15. Systém souborů

Implementace abstrakce souboru
Soubor = data dostupná pod jednoznačnou identifikací (jméno souboru)

Fyzická úroveň	**Logická úroveň**
V/V po sektorech	V/V po slabikách (záznamech)
adresa fyzického sektoru	pojmenovaný soubor
žádná ochrana	práva pro uživatele, skupiny

Funkce systému souborů:
1. Struktura souborů
2. Alokace diskového prostoru
3. Souborové operace
4. Ochrana

1. Struktura souborů

a) Logická struktura souboru (na úrovni rozhraní jádra):

- bez struktury - pole slabik (Unix):
  write(fd, buf, length)
- logické záznamy pevné délky:
  create_file(name, record_length),
  write_record(fd, buf, record_number)
  + rychlý náhodný přístup k záznamům
  + lze aktualizovat záznamy uvnitř souboru
  - velký prostor nevyužit, pokud nejsou záznamy plné
- logické záznamy proměnné délky:
  write_record(fd, buf, length)
  - vystavení na záznam složité - sekvenční průchod
  - záznamy lze aktualizovat jen pokud se nezmění jejich délka
- index-sekvenční soubory - záznamy proměnné/pevné délky s klíčem (celé číslo), rychlé vyhledání záznamu podle klíče (B-strom)

b) Vnitřní struktura souboru (na úrovni rozhraní V/V)

- pole alokačních bloků
- alokační blok = několik souvislých sektorů (512 byte) na disku
- úzce souvisí s alokací diskového prostoru (bod 2)

Problém zobrazení logická adresa → fyzická adresa

čti/zapiš(soubor, záznam) → V/V operace čti/zapiš sektor C,H,S

Příklad:

read byte 1000-1030 →

vyhledej diskový sektor obsahující 1000
přečti sektor do paměti
přesuň data
vše? ne → vyhledej diskový sektor obsahující 1024 a ↑

write byte 1000-1030 →

vyhledej diskový sektor obsahující 1000
přečti sektor do paměti
zapiš data do sektoru
zapiš sektor na disk
vše? ne → vyhledej diskový sektor obsahující 1024 a ↑

Pokud je soubor zapisován sekvenčně, jsou čtení při zápisu zbytečné (přepíšou se postupně všechna data v sektoru) – ale jak to má systém dopředu vědět? Nemůže -> **musí odkládat zápisové operace.**

Fyzický sektor – historicky 512 byte
Disky dnes s interně 4KB sektorem, sektory 512B jsou **emulovány**! Stejný problém zápisu jako částečná modifikace sektoru z jádra systému, ale tentokráte uvnitř disku – nutné zapisovat sekvenčně násobky 4KB (nebo přejít na 4KB sektory).

Alokace alokačních bloků pro soubor:

- různé typy přístupu k souborům - sekvenční, náhodný
- malé soubory - je jich hodně, AB nesmí zabírat moc místa
- velké soubory - moc AB, AB pokud možno souvisle uložené na médiu pro rychlý přístup

A) Souvislá alokace

- nutná prealokace místa pro soubor, nelze zvětšovat

+ rychlý přístup
- problém externí fragmentace (alokace úseků)

B) Lineární seznam

Jednosměrně nebo obousměrně vázaný seznam alokačních bloků, indexy na další/předchozí součástí bloků.
+ jednoduché vkládání, rušení záznamů proměnné délky, zvětšování souboru
+ nulová externí fragmentace
- pomalý náhodný přístup
- slabá lokalita, náhodné adresy sousedních logických bloků

C) Index-sekvenční organizace (pevné alokační bloky)

Prostor je alokován po alokační jednotkách (volba velikosti), index obsahuje seznam alokačních jednotek.
Problém velikosti tabulky popisující zobrazení:

- globální index pro celý systém souborů (FAT)
- víceúrovňové indexy pro každý soubor zvlášť (UFS)

UFS (původně alokační blok = sektor 512B):

i-node - prvních 10 alokačních bloků přímo (1. úroveň)
11. alokační blok adresy alokačních bloků (2. úroveň)
12. alokační blok adresy alok. blok alok. bloků (3. úroveň)
13. alokační blok adresy alok. blok^3 (3. úroveň)

DÚ: Jaká byla max. velikost souboru?

