**Dokumentation**

**Das Dateisystem MyFS**

Das Dateisystem MyFS basiert FUSE, einem Software- Interface für Unix- artige Computersysteme, das es auch eher unerfahrenen Benutzern erlaubt, ein eigenes Dateisystem zu entwickeln.

**Aufgabenumfang**

Aufgabe 1: Erstellen eines Read-Only-Dateisystems.

Hierfür war gefordert, mittels FUSE eine Containerdatei in ein bestehendes Filesystem zu laden.

Diese Containerdatei sollte direkt bei der Erstellung mit einer Anzahl Dateien gefüllt werden.

Ein Lesezugriff auf diese Dateien sollte möglich sein, Schreiben oder Dateien wieder löschen jedoch nicht.

Aufgabe 2: Erstellen eines Read-Write-Dateisystems

Das bereits bestehende Filesystem soll in diesem Aufgabenteil folgendermaßen erweitert werden:

Es soll bei Einbindung des FUSE-Dateisystems in ein anderes Dateisystem ein leere Containerdatei erstellt werden. Diese hat eine feste Größe.

In dieser Containerdatei soll es nun möglich sein, neue Dateien zu erstellen, sowie vorhandene zu lesen, zu verändern und wieder zu löschen.

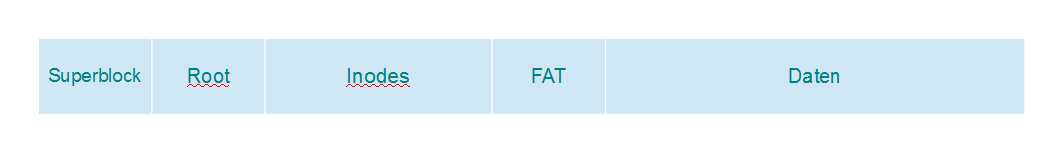
**Lösungsansatz:**

**Aufbau des MyFS:**

Das Dateisystem kann maximal 64 Dateien verwalten, von denen jede eine Maximalgröße von 2^32 -1 Byte haben kann. Es ist aus Speichersicht in Datenblöcke à 512 Byte aufgeteil.

Es besteht aus folgen Komponenten, die hintereinander im Speicher liegen:

Superblock, Root, Inodes, FAT und Daten.



Jede Komponente hat eine feste Anzahl von Speicherblöcken von je 512 Byte:

Superblock: 1 Block

Root: 1 Block

Inodes: 64 Blöcke; pro Inode 1 Block (dies entspricht somit auch der maximalen Anzahl an Dateien)

FAT: maximale Größe: 2^22 Blöcke bei maximaler Dateienzahl- und Größe, bei einer maximalen Anzahl von 128 Adressen pro Block entspricht.

1 Block minimale Größe falls es keine Dateien gibt

Daten: 2^29 Blöcke maximale Größe bei der jeweiligen Maximalgröße von 2^21 Blöcken pro Datei, was einer Dateigröße von 2 GiB entspricht.

0 Blöcke minimale Größe falls es keine Datei gibt.

Minimale Größe einer Datei: 1 Block

**Die Komponenten:**

Superblock:

Klasse: SuperBlockManager

Im Superblock werden alle grundlegenden Informationen, wie z.B. der Name und Größe des Dateisystems, sowie die maximale Anzahl von Dateien, und vieles mehr gespeichert.

Inhalt des Superblocks:

* Name des Filesystems
* Größe der Datenblöcke, die beschrieben werden können
* Maximale Dateigröße
* Maximale Anzahl von Dateien
* Größe der FAT Tabelle
* Anzahl der Adressen pro FAT Block
* Adresse des Root-Blocks
* Adresse des ersten Inode-Blocks
* Adresse des ersten FAT-Eintrags
* Adresse des ersten Daten-Blocks

Der SuperblockManager beinhaltet zwei Methoden:

init() Allokiert einen Block für den Superblock und beschreibt diesen mit den Daten des Superblock.

Falls der Superblock eine größeren Datenumfang hat wird keiner angelegt und es gibt eine Fehlermeldung.

ToDo

load()

Inodes:

Klasse: inodeManager

Jede Inode hat die Größe eines Blocks. Sie beinhaltet Informationen über die Datei, der sie zugeordnet ist, wie z.B. Dateiname, Größe der Datei, Erstellungsdatum usw.

Inhalt einer Inode:

* ID- Nummer der Inode
* Name der Datei
* Größe der Datei in Byte
* Anzahl der von der Datei beschriebenen Datenblöcke
* Für die Datei geltenden Zugriffsrechte (mode-> read, write, execute)
* Zeitpunkt des letzten schreibenden oder lesenden Zugriffs (atime)
* Zeitpunkt der letzten Veränderung des Inhalts der Datei (mtime)
* Zeitpunkt der letzten Änderung sowohl am Inhalt als auch an den Metadaten (ctime)
* Zeiger auf den ersten FAT- Eintrag
* ID des Benutzers
* Id der Benutzergruppe

FAT:

Klasse: fatManager

ToDo: Wofür steht start in writeFat() ; was mapt das FAT eigentlich auf die Adressen?

In der FAT-Tabelle stehen die Adressen aller Blöcken, die mit Dateien belegt sind, zusammen mit den jeweiligen Inode-IDs der Datei.

Der fatManager beinhaltet zwei Methoden:

writeFat(): berechnet den nächsten freien Platz in einem FAT-Block berechnet und tätigt dort einen neuen Eintrag.

readFat(): gibt einen existierenden FAT-Eintrag zurück

Root:

Klasse: rootManager

Das Root enthält eine Tabelle, die anzeigt, welche Inodes bereits belegt sind.

Der rootManager beinhaltet vier Methoden:

setInode(): trägt eine neue Inode in die Tabelle ein.

getRootBlock(): gibt einen Root-Block zurück

load(): allokiert Speicherplatz für einen Root-Block an der dafür vorgesehen

Adresse innerhalb der Struktur des Dateisystems.

isValid(): gibt zurück, ob eine bestimmte Inode-Adresse bereits belegt ist, oder nicht.

ToDo

Daten:

Klasse: