Zjednodušený souborový systém založený na pseudoFAT

Autor: Milan Vlachovský

Obsah

1. Úvod

Cílem tohoto projektu bylo vytvořit zjednodušený soubový systém, který funguje na pricipech známého souborového systému FAT. Jedná se pouze o simulaci souborového systému vytvořenou, v rámci univerzitního předmětu, pro studijní účely.

Jako programovací jazyk bylo potřeba zvolit jazyk nižší úrovně, ve kterém je možné manipulovat přímo s pamětí programu. Byl zvolen jazyk Go díky jeho efektivní správě paměti, podpoře paralelismu a relativně nízké úrovni abstrakce ve srovnání s jinými moderními jazyky.

2. Popis programu

Program implementuje zjednodušený souborový systém založený na principech FAT (File Allocation Table). Cílem je umožnit správu souborů a adresářů v rámci virtuálního disku, který je uložen jako binární soubor. Program poskytuje základní operace, jako je vytváření, mazání a přesouvání souborů, práce s adresáři a načítání či ukládání dat.

- Podporované příkazy:
    - `format <velikost>` – naformátování souborového systému na disk o zadané velikosti (se smazáním všech dat)
    - `mkdir a1` – vytvoření adresáře
    - `rmdir a1` – smazání prázdného adresáře
    - `cd a1` – změna aktuálního adresáře
    - `pwd` – zobrazení aktuální cesty
    - `incp s1 s2` – nahrání souboru z disku do souborového systému
    - `outcp s1 s2` – export souboru ze souborového systému na disk
    - `cp s1 s2` – kopírování souboru
    - `mv s1 s2` – přesunutí nebo přejmenování souboru
    - `rm s1` – smazání souboru
    - `ls a1` – výpis obsahu adresáře
    - `cat s1` – zobrazení obsahu souboru
    - `info s1/a1` – informace o souboru (v jakých clusterech se nachází)
    - `load s1` – vykonání příkazů ze souboru
    - `check` – kontrola konzistence souborového systému
    - `bug s1` – záměrné poškození souboru (pro simulaci chyby)
    - `exit` – ukončení programu

> Ve všech případech představují `s1`, `s2` a `a1` cesty k souborům nebo adresářům ve virtuálním souborovém systému.

Program vytváří virtuální disk ve formě binárního souboru, do kterého data ukládá a ze kterého je načítá. Při zadání neplatného příkazu (neznámý příkaz, chybné argumenty, ...) program vypíše chybovou hlášku a pokračuje ve svém běhu. Program omezuje velikost disku pouze na 4 GiB (z praktických důvodů).

3. Architektura systému

Hlavní struktury

Souborový systém je strukturován do několika klíčových částí:

- Hlavní metadata souborového systému:
    - Struktura (`struct`) FileSystem
    - Obsahuje informace definující rozložení souborového systému na disku.
    - Struktura je 31 bytů dlouhá a obsahuje následující informace:
        - velikost disku,
        - počet FAT tabulek,
        - počáteční adresy FAT tabulek,
        - počáteční adresa datové oblasti,
        - velikost clusteru,
        - identifikátor autora souborového systému.

```
// FileSystem is a struct representing the pseudo FAT file system. It
is 31 bytes long.
//
// WARNING: The variables are ordered in a way that they are aligned
in memory with the
// smallest possible padding. This is important for the byte handling
in the loader.go.
type FileSystem struct {
  // DiskSize is the size of the disk in bytes
  DiskSize uint32
  // FatCount is the number of records in the FAT
  FatCount uint32
  // Fat01StartAddr is the start address of the first FAT
  Fat01StartAddr uint32
  // Fat02StartAddr is the start address of the second FAT
  Fat02StartAddr uint32
  // DataStartAddr is the start address of the data region
  DataStartAddr uint32
  // ClusterSize is the size of a cluster in bytes
  ClusterSize uint16
  // Signature is the ID of the author of the file system
  Signature [consts.StudentNumLen]byte
}
```

> Všech velikosti proměnných byly zvoleny tak, aby bylo ušetřeno co nejvíce místa na disku vzhledem k maximální velikosti souborového systému. Velikost clusteru byla zvolena na základě průměrné velikosti souboru v běžném uživatelském prostředí.

