Betriebssysteme Labor

Hochschule Karlsruhe - Technik und Wirtschaft

Erstellung eines Filesystem

selu1014, zach1011, atat1011, lira1011

23. Juni 2019

Zielsetzung

Ziel ist es, ein voll Funktionsfähiges Filesystem zu Programmieren. Hierfür ist das Template für die Aufgaben zum Labor Betriebssysteme. Wenn die notwendige Arbeitsumgegebung eingerichtet wurde, sollte sich das Template-Projekt korrekt übersetzen lassen und dann die Funktionalität des Simple and Stupid File System bereitstelle.

Teil 1 - Read Only Filesystem

Erstellt werden soll ein Dateisystem MyFS, dass verwendet wird, um Datenträger zu formatieren. Unterstütz werden alle üblichen Datei mit den Attributen wie Name, Größe, Zugriffsrechte etc. Diese Dateien sind hierbei in einem einzigen Verzeichnis angeordnet (also gibt es keine Ordnerstruktur). Eine mit MyFS formatierter Datenträger kann (wie jeder Datenträger mit einem bekannten Dateisystem) in den Verzeichnisbaum eingebunden werden.

Teil 2 - Read Write Filesystem

Erstellt werden soll ein Dateisystem MyFS, dass aufbauend auf Teil 1 zusätzlich auch schreiben kann.

Inhaltsverzeichnis

Teil I

Grundlegende Informationen

Kapitel 1

Dokumentation

1.1 Aufgabenstellung

Die Aufgabenstellung bestand darin ein Filesystem auf Fuse Ebene zu Programmieren. Dies soll dazu dienen beliebige Datenträger zu formatieren. Damit kann man beliebige Dateien mit den üblichen Attributen wie Name, Zugriffsrechte und Zugriffsdatum sowie den Zeitstempel. Ebenso soll in dem Filesystem ermöglicht werden, dass wenn eine Datei formatiert wurde, diese in einem freien Platz im Verzeichnis eingetragen werden kann. Die Einbindung wiederrum findet in einem frei wählbaren und leerem Verzeichnis statt, in diesem der Inahlt des Datenträgers erscheinen soll.

Nicht wie bei üblichen Dateisystemen, bei denen die Daten auf dem Datenträger direkt abgelegt werden, wird in diesem Filesystem mit einer Container Datei gearbeitet. Auf diese wird später detaillierter eingegangen. Außerdem arbeitet MyFS nicht auf Kernel sondern auf Fuse Ebene. Auch hierbei wird später genauerter eingegangen. Somit ist MyFS ein Voirtuelles Dateisystem.

Auf üblichen Dateisystemem, die auf Kernel Ebene arbeiten, finden Anfragen auf dem User Space des VFS statt. Danach der Block- und IO-Layer sowie dem Treiber. Auf MyFS wird diese Vorgangskette vermieden.

Im zweiten Teil der Aufgabe, also das Read Write Filesystem, war das Ziel mittels verschiedenen Kommandozeilen die Container Datei zu erstellen und es schließlich zu Mounten. Dabei muss die Container Datei sowie die gewünschten Dateien als Parameter übergeben, was wie folgt aussehen kann:

```
./mkfs.myfs container.bin datei1.txt datei2.pdf ... dateiN.mp4
```

Um die nun gefüllte Container Datei und das Filesystem zu mounten war folgender Befehl nötig. Hierbei übergibt man die Container Datei, die Log Datei und schließlich das Mount Verzeichnis.

```
./mount.myfs container.bin log.txt mount}
```

Somit ist unser Filesystem mit den N Dateien einsatzbereit.

Hierbei ist jedoch stark zu differenzieren. Im ersten Aufgabenteil handelt es sich um ein sogenanntes Read Only Filesystem. Das beduetet das lediglich Daten in das Filesystem geschrieben und gelesen werden können. Eine Änderung der Datei ist dabei nicht möglich. Erst im zweiten Aufgabenteil, wird diese Funktion gewährleistet.

