FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

Základy operačních systémů

KIV/ZOS

Pavel Třeštík
A17B0380P

11. února 2021

Obsah

1 Úvod

Cílem této práce je vytvořit zjednodušný souborový systém (anglicky file system, dále jen (pseudo) FS) založený na reálném FS i-uzlů (anglicky i-node, dále jen inode). Dále nad tímto systémem implementovat některé základní linuxové příkazy jako například "cd, ls"a další. Tyto příkazy jsou zadávány do konzole, která bude součástí programu.

Konkrétní příkazy, které mají být implementovány jsou:

- cp - zkopíruje soubor
- mv - přesune soubor
- rm - smaže soubor
- mkdir - vytvoří adresář
- rmdir - smaže adresář
- ls - vypíše obsah adresáře
- cat - vypíše obsah souboru
- cd - změní aktuální cestu do adresáře
- pwd - vypíše aktuální cestu
- info - vypíše informace o soubrou/ adresáři
- incp - zkopíruje soubor pevného disku do pseudo FS
- outcp - zkopíruje soubor z pseudo FS na pevný disk
- load - načte soubor z pevného disku a vykonavá příkazy z tohoto file
- format - formátuje soubor na pseudo FS
- slink - vytvoří symbolický link

2 Analýza

Programu se předá název souboru jako argument při spuštění. Do tohoto souboru je ukládán pseudo FS.

2.1 File system

File systém má 4 hlavní struktury, ze kterých se skládá. Tyto struktury jsou ve FS poskládány dle obrázku 2.1.

Tyto struktury jsou:

1. blok (anglicky block) - uchovává data
2. inode - uchovává metadata o souboru/ adresáři
3. bitmapa - mapa určující obsazenost bloků/ inode
4. superblock - uchovává metadata FS a adresy ostatních struktur

Obrázek 2.1: Struktura file systému

2.1.1 Blok

Je to blok uložiště o konstantní velikosti.

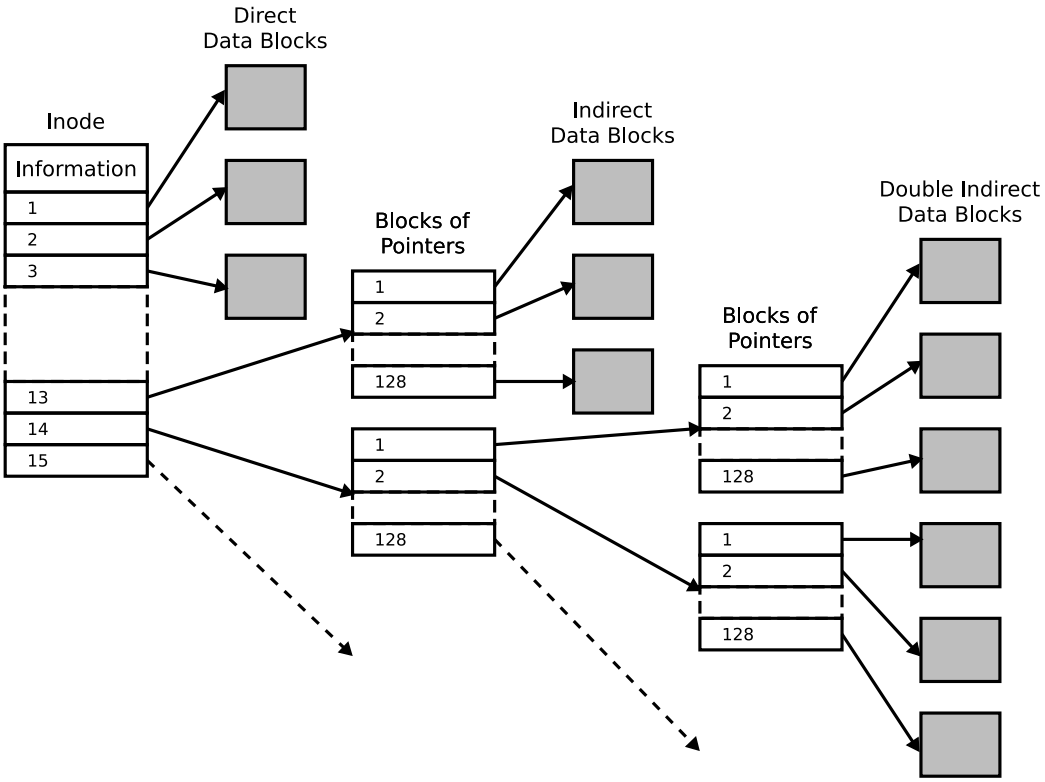
2.1.2 Inode

Je to struktura, která reprezentuje každý adresář a soubor ve FS. Uchovává základní informace o adresáři/ souboru jako například velikost, ID a další. V inode jsou také uchovávány odkazy na bloky.

