Ext4-Übung

Florian Mayer

Oktober 2014

Inhaltsverzeichnis

1 Einführung

Im ersten Kapitel dieses Dokuments werden die Werkzeuge des e2fsprogs-Pakets näher beleuchtet. Danach folgt die eigentliche Übung

1.1 Bestandteile des e2fsprogs-Pakets

Die Tools – soweit nicht anders vermerkt – arbeiten jeweils mit ext2, ext3 oder ext4 Dateisystemen. “ExtX” bezeichnet zusammenfassend ext2- ext3- oder ext4-Systeme.

e2fsk Überprüft ein ExtX-System auf Inkonsistenzen und behebt diese, wenn möglich.

mke2fs Wird vom Frontendprogramm `mkfs` verwendet um neue ExtX-Dateisysteme anzulegen.

resize2fs Kann genutzt werden, um ein ExtX-Dateisystem an eine gewachsene oder verkleinerte Partition anzupassen.

tune2fs Setzt oder modifiziert Dateisystemparameter.

dumpe2fs Schreibt Superblock- bzw. Blockgruppeninformationen auf die Standardausgabe.

filefrag Zeigt den Grad der Dateifragmentierung an.

e2label Verändert das Dateisystemlabel eines ExtX-Systems.

findfs Sucht nach einem Dateisystem mit einem Label oder einer UUID.

e2freefrag Wie `filefrag`, jedoch mit dem Unterschied, dass nur die Fragmentierung des freien Speicherplatzes aufgezeigt wird.

chattr Ähnlich zu `chmod` oder `chown`. Funktioniert auf vielen verschiedenen Dateisystemen.

e2image Speichert kritische ExtX-Metadaten in einer Datei.

e4defrag Defragmentiert ein Ext4-Dateisystem während es parallel weiterhin genutzt werden kann.

findsuper Findet Ext2-Superblocks. Veraltet und lt. Dokumentation "schnell zusammengehackt".

1.2 Verwendete Programme

Im Rahmen dieses Dokuments werden lediglich die Programme `debugfs` und `dumpe2fs` verwendet. Die Manualseiten könnten, wie gewohnt, mit

```
$ man debugfs
$ man dumpe2fs
```

angesehen werden. Das Studium dieser Handbücher ist dringend empfohlen.

2 Übung

2.1 Vorbereitungen des Test-Dateisystems

Operationen auf Dateisysteme mit sog. Debugging-Kommandos sind gefährlich. Löscht man z.B. versehentlich einen Inode, so ist es nur noch schwer möglich die zugehörige Datei wiederherzustellen. Es wird daher beschrieben, wie eine sog. Sandbox-Umgebung (also ein vom System abgekapseltes Dateisystem) eingerichtet werden kann. Die hier geschilderte Methode steht sofort unter GNU/Linux zur Verfügung, da sie integraler Kernel-Bestandteil ist.

- `dd if=/dev/zero of=~/foo.disk count=1000`
  Erzeugt eine genullte Datei, bestehend aus 1000 Blöcken zu jeweils 0,5KiB.
- `losetup --list`
  Zeigt bereits vorhandene loop devices (footnote) an. Bereits vorhandene Loop-Geräte müssen mittels `sudo losetup -d /dev/loop<X>` ausgehängt werden.
- `sudo losetup /dev/loop0 ~/foo.disk`
  Erzeugt eine blockorientierte Gerätedatei, die alle Lese- und Schreiboperationen auf die Datei ~/foo.disk abbildet. Das Kommando muss sodann mithilfe von `losetup --list` überprüft werden. Der Vorgang war erfolgreich, wenn nun die neue Konfiguration ausgegeben wird.
- `sudo parted /dev/loop0 mklabel msdos`
  Erstellt eine Partitionstabelle mit *msdos*-Format. Das bedeutet, dass ein normaler MBR erstellt wird.
- `sudo parted /dev/loop0 mkpart primary 0 500K`
  Das erste Kommando erzeugt einen neuen MBR und das zweite richtet eine primäre Partition über die volle Länge des virtuellen Blockgeräts ein.

- `sudo mkfs.ext4 /dev/loop0p1`
  Erzeugt das zu untersuchende Dateisystem. Achtung: Hier darf /dev/loop0 nicht verwendet werden, da dies keine Partition ist! Die Gerätedatei der Partition (*/dev/loop0p1*) wird automatisch vom System erkannt und steht daher sofort nach erzeugung der Partition zur Verfügung.

