1 Abbildung Spur/Sektor → FAT-Eintrag

Auf einer einseitigen Diskette mit 80 Spuren (Nr. 0 bis 79) und 15 Sektoren pro Spur (Nr. 1 bis 15) seien jeweils 4 aufeinander folgende Sektoren zu einer Sektorgruppe zusammengefasst. Die ersten drei Spuren seien für Systemzwecke reserviert; auf dem ersten Sektor der nächsten Spur beginne der Nutzdatenbereich mit der Sektorgruppe Nr. 2. Geben Sie eine Formel an oder schreiben Sie eine Pascal-Funktion, die aus einer Spur- und Sektornummer innerhalb des Nutzdatenbereiches die Nummer des zugehörigen *FAT*-Eintrags berechnet.

Man definiert zunächst eine über alle Spuren fortlaufende Sektornumerierung Sn beginnend bei 0 gemäß

```
Sn = (Sp - 3)*15 + Sk - 1 .
```

Daraus kann man mittels ganzzahliger Division durch 4 eine Sektorgruppennumerierung Sg0 beginnend bei 0 ableiten:

```
Sg0 = ((Sp - 3)*15 + Sk - 1) DIV 4 .
```

Durch Berücksichtigung der Anfangsnummer 2 erhält man schließlich

```
FAT-Nr. = int (((Sp - 3)*15 + Sk - 1)/4) + 2 .
```

2 Ablauf Dateizugriff

Bitte skizzieren Sie die wesentlichen Schritte von der Entgegennahme eines logischen Dateinamens wie z.B. \home\src\test.c bis zum endgültigen Zugriff auf die Daten von test.c. Die Abbildungsfunktion einer Datei/Katalogs auf eine physikalische Plattenadresse (Kopf/Spur/Sektor) können Sie als gegeben voraussetzen.

- Rootverzeichnis einlesen
- Nach Directoryeintrag „home“ suchen, ggf. weitere Sektoren einlesen
- Aus Directory-Eintrag von „home“ die Clusternummer entnehmen und einlesen
- Hier nach Directoryeintrag „src“ suchen, ggf. weitere Cluster einlesen (dazu ggf. in der FAT der Verkettung folgen)
- Aus Directory-Eintrag von src die Clusternummer entnehmen und einlesen
- Hier nach Directoryeintrag „test.c“ suchen, ggf. weitere Cluster einlesen (dazu ggf. in der FAT der Verkettung folgen)
- Aus Directory-Eintrag von test.c die Clusternummer entnehmen und einlesen

3 FAT-Inkonsistenzen

a) Welche Fälle von Inkonsistenzen kann man sich in der FAT einer Festplatte vorstellen? (Hinweis: Vergegenwärtigen Sie sich zuerst den Normalfall, also, wie eine Datei unter MS-DOS korrekt in der FAT repräsentiert ist.)

Skizzieren Sie ein Beispiel für jeden Fall von Inkonsistenz!

Machen Sie für jeden Fall einen Vorschlag, wie die Inkonsistenz so beseitigt werden kann, dass möglichst wenig Daten verloren gehen! (Es sei immer genügend freier Platz auf der Festplatte zum Anlegen von Kopien usw. als vorhanden vorausgesetzt.)

b) Wodurch können die Inkonsistenzen überhaupt auftreten?

1. Längenfehler:

Die Dateilänge im Katalogeintrag entspricht nicht der Anzahl der Sektorgruppen in der FAT-Teilliste der Datei.

1.1. Länge zu groß (d. h., FFFF tritt zu früh auf):

→ Längenangabe korrigieren.

1.2. Länge zu niedrig:

a) Kopie mit künstlichem Namen der gemäß Längenangabe verkürzten Datei anlegen und

b) Längenangabe beim "Original" korrigieren.

2. Dateilose FAT-Teillisten (lost clusters):

Einzelner FAT-Eintrag oder komplette Teilliste hat keinen zugehörigen Katalogeintrag, der auf den Anfang verweist.

→ Katalogeintrag mit künstlichem Namen erzeugen.