Vlastnosti:
+ jednoduchá implementace
+ zvětšování, zmenšování souborů
+ náhodné vystavování indexové, pro malé soubory přímé
- pro velké soubory průchod víceúrovňovými indexy
- bez optimalizované alokace po počátečním zaplnění disku náhodné pozice alokačních bloků souboru

Řešení:
- větší alokační bloky (1KB, 2KB), problém interní fragmentace

D) Index-sekvenční organizace (proměnné alokační bloky)
JFS (IBM), XFS (SGI), ZFS (Sun)
index alokačních bloků proměnné délky:

ofset v souboru, adresa AB, délka (počet AB)
ofset v souboru, adresa AB, délka
...

Problém alokace úseků proměnné velikosti, z praktických důvodů omezení na násobky (mocniny 2) nejmenšího alokačního bloku, s limitem 128KB, apod. AB mohou být v ideálním případě rovné velikosti souboru = souvislá alokace.

2. Alokace diskového prostoru

Volba velikosti alokačního bloku, měření viz popis BSD FFS:
512 B - 95% využití diskového prostoru
1 KB - 90 %, pomalé čtení/zápis 20-50 KB/s
2 KB - 80 %
4 KB - 60 %, rychlé 140-220 KB/s
Optimalizace diskových operací – vyžadovány alokační bloky souboru blízko u sebe, ideálně souvislé alokace.
a) souvislá alokace – problém vyhledání volného úseku proměnné velikosti, správa úseků proměnné velikosti
b) alokační bloky pevné velikosti – lze použít libovolný AB, ale je vhodné najít co nejbližší k poslednímu AB souboru
c) alokační bloky proměnné velikosti – jako a + b

Optimalizace výběru volného bloku

- lineární seznam volných bloků (UFS) – nelze
- bitová mapa – pro celý disk je příliš velká, nelze udržovat trvale v paměti, ani prohledávat na disku
- skupiny cylindrů, pro každý samostatná bitová mapa a sumární statistika – jedna bitová mapa rozumné velikosti, lze v ní hledat optimální skupinu volných AB (UFS velikost rovna AB*8)
- index-sekvenční organizace – volnou část disku popsat jako soubor se seřazenými volnými úseky podle čísla AB (B-strom)

Okamžik alokace diskového prostoru pro soubor:

- prealokace – nutná deklarace velikosti při otevření souboru,
- dynamická alokace – alokační bloky se přidělují při zápisu do souboru. **Problém:** není dopředu známá velikost, volný prostor v okolí konce může být využit pro jiný současně vytvářený soubor,
- odložená alokace – alokační bloky se přidělují až v okamžiku zápisu na disk, pokud je soubor dočasný a vzápětí je smazán, nemusí dojít k alokaci AB vůbec (XFS, ZFS).

Příklad: (Unix) BSD FFS – alokace datových bloků:

- Alokační bloky se dělí na fragmenty (FAB), jeden AB=8FAB
- Fragmenty mohou být pro daný soubor alokovány pouze sekvenčně na konci souboru a musí jich být méně než celý blok. Pokud je soubor menší než AB, skládá se pouze ze sekvenčních FAB. Pokud soubor naroste a vyžaduje více fragmentů, přesune se obsah co celistvého AB.
- Každá skupina cylindrů má bitovou mapu popisující stav všech FAB ve skupině.
- Velikost bitové mapy je AB*8, pro 16KB AB tedy 128K bitů, čili popisuje 128K*FAB (256MB).
- Přímo adresované bloky alokovány ve skupině cylindrů podle umístění i-uzlu, co nejblíže u sebe.
- Další skupiny po 2048 blocích vždy první blok ve skupině cylindrů, která je zaplněná méně než je průměr (hledá se od následující do konce, pak od začátku), další bloky ve stejné skupině jako předchozí, co nejblíže předchozímu bloku.
- Alokace i-uzlů pro soubory ve stejné skupině cylindrů jako adresář, i-uzlů pro adresáře implicitně ve stejné skupině cylindrů jako nadřazený, ale jen do limitu 255 sekvenčně vytvořených, další jsou v jiné skupině cylindrů (sleduje se rovnoměrné rozložení adresářů ve skupinách cylindrů i počet adresářů vzhledem k počtu datových souborů v každé skupině). Kořenové adresáře jsou alokovány v náhodně zvolené skupině cylindrů.
- Rezervace prostoru pro metadata na začátku skupiny cylindrů (2013).

3. Operace se soubory

a) **Zpřístupnění souboru** - vyhledání informací o souboru na základě jména, metadata
b) **Čtení/zápis** - obvykle na základě deskriptoru souboru
c) **Manipulace se souborem** - podle jména, podle deskriptoru

3a) Zpřístupnění souboru, metadata

Jméno souboru → informace o souboru (zobrazení)
Systém souborů musí obsahovat mimo vlastní data souborů také **metadata** popisující organizaci souborů a pojmenování souborů.

