Filesysteme

Hans Jürgen Ohlbach

22. November 2017

Keywords: Filesysteme

Inhaltsverzeichnis

1 Einleitung **2**

2 Partitionen auf einer Festplatte **2**

3 Cluster **3**

4 Clusterlisten: FAT und iNodes **4**

4.1 File Allocation Table (FAT) 4

4.2 Index-Knoten (iNodes) 6

5 Dateinamen **7**

6 Verzeichnisse (Ordner, Directories) **8**

7 Hard- und Softlinks **8**

8 Attribute **9**

9 Journaling **9**

10 Network File System (NFS) **10**

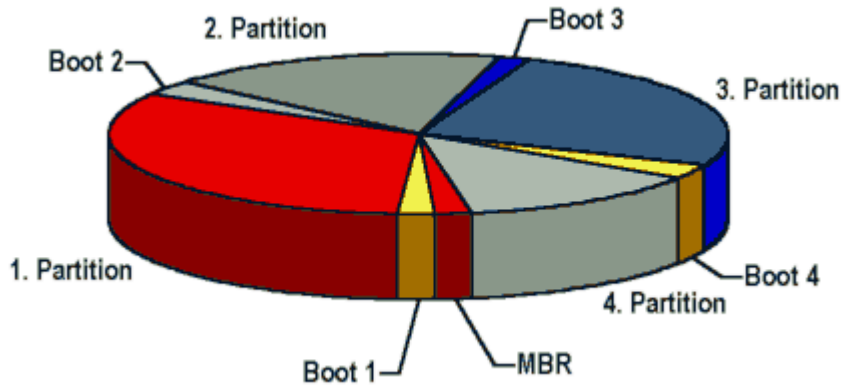
1 Einleitung

Eine ganz wesentliche Komponente eines Betriebssystems ist sein Filesystem. Das Filesystem bestimmt, wie genau die vielen Dateien[1] auf der Festplatte abgelegt werden können, und wie der Computer darauf zugreifen kann. Es gibt allerdings nicht *das Filesystem*, sondern einen ganzen Zoo davon: FAT-12, FAT-16, FAT-32, NTFS, Ext2, Ext3, Ext4, Btrfs, XFS, F2FS, ReiserFS, um einige zu nennen. Manche Filesysteme sind optimiert für CD-ROMs, (ISO 9660, Joliet, Rock Ridge), für Flash Speicher und SSDs (exFAT), für Magnetbänder, usw. Es würde den Rahmen dieses Miniskript komplett sprengen, um auf jedes der Systeme im Detail einzugehen. Daher konzentrieren wir uns in erster Linie auf Techniken und Eigenschaften, die einzelne von den Systemen haben.

2 Partitionen auf einer Festplatte

Die magnetische Festplatte ist immer noch das primäre Speichermedium, auch wenn sie allmählich von den SSDs verdrängt wird. Ihre logische Dateistruktur ist folgendermaßen aufgebaut: Der erste Sektor ist der *Master Boot Record* (MBR). In ihm wird u.A. die Startroutine, der *Bootloader*, gespeichert, mit der der Start des Betriebssystem angestoßen wird. Es ist auch möglich, dass der Bootloader woanders liegt, und im MBR nur ein Verweis auf den eigentlichen Bootloader steht. Falls die Platte mehrere Betriebssysteme enthält, erlaubt der Bootloader, eines davon auszuwählen. Die eigentliche Startroutine für das jeweilige Betriebssystem steht am Beginn der jeweiligen Partition.

Anschließend kommen 1-4 *primäre Partitionen*. Eine Partition kann ein komplettes Betriebssystem, oder auch nur der Swapspace für die Speicherverwaltung enthalten. Die Informationen über die Partitionen sind ebenfalls im MBR als *Partitionstabelle* gespeichert. Es gibt besondere Programme, mit denen man die Partitionstabelle ansehen und manipulieren kann (in Linux ist das z.B. fdisk).



Bei den heutigen riesengroßen Platten kann es nützlich sein, mehr als 4 Partitionen zu haben. Dafür lässt sich eine *erweiterte Partition* einrichten, die wiederum in mehrere *logische Partitionen* unterteilt werden kann.

[1] Die Namen „File“ und „Datei“ sind synonym.