3. Fehler innerhalb der FAT-Teilliste einer Datei: "Listenriß":

3.1 Eintrag mit 0 tritt in der Teilliste auf:

a) Dateiinhalt gemäß dieser Teilliste inklusive dieser Sektorgruppe mit künstlichem Namen kopieren.

b) Datei verkürzen, dabei letzte Sektorgruppe entfernen.

3.2 Eintrag mit FFFF tritt in der Teilliste auf: -> Längenfehler

3.3 Eintrag mit FFF7 (fehlerhafter Sektor) tritt in der Teilliste auf:

→ versuchen, die betreffende Sektorgruppe zu lesen, ansonsten weiter wie unter 3.1.

In allen Fällen wird zusätzlich ein Längenfehler zu korrigieren sein, und möglicherweise ist auch eine dateilose FAT-Teilliste vorhanden. Jeweilige Behandlung wie oben beschrieben.

4. Listenvereinigung:

Ein FAT-Eintrag hat (mindestens) zwei Vorgänger.

→ Dateiinhalte gemäß der zweiten bis n-ten Teillisten in eigene, getrennte Dateien gegebenenfalls mit künstlichen Namen kopieren. Überzählige Verweise auf den betroffenen FAT-Eintrag auftrennen.

Sonderfall Zyklus: Verweis auf den betroffenen FAT-Eintrag vor dem Kopieren auftrennen, Datei-Ende-Kennung erzeugen.

5. Verweis übers FAT-Ende hinaus:

→ wie Längenfehler behandeln.

6. Zwei in sich konsistente, aber verschiedene FAT-Versionen:

Die betroffenen Dateien ermitteln, von den Versionen in der ersten FAT Kopien anlegen und in der zweiten FAT eintragen; erste FAT der zweiten angleichen (oder umgekehrt).

4 Clustergrößen

- Welche Clustergrößen werden von FAT16 bei der Verwendung einer 2GB-Festplatte verwendet? (Annahme: es liegt nur eine Partition vor!) Hinweis: Betrachten Sie hierzu die Maximalanzahl von Clustern, die bei MSDOS verwendet werden können!
- Welcher Verschnitt ergibt sich in folgender Situation:
  - 4271 System- und Anwendungsdateien, durchschnittl. Größe 92.400 Byte
  - 2546 Benutzerdateien, durchschnittl. Größe 207.000 Byte
- Würden Sie eine andere Partitionierung vorschlagen?

(Hinweis: Bei der Verwendung mehrerer Partitionen wird für jede Partition aufgrund ihrer Größe auch die Clustergröße beeinflusst. Allerdings ist damit natürlich auch die Partitionsgröße klar festgeschrieben, so dass es bei zu klein gewählter Partitionierung evtl. später beim Hinzufügen weiterer Dateien zu Platzproblemen kommen kann.

2GB → 32KByte Cluster (2x2^30 / 2^16)

Verschnitt: Systemdateien 92.400 Byte

→ 3 Cluster (98.304 Byte)

→ 5904 Byte Verschnitt pro Datei

→ 25.215.984 Byte Verschnitt gesamt (24 MByte)

Verschnitt: Benutzerdateien 207.000 Byte

→ 7 Cluster (229.376 Byte)

→ 22.376 Byte Verschnitt pro Datei

→ 56.969.296 Byte Verschnitt gesamt (54.3 MByte)

Gesamtverschnitt: 4% der Plattenkapazität

Andere Partitionierung:

4271 Systemdateien → 376 MByte

2546 Benutzerdateien → 502 MByte

Möglich 2 x 1GB-Partition → 16 KByte Cluster

Systemdateien: 6 Cluster,
5904 Byte Verschnitt=24 Mbyte Gesamtverschnitt

Benutzerdateien: 13 Cluster
5992 Byte Verschnitt= 14.5 Mbyte Gesamtverschnitt

Gesamtverschnitt: 1,9% der Plattenkapazität

Ersparnis: 39,8 MByte

5 Unix Dateistruktur

In einem Dateisystem unter *Unix System V* mit Blockgröße 1 Kbyte und einer Blocknummernlänge von 4 byte seien alle Blöcke bis zur Nr. 99 lückenlos belegt und alle danach frei. In diesem Dateisystem wird durch eine Reihe von sequentiellen Schreiboperationen eine neue Datei mit 524 Kbyte Nutzdaten erzeugt. Unter der Voraussetzung, dass freie Blöcke mit 1 beginnend nach aufsteigenden Nummern belegt werden, geben Sie die Nummern der von der Datei belegten Blöcke an, kennzeichnen Sie unter ihnen die Nutzdatenblöcke und erläutern Sie die Verweisstruktur!

