

Správa souborů

ZOS 2024, 03 A 04, L. PEŠIČKA, V3

Souborové systémy

- potřeba aplikací **trvale** uchovávat data
- **hlavní požadavky**
 - možnost uložit velké množství dat
 - potřeba uchovávat strukturovaně => adresářový strom a soubory
 - informace zachována i po ukončení procesu
 - data přístupná více procesům
- **společné problémy** při přístupu k zařízení
 - alokace prostoru na disku
 - pojmenování dat
 - ochrana dat před neoprávněným přístupem
 - zotavení po havárii (výpadek napájení)

Soubor - definice

soubor je **pojmenovaná sada dat**, uložená na **datovém médiu**, se kterou lze pracovat nástroji operačního systému jako s **jedním celkem**

Soubor

- název
- velikost v bytech
- datový obsah

Soubor - příklad

- bakalarka.pdf
 - **Název souboru**: bakalarka.pdf
 - **Velikost v bytech**: 1 852 457 B
 - Časové značky (vytvoření, změny, případně posledního přístupu)
 - Přístupová práva (žádná, základní unixová, ACL)
 - Datový obsah (rozumí mu např. Acrobat Reader)
- Datovému obsahu rozumí program, který s daným typem souboru umí pracovat, např. Acrobat Reader -> .pdf, Word -> .docx
- Operační systém pracuje se souborem jako celkem, umí jej přesunout do jiného adresáře, přejmenovat, smazat, zpřístupnit procesu (běžící Word). Po otevření souboru lze se souborem pracovat pomocí operací read/write.
- Soubor **musí být umístěný v** některém **adresáři** (aby byl přístupný)

Souborový systém

Definice:

Způsob organizace dat ve formě **souborů (a adresářů)**, tak aby k nim bylo možné snadno přistupovat.

Souborový systém – **popisuje způsob uložení souborů** v elektronické paměti, která je buď v počítači (**disková oblast, CD**), nebo zpřístupněna přes počítačovou síť (**Samba, NFS**).

Souborový systém říká, **jak** jsou soubory na disk ukládány

Souborový systém

- souborový systém (file system, fs)
 - pravidla pro ukládání souborů a snadný přístup k nim
 - datové struktury a algoritmy
 - Prostor na ukládání dat je členěn na datové bloky stejné velikosti.
 - Jak poznám, který datový blok je volný a který obsazený?
 - Jak je určeno, že určitý blok patří danému souboru?
 - Jak jsou organizované adresáře?
 - Jsou nějak řešena přístupová práva?
- část OS, poskytuje mechanismus pro ukládání a přístup k datům

Souborové systémy(fs)

- Současné OS – implementují více různých fs
 - kompatibilita (starší verze daného OS, ostatní OS)
- Windows:
 - primární je **NTFS**
 - ostatní: FAT12, FAT16, **FAT32**, **exFAT**, ISO 9660 (CD-ROM), ReFS (FAT16 - 16ti bitové adresy clusterů, FAT32 - 32bitové adresy, ale používá se 28 bitů)
- Linux
 - ext2, ext3, **ext4**
 - **XFS**, **btrfs (B-tree fs)**, **ZFS**, **JFS**, **ReiserFS**
 - ostatní: FAT12 až 32, ISO 9660, Minix, VxFS, OS/2 HPFS, SysV fs, UFS, NTFS

Základní znalosti

- V PC můžeme mít **více pevných disků**:
Např. v Linuxu: `/dev/sda`, `/dev/sdb`
- Každý disk se může dělit na **několik oddílů (partitions)**:
`/dev/sda1`, `/dev/sda2`, `/dev/sda3` (1. disk má 3 oddíly)
`/dev/sdb1` (2. disk má 1 oddíl)
- Každý oddíl – nějaký **filesystem**:
`/dev/sda1` ext4
`/dev/sda2` swap (odkládací paměť)
`/dev/sda3` ntfs
`/dev/sdb1` fat32

Základní znalosti

`fdisk /dev/sda`

- **Rozdělení** disku na oddíly (partitions)
- Zobrazení a možnost toto rozdělení změnit

`/sbin/mkfs.ext4 /dev/sda1`

- **Formátování** oddílu na vybraný filesystem (zde ext4)
- Formátování
 - inicializace paměti, aby byla připravena k použití
 - zapíše na disk metadata popisující prázdné médium dle pravidel daného souborového systému

Dělení disku na oddíly

2 způsoby dělení disku

- **Master Partition Table (MPT)**
 - Master Boot Record (MBR) – na počátku disku
 - Umožňuje **4 oddíly primární**
 - Chceme-li jich více, uděláme **3 primární a 1 extended**
 - Extended lze dělit na další oddíly
- **GUID Partition Table (GPT)**
 - Nelimituje na 4 oddíly, např. Microsoft 124 oddílů
 - Používá např. Mac OS, ale i dnešní PC s Windows

Struktura MBR

Struktura MBR				
Adresa			Popis	Délka v bajtech
Hex	Oct	Dec		
0000	0000	0	Kód zavaděče 	440 (max 446)
01B8	0670	440	Volitelná signatura disku	4
01BC	0674	444	Obvykle nuly; 0x0000	2
01BE	0676	446	Tabulka rozdělení disku (MPT) (4 položky po 16 bajtech, IBM schéma oddílů)	64
01FE	0776	510	55h	2
01FF	0777	511	AAh	
Celková délka MBR: 446 + 64 + 2 =				512

Zdroj obr.: wikipedia

512 bytů

Na začátku disku je MBR záznam, který obsahuje:

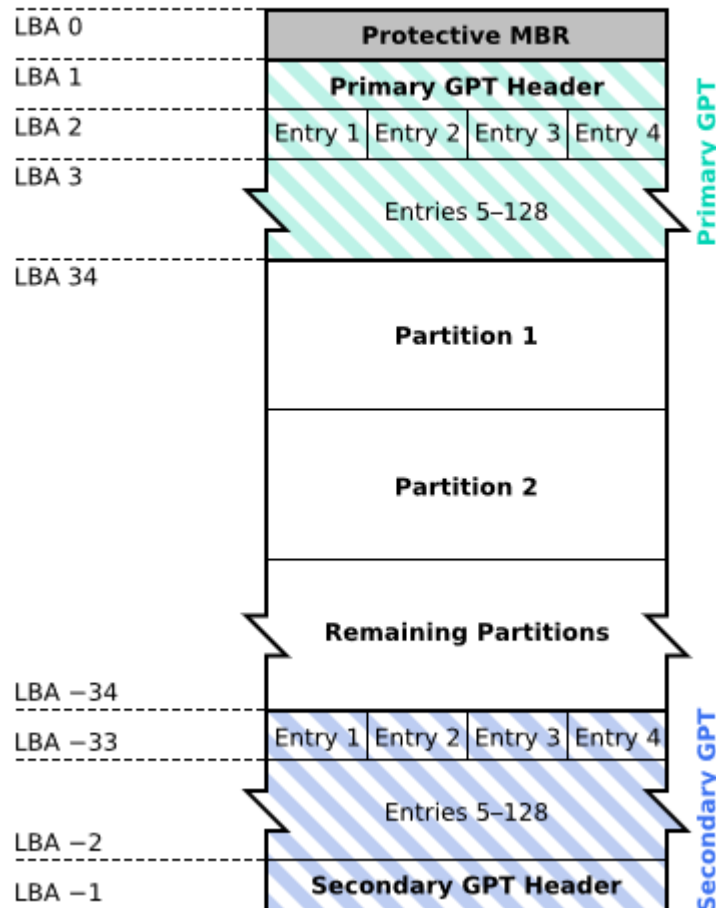
Kód zavaděče

(pokud se z disku bootuje, tak tento kód je puštěn z BIOSu)

Tabulka rozdělení disku
(4 partitions)

GUID Partition Table (GPT)

GUID Partition Table Scheme



Nejsme omezeni na 4 primární oblasti disku

Součástí standardu **UEFI**, který postupně nahradil klasický BIOS (umožňuje např. secure boot)