1.2 Vorgaben

Das Read Only Filesystem hat Regeln die zu beachten sind. Darunter fallen:

- Mindestplatz von 30,1 MB
- Maximale Dateinamen Länge von 255 Charakteren
- Blockgröße von 512 Byte
- Maximale Dateneinträge von 64 Dateien

Ebenso ist diverser Code der einige Funktionalitäten bereitstellen sowie ein Testframework gegeben.

Das Filesystem muss, laut Vorgabe, folgendes erfüllen bzwl folgende Eigenschaften in der Grundstruktur aufweisen:

Superblock Informationen zu dem Filesystem (Größe, Position der Einträge, ...)

DMAP Verzeichnis der freien Datenblöcke

FAT Dateizuordnungstabelle

Root Datein im Filesystem mit folgenden Einträgen

- Dateiname
- Dateigröße
- Benutzer / Gruppe - ID
- Zugriffsberechtigungen (mode)
- Zeitpunkt letzer Zugriff (atime)
- Zeitpunkt letzte Veränderung (mtime)
- Zeitpunkt letzte Statusänderung (ctime)
- Zeiger auf ersten Datenblock

Daten Daten der Datei

Ein Datenblock hat dann folgenden Aufbau / Grundstruktur.

Superblock	DMAP	FAT	Root	Daten

Teil II

Read Only Filesystem

Kapitel 2

Dokumentation

2.1 Aufbau

2.1.1 container.bin

Der Grundbestandteil eines Dateisystemes ist die Festplatte, da MyFS nicht auf Kernel Ebene sondern auf Fuse Ebene arbeitet, wird hierbei auf eine Container Datei zurückgegriffen, welches die Festplatte simuliert.

Diese Container Datei hat, laut Vorgabe, für 30,1 MB Platz und wird in Blöcke der Größe 512 Byte aufgeteilt. Somit hat die Container Datei Platz für 64 Dateien und eine Gesamtgröße von 65.536 Blöcken, also 32 MB.

Ein Block besteht aus Superblock, DMAP, FAT und Root. Diesen Block wird wie folgt aufgeteilt:

Superblock Wird durch den Blockindex 0 andressiert. Wird hier nicht behandelt.

DMAP Wird durch den Blockindex 1 - 128 adressiert und beinhaltet die Daten. Dabei ist zu beachten, das jeweils nur ein Block vergeben wird. Dieser Block ist entweder beschrieben oder mit *e* als Leer (Empty) gekennzeichnet.

FAT Wird durch den Blockindex 129 - 384 adressiert und beinhaltet eine Art Inhaltsverzeichnis, an welcher Stelle die Blöcke stehen.

Root Wird druch den Blockindex 185 - 448 adressiert und beinhaltet die einzelnen Einträge. Hierbei sind keine Ordnerstrukturen erlaubt. Ebenso werden hier Attribute wie Name, Größe, Benutzer / Benutzergruppe, Zugriffszeiten etc. gespeichert. In dem Root Verzeichnis sind nur Platz für 64 Dateien.

Daten Wird durch den Blockindex 449 - 65.535 adressiert und beinhaltet die Dateien.

2.1.2 log.txt

Jedes anständige Programm hat eine Log Datei, in die Änderungen oder auch Zugriffe gespeichert werden. So auch MyFS. In der Log Datei werden Fehler, Errors oder auch Informationen zu den Dateien gespeichert bzw. geloggt.