- FAT tabulky:
    - Jedná se o dvourozměrné pole int32 (standardně používá systém FAT 2 tabulky) s délkou odpovídající počtu clusterů.
    - Používá se k mapování alokovaných a volných clusterů na virtuálním disku do datové oblasti.
    - Speciální hodnoty označují nevyužité bloky (`FatFree int32 = -1`), konec souboru (`FatFileEnd int32 = -2`) a poškozené bloky (`FatBadCluster int32 = -3`).
- Datová oblast:
    - Jedná se o byte pole délky alokovatelého prostoru na disku (počet clusterů * velikost clusteru).
    - Obsahuje metadata samotných souborů a adresářů včetně jejich obsahu (jedná-li se o soubory).
    - Každý soubor či adresář si drží v počátečním clusteru referenci sám na sebe (ve formě `DirectoryEntry` struktury).
    - Následně:
        - je-li soubor, další clustery obsahují data souboru;
        - je-li adresář, další clustery obsahují reference na soubory a adresáře v něm obsažené.
    - Do dat z datové oblasti je přistupováno pomocí FAT tabulek:
        - Offset začátku jednotlivých záznamů v poli je vypočítán jako `(clusterIndex * ClusterSize)`.

Vedlejší struktury

Další důležité struktury:

- Záznam (`DirectoryEntry`) souboru nebo adresáře:
    - Jedná se o strukturu o velikosti 20 bytů, která obsahuje metadata souboru nebo adresáře.
    - Struktura obsahuje následující informace:
        - název souboru nebo adresáře,
        - příznak, zda se jedná o soubor,
        - velikost souboru,
        - startovací cluster souboru,
        - startovací cluster rodičovského adresáře.

```
// DirectoryEntry is a struct representing an item in a directory. It
is 20 bytes long.
type DirectoryEntry struct {
  // Name is the name of the file or directory
  Name [consts.MaxFileNameLength]byte
  // IsFile is a flag indicating if the item is a file
  IsFile bool
  // Size is the size of the file in bytes
  Size uint32
  // StartCluster is the start cluster of the file
  StartCluster uint32
  // ParentCluster is the start cluster of the parent directory
  ParentCluster uint32
}
```

> Každý DirectoryEntry záznam zabírá 1 cluster. Bylo tak zvoleno na základě standardu FAT. Znamená to, že pokud by souborový systém obsahoval převážně adresáře nebo velmi malé soubory, docházelo by k neefektivnímu využití místa na disku. Omezení je ale pro účely této práce dostačující.

> Kořenový adresář je reprezentován jako záznam s názvem `/` a startovacím clusterem 0. Jeho rodičovský cluster odkažuje na sebe samého. V programu jsou nastaveny omezení, aby kořenový adresář nemohl být smazán ani jinak modifikován.

Princip fungování soubového systému

Souborový systém funguje na principu alokace souborů a adresářů v rámci pevně definovaných clusterů, které jsou propojeny pomocí FAT tabulky. Každý soubor nebo adresář je uložen v jednom nebo více clusterech, jejichž pořadí je udržováno právě v této tabulce.

1. Vytváření souboru/adresáře

Proces vytvoření souboru probíhá v několika krocích:

- Vyhledání volného místa v nadřazeném adresáři.
  - Systém zpracuje zadanou cestu a pokusí se získat nadřazený adresář pro cílový záznam (pokud je cesta validní).
  - Systém načte adresářovou strukturu nadřazeného adresáře a zkontroluje, zda název záznamu v seznamu potomků již neexistuje.
  - Pokud neexistuje, je přidán nový záznam do seznamu potomků (včetně aktualizace FAT tabulky).
- Alokace prvního clusteru pro samotný soubor/adresář.
  - Systém prohledá FAT tabulku a nalezne první volný cluster.
  - Tento cluster je zapsán jako počáteční (`StartCluster`) v DirectoryEntry.
- Uložení dat souboru.
  - Pokud soubor zabírá více než jeden cluster, je každý další cluster propojen v FAT tabulce s předchozím.
  - Poslední cluster v řetězci obsahuje značku `FatFileEnd`.

