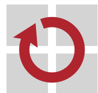
Praktikum angewandte Systemsoftwaretechnik (PASST)

Dateisysteme / Aufgabe 5

13. Januar 2021

Dustin Nguyen, Tobias Langer, Jonas Rabenstein, Phillip Raffeck

Lehrstuhl für Informatik 4
Friedrich-Alexander-Universität Erlangen-Nürnberg


Lehrstuhl für Verteilte Systeme
und Betriebssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Rückblick

Rückblick

- Linux Kernelmodule
  - Dynamisch lad-/entladbar werden
- **U**niversal **S**erial **B**us
  - Kein wirklicher Bus
  - Verschiedene Transferarten
  - Kommunikation über Endpoints und Pipes unter Linux
- `sysfs`
  - FileIO-basierte Schicht zwischen Kernel-/Userspace
  - Stellt Zugriff auf Kernelobjekte (`kobjects`) bereit

Motivation

Leaving the beaten track

- Dateisystem ist zentraler Teil eines Betriebssystems
  - Dateiein- und ausgabe
  - Schnittstelle zum Linux-Kernel
- Entwicklung betrifft dabei viele Komponenten
  - Verwaltung von Dateisysteminstanzen, Inodes, …
  - Organisation von (Meta-)Daten
  - Adressraummappings
- Allerdings…
  - Kaum/wenige verlässliche Dokumentation
  - Vieles muss anhand vorhandenem Code abgeleitet werden

Lernziele

Lernziele

Im Anschluss an diese Aufgabe solltet Ihr...

- die Kernkomponenten eines Dateisystems aufzählen
- die Funktionsweise des **VFS** erklären
- eigene Dateisystemtreiber programmieren

...können.

Agenda

Agenda

Dateisysteme

Virtual Filesystem

Interaktion mit dem Speichermedium

Zusammenfassung

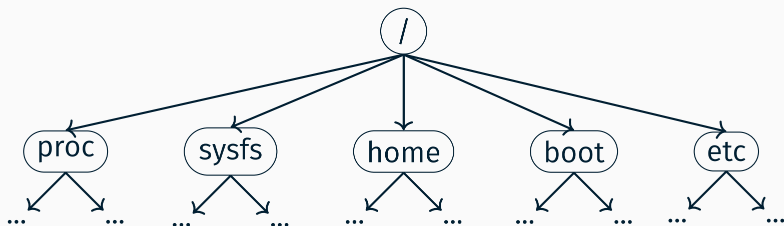
Aufgabe 5

Dateisysteme

Von Dateisystemen

- Dateisystem
  - Organisation von Daten auf Datenträger
  - Zuordnung von (Meta-)Daten zu Ablageort
  - Beispiele: ext{2,3,4}, ntfs, ...
- Dateisystemtreiber
  - Setzt Zugriff auf Dateisystem um
  - Implementiert `open()`, `read()`, `write()`, ...
- Zusätzlich: Pseudo-Dateisysteme: procfs, sysfs
  - Unix-Philosophie: „Everything is a file"
  - Exportieren Kernelstrukturen als Dateischnittstelle

Der Dateibaum unter Linux

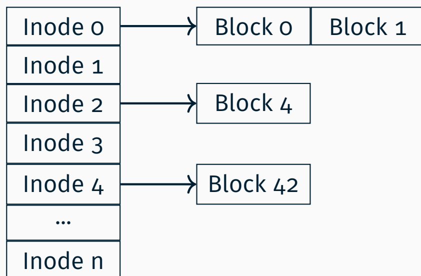


- Dateien sind als Baum organisiert
- Einträge können eigene „Mountpoints“ sein
  - `boot/` als eigenes Dateisystem
  - `proc/` als Pseudo-Dateisystem
  - Dateisystem-Operationen können dort eine vollständig andere Semantik haben

Bestandteile eines einfachen Dateisystems

Prinzipieller Aufbau eines Unix-Dateisystems:

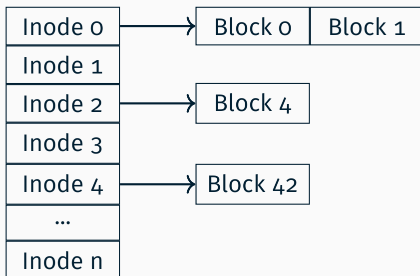
- Inodes
  - Metadaten der Dateien
  - Verweise auf Dateiblöcke
- Blöcke
  - Nutzdaten
  - Verzeichnisinhalte



Bestandteile eines einfachen Dateisystems

Prinzipieller Aufbau eines Unix-Dateisystems:

- Inodes
  - Metadaten der Dateien
  - Verweise auf Dateiblöcke
- Blöcke
  - Nutzdaten
  - Verzeichnisinhalte



Blockgröße $\neq$ Blockgröße

- Blockgröße wird durch Dateisystem vorgegeben
- Entspricht nicht phys. Blockgröße des Datenträgers

Inodes - Daten über Daten...