2.1.3 mount

Da MyFS nicht auf Kernel Ebene sondern auf Fuse Ebene arbeitet, haben wir keine Platte auf der die Informationen gespeichert werden. Daher bedient sich MyFS einem mount Ordner der

dies Simuliert. Dafür wird dieser in ein Filesystem gewandelt bzw. gemountet.

```
./mount.myfs container.bin log.txt mount
```

Mit hilfe des folgenden Befehls kann genau dies Rückgängig gemacht werden.

```
fusermount -u mont
```

2.1.4 mkfs.myfs

Um die Grundstruktur aus Kapitel 1.3.1 umsetzten zu können waren erweitere Methoden notwendig. Hierbei handelt es sich weitestgehend um Initialisierungen und Zuweisungen.

initializeObjects() Initialisert den Blockdevice und den Superblock.

initDMapAndFat() Initialisert die DMAP und die FAT mit leeren Einträgen.

writeSuperBlockToContainer() Beschreibt den Container mit den Einträgen der DMAP.

writeFatToContainer() Beschreibt den Container mit den Einträgen der FAT.

writeFilesToContainer() Beschreibt den Container mit den Daten.

print() Gibt auf den Console diverse Kontroll Informationen aus.

inputChecks() Kontrolliert eine Ordnungsgemäße Parameterübergabe.

2.1.5 myfs

Um den oben genannten Bereich Codieren zu können, bedient sich MyFS diverser Methoden. Fuse bietet selbstverständlich mehr Methoden zur Erstellung eines Filesystems an. Da es sich bei MyFS um ein Read Only Filesystem handelt, sind folgende 3 Methoden zur Implementierung eines solchen Read Only Filesystem vollkommen ausreichend.

In dem Teil 2 des Filesystem wird lediglich an diesem Methoden etwas geändert. Denn es kommen unter anderem fuseWrite() hinzu.

fuseGetAttr() Setzt die oben genannten Attribute der Datei, falls die Datei existiert.

fuseOpen() Öffnet eine Datei und überprüft sie auf Fehler. Hat eine Datei diese Hürde überstanden, bekommt sie ihre Position in der Liste der anderen Dateien.

fuseRead() Ließt eine Datei und schreibt die Bytes in den Buffer.

2.2 Testfälle

Damit die Funktionalität von MyFS garantiert werden kann werden sich verschiedenen Tests bedient. Darunter fallen die Tests von der Grundstruktur sowie der Ausführung.

Unter Grundstruktur versteht man die DMAP, FAT und Root, d. h. ob die Container Datei korrekt abspeichert und ggf. die Daten zum Lesen bereit stellt. Um einen Optimalen Code zu erzeugen bedient man sich diesen Test währrend der Entwicklung um schon kleinere Fehler zu vermeiden.

Im folgenden wird ein Bash - Testscript aufgeführt das die Grundfunktionalität überprüft. Um dies zu starten muss in die Kommandozeile *bash readDifferentFormats.sh datei* eingegeben werden. Das Bash Script kümmert sich um die Restliche Ausführung des Filesystem. Um mehrere Dateien zu überprüfen werden anderen Testmechanismen angewendet.

```
data=$1
echo "-----------------------------------------"
echo "Test Script startet"
echo "-----------------------------------------"
echo "> Es wird folgende Datei getestet: $data"
echo "> Das Filesystem wird vorbereitet und mit der Datei $data besch
echo "> Start Filesystem Vorbereitung"
fusermount -u mount
rm -rf container.bin
rm -rf log.txt
rm -rf mount
make clean
make
mkdir mount
touch container.bin
./mkfs.myfs container.bin $data
./mount.myfs container.bin log.txt mount
echo "> Navigiere ins Filesystem und gebe den Inhalt aus"
cd mount
ls
echo "-----------------------------------------"
echo "Test Script end"
echo "-----------------------------------------"
```

2.3 Mögliche Optimierungen

Um ein möglichst schnelles Filesystem zu garantieren bedient man sich an diversen Optimierungen, darunter fällt die Überprüfung der richtigen Parameter. Sind diese falsch bzw. nicht korrekt angegeben wird bereits ein Fehler zurückgeben werden.

Im folgenden ist ein Code Fragment was unter anderem genau dies ausführt. Es wird geschaut ob die Maximale Anzahl an Dateien eingehalten wird. Ebenso muss das Container File gegeben sein und es muss mindestens eine Datei zum Speichern übergeben werden.

