrenzenden Bereichen liegen und sich damit der gesamte freie Speicherplatz (oder zumindest der größte Teil davon) in einem einzelnen oder mehreren großen zusammenhängenden Teilen der Platte befindet. Windows besitzt für genau diesen Zweck ein Programm namens *defrag*. Windows-Benutzer sollten es regelmäßig laufen lassen, außer auf SSDs.

Defragmentierung funktioniert besser bei Dateisystemen, bei denen sehr viel Speicherplatz in einem zusammenhängenden Bereich am Ende der Partition frei ist. Damit ist das Defragmentierungsprogramm in der Lage, eine zerstückelte Datei vom Anfang der Partition auszuwählen und alle Blöcke dieser Datei auf den freien Platz zu kopieren. Dadurch wird ein zusammenhängender Speicherblock am Anfang der Partition frei, in dem die Originaldatei oder eine andere Datei an einem Stück untergebracht werden kann. Dieser Prozess kann dann mit dem nächsten freien Stück Speicherplatz wiederholt werden und immer so weiter.

Einige Dateien, darunter die Paging-Datei, die Hibernation-Datei und das Journaling-Log, können nicht verschoben werden, da der Verwaltungsaufwand hier viel größer als der Nutzen wäre. In einigen Systemen sind dies ohnehin zusammenhängende Bereiche mit fester Größe, bei denen also gar keine Defragmentierung nötig ist. Dieser Mangel an Mobilität stellt lediglich dann ein Problem dar, wenn sich die Dateien in der Nähe des Partitionsendes befinden und der Benutzer die Partitionsgröße reduzieren möchte. Die einzige Möglichkeit ist dann, all diese Dateien zu löschen, die Größe der Partition zu verändern und danach die Dateien neu zu erzeugen.

Die Dateisysteme unter Linux (speziell ext2 und ext3) leiden im Allgemeinen weniger unter Defragmentierung als Windows-Systeme. Dies liegt an der Art und Weise, wie hier Plattenblöcke ausgewählt werden, sodass eine manuelle Defragmentierung selten nötig ist. Auch bei SSDs gibt es das Problem der Fragmentierung nicht. Im Gegenteil, das Defragmentieren einer SSD ist sogar kontraproduktiv: nicht nur, dass es keinen Performanzgewinn gibt, sondern SSDs verschleißen dadurch auch – Defragmentierung verkürzt also lediglich ihre Lebensdauer.

4.5 Beispiele von Dateisystemen

In den folgenden Abschnitten werden wir verschiedene Dateisysteme beispielhaft betrachten. Die Spannweite reicht dabei von den recht einfachen bis zu den hochkomplizierten Systemen. Da die modernen UNIX-Dateisysteme und das Dateisystem von Windows 8 im Kapitel über UNIX (*Kapitel 10*) und Windows 8 (*Kapitel 11*) behandelt werden, wollen wir hier nicht auf diese Systeme eingehen. Ihre Vorgänger werden wir jedoch untersuchen.

4.5.1 Das MS-DOS-Dateisystem

Das Dateisystem, mit dem der erste IBM-PC herauskam, war das MS-DOS-Dateisystem. Es war durchgängig bis Windows 98 und Windows ME das Hauptdateisystem und wird immer noch von Windows 2000, Windows XP und Windows Vista unterstützt, obwohl es auf neuen PCs (außer für Disketten) nicht länger Standard ist. Dennoch ist dieses Dateisystem zusammen mit einer Erweiterung (FAT-32) weit verbreitet für viele eingebettete Systeme. Die meisten Digitalkameras benutzen es. Viele MP3-Player benutzen es ausschließlich. Es ist zudem das typische Dateisystem auf USB-Sticks. Der populäre iPod von Apple benutzt es als Standarddateisystem,[1] obwohl begabte Hacker den iPod neu formatieren und ein anderes Dateisystem installieren können. Damit ist die Anzahl an elektronischen Geräten, auf denen das MS-DOS-Dateisystem eingesetzt wird, heute weitaus größer, als es in der Vergangenheit jemals war, und sicher viel größer als die Anzahl der Geräte, die das modernere Dateisystem NFTS benutzen. Schon allein aus diesem Grund lohnt es sich, es ein wenig genauer zu betrachten.