Direkte Verweise auf die Nutzdatenblöcke:

100, ..., 109 (→ 10 Kbyte).

Einfach indirekte Verweise im Block 110 auf die Nutzdatenblöcke:

111, ..., 366 (→ 266 Kbyte).

Doppelt indirekter Verweis im Block 367 auf den indirekten Block 368 und von dort aus auf die Nutzdatenblöcke:

369, ..., 624 (→ 522 Kbyte).

Zweiter doppelt indirekter Verweis im Block 367 auf den indirekten Block 625 und von dort aus auf die Nutzdatenblöcke:

626, 627 (→ 524 Kbyte).

6 Sicheres Löschen von Dateien

Beim Löschen einer Unix-Datei wird sie nicht vollkommen von der Platte (oder Diskette) entfernt, sondern in der Plattenbuchhaltung nur als "gelöscht" gekennzeichnet. Die Datenblöcke der gelöschten Datei werden jedoch nicht überschrieben.

Es ist daher nicht auszuschließen, dass man den Inhalt gelöschter Dateien wiederherstellen kann, solange ihre Daten noch nicht anderweitig überschrieben wurden. Für sicherheitskritische Daten ist die Löschmethode von Unix somit unzureichend.

Skizzieren Sie daher zwei Algorithmen zur Lösung des dargestellten Problems. Sie dürfen dazu *Superuser*-Rechte und die Möglichkeit voraussetzen, beliebige Datenblöcke auf der Platte direkt (unter Umgehung des Dateisystems) anzusprechen. Außerdem können Sie so tun, als seien Ihnen alle wesentlichen Datenstrukturen und Nummerierungskonventionen des Dateisystems bekannt.

a) Skizzieren Sie die Vorgehensweise für einen Algorithmus, dem der Name -einer im aktuellen Katalog befindlichen, noch **ungelöschten** Datei übergeben wird. Er überschreibt den Inhalt der gegebenen Datei nicht wieder herstellbar und sorgt anschließend dafür, dass sie auch in der Plattenbuchhaltung gelöscht wird. Sie dürfen voraussetzen, dass es nur einen Verweis auf die Datei gibt und dass kein Prozess die Datei vorher geöffnet hat.

b) Skizzieren Sie einen zweiten Algorithmus, der die noch nicht wieder verwendeten Blöcke aller bereits auf normale Weise **gelöschten** Dateien einer Platte nicht wieder herstellbar überschreibt!

c) Wie müssen Sie den Algorithmus von Aufgabenteil b) erweitern, um zusätzlich auch die Namen der gelöschten Dateien sowie gelöschter Kataloge nicht rekonstruierbar zu machen? Muss auch in gelöschten Katalogen nach gelöschten Dateien gesucht werden?

a) Zunächst muss im Katalog "." der zugehörige Eintrag zu der Datei ermittelt werden. Darin findet man den Verweis auf die *Inode*-Nummer der Datei. Den zugehörigen *Inode* erhält man über die *Inode*-Tabelle hinter dem Superblock. Den Plattenblock, in dem der betreffende *Inode* sich befindet, liest man von der Platte. Da nach Voraussetzung kein Prozess die Datei vorher geöffnet hat, sind die Informationen darin aktuell.

