Studienarbeit 1

Reiser4 Dateisystemtreiber für Windows XP

Hochschule für Technick Rapperswil HSR

Wintersemester 2005 / 2006

Studenten: Josias Hosang, Oliver Kadlcek, Christian Oberholzer

Betreuer: Prof. Dipl.-Ing. Eduard Glatz

Inhaltsverzeichnis

Teil I: Einführung

1 Abstract

Innerhalb der letzten Jahre entwickelte sich ReiserFS zu einem der wichtigsten und besten Dateisysteme unter Linux, insbesondere wurde es zum Standarddateisystem der SUSE-Distribution. Um auf Dualboot Systemen unter Windows XP den Zugriff auf Reiser4-Partition zu ermöglichen, wurde im Rahmen dieser Studienarbeit ein Reiser4-Dateisystemtreiber entwickelt.

Ausgehend von den Beschreibungen des Reiser4-Dateisystems und der Dokumentation des Microsoft Installable File System Kits konnte die offene Reiser4-Implementierung für Linux nach Windows XP portiert und die Lesefunktionalität vollständig und recht stabil implementiert werden. Andere Treiberschnittstellen wurden so weit realisiert, wie es für den normalen Betrieb unter Windows notwendig ist.

Neben der Treibersoftware und der üblichen Projektdokumentation entstand ein umfangreiches Analysedokument über die Funktionsweise und den Aufbau von Reiser4 sowie eine Übersicht über die Windows-Dateisystemtreiberentwicklung.

2 Aufgabenstellung

Studienarbeit WS05/06: „ReiserFS-Dateisystemtreiber für Windows XP - reiser4xp“

Gruppe: Josias Hosang / Oliver Kadlcek / Christian Oberholzer

2.1 Einführung

Heute werden von vielen Informatikfachleuten oder anderen interessierten Personen Windows/Linux-Dual-Boot-Systeme eingesetzt. Während es von Linux her keine Mühe bereitet die Windows-Partitionen zu lesen (FAT/NTFS) kann umgekehrt von Windows her nicht auf Partitionen des Linux zugegriffen werden. In letzter Zeit wurde ReiserFS zu einem der wichtigsten und besten Dateisysteme, das unter Linux angeboten wird. Es wurde unter anderem sogar zum Standarddateisystem des SUSE-Linux. Thema dieser Arbeit ist die Entwicklung eines ReiserFS-Dateisystemtreibers für Windows XP. Das Thema wurde von den Studierenden vorgeschlagen.

2.2 Aufgabe

Ausgehend von den Beschreibungen des ReiserFS-Dateisystems, der Dokumentation des IFS (Interface File System) Kit und der offenen Implementierung für Linux soll ein Windows XP Dateisystemtreiber entwickelt werden. Die entwickelte Software stellt eine wertvolle Hilfe für den Betrieb von Windows/Linux Dual Boot Systemen dar. Sie kann zudem als Basis für weitergehende Entwicklungen von Unix-Dateisystemtreibern für Windows dienen. Es ist geplant die erstellte Software unter die GPL (General Public License) zu stellen. Dieses Thema gibt einen profunden Einblick in die Systemprogrammierung, ihre Technologien und Tools.

2.3 Hinweise

Es wird eine Durchführung der Arbeit nach den Grundsätzen des Software Engineerings erwartet. Dazu gehört eine Projektplanung mit Sollwerten, eine Erfassung der Istwerte und eine abschliessende Beurteilung des Projektverlaufs im Soll-/Istvergleich. Insbesondere sind terminliche und aufwandmässige Planabweichungen zu dokumentieren und zu begründen.

2.3.1 Literaturhinweise

- Buch zu Windows-Dateisystemtreiber: http://www.amazon.de/exec/obidos/ASIN/1565922492/qid=1119366602/sr=1-11/ref=sr_1_2_11/028-3178522-5768560
- Berichte früherer Studienarbeiten (bei Betreuer erhältlich)

2.3.2 Benötigte Software

Der Microsoft IFS-Kit für das Windows XP wird für die Dauer der Studienarbeit leihweise zur Verfügung gestellt.

2.4 Erwartete Resultate

Als Resultat dieser Arbeit soll eine demonstrierbare Software auf Stufe Prototyp vorliegen, ergänzt mit einer Dokumentation, die wichtige Überlegungen der Analyse, des Entwurfs und der Implementierung festhält.

Primäres Ziel soll es sein, dass mit dem Prototypen Dateien und Verzeichnisstrukturen einer ReiserFS-Partition gelesen werden können. Als optionale Erweiterung können bei genügend Zeit Schreib- und Löschfunktionalitäten implementiert werden.

Die entwickelte Software muss auf einem Standard-Windows-Rechner installiert, sowie bei Verwendung der dokumentierten Entwicklungs-Tools auch generiert werden können. Es ist eine Installationsbeschreibung bereitzustellen. Ergänzend soll ein nach den unten stehenden Anforderungen aufgebauter Bericht entstehen. Abzugeben sind zwei CD-ROMs (MS Windows kompatibel), die die Software und den Bericht in elektronischer Form enthalten (PDF-Dateien und Originaldateien des benutzten Textsystems). Der Bericht ist zusätzlich zweimal in gedruckter Form mit Ringbindung (kein Ordner!) bereitzustellen. Basis für die Bewertung der Arbeit stellt der gedruckte Bericht, sowie die auf der CD bereitgestellte Software dar.

Geforderte Berichtsinhalte:

- **Resultatdokumentation:**
  Sie beschreibt alle Resultate der Arbeit und soll projektbegleitend gemäss dem für die Arbeit gewählten Vorgehensmodell erstellt werden. Sie soll alle wichtigen Informationen enthalten, die ein Ingenieur benötigt, der die Arbeitsresultate ohne vorgängige Kenntnis des Projekts weiterverwenden möchte.
- **Projektdokumentation:**
  Sie umfasst jegliche Dokumentation, die sich auf die Durchführung der Studienarbeit bezieht. Dazu gehören auch ein nachgeführter Projektplan (Arbeitspakete, Zeitplan mit Meilensteinen, Plan- und Ist-Aufwände), Kurzprotokolle aller Besprechungen und ein Projektschlussbericht (was wurde erreicht bzw. nicht erreicht, was ist in der Durchführung gut/schlecht gelaufen, Schlussfolgerungen). Im Bericht ist die Aufteilung der Arbeit innerhalb der Gruppe auszuweisen (thematisch und in Anzahl Arbeitsstunden).

Im weiteren soll die Gliederung und der Inhalt des Berichts den Regelungen der Abt. Informatik entsprechen (siehe Richtlinien unter den Web-Seiten der Abt. Informatik: http://i.hsr.ch, Menü „Studienarbeiten->DokuAnleitung“). Teildokumente sollen derart in den Bericht integriert werden, dass ein hierarchisches Gesamt-Inhaltsverzeichnis mit durchgängiger Seitennummerierung ermöglicht wird.

Datenblätter, Handbücher und allfällige Kopien anderer Dokumente, die nicht selbst erstellt wurden, aber für weitere Arbeiten mit den Systemen nützlich sind, sollen separat zum Gesamtbericht abgegeben werden (Abgabeform frei; evtl. auch nur auf Abgabe-CD). Berichtsinhalte, die nicht selbst erarbeitet wurden, sind mit ihrer Quelle zu bezeichnen. Hinweise zum korrekten Zitieren sind unter folgenden URLs zu finden:

- http://www.plagiarism.org/research_site/e_citation.html
- http://www.plagiarism.org/research_site/e_what_is_plagiarism.html

2.5 Termine

Mo 24. Okt 2005	Arbeitsbeginn
Fr 10. Feb 2006	Abgabe der Kurzbeschreibung per Email an Abteilungssekretariat Frau J. Ebnöther (jebnoeth@hsr.ch). Es ist das Formular unter http://www.hsr.ch/downloads_ext/kurzfass.doc zu benutzen.
Fr 10. Feb 2006 (17:00)	Abgabe des Berichts an Betreuer

2.6 Betreuung

Betreuer: E. Glatz, Büro 1.113, Tel. 055 222 49 04, email: eglatz@hsr.ch

Während der Durchführung der Arbeit findet regelmässig jede Woche eine Besprechung mit dem Betreuer statt. Dazu werden entsprechende Termine bei Arbeitsbeginn festgelegt.

Rapperswil, den 23. Okt. 2005
Prof. Eduard Glatz

3 Management Summary

3.1 Ausgangslage

3.1.1 Reiser4

Zur Zeit wird auf vielen Linux Rechnern das Dateisystem ReiserFS[1] eingesetzt. Vor allem seit ReiserFS von der SUSE Linux Distribution als Standarddateisystem eingesetzt wird, ist es zunehmend beliebter. Damit verdrängt ReiserFS das traditionelle Dateisystem ext3 immer mehr. Der Grund dafür liegt vor allem darin, dass ReiserFS bereits deutliche Geschwindigkeitsvorteile gegenüber ext3 besitzt.

Reiser4 ist eine Weiterentwicklung von ReiserFS. Zur Zeit ist es noch in Entwicklung und wird als „experimental", das heisst höchst instabil, eingestuft. Erste Performance Tests zeigen jedoch, dass Reiser4 wiederum deutlich schneller ist als ReiserFS. Daher wird allgemein erwartet, dass Reiser4 in den nächsten Jahren ReiserFS 3 ablösen und zu einem der meistgenutzten Dateisysteme unter Linux wird.

3.1.2 Ziel

Viele Entwickler besitzen ein sogenanntes Dual-Boot System, das heisst ein System mit Windows auf einer der Partitionen und Linux auf einer anderen. Damit können sie gleichzeitig Anwendungen für Windows und für Linux schreiben oder sicherstellen, dass eine Anwendung unter beiden Betriebssystemen funktioniert.

Das Problem dabei ist, dass nur das FAT32 Dateisystem von beiden Betriebssystemen geschrieben und gelesen werden kann. Dieses Dateisystem ist allerdings unterdessen veraltet und es wäre wünschenswert, wenn es ein modernes Dateisystem gäbe, das unter beiden Betriebssysteme mindestens gelesen werden kann.

Diese Arbeit versucht dieses Problem anzugehen, indem mit Reiser4 ein Dateisystem für Windows verfügbar gemacht wird, für das in Zukunft eine grosse Verbreitung zu erwarten ist. Es soll ein Treiber geschrieben werden, welcher folgende Ziele erfüllt:

Nr	Ziel	Kann / Muss
1	Der Treiber muss eine Reiser4 Partition erkennen und mounten können.	Muss
2	Der Treiber muss Informationen über eine Reiser4 Partition lesen können. Diese Informationen bestehen aus Label, Anzahl freie und Anzahl belegte Bytes.	Muss
3	Der Treiber muss die Verzeichnisstruktur einer Reiser4 Partition lesen können und auf entsprechende Anfragen von Windows korrekt reagieren.	Muss
4	Windows bietet für Dateisystemtreiber verschiedene Interfaces zum Lesen von Dateien an. Der Treiber muss eine, vorzugsweise die einfachste, dieser Möglichkeiten unterstützen.	Muss
5	Gewisse Programme benötigen die Funktionalität, dass Daten einer Datei in den Adressraum des Haupspeichers abgebildet werden können. Dieses Feature heisst „Memory Mapped Files" und muss vom Treiber unterstütz werden.	Muss
6	Die gewöhnliche Art auf Dateien zuzugreifen ist relativ langsam. Deshalb hat Microsoft eine weitere Möglichkeit in Windows eingeführt, um schnelle Zugriffe auf Dateien zu ermöglichen. Dieses Feature wird FastIO genannt. Treiber können FastIO optional unterstützen.	Kann

Die Illustration 1 zeigt die Interaktion zwischen Betriebssystem und Treiber.

1 Von Hans Reiser wird offiziell mit ReiserFS immer die Version 3 des Dateisystems gemeint, während für die Version 4 immer die Bezeichnung Reiser4 verwendet wird.

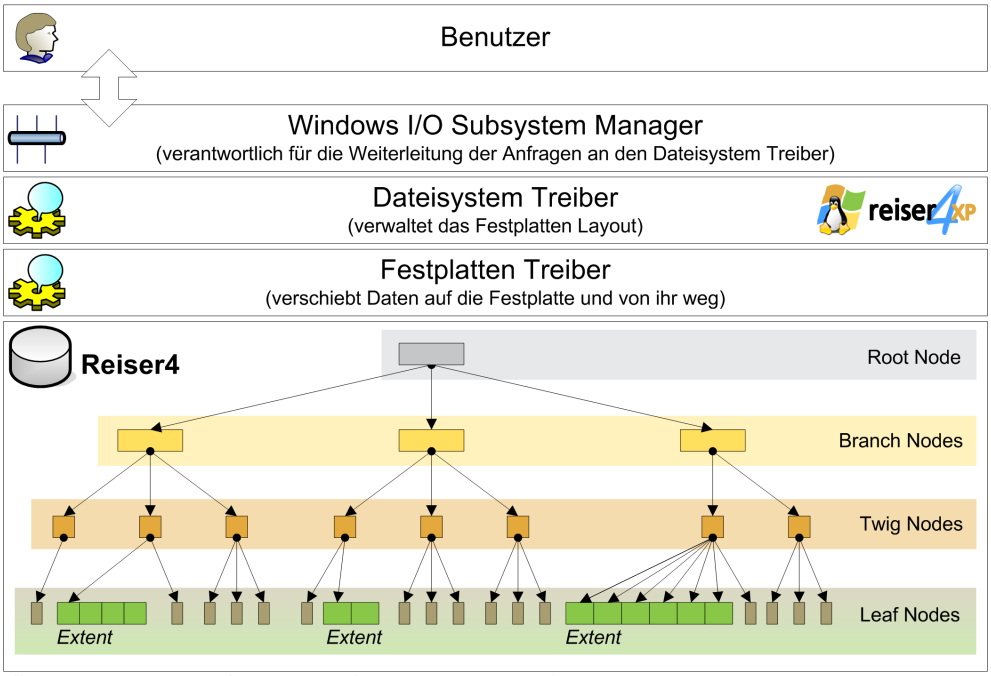

Illustration 1: Interaktion zwischen Benutzer und Dateisystem

3.1.3 Ähnliche Arbeiten

Es existieren diverse Arbeiten über ReiserFS:

- „The structure of the Reiser file system“ [FBU01]. Dieses Dokument ist eine Beschreibung der On-Disk Strukturen von ReiserFS Version 3.61.
- „rfstool“ [GKU01]. rfstool wurde von Gerson Kurz entwickelt und ist eine Kommandozeilenapplikation, mit der Daten von einer ReiserFS Partition gelesen werden können.
- „rfsd“ [MPI01]. rfsd ist ein Dateisystemtreiber für ReiserFS. Er wurde offenbar als Arbeit an einer Universität entwickelt.

Alle erwähnten Arbeiten behandeln ReiserFS und nicht Reiser4. Es konnte keine einzige Arbeit über Reiser4 gefunden werden. Diese Tatsache war ein weiterer Grund für die Durchführung der vorliegende Arbeit reiser4xp.

3.2 Durchführung der Arbeit

3.2.1 Involvierte Personen

An der Arbeit waren folgende Personen beteiligt:

Person	Funktion
Prof. Dipl. Ing. Eduard Glatz	Betreuer der Arbeit
Josias Hosang	Projektausführend
Oliver Kadlcek	Projektausführend
Christian Oberholzer	Projektausführend

3.2.2 Entwicklung

Die Entwicklung wurde nach den definierten Zielen gegliedert und in kleinen Teilschritten durchgeführt. Für jeden Entwicklungsschritt wurde ein Prototyp erstellt, in welchem die neue Funktionalität enthalten war. Nach jedem Teilschritt wurde der resultierende Prototyp getestet. Folgende Prototypen wurden implementiert:

Prototyp 1: Treiberrumpf

Der erste Prototyp umfasste die Ziele1 und 2. Er wurde am 25.11.2005 termingemäss fertiggestellt.

Prototyp 2: VolumeInfo

Der zweite Prototyp umfasste das dritte Ziel und wurde am 30.12.2005 termingemäss fertiggestellt.

Prototyp 3: Recognizer

Als dritter Prototyp war der sogenannte File System Recognizer geplant. Der File System Recognizer ist ein eigener Treiber und kann vorhandene Reiser4 Partitionen erkennen und danach den eigentlichen Reiser4 Treiber dynamisch laden. Dadurch können Resourcen gespart werden, da der Recognizer kleiner ist als der echte Treiber. Während der Evaluation von Hilfsmitteln für die Implementierung dieses Prototypen wurde festgestellt, dass der File System Recognizer für die gegebene Aufgabenstellung nicht benötigt wird und keinen Vorteil bringt. Daher wurde dieser Prototyp weggelassen.

Prototyp 4: Lesen

Der vierte Prototyp umfasste alle anderen Ziele (4, 5, 6 und 7) und wurde mit Verspätung von einer Woche am Ende der Studienarbeit am 10.02.2006 fertiggestellt. Zur Verspätung kam es, weil sich die Fehlersuche aufwändiger als erwartet erwies und das Sekundärziel FastIO noch implementiert wurde.

3.3 Erreichte Ziele

Die folgende Tabelle gibt einen Überblick über die erreichten Ziele.

Nr	Kurzbeschreibung	Status	Datum
1	Treiber starten und Partition mounten	Erreicht	25.11.2005
2	Informationen über eine Partition lesen	Erreicht	25.11.2005
3	Verzeichnisstruktur lesen	Erreicht	30.12.2005
4	Einfaches Lesen von Dateien	Erreicht	03.02.2006
5	Memory Mapped Files	Erreicht	03.02.2006
6	FastIO	Erreicht	09.02.2006

Generell gilt, dass der Treiber noch nicht absolut stabil funktioniert. Trotz der Testbemühungen während den letzten zwei Wochen der Studienarbeit muss der Treiber als „experimental“ oder „unstable“ eingestuft werden und kann während dem laufenden Betrieb abstürzen. Ein Absturz des Treibers hat für gewöhnlich einen Absturz des Gesamtsystems zur Folge.

Die angegebenen Features werden auf normalen Festplatte unterstützt. Sogenannte „Removable Devices“, das heisst zum Beispiel USB Massenspeichergeräte, werden nicht unterstützt. Nach einigen Tests mit diesen Massenspeichergeräten wurde festge-

stellt, dass der Treiber mit gewissen Einschränkungen auch für diese Geräte funktioniert. Die Einschränkungen sind:

- Das Gerät wird einmal eingesteckt. Wiederholtes Ein- und Ausstecken führt zum Absturz
- Während einer Operation, das heisst zum Beispiel während dem Lesen vom entsprechenden Gerät darf das Gerät unter keinen Umständen ausgesteckt werden.

3.4 Ausblick

Als Prototyp enthält der vorliegende Treiber zahlreiche offene Punkte. In einer Fortsetzungsarbeit könnte folgendes umgesetzt werden:

- Schreibfunktionalität / Journaling. Dies ist die umfangreichste Erweiterungsmöglichkeit, vor allem, weil das Schreiben nicht ohne Journaling funktioniert.
- Genauere Abblindung von Linux-Konzepten. Hierzu gehört die Durchsetzung von Benutzerrechten sowie die Unterstützung von Softlinks.
- Verbesserung des bestehenden Treibers, beispielsweise durch die korrekte Behandlung von Removable Devices, das saubere Herunterfahren des Treibers und die Implementation weiterer Reiser4-Plugins.
- Kleinere Dinge, wie das Beheben aller Speicherlöcher oder die allgemeine Verbesserung der Quellcode-Qualität.
- Durchführung weiterer Tests, insbesondere Dauerbelastungstests.

In Teil III, Kapitel 8.3 „Ausblick“ werden die einzelnen Punkte zusammen mit Lösungsmöglichkeiten genauer beschrieben.

Teil II: Projektmanagement

1 Projektplan

1.1 Einführung

1.1.1 Zweck

Dieses Dokument beinhaltet die detailierte Informationen zum Projektmanagement, insbesondere die Definition von Meilensteinen und Releases sowie die Risikoanalyse.

1.1.2 Gültigkeit

Dieses Dokument ist gültig für die Studienarbeit „reiser4xp“ während des Wintersemesters 2005 / 2006.

1.2 Projektübersicht

1.2.1 Ziel

Das Ziel der Studienarbeit ist die Entwicklung eines WindowsXP Dateisystemtreibers, der es Benutzern ermöglicht, auf eine lokale Reiser4-Partition lesend zuzugreifen.

1.2.2 Annahmen und Einschränkungen

Es wird von einer Richtarbeitszeit von 14 Stunden pro Woche und Teammitglied ausgegangen. Sollte sich während der Enwicklung der Umfang der Software als zu gross erweisen, wird Funktionalität niedriger Priorität gestrichen.

1.2.3 Organisationsstruktur

An der Studienarbeit beteiligt sind:

Name	Kürzel
Josias Hosang	jho
Oliver Kadlcek	oka
Christian Oberholzer	cob

Für die Betreuung zuständig ist Prof. Dipl.-Ing. Eduard Glatz.

1.2.4 Termine

Folgende Termine sind von der HSR vorgegeben:

Datum	Beschreibung
24.10.2005	Projektstart
10.02.2006	Abgabe

Während der gesamten Projektdauer findet jeweils einmal pro Woche eine Besprechung mit dem Betreuer statt.

Im Verlaufe des Projekts hat sich die Terminstruktur verändert, da die Schulleitung den Abgabetermin vom 3.2. auf den 10.2. verschoben hat. Die Termine wurden im ganzen Projekt angepasst und gegebenenfalls um eine Woche nach hinten ver-

schoben. Natürlich hatte das auch Auswirkungen auf den Zeitplan. Für das Projektmanagement ist der vorliegende angepasste Projektplan massgebend.

1.2.5 Prozessmodell

Als Prozessmodell wird eine Kombination von RUP und Prototypmodell verwendet. Grundsätzlich wird iterativ entwickelt, am Ende jeder Iteration steht ein neuer Prototyp. Auf diese Weise kann schrittweise komplexe Funktionalität hinzugefügt und ausgiebig getestet werden. Nach der Fertigstellung eines Prototyps werden jeweils die genauen Anforderungen für das nächste Release festgelegt. Deshalb werden im Gegensatz zum normalen RUP keine UseCases für die Ermittlung der funktionalen Anfoderungen benutzt.

Da es sich bei einem Treiber um ein abstraktes Gebilde und kein Problem auf Benutzerebene handelt, wird auf eine Domainanalyse verzichtet. Stattdessen wird nur die Softwarearchitektur und das Design beschrieben. Ein Teil der Vorgabe dafür stammt aus dem bestehenden Reiser4 Treiber für Linux, ein anderer Teil vom Windows XP-Treibermodell.

1.3 Meilensteine

Zu Projektbeginn werden alle Meilensteine festgelegt, die zu diesem Zeitpunkt sicher bekannt sind. Später während des Projektverlauft wird für jeden Prototyp ein zusätzlicher Meilenstein erstellt.

Name	Datum	Beschreibung
MS1: Ende Elaboration	11.11.2005	Projektplan bis auf Prototypen, Anfoderungsspezifikation und Architekturdokumentation abgeschlossen.
MS2: Treiberrumpf	25.11.2005	Getesteter PT1.
MS3: VolumeInfo	30.12.2005	Getesteter PT2.
MS4: Ende Analyse	13.01.2006	Analyse und Portierung von Reiser4 zum Grossteil abgeschlossen.
MS5: Recognizer	27.01.2006	Getesteter PT3.
MS6: Schlussrelease	03.01.2006	Getesteter PT4.
MS7: Abgabe	10.02.2006	Dokumentation abgeschlossen.

1.4 Prototypen

Prototypen können nach Bedarf während der gesamten Projektdauer definiert werden, da zu Beginn die Anforderungen noch nicht im Detail feststehen.

Name	Datum	Beschreibung
PT1: Treiberrumpf	25.11.2005	Leerer, funktionsfähiger Treiber mit Logging-Funktionalität und Memory-Manager. FA1 vollständig implementiert.
PT2: VolumeInfo	30.12.2005	Lesen von Partitionsinformationen und des Verzeichnisbaums, d.h. FA2 und FA3 implementiert.
PT3: Recognizer	27.10.2006	Lauffähiger Recognizer, der eine Reiser4 Partition erkennt und den echten Treiber lädt. FA1 wird dabei leicht modifiziert vom Treiber in den Recognizer transferiert.
PT4: Lesen	03.02.2006	Schlussrelease. Lesen von Dateiattributen (FA5), einfaches lesen von Dateien sowie lesen von Dateien unter Berücksichtigung von Windows-Caching und -MemoryMapping (FA4a und b). Falls genug Zeit bleibt FastIO.

1.5 Arfetakte

Die Artefakte beschreiben jene Dokumente, welche mit dem Erreichen eines Meilensteins vorhanden sein müssen.

Meilenstein/Prototyp	Artefakte
MS1: Ende Elaboration	Projektplan.odt (bis auf Prototypen) Projektplan.mpp Zeitplan.xls Anforderungsspezifikation.odt Architektur und Design.odt
MS2: Treiberrumpf	Test PT1 Treiberrumpf.odt
MS3: VolumeInfo	Test PT2 VolumeInfo.odt
MS4: Ende Analyse	Analyse Reiser4.odt
MS5: Recognizer	Test PT3 Recognizer.odt
MS6: Schlussrelease	Test PT4 Lesen.odt Umsetzung.odt Windows Einführung.odt Entwicklungsumgebung.odt Ausblick.odt
MS7: Abgabe	Abstract.odt Management Summary.odt Aufgabenstellung.odt Sitzungsprotokolle

1.6 Risikoanalyse

Risiko	Auswirkung	Massnahme	Kosten der Massnahme [h]	Max. Schaden [h]	Wahrscheinlichkeit des Eintretens	Gewichteter Schaden [h]
RS1: Unzureichende Dokumentation ReiserFS	Verzögerung im Projekt verlauf, Mehraufwand.	-	-	20	50%	10
RS2: Schwerwiegende Fehler im Treiber	Unstabiles OS, Bluescreens	Ausführliche Tests.	51	71	10%	7.1
RS3: Ausfall Testsystem	Analyse und Debugging müssen auf dem verbleibenden System durchgeführt werden.	Verwendung zweier Testsysteme.	-	3	10%	0.3
RS4: Datenverlust	Teile der Arbeit gehen verloren.	Einsatz von Perforce, häufige Commits.	-	5	5%	0.25
RS5: Ausfall Teammitglied	Verzögerung im Projektverlauf.	Überstunden für restliche Team-mitglieder.	-	28	10%	2.8
Total			51	127		21

In der Zeitplanung wird Schaden im Umfang von 21 Stunden eingeplant.

1.7 Qualitätsmassnahmen

1.7.1 Dokumentation

Die aktualisierung der Dokumente hat hohe Priorität. Programmcode wird laufend mit Doxygen in sinnvollem Umfang dokumentiert (siehe Anhang D „Programmierrichtlinien“).

1.7.2 Tests

Um eine möglichst grosse Stabilität des Treibers zu gewährleisten, werden zu jedem Prototyp Tests durchgeführt und deren Resultate dokumentiert. Im Vorfeld zum Test wird jeweils eine Testspezifikation erstellt. Um die Testabläufe zu erleichtern, wird bereits früh ein Memory-Manager für das Auffinden von Speicher-Lecks sowie ein Logger für die Ausgabe von Debug-Informationen implementiert.

Um alle Aspekte eines Treibers zu testen sind zwei verschiedene Testtypen notwendig. Einerseits Blackbox-Tests, welche den Treiber „von aussen“, aus der Sicht des Benutzers prüfen, um sicherzustellen, dass alle Anfoderungen erfüllt wurden. Diese Tests werden ohne Ausnahme für alle funktionalen Anforderungen durchgeführt, so weit automatisiert wie möglich, stellenweise auch manuell.

Andererseits sollten Whitebox-Tests (Unit-Tests) durchgeführt werden, welche die Reaktion des Treibers auf ungewöhnliche oder falsche Anfragen prüfen, welche im normalen Betrieb kaum auftreten. Dadurch kann die Robustheit des Treiber zusätzlich gesteigert werden. Tests dieser Art können sehr komplex sein und erfordern einen erheblichen Implementierungsaufwand, weshalb keine eigenen Whitebox-Tests im engen Zeitrahmen dieser Studienarbeit durchgeführt werden. Dafür wird der Driver Verifier von Microsoft benutzt, welcher ebenfalls die Robustheit eines Treibers testet. Er ist allgemeiner ausgelegt als ein selbst geschriebener, angepasster Test und deckt dadurch nicht alle denkbaren Fehlersituationen ab, genügen aber für die Ziele dieser Studienarbeit. Ein komerzieller Treiber müsste selbverständlich genauer untersucht werden.

Während der Entwicklung wird der Treiber laufend ohne spezielle Dokumentation auf verschiedenen Systemen mit unterschiedlichen Applikationen getestet. Es ist dehalb zu erwarten, dass in den dokumentierten Tests der Prototypen wenig zusätzliche Fehler auftreten werden.

2 Anforderungsspezifikation

2.1 Einführung

2.1.1 Zweck

Dieses Dokument legt die Anforderungen an „reiser4xp“ fest.

2.1.2 Gültigkeit

Dieses Dokument ist gültig für die Studienarbeit „reiser4xp“ während des Wintersemesters 2005 / 2006.

2.2 Allgemeine Beschreibung

2.2.1 Ziele

Die Ziele der Studienarbeit sind durch die Aufgabenstellung gegeben.

2.2.2 Annahmen

reiser4xp beschränkt sich auf Lese-Funktionalität, hohe Performance ist nicht das Primärziel.

2.2.3 Abhängigkeiten

Es sind keine Abhängigkeiten von anderen Projekten vorhanden.

2.3 Funktionale Anforderungen

Die funktionalen Anfoderungen werden so weit auf der Benutzerebene definiert wie möglich und in teilweise technischere Teilanforderungen unterteilt. Teilanforderungen der Priorität 1 müssen zwingend erfüllt werden, damit die übergeordnete Anforderungen als erfüllt gilt. Weitere Teilanforderungen werden in aufsteigender Reihenfolge implementiert. Teilanforderungen gleicher Priorität werden soweit möglich parallel umgesetzt, ansonsten nach dem Ermessen des Programmieres.

Die Implementierungsreihenfolge der übergeordneten funktionalen Anforderungen richtet sich grob nach der verwendeten Nummerierung, wird aber erst im Projektverlauf genau festgelegt.

FA1: Partition erkennen

Teilanforderung	Beschreibung	Priorität
FA1a: Partition erkennen	Sicheres erkennen einer Reiser4 Partition.	1
FA1b: Treiber laden	Laden des vollständigen Treibers.	1

FA2: Partitionsinformationen lesen

Teilanforderung	Beschreibung	Priorität
FA2a: Alle Partitionsinformationen lesen	-	1

FA3: Verzeichnisbaum lesen

Teilanforderung	Beschreibung	Priorität
FA3a: Gesamten Verzeichnisbaum lesen	-	1

FA4: Datei lesen

Teilanforderung	Beschreibung	Priorität
FA4a: Einfaches lesen	Daten ohne Caching oder Memory-Mapping lesen	1
FA4b: Memory-Mapping	Daten unter Berücksichtigung von Caching und Memory-Mapping lesen.	1
FA4c: FastIO	Schnelles lesen von Daten mit Hilfe von FastIO	2

FA5: Dateiattribute lesen

Teilanforderung	Beschreibung	Priorität
FA5a: Alle Dateiattribute lesen	-	1

2.4 Nichtfunktionale Anforderungen

2.4.1 Zuverlässigkeit

Die Erkennung einer Reiser4-Partition muss bei einer intakten Partition immer zuverlässig funktionieren.

Das Lesen einer beliebigen Testdatei in jedem beliebigen Verzeichnis muss in 9 von 10 Fällen fehlerfrei möglich sein.

Datei- und Partitionsattribute müssen in 9 von 10 Fällen korrekt ausgelesen werden.

2.4.2 Leistung

Die Leistung des Treibers ist in der vorliegenden Studienarbeit nur sekundär von Bedeutung und wird erst berücksichtigung, sobald die gesamte Funktionalität implementiert ist.

2.4.3 Modifizier- und Erweiterbarkeit

Um die Erweiterbarkeit von reiser4xp zu gewährleisten, wird das PlugIn-System des original Linux-Treibers übernommen. Implementiert werden nur die für den Betrieb minimal notwendigen Standard-PlugIns.

3 Schlussbemerkung Management

3.1 Zeitplanung

Die im Projektplan fest gehaltene Zeitplanung konnte ziemlich genau eingehalten werden.

Den eingetragenen Terminen, sprich den Projektabgaben, sowie den wöchentlichen Teamsitzungen mit dem Betreuer, konnte vollständig nach gekommen werden. Die Projektabgabe hat sich von Seite der Schulleitung um eine Woche nach hinten geschoben und die Zeitschätzung wurde dementsprechend angepasst. Es konnten mehr Sekundärziele eingeplant und auch erreicht werden.

Die gesetzten Meilensteine wurden mit einer Ausnahme termingerecht erreicht. Der Meilenstein MS6, welcher dem Schlussrelease (Prototyp 4) entspricht, konnte nicht ganz eingehalten werden. Beim Erreichen des Termines von MS6 wurde die Programmierung gestoppt, um die anderen anstehenden Arbeiten zu erledigen. Am Ende blieb noch Zeit, das Schlussrelease eine Woche später als geplant fertig zu stellen. Nachfolgende Meilensteine wurden jedoch nicht verzögert.

Die Risikoanalyse stellte sich als zutreffend heraus. Es gab einige schwerwiegende Fehler. Es kostete einige Zeit, diese wieder zu beheben. Wir brauchten ungefähr 13 Stunden, um sie zu finden, zu beheben und das System erneut zu testen, was dem Risiko RS2 (Schwerwiegender Fehler im Treiber) zugeordnet werden kann. Zum Teil war es auch schwierig, ausreichende Dokumentation für Systemaufrufe oder -fehler zu finden. Dieses Risiko war auch vorgesehen (RS1, Unzureichende Dokumentation). Wir rechneten gesamthaft 6.5 Stunden diesem Risiko an. Die gesamt eingeplante Zeit für eintretende Risiken berechneten wir mit 21 Stunden, 19.5 davon wurden in Anspruch genommen. Die Risikoplanung machten wir so, dass Zeit für die Massnahmen zur Vermeidung der Risiken und das Beheben der Folgen als zusätzlicher Aufwand im Projekt eingerechnet wurde. Das erklärt zum Teil den etwas höheren Zeitaufwand pro Person.

Die Tests wurden in dem Masse durchgeführt, welches der enge Zeitrahmen des Projektes erlaubt. Siehe auch Teil III, Kapitel 7 „Tests“.

3.2 Aufwandschätzung

Mit Ausnahme der folgenden Arbeitspakete erwies sich die Aufwandschätzungen als zutreffend.

