INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

VYSOKÁ ŠKOLA BÁŇSKÁ – TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA STROJNÍ

OPERAČNÍ SYSTÉMY

IMPLEMENTACE SYSTÉMU SOUBORŮ

doc. Dr. Ing. Oldřich Kodym

Ostrava 2013

ISBN 978-80-248-3053-7

Tento studijní materiál vznikl za finanční podpory Evropského sociálního fondu (ESF) a rozpočtu České republiky v rámci řešení projektu: CZ.1.07/2.2.00/15.0463, MODERNIZACE VÝUKOVÝCH MATERIÁLŮ A DIDAKTICKÝCH METOD

OBSAH

8. IMPLEMENTACE SYSTÉMU SOUBORŮ

OBSAH KAPITOLY:

Struktura systému souborů, zpřístupňování systému souborů.

Metody alokace prostoru pro soubor.

Management volného prostoru.

Zálohování a archivace.

MOTIVACE:

Chod operačního systému využívá mnoha standardních mechanizmů známých z jiných oblastí řízení i běžného života. Souborový systém je příkladem aplikace metod organizace rozsáhlých hierarchických systémů. Systém souborů je uložen na odkládacím zařízení, jehož hlavním úkolem je neustále uchovávat velké množství dat. Nyní si vysvětlíme aspekty týkající se uchovávání a přístupu k souborům na zatím nejpoužívanějším mediu, magnetickém disku. Ukážeme si postupy, jak alokovat diskový prostor, jak ho uvolňovat, jaké jsou návaznosti ostatních částí výpočetního systému na odkládací prostor.

CÍL:

Implementace systému souborů – oddíl, připojování, MBR

Alokace diskového prostoru – souvislá, spojovaná

Alokace diskového prostoru – indexová

Management volného prostoru – vektor bitů, spojový seznam

1. ÚVOD

Jak již jsme řekli, systém souborů provádí mechanismy on-line ukládání a přístupu k datům i programům. Systém souborů je uložen na odkládacím zařízení, jehož hlavním úkolem je neustále uchovávat velké množství dat. Nyní si vysvětlíme aspekty týkající se uchovávání a přístupu k souborům na zatím nejpoužívanějším mediu, magnetickém disku. Ukážeme si postupy, jak alokovat diskový prostor, jak ho uvolňovat, jaké jsou návaznosti ostatních částí výpočetního systému na odkládací prostor.

2. STRUKTURA SYSTÉMU SOUBORŮ

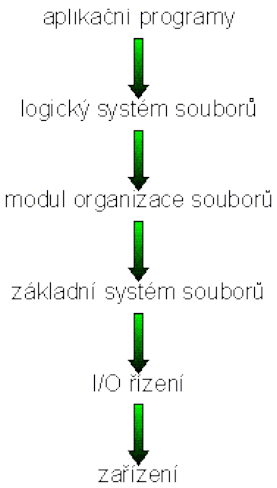
Disky představují plochu, na které je udržována odkládací paměť. Aby I/O přenos mezi diskem a pamětí byl co nejefektivnější, je organizován do *bloků*. Blok je jednotka I/O přenosu mezi diskem a pamětí a představuje jeden nebo více diskových sektorů. Diskový sektor může být velikosti 32 – 4096 B, většinou bývá 512 B (dnešní disky přechází na 4 KB). Disk má dvě hlavní charakteristiky, které z něj dělají vhodné medium pro odkládací prostor:

- Může být přepsán na konkrétním místě – je možno načíst blok do paměti, modifikovat a pak vrátit zpět na totéž místo
- Je možno adresovat kterýkoliv konkrétní blok informací na disku. K souboru je tedy možno přistupovat jak sekvenčně, tak i náhodně a přepínat mezi různými soubory pouhým přesunem čtecí hlavy a počkat až se disk otočí.

2.1 Organizace systému souborů

Při tvorbě systému souborů je třeba vyřešit dva hlavní problémy. První problém je definovat, jak bude vypadat systém souborů z hlediska uživatele. Tato úloha vyžaduje definovat soubor, jeho atributy, operace se souborem a adresářovou strukturu potřebnou k organizování souboru. Za druhé je třeba vytvořit algoritmy a datové struktury, které umožní mapování této logické struktury na fyzické zařízení.

Systém souboru bývá obecně definován do několika úrovní. Příklad takové struktury je na následujícím obrázku. Každá úroveň na obrázku využívá vlastností úrovně nižší k vytvoření vlastností využitelných na úrovni vyšší.



Obrázek 1 - Úrovně systému souborů

Nejnižší úroveň – *I/O řízení* – tvoří *ovladače* jednotlivých *zařízení (device drivers)* a obsluha přerušení zajišťujících přenos mezi pamětí a diskem. Ovladač zařízení může být chápán jako

překladač. Jeho vstupem je příkaz vyšší úrovně (např. načti blok 123) a výstupem sekvence hardwarových instrukcí, které užije hardwarový ovladač, který tvoří rozhraní mezi diskem a výpočetním systémem.

Ovladač zařízení většinou zapíše specifický sled bitů na speciální místo v pamětí hardwarového I/O ovladače a tím mu sdělí typ operace a místo na zařízení, kde bude operace provedena.

Základní systém souborů je zapotřebí na to, aby převedl I/O požadavek do tvaru pochopitelnému ovladači zařízení (každý fyzický blok je identifikován svou numerickou adresou, např. drive 1, cylinder 73, surface 2, sector 10).

