Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Labor „Betriebssysteme Übung“ Wintersemester 2017/18

- Aufgabenteil 2 -

Oliver P. Waldhorst

Zielsetzung

Zielsetzung des Labors

- Ergänzung / Vertiefung der Inhalte der Vorlesung „Betriebssysteme“
  - Insbesondere Funktionsweise von Dateisystemen und deren Verwendung in Linux (UNIX)
- Vertiefung des (Betriebs-)systemnahen Programmierens unter C++
- Entwickeln von Software im Team

Organisatorisches

Umfang
- 3 ECTS / 2 SWS (entspricht einem Arbeitsaufwand pro Person von 90h!)
- Gruppenarbeit von 3-4 Studierenden

Veranstaltungen
- Jeweils mittwochs von 14.00 (11.30) – 18.30 (17.00) in LI137
- **Ausnahme: Mittwoch, 20.12.2017: 11:30 – 16:00**

Zeitplan
- Teil 1: 5 Termine (08.11.17 - 06.12.17)
- Teil 2: 5 Termine (13.12.17 - 24.01.18)
- **Letzte Möglichkeit zur Abgabe ist Mittwoch, 24.01.18**

Bewertung
- Unbenoteter Schein
- Bei Erledigung in diesem Semester: Eine Notenstufe Bonus in Betriebssysteme-Klausur

Die Aufgabenstellung

Aufgabe 1: Read-Only File System

- *Eine Containerdatei wird mittels eines Kommandos `mkfs.myfs` erstellt*
- *Beim Erstellen werden ausgewählte Dateien in die Containerdatei kopiert (einmalig)*
- *Wenn eine Containerdatei mittels FUSE in Verzeichnisbaum eingebunden wird, können enthaltene Dateien gelesen, aber (noch) nicht verändert oder gelöscht werden*

Aufgabe 2: Read-Write File System

- `mkfs.myfs` erstellt leere Containerdateien mit fester Größe (optional kann Kopieren beibehalten werden)
- Wenn eine Containerdatei mittels FUSE in Verzeichnisbaum eingebunden wird, können enthaltene Dateien gelesen, verändert und gelöscht werden, neue Dateien können eingefügt werden

Aufgabe 3: Dokumentation

Aufgabe 2: Read-Write File System

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Teilaufgabe 2a: Anpassen des Aufbaus von MyFS-Containerdateien

Teilaufgabe 2b: Implementierung der Operationen zum Anlegen, Schreiben, Ändern, Löschen von Dateien

Aufgabe 2:
Was soll (mindestens) funktionieren?

Anlegen einer leeren Datei:

```
touch neuedatei1.txt
```

→ Rechte beachten

Anlegen und Schreiben einer Datei:

```
echo Hello World > neuedatei2.txt
```

Überschreiben einer Datei:

```
echo Hello World > neuedatei1.txt
echo Hello World 2 > neuedatei2.txt
```

Anhängen an eine Datei:

```
echo Hello World 2 >> neuedatei1.txt
```

→ Hello World Hello World 2
Hello World 2

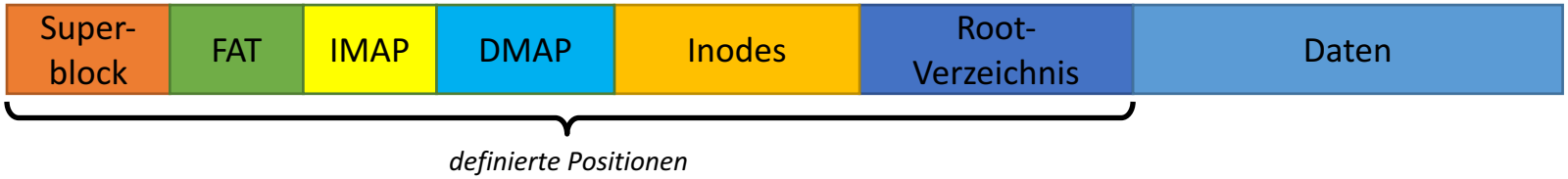
Löschen einer Datei:

```
rm neuedatei2.txt
```

2a: Aufbau von MyFS-Containerdateien

Die MyFS-Containerdatei soll Strukturen zur Verwaltung von freien Inodes und Datenblöcken enthalten (falls noch nicht vorhanden)

- Z.B. mit IMAP bzw. DMAP:



Hinweis: Sorgen Sie dafür, dass in der Container-Datei genügend Platz für Testdaten ist (30MB sollten ausreichen...)

