FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

Semestrální práce

Základy operačních systémů

Ondřej Drtina
A17B0202P
drtinao@students.zcu.cz

Obsah

1 Zadání

Úkolem bylo v jazyce C vytvořit zjednodušený souborový systém založený na i-uzlech. V rámci vytvořeného souborového systému mělo být umožněno uživateli provádět klasické akce známé z Unixových operačních systémů, tedy např.: cp (pro zkopírování souboru), mv (pro přesun souboru), pwd (výpis aktuálně procházeného adresáře). Příkazů je samozřejmě více a jejich celý výčet lze nalézt v zadání (viz dále).

1.1 Celé znění úlohy

Celé zadání úlohy je možno nalézt online zde.

2 Analýza problému

Před řešením úlohy jsem si nastudoval princip funkce souborového systému založeného na i-uzlech, jelikož jeho pochopení je pro řešení úlohy klíčové. Souborový systém založený na i-uzlech se jako výchozí využívá ve většině distribucí systému Linux (např. Ubuntu), a tak o něm lze na internetu najít dostatek informací. Stručný popis systému založeného na i-uzlech lze nalézt níže.

2.1 Základní pojmy

V rámci systému založeného na i-uzlech se setkáme zejména s následujícími pojmy - viz podkapitoly.

2.1.1 Cluster

Část disku, která je určena pro uživatelská data, je rozdělena na tzv. clustery. Clustery mají jednotnou velikost a každý soubor zabírá minimálně jeden cluster.

2.1.2 I-uzel

Jedná se o strukturu, jež reprezentuje jednu položku uloženou na disku (soubor či adresář). Součástí i-uzlu bývá jemu přiřazené číslo, počet referencí, celková velikost souboru či složky a v neposlední řadě obsahuje také čísla clusterů, na nichž se vyskytuje obsah souboru. V případě větších souborů jsou využity i nepřímé odkazy na clustery (odkaz na cluster, jež obsahuje čísla dalších clusterů).

2.2 Součásti systému založeného na i-uzlech

V případě systému založeného na i-uzlech je nutno na disku rezervovat určitou část paměťového prostoru pro určité položky, jež jsou pro správnou funkci souborového systému nezbytné. Jmenovitě se jedná o: superblock, bitmapu reprezentující stav clusterů, bitmapu reprezentující stav i-uzlů, i-uzly, clustery.

2.2.1 Superblock

Tato komponenta bývá většinou uložena na začátku paměťového prostoru a obsahuje základní informace o disku. Typicky se jedná o číselný údaj reprezentující celkovou velikost disku, počet dostupných clusterů, velikost jednoho clusteru a adresu začátku prostoru určeného pro uživatelská data. Dle způsobu implementace může obsahovat i další informace, například název disku.

2.2.2 Bitmapa reprezentující stav clusterů

Pomocí tohoto prvku lze zjistit stav jednotlivých clusterů (volný / obsazený). Konkrétní podoba se liší dle implementace.

2.2.3 Bitmapa reprezentující stav i-uzlů

Slouží k získání stavu jednotlivých i-uzlů (volný / obsazený). Je možno implementovat více způsoby.

2.2.4 I-uzly

Popis viz předchozí kapitola. Reálný souborový systém uchovává řádově desetitisíce i-uzlů.

2.2.5 Clustery

Popis viz předchozí kapitola. V případě skutečného souborového systému jejich počet většinou převyšuje počet i-uzlů.

3 Popis implementace

3.1 Naformátování disku

3.1.1 Postup

Jedná se o proces, v rámci kterého jsou do uživatelem zvoleného souboru reprezentujícího souborový systém uloženy základní struktury nutné pro správnou funkčnost souborového systému. Do virtuálního disku (souboru) jsou postupně zapsány následující součásti (pořadí zápisu shodné s pořadím podkapitol).

