Semestrální práce z předmětu KIV/ZOS

Souborový systém založený na i-uzlech

Patrik Janoušek

A17B0231P
janopa@students.zcu.cz

Západočeská univerzita v Plzni
Fakulta aplikovaných věd

Obsah

1 Zadání

Úkolem této semestrální práce je vytvořit souborový systém založený na i-uzlech s vlastní příkazovou řádkou. Souborový systém využívá již existujícího souborového systému, kam se umístí jeho datový soubor, se kterým později pracuje. Dále implementuje všechny základní funkcionality pro práci se soubory, které je typické pro unixové prostředí.

Semestrální práce ZOS 2019 (verze dokumentu 01)

Tématem semestrální práce bude práce se zjednodušeným souborovým systémem založeným na i-uzlech. Vaším cílem bude splnit několik vybraných úloh.

Základní funkčnost, kterou musí program splňovat. Formát výpisů je závazný.

Program bude mít jeden parametr a tím bude název Vašeho souborového systému. Po spuštění bude program čekat na zadání jednotlivých příkazů s minimální funkčností viz níže (všechny soubory mohou být zadány jak absolutní, tak relativní cestou):

1) Zkopíruje soubor s1 do umístění s2

```
cp s1 s2
Možný výsledek:
OK
FILE NOT FOUND (není zdroj)
PATH NOT FOUND (neexistuje cílová cesta)
```

2) Přesune soubor s1 do umístění s2

```
mv s1 s2
Možný výsledek:
OK
FILE NOT FOUND (není zdroj)
PATH NOT FOUND (neexistuje cílová cesta)
```

3) Smaže soubor s1

```
rm s1
Možný výsledek:
OK
FILE NOT FOUND
```

4) Vytvoří adresář a1

```
mkdir a1
Možný výsledek:
OK
PATH NOT FOUND (neexistuje zadaná cesta)
EXIST (nelze založit, již existuje)
```

5) Smaže prázdný adresář a1

```
rmdir a1
Možný výsledek:
OK
FILE NOT FOUND (neexistující adresář)
NOT EMPTY (adresář obsahuje podadresáře, nebo soubory)
```

6) Vypíše obsah adresáře a1

```
ls a1
Možný výsledek:
-FILE
+DIRECTORY
PATH NOT FOUND (neexistující adresář)
```

7) Vypíše obsah souboru s1

```
cat s1
Možný výsledek:
OBSAH
FILE NOT FOUND (není zdroj)
```

8) Změní aktuální cestu do adresáře a1

```
cd a1
Možný výsledek:
OK
PATH NOT FOUND (neexistující cesta)
```

9) Vypíše aktuální cestu

```
pwd
Možný výsledek:
PATH
```

10) Vypíše informace o souboru/adresáři s1/a1 (v jakých clusterech se nachází)

```
info a1/s1
Možný výsledek:
```

```
NAME – SIZE – i-node NUMBER – přímé a nepřímé odkazy
FILE NOT FOUND (není zdroj)
```

11) Nahraje soubor s1 z pevného disku do umístění s2 v pseudoNTFS

```
incp s1 s2
Možný výsledek:
OK
FILE NOT FOUND (není zdroj)
PATH NOT FOUND (neexistuje cílová cesta)
```

12) Nahraje soubor s1 z pseudoNTFS do umístění s2 na pevném disku

```
outcp s1 s2
Možný výsledek:
OK
FILE NOT FOUND (není zdroj)
PATH NOT FOUND (neexistuje cílová cesta)
```

13) Načte soubor z pevného disku, ve kterém budou jednotlivé příkazy, a začne je sekvenčně vykonávat. Formát je 1 příkaz/1řádek

```
load s1
Možný výsledek:
OK
FILE NOT FOUND (není zdroj)
```

14) Příkaz provede formát souboru, který byl zadán jako parametr při spuštení programu na souborový systém dané velikosti. Pokud už soubor nějaká data obsahoval, budou přemazána. Pokud soubor neexistoval, bude vytvořen.

```
format 600MB
Možný výsledek:
OK
CANNOT CREATE FILE
```

Budeme předpokládat korektní zadání syntaxe příkazů, nikoliv však sémantiky (tj. např. cp s1 zadáno nebude, ale může být zadáno cat s1, kde s1 neexistuje).