In den Windows Betriebssystemen werden die Partitionen mit den Buchstaben C bis Z angesprochen. In Unix und Linux werden die Festplatten selbst mit hda, hdb usw. angesprochen. Die primären Partitionen auf der ersten Festplatte werden mit hda1, hda2, hda3, hda4 angesprochen, die erweiterte Partition mit hda5, und die logischen Partitionen mit hda6 usw. Für die anderen Festplatten ist es analog.

Der ursprüngliche Grund für die Partitionierung von Festplatten war, dass die Filesysteme eine Adressierung hatten, die die Maximalgröße der Partition beschränken. Z.B. war die Partitionsgröße eines der ersten Filesysteme, FAT-16, auf 2 GByte beschränkt.

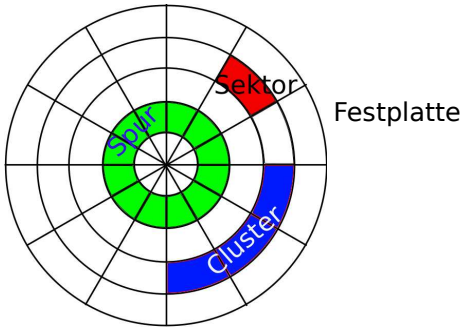
Heutzutage gibt es noch andere Gründe für eine Partitionierung:

- man möchte mehrere Betriebssysteme bootfähig speichern,
- man möchte unterschiedliche Filesysteme parallel zur Verfügung haben,
- man braucht einen speziellen Swapspace für die virtuelle Speicherverwaltung,
- man möchte große Datenmengen separat speichern.
- ...

3 Cluster

Filesysteme dienen natürlich dazu, Dateien abzuspeichern. Die Größe der Dateien kann aber extrem abweichen, von ein paar wenigen Bytes, bis zu GByte großen Videofiles. Alle diese Dateien am Stück auf einer Festplatte abzuspeichern bringt riesige Fragmentierungsprobleme mit sich. Daher hat man schon früh entschieden, Dateien generell in sog. *Cluster* aufzuteilen, und die Cluster auf freie Stellen auf der Platte zu verteilen.

Jede Festplatte ist in *Spuren* und *Sektoren* aufgeteilt. Ein Sektor ist die kleinste Einheit, die am Stück gelesen oder geschrieben werden kann. Typische Sektorgrößen sind 1-4 KByte. Man hätte daher ein Cluster genau einen Sektor groß sein lassen können. Bei einer Festplatte von 1 TByte und einer Sektorgröße von 1 KByte hätte man dann ca. eine Milliarde Cluster, die man alle durchnummerieren müsste. Dafür braucht man 30 Bit Adressbreite. Viele Filesysteme ließen das nicht zu. Um damit flexibler zu sein, wurden *Cluster* als eine kleine Menge von aufeinanderfolgenden Sektoren eingeführt. Z.B. könnte ein Cluster aus 4 Sektoren bestehen. Die Cluster sind alle gleich groß und durchnummeriert, so dass man mit der Clusternummer auch direkt den Sektor bestimmen kann, wo das Cluster anfängt.



Eine Datei wird also in eine Menge von Clustern aufgeteilt. Die Cluster einer Datei können beliebig auf der Platte verteilt sein.

Die Clustergröße muss bei der Einrichtung des Filesystems festgelegt werden, und kann dann nicht mehr verändert werden. Folgende Gesichtspunkte sind dabei zu beachten:

- Ist die Clustergröße klein, dann werden große Dateien in sehr viele Cluster aufgeteilt. Eine ganze Datei zu laden bedeutet dann, dass der Lesekopf auf der Platte viel hin- und herspringen muss. Das kostet Zeit.
- Ist die Clustergröße dagegen zu groß, kommt es zu interner Fragmentierung. Kleine Dateien nutzen nicht ein ganzes Cluster aus, und lassen dann Lücken.