2. Načítání souboru/adresáře

Proces načítání souboru/adresáře probíhá následovně:

- Vyhledání souboru/adresáře.
  - Systém se pokusí najít záznam na základě poskytnuté cesty.
  - Čtení obsahu přes FAT tabulku
    - Počáteční cluster (`StartCluster`) se použije jako výchozí bod.
    - Z prvního clusteru se načte `DirectoryEntry` představující soubor/adresář.
    - Jakékoliv další clustery představují potomky adresáře nebo data souboru.
    - Systém prochází FAT tabulku a načítá data ze všech přidělených clusterů, dokud nenarazí na `FatFileEnd`.

3. Mazání souboru/adresáře

Při mazání souboru je třeba provést následující kroky:

- Vyhledání souboru/adresáře:
  - Systém analyzuje cestu a nalezne odpovídající záznam `DirectoryEntry` v nadřazeném adresáři.
  - Pokud se jedná o adresář, je nutné zkontrolovat, zda není prázdný (neobsahuje žádné soubory nebo podadresáře).
- Odstranění záznamu z nadřazeného adresáře:
  - `DirectoryEntry` odpovídající souboru nebo adresáři je odstraněn z nadřazeného adresáře (včetně aktualizace FAT tabulky - zkrácení řetězce clusterů o jeden cluster).
  - Oblast záznamu z datové sekce je vynulována.
- Uvolnění clusterů:
  - Každý cluster, který soubor nebo adresář využívá, je označen jako `FatFree` ve FAT tabulce.
  - Oblast záznamu z datové sekce, která podle FAT tabulky patří k souboru/adresáři, je vynulována.

> Při mazání adresáře dochází k vytváření volných míst v datové oblasti a FAT tabulkách. Následné vkládání nových větších záznamů způsobuje fragmentaci souborového systému. S fragmentací se však v systémech FAT počítá a je běžným jevem. V reálném souborovém systému by bylo nutné implementovat mechanismus defragmentace, který by zajišťoval, že soubory jsou uloženy v paměti co nejefektivněji.

4. Přesun souboru/adresáře

Při přesunu souboru nebo adresáře dochází k následujícím krokům:

- Vyhledání zdrojového záznamu:
  - Systém nalezne odpovídající `DirectoryEntry` souboru nebo adresáře.
- Zkontrolování existence cílového umístění:
  - Systém se ujistí, že cílová cesta existuje a že již neobsahuje soubor/adresář se stejným jménem.
- Vytvoření nového záznamu v cílovém adresáři:
  - Do cílového adresáře se přidá nový DirectoryEntry, který odkazuje na stejná data jako původní záznam.
- Odstranění starého záznamu:
  - Původní DirectoryEntry je odstraněn ze zdrojového adresáře.
  - FAT tabulka je aktualizována:
    - Odebrání clusteru z nadřazeného adresáře.
    - Přidání clusteru do cílového adresáře.
    - Clusterové řetězce, které obsahovaly data souboru/adresáře, zůstávají nezměněny.

5. Kopírování souboru/adresáře

Používá principy vytváření nového záznamu. Dochází pouze ke kopírování obsahu souboru/adresáře do nového umístění a změně počátečních clusterů (podle nalezeného volného místa).

4. Návod na zprovoznění

Požadavky

Pro sestavení a spuštění projektu je třeba mít nainstalováno následující:

- Go 1.22 nebo novější
- make
    - > Na Windows je možné použít například make z chocolatey, či jiné alternativy.

Sestavení projektu

Pro sestavení celého projektu byly vytvořeny soubory *Makefile* a *Makefile.win*, které obsahují instrukce pro sestavení projektu na Unixových a Windows OS. Pro sestavení projektu na Unixových OS stačí spustit z kořenové složky projektu příkaz:

```
make
```

a pro Windows OS stačí spustit příkaz:

```
make -f Makefile.win
```

Skript sestaví spustitelný soubor ve složce *bin/*. Spustitelný soubor je pojmenován *myfs*, případně na Windows *myfs.exe*.

Spuštění programu

Program lze spustit s následujícím příkazem:

```
./bin/myfs <cesta k virtuálnímu disku>
```

> Soubory kopírované do virtuálního disku a naopak jsou vždy kopírovány relativně ke složce, ve které je spuštěn program.