Informace k zadání a omezením

- Maximální délka názvu souboru bude 8+3=11 znaků (jméno.přípona) + \0 (ukončovací znak v C/C++), tedy 12 bytů.
- Každý název bude zabírat právě 12 bytů (do délky 12 bytů doplníte \0 - při kratších názvech).

Nad vytvořeným a naplněným souborovým systémem umožněte provedení následujících operací:

- Defragmentace (defrag) – pokud login studenta začíná **a-i**
  Datové bloky budou organizovány tak, že nejprve budou obsazené a následně volné (předpokládáme dostatek volného místa – minimálně ve velikosti největšího souboru).

- Kontrola konzistence (check) – pokud login studenta začíná **j-r**
  Zkontrolujte, zda jsou soubory nepoškozené (např. velikost souboru odpovídá počtu alokovaných datových bloků) a zda je každý soubor v nějakém adresáři. Součástí řešení bude nasimulovat chybový stav, který následná kontrola odhalí.

- Symbolický link (slink s1 s2) – pokud login studenta začíná **s-z**
  Vytvoří symbolický link na soubor s1 s názvem s2. Dále se s ním pracuje očekávaným způsobem, tedy např. cat s2 vypíše obsah souboru s1.

Odevzdání práce

Práci včetně dokumentace pošlete svému cvičícímu e-mailem. V případě velkého objemu dat můžete využít různé služby (leteteckaposta.cz, uschovna.cz).

Osobní předvedení práce cvičícímu. Referenčním strojem je školní PC v UC326. Práci můžete ukázat i na svém notebooku. Konkrétní datum a čas předvedení práce si domluvte e-mailem se cvičícím, sdělí vám časová okna, kdy můžete práci ukázat.

Do kdy musím semestrální práci odevzdat?

- Zápočet musíte získat do mezního data pro získání zápočtu (10. února 2020).
- A samozřejmě je třeba mít zápočet dříve, než půjdete na zkoušku (alespoň 1 den předem).

Hodnocení

Při kontrole semestrální práce bude hodnocena:

- Kvalita a čitelnost kódu včetně komentářů
- Funkčnost a kvalita řešení
- Dokumentace

2 Analýza

2.1 Datové struktury

Pro vytvoření souborového systému založeného na i-uzlech je potřeba několik základních struktur. Těmi jsou superblok, bitmapy pro obsazenost datových bloků a i-uzlů, i-uzel a samotný datový blok.

2.1.1 Superblok

Superblok je základní datová struktura, která je nezbytná pro fungování celého souborového systému. Uchovává základní informace o jeho struktuře, jako např. popis, počáteční adresy jeho částí, celkovou velikost, atd.

2.1.2 Bitmapa

Bitmapa je jednoduchá datová struktura, ve které je možné nastavit hodnotu biotu na konkrétní pozici na 0 nebo 1. Tato jednoduchá datová struktura se dá v souborovém systému použít ke značení, zda je daný i-uzel nebo datový blok volný k použití, nebo je obsazen.

