Semestrální práce z předmětu KIV/ZOS

Práce s pseudo FAT tabulkou

Zdeněk Valeš - A13B0458P - valesz@students.zcu.cz

13.12. 2016

1 Zadání

V jazyce C/C++ vytvořte program pro obsluhu pseudo FAT tabulky. Program musí být schopný přidávat a mazat soubory i adresáře, vypisovat použité clustery u souboru, obsah souboru a vypsat adresářovou strukturu.

Program musí být schopný najít a opravit vadné clustery. Vadný cluster bude mít na konci a začátku sekvenci `0xFF 0xFF 0xFF`.

2 Formát souboru s FAT

Tabulka se skládá ze tří částí - popis FAT, samotná FAT tabulka její kopie a datová sekce, kde jsou uložené clustery.

2.1 Popis FAT

Následující tabulka popisuje první část FS. Reálně tato část zabírá 272 bytů, kvůli zarovnání.

Datový typ	délka (B)	Popis
char	250	Slovní popis souborového systému.
int8_t	1	Typ FAT tabulky.
int16_t	2	Velikost jednoho clusteru v bytech.
int32_t	4	Počet clusterů použitelných pro data.
char	9	Login autora FS.

2.2 FAT a její kopie

Délka této části vychází z údajů z předchozí sekce, závisí na počtu clusterů (=počet záznamů) a počtu kopií. Každý záznam ve FAT je typovaný na `int32_t`.

Následující tabulka zobrazuje možné hodnoty, které může položka ve FAT nabývat:

Hodnota	Popis
0x7FFF FFFe	Volný cluster
0x7FFF FFFd	Konec souboru - poslední cluster v souboru.
0x7FFF FFFc	Vadný cluster.
0x7FFF FFFb	Cluster obsahuje adresář.

2.3 Datová sekce

V této části se nachází datové clustery. Pokud je cluster adresář, bude obsahovat seznam `Directory` struktur, které tvoří položky v adresáři. Počet položek v adresáři je omezen velikostí clusteru.

Následující tabulka popisuje strukturu `Directory`.

Datový typ	délka (B)	Popis
char	13	Jméno souboru.
bool	1	Položka je soubor (true), nebo adresář (false).
int32_t	4	Velikost souboru. V případě adresáře 0.
int32_t	4	Počáteční cluster.

3 Popis použití programu

Program lze přeložit a sestavit automatizačním nástrojem `soncs` z kořenového adresáře projektu.

Obecná forma spuštění příkazu je `./zos-fat cesta/k/fat prikaz [p1] [p2]`. Následující tabulka popisuje možné příkazy a jejich parametry.

Příkaz	Popis
-a	`p1` je plná cesta ke zdrojovému souboru a `p2` je plné jméno cílového souboru ve FAT. Adresářová cesta ve FAT musí existovat, jméno souboru v cílovém adresáři musí být unikátní.
-f	Smaže soubor `p1`. `p1` musí být plná cesta ve FAT.
-c	Vypíše čísla clusterů souboru `p1`. `p1` musí být plná cesta souboru ve FAT.
-m	Vytvoří adresář se jménem `p1` v cestě `p2`. Adresářová cesta `p2` musí být zakončena znakem '/'.
-r	Smaže adresář daný cestou `p1`. Adresářová cesta `p1` musí být zakončena znakem '/' a adresář musí být prázdný.
-l	Vypíše obsah souboru `p1`. `p1` musí být plná cesta ve FAT.
-p	Vypíše celou adresářovou strukturu.
-v	Projde FAT a opraví vadné clustery.

3.1 Příklady

Smazání souboru: `./zos-fat fat.fat -f /msg.txt`
Smazání adresáře: `./zos-fat fat.fat -r /direct-1/`
Výpis souboru: `./zos-fat fat.fat -l /pohadka.txt`
Vytvoření adresáře: `./zos-fat fat.fat -m new-dir /direct-1/`
Nahrání souboru: `./zos-fat fat.fat /tmp/soubor.txt /direct-1/soubor.txt`

4 Popis řešení

Běh programu je rozdělen do tří částí. V první části dojde k načtení parametrů a přečtení FAT tabulky, v druhé části k vykonání příkazů a ve třetí části k výpisu výsledku.

Implementace obsluhy FAT se nachází v souborech `fat.c` a `fat.h`. Implementace příkazů v souborech `commands.c` a `commands.h`.

Několik málo funkcí je otestováno v souborech `tests.c` a `tests.h`. Testování může být zapnuto přepínačem `RUN_TESTS` v `main.c`. Celková funkcionalita je pak otestována

skriptem `test.sh` v adresáři `testings`. Pro spuštění tohoto testu je potřeba nastavit ve skriptu správnou cestu k spustitelnému souboru.