Index Node (Inode):

Number	Type	Size	Owner	Ref. Counter	...	Pointer to Blocks

Mögliche Verwaltungsdaten:

- Eindeutige ID
- Typ & Größe der Datei
- Besitzer & Zugriffsrechte
- Verweis auf die Nutzdaten
- Verweiszähler
- ...

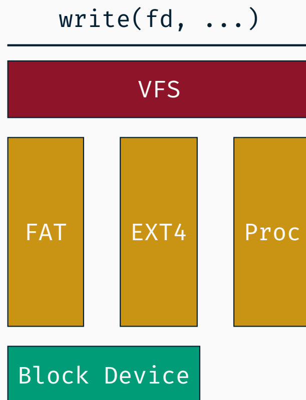
Linux-Kernel Inode:

```
struct inode {
    umode_t                  i_mode;
    kuid_t                   i_uid;
    kgid_t                   i_gid;
    struct super_block*      i_sb;
    unsigned long            i_ino;
    loff_t                   i_size;
    // ...
};
```

Virtual Filesystem

Das Linux Virtual Filesystem (VFS)

- Abstraktion für Dateisysteme
  - Generische Schnittstelle
  - Implementiert `write()`, `open()`, ...
- Komponenten eines Dateisystems
  - Dateisystemtyp
  - Superblock
  - Inode
  - Datei



Treiberprogrammiermodell im Kernel

Ähnlichkeiten zur **Objektorientierung**:

Treiberprogrammiermodell im Kernel

Ähnlichkeiten zur **Objektorientierung**:

- *Vererbung* durch Schachtelung von Strukturen:

```
struct passtfs_inode_info { // "Derived Class"
    int i_sblock;
    int i_eblock;
    struct inode i_inode;   // "Base Class"
};
```

Treiberprogrammiermodell im Kernel

Ähnlichkeiten zur **Objektorientierung**:

- *Vererbung* durch Schachtelung von Strukturen:

```
struct passtfs_inode_info { // "Derived Class"
    int i_sblock;
    int i_eblock;
    struct inode i_inode;   // "Base Class"
};
```

- *Konstruktoren* durch Initialisierer-Funktionen

Treiberprogrammiermodell im Kernel

Ähnlichkeiten zur **Objektorientierung**:

- *Vererbung* durch Schachtelung von Strukturen:

```
struct passtfs_inode_info { // "Derived Class"
    int i_sblock;
    int i_eblock;
    struct inode i_inode;   // "Base Class"
};
```

- *Konstruktoren* durch Initialisierer-Funktionen

- *Funktionspezialisierung* durch Callbacks[1]:

```
const struct file_operations passtfs_dir_file_ops = {
    .read           = generic_read_dir,
    .iterate_shared = passtfs_iterate_shared,
};
```

[1] Linux stellt teilweise generische Implementierung bereit

Der Dateisystemtyp im VFS

Dateisystem

Superblock	Inode Freilist	Block Freilist	Inodes	Blöcke

Dateisystemtyp

- Repräsentiert durch `struct file_system_type`
- Erzeugt Dateisysteminstanzen

Dafür benötigt der Treiber ...

- Attribute des Dateisystems (Namen, Statusflags, ...)
- `fill_super`-Callback

Der Superblock im VFS

Dateisystem

`Superblock`	`Inode Freilist`	`Block Freilist`	`Inodes`	`Blöcke`

Superblock - Instanz des Dateisystems

- Repräsentiert durch `struct super_block`
- Generische Kernelschnittstelle

Dafür benötigt der Treiber ...

- `struct super_operations`
- Implementierung der unterstützten Callbacks

Ergebnis im Userspace

- Mounten eines Dateisystems

Inodes im VFS

Dateisystem

Inode

- Repräsentiert durch `struct inode`
- Generische Kernelschnittstelle

Dafür benötigt der Treiber …

- `struct inode_operations`
- Implementierung der unterstützten Callbacks

Ergebnis im Userspace

- open, `link`, `rename`, … Syscalls funktionieren

Dateien im VFS

Dateisystem

Superblock	Inode Freilist	Block Freilist	Inodes	Blöcke

Geöffnete Dateien

- Repräsentiert durch `struct file`

Dafür benötigt der Treiber ...