- Nun muss das frische Dateisystem nur noch eingebunden werden:

```
$ sudo su
$ mkdir /mnt/foo
$ mount /dev/loop0p1 /mnt/foo
```

2.2 Allgemeine Fragen

1. Wo liegt der Superblock, bzw. ab welchem Offset beginnt dieser. Wie groß ist er?
2. Wieviel Inodes und Datenblöcke wurden beim Erstellen des Dateisystems erzeugt?
3. Wie groß ist ein Ext4-Datenblock und wie viele sind davon anfangs tatsächlich durch Dateien nutzbar?
4. Wieviele Bytes ist ein Inode groß?
5. Gibt es ein Journal?

2.3 Dumpe2fs

Im Folgenden ist eine Ausgabe des `dumpe2fs`-Kommandos aufgeführt. Beantworten Sie damit die folgenden Fragen unter der Annahme, dass Sie sich auf einem *32-Bit-System* befinden.

1. Über wie viele Bytes erstreckt sich ein ext4-Block?
2. Wie groß kann das Dateisystem maximal werden?
3. Wie viele Inodes kann es maximal geben?
4. Wie groß kann eine Datei, die in eine einzige Block-Gruppe passt, maximal sein?

```
$ dumpe2fs /dev/loop0p1
dumpe2fs 1.42.12 (29-Aug-2014)
[....]
Default mount options:    user_xattr acl
Filesystem state:         clean
Errors behavior:          Continue
Filesystem OS type:       Linux
Inode count:              64
Block count:              488
Reserved block count:     24
Free blocks:              460
Free inodes:              53
First block:              1
Block size:               1024
Fragment size:            1024
Reserved GDT blocks:      1
Blocks per group:         8192
Fragments per group:      8192
Inodes per group:         64
Inode blocks per group:   8
[...]
```

2.4 Einlesen des Superblocks

Schreiben Sie ein C-Programm mit dem sie den Superblock einlesen und einige Informationen auf der Kommandozeile ausgeben.

- Anzahl der Inodes
- Insgesamte Anzahl der Blöcke
- Erster Datenblock
- Anzahl freier Inodes
- Anzahl freier Datenblöcke
- Magic Number (Ausgabe in Hex)
- 128-bit UUID des Dateisystems (Ausgabe in Hex)
- Betriebssystem des Erstellers

2.5 Anlegen von Testdaten

1. Erstellen Sie im neuen Dateisystem den angegebenen Verzeichnisbaum mithilfe der Werkzeuge `touch`, `mkdir` und `nano`/`echo`. Hierfür steht ihnen ein Bash-Skript zur Verfügung. Der Aufruf unten in der Baumansicht aufgeführt.

```
/+-> t1
 |   +-> dirA
 |      +-> foo.txt  -- Inhalt: "FOO.TXT"
 |   +-> a.txt -- Inhalt "A"
 |   +-> b.txt -- Inhalt "B"
 |   +-> c.txt
 +-> t2
 |   +-> dirB
 |   +-> dirC
 +-> foo.txt -- Inhalt "FOO"

$ cd /mnt/foo && ./
```

   c.txt: Erstellen Sie diese Datei mithilfe von `echo abc{,,}{,,}{,,}{,,}{,,}{,,}{,} > c.txt`[1]

2.6 Debugfs

In dieser Aufgabe machen Sie sich mit dem Dateisystemdebugger `debugfs` vertraut.

1. Starten Sie debugfs (`debugfs /dev/loop0p1`).
2. Welcher Inode stellt das Verzeichnis "t1" dar, welcher "t2"? (Hinweis `ls -l`[2])
3. Welcher Inode stellt die Verzeichniswurzel dar und auf welchem Block (i.S.v. Ext4) liegt dieser? (Hinweis `imap <inode nummer> ODER <pfad>`)

[1] Das Kommando schreibt 2048 mal die Zeichenkette "abc" in die Datei, wobei die einzelnen Strings durch Leerzeichen getrennt sind (bis auf den letzten). Dies erzeugt eine Datei mit *exakt* 8*1024 Bytes. Das verwendete Konstrukt nennt sich *Brace-Expansion* und ist Sprachmittel der Bash.

[2] Es handelt sich hierbei um ein Kommando von *debugfs*, ist also insbesondere nicht mit dem *ls*-Kommando zu verwechseln.

4. Wechseln Sie mit `cd` in das Verzeichnis “t1” und lassen Sie sich alle Inodes anzeigen. Benutzen Sie das “cat”- Kommando, um sich den Inhalt der Dateien a.txt und b.txt ausgeben zu lassen.

5. Wo liegt der Block des Inodes für a.txt?

6. Lassen Sie sich mittels `stat <inode nummer>` den Inodestatus für die Datei a.txt ausgeben. Auf welchem Block liegen die Daten?