Každý adresář má právě jeden blok a tudíž je limitovaný počet položek, které mohou být v adresáři uloženy. Soubory mají proměnné počty bloků,

ale stále jsou limitovány maximálním počtem bloků. Maximální počet bloků je také spjatý s velikostí bloku.

Na obrázku 2.2 je znázorněna struktura inode. V tomto obrázku jsou indirect bloky až 13 a 14 odkaz inode. V této práci je ale struktura inodu zjednodušená a má pouze 5 přímých odkazů, 1 nepřímý odkaz první úrovně (odkaz 13 v obrázku) a 1 nepřímý odkaz druhé úrovně (odkaz 14 v obrázku).



Obrázek 2.2: Struktura inode
Zdroj: https://en.wikipedia.org/wiki/Inode_pointer_structure

2.1.3 Bitmapa

Pro každý blok/ inode je v příslušné bitmapě jeden bit, který určuje zda-li je blok/ inode na té pozici obsazený.

2.1.4 Superblock

Mimo uchovávání metadat, jako například "podpis"tvůrce FS, uchovává umístění ostatních částí. V superblocku se uchovávají adresy bitmap, inode a bloků. Je zde také zaznamenán počet bloků a velikost jednoho bloku.

2.2 Příkazy

Příkazy mají fungovat na principu stejnojmenných příkazů v systému Linux. Protože příkazy mají být implementovány nad zjednodušeným pseudo FS, tak některé příkazy mohou fungovat trochu jinak než v typickém Linuxu nebo mohou mít úplně jinou funkci (např. příkaz `info`). Většina příkazů má jeden nebo dva parametry, které odkazují na soubor nebo adresář ve FS. Všechny soubory a adresáře jsou zadány absolutní nebo relativní cestou, proto musí být FS schopný najít soubor z aktuální pozice i z root adresáře FS.

3 Programátorská dokumentace

Program je psán v jazyce C, využítím pouze standardních knihoven.

3.1 Moduly

Moduly do kterých je aplikce rozdělená.

- `main` - spouštěcí modul programu
- `general_functions` - poskytuje strukturu linked list a makra
- `console` - modul spouští konzoli, kterou uživatel ovládá program. Rozkládá vstup na části a volá příslušné funkce
- `fs_manager` - zpracuje vstup, který převezme od modulu `console` a volá obslužné funkce z modulu `commands` se zpracovanými vstupy
- `commands` - vykonává funkce příkazů a provádí změny ve FS
- `file_system` - poskytuje funkce manipulující s FS a použité FS struktury

3.2 Implementace příkazů

Všechny příkazy jsou implementovány v modulu `commands`. Každý příkaz využívá nějákou funkci z modulu `file_system`, aby vykonal akci nad FS.

3.2.1 cp

Prototyp: `int copy(int32_t sparent, int32_t tparent, char* sname, char *tname)`. Vstupy funkce jsou jméno zdroje (`sname`) a cíle (`tname`) a ID adresářů ve kterých se tyto jmena mají nacházet. Funkce hledá zdroj a cíl v adresáři podle jména a podle toho, zda úspěšně najde výsledek reaguje.

Pokud `sname` neexistuje, funkce končí chybou. Na cíl funkce reaguje podle toho, zda-li `tname` existuje, je adresář nebo je soubor. Pokud `tname` neexistuje, vytvoří se v adresáři s ID `tparent` soubor se jménem `tname`. Pokud je `tname` adresář vytvoří se v něm soubor se jménem `sname`. Pokud je `tname` existující soubor, vymažou se data souboru a nahradí se daty souboru `sname`.