Zdroj obr.: wikipedia

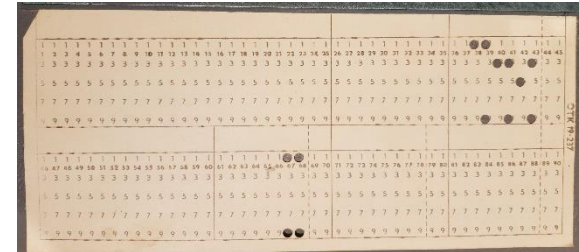
GPT

- záložní kopii tabulky ukládá na konci disku
- Velikost GPT je na disku s 512B sektory $34 \times 512B = 16KB$
- LBA (Logical Block Addressing)
 - Metoda adresování sektorů na pevném disku pomocí indexu
 - velikost **512B** (tradiční), ale jsou i jiné, např. novější **4096B**
 - Dnešní nové disky 4096B (např. 4TB, 10TB disk)
 - Číslování bloků lineárně od nuly
 - Nástupce původní adresace CHS (cylinder – hlava – sektor)
 - <https://neosmart.net/blog/2015/chs-lba-and-4k-advanced-format-drives/>
- První oddíl začíná na LBA 34

Historický vývoj

- první systémy (děrné štítky, děrné pásky)

- vstup – děrné štítky, výstup – tiskárna
- soubor = množina děrných štítků



Zdroj obrázku: wikipedia

- magnetické pásky

- vstup i výstup – pásky
- soubor = pojmenovaná množina záznamů na magnetické pásce
- záznam = strukturovaný datový objekt tvořený konečným počtem položek

- Magnetické a optické disky

- SSD disky (bez pohyblivých částí)

Soubor

- Soubor je **pojmenovaná** sekvence bytů.
- Operační systém poskytuje abstrakci fs, aby programy mohly pracovat stejným způsobem s různými fs.

Soubor

- Jeden druh dat (textový, obrázek, zvuk, program)
- Složený (archiv .zip, ISO obraz disku atp.)

Programátora většinou nezajímá, na jakém fs soubor leží, potřebuje jen být schopen s ním pracovat.

Uživatelské rozhraní fs (file systémů)

- vlastnosti fs z pohledu uživatele
 - konvence pro pojmenování souborů
 - vnitřní struktura souboru
 - typy souborů
 - způsob přístupu
 - atributy a přístupová práva
 - služby OS pro práci se soubory



Konvence pro pojmenování souborů

- vytvoření souboru – proces určuje jméno souboru
- různá pravidla pro vytváření jmen
- Windows x Unix a Linux
- rozlišuje systém malá a velká písmena?
 - Win32API nerozlišuje: **ahoj**, **Ahoj**, **AHOJ** stejná
 - UNIX rozlišuje: **ahoj**, **Ahoj**, **AHOJ** rozdílná jména

Ukázka

Snaha vytvořit textový soubor ahoj.txt a Ahoj.txt ve stejném adresáři pod Windows:

Tento počítač > Nový svazek (D:) > work > a

Název	Datum změny	Typ	Velikost
 Ahoj (2).txt	05.12.2016 9:24	Textový dokument	0 kB
 ahoj.txt	05.12.2016 9:24	Textový dokument	0 kB

Vyzkoušejte v Linxu na eryxu (budou 2 odlišné soubory):

vim ahoj.txt

vim Ahoj.txt

Pojmenování souborů

- jaká může být délka názvu souboru?
 - Do Windows 10 AU - 256 znaků NTFS (včetně cesty)
 - Windows 10 AU – lze limit překonat
(<https://www.howtogeek.com/266621/how-to-make-windows-10-accept-file-paths-over-260-characters/>)
 - UNIX obvykle alespoň 256 znaků (dle typu fs)
- množina znaků?
 - všechny běžné – názvy písmena a číslice
 - znaková sada UNICODE
 - βετα – legální jméno souboru
 - Linux – všechno kromě / a char(0)

Pojmenování souborů

■ přípony?

- MS DOS – jméno souboru 8 znaků + 3 znaky přípona
- Dnešní OS: Windows 10, Linux – klidně i více přípon nebo žádná

■ další omezení?

- Některé OS – mezera nesmí být první a poslední znak
- Zamyslet se, zda něco takového opravdu potřebujeme
- Ale Windows 10:
`mkdir "sMezerou"`
`cd "sMezerou"`

Typy souborů

OS podporují více typů souborů:

- **obyčejné soubory**

- data zapsaná aplikacemi
- obvykle rozlišení textové x binární
- **textové** - řádky textu ukončené znaky CR (MAC), LF (UNIX), nebo CR+LF (MS DOS, Windows)
- binární – všechny ostatní
- OS rozumí strukturu **spustitelných** souborů (binárky, skripty)

Typy souborů

■ adresáře

- udržují strukturu souborového systému
- můžeme si zobrazit obsah adresáře (ls) a nastavit jej jako pracovní (cd)

■ speciální - Linux , UNIX:

- znakové speciální soubory (/dev/tty)
- blokové speciální soubory (/dev/sda)
 - bloky můžeme cachovat, přeskakovat atp.
- pojmenované roury (příkazy mkfifo, mknod)
- symbolické odkazy (příkaz ln -s)

Vnitřní struktura (obyčejného) souboru

- Vnitřní struktura souboru
 - **nestrukturovaná posloupnost bytů**
 - posloupnost záznamů
 - strom záznamů
- **nestrukturovaná posloupnost bytů** (nejčastěji)
 - OS obsah souboru nezajímá, interpretace je na aplikacích
 - maximální flexibilita
 - Nestrukturovaná z pohledu OS, ale programy mohou strukturovat, jak chtějí

Vnitřní struktura souboru – posloupnost záznamů

■ posloupnost záznamů pevné délky

- každý záznam má vnitřní strukturu
- Operace
 - čtení – vrátí záznam,
 - zápis – změnění / přidá záznam
- v historických systémech, dnes se téměř nepoužívá
- záznamy 80 znaků obsahovaly obraz děrných štítků

Vnitřní struktura souboru – strom záznamů

■ strom záznamů

- záznamy nemusejí mít stejnou délku
- záznam obsahuje pole **klíč** (na pevné pozici v záznamu)
- záznamy **seřazený** podle klíče, aby bylo možné vyhledat záznam s požadovaným klíčem
- mainframy pro komerční zpracování dat
- Také historické

Způsob přístupu k souboru (!!)

Sekvenční nebo přímý

- **sekvenční** přístup

- procesy mohou číst data pouze v pořadí, v jakém jsou uloženy v souboru
- tj. od prvního záznamu (bytu), nemohou přeskakovat
- možnost „přetočit pásku“ a číst opět od začátku, **rewind()**
- hlavně v prvních OS, kde data na magnetických páskách

Způsob přístupu k souboru (!!)

- **přímý** přístup (random access file)
 - čtení v libovolném pořadí nebo podle klíče
 - přímý přístup je nutný např. pro databáze
 - uživatel – např. přeskakování děje filmu
 - určení začátku čtení
 - každá operace určuje pozici
 - OS udržuje pozici čtení / zápisu, novou pozici lze nastavit speciální operací **seek**

Pokud nemám operaci seek, musím přistupovat sekvenčně

Způsob přístupu k souboru

- všechny běžné současné OS – soubory s přímým přístupem
- v některých OS pro mainframy:
 - při vytvoření souboru se určilo, zda je sekvenční nebo s přímým přístupem
 - OS mohl používat rozdílné strategie uložení souboru
- Speciální RT systémy – někdy mohou využít sekvenční soubory