Pojmenování souborů:
- jednoúrovňový adresář
- víceúrovňový adresář - strom/acyklický graf (problém rychlosti vyhledání jména)

Umístění informací o souboru:
- v adresáři – omezení na stromovou necyklickou strukturu
- mimo (i-node) – obecný graf
  Jméno souboru → index (i-node) → informace o souboru

Organizace adresářů a hledání v adresářích:
- lineárně (UFS, BSD FFS),
- B-strom (XFS, ZFS, NTFS),
- hash (pouze v paměti, FreeBSD)

Správa adresářů:

- **oddělená obsluha** - nelze použít rozhraní pro práci se soubory, možná optimalizace, změna struktury (Win32 *FindFirstFile()*, *FindNextFile()*, *FindClose()*)
- **adresář jako soubor** - zapisovat může pouze systém, jinak standardní soubor, používají se stejné operace čtení, interpretace obsahu adresáře je na úrovni knihoven (*opendir()*, *readdir()*, *closedir()*)

Délka jména souboru:

- první Unixy pevná (14 znaků), jednodušší implementace
- proměnná (max. 255 znaků), problém alokace prostoru v adresáři (externí fragmentace po smazaných jménech)

Atributy (informace o souborech - Unix):

- typ souboru (adresář, speciální soubor, roura, socket)
- majitel, skupina (původně 16bitů, nyní 32bitů)
- práva (majitel, skupina, ostatní, R/W/X)
- počet odkazů (tvrdý link)
- velikost (původně 32, nyní 64 bitů)
- čas posledního přístupu, změny, změny stavu (tradičně 32bitů, postupný přechod na 64bitů – problém Y2.038K)
- indexy přímých a nepřímých alokačních bloků

Uložení atributů

Fixní struktura - na disku obdobně jako struktura v paměti, fixní počet atributů, náročné na změnu velikosti, přidávání atributů (klasické UFS, FFS, EXT)

Dynamické seznamy atributů - formou seznamu (NVlist), omezeno obvykle na AB = jméno atributu, délka hodnoty, hodnota (novější systémy), často implementované i s fixními jako *Extended attributes*.

Příklad: UFS2 <ufs/dinode.h>

```
#define NXADDR  2   /* External addresses in inode. */
#define NDADDR  12  /* Direct addresses in inode. */
#define NIADDR  3   /* Indirect addresses in inode. */
struct ufs2_dinode {
   u_int16_t di_mode;        /*   0: IFMT, permissions*/
   int16_t   di_nlink;       /*   2: File link count. */
   u_int32_t di_uid;         /*   4: File owner. */
   u_int32_t di_gid;         /*   8: File group. */
   u_int32_t di_blksize;     /*  12: Inode blocksize. */
   u_int64_t di_size;        /*  16: File byte count. */
   u_int64_t di_blocks;      /*  24: Blocks actually */
   ufs_time_t di_atime;      /*  32: Last access. */
   ufs_time_t di_mtime;      /*  40: Last modified. */
   ufs_time_t di_ctime;      /*  48: Last inode change*/
   ufs_time_t di_birthtime;  /*  56: Inode creation. */
   int32_t   di_mtimensec;   /*  64: Last modified.  */
   int32_t   di_atimensec;   /*  68: Last access. */
   int32_t   di_ctimensec;   /*  72: Last inode change*/
   int32_t   di_birthnsec;   /*  76: Inode creation */
   u_int32_t di_gen;         /*  80: Generation num */
   u_int32_t di_kernflags;   /*  84: Kernel flags. */
   u_int32_t di_flags;       /*  88: Status chflags. */
   u_int32_t di_extsize;     /*  92: Extern attr size */
   ufs2_daddr_t di_extb[NXADDR];/*  96: EA blocks. */
   ufs2_daddr_t di_db[NDADDR]; /* 112: Direct blocks. */
   ufs2_daddr_t di_ib[NIADDR];  /* 208: Indirect blks */
   u_int64_t di_modrev;    /* 232: i_modrev for NFSv4 */
   ino_t  di_freelink; /* 240: SUJ: Next unlinked ino */
    uint32_t         di_spare[3];
};
```

Odolnost proti výpadku, zotavení po havárii

Příklad: smazání souboru = zrušení položky v adresáři (zápis do adresáře), uvolnění i-nodu (zápis do tabulky i-nodů), uvolnění alokačních bloků (zápis uvolněných alokačních bloků do indexů, bitové mapy), aktualizace počtu volných/obsazených bloků a i-nodů (zápis do superbloku). Pokud jsou tyto zápisové operace realizovány asynchronně, vzniká při výpadku nekonzistentní a nezotavitelný stav metadat na disku (UFS).