- Für die Analyse Reiser wurde ein bisschen weniger Zeit gebraucht, was von der Tatsache herrührt, dass im Verlaufe des Projekts von der Implementation einer eigenen Bibliothek auf die Portierung des bestehenden Codes umgestellt wurde, was nicht so ausführliches Codestudium nötig machte.
- Für den Treiberrumpf wurde nur die Hälfte der Zeit gebraucht. Für den Aufwand machten wir eine vorsichtige und grosszügige Schätzung, da vielleicht unerwartete Schwierigkeiten auftreten hätten können. Die Realisierung erwies sich jedoch als problemlos.
- Für die gesamten Implementierungsarbeit wurde etwa die Hälfte mehr Zeit gebraucht. Es war sehr schwer, die Aufwände abzuschätzen, da es für uns ein recht ungewohntes Gebiet war. In diesem Zusammenhang stellt sich die Schätzung als doch recht gut heraus.
- Für die Besprechung mit dem Betreuer brauchten wir etwa ein Viertel weniger als geplant. Die eigesetzte Stunde pro Woche, stellte sich als zu grosszügig heraus.

Im Total arbeitete jedes Teammitglied im Schnitt 235 Stunden über die gesamte Projektdauer hinweg gesehen. Gesamthaft wurden 710 Stunden aufgewendet. Durch die Vorgabe eingeplant waren 210 Stunden pro Person, was einem Total von 630 Stunden entspricht, wobei noch die 21 Stunden der Risikoanalyse dazu gerechnet werden müssen. Damit liegen wir etwa 60 Stunden über dem Soll. Unserer Meinung nach befindet sich das durchaus im Rahmen, vor allem in Anbetracht, dass eine ziemlich funktionstüchtige und inhaltsreiche Arbeit entstanden ist.

Die genau Zeiterfassung befindet sich im Anhang F „Projektplan und Zeiterfassung“.

Teil III: Technischer Bericht

1 Übersicht

Der technische Bericht ist in fünf Teile gegliedert.

Zuerst werden im Kapitel 2 „Architektur“ verschiedene Treibermodelle verglichen und Designentscheide getroffen.

Danach werden in den Kapiteln 3 „Linux VFS“ und Kapitel 4 „Einführung in die Dateisystemtreiberentwicklung unter Windows“ grundlegende Konzepte vermittelt, welche das Verständnis des nachfolgenden Kapitels 5 „Umsetzung“ erleichtern.

Nachdem darin das Vorgehen bei der Entwicklung dargelegt wird, folgen im Kapitel 6 „Analyse Reiser4“ Details über den Aufbau und die Funktionsweise von Reiser4.

Das Kapitel 7 „Tests“ enthält die Spezifikationen und Resultate sämtlicher durchgeführter Tests zusammen mit einer kurzen Analyse der aufgetretenen Fehler.

Im letzten Kapitel 8 „Schlussfolgerungen“ werden die Resultate der Arbeit zusammengefasst und Erweiterungs- und Verbesserungsmöglichkeiten aufgezeigt.

2 Architektur

2.1 Einführung

In diesem Dokument werden erst verschiedene Treibermodelle miteinander Verglichen, danach wird die grundlegende Architektur des ausgewählten Modells beschrieben.

2.2 Mögliche Vorgehensweisen und Entscheidung

2.2.1 Mögliche Vorgehensweisen

Bei einer ersten Betrachtung der Problemstellung sind, abgesehen davon ungeplant „drauflos" zu entwickeln, drei verschiedene Vorgehensweisen denkbar. Diese drei Möglichkeiten werden im Folgenden vorgestellt.

Schnittstelle Windows zu Linux

Diese Variante sieht im Wesentlichen vor, den Linux Treiber-Code nicht zu verändern. Sie basiert vielmehr darauf, dass es möglich ist die Linux-API auf Windows zu simulieren. Damit könnte der Treiber unverändert auf einer Art Simulationsschicht laufen welche sämtliche Aufrufe umwandelt und an die Windows-API weiterreicht. Diese Abtraktionsschicht könnte sogar so allgemein ausgelegt werden, dass sie völlig unabhängig vom Dateisystem nur auf Linux-VFS Funktionen aufbaut. Dann wäre auch eine einfache Portierung anderer Linux-Dateisysteme möglich, allerdings gibt es hier sicher technische Grenze. Siehe auch Illustration 1.

Vorteile:

- Der Reiser4-Treiber muss nicht verändert werden, d.h. es ist keine Einarbeitungszeit in den Reiser4-Treiber nötig und es ist nicht nötig zu verstehen, wie Reiser4 seine Daten auf der Festplatte ablegt.
- Sobald die Simulationsschicht eine Linux-Treiberumgebung simulieren kann wäre es möglich mit wenig bis gar keinem Aufwand auch andere Linux Dateisysteme (wie etwa z.b. XFS) auf Windows laufen zu lassen.

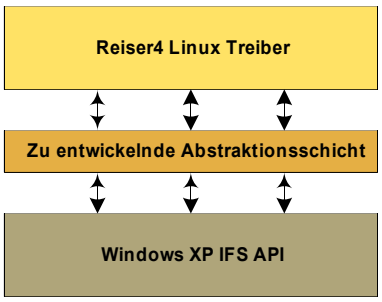

Illustration 2: Schnittstelle Windows zu Linux

Nachteile:

- Es ist nicht garantiert, dass sich die Linux-API auf die Windows-API abbilden lässt. Möglicherweise sind die Interfaces so verschieden, dass es sehr aufwändig oder sogar unmöglich ist eine Simulationsschicht zu entwickeln.

Entwicklung des Reiser4-Systems in User-Mode

Der Ansatz sieht vor zuerst die Funktionalität des Dateisystems in einer Bibliotheke zusammenzufassen. Diese kann zuerst im User-Mode des Betriebssystems entwickelt und auch getestet werden. Erst wenn sie fehlerfrei funktioniert wird ein Kernel-Mode Treiber entwickelt und die bereits funktionierende Bibliotheke eingebunden. Dadurch können die Komplexitäten der Treiberentwicklung und des Dateisystems voneinander getrennt werden. Siehe auch Illustration 2.

Vorteile:

- Die Vorgehensweise "funktioniert". In [NDS01] wurde damit bereits erfolgreich Ext2 nach Windows portiert.
- Die Schwierigkeiten der Treiberentwicklung und des Dateisystems können voneinander getrennt behandelt werden.

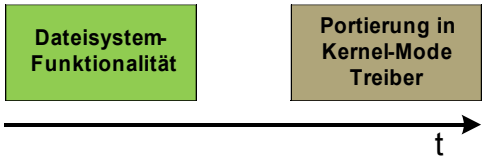

Illustration 3: Übergang User-Mode zu Kernel-Mode

Nachteile:

- Die Differenzen zwischen Kernel- und User-Mode sind relativ gross. Das Reservieren von Speicher ist komplett anders, ebenso die Synchronisierung, welche im User-Mode nicht nötig ist. Reservieren von Speicher und die Synchronisierung von Programmcode sind aber zentrale Elemente eines Treibers. Dadurch könnte es nötig werden die User-Mode Bibliotheke umzuschreiben sobald die Entwicklung des Kernel-Mode Treibers begonnen wird, was zu einer Verdoppelung der Arbeit führen könnte.
- Aufgrund dieser Differenzen ist es nicht sicher ob der Treiber der NDS Studenten nur per Zufall funktioniert hat, oder ob deren Lösung die Resourcen des Betriebssystems einfach suboptimal eingesetzt hat.

Entwicklung eines Kernel-Mode Treiberskeletts

Ähnlich wie der letzte Ansatz sieht diese Möglichkeit vor die Komplexitäten des Dateisystems von derjenigen der Treiberentwicklung zu trennen. Der Unterschied liegt jedoch in der Reihenfolge. Hier ist vorgesehen zuerst die Schwierigkeiten des Kernel-Mode Treibers zu bewältigen und ein einfaches "Treiberskelett" zu entwickeln. Im Anschluss darauf soll die Funktionalität des Dateisystems in diesen Rahmen eingepasst werden. Siehe auch Illustration 3.

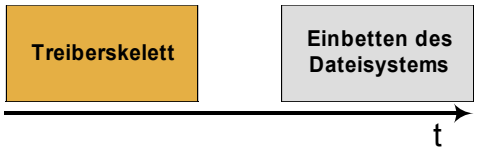

Illustration 4: Einbinden des Dateisystems in funktionierendes Treiberskelett

Vorteile:

- Es kann angenommen werden, dass die Vorgehensweise ähnlich gut funktioniert wie der Ansatz in welchem zuerst die Dateisystemfunktionalität implementiert wird. Die Komplexität der verschiedenen Teilsysteme wird ebenfalls getrennt behandelt.
- Kernel-Mode spezifische Elemente wie die Speicherallozierung oder die Synchronisierung können bereits von Anfang an mit einbezogen werden.
- Dadurch, dass bei der Entwicklung des Dateisystems bereits die Synchronisierung miteinbezogen wird kann die Portierung näher am Originalsource gehalten werden und wird dadurch einfacher.

Nachteile:

- Schwierigeres Debugging für Dateisystemfunktionalitäten im Kernel-Mode

2.2.2 Auswahl

Bei der Bewertung der verschiedenen Möglichkeiten wurde viel Wert auf eine möglichst sichere Durchführung und auf minimalen Aufwand gelegt. Ausgewählt wurde eine Kombination zweier Varianten: Erst wird ein Kernel-Mode Treiberskellett entwickelt und getestet, danach wird der bestehende Reiser4-Code nach Windows portiert und an das neue Betriebssystem angepasst. Eine komplette Abstraktions-

schicht wird jedoch nicht entwickelt, stattdessen wird das Treiberskellett Schritt für Schritt um Aufrufe des Reiser4-Codes über VFS-Funktionen ergänzt.

3 Einführung Linux VFS

Das Linux VFS (Virtual File System) wurde entwickelt um eine einheitliche Schnittstelle für alle Dateisysteme zu bilden. Applikationen und das Betriebssystem können dadurch eine API verwenden um auf viele verschiedene Dateisysteme zugreifen zu können.

Zum besseren Verständnis für die Portierung von Reiser4 ist es nützlich, gewisse Elemente des VFS zu kennen. Der folgende Abschnitt bietet einen kurzen Einstieg in das VFS. Für eine genaue Abhandlung ist es nützlich das entsprechende Kapitel in [MAU01] nachzuschlagen.

Eine weitere nützliche Quelle stellt das Dokument [RGOOCH] welches eine kurze Einführung in das VFS bietet. Der Artikel ist allerdings leider nicht mehr ganz aktuell da er für eine ältere Kernel-Version geschrieben wurde.

3.1 Initialisierung des Treibers

Unter Linux wird ein Treibermodul über die Funktionen `module_init` und `module_exit` Initialisiert und entfernt. Unter Reiser4 sind dies die Funktionen `init_reiser4` respektive `done_reiser4`. Um einen Linux-Treiber zu analysieren sind diese beiden Funktionen ein sehr guter Startpunkt. In der Initialisierungsfunktion wird Reiser4 als Linux VFS Dateisystem registriert und mit der Registrierung werden Zeiger auf Funktionen übergeben welche einen neuen Superblock erstellen bzw. den Superblock wieder löschen können. Diese Funktionen werden vom VFS verwendet wenn ein Dateisystem gemountet bzw. wieder unmountet werden soll.

3.2 Objekte

Die für die Entwicklung eines Dateisystems wichtigsten Elemente jdes VFS sind die Strukturen `file`, `dentry`, `inode` und `super_block`. Die folgende Grafik gibt eine vereinfachte Übersicht darüber wie es auf einem laufenden System aussehen könnte.

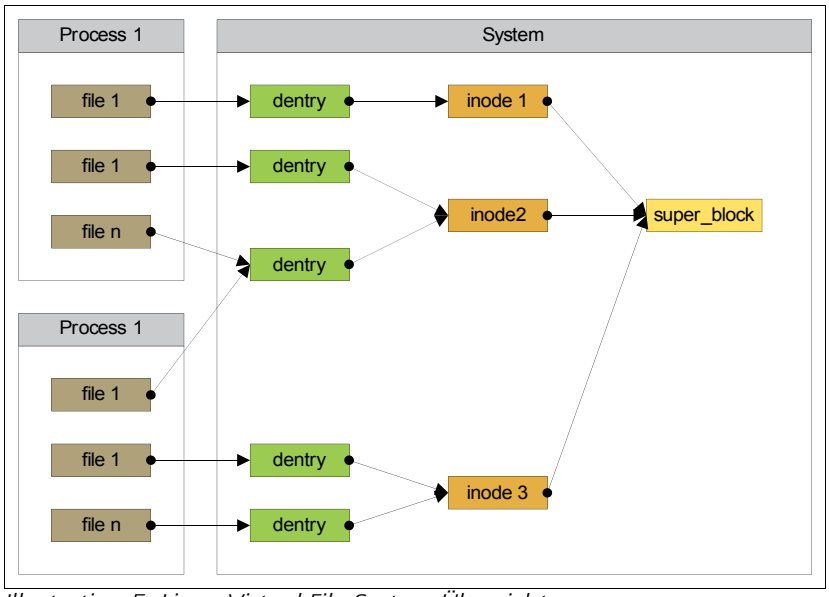

Illustration 5: Linux Virtual File System Übersicht

3.2.1 super_block

Für jedes gemountete Dateisystem wird von Linux ein einzelner Superblock angelegt. Darin werden Informationen über das geladene Dateisystem wie zum Beispiel eine Liste aller geöffneten Dateien, die Blockgrösse des Dateisystems und ähliches verwaltet.

3.2.2 inode

Ein Inode stellt ein Objekt im Dateisystem dar. Dies können Verzeichnisse, reguläre Dateien, Pipes, FIFO's oder ähnliche Objekte sein. Für jedes Objekt gibt es zu jeder Zeit nur genau einen inode.

3.2.3 dentry

Die langsamste Funktion eines Dateisystems stellt jeweils die Umwandlung eines Dateipfades (Bsp. „`/mnt/dev/verzeichnis1/datei.txt`“) in das entsprechende inode dar. Deshalb wurden die Namen von Dateien von den inodes gelöst und in separaten Objekten, den dentries, verwaltet. Ein dentry implementiert hauptsächlich Cache-Funktionen und enthält einen Zeiger auf den zu ihm gehörenden inode.

3.2.4 file

Immer wenn ein Prozess eine Datei öffnet bekommt er vom VFS ein Handle zurück. Dieses Handle wird intern auf eine Instanz der file-Struktur aufgelöst. In dieser Struktur werden Informationen wie der aktuelle File-Pointer für Lese- und Schreiboperationen verwaltet. Ein file-Objekt zeigt zusätzlich auf denjenigen dentry der von ihm geöffnet wurde.

4 Einführung in die Dateisystemtreiber-entwicklung unter Windows

Die folgenden Abschnitte wurden übersetzt und zusammengefasst aus dem Buch Windows NT File System Internals [RNAGAR01] übernommen. Sie sollen Lesern, die mit der Entwicklung von Dateisystemtreibern unter Windows nicht vertraut sind, als Einführung dienen. Die darin enthaltenen Informationen sind eine Hilfe für das Verständnis der nachfolgenden Kapitel.

4.1 Dateisystem Treiber für lokale Festplatten

Ein lokales Dateisystem verwaltet Daten auf einer Festplatte, welche direkt an einem Host angeschlossen ist. Der Dateisystem Treiber bekommt dabei Anfragen zum Öffnen, Erstellen, Lesen, Schreiben und Schliessen von solchen Dateien.

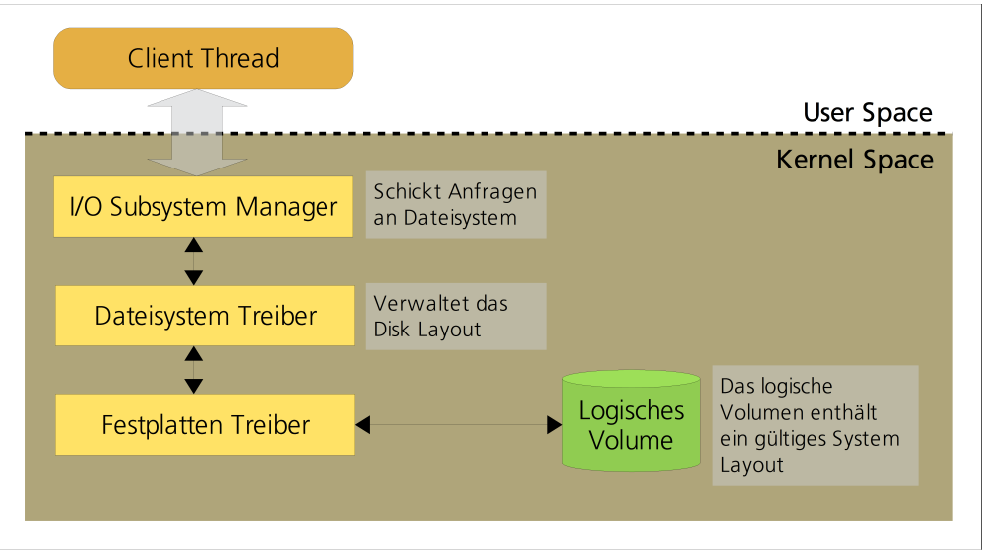

Abbildung 6: Lokales Dateisystem

Der Festplattentreiber transferiert Daten von und zu der logischen Disk. Eine logische Disk ist eine Abstraktion eines Speichers. Das Filesystem 'sieht' nur eine Anordnung von Speicherblocks. Eine logische kann ein Teil einer physischen Disk sein (Partition), oder eine ganze physische Disk.

Das logische Volume muss ein gültiges Dateisystem Layout haben, in unserem Fall Reiser4. Das Layout beinhaltet auch Metadaten zum Dateisystem.

Bevor der Benutzer auf Daten eines logischen Volumes zugreifen kann, muss das Volume gemountet werden. Mounten heisst die Metadaten lesen und verwalten, sowie geeignete in-memory Strukturen, basierend auf Metadaten, zu erstellen.

Dateisystemtreiber sind Teil des Windows I/O Managers und müssen deshalb auch diesen Schnittstellen genügen, wie alle Kernel Mode Treiber. Ein Dateisystem Treiber kann dynamisch geladen werden und theoretisch dynamisch entladen werden.

4.1.1 Benutzung des Dateisystems

Der Benutzer kann auf zwei verschiedene Arten auf die Dienste zugreifen, welche von einem Dateisystem Treiber zur Verfügung gestellt werden.

- Standart Systemaufrufe verwenden, was eigentlich die meist gebrauchte Art ist. Aufrufe sind Öffnen, Lesen und Schreiben, sowie Schliessen von Dateien.

- Selber I/O Kontrollanfragen zum Treiber schicken, und somit spezielle Services (welche im Dateisystem Treiber implementiert sind) benutzen, dazu verwendet er das File System Control (FSCTL) Interface.

Eine typische Benutzung eines Dateisystem Treibers ist zum Beispiel den Inhalt einer Datei `C:\Ordner\Datei` zu lesen. Im Win32 Subsystem geschieht folgender Ablauf:

- Die Datei öffnen
  Normalerweise wird die Win32 Funktion `CreateFile()` mit einem Namen und Zugriffsoptionen aufgerufen. Intern ruft das Subsystem `NtCreateFile()` auf. An diesem Punkt wechselt die CPU in den Kernel Mode, da diese Methode im I/O Manager implementiert ist. Der NT Object Manager parst den Pfad und gibt dem I/O Manger die Kontrolle wieder zurück, um den Dateisystem Treiber zu identifizieren, welcher zuständig ist für das logische, gemountete Volume `C:`. In diesem Treiber ruft dann der I/O Manager die `create` und `open` Routine auf.
  Der Dateisystem Treiber verrichtet seine Arbeit und gibt das Resultat an den I/O Manager zurück, welcher wiederum an das Win32 Subsystem zurück gibt. Das System geht zurück in den User Mode.
- Den Datei Inhalt lesen
  Falls das Öffnen erfolgreich war, kann der aktive Prozess eine Leseanfrage starten, indem er den Offset des Startes und die Anzahl Bytes, die gelesen werden sollen angibt. Typischerweise ruft die `ReadFile()` Funktion die Systemroutine `NtReadFile()` auf, welche auch im I/O Manager implementiert ist.
  Der aufrufende Prozess muss einen gültigen File Handle übergeben, welchen er vorher aus der Create-Anfrage erhalten hat. Der I/O Manager kann aus dem File Handle eine interne Datenstruktur (file object) identifizieren, welche zur Open-Anfrage von 1. gehört. Aus dem file object kann der I/O Manager die offene Datei auf dem Volume lokalisieren und die Kontrolle dem Dateisystem Treiber übergeben.
  Der Treiber versucht soviel wie möglich der angefraten Daten zu lesen und gibt das Resultat an den I/O Manager zurück, welcher diese weitergibt an den aufrufeneden Prozess.
- Die Datei wieder schliessen
  Nachdem alle Inhalte abgefragt wurden, wird eine Close-Operation auf dem offenen File Handle ausgeführt. Dieser Aufruf informiert das System, dass der Handle nicht weiter benötigt wird. Es wird die Win32 Funktion `CloseHandle()` aufgerufen, welche wiederum die Systemdienst Routine `NtClose()` ausführt.

Es gibt noch andere Datei Operationen, welche aufgerufen werden können. Der Ablauf ist jedoch immer der selbe: Ein Prozess öffnet eine Datei, führt Operationen darauf aus und schliesst sie wieder. Die NT Systemdienste sind in allen Threads eines Windows NT Systems verfügbar, sowohl user-mode, als auch kernel-mode Threads.

4.1.2 Die Dateisystem Treiber Schnittstelle

Damit verschieden Datreisystem Treiber unter Windows unterstützt werden können, muss eine definierte Schnittstelle existieren. Die Entwickler des I/O Subsystems strebten genau dieses Ziel an. Dies erlaubt eine modularisierte und relativ einfache Entwicklung von Treibern. Es existieren Methoden für ein Dateisystem, um sich im Betriebssystem zu installieren, laden und registrieren. Auch die Anfragepackete, welche der I/O Manager schickt, sind fest definiert (I/O request packets – IRPs). Die IRPs werden normalerweise im I/O Manager auf User Mode Anfragen erstellt und an einen kernelmode Treiber gesendet. Oder aber eine Kernel Mode Komponente erzeugt einen IRP und gibt ihn an einen Treiber tiefer unter in der Treiberhierarchie weiter. IRPs sind die einzige Möglichkeit, Dienste im I/O Subsystem zu benutzen und ermöglichen somit die Schichtung von Treibern. Ein IRP bleibt unbehandelt, bis es durch einen Empfänger bearbeitet wird und mit `IoCompleteRequest()` abgeschlossen wird und somit nicht wieder verwendbar ist. Für Dateisystem Treiber gibt es jedoch noch die Möglich-

keit von Fast I/O, welche es erlaubt, Funktionen im Treiber und Cache Manager direkt aufzurufen, statt den Weg über die IRPs zu nehmen.

4.1.3 Kernel Speicher Alloziieren

Jeder kernelmode Treiber braucht Speicher um private Daten zu speichern. Um diesen Speicher zu erhalten, startet er eine Anfrage an den NT Virtual Memory Manager (VMM) mit der Angabe, ob der Speicher paged oder nonpaged sein soll. Nonpaged Speicher ist pro System limitiert und abhängig vom Systemtyp und vom verfügbaren physischen Speicher. Mit der nicht blockierenden Methode (d.h. gibt immer Speicher oder `NULL` zurück) `ExAllocatePool()` wird der Speicher alloziiert.

4.1.4 Windows NT Object Name Space

Wie bereits erwähnt, versuchten die Designer von Windows NT das System objektorientiert aufzubauen. Es gibt einige Objekttypen mit zugehörigen Methoden in Bezug auf ein Dateisystem. In einem Dateisystem Treiber sind folgende Objekte relevant: driver object, device object, file object und directory object (ein Container für andere Objekttypen). Die Objekte, welche vom Object Manager verwaltet werden, können einen Namen zugewiesen haben. Dies erlaubt auch, auf die Objekte aus anderen Prozessen zuzugreifen.

Der Object Manger ist hierarchisch aufgebaut mit einem Root Verzeichnis '\'. Alle Objekte können mit einem absoluten Pfad ausgehend vom Root angesprochen werden. Ausserdem können spezielle Objekttypen erstellt werden, die symbolic link objects, welche ein Alias für ein anderes benanntes Objekt darstellen.

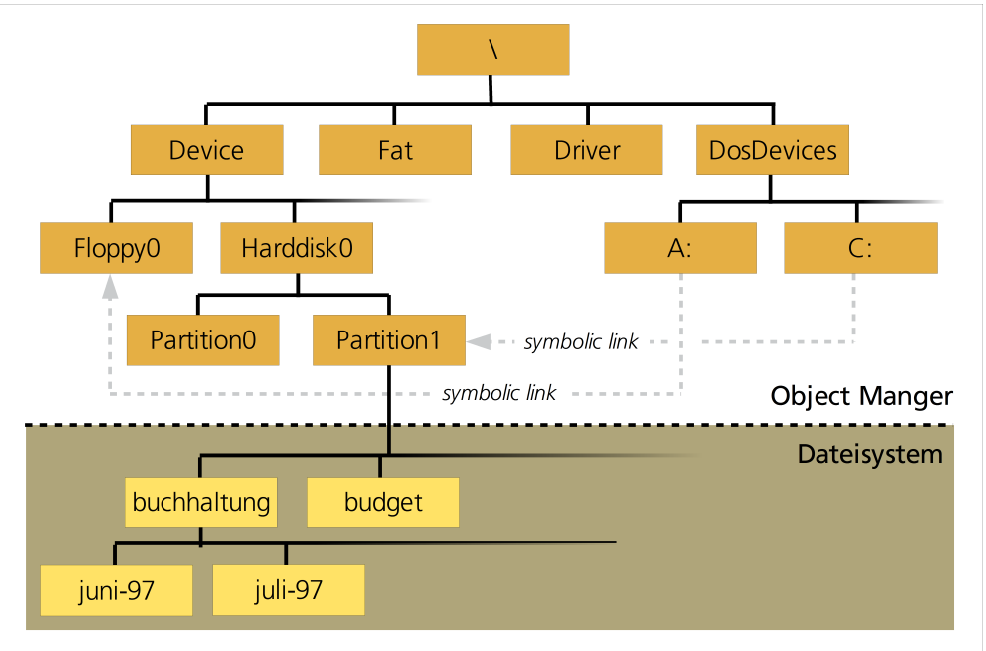

Abbildung 7: Namensraum der Windows-Objekte

4.2 Hilfsmittel bei der Treiberentwicklung

4.2.1 Exception Handling

Der Windows NT Kernel unterstützt ein Exception Handling. Eine Exception ist ein untypisches Ereigniss, das in einem Thread beim Ausführen von Instruktionen auftritt und wird im selben Threadkontext verarbeitet. Da jedoch Exception in einem Treiber relativ schwerwiegende Fehler sind, ist deren Handling nicht sehr einfach bis unmöglich (ausser den Treiber zu beenden).

4.2.2 Event Logging

Um bestimmte Informationen, z.B. Fehler, Warnungen oder auch Statusmeldungen aus Kernel Mode Treibern an den System Administrator zurück zu geben, kann der Event Log verwendet werden. Das Event Log ist eine zentrale Sammlung von Nachrichten mit definiertem Format aus verschiedenen Modulen im System.

4.2.3 Synchronisations Mechanismen

Eine Hauptaufgabe eines Treibers ist es zu verhindern, dass Daten korrumpiert werden. Die häufigste Ursache für korrupte Daten ist fehlende Synchronisation zwischen nebenläufigen Threads, welche auf die selben Daten zugreifen. Die NT Executive stellt verschiedene Arten von Synchronisations Mechanismen zur Verfügung. Hier vorgestellt werden nur diejenigen, welche in der vorliegenden Arbeit eingesetzt wurden.

Spin Locks

Spin Locks werden für gegenseitige Ausschlüsse verwendet, das heisst nur jeweils ein Thread kann eine critical region betreten. Falls ein anderer Thread den Spin Lock erhalten möchte und ihn schon ein anderer Thread hält, versucht er ständig den Spin Lock zu erhalten (busy-waiting, spinning), bis er erfolgreich ist.

Read/Write Locks

Ein bei Dateisystem häufig gebrauchter Synchronisations Mechanismus sind Read/Write Locks. Die Windows NT Executive stellt dafür eine `ERESOURCE` Struktur zur Verfügung, welche nur einem Thread exklusiven Schreibzugriff oder mehreren Threads Lesezugriff (shared) gewährt. `ERESOURCE`s müssen vor dem deallozieren des Speichers deinitialisiert werden.

4.2.4 Hilfsfunktionen (RTLs)

Die Windows NT Executive stellt eine Anzahl von Hilfsfunktionen via Runtime Library zur verfügung. Sie helfen bei folgenden Anwendungen:

- Manipulationen mit double linked lists
- Abfragen und Schreiben von Windows NT Registry Einträgen
- Konvertierungen (char zu string, ...)
- String Manipulationen (ASCII und Unicode)
- Speicher Blocks Kopieren, Löschen, Verschieben, Füllen und Vergleichen
- Arithmetische Operationen mit 32-bit integer und 64-bit large integer, sowie longs
- Zeit Manipulationen und Konvertierungen

4.3 Konzepte des NT I/O Managers

4.3.1 Share Access Rights

Beim Öffnen einer Datei kann eine Applikationen spezifizieren, welche Rechte (lesen, schreiben oder löschen) bei nachfolgendem, gleichzeitigem Öffnen der Datei gewährt werden. Wenn Dateisystemtreiber Share Access Rights unterstützen wollen, sind sie für die Umsetzung verantwortlich.

4.3.2 Byte-Range Locks

Auf einen File Stream können Byte-Range Locks angefordert werden. Damit kann eine Applikation eine Teil einer Datei für das Lesen oder Schreiben sperren. Dateisystemtreiber sind verantwortlich, die Locks durchzusetzen, müssen dieses Konzept aber nicht gezwungener Massen unterstützen.

4.3.3 FastIO

FastIO wurde in Windows eingeführt, weil festgestellt wurde, das die Verwendung von IRPs für die Ein- und Ausgabe relativ viel Overhead beinhaltet und dadurch langsam ist. Um FastIO zu unterstützen, muss ein Treiber einen Satz von Funktionen zur Verfügung stellen, die vom I/O-Manager anstelle der herkömmlichen, IRP-basierten Dispatch-Routinen aufgerufen werden.

4.3.4 Asynchrone I/O

Der I/O Manager unterstützt asynchrone I/O, das heisst ein Thread kann eine Anfrage stellen und dann andere Aufgaben ausführen, bevor er das Resultat zurückbekommt.

Wenn eine Applikation eine eine Anfrage asynchron ausführen will, ist dies für einen Treiber über einen Paramter ersichtlich. Dann sollte er die Anfrage sofort ohne zu blockieren erledigen. Wird festgestellt, dass auf eine benötigte Resource gewartet werden muss, wird die Operation mit einer entsprechenden Rückmeldung abgebrochen und für die spätere Erledigung durch einen speziellen Thread des Betriebssystems (System Worker Thread) in eine Warteschlange eingereiht. Ein System Worker Thread holt zu einem nicht beeinflussbaren Zeitpunkt das IRP wieder aus der Warteschlange reicht es zur Erledigung an den Treiber weiter. Der System Worker Thread darf jederzeit blockiert werden.

4.3.5 Interrupt Request Level (IRQL)

Jeder Thread im System wird auf einem bestimmten Interrupt Request Level ausgeführt. Jeder Thread wird ausgeführt, solange kein anderer Thread mit einem höheren IRQL ihn verdrängt und selbst eine Aufgabe erledigt.

Das IRQ Level reicht von `PASSIVE_LEVEL` (0), was als Standart für alle Benutzer Threads und System Worker Theads gesetzt wird, bis zu `HIGH_LEVEL` (31). Die meisten Dateisystem Routinen werden auf `PASSIVE_LEVEL` ausgeführt, lower-level Treiber meist auf `DISPATCH_LEVEL` (2). Um Synchronisation über die IRQLs hinweg zu erhalten müssen Spin Locks eingesetzt werden, welche immer auf `DISPATCH_LEVEL` angefordert werden.

4.4 Benutzte Daten Strukturen

4.4.1 `DRIVER_OBJECT`

Die `DRIVER_OBJECT` Struktur repräsentiert ein geladener Treiber im Speicher. Zu beachten ist, dass ein Treiber nur einmal geladen werden kann.

```
typedef struct _DRIVER_OBJECT {
        [...]
        /* a linked list of all device objects created by the driver */
        PDEVICE_OBJECT   DeviceObject;
        PVOID            DriverInit;
        [...]
        PDRIVER_DISPATCH MajorFuntion[ IRP_MJ_MAXIMUM_FUNCTION ];
} DRIVER_OBJECT;
```

Wie erwähnt müssen die IRPs an eine I/O Treiber Routine übergeben werden. Im DRIVER_OBJECT gibt es ein Array, in dem Funktionszeiger gespeichert werden können. Der Kernel Mode Treiber ist verantwortlich, dass dieses Array initialisiert wird mit Funktionen, welche der Treiber unterstützen soll.

Beim Laden des Treibers geschieht folgendes:

- Den Namen bestimmen und prüfen, ob der Treiber bereits geladen ist
- Wenn der Treiber nicht geladen ist, wird die ausführbare Datei vom Virtual Memory Manager gemappt
- Das DRIVER_OBJECT wird vom Object Manager erzeugt
- Jeder MajorFunction Array Eintrag wird initialisiert mit einer Standart Routine
- DriverInit wird mit der DriverEntry Funktion initialisiert
- Zuletzt wird die Treiber Initialisierungsroutine aufgerufen, wo der Treiber sich selbst und die Funktionszeiger initialisieren kann

4.4.2 DEVICE_OBJECT

DEVICE_OBJECT Strukturen werden gebraucht, um logische, virtuelle oder physische Devices darzustellen und diesen spezifische Befehle zu senden.

4.4.3 FILE_OBJECT

Ein FILE_OBJECT ist die in-Memory Struktur eines offenen Objektes im I/O Manager. Zum Beispiel nach einer erfolgreichen Öffnen Anfrage einer on-disk Datei, wird ein FILE_OBJECT erzeugt. Ein FILE_OBJECT enthält ein FsContext und ein FsContext2. Theoretisch kann darin irgend etwas gespeichert werden, normalerweise werden sie aber gebraucht, um auf den FCB und CCB zu verlinken. Wird Caching verwendet, muss im FsContext das FSRTL_COMMON_FCB_HEADER verlinkt werden.

4.4.4 Volume Parameter Block (VPB)

Der VPB ist die Verbindung zwischen dem device object, welches das Dateisystem auf einem gemounteten Volume repräsentiert und dem device object, welches die physische oder virtuelle disk und die physikalische Anordnung der Daten darstellt.

4.4.5 File Control Block (FCB)

Ein FCB stellt einen offenen Stream eines on-disk Objektes im Systemspeicher dar und er wird vom Dateisystem Treiber erstellt. Ein FCB muss nicht zwingend ein File, er kann auch ein Verzeichnis, Volume oder sonst ein Objekt, das ein Benutzer öffnen kann, repräsentieren. Ein Analogon in der UNIX Welt zum FCB ist das Inode. Im FCB ist häufig eine Struktur FSRTL_COMMON_FCB_HEADER enthalten, welche für das Caching von Daten notwendig ist. Ein FCB sollte über ERSOURCEs synchronisiert werden.