- `struct file_operation`
- Implementierung der unterstützten Callbacks

Ergebnis im Userspace

- Dateien können gelesen & geschrieben werden
- Verzeichnisinhalte können gelesen werden

Dentries

Dentry (Directory entry):

- Repräsentiert einen Verzeichniseintrag
- Verweis auf Inode und Eltern-Dentry
- Werden für schnelleren Zugriff zwischengespeichert
- Werden nicht auf der Festplatte gespeichert

Interaktion mit dem Speichermedium

Dateisystemtreiber

Die Aufgaben eines Dateisystemtreibers bestehen aus:

- Verwaltung von **Metadaten**
  - Einheit: Speicherblöcke
  - Interface: `sb_bread`
- Verwaltung von **Dateiinhalten**
  - Einheit: Dateiausschnitte
  - Interface: `address_space_operations`

Buffer Heads

- Kernel abstrahiert Zugriff auf Medium:

```
struct buffer_head *sb_bread(struct super_block *sb,
sector_t block)
```

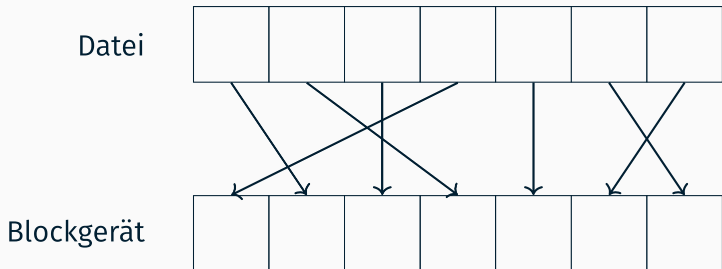
  - Speichert *logischen* Block im sog. Buffer Head
  - Akzeptiert *logische* Blockadresse
  - Speicht assozierte Daten zwischen

- Ermöglicht Zugriff mit Blockgranularität

  → z.B. Lesen des Superblock

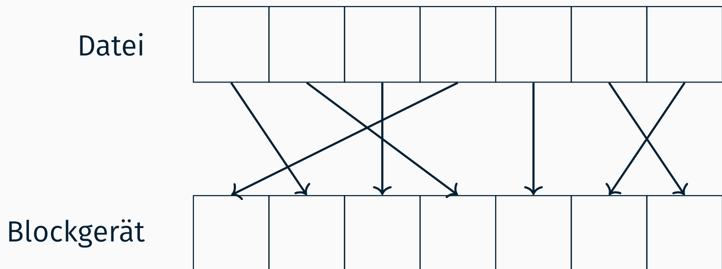
Adressraumoperationen (1/2)

- **VFS** kapselt Zugriff auf Dateiinhalte
- Dateioperationen interagiert indirekt mit Speichermedium



Adressraumoperationen (1/2)

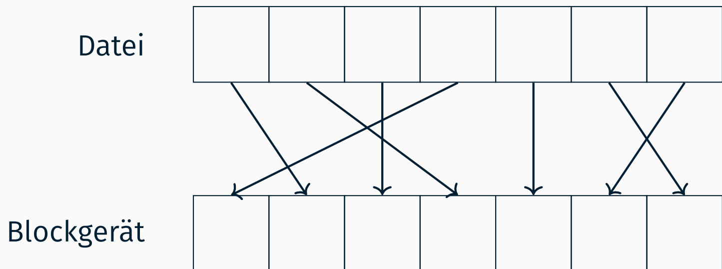
- **VFS** kapselt Zugriff auf Dateiinhalte
- Dateioperationen interagiert indirekt mit Speichermedium



- **Frage:** Woher kennt VFS Dateisystemlayout?

Adressraumoperationen (1/2)

- **VFS** kapselt Zugriff auf Dateiinhalte
- Dateioperationen interagiert indirekt mit Speichermedium



- **Frage:** Woher kennt VFS Dateisystemlayout?
- **Antwort:** Abbildungsfunktion von Dateisystemtreiber

Adressraumoperationen (2/2)

- Dateisystemtreiber implementiert `get_block` Funktion:

