# Filesystem založený na I-uzlech

**Ondřej Vaic**

A17B0385P
ondra.vaic@gmail.com

Seminární práce pro předmět
KIV/ZOS

Fakulta aplikovaných věd
Západočeská Univerzita
28/01/2020

# Semestrální práce ZOS 2019 (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

```
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:

- Defragmentace (defrag) – pokud login studenta začíná **a-i**
  Datové bloky budou organizovány tak, že nejprve budou obsazené a následně volné (předpokládáme dostatek volného místa – minimálně ve velikosti největšího souboru).

- 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 (10. února 2020).
- A samozřejmě je třeba mít zápočet dříve, než půjdete na zkoušku (alespoň 1 den předem).

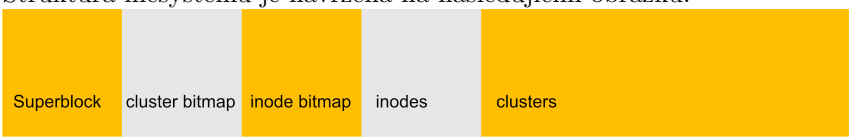
## Hodnocení

Při kontrole semestrální práce bude hodnocena:

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

# 1 Struktura filesystému

Struktura filesystému je navržena na následujícím obrázku.




# 2 Datové struktury

Všechny níže uvedené třídy je možné uložit do filesystému, a následně je z něj načíst. Každou třídu je možné uložit pomocí:

```
void Save(const string& fileSystemName, int32_t offset);
```

`fileSystemName` je jméno souboru který simuluje filesystém, a `offset` je adresa v bytech. Každou třídy je možné načíst z disku a vytvořit její instanci pomocí zavolání jejího konstruktoru se stejnými parametry jako `Save`.

## 2.1 SuperBlock

Struct `SuperBlock` uchovává hlavní data o filesystému. Je uložena od adresy 0. Jsou v ní data která popisují filesystém. Jsou v ní uložena tyto data.

```
int32_t disk_size;                          //celkova velikost VFS
int32_t cluster_size;                       //velikost clusteru
int32_t cluster_count;                      //pocet clusteru
int32_t inode_count;                        //pocet clusteru
int32_t cluster_bitmap_start_address;       //adresa pocatku bitmapy
datovych bloku

int32_t inode_bitmap_start_address;         //adresa pocatku bitmapy i-uzlu
int32_t inode_start_address;                //adresa pocatku i-uzlu
int32_t data_start_address;                 //adresa pocatku datovych bloku
```

### 2.1.1 Výpočet superblocku

U některých atributu není z počátku jasné, jak se dopočítají. Dále budou uvedeny výpočty jednotlivých atributů.

Nejdříve je vypočítán počet i-uzlů.

```
int32_t inodeCount = sizeInBytes / FILE_EXPECTED_MIN_SIZE;
```

Poté se vypočítá kolik místa zabere superblock, i-uzly a jejich bitmapa.

```
int32_t metaDataPartSize = sizeof(SuperBlock) + inodeCount *
sizeof(PseudoInode) + inodeCount / 8 + 1;
```

$inodeCount/8 + 1$ se přičítá za bitmapu i-uzlů. Děleno 8 protože adresy jsou počítány v bytech, a tedy 1/8 bytu pro každý i-uzel.

Dále je nutné vypočítát počet clusterů.

```
int32_t clusterCount = ((sizeInBytes - metaDataPartSize) * 8) /
(CLUSTER_SIZE_B * 8 + 1);
```

$(sizeInBytes - metaDataPartSize) * 8$ je velikost zbývajícího místa na clustery a jejich bitmapu v bitech. To celé děleno $(CLUSTER\_SIZE\_B * 8 + 1)$, to je velikost clusteru v bitech + 1 protože pro každý cluster je potřeba ještě 1 bit do bitmapy. Další výpočty jsou už jednoduché.

```
int32_t clusterBitmapStartAddress = sizeof(SuperBlock);
//address is in bytes, so clusterCount / 8, +1 because integer division
int32_t inodeBitmapStartAddress = clusterBitmapStartAddress +
clusterCount / 8 + 1;
int32_t inodeStartAddress = inodeBitmapStartAddress +
inodeCount / 8 + 1;
int32_t dataStartAddress = inodeStartAddress +
inodeCount * sizeof(PseudoInode);
```

## 2.2 Bitmap

Bitmapa je buď bitmapa i-uzlů nebo bitmapa clusterů. Její jediný atribut je `vector<bool> bitmap`. False znamená že je cluster nebo i-node na daném indexu volný. Například adresa ntého clusteru se poté z indexu vypočítá následovně.

```
int32_t address = superBlock.data_start_address + emptyClusterIndex
* CLUSTER_SIZE_B;
```

Bitmapa je ukládána do souboru jako pole bitů. C++ nepodporuje uložení jednotlivých bitů. Nejmenší možná zapsatelná jednotka je byte. Pro uložení je tedy potřeba procházet pole, a vytvářet pomocí binárních operátorů byte - `unsigned char` pro 8 bitů v bitmapě, a ten poté zapsat. Podobně je poté nutné načíst bity z disku. V tomto případě se načte jeden byte, a z něj musí být vyňaty jednotlivé bity. Následuje ukázka uložení bitmap, v té je použita funkce.

```
unsigned char Utils::SetBit(unsigned char num, unsigned char pos)
```

Ta nastaví jeden bit v num na pozici pos na 1.