   Im *Inode* finden sich die 10 direkten Verweise auf die "direkten Plattenblöcke" sowie die Verweise auf die einfach, doppelt und dreifach

indirekten Blöcke. Durch diese hangelt man sich bei Bedarf hindurch, bis man jeweils die eigentlichen Datenblöcke gefunden hat, und überschreibt sie mit einem beliebigen Wert, z. B. 0. Anschließend wird die Datei mit dem normalen Unix-Systemaufruf *unlink()* gelöscht. Da es nach Voraussetzung nur einen Verweis auf die Datei gibt, gelingt dies bereits mit einem *unlink()*-Aufruf.

b) Man hangelt sich durch die Freiliste im Superblock und überschreibt die zugehörigen Plattenblöcke mit einem beliebigen, aber festen Wert. Dabei erfasst man vielleicht auch Plattenblöcke, die nie benutzt worden sind; das schadet aber nichts. - Besser: Man erzeugt eine neue Datei und schreibt in sie immer wieder den festen Wert hinein, bis die Platte voll ist, und löscht sie wieder.

c) Zusätzlich zu Teil b) müssen in den Katalogen des gesamten Dateisystems die Katalogeinträge der gelöschten Dateien und Kataloge gesucht (gekennzeichnet durch 0 statt Inode-Nummer) und überschrieben werden.

Die Katalogeinträge von Dateien, die sich in gelöschten Katalogen befunden haben, werden beim Überschreiben der Plattenblöcke nach Teil b) automatisch zerstört, da ein Katalog ja nur eine spezielle Datei darstellt, dessen -Plattenblöcke nach seinem Löschen in die Freiliste im Superblock aufgenommen werden.

7 Directory-Verwaltung

MS-DOS verlängert Katalogdateien nur, d. h., es gibt die für einen Katalog einmal allozierten Sektorgruppen nicht wieder frei, auch wenn im Katalog so viel Einträge wieder gelöscht wurden, wie in eine Sektorgruppe passen würden. Skizzieren Sie für zwei Verfahren die detaillierten Schritte, um den überflüssig allozierten Platz eines Katalogs (≠ Stammkatalog) wieder freizubekommen, und zwar

a) mit normalen MS-DOS-Befehlen. Sie dürfen voraussetzen, dass für alle Ihre Aktionen genügend freier Platz auf der Platte zur Verfügung steht und dass der Katalog keine Unterkataloge besitzt. Hinweis: Sie benötigen zur Lösung keine ausgefallenen MS-DOS-Befehle.

b) mit Hilfe eines eigens dafür geschriebenen Dienstprogramms, das Zugriff auf die *File Allocation Table (FAT)* hat. Es soll auch ohne freien Platz auf der Platte und beim Vorhandensein von Unterkatalogen funktionieren.

c) Vergleichen Sie das *first-fit*-Verfahren mit dem *rotating-first-fit*-Verfahren zur Allokation von Sektorgruppen auf einer unter MS-DOS benutzten Platte im Hinblick auf
   - die Zerstückelung von Dateien
   - die Entstehung von Rückwärtsverweisen in der *File Allocation Table (FAT)*
   - den benötigten Platz auf der Platte

aa) Der Ausgangskatalog sei A. Man legt zuerst irgendwo, aber nicht als Unterkatalog von A, einen Hilfskatalog H an, kopiert dorthin alle Dateien aus A, löscht alle Dateien in A und anschließend A selbst. Dann legt man A am alten Platz neu an, kopiert dorthin alle Dateien aus H zurück, löscht alle Dateien in H und anschließend H selbst.

ab) Das Dienstprogramm schiebt zunächst durch Umkopieren der zuhinterst liegenden Katalogeinträge in vorne liegende Lücken den Inhalt des Katalogs zusammen, prüft sodann, wie viele Sektorgruppen am Ende der Katalogdatei frei geworden sind, kennzeichnet diese in der *FAT* durch den Eintrag 0 als frei und trägt in den *FAT*-Eintrag der nun letzten belegten Sektorgruppe der Katalogdatei als Endekennzeichen 0FFFFH ein.

b1) Zerstückelung von Dateien:

first-fit-Verfahren: bildet vor allem am Anfang der Platte sehr zerstückelte Dateien, da sich dort kleine Lücken (mit wenigen Sektorgruppen) sammeln. Neigt generell eher zur Zerstückelung.

rotating-first-fit-Verfahren: Zerstückelung tritt erst auf, nachdem das Verfahren vom Ende der Platte an den Anfang zurückgekehrt ist. Bis dahin ist die Chance, daß weiter vorne wieder größere Lücken entstanden sind, gewachsen. Daher eher weniger Zerstückelung.

b2) Entstehung von Rückwärtsverweisen in der File Allocation Table (FAT):

first-fit-Verfahren: nur bei Verlängerung von Dateien, und zwar, wenn in der Zwischenzeit Sektorgruppen weiter vorn in der FAT frei geworden sind.

rotating-first-fit-Verfahren: nur, wenn bei der Allokation von Sektorgruppen das Verfahren über das Ende der FAT hinaus zum Anfang zurückkehrt. Dies kann beim ersten Beschreiben und bei der Verlängerung einer Datei, wird aber insgesamt seltener geschehen.

b3) Benötigter Platz auf der Platte

bei beiden Verfahren gleich.