2.1.3 I-uzel

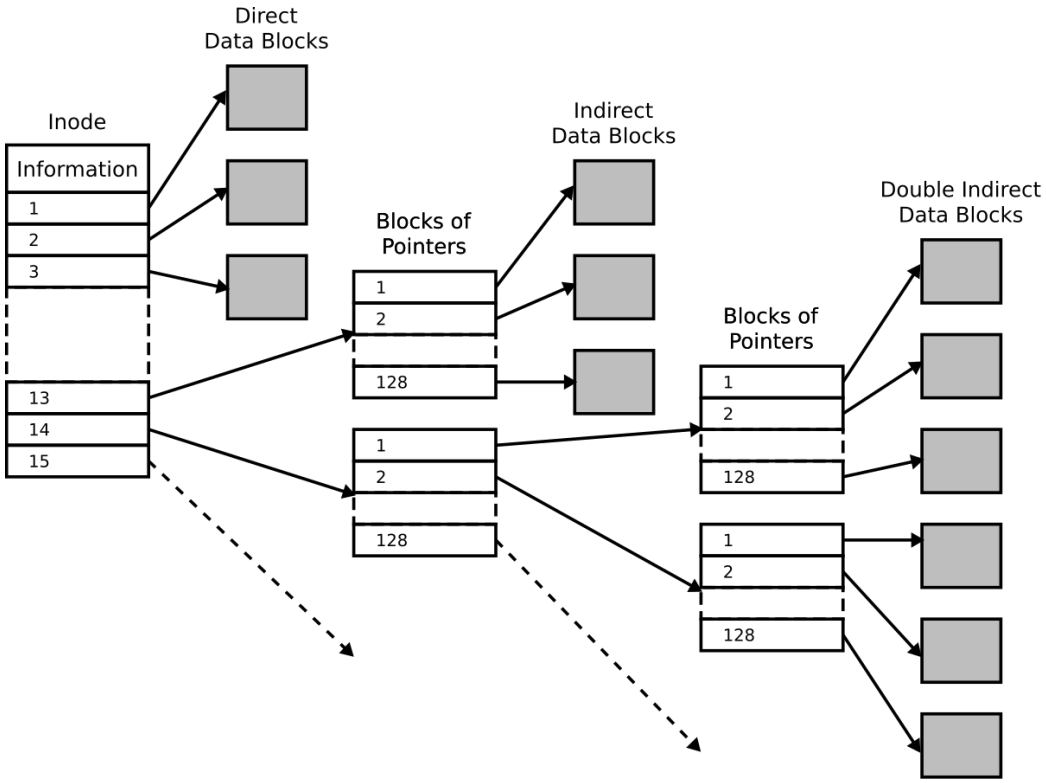
I-uzel je datová struktura, která může být do velké míry chápána jako soubor. Na jeden soubor či složku totiž připadá právě jeden i-uzel. V této datové struktuře jsou obsaženy informace, pro možnost přečtení již zapsaného souboru. Jsou jimi například typ (složka/soubor), velikost zapsaných dat, počet naalokovaných datových bloků, a přímé/nepřímé odkazy na datové bloky.

Přímé i nepřímé odkazy v i-uzlu odkazují na samotné datové bloky, kde jsou zapsaná požadovaná data. Každý typ odkazu k této problematice přistupuje mírně odlišně.

Několik přímých odkazů je uloženo přímo v i-uzlu (zpravidla jednotky) V těchto odkazech jsou pak uloženy přímo odkazy na jednotlivé datové bloky s uloženými daty.

Nepřímé odkazy už fungují mírně komplikovaněji. Odkaz, který je uložen v i-uzlu, neodkazuje na datový blok s daty, ale na datový blok s dalšími odkazy. Až tyto odkazy v datovém bloku nadále odkazují na jednotlivé datové bloky s daty.

Nadále máme dvojité nepřímé odkazy, nicméně ty fungují velmi podobně. Jen s tím rozdílem, že se používá dvojitý odkaz (oproti jednoduchému). Odkaz v i-uzlu tedy odkazuje na datový blok s odkazy, kde každý odkaz odkazuje



Obrázek 1: Vizualizace struktury i-uzlu

na další datový blok s odkazy, které už odkazují na jednotlivé datové bloky s daty.

Takto bychom samozřejmě mohli pokračovat dále. Nicméně s každou násobností nepřímých odkazů velmi výrazně naroste maximální velikost souboru v souborovém systému. Z tohoto důvodu by třeba pětinásobné nepřímé odkazy nedávali smysl. Například souborový systém ext4 využívá nepřímé odkazy až do násobnosti tři.

2.1.4 Datový blok

Datový blok představuje oblast v souborovém systému, kam je možné ukládat různá data. Můžou jimi být například přímo obsahy jednotlivých souborů, seznam souborů ve složce nebo odkazy na další datové bloky.