Modul organizace souborů realizuje rozhraní mezi logickým a fyzickým systémem soubor. Má informace o souborech a jejich logických blocích, stejně jako o fyzických blocích. Podle použitého mechanismu alokace a při znalosti o uložení souborů je *modul organizace souborů* schopen převést logickou adresu bloku na fyzickou, kterou potřebuje základní systém souborů k zajištění I/O operace. Každý logický blok souborů má své číslo od 0 (resp. 1) po *N*, zatímco fyzické bloky většinou neobsahují data v tomto logickém uspořádání, takže je třeba převod čísla bloku z logického na fyzické.

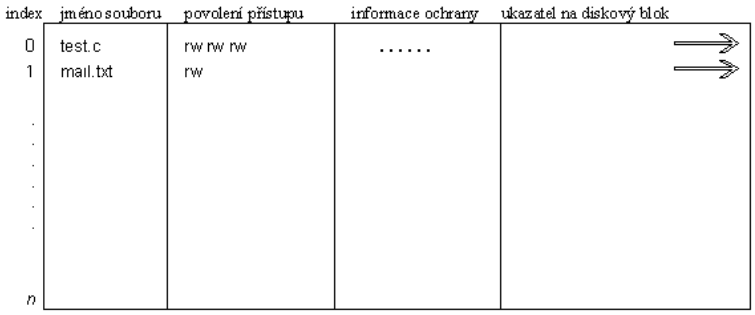
Modul organizace souborů v sobě zahrnuje manažer volného prostoru, který prochází nealokované bloky a dává je na požádání k dispozici modulu organizace souborů.

Logický systém souborů užívá adresářovou strukturu a symbolických jmen souborů k tomu, aby modulu organizace souborů podával zmíněné informace. Logický systém souborů také zajišťuje ochranu a zabezpečení.

Chce-li aplikační program vytvořit nový soubor, zavolá logický systém souborů. Logický systém souborů zná formát adresářové struktury. Pro vytvoření nového souboru načte do paměti obsah patřičného adresáře, zapíše do něj novou položku a uloží ho zpět na disk. S adresářem může být nakládáno jako se souborem, který má indikační položku definující, že se jedná o adresář. Po aktualizaci adresáře tedy může logický systém souborů zavolat modul organizace souborů, aby namaloval adresář na čísla diskových bloků, která mu patří v základním systému souborů a v systému I/O operací.

Jakmile je adresář aktualizován, může ho využít logický systém souborů k vykonání I/O operací. Je-li soubor otvírán, je v něm třeba nalézt položku patřící otvíranému souboru. Aby nemuselo být toto prohledávání vykonáváno před každou I/O operací, OS většinou ukládá adresářové položky otevřených souborů do paměti do tabulky otevřených souborů (urychlení, zjednodušení).

První odkaz na soubor (většinou operace **open**) vyvolá prohledání adresáře a zkopírování patřičné položky do tabulky otevřených souborů v paměti. Ukazatel do této tabulky (*file deskriptor* nebo *file control block*) vrátí operace **open** a všechny operace se souborem po ní následující se odvolávají prostřednictvím ukazatele na tuto položku. Když je soubor uzavřen všemi uživateli, kteří ho otevřeli, je adresářová položka z tabulky opět zkopírována na své místo na disku.



Obrázek 2 - Typická tabulka otevřených souborů

Některé systémy rozvíjejí toto schéma užitím víceúrovňových tabulek. Např. v BSD Unix má každý proces svou vlastní tabulku otevřených souborů, která pouze ukazuje do systémové tabulky otevřených souborů, která ukazuje do tabulky aktivních *inods*. Tabulka aktivních inods je cache právě užívaných inods v paměti a obsahuje položky odkazující na diskové bloky. BSD Unix je typický svým užitím cache kdekoliv je to možné. Užití této cache mívá až 85% účinnost.

2.2 Připojováni (Mounting) systému souborů

Stejné jako musí být soubor nejdříve otevřen (voláním **open**), musí být i systém souborů nejdříve připojen, než budou soubory na něm uložené přístupné procesům. OS zná jméno každého zařízení a jeho umístění v celkové struktuře souborů tzv. *bod připojení (mounting point)*. V Unixu např. systém souborů obsahující domácí adresáře uživatelů může být připojen do adresáře */home*. V adresování všech souborů na něm uložených je třeba na prvním místě uvést adresář */home* (např. */home/sarka/book*. Pokud by tento systém souborů byl připojen do adresáře */user*, byla by cesta k témuž souboru */user/sarka/book*.

Nejdříve systém zkontroluje, jestli zařízení obsahuje korektní systém souborů. Udělá to tak, že vyžádá od ovladače zařízení adresář zařízení a zkontroluje ho, jestli má předpokládaný formát. Na závěr OS zaznamená ve své adresářové struktuře, že systém souborů je připojen na bodě připojení. Toto schéma umožňuje operačnímu systému přenášet systémy souborů v adresářové struktuře jak je zapotřebí.

Jak je to u MacOS? Jakmile tam OS poprvé zaznamená disk (pevné disky jsou zaznamenány při bootování systému, floppy disky když je do nich poprvé vložena disketa), MacOS hledá na tomto zařízení systém souborů. Pokud ho najde, automaticky připojí disk na kořen adresářového stromu a zobrazí ikonu tohoto disku se jménem systému souborů (to je uloženo v adresáři zařízení) na pracovní ploše. Uživatel potom může kliknutím na ikonu zobrazit nově připojený systém souborů.