8 Unix- und NT-Dateisysteme (Klausur: SS 1999; Gewichtung: mittel)

Pro Festplattenblock wird bei dem in der Vorlesung betrachteten Unix-Dateisystem ja in dem Inode ein Zeiger (4 Byte) auf diesen Datenblock abgelegt und ggf. bei größeren Dateien werden noch zusätzliche Blöcke für die einfach, zweifach und dreifach indirekte Adressierung benötigt. Ferner haben wir gesehen, dass bei einer Blockgröße von 1KByte die größte auf diese Weise adressierbare Datei bei ca. 16 GByte liegt.

a) Bitte berechnen Sie den zusätzlichen Speicherbedarf (d.h. Overhead), der sich bei der maximal großen Datei aufgrund der indirekten Adressierung ergibt (ohne des im Inode für die Zeiger benötigten Platzes), d.h. wie viele zusätzliche Blöcke werden bei Blockgröße 1KByte und Zeigerlänge 4 Byte zusätzlich zur Datei benötigt? Welchem Platz in MByte (=2^{20} Byte) entspricht dies?

b) Welcher Overhead würde sich ***überschlägig*** ergeben, wenn Sie die Größe der Datei unverändert lassen, aber die Blockgröße auf 2 KByte erhöhen? (Hinweis: Bauen Sie nicht den ganzen Verweisbaum auf, sondern be-

rechnen Sie nur die Anzahl der Blöcke, die dann auch tatsächlich auf Datenblöcke verweisen. Die anderen indirekten Knoten können Sie vernachlässigen!) Geben Sie das Ergebnis wieder in MByte an!

c) Welcher Platzbedarf für den "Verwaltungs-Overhead" bei der gleichen Dateigröße ergibt sich bei Windows NT unter der Annahme, dass die Datei unfragmentiert ist?

d) Welche Konsequenzen würde bei NTFS ein hoher Fragmentierungsgrad haben? (nur Stichwörter!)

a) 66051 Blöcke (1 + 257 + 256*257+1) = 64,5 MByte

b) 16448 Blöcke (à 2 KByte) – 10 Blöcke aus den direkten Einträgen = 16438 Blöcke; entspr. 32,1 MByte

c) 0 – paßt vollständig in den MFT-Eintrag

d) weitere Datenblocknummern werden in der ATTR-Liste aufgeführt.

9 E/A-Optimierungsverfahren (Klausur 2000; Gewichtung: mittel)

Gegeben sei eine Platte mit 200 Spuren (Zylindern). Im Moment befindet sich der Schreib-/ Lesekopf auf Spur 100. Es sind Zugriffswünsche für folgende Spuren eingetroffen (ohne Unterscheidung von Schreib- oder Leseaufträgen):

160 20 180 30 110 120 30

Alle 80 ms (= 50 Spurwechsel) komme ein neuer Auftrag hinzu. Diese Aufträge seien:

10 150 60 50 25

Falls ggf. ein neuer Auftrag genau zu einem Zeitpunkt eintrifft, wo sich der Kopf gerade auf dieser Position befindet, so wird dieser Auftrag sofort mit ausgeführt.

Bestimmen Sie für das Verfahren „einfache Fahrstuhlstrategie (Pickup)“ die Abarbeitungsreihenfolge der Aufträge unter Berücksichtigung der neu eingetroffenen Aufträge und berechnen Sie die Anzahl Σ der gewechselten Spuren.

Lösung einfache Fahrstuhlstrategie:

110 – 120 – 160 – 60 – 30 – 30 – 20 – 50 – 150 – 180 – 25 – 10 Summe: 530