3.2.2 mv

Prototyp: `int move(int32_t sparent, int32_t tparent, char* sname, char *tname)`. Chová se podobně jako cp, ale funguje velmi odlišně. Protože není třeba zachovat původní soubor `sname`, tak příkaz jednoduše přesune/ přejmenuje adresářovou položku `sname` do/ na `tname`.

3.2.3 rm

Prototyp: `int remove_file(int32_t where, char *name)`. Hledá položku se jménem `name` v adresáři s inode ID where. Vrací chybu, pokud soubor není nalezen. Pokud je soubor nalezen, jsou vymazána data souboru a položka souboru z adresáře `where`.

3.2.4 mkdir

Prototyp: `int make_directory(char *name, int32_t parent_nid)`. Načte inode s ID `parent_nid`, zkontroluje, zda-li jméno name již existuje a pokud neexistuje, tak vytvoří nový inode pro adresář a přidá položku s `name` do načteného adresáře.

3.2.5 rmdir

Prototyp: `int remove_directory(int32_t node_id)`. Načte inode s ID `node_id`, zkontroluje, zda-li je adresář prázdny a následně uvolní blok, do kterého jsou ukládány položky a vynuluje inode s `node_id`.

3.2.6 ls

Prototyp: `int list_dir_contents(int32_t node_id)`. Načte inode s ID `node_id`, pokud se jedná o adresář, načítá a vypisuje položky v něm uložené.

3.2.7 cat

Prototyp: `int cat_file(int32_t where, char *name)`. Načte inode adresáře `where` a hledá v něm položku `name`. Po nalezení položky, vypíše její obsah. Načítá po velikosti bloku.

3.2.8 cd

Prototyp: `int change_dir(int32_t target_id)`. Načte inode s ID `target_id` a pokud se jedná o adresář, nastaví globální proměnou `position`, která určuje aktuální pozici ve FS, na načtený inode.

3.2.9 pwd

Prototyp: `int print_working_dir()`. Funkce postupně prochází adresáře pomocí ".."(předchozí adresář), dokud nedojde do root adresáře. Průchodem ukládá každý prošlý adresář do linked listu a ten po dosažení root adresáře vypíše.

3.2.10 info

Prototyp: `int node_info(int32_t where, char *name)`. Vypíše informace o adresáři nebo souboru `name` nacházející se v adresáři s inode ID `where`.

3.2.11 incp

Prototyp: `int in_copy(char *source, int32_t t_node, char *source_name, char *target_name)`. Načte soubor z uložiště mimo FS této práce a zkopíruje data tohoto souboru do pseudo FS. Zdroj souboru je `source` a `source_name` je pouze název souboru v případě, že soubor byl zadán cestou. Dále je načten inode s ID `t_node`, který by měl adresář, ve kterém se hledá položka `target_name`. Příkaz reaguje stejně jako **cp** na existenci a typ souboru/ adresáře `target_name`.

3.2.12 outcp

Prototyp: `int out_copy(int32_t where, char *name, char *target)`. Plní opačnou funkci k **incp**. Tedy koupíruje soubor se jménem `name` v adresáře s inode ID `where` do souboru `target`, který se nachází mimo pseudo FS.

3.2.13 load

Prototyp: `void load(char *file)`. Tento příkaz není implementován v modulu `commands`. Funkce načte soubor `file` a provádí příkazy, které jsou v souboru uloženy. Formát souboru je jeden příkaz na řádku. Tato funkce je implementována v modulu `console`, protože musí rozložit načtený vstup a volat příslušné obslužné funkce.

3.2.14 format

Prototyp: `int format(char *size)`. Je druhý příkaz, který není implementovaný v modulu `commands`, ale je implementován v modulu `fs_manager`. Je očekáván argument jako číslo s jednotkami (např. 100MB). Po převední jednotek na číslo, se začne vytvářet FS do souboru, který byl zadán jako argument programu. Nejdříve se vytvoří struktura `superblock`, do které jsou spočítány adresy počátků ostatních částí FS. Adresy jsou počítány nejdříve pomocí inkrementu 1:5. Tedy 1B do bitmapy inode (8 inode v každém inkrementu) ku 5B do bitmapy bloků (tedy 40 bloků). Pokud již není možné do zadané velikosti FS vložit tento inkrement je zbytek místa doplněn bloky po 8 (tedy 1B v bitmapě bloků). Postupně se potom zapisují data do souboru. Nejdříve se zapíše `superblock`, poté první byte inode bitmapy, kde se první inode rovnou inicializuje jako zabraný. Po doplnění 0 do začátku bitmapy bloků, je stejně jako u inode bitmapy první blok inicializovaný a poté doplněn 0 do inode adresy. Poté je incializován root inode s ID 1 a zapsán do souboru. Tato inode struktura je poté vynulována a dále zapisována do souboru do dosažení adresy bloků. Na začátku adresy bloků jsou inicializovány adresářové položky "."a ".."root adresáře a zbytek souboru je doplněn nulami do spočítané velikost.