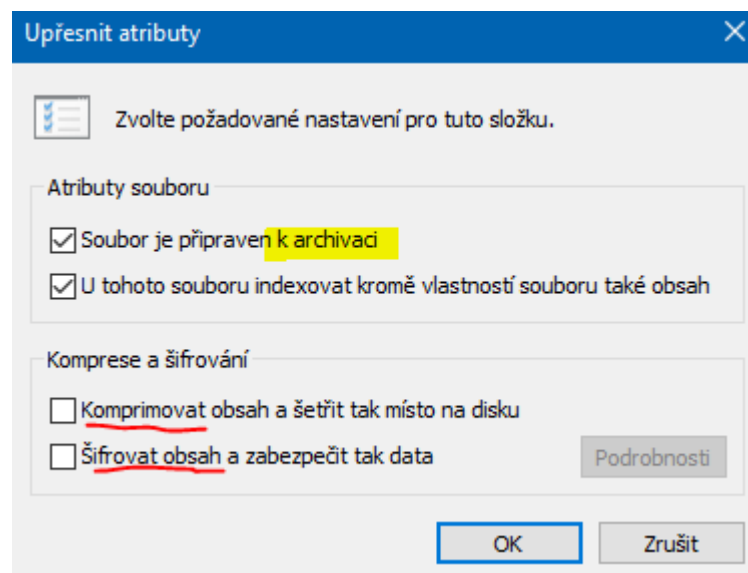
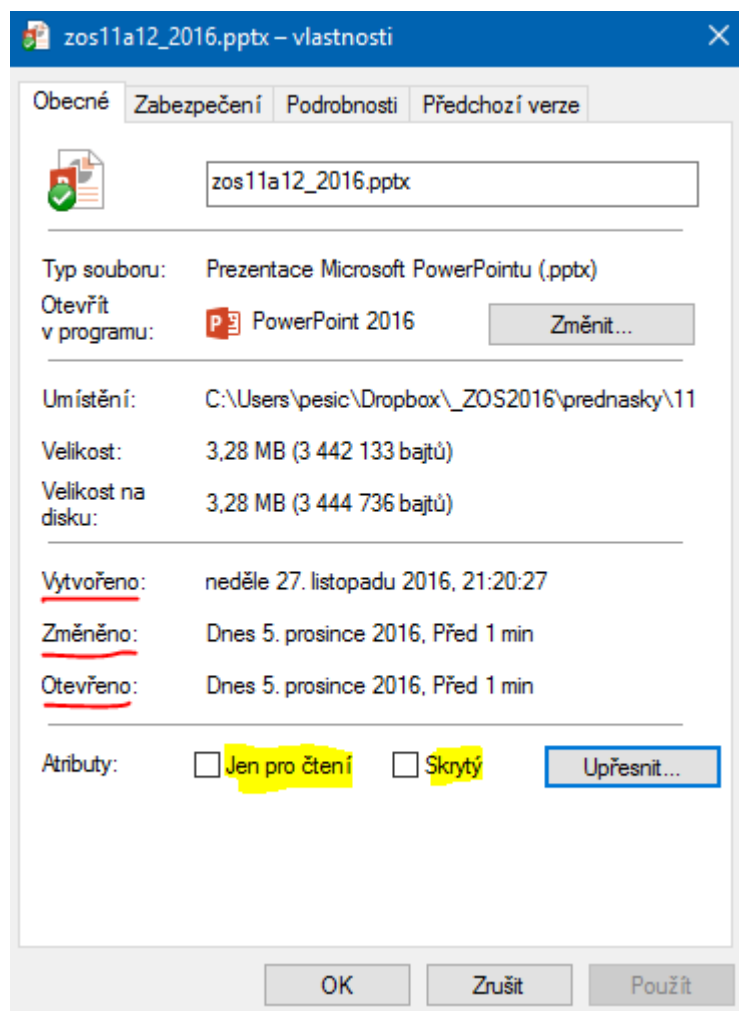
Atributy

- informace sdružená se souborem
- některé atributy interpretuje
 - OS
 - jiné systémové programy
 - a jiné aplikace
- významně se liší mezi jednotlivými OS
- ochrana souboru
 - kdo je vlastníkem, množina přístupových práv, heslo, ...

Atributy - pokračování

- příznaky
 - určují vlastnosti souboru
 - **hidden** – neobjeví se při výpisu
 - **archive** – soubor nebyl zálohován
 - **temporary** – soubor bude automaticky zrušen
 - **read-only**,
 - text/binary, random access, ...
- velikost, datum vytvoření, poslední modifikace, poslední přístup

Atributy – ukázka Windows 10



Služby OS pro práci se soubory

- většina současných – základní model dle UNIXu

Základní pravidla

- **veškerý I/O prováděn pouze pomocí souborů**
 - obyčejné soubory – data, spustitelné programy
 - zařízení – disky, tiskárny
 - se všemi typy - zacházení pomocí **stejných** služeb systému
- obyčejný soubor – posloupnost bytů
 - význam znají pouze programy, které s ním pracují
 - interní struktura souboru OS nezajímá
- seznam souborů – **adresář**
 - adresář je také soubor (!)
 - adresář obsahuje informace o souborech a podadresářích

Základní pravidla - dokončení

- speciální soubory – pro přístup k zařízením
 - Linux – například `/dev/sda`, `/dev/tty`
- MS DOS používal označení – `PRN:`, `COM1:`

Jednotný přístup nebyl vždy

- Před příchodem UNIXu jednotný přístup nebyl samozřejmostí
- Téměř všechny moderní systémy základní rysy UNIX modelu převzaly
- většina systémů před UNIXem
 - samostatné služby pro čtení / zápis terminálu, na tiskárnu do souboru

systémy poskytovaly „více služeb“ - komplexní

x

model podle UNIXu – podstatně menší složitost

Základní služby pro práci se soubory

■ otevření souboru

- než s ním začneme pracovat
- úspěšné – služba pro otevření souboru vrátí **popisovač souboru (file descriptor)** – malé celé číslo
- popisovač souboru používáme v dalších službách
 - čtení apod.

Základní služby pro práci se soubory

otevření souboru:

fd = open (jmeno, způsob)

- jméno – řetězec pojmenovávající soubor
- způsob – pouze pro čtení, zápis, obojí
- fd – vrácený popisovač souboru

nalezne informace o souboru na disku a vytvoří pro soubor potřebné datové struktury

popisovač souboru – **index to tabulky souborů** uvnitř OS

Základní služby pro práci se soubory

vytvoření souboru:

`fd=creat(jméno, práva)`

- vytvoří nový soubor s daným jménem a otevře pro zápis
- pokud soubor existoval – zkrátí na nulovou délku
- fd – vrácený popisovač souboru

Opravdu je creat nikoliv create

Základní služby pro práci se soubory

operace **čtení** ze souboru:

read(fd, buffer, počet_bytů)

- přečte *počet_bytů* ze souboru *fd* do *bufferu*
- může přečíst méně – zbývá v souboru méně
- přečte 0 bytů – konec souboru

Základní služby pro práci se soubory

operace **zápisu** do souboru

write (fd, buffer, počet_bytů)

- Zapiše do souboru z bufferu daný počet bytů
- Zápis uprostřed souboru – přepíše, na konec – prodlouží

read() a write()

- vrací počet skutečně zpracovaných bytů
- jediné operace pro čtení a zápis
- samy o sobě poskytují sekvenční přístup k souboru

Základní služby pro práci se soubory

nastavení pozice v souboru:

lseek (fd, offset, odkud)

nastaví offset příští čtené/zapisované slabiky souboru

odkud

- od začátku souboru
- od konce souboru (záporný offset)
- od aktuální pozice

poskytuje **přímý přístup** k souboru („přeskakování děje filmu“)

Základní služby pro práci se soubory

zavření souboru

close (fd)

uvolní datové struktury alokované OS pro soubor

Památovat

služba	popis
creat	Vytvoří soubor
open	Otevře soubor
read	Čtení
write	Zápis
lseek	Přímý přístup k souboru změna pozice pro čtení, zápis
close	Uzavření souboru

Příklad použití rozhraní – kopírování souboru

```
int src, dst, in;                                /* není to kompletní program, jen ukázka */
src = open("puvodni", O_RDONLY);                /* otevření zdrojového souboru */
dst = creat("novy", MODE);                       /* vytvoření cílového souboru */
while (1)
{
    in = read(src, buffer, sizeof(buffer));      /* čteme */
    if (in == 0)                                  /* konec souboru? */
    {
        close(src);                               /* zavřeme soubory */
        close(dst);
        return;                                    /* ukončení */
    }
    write(dst, buffer, in);                       /* zapíšeme přečtená data */
}
```

Další služby pro práci se soubory

- změna přístupových práv, zamykání, ...
- závislé na konkrétních mechanismech ochrany
- např. UNIX
 - zamykání `fcntl (fd, cmd)`
 - zjištění informací o souboru (typ, příst. práva, velikost) !
 - `stat (file_name, buf)`, `fstat (fd, buf)`
 - *man stat*, *man fstat*

Paměťově mapované soubory (!!)