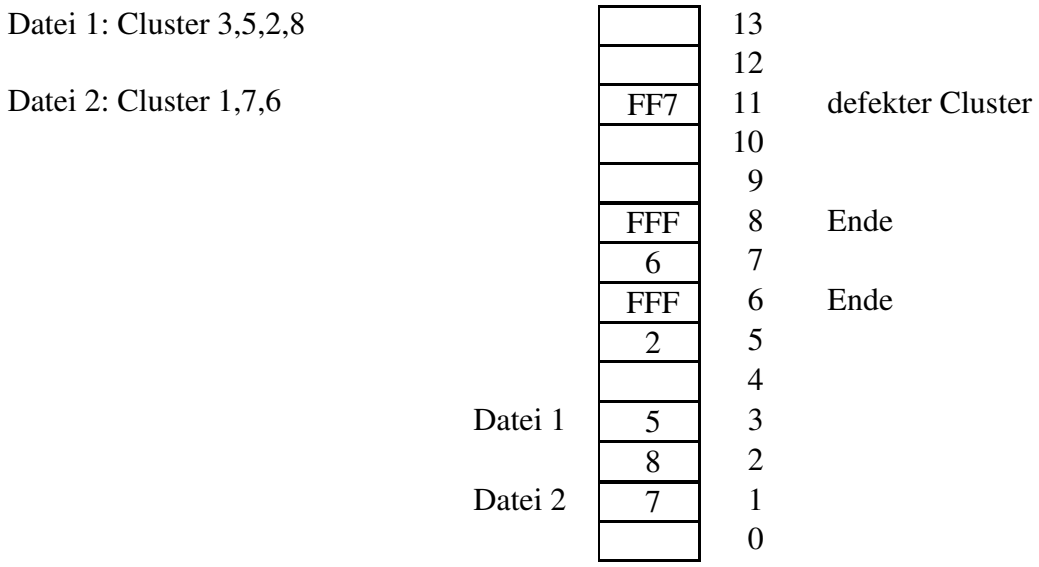
Das Lesen und Schreiben von Dateien geht am schnellsten, wenn alle Cluster einer Datei hintereinander liegen. Es gibt sog. *Defragmentierungsprogramme*, die versuchen, die Cluster so umzuordnen, dass die Cluster einer Datei hintereinander liegen. Filesysteme wie ext4 versuchen allerdings, Fragmentierung zu vermeiden, indem sie Dateien nicht sequentiell auf der Festplatte, sondern gleichmäßig platziert speichern. Dies erlaubt meistens freie Expansion bei Wachstum der Datei über ihre anfänglichen Grenzen hinaus.

4 Clusterlisten: FAT und iNodes

Eine Datei wird also in einer Menge von Clustern abgespeichert. Um all diese Cluster wiederzufinden, muss die Liste der Clusternummern existieren und in einer passenden Form abgelegt werden. Es gibt dazu verschiedene Methoden.

4.1 File Allocation Table (FAT)

Die FAT erlaubt, die Clusternummern in einem Array abzuspeichern. Diese Technik wurde in den ersten Filesystemen von DOS und Windows verwendet. An folgendem Beispiel sieht man, wie das im Prinzip funktioniert.



Für eine Datei braucht man nur die erste Clusternummer. Bei Datei 1 oben ist das die Nummer 3. In der FAT-Zelle Nummer 3 steht die 5, also ist der Anfang der Liste (3,5). In der FAT-Zelle 5 steht die 2, also ist die Liste jetzt (3,5,2). In der FAT-Zelle 2 steht die 8. Damit ist die Liste (3,5,2,8). In der Zelle 8 steht die Hex-Zahl FFF als Endemarkierung.

Mit der ersten Clusternummer der Datei kann man also mit ein paar Arrayzugriffen die ganze Clusterliste aufsammeln. Die FAT hat sogar ein Markierung für defekte Cluster: FF7. Wenn eine Datei gelöscht wird, löscht man einfach die entsprechenden Einträge in der FAT.

Es gibt die Filesysteme FAT-12, FAT-16 und FAT-32. Die Nummern beziehen sich auf die Adressbreite der Cluster:

FAT-12:	$2^{12} = 4.096$	Cluster	(4078 nutzbare Cluster)
FAT-16:	$2^{16} = 65.517$	Cluster	
FAT-32:	$2^{28} = 268.435.456$	Cluster	(nicht alle 32 Bits werden genutzt)

Da eine Datei mindestens einen Cluster beansprucht, gibt das auch die maximale Anzahl von Dateien wieder.