3. METODY ALOKACE

Možnost přímého přístupu k disku dává široké možnosti v implementaci souborů. Zásadním problémem je, jak alokovat diskový prostor pro tyto soubory, aby disk byl využit co nejefektivněji a přístup k souborům co nejrychlejší. Existují tři hlavní metody alokace diskového prostoru: *souvislá (contiguous), spojovaná (linked) a indexovaná (indexed) alokace*. Každá metoda má své výhody i nevýhody. Jsou OS, které implementují všechny tři metody, ve valné většině však systém užívá metodu jednu.

3.1 Souvislá alokace

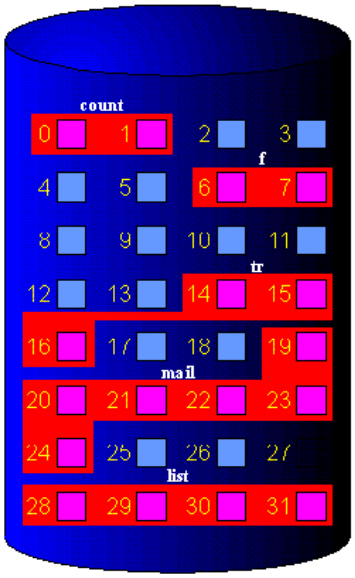
V souvislé alokaci každý soubor obsazuje sled za sebou jdoucích diskových bloků. Adresování na takovém disku je lineární. Poznamenejme, že přístup k bloku *b*+1, bylo-li před

tím přistupováno k bloku *b*, nevyvolá většinou posun čtecí hlavy. Když je přesun hlavy nutný (z posledního sektoru předcházejícího cylindru na první sektor následujícího cylindru), je to posun pouze o jednu stopu.

Počet nutných vyhledávání k alokování souboru je tedy minimální. Tato metoda alokace vede k dobrým výkonům např. OS fy IBM - VM/CMS.

Souvislá alokace souborů je dána diskovou adresou prvního bloku a délkou souboru v blocích. Jestliže je soubor dlouhý *n* bloků a adresa prvního bloku je *b*, potom tento soubor zabírá bloky *b*, *b* + 1, *b* + 2, ..., *b* + *n* – 1. Adresářová položka každého souboru potom obsahuje adresu prvního bloku a délku prostoru alokovaného tomuto souboru (viz následující obrázek).

Přístup k souboru, který byl alokován souvisle je jednoduchý. Při sekvenčním přístupu si systém pamatuje adresu naposledy čteného bloku, a pokud je třeba, načte blok následující. Při přímém přístupu k bloku *i* souboru, který je uložen od bloku *b* je třeba nacist blok *i* + *b*. Souvislá alokace tedy podporuje jak sekvenční, tak i přímý přístup k souboru.



adresář

soubor	start	délka
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Obrázek 3 - Souvislá alokace diskového prostoru

Potíž souvislé alokace spočívá v nalezení dostatečně velkého souvislého prostoru k uložení nového souboru. Je třeba užít nějakého mechanismu managementu volného diskového prostoru.

Užití kontinuální alokace s sebou přináší také značnou vnější fragmentaci disku. Tak, jak je diskový prostor alokován pro jednotlivé soubory a tyto soubory jsou potom opět mazány, rozpadá se disk do malých volných oblastí. Problém nastává v okamžiku, kdy ani největší malá oblast nestačí na vyplnění požadavku o uložení souboru. V závislosti na velikosti diskového prostoru a průměrné velikosti souboru může být fragmentace velkým či malým problémem.

I když je předem známa maximální velikost souboru, může být tento mechanismus alokace velmi neúčinný. Uvažme soubor, který může dosáhnout ohromné velikosti, ale jeho přírůstky jsou malé a pomalé. V tom případě soubor dlouhou dobu zabírá zcela zbytečně diskový prostor a dochází k vnitřní fragmentaci.

Aby zabránily alespoň některým ze zmiňovaných nedostatků, užívají některé OS (např. RSX-IIB) modifikovaný algoritmus souvislé alokace. Na začátku naalokují souboru určitý celistvý kus diskového prostoru. Pokud tento prostor časem není dostatečný, je vytvořen další soubor

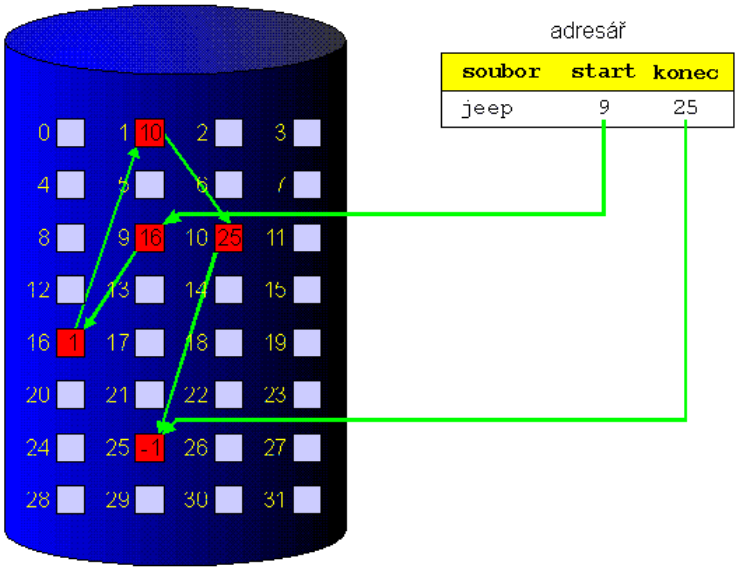
stejných vlastností jako původní – *extend*, který je logickým pokračováním souboru původního. V jeho adresářové položce je nastaven čítač pokračování jako inkrement předchozího.