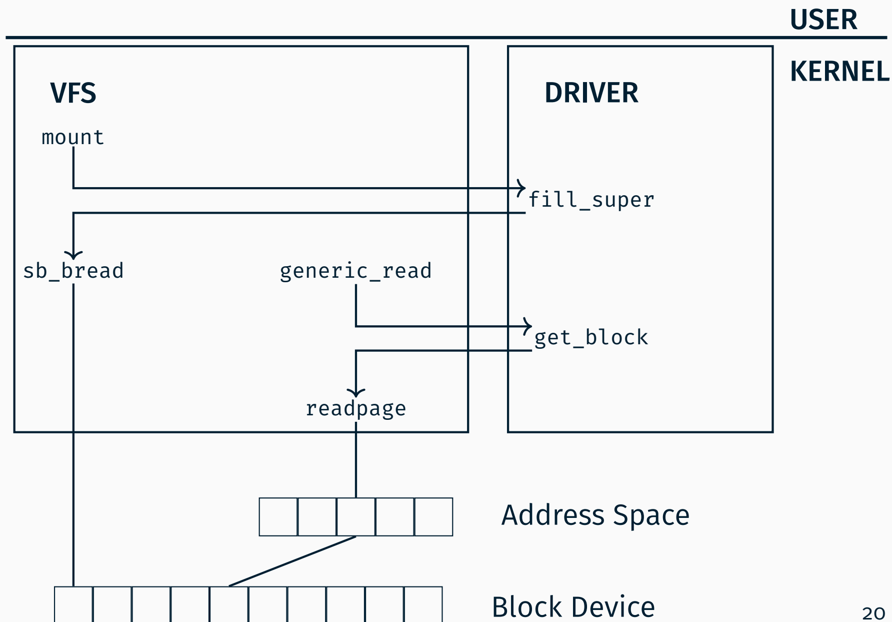
  ```
  int get_block(struct inode *inode, sector_t iblock,
              struct buffer_head *bh_result, ...)
  ```

  - `inode`: Assoziierte Datei
  - `iblock`: Dateioffset (in *logischen* Blöcken)
  - `bh_result`: Ergebnisbuffer

- Generische Adressraumoperationen erhalten `get_block` Funktion als Parameter

  ```
  int writepage(struct page *page, struct writeback_control *wbc)
  {
      return block_write_full_page(page, get_block, wbc);
  }
  ```

Übersicht



Schrittweises Vorgehen

✓ `TODO-LIST:`

- ☐ Kernelmodul
- ☐ Mount
- ☐ Anzeigen von Ordnerinhalten
- ☐ Lesen von Dateien
- ☐ Schreiben von Dateien

Zusammenfassung

Zusammenfassung

- Dateisysteme
  - Speicherung und Struktur von Daten auf Datenträger
  - Ablage als Metadaten, Inodes & Datenblöcke
  - Außerdem: Pseudodateisysteme
- Virtual File System
  - Einheitliche Dateisystemschnittstelle des Linux-Kernels
  - Stellt generische Basisoperationen bereit
  - Erweitert durch konkrete Dateisystemtreiber

Aufgabe 5

Aufgabe

- Einarbeiten in die benötigten APIs im Linux-Kernel
  - Dokumentation, Codebeispiele
  - Empfohlenes Dateisystem zum Verständnis: bfs, minix
- Lesen & Verstehen der Spezifikation
- Programmieren des Dateisystems
- Testen des bereitgestellten Images

Abgabe: Bis 08. Februar 2021 durch Vorführung in einer Rechnerübung

Ein einfaches Dateisystem (1/3)

Dateisystem

Superblock	Inode Freilist	Block Freilist	Inodes	Blöcke

- Feste Zahl an Inodes & Blöcken
- Block- & Inodefreilisten als Bitmaps
- Superblock enthält statische Informationen
- Dateien werden als Blockintervall gespeichert

Superblock

Magic Number	Blockgröße	Anzahl Inodes	Anzahl Blöcke

- Magic Number: `0x53462d5453534150`
- Blockgröße (idR. 512 Byte)
- Absolute Offsets der Dateisystemsegmente (in Blöcken)

Frage

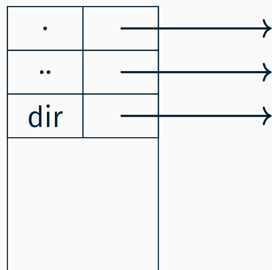
- Welche Vor-/Nachteile hat das Layout?

Ein einfaches Dateisystem (3/3)

Verzeichniseintrag:

```
struct passtfs_raw_diren {
    __le32 inode_number;
    char name[PASSTFS_NAMELEN];
};
```

Verzeichnisse:



Verzeichnisse

Verzeichnisse können als Arrays von Namen/Inodenummern Einträgen interpretiert werden.

Fragen?