FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

Semestrální práce z KIV/ZOS

Virtuální souborový systém

Autor: Vít Novotný

Datum: 24. prosince 2025

Obsah

1 Cíl práce

Cílem této práce bylo navrhnout a implementovat jednoduchý virtuální souborový systém, který simuluje základní principy reálných souborových systémů používaných v operačních systémech. Implementace je realizována v jazyce Rust a celý souborový systém je uložen v jednom binárním souboru na hostitelském souborovém systému.

Výsledný systém podporuje hierarchickou strukturu adresářů, práci se soubory, alokaci datových bloků pomocí bitmap a využívá koncept i-uzlů (inode) pro správu metadat.

2 Architektura

2.1 Fyzická organizace disku

Virtuální disk je reprezentován jedním binárním souborem, se kterým se pracuje pomocí blokového přístupu. Disk je rozdělen do následujících logických sekcí:

- superblock obsahující globální metadata systému,
- bitmapa i-uzlů,
- bitmapa datových bloků,
- tabulka i-uzlů,
- datová oblast.

Jednotlivé sekce jsou uloženy lineárně a jejich pozice je určena informacemi uloženými v superblocku.

2.2 Datové struktury

2.2.1 Superblock

Superblock obsahuje globální metadata souborového systému. V implementaci je uložen zejména celkový rozsah souborového systému (`fs_size`), identifikační signatura (`"ELFS"`) a identifikátor kořenového i-uzlu (`root_inode_id`). Dále obsahuje začátky a počty bloků jednotlivých oblastí (bitmapa, inode tabulka a datové bloky) ve formě dvojic `<start, count>`.

Velikost bloku není v superblocku uložena; je definována jako pevná konstanta `BLOCK_SIZE = 4096 B`.

2.2.2 I-uzel (Inode)

I-uzel reprezentuje soubor, adresář nebo symbolický odkaz. Struktura obsahuje logickou velikost souboru v bytech (`file_size`), identifikátor (`id`), odkazy na datové bloky a typ objektu (`file_type`). Pro adresování dat jsou použity přímé odkazy `single_directs` (5 položek), dále jednonásobná nepřímá adresa (`single_indirect`) a dvojnásobná nepřímá adresa (`double_indirect`). Implementace nepoužívá trojnásobnou nepřímou adresaci.

Pro podporu odkazů systém udržuje počet referencí (`link_count`). Struktura je zarovnána na pevnou velikost (`INODE_SIZE = 48 B`) a obsahuje rezervovanou část pro padding a případná budoucí rozšíření.

2.2.3 Položka adresáře (Directory entry)

Adresář je uložen jako soubor složený z položek adresáře. Jedna položka obsahuje fixní jméno délky `DIR_NAME_LEN = 12 B` a identifikátor cílového i-uzlu (`inode_id`). Nevyužité položky jsou značeny speciální hodnotou `inode_id = DIR_INODE_UNUSED`.

3 Implementace

Virtuální souborový systém je implementován v jazyce Rust a je navržen modulárně s důrazem na přehlednost a jednoznačné oddělení odpovědností. Architektura systému rozlišuje jádro souborového systému, aplikační logiku a uživatelské rozhraní, což usnadňuje orientaci v kódu i jeho další rozšiřování.

Projekt je rozdělen do následujících modulů:

- `main.rs` – vstupní bod programu a spuštění aplikace,
- `filesystem.rs` – hlavní rozhraní a logika souborového systému,
- `layout.rs` – datové struktury a rozložení virtuálního disku,
- `io.rs` – blokový vstup a výstup nad virtuálním diskem,
- `context.rs` – běhový kontext aplikace,
- `cli.rs` – parsování uživatelských příkazů,
- `file_man.rs` – aplikační logika práce se soubory a adresáři,
- `tui.rs` – textové uživatelské rozhraní,
- `exit_codes.rs` – jednotné návratové kódy programu,
- `consts.rs` – globální konstanty.

Soubor `main.rs` slouží jako vstupní bod aplikace. Zajišťuje inicializaci běhového prostředí, vytvoření globálního kontextu a spuštění hlavní smyčky programu, ve které jsou zpracovávány uživatelské příkazy.

Jádro virtuálního souborového systému je implementováno v modulu `filesystem.rs`. Tento modul poskytuje základní operace nad souborovým systémem, jako je práce s i-uzly, alokace a uvolňování datových bloků, čtení a zápis dat a práce s adresářovou strukturou. Slouží jako centrální rozhraní, které je využíváno vyššími vrstvami aplikace.