3 Implementace

Souborový systém byl především z časových důvodů implementován v jazyce Go. I přes toto rozhodnutí však nebylo zaznamenáno, že by samotná implementace jazyka jakkoliv snižovala výkon souborového systému.

3.1 Struktura souborového systému

Souborový potřebuje pro svou funkčnost pevnou strukturu jednotlivých jeho částí. Tyto části jsou pevně rozděleny již při formátování souborového systému a není možné je následně měnit.

Souborový systém je na začátku rozdělen na 2 části. Na datovou část o velikosti 95% celkové velikosti datového souboru, a na část, sloužící pro uložení i-uzlů, bitmap a superbloku, o velikosti 5% celkové velikosti.

Na začátku souborového systému je uložen superblok, následuje bitmapa pro označování obsazených datových bloků. A nakonec je zbylé místo využito na bitmapu použitých i-uzlů a samotné i-uzly. Velikost posledních dvou částí je vypočtena tak, aby se maximalizoval počet i-uzlů, a zároveň pro ně bylo možné vytvořit dostatečně velkou bitmapu.

Při formátování souborového systému je celý datový soubor vynulován a ne jeho počátek je zapsán superblok s počátky adres výše uvedených částí. Díky tomu je formátování nenáročnou a rychlou operací.

3.2 Omezení souborového systému

Žádný souborový systém není plně bez omezení, i běžné souborové systémy mají řadu technických omezení. Jen na ně při běžném použití nenarazíme. U tohoto souborového systému tomu není jinak.

Vytvořený souborový systém nemá žádnou ochranu proti jeho poškození. Ačkoliv poškození datové části pravděpodobně přežije, tak poškození ostatních jeho částí může vést ke ztrátě jednoho či více souborů. Poškození superbloku je pak nevratné a vede k poškození celého souborového systému. Kromě pokusu o ruční opravu zde není žádná jiná cesta jak souborový systém znovu obnovit.

Maximální velikost souboru se pak odvíjí podle velikosti datového bloku. Tato hodnota se dá spočítat podle vzorce:
$(pocet\_primych\_odkazu * velikost\_bloku + \frac{velikost\_bloku^2}{velikost\_odkazu} + \frac{velikost\_bloku^3}{velikost\_odkazu})$

3.3 API souborového systému

API, nebo-li aplikační rozhraní, souborového systému bylo implementováno tak, aby se co nejvíce podobalo práci se soubory, jak jí známe ze standardních souborových systémů (resp. operačního systému). Obsahuje tak operace jako například `Open`, `Read`, `Write`, `Rename` a další. Díky tomuto byla pak implementace příkazové řádky souborového systému triviální operací, jelikož už nebylo potřeba řešit žádné specifické problémy při práci s vytvořeným souborovým systémem.

3.4 Dostupné příkazy v příkazové řádce

Dostupné příkazy příkazové řádky jsou již velmi dobře zdokumentovány v přiloženém zadání. Z tohoto důvodu již nebudou opětovně zmíněny. Jediný příkaz, který je implementován nad rámec zadání, je `badrm`. Tento příkaz je implementován pouze pro ověření funkčnosti kontroly konzistence souborového systému. Jedná se o modifikaci příkazu `rm` s tím rozdílem, že neuvolní i-uzel zvoleného souboru. To má za následek to, že vznikne i-uzel, ke kterému nejde přistoupit, jelikož není v žádné složce v souborového systému. Tento příkaz tedy neslouží pro běžné smazání souboru, a je navržen tak, aby účelně poškodil souborový systém.

3.5 Použité knihovny

Vytvořený souborový systém používá pouze jednu knihovnu třetí strany, a tou je knihovna jménem *ishell*. S pomocí této knihovny bylo možné velmi rychle a efektivně vytvořit prostředí příkazové řádky, které disponuje pokročilejšími funkcemi, jakými jsou například historie příkazů, vyhledávání v historii příkazů, automatické doplňování, atd.