2b: Implementierung der Operationen

Folgende Operationen in der Klasse `MyFS` müssen zusätzlich zu Aufgabe 1 (mindestens) implementiert werden:

- Zum Anlegen einer Datei
  `MyFS::fuseMknod()`
  (ggf. `MyFS::fuseCreate()`)
- Zum (Über-)Schreiben einer Datei
  `MyFS::write()`
- Zum Löschen einer Datei
  `MyFS::unlink()`

Hinweise zu den FUSE Operationen (1)

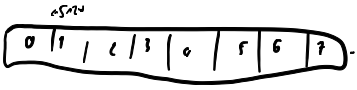
```
int MyFS::fuseMknod(const char *path,
      mode_t mode, dev_t dev)
```

- Erzeugt eine neue Datei `path`
- Wird nur aufgerufen, wenn `MyFS::fuseGetattr()` für `path` zuvor `-ENOENT` zurückgeliefert hat!
- Sinnvolle Fehlercodes
  - `EEXIST` – Datei existiert bereits
  - `ENOSPC` – Kein freier Platz im Dateisystem

Hinweise zu den FUSE Operationen (2)

```
int MyFS::fuseWrite(const char *path,
      const char *buf, size_t size, off_t offset,
      struct fuse_file_info *fileInfo)
```

- Schreibt `size` Bytes aus `buf` ab Position `offset` in die durch `path` oder `N`gegebene Datei
  - Datei wird vorher geöffnet, daher bietet sich die Verwendung von `fileInfo` an
- Rückgabewert ist die Anzahl der geschriebenen Bytes
- Bitte auch hier Puffern (vgl. `MyFS::fuseRead`)!
- Sinnvolle Fehlercodes
  - `EBADF` – Datei nicht zum Schreiben geöffnet
  - `ENOSPC` – Kein freier Platz in Container-Datei



read: offset: 518
size: 10

total: offset: 518
size: 16

} beide nun Zugriff auf Block 1 gewährt

Hinweise zu den FUSE Operationen (3)

```
int MyFS::unlink(const char *path)
```

- Löscht die Datei `path`
- Verzeichniseitrag wird entfernt, Inode und Datenblocks freigegeben
- Sinnvolle Fehlercodes
  - `ENOENT` – Datei existiert nicht

Bewertung

Teil 1 + 2

- Testfälle, Erklärung des Codes

Teil 3

- Vollständigkeit und Verständlichkeit der Dokumentation
  - Aufgabenstellung (in eigenen Worten)
  - Lösungsansatz und Umsetzung
  - Programmausführung und Testfälle
  - …

10 Seiten

Fragen?

Literatur

[1] R. Arpaci-Dusseau, A. Arpaci-Dusseau, Operating Systems: Three Easy Pieces, (V. 0.90). Arpaci-Dusseau Books, 2015. http://pages.cs.wisc.edu/~remzi/OSTEP/ (Kapitel 39 und 40).

[2] R. Stevens, S. Rago, Advanced Programming the UNIX Environment (3rd Edition). Addison Wesley, 2013. (Kapitel 3 und 4)

[3] X. Pretzer, Building File Systems with FUSE. https://stuff.mit.edu/iap/2009/fuse/fuse.ppt (abgerufen 06.11.2017)

[4] M. Q. Hussain, Writing a Simple Filesystem Using FUSE in C. http://www.maastaar.net/fuse/linux/filesystem/c/2016/05/21/writing-a-simple-filesystem-using-fuse/ (abgerufen 06.11.2017)

[5] J. Pfeiffer, Writing a FUSE Filesystem: a Tutorial. https://www.cs.nmsu.edu/~pfeiffer/fuse-tutorial/ (abgerufen 12.10.2017)

[6] fuse-examplefs. https://code.google.com/archive/p/fuse-examplefs/ (abgerufen 06.10.2017)

[7] libfuse API documentation. https://libfuse.github.io/doxygen/index.html (abgerufen 06.11.2017)