Semestrální práce z předmětu KIV/ZOS

Virtuální souborový systém založený na i-nodech

Tomáš Linhart

A18B0255P

Západočeská univerzita v Plzni
ZS 2020 / 2021

Obsah

Semestrální práce ZOS 2020 (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, nebo přejmenuje s1 na 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:

- Hardlink (ln s1 s2) – pokud login studenta začíná **a-i**
  Vytvoří hard link na soubor s1 s názvem s2. Dále se s ním pracuje očekávaným způsobem, tedy např. cat s2 vypíše stejný obsah jako cat s1.

- 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 (12. února 2021).
- A samozřejmě je třeba mít zápočet dříve, než půjdete na zkoušku (alespoň 1 den předem).

Hodnocení

Při kontrole semestrální práce bude hodnocena:

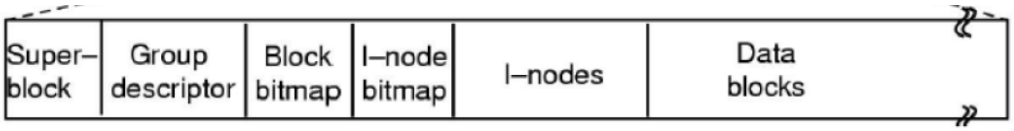
- Kvalita a čitelnost kódu včetně komentářů
- Funkčnost a kvalita řešení
- Dokumentace

1. Úvod

Cílem práce je vytvořit virtuální souborový systém, který bude používat souborové uzly (i-node). Program si virtuální souborový systém načte ze souboru specifikovaném v parametru aplikace. Speciálním příkazem je kontrola konzistence souborového systému, kdy je ověřováno, zda veškeré i-nody jsou ve složce (tzn. jsou dostupné) a zda velikost specifikovaná v i-node odpovídá počtu alokovaných bloků. Pro ověření funkčnosti bude připravena funkce, která změní velikost souboru zapsanou v i-node.

2. Struktury

Souborový systém využívající i-node je rozdělený na několik částí zobrazených v obrázku níže. Tento obrázek je převzatý z prezentace na cvičení předmětu KIV/ZOS.



Obrázek 1: Struktura filesystému využívajícího i-nodů.

2.1 Superblock

Superblock obsahuje základní velikosti o celém filesystému - Podpis autora, popis filesystému, velikost na disku, počet datových bloků, adresu počátku datových bloků a další. Velikost superblocku je přesně daná, aby bylo možné ho načíst pro libovolně velký filesystém.

2.2 Block bitmap

Bitmapa datových bloků obsahuje informace o tom, které datové bloky jsou obsazené a které jsou ještě volné. V aktuální práci je pro jednoduchost využito toho, že hodnota obsazení bitmapy zabírá jeden byte, namísto jednoho bitu. To dovoluje načíst hodnoty přímo do pole int8_t a snadno s nimi pracovat.

2.3 I-Node bitmap

Bitmapa i-nodů obsahuje, které i-nody jsou ještě dostupné a které jsou volné. V této práci ovšem není využita a je nahrazena dynamickým polem i-nodů. Obsazenost i-nodů poté je prováděna iterací přes toto pole a ověřováním, zda se ID nerovná ID_ITEM_FREE, tedy hodnotě -1.

2.4 I-nodes

Jedná se o oblast, ve které jsou uloženy všechny i-nody. Jednotlivý i-node obsahuje informace o velikosti dat v něm uložených, zda je daný i-node složka a odkazy na přímé a nepřímé datové bloky. V i-nodu je také specifikovaný

počet referencí. Tato hodnota ovšem není nikde v práci využita, používá se pouze pro hardlinky.

2.5 Data blocks

Oblast datových bloků slouží pro samotné ukládání dat. Do této oblasti jsou ukládány také bloky sloužící jako nepřímé datové odkazy. Tyto bloky obsahují pouze adresy na další datové bloky a jsou využívány, pokud počet přímých odkazů v i-nodu není dostatečný.

3. Popis implementace

Program je vytvořen v jazyce C. Velikost jednoho datového bloku byla zvolena na 4096B (4KB). To umožňuje uložení větší maximální velikosti souboru - konkrétně $((1024 * 2) + 5) * 4096 = 8409088B$ (cca 8MB). Hodnoty jsou vypočteny tak, že můžeme uložit 1024 adres v datovém bloku (4096 / 4B na adresu) a máme možnost využít dva nepřímé datové bloky. Dále máme pět přímých datových bloků a na každou adresu můžeme uložit 4096B. Vzhledem k zadání je délka názvu souboru omezena na 12 znaků, což se fakticky rovná 11 znaků + NULL terminátor. Toto omezení je pouze umělé, ukládání delších názvu by nebyl problém.