Datové struktury a fyzické rozložení virtuálního disku jsou definovány v souboru `layout.rs`. Jsou zde popsány struktury reprezentující superblock, i-uzly a další metadata tak, aby jejich binární reprezentace odpovídala skutečnému uložení dat v binárním souboru představujícím virtuální disk.

Nízkoúrovňová práce se souborem virtuálního disku je zapouzdřena v modulu `io.rs`. Tento modul zajišťuje blokové čtení a zápis dat na základě indexu bloku pomocí posunu v souboru a práce s pevnou velikostí bloků.

Běhový stav aplikace je uchováván v modulu `context.rs`, který propojuje otevřený soubor virtuálního disku, instanci souborového systému a další informace potřebné během zpracování uživatelských příkazů.

Zpracování uživatelského vstupu je řešeno v modulu `cli.rs`, který provádí parsování příkazové řádky a rozpoznání jednotlivých příkazů. Vlastní aplikační logika operací nad soubory a adresáři je soustředěna v modulu `file_man.rs`, který využívá rozhraní jádra souborového systému.

Textové uživatelské rozhraní je implementováno v modulu `tui.rs`, který zajišťuje formátování výstupu, zobrazování informací a chybových hlášení. Jednotné návratové kódy programu jsou definovány v souboru `exit_codes.rs`.

4 Podporované příkazy

Virtuální souborový systém poskytuje sadu příkazů odpovídajících zadání semestrální práce. Příkazy jsou zadávány prostřednictvím interaktivního shellu a jejich zpracování je odděleno od jádra souborového systému.

Každý příkaz je implementován v samostatném zdrojovém souboru ve složce `commands/`, přičemž název souboru přímo odpovídá názvu příkazu (např. `cp.rs`, `mkdir.rs`). Tento přístup zajišťuje přehlednost kódu a umožňuje snadné rozšiřování o další příkazy.

4.1 Základní operace se soubory a adresáři

- `cp s1 s2` – kopírování souboru v rámci souborového systému,
- `mv s1 s2` – přesun nebo přejmenování souboru,
- `rm s1` – odstranění souboru,
- `mkdir a1` – vytvoření adresáře,
- `rmdir a1` – odstranění prázdného adresáře,
- `ls [a1]` – výpis obsahu adresáře,
- `cat s1` – zobrazení obsahu souboru,
- `cd a1` – změna aktuálního pracovního adresáře,
- `pwd` – výpis aktuální pracovní cesty.

Tyto příkazy podporují chybové stavy definované zadáním, například `FILE NOT FOUND`, `PATH NOT FOUND`, `EXIST` nebo `NOT EMPTY`.

4.2 Informační a systémové příkazy

- `info s1/a1` – zobrazení metadat souboru nebo adresáře,
- `statfs` – výpis statistik souborového systému,
- `format <velikost>` – inicializace a formátování virtuálního souborového systému,
- `exit` – korektní ukončení programu.

Příkaz `format` vytváří nový souborový systém o zadané velikosti a inicializuje všechny jeho interní struktury. Příkaz `statfs` poskytuje přehled o využití bloků, i-uzlů a adresářů.

4.3 Integrace s hostitelským systémem

- `incp s1 s2` – import souboru z hostitelského souborového systému,
- `outcp s1 s2` – export souboru z virtuálního systému na disk,
- `load s1` – načtení a provedení příkazů ze skriptového souboru.

Tyto příkazy umožňují propojení virtuálního souborového systému s hostitelským prostředím a dávkové provádění operací.

4.4 Odkazy v souborovém systému

- `slink s1 s2` – vytvoření symbolického odkazu,
- `rmslink s1` – odstranění symbolického odkazu.

Symbolické odkazy jsou implementovány jako samostatný typ položky. Při práci s příkazy `cat`, `ls` a `info` je rozlišeno, zda se jedná o běžný soubor, adresář nebo symbolický odkaz. Při čtení symbolického odkazu je zpřístupněn obsah cílového souboru.

Striktní oddělení jednotlivých příkazů do samostatných modulů odpovídá principům modulárního návrhu a umožňuje snadnou rozšiřitelnost systému bez zásahu do jádra souborového systému.

5 Závěr

V rámci práce jsem úspěšně implementoval jednoduchý virtuální souborový systém, který demonstruje základní principy správy dat v operačních systémech. Systém umožňuje práci se soubory a adresáři, podporuje hierarchickou strukturu a využívá klasické koncepty jako i-uzly a bitmapovou alokaci.

Implementaci lze dále rozšiřovat například o podporu přístupových práv, cache nebo mechanismy zajišťující konzistenci dat.