- někdy se může zdát open/read/write/close nepohodlné
- možnost mapování souboru do adresního prostoru procesu
- služby systému `mmap()`, `munmap()`
- mapovat je možné i jen část souboru
- k souboru pak přistupujeme např. přes pointery v C

Paměťově mapované soubory - příklad

- délka stránky 4KB
- soubor délky 64KB
- chceme mapovat do adresního prostoru od adresy 512KB
- $512 * 1024 = 524\,288$.. od této adresy mapujeme
- 0 až 4KB souboru bude mapováno na 512KB – 516KB
- čtení z 524 288 čte první byte souboru atd.

Implementace paměťově mapovaných souborů

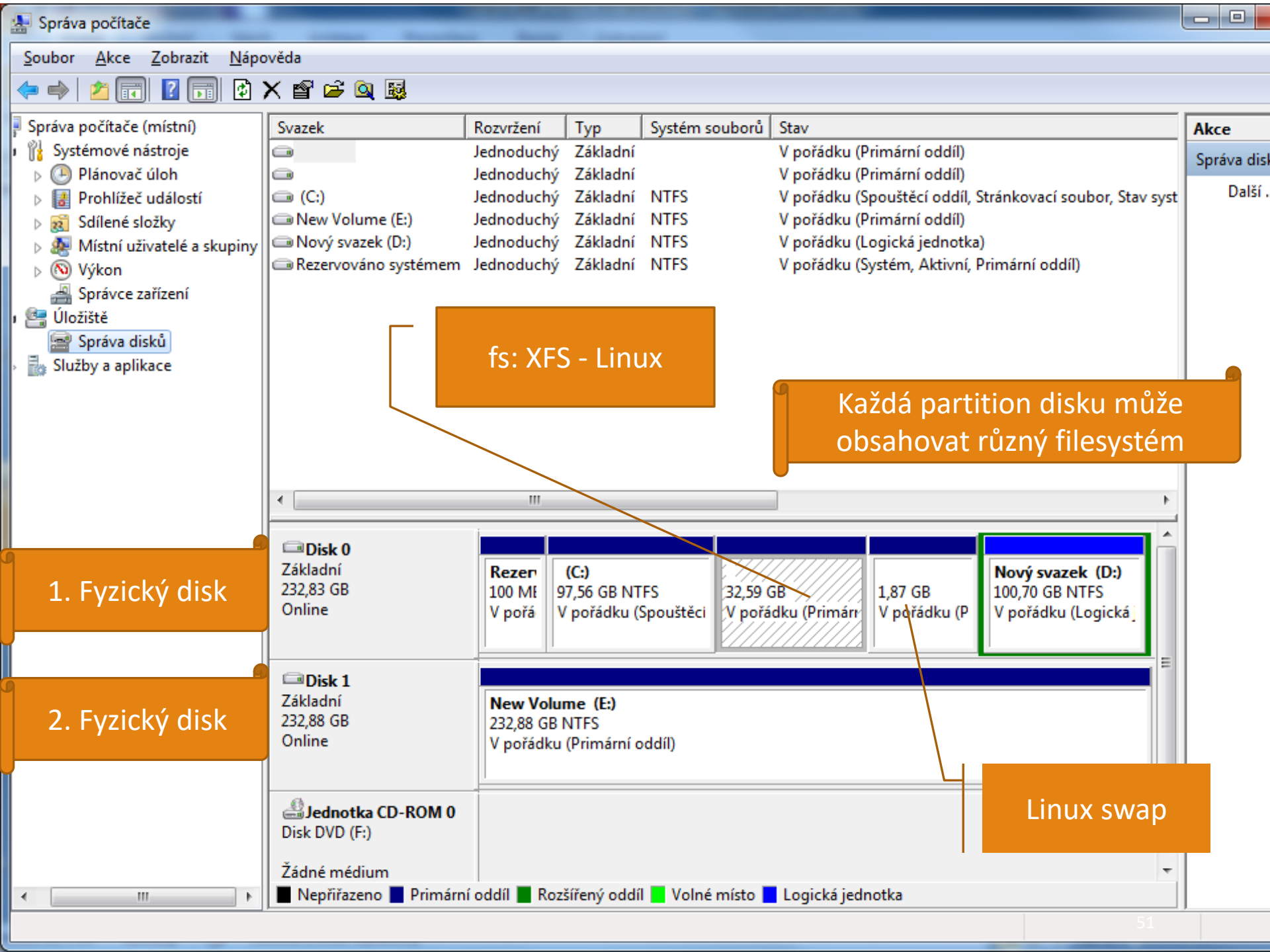
- OS použije soubor jako **odkládací prostor** (swapping area) pro určenou část virtuálního adresního prostoru
- čtení / zápis na adr. 524 288 způsobí **výpadek stránky**
- do rámce se **načte** obsah první stránky souboru
- pokud je modifikovaná stránka vyhozena (nedostatek volných rámců), **zapíše** se do souboru
- po skončení práce se souborem se **zapíše** všechny modifikované stránky

Problémy paměťově mapovaných souborů

- nejmenší jednotka je stránka
- problém nekonzistence pohledů na soubor, pokud je zároveň mapován a zároveň se k němu přistupuje klasickým způsobem

Adresářová struktura

- Jeden oddíl disku obsahuje jeden filesystem (fs)
- filesystem – 2 součásti:
 - množina souborů, obsahujících data
 - adresářová struktura – udržuje informace o všech souborech v daném fs
- adresář překládá jméno souboru na informace o souboru (umístění, velikost, typ ...)



- Správa počítače (místní)
- Systémové nástroje
 - Plánovač úloh
 - Prohlížeč událostí
 - Sdílené složky
 - Místní uživatelé a skupiny
 - Výkon
 - Správce zařízení
- Úložiště
 - Správa disků
 - Služby a aplikace

Svazek	Rozvržení	Typ	Systém souborů	Stav
	Jednoduchý	Základní		V pořádku (Primární oddíl)
	Jednoduchý	Základní		V pořádku (Primární oddíl)
(C:)	Jednoduchý	Základní	NTFS	V pořádku (Spouštěcí oddíl, Stránkovací soubor, Stav syst
New Volume (E:)	Jednoduchý	Základní	NTFS	V pořádku (Primární oddíl)
Nový svazek (D:)	Jednoduchý	Základní	NTFS	V pořádku (Logická jednotka)
Rezervováno systémem	Jednoduchý	Základní	NTFS	V pořádku (Systém, Aktivní, Primární oddíl)

- Akce
- Správa disků
- Další...

fs: XFS - Linux

Každá partition disku může obsahovat různý filesystem

1. Fyzický disk

2. Fyzický disk

Linux swap

Disk 0

Základní

232,83 GB

Online

Rezer

100 MB

V pořá

(C:)

97,56 GB NTFS

V pořádku (Spouštěcí

32,59 GB

V pořádku (Primární

1,87 GB

V pořádku (P

Nový svazek (D:)

100,70 GB NTFS

V pořádku (Logická

Disk 1

Základní

232,88 GB

Online

New Volume (E:)

232,88 GB NTFS

V pořádku (Primární oddíl)

Jednotka CD-ROM 0

Disk DVD (F:)