3.2 Spojovaná alokace

Spojovaná alokace řeší všechny problémy alokace kontinuální. Při spojované alokaci je každý soubor spojovým seznamem diskových bloků a tyto bloky mohou být rozmístěny kdekoliv na disku. Každá adresářová položka obsahuje ukazatel na první a poslední blok souboru. Např. soubor obsahující 5 bloků může začínat na bloku 9, pokračovat blokem 16, pak blokem 1, 10 a na závěr blokem 25 (viz následující obrázek).

Každý blok obsahuje ukazatel na následující blok. Tyto ukazatele nejsou uživateli přístupné. Tzn., že jestliže velikost bloku je 512 B a adresa bloku zabírá 4 B, je uživatelsky přístupno pouze 508 B v každém bloku.

Chceme-li vytvořit nový soubor, jednoduše vytvoříme novou položku v adresáři, ukazatel na první diskový blok inicializujeme na hodnotu *nil* a velikost souboru je stanovena na 0. Zápis do souboru vyvolá žádost o volný blok, který nalezne modul managementu volného prostoru. Do tohoto bloku bude proveden zápis a blok bude připojen na konec souboru. Při čtení souboru jednoduše načítáme jednotlivé bloky, tak jak je na sebe váží ukazatele.



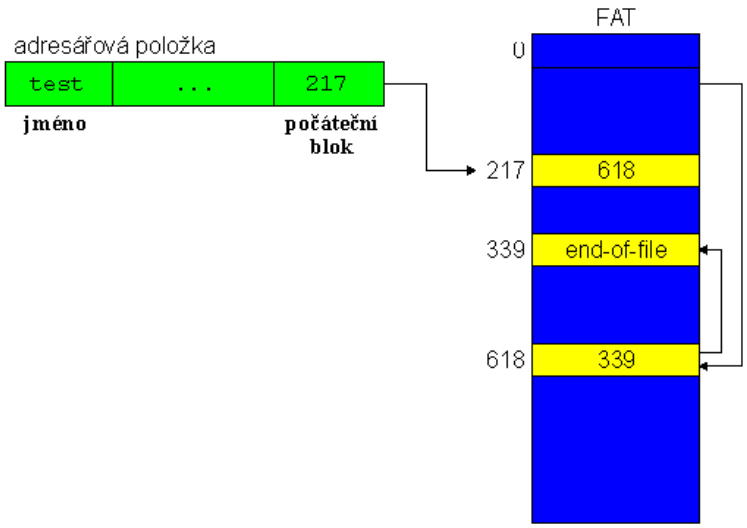
Obrázek 4 - Spojovaná alokace diskového prostoru

Nedochází tu k žádné vnější fragmentaci, neboť k vyplnění žádosti o nový blok může být použit kterýkoliv z volných. Není zde také třeba dopředu deklarovat velikost souboru při jeho vytváření. Soubor může postupně narůstat tak dlouho, dokud budou na disku volné bloky. Stejně tak není třeba provádět žádné zhušťování.

Přesto má spojovaná alokace svá úskalí. Hlavní problém je, že efektivně může být užita pouze pro sekvenční přístup k souboru. Chci-li ale najít *i*-ty blok souboru, musím své hledání začít na prvním bloku a *i* krát se posunout po ukazateli z bloku na blok. Každý takový přesun vyžaduje načtení informace a někdy posun po disku. Spojovaná alokace má tedy špatnou podporu přímého přístupu k souboru.

Další nevýhodou je diskový prostor nutný k ukládání ukazatelů. Jestliže je na ukazatel třeba 4 B při 512 B blocích, je 0,78 % diskového prostoru užito pro ukazatele a ne pro uživatelské informace. Každý soubor tedy zabírá více místa než při jiné metodě alokace.

Alternativou je užití *tabulky alokace souborů (file–allocation table).* Tuto jednoduchou a efektivní metodu alokace diskového prostoru využívají MS DOS a OS/2. Určitá část na začátku každé partition je ponechána stranou na tabulku souboru. V této tabulce je položkou zanesen každý diskový blok resp. klastr a je v ní indexován číslem bloku. FAT je potom užita jako seznam linků. Každá adresářová položka obsahuje číslo prvního bloku souboru. Položka FAT indexovaná tímto blokem obsahuje číslo následujícího bloku v souboru. Tento řetěz pokračuje až k poslednímu bloku, jehož položka ve FAT má speciální hodnotu EOF.



Obrázek 5 - Tabulka alokace souboru

Nevyužité bloky ve FAT jsou indikovány hodnotou 0. Alokování nového bloku souboru potom představuje jednoduchou úlohu nalezení první položky tabulky, jejíž hodnota je 0 a nahradit hodnotu EOF dřívějšího posledního bloku souboru odkazem na tento nový blok, jehož hodnota je nastavena na EOF. Ilustrativní příklad uvádí obrázek 85. Soubor sestava z bloku 217, 618 a 3311.

