Praktikum 13

Kenntnisse zum FAT-Verzeichnisaufbau vertiefen

In diesem und im folgenden Praktikum werden Sie Daten aus einem FAT-Dateisystem lesen. Die entsprechenden Betriebssystemfunktionen (den sogenannten Dateisystem-Treiber) werden Sie dabei selbst in Java implementieren. Den Java-Code für einige Grundfunktionen sowie den Inhalt einer SD-Karte finden Sie auf Ilias.

Vorbereitung: Importieren Sie die `.java`-Dateien für die Grundfunktionen und das Image der SD-Karte (`rb1-fat.img`) in eine beliebige Java-Entwicklungsumgebung.

Das Image ist mit der FAT-Variante FAT16 mit 512 Byte großen Sektoren formatiert. Die wesentlichen Teile sind auf einem Speichermedium wie folgt abgelegt:

Contents	Boot Sector	FS Information Sector (FAT32 only)	More reserved sectors (optional)	File Allocation Table #1	File Allocation Table #2	Root Directory (FAT12/16 only)	Data Region (for files and directories) ... (To end of partition or disk)
Size in sectors	(number of reserved sectors)			(number of FATs)*(sectors per FAT)		(number of root entries*32)/Bytes per sector	NumberOfClusters*SectorsPerCluster

Sie werden die Klassen `DiscImage` und `Block` benutzen, die Klasse `BootRecord` wird nur intern genutzt.

Um zum Beispiel den Sektor mit der Nummer 0 zu lesen, gehen Sie wie folgt vor:

```
Block sector = new Block(512);   // Speicher reservieren
discImage.readSector(0, sector); // Sektor in Speicher lesen
```

Mit den `get`…-Methoden der Klasse `DiscImage` können Sie die Nummer bestimmter Sektoren herausfinden, die `get`…-Methoden der Klasse `Block` erlauben den Zugriff auf Daten innerhalb des gelesenen Sektors.

Modifizieren Sie im Folgenden nur die Methode `printRootDir()` in der Datei `FatReader.java`.

Aufgabe 1: Geben Sie den ersten Sektor des Root Directories mit Hilfe von `Block.toString()` als sogenannten Hexdump aus. Ähnlich dem Memory View des AVR Studios beginnt jede Zeile des Hexdumps mit einem hexadezimalen Offset. Darauf folgt ein Datenbereich, in dem die Bytewerte ab diesem Offset in hexadezimaler Form dargestellt sind. Es schließt sich ein Bereich an, in dem dieselben Daten soweit möglich als ASCII-Zeichen dargestellt sind. Die Tabelle im Anhang gibt an, wie ein einzelner Verzeichniseintrag strukturiert ist.

Ab welchem Offset beginnt der erste Verzeichniseintrag, wo der zweite und wo der dritte Eintrag? Bestimmen Sie aus dem Hexdump die Größe der dritten Datei als Dezimalzahl. Bestimmen Sie die Dateigröße danach nochmals mit Hilfe von `Block.get`… direkt.

Aufgabe 2 (programmiertechnisch schwierig, optional und außer Konkurrenz): Geben Sie das vollständige Root Directory auf den Bildschirm aus. Dieses kann mehr als einen Sektor umfassen! Unterdrücken Sie alle Einträge, die in eine der folgenden drei Kategorien fallen:

```
// Eintrag kategorisieren
boolean isEmpty = dateiname.charAt(0) == '\u0000';
boolean isDeleted = dateiname.charAt(0) == '\u00e5';
boolean isVfat = attribute == 0x000f;
```

Die Bildschirmausgabe soll die folgenden Daten enthalten: Attribute (nutzen Sie die Methode `attrToString`), erster Cluster mit Daten, Dateigröße, Dateiname, Erweiterung

Anhang

Verzeichniseintrag: Ein FAT-Verzeichniseintrag ist 0x20=32 Byte lang und wie folgt aufgebaut (siehe auch Vorlesung, Tabelle aus Wikipedia):

Offset (hex)	Länge (in Byte)	Inhalt
00	8	Dateiname ohne Erweiterung Die nicht genutzten Bytes werden mit Leerzeichen aufgefüllt.
08	3	Erweiterung Die nicht genutzten Bytes werden mit Leerzeichen aufgefüllt.
0B	1	Dateiattribute. Bit 0: Schreibgeschützt; Bit 1: Versteckt; Bit 2: Systemdatei; Bit 3: Volume-Label; Bit 4: Unterverzeichnis; Bit 5: Archiv; Bit 6–7: ungenutzt Die zusätzlichen Pseudo-Verzeichniseinträge für VFAT (siehe weiter unten) haben das Attribut 0x0F (Schreibgeschützt, Versteckt, Systemdatei, Volume-Label)
0C	1	reserviert
0D	1	Erstellzeitpunkt in 10ms. Von 0 bis 199
0E	2	Erstellzeitpunkt (Format wie Zeit der letzten Änderung bei Offset 0x16)
10	2	Erstelldatum (Format wie Datum der letzten Änderung bei Offset 0x18)
12	2	Datum des letzten Zugriffs (Format wie Datum der letzten Änderung bei Offset 0x18)
14	2	Bei FAT32 die oberen beiden Bytes des Clusters
16	2	Zeit der letzten Änderung (5 / 6 / 5 Bits für Stunde / Minute / Sekunden) Die Auflösung der Sekunden beträgt 2 s (0..29)
18	2	Datum der letzten Änderung (7 / 4 / 5 Bits für Jahr / Monat / Tag) Jahr: Jahr seit 1980; z. B. für 2007 = 27
1A	2	(Offset des Start-Clusters) + 2
1C	4	Dateigröße in Byte

Zahlen, die mehrere Bytes lang sind, sind im Format *Little Endian* gespeichert (least significant byte first). Ein Sektor enthält mehrere Verzeichniseinträge, die hintereinander gespeichert sind.

Disk Editor: Manchmal ist es nützlich, auf den Inhalt einer kompletten Datei oder eines Gerätes in hexadezimaler Form zuzugreifen. Neben `FatReader` ist dies mit einem sogenannten Disk Editor möglich. Auf den Linux-Systemen im Praktikum können Sie dazu mit `ghex2 rb1-fat.img` ein entsprechendes Programm starten. Wenn Sie im Menü mit *Bearbeiten/Einstellungen* im Tab *Bearbeitung* die Checkbox *Offset-Spalte zeigen* gewählt haben, können Sie bequem auf den Inhalt einzelner Sektoren und Cluster zugreifen und weitere Dateisystem-Interna erkunden.