Řešení:

1. **Synchronní zápis** metadat v pevně definovaném pořadí (BSD FFS <= 4.3) – pomalé operace se soubory (rušení).
2. **Soft metadata update** - zápis diskových bloků v tom pořadí, jak to vyžaduje V/V vyrovnávání, ale se zaručenou konzistencí metadat na disku dle grafu závislosti metadat uložených v paměti (viz *McKusick: Soft Metadata Update*).
   Princip: pokud má být zapsán blok, který je závislý na něčem, co ještě zapsáno nebylo, reverzují se v něm dočasně pro zápis všechny změny učiněné v paměti tak, aby odpovídaly stavu ostatních bloků na disku. Po zápisu se blok v paměti vrací do původního stavu. Tím je zaručena konzistence dat na disku i v paměti.
3. **Transakční zpracování** (žurnál změn, MS NTFS, Veritas VxFS, IBM JFS, Linux EXT3, SGI XFS, Sun ZFS):
   - seznam operací (buď jen metadata nebo včetně dat) → log
   - značka (commit) → log
   - provedení operací
   - značka (done) → log (nebo smazání logu)

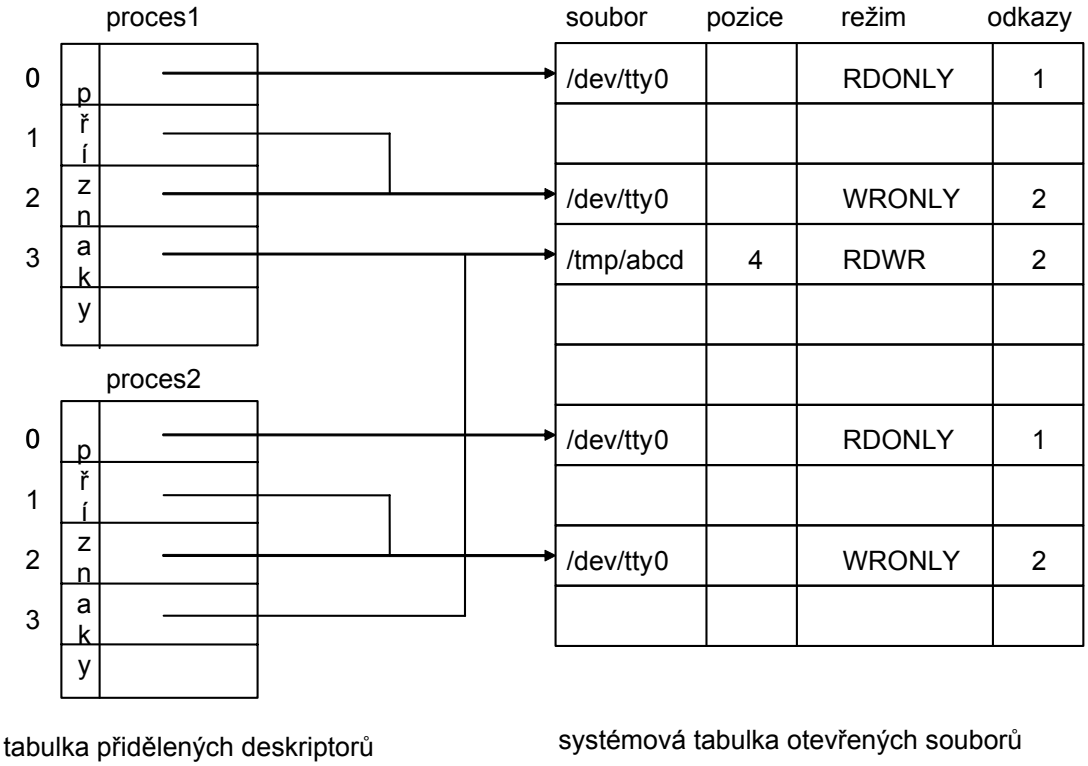
   Při výpadku se provedou operace z logu, které nebyly zapsány na disk při výpadku.