Žádné médium

Nepřiřazeno

Primární oddíl

Rozšířený oddíl

Volné místo

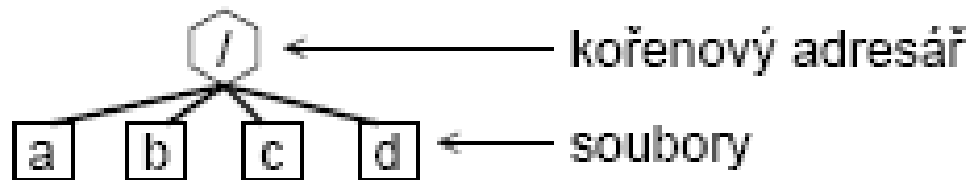
Logická jednotka

Základní požadavky na adresář (příkazy)

- procházení souborovým systémem (**cd**)
 - `cd /`, `cd ..`, `cd adr1`, `cd /etc`
- výpis adresáře (**ls**)
 - `ls`, `ls -l`, `ls -la`
- vytvoření a zrušení souboru a adresáře (**rm**, **rmdir**)
- přejmenování souboru (**mv**)
 - `mv` kromě přejmenování i přesun do jiného adresáře
- dále schémata logické struktury adresářů
 - odpovídá historickému vývoji OS

Schémata logické struktury adresářů

- **jednoúrovňový adresář**
- původní verze MS DOSu (pozdější verze už ne!)
- všechny soubory jsou v jediném adresáři
- všechny soubory musejí mít jedinečná jména
- problém zejména pokud více uživatelů

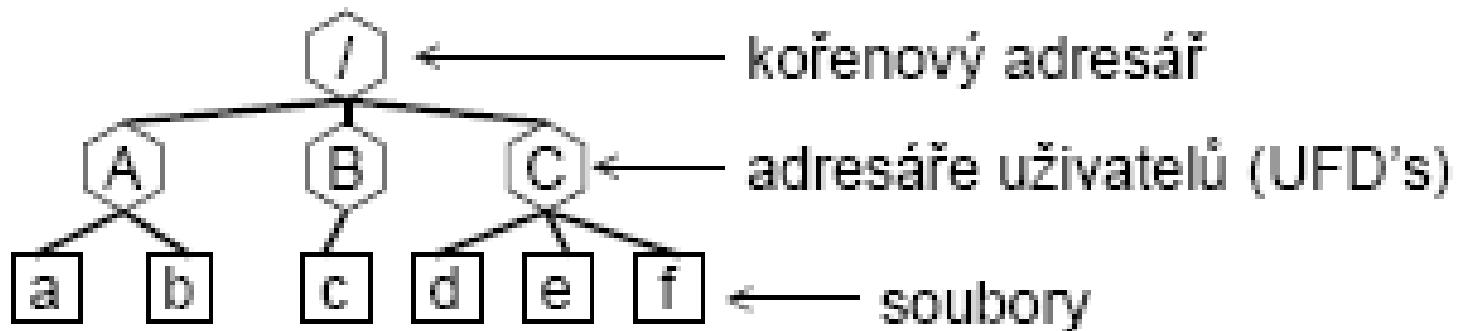


nelze mít dva soubory
nazvané příklad.c

Dvouúrovňový adresář

- adresář pro **každého uživatele** (User File Directory, UFD)
- OS prohledává pouze UFD , nebo pokud specifikováno adresář jiného uživatele **[user] file**
- výhoda – 2 uživatelé mohou mít soubor se stejným názvem
- speciální adresář pro systém
 - příkaz se hledá v adresáři uživatele
 - pokud zde není, vyhledá se v systémovém adresáři

Dvouúrovňový adresář – pokr.



každý uživatel může být nanejvýš jeden soubor
nazvaný priklad.c

Adresářový strom

- zobecnění předchozího
- např. MS DOS, Windows NT
- adresář – množina souborů a adresářů
- souborový systém začíná kořenovým adresářem
 - Značený „/“ na Unixu
 - u MS DOS „\“, protože znak / se používal pro volby
- cesta k souboru – jméno uvedené ve volání open, creat
 - absolutní
 - relativní

Cesta k souboru

absolutní cesta začíná:
/ (Linux)
C:\ (windows)

■ absolutní

- kořenový adresář a adresáře, kudy je třeba projít, název souboru
- oddělovače adresářů – znak „/“
- např. */home/user/data/v1/data12.txt*

■ relativní

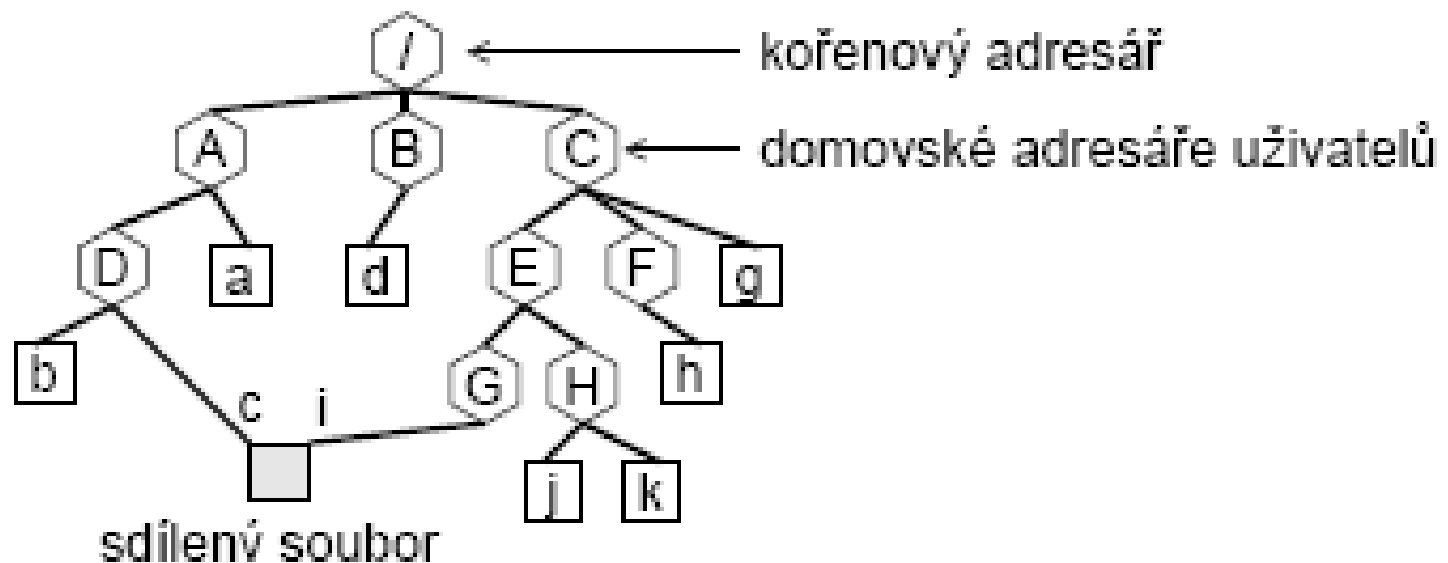
- aplikace většinou přistupují k souborům v jednom adresáři
- defaultní prefix = pracovní adresář
- cesta nezačíná znakem /
- př. *data12.txt* , *data/v1/data12.txt*

Acyklický graf adresářů

- **linky** na stejný soubor z více míst
- použití: týmová spolupráce na určitém projektu
- **sdílení společného souboru** nebo podadresáře
 - stejný soubor (adr.) může být viděn ve dvou **různých** adresářích
- flexibilnější než strom, komplikovanější
- **rušení souborů / adresářů** – kdy můžeme zrušit?
 - se souborem sdružen počet odkazů na soubor z adresářů
 - každé zrušení sníží o 1, když 0 = není odkazován
- jak **zajistit aby byl graf acyklický**?
 - algoritmy pro zjištění cyklu, drahé pro fs



Acyklický graf adresářů



stejný soubor viděný v různých cestách

Obeční graf adresářů

- obtížné zajistit, aby graf byl acyklický
- prohledávání grafu
 - omezení počtu prošlých adresářů (Linux)
- rušení souboru
 - pokud cyklus, může být počet odkazů > 0 i když je soubor již není přístupný
 - garbage collection – projít celý fs, označit všechny přístupné soubory; zrušit nepřístupné; (drahé, zřídka používáno)

Nejčastější použití

- adresářový strom (MS DOS)
- acyklický graf (UNIX, Linux)
hard links – sdílení pouze souborů – nemohou vzniknout cykly
 - In adr1 hlink
 - v Linuxu vypíše: **hard link not allowed for directory**
 - Hard linky nejsou povolené na adresáře
 - Hard linky lze vytvářet pouze uvnitř systému souborů
- POZOR!
Je nutné si uvědomit rozdíl mezi pojmy adresářový strom a acyklický graf.