Über die Größe der Cluster kann man natürlich die Gesamtgröße des Filesystems beeinflussen. Im Lauf der Entwicklung hat man die Clustergröße auch hoch gesetzt, bei FAT-16 bis 64 KByte, und bei FAT-32 bis 32 Kbyte. Damit ergibt sich eine maximale Größe des Filesystems von 4 GByte bei FAT-16 und 32 GByte bei FAT-32, also viel zu klein für moderne Festplatten.

Microsoft hat 2006 das Filesystem exFAT eingeführt, in erster Linie aber für Flash-Speicher. Hier ist die Clustergröße bis 128 KByte und das Filesystem kann bis 256 TByte umfassen.

Der Aufbau eines FAT-Filesystems in einer Partition sieht jetzt folgendermaßen aus:

Bootsektor	reservierte Sektoren	FAT 1	FAT 2	root-Verzeichnis	Datenbereich

Im Bootsektor steht der Maschinencode zum Laden des Betriebssystems, sowie Information über die Struktur des Filesystems. In den reservierten Sektoren kann man z.B. einen Bootmanager unterbrin-

gen. Aus Sicherheitsgründen gibt es zwei Kopien der FAT. Das root-Verzeichnis enthält die Metadaten für die Dateien.

Die Zugriffe über die FAT sind sehr effizient, aber nur dann wenn die FAT selbst beim Booten in den Arbeitsspeicher geladen wird. Je größer die Partitionen wurden, um so mehr Speicherplatz war dafür nötig.

4.2 Index-Knoten (iNodes)

Eine andere Art der Speicherung von Listen von Clusternummern sind die *Index Knoten* oder *iNodes*. Ein iNode ist einfach eine genormt große Liste von Clusternummern, z.B. 4 Clusternummern. Wenn eine Datei kleiner ist, bleiben die letzten Element der Liste leer (einfach indirekt). Wenn eine Datei größer ist, dann zeigt das letzte Element der Liste auf einen weiteren iNode, in dem weitere Clusternummern stehen (zweifach indirekt). Reicht das immer noch nicht, dann gibt es noch einen Verweis auf einen weiteren iNode (dreifach indirekt), usw.

Ein Beipiel könnte so aussehen:

		iNodes			
Datei 3	4	20	18	3	
	3	5	19	2	
	2	1	11	7	3
Datei 2	1	16	15	13	2
Datei 1	0	4			

Im Beispiel gibt es 5 iNodes mit den Nummern 0 ... 4. Die letzte Spalte enthält die Verweise auf weitere iNodes.

iNode 0 beschreibt Datei 1 mit Clusternummer 4.
iNode 1 beschreibt Datei 2 mit den Clusternummern 16,15,13,1,11,7,5,19,2 (zweifach indirekt).
iNode 4 beschreibt Datei 3 mit den Clusternummern 20,18,3.

Ein Vorteil der iNodes ist, dass nur jeweils die iNodes einer Datei oder eines Verzeichnisses in den Arbeitsspeicher geladen werden müssen. Im Vergleich zur FAT spart das viel Platz.

Die iNodes haben eine feste Größe, damit man mit der iNode-Nummer sofort an den iNode selbst springen kann. Hätten sie variable Größe, müsste man suchen.

iNodes werden insbesondere in den Filesystemen ext2, ext3 und ext4 eingesetzt.

5 Dateinamen

Ein typischer Dateiname besteht aus dem eigentlichen Namen und optional einem oder mehreren mit Punkt getrennten *Suffixen*. Der Dateiname `Filesysteme.pdf`, z.B. hat als eigentlichen Namen `Filesysteme`, und als Suffix `pdf`. Die Suffixe sollen Auskunft über den Typ der Datei geben.

Einige bekannte Suffixe, die sich über die Zeit eingebürgert haben, sind:

Suffix	Bedeutung
bak	Backup Datei
c	C-Quelldatei (Programmiersprache C)
cpp	C++-Quelldatei
html	HTML-Datei (früher auch htm)
jgp	Bilddatei
dll	Dynamically Loadable Library
pdf	Portable Document Format
ps	Postscript Format
o	Übersetzte Objektdatei
gif	Graphical Interchange Format
tex	Eingabedatei des TEX-Textsystems
txt	Gewöhnliche Textdatei
gz	komprimierte Datei

Auch mehrere Suffixe sind erlaubt. `Brief.txt.gz` bezeichnet eine komprimierte Textdatei.