   Zajišťuje odolnost primárně pro metadata, ztrátu dat je třeba řešit synchronním režimem V/V (O_DSYNC)
   Sun/Illumos ZFS řeší i konzistenci dat – součástí metadat je signatura kopií datových bloků (RAID-1 nebo RAID-5), datové bloky se spravují metodou COW (copy on write), dokud není dokončena kompletní transakce, zůstávají původní datové bloky při zápisu nezměněny, zápis probíhá do nově alokovaných bloků.

3b) Přístup k souboru - čtení/zápis

Vyhledání souboru je náročná operace, nemá smysl ji opakovat pro každou operaci - otevřený soubor je identifikován v tabulce otevřených souborů, přistupuje se k němu pomocí deskriptoru.

Tabulka otevřených souborů



Odkazy ze dvou tabulek deskriptorů dvou různých procesů mohou identifikovat stejnou položku v systémové tabulce otevřených souborů - mohou *sdílet* otevřený soubor. Při tom je třeba dát pozor na to, že **sdílí i pozici v souboru**.

Spuštěný proces dědí tabulku deskriptorů z rodičovského procesu. Spuštění nového programu (*exec()*) nemění tabulku deskriptorů, což se využívá pro přesměrování vstupu (dekriptor 0), výstupu (1) a chybového výstupu (2). Je třeba dát pozor na další deskriptory, pokud není cílem je předat novému programu, měl by je dětský proces před *exec()* všechny **uzavřít**!

- pohyb po záznamech, alokačních blocích - zobrazení logické struktury na fyzickou
- změny v záznamech, alokačních blocích - zachování okolí změněných dat

3c) Manipulace se soubory

- Vesměs manipulace nad adresáři, metadaty
- Operace ustálené a podobné:
  - POSIX 1003.1: *stat(), link(), unlink(), chmod(), chown(), mkdir(), rmdir()*, ...
  - WIN32: *GetFileType(), GetFileTime(), GetFileSize(), GetFileAttributes, CreateHardLink(), DeleteFile(), GetFileSecurity(), SetFileSecurity(), CreateDirectory(), RemoveDirectory()*, ...
- Problém atomičnosti adresářových operací (*rename(), MoveFile()*), konzistence, odolnost vůči výpadku (viz a))
- Implementace v jádře (malá flexibilita) → mikrojádro

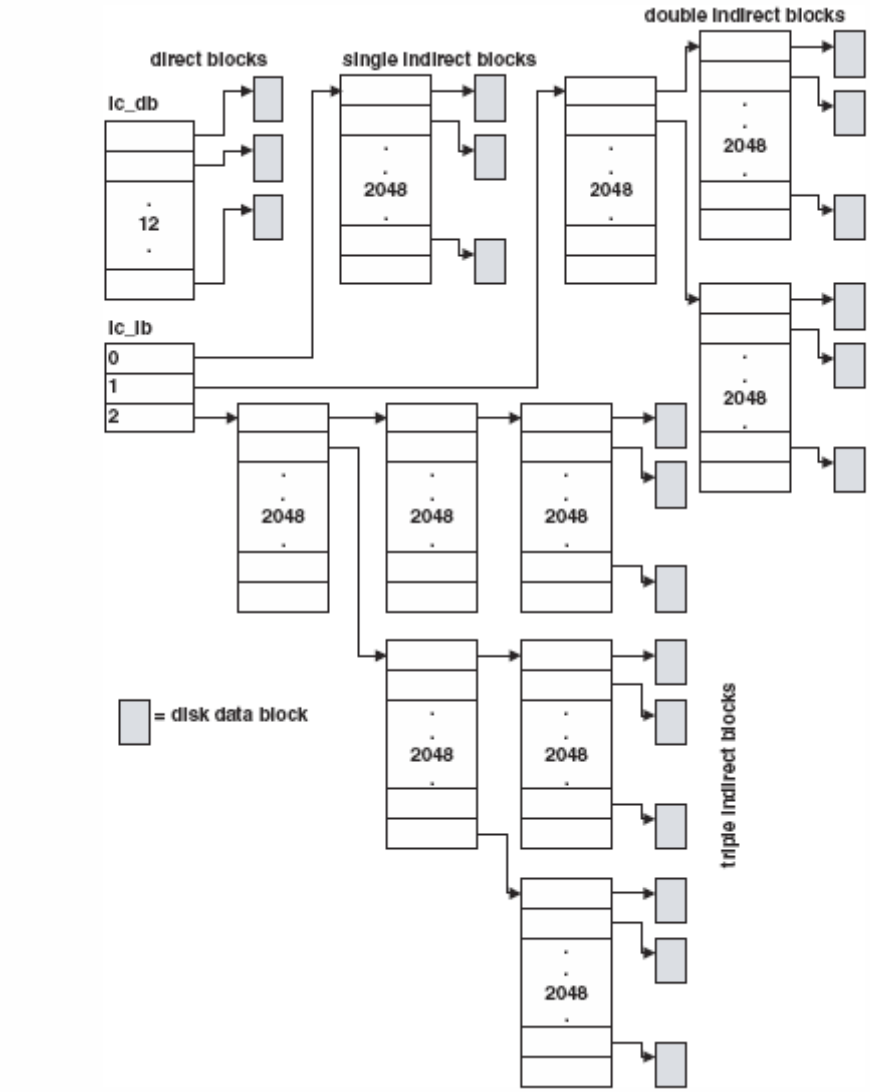
Příklady implementací (podrobněji viz Web předmětu)