Diese kleineren Zeitersparnisse machen einen deutlicher besseren Performance. Zum Beispiel kann auf MyFS gestreamt werden.

```
int inputChecks(int argc, char *argv[]) {
        if (argc > 2 + NUM_DIR_ENTRIES) {
                cout << "Error(to much files): <<
                You provided more then 64 files. " << endl;
                return -1;
        }
        if (argc < 2) {
                cout << "Error(no container file): <<
                No container file has been provided. " <<
                "Please (create and) provide the file 'container.bin'."
                << endl;
                return -1;
        }
        if (argc < 3) {
                cout << "Error(no file): <<
                No file has been provided. " <<
                "Please (create and) provide at least <<
                one file for the file system." << endl;
                return -1;
        }
        ...
}
```

Teil III

Read Write Filesystem

Kapitel 3

Dokumentation

3.1 Aufbau

3.1.1 myfs

Um die neue Funktionalität des Filesystem nutzen zu können, werden neue Fuse Methoden benötigt.

fuseWrite() Wird aufgerufen wenn eine Datei z. B. gelöscht werden soll.

3.2 Testfälle

Damit auch hier die Funktionalität von MyFS garantiert werden kann werden sich verschiedenen Tests bedient. Darunter fallen das Löschen, Schreiben und Verändern von Daten.

Im folgenden wird ein Bash - Testscript aufgeführt das diese Grundfunktionalität überprüft. Es wird eine Datei in dem Filesystem angelegt. Ebenso wird das Verändern des Inhaltes getestet. Dabei wird der Inhalt dieser Datei neu geschrieben sowie ergänzt. Schließlich wird sie gelöscht.

Um dies zu starten muss in die Kommandozeile *bash readDifferentFormats.sh datei* eingegeben werden. Das Bash Script kümmert sich um die Restliche Ausführung des Filesystem. Um mehrere Dateien zu überprüfen werden anderen Testmechanismen angewendet.

```
#!/bin/bash

parameter=$1

echo "$(tput setaf 2)"
echo "-------------------------------------"
echo "Test Script startet"
echo "-------------------------------------"

echo "$(tput setaf 4)"
echo "> Es wird folgende Datei getestet: $parameter"
echo "> Das Filesystem wird vorbereitet und mit der Datei beschrieben"
echo "> Start Filesystem: Vorbereitung"

echo "$(tput setaf 5)"
fusermount -u mount
rm -rf container.bin
rm -rf log.txt
rm -rf mount
make clean
make
mkdir mount
touch container.bin

echo "$(tput setaf 4)"
echo "> Start Filesystem: Mit Datei beschreiben"

echo "$(tput setaf 5)"
./mkfs.myfs container.bin $parameter
./mount.myfs container.bin log.txt mount

echo "$(tput setaf 4)"
echo "> Navigiere ins Filesystem und gebe den Inhalt aus"

echo "$(tput setaf 5)"
cd mount
ls

echo "$(tput setaf 4)"
```

```
echo "> Inhalt der Datei"

echo "$(tput setaf 5)"
cat $parameter

echo "$(tput setaf 5)"
echo "> In die Datei wird nun 'Hello World' geschrieben"
echo "Hello World" > $parameter

echo "$(tput setaf 4)"
cat $paramter

echo "$(tput setaf 5)"
echo "> In die Datei wird nun 'Hello World 2' angehaengt"
echo "Hello World 2" >> $parameter

echo "$(tput setaf 4)"
cat $paramter

echo "$(tput setaf 5)"
echo "> Die Datei wird nun geloescht"
rm -rf $parameter

echo "$(tput setaf 4)"
ls

echo "$(tput setaf 2)"
echo "----------------------------------------"
echo "Test Script endet"
echo "----------------------------------------"
```

3.3 Mögliche Optimierungen