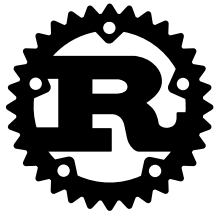
KernelModule  ust>

Clara Granzow, Clemens Tiedt

Projektverlauf

Anfang

- Grobe Projektidee: Ein Kernelmodul in Rust schreiben
- Erste konkrete Idee: `Ext2` nachbauen
  - Problem: Deutlich zu groß für ein Semester Arbeit

Aktuell

- `ramfs` in Rust implementieren

Zukunft

- Bestehende Probleme im `ramfs`-Code lösen
- Komplexeres FS implementieren

Wie schreibt man ein Rust-Kernelmodul?

- **Schritt 1:** Start-Code in C schreiben, der Rust-Funktionen per FFI aufruft
- **Schritt 2:** Statische Rust-Library schreiben und via `kbuild` linken
  - Rust-Funktionen mit `pub extern "C"`-Deklaration nach außen sichtbar machen
  - Herausforderung: Keine Rust-Standardbibliothek im Kernel nutzbar

- Bei Portierung von vorhandenem C-Code: Noch nicht portierte Funktionen einfach via FFI einbinden

Bindings für Kernel-Strukturen

Variante	Wenig manueller Aufwand	Type Safety	Übersetzung von inline Funktionen und Makros
Bindings selbst schreiben	✗	✓	✗
Opaque Typen verwenden	✓	✗	✗
Bindgen nutzen	✓	✓	✗

- Unsere Wahl: bindgen
  - Nicht perfekt, aber beste Trade-Offs
  - Eigene C-Wrapper für inline-Funktionen
  - Bestehendes Problem: bindgen für Kernel-Header funktioniert nicht auf Kernel 5.x

    → Da sich die genutzten Interfaces nicht geändert haben, Bindings auf Kernel 4.x generieren

“Safety” in Rust

- Rust-Code ist standardmäßig sicher in Bezug auf Speicher
  - Kein “Use after free”, dangling pointers etc.
- C-Code per FFI ist standardmäßig unsafe
  - Rust-Compiler kann nicht prüfen, ob Speicher ordnungsgemäß genutzt wird
- Lösung: Einzelnen, unsafen Code, dessen korrekte Funktionsweise garantiert werden kann, in safe Rust-Wrapper einbinden

Beispiel: ramfs_mknod

```
static int
ramfs_mknod(struct inode *dir, struct dentry *dentry, umode_t mode, dev_t dev)
{
        struct inode *inode = ramfs_get_inode(dir->i_sb, dir, mode, dev);
        int error = -ENOSPC;

        if (inode)
        {
                d_instantiate(dentry, inode);
                dget(dentry);
                error = 0;
                dir->i_mtime = dir->i_ctime = current_time(dir);
        }
        return error;
}
```

Interner Zustand für Fehler

Expliziter Null-Pointer-Check

Repräsentation eines Fehlers durch Integer

Beispiel: ramfs_mknod

```
#[no_mangle]
pub extern "C" fn ramfs_mknod(
        dir: *mut inode,
        dentry: *mut dentry,
        mode: umode_t,
        dev: dev_t,
) -> cty::c_int {
        use bindings::{current_time, d_instantiate, ENOSPC};
        let inode = unsafe { ramfs_get_inode((*dir).i_sb, dir, mode, dev) };
        let mut error = -(ENOSPC as i32);

        if inode != core::ptr::null_mut() {
                unsafe { d_instantiate(dentry, inode) };
                unsafe { c_dget(dentry) };
                error = 0;
                let current_time = unsafe { current_time(dir) };
                unsafe { (*dir).i_mtime = current_time };
                unsafe { (*dir).i_ctime = current_time };
        }
        error
}
```

Beispiel: ramfs_mknod

```
#[no_mangle]
pub extern "C" fn rs_ramfs_mknod(
        dir: *mut inode,
        dentry: *mut dentry,
        mode: umode_t,
        dev: dev_t,
) -> Result<(), cty::c_int> {
        use bindings::ENOSPC;
        use c_fns::{rs_d_instantiate, rs_dget, rs_ramfs_get_inode};

        match rs_ramfs_get_inode(unsafe { (*dir).i_sb }, dir, mode, dev) {
                Some(inode) => {
                        rs_d_instantiate(dentry, inode);
                        rs_dget(dentry);
                        unsafe { (*dir).set_mctime_current() };
                        Ok(())
                },
                None => Err(-(ENOSPC as i32)),
        }
}
```

Kein funktionsinterner Zustand

Keine unentdeckten Null-Pointer möglich

Repräsentation des Fehlers durch einen Fehlerwert