3.2.15 slink

Prototyp: `int symbolic_link(int32_t src, int32_t par, char *name)`. Vytvoří nový inode, který ukazuje na inode ID `src`, pomocí direct1. Tento inode je uložen pod jménem `name` do adresáře s inode ID `par`. Podle definice se tedy jedná o "smart link", protože link je vázáný na inode ID, tedy funguje i při přesunutí linku do jiného adresáře nebo přejmenování zdrojového souboru/ adresáře. "Symbolic link"pouze ukazuje na položku v adresáři a při přesunutí nebo přejmenování zdrojového souboru/ adresáře, je link rozbit.

4 Uživatelská dokumentace

4.1 Přeložení a spuštění programu

Jedná se o konzolovou aplikaci. Se zdrojovými soubory je přidán i **makefile**. Požadavky jsou překladač **gcc** a příkaz **make**. Přeložení aplikace je tedy velmi jednoduché. Stačí zavolat příkaz **make** v adresáři s **makefile**.

Po provední příkazu **make** je vytvořen spustitelný soubor **runfs**. Tento soubor požaduje právě jeden parametr, který odkazuje na soubor, kde bude vytvořen pseudo FS.

4.2 Příkazy

Všechny argumenty příkazů mohou být zadány absolutní nebo relativní cestou.

- cp s1 s2 - zkopíruje soubor s1 do/ na s2 (s2 může být adresář i soubor)
- mv s1 s2 - přesune soubor s1 do/ na s2 (pokud s2 neexistuje, s1 je přejmenován na s2)
- rm s1 - smaže soubor s1
- mkdir a1 - vytvoří adresář se jménem a1
- rmdir a1 - smaže adresář se jménem a1
- ls a1 - vypíše obsah adresáře a1
- cat s1 - vypíše obsah souboru s1
- cd a1 - změní aktuální cestu do adresáře a1
- pwd - vypíše aktuální cestu
- info s1/ a1 - vypíše informace o souboru/ adresáři s1/ a1
- incp s1 s2 - zkopíruje soubor s1 pevného disku do pseudo FS adresáře/ souboru s2

- outcp s1 s2 - zkopíruje soubor s1 z pseudo FS na pevný disk do s2
- load s1 - načte soubor s1 z pevného disku a vykonavá příkazy z tohoto file
- format xxxYY- formátuje soubor na pseudo FS, xxx je číslo a YY jsou jednotky (př. 100MB)
- slink s1 name - vytvoří symbolický link se jménem name ukazující na s1

5 Závěr

Cílem práce bylo implementovat file system založený na inode a některé základní příkazy pro užití a testování tohoto FS. Oproti reálnému FS využívající inode je ale práce zjednodušena.

Práce je stabilní a příkazy z většiny fungují jako příkazy systému Linux. Rozdíl je například u příkazu info, který má kompletně jinou funkci v Linuxu než v této práci.

Ovšem i tak jsou na práci věci, které by se daly vylepšit. Především výkonostní věci. Příkladem může být přepisování souboru některým příkazem jako mv/ cp. Ve stávající implementaci, pokud cílový soubor existuje, tak jsou nejdříve odstraněny všechny reference na bloky dat tohoto souboru a pak jsou znovu přiděleny nové bloky. Toto je ovšem poměrně neoptimální řešení, protože pokud na příklad bude zrojový soubor větší než původní, tak není nutné odstranit všechny reference a znovu přidělit potřebný počet bloků. Stačilo by pouze přidat počet bloků, aby bylo možné uložit zdrojový soubor a přepsat již alokované bloky.