Um eine Datei lesen zu können, muss ein MS-DOS-Programm den Systemaufruf `open` starten, um ein Handle für diese Datei zu bekommen. Der `open`-Systemaufruf spezifiziert einen Pfad, der entweder absolut oder relativ zum aktuellen Arbeitsverzeichnis sein kann. Dieser Pfad wird Komponente für Komponente verfolgt, bis das Zielverzeichnis gefunden ist, das dann in den Speicher eingelesen und anschließend nach der zu öffnenden Datei durchsucht wird.

Obwohl MS-DOS-Verzeichniseinträge eine variable Länge haben, benutzen sie eine feste Länge von 32 Byte. Das Format eines MS-DOS-Verzeichniseintrags ist in ▶*Abbildung 4.30* zu sehen. Es enthält den Dateinamen, die Attribute, das Datum und 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.

1 Anm. d. Fachlektors: Das gilt aber nur, wenn das Gerät an einem Windows-PC betrieben wird. Beim Betrieb an einem Apple OS-X-Rechner wird normalerweise das OS-X-Dateisystem HFS verwendet.

Außerdem können Systemdateien nicht fälschlicherweise durch das *del*-Kommando gelöscht werden. Die Hauptkomponenten von MS-DOS haben dieses Bit gesetzt.

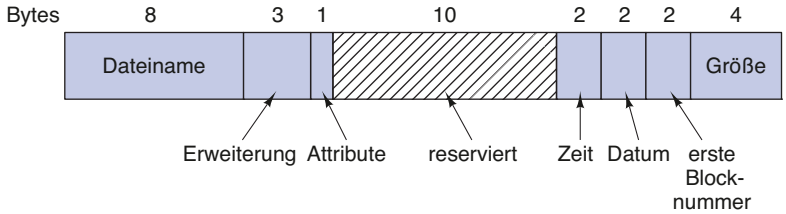

Abbildung 4.30: Der MS-DOS-Verzeichniseintrag.

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 65536 unterschiedliche Werte speichern kann (ein Tag hat 86400 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 64000 Einträgen im Speicher zu indizieren. Durch Verfolgen der Kette können alle Blöcke gefunden werden. Die Funktionsweise der FAT ist in ▶ *Abbildung 4.12* 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 OS-X-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} 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 4096 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. (32768 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 (64000 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} 2^{15} Byte groß sein, doch sind sie tatsächlich auf 2 TB (2048 GB) beschränkt. Das System verfolgt die Partitionsgrößen nämlich intern in 512 Byte großen Sektoren über eine 32-Bit-Nummer und 2^{9} 2^{32} ergibt 2 TB. Die maximale Partitionsgröße für die verschiedenen Blockgrößen und alle drei FAT-Typen ist in ▶ *Abbildung 4.31* 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		1024 MB	2 TB
32 KB		2048 MB	2 TB

Abbildung 4.31: 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 64000 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 512000 Blöcke, also muss die FAT 512000 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 4.13* benutzt. Ein Verzeichniseintrag enthält nur zwei Felder: den Dateinamen (14 Byte) und die Nummer des I-Node für diese Datei (2 Byte), siehe ▶ *Abbildung 4.32*. Diese Parameter begrenzen die Anzahl der Dateien pro Dateisystem auf 64 000.

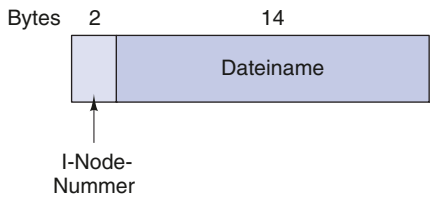

Abbildung 4.32: UNIX-V7-Dateieintrag.

Wie der I-Node aus ▶ *Abbildung 4.13* 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 4.13* 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 4.33* zu sehen.