Základní služby pro práci s adresáři (!)

- téměř všechny systémy dle UNIXu

pracovní adresář – služby:

- **chdir (adresář)**
 - nastavení pracovního adresáře
- **getcwd (buffer, počet_znaků)**
 - zjištění pracovního adresáře

Práce s adresářovou strukturou (!)

vytváření a rušení adresářů

- mkdir (adresář, přístupová_práva)
- rmdir (adresář) – musí být prázdný

zrušení souboru

- remove (jméno_souboru)
- volá **unlink()** pro soubory, rmdir() pro adresáře

přejmenování souboru

- rename (jméno_souboru, nové_jméno)
- provádí také přesun mezi adresáři

Práce s adresářovou strukturou

čtení adresářů – UNIX / POSIX

- DIRp = **opendir** (adresář)
 - otevře adresář
- položka = **readdir** (DIRp)
 - čte jednotlivé položky adresáře
- **closedir** (DIRp)
 - zavře adresář
- **stat** (jméno_souboru, statbuf)
 - info o souboru, viz man 2 stat

př. DOS: findfirst / findnext

ke všem uvedeným
voláním získáte v
Linuxu
podrobnosti pomocí:

man 2 opendir
man 2 readdir
man 2 stat

Pevný disk: cluster, sektor

Cluster

- nejmenší **alokovatelná** jednotka pro uložení souboru
- Logická jednotka, kterou používá souborový systém
- Soubor velikosti 1B zabere celý cluster
- Např. FAT32 – 32bitová adresa
(z nich používá jen 28bitů pro čísla clusterů, 4 bity for future use)
- Skládá se z 1 nebo více sektorů

Sektor

- Nejmenší fyzická jednotka pro ukládání dat na pevném disku (zařízení)
- Data na disku jsou ukládána do sektorů, každý sektor má jedičnou adresu
- **512B, 4KB, ...**

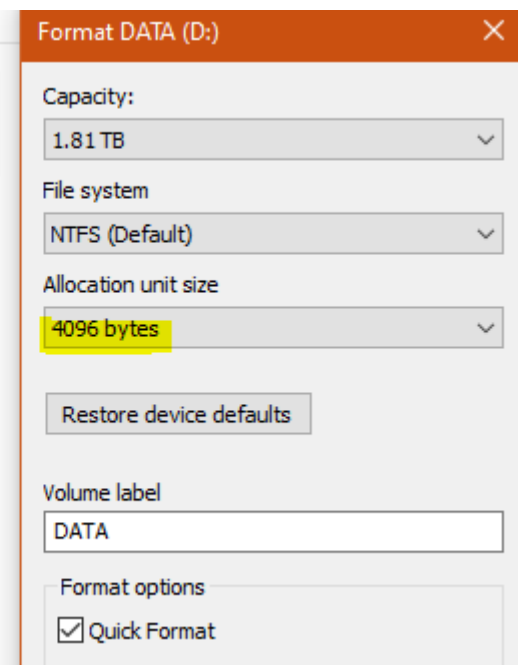
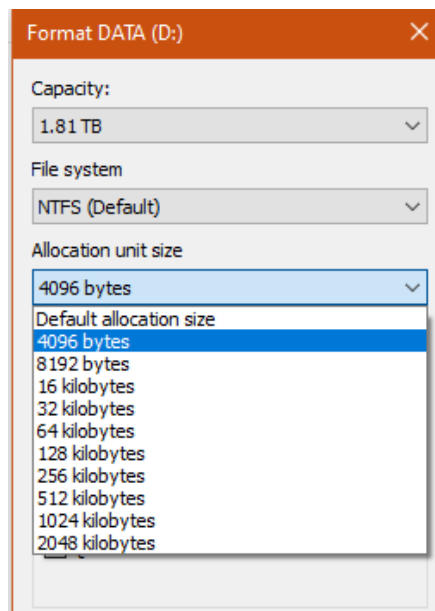
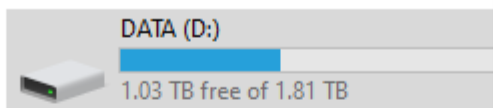
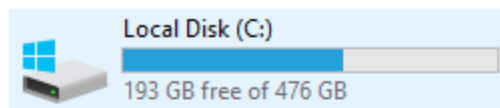
Příklady

- 512B cluster - 512B sektor disku
- 4KB cluster – 8 sektorů po 512B x nebo 1 sektor 4KB
- <https://notebook.cz/clanky/technologie/2015/advanced-format-4kb-disky>

Jakou velikost clusteru používáme?

Win: Správce souborů – disk – pravá myš – zvolit Format 😊

Devices and drives (3)



LBA vs cluster

<https://stackoverflow.com/questions/5774164/lba-and-cluster>

main difference is that
a **cluster** number addresses a group of continous sectors,
while **LBA** addresses a single sector on the disk.

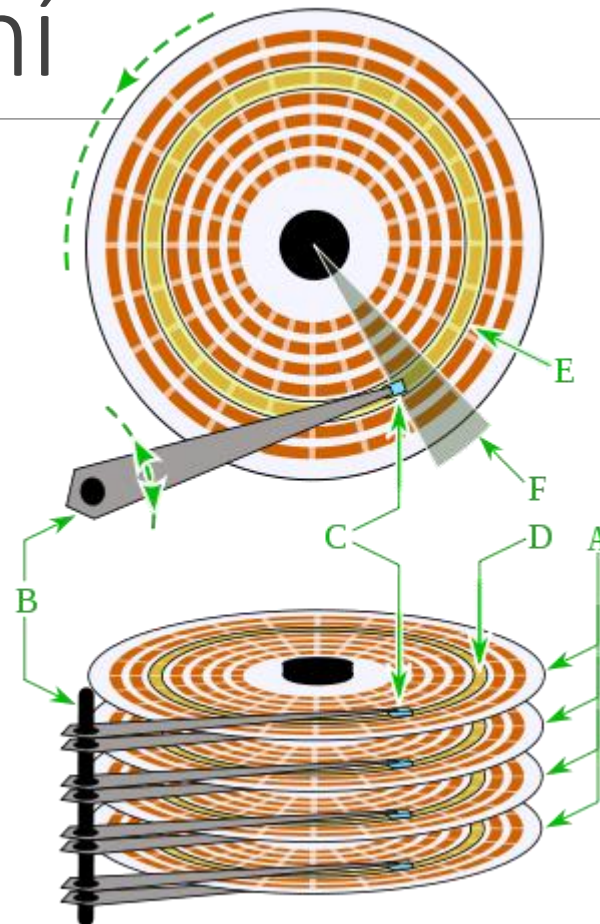
Pevný disk (rotační)

- Pevný disk je složen z jedné nebo více **ploten**, na kterých je magnetická vrstva.
- Záznam je prováděn do jednotlivých **stop** – soustředné kružnice.
- Každá plotna má nad sebou a pod sebou čtecí/záznamovou **hlavu**.
- U více ploten – všechny hlavy jsou nastaveny na stejnou stopu (tzv. **cylindr** – plocha procházející všemi plotnami)
- Stopy se dělí na **sektory**.
- Dříve stejný počet sektorů na vnitřní i vnější stopě, později proměnlivý, aby byla stejná hustota záznamu.