4.4.6 Context Control Block (CCB)

Ein CCB wird vom Dateisystem Treiber erstellt, um eine offene Instanz eines File Streams darzustellen. Folglich existiert pro erfolgreich geöffnete Datei ein zugehöriger CCB. Im Prinzip ist der CCB eine Entsprechung zu einem user mode Datei Handle auf kernelmode Level. Ein CCB (und auch ein FCB) werden nur vom Dateisystem Treiber erstellt, verwaltet und gebraucht. Windows kennt diese Strukturen nicht.

4.5 NT Cache Manager

Der Cache Manager ist ein itegraler Bestandteil von Windows und bietet die Funktionalität Dateiinhalte zu cachen. Dazu interagiert er mit dem Dateisystem Treiber und dem NT Virtual Memory Manager.

Zu den Verantwortlichkeiten des Cache Managers zählen:

- Konsistenter, systemweiter Cache für Daten eines sekundären Speichers
- Vorlesen von Dateiinhalten
- Verzögertes Schreiben von veränderten Daten im Cache

Beim Cache Manager handelt es sich um ein komplexes Thema, das im Rahmen dieser Studienarbeit nicht weiter erläutert wird.

5 Umsetzung

In diesem Kapitel wird nach einer Einführung die Entwicklung des Treibers Schritt für Schritt nach Prototyp geordnet beschrieben. Dabei werden erst die Ziele des Prototyps zusammengefasst, danach wird grob dargestellt, was in der Windows-Treiberschnittstelle auf welche Weise Implenentiert wurde und welche erwähnenswerten Probleme aufgetreten sind. Die eingesetzten Reiser4- und VFS-Funktionen wurden im Kapitel 8 „Einführung Linux VFS“ bereits beschrieben.

Als Referenz für die Implementierung der Windows-Treiberschnittstellen diente das Buch „Windows File System Internals“ [RNAGAR01], die Windows DDK Dokumentation [MS02] inklusive FASTFAT-Treiber, sowie ein freier RomFS-Treiber [ROMFS01].

Details zur Implementierung können dem auf CD mitgeliefeten Quellcode entnommen werden.

5.1 Abbildung von Linux-Konzepten in Windows

5.1.1 Dateinamen

Standardmässig bestehen Linux Dateinamen nur aus US-ASCII-Zeichen, eine Umstellung ist zwar über Umwege möglich, wurde aber nicht weiter untersucht.

Windows XP hingegen verwendet Unicode Dateinamen. Dies stellt kein Problem dar, da eine Umwandlung von US-ASCII nach Unicode verlustfrei möglich ist.

5.1.2 Dateiattribute und Benutzerrechte

Alle drei Linux-Attribute lassen sich gemäss der Tabelle in [NDS02] direkt auf die entsprechenden Windows-Varianten oder Kombinationen davon abbilden.

Die Abbildung von Linux-Benutzerrechten in Windows ist nicht ohne weiteres möglich und wurde im Rahme dieser Studienarbeit nicht vorgenommen. Schwierigkeiten und Lösungsansätze werden im Kapitel 8.3 „Ausblick“ genauer beschrieben.

Die Änderungs- und Zugriffszeit von Dateien konnte richtig übernommen werden, die Erstellungszeit existiert unter Linux nicht.

5.1.3 Links

Hardlinks können problemlos in Windows dargestellt werden, da sich dieses Konzept allein durch das Dateisystem umsetzen lässt und das Betriebssystem nichts über die Existenz der Hardlinks wissen muss, solange diese nur gelesen werden.

Bei Softlinks wird unter Linux die Zieldatei in einem vom Root-Verzeichnis der Systempartition ausgehenden, absoluten Pfad dargestellt. Wenn also eine Partition beispielsweise in `/mnt/reiser/` gemountet ist, beginnt der Zielpfad eines Softlinks auf eine Datei dieser Partition immer mit `/mnt/reiser/`. Beim Auflösen des Links unter Windows stellt dies ein grosses Problem dar, denn die Information, in welchem Verzeichnis die Partition unter Linux gemountet war, ist nicht mehr verfügbar. Aus diesem Grund werden Softlinks von reiser4xp nicht unterstützt. Im Kapitel 8.3 „Ausblick“ werden mögliche Lösungen beschrieben.

5.2 PT1 Treiberrumpf

5.2.1 Ziele

Der Prototyp am Ende dieses Arbeitsschrittes soll ein funktionsfähiger, leerer Treiber sein, in welchen die Funktionalitäten von Reiser4 eingebaut werden können. Er umfasst alle Schnittstellen zu Windows und implementiert diese mit leeren Funktionen. Folgende Hilfsfunktionen werden ebenfalls benötigt um später das Debugging des Dateisystems zu erleichtern:

- Ein Memory-Management: Ein separates Modul erfasst die allozierten Speicherbereiche und warnt, falls einige Bereiche nicht mehr freigegeben werden. Zusätzlich können Statistiken Informationen über den Speicheraufwand des Treibers liefern.
- Logging: Das Logging kann Events ins Windows Event-Log schreiben, was vor allem für einen produktiven Einsatz wichtig wäre, wenn kein Kernel-Debugger angeschlossen ist, der Fehlermeldungen anzeigen kann. Eine Vorlage dafür findet sich im Buch „Windows NT File System Internals". Zusätzlich können strukturierte Debug-Informationen im Kernel-Debugger ausgegeben werden.

5.2.2 Aufbau

Aus der Zielstellung ergibt sich die in Illustration 8 gezeigte Einteilung des Treiberskeletts in verschiedene Module.

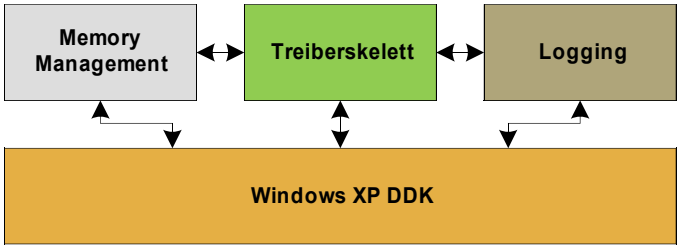

Illustration 8: Einteilung des Treibersekeletts in Module

5.2.3 Umsetzung

Laden des Treibers

Als Standarteinstiegspunkt eines kernelmode Treibers wird die Funktion `DriverEntry` vom System aufgerufen. Für ein erfolgreiches Laden des Treibers muss also mindestens diese Funktion implementiert werden. Darin wird erst ein `DEVICE_OBJECT` erstellt, welches im BS die Treiber-Implementation selbst darstellt. Auch wenn mehrere Partitionen mit einem bestimmten Dateisystem vorhanden sind, wird nur ein `DEVICE_OBJECT` für den Treiber erstellt. Danach werden alle IRPs, welche später empfangen werden sollen mit Hilfe von Funktionszeigern auf verschiedene Funktionen abgebildet. Zuletzt registriet sich der Treiber als Dateisystem-Treiber beim Betriebssystem.

Bei einem Unload des Treibers sollte `DriverUnload` aufgerufen werden, aus noch ungeklärten Gründen geschieht dies jedoch nicht.

Registrierte IRPs

Folgende Funktionen (IRPs) werden voraussichtlich in zukünftigen Prototypen benötigt und wurden bereits im Treiberrumpf auf Funktionen abgebildet:

- IRP_MJ_FILE_SYSTEM_CONTROL
- IRP_MJ_QUERY_VOLUME_INFORMATION
- IRP_MJ_CREATE
- IRP_MJ_CLEANUP
- IRP_MJ_CLOSE
- IRP_MJ_DIRECTORY_CONTROL
- IRP_MJ_QUERY_INFORMATION
- IRP_MJ_READ
- IRP_MJ_LOCK_CONTROL
- IRP_MJ_DEVICE_CONTROL
- IRP_MJ_SHUTDOWN

Eine genaue Beschreibung der einzelnen IRPs erfolgt später im Kapitel des zugehörigen Prototyps.

Erkennen einer Reiser4-Partition

Um die funktionale Anforderung FA1a zu erfüllen (Erkennen einer Reiser4 Partition), ist im Treiberrumpf ein Teil des IRPs `IRP_MJ_FILE_SYSTEM_CONTROL` implementiert. Es wird vom Betriebssystem gesendet, wenn eine Benutzerapplikation eine (möglicherweise benutzerdefinierte) Anfrage an den Treiber schickt, ein Dateisystem geladen oder eine Partition (Volume) gemountet werden soll. Momentan wird letztere Funktionalität implementiert. Diese entspricht der Minor-Function `IRP_MN_MOUNT_VOLUME`.

Beim Erhalten einer Mount-Anfrage wird zuerst versucht, den Reiser4 Super-Block von der Partition zu lesen. Gelingt dies und ist der darin enthaltene Magic-String korrekt, wird ein `VolumeDeviceObject` erstellt und beim BS angemeldet. Pro gemounteter Reiser4-Partition muss jeweils ein separates `VolumeDeviceObject` erstellt werden. Erst nach dem Mount-Vorgang erhält der Treiber später vom BS weitere IRPs, welche sich auf ein spezifisches Volume beziehen (z.B. Lesen).

Debugging, Exception Handling, Logging

Für das Debugging wird das Hilfsmakro `rfsD_dbg_trace` erstellt, welches erlaubt, für Debug Ausgaben über KdPrint ein Level festzulegen. So können, über eine globale Variable gesteuert, nur Ausgaben ab einem bestimmten Level angezeigt werden. Damit lassen sich momentan uninteressante Informationen unterdrücken, ohne dass der Quellcode geändert werden muss. Ausserdem kann eine Einrückungshierarchie erstellt werden, was die Debug Ausgabe einiges leserlicher macht. Speziell zu erwähnen sind noch die Ausgaben, welche anzeigen, dass eine Funktion aufgerufen, respektive terminiert wurde. Dies erlaubt uns später, Fehler genau zu lokalisieren.

Für das Exception Handling besteht ein globaler Exception Handler, welcher alle Exceptions fängt, einige Aufräumarbeiten (Abschluss des aktuellen IRPs, entladen des Treibers) verrichtet, die Exception im Event-Log und im Debugger ausgibt und dann einen Fehlercode zurückgibt. Da Windows das Entladen von Dateisystem-Treibern nicht unterstützt, wird im Exception-Handler der Treiber nur so deregistriet, dass er keine IRPs mehr erhält, er bleibt jedoch geladen.

Memory Manager

Der Memory-Manager ist eines der wichtigen Debug-Elemente. Er ist im globalen Datenbereich untergebracht und überwacht jede durch den Treibergemachte Allokation. Er führt diese Funktion durch, indem er über ein Makro die Allokations- und Freigabefunktionen überschreibt und durch eigene Allokationsfunktionen ersetzt. In diesen internen Funktionen wird das Memory mit der gewünschten Variante alloziert und initialisiert. Es wird zusätzlich eine Allocation Unit Struktur instanziert, welche auf die gemachte Allokation zeigt und in einer globalen Hashtabelle abgelegt wird. Wird wiederum ein Speicherbereich freigegeben wurde, dann wird die entsprechende Allocation Unit in der Hashtabelle gesucht und über diverse Checks geprüft ob die Allokation korrekt behandelt und richtig freigegeben wurde. Diese Checks beinhalten unter anderem die Überprüfung auf Buffer Over- und Underruns. Danach wird die Allocation Unit aus der Hashtabelle entfernt und der Speicherbereich im System freigegeben. Über die globale Hashtabelle kann jederzeit herausgefunden werden, welche Speicherbereiche noch alloziert sind. Diese Funktion wird vor allem beim Beenden des Treibers genutzt, nachdem alle Allokationen freigegeben sein sollten. Zusammengefasst bietet der Memory-Manager folgende Features an:

- Memory-Leaks können gefunden werden. Der Memory-Manager legt für jeden Allocate-Befehl eine Struktur an, in welcher er zahlreiche Daten zur Allokation speichert. Beim Beenden des Treibers schreibt er alle bis dorthin nicht wieder freigegebenen Allokationen auf die Konsole. Jede Allokation wird dabei durch die Quellcodedatei und Zeilennummer von welcher aus sie gemacht wurde beschrieben. Damit können normalerweise Memory-Leaks schnell gefunden und behoben werden.
- Buffer Over- und Underruns können gefunden werden. Wenn eine Allokation freigegeben wird überprüft der Memory-Manager, dass die Applikation weder vor noch hinter den allozierten Speicherbereich geschrieben hat.
- Es wird jeweils beim Freigeben einer Allokation überprüft, dass dazu auch die richtige Funktion verwendet worden ist.
- Es wird eine Statistik geführt wie viele Allokationsbefehle insgesamt ausgeführt worden sind und wie viele Speicherbereiche maximal zur gleichen Zeit alloziert waren. Diese Angaben können helfen, wenn die Applikation optimiert werden soll.

Damit der Memory-Manager funktioniert, muss er überall als erstes Include eingebunden werden, da er alle Allokations-Funktionen mit einem Makro überschreibt.

5.3 PT2 Volumeinfo

5.3.1 Ziele

In diesem Prototyp werden zwei unabhängige Dinge umgesetzt:

- Das Lesen von Partitionsinformationen wie Volume-Name oder Grösse (FA2). Hierzu muss der Master Superblock der Reiser4-Partition von der Festplatte ausgelesen werden.
- Auslesen des Verzeichnisbaums (FA3). Beliebige Verzeichnisse sollen sich darstellen und navigieren lassen. Dazu muss bereits der gesamte Reiser4 Baum richtig ausgelesen werden.

Um die Entwicklung fortsetzen zu können, müssen ausserdem die Anfragen `IRP_MJ_CREATE`, `IRP_MJ_CLEANUP` und `IRP_MJ_CLOSE` implementiert werden, welche für das Öffnen bzw. Schliessen von Dateien und Verzeichnissen verantwortlich sind.

5.3.2 Umsetzung

Öffnen und Schliessen von Dateien und Verzeichnissen

Um eine Datei zu öffnen (`IRP_MJ_CREATE`), muss erst anhand ihrer Share Access Rights die Zulässigkeit der Operation geprüft werden, falls die Datei bereits offen ist. Kann mit dem Öffnen fortgefahren werden, wird Reiser4-seitig ein Lookup auf den vom BS übergebenen Dateinamen gestartet. Es kann sich dabei um ein relatives oder absolutes Öffnen handeln. Bei ersterem wird vom BS in einem zusätzlichen Parameter das relative Verzeichnis (in Form eines Dentry) übergeben, bei letzterem ist das relative Verzeichnis implizit das Root-Verzeichnis der Partition. Wurde das Objekt gefunden, gibt das Lookup das entsprechende Dentry zurück, in dessen Inode nun alle Windows-seitig benötigten Datenstrukturen initialisiert werden. Schliesslich wird ein CCB, welches einem Datei-Handle im Usermode entspricht, für das gefundene Objekt erstellt und mit dem gefundenen Dentry assoziiert. Damit das Inode und das CCB des Objekts in späteren Anfragen verfügbar sind, werden sie mit der Windows-Struktur `FILE_OBJECT` über zwei Zeiger verknüpft. Das `FILE_OBJECT` wird von nun an bei jeder Anfrage des BS als Parameter an den Treiber übergeben, das BS selber kennt und benutzt aber weder das CCB noch das Inode.

Das Schliessen von Dateien und Verzeichnisse verläuft analog zum Öffnen. Dazu wird vom BS erst `IRP_MJ_CLEANUP` und danach `IRP_MJ_CLOSE` gesendet, die eigentliche Arbeit wird in letzterem erledigt. Erwähnenswert ist hier, das bei jedem Close das entsprechende CCB geschlossen und freigegeben wird, das Inode und das Dentry aber werden erst gelöscht, wenn keine weiteren Handles mehr auf das Objekt existieren. Dies wird treiberseitig durch einfaches Reference-Counting sichergestellt. Hier können Memory-Leaks entstehen, wenn Dateien geöffnet aber nicht mehr geschlossen werden.

Lesen von Partitionsinformationen

Partitionsinformationen werden vom Betriebssystem über das IRP `IRP_MJ_QUERY_VOLUME_INFORMATION` angefordert. Dabei können verschiedene Infomationen in Form von InfoClasses verlangt werden, je nach InfoClass muss der übergebene Rückgabe-Buffer mit einer anderen Struktur gefüllt werden. Die meisten Informationen können direkt aus dem Master Superblock übernommen werden und sind nicht weiter erwähnenswert, interessant ist aber das Feld `FileSystemAttributes` der InfoClass `FS_INFORMATION_CLASS`. Hierdurch wird dem BS mitgeteilt, welche Eigenschaften das Dateisystem hat und welche Konzepte (z.B. Streams) unterstützt werden. Reiser4xp signalisiert hier, dass es sich um ein readonly Dateisystem handelt (`FILE_READ_ONLY_VOLUME`) und dass die Gross- und Kleinschreibung von Dateinamen berücksichtigt wird (`FILE_CASE_SENSITIVE_SEARCH` und `FILE_CASE_PRESERVED_NAMES`).

Auslesen des Verzeichnisbaums

Soll der Inhalt eines Verzeichnisses ausgelesen werden, schickt das Betriebssystem die Anfrage `IRP_MJ_DIRECTORY_CONTROL` an den Treiber (Minor Function `IRP_MN_QUERY_DIRECTORY`). Dabei wird das auszulesende Verzeichnis als Parameter geliefert, neben weiteren Parametern, die bestimmen, welchem Such-Pattern die Verzeichniseinträge entsprechen sollen, ob nur ein Verzeichniseintrag zurückgegeben werden darf und ob die Suche fortgesetzt wird oder von vorne gestartet werden muss. Bei einer fortgesetzten Suche können einzelne Parameter fehlen, dann muss auf im vorherigen Durchgang gespeicherte Werte zurückgegriffen werden. Die Rückgabe einzelner Verzeichniseinträge scheint das BS zu benutzen, um zu sondieren, ob das Zielverzeichnis überhaupt ausgelesen werden kann und ob der bereitgestellte Rückgabebuffer genug gross ist.

Der Dateisystemtreiber soll nun die Parameter auswerten und den Rückgabebuffer mit den gewünschten Verzeichniseinträgen füllen. Wie bei den Partitionsinformationen wird auch hier über eine InfoClass angegeben, welche Inforamationen zurückgegeben werden müssen. Neben dem Dateinamen werden normalerweise auch diverse Dateiattribute verlangt.

Durch die Kombination der Parameter, der verschiedenen InfoClasses und durch die Tatsache, dass der Rückgabebuffer selten für alle Einträge genug Platz bietet ist es relativ schwierig, die Anfrage fehlerfrei umzusetzen. Grundsätzlich werden folgende Schritte unternommen:

- Auswertung der Paramter. Danach ist bekannt, welches das Zielverzeichnis ist, wo die Suche gestartet werden soll und wie das Such-Pattern lautet. Die verlangte InfoClass steht ebenfalls fest.
- In einer Schleife wird der Rückgabebuffer solange mit Einträgen gefüllt, bis er voll ist, oder keine weiteren Einträge vorhanden sind. Um die Inforamationen für die Einträge zu gewinnen, wird erst über die VFS-Funktionen `readdir` und `llseek` der Name des Eintrags ermittelt, danach wird daraus mit Hilfe eines Lookups das zugehörige Inode ermittelt. Dort werden schliesslich die benötigten Attribute ausgelesen. Für jeden Verzeichniseintrag ein Lookup zu starten kostet recht viel Zeit, ist aber nötig, weil unter Linux bei `readdir` keine Datei-Attribute zurückgegeben werden und das Inode deshalb noch nicht geladen wird.
- Beim Abschluss der Anfrage wird dem BS mitgeteilt, ob weitere Einträge folgen, die Aufzählung am Ende ist oder der Rückgabebuffer zu klein war um überhaupt einen einzelnen Eintrag zu fassen.

Je nach Rückgabewert kann die Anfrage nun vom BS durch das Senden eines weiteren `IRP_MN_QUERY_DIRECTORY` fortgesetzt werden.

Da neben dem Namen der einzelnen Verzeichniseinträge auch deren Attribute zurückgegeben werden müssen, musste gezwungenermassen bereits in diesem Prototyp das Auslesen von Dateiattributen implementiert werden, obwohl dies erst für später geplant war. Weil dadurch schon alle Dateiattribute verfügbar waren, wurde auch gleich das `IRP_MJ_QUERY_INFORMATION` umgesetzt. Es verhält sich im Grunde gleich wie `IRP_MN_QUERY_DIRECTORY`, ausser, dass jeweils nur die Attribute einer einzelnen Datei verlangt werden und die Dateien vorher schon geöffnet wurden (es existiert also schon ein Inde usw.).

5.3.3 Probleme

Während der Umsetzung der neuen Funktionalität sind Schwierigkeiten mit Teilen des vorherigen Prototyps aufgetaucht:

- Das Exception-Handling konnte nicht wie geplant eingesetzt werden, da Exceptions im Kernelmode wirklich schwerwiegende Fehler darstellen und nicht für die Rückgabe eigener Fehlerwerte „missbraucht“ werden sollten. Trotzdem können sie im normalen Betrieb auftauchen und sind normalerweise kein Grund, den Treiber zu stoppen. Aus diesem Grund wurde der Exception-Handler neu geschrieben. Neu gibt er eine Fehlermeldung im Debugger aus und schliesst das aktuelle IRP mit einem entsprechenden Fehlercode ab.
- Das Logging im Windows Event-Log wurde wieder abgeschaltet, denn einzelne Fehlercodes im Log ohne den Kontext des Fehlers (z.B. ein Stack-Trace) sind völlig wertlos. Während der Entwicklungsphase ist das Fehlen des Loggings egal, da der Kernel-Debugger allein genug Informationen bereitstellt. Ein Problem würde erst auftreten, wenn der Treiber ausgeliefert wird und bei einem Benutzer einen Systemabstutz verursacht. Als Entwickler hätte man keine Informationen über den Fehler und es wäre schwierig, ihn zu reproduzieren. Deshalb müsste bei einem kommerziellen Treiber hier eine Lösung gefunden werden.

- Das Makro `rfsD_dbg_trace` musste um Synchronisation mittels Spinlocks erweitert werden, da die Debugausgaben durch Multithreading durcheinander gebracht wurden. Durch die extrem grosse Anzahl der Debugausgaben im gesamten Code kam es jedoch zu Deadlocks und anderen seltsamen Problemen. Aus diesem Grund wurde die automatisch Protokollierung der Funktionsein- und austritte abgeschaltet, was die Ausgabenmenge erheblich reduziert und das Deadlock-Problem gelöst hat. Nachteile entstehen dadurch nicht, da die Funktionsaufrufe auch im Callstack des Kernel-Debuggers ersichtlich sind.

5.4 PT3 Recognizer

Der geplante Recognizer wurde nicht implementiert, weil sein Nutzen fragwürdig ist. Denn man kann davon ausgehen, dass jemand, der einen Dateisytemtreiber für Reiser4 installiert, auch mindestens eine Reiser4-Partition besitzt. Sobald nun der Windows Explorer gestartet wird, versucht dieser von der Partition zu lesen und der gesamte Treiber muss in jedem Fall geladen werden, auch wenn ein Recognizer vorhanden ist.

5.5 PT4 Lesen

5.5.1 Ziele

In diesem Prototyp wird das Lesen von Dateien vollständig implementiert. Dazu wird erst die normale, IRP-basierte Anfrage mit Caching- und Pagefile-Unterstützung umgesetzt, danach folgt das als Sekundärziel geplante FastIO. Um FastIO korrekt zu realisieren, werden auch Byte-Range-Locks implementiert.

Das ursprünglich ebenfalls hier vorgehene Auslesen von Dateiattributen wurde bereits im vorherigen Prototyp umgesetzt.

Zusätzlich werden die IRPs `IRP_MJ_DEVICE_CONTROL` (weiterleiten von Anfragen an den Festplattentreiber) und `IRP_MJ_SHUTDOWN` (herunterfahren des Treibers) implementiert, die zwar keiner funktionalen Anforderung entsprechen, aber für das korrekte Verhalten des Treiber nötig sind.

5.5.2 Umsetzung

IRP-basiertes Lesen

Soll eine Datei gelesen werden, schickt das BS die Anfrage `IRP_MJ_READ`. Als Parameter wird unter anderem das zu lesende `FILE_OBJECT` zusammen mit Offset und Länge übergeben. Zuerst muss vom Treiber sichergestellt werden, dass der zu lesende Bereich innerhalb der Datei liegt. Da die Grösse des zugegörigen Inodes direkt abrufbar ist, stellt das kein Problem dar. Danach wird geprüft ob Byte-Range-Locks auf der Datei liegen und die Operation gegebenenfalls abgebrochen werden muss.

Die Leseanfrage kann von hier an auf zwei Arten bedient werden:

- Wenn Caching für die Datei gewünscht ist (Normalfall) wird die Anfrage direkt an den Windows Cache Manager weitergeleitet. Danach ist die Arbeit für den Treiber erledigt. Wird die Datei zum ersten Mal gelesen, muss vorher das Caching initialisiert werden. Der genaue Funktionsweise des Cache Managers ist in [RNAGAR01] beschrieben.
- Falls kein Caching gewünscht ist (entweder explizit von einer Applikation verlangt oder wenn der Cache Manager den Cache einer Datei füllen will), wird die VFS-Funktion `read` für das zugehörige Inode aufgerufen.

In beiden Fällen wird zum Schluss der übergebene Rückgabebuffer mit den Daten gefüllt.

Wenn eine Applikation ein Memory Mapping für eine Datei erstellen will, muss diese als Pagefile eingelesen werden. Das Verfahren ist gleich wie das oben beschriebene, mit Ausnahme, dass die Synchronisation ein wenig anders erfolgen muss. Datails dazu sind im Quellcode ersichtlich.

Byte-Range-Locking

Das Erstellen und Freigeben von Byte-Range-Locks geschieht über die Anfrage `IRP_MJ_LOCK_CONTROL`. Dafür muss nicht viel getan werden, das Meiste erledigt die DDK Funktion `FsRtlProcessFileLock`.

FastIO

Neben dem Lesen beinhaltet FastIO unter anderem auch Funktionen für das Abfragen von Dateiattributen und für die Verwaltung von Byte-Range-Locks. Da das BS auf diese zurückgreifen will, sobald FastIO eingeschaltet ist, müssen alle implementiert werden. Die vielleicht wichtigste Funktion ist aber `rfsS_fast_io_check_if_possible`. Sie wird vom BS aufgerufen um festzustellen, ob FastIO für eine Datei möglich ist. Das ist immer dann der Fall, wenn keine Byte-Range-Locks auf der Datei liegen.

Lesen über FastIO erfolgt direkt über die Windows DDK Funktion `FsRtlCopyRead`, hier muss nichts weiter getan werden. Die Funktionen für das Abfragen von Dateiattributen sind fast identisch mit den IRP-basierten Varianten. Die Locking-Funktionalität setzt und entfernt Byte-Range-Locks auf Dateien und ist nicht weiter interessant.

Device Control und Shutdown

Das IRP `IRP_MJ_DEVICE_CONTROL` wird gesendet, wenn eine Applikation IOCTL Anfrage startet. Die Anfrage kann vom Dateisystemtreiber selber behandelt, oder an den Festplattentreiber weitergeleitet werden. Reiser4xp leitet alle Anfragen weiter.

`IRP_MJ_SHUTDOWN` signalisiert, dass das BS heruntergefahren wird und der Treiber entladen werden soll. Im vorliegenden Treiber wird hier versucht, noch einige Resourcen freizugeben, damit am Schluss eine brauchbare Memory Leak Statistik entsteht.

5.5.3 Probleme

Das grösste Problem in diesem Prototyp sind die vielen entstehenden Memory Leaks. Zu Grossteil entstehen sie, weil Dateien (vom Windows Explorer) geöffnet, gelesen aber nicht wieder geschlossen werden. Dabei handelt es sich laut [OSR02] um normales Verhalten des Betriebssystems, das auf diese Weise versucht, eine Art Caching von Datei-Handles vorzunehmen.

5.6 Installation

Während der Entwicklung wurde der Treiber auf dem Testsystem jeweils von Hand in der Windows Registry eingetragen und über die Kommandozeile gestartet. Die Zuordnung eines Laufwerksbuchstaben an die Reiser4-Partition erfolgte ebenfalls über die Registry. Das stellte im Entwicklungsprozess kein Problem dar, kann aber einem Benutzer des Treibers nicht zugemutet werden. In diesem Kapitel wird beschrieben, was unternommen wurde, um die Installation so einfach wie möglich zu gestalten.

5.6.1 Zuweisung von Laufwerksbuchstaben

Die grösste Schwierigkeit bei der Zuordnung eines Laufwerksbuchstaben an eine Partition ist, dass diese Prozedur sehr schlecht Dokumentiert ist. Aus diesem Grund ist es speziell in diesem Abschnitt möglich, dass gemachte Angaben ungenau oder falsch sind. Die Informationen stammen grösstenteils aus der MSDN [MS03]

In den Windowsversionen vor Windows 2000 lief die Zuordnung nur über die sogenannten DosDevices. Dabei können über die Registry (`HKLM\System\CurrentControlSet\Control\Session Manager\Dos Devices`) Laufwerksbuchstaben mit Partitionen verknüpft werden. Die Partitionen werden dabei in der Form `\Device\HarddiskX\PartitionY` angegeben. Das Selbe kann auch über die Windows-Funktion `DefineDosDevice` erfolgen, allerdings verschwindet so die Zuordnung nach einem Neustart wieder. Die Einträge in der Registry sind persistent, dafür treten sie erst nach einem Neustart in Kraft.

Seit Windows 2000 sollte die Zuordnung vorzugsweise über die Funktion `SetVolumeMountPoint` vonstatten gehen. Die Funktion benötigt als Parameter den gewünschten Laufwerksbuchstaben und einen Windows Volume-Identifier der Form `\\?\Volume{GUID}`.

Eine zusätzliche Möglichkeit ist die direkte Kommunikation mit Windows Mount Manager über IOCTLs. Dies wurde im Rahmen dieser Studienarbeit nicht weiter untersucht, genauere Informationen sind in [MS03] zu finden.

Für reiser4xp wurde als erstes versucht, die Zuweisung mit der moderneren Variante `SetVolumeMountPoint` vorzunehmen. Dies ist an der Tatsache gescheitert, dass der Reiser4-Partition vom BS kein Volume-Identifier zugeteilt wurde. Interessanterweise war bei Partitionen auf USB-Festplatten ein Volume-Identifier vorhanden. Dort ist die Zuordnung aber gar nicht nötig, weil USB-Festplatten automatisch einen Laufwerksbuchstaben erhalten. Deshalb wurde ein Programm geschrieben (DriveLetterUtil), welches `DefineDosDevice` und den oben beschriebenen Registry-Eintrag für die Zuordunung des Laufwerksbuchstaben benutzt. Auf diese Weise ist das Laufwerk sofort verfügbar und bleibt auch nach einem Neustart vorfügbar.

DriveLetterUtil hat leider einige Schwächen. So werden teilweise Lauftwerksbuchstaben von Netzwerklaufwerken dem Benutzer als frei gemeldet, obwohl diese schon in gebrauch sind. Ebenso werden Reiser4-Partitionen auf Removable Media nicht als solche erkannt und zur Zuordnung angeboten, obwohl dies über DosDevices gar nicht möglich ist. Wieso die Erkennung nicht funktioniert, konnte nicht festgestellt werden.

5.6.2 Installationsvoraussetzungen

Für eine problemlose Installation des Treibers und die teilweise Ersetzung des Recognizers wurden verschiedene Arten eines Installers ausprobiert.

Grundsätzlich muss die kompilierte Treiberdatei im `C:\Windows\System32\Drivers` Ordner liegen und ein Service in der Registry eingetragen sein, welcher die Datei beim Systemstart lädt. Zusätzlich musste ein Tool entwickelt werden (DriveLetterUtil), damit einer neu erkannten reiser4 Partition manuell ein Laufwerksbuchstaben zugewiesen werden kann, da Windows dies nur für FAT oder NTFS formatierte Partitionen zur Verfügung stellt. Das Tool soll nach der Installation gestartet werden.

Für alle getesteten Varianten ist eine Treiber INF Datei notwendig:

```
[Version]
Signature                = "$WINDOWS NT$"
Provider                 = %Provider%
DriverVer                = 02/01/2006,1.0.0.4
DriverPackageType        = ClassFilter
```

```
DriverPackageDisplayName  = %Desc%

[DestinationDirs]
rfs.Files                 = 12

[SourceDisksFiles]
reiser4xp.sys             = 1

[SourceDisksNames]
1                         = %InstDisk%

; INSTALL STUFF
[DefaultInstall.NT]
CopyFiles                 = rfs.Files
AddReg                    = rfs.AddReg

[DefaultInstall.NT.Services]
AddService                = %Name%,%SPSVCINST_ASSOCSERVICE%,rfs.AddService,rfs.EventLog

; OWN SECTIONS
[rfs.AddReg]
HKLM,%RegPath%,"DisplayName",%REG_SZ%,%Name%
HKLM,%RegPath%,"ErrorControl",%REG_DWORD%,%SERVICE_ERROR_NORMAL%
HKLM,%RegPath%,"Group",%REG_SZ%,"File System"
HKLM,%RegPath%,"ImagePath",%REG_EXPAND_SZ%,"%12%\reiser4xp.sys"
HKLM,%RegPath%,"Start",%REG_DWORD%,%SERVICE_SYSTEM_START%
HKLM,%RegPath%,"Tag",%REG_DWORD%,1
HKLM,%RegPath%,"Type",%REG_DWORD%,%SERVICE_FILE_SYSTEM_DRIVER%

[rfs.AddService]
DisplayName               = %Name%
Description               = %Desc%
ServiceType               = %SERVICE_FILE_SYSTEM_DRIVER%
StartType                 = %SERVICE_SYSTEM_START%
ErrorControl              = %SERVICE_ERROR_NORMAL%
ServiceBinary             = %12%\reiser4xp.sys

[rfs.EventLog]
AddReg                    = rfs.EveReg

[rfs.EveReg]
HKR,,"EventMessageFile",%REG_EXPAND_SZ%,"%%SystemRoot%%\System32\IoLogMsg.dll;%%SystemR
oot%%\system32\Drivers\reiser4xp.sys"
HKR,,"TypesSupported",%REG_DWORD%,7

[rfs.Files]
reiser4xp.sys
```

Die genaue Spezifikation ist der DDK Hilfe zu entnehmen. Um die Syntax zu überprüfen kann das mit der DDK mitgelieferte ChkInf verwendet werden (benötigt jedoch Perl). Das Schreiben einer funktionierenden INF Datei stellte sich als sehr aufwändig heraus, da nicht immer ganz nachvollziehbar ist, was Windows genau braucht oder macht.