Pokud není FAT cacheována, vyvolává značné množství pohybu po disku. Hlava se musí přesunout na začátek partition, aby načetla položku z FAT, potom nalézt lokaci bloku, který má být užit a přesunout se na toto místo. V nejhorším případě k tomuto může dojít u každého bloku.

FAT podporuje náhodný přístup k blokům souboru, protože v tabulce lze nalézt všechny potřebné informace.

3.3 Indexová alokace

Spojovaná alokace řeší problém vnější fragmentace a problém deklarace velikosti, který je u alokace souvislé. Spojovaná alokace ovšem spatně podporuje přímý přístup k souboru, protože ukazatel na následující blok je součástí bloku předcházejícího, z čehož nutně plyne degradace na sekvenční přístup k souboru.

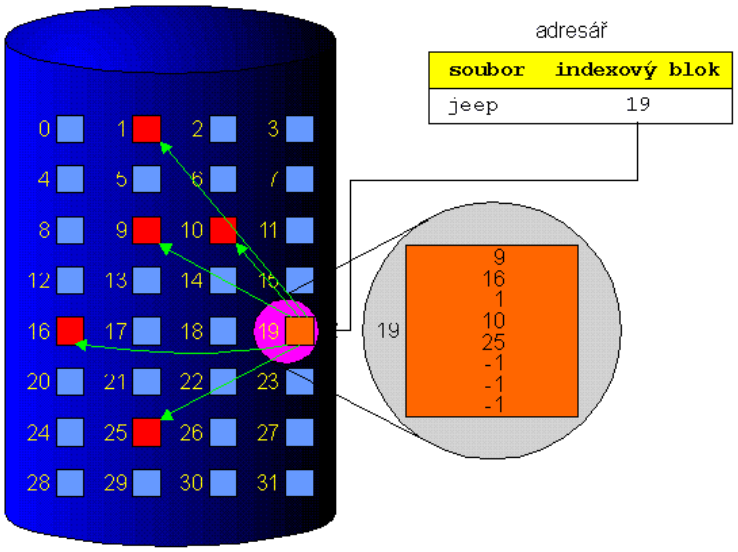
Indexovaná alokace řeší tento problém tak, že všechny ukazatele uloží do jednoho zvláštního bloku – *bloku indexů*. Každý soubor má svůj vlastní indexový blok, který obsahuje seznam adres diskových bloků. *I*-tá položka v indexovém bloku ukazuje na *i*-ty blok souboru. Adresářová položka potom obsahuje adresu indexového bloku (viz následující obrázek).

Pro načtení *i*-teho bloku užijeme adresu uloženou v *i*-te položce indexového bloku. Jedná se zde prakticky o ekvivalent stránkování paměti.

Při vytvoření nového souboru jsou všechny ukazatele v jeho indexovém bloku nastaveny na hodnotu *nil*. Jestliže je *i*-ty blok poprvé ukládán na disk, dodá manažer volného prostoru

neobsazený blok, do nějž se pak provede zápis a jeho adresa se uloží jako *i*-ta položka indexového bloku.

Indexová alokace podporuje přímý přístup bez nebezpečí vnější fragmentace, neboť požadavek na volný blok může uspokojit kterýkoliv volný.



Obrázek 6 - Indexová alokace diskového prostoru

Možným nebezpečím je tu ale interní fragmentace. Diskový prostor nutný pro indexový blok je obecně větší, než prostor potřebný pro ukazatele při spojované alokaci. Uvažujme poměrně častý případ, kdy máme soubor sestávající z jednoho či dvou bloků. Při spojované alokaci zahrnuje režie prostor potřebný pro dva ukazatele. Při indexované alokaci musíme alokovat celý blok, i když v něm bude pouze jeden či dva ukazatele různé od *nil*.

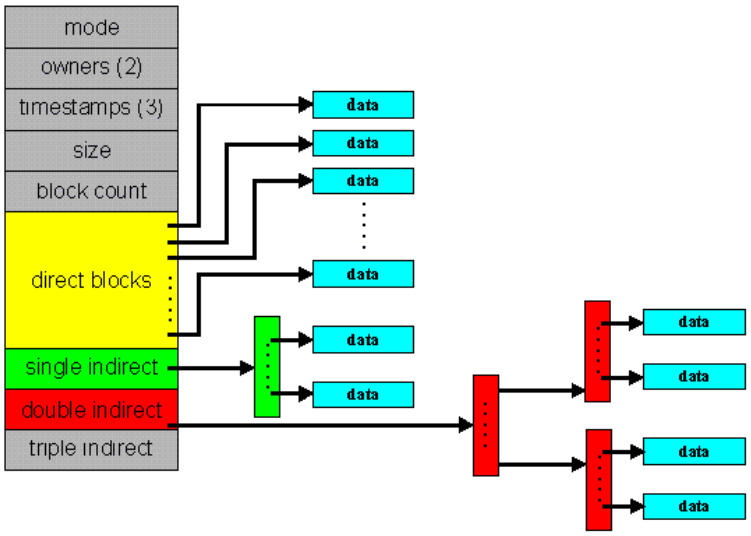
Předcházející úvaha vede k zamýšlení nad velikostí indexového bloku. Každý soubor musí mít svůj indexový blok, takže naší logickou snahou je, aby tento blok byl co nejmenší. Pokud bude ale indexový blok příliš malý, nebude schopen držet všechny ukazatele na všechny bloky velkého souboru. Řešení tohoto problému mohou být následující:

- **Spojová struktura** - indexový blok je běžně jeden diskový blok. Pro umožnění existence velkých souborů je možno spojit několik diskových bloků dohromady. Indexový blok může obsahovat např. malou hlavičku, kde je zaznamenáno jméno souboru a pak následuje seznam prvních 100 ukazatelů na diskové bloky. Následující adresa (poslední slovo v indexovém bloku) je *nil* pro malý soubor nebo ukazatel na další indexový blok (u velkého souboru).
- **Víceúrovňový index** - variantou spojové struktury je užit indexový blok pouze jako seznam odkazů na další indexové bloky, které potom ukazují na diskové bloky souboru. Při odkazu na nějaký blok souboru OS užije index první úrovně k tomu, aby nalezl index druhé úrovně, kde pak najde adresu požadovaného bloku dat. Tento přístup může potom pokračovat indexem trojúrovňovým, čtyřúrovňovým atd. podle toho, jak maximálně veliký soubor se může na disku objevit. Máme-li blok velikosti 2048 B, můžeme v něm uložit 512 4B ukazatelů na diskové bloky. Index dvou úrovní umožní adresovat 4 194 304 bloků, což představuje soubor o velikosti 8,5 GB. Tato velikost přesahovala kapacitu tehdejších disků.
- **Kombinovaný přístup** – BSD Unix používá kombinaci obou výše zmiňovaných přístupů. Indexový blok (neboli Inode) obsahuje, řekneme 15 ukazatelů. Z nich prvních 12 ukazuje *přímo* na prvních 12 bloků souboru. Z toho plyne, že malé soubory (při blocích velikosti 4

kB soubory do 48 kB) nepotřebují vlastní indexový blok a vystačí si s Inodem. Inody všech souborů jsou uchovávány ve vyhrazené oblasti na disku.

Tři ukazatelé ukazují na víceúrovňové indexy. První z nich ukazuje na index druhé úrovně, který obsahuje adresy dalších bloků souborů počínaje třináctým. Druhý odkazuje na trojúrovňový index a poslední případně u extrémně velkých souborů i na čtyřúrovňový index bloků. Index čtvrté úrovně není na obrázku uveden.

Užitím této metody je počet odkazovatelných bloků větší než počet bloků, které je možno adresovat pomoci 4 B. Inode je vidět na následujícím obrázku.



Obrázek 7 - Inode v Unixu

Poznamenejme, že indexová alokace trpí některými výkonnostními problémy stejně jako spojovaná alokace. Indexové bloky používaných souborů mohou a také bývají cachovány v paměti, ale jednotlivé bloky souborů mohou být rozptýleny různé po disku, což představuje problémy s posunem hlav a otáčením disku.

4. MANAGEMENT VOLNÉHO PROSTORU

Protože velikost celkového diskového prostoru je omezena, je nutno opětovně využívat všechen prostor, který je uvolněn při smazání souboru (je-li to možné). U optických disků, na které lze provést zápis pouze jednou tato starost odpadá.

Kvůli přehledu o volném diskovém prostoru udržuje OS *seznam volných bloků (free–space list)*. Seznam volných bloků obsahuje všechny bloky, které jsou volné, tj. nejsou alokovány pro nějaký soubor nebo adresář.

Při vytváření nového souboru vyhledáme v seznamu volných bloků požadované množství diskového prostoru a tento prostor novému souboru alokujeme. Nové alokované bloky jsou potom odebrány ze seznamu volných bloků. Jestliže je soubor smazán, bloky mu alokované se do seznamu volných bloků přidají.

4.1 Vektor bitů

Často je seznam volného prostoru implementován jako *bitová mapa* nebo *vektor bitů*. Každý blok je tu reprezentován jedním bitem. Jestliže je blok volný, hodnota bitu je 1, jestliže je alokován, je jeho hodnota 0.

Uvažujme např. disk, kde volné jsou bloky 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, 27 a zbytek disku tvoří bloky alokované. Bitová mapa volného prostoru potom bude vypadat následovné:

`00111100111111000110000001110000000000…`

Hlavní výhodou tohoto řešení je malá velikost bitové mapy a její jednoduché prohledávání, pokud chci najít první volný blok. Mnoho počítačů má navíc implementovány operace pro manipulaci s bity, které mohou byt efektivně využity k implementaci tohoto seznamu volných bloků.

Např. procesory Intel 386 a 486 a Motorola 68020 až po 68040 (tedy procesory uzívané na počítačových platformách IBM PC kompatibilní a Macintosh) mají implementovány instrukce, které vrací offset prvního nenulového bitu ve slově. Proto také počítače Apple Macintosh užívají pro management volného prostoru metodu vektoru bitů. K nalezení prvního volného bloku potom MacOS prochází sekvenčně každé slovo v bitové mapě a hledá první nenulové, neboť slovo s hodnotou 0 značí všechny bloky alokované.