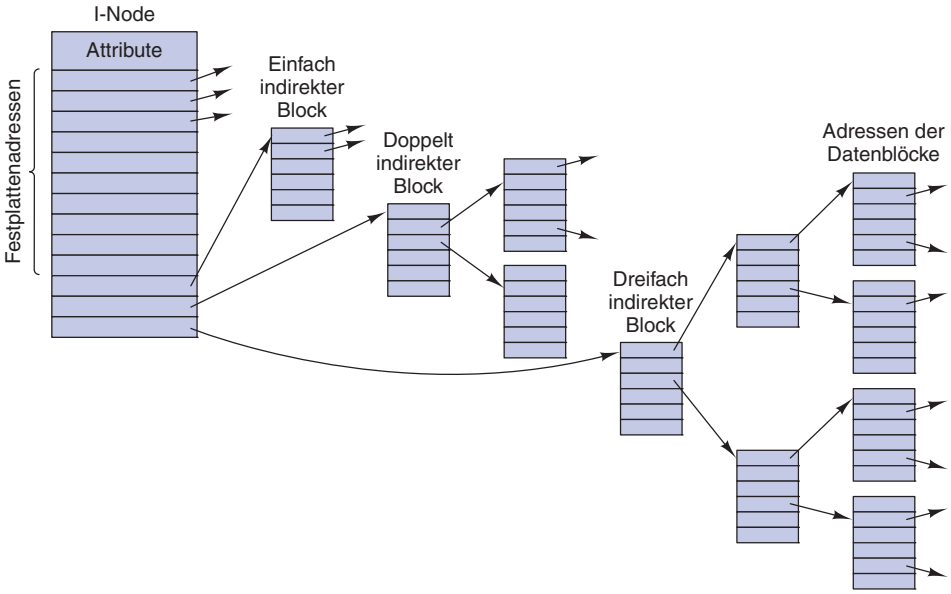

Abbildung 4.33: I-Node unter UNIX.

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 4.34* dargestellt.

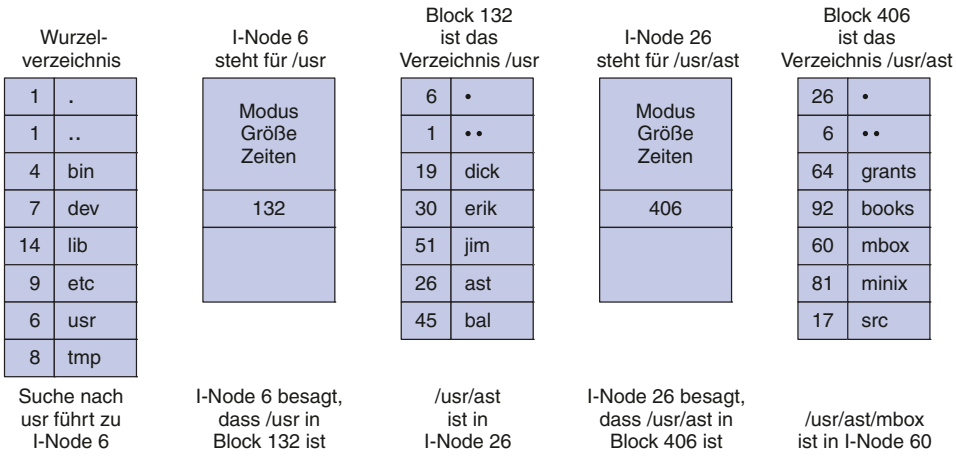

Abbildung 4.34: Schritte, um */usr/ast/mbox* zu finden.

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 *../dick/prog.c* finden soll, einfach nach .. im Arbeitsverzeichnis. Sie findet die I-Node-Nummer des Vorgängerverzeichnisses und sucht dieses nach *dick* 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.

4.5.3 CD-ROM-Dateisysteme

Als letztes Beispiel für ein Dateisystem betrachten wir die Dateisysteme auf CD-ROMs. Diese Systeme sind besonders einfach strukturiert, da sie für einmal beschreibbare Medien entwickelt wurden. Unter anderem besitzen sie keine Mittel, um über die freien Blöcke zu wachen, denn auf einer CD-ROM können nach ihrer Herstellung keine Dateien mehr gelöscht oder hinzugefügt werden. Im Folgenden werden wir einen Blick auf das Hauptdateisystem der CD-ROM werfen und auch zwei Erweiterungen dafür betrachten. Auch wenn CD-ROMs heute veraltet sind, so sind sie doch auch einfach, und die Dateisysteme, die auf DVD und Blu-ray eingesetzt werden, basieren auf dem CD-ROM-Dateisystem.