Manuální spuštění pomocí Go

Jako alternativu k sestavení projektu je možné spustit projekt přímo pomocí Go. Je potřeba ho spouštět ze složky *src/*. Program lze spustit pomocí následujícího příkazu:

```
go run main.go <cesta k virtuálnímu disku>
```

5. Struktura projektu

Projekt je rozdělen následovně:

- *Kořenová složka* — Obsahuje soubory pro sestavení projektu, složku *src/* obsahující zdrojové kódy projektu, složku *bin/* pro výstupní soubory a složku *docs/* s dokumentací.

Kořenová složka

- *Makefile* — Soubor pro sestavení projektu na Unix OS.
- *Makefile.win* — Soubor pro sestavení projektu na Windows OS.
- *docs/* — Složka obsahující dokumentaci.
    - *docs/doc.md* a *docs/doc.pdf* — Tento dokument ve formátu Markdown a PDF.
    - *docs/client_ref.html* — Odkaz na dokumentaci klientské části.
    - *docs/server_ref.html* — Odkaz na dokumentaci serverové části.
- *src/* — Složka obsahující zdrojové kódy projektu.
    - *src/arg_parser/arg_parser.go* — Modul pro zpracování argumentů příkazové řádky.
    - *src/cmd/* — Složka obsahující moduly pro zpracování příkazů.
        - *src/cmd/command.go* — Definice struktury příkazu.
        - *src/cmd/command_parser.go* — Modul pro parsování příkazu.
        - *src/cmd/command_validator.go* — Modul pro validaci příkazu.
        - *src/cmd/command_executor.go* — Modul pro vykonání příkazu.
    - *src/consts/* — Složka obsahující definované konstanty používané v projektu.
        - *src/consts/cmds.go* — Seznam podporovaných příkazů v souborovém systému.
        - *src/consts/exit_codes.go* — Definice návratových kódů programu.
        - *src/consts/fat_flags.go* — Konstanty určující speciální hodnoty v FAT tabulce (např. `FatFileEnd`, `FatFree`).
        - *src/consts/formats.go* — Definice všech jednotek a podporovaných znaků/symbolů.
        - *src/consts/limits.go* — Definované limity souborového systému (např. maximální velikost názvu souboru, ...).
        - *src/consts/msg.go* — Obsahuje textové zprávy používané pro výstupy.
    - *src/custom_errors/errors.go* — Definuje vlastní chybové typy a konstanty chybových hlášení používané v projektu.
    - *src/logging/logging.go* — Modul pro správu logování, umožňuje zapisovat zprávy různých úrovní (INFO, WARNING, ERROR, ...).
    - *src/pseudo_fat/structures.go* — Definice základních datových struktur souborového systému, včetně `FileSystem` a `DirectoryEntry`.
    - *src/utils/* — Složka obsahující pomocné utility pro práci se souborovým systémem.

        - *src/utils/data_transform.go* — Nástroje pro konverzi dat mezi různými formáty, například serializace a deserializace binárních struktur.
        - *src/utils/loader.go* — Modul pro načítání souborového systému do binárního souboru.
        - *src/utils/path.go* — Nástroje pro zpracování cest k souborům a adresářům.
        - *src/utils/pretty_print.go* — Modul pro formátování výstupu a přehledné zobrazování informací pro určité struktury.
        - *src/utils/pseudo_fat_fs_operations.go* — Implementace algorithmů pro práci se souborovým systémem.
            - **Zde se nachází implementace všech podporovaných příkazů, jako je vytváření, mazání, kopírování, přesunování souborů a adresářů, práce s adresáři, načítání a ukládání dat, ...**

    - *src/main.go* — Hlavní soubor projektu, který inicializuje souborový systém, načítá příkazy a zajišťuje jejich vykonání.

    - *src/test.cmds*, *src/testh.cmds*, *src/testr.cmds* — Soubory obsahující testovací sady příkazů pro ověření správné funkčnosti implementace souborového systému (pomocí příkazu `load`).