Pevný disk rotační

- A – plotny
- B – otočné raménko nesoucí
čtecí/zápisové hlavy
- C – čtecí/zápisová hlava
- D – stopy tvoří cylindr
- E – stopa
- F – sektor (část stopy)

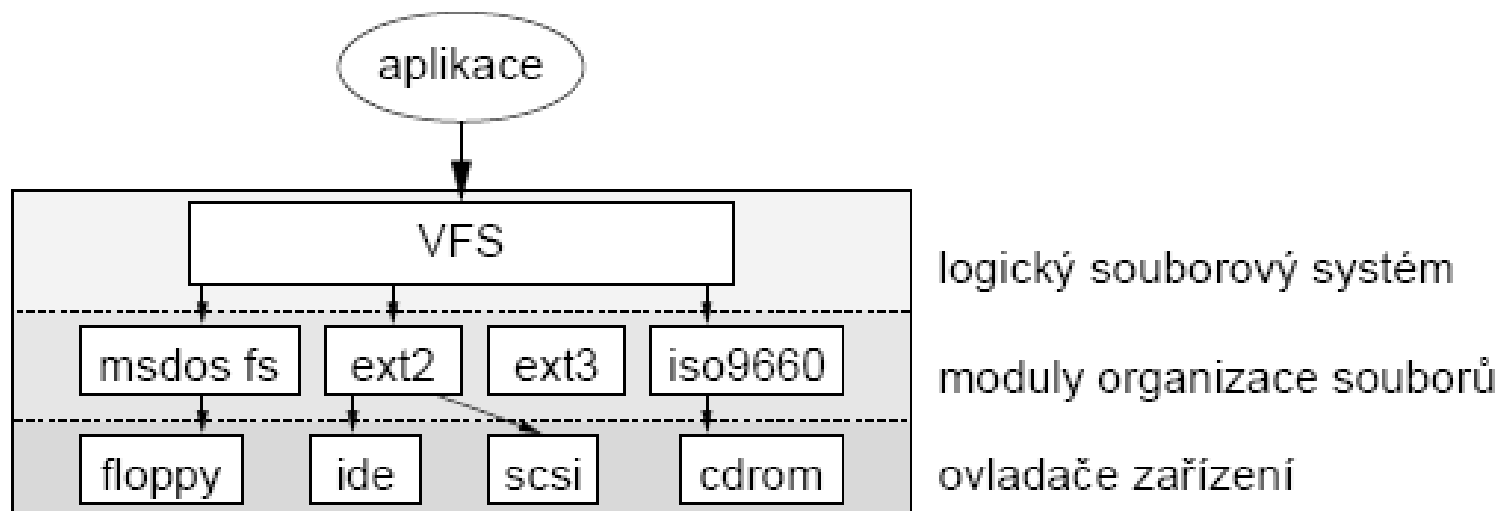
Zdroj obrázku: wikipedia



Pevný disk: CHS a LBA adresování

- CHS (Cylinder-Head-Sector, Stopa-Hlava-Sektor)
 - Starší způsob adresování sektorů
 - Uvedeme stopu, hlavu disku, číslo sektoru na stopě
 - Omezení velikosti disku,
BIOS uměl jen 1024 cylindrů, 256 hlav, 64 sektorů
 - Triky – elektronika disku přemapovala počet válců na vyšší počet hlav, abychom mohli adresovat větší disky
- LBA (Logical Block Addressing) adresování
 - Nahradilo původní CHS
 - Čísluje sektory na disku lineárně a není třeba znát geometrii disku

Implementace souborových systémů (!!!)



Implementace fs - vrstvy

1. Logický (virtuální) souborový systém
 - Volán aplikacemi
2. Modul organizace souborů
 - Konkrétní souborový systém (např. ext3)
3. Ovladače zařízení
 - Pracuje s daným zařízením
 - Přečte/zapíše logický blok

Ad 1 – virtuální fs

- Volán aplikacemi
- Rozhraní s moduly organizace souborů
- Obsahuje kód **společný** pro všechny typy fs
- **Převádí** jméno souboru na informaci o souboru
- **Udržuje informaci** o otevřeném souboru
 - Pro čtení / zápis (režim)
 - Pozice v souboru
- **Ochrana a bezpečnost** (ověřování přístupových práv)

Ad 2 – modul organizace souborů

- Implementuje **konkrétní** souborový systém
 - ext3, xfs, ntfs, fat, ..
- **Čte/zapisuje datové bloky** souboru
 - Bloky souboru číslovány 0 až N-1 (nebo a_1, a_2, \dots, a_N)
 - Převod čísla bloku souboru na diskovou adresu
 - Volání ovladače pro čtení či zápis bloku
- **Správa volného** prostoru + **alokace** volných bloků souborům
- Údržba datových struktur filesystemu

Ad 3 – ovladače zařízení

- Nejnižší úroveň
- Zařízení: SATA disk, SCSI disk, DVD mechanika
- Interpretují požadavky:
přečti logický blok 6456 ze zařízení 3

Pozn. číslo zařízení – Linux (hlavní a vedlejší)

Jak VFS pracuje s konkrétním filesystemem (!)

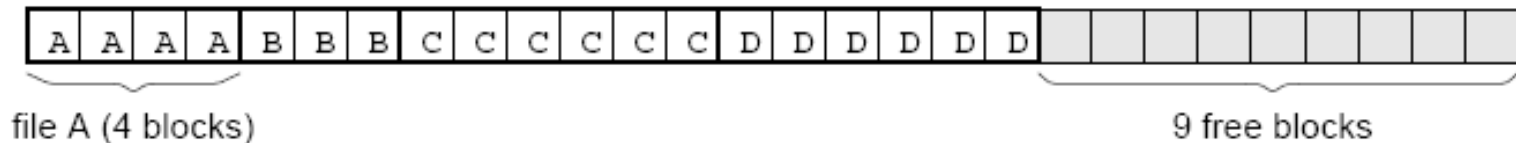
- Nový filesystem, který chceme používat se nejprve v systému **zaregistruje**
- Díky registraci VFS ví, jak zavolat jeho metody open, read, write pro konkrétní soubor
- Při požadavku na soubor VFS napřed zjistí, na kterém filesystemu leží:
 - Viz např. příkaz mount
 - /bin/ls může ležet na ext4, /home/pesi/f1.txt na xfs
 - Pro čtení /bin/ls zavolá VFS->ext4->read
 - Pro čtení /home/pesi/f1.txt zavolá VFS->xfs->read

Implementace souborového systému

- Nejdůležitější:
 - které diskové bloky patří každému ze souborů 😊
 - které diskové bloky jsou volné
- Předpokládáme:
 - fs je umístěn v nějaké diskové partition
 - bloky v diskové oblasti jsou očíslovány od 0

Kontinuální alokace

- Soubor jako **kontinuální posloupnost** diskových bloků
- Příklad: bloky velikosti 1KB, soubor A (3.5KB) by zabíral 4 po sobě následující bloky
- Implementace
 - Potřebujeme znát číslo prvního bloku
 - Znat celkový počet bloků souboru (např. v adresáři)
- **Velmi rychlé čtení**
 - Hlavičku disku na začátek souboru, čtené bloky jsou za sebou



K obrázku, ještě lépe je bloky označovat: a1, a2, a3, a4

Kontinuální alokace

- Problém – dynamičnost OS
 - soubory vznikají, zanikají, mění velikost

Nějaké řešení?

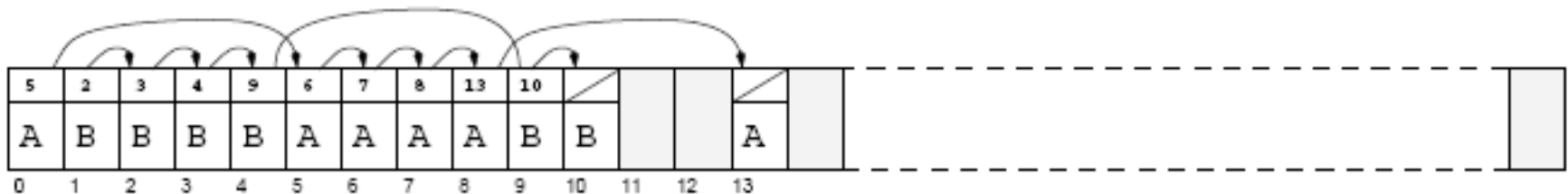
- Nejprve zapisovat sériově do volného místa na konci
- Po zaplnění využít volné místo po zrušených souborech
- Pro výběr vhodné díry – potřebujeme znát konečnou délku souboru – většinou nevíme..

