

Systeme I: Betriebssysteme Übungsblatt 4 - Dateisysteme

Aufgabe 1 (1+1 Punkte)

Es ist in der Praxis üblich, die Größe einer Datenmenge in KB (auch KByte, „Kilobyte“), MB (auch MByte, „Megabyte“) oder GB (auch GByte, „Gigabyte“) anzugeben.

Anders als sonst bezeichnet ein Kilo hier aber in der Regel nicht 1000, sondern $2^{10} = 1024$ Einheiten, ein Mega nicht 1000000, sondern $2^{20} = 1048576$ Einheiten usw. Ein KB entspricht daher genau 2^{10} Byte, ein MB entspricht 2^{20} Byte und ein GB entspricht 2^{30} Byte. Die Notation und Interpretation ist jedoch oft uneinheitlich.

Es gibt Normierungsbestrebungen, die versuchen, die in diesem Zusammenhang häufig auftretende Verwirrung zu beseitigen. So wurden von der IEC 1998 die Bezeichnungen KiB, MiB etc. eingeführt (vgl. <https://de.wikipedia.org/wiki/Binärpräfix>):

SI-Dezimalpräfixe				IEC-Binärpräfixe		
k	Kilo	$10^3 = 1.000$		Ki	Kibi	$2^{10} = 1.024$
M	Mega	$10^6 = 1.000.000$		Mi	Mebi	$2^{20} = 1.048.576$
G	Giga	10^9		Gi	Gibi	2^{30}
T	Tera	10^{12}		Ti	Tebi	2^{40}

Wir verwenden bei der Angabe von Datei- und Festplattengrößen KiB, MiB und GiB im Sinne von 2^{10} , 2^{20} bzw. 2^{30} Byte.

- a) Vervollständigen Sie folgende Tabelle, indem Sie die Größenangaben in Bit und Byte (=8 Bit) umrechnen und in 2er-Potenzen und in Dezimaldarstellung angeben.

Angabe	Angabe in Bits		Angabe in Bytes	
	2er-Potenz	dezimal	2er-Potenz	dezimal
2 Byte	2^4 Bit	16 Bit	2^1 Byte	2 Byte
2048 MiB				
32 Byte				
16 MiBit				
1024 KiBit				

- b) Sie haben eine „3,0 TB“-Festplatte gekauft. Was glauben Sie, welche Interpretation von „TB“ der Festplattenhersteller gewählt hat? Wie groß ist der Unterschied (in GiB) zwischen den Interpretationen?

Aufgabe 2 (1.5+1.5 Punkte)

Wahlfreier Zugriff bei I-Nodes und bei FAT

Betrachten Sie ein Dateisystem mit Blockgröße b in Byte und Zeigergröße z in Byte.

- a) Angenommen, es handelt sich um ein UNIX-Dateisystem basierend auf I-Nodes. Gehen Sie von der Implementierung in **System V** aus, d.h. jeder I-Node enthält 10 direkte Zeiger sowie jeweils einen einfach-, zweifach- und dreifach indirekten Zeiger.

Wie läuft ein wahlfreier Zugriff auf das Byte Nummer $n = 50000$ einer Datei konkret ab? Die Blockgröße sei $b = 4$ KiB und die Zeigergröße sei $z = 4$ Byte, die Anzahl der direkten Zeiger ist 10. Die Nummerierung der Bytes fängt mit der Nummer 0 an. Geben Sie an, welche Zeiger daran beteiligt sind, an welcher Position in den Blöcken diese zu finden sind und wohin sie zeigen. Geben Sie den Rechenweg mit an. Bezeichnen Sie dabei das vorderste Element in einem Block als „0. Element“.

- b) Wie läuft der wahlfreie Zugriff auf ein beliebiges Byte Nr. n einer Datei ab, falls statt eines I-Node-Systems nun ein FAT32-Dateisystem verwendet wird? Geben Sie einen allgemeinen Ausdruck für die Anzahl der zu verfolgenden Verweise N innerhalb der FAT in Abhängigkeit von der Blockgröße b an. Wie wird sich die Zugriffszeit in Abhängigkeit von n verhalten?

Aufgabe 3 (2,5+2+1,5 Punkte)

Bemerkung: Diese Aufgabe enthält Befehle, die nur mit Rootrechten ausgeführt werden können. Aus diesem Grund müssen Sie den Code nicht ausführen und können die Aufgabe mit Hilfe der angegebenen Ausgaben (sowie ggf. der entsprechenden `man` pages der Befehle) lösen.

Unter Linux können Sie Dateisysteme auf einem *block device* (z.B. einer Festplattenpartition) mit dem Befehl `mkfs.xyz` erstellen, wobei `xyz` für das Dateisystem steht. In dieser Aufgabe verwenden wir der Einfachheit halber eine Datei¹ als Speichermedium und erzeugen dann eine FAT12-Partition:

Listing 1: Dateisystem erstellen

```
1 # Erstelle 4MiB Datei, die mit Nullen gefuehlt ist.
2 $ dd if=/dev/zero of=blockdata.img bs=1K count=4096
3 # Erstelle Dateisystem
4 $ mkfs.fat -F 12 -s 4 -S 512 -v blockdata.img
```

Dieses Dateisystem können wir nun einhängen und mit zwei Dateien füllen (vgl. `man`-Page zu `dd`).