6. Popis implementace

Do popisu implementace budou převážně zahrnuty jen ty nejdůležitější části kódu potřebné pro pochopení principu fungování aplikace. Projekt je strukturován tak, aby umožňoval efektivní zpracování příkazů a manipulaci se souborovým systémem.

Hlavní komponenty a jejich role

Projekt je rozdělen do několika klíčových modulů, které spolu úzce spolupracují:

- *main.go* — hlavní vstupní bod aplikace, inicializuje souborový systém, spouští vlákna pro zpracování příkazů a spravuje ukončení programu.
- *cmd/* — obsahuje logiku pro parsování, validaci a vykonávání příkazů.
- *pseudo_fat/* — definice pseudo FAT souborového systému, včetně základních datových struktur.
- *utils/* — pomocné funkce pro práci s cestami, konverze dat a **správa souborového systému**.
- logging/ — zajišťuje logování událostí v systému.

Průběh vykonání příkazu

1. Inicializace aplikace (`main.go`)

    - Program začíná zpracováním argumentů příkazové řádky.
        - `arg_parser.GetFilenameFromArgs()`
    - Ověří, zda existuje soubor reprezentující souborový systém. Případně vytvoří nový.
        - `getFileFromPath()`.
    - Poté se pokusí načíst souborový systém do paměti (případně neinicializovaný souborový systém, pokud v souboru nebyly žádné data).
        - `utils.GetFileSystem()`.
    - Vytvoří se gorutiny pro čtení příkazů (`acceptCmds()`) a jejich vykonávání (`interpretCmds()`).

- Program čeká ve funkci (`handleProgramTermination()`) dokud není ukončen uživatelem (příkaz `exit` nebo poslání signálu `SIGINT`).

2. Čtení příkazů (`acceptCmds()`)

   - Gorutina `acceptCmds()` nepřetržitě čte vstup uživatele (`bufio.Scanner`).
   - Každý vstup je parsován (`cmd.ParseCommand()`).
     - Pokud je příkaz validní (`cmd.ValidateCommand()`), je odeslán do kanálu `cmdBufferChan`.

3. Vykonání příkazu (`interpretCmds()`)

   - Gorutina `interpretCmds()` čte příkazy z `cmdBufferChan`.
   - Každý příkaz je předán k vykonání (`cmd.ExecuteCommand()`).
     - `cmd.ExecuteCommand()` zpracuje příkaz, vykoná jednodušší operaci (změna aktuálního adresáře, výpis adresáře, ...) nebo zavolá příslušnou funkci z *utils/pseudo_fat_fs_operations.go* pro provedení operace nad souborovým systémem.
     - V případě chyby se vypíše odpovídající chybová hláška (*custom_errors*).
     - Pokud byl příkaz úspěšný, případné změny se uloží do souborového systému (`utils.WriteFileSystem`).

Implementace samotných operací nad souborovým systémem se nachází v souboru *utils/pseudo_fat_fs_operations.go*. Vetšina funkcí je nazvána výstižně podle příkazu, který implementuje (např. `Mkdir()`, `Rmdir()`, `CopyInsideFS()`, ...).

> V produkčním světě by bylo potřeba funkce z tohoto souboru více dekomponovat, ale pro rychlou navigaci čtenáře skrze operace, které bylo nutné vykonat nad souborovým systémem, byly fuknce ponechány v méně modulární podobě (aby čtenář nemusel přeskakovat mezi soubory/řádky do různých funkcí).

Chyba při kopírování souborových dat do datového regionu

Při kontrole práce zadavatelem byla nalezena chyba zápisu souboru do souborového systému. Došlo k chybě při kopírování dat do paměti. Konkrétně při použití funkce jazyka `copy()` pro přenos dat ze zdrojového pole (`fileDataRef`) do datového regionu (`dataRef`).

Problém spočíval v tom, že při určování rozsahu dat ke kopírování nebyla nastavena limitní hodnota. To vedlo k situaci, kdy Go kopírovalo více dat, než bylo požadováno a přepisovalo clustery, které nepatřily k souboru. Tím docházelo k poškození souborového systému.