Schon die allerersten Filesysteme brauchten Dateinamen. Dort waren aber die Ressourcen begrenzt. Daher hat man auch die Länge der Dateinamen begrenzt. In den ersten DOS-Systemen von Microsoft war das Namensformat 8.3, d.h. 8 Buchstaben für den eigentlichen Namen und 3 Buchstaben für den Suffix. Bsp. `Institut.htm`. Daher kommt z.B., dass manche HTML-Dateien immer noch den Suffix htm haben, statt html.

Erst ab Windows 95 waren längere Dateinamen erlaubt, aber auch nur durch einen Trick, mit dem Nebeneffekt, dass die Dateien zwei Dateinamen hatten, einen kurzen und einen langen. Dieser kurze Dateiname wird tatsächlich immer noch gespeichert, man kann ihn für Dateien in einem Verzeichnis in Windows mit dem Befehl `dir /x` aufrufen. Für das Verzeichnis `Documents` z.B. ist er `DOCUME~1`.

Inzwischen sind viele der Beschränkungen soweit gelockert, dass sie kaum noch einschränkend sind. Die Länge der Dateinamen ist aber immer noch beschränkt. Insbesondere ist in Windows die gesamte Länge des Pfadnamens zu einer Datei beschränkt.

6 Verzeichnisse (Ordner, Directories)

Verzeichnisse sind nichts anderes als kleine Dateien mit den Informationen über die Dateien und Unterverzeichnisse, die sie enthalten.

Die Komponenten sind also:

- Kennung als Verzeichnis,
- Name des Verzeichnisses,
- Attribute des Verzeichnisses,
- Liste der iNode-Nummern bzw. FAT-Nummern der Dateien und der Unterverzeichnisse.

Da Verzeichnisse einfach Dateien sind, haben sie selbst eine iNode- bzw. FAT-Nummer, und können daher wie die anderen Dateien behandelt werden.

Verzeichnisse sind oft sehr kleine Dateien, die in einem Cluster nur wenig Platz brauchen. Hat man viele davon, dann gibt es auch viel interne Fragmentierung. In manchen Filesystemen trägt man dem Rechnung, indem man mehrere dieser Dateien in ein Cluster packt.

7 Hard- und Softlinks

Manchmal ist es nützlich, wenn man in verschiedenen Verzeichnissen *dieselbe* Datei oder *dasselbe* Unterverzeichnis hat. Dann muss man nicht das Verzeichnis wechseln, um an die Datei zu kommen. Um das zu lösen, gibt es zwei Methoden:

Hardlinks: Da Dateien und Verzeichnisse eindeutig durch ihre iNode-Nummer oder FAT-Nummer gekennzeichnet sind, kann man diese Nummern einfach in mehreren Verzeichnissen verwenden.

In den Dateien, die die Verzeichnisse repräsentieren, hat man in der Liste der iNode-Nummern bzw. FAT-Nummern dann dieselbe Nummer.

Ein Problem ergibt sich zunächst, wenn man so eine Datei löscht, und seine Nummer nur aus *einem* Verzeichnis entfernt. Im anderen Verzeichnis ist die Nummer dann noch drin, führt aber ins Leere. Um das zu verhindern, werden *Refenrenzzähler* mitgeführt. D.h. die verlinkte Datei merkt sich, wieviele Hardlinks auf sie zeigen. Erst wenn der Zähler auf 0 ist, wird die Datei wirklich gelöscht.

Softlinks: Ein Softlink ist einfach eine Datei, in der als einziges ein Dateiname oder auch eine URL steht. Der Zugriff auf die verlinkte Datei ist jetzt indirekt. Der eigentliche Name muss zuerst ausgelesen werden, und dann kann auf die eigentliche Datei zugegriffen werden.

Ein großer Vorteil ist jedoch, dass dort nicht nur ein Dateiname stehen kann, sondern eine beliebige URL, die auf eine Ressource im Internet zeigt, z.B. auf einen HTML-File auf einem ganz anderen Computer. Auf diesen kann man dann auch zugreifen. Das ginge mit Hardlinks nicht.