3.1.2 Superblock

Superblock je reprezentován pomocí struktury, jež byla dodána vyučujícím. Obsahuje běžné položky jako je název disku, velikost clusteru, počet clusterů, celkovou velikost disku atd. Pro lepší orientaci v souborovém systému jsem si do struktury přidal i položku, jež obsahuje počáteční adresu bitmapy s i-uzly. Níže přikládám kód struktury reprezentující superblock.

```
1 struct superblock {
2     char signature[9]; //autor FS
3     char volume_descriptor[251]; //popis FS
4     int32_t disk_size; //velikost disku
5     int32_t cluster_size; //velikost clusteru
6     int32_t cluster_count; //pocet clusteru
7     int32_t bitmap_clus_start_address; //pocatek
          adresy s bitmapou clusteru
8     int32_t bitmap_in_start_address; //pocatek adresy
          s bitmapou i-uzlu
9     int32_t inode_start_address; //pocatek adresy s
          i-uzly
10    int32_t data_start_address; //pocatek adresy dat
11 };
```

Tato struktura je po inicializaci uložena na začátek souboru reprezentující souborový systém.

3.1.3 Bitmapa reprezentující stav clusterů

Bitmapa clusterů je realizována pomocí mnou vytvořené struktury, která obsahuje dvě proměnné typu int (jedna udává počet volných clusterů, druhá počet obsazených). Dále je ve struktuře obsaženo pouze pole, jehož velikost je shodná s počtem clusterů - pomocí něj lze zjistit stav jednotlivých clusterů. Pokud je cluster s požadovaným číslem volný, obsahuje pole na odpovídající pozici 0, jinak 1. Konkrétní podobu struktury lze vidět níže.

```
1 struct bitmap_clusters{
2     int *cluster_bitmap; /* pole se stavem clusteru */
3     int cluster_free_total; /* pocet volnych clusteru
          */
4     int cluster_occupied_total; /* pocet obsazenych
          clusteru */
5 };
```

Struktura je po naplnění uložena do souboru reprezentující souborový systém. V daném souboru se před bitmapou s clustery nachází superblock.

3.1.4 Bitmapa reprezentující stav i-uzlů

Definice struktury s bitmapou obsahující stav i-uzlů je velice podobná definici bitmapy s clustery. I v tomto případě potřebujeme pole, jehož velikost je shodná s počtem i-uzlů a dvě proměnné typu int (jedna udává počet volných i-uzlů a druhá počet obsazených). Konkrétní podoba struktury viz níže.

```
1 struct bitmap_inodes{
2     int *inodes_bitmap; /* pole se stavem i-uzlu */
3     int inodes_free_total; /* pocet volnych i-uzlu */
4     int inodes_occupied_total; /* pocet obsazenych
          i-uzlu */
5 };
```

Struktura je po naplnění uložena do souboru reprezentující souborový systém. V tomto souboru se před bitmapou s i-uzly nachází superblock + bitmapa se stavem clusterů.

3.1.5 I-uzly

Reprezentace i-uzlů je v programu zajištěna pomocí struktury, jež byla dodána vyučujícím. Klíčovými prvky struktury jsou: číslo i-uzlu, počet referencí, velikost souboru a samozřejmě čísla clusterů, na kterých je odpovídající soubor uložený. Pro clustery je v rámci struktury zavedena podpora pro až druhý nepřímý odkaz (tzn. struktura obsahuje číslo clusteru, jež obsahuje čísla dalších clusterů, na kterých jsou uvedena čísla clusterů obsahujících data). Zmiňovanou strukturu přikládám níže.

```
1 struct pseudo_inode {
2     int32_t nodeid; /* cislo uzlu */
3     bool isDirectory; /* soubor / adresar */
4     int8_t references; /* pocet odkazu */
5     int32_t file_size; /* velikost souboru */
6     int32_t direct1; /* 1. primy odkaz */
7     int32_t direct2; /* 2. primy odkaz */
8     int32_t direct3; /* 3. primy odkaz */
9     int32_t direct4; /* 4. primy odkaz */
10    int32_t direct5; /* 5. primy odkaz */
11    int32_t indirect1; /* 1. neprimy odkaz */
12    int32_t indirect2; /* 2. neprimy odkaz */
13 };
```

Struktura je po inicializaci uložena do souboru reprezentujícího souborový systém. V tomto souboru se před ní nachází superblock + bitmapa se stavem clusterů + bitmapa se stavem i-uzlů. Počet takto uložených struktur je roven počtu celkovému počtu i-uzlů.