3.1 superblock.c

Tento soubor obsahuje pouze funkci, která inicializuje superblock při formátování souboru. Vypočítá velikosti oblastí pro i-node (5% celkové velikosti) a bitmapu a vypočítá adresy v souboru.

3.2 utils.c

V tomto souboru se nachází různé velmi jednoduché funkce, které slouží jako pomocné funkce pro hlavní části programu, jako například: převedení stringu na int, přidání znaku na začátek řetězce a další.

3.3 main.c

Vstupní bod programu - Pouze načte parametry a inicializuje VFS, poté spustí nekonečnou smyčku, která čte příkazy od uživatele.

3.4 directory.c

Tento soubor obsahuje pouze jednu funkci, která vytvoří novou strukturu dir_item, která obsahuje ID i-nodu a jméno souboru. Struktury dir_item jsou využívány ve spojovém seznamu, kdy každá složka si drží spojový seznam všech svých složek a souborů. Tohoto je využito pro jednodušší prohledávání složek.

3.5 commands.c

Tento soubor obsahuje všechny funkce nutné pro zpracování příkazů zadaných uživatelem. Na začátku je funkce, která určí, o jaký příkaz se jedná, načte parametry a zavolá funkci, která se postará o vykonání příkazu. Nejsložitějšími příkazy jsou příkazy mv, cp, incp. Jejich funkčnost je ovšem velmi podobná - nejdříve je nutné získat volný i-node a volné datové bloky (datové bloky nejsou nutné pro mv) pro nový soubor. Poté jsou postupně do zásobníku o velikosti jednoho datového bloku (4096B) načítány datové bloky ze zdrojového souboru a jsou vkládány do cílového souboru. Nakonec je nutné aktualizovat cílovou složku (Přidat do spojového seznamu souborů; pro mv je nutné aktualizovat i zdrojovou - soubor ze složky odstranit), bitmapu, inode v souboru a rekurzivně změnit velikosti všech složek nahoru ke kořenovému adresáři. Příkaz outcp je velmi podobný, akorát cílový soubor je mimo virtuální souborový systém a využívá klasický fwrite. Při vykonávání příkazu rm je nutné projít veškeré datové bloky daného souboru a nastavit je na nulové. Poté teprve je možné vynulovat i-node (hodnoty nastavit na ID_ITEM_FREE) a aktualizovat bitmapu a i-nody v souboru. Ostatní příkazy jsou poměrně snadné a jejich funkcionalitu je snadné vyčíst ze zdrojových kódů. Pro mé zadání byla podmínka připravit příkaz pro kontrolu konzistence dat. Tento příkaz funguje na principu, že projde mapu inodů a uloží se do seznamu ID všech inodů, které jsou alokované (tzn jejich ID se nerovná ID_ITEM_FREE). Poté rekurzivně prochází celý souborový systém a pro každý inode odstraní ID z daného seznamu. Poté projde seznam a pokud se v seznamu nachází nějaké ID inodu, je kontrola konzistence prohlášena za chybnou, jelikož se nepodařilo dostat ke všem i-nodům. Druhým krokem je opět rekurzivní průchod souborovým systémem - pro každý soubor (ne složku) je zjištěný reálný počet alokovaných bloků. Následně je vypočítán očekávaný počet alokovaných datových bloků - tato čísla se musí shodovat, jinak je kontrola konzistence prohlášena za chybnou. Pro ověření funkčnosti je připravena funkce size, kterou je možné ručně nastavit velikost souboru pro daný i-node. Nakonec mám připravený příkaz debug, kterým zjistím údaje o superblocku, datovou bitmapu a ID alokovaných i-nodů. Program lze ukončit příkazem exit.