Ein paar Jahre nach dem Debüt der CD-ROM wurde die CD-R (CD Recordable) eingeführt. Anders als bei der CD-ROM ist es hier möglich, Dateien nach dem ersten Brennen hinzuzufügen, aber diese werden einfach am Ende der CD-R angehängt. Dateien werden nie wirklich gelöscht (auch wenn das Verzeichnis aktualisiert werden kann, um vorhandene Dateien zu verstecken). Durch dieses „Nur-Anhängen“ wurden die

grundsätzlichen Eigenschaften des Dateisystems nicht verändert. Insbesondere befindet sich der gesamte freie Speicherbereich weiterhin in einem zusammenhängenden Stück am Ende der CD.

Das ISO-9660-Dateisystem

Der gebräuchlichste Standard für CD-ROM-Dateisysteme wurde als internationaler Standard im Jahre 1988 unter dem Namen **ISO 9660** anerkannt. Praktisch jede CD-ROM, die sich zurzeit auf dem Markt befindet, ist zu diesem Standard kompatibel, manchmal mit den Erweiterungen, die weiter unten vorgestellt werden. Ein Ziel dieses Standards war es, jede CD-ROM auf jedem Computer, unabhängig von der benutzten Byte-Ordnung und dem jeweiligen Betriebssystem lesbar zu machen. Als Konsequenz wurde das Dateisystem in einigen Bereichen eingeschränkt, um auch für das schwächste Betriebssystem (wie MS-DOS) lesbar zu sein.

CD-ROMs besitzen keine konzentrischen Zylinder wie die magnetischen Platten, sondern stattdessen eine einzige kontinuierliche Spirale, die die Bits in einer linearen Sequenz enthält (obwohl Suchsprünge auf der Spirale möglich sind). Die Bits entlang der Spirale sind in logische Blöcke (auch logische Sektoren genannt) von 2352 Byte unterteilt. Einige davon sind für Präambeln, Fehlerkorrektur und anderes Beiwerk vorgesehen. Die Datenportion beträgt in jedem logischen Block 2048 Byte. Wenn die CDs für Musik benutzt werden, dann haben sie sogenannte Einleitungsbereiche (*leadin*) und Schlussbereiche (*leadout*) sowie Lücken zwischen den Tracks. Dies ist etwas, das es bei Daten-CDs nicht gibt. Oft wird die Position eines Blocks auf der Spirale in Minuten und Sekunden angegeben. Diese Angabe kann in eine lineare Blocknummer über den Umrechnungsfaktor von 1 s = 75 Blöcke umgewandelt werden.

ISO 9660 unterstützt Gruppen von CD-ROMs mit $2^{16}-1$ CDs pro Gruppe. Jede individuelle CD-ROM kann ebenfalls in logische Einheiten (Partitionen) unterteilt werden. Wir werden uns im Folgenden aber auf ISO 9660 für eine einzelne, nicht partitionierte CD-ROM konzentrieren.

Jede CD-ROM beginnt mit 16 Blöcken, deren Funktion nicht durch den ISO-9660-Standard definiert ist. Der Hersteller könnte diese Region nutzen, um z.B. ein Bootprogramm zur Verfügung zu stellen, mit dessen Hilfe der Computer von der CD-ROM aus hochgefahren werden kann, aber auch andere (schändliche) Verwendungszwecke sind denkbar. Als Nächstes kommt ein Block, der den **Primärvolumendeskriptor** (*primary volume descriptor*) enthält, welcher generelle Informationen über die CD-ROM enthält. Unter diesen Informationen finden sich der Systemidentifikator (32 Byte), der Volumenidentifikator (32 Byte), der Herausgeberidentifikator (128 Byte) und der Datenvorbereitungsidentifikator (128 Byte). Die Hersteller können diese Felder beliebig füllen. Zum Erhalten der Plattformunabhängigkeit muss lediglich beachtet werden, dass nur Großbuchstaben, Zahlen und eine sehr kleine Anzahl von Sonderzeichen benutzt werden.

Der Primärvolumendeskriptor enthält außerdem die Namen von drei Dateien, die eine Zusammenfassung, die Urheberrechtsbestimmungen bzw. bibliografische Informationen umfassen können. Zusätzlich sind einige Schlüsselwerte gespeichert, wie die

Copyright