3.1.6 Část prostoru pro uživatelská data

Po zapsání zmíněných struktur do souboru je doposud nevyužitá část souboru přepsána nulami, což zajistí vymazání veškerých uživatelem uložených dat. Do tohoto prostoru se následně ukládají data uživatele. Velikost tohoto prostoru lze získat pomocí vzorce: (počet clusterů * velikost jednoho clusteru).

3.2 Princip funkce vybraných příkazů

Zde naleznete pouze popis příkazů, které nejsou v běžných Unixových systémech dostupné. Zbylé, běžné, příkazy jsou přehledně zdokumentovány

v přiloženém zdrojovém kódu (anglicky).

3.2.1 incp

Zřejmě nejdůležitější funkce, jež slouží pro import souborů do souborového systému. Před začátkem importu proběhne kontrola, na základě které zjistíme, zda je v systému dostupný požadovaný počet clusterů - pokud ne, je import zamítnut. Počet požadovaných clusterů pro data získáme pomocí vzorce (velikost souboru / velikost jednoho clusteru). Je však potřeba připočítat i clustery potřebné pro uložení odkazů na clustery, pokud překročíme kapacitu přímých odkazů. Jelikož máme dostupných pouze 5 přímých odkazů, tak k využití nepřímých odkazů dojde již při velikosti souboru překračující (5 * velikost jednoho clusteru). Pokud je v systému nalezen požadovaný počet clusterů (pro soubor, případně i odkazy), je zahájeno čtení souboru a postupné ukládání jeho částí (velikost jednoho clusteru) do získaných clusterů. Adresu clusteru získáme jako: (počátek adresy clusterů + velikost clusteru * číslo clusteru). Následně je nalezen volný i-uzel, jemuž jsou clustery přiřazeny a vytvořen odkaz v aktuálně procházeném adresáři.

3.2.2 outcp

Příkaz slouží k vykopírování souboru z virtuálního souborového systému do skutečného souborového systému, jež využívá disk uživatele. Základem je nalezení i-uzlu, který obsahuje odkazy na požadovaný soubor. Poté je vytvořen soubor ve skutečném souborovém systému a do něj je postupně vepisován obsah jednotlivých clusterů, na něž odkazuje nalezený i-uzel reprezentující soubor. Při získávání obsahu posledního clusteru je nutno ověřit celkovou velikost souboru a eventuelně přečíst pouze část clusteru, jelikož poslední cluster nemusí být zcela využit.

3.2.3 defrag

Provede defragmentaci souborového systému. V systému jsem se rozhodl implementovat tzv. úplnou defragmentaci. Zmíněný typ defragmentace zajistí, že veškerá data v rámci souborového systému jsou uložena za sebou (nikoliv pouze části jednoho souboru jako u částečné defragmentace). Postup, který jsem využil je relativně jednoduchý, avšak při větším množství souborů by byl časově náročný. Během defragmentace procházím od začátku veškeré obsazené i-uzly a jejich clustery si čísluji od nuly. Pokud je tedy například jako první i-uzel, který obsahuje odkazy na 140 clusterů, očísluji si je jako 0-139 (clustery, které budu chtít i-uzlu později přiřadit). Udělám si

seznam souborů, které obsazují clustery v rozmezí 0-139 a zároveň nejsou souborem, který patří právě procházenému i-uzlu. Takové soubory jsou pak přesunuty na jiné volné clustery a je aktualizován obsah i-uzlu, jež je s daným souborem spojen. Clustery 0-139 jsou nyní volné a zkopíruji do nich tedy obsah clusterů, na než ukazuje zmíněný první i-uzel a aktualizuji obsah tohoto i-uzlu. Dále pak postupuji analogicky - tzn. uvolním požadovaný prostor, nakopíruji obsah clusterů a závěrem aktualizuji obsah i-uzlu.