3.6 vfs.c

V souboru vfs.c jsou veškeré funkce, které souvisí se čtením, zápisem a aktualizací dat ve virtuálním souborovém systému. Je zde veškerá logika, která stojí za hledáním nových datových bloků a prázdných inodů. Jsou zde funkce

pro vytvoření nového i-nodu a vytvoření složky. Při spuštění programu je zavolána funkce load_vfs, která načte virtuální souborový systém z reálného souboru uloženého na disku. Nejdříve snadno načte superblock, jelikož má pevně zadanou délku. Poté již postupně načítá bitmapu a i-nody za pomocí adres získaných ze superblocku. Pro usnadnění operací je vytvořena struktura *directory*, která obsahuje odkaz na adresář o úroveň vyšší, a odkazy na struktur *dir_item*, aktuálního prvku, na první podadresář a na první soubor ve složce. Toto nám umožňuje jednodušeji procházet souborovou strukturou, kdy máme seznam všech dceřinných souborůsložek a zároveň i odkaz na rodičovský adresář. Tuto strukturu je ovšem nutné vytvořit při načtení souboru virtuální souborového systému. Procházíme i-nody daného adresáře a pokud daný i-node je složka (Můžeme snadno ověřit - v i-node je proměnná isDirectory), rekurzivně projdeme tento adresář a přidáme ho do seznamu všech složek (Seznam *all_dirs* se nachází ve struktuře *vfs*) na pozici daného inodu. Pro procházení souborem virtuálního souborového systému je připraveny funkce seek_data_block a další, které usnadňují výpočet adres a posouvají se v datovém bloku podle zadaného čísla bloku.

4. Uživatelská příručka

Uživatelská příručka obsahuje návod jak program zkompilovat a jak ho spustit

4.1 Kompilace

Níže můžete najít návod, jak zkompilovat server na systému Linux. Kompilování na systému Windows není ověřeno.

4.1.1 Linux

Pro kompilaci na systému Linux je nutné mít stažený program *gcc* a *make*, které je možné stáhnout příkazem *sudo apt-get install gcc make -y*. Poté je možné program sestavit zavoláním příkazu

make

ve složce se zdrojovými kódy programu.

4.2 Spuštění aplikace

Server na Linuxu spustíme pomocí příkazu

./zos_ vfs [Jméno_ filesystému]

Parametr *Jméno_ filesystému* specifikuje, v jakém souboru bude uložený virtuální souborový systém. Tento parametr je povinný. Filesystém je nutné při prvním spuštění naformátovat. Naformátování připraví virtuální souborový systém, jeho superblok, bitmapu, oblast i-nodů a vynuluje veškeré datové clustery. Soubor obsahující virtuální souborový systém na disku poté zabírá místo specifikované formátováním. Po naformátování je možné souborový systém používat, příkazy je možné zobrazit napsáním příkazu *help*.

4.3 Dostupné příkazy

Příkazy je možné získat přímo z programu pomocí příkazu help. Pro přehlednost jsou uvedeny také níže v tabulce.

Příkaz	Funkce
cp s1 s2	Zkopíruje soubor na cestě s1 na cestu s2
mv s1 s2	Přesune soubor na cestě s1 na cestu s2
rm s1	Odstraní soubor s1
mkdir a1	Vytvoří adresář a1
rmdir a1	Odstraní adresář a1
ls a1	Vypíše obsah adresáře a1
cat s1	Vypíše obsah souboru s1
cd a1	Změní aktuální složku na adresář na adrese a1
pwd	Vypíše cestu k aktuální složce
info a1/s1	Vypíše informace o daném souboru / složce
incp s1 s2	Přesune soubor z reálného souborového systému do virtuálního
outcp s1 s2	Přesune soubor z virtuálního souborového systému do reálného
load s1	Přečte příkazy řádek po řádku ze souboru s1 z reálného souborového systému
format 600M	Naformátuje virtuální souborový systém (VFS)
check	Provede kontrolu konzistence souborového systému
size inode_id 600M	Změní velikost i-nodu s daným ID na velikost specifikovanou parametrem

Tabulka 4.1: Přehled příkazů

5. Závěr

Výsledkem práce je program představující virtuální souborový systém založený na využití i-nodů. Veškeré příkazy včetně kontroly konzistence byly implementovány. Program je rozdělený do jednotlivých logických částí pomocí různých .c a .h souborů. Program se mi z časových důvodů nepodařilo řádně otestovat, avšak při běžném využití (Happy-day scénář) byl program funkční a bez chyb. Funkčnost programu byla ověřována pouze v operačním systému Linux (Respektive prostředí WSL - Windows Subsystem for Linux). Funkčnost v operačním systému Windows ověřena nebyla, a proto některé části programu nemusí fungovat správně.

Seznam obrázků