Listing 2: Mounten und Dateien auf Dateisystem erstellen

```
1 $ mkdir my_mountpoint
2 $ sudo mount -o uid=$(whoami) blockdata.img my_mountpoint/
3 $ dd if=/dev/zero of=my_mountpoint/vielenullen_0.bin bs=1K count=7
4 $ dd if=/dev/zero of=my_mountpoint/vielenullen_1.bin bs=1M count=2
5 $ ls -lh blockdata.img
6 -rw-rw-r-- 1 henkolk henkolk 4,0M Nov  6 16:39 blockdata.img
7 $ du my_mountpoint/
8 2072    my_mountpoint/
```

¹Unter Linux werden auch Festplattenpartitionen über `/dev/sdXN` in Form einer Datei angesprochen, z.B. `mkfs.ext4 /dev/sdb2`

Nach dem Aushängen des Dateisystems führen wir nun einen Dateisystem-Check durch und lassen uns dabei zusätzliche Informationen (-v) sowie die Dateien (-l) ausgeben.

Listing 3: Dateisystemüberprüfung mit fsck

```
1 $ sudo umount my_mountpoint/
2 $ fsck.fat -vl blockdata.img
3 fsck.fat 3.0.28 (2015-05-16)
4 Checking we can access the last sector of the filesystem
5 Boot sector contents:
6 System ID "mkfs.fat"
7 Media byte 0xf8 (hard disk)
8     512 bytes per logical sector
9     2048 bytes per cluster
10    1 reserved sector
11 First FAT starts at byte 512 (sector 1)
12    2 FATs, 12 bit entries
13    FAT_SIZE bytes per FAT (= [...] sectors)
14 Root directory starts at byte [...] (sector [...])
15     512 root directory entries
16 Data area starts at byte 23040 (sector 45)
17    2036 data clusters (4169728 bytes)
18 32 sectors/track, 64 heads
19     0 hidden sectors
20    8192 sectors total
21 Checking file /vielenullen_0.bin (VIELEN~1.BIN)
22 Checking file /vielenullen_1.bin (VIELEN~2.BIN)
23 Checking for unused clusters.
24 blockdata.img: 2 files, USED_CLUSTERS/2036 clusters
```

Hierbei berechnet sich die Blockgröße („cluster size“) aus der Anzahl der Sektoren pro Block multipliziert mit der Sektorengöße.

Bemerkung. Ein *Sektor* ist die kleinste adressierbare Einheit eines Speichergerätes (z.B. einer Festplatte), die auch direkt vom jeweiligen Geräte-Controller unterstützt wird. Die Sektorgöße beträgt typischerweise 512 Bytes, es sind jedoch auch Vielfache davon möglich. Auf Dateisystemebene wird die Festplatte in Blöcke als „Verwaltungseinheit“ unterteilt. So bestehen reguläre Dateien immer aus mindestens einem Block. Da die eigentlichen Hardwareoperationen auf Sektoren operieren, muss die Blockgröße immer ein ganzzahliges Vielfaches (und häufig zusätzlich eine Zweierpotenz) der Sektorgöße sein. Typische Werte für die Blockgröße sind 512 Bytes oder 4 KiB. Bei Microsoft-Dateisystemen wird die Blockgröße auch als „cluster size“ bezeichnet.

Beantworten Sie die folgenden Aufgaben:

- a) Wir führen die Listings 1, 2 und 3 aus.
 - 1) Berechnen Sie die Größe `FAT_SIZE` (Zeile 13) einer einzelnen *file allocation table* (FAT). Was für eine Funktion könnte die zweite FAT besitzen?
 - 2) Berechnen Sie die Anzahl der belegten Blöcke `USED_CLUSTERS` (Zeile 24) an. Zu welchen Arten von Fragmentierung in welcher Größe kommt es?

Bemerkung. Die FAT beinhaltet auch Einträge für Sektoren, in denen keine Daten liegen. So sind beispielsweise auch Einträge für Sektoren vorhanden, in denen die FAT selbst liegt.

- b) In welchem Verhältnis stehen nach Ausführen von Listings 1 und 2 die Nutzdaten (tatsächliche Dateiinhalte) zum „Overhead“ des FAT12-Dateisystems?

Betrachten Sie den Befehl

```
$ dd if=/dev/urandom of=my_mountpoint/zufall.bin bs=2K count=MAX_COUNT,
```

den wir nach Listing 2 ausführen könnten. Welchen Wert müssen Sie für MAX_COUNT wählen, um dieses Verhältnis zu maximieren?

Bemerkung. Als *Overhead* bezeichnen wir in dieser Aufgabe Bereiche des Speichermediums, die für die Verwaltung benötigt werden und somit nicht für *Dateiinhalte* zur Verfügung stehen. Wir zählen freien Speicher nicht zum Overhead.

- c) Wir betrachten ein FAT12-Dateisystem mit fester Mediengröße und fester Sektorengöße. Können Sie das Verhältnis von Nutzdaten zur Größe des Speichermediums bei der Dateisystemerstellung beeinflussen? Falls ja, welche Nachteile müssen Sie dafür in Kauf nehmen?

Abgabe: Als PDF-Datei über Ilias bis 27. November 2017, 23:59:00 Uhr.