V prvním slově s nenulovou hodnotou je vyhledán vzpomínanou instrukcí první nenulový bit, který svou pozicí udává adresu volného bloku. Výpočet čísla bloku je:

"počet bitů ve slově" * "počet slov s hodnotou 0" + "offset prvního 1 bitu"

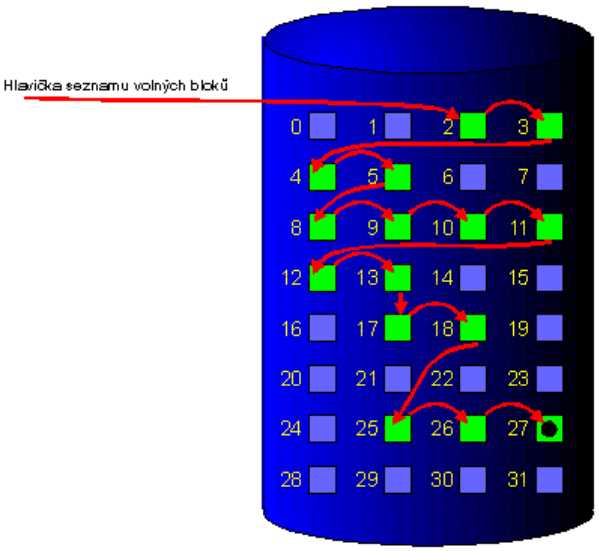
Nevýhodou implementace seznamu volných bloků pomocí bitového vektoru je, že celý vektor musí být uchováván v operační paměti. To je možné u menších disků, jaké mají např. mikropočítače, ale u disků s větší kapacitou to představuje určitý problém. Pokud bychom tímto algoritmem mapovali volný prostor disku s kapacitou 1,3 GB, potřebovali bychom v paměti pro vektor bitů 310 KB při 512 B blocích.

4.2 Spojový seznam

Dalším možným řešením je vytvořit spojový seznam všech volných bloků a uchovávat v paměti pouze ukazatel na první volný. První volný blok v sobě obsahuje ukazatel na druhý volný atd. Disk obsazený stejně jako v předcházející kapitole by při managementu volného prostoru spojovým seznamem vypadal tak, jak ukazuje následující obrázek.

Toto schéma, stejně jako každá spojová struktura, je nevhodná pro průchody seznamem. Při průchodu je třeba načítat každý blok, což stojí hodně času, nicméně procházení seznamem volných bloků není právě častá operace. Nejčastěji OS potřebuje volný blok, který může alokovat souboru, a tak užije první blok seznamu.

Závěrem pouze připomeňme, že FAT v sobě zahrnuje i seznam volných bloků a není tudíž potřeba užívat vlastní metodu na správu volného prostoru disku.



Obrázek 8 - Spojový seznam volných bloků disku

4.3 Seskupování

Dalším možným řešením seznamu volných bloků je uchovávat v prvním volném bloku adresy *n* dalších volných bloků. Prvních *n* – 1 bloků je skutečně volných, *n*-tý obsahuje adresy dalších *n* volných bloků atd. Výhodou tohoto řešení je, že je možno rychle a jednoduše získat najednou adresy více volných bloků, na rozdíl od řešení pomocí spojového seznamu.

4.4 Počítání

Další přístup využívá toho, že na disku obecně existují spojité úseky volných bloků. Např., pokud je diskový prostor alokován spojitě nebo díky klastrům. Potom je výhodnější, spíše než uchovávat seznam *n* volných diskových adres, uložit pouze adresu prvního bloku a počet dalších volných bloků, které tvoří s prvním souvislý úsek. Přesto, že každá položka takovéhoto seznamu vyžaduje větší prostor, než je třeba pro adresu diskového bloku, výsledný prostor obsazený tabulkou je menší než při běžné metodě, protože počet dalších volných bloků je téměř vždy větší než 1.

5. IMPLEMENTACE ADRESÁŘE

Výběr metody alokace prostoru pro adresář a algoritmu pro správu adresáře má velký vliv na efektivitu, výkon a spolehlivost systému souborů. Je proto důležité pochopit charakteristické rysy možných implementaci.

5.1 Lineární

Nejjednodušší metodou implementace adresáře je užít lineární seznam jmen souborů a ukazatelů na datové bloky. Lineární seřazení položek v adresáři implikuje lineární algoritmus vyhledání určité položky. Tato metoda je velmi jednoduchá na naprogramování, ale značně časově náročná.

Chceme-li vytvořit nový soubor, musíme nejprve prohledat celý adresář, jestli se v něm nenachází soubor stejného jména. Pokud ne, můžeme v druhém kroku vytvořit v adresáři novou položku pro tento soubor.

Chceme-li smazat soubor v adresáři, musíme jeho položku vyhledat a uvolnit prostor, který jí byl alokován. Abychom zajistili možné další využiti tohoto uvolněného prostoru, je třeba provést jednu z několika možných operací. Můžeme tuto položku označit jako neobsazenou

např. prázdným jménem souboru nebo stavovým bitem nebo ji můžeme uložit do seznamu volných adresářových položek. Třetí možností je nakopírovat poslední adresářovou položku na uvolněné místo a zmenšit tak velikost adresáře. K minimalizaci času nutného ke smazání souboru je možné užít spojovou strukturu adresářových položek.