3.2.4 load

Jedná se o speciální funkci souborového systému, jež sekvenčně provede příkazy uvedené v textovém souboru. Implementace byla jednoduchá - postupně čtu jednotlivé znaky ze souboru a ukládám je do bufferu, dokud nenarazím na konec řádky. Poté je obsah bufferu analyzován a na základě obsahu je proveden požadovaný příkaz. Následně je čtena další řádka atd.

3.2.5 help

Přidal jsem i nápovědu, v rámci níž se uživatel dozví syntaxi příkazů.

4 Uživatelská příručka

Při spuštění programu je požadován jeden parametr - název souboru, v rámci něhož bude probíhat simulace. Jako první musí být proveden příkaz format, jež provede přípravu datového souboru. Poté může být použit jakýkoliv z následujích příkazů.

4.1 cp

Provede příkaz cp (syntax: cp f1 f2), který zkopíruje soubor f1 do umístění f2.

4.2 mv

Provede příkaz mv (syntax: mv f1 f2), který přesune soubor f1 do umístění f2.

4.3 rm

Provede příkaz rm (syntax: rm f1), smaže soubor f1.

4.4 mkdir

Provede příkaz mkdir (syntax: mkdir d1), vytvoří složku d1.

4.5 rmdir

Provede příkaz rmdir (syntax: rmdir d1), smaže složku d1.

4.6 ls

Provede příkaz ls (syntax: ls d1), vypíše obsah složky d1.

4.7 cat

Provede příkaz cat (syntax: cat f1), vypíše obsah souboru f1.

4.8 cd

Provede příkaz cd (syntax: cd d1), přepne uživatele do složky d1.

4.9 pwd

Provede příkaz pwd (syntax: pwd), vypíše aktuální adresář.

4.10 info

Provede příkaz info (syntax: info d1/f1), vypíše informace o souboru-/složce.

4.11 incp

Provede příkaz incp (syntax: incp f1 f2), zkopíruje soubor f1 do virtuální souborového systému pod názvem f2.

4.12 outcp

Provede příkaz outcp (syntax: outcp f1 f2), zkopíruje soubor f1 z virtuálního souborového systému do reálného souborového systému pod názvem f2.

4.13 load

Provede příkaz load (syntax: load f1), sekvenčně provede příkazy ze souboru f1.

4.14 format

Provede příkaz format (syntax: format xMB), jež připraví datový soubor o velikosti x MB.

4.15 defrag

Provede příkaz defrag (syntax: defrag), jež provede defragmentaci souborového systému.

4.16 exit

Provede příkaz exit (syntax: exit), uzavření datového souboru a ukončení programu.

5 Závěr

5.1 Funkčnost programu a jeho přínos

Funkčnost programu jsem ověřil a pracuje dle očekávání. Během testování jsem využíval nástroj Valgrind, který při zadávání očekávaných příkazů do programu nehlásil žádné chyby a pravděpodobnost úniku paměti je tedy minimální. Díky tvorbě semestrální práce jsem si ujasnil, jak pracuje souborový systém založený na i-uzlech.

5.2 Problémy spojené s tvorbou programu

Během analýzy a následného programování úlohy jsem narazil na několik problémů, které mě při tvorbě programu značně zdržely. Například jsem se téměř čtyři dny zabýval defragmentací disku. Na vině byl prostý překlep - záměna indexu i a j ve vnořeném for cyklu. Mezi další dlouho řešené problémy řadím i problémy související s uvolňováním paměti. Jelikož jsem se jazyku C poměrně dlouho nevěnoval, byly pro mě občasné potíže s uvolňováním paměti běžné. Postupem času jsem však veškeré problémy s pamětí vyřešil pomocí programu Valgrind, který mi pomohl s odhalením místa v programu, kde k problémům s pamětí docházelo.

5.3 Zhodnocení zadání

Zadání úlohy se mi zpočátku zdálo velice triviální, ale nakonec se ukázalo, že naprogramování této úlohy vyžaduje nastudování několika algoritmů, z nichž některé jsou, minimálně pro začátečníka v jazyce C, poměrně náročné na implementaci. Po prostudování algoritmů a vyřešení problémů zmíněných v předchozí kapitole se mi však úlohu podařilo vyřešit.