5.6.3 Installation per Rechtsklick

Wenn die DefaultInstall Section in der INF Datei vorhanden ist und die sys Datei im gleichen Ordner liegt, kann der Treiber über einen Rechtsklick > Installieren installiert werden. Dies ist jedoch nicht sehr komfortabel und die Deinstallation verläuft nicht ganz reibungslos.

5.6.4 Installation über Batch File

Über den Funktionsaufruf InstallHinfSection in der rundll kann der Treiber auch mittels INF Datei installiert werden.

```
> rundll32.exe setupapi.dll,InstallHinfSection
      DefaultInstall 129 .\reiser4xp.inf
```

Dieser Aufruf könnte in eine Batch Datei verpackt werden. Der Deinstallationsvorgang verläuft ähnlich, mit Angabe der DefaultUninstall Section. 129 setzt den Installationspfad auf den Ort der INF und fordert auf jeden Fall einen Neustart des Systems.

5.6.5 Verpacken in CAB

Um das Problem der losen Dateien bei den beiden vorhergehenden Varianten zu lösen, kann das mit Windows ausgelieferte IExpress verwendet werden, mit dem ein selbst extrahierendes CAB File erstellt werden kann. Bei der zweiten Variante kann zusätzlich angegeben werden, dass direkt nach dem Entpacken das DriveLetterUtil gestartet wird.

5.6.6 Microsoft Driver Install Framework 2.01

Eine schöne Möglichkeit ist das Driver Install Framework (DIFx) von Microsoft. Es gibt in diesem Framework drei Installationsarten:

- DIFxAPI: Für Installationen aus eigenem Code
- DIFxApp: Installation mit einem MergedModul oder der WiXLib
- DPInst: Installation mit einem konfigurierbaren Installer

Bei allen Dateien brauche es die erwähnte INF Datei auch.

DPInst

Mit DPInst wird ein Wizard gestartet, welcher sehr einfach (per XML Datei) angepasst werden kann (Sprachen, eigenes Bild, EULA). Der Nachteil ist, dass die Dateien auch lose sind und ebenfalls mit IExpress gepackt werden müssten. Eine Deinstallation kann sehr einfach über die Software in der Systemsteuerung vorgenommen werden kann. Der Nachteil ist, dass das DriveLetterUtil nur einmal nach dem Installieren ausgeführt werden kann und dann kein Ummappen der Laufwerkbuchstaben mehr möglich ist.

DIFxApp

Die wohl beste Möglichkeit bietet wohl das MergedModule. Damit kann in VisualStudio ein Setup Projekt erstellt werden, das direkt die neuste Version des reiser Codes kompilieren kann und einbinden, auch die nueste Version des DriveLetterUtil wird automatisch eingebunden. Das MergedModule des DIFx wird hineinkopiert und kann mit Hilfe der INF Datei den Treiber installieren und registrieren. Es kann eine ausführliche Installationsanleitung integriert werden und DriveLetterUtil wird noch während der Installation ausgeführt. Ausserdem ist es nach der Installation über das Startmenü verfügbar. Die generierte msi entspricht dem aktuellsten Standard von Microsoft Installern und kann auch sehr einfach wieder komplett Deinstalliert werden.

Ein kleiner Nachteil ist, dass mit ORCA (der msi table editor, enthalten in den Microsoft Debugging Tools) noch manuell eine Referenz auf die INF Datei erstellt werden muss. ORCA zeigt die ganze msi table (eine Art Datenbank) an. In der Tabelle Component sind alle Dateien, welche im msi enthalten sind aufgeführt. In der Tabelle MsiDriverPackages (mit dem MergedModule eingefügt) muss ein neuer Eintrag erstellt werden, der auf das Component der INF Datei verweist. Zusätzlich kann ein Flag angegeben werden:

Wert	Bedeutung
0	Standardwert.
1	Installiert den Treiber auch wenn eine neuere Version vorhanden ist.
2	Fordert nicht auf, das zum Treiber gehörende Gerät anzuschliessen.
4	Verhindert den Eintrag in die Software Systemsteuerung.

Wert	Bedeutung
8	Erlaubt die Installation von unsignierten Treibern.
16	Entfernt die Binärdatei bei der Deinstallation.

Bei der Installationsdatei von reiser4xp wurde 16 + 8 + 2 = 26 als Flag verwendet. Wünschenswert wäre, wenn die Installation zu einem Neustart auffordern würde, was momentan nicht der Fall ist. Die Gründe dafür konnten nicht ermittelt werden.

6 Analyse Reiser4

6.1 Allgemeine Bemerkungen

6.1.1 Quellen

Grundlage dieses Abschnitts waren verschiedene im folgenden aufgelistete Artikel:

- Reasons why Reiser4 is great for you [NSYS01]
- B-Baum [WIKI01]
- Overview of the Virtual File System [RGOOCH]
- Linux Kernelarchitektur [MAU01]

Diese Dokumente stellen zugleich die Basis für die vorliegende Dokumentation dar. Da es aufwändig ist, sich die Grundlagen aus allen Dokumenten zusammenzusuchen wurden diese, wenn es nötig war übersetzt und übernommen. Teilweise wurden eigene Erläuterungen hinzugefügt. Das heisst dass Teile der nachfolgenden Abschnitte aus den obenstehenden Dokumenten übernommen worden sind.

6.1.2 Darstellung

Für die Beschreibung von Strukturen werden 2 Fälle unterschieden. Es gibt Strukturen, welche auf der Festplatte abgelegt sind. Andere wiederum werden nur während der Treiberausführung benötigt und sind daher nur im Hauptspeicher zu finden. Während dieser Einführung und der späteren detaillierten Analyse werden diese beiden Arten von Strukturen durch verschiedene Notationen dargestellt. Die auf die Festplatte abgebildeten Strukturen werden durch eine eigene Notation beschrieben und alle anderen Strukturen in C-Code dargestellt.

6.2 Das Grundkonzept von Reiser4

Das wichtigste Konzept von Reiser4 ist die Erweiterbarkeit. Ausser dem Basiskonzept, dem balancierten Suchbaum und der Schlüsselstruktur, gibt es praktisch keine Funktion welche nicht (zumindest theoretisch) durch die Implementierung eines eigenen sogenannten Plugins umdefiniert oder erweitert werden kann. Plugins können nicht wie bei normalen Userprogrammen während der Laufzeit eingebunden werden sondern müssen in den Treiber hineinkompiliert werden. Sie besitzen dadurch gewisse Restriktionen welche eingehalten werden müssen, damit bereits auf die Festplatte geschriebene Daten nicht verloren gehen. Das Konzept der Plugins hat folgende Vor- und Nachteile:

- Das System bietet eine enorme Flexibilität. Beispielsweise werden Dateinamen normalerweise im ASCII-Format gespeichert. Unter Windows XP werden jedoch normalerweise Unicode-Strings verwendet. Damit auch unter ReiserFS Unicode-Namen gespeichert werden müsste nur ein zusätzliches Metadaten-Plugin definiert werden welches diesen Unicode-Namen speichert. Die Grundstruktur des Treibers würde davon nicht betroffen.
- Der Nachteil ist, dass das Grundsystem des Treibers durch die Plugins und die zusätzliche Indirektion komplizierter wird. Die Einarbeitungszeit welche nötig ist um den Treiber zu verstehen wird dadurch vergrössert.
- Die grosse Flexibilität macht es schwierig den Treiber als ein Ganzes zu dokumentieren. Es kann sein, dass dem Leser scheinbar zusammenhangslose Konzepte

begegnen welche erst später einen Sinn und Verbindungen zu anderen Konzepten bekommen.

6.3 Der balancierte Suchbaum

Grosse Teile der folgenden Behandung des Reiser4 Suchbaumes sind [NSYS01] entnommen und frei übersetzt. Dieser Artikel ist auch die Quelle aller nachfolgenden Abbildungen von Bäumen, B-Bäumen und B⁺-Bäumen.

6.3.1 Die Elemente eines Suchbaumes

Leaves, Internal Nodes und Levels

Ein Baum besteht aus Leaves und Internal Nodes. Ein Leaf besitzt keine Child-Knoten während ein Internal Node Children besitzt. Unter Level wird die Ebene verstanden auf welcher ein Knoten gefunden werden kann. Da der Reiser4-Baum nicht von Oben nach Unten, sondern von Unten nach Oben wächst befinden sich die Leaf-Nodes auf Level eins. Darauf folgt die erste Ebene mit Internal Nodes auf Level zwei. Die Numerierung geht so weiter bis zum Root Node welcher den höchsten Level besitzt.

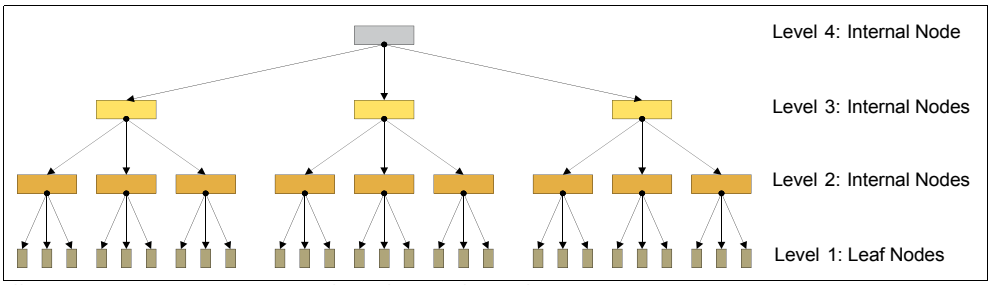

Illustration 9: Leaves, Internal Nodes und Level

Items

Items sind keine direkten Elemente des Baumes. Sie sind aber trotzdem nötig um die weiteren Konzepte des Baumes zu verstehen. Ein Item ist ein beliebiges Element des Dateisystems. Es könnte Beispielsweise die Einträge eines Verzeichnisses, die Daten einer Datei oder Metainformationen über eine Datei beinhalten. Für die folgende Behandlung des Baumes spielt der Inhalt keine Rolle, es ist nur wichtig, dass ein Item einen Schlüssel besitzt und nach diesem Schlüssel einem Knoten im Baum zugeordnet ist.

Units

Units sind Elemente welche in Items enthalten sind. Units sind atomare Einheiten, das heisst sie sind nicht weiter aufteilbar. Sie definieren eine bestimmte Einheit welche nützlich ist für das übergeordnete Item. Eine Unit kann beispielsweise folgende Definition haben:

- Eine Unit ist ein Byte: für Datei- und andere binäre Objekte
- Eine Unit ist ein Verzeichniseintrage: für Verzeichnis-Objekte

Schlüssel

Schlüssel von Reiser4 bestehen aus einem Array von 64-Bit Werten. Abhängig davon wie Reiser4 kompiliert worden ist besteht dieses Array entweder aus drei Komponenten (kleine Schlüssel) oder aus vier Komponenten (grosse Schlüssel). Der

Treiber speichert auf der Partition in einem Bit-Feld ob er kleine oder grosse Schlüssel verwendet.

Formatted Nodes und Unformatted Leaves

Jeder Knoten im Reiser4-Baum kann optional Items enthalten. Als Formatted Nodes werden diejenigen Knoten des Baumes bezeichnet welche Items enthalten. Da Zeiger auf weitere Knoten im Baum in Items gespeichert werden sind alle Internal Nodes zugleich erzwungenermassen auch Formatted Nodes. Trotzdem dürfen Internal Nodes nicht mit Formatted Nodes gleichgesetzt werden, da es auch Leaves geben kann welche Formatted Nodes sind.

Bei grossen Dateien kann es sinnvoll und effizienter sein die Dateien ohne Verwendung von Items zu speichern. Dies betrifft bei Reiser4 standardmässig Dateien welche grösser sind als 16 Kilobytes. Als Unformatted Leaves werden Leaf Nodes bezeichnet welche solche Rohdaten ohne Formatierungsinformationen beinhalten. Hans Reiser bezeichnet solche Unformatted Leaves auch kurz als Unfleaves.

Gemäss Definition von Hans Reiser können Rohdaten von Dateien nur im Leaf-Level gespeichert werden. Diese Tatsache bringt mit sich, dass Unformatted Nodes nur im Leaf-Level vorkommen können.

Node Pointer und Extent Pointer

Als Node Pointer werden Zeiger bezeichnet welche von Internal Nodes auf weitere Internal Nodes zeigen. Als Extent Pointer wird ein Zeiger bezeichnet welcher von einem Internal Node auf Unformatted Leaves zeigt. Zwischen den beiden Zeigertypen wird unterschieden, da sie auf der Festplatte unterschiedlich abgelegt sind.

Twigs, Branches und Root Node

Als Twigs werden die Parent-Knoten von Leafes bezeichnet. Nur Twigs können Extent Pointers beinhalten und sie befinden sich immer auf Level 2. Als Branch Nodes werden Internal Nodes bezeichnet welche keine Twigs sind. Der Root Node ist derjenige Knoten des Baumes mit dem höchsten Level.

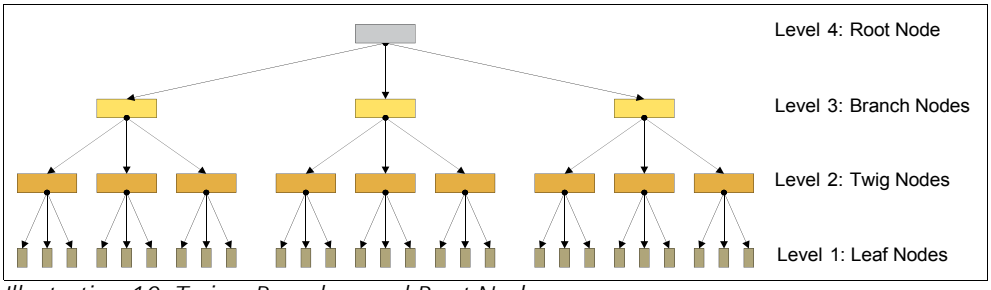

Illustration 10: Twigs, Branches und Root Node

Minimale Höhe des Baumes

Um die Algorithmen zur Traversierung und Balancierung des Baumes zu vereinfachen wurde definiert, dass ein Reiser4-Baum immer mindestens die Höhe 2 besitzt. Das heisst der Root Node ist immer ein Internal Node.

Verzweigungsgrad

Der Verzweigungsgrad (in englisch „Fanout-Rate“ genannt) bestimmt wie viele Children ein bestimmter Knoten hat. Beispiele sind:

- Ein Baum mit der Verzweigungsgrad 1 sieht aus wie eine einfach gelinkte Liste
- Ein Baum mit der Verzweigungsgrad 2 ist ein binärer Baum.

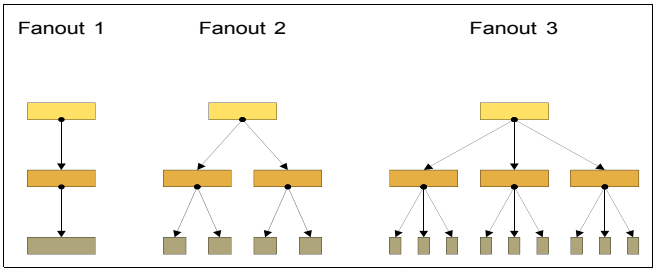

Illustration 11: Fanout von 1, 2 und 3

6.3.2 Der Reiser4 Dancing Tree

Der Reiser4 Baum ist die Folge einer Evolution welche beim B-Baum angefangen hatte. Dieser wurde zum B^+-Baum, Reiser3-Baum und schlussendlich zum Reiser4-Baum weiterentwickelt. Dieser Abschnitt folgt der historischen Entwicklung und stellt die Ideen und Gründe für die verschiedenen Verbesserungen dar.

Der B-Baum

Der folgende Abschnitt ist mehr oder weniger eine Zusammenfassung des Artikels unter [WIKI01].

Der Hintergrund für die Entwicklung des B-Baumes war die Notwendigkeit eine Datenstruktur zu entwickeln welche die Eigenschaften von persistentem Speicher (zum Beispiel Festplatten) berücksichtigt. Dies war notwendig für die Entwicklung von Datenbanksystemen oder Dateisystemen bei welchen die grosse Datenmenge verhindert, dass alle Daten zeitgleich in den Hauptspeicher passen. Beim Zugriff auf Festplatten stellt das mechanische Umpositinieren der Leseköpfe die grösste Zeitverzögerung dar, während das Auslesen danach relativ schnell und ohne Mitwirkung des Prozessors über DMA ausgeführt werden kann. Die Folge davon ist, dass sich binäre Suchbäume für die Speicherung auf solchen Datenträgern nicht eignen da sie eine grosse Menge wahlfreier Zugriffe (Random-Access) auf das Speichermedium benötigen wobei jeder Zugriff eine Umpositionierung des Lesekopfes zur Folge hätte.

Eine weitere Eigenschaft der Hardware ist, dass beim Lesen immer mindestens ein Sektor (512 Bytes) ausgelesen werden muss. B-Bäume berücksichtigen diese Tatsache damit, dass die Grösse eines Knotens immer ein vielfaches der Sektorgrösse sein muss. Dadurch können B-Bäume den Verzweigungsgrad (das heisst die Anzahl von Verweisen auf Kindknoten) im Vergleich zu binären Suchbäumen drastisch steigern. Der grosse Verzweigungsgrad bringt einerseits eine Reduzierung der Höhe des Suchbaumes mit sich und andererseits wird die Häufigkeit der Balance-Operation gesenkt.

Im praktischen Fall kann bei B-Bäumen sogar von einer konstanten Anzahl von Festplatten-Zugriffen pro Operation ausgegangen werden. Da ein Baum mit Fanout-Rate t und Höhe h $t^{h+1}-1$ Schlüssel speichern kann, können bei einer Fanout-Rate von zum Beispiel 1024 und einer Höhe von 4 bereits $1024^{4+1}-1=2^{50}-1$ Schlüssel gespeichert werden. Oft werden dann eine gewisse Anzahl Level ausgehend vom Root Node zusätzlich im Hauptspeicher gehalten was die Anzahl von Festplattenzugriffen weiter senkt.

Die folgende Grafik zeigt das Beispiel eines B-Baumes mit zwei Leveln.

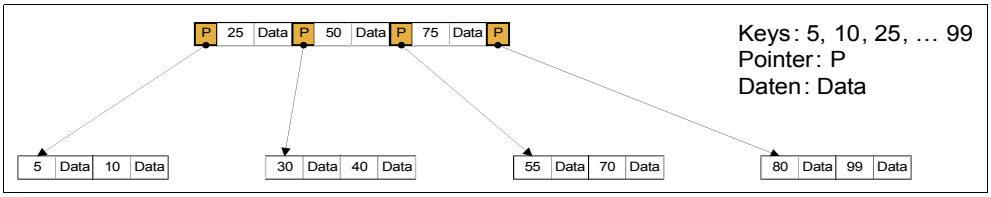

Illustration 12: B-Baum

Der B⁺-Baum

Der B⁺-Baum ist eine Variation des B-Baumes. Anders als beim B-Baum speichert er Objekte nur in Leaves während Internal Nodes nur Schlüssel und Pointer auf Child Knoten enthalten. Dadurch kann der Verzweigungsgrad im Vergleich zu B-Bäumen weiter gesteigert und dadurch die Anzahl der Festplattenzugriffe weiter verringert werden. Zusätzlich können mehr Internal Nodes intern im Cache gehalten werden, da insgesamt weniger Internal Nodes existieren.

Die folgende Grafik zeigt das Beispiel eines B⁺-Baumes mit zwei Leveln.

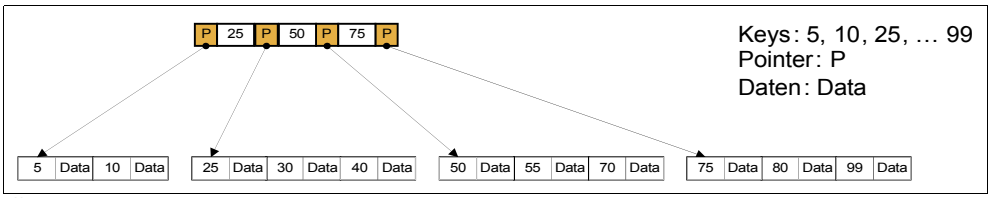

Illustration 13: B⁺-Baum

6.3.3 Der Reiser3-Baum und BLOBS

Zusammengefasst war die Grundidee der B⁺-Baume einerseits den Verzweigungsgrad zu erhöhen und andererseits wurden die Internal Nodes durch die Entfernung der binären Daten kleiner, so dass im optimalen Fall alle Internal Nodes dauernd in einem Cache im Hauptspeicher abgelegt werden können.

BLOBS (Binary Large Objects) sind binäre Objekte (zum Beispiel Dateien) welche grösser als ein Block (das heisst ein Knoten) auf der Festplatte sind. BLOBS werden gespeichert indem Zeiger auf Datenblocks angelegt werden. Diese Zeiger werden normalerweise im Leaf Level gespeichert. Dadurch entsteht ein weiteres BLOB-Level unter dem eigentlichen Leaf Level. Beispiel eines Reiser3-Baumes:

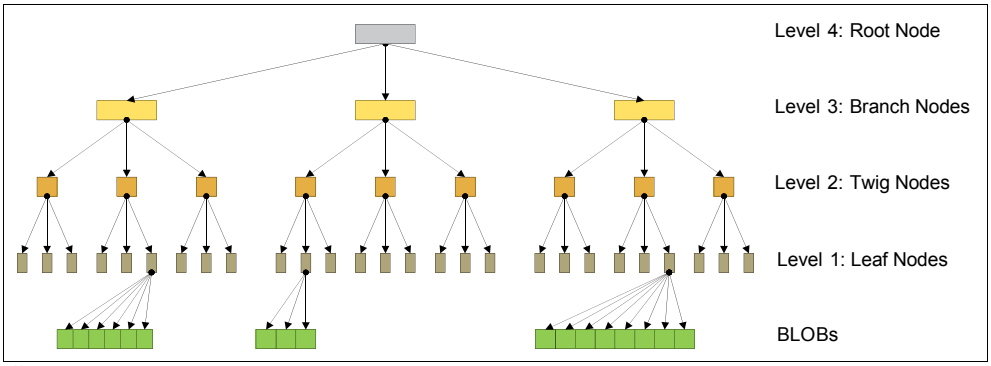

Illustration 14: Reiser3 Baum

Reiser4-Baum

Durch die Einführung der BLOBs im Reiser3-Baum wurden wieder dieselben Probleme eingeführt welche die Entwicklung vom B- zum B⁺-Baum ausgelöst hatten. Nun war es im Leaf Level wieder der Fall, dass die Knoten gleichzeitig Zeiger auf andere Knoten und binäre Daten enthielten. Die Knoten im Leaf Level konnten deshalb nicht im Hauptspeicher gehalten werden, was wiederum zu mehr Festplattenzugriffen als nötig geführt hat.

In Reiser4 änderte dies wieder. Die BLOBs wurden auf das gleiche Level wie die Leaf Nodes gebracht und damit sind die Zeiger auf BLOBs neu im Twig Level gespeichert. Dadurch wird wieder klar zwischen Knoten mit Daten und Knoten mit Zeigern unterschieden. Gemäss Reiser können bei typischen Konfigurationen alle Internal Nodes gleichzeitig im Hauptspeicher gehalten werden. Beispiel eines Reiser4-Baumes:

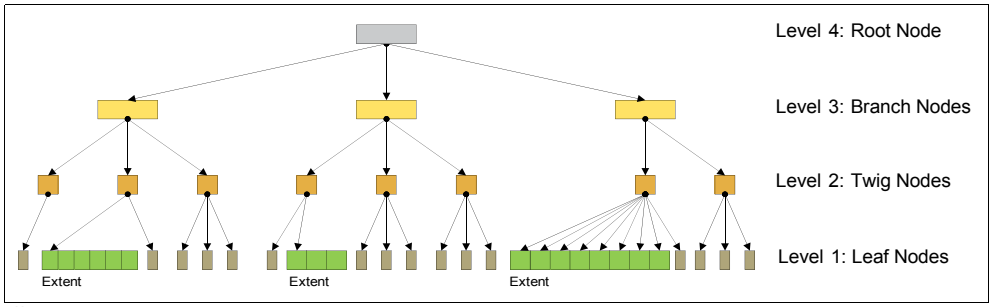

Illustration 15: Reiser4 Baum

6.4 Reiser4 Schlüssel

6.4.1 Übersicht

Wie bereits in der Einführung erwähnt, bestehen die Schlüssel von Reiser4 aus einem Array von 64-Bit Werten. Abhängig davon wie Reiser4 kompiliert worden ist besteht dieses Array entweder aus drei Komponenten (kleine Schlüssel) oder aus vier Komponenten (grosse Schlüssel). Der Treiber speichert auf der Partition in einem Bit-Feld ob er kleine oder grosse Schlüssel verwendet. Es ist weder ein für grosse Schlüssel kompilierter Treiber zu einer Partition von kleinen Schlüsseln noch ein für kleine Schlüssel kompilierter Treiber zu einer Partition von grossen Schlüsseln kompatibel.

Die vier (bzw. drei) Komponenten des Schlüssels werden folgendermassen benennt:

Name	Index (grosse Schlüssel)	Index (kleine Schlüssel)	Anmerkungen
locality	0	0	Diese Komponente enthält unter anderem den Typ des Schlüssel.
ordering	1	-	Diese Komponente ist nur in grossen Schlüssel vorhanden
objectid	2	1	
offset	3	2	

Für die Funktionsweise von Reiser4 spielen die Schlüssel eine zentrale Rolle. Jedes im Reiser4 Baum abgelegte Element besitzt einen Schlüssel. Abhängig vom Objekttyp wird jedoch zwischen verschiedenen Schlüsseltypen unterschieden. Die Komponenten des Schlüssels werden jeweils basierend auf dem Schlüsseltyp unterschiedlich interpretiert. Der Typ eines Schlüssels ist immer in den niedrigstwertigen 4 Bit der `locality` Komponente gespeichert. Es gibt die folgenden Arten von Reiser4 Schlüsseln:

Typ	Wert	Beschreibung
KEY_FILE_NAME_MINOR	0	Dieser Schlüsseltyp wird auf Verzeichniseinträge angewendet und besteht aus deren Namen. Siehe Übersicht und ein weiteres folgendes detailliertes Kapitel.
KEY_SD_MINOR	1	Dieser Schlüsseltyp wird für Stat Data Elemente (Informationselemente über Dateien und Verzeichnisse) verwendet. Er ist definiert, dass aus einem Inode immer der Schlüssel des Items mit den dazugehörenden Stat Data hergeleitet werden kann.
KEY_ATTR_NAME_MINOR	2	Dieser Schlüsseltyp wird in Reiser4 momentan nicht verwendet.
KEY_ATTR_BODY_MINOR	3	Dieser Schlüssel wird ebenfalls in Reiser4 momentan nicht verwendet.
KEY_BODY_MINOR	4	Der Typ KEY_BODY_MINOR wird verwendet für alle nicht durch KEY_FILE_NAME_MINOR und KEY_SD_MINOR abgedeckten Elemente verwendet.

6.4.2 KEY_FILE_NAME_MINOR

Schlüssel von diesem Typ werden verwendet um Verzeichniseinträge zu referenzieren. Das wichtigste dabei ist, dass der Schlüssel rekonstruierbar ist. Diese Fähigkeit wird für die Suche nach Dateinamen im Baum benötigt. Dabei steht nur der Dateinamen zur Verfügung und es muss daraus für eine korrekte Suche immer derselbe Schlüssel extrahiert werden können. Die Komponenten des Schlüssels sind folgendermassen definiert:

locality

Bitstellen	Beschreibung
0-59	Enthält die Object-Id des Parent-Verzeichnisses.
60-63	Schlüsseltyp. Diese 4 Byte enthalten die Konstante `KEY_FILE_NAME_MINOR`

ordering

Bitstellen	Beschreibung
0-6	Fibration Code
7	Dieses Bit wird Longname Mark genannt. Es wird gesetzt sobald der Name des referenzierten Objektes länger als 23 Buchstaben ist.
8 - 63	7 Buchstaben des Namens

objectid

Bitstellen	Beschreibung
0-63	8 Buchstaben des Namens

offset

Das offset Member des Keys unterscheidet sich, je nach dem ob der Name des Objektes mehr oder weniger als 23 Buchstaben besitzt. Es ist bei 23 oder weniger Buchstaben folgendermassen organisiert:

Bitstellen	Beschreibung
0-63	8 Buchstaben des Namens

Bei mehr als 23 Buchstaben sieht das Member hingegen so aus:

Bitstellen	Beschreibung
0-63	Hashwert über den Schluss des Namens, das heisst über alle nicht im Schlüssel

Bitstellen	Beschreibung
	enthaltenen Buchstaben. (Ab dem 16. Buchstaben des Namens)

6.4.3 KEY_SD_MINOR

Dieser Schlüssel kann aus den Komponenten eines Inodes generiert werden. Er zeigt dann auf das Stat Data Item welches die Eigenschaften des Inodes enthält. Die Komponenten des Schlüssels sind folgendermassen definiert:

locality

Bitstellen	Beschreibung
0-59	Enthält die Locality Id des Inodes.
60-63	Schlüsseltyp. Diese 4 Byte enthalten die Konstante `KEY_SD_MINOR`

ordering

Bitstellen	Beschreibung
0-63	Enthält den Ordering Wert des Inodes.

objectid

Bitstellen	Beschreibung
0-63	Enthält die Object Id des Inodes.

offset

Bitstellen	Beschreibung
0-63	Enthält immer 0.

6.4.4 KEY_BODY_MINOR

Dieser Schlüssel kann aus den Komponenten eines Inodes und aus dem Offset ab welchem aus einer Datei gelesen werden soll generiert werden. Er kann bei Leseoperationen dynamisch generiert werden und mit ihm kann das Item gefunden werden welches die Daten enthält welche gelesen werden sollen. Die Komponenten des Schlüssels sind folgendermassen definiert:

Locality

Bitstellen	Beschreibung
0-59	Enthält die Locality Id des Inodes.
60-63	Schlüsseltyp. Diese 4 Byte enthalten die Konstante `KEY_BODY_MINOR`.

ordering

Bitstellen	Beschreibung
0-63	Enthält den Ordering Wert des Inodes.

objectid

Bitstellen	Beschreibung
0-63	Enthält die Object Id des Inodes.

offset

Bitstellen	Beschreibung
0-63	Das offset Member enthält das Offset des ersten im Item enthaltenen Bytes.

6.5 Reiser4 Disk-Layout

Es folgt eine kurze Beschreibung des Disk-Layouts von Reiser4. Sinn und Zweck dieses Abschnittes ist es dem Leser eine kurze Übersicht und Vorstellung davon zu geben wie Reiser4 Daten auf der Festplatte ablegt. Damit soll die Grundlage für die späteren detaillierten Beschreibungen von Strukturen, deren Inhalten und Algorithmen gebildet werden.

Da es von einem Plugin abhängt wo die Daten auf der Festplatte zu finden sind, kann keine Übersicht über die Einteilung der Partition in verschiedene Elemente wie Journal, Knoten des Baumes, Bitmaps und ähnliches gegeben werden. Es wäre beispielsweise denkbar, dass bei einem Plugin das Journal am Anfang einer Partition steht während es bei einem anderen Plugin am Ende der Partition zu finden ist. Die folgenden Unterkapitel wurden deshalb nicht nach der Reihenfolge ihres Auftretens auf der Partition, sondern nach der Reihenfolge ihrer Verwendung geordnet.

6.5.1 Master Superblock

Dieser Superblock befindet sich an Stelle `REISER4_MAGIC_OFFSET` auf der Festplatte. `REISER4_MAGIC_OFFSET` ist im Linux-Treiber momentan hardcoded als $16*4096=65536$. Er wird durch die Struktur `reiser4_master_sb` abgebildet welche wie folgt definiert ist:

```
structure reiser4_master_sb {
  char magic array 16
  signed_16 disk_plugin_id print_decimal
  signed_16 blocksize print_decimal
  signed_8 uuid array 16 print_hexadecimal
  char label array 16
  signed_64 diskmap print_binary
}
```

Grundsätzlich enthält der Superblock den Magic-String mit welchem die Reiser4-Partition von anderen Partitionen unterschieden wird und die Id des Disk-Format-Plugins welches bestimmt wie die Elemente von Reiser4 auf der Festplatte verteilt sind.

6.5.2 Disk Format Plugin Superblock

Es ist nicht zwingend notwendig, dass ein Disk Format Plugin einen Superblock besitzt. Das Default-Plugin (bisher auch das einzige Implementierte Disk Format Plugin) besitzt jedoch einen Superblock in welchem sich die Angaben zum Disk Layout befinden. Details zum Format der einzelnen Superblocks kann bei den Disk Format Plugins nachgelesen werden. Nachfolgend das Layout aufgeführt welches von Reiser4 standardmässig verwendet wird.

```
structure format40_disk_super_block {
  unsigned_64 block_count print_decimal
  unsigned_64 free_blocks print_decimal
  unsigned_64 root_block print_decimal
  unsigned_64 oid print_decimal
  unsigned_64 file_count print_decimal
  unsigned_64 flushes print_decimal
  unsigned_32 mkfs_id print_decimal
  char magic array 16
  unsigned_16 tree_height print_decimal
  unsigned_16 formatting_policy print_decimal
  unsigned_64 flags print_binary
```

```
  char not_used array 432
}
```

6.5.3 Nodes

Wie die Knoten des Baumes auf der Festplatte genau abgelegt werden wird durch das Node Plugin bestimmt. Ein Knoten ist die Abbildung eines Internal Nodes. In der Software werden Internal Nodes durch die Strukturen znode und jnode dargestellt. Ein Überblick über diese beiden Strukturen folgt in einem der folgenden Abschnitte. Eine detaillierte Beschreibung ist im Kapitel zur Analyse von Reiser4 zu finden. Zum aktuellen Zeitpunkt besitzt Reiser4 nur ein einzelnes Node Plugin, das Node40-Plugin. Wie beim Disk Format Plugin wird hier das Layout dieses Plugins beschrieben.

Layout eines Node40Elementes

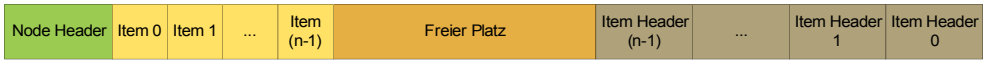

Illustration 16: Node 40 Layout

Node Header

```
structure common_node_header {
  signed_16 plugin_id print_decimal
}
structure node_header40 {
  common_node_header common_header
  unsigned_16 nr_items print_decimal
  unsigned_16 free_space print_decimal
  unsigned_16 free_space_start print_decimal
  unsigned_32 magic print_hexadecimal
  unsigned_32 mkfs_id print_hexadecimal
  unsigned_64 flush_id print_decimal
  unsigned_16 flags print_binary
  unsigned_8 level print_binary
  unsigned_8 pad print_decimal
}
```

Item

Ein Item ist eine beliebige Bytefolge. Die genaue Bedeutung wird von dem jeweiligen Item-Plugin welches im Item-Header angegeben ist bestimmt.

