



FSCK – File System Check

Marco Bindner

Seminar Dateisysteme

Sämtliche Informationen über den Ablauf des Filesystemchecks beziehen sich auf das Ext2 Dateisystem.



Gliederung

- Allgemeines
- Aufbau
- Zusammenfassung

Allgemeines

- FSK - „File system consistency check“
- Software zur Prüfung verschiedener Dateisysteme unter UNIX/LINUX
- Basiert auf fsck für das MINIX Dateisystem
- Komplet neu geschrieben

Die Ursprungsversion von fsck bzw e2fsck basiert auf dem von Linus Torvald geschriebenen fsck für das MINIX Dateisystem und wurde aus performance Gründen komplett neu geschrieben.

Allgemeines

- Optimierung auf möglichst kurze Laufzeit
- Optimierung auf wenig Plattenzugriffe
- Optimierung auf wenig Suchoperationen
- Ideen von E. Bina und P. Emrath

Um möglichst wenig Suchoperationen, Plattenzugriffe und somit auch eine kurze Laufzeit zu erreichen wurden ca 85% der E/A-Operationen auf die ersten beiden Durchlaufphasen gelegt.

Die dabei zugrundeliegenden Ideen stammen von E. Binath und P. Emrath



Gliederung

- Allgemeines
- Aufbau
- Zusammenfassung

Aufbau

e2fsck

- Phase 1: Überprüfe Inodes
- Phase 2: Sammle Verzeichnisse
- Phase 3: Überprüfe Verzeichnissbaum
- Phase 4: Überprüfe Reference counts
- Phase 5: Überprüfe Bitmaps



Phase 1

- Testet Inodes als nicht verbundene Objekte
- Sammelt sehr viele Informationen um später Plattenzugriffe bei konsistentem FS zu minimieren
- E/A lastig
- Längste Phase



Phase 1 - Tests

- Gültigkeit des mode Felds jedes Inodes
- Korrektheit von Größe und Blockzahl der Inodes
- Eindeutige Zuweisung eines Datenblocks zu einem einzigen Inode
 - -> Phase 1b bis 1d werden durchgeführt

Phase 1 b) bis 1 d) werden nur durchlaufen wenn Verweis zweier Inodes auf den gleichen Inode festgestellt wurden.

Phase 1 – Informationen

Gesammelt:

- inode_used_map
- inode_dir_map
- inode_reg_map
- inode_bad_map
- inode_bb_map
- inode_imagic_map
- block_found_map
- block_dup_map
- dir_map

Informationen werden in sämtlichen späteren Phasen benötigt.

Phase 1 b) - d)

- Phase 1 b)
 - Datenblöcke der Inodes werden erneut gescannt um eine Liste der Duplikate zu erstellen
- Phase 1 c)
 - Baumtraversierung wird durchgeführt um Pfadnamen der betroffenen Inodes auszugeben
- Phase 1 d)
 - Für jeden von mehreren Inodes benutzten Datenblock wird wahlweise der Datenblock dupliziert oder der Inode gelöscht



Phase 2

- Testet Verzeichnisse als unverbundene Objekte
 - Möglich da Verzeichniseinträge keine Blöcke übergreifen
- Sortiert Blöcke in aufsteigender Folge nach Nummer
 - Minimiert Zeit für Suche auf der Platte

Phase 2 - Tests

- Länge des Verzeichniseintrags (`rec_len`) liegt zwischen 8 Byte und dem verbleibenden Platz im Verzeichnisblock
- Die Länge des Namens des Verzeichniseintrags (`name_len`) beträgt weniger als (`rec_len - 8`)
- Inode Nummer im Verzeichnis liegt in erlaubten Grenzen

Unzulässige Angaben werden entsprechend korrigiert.



Phase 2 - Tests

- Die Inode Nummer verweist auf einen benutzten Inode
- Erster Eintrag ist “.” und sein Inode verweist auf den Inode des Verzeichnisses
- Der zweite Eintrag ist “..”

Der Verweis des “..” Eintrags wird erst in Phase 3 geprüft.

Phase 2 - Informationen

Gesammelt:

- Inode Nummern der Verzeichnisse

Benutzt:

- Verzeichnissinformationen aus Phase 1
- inode_used_map
- inode_bad_map
- inode_dir_map

Informationen werden in späteren Phasen benötigt.



Phase 3

- Test des Verzeichnisbaums
- Jedes Verzeichnis das nicht auf Root zurückgeführt werden kann wird unter /lost+found eingehängt
- Kaum E/A Operationen
- CPU lastig

Phase 3 - Algorithmus

- Prüfe ob ROOT existiert. Wenn nein erstelle neu. Markiere das ROOT Verzeichnis „done“.
- Iteriere über alle Verzeichnis Inode. Verfolge den Verzeichnisbaum mit Hilfe von dirinfo.parent aufwärts bis er ein als „done markiertes Verzeichnis erreicht. Markiere es als „done“. Wenn das nicht möglich ist hänge es unter /lost+found ein.
- Wird dabei ein Verzeichnis zweimal durchlaufen ist ein Loop entdeckt. Hänge das Verzeichnis unter /lost+found ein.



Phase 4

- Vergleicht link counts (cached in Phase 1) mit den internen counter aus Phase 2/3
- Alle nicht gelöschten Dateien mit link_count=0 werden an /lost+found angehängt
- Kaum E/A Operationen
- CPU lastig



Phase 5

- Vergleicht die Block-Bitmaps und Inode-Bitmaps des FS mit den in Phase 1-4 erstellten Bitmaps
- Bei Inkonsistenz wird der entsprechende Bitmap eintrag des FS überschrieben



Gliederung

- Allgemeines
- Ablauf
- Zusammenfassung



Zusammenfassung



- Kann viele aber nicht alle Fehler korrigieren
- FSCK bei Ext2 sehr zeitaufwendig
- Methoden zur Verringerung des Aufwandes sollten angewandt werden (z.B. Journaling)



Literatur

- <http://e2fsprogs.sourceforge.net/>
Quellcode e2fsck
- <http://e2fsprogs.sourceforge.net/ext2intro.html>
Ext2 Informationen
- <http://www.tldp.org/LDP/khg/HyperNews/get/fs/ext2intro.html>
Ext2 Informationen mit Bezug auf fsck
- Daniel P. Bovet & Marco Cesati
Understanding the Linux Kernel, 3rd Edition
- Uresh Vahalia: UNIX Internals - The New Frontiers