Unix File System (UFS původní)

- alokační blok = 1 sektor (System V: 2-4 sektory, Solaris 16 s.)
- volné bloky - index sekvenční
- alokace - první volný blok bez optimalizace, seznam prvních 10 AB v i-uzlu, další 3 AB nepřímé 1, 2 a 3 úrovňové
- umístění informací o souborech - i-uzly na začátku disku

BSD Fast File System (FFS)

- větší alokační blok (4KB-16KB), možnost dělení jednoho AB na fragmenty (512-2KB) - pouze pro konec souboru
- volný prostor popsán bitovou mapou fragmentů
- volné i-nódy - bitová mapa
- diskový prostor rozdělen na skupiny cylindrů (CG, cylinder group), každá obsahuje kopii superbloku, bitovou mapu volných fragmentů, bitovou mapu volných i-uzlů, i-uzly, datové bloky
- maximální velikost skupiny cylindrů je dána maximální velikostí bitové mapy, která se musí vlézt do jednoho alokačního bloku (16KB = 128Kb = 256MB/2KB fragmenty)
- alokace prostoru - optimalizace (nejbližší volný blok, nové soubory do stejné skupiny, i-nódy pro položky adresáře ve stejné skupině, blok nového adresáře v nejvolnější skupině)
- fragmenty jsou alokovány pouze pro konec souboru po uzavření, pokud je menší než jeden alokoční blok. Když se zvětšuje, překopíruje se do plného alokačního bloku.
- původní FFS měl adresy AB a časy 32bitové (UFS1), UFS2 má vše 64bitové, navíc externí atributy (ACL) a snapshoty (snímky stavu).



Diskové adresy 8 byte, alokační blok 16 KB (dnes 32 KB), fragment 2 KB (4 KB), adresovatelné musí být fragmenty, max. velikost je 2^{64}*2KB

přímé adresy AB (frag.addr): 12 * AB = 192 KB

1. úroveň: (12+2048) * AB = 32 MB

2. úroveň: (12+2048 + 2048*2048) * AB = 64 GB

3. úroveň: 128 TB

(pro větší FS je třeba volit větší AB, lze až 64 KB)

EXT2/3 (Linux)
zjednodušený BSD FFS:

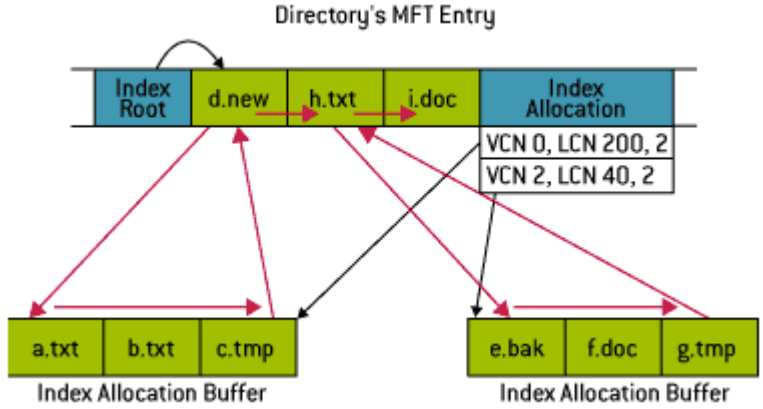
- skupiny cylindrů s vlastními tabulkami i-uzlů a bitovou mapou volných alokačních bloků
- bez fragmentů, alokační blok pouze 4KB (nebo 1, 2 KB)
- bez optimalizací FFS (pouze prealokace při zápisu a přidělování prostoru ve stejné skupině)
- omezená velikost souboru (2 GB)
- omezená velikost systému souborů (4G*AB jako UFS1)
- 32-bitová čísla i-nodů (max. 4G souborů)
- EXT3 doplňuje žurnál (jako soubor), struktura nezměněna
- EXT4 (2.6.28) řeší problém limitů velikosti, dovoluje proměnné velikosti AB, časové údaje 64bitové