Item Header

Der Item Header enthält eine Beschreibung des zu ihm gehörenden Items. Dazu gehören unter anderem das zur Interpretation des Inhalts nötige Item Plugin und das Offset im Knoten an welchem das Item steht. Er hat folgende Struktur:

```
structure item_header40 {
  reiser4_key key
  unsigned_16 offset print_decimal
  unsigned_16 flags print_binary
  unsigned_16 plugin_id print_decimal
}
```

6.5.4 Items

Im letzten Abschnitt wurde ersichtlich, dass Nodes eigentlich nur Container für eine weitere kleinere Einheit sind. Alle Elemente von Reiser4 werden in Form von Items mit

einem Reiser4-Schlüssel versehen und in Nodes abgelegt. Sie werden durch Item-Plugins beschrieben und besitzten unterschiedliche Disk-Layouts.

Es gibt Item Plugins für Verzeichniseinträge, Internal Node Pointer, Extent Pointer, direkte Dateiinhalte und Weiteres. Aufgrund der Menge werden sie an dieser Stelle nicht beschrieben. Eine Beschreibung des Disk-Layouts von Items, derein Einteilung in Units und weitere Details folgen im Abschnitt 6.7 „Plugin Infrastruktur“.

6.6 Die wichtigsten Software-Konzepte

In Illustration 10 wird eine Übersicht über die wichtigsten Strukturen und Objekte von Reiser4 gegeben.

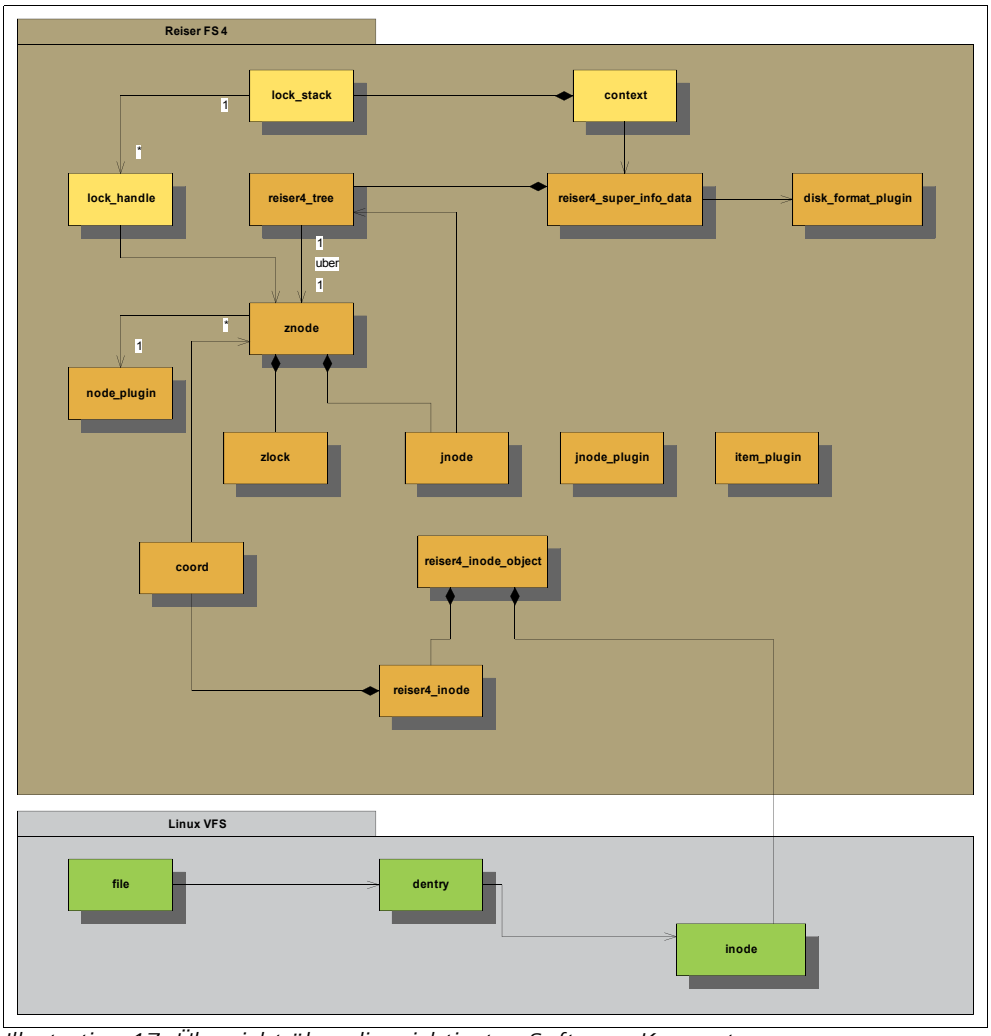

Illustration 17: Übersicht über die wichtigsten Software-Konzepte

reiser4_super_info_data

Der Treiber legt für jede geladene Partition ein Objekt dieser Struktur an. Es beinhaltet einerseits den Reiser4 Baum (Ein Element vom Typ `reiser4_tree`) und andererseits Informationen über die Partition wie die Grösse, die Anzahl der besetzten und freien Blocks und Ähnliches. Die meisten Funktionen benötigen eine Referenz auf die Instanz dieser Struktur. Deshalb wird sie entweder auf dem Stack von Funktion zu Funktion weitergereicht oder sie ist aus dem aktuellen Reiser4 Kontext abrufbar. Die Struktur

kann als zentrale Instanz von Reiser4 betrachtet werden von welcher aus alle anderen Elemente erreicht werden können.

disk_format_plugin

Das Disk Format Plugin ist eine Einrichtung welche auch im Reiser4-Kern ohne Plugin hätte implementiert werden können. Das Disk Format Plugin speichert einerseits die Position der zentralen Elemente wie Journal und Root Node, andererseits übernimmt es aber auch einen grossen Teil der Initialisierung von Reiser4. Eine ausführlichere Beschreibung kann im Kapitel über die Plugins von Reiser4 gefunden werden.

reiser4_tree

Der Reiser4 Baum wird durch diese Struktur implementiert. Da sie immer benötigt wird ist sie direkt als Member von `reiser4_super_info_data` implementiert. Es existiert dadurch eine Instanz des Reiser4 Baumes pro gemounteter Partition. Die Struktur enthält und verwaltet alle zum Aufbau des Baumes und dessen Synchronisierung nötigen Elemente.

jnode

Ein jnode kann für mehrere Dinge verwendet werden. Er wird für folgende Elemente des Dateisystems benötigt:

- Für die Repräsentation von Formatted- und Unformatted-Blocks im Memory
- Für die Repräsentation von Bitmap Blocks im Memory
- Für die Repräsentation von Blocks des Wandering Logs (Journal) im Memory

Für das Verstehen des Read-Only Treibers ist vor allem der erste Punkt wichtig. Die anderen beiden Funktionsweisen werden nur zur Modifizierung der Partition benötigt. Jeder durch ein `jnode` repräsentierte Block ist 4096 Bytes gross.

znode

Ein znode stellt einen Knoten des Reiser4 Baumes dar. Alle Internal Nodes des Reiser4 Baumes sind znode's. Jeder znode ist zugleich auch ein jnode (Formatted Node). Der znode enthält aber weitere Elemente welche für die Verlinkung und die Synchronisierung im Baum.

zlock

Ein zlock ist eine Struktur zur Synchronisierung des Zugriffs auf die znode's. Jeder znode besitzt ein eigenes Lock welches vom Treiber wie ein Read-Write-Lock verwendet werden kann. Durch `zlock`, `lock_handle` und `lock_stack` kann herausgefunden werden welcher Thread welche `znode`'s wie gelockt hat.

jnode_plugin

Jeder jnode ist von einem der oben erwähnten Typen. Für jeden dieser Typen existiert ein jnode_plugin. Diese Plugins implementieren die Funktionen durch welche sich die verschiedenen jnode-Typen unterscheiden.

coord

Ein coord-Objekt stellt die „Koordinate“ eines Elementes auf der Festplatte dar. Um ein Element eindeutig zu referenzieren werden einerseits ein Zeiger auf den besitzenden `znode` (das heisst den Internal Node) und andererseits Indices auf das

Item und die darin befindliche Unit gespeichert. Zusätzlich besitzt eine Koordinate den Index des für das Item nötigen `item_plugin`'s.

node_plugin

Das `node_plugin` bestimmt wie ein `jnode` auf der Festplatte strukturiert wird. Wie im Abschnitt zu den jnode's beschrieben enthält ein jnode 4096 Bytes Speicherplatz. Das node_plugin bestimmt wie die Items in diesem Speicherbereich abgelegt werden. Es ist daher eines der wichtigsten Plugins von Reiser4.

item_plugin

Jedes der in den Knoten abgelegten Items ist von einem bestimmten Typ. Für jeden Item Typ existiert ein eigenes `item_plugin` welches die Daten des Plugins in einen Knoten schreiben und wieder daraus herauslesen kann. Es gibt beispielsweise `item_plugin`'s für Verzeichniseinträge, Internal Node Pointers oder Extent Node Pointers. Sie sind daher neben den `node_plugin`'s die zweite für die Speicherung und das Lesen des Reiser4 Baumes zentrale Einrichtung.

context

Ein `context` Objekt speichert Angaben über den aktuellen Aufruf des Treibers. Wann immer vom Betriebssystem eine Funktion des Treibers aufgerufen wird legt der Treiber ein neues Objekt von diesem Typ an. Dieses Objekt wird dann in einer globalen Liste mit dem Thread assoziiert und kann von allen im Folgenden aufgerufenen Funktionen genutzt werden. Basierend darauf werden genommene Locks oder allozierte Speicherbereiche mit dem `context` assoziiert und können so bei Aufruf-Ende wenn nötig ohne grossen Verwaltungsaufwand wieder an das System zurückgegeben werden. Damit kann einfacher und besser kontrolliert werden dass keine Resourcen-Löcher entstehen.

lock_handle

Ein `lock_handle` beschreibt die Umstände und Elemente eines auf einen `znode` genommenen Locks. Es ermöglicht, dass mehrere Threads gleichzeitig ein Read-Lock auf einen `znode` nehmen können.

lock_stack

Jeder `context` besitzt einen `lock_stack`. Auf dem lock_stack werden alle mit dem aktuellen Thread assoziierten `lock_handle`'s registriert.

inode_object

Das `inode_object` besitzt zwei Elemente. Das eine Element ist der Linux-VFS-inode und das andere der `reiser4_inode`. Es dient dazu die beiden Objekte miteinander zu verknüpfen, so dass immer über einen Cast von einem zum anderen Typ gewechselt werden kann. Dadurch wird auch verhindert, dass jeweils zwei Objekte ein linux- und ein Reiser4-Inode instanziert werden müssen.

reiser4_inode

Der `reiser4_inode` enthält alle zur Beschreibung eines Objektes im Dateisystem nötigen Angaben. Ein Objekt kann dabei beispielsweise eine Datei oder ein Verzeichnis sein. Dies umfasst vor allem eine Ansammlung von Plugins zur Bearbeitung des inode Inhaltes.

inode, file, dentry

Die Strukturen inode, file und dentry werden vom Linux Virtual File System definiert und genutzt. Ihre Funktionsweise wird weiter oben beschrieben. Sie sind in diesem Kapitel der Vollständigkeit halber aufgeführt, da sie manchmal von den den Interface-Funktionen zum Linux VFS an interne Funktionen weitergegeben und deshalb benötigt werden. Weitere Hinweise dazu können in den beiden folgenden Quellen gefunden werden: [MAU01], [RGOOCH]

6.7 Plugin Infrastruktur

Eines der Ziele von Reiser4 ist grosse Flexibilität und leichte Erweiterbarkeit. Es wird versucht diese Ziele durch die Implementierung der Plugin-Infrastruktur zu erreichen. Dazu ist es nötig Funktionen zur Verfügung zu stellen die es einem Entwickler ermöglichen seine eigenen Plugins leicht in das Gesamtsystem einzubinden ohne dabei grosse Veränderungen am Quellcode des Dateisystems vornehmen zu müssen. Dieser Abschnitt beschreibt welche Funktionalitäten Reiser4 dazu anbietet.

6.7.1 Plugin Typ

Jedes Plugin in Reiser4 ist von einem bestimmten Plugin-Typ. Beispiele für Plugin-Typen sind Node-Plugins, Disk-Format-Plugins oder Item-Plugins. Die verschiedenen vorhandenen Plugin-Typen sind in der Enumeration `reiser4_plugin_type` in der Headerdatei `plugin_header.h` aufgelistet.

Zusätzlich wird jedes Plugin über einen Eintrag im Array `plugins[]` beschrieben. Jeder Eintrag ist eine Instanz der Struktur `reiser4_plugin_type_data`:

```
typedef struct reiser4_plugin_type_data {
  reiser4_plugin_type type_id;
  const char *label;
  const char *desc;
  int builtin_num;
  void *builtin;
  plugin_list_head plugins_list;
  size_t size;
} reiser4_plugin_type_data;
```

Wobei die einzelnen Elemente der Struktur folgende Funktion haben:

Member	Beschreibung
type_id	Beschreibt den Typ des Plugins. Dieses Member ist eigentlich „überflüssig“ wird vom Treiber aber dazu verwendet um zu prüfen ob das plugins[]-Array vom Programmierer richtig initialisiert wurde.
label	Zeiger auf einen String mit der Bezeichnung des Plugin-Typs
desc	Eine kurze Beschreibung der Funktion welche mit Plugins dieses Typs erfüllt werden soll.
builtin_num	Anzahl der im Treiber mitgelieferten Plugins dieses Typs.
builtin	Zeiger auf ein Array aller vom Treiber mitgelieferten Plugins dieses Typs.
plugins_list	Vermutlich eine Liste aller installierten Plugins dieses Typs. Vermutlich dienst dieses Member dazu später einmal dynamisch Plugins in den Treiber laden zu können.
size	Grösse eines Plugins dieses Typs in Bytes. Dieses Member wird dazu benötigt um den builtin-Zeiger richtig verwenden und ohne genauere Kenntnis des einzelnen Plugins durch dieses Array navigieren zu können.

Der typische Eintrag im `plugins[]`-Array sieht nach C90-Standard folgendermassen aus:

```
[REISER4_FILE_PLUGIN_TYPE] = {
  .type_id = REISER4_FILE_PLUGIN_TYPE,
  .label = "file",
  .desc = "Object plugins",
  .builtin_num = sizeof_array(file_plugins),
  .builtin = file_plugins,
  .plugins_list = TYPE_SAFE_LIST_HEAD_ZERO,
  .size = sizeof(file_plugin)
},
```

Das Array `plugins[]` bietet damit für den Treiber eine direkte Zugriffsmöglichkeit auf jedes Plugin wenn man den Typ und die Id eines Plugins kennt.

6.7.2 Struktur eines Plugins

Die Definition eines Plugins ist danach sehr frei. Die einzige von Reiser4 auferlegte Bedingung ist, dass die erste Membervariabel jedes Plugins vom Typ `plugin_header` sein muss. Dieser enthält wiederum Plugin Typ und Plugin Id zur Überprüfung, dass der Programmierer das Array richtig initialisiert hat und Strings welche das Plugins beschreiben. Zusätzlich enthält er ein Array von Funktionen welche auf alle Plugins anwendbar sind.

6.8 Algorithmen

6.8.1 Lookup eines Objektes

Der folgende Algorithmus soll zeigen wie eine Datei oder ein Verzeichnis aufgrund des Namens und des übergeordneten Verzeichnisses gefunden werden kann.

```
Algorithmus lookup (parent : dentry, name)
  begin
    direntry_coord <- get_dir_plugin(dentry)->build_entry_key(parent, name);
    if direntry_coord->item_type != ITEM_DIRECTORY_ENTRY then
      lookup <- error
      return
    endif
    object_key <- get_item_plugin(direntry_coord)->extract_key(direntry_coord)
    object_inode <- iget(object_key)
    lookup <- object_inode
    return
  end
```

6.8.2 Traversierung des Reiser4 Baumes

Die folgenden Abschnitte beschreiben alle Elemente, welche bei der Traversierung des Reiser4 Baumes von Bedeutung sind. Im letzten Abschnitt wird dann der Algorithmus präsentiert.

cbk_handle

Die Struktur cbk_handle wird benötigt um den Algorithmus zur Traversierung im Baum einheitlich gestalten zu können. Die Traversierung des Reiser4 Suchbaumes kann über ein Set von Funktionen gestartet werden. Gemeinsam ist diesen Funktionen, dass sie aus den übergebenen Parametern eine Instanz der Struktur `cbk_handle` generieren in welcher alle Suchparameter in einer standardisierten Form abgelegt werden können. Folgende Member des `cbk_handle` beeinflussen bzw. steuern den Traversierungsalgorithmus:

Member	Beschreibung
tree	Enthält einen Zeiger auf die `reiser4_tree` Struktur des Baumes welcher traviersiert wird.
key	Enthält einen Pointer auf den im Baum gesuchten Schlüssel.
coord	Zeiger auf eine Struktur vom Typ `coord`. In diese Struktur wird das Resultat der Suche gespeichert.
lock_mode	Spezifiziert den Locking-Typ welcher auf dem Zielknoten angewendet wird. Das heisst auf denjenigen Knoten der das Ziel der Suche enthält.
bias	Spezifziert die Genauigkeit der Suche. Es gibt zwei Möglichkeiten: • Es wird nach exakt dem übergebenen Schlüssel gesucht • Es wird der grösste Schlüssel gesucht der gerade noch kleiner als der angegebene Schlüssel ist.
lock_level	Spezifiziert den Level des Baumes ab welchem die Traversierung Write-Locks nimmt.
stop_level	Spezifiert den Level des Baumes an welchem die Suche beendet wird. Wurde das gesuchte Element nicht zwischen `lock_level` und `stop_level` gefunden wird ein Not-Found Fehler zurückgegeben.
flags	Dieses Member speichert eine Reihe von Flags welche an den Algorithmus übergeben werden bzw. während der Ausführung des gesetzt werden.
ra_info	Informationen über das Readahead verhalten wenn während der Traversierung Knoten von der Festplatte gelesen werden.

vroot

Das sogenannte vroot ist eine Optimierung für die Suche im Reiser4-Baum. Jeder Inode kann ein vroot besitzen. Die Idee dahinter ist, dass obwohl die einzelnen Elemente eines Inodes über mehrere znodes verteilt sein können, alle Elemente des Inodes im Baum dennoch nahe beieinanderliegen und es daher normalerweise nicht nötig ist die Suche beim Root-Knoten zu starten. Stattdessen wird das sogenannte vroot definiert als derjenige znode von welchem aus alle Elemente des betreffenden Inodes gefunden werden können. Ist nun auf einem Inode ein solcher vroot definiert wird eine Suche bei diesem vroot anstatt dem absoluten Root-Knoten gestartet.

Konzept

Der folgende Algorithmus ist auf einem sehr hohen Level geschrieben. Er soll zuerst das Grundkonzept der Suche im Baum klären.

```
Algorithmus traverse_tree (h : cbk_handle)
  begin
    if vroot defined then
      node = vroot
    else
      node = root node
    endif
    if check_cache_for_key(h.key) then
      traverse_tree <- get_cache_entry()
      return
    else
      -- do level lookup
      node <- do_level_lookup(h)
      until node = invalid or node.key = searched_key do
        node <- do_level_lookup(node, key)
      loop
      traverse_tree <- node
      return
    endif
  end

Algorithmus do_level_lookup(h : cbk_handle)
  begin
    do_level_lookup <- do_node_lookup(h)
    return
  end
```

```
Algorithmus do_node_lookup(h : cbk_handle)
  begin
    do_node_lookup <- get_node_plugin(znode).lookup(h.active.lh, h.key, h.coord)
    return
  end
```

Wie gut erkennbar ist wird in einer Schlaufe der Knoten beginnend beim höchsten Level gesucht. Der höchste Level ist dabei entweder der vroot des Inodes für welchen gesucht wird oder der Root Knoten des Reiser4 Baumes. Pro Schlaufendurchgang wird jeweils ein Level-Lookup ausgeführt. Dieses Level-Lookup lädt wenn nötig den znode des aktuellen Levels von der Festplatte und initialisiert ihn. Danach gibt es die Kontrolle an die generische Node-Lookup-Funktion weiter. Im Node-Lookup wird überprüft ob die im `cbk_handle` definierten Levels erreicht worden sind und die Suchkontrolle danach an das verantwortliche Node-Plugin weitergeleitet. Das Node-Plugin kann beliebig implementiert sein.

6.9 Reiser4 Built-In Plugins

6.9.1 Plugin Header

Alle Plugins besitzen ein Member 'h' vom Typ `plugin_header`. Der Header ist wie folgt definiert:

```
typedef struct plugin_header {
  reiser4_plugin_type type_id;
  reiser4_plugin_id id;
  reiser4_plugin_ops* pops
  const char* label;
  const char* desc;
  plugin_list_link linkage;
}  plugin_header;
```

Die Member der Struktur bedeuten folgendes:

Member	Beschreibung
type_id	Die Enumeration reiser4_plugin_type beschreibt die verschiedenen Plugin-Typen die in Reiser4 verwendet werden.
id	Die Id des Plugins wird verwendet um das Plugin von anderen Plugins des selber Typs zu unterscheiden.
pops	*Es gibt ein Set von Funktionszeigern welches auf alle Plugins anwendbar ist. Ein Plugin kann diese Funktionen implementieren wenn es möchte und dieses Member mit einem Zeiger auf diese Funktionen initialisieren.*
label	Jedes Plugin besitzt einen Namen welcher durch dieses Member dargestellt wird.
desc	Eine genauere Beschreibung der Funktionen des Plugins
linkage	Nicht näher spezifiert.[2]

2 Dieses Member ist für einen Read-Only Treiber nicht relevant. Da gemäss der Aufgabenstellung nur ein Read-Only Treiber entwickelt werden muss wurde die Analyse der unwichtigen Elemente von Strukturen weggelassen. Im Rest des Dokumentes sind alle diese Member mit „Nicht näher spezifiziert." gekennzeichnet.

6.9.2 Disk Format Plugins (REISER4_FORMAT_PLUGIN_TYPE)

Disk Format Plugins beschreiben, wie Elemente von Reiser4 auf der Festplatte abgelegt sind. Sie sind wie folgt definiert:

```
typedef struct disk_format_plugin {
  plugin_header h;
  int (*get_ready)(struct super_block*, void* data);
  const reiser4_key* (*root_dir_key)(const struct super_block*);
  int (*release)(struct super_block*);
  jnode* (*log_super)(struct super_block*);
  void (*print_info)(const struct super_block*);
  int (*check_open)(const struct inode* object);
} disk_format_plugin;
```

Die Member der Struktur bedeuten folgendes:

Member	Beschreibung
h	Der Plugin Header.
get_ready	Diese Funktion wird vom Reiser4-Init-Code aufgerufen während eine neue Partition gemountet wird. Er sollte die Disk-Format-Spezifischen Daten von der Festplatte lesen und entsprechende Initialisierungen vornehmen.
root_dir_key	Diese Funktion gibt den Schlüssel des Root-Eintrags zurück.
release	Wird aufgerufen beim Unmount wenn der Superblock freigegeben wird.
log_super	Nicht näher spezifiert.
print_info	Nicht näher spezifiert.
check_open	Diese unktion wird immer aufgerufen nachdem ein Inode geladen wurde und kann dessen Korrektheit überprüfen.

Format40 Plugin

Das Format40-Plugin ist das Default-Disk-Plugin von Reiser4. Es ist zugleich das einzige Plugin mit welchem der Treiber standardmässig ausgeliefert wird. Auf der Festplatte besitzt das Plugin einen eigenen Superblock welcher an der Stelle `FORMAT40_OFFSET = REISER4_MASTER_OFFSET + PAGE_CACHE_SIZE` abgelegt ist. `REISER4_MASTER_OFFSET` ist dabei das Offset an welchem der Master-Superblock abgelegt ist (65536) und `PAGE_CACHE_SIZE` ist eine vom Linux-Kernel vorgegebene Maschienenabhängige Konstante. Auf normalen I386 Rechnern gilt `PAGE_CACHE_SIZE = 4096`. Der Format40-Superblock ist wie folgt definiert:

```
structure format40_disk_super_block
{
  unsigned_64 block_count print_decimal
  unsigned_64 free_blocks print_decimal
  unsigned_64 root_block print_decimal
  unsigned_64 oid print_decimal
  unsigned_64 file_count print_decimal
  unsigned_64 flushes print_decimal
  unsigned_32 mkfs_id print_decimal
  char magic array 16
  unsigned_16 tree_height print_decimal
  unsigned_16 formatting_policy print_decimal
  unsigned_64 flags print_binary
  char not_used array 432
}
```

Die Membervariabeln bedeuten folgendes:

Member	Beschreibung
block_count	Gibt die Anzahl der auf der Partition vorhandenen Reiser4-Blocks an. Jeder Block ist normalerweise 4096 Bytes gross.

Member	Beschreibung
free_blocks	Speichert die Anzahl der noch freien Blocks auf dem Dateisystem.
root_block	Gibt den ReiserFS Block an in welchem der Root-Node abgelegt ist. Die Adresse kann über root_block * master_superblock.blocksize berechnet werden.
oid	Nicht näher spezifiert.
file_count	Enthält die Anzahl der auf der Partition abgelegten Dateiobjekte.
flushes	Nicht näher spezifiert.
mkfs_id	Nicht näher spezifiert.
magic	Der Magic-String für das DiskFormat40-Plugin. Der Magic-String muss immer 'ReIsEr40FoRmAt' lauten.
tree_height	Speichert die aktuelle Höhe des Reiser4-Trees.
formatting_policy	Nicht näher spezifiert.
flags	Nicht näher spezifiert.
not_used	Nicht näher spezifiert.

6.9.3 Node Plugins (REISER4_NODE_PLUGIN_TYPE)

Node Plugins beschreiben wie Reiser4 Knoten auf der Festplatte ablegt. Jeder Knoten besitzt genau die Grösse eines `jnode`'s. In jedem Knoten ist eine Anzahl von Items abgespeichert welche verschiedene Funktionen unterstützen können. Node Plugins organisieren nun die Verteilung der enthaltenen Items auf den verfügbaren Platz auf dem Speichermedium.

Node40 Plugin

Das Node40 Plugin teilt den Bereich den es auf der Festplatte belegt wie folgt ein:

Illustration 18: Node40 Disk Layout

Jeder Knoten des Node40 Plugins besitzt einen Header welcher die wichtigsten Angaben zum Inhalt des Knotens enthält. Er ist wie folgt definiert:

```
structure common_node_header {
  signed_16 plugin_id print decimal
}
structure node_header40 {
  common_node_header common_header
  unsigned_16 nr_items print_decimal
  unsigned_16 free_space print_decimal
  unsigned_16 free_space_start print_decimal
  unsigned_32 magic print_hexadecimal
  unsigned_32 mkfs_id print_hexadecimal
  unsigned_64 flush_id print_decimal
  unsigned_16 flags print_binary
  unsigned_8 level print_binary
  unsigned_8 pad print_decimal
}
```

Die Membervariabeln bedeuten folgendes:

Member	Beschreibung
plugin_id	Enthält die Id dieses Plugins. Enthält immer `NODE40_ID`.
nr_items	Speichert die Anzahl an Items welche sich momentan in diesem Knoten befinden.
free_space	Speichert die Anzahl der in diesem Knoten für neue Objekte freien

Member	Beschreibung
	Bytes.
free_space_start	Bezeichnet das Byte-Offset im Knoten ab welchem der freie Speicherplatz beginnt. Das Offset zeigt direkt hinter den letzten Item-Body.
magic	Ein Magic-Feld welches immer mit demselben Doppelwort gefüllt ist und dazu dient zu erkennen falls Fehler auftreten.
mkfs_id	Nicht näher spezifiert.
flush	Nicht näher spezifiert.
flags	Dieses Feld ist reserviert für Flags. In Reiser4 konnte keine Anwendung dieses Feldes gefunden werden.
level	Speichert das Level im Baum auf welchem sich dieser Knoten befindet.
pad	Ein Padding-Wert damit die Struktur Wort-Aligned ist.

Jedes Item im Knoten besteht aus zwei Elementen (Item Header und Item Body). Der „Item Header" ist am Schluss des Knotens gespeichert. Das Node40 Plugin speichert die Header jeweils von rechts beginnend und nach links, d.h. nach innen wachsend. Im Header sind die zum Interpretieren eines Items benötigten Daten abgelegt. Eine Ausnahme ist die Grösse des Items, welche aus den Offset-Membern der Item Header berechnet werden kann und deshalb nicht abgelegt wird, um Platz zu sparen.Ein Item Header sieht wie folgt aus:

```
structure item_header40 {
  reiser4_key key
  unsigned_16 offset print_decimal
  unsigned_16 flags print_binary
  unsigned_16 plugin_id print_decimal
}
```

Die Membervariabeln bedeuten folgendes:

Member	Beschreibung
key	Speichert den diesem Item zugeordnete Reiser4-Schlüssel.
offset	Speichert das offset im Knoten ab welchem der Inhalt des Items gespeichert ist.
flags	Dieses Feld ist reserviert für Flags. In Reiser4 konnte keine Anwendung dieses Feldes gefunden werden.
plugin_id	Speichert die Id des zum Item gehörenden Item Plugins. Das angegebene Plugin kann die Daten des Items korrekt interpretieren.

Ein Item Header bezieht sich jeweils auf den Item Inhalt. Der Inhalt der Items (Item Body) wird beginnend nach dem Node Header nach rechts wachsend abgelegt. Er kann über die im Item Header gemachten Angaben referenziert werden und eine beliebige Grösse haben. Für das Node Plugin ist ein Item einfach eine beliebige Folge von Bytes. Die Daten werden jeweils vom Plugin, das zum Item gehört, interpretiert.

6.9.4 File Plugins (REISER4_FILE_PLUGIN_TYPE)

File Plugins übernehmen die Implementierung von Objekten in Reiser4. Hans Reiser bezeichnet sie in einem Kommentar treffender als Object Plugins. Sie behandeln die speziellen Eigenschaften von verschiedenen Linux-Dateitypen. Mit Dateitypen ist dabei nicht die Unterscheidung zwischen Bild, Text oder anderen Dateien, sondern die für Unix wichtige Unterscheidung zwischen regulären Dateien, symbolischen Links, Verzeichnissen, Pipes, Fifos, Devices und Sockets gemeint. File Plugins implementieren diese Eigenschaften von Linux-Dateien. Welches File Plugin verwendet wird, wird

entweder aus den Stat Daten gelesen oder auf der Basis des `i_mode` Members der Inode-Struktur „geraten“.

Grundsätzlich ist das File Plugin verantwortlich für die Implementierung der VFS-Operationen auf der angegebenen Datei. Das ist der Grund wieso verschiedene File Plugins für reguläre Dateien existieren. Die Daten einer regulären Datei können vor dem Schreiben beziehungsweise nach dem Lesen durch das File Plugin modifiziert werden. Sinnvolle Beispiele dafür sind Komprimierung und/oder Verschlüsselung.

Es gibt folgende File Plugins:

Regular File Plugin

Das Regular File Plugin implementiert die VFS-Operationen für reguläre Dateien. Es nimmt keine Veränderungen an den gegebenen Daten vor und legt sie gemäss Bestimmungen des Formatting Plugins entweder als Tail oder als Extent Item auf der Festplatte ab.

Directory File Plugin

Das Directory File Plugin implementiert die VFS-Operationen für Verzeichnis-Objekte.

Symlink File Plugin

Das Symlink File Plugin implementiert die VFS-Operationen für symbolische Links.

Special File Plugin

Das Special File Plugin implementiert die VFS-Operationen für Fifo-, Device- oder Socket-Objekte.

Cryptocompress File Plugin

Das Cryptocompress File Plugin implementiert die VFS-Operationen für reguläre Dateien. Anders als beim Regular File Plugin werden die Daten jedoch komprimiert und/oder verschlüsselt bevor sie gespeichert werden und natürlich auch wieder in den Originalzustand zurückversetzt nachdem sie von der Festplatte gelesen worden sind.

6.9.5 Directory Plugins (REISER4_DIR_PLUGIN_TYPE)

Die Directory Plugins implementieren die zur Suche nach Objekten benötigte Funktionalität. Die zur Zeit einzige Funktion von Directory Plugins ist es den Schlüssel eines Verzeichniseintrages zu generieren.

Hashed Directory Plugin

Dieses Plugin generiert den Verzeichniseintrag wie im Kapitel zu den Schlüsseln beschrieben. Der Schlüssel besteht danach aus der Object-Id des Verzeichnisses und dem Namen der Datei.

Seekable Hashed Directory Plugin

Dieses Plugin generiert den Schlüssel eines Verzeichniseintrages sehr einfach. Es verwendet einerseits wie das Hashed Directory Plugin die Object-id des Verzeichnisses. Zusätzlich wird aber nur aus dem Dateinamen ein Hashwert generiert und in objectid Teil des Schlüssels abgelegt. Der Dateiname wird nicht im Schlüssel gespeichert.

6.9.6 Item Plugins (REISER4_ITEM_PLUGIN_TYPE)

Item Plugins speichern den Inhalt von Objekten. Sie sind gehören daher zu den wichtigeren Plugins des Dateisystem. Reiser4 unterscheidet bei den Item Plugins noch einmal zwischen einer Gruppierung von Item-Typen. Diese Typen sind:

Typ	Beschreibung
STAT_DATA_ITEM_TYPE	Stat Data Items speichern Meta-Informationen über ein bestimmtes Objekt im Dateisystem. Diese Werte sind zum Beispiel der Besitzer einer Datei, das letzte Änderungsdatum und ähnliches. Standardmässig besitzt Reiser4 dazu ein einziges Plugin das „Static Stat Data Plugin". Plugins vom Typ Stat Data Item müssen das `sd_ops`-Interface implementieren.
DIR_ENTRY_ITEM_TYPE	Directory Entry Items können Verzeichniseinträge speichern. Reiser4 besitzt standardmässig zwei Plugins von dieser Sorte das „Compound Dir Entry Plugin" und das „Simple Dir Entry Plugin". Plugins dieses Typs müssen das `dir_entry_ops`-Interface implementieren.
INTERNAL_ITEM_TYPE	Internal Items sind Items welche Zeiger auf weitere Knoten im Reiser4 Baum enthalten. Reiser4 besitzt nur ein einzelnes Plugin von diesem Typ (das „Internal Item Plugin") und es scheint auch nicht sinnvoll für diese einfache Aufgabe weitere Plugins hinzuzufügen. Plugins dieses Typs müssen das `internal_item_ops`-Interface implementieren.
UNIX_FILE_METADATA_ITEM_TYPE	Diese Items beinhalten die Information wie Dateien ausgelesen werden können. Es gibt verschiedene Arten von Metadata Items, das „Tail Item Plugin", das „Cryptocompress Tail Item Plugin" und das „Extent Item Plugin". Diese Plugins unterscheiden sich darin, dass die Daten entweder direkt im Knoten abgelegt sind oder auf Unformatted Leafs verwiesen wird welche dann die Daten enthalten. Extent Items sind immer im Twig-Level des Baumes zu finden, während sich Tail und Cryptocompress Tail Items immer im Leaf Level befinden müssen. Plugins dieses Typs müssen das `file_ops`-Interface implementieren.
OTHER_ITEM_TYPE	Dieser Typ beschreibt alle anderen Items welche nicht in das Konzept der anderen vier Item Typen passen. Reiser4 besitzt auch ein Plugin von diesem Typ, das „Block Box Item Plugin". Diese Plugins müssen nur das normale Item Plugin Interface implementieren.

