

Inhaltsverzeichnis

1	Einleitung	2
1.1	Ziel	2
1.2	Technische Umgebung	2
1.3	Struktur der Containerdatei	2
1.4	Projektstruktur	2
2	Implementierung	3
2.1	Superblock	3
2.2	DMAP	3

1 Einleitung

1.1 Ziel

Als Ergänzung zu den Inhalten der Vorlesung "Betriebsysteme" und zur Vertiefung des systemnahen Programmierens unter C++, wurde das Dateisystem MyFs erstellt.

1.2 Technische Umgebung

Da Dateisystemverwaltung auf Kernel Ebene stattfindet, wurde File System In User Space (Fuse) als Schnittstelle zwischen dem Kernel und MyFs benutzt. Somit stand auch das Zielbetriebsystem fest, es wurde Linux aufgrund der besseren Handhabung von Fuse unter Linux. Entwickelt wurde das Dateisystem mit Hilfe von XCode unter MacOS in C++. Zur Versionsverwaltung und zur Projektverwaltung wurde GitHub benutzt.

1.3 Struktur der Containerdatei

Die Containerdatei wurde intern in 5 Abschnitte unterteilt.

1. Superblock \Rightarrow *Informationen über das Dateisystem*
2. DMAP \Rightarrow *Übersicht freier Datenblöcke*
3. FAT \Rightarrow *Verlinkung von Datenblöcken zu Dateien*
4. Root \Rightarrow *Informationen über Dateien*
5. Datenblöcke \Rightarrow *Dateien in MyFs geteilt in Blöcke*

1.4 Projektstruktur

Bei der Projektplanung haben wir uns an dem MVC Pattern orientiert. Wir haben getrennte Klassen für die einzelnen Abschnitte (siehe 1.3), welche die Logik/Modell stellen. Dazu gibt es die Klasse "FilesystemIO" welche Methoden für das Schreiben in die Containerdatei, sowie das Lesen daraus zur Verfügung stellt und in dem Pattern als View angesehen werden kann. Objekte dieser Klassen werden von einer übergeordneten Klasse "MyFS" erzeugt. Diese ruft Methoden aus den Klassen auf, um die Ausführung des Programms zu steuern und kann als Control in dem MVC Pattern angesehen werden.

2 Implementierung

2.1 Superblock

Der Superblock enthält wichtige Informationen über das Dateisystem an sich. In ihm werden sowohl Informationen die für das Betriebssystem und den Benutzer relevant sind wie Größe des Dateisystems, der freie Speicherplatz und maximale Speichergröße. Als auch für den Betrieb des Dateisystems wichtige Daten wie die jeweiligen Startadressen der Bestandteile und die Größe dieser. Von der Implementierung ist es eine Struct mit 16 bit unsigned Integer werten.

2.2 DMAP

Die DMAP ist dazu da, einen Überblick zu schaffen, welche Datenblöcke frei sind. Im Endeffekt handelt es sich um einen Integer Array dessen Länge, der Anzahl vorhandenen Datenblöcke im Dateisystem entspricht. Dabei bedeutet eine "0" an Stelle x im Array, dass der x-te Datenblock frei ist, dementsprechend eine "1", dass der Datenblock belegt ist.

Die Klasse DMAP stellt 5 öffentliche Methoden, es können Blöcke als belegt und frei gesetzt werden. Außerdem gibt die Methode "getFreeBlock" den nächsten freien Block zurück. Wir haben uns dazu entschlossen, den Wert für den nächsten freien Block jederzeit im Hintergrund als private Variable zu halten und erst nachdem dieser Block beschrieben wurde, den neuen nächsten freien Block zu berechnen. Für die Initialisierung, wurden zwei Methoden implementiert "getAll" welchen den kompletten DMAP Array zurückgibt, und "setAll" welche den DMAP Array mit dem übergebenen Array überschreibt. Diese zwei Methoden werden benutzt, um die komplette DMAP in die Containerdatei auf der Festplatte zu schreiben bzw. sie wiederherzustellen.

2.3 FAT

Die Aufgabe der FAT ist die Verbindung zwischen einzelner Datenblöcke zu einer Datei herzustellen.