Log Structured File System (LFS – BSD, NILFS - Linux)

- každý zápis dat i metadat se strádá v paměti a zapisuje po velkých blocích (1MB) chronologicky do logu na konec (data, metadata), všechny zápisy jsou sekvenční,
- každý blok logu obsahuje: změněné bloky, změněné indexy alokačních bloků, změněné i-nody, změněné bloky mapy i-nódů (mapuje i-nody na umístění obsahu v poslední verzi),
- data a metadata se čtou z poslední verze odpovídajích bloků v logu – nutná evidence alokovaných bloků a metadat v paměti,
- pravidelně se dělá checkpoint – zapíše se na pevné místo poslední platná verze mapy i-nodů,
- čtení dat je obecně pomalejší, protože nejsou alokační bloky souboru souvislé – větší vystavování (nevadí u SSD disků),
- při zaplnění logu je aktivován cleaner (prochází log od začátku a uvolňuje duplicitní bloky pro něž je novější kopie),
- odolnost proti výpadku - většina dat je vícekrát, přijde se pouze o poslední zápisy,
- vhodný pro SSD/SMR disky – nepřepisuje stále stejné místo.

NTFS

- Všechny soubory a adresáře jsou popsány v MFT (Master File Table, obdoba tabulky inodů) na začátku disku.
- Položka MFT (1KB) obsahuje hlavičku (42 byte) a seznam atributů pro daný soubor.
- Pokud se seznam atributů nevleze do jedné položky MFT, obsahuje atribut *Attribute List* odkazy na pokračovací položky v MFT (v jejich hlavičkách je odkaz zpět na bázový záznam).
- Každý atribut má typ, délku (záznamu), jméno (volitelně), alokovanou velikost, velikost obsahu, komprimovanou velikost a obsah.
- Atributy jsou buď rezidentní (obsah je přímo za hlavičkou atributu v MFT) nebo externí (obsah v AB mimo), data souboru jsou jedním z typů atributů. Obsah externích atributů je uložen v datových alokačních blocích. Jejich seznam ve formě komprimovaného *runlist* (VCN, LCN, délka) je součástí externího atributu.
- Mimálně je v položce MFT atribut *Standard Information* (časy, Security ID, Owner ID, R/W), *File Name* (jméno, další časy a index nařazeného adresáře, alokovaná velikost, skutečná velikost nepojmenovaného *Data*, jmen může být více), *Data* nebo *Index Root, Index Allocation* a *Bitmap* (pro adresář, B-strom jmen souborů v adresáři+čas a velikost).



- Typy atributů mohou být doplňovány, součásti MFT je tabulka (*AttrDef*) mapování jmen a typů atributů na interní id.

- Prvních 15 položek MFT je vyhrazených (*MFT, kopie, log, volume, attrdef, root dir, bitmap, boot, badclust, secure, upcase, extend*).

Příklad: NFI c:

```
File 28              Větší adresář
\WINDOWS
    $STANDARD_INFORMATION (resident)
    $FILE_NAME (resident)
    $OBJECT_ID (resident)
    $INDEX_ROOT $I30 (resident)
    $INDEX_ALLOCATION $I30 (nonresident)
        logical sectors 15490840-15491167
        logical sectors 32437496-32437503
        logical sectors 28623976-28623983
        logical sectors 28651624-28651631
    $BITMAP $I30 (resident)
File 29              Malý adresář
\WINDOWS\system32
    $STANDARD_INFORMATION (resident)
    $ATTRIBUTE_LIST (resident)
    $FILE_NAME (resident)
    $OBJECT_ID (resident)
    $INDEX_ROOT $I30 (resident)
    $BITMAP $I30 (resident)
File 8687            Malý soubor
 \CONFIG.SYS
     $STANDARD_INFORMATION (resident)
     $FILE_NAME (resident)
     $DATA (resident)
File 2453            Velký soubor
 \WINDOWS\system32\config\software
     $STANDARD_INFORMATION (resident)
     $FILE_NAME (resident)
     $DATA (nonresident)
         logical sectors 142760-213927
         logical sectors 25200568-25201079
         logical sectors 6494120-6494631
         logical sectors 27997936-27998447
         logical sectors 31597976-31598487
```