7. Geben Sie den Datenblock mittels `block_dump` aus.

8. Wie viele Blocks benötigt die Datei c.txt? (Hinweis: `blocks <inode nummer>`)

9. Geben sie auch noch einen der ausgegebenen Blöcke auf der Standardausgabe aus (`block_dump`).

2.7 Inodes

1. Lassen Sie sich Inode der Datei *a.txt* mithilfe der Kommandos `dd` und `hexdump` anzeigen.

2. Schreiben Sie ein C-Programm mit dem Sie sich die folgenden Daten des Inodes der Datei *c.txt* ausgeben lassen.

   - User-ID
   - Erstellungsdatum (nicht formatiert, am besten in hex ausgeben)
   - Zugriffsdatum (ebenfalls nicht formatiert)
   - Berechtigungen (Dateimodus)
   - Anzahl der Hardlinks
   - Dateigröße in Bytes

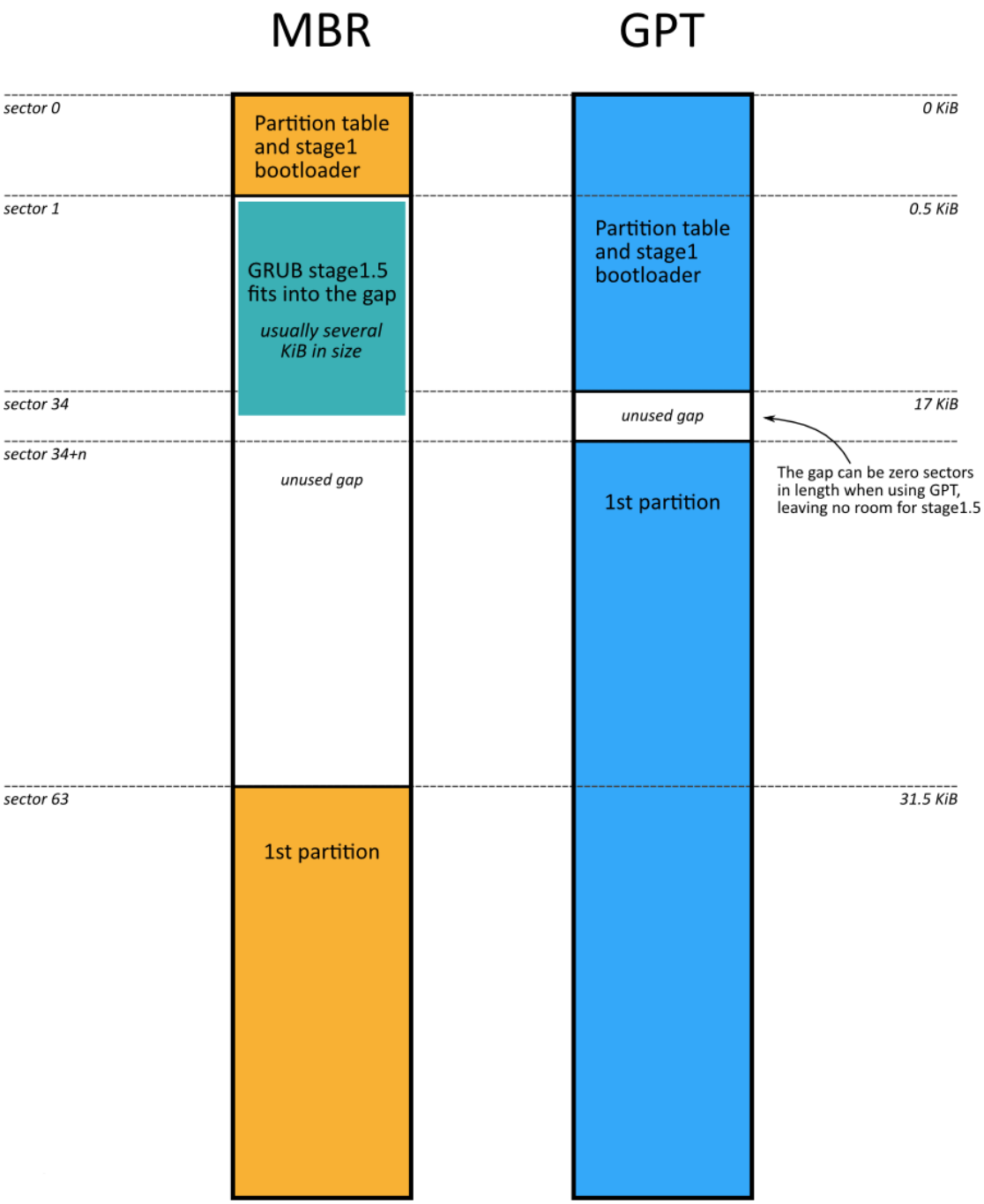
3. Schreiben Sie ein zweites Programm mit dessen Hilfe sie die Berechtigungen des eben ausgelesenen Inodes auf den Modus 777, und die UID auf 4711 setzen. Eine Sinnhaftigkeit dieser Veränderung sei dahingestellt.