```
int off = offset;
for (int i = 0; i < bitmap.size(); ++i) {

    if(bitmap[i]){
        byte = Utils::SetBit(byte, (7 - (i % 8)));
    }

    if((i + 1) % 8 == 0){
```

```
            stream.write((char*)&byte, sizeof(byte));
            off++;
            byte = 0;
        }
    }
```

### 2.3 PseudoInode

Struct `PseudoInode` popisuje jeden i-uzel. Jsou v něm následující atributy.

```
int32_t dot;                        //adresa souboru
int32_t dotDot;                     //adresa slozky ve ktere soubor je

bool isDirectory;                   //soubor, nebo adresar
int32_t file_size;                  //velikost souboru v bytech
int32_t direct[NUM_DIRECT_NODES];//primy odkaz na datove bloky
int32_t indirect1;                  //1. neprimy odkaz (odkaz - datove bloky)
int32_t indirect2;                  //2. neprimy odkaz
(odkaz - odkaz - datove bloky)
```

### 2.4 Indirect

Struct `Indirect` reprezentuje jeden nepřímý blok. Jediný atribut je tedy pole přímých odkazů.

```
int32_t direct[NUM_DIRECT_IN_CLUSTER];
```

Velikost pole se vypočítá jednoduše jako velikost clusteru děleno 4, kvůli 4 bytovým adresám.

```
NUM_DIRECT_IN_CLUSTER = CLUSTER_SIZE_B / 4;
```

### 2.5 DirectoryItem

Struct `DirectoryItem` popisuje jeden soubor. Je v něm pouze jeho jméno a adresa jeho i-uzlu.

```
int32_t inode;              //inode odpovidajici souboru
char item_name[12];         //8+3 + /0 C/C++ ukoncovaci string znak
```

## 3 FileSystem

Nejdůležitější třída je `FileSystem`. V ní je téměř všechna logika file systému. V konstruktoru se nejdříve inicializují důležité atributy. Z disku se načte `superBlock`, `root`, `clusterBitmap` a `inodeBitmap` a `currentDir` se inicializuje na `root`. Poté je incializována mapa `commandMap`. V té jsou jednotlivým příkazům (například

*format*) přiřazeny metody které se mají vykonat. Poté je nad instancí `FileSystem` zavolána metoda `Run`. Ta vypadá následovně.

```
while(true){
    cout << getCurrentPathDescriptor() + ":";

    vector<string> commandSplitStrings = Utils::GetCommand();
    if(commandSplitStrings[0] == "x"){
        break;
    }

    if (commandMap.find(commandSplitStrings[0]) == commandMap.end()){
        cout << "Incorrect command" << endl;
        continue;
    }

    bool result = commandMap[commandSplitStrings[0]](commandSplitStrings);
}
```

Tento návrh umožňuje čitelný kód bez zbytečných if statementů nebo switch statementů. Nevýhodou naopak je, že všechny metody musí mít stejnou signaturu.

# 4 Kontrola file systému

File systém je možné zkontrolovat pomocí příkazu `check`. `check` zkontroluje jestli velikost souboru odpovídá tomu kolik clusterů má naalokovaných a jestli jsou všechny soubory v nějaké složce.

Pro rozbití file systému jsou implementovány dvě mětody `corrupt1` a `corrupt2`. `corrupt1` načte i-uzel souboru a nastaví jeho velikost na 165 bytů.

`corrupt2` načte i-uzel souboru, pomocí něj najde složku ve které je soubor uložen. Poté projde všechny odkazy i-uzlu rodičovské složky, a tu položku, která ukazuje na cluster zadaného souboru nastaví na 0 - tedy volný cluster.

Pro kontrolu file systému pomocí `check` jsou nejdříve načteny všechny i-uzly. Každý i-uzel je následně zkontrolován. Kontrola správné velikosti je jednoduchá. Stačí získat počet clusterů, na které i-uzel odkazuje, a poté porovnat:

```
(numClusters != inode.file_size / CLUSTER_SIZE_B + 1)
```

Pokud je tato podmínka true, pak velikost souboru nesedí.

Dále je provedena kontrola toho jestli je inode skutečně ve složce. Pomocí `inode.dotDot` je zjištěno ve které složce se má soubor nacházet. Poté se projdou všechny soubory v této složce, a pokud žádný nemá stejné jméno jako zrovna kontrolovaný soubor, pak nastala chyba.

```
vector<int32_t> fileAddresses = getReferencedClusters(parentDirectory.inode);
for (int32_t fileAddress : fileAddresses) {
    DirectoryItem file(name, fileAddress);
```

```
        string fileName = file.item_name;

        if(fileName == currentFile.item_name){
            parentContainsCurrentFile = true;
        }
    }

    if(!parentContainsCurrentFile){
        cout << "FILE " << currentFile.item_name << " IS IN NO DIRECTORY" <<endl;
    }
```

# 5 Překlad

Seminární práci je možné přeložit pomocí nástroje `Make`. Ve složce src stačí z příkazové řádky spustit příkaz *make*.

# 6 Spuštění

Filesystém se spustí z příkazové řádky příkazem *./Zos [jmeno disku]*. Dále je nutné disk formatovat pomocí *format [velikost v MB]*. Poté je možné provádět základní operace uvedené v zadání. Program se ukončuje příkazem *x*.