Ein weiterer Vorteil ist, dass man jeden Softlink auf eine Datei mit eigenen Zugriffsrechten versehen kann. Damit kann man z.B. erreichen, dass ein Benutzer von seinem Verzeichnis aus die eigentliche Datei nur lesen, ein anderer aber von seinem Verzeichnis aus die eigentliche Datei lesen und beschreiben kann.

Ein Nachteil ist dagegen, dass die Technik mit dem Referenzzähler nicht mehr funktioniert. Da es im ganzen Internet Softlinks auf eine Datei geben kann, ist es unmöglich festzustellen, wieviele das sind. Es kann also passieren, dass ein Softlink noch auf eine schon gelöschte Datei zeigt (engl. dead link).

In Windows werden Softlinks als sog. *Verknüpfungen* erstellt. In Linux und Unix gibt es das Kommando `ln`, mit dem man Hard- und Softlinks erstellen kann.

8 Attribute

Neben ihrem Namen haben Dateien und Verzeichnisse eine Vielzahl von anderen *Attributen*. Welches das konkret sind, hängt vom Filesystem ab.

Beispiele sind:

- Länge der Datei
- Typ: binär, ASCII, UTF-8
- Ersteller der Datei
- Erstellungszeit
- Zeit des letzten Zugriffs
- Zeit der letzten Änderung
- Eigentümer der Datei
- Passwort für den Zugriff auf die Datei
- Read-Only Flag, 0: lesen-schreiben, 1: nur lesen
- Hidden Flag, 0: sichtbar, 1: unsichtbar
- Lock Flag, 0: nicht gesperrt, 1: gesperrt
- Temporary Flag, 0: normal, 1: Datei nach Prozessende löschen
- ein Schlüssel für verschlüsselte Dateien.

9 Journaling

Dass ein Computer abstürzt ist sicher jedem schon mal passiert. Wenn das mitten in einer Operation des Filesystems passiert, kann es zu inkonsistenten Zuständen im Filesystem kommen. Solche inkonsistenten Zustände zu finden und zu reparieren, ist komplex und zeitaufwendig.

Um das zu beschleunigen, hat man das Verfahren des *Journalings* eingeführt. Dabei wird eine Log-Datei mitgeführt, in dem jede Operation im Filesystem vermerkt wird. Vor Beginn der Operation wird eingetragen, welche Operation durchgeführt werden soll. Wenn diese erledigt wurde, wird das entweder auch vermerkt, oder der Eintrag wird wieder gelöscht.

Stürzt der Computer mitten in der Operation ab, dann muss das Reparaturprogramm nur im Journal nachsehen, bei welcher Operation der Computer abgestürzt ist, und kann dann ganz gezielt die problematische Stelle untersuchen.

Moderne Filesysteme haben Journaling eingebaut. Manche lassen es aber auch bewusst weg, um nicht zuviel Plattenzugriffe zu veranlassen.

10 Network File System (NFS)

In größeren Einrichtungen liegen die Dateien nicht immer auf dem lokalen Computer eines Benutzers, sondern u.U. über viele Platten verteilt auf vielen verschiedenen Servern. Um auf diese Dateien zuzugreifen, wurde von Sun Microsystems ein Protokoll entwickelt, das den Zugriff auf Dateien über ein Netzwerk ermöglicht.

Die wichtigsten Eigenschaften des NFS sind:

- Benutzer können auf Dateien, die sich auf einem entfernten Rechner befinden, so zugreifen, als ob sie auf ihrer lokalen Festplatte abgespeichert wären.
- Die Kommunikation läuft über den nfs-Dämon (nfsd).
- Unterstützte Operationen sind:
  - open, close: Öffnen und Schließen einer Datei
  - read, write: Lesen und Schreiben
  - create, unlink: Erzeugen und Löschen
  - mkdir, rmdir: Erzeugen und Löschen eines Verzeichnisses
  - readdir: Lesen von Verzeichniseinträgen.

Stichwortverzeichnis

Bootloader, 2

Cluster, 3

exFAT, 5

Hardlink, 8

Index Knoten, 6
iNodes, 6

Journaling, 9

Master Boot Record, 2

Network File System, 10
NFS, 10

Partition, 2
Partitionstabelle, 2
primäre Partition, 2

Sektor, 3
Softlink, 8
Spur, 3
Suffix, 7