Jedes existierende Reiser4 Item Plugin ist einem dieser Typen zugeordnet und implementiert dessen Funktionalität.

Stat Data Item Plugin

Das Stat Data Item Plugin ist das standardmässig mit Reiser4 mitgelieferte Plugin zur Speicherung von Metadaten. Zur Speicherung der Metadaten nutzt das Stat Data Item Plugin die Plugins vom Typ „SdExt Plugin". Auf der Festplatte sieht das Layout des Stat Data Item Plugins wie folgt aus.

Illustration 19: Stat Data Item Disk Layout

Das einzige in diesem Plugin definierte Element ist `StatDataBase` der Header des Stat Data Items. Dieser ist wie folgt definiert:

```
structure reiser4_stat_data_base {
  unsigned_16 extmask print_binary
}
```

Das Member `extmask` ist dabei ein Bitfeld welches für jedes existierende SdExt Plugin dessen Anwesenheit im Item oder dessen Fehlen im Item bezeichnet. Die vorhandenen Plugindaten sind der Reihe nach (beginnend mit dem Least Significant Bit von `extmask`) hinter `StatDataBase` abgelegt. Das heisst in der gegebenen Grafik entspricht das „SdExt Plugin 1" den gespeicherten Daten des SdExt Plugin mit der Id 0 falls das niedrigstwertige Bit in `extmask` gesetzt ist. Beim Laden eines Inodes geht Reiser4 beginnend mit dem niedrigstwertigen Bit durch die `extmask` und lädt jedes im Item gespeicherte SdExt Plugin.

Simple Dir Entry Item Plugin

Dieses Item Plugin ist in der Lage einen einzelnen Verzeichniseintrag zu speichern. Dadurch ist dessen Implementation relativ einfach. Es ist anzunehmen, dass das Plugin bei Reiser4 deshalb während einer frühen Entwicklungsphase genutzt worden ist. Auf gängigen Reiser4-Partitionen ist es nicht mehr zu finden, da das Compound Dir Entry Item Plugin diverse Vorteile bietet.

Die auf der Festplatte abgelegten Daten sind wie folgt definiert:

```
structure directory_entry_format {
  obj_key_id id
  char name array (item_length - sizeof(obj_key_id))
}
```

Die Members sind wie folgt definiert:

Typ	Beschreibung
Id	Der Verzeichniseintrag zeigt auf das untergeordnete Element. Dessen Schlüssel kann teilweise aus dem Schlüssel des Verzeichniseintrags berechnet werden. Dieses Member speichert denjenigen teil des Schlüssels welcher nicht berechnet werden kann.
name	In diesem Member ist der 0-Terminierte Name des Verzeichniseintrags gespeichert. Das Member enthält nur dann Daten wenn das Verzeichnis einen langen Dateinamen besitzt, d.h. einen Dateinamen der nicht in den Schlüssel passt. Sonst wird nicht einmal das 0-Zeichen gespeichert.

Compound Dir Entry Item Plugin

Dieses Plugin stellt eine Weiterentwicklung des Simple Dir Entry Item Plugins dar. Es ermöglicht die Speicherung von mehreren Verzeichniseinträgen in einem Item. Die dahinterliegende Idee ist, dass dadurch der Platz des Item Headers im Node Plugin eingespart werden kann. Jeder Verzeichniseintrag in diesem Plugin stellt eine Unit dar. Das Plugin ist auf der Festplatte wie folgt abgelegt:

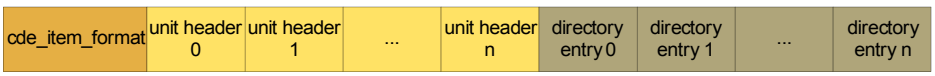

Illustration 20: Compound Dir Entry Item Plugin Disk Format

Dabei werden die folgenden beiden Strukturen genutzt:

```
structure cde_unit_header {
  de_id hash
  unsigned_16 offset print_decimal
}
structure cde_item_format {
  unsigned_16 num_of_entries print_decimal
  cde_unit_header entry array num_of_entries
}
```

```
structure directory_entry_format {
  obj_key_id id
  char name array
}
```

Die Struktur `directory_entry_format` ist dabei bereits aus dem Simple Directory Entry Item Plugin bekannt.

Internal Item Plugin

Dieses Plugin implementiert einen Link auf einen Childknoten des Knotens in welchem sich das Internal Item befindet. Das Disk Layout des Item Plugins sieht wie folgt aus:

```
structure internal_item_layout {
  unsigned_64 block_nr block print_decimal
}
```

Das einzige Member ist block. Es speichert die Nummer des Blocks in welchem der Childknoten gespeichert. Ist.

Extent Item Plugin

Extent Items sind nur im Twig-Level zu finden. Sie zeigen auf Blocks von Unfleaves und speichern so grosse Dateien. Auf die Festplatte wird ein Extent Item wie in Illustration 16 gezeigt gespeichert.

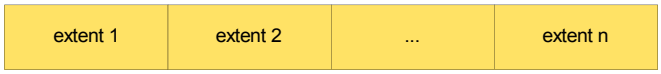

Illustration 21: Extent Item Disk Layout

Jedes extent Element ist von der Struktur `reiser4_extent` welche folgendermassen definiert ist.

```
structure reiser4_extent {
  unsigned_64 start
  unsigned_64 width
}
```

Eine Unit in einem Extent Item ist definiert als ein solches reiser4_extent Element. Jedes zeigt auf eine Reihe von Unfleaves. Alle diese Unfleaves in der gegebenen Reihenfolge zusammengefügt ergeben die Datei, auf welche das Extent Item zeigt. Wichtig ist, dass vom letzten Unfleaf nicht unbedingt alle Bytes genutzt sind falls die Datei nicht genau die Grösse eines vielfachen der Blockgrösse besitzt.

Tail Item Plugin

Tail Items befinden sich immer im Leaf Level. Sie enthalten die Inhalte von Dateien welche so klein sind, dass es sich nicht lohnt sie in eigene Unfleaves zu speichern. Sie besitzen daher auch kein Format. Der Inhalt des Items ist gleichzeitig auch der Inhalt der Datei. Beim Lesen kann er direkt in den übergebenen User Buffer kopiert werden. Bei diesem Item Plugin gelten einzelne Bytes als Unit.

Cryptocompress Tail Item Plugin

Das Cryptocompress Tail Item Plugin wird grundsätzlich wie das Tail Item Plugin verwendet. Der Unterschied liegt darin, dass es den Inhalt einer mit dem Cryptocompress File Plugin geschriebenen Datei enthält und einen Header besitzt. Das Format des Items auf der Festplatte sieht deshalb folgendermassen aus

```
structure ctail_item_format {
  unsigned_8 cluster_shift
  unsigned_8 body array filesize
}
```

Blackbox Item Plugin

Dieses Item Plugin besitzt keine uns bekannte Funktionalität. Möglicherweise nutzt Hans Reiser dieses Plugin um darin irgendwelche Closed-Source Elemente abzulegen.

6.9.7 Formatting Plugins (REISER4_FORMATTING_PLUGIN_TYPE)

Die Aufgabe der Formatting Plugins ist die Entscheidung ob der Inhalt einer Datei als Tail Item oder als Extent Item abgelegt werden soll. Standardmässig wird das Small File Formatting Plugin verwendet. Momentan sind alle Formatting Plugins in der Datei `tail_policy.c` implementiert. Formatting Plugins sind folgendermassen definiert:

```
typedef struct formatting_plugin {
  plugin_header h;
  int (*have_tail) (const struct inode * inode, loff_t size);
} formatting_plugin;
```

Die Funktion `have_tail` wird aufgerufen bevor ein Inode geschrieben wird und soll entscheiden ob der Inode als Tail oder als Extent Item gespeichert wird. Ein Rückgabewert von 0 bedeutet, dass der Inode als Extent Imte abgelegt wird, während ein Rückgabewert von 1 bedeutet, dass er als Tail Item abgelegt wird.

Never Tails Formatting Plugin

Dieses Plugin gibt in `have_tail` immer 0 zurück. Das heisst, dass Inodes immer in Extent Items abgelegt werden, solange dieses Plugin aktiv ist.

Always Tail Formatting Plugin

Dieses Plugin gibt in `have_tail` immer 1 zurück. Das heisst, dass Inodes immer in Tail Items abgelegt werden, solange dieses Plugin aktiv ist.

Small File Formatting Plugin

Dieses Plugin entscheidet in `have_tail` je nach Grösse der Datei. Ist die Datei grösser als vier Blocks (momentan also grösser als 16 Kilobytes) dann wird es als Extent Item abgelegt. Alle kleineren Dateien werden als Tail Item abgelegt.

6.9.8 JNode Plugins (REISER4_JNODE_PLUGIN_TYPE)

Die JNode Plugins werden benötigt um die verschiedenen JNode Typen zu implementieren. Da in reiser4xp `jnode`'s nur für Formatted Nodes benötigt werden ist auch nur das dafür benötigte Plugin, mit dem Typ `JNODE_FORMATTED_BLOCK`) implementiert.

6.9.9 Regular File Plugins (REISER4_REGULAR_PLUGIN_TYPE)

Das Design von diesem Plugin ist nicht logisch. Es wäre logisch wenn das Regular File Plugin das Lesen und Schreiben der Daten einer regulären Linux Datei implementieren würde. Dies ist jedoch nicht der Fall. Das Regular File Plugin spezifiziert nur welches der File Plugins verwendet werden soll. Reiser4 nutzt dieses Plugin wenn eine neue Datei angelegt wird. Es legt dann das im Regular File Plugin spezifizierte File Plugin für den neuen Inode an.

Die folgenden Regular File Plugins sind in Reiser4 implementiert:

Plugin	Beschreibung
Unix File Regular Plugin	Verweist auf das Regular File Plugin
Cryptocompress Regular Plugin	Verweist auf das Cryptocompress File Plugin.

6.9.10 Hash Plugins (REISER4_HASH_PLUGIN_TYPE)

Wenn Datei- oder Verzeichnisnamen zu lang sind um in einen Reiser4 Schlüssel zu passen wird aus dem Dateinamen ein Hashwert generiert welcher dann einen Teil des Schlüssels bildet. Zur Generierung von Hashwerten wird jeweils eines der vorhandenen Hash Plugins verwendet. Die folgenden Angaben stammen zum grössten Teil von [NSYS02]. Reiser4 besitzt die folgenden Hash Plugins:

Typ	Beschreibung
Rupasov	Diese Hash-Funktion wurde erfunden von Yury Yu. Rupasov (yura@yura.polnet.botik.ru). Sie ist schnell und generiert für ähnliche Namen ähnliche Hashwerte. Hans Reiser empfielt diese Hashfunktion nicht zu nutzen, da sie oft doppelte Hashwerte generiert.
Tea	Nicht näher spezifiert.
R5	Der R5 Algorithmus ist eine weiterentwicklung des Rupasov Hashes. Er wird standardmässig von Reiser4 verwendet.
FNV1	Nicht näher spezifiert.

6.9.11 Fibration Plugins (REISER4_FIBRATION_PLUGIN_TYPE)

Im entsprechenden Kapitel über die Reiser4 Keys wurde erklärt, dass grundsätzlich der erste Teil eines Schlüssels jeweils aus dem Namen einer Datei oder eines Verzeichnisses besteht. Die Konsequenz daraus ist, dass Dateien mit ähnlichem Namen im Baum nebeneinander abgelegt werden. Dies kann unter Umständen zu reduzierter Performance führen. Beispielsweise möchte man vermeiden, dass Quellcode und der daraus resultierende Object-Code (gespeichert in *.o-Files) gleich nebeneinander zu liegen kommen. Besser wäre es wenn die Object-Files an einem vom Quellcode getrennten Ort im Baum abgelegt würden.

Fibration Plugins sollen dieses Problem beheben. Die ersten 7 Bit in einem Schlüssel sind reserviert für den sogenannten Fibration Code. Der Dateiname wird erst danach in den Schlüssel gefüllt. Dieser Fibration Code wird jeweils durch das Fibration Plugin aus dem Dateinamen generiert. Je nach Plugin können so verschiedene Strategien zur physikalischen Trennung von Dateien implementiert werden.

Einfaches Fibration Plugin (FIBRATION_LEXICOGRAPHIC)

Dieses Plugin generiert keinen Fibration Code. Der Fibration Code wird unter Verwendung dieses Plugins immer 0 sein.

Object-File Fibration Plugin (FIBRATION_DOT_O)

Dieses Plugin unterscheidet zwischen .o-Dateien und anderen Dateien. Für .o-Dateien wird ein Fibration Code von eins verwendet, für alle anderen einer von null.

Ext1 Fibration Plugin (FIBRATION_EXT_1)

Dieses Fibration Plugin unterscheidet zwischen Dateien mit einer einstelligen Dateiendung und anderen Dateien. Dateien mit einer einstelligen Endung erhalten den ASCII-Code der Endung und alle anderen erhalten 0 als Fibration Code

Ext3 Fibration Plugin (FIBRATION_EXT_3)

Das Ext3 Fibration Plugin funktioniert ähnlich wie das Ext1 Fibration Plugin. Der Unterschied liegt darin, dass bei Dateien mit einer dreistelligen Dateiendung eine Art Hash aus der Endung als Fibration Code zurückgegeben wird.

6.9.12 SdExt Plugins (REISER4_SD_EXT_PLUGIN_TYPE)

SdExt Plugins erweitern das Stat Data Item Plugin. Jedes der SdExt Plugins speichert gewisse Dateieigenschaften. Aufgrund der speziellen Eigenschaft des Stat Data Item Plugins kann es maximal 32 solche SdExt Plugins geben.

Leight Weight Stat Plugin

Das Leight Weight Stat Plugin speichert die Basis-Informationen über einen bestimmten Inode. Es ist folgendermassen strukturiert:

```
structure reiser4_light_weight_stat {
  unsigned_16 mode print_hexadecimal
  unsigned_32 nlink print_decimal
  unsigned_64 size print_decimal
}
```

Die Member besitzen die folgende Beschreibung

Member	Beschreibung
mode	Das mode Member spezifiziert was für einen Typ der Inode hat, das heisst ob der Inode ein Verzeichnis, eine reguläre Datei oder etwas anderes ist.
nlink	Nicht näher spezifiziert.
size	Grösse des Inodes in Bytes.

Unit Stat Plugin

Dieses Plugin implementiert die Standard Unix-Eigenschaften einer Datei. Dies sind einerseits die Besitzer (User, Group) und andererseits die letzten Zugriffsdaten. Nachfolgend die auf der Festplatte gespeicherte Struktur:

```
structure reiser4_unix_stat {
  unsigned_32 uid print_decimal
  unsigned_32 gid print_decimal
  unsigned_32 atime print_decimal
  unsigned_32 mtime print_decimal
  unsigned_32 ctime print_decimal
  unsigned_64 bytes print_decimal
}
```

Beschreibung der Member:

Member	Beschreibung
uid	Definiert die Id des Benutzers der den Inode besitzt.
gid	Definiert die Id der Gruppe die den Inode besitzt.
atime	Definiert die Zeit zu welcher der letzte Zugriff auf den Inode erfolgte.

Member	Beschreibung
	Die Zeitangabe ist die Anzahl der 100-Nanosekunden-Intervalle die seit dem 1. Januar 1970 vergangen ist.
mtime	Definiert die Zeit zu welcher der tatsächliche Inhalt des Inodes (also einer Datei oder eines Verzeichnisses) zuletzt verändert worden ist. Die Zeitangabe ist die Anzahl der 100-Nanosekunden-Intervalle die seit dem 1. Januar 1970 vergangen ist.
ctime	Definiert die Zeit zu welcher zuletzt die Attribute eines Inodes (also Besitzer, Zugriffsrechte, etc.) geändert worden sind. Die Zeitangabe ist die Anzahl der 100-Nanosekunden-Intervalle die seit dem 1. Januar 1970 vergangen ist.
bytes	Nicht näher spezifiert.

Large Times Stat Plugin

Das Large Times Stat Plugin kann genauere Zeitangaben als das normale Unix Stat Plugin speichern. Es kann Zeitangaben bis auf eine Nanosekunde genau speichern. Auf der Festplatte sieht die Struktur des Plugins folgendermassen aus:

```
structure reiser4_large_times_stat {
  unsigned_32 atime print_decimal
  unsigned_32 mtime print_decimal
  unsigned_32 ctime print_decimal
}
```

Beschreibung der Struktur-Member:

Member	Beschreibung
atime	Definiert die Zeit zu welcher der letzte Zugriff auf den Inode erfolgte. In welchem Zeitmassstab die Angabe gemacht wird ist nicht bekannt.
mtime	Definiert die Zeit zu welcher der tatsächliche Inhalt des Inodes (also einer Datei oder eines Verzeichnisses) zuletzt verändert worden ist. In welchem Zeitmassstab die Angabe gemacht wird ist nicht bekannt.
ctime	Definiert die Zeit zu welcher zuletzt die Attribute eines Inodes (also Besitzer, Zugriffsrechte, etc.) geändert worden sind. In welchem Zeitmassstab die Angabe gemacht wird ist nicht bekannt.

Symlink Stat Plugin

Wenn ein Inode ein Symlink ist, dann besitzt das Stat Data Item Plugin ein Element von diesem Plugin-Typ. Es speichert das Ziel des Symlinks. Die auf der Festplatte gespeicherte Struktur sieht folgendermassen aus:

```
structure reiser4_symlink_stat {
  CHAR body array item_length
}
```

Das einzige Member body enthält den Zielpfad.

Plugin Stat Plugin

Jeder Inode besitzt ein Set von Plugins. Dieses Plugin Set kann dann die Operationen des Inodes ausführen (je nach Operation wird das passende Plugin verwendet). Das Plugin Stat Plugin speichert die Id's aller für das Plugin Set benötigten Plugins. Die auf der Festplatte gespeicherte Struktur sieht folgendermassen aus:

```
structure reiser4_plugin_slot {
```

```
  unsigned_16 pset_memb print_decimal
  unsigned_16 id print_decimal
}
structure reiser4_plugin_stat {
  unsigned_16 plugins_no
  reiser4_plugin_slot slot array plugins_no
}
```

Das Plugin speichert ein Array von `reiser4_plugin_slot` Elementen. Die Anzahl der Elemente wird durch `plugins_no` definiert. Jedes Element enthält mit `pset_memb` die Id des referenzierten Plugin Set Typs und mit id die Id des verwendeten Plugins.

Flags Stat Plugin

Dieses Plugin speichert das `i_stat` Member eines Inodes.

Capabilities Stat Plugin

Dieses Plugin besitzt bei Reiser4 keine Implementierung.

Crypto Stat Plugin

Dieses Plugin wird für das Cryptocompress File Plugin verwendet. Dessen genauer Aufbau ist daher für den Read-Only Treiber nicht relevant.

7 Tests

7.1 Test PT1 Treiberrumpf

7.1.1 Blackboxtest Treiberrumpf laden (FA1b)

Voraussetzungen

Der Treiber wurde in der Windows-Registry manuell mit dem OSR Driver Loader (siehe [OSR01]) als Service registriet.

Testspezifikation

Nach dem Start des Treibers über das Command-Prompt mit `net start reiser4xp` läuft das Betriebssystem weiter und es werden keine Exceptions geworfen.

An den entsprechenden Debug-Ausgaben im Kernel-Debugger ist ersichtlich, das der Treiber läuft und verschiedene Funktionen von Windows aufgerufen werden.

Resultat

Start mehrmals erfolgreich durchgeführt. Kernel-Debugger zeigt verschiedene Funktionsaufrufe. Da die meisten Funktionen noch nicht implementiert wurden, erscheinen gelegentlich Fehlermeldungen, das System läuft jedoch stabil weiter.

Fazit

FA1b wurde vollständig implementiert.

7.1.2 Blackboxtest Partition erkennen (FA1a)

Voraussetzungen

Blackboxtest Treiberrumpf laden erfolgreich abgeschlossen.

Testspezifikation

Nachdem das Disk Management Tool (unter Administration > Computer Management) gestartet wurde, ruft Windows den Treiber für jede gefundene Partition auf. Der Treiber prüft jede Partition auf eine Magic String und gibt eine Meldung im Kernel-Debugger aus, wenn eine Reiser4-Partition gefunden wurde. Die Meldung darf natürlich nur bei echten Reiser4-Partition ausgegeben werden. Unter Linux ist ersichtlich, wo sich überall ein Reiser4 Dateisystem befinden muss.

Resultat

Die Nachricht wird bei der korrekten Partition (hier Nummer 4) ausgegeben. Die Funktionen werden auch bei den anderen ext2-Partitionen aufgerufen (Linux, Swap), dort wird die KdPrint-Nachricht jedoch wie erwartet nicht ausgegeben.

Fazit

Der Test verlief erfolgreich, FA1a ist fertig implementiert.

7.1.3 Exception-Handler

Voraussetzungen

Blackboxtest Partition erkennen erfolgreich abgeschlossen.

Testspezifikation

Im Treibercode werden künstlich an beliebigen Stellen einige Exceptions geworfen. Sie sollten im Kernel-Debugger, sowie im Event Log von Windows erscheinen.

Resultat

Die Exception wird korrekt angezeigt im Debugger. Im Event Log wird der Eintrag gemacht, die Beschreibung ist für den Normalbenutzer jedoch nicht sehr verständlich.

Fazit

Der Test verlief erfolgreich, Exception-Handler ist abgeschlossen.

7.1.4 Memory Manager

Voraussetzungen

Keine.

Testspezifikation

In einer Testfunktion im Treibercode wird künstlich Speicher alloziiert und nicht wieder freigegeben und künstlich ein Memory Leak erzeugt. Der Memory Manager soll das Leak im Kernel-Debugger anzeigen.

Resultat

Der Memory Manager findet das Memory Leak.

Fazit

Der Test verlief erfolgreich, Memory Manager ist abgeschlossen.

7.2 Test PT2 Volumeinfo

7.2.1 Blackboxtest Partitionsinformationen lesen (FA2)

Voraussetzungen

Kernel-Debugger ist verbunden, Treiber ist gestartet. Eine Reiser4 Partition ist dem Laufwerksbuchstaben `R:` zugeordnet. Seriennummer, Grösse und freier Speicher der Partition sind bekannt (unter Linux ersichtlich).

Testspezifikation

Laut Anforderungsspezifikation müssen die Partitionsinformationen in neun von zehn Fällen richtig zurückgegeben werden. Der Test wird im Blackbox-Verfahren auf Benutzerebene manuell durchgeführt. Dabei werden die Partitionsinformationen je zehn mal auf folgende Arten abgefragt:

Command Prompt

Über das Windows Command Prompt werden mit folgenden Befehlen die Partitionsinformationen angezeigt:

```
> R:
> dir
```

Der angezeigte Volumename muss in jedem Fall „Reiser4-Partition“ lauten. Die Seriennummer der unter Linux ermittelten Seriennummer entsprechen.

Windows Explorer

Anzeige der Volume-Eigenschaften über den entsprechenden Eintrag im Kontextmenü des Volumes im Windows Explorer.

Der Volumename muss wiederum „Reiser4-Partition“ lauten, unter „Dateisystem“ muss reiser4 aufgeführt sein. Belegeter und Freier Speicher müssen den unter Linux ermittelten Werten entsprechen.

Disk Management

Im Disk Management (unter Administration > Computer Management) muss der Name des Volumes „Reiser4-Partition“ lauten, Kapazität und freier Speicher müssen den unter Linux ermittelten Werten entsprechen. Der Partitionstyp wir hier nicht angezeigt, wenn der Treiber erst nach dem Systemstart geladen wurde und muss daher nicht geprüft werden.

Resultat erster Testlauf

Die Partitionsinformationen werden je zehn mal korrekt angezeigt.

- CommandPrompt: folgende Infos werden ausgegeben:
  Volumeseriennummer und das Volumelabe.
- Windows Explorer: folgende Infos werden grafisch dargestellt:
  Volumelabel, Filesystem (Type), Used Diskspace, Free Diskspace.
- Disk Management: folgende Infos werden dargestellt:
  Volumelabel, Filesystem (Type), Free Diskspace.

Fazit erster Testlauf

Alle Tests verliefen erfolgreich, FA2 ist abgeschlossen.

7.2.2 Blackboxtest Verzeichnisbaum lesen (FA3)

Voraussetzungen

Kernel-Debugger ist verbunden, Treiber ist gestartet. Dem Laufwerksbuchstaben `R:` ist eine Reiser4-Partition mit einer beliebigen Datei- und Verzeichnisstruktur zugeordnet, die jedoch noch keine speziellen Dateien wie beispielsweise symbolic Links enthal-

ten darf. Java Runtime Environment 1.4 oder höher ist auf dem Testsystem unter Linux und Windows installiert, ebenso muss das Java-Testprogramm „EnumDirTest.jar“ sowohl unter Linux als auch unter Windows verfügbar sein. Wegen der plattformunabhängigkeit von Java kann genau das selbe Programm verwendet werden, es sollte sich auf alle Plattformen gleich verhalten.

Testspezifikation

Um die festgelegten Anforderungen zu erfüllen, muss der Inhalt eines beliebigen Verzeichnisses in neun von zehn Fällen richtig zurückgegeben werden. Der Test wird im Blackbox-Verfahren auf Benutzerebene mit Hilfe von „EnumDirTest.jar“ zum Grossteil automatisch durchgeführt. Um für die Richtigkeit des Verzeichnisbaums eine Referenz zu haben, muss der Test auch unter Linux durchgeführt werden.

Zuerst wird „EnumDirTest.jar“ unter Linux gestartet. Der erste Parameter ist dabei das zu testende Verzeichnis einer Reiser4-Partition, der zweite der Name einer Zieldatei für die Ausgabe. Das Programm iteriert rekursiv durch das angegebene Verzeichnis und gibt Informationen (Name, Grösse in Bytes, letzte Änderung) zu allen gefundenen Dateien und Verzeichnisse aus.

Die Ausgabedatei dient nun als Referenz, solange der Inhalt des Testverzeichnisses nicht ändert und muss sich deshalb für die Fortführung des Tests an einem von Windows erreichbaren Ort befinden.

Danach wird das Testprogramm unter Windows auf das gleiche Verzeichnis der Reiser4-Partition angewendet, natürlich mit einer anderen Ausgabedatei. Zum Schluss werden die beiden Ausgabedateien mit `diff` verglichen. Sind sie identisch, war der Testlauf erfolgreich.

Resultat erster Testlauf

Unter Linux sind Grösse und Änderungsdatum von Dateien, die nicht-US-ASCII Zeichen enthalten (hier ein „ñ“) immer Null.

Unter Windows stürzt das Testprogramm ab. Die Ausgabedatei enthält einige richtige Einträge.

Analyse erster Testlauf

Bei den falschen Werten unter Linux scheint es sich um eine Eigenart der Linux-Implementation der Java-Bibliothek zu handeln, denn die Werte werden beispielsweise in der Shell richtig angezeigt. Unter Windows arbeitet das Testprogramm ebenfalls korrekt, auch mit Sonderzeichen im Dateinamen.

Das Windows-Problem tritt nur bei Dateien mit sehr langen Namen (ab etwa 100 Zeichen) auf, weil der Treiber einen Überlauf des Rückgabebuffers für die Dateiinformationen nicht korrekt an das BS meldet.

Fazit erster Testlauf

Für das erste Problem wurde keine Lösung gefunden. Da hier die Fehlerquelle das Testprogramm ist und nicht etwa der Treiber , wird in Zukunft auf Dateiennamen mit Sonderzeichen verzichtet.

Der Test war nicht erfolgreich und muss nach der Korrektur des übrig bleibenden Fehlers wiederholt werden.

Resultat zweiter Testlauf

Das Testprogramm wird fehlerfrei ausgeführt. Der Vergleich mit der Referenzdatei zeigt aber Unterschiede: Dateien mit einer Erweiterung von nur einem Buchstaben (z.B. „.h“ oder „.c“) werden unter Windows nicht zurückgegeben.

Analyse zweiter Testlauf

Der Node-Schlüssel von Dateien mit einer Erweiterung von nur einem Buchstaben hat eine bestimmte Form. Der Fehler triff auf, weil durch eine falsch gesetzte Konstante im Reiser4-Code genau diese Schlüssel im Baum nicht gefunden werden können.

Fazit zweiter Testlauf

Nach der Korrektur wird der Test nochmal wiederholt.

Resultat dritter Testlauf

Das Testprogramm wird fehlerfrei ausgeführt, die Ausgabedatei entspricht bis auf die oben erwähnte Einschränkung der Referenzdatei.

Fazit dritter Testlauf

Alle Tests verliefen erfolgreich, FA3 ist für normale Dateien und Verzeichnisse abgeschlossen. Später wird der Test für spezielle Dateien wie symbolic Links wiederholt.

7.3 Test PT4 Lesen

7.3.1 Blackboxtest Datei lesen (FA4)

Voraussetzungen

Kernel-Debugger ist verbunden, Treiber ist gestartet. Eine Reiser4-Partition ist dem Laufwerksbuchstaben `R:` zugeordnet. Java Runtime Environment 1.4 oder höher ist auf dem Testsystem unter Linux und Windows installiert, ebenso muss das Java-Testprogramm „ReadFileTest.jar“ sowohl unter Linux als auch unter Windows verfügbar sein. Wegen der plattformunabhängigkeit von Java kann genau das selbe Programm verwendet werden, es sollte sich auf alle Plattformen gleich verhalten.

Testspezifikation

Laut Anforderungsspezifikation müssen Dateien in neun von Zehn Fällen korrekt gelesen werden. Der Test wird im Blackbox-Verfahren auf Benutzerebene mit Hilfe von „ReadFileTest .jar“ zum Grossteil automatisch durchgeführt. Zuerst werden unter Linux mit dem Testprogramm (genaue Beschreibung im Sourcecode) folgende Testdateien erstellt:

- testfile1.bin: Null Byte grosse Datei
- testfile2.bin: 4096 Byte grosse Datei (entspricht der Grösse einer Windows Page), gefüllt mit einer Folge des Integers (Java) 8745
- testfile3.bin: 4096 + 1 Byte grosse Datei (entspricht knapp zwei Windows Pages), gefüllt mit einer Folge des Integers (Java) 8745
- testfile4.bin: 16 KB grosse Datei (wird in Reiser4 gerade noch direkt im Baum gespeichert), gefüllt mit einer Folge des Integers (Java) 8745

- testfile5.bin: 16 KB + 1 Byte grosse Datei (wird gerade nicht mehr direkt im Reiser4-Baum gespeichern), gefüllt mit einer Folge des Bytes 3
- testfile6.bin: ~10 MB grosse Datei, gefüllt mit einer Folge des Strings „Test“

Danach werden die Testdateien unter Windows vom Testprogramm gelesen und auf ihre Korrektheit geprüft. Über eine vorher unter Linux automatisch erstellte Konfigurationsdatei weiss das Programm, welchen Inhalt die Dateien haben sollten und was ihre richtige Grösse ist.

Resultat erster Testlauf

- testfile1.bin: Test erfolgreich
- testfile2.bin: Test erfolgreich
- testfile3.bin: Test erfolgreich
- testfile4.bin: Test erfolgreich
- testfile5.bin: Test erfolgreich
- testfile6.bin: Test erfolgreich

Fazit erster Testlauf

Alle Tests verliefen erfolgreich, FA4 ist abgeschlossen.

7.3.2 Blackboxtest Dateiattribute lesen (FA5)

Dieser Test wird nach den Spezifikationen des Tests PT2 Volumeinfo / Blackboxtest Verzeichnisbaum lesen (FA3) durchgeführt.

Voraussetzungen

Es gelten die gleichen Voraussetzungen wie beim Blackboxtest Verzeichnisbaum lesen, mit der Ausnahme, dass die Partition nun auch symbolic Links enthalten darf.

Testspezifikation

Gemäss Blackboxtest Verzeichnisbaum lesen.

Resultat

Das Problem mit nicht-US-ASCII Dateinamen wurde bereits dokumentiert und bleibt bestehen.

Softlinks werden unter Windows nicht angezeigt.

Hardlinks werden korrekt dargestellt und aufgelöst.

Alle Dateiattribute werden korrekt gelesen.

Analyse

Die Softlinks werden absichtlich nicht angezeigt, da sie unter Windows nicht aufgelöst werden können. Das liegt daran, das für die Auflösung bekannt sein müsste, in welchem Verzeichnis die Reiser4-Partition unter Linux gemountet war, denn die Zielpfade der Softlinks sind jeweils absolut von der Root-Partition ausgehend gespeichert (beginnen z.B. immer mit `/mnt/reiser4`)

Fazit

Das Problem mit den Softlinks kann nicht gelöste werden, da keine geeignete Lösung gefunden wurde.

Ansonsten war der Test erfolgreich. FA5 ist somit abgeschlossen.

8 Schlussfolgerungen

8.1 Zusammenfassung

Nach einem Studium verschiedener Beispieltreiber und Dokumentationen konnte zu Projektbeginn relativ schnell der erste Prototyp spezifiziert und umgesetzt werden. Danach startete die Hauptarbeit an der Portierung von Reiser4 sowie die Implementation der komplexeren Treiberschnittstellen, was den Fortschritt ein wenig verlangsamte. Durch das gewählte Prozessmodell konnten die zahlreich auftretenden Probleme isoliert analysiert und gelöst werden. Erschwert wurde die Entwicklung allgemein durch mangelhafte oder veraltete Dokumentation, Windows-seitig auch durch fehlerhafte Beispieltreiber. Hauptprobleme bei der Portierung des bestehenden Reiser4-Treibers waren die langwierige Einarbeitung in den Code und die Nachbildung von Linux Kernel-Funktionalität unter Windows. Ansonsten traten zu unserem Erstaunen verhältnismässig wenig Schwierigkeiten bei der Abbildung von Linux-Konzepten in Windows aus.

Abschliessend lässt sich sagen, dass die gesteckten Projektziele erreicht wurden. Der entstandene Prototyp enthält zwar noch einige offene Punkte und ist teilweise unstabil, kann aber gut eingesetzt werden. Damit scheint reiser4xp nach unseren Recherchen der erste Reiser4-Dateisystemtreiber für Windows zu sein.

Die gefundenen Verbesserungmöglichkeiten wurden dokumentiert und könnten in einer Fortsetzungsarbeit umgesetzt werden.