Kód před opravou:

```
// write the file data to the filesystem
prevIndex := freeClusterIndex
for i, clusterIndex := range freeClusterIndicesData {
  addToFat(fatsRef, prevIndex, clusterIndex)
  byteOffset = int(clusterIndex) * int(pFs.ClusterSize)
  copy(dataRef[byteOffset:], fileDataRef[i*int(pFs.ClusterSize):])
  prevIndex = clusterIndex
}
```

Kód po opravě:

```
// write the file data to the filesystem
bytesRemaining := len(fileDataRef)
prevIndex := freeClusterIndex
for i, clusterIndex := range freeClusterIndicesData {
  addToFat(fatsRef, prevIndex, clusterIndex)
  byteOffset = int(clusterIndex) * int(pFs.ClusterSize)
  fileDataRefStartOffset := i * int(pFs.ClusterSize)
  fileDataRefEndOffset :=
    min((i+1)*int(pFs.ClusterSize),
        fileDataRefStartOffset+bytesRemaining
       )
  currentSourceBytes :=
    fileDataRef[fileDataRefStartOffset:fileDataRefEndOffset]
  copiedBytesCount := copy(dataRef[byteOffset:], currentSourceBytes)
  bytesRemaining -= copiedBytesCount
  prevIndex = clusterIndex
}
```

Tato chyba je analogická k problému známému z jazyka C, kdy se používá strcpy() namísto bezpečnější varianty strncpy(), což může vést k přetečení bufferu.

7. Testování

Testování projektu bylo provedeno manuálně uživatelským testováním. Žádné automatické testy nebyly implementovány. Testování bylo zaměřeno na ověření správné funkčnosti základních operací souborového systému, jako je vytváření, mazání, kopírování a přesouvání souborů a adresářů. Dále byla testována správná práce s cestami, validace vstupu a detekce chyb.

8. Závěr

Tento projekt implementuje zjednodušený souborový systém založený na principu FAT (File Allocation Table), přičemž využívá jazyk Go pro efektivní správu paměti a manipulaci s binárními daty. Cílem bylo vytvořit funkční a srozumitelnou simulaci souborového systému, která umožňuje uživateli provádět základní operace nad virtuálním diskem, jako je vytváření, mazání, přesouvání a kopírování souborů a adresářů.

Při návrhu a implementaci bylo třeba řešit několik klíčových výzev, mezi které patřilo:

- Správa clusterů a FAT tabulek, včetně správného propojení řetězců clusterů pro soubory a adresáře.
- Efektivní práce s binárními daty, včetně serializace a deserializace struktur souborového systému.
- Zajištění integrity souborového systému a detekce chyb, například při manipulaci se soubory a jejich datovými bloky.
- Validace a zpracování cest, což zahrnovalo podporu jak relativních, tak absolutních cest.

Projekt je navržen modulárně, což umožňuje snadné rozšíření a přizpůsobení. Struktura kódu je přehledně rozdělena mezi moduly pro práci s příkazy, souborovým systémem, cestami a pomocnými funkcemi, což zajišťuje dobrou čitelnost a udržovatelnost.

Shrnutí dosažených cílů:

- Základní operace souborového systému: Implementovány běžné operace jako mkdir, rm, mv, cp, ls, pwd, info a další.
- Práce s virtuálním diskem: Umožňuje formátování souborového systému, správu souborů a adresářů.
- Ošetření běžných chyb: Validace vstupu, kontrola konzistence souborového systému (check), detekce a simulace chyb (bug).
- Testování a ladění: Byly odhaleny a opraveny chyby, například problém s přepisováním paměti při kopírování souborů.
- Přehledná a udržovatelná architektura: Modulární rozdělení zdrojového kódu, dobře definované rozhraní mezi komponentami.

Projekt je připraven na další možná rozšíření jako je implementace defragmentace, implementace dalších typů souborů jako jsou například symbolické odkazy, a další.

Projekt splnil svůj hlavní účel – implementovat funkční souborový systém a umožnit jeho efektivní správu. Přestože se jedná pouze o simulaci souborového systému, při jeho návrhu a implementaci bylo nutné řešit řadu reálných problémů, které jsou běžné v oblasti správy souborů a dat. Tento projekt poskytl cenné zkušenosti s návrhem a implementací nízkoúrovňového souborového systému.