3.6 Struktura implementace

Při programování souborového systému byl kladen důraz na oddělení základních částí implementace. Je tak oddělena práce se samotným úložištěm, i-uzly, tak i například alokace paměti pro i-uzly. Je tak teoreticky možné do budoucna zlepšit způsob alokace datových bloků tak, aby nebylo nutné alokovat pouze po jednom datovém bloku. Jelikož je alokace poměrně drahou operací, tak by tímto bylo teoreticky možné výrazným způsobem zvýšit výkon souborového systému.

Celý souborový systém je implementován v modulu `vfs`, který už není dále členěn do modulů. Nicméně jak již bylo naznačeno výše, stále se zde

dodržuje oddělení jednotlivých problematik souborového systému.

Další modul přímo spjatý se souborovým systémem nese název `vfsapi`. Jedná se aplikační rozhraní určené pro použití třetí stranou. Je takto oddělené ve vlastním modulu, aby bylo zřejmé, co je určené pro vnější volání, a co už je funkce určená právě pro tvorbu API, a při nesprávném použití může způsobit nekonzistenci souborového systému.

Posledním modulem je pak modul `shell`, který obsahuje implementaci jednotlivých příkazů příkazové řádky. Jednoduše řečeno se jedná o velmi tenkou vrstvu mezi příkazovým řádkem a API souborového systému.

4 Uživatelská příručka

4.1 Překlad a sestavení projektu

Pro překlad a sestavení projektu je potřeba mít pouze nainstalovaný programovací jazyk go, a to minimálně ve verzi `1.13.6`. Následně už můžeme spustit kompilaci pouze pomocí `go build .`. Go automaticky stáhne všechny závislosti a provede překlad projektu. Pokud chceme projekt přímo spustit, tak je také možné použít příkaz `go run .`. Pro správnou funkčnost obou zmíněných příkazů je potřeba být v kořenovém adresáři projektu.

4.2 Ovládání

Po spuštění projektu je nadále možné pracovat se souborovým systémem jak je zvykem z prostředí GNU/Linux. Jsou implementovány všechny základní příkazy, jejichž podrobná dokumentace je dostupná v zadání semestrální práce.

5 Závěr

V rámci semestrální práce bylo nutné pochopit a implementovat základní principy souborových systémů. Ačkoliv se to zprvu jednalo o téměř nemožný úkol, tak se po řádném dekomponování problému ukázalo, že to není tak neproveditelné, jak by se mohlo zdát.

Za největší problém implementace souborového systému bych označil to, že dlouho není vidět žádný použitelný výsledek. Nejdříve bylo nutné vůbec navrhnout strukturu souborového systému, následně implementovat bitmapy, i-uzly, alokátor datových bloků, a další komponenty Doba, od vytvoření projektu do zapsání prvního souboru, je tedy poměrně dlouhá. Reálné vytvoření

souboru je dokonce tak pokročilá záležitost, že bylo implementováno v podstatě až při finalizaci celé semestrální práce. Jednalo se totiž v podstatě o pouhé propojení již připravených komponent, které bylo nutné předtím implementovat. Z těchto důvodů byly při vývoji aktivně používány jednotkové testy, které výrazně usnadnili detekci a opravování chyb v průběhu vývoje. Díky nim bylo možné ihned odhalit chybně implementovanou funkcionalitu, nebo rozbití již nějaké stávající.

Co se týče samotného souborového systému, tak za jeho primární nedostatek bych označil jeho výkon a neodolnost vůči vnějším vlivům (chyba disku, výpadek proudu, náhlé odpojené disku, ...). Ačkoliv nakopírovat soubor o velikosti 40MB trvá zhruba jednu sekundu, tak nakopírování souboru o velikosti 1GB je záležitostí na několik minut. Tento jev přisuzuji právě pomalému alokátoru, který hledá volné datové bloky lineárně a vždy od začátku. Ke zrychlení by mohlo dojít například, kdyby se nealokovalo pouze po jednom datovém bloku, ale například po čtyřech.

Souborový systém má samozřejmě i spoustu dalších nedostatků, nicméně pro jejich vyřešení už by se nemohlo jednat o semestrální práci, ale spíše o dlouhodobý několikaletý projekt.

I přes zmíněné nedostatky jsem však s výslednou prací spokojen a podle zadání je splněna.

Seznam obrázků