Porovnání systémů souborů

	max. velikost	***max. soubor***	***max. souborů***	***adresář***
FAT32	*8 TB*	*4 GB*	*4 G*	*lineární*
NTFS	*16 EB*AB*	*16 EB*	*4 G*	*B+ strom*
*NTFS**	*16 TB*		*AB=4KB*	
EXT2 4KB	*16 TB*	*2 TB*	*4G*	*lineární*
EXT4	*1 EB*	*16 TB*	*4 G*	*h-B strom*
UFS1	*2GB*AB*	*2 TB*	*NCG* AB*8*8*	*lineární*
UFS2	*16EB*AB*	*8 EB*	*NCG* AB*8*8*	*lineární*
XFS	*16 EB*	*8 EB*	*16 E*	*B+ strom*
ZFS	*16 EB*	*8 EB*	*256 T*	*B+ strom*

EB = Exabyte (1024*1024 TB)

Aktuální problémy:

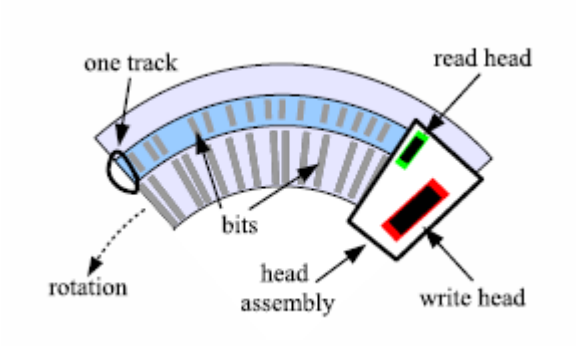
SSD (flash) disky:

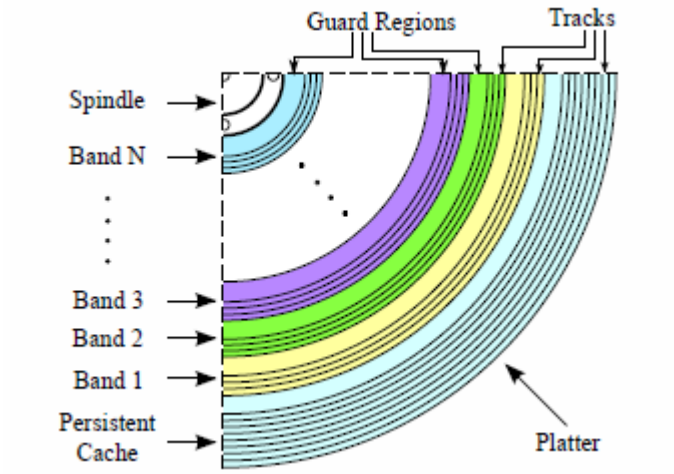
- náhodné čtení stejně rychlé jako sekvenční (není třeba defragmentace)
- zápis obvykle řádově pomalejší
- problém velikosti stránky (min. blok zápisu) – 4KB (32nm), 8KB (25nm) - nesnáší zápis malých bloků (sektory, write amplification)
- blok pro mazání značně větší, musí se mazat najednou celý (~256 KB, cca. 2 ms)
- počet zápisů do stejné stránky omezen (10k/5k/3k)
- rozpoznání uvolněných bloků (operace TRIM)
- vadí časté zápisy do stejného sektoru (např. aktualizace timestampu posledního přístupu v i-nodech nebo změnový žurnál)

4KB sektory – problém zarovnání metadat a alokačních bloků

SMR (Shingled Magnetic Recording) disky:

- zápisová hlava je větší, přepisuje několik stop
- zápis musí probíhat jen jedním směrem sekvenčně, několik otevřených oblastí na disku (řádově desítky MB), kam lze přidávat data
- čtení může probíhat kdekoli (nezúžená stopa je čtena jen z části)





Zápisy může disk řešit:

- autonomně (*drive-managed*, Seagate Archive 8TB), tváří se jako normální disk, ve skutečnosti nezapisuje na zadané adresy LBA, ale buď do otevřených oblastí zápisu (mapuje sektory podobně jako SSD disky) nebo do cache na začátku disku a pak v klidu přepisuje celé změněné oblasti najednou (adresy sektoru jsou pevné jako u normálních disků) - nevyžaduje změnu systémů souborů.
- ponechat na OS (*host-managed*, Hitachi Ultrastar Archive Ha10), akceptuje zápisy pouze na konce otevřených oblastí, ostatní zápisy odmítá. Pomocí SATA příkazů může systém zjistit počty a adresy začátků otevřených oblastí zápisu - vyžaduje speciální systém souborů (LFS).