8.2 Erreichte Ziele

Der zum Projektende vorliegende Treiber erfüllt alle spezifizierten funktionalen Anforderungen, sowie das Sekundärziel FastIO. Im Einzelnen wurde folgendes realisiert:

- Analyse des Aufbaus und der Funktionsweise von Reiser4 im für die Studienarbeit benötigten Umfang
- Portierung des relevanten Reiser4-Quellcodes nach Windows
- Nachbildung der benötigten Linux-Kernel-Funktionalität
- Öffnen und schliessen von Dateien, Verzeichnissen und Partitionen
- Unterstützung für Share Access Rights beim Öffnen von Dateien und Verzeichnissen
- Auflistung von Verzeichnisinhalten
- Lesen von Datei-, Verzeichnis- und Partitionsattributen
- Lesen von Dateien mit Unterstützung für Caching und Memory-Mapped Files
- Sperren von Dateibereichen mittels Byte-Range-Locks
- Schnelles lesen von Dateien und deren Attribute durch FastIO

Reiser4xp wurde ausschliesslich für Windows XP entwickelt und getestet. Möglicherweise ist auch der Betrieb unter Windows 2000 möglich, mit älteren Betriebssystemversionen funktioniert der Treiber nicht. Unterstützt wird nur die Dateisystemversion Reiser4 mit Standardplugins.

Während der gesamten Projektdauer wurden Tests durchgeführt, um eine möglichst grosse Stabilität des Treibers zu erreichen. Die vorliegende Version läuft so stabil, dass sie sinnvoll im kleinen Rahmen eingesetzt werden kann, verursacht aber trotzdem gelegentlich Systemabstürze.

8.3 Ausblick

In der vorliegenden Treiberversion handelt es sich wie von Anfang an geplant um einen Prototyp. Dadurch bestehen viele Erweiterungsmöglichkeiten, die in diesem Kapitel erläutert werden.

Neben den dokumentierten Erweiterungsmöglichkeiten verbleiben auch einige Fehler, die gelegentlich zu Systemabstürzen führen, deren Quelle aber nicht lokalisiert werden konnte. Diese Fehler zu finden ist eine schwierige Aufgabe, da sie unvorhersehbar sind und teilweise nur auftreten, wenn kein Kernel-Debugger angeschlossen ist.

8.3.1 Schreibfunktionalität

Die offensichtlichste Erweiterungsmöglichkeit ist die Ergänzung des bestehenden Treibers um Schreibfunkktionalität. Dazu müssten vor allem die Anfrage `IRP_MJ_WRITE` und die Schreib-Callbacks des Cache Managers implementieren werden. In einem zweiten Schritt könnte man schliesslich das Schreiben über FastIO realisieren. Vermutlich würde man dabei analog zum Lesen Vorgehen, genaueres zu diesem Thema wurde in dieser Studienarbeit nicht ermittelt.

8.3.2 Unterstützung von Removable Devices

Der vorliegende Treiber unterstützt nur Reiser4-Partition auf nicht entfernbaren Festplatten vollständig. Das Lesen von Partition auf Removable Devices wie beispielsweise USB-Festplatten funktioniert grösstenteils, wurde aber nicht ausführlich getestet. Ein bekanntes Problem ist das Aus- und wieder Einstecken einer Festplatte. Dabei kommt es momentan zu einem Bluescreen, weil scheinbar beim wieder Einstecken alte, nicht mehr gültige Resourcen, die vom vorherigen Mount-Prozess übriggeblieben sind, wiederverwendet werden. Um dieses Problem zu lösen müsste die Anfrage des IRPs `IRP_MJ_FILE_SYSTEM_CONTROL` (minor Function `IRP_MN_USER_FS_REQUEST`) implementiert werden, in der man die Freigabe aller Resourcen im Zusammenhang mit dem ausgesteckten Volume forcieren würde.

Um die Robusheit des Treibers bei Removable Devices zu erhöhen, sollte zusätzlich die Verifizierung (Verifying) von Volumen implementiert werden. Grundsätzlich muss dafür vor jedem Zugriff auf die Partition geprüft werden, ob der Datenträger noch gültig ist oder eventuell entfernt wurde. Ist es nicht mehr gültig wird die aktuelle Anfrage mit einem entsprechenden Fehlercode beendet.

8.3.3 Abbildung von Linux-Benutzerrechten

Momentan hat jeder Windows-Benutzer Linux Root-Rechte auf allen Reiser4 Partitionen. Für die Durchsetzung der Linux-Benutzerrechte müsste ein Weg gefunden werden, diese auf Windows-Benutzer abzubilden. Dazu könnte eine der folgenden Möglichkeiten in Betracht gezogen werden:

Manuelle Zuordnung mit Hilfe eines separaten Tools. Die Zuordnungen werden in der Registry gespeichert und im Treibercode beim Öffnen von Dateien oder Verzeichnissen ausgelesen und geprüft. Diese für den Administrator zeitraubende Lösung wäre höchstens für eine kleine Benutzerzahl praktikabel.

Automatische Zuordnung. Hier müsste erst untersucht werden, ob es irgendwie möglich ist, die Abbildung automatisch aus der Benutzerdatenbank eines geeigneten Servers zu generieren. Dies wäre sicher die bevorzugte Lösung.

Es stellt sich jedoch grundsätzlich die Frage, ob der Treiber in grösseren Systemen mit vielen Benutzern überhaupt einen Nutzen hätte. Wenn man davon ausgeht, dass er vor allem privat auf Dualboot-Einzelarbeitsplätzen eingesetzt wird, genügt auch die aktuelle Implementierung.

8.3.4 Unterstützung von Softlinks

Softlinks werden momentan wegen des absoluten Zielpfads nicht aufgelöst. Um das Problem zu lösen, könnte man nun den Zielpfad eines Softlinks vom Root-Verzeichnis her abarbeiten und die Reiser-Partition nach jedem einzelnen Verzeichns im Pfad durchsuchen, bis ein möglicher Treffer gefunden wurde. Danach könnte man versuchen, die Zieldatei von dort aus zu öffnen. Dieses Vorgehen ist jedoch sehr fragwürdig, weil nicht sichergestellt werden kann, dass der Softlink richtig aufgelöst wird. Es besteht immer die Gefahr, dass man zufällig eine scheinbar passende Datei findet.

Als zweite Möglichkeit bietet sich auch hier die Bereitstellung eines Tools an, mit dessen Hilfe man den Linux-Mountpoint einer Reiser4-Partition manuell eintragen könnte. Dies ist zwar auch ein wenig umständlich, dürfte aber funktionieren.

8.3.5 Optimistischere Behandlung von asynchronen Anfragen

Momentan werden alle asynchronen Anfragen, welche den aufrufenden Thread nicht blockieren dürfen, sofort in eine Queue eingereiht, um später von einem System-Worker-Thread synchron abgearbeitet zu werden. Dies ist ein ziemlich pessimistisches Verhalten, weil gar nicht erst versucht wird, die Anfrage direkt zu verarbeiten, obwohl es vielleicht nirgends zu einer Blockierung kommen würde. Der Grund für diese Implementierung ist, dass es mit sehr viel Aufwand verbunden wäre, den Reiser4-Code entsprechend anzupassen. Hier müsste eine gute Lösung gefunden und kosequent sowohl im Reiser- wie auch im Windows-Teil durchgesetzt werden.

8.3.6 Unmount

Das Unmount von Volumes wird beim Beenden des Systems oder nachdem entfernbare Medien wie USB Massenspeichergeräte ausgesteckt worden sind wurde in der aktuellen Version von reiser4xp wieder auskommentiert, da es sonst zu einem Stillstand des Systems führte. Das richtige Beenden ist durchaus noch mit einem Aufwand von knapp einer Woche oder sogar noch mehr verbunden.

8.3.7 Unterstützung weiterer Plugins

Während dieser Arbeit war es nur das Ziel die Standardplugins von Reiser4 zu portieren. Damit ist der Betrieb mit einer normalen Installation gewährleistet. Sobald ein Benutzer jedoch spezielle Wünsch“ hat, das heisst seine Dateien zum Beispiel verschlüsselt abspeichern, funktioniert die implementierte Lösung jedoch nicht mehr.

Vor allem die Portierung des Cryptocompress File Plugins dürfte sehr aufwändig sein.

8.3.8 Memory Leaks

Die abgegebene Version von reiser4xp enthält noch diverse Memory Leaks. Der Treiber kann jedoch alle Memory Leaks „kontrollieren“ und gibt diese beim Beenden des Treibers im Checked-Build auf dem Remote Debugger aus. Eine mögliche, zwar mühsame aber dennoch teilweise interessante, Fortsetzung könnte es sein diese Memory Leaks zu beheben.

8.3.9 64 Bit Kompatibilität

Eine weitere Möglichkeit wäre es den Treiber auf 64Bit kompatibilität zu überprüfen und wo nötig entsprechend zu erweitern.

8.3.10 Ausführliche Tests

Um die Stabilität des Treibers zu erhöhen, müssten zahlreiche weitere Tests durchgeführt werden, insbesondere Langzeitbelastungstests, für die während der Studien-

arbeit die Zeit fehlte. Daneben sollte eine Reihe von Programmen geschrieben werden, die absichtlich möglichst viele verschiedene Systemfunktionen falsch aufrufen und andere extreme Situationen provozieren.

Es sind ausserdem noch die folgenden bekannten Fehler vorhanden deren Ursache noch nicht gefunden werden konnte:

Grosse Verzeichnisse

Eines der Testprogramme war darauf ausgerichtet grosse Verzeichnisse zu erstellen. Bei einer Verzeichnisgrösse von ca. 10000 Dateien mit zufälligen Dateinamen (mit einer Länge zwischen zwei und 100 Zeichen) kann das „dir"-Kommando in der DOS Konsole nicht korrekt ausgeführt werden. Im Explorer wird ebenfalls nichts angezeigt.

Dauertest beim Abspielen von MP3s

Gegen Ende der Arbeit wurden Dauertests durchgeführt. Wenn während bis zu ca. einer Stunde am Stück MP3-Lieder abgespielt werden, tritt nach einer gewissen Zeit eine Access Violation in einer Windows Funktion auf.

8.3.11 Refactoring

Wie bei jedem anderen Programm lässt sich auch die Qualität des Treibercodes durch diverse Refactorings steigern. Trotzdem ist es nicht zu empfehlen, die Struktur des Treibers grundlegend zu verändern, selbst wenn gewisse Funktionen sehr umfangreich sind und einige Einarbeitungszeit erfordern. Diese Struktur ist bei allen untersuchten Treiber nahezu identisch, da bei der Implementation von Schnittstellen des Betriebssystem wenig Spielraum geboten wird und die Vorgaben ja immer die selben sind. Ein Vorteil dieser Strukturierung ist, das man sich auch in fremden Treibern relativ schnell zu recht findet und den Code besser vergleichen kann.

8.4 Fortsetzungsarbeiten

So weit es sich momentan beurteilen lässt, wäre die Implementierung der Schreibfunktionalität umfangreich genug, um in einer weiterführenden Studien- und / oder Dimplomarbeit umgesetzt zu werden. Allein die Einarbeitung in die Treiberprogrammierung und den bestehenden Code würde recht viel Zeit in Anspruch nehmen. Der grösste Teil der Arbeit bestünde aber aus dem Portieren des Reiser4-Schreib-Codes. Dazu müsste auch die Journaling-Funktionalität übernommen werden, da das Schreiben unter Reiser4 Journaling zwingend voraussetzt. Der Windows-Teil der Schreibfunktionalität dürfte weniger arbeitsintensiv ausfallen, da sich hier vielerorts auf bestehenden Code aufbauen lässt. Dafür könnte zusätzlich zum Beispiel die Unterstützung für Removable Devices vervollständigt werden.

Eine weitere mögliche Fortsetzungsarbeit wäre es, die diversen kleineren Dinge des Read-Only Treibers zu erledigen. Dazu könnten nach belieben aus den oben genannten Problemen diverse Punkte ausgewählt und umgesetzt werden. Vor allem die Portierung aller Plugins, das Refactoring, das Unmount und das Entfernen der Memory Leaks wären sehr gute Verbesserungen des Treibers. Eine solche Arbeit wäre auch ideal dazu geeignet sich in den Treibercode einzuarbeiten und sich damit auf eine eventuelle Implementierung der Schreibfunktionalität vorzubereiten.

9 Entwicklungs- und Testumgebung

9.1 Entwicklungsumgebung

Auf allen drei Entwicklungslaptops wurde das Installable Filesystem Kit (IFS, Version 2600) von Microsoft für das Kompilieren und Binden des Treibers eingesetzt. Es handelt sich dabei um eine speziell auf Dateisystem-Treiber ausgerichtete Version des Windows DDKs. Im Gegensatz zum normalen DDK wird das IFS-Kit jedoch nicht frei zum Download angeboten, da es von Microsoft als vertraulich eingestufte Header-Dateien enthält. Das IFS-Kit wurde uns freundlicherweise von Prof. Dipl. Ing. Eduard Glatz zur verfügung gestellt. Übersetzt wurde der Treiber jeweils ausschliesslich für die Zielplattform Windows XP. Nähere Informationen zum IFS-Kit sind online bei Microsoft auf [MS01] erhältlich.

Um die Programmierung komfortabler zu gestalten, wurde zusätzlich das Freeware-Tool „DDKBuild“ (Version 3.12) von Hollis Technology Solutions benutzt. Es handelt sich dabei um eine Batch-Datei, welche das Erstellen von DDK-Treibern aus Mircosoft Visual Studio heraus erleichtert. Eine genaue Beschreibung ist online auf [HOLLIS01] verfügbar.

Beim verwendeten Visual Studio handelt es sich um die Version .NET 2005, sämtliche Entwicklungsrechner liefen unter Windows XP SP2.

Für das Debugging des Treibers wurde die Host-Version der Microsoft Debugging Tools 6.5 installiert, als Kernel-Debugger Frontend diente WinDbg. Die Testrechner wurden über FireWire verbunden.

9.2 Testumgebung

Als Testrechner waren zu Beginn folgende zwei Rechner vorgesehen:

IBM ThinkPad T22

- Pentium III 689 Mhz, 256 MB RAM
- IBM DJSA-220 Festplatte, Microsoft Treiber Version 5.1.25.35.0
- Windows XP Checked Build 2600.xpsp_sp2_rtm.040803-2158
- SUSE Linux 9.3, Kernel 2.6.11
- Reiser4, Default-Plugins
- Partitionen: [NTFS `C:\`] [Ext3] [Linux Swap] [Reiser4, 2.5 GB, `R:\`]

Asus Barebone

- Celeron 2.4 Ghz, 448 MB RAM
- „Standard Disk Drive“, Microsoft Treiber Version 5.1.25.35.0
- Windows XP Checked Build 2600.xpsp_sp2_rtm.040803-2158
- SUSE Linux 10, Kernel 2.6.13
- Reiser4, Default-Plugins
- Partitionen: [Linux Swap] [Reiser3] [NTFS `C:\`] [FAT32 `D:\`] [Reiser4, 17.8 GB, `R:\`]

Da aus unerklärlichen Gründen das Kernel-Dubugging über FireWire beim Asus Barebone nicht funktionierte, wurde schlussendlich nur das IBM ThinkPad zum Debuggen eingesetzt.

Der Treiber wurde auf den Testrechnern von Hand in der Registry installiert. Unter `HKLM\System\CurrentControlSet\Services\reiser4xp` wurden folgende Werte hinzugefügt:

Name	Typ	Wert
DisplayName	REG_SZ	Reiser4x
ErrorControl	REG_DWORD	0x1
ImagePath	REG_EXPAND_SZ	\??\C:\Windows\System32\drivers\reiser4xp.sys
Start	REG_DWORD	0x3
Type	REG_DWORD	0x2

Der Starttyp `0x3` bedeutet, dass er Treiber manuell nach dem Systemstart über die Kommandozeile mit `net start reiser4xp` gestartet werden muss.

Um der getesteten Reiser4-Partition einen Laufwerksbuchstaben zuzuordnen, wurde folgender Eintrag in der Registry unter `HKLM\System\CurrentControlSet\Control\SessionManager\Dos Devices` vorgenommen:

Name	Typ	Wert
R:	REG_SZ	\Device\Harddisk0\Partition4

9.3 Vorgehen Debugging

Das Debuggen eines Treibers erfordert ein etwas anderes Vorgehen und andere Software als bei einer normalen Benutzerapplikation. Grundsätzlich ist ein spezieller Kernel-Debugger nötig, der auf unterster Ebene eng mit dem Betriebssystem zusammenarbeitet. Dadurch wird es möglich, selbst bei schweren Systemabstürzen (Bluescreens) noch nützliche Informationen über den Fehler zu erhalten. Dazu sollte der Debugging-Host, also jene Komponente, die das Benutzerinterface des Debuggers darstellt, idealerweise auf einen andere Computer als dem eigentlichen Testrechner laufen. Wäre das nicht der Fall, würde bei einem Systemabstutz natürlich auch der Debugger nicht mehr reagieren. Trotz dieses Nachteils gibt es auf dem Markt auch Kernel-Debugger (beispielsweise DebugView, [SYSINT01]), die mit dem zu testenden Treiber auf einem einzelnen Rechner betrieben werden können. Für reiser4xp wurde die erste Variante mit zwei Computern eingesetzt.

Selbst kleine Fehler im Treiber können zu Systemabstürzen und damit zu einem Neustart des Rechners führen. Da sich gewisse Fehlersituation schwer reproduzieren lassen ist es besonders wichtig, zum Zeitpunkt des Absturzes möglichst viele Informationen über den Fehler und den internen Zustand des Treibers im Debugger auszugeben. Zu Projektbeginn wurden dazu ein drei C-Makros definiert, welche einerseits die Ausgabe und Filterung von Debuginformationen vereinfachen, andererseits den Eintritt in Funktionen deren Abschluss protokollieren. Die Protokollierung der Funktionsaufrufe erwies sich mit steigendem Codeumfang als unbrauchbar, da es häufig zu unerklärlichen Deadlocks und anderen Synchronisationsproblem kam, und der Debugger duch die Informationsflut zu sehr abgebremst wurde. Mit steigender Erfahrung bei der Benutzung des Kernel-Debuggers wurde deshalb vermehrt auf im integrierten Funktionen zurückgegriffen. Funktionseintritte wurden mit Hilfe des Callstacks überwacht, anstatt Variabeln im als Text im Debugger auszugeben, wurden im Code Haltepunkt gesetzt. Dadurch konnten beispielsweise lokale Variabeln direkt im „Locals"-Fenster des Debuggers eingesehen werden.

Der Windows Kernel-Debugger bietet unzählige weitere Funktionen, die das Debuggen erleichtern können, leider fehlte im Rahmen dieser Studienarbeit die Zeit, sich richtig einarbeiten zu können. Die Verschiedenen Möglichkeiten sind in der Hilfe der Windows Debugging Tools genauer beschrieben.

Wie bereits im Projektplan erwähnt, wurde zum Debuggen zusätzlich der Driver Verifier von Microsoft eingesetzt. Dieses standardmässig mit Windows mitgelieferte Programm provoziert, wenn es eingeschaltet wird, automatisch verschiedene Fehlersituationen, indem es an den Treiber gesendete Anfragen verändert. Beispielsweise werden Buffer so verkleinert, damit die angefragten Informationen keinen mehr Platz darin finden. Zusätzlich werden die vom Betriebssystem im Debugger ausgegebenen Inforamationen durch nützliche Einzelheiten ergänzt. Der Einsatz des Verifiers ist nicht unbedingt notwendig, wird aber von Microsoft empfohlen, da so Fehler, die im normalen Betrieb vielleicht lange verborgen geblieben wären, schon viel früher auftauchen.

Als besonders heimtückisch und schwierig zu Debuggen haben sich Fehler im Zusammenhang mit Concurrency erwiesen. Einerseits können Thread-Wechsel bekanntermassen jederzeit stattfinden, auch wenn man gerade auf einen bestimmten Fehler im aktuellen Thread wartet. Andererseits kann beispielsweise der Cache-Manager des Bestriebssystems jederzeit asynchron Leseoperationen starten. Für diese Probleme wurde keine echte Lösung gefunden. Ideal wäre, wenn man z.B. über den Debugger das Multithreading-Verhalten von Windows beeinflussen könnte, allerdings dürfte das technisch schwer realisierbar sein.

Eine wertvolle Hilfe beim Debuggen war auch der zu Anfang implementierte Memory-Manager. Mit dessen Hilfe konnten Buffer-Overruns und Underruns sowie Memory-Leaks zuverlässig und frühzeitig erkannt werden.

Teil IV: Anhang

A Glossar

BLOB	Binary Large Objekts. Grosser, zusammenhängende Blocks binärer Daten.
BS	Betriebssystem.
Byte-Range-Lock	Schreib- bzw. Lesesperren auf Dateien unter Windows.
Cache	Zwischenspeicher.
CCB	Context Control Block. Entsprechung eines Datei-Handles im Windows Kernelmodus
DDK	Driver Development Kitl. Blbliothek und Umgebung für die Treiberentwicklung.
Dentry	Name eines Inodes.
EResource	Synchronisationsobjekt vom Windows Kernel.
FastIO	Windows-Treiberschnittstelle für schnelle Ein- und Ausgabe.
FCB	File Control Block. Darstellung einer Datei auf der Fesplatte unter Windows
File Handle	Zeiger auf eine Datei.
FSCTL	File System Control Code. Anfrage an ein Dateisystem im Windows IO Manager.
GPL	GNU Public License. Lizenz für freie Software.
GUID	Global Unique Identifier. „Weltweit" einmalige Identifikationsnummer.
Hardlink	Referenzgezählter Link auf ein Inode unter Linux.
Inode	Beliebiges Objekt in einem Linux-Dateisystem.
IOCTL	IO Control Code. Anfragecode des Windows IO Managers.
IRP	IO Request Packet. Kommunikationspaket des Windows IO Managers.
Journaling	Protokollierung von Dateisystemzugriffen.
MSDN	Microsoft Developer Network. Entwicklerdokumentation für Microsoftumgebungen.
Pagefile	Auslagerungsdatei des Hauptspeicher unter Windows.
Plugin	Austauschbare Softwarekomponente.
Portierung	Anpassen eines Programms an ein anderes Betriebssystem.
Removable Media	Datenträger, die während dem Betrieb des Betriebssystems entfernt werden können.
Share Access Rigth	Beim Öffnen von Dateien gewährte Rechte.
Softlink	Link auf ein Inode unter Linux.
Struktured Exception Handling	Fehlerbehandlungsmechanismus von Windows.
SUSE	Linux-Distribution.
VFS	Virtual File System. Dateisystemschnittstelle unter Linux.
VMM	Virtual Memory Manager. Verwaltung des virtuellen Speichers unter Windows.
VPB	Volume Parameter Block. Verbindung zwischen einem Laufwerk und der physikalischen Partition.

B Literaturverzeichnis

[FBU01]: Florian Buchholz, **The structure of the Reiser file system**; http://homes.cerias.purdue.edu/~florian/reiser/rei, *Februar 2006*

[GKU01]: Gerson Kurz, **rfstool**; http://p-nand-q.com/download/rfstool/overview.html, *Februar 2006*

[MPI01]: Mark W Piper, **rfsd**; http://rfsd.sourceforge.net/, *Februar 2006*

[NDS01]: D. Lüönd, D. Zwirner, J. Krebs, **Ext2 Dateisystemtreiber für Windows XP**; , *2002*

[MAU01]: Wolfgang Mauerer, **Linux Kernelarchitektur**; *Carl Hanser Verlag*, *2004*

[RGOOCH]: Richard Gooch, **Overview of the Virtual File System**; http://www.atnf.csiro.au/people/rgooch/linux/vfs.t, *Januar 2006*

[RNAGAR01]: Rajeev Nagar, **Windows NT File System Internals**; *O'Reilly*, *1997*

[MS02]: Microsoft, **Windows Driver Development Kit**; , *2001*

[ROMFS01]: Bo Brantén, **RomFS File System Driver**; , *2003*

[NDS02]: D. Lüönd, D. Zwirner, J. Krebs, **Ext2 Dateisystemtreiber für Windows XP**; , *2002*

[OSR02]: OSR Open Systems Resources, **IFS FAQ**; http://www.osronline.com/article.cfm?article=17#Q40, *Juli 2003*

[MS03]: Microsoft, **Microsoft Developer Network**; , *2005*

[NSYS01]: Hans Reiser, **Reasons why Reiser4 is great for you**; http://www.namesys.com/v4/v4.html, *Dezember 2005*

[WIKI01]: Wikipedia Community, **B-Baum**; http://de.wikipedia.org/wiki/B-Tree, *Dezember 2005*

[NSYS02]: Hans Reiser, **Reiser4 Hash Options**; http://www.namesys.com/mount-options.html, *Januar 2005*

[OSR01]: OSR Open Systems Resources, Inc., **OSR Online**; http://www.osronline.com, *November 2005*

[MS01]: Microsoft, **IFS Kit - Installable File System Kit**; http://www.microsoft.com/whdc/devtools/ifskit/default.mspx, *Januar 2006*

[HOLLIS01]: Hollis Technology Solutions, **DDKBUILD - Integrating the Windows DDK with Visual Studio .Net**; http://hollistech.com/Resources/ddkbuild/ddkbuild.htm, *November 2005*

[SYSINT01]: SysInternals, **DebugView**; http://www.sysinternals.com/Utilities/DebugView.html, *Februar 2006*

C Persönliche Berichte

1 Josias Hosang

Für die erste Studiearbeit wurde ich von Christian Oberholzer und Oliver Kadlcek angefragt, ob ich mit ihnen ein selbst ausgewähltes Projekt erarbeiten möchte. Ich kann noch nicht auf eine so grosse Programmiererfahrung zurück blicken und konnte mir die genaue Vorgehensweise bei der Entwicklung eines Treiber nicht so gut vorstellen. Ich war aber zuversichtlich, mich in die Materie einzuarbeiten und wagte die Herausforderung. Ausserdem komme ich wahrscheinlich nicht so schnell wieder in die Situation, einen Treiber zu entwerfen.

Die Einarbeitung vor und zu Beginn des Semesters war interessant; nur schon die Kommunikation mit Hans Reiser mit dem Ergebnis, dass wir auf uns alleine gestellt sind in Bezug auf die Dokumentation von reiser4.

Christian Oberholzer konzentrierte sich hauptsächlich auf die Linux Seite des Treiber, das heisst den reiser4 Code. Ich hatte kaum Einblick in diesen Bereich der Entwicklung, was einerseits ein wenig schade ist, jedoch auch gar nicht anders möglich gewesen wäre. Die Einarbeitungszeit hätte viel zu lange gedauert. So mussten wir uns auf die sehr gute Dokumentation von Christian verlassen und erhielten immer wieder einen Überblick in kurzen Diskussionen.

Oliver Kadlcek und ich, wir kümmerten uns um die Treiberschnittstelle unter Windows. Schritt für Schritt erarbeiteten wir die Konzepte; zum Teil parallel und zum Teil an gleichen Codestücken, welche knifflige Abläufe enthielten. Ich möchte erwähnen, dass Oliver sehr viel mehr an der Erarbeitung gewirkt hat als ich; mir hat immer ein wenig der Gesamtüberblick gefehlt und ich war halt langsamer mit der Progammierung in C. Der Vorteil war, dass ich dadurch sehr viel profitiert und gelernt habe. Zum Teil habe ich dann auch andere Arbeiten übernommen, wie Dokumentation, Zeiterfassung, grafische Sachen oder den Installer.

Die Zusammenarbeit im Team war sehr produktiv und hat Spass gemacht. Meinungsverschiedenheiten konnten wir gut diskutieren und eine optimale Lösung finden. Auch die Betreuung durch Prof. E. Glatz empfand ich als angenehm. Durch die wöchentlichen Sitzungen konnte er und wir immer den Stand unserer Arbeit überprüfen. Herr Glatz gab uns viel Freiheit in der Umsetzung, was uns sehr gelegen kam. So konnten wir uns auf Dinge konzentrieren die uns und ihm wichtig waren und mussten uns nicht stur an eine Vorgabe halten. Ich glaube das hat sehr viel zu einer gehaltsreichen Arbeit beigetragen.

Im ersten Quartal, bevor wir die Arbeitsplätze im Studienarbeitsraum hatten, war ein wenig umständlich, dass wir keinen festen Platz für unsere Arbeit hatten und immer die Testsysteme herumtransportieren mussten. Freundlicherweise wurde uns von Herrn Glatz ein Platz im Schrank zur Verfügung gestellt. Auch für die Testsysteme waren wir selbst besorgt. In diesen zwei Punkten wäre eine grössere Unterstützung seitens der HSR schön gewesen.

Mit dem Resultat schlussendlich bin ich sehr zufrieden. Wir haben einen funktionsfähigen Treiber erschaffen, welcher zusätzlich der erste für reiser4 ist, was immer eine weitere Motivation in diesem Projekt darstellte. Bei der Implementation haben wir eigentlich alle Ziele erreicht und zusammen mit der Dokumentation eine umfang- und inhaltsreiche Arbeit erstellt.

2 Oliver Kadlcek

Als mir Christian Oberholzer zum ersten Mal von der Idee berichtete, in der ersten Studienarbeit einen Reiser4 Dateisystemtreiber für Windows zu programmieren, fand ich das Vorhaben gewagt. Da mich aber ungewöhnliche und herausfordernde Themen interessieren, entschloss ich mich bald für die Mitarbeit am Projekt. Schnell war mit Josias Hosang auch das dritte Teammitglied gefunden.

Die Vorbereitungen für die Studienarbeit verliefen zum Glück reibungslos. Vor allem konnten wir Herrn Glatz, welcher bereits zwei sehr ähnliche Arbeiten begleitet hat, als Betreuer gewinnen. Er stellte uns schon sehr früh die Dokumentationen der beiden Arbeiten sowie das IFS Kit zur Verfügung. Dadurch konnten wir gegen Ende der Sommerferien langsam mit der Einarbeitung ins Thema beginnen und erste Abklärungen treffen. Ich bin überzeugt davon, dass diese frühe Auseinandersetzung mit dem Thema mit zum Erfolg der Arbeit beigetragen hat. Dadurch hatten wir zu Semesterbeginn bereits ein Bild davon, was ungefähr zu tun ist und mit welchem Umfang wir rechnen müssen. Das half uns bei der Planung.

Die Umsetzung verlangte viel Durchhaltevermögen. Wie schon die fünf Studenten vor uns bemerkt haben, ist die Entwicklung eines Dateisystemtreibers eine schwierige Aufgabe. Die Dokumentation ist häufig mangelhaft, die Fehlersuche ist durch das unvorhersehbare Verhalten des Kernel-Debuggers und die ständigen Neustarts mühsam, und die richtige und vor allem robuste Implamantation der Schnittstellen ist schwierig. So entstehen Fehler, die man teilweise über Stunden oder Tage suchen muss und sobald man einen Fehler beseitigt hat, taucht irgendwo ein neuer auf. Trotz dieser Schwierigkeiten fand ich das Projekt sehr Interessant, denn es eröffnete uns tiefe Einblicke in die Systemprogrammierung unter Windows und Linux.

Mit dem Resultat der Arbeit bin ich sehr zufrieden. Wir konnten wie geplant einen verwendbaren Prototypen realisieren. Es bleiben aber viele Verbesserungsmöglichkeiten, die ich einerseits gerne implenentiert hätte, andererseits bin ich auch froh, dass das Projekt abgeschlossen ist und kann mir auch nicht vorstellen, in der zweiten Studiearbeit oder der Diplomarbeit am Projekt weiterzuarbeiten.

Die Arbeit mit Josias Hosang und Christian Oberholzer hat mir Spass gemacht und war effizient. Meinungsverschiedenheiten haben meiner Meinung nach immer zu einer Verbesserung des Endproduktes beigetragen. Die Betreueung durch Herrn Glatz war für meinen Geschmack sehr gut, denn wir hatten viel Freiheit bei der Umsetzung.

Schlecht war in meinen Augen, dass uns die HSR nicht für die gesamte Projektdauer Arbeitsplätze im Studienarbeitszimmer zu verfügung gestellt hat. Dadurch mussten wir unsere Testrechner selber organisieren und ständige freie Zimmer zum arbeiten suchen.

3 Christian Oberholzer

Wir hatten uns mit dieser Arbeit relativ hohe Ziele gesteckt. Das grösste Problem am Anfang war, dass wir einerseits keine Ahnung von der Treiberentwicklung unter Windows hatten und andererseits auch keine Kenntniss über Reiser4 besassen. Die Grundlagen mit welchen wir arbeiteten waren daher nur das IFS Kit von Microsoft, einige Beispieltreiber und der Quellcode der Linuxversion von Reiser4. Das führte dazu, dass wir den Aufwand für die einzelnen gesetzten Ziele am Anfang nicht genau planen konnten und deshalb in kleinen Schritten planten. Immer wenn wir das Ziel einer Planungsphase erreicht hatten, wurde der nächste Schritt geplant.

Ich finde, dass die tatsächlich benötigte Zeit für die Implementierung danach einem interessanten Muster folgte. Der Anfang, das heisst die Informationsbeschaffung und

Einarbeitung, dauerte sehr lange. Wir haben viel Zeit gebraucht um den Treiberrumpf und einfache Funktionen zu implementieren und wir haben ebenfalls lange benötigt, um die Grundstrukturen von Reiser4 zu begreifen. Im zweiten Abschnitt der Arbeit ging dann alles sehr schnell. Wir hatten innerhalb kürzester Zeit das Lesen von Verzeichnisstrukturen und Dateien in einer ersten Fassung implementiert. Im letzten Abschnitt der Arbeit sind wir wieder sehr langsam vorangekommen und es war mühsam die vielen Fehler zu finden. Immer wenn wir der Meinung waren, eine mehr oder weniger stabile Version des Treibers zu haben, tauchten wieder neue, noch mühsamere Fehler auf.

Was die Teamarbeit betrifft wollten wir zuerst die Implementierungsaufgaben ab und zu untereinander austauschen. Die Idee dahinter war, dass jeder am Ende der Arbeit mindestens einmal mit der Implementierung von Windows Treibercode und der Portierung von Reiser4 Code zu tun hatte. Es hat sich jedoch im Laufe der Arbeit herausgestellt, dass diese Aufgabenteilung nicht möglich war weil jeweils wieder zuviel Einarbeitungszeit nötig gewesen wäre. Ich denke, dass diese Spezialisierung einer der Gründe für das gute Resultat der Arbeit ist. Jeder Beteiligte konnte am Schluss der Arbeit genau Auskunft über seinen Teil des Treibers geben und die Fehlersuche konnte so speditiv erledigt werden.

Ich denke etwas weiteres sehr Gutes war die gemischte Ansicht unseres Teams über die Art der Implementierung oder die Anzahl implementierenden Features und die dafür benötigte Zeit. Wenn ich jeweils wieder einmal daran war über das Ziel hinauszuschiessen haben mich Oliver und Josias jeweils gebremst und wieder an die richtigen Verhältniss erinnert. Andererseits denke ich, dass ich sie ab und zu auch dazu bewegen konnte mehr zu implementieren als sie ursprünglich gedacht hatten. Wenn verschiedene Möglichkeiten zur Implementierung eines Features vorhanden waren haben wir, teilweise nach hitzigen Diskussionen, uns immer auf eine gute und leicht umsetzbare Variante geeinigt.