3 Anhang

3.1 Layout einer HDD

Im Folgenden wird der grundlegenden Aufbau von Datenträgern unter Linux dargestellt.

Abbildung 1: Partitionslayout



3.2 HDD-Partitionslayout

Offset	Inhalt	Größe in Bytes
0x0000	Bootloader	446
0x01BE	1. Partitionseintrag	16
0x01CE	2. Partitionseintrag	16
0x01DE	3. Partitionseintrag	16
0x01EE	4. Partitionseintrag	16
0x01FE	0x55	1
0x01FF	0xAA	1

Tabelle 1: MBR Layouttabelle

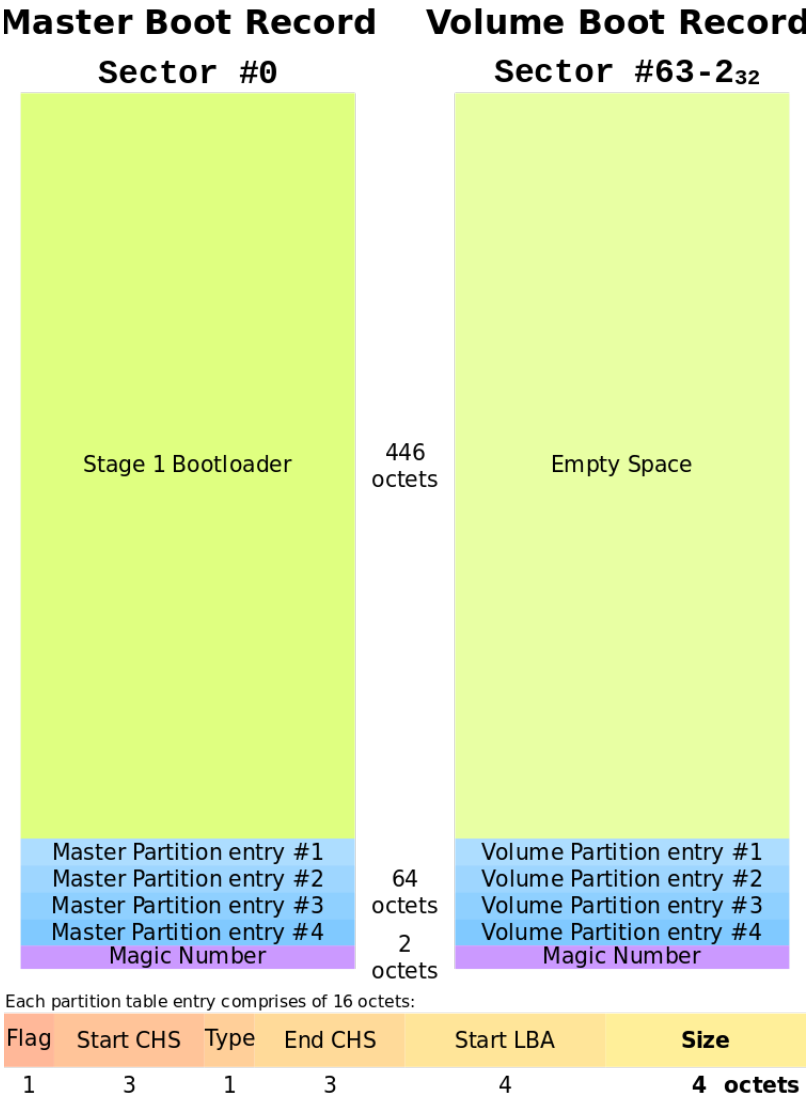


Abbildung 2: MBR-Layout

3.3 Blockgruppen Layout von Ext4

Die folgende Tabelle beschreibt den grundsätzlichen Aufbau einer ext4-Blockgruppe. Die Menge der Blöcke, die in einer Blockgruppe enthalten sind, lässt sich leicht mithilfe des `dumpe2fs`-Kommandos ermitteln (vgl. auch 2.3).

Daten	Größe
Füllbytes vor der Gruppe 0	1024 Bytes
Super Block	1 Block (1KiB, 2KiB, 4KiB, . . . , 64KiB)
Group-Descriptors	n Blöcke
Reservierte Group-Descriptors	k Blöcke. Werden für's nachträgliche Vergrößern benötigt
Datenblock Bitmap	1 Block
Inode Bitmap	1 Block
Inode Tabelle	j Blöcke (Im Test-Dateisystem 8 1KiB-Blöcke)
Datenblöcke	Der Rest

Tabelle 2: Blockgruppen Layouttabelle

Abbildung 3: Blockgruppenlayout