4.1 Přidání souboru

Při nahrávání nového souboru je nejdříve zkontrolována platnost cest (zdrojové a cílové). V jednom adresáři nemohou být dva soubory se stejným jménem. Pak následuje kontrola velikosti. Pokud je nahrávaný soubor větší, než místo dané počtem volných clusterů, není možné jej do FAT nahrát.

Samotné čtení ze souboru a nahrávání dat do FAT je řešeno pomocí schématu producent-konzument. Kde hlavní vlákno je producent, který čte data ze zdrojového souboru a vedlejší vlákno je konzument, který tato data zapisuje do souboru s FAT.

Po přečtení zdrojového souboru se uloží nový `Directory` záznam a provede se update FAT tabulky.

4.2 Smazání souboru

Před smazáním souboru je nejdříve ověřena jeho existence ve FAT. Pokud soubor existuje, všechny jeho clustery se označí jako nepoužité a popisovač souboru (`Directory`) bude z FAT vymazán. Tímto způsobem není třeba procházet celou FAT a přepisovat clustery.

4.3 Výpis clusterů

Před výpisem čísel clusterů souboru je nejdříve ověřena jeho existence ve FAT. Pokud soubor existuje, program projde FAT tabulku a bude postupně vypisovat čísla clusterů.

Funkce vypisování clusterů funguje pouze pro soubory. Pro adresář vrátí `PATH_NOT_FOUND`.

4.4 Vložení adresáře

Před vložením adresáře je nejdříve ověřena existence cílové cesty (kde bude adresář vytvořen). Pak program zkontroluje, jestli v cílovém adresáři už neexistuje adresář s tímto jménem. Soubor se stejným jménem existovat může.

Po těchto kontrolách je do clusteru s rodičovským adresářem (na 1. volné místo) uložen nový `Directory` záznam a provede se update FAT tabulky.

4.5 Smazání adresáře

Před smazáním adresáře proběhne ověření jeho existence a kontrola, jestli je adresář prázdný. Pokud existuje a je prázdný, `Directory` záznam je smazán a provede se update FAT tabulky.

4.6 Výpis obsahu souboru

Před výpisem obsahu souboru je nejdříve ověřena jeho existence. Pokud soubor existuje, je postupně vypsán na konzoli.

Výpis na konzoli je paralelizován pomocí schématu producent-konzument, kdy hlavní vlákno je producent, který čte ze souboru do bufferu a vedlejší vlákno je konzument, který bere data z bufferu a vypisuje je na konzoli. Do bufferu se ukládá i pořadové číslo načteného bloku dat, takže výpis pak není přeházený.

4.7 Výpis adresářové struktury

Výpis adresářové struktury je řešen rekurzivním algoritmem. Vždy na začátku iterace proběhne nahrání souborů v současném adresáři. Pak následuje jejich postupný výpis a pokud se narazí na adresář, dojde k rekurzi a proces se opakuje.

4.8 Oprava bad blocků

Oprava chybných clusterů má dvě fáze. V první fázi se projdou všechny clustery označené jako volné. Pokud je detekován cluster špatný, označí se ve FAT jako chybný a data v něm budou ponechána beze změny.

V druhé fázi je procházen adresářový strom. Root není kontrolován - předpokládá se, že root je ve fat vždy na 0. místě a proto jej nelze přesunout. Adresářový strom je opět procházen rekurzivně. V případě, že je detekován chybný adresář, dojde k přesunu celého clusteru a změně ve FAT. Přesunutý cluster bude mít změněn poslední byte na `0`, aby nebyl při dalších průchodech znovu detekovaný.

Clustery u souborů jsou procházeny podobně jako spojové seznamy. Pokud je detekován špatný cluster uvnitř souboru, je pouze přesunut do jiného. Pokud je detekován špatný první cluster souboru, dojde k přesunu a updatu `Directory` záznamu. Přesunutý cluster bude mít opět změněn poslední byte na `0`.

5 Závěr

Program pracuje správně a dokáže obsloužit jednoduchou FAT tabulku. Z práce je patrná jednoduchost FAT oproti modernějším file systémům (EXT3, NTFS).

Program by se dal na některých místech více paralelizovat (oprava bad blocků), nicméně by to vyžadovalo držení větší části FS v paměti a i přes to by spousta operací zůstala sekvenčních.

Další z možných vylepšení je nahrazení rekurzivních algoritmů. V tomto případě se sice jedné o jednoduché řešení, ale na slabších systémech by mohlo snadno dojít k přetečení zásobníku.