Reálným nedostatkem spojové adresářové struktury je lineární vyhledávání položky. Informace z adresáře jsou užívány velmi často a implementace pomalého algoritmu přístupu je viditelné i pohledem běžného uživatele počítače. Ve skutečnosti mnohé OS implementují softwarovou cache pro uložení nejčastěji používaných adresářových informaci. Softwarová cache pomáhá ubránit se opětovnému čtení týchž dat z disku. Uspořádaný seznam umožňuje lepší vyhledávání a snižuje průměrnou dobu hledání. Takový vyhledávací algoritmus je však složitější co do naprogramování.

Požadavek na uspořádaný seznam adresářových položek může také komplikovat vytváření a mazání souborů. K zachování uspořádané struktury můžeme někdy potřebovat přenést podstatnou část položek v adresáři. (Přesto jako výhodu můžeme uvést, že pokud potřebujeme vytvořit uspořádaný seznam souborů v adresáři, nemusíme nic nijak řadit a stačí vypsat položky adresáře tak, jak v něm jdou za sebou.

5.2 Hashovací tabulka

Další datovou strukturou, která může být užita k implementaci adresáře je hashovací tabulka. I tady jsou adresářové položky uchovávány v lineárním seznamu, ale k jeho procházení je užito hashování. Hashovací tabulka použije hodnotu vypočítanou ze jména souboru a vrátí ukazatel na jméno souboru v lineárním seznamu. Tím se významně zkrátí doba prohledávání adresáře.

Zlepší se i vytváření a mazání položek, i když je třeba ošetřit některé nové mimořádné situace (např. když dvě různá jména vrací stejnou adresu do lineárního seznamu). Nejzávažnější nevýhodou hashovací tabulky je její pevná délka a podmíněnost hashovací funkce touto délkou.

Například uvažujme hashovací tabulku o 64 položkách. Hashovací funkce převede jméno souboru na celé číslo od 0 do 63, pravděpodobně bude užito funkce *mod 64*. Pokud je třeba v adresáři vytvořit položku pro 65. soubor, musíme prodloužit hashovací tabulku na řekněme 128 položek. Potom musíme také změnit hashovací funkci na funkci, která bude vracet hodnoty z množiny 0 – 127 a musíme reorganizovat existující adresářové položky podle této funkce.

6. OBNOVA

Protože jsou soubory uloženy jak v paměti, jak i na disku, musíme zajistit, že výpadek systému nebude mít za následek ztrátu dat nebo jejich nekonzistentnost.

6.1 Testování konzistence

Jak jsme si ukázali jinde, je část informací o adresáři uložena v hlavní paměti (cache), aby byl urychlen přístup k položkám adresáře. Adresářové informace v paměti jsou obecně novější než jejich ekvivalent uložený na disku, protože zápis do cacheovaných adresářových položek nemusí nutně vyvolat zápis téhož na disk.

Uvažme z tohoto pohledu poruchu počítače. V tomto případě je tabulka otevřených souborů obecně ztracena a s ní i všechny změny v adresářích otevřených souborů. Tato událost může ponechat systém v nekonzistentním stavu, protože aktuální stav některých souborů nemusí být zobrazen v jejich adresářových položkách. Proto je po rebootování počítače často spouštěn program, který zajisti opravy případných nekonzistencí.

Program pro zjišťování konzistence porovnává data v adresáři s bloky na disku a snaží se opravit všechny nekonzistence, které objeví. Algoritmy alokace a správy volného prostoru definuji typy problémů, které má zjišťovat, hledat a také jak tyto chyby úspěšně odstranit.

Je-li např. užita spojovaná alokace a každý blok obsahuje ukazatel na svého následníka, může být adresářová položka tohoto souboru restaurována za pomoci jeho datových bloků. Ztráta adresářové položky při indexové alokaci může mít katastrofální následky, protože datové bloky o sobě navzájem neví. Z tohoto důvodu Unix cachuje položky pro čtení, ale jakmile nějaký zápis vyvolá změnu inode, provede se zápis inode dříve než operace, která ho vyvolala.

6.2 Backup and restore

Protože na magnetickém disku může dojit k chybě a tím ke ztrátě dat, je třeba provádět zálohování těchto dat. K tomu se používají speciální systémové programy (*backup*), které přenášejí systém souborů z disku na magnetickou pásku, optický disk apod. Obnova ztraceného souboru nebo celého disku spočívá v provedení operace *restore*, která obnoví data ze zálohovacího zařízení.

Abychom minimalizovali potřebné kopírování, můžeme užít informaci o každém souboru z jeho adresářové položky. Program *backup* většinou uchovává datum posledního zálohování. V tom případě nemusíme zálohovat soubory, jejichž datum poslední změny je dřívější než datum poslední zálohy. V jiných případech lze využít „archivního bitu“, který je součástí obsahu adresářové položky souboru. Typicky plán zálohování může vypadat např. takto.:

- **1. den** - Kopie všech souborů z disku na zálohovací medium
- **2. den** - Kopie souborů, které se změnily od 1. dne na jiné medium
- **3. den** - Kopie souborů, které se změnily od 2. dne na jiné medium
- ...
- **N. den** - Kopie souborů, které se změnily od N-1. dne na jiné medium, potom přejdi na 1. den

Nový zálohovací cyklus lze provádět na další sadu n medií, nebo jím přepsat postupně předcházející zálohy. Při takto prováděném zálohování můžeme obnovit celý disk, jestliže začneme u prvního media a postupně obnovíme soubory také ze všech následujících až do N.