Viel Zeit gekostet hat uns am Anfang des Semesters jeweils die Suche nach einem freien Zimmer in welchem wir Arbeiten konnten. Ausserdem hatten wir dadurch, dass wir auf dem Barebone nicht testen konnten, jeweils nur eine Maschiene zum Testen. Eine bessere Unterstützung in diesen Punkten durch die HSR hätte uns einige Zeit gespart.

Zum Schluss bin ich mit dem erreichten Resultat sehr zufrieden. Wir haben mehr Features implementiert als am Anfang geplant waren, wir haben sehr viel über die Entwicklung von Treibern unter Windows und Linux gelernt und wir haben eine Stabilität und Geschwindigkeit des Treibers erreicht die es mir erlaubten, an einem Abend einige Stunden ohne Unterbruch oder Absturz Film zu schauen.

D Programmierrichtlinien

1 Einführung

1.1 Zweck

Dieses Dokument ist ein Teil des Qualitätsmanagements des Projektes reiser4xp. Es beschreibt wie für das Projekt programmiert werden soll.

1.2 Gültigkeit

Die „Programming Guidelines“ sind über die gesamte Projektdauer von reiser4xp gültig.

1.3 Referenzen

Für die Programming Guidelines bestehen keine nötigen Grundlagendokumente.

1.4 Übersicht

Im folgenden Verlauf dieses Dokumentes wird beschrieben welche Richtlinien bei der Erstellung von Programmcode eingehalten werden sollen. Die Richtlinien sind bewusst so ausgelegt, dass dadurch zwar einheitlicher Programmcode entsteht, die Kreativität und Eigenheiten der einzelnen Mitarbeitenden aber nicht eingeschränkt werden. Erfahrungsgemäss wird effizienter und mit weniger Fehlern programmiert wenn man nicht ständig darauf achten muss die Richtlinienn nicht zu verletzen.

2 Code Organisierung und Stil

Generell gilt die Regelung, dass pro „Modul“ eine Header-Datei und eine Code-Datei erstellt werden soll. In die Header-Datei werden sämtliche Struktur Deklarationen welche in anderen Modulen genutzt werden müssen und Funktionsprototypen deklariert. Elemente welche in anderen Modulen nicht benötigt werden sollen in der .c-Datei deklariert bzw. implementiert sein.

2.1 Kommentare

Der Code wird mit den nachfolgenden Elementen kommentiert und dokumentiert.

Innerhalb von Funktionen und Methoden sollte sinnvoll kommentiert werden. Als Richtlinie ob ein Kommentar sinnvoll ist soll folgendes gelten: Es ist zum Beispiel nicht nötig eine Zeile „int a = 5“ mit „speichert den Wert 5 in die Variabel a“ zu kommentieren. Es ist hingegen sehr nützlich wenn über einer komplexen boolschen while-Schlaufenbedingung in kurzen Worten die Idee hinter der boolschen Verknüpfung erklärt wird.

Die speziell genannten Dokumentationselemente ermöglichen die Erstellung einer Online-Software-Dokumentation mittels Doxygen. Deshalb ist es wichtig, die Form der Kommentare einzuhalten. Für weitere Dokumentationselemente welche von Doxygen unterstützt werden, soll die Dokumentation (http://www.doxygen.org) gelesen werden.

2.2 Dateiheader

Jede Datei sollte einen Kommentarheader besitzen, in welchem sich einerseits die GPL-Lizenzbestimmungen befinden und der andererseits in Doxygen-Format beschreibt, was sich in der Datei befindet.

```
//-----------------------------------------------------------------------------
/*
    reiser4xp filesystem driver

    Copyright (C) 2005 Josias Hosang,
                       Oliver Kadlcek,
                       Christian Oberholzer

    This program is free software; you can redistribute it and/or
    modify it under the terms of the GNU General Public License
    as published by the Free Software Foundation; either version 2
    of the License, or (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program; if not, write to the Free Software
    Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301,
    USA.
*/
//-----------------------------------------------------------------------------
/*! \file [DATEINAME]
    \brief [KURZBESCHREIBUNG]
    \version [VERSION]
    \date [DATUM] [AUTOR], [ÄNDERUNG1]
    \date [DATUM] [AUTOR], [ÄNDERUNG2]
    \author [AUTOR], [AUTOR]
    \[WEITERE TAGS]

    [LANGE BESCHREIBUNG]
*/
//-----------------------------------------------------------------------------
```

Kurze Beschreibung der einzufügenden Elemente:

- [DATEINAME]: Name der Datei
- [KURZBESCHREIBUNG]: Eine kurze Beschreibung vom Zweck des Quellcodes der Datei. Die Kurzbeschreibung sollte nicht mehr als ca. 20 Zeichen enthalten
- [VERSION]: Version des Quellcodes in der Datei, optional
- [DATUM]: Datum in der From TT.MM.JJJJ einer Änderung. Angehängt der Autor welcher die Änderung vorgenommen hat und eine kurze Beschreibung der Änderung.
- [AUTOR]: Namen aller Personen die etwas an der Datei geändert haben.
- [WEITERE TAGS]: Alle weiteren nötigen Doxygen Tags. (Beispielsweise weitere Tags: \bug, \todo, \ingroup, etc.)
- [LANGE BESCHREIBUNG]: Eine etwas ausführlichere Beschreibung vom Zweck des Quellcodes in der Datei. Nur nötig, falls die Kurzbeschreibung nicht ausreicht.

2.3 Strukturheader

```
//-----------------------------------------------------------------------------
/*! \brief [KURZBESCHREIBUNG]
    \version [VERSION]
    \date [DATUM] [ÄNDERUNG]
```

```
    \date [DATUM] [ÄNDERUNG]
    \author [AUTOR]
    \[WEITERE TAGS]

    [LANGE BESCHREIBUNG]
*/
```

Kurze Beschreibung der einzufügenden Elemente:

- [KURZBESCHREIBUNG]: Eine kurze Beschreibung vom Zweck der Struktur. Die Kurzbeschreibung sollte nicht mehr als ca. 20 Zeichen enthalten
- [VERSION]: Version der Struktur, optional
- [DATUM]: Datum in der From TT.MM.JJJJ einer Änderung. Angehängt der Autor welcher die Änderung vorgenommen hat und eine kurze Beschreibung der Änderung.
- [AUTOR]: Namen aller Personen welche etwas an der Struktur geändert haben
- [WEITERE TAGS]: Alle weiteren nötigen Doxygen Tags. (Beispielsweise weitere Tags: \bug, \todo, \ingroup, etc.)
- [LANGE BESCHREIBUNG]: Eine etwas ausführlichere Beschreibung vom Zweck der Struktur. Nur nötig, falls die Kurzbeschreibung nicht ausreicht.

2.4 Funktionsheader

```
//------------------------------------------------------------------------------
/*! \brief [KURZBESCHREIBUNG]
    \param [PARAMETER]
    \return [RÜCKGABEWERT]
    \[WEITERE TAGS]

    [LANGE BESCHREIBUNG]
*/
```

Kurze Beschreibung der einzufügenden Elemente:

- [KURZBESCHREIBUNG]: Eine kurze Beschreibung vom Zweck der Funktion. Die Kurzbeschreibung sollte nicht mehr als ca. 20 Zeichen enthalten
- [PARAMETER]: Zuerst der Name des Parameters danach durch einen Space getrennt eine Beschreibung für jeden Parameter. D.h. Welche Daten erwartet werden.
- [RÜCKGABEWERT]: Eine genaue Beschreibung was für Werte von der Funktion zurückgegeben werden.
- [WEITERE TAGS]: Alle weiteren nötigen Doxygen Tags. (Beispielsweise weitere Tags: \bug, \todo, \ingroup, etc.)
- [LANGE BESCHREIBUNG]: Eine etwas ausführlichere Beschreibung vom Zweck der Funktion. Nur nötig, falls die Kurzbeschreibung nicht ausreicht.

2.5 Member Variabeln

```
/*!
    \var [VARIABELNAME]
    \brief [KURZBESCHREIBUNG]

    [LANGE BESCHREIBUNG]
```

```
*/
```

Kurze Beschreibung der einzufügenden Elemente:

- [VARIABELNAME]: Name der Variabel
- [KURZBESCHREIBUNG]: Eine kurze Beschreibung vom Zweck der Variabel,. Die Kurzbeschreibung sollte nicht mehr als ca. 20 Zeichen enthalten.
- [LANGE BESCHREIBUNG]: Eine etwas ausführlichere beschreibung vom Zwekc der Variabel. Nur nötig falls die Kurzbeschreibung nicht ausreicht.

2.6 Naming

Variabeln

Variabeln werden immer in Kleinbuchstaben gehalten. Einzelne „Worte“ im Variablnamen wird durch ein '_'-Zeichen getrennt. Beispiel:

```
int my_var;
```

Strukturen

Strukturen werden nach den in Java oder auch in C++ ebenfalls üblichen Konventionen benannt. Ein Strukturname beginnt immer mit einem Grossbuchstaben. Jedes „Wort“ im Namen wird durch einen Grossbuchstaben begonnen. Strukturenamen sollten nie '_'-Zeichen enthalten. Beispiel:

```
struct MyStruct
{
};
```

Funktionen

Funktionen werden wie Variabeln benannt. Funktionsnamen bestehen grundsätzlichen nur aus Kleinbuchstaben. Einzelne „Worte“ im Namen werden durch '_'-Zeichen getrennt. Zusätzlich besitzt jede Funktion eine Art Header. Dieser besteht aus der Zeichenfolge rfs[Modulkürzel]. Das Modulkürzel ist besteht aus einem oder mehreren Grossbuchstaben, Beispielsweise „T“. Beispiel für eine Funktion aus dem Modul T:

```
void rfsT_my_function()
{
}
```

Konstanten

Konstanten enthalten nie Kleinbuchstaben. Einzelne „Worte“ im Konstantennamen werden durch '_'-Zeichen getrennt. Beispiel:

```
#define MY_CONSTANT 5
```

Guards

Guard-Defines welche Header vor mehrfachem include schützen sollten wie Konstanten benannt werden. Um doppelte Guard-Namen zu verhindern soll der Pfad-der Datei vom Root-Directory aus verwendet werden. Beispiel:

```
#ifndef REISER4_DIRECTORY1_DIRECTORY2_FILENAME_H
#define REISER4_DIRECTORY1_DIRECTORY2_FILENAME_H

#endif /* REISER4_DIRECTORY1_DIRECTORY2_FILENAME_H */
```

2.7 Deklaration

Wie deklariert wird kann jeder nach eigenem Ermessen selber entscheiden.

2.8 Expressions und Statements

Wie Expressions und Statements genau aussehen kann jeder selber entscheiden.

2.9 Memory Management

Es wird im Treiberrumpf ein Memory Manager implementiert. Er überwacht das Reservieren von Speicher und entdeckt, wenn allozierter Speicher nicht mehr freigegeben wurde. Jeder sollte regelmässig nach Änderungen am Quellcode den Output des Managers überprüfen und allfällige Speicherlöcher beheben.

2.10 Error Handling und Exceptions

Fürs Error-Handling soll das „Structured Exception Handling“ des „Windows DDK“ verwendet werden. Fehler sollten nicht in Form von Return Codes zurückgegeben werden, stattdessen wird im Fehlerfall eine Exception geworfen. Zusätzlich zum Structured Exception Handling sollen die Möglichkeiten des Windows System Log verwendet werden.

E Sitzungsprotokolle

1 Sitzungsprotokoll, 4. November 2005

1.1 Sitzungsinformationen

Projekt: reiser4xp

Woche: 2

Datum: Freitag, den 4. November 2005

Zeit: 15:00-16:10

Teammitglieder / Kürzel

Josias Hosang /jho

Oliver Kadlcek / oka

Christian Oberholzer / cob

Traktanden

- Projektstatus besprechen
- Weiteres Vorgehen besprechen
- Diverses

1.2 Diskussion / Beschlüsse

Projektstatus besprechen

- Grössere Diskussion über die Zeiterfassung und die dafür nützlichen Tools. Anschliessend festhalten am Beschluss die Zeiterfassung mit einer Kombination aus MS Excel/MS Project zu machen
- Kurzes betrachten der anderen angefangenen Dokumente

Weiteres Vorgehen besprechen

- Aufgaben bis nächste Woche
- Testsystem funktioniert
- Requirements formuliert
- Architektur / Designdokument soweit als möglich fertig

Diverses:

- Allgemeine Diskussion über Textverarbeitungsprogramme
- Diskussion über Prozessmodelle
- Iteratives Prozessmodell anwenden
- Anwendungsfälle / Use Cases in diesem Fall nicht unbedingt nötig

- Das Prozessmodell ist frei wählbar, Prototypenmodell ist ok
- Wichtig ist etappenweises Vorgehen (inkrementell/iterativ)
- Idee für das Treiberskelett: Könnte für Tests hartkodierte Daten zurückgeben
- Idee zum Logging: Man könnte eventuell bei Abstürzen das Problem eingrenzen indem bei jeder Funktion Beginn und Ende geloggt werden

1.3 Nächste Sitzung

Nächster Termin: Sitzung mit Betreuer am Freitag, den 11.11.2005

Zeit: 15:00

Dauer: ca. 30 Minuten erwartet

2 Sitzungsprotokoll, 11. November 2005

2.1 Sitzungsinformationen

Projekt: reiser4xp

Woche: 3

Datum: Freitag, den 11. November 2005

Zeit: 15:10 - 16:15

Teammitglieder / Kürzel

Josias Hosang / jho

Oliver Kadlcek / oka

Christian Oberholzer / cob

Traktanden

- Besprechung Projektplan
- Besprechung Anforderungsspezifikation
- Besprechung Architektur und Design
- Besprechung Programmierrichtlinien
- Besprechung Zeitplan
- Weiteres Vorgehen

2.2 Diskussion / Beschlüsse

Besprechung Projektplan

- Diskussion über den Unterschied zwischen Meilenstein und Prototyp. Im Text muss explizit erwähnt werden, dass Prototypen spezielle Meilensteine sind.
- Baldmöglichst sollte für das Jahresende ein aussagekräftiger Prototyp definiert werden, der eine Kontrolle des Projektfortschritts ermöglicht.

- Die Risikoanalyse könnte um ein oder zwei Risiken erweitert werden. Zieht man „Copy-Paste“-Risiken wie „Ausfall Teammitglied“ ab, sind momentan nur zwei projektspezifische Risiken vorhanden. Möglich wäre z.B. „Ausfall Testsystem“, da in unserem Fall ersatzweise nicht einfach ein normal installierter PC verwendet werde kann. Aktuelle Risiken verursachen tendenziell zu wenig Schaden.
- Allgemeine Diskussion über Kernel-Debugger: Normalerweise kein Support vom Herstellter.
- Diskussion von Testabläufen. Grundsätzlich soll erst die Testumgebung beschrieben werden, danach wird eine Testpszifikation erstellt und zuletzt werden die Tests möglichst automatisiert durchgeführt. Wichtig ist, dass im Fehlerfall nicht alle Teammitglieder blockiert werden und die Entwicklung trotzdem vorangehen kann. Diskussion verschiedener Zeigerfehler.

Besprechung Anforderungsspezifikation

- FA3 „Verzeichnisstruktur lesen“ sollte in „Verzeichnisbaum lesen“ umbenannt werden. Vergleich zwischen FAT-Verzeichnisbaum und Reiser4-Verzeichnisbaum. Diskussion über den Speicherort der Reiser4-Metadaten.

Besprechung Architektur und Design

- So weit in Ordnung.

Besprechung Programmierrichtlinien

- Es ist nicht ersichtlich, wie im Code selber dokumentiert werden soll. „Sinnvoll“ sollte deswegen genauer umschrieben werden, z.B. „mit gesundem Menschenverstand“, „triviales wird nicht kommentiert“ oder „in abhängigkeit der Komplexität“.
- Beim Memory-Manager ist der Bezug zum Architektur-Dokument unklar. Ebenso ist nicht beschrieben, welcher Algorithmus für die Entdeckung von Speicherlöchern verwendet wird und welche Limitationen bestehen. Reicht es beispielsweise aus, den belegten Speicher zu Programmstart und -ende zu vergleichen?

Besprechung Zeitplan

- Einfache Lösung mit Standardprogrammen, entspricht den Anforderungen.

Weiteres Vorgehen

- Start der Implementierung des Treiberrumpfs, nächte Woche werden erste Ergebnisse erwartet.
- Beginn der Analyse von Reiser4, ebenfalls erste Resultate.

2.3 Nächste Sitzung

Nächster Termin: Sitzung mit Betreuer am Freitag, den 18.11.2005

Zeit: 15:00

Dauer: ca. 30 Minuten erwartet

3 Sitzungsprotokoll, 18. November 2005

3.1 Sitzungsinformationen

Projekt: reiser4xp

Woche: 4

Datum: Freitag, den 18. November 2005

Zeit: 15:05 - 16:00

Teammitglieder / Kürzel

Josias Hosang / jho

Oliver Kadlcek / oka

Christian Oberholzer / cob

Traktanden

- Momentaner Stand
- Besprechnung Analyse
- Besprechnung Technischer Bericht

3.2 Diskussion / Beschlüsse

Momentaner Stand

- Ein Punkt ist der angefangene Treiberrumpf. Er ist momentan kompilierbar und installierbar. Die Disk-Information kann gelesen werden, wa eine der beiden wichtigen Funktionen darstellt. Fehlt noch die andere wichtige, das Mounten. An der Erkennung der ReiserPartition wird momentan gearbeitet.
- Als Problem hat sich der Unload des Treibers, zwecks Ausprobieren einer neuer Version gestellt. Es werden dahingehend noch in der nächsten Woche Recherchen angestellt, ansonsten wird halt die etwas mühsamere Lösung mit einem Neustart vor jedem Neuinitialisieren des Treibers ins Auge gefasst. Erwöhnt wurde auche eine Lösung mit einer Virtual Machine, wovon Herr Glatz abrät, wenn direkt vond er Platte gelesen werden will.
- Weitere Tücken scheint der Debugger zu haben. Sowohl über Firewire als auch seriell scheinen einzelne Befehle nicht zu funktionieren (z.B. .kdfiles für direktes laden der neusten Treiberversion auf den Testrechner).
- In der Analyse hat cob den MagicString für die Identifikation der Reiser Partition gefunden und eine Funktion für das Lesen von Nodes.

Besprechung Analyse

- cob möchte mehr Dokumentation über den Linux Source Code. Es wird wohl keine gute geben, ausser der Source Code selber. Herr Glatz wird aber am Montag das Buch „Kernel Architecture“ mitbringen , welches einige Ansätze behandelt.
- In dem Analyse Dokument könnten künftig noch Grafiken verwendet werden

Besprechung Technischer Bericht

- Es kam die Frage auf, was der Technische Bericht eigentlich für einen Zweck und somit Inhalt hat. Aus der Diskussion sind folgende Antworten entstanden. Die Analyse wird ein Teil des Technischen Berichts sein. Die Doxygen Kommentare sollen auch integriert werden und darum auch daurauf ausgerichtet werden, den Code allgemein zu erklären. Ein weiterer Bestandteil sollte der Lösungsentwurf sein, welcher bei uns im Dokument „Architektur und Design“ bereits besteht.
- Der Zweck des Technischen Berichts sollte es sein, dass ein Uneingeweihter, die Gründe sieht, warum etwas auf eine bestimmte Art gelöst wurde. Optimalerweise sollte das in einem Top-Down Ansatz geschehen, das heisst ein Gesamtüberblick und Einführungen/Zusammenfassungen sind wichtige Bestandteile. Die Erstellung des Dokumentes kann aber durchaus iterativ oder in einem Down-Top Ansatz erfolgen. Er sollte eine kleine Einführung in die Treiberentwicklung enthalten und auch Probleme und Stolpersteine in unserer Arbeit darstellen.

Weiteres Vorgehen

- Weiterentwicklung Treiberrumpf.
- Weiterführen Analyse (ev. wurde auf dieser 'Schatzsuche' bereits eine entscheidende Funktion gefunden).

3.3 Nächste Sitzung

Nächster Termin: Sitzung mit Betreuer am Freitag, den 25.11.2005

Zeit: 15:00

Dauer: ca. 30 Minuten erwartet

4 Sitzungsprotokoll, 25. November 2005

4.1 Sitzungsinformationen

Projekt: reiser4xp

Woche: 5

Datum: Freitag, den 25. November 2005

Zeit: 15:05 - 16:00

Teammitglieder / Kürzel

Josias Hosang / jho

Oliver Kadlcek / oka

Christian Oberholzer / cob

Traktanden

- Momentaner Stand
- Besprechnung Analyse
- Besprechnung Technischer Bericht

4.2 Diskussion / Beschlüsse

Momentaner Stand

- Der Treiberrumpf wurde vorgestellt. Die aktuelle Situation sieht so aus, dass der Treiber kompilier- und installierbar ist. Es fehlen allerdings das Logging und der Memory Manager. Damit wurden die gesteckten Ziele nicht ganz erreicht.
- Der aktuelle Stand des technischen Berichtes wurde vorgestellt.
- Der aktuelle Stand der Analyse welche nicht viele weitere Erkenntnisse gebracht hat wurde vorgestellt.

Weiteres Vorgehen

- Den Rumpf fertigstellen, d.h. den Memory Manager und das Logging implementieren.
- Den Umfang des nächsten Release festlegen
- Die Analyse weiterbringen. Es werden mehr Erkenntnisse über ReiserFS benötigt um auch stetig weiterentwickeln zu können.

4.3 Nächste Sitzung

Nächster Termin: Sitzung mit Betreuer am Freitag, den 2.12.2005

Zeit: 15:00

Dauer: ca. 30 Minuten erwartet

5 Sitzungsprotokoll, 2. Dezember 2005

5.1 Sitzungsinformationen

Projekt: reiser4xp

Woche: 6

Datum: Freitag, den 2. Dezember 2005

Zeit: 15:10 - 16:30

Teammitglieder / Kürzel

Josias Hosang / jho

Oliver Kadlcek / oka

Christian Oberholzer / cob

Traktanden

- Momentaner Stand
- Demo MemoryManager / Logger
- Besprechnung des weiteren Vorgehens

5.2 Diskussion / Beschlüsse

Momentaner Stand

- Der Treiberrumpf ist nun samt Logger und MemoryManager fertig implementiert.
- Anforderungen und Termin des neuen Prototyps wurden festgelegt.

Demo MemoryManager / Logger

- Der MemoryManager funktioniert wie erwartet, beim Herunterfahren von Windows werden verschiedene Speicher-Statistiken im Debugger ausgegeben.
- Die Logging-Funktionalität ist implementiert, bringt während der Entwicklung jedoch praktisch keine Vorteile und wird deshalb bis auf weiteres nicht produktiv eingesetzt.

Weiteres Vorgehen

- Analyse wird weitergeführt, das aktuelle Ziel ist das Lesen des Baums von Festplatte.
- Bis am 9.12.2005 sollte das Zurückgeben von VolumeInformations (z.B. Laber, freier Speicher) einigermassen funktionieren.

5.3 Nächste Sitzung

Nächster Termin: Sitzung mit Betreuer am Freitag, den 9.12.2005

Zeit: 11:50

Dauer: ca. 20 Minuten erwartet

6 Sitzungsprotokoll, 9. Dezember 2005

6.1 Sitzungsinformationen

Projekt: reiser4xp

Woche: 7

Datum: Freitag, den 9. Dezember 2005

Zeit: 11:50 - 12:05

Teammitglieder / Kürzel

Josias Hosang / jho

Oliver Kadlcek / oka

Christian Oberholzer / cob

Traktanden

- Momentaner Stand
- Demo Treiber
- Besprechnung des weiteren Vorgehens

6.2 Diskussion / Beschlüsse

Momentaner Stand

- Der Treiber zeigt das Volume Label im Computer Management an (die Grössenangaben und der Filesystemtyp funktionieren noch nicht wie gewünscht). Im Windows Explorer wird das Laufwerk auch noch nicht angezeigt.
- Die Implementation der reiser4-Bibliothek ist ziemlich fortgeschritten. Momentan werden Funktionen für das Auslesen des Baumes, insbesondere des Roots portiert.

Demo Treiber

- Es findet eine kurze Demonstration des momentanen Treibers statt, um die momentanen Funktionalitäten aufzuzeigen.

Weiteres Vorgehen

- Analyse wird weitergeführt, das aktuelle Ziel ist das Lesen des Baums von Festplatte.
- Bis am 16.12.2005 sollte das Anzeigen des Laufwerkes mit Laufwerkbuchstaben im Explorer funktionieren und die Volumeinformationen vollständig gelesen werden.

6.3 Nächste Sitzung

Nächster Termin: Sitzung mit Betreuer am Freitag, den 16.12.2005

Zeit: 15:00

Dauer: ca. 20 Minuten erwartet

7 Sitzungsprotokoll, 16. Dezember 2005

7.1 Sitzungsinformationen

Projekt: reiser4xp

Woche: 8

Datum: Freitag, den 16. Dezember 2005

Zeit: 11:50 - 12:05

Teammitglieder / Kürzel

Josias Hosang / jho

Oliver Kadlcek / oka

Christian Oberholzer / cob

Traktanden

- Momentaner Stand
- Demonstration des Treibers

7.2 Diskussion / Beschlüsse

Momentaner Stand

- Der Treiber mountet die Partition und sie wird im Explorer unter einem Buchstaben angezeigt
- Der Eigenschaftendialog der Partition wird angezeigt.

Demo des Treibers

- Es findet eine kurze Demonstration des momentanen Treibers statt, um die momentanen Funktionalitäten aufzuzeigen.

Weiteres Vorgehen

- Ziel ist es weiter auf das Erreichen des Meilensteins bis Ende Jahr hinzuarbeiten.

7.3 Nächste Sitzung

Nächster Termin: Sitzung mit Betreuer am Freitag, den 23.12.2005

Zeit: 15:00

Dauer: ca. 20 Minuten erwartet

8 Sitzungsprotokoll, 23. Dezember 2005

8.1 Sitzungsinformationen

Projekt: reiser4xp

Woche: 8

Datum: Freitag, den 23. Dezember 2005

Zeit: 11:50 - 12:05

Teammitglieder / Kürzel

Josias Hosang / jho

Oliver Kadlcek / oka

Christian Oberholzer / cob

Traktanden

- Momentaner Stand

8.2 Diskussion / Beschlüsse

Momentaner Stand

- Reiser4-Seitig ist der Lookup von Dateien implementiert.
- Der Windows-Teil funktioniert noch nicht richtig, der Fehler ist momentan noch unbekannt. Daher keine Demo möglich.

Weiteres Vorgehen

- Nach den Ferien soll der PT3 abgeschlossen sein, d.h. Verzeichnis-Auslistungen müssen funktionieren.

8.3 Nächste Sitzung

Nächster Termin: Sitzung mit Betreuer am Freitag, den 13.01.2006

Zeit: 15:00

Dauer: ca. 20 Minuten erwartet

9 Sitzungsprotokoll, 13. Januar 2006

9.1 Sitzungsinformationen

Projekt: reiser4xp

Woche: 11

Datum: Freitag, den 13. Januar 2006

Zeit: 15:00 - 15:50

Teammitglieder / Kürzel

Josias Hosang / jho

Oliver Kadlcek / oka

Christian Oberholzer / cob

Traktanden

- Demo
- Ausblick
- Test
- Dokumente

9.2 Diskussion / Beschlüsse

Demo

- Das Browsen funktioniert beinahe einwandfrei. In einem Verzeichnis mit sehr langen Dateinamen, z.T. mit Sonderzeichen stürzt das System ab, was aber ein bekanntes Problem ist. Allgemein sind die Character Sets ein Problem, auch bei den Tests. Künftig könnte es auch Probleme mit der Case Sensitivity geben.

Ausblick

Es fand eine Duskussion über mögliche Erweiterungen für unser Projekt statt:

- Um Benutzerrechte von Dateien zu unterstützen, müssten erst Linuxbenutzer auf Windowsbenutzer gemappt werden.

- Weiter könnte eine Implementation für die Dateirechtverwaltung vorgenommen werden, was aber nur Sinn macht zusammen mit dem Benutzermapping.
- Eien naheliegende Erweiterung wäre die Schreibunterstützung für reiser4 unter Windows, vom Umfang her würde die aber vermutlich eine ganze Studienarbeit in Anspruch nehmen.
- Die Rolle des Transaktions Logs könnte genauer untersucht werden.
- FastIO wäre eine Möglichkeit die Performance noch zu steigern.

Wir werden diese Erweiterungen in einem Kapitel ausführen und abschätzen, was diese in etwa für einen Aufwand darstellen könnten. Für unsere Studienarbeit sind diese Erweiterungen nicht vorgesehen (ausser optional FastIO), da die Zeit zu knapp ist.

Test

Es wurde angefangen einen Test durchzuführen, um die Verzeichnisstruktur zu vergleichen. Mit der Verwendung von real live Daten tauchten aber massive Probleme auf, und der Test musste zurück gestellt werden.

Wichtig wären Tests sowohl mit real live Daten (um unerwartete Fehler zu finden, wie es tatsächlich auch geschah), als auch mit künstlichen Daten, um Randbedingungen zu testen.

Dokumente

Cob zeigt Herrn Glatz den Fortschritt des Analyse Dokumentes, welches schon sehr weit entwickelt ist.

Der Projektplan wurde auf Grund des späteren Abgabedatums angepasst. Wichtig ist, dass die Verschiebungen dokumentiert werden.

9.3 Nächste Sitzung

Herr Glatz fragt an, ob wir die letzten Sitzungen jeweils um Donnerstag um 15 Uhr durchführen könnten. Für uns stellt das kein Problem dar.

Nächster Termin: Sitzung mit Betreuer am Donnerstag, den 19.01.2006

Zeit: 15:00

Dauer: ca. 30 Minuten erwartet

10 Sitzungsprotokoll, 19. Januar 2006

10.1 Sitzungsinformationen

Projekt: reiser4xp

Woche: 12

Datum: Donnerstag, den 19. Januar 2006

Zeit: 15:00 - 15:15

Teammitglieder / Kürzel

Josias Hosang / jho

Oliver Kadlcek / oka

Christian Oberholzer / cob

Traktanden

- Stand

10.2 Diskussion / Beschlüsse

Stand

Es fand eine kurze Diskussion über den aktuellen Projektstand statt:

- Der Fehler mit den langen Dateinamen ist behoben worden.
- Bei Sonderzeichen funktioniert das Testprogramm nicht richtig. Die Darstellung im Windows Explorer funktioniert jedoch korrekt. Daher können die Testdaten behalten werden. Das Fehlverhalten des Testprogrammes muss allerdings berücksichtigt werden.
- Die Dokumentation ist weitergeschrieben worden.
- Die Arbeiten an der Funktionalität zum Lesen von Dateien wurden weitergeführt.

10.3 Nächste Sitzung

Nächster Termin: Sitzung mit Betreuer am Donnerstag, den 26.01.2006

Zeit: 15:00

Dauer: ca. 30 Minuten erwartet

11 Sitzungsprotokoll, 26. Januar 2006

11.1 Sitzungsinformationen

Projekt: reiser4xp

Woche: 13

Datum: Donnerstag, den 26. Januar 2006

Zeit: 15:05 - 15:30

Teammitglieder / Kürzel

Josias Hosang / jho

Oliver Kadlcek / oka

Christian Oberholzer / cob

Traktanden

- Aktueller Stand
- Weiteres Vorgehen
- Dokumente

11.2 Diskussion / Beschlüsse

Aktueller Stand

- Implementation der Lesefunktionalität schreitet voran.
- Seit dem letzten Mal wurden einige Memory-Leaks behoben

Weiteres Vorgehen

- Es wurde beschlossen, den Recognizer nicht zu implementieren, da er keinen echten Nutzen bringt: Wenn jemand den Treiber installiert ist davon auszugehen, dass er auch eine Reiser4-Partition besitzt. Daher kann der Treiber immer vollständig geladen werden.
- Da USB-Festplatten eine spezielle Behandlung beim Unmount erfordern, wird aus Zeitgründen auf eine korrekte Implementation verzichtet.
- Allgemein soll lieber mehr Gewicht auf einen stabielen Treiber gelegt werden als auf viele Features.
- Bis zum nächsten Mal sollte das Lesen funktionieren.

Dokumentation

- Die Strukturierung der Dokumentation wurde diskutiert. Sie sollte „logisch“ und „vernünftig“ sein. Der Informationsgehalt soll möglichst hoch sein, Dinge wie eine Änderungsgeschichte sind uninteressant.
- Besonders Design-Entscheidungen müssen gut dokumentiert werden.
- Die Dokumentation sollte für jemanden, der die Arbeit weiterführen will verständlich sein.
- Mit Doxygen generierte Code-Dokumentation sollte nicht ausgedruckt werden und muss auch nicht ungedingt auf die CD.

11.3 Nächste Sitzung

Nächster Termin: Sitzung mit Betreuer am Donnerstag, den 02.02.2006

Zeit: 15:00

Dauer: ca. 30 Minuten erwartet

12 Sitzungsprotokoll, 2. Februar 2006

12.1 Sitzungsinformationen

Projekt: reiser4xp

Woche: 14

Datum: Donnerstag, den 2. Feb 2006

Zeit: 15:10 - 15:40

Teammitglieder / Kürzel

Josias Hosang / jho

Oliver Kadlcek / oka

Christian Oberholzer / cob

Traktanden

- Demo
- Stand

12.2 Diskussion / Beschlüsse

Demo

- Als Demo wurde ein MP3 von der Reiser4 Partition abgespielt, was das Funktionieren des Lesens demonstrieren soll (sogar einigermassen performant).
- Es werden auch mehrere Partitionen (eine zusätzliche USB Festplatte) erkannt. Allerdings kann die USB Platte nicht getrennt werden und falls sie einfach ausgesteckt und wieder eingesteckt wird und vorher Dateien geöffnet wurden, entsteht ein BlueScreen.
- Das Zuordnen von Laufwerksbuchstaben an von Reiser4 Partitionen mit Hilfe des DriveLetterUtil wird demonstriert.

Stand

- Symbolic Links funktionieren nicht. Das liegt daran, das die Zielpfade absolut (von der Linux Root-Partition ausgehend) sind und unter Windows deshalb nicht aufgelöst werden können. In reiser4xp werden Symbolic Links aus aus diesem Grund nicht angezeigt.
- Hardlinks funktionieren
- Das Header Dateien Problem ist gelöst (Ein-Buchstaben Dateiendungen)
- Lesen (auch gecached) funktioniert mehrheitlich (noch nicht getestet)
- FastIO enthält viele kleine Funktionen, was aufwändig zu implementieren ist, ev. bleibt noch Zeit.

12.3 Nächste Sitzung

In der letzten Woche sollte alles fertig gestellt werden und keine offizielle Sitzung mehr stattfinden. Es wird noch einmal ein ausführliche Demo mit unserem Betreuer abgehalten.

Nächster Termin: Demo mit Betreuer am Donnerstag, den 09.02.2006

Zeit: 15:00

Dauer: ca. 30 Minuten erwartet

F Projektplan und Zeiterfassung