Lze dnes využít kontinuální alokaci?

- Dnes se používá pouze na read-only a write-once médiích
- Např. v ISO 9660 pro CD ROM
- Výhodou DVD je, že dopředu víme velikosti souborů, které vypalujeme a tyto soubory se také typicky dále již nemění

Seznam diskových bloků

- Svázat diskové bloky do seznamu – nebude vnější fragmentace
- Na začátku diskového bloku je uložen odkaz na další blok souboru, zbytek bloku obsahuje data souboru
- Pro přístup k souboru stačí znát pouze číslo prvního bloku souboru (může být součástí záznamu v adresáři) a velikost (kolik využito z posledního bloku)



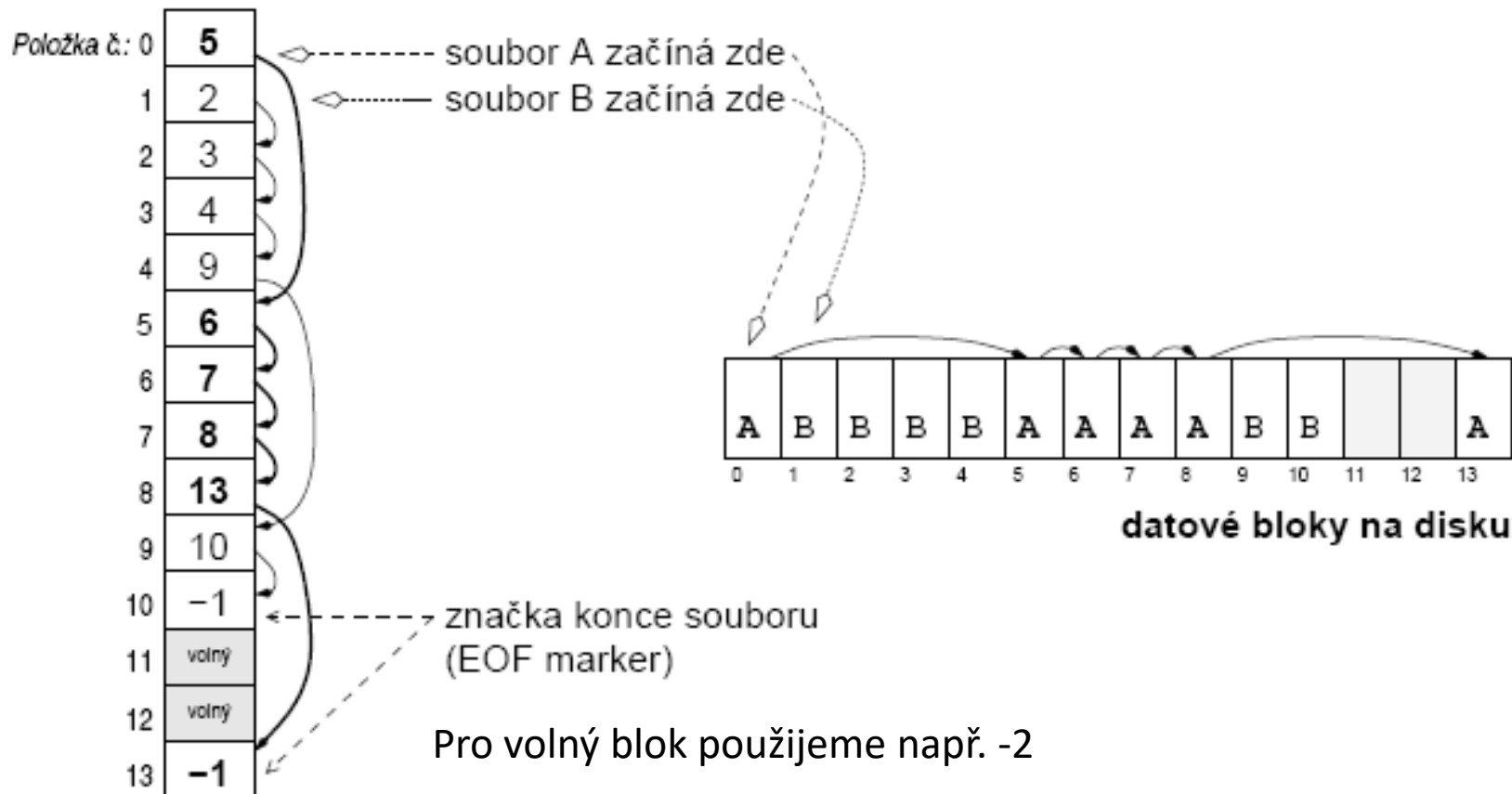
Seznam diskových bloků

- Sekvenční čtení – bez potíží
- Přímý přístup – simulován sekvenčním, pomalé (musí dojít ke správnému bloku)
- Velikost dat v bloku není mocnina dvou
 - Část bloku je zabraná odkazem na další blok
 - Někdy může být nevýhodou (kešování, optimalizace, ...)

FAT (!!)

- Přesunutí odkazů do samostatné tabulky FAT
- **FAT (File Allocation Table)**
 - Každému diskovému bloku **odpovídá** jedna položka ve FAT tabulce
 - Položka FAT obsahuje číslo **dalšího bloku** souboru (je zároveň odkazem **na další položku** FAT!)
 - Řetězec odkazů je ukončen speciální značkou, která není platným číslem bloku (poslední blok souboru)
 - Volný blok – značí, že je odpovídající datový blok volný
 - Bad block – značí, že je odpovídající datový blok poškozený

Tabulka FAT



Položce číslo X ve FAT odpovídá datový blok X na disku

Položka ve FAT obsahuje odkaz na další datový blok na disku a tedy i na další položku ve FAT tabulce

Tabulka FAT

Položka č.: 0	5	
1	2	
2	3	
3	4	
4	9	
5	6	
6	7	
7	8	
8	13	
9	10	
10	-1	
11	volný	
12	volný	
13	-1	

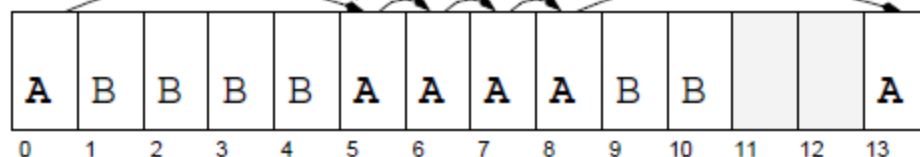
soubor A začíná zde
soubor B začíná zde

Fakt tu je !
Jdem dál na
6

značka konce souboru
(EOF marker)

5 znamená, že na indexu 5 je
další odkaz na blok souboru A

Na indexu 5 je i datový blok
souboru A (kus filmu)



datové bloky na disku

Co je na FAT důležité?

Pokud bych chtěl např. k souboru B přidat další datový blok, nemusím s ničím hýbat, pouze do FAT(10) vložím číslo 11, a do FAT(11) dám -1 a soubor B je prodloužený

FAT

FAT je ukázka implementace souborového systému, kde v druhé části máme **datové bloky** (obsahující např. části jednoho filmu) a v první části máme **indexy**, které nám říkají, pod jakým číslem se nalézá další datový blok daného souboru

Výhodou je, že s určitým souborem můžeme manipulovat, zrušit ho, prodloužit, atd., aniž bychom ovlivnili pozici ostatních souborů na disku

FAT - defragmentace

- Úplná – obsazené bloky souborů půjdou na disku za sebou, poté následuje volné místo
- Částečná – upraví se jen tak, že napřed je obsazený prostor soubory a za ním volné bloky

FAT souborový systém

- co obsahuje (!!)

Boot sektor

- blok parametrů disku
 - Verze, počet sektorů na cluster, počet FAT, název svazku (label), ..
- spouštěcí kód svazku
 - Pro bootování z této oblasti (ale dnes většinou z jiných file systémů)

▪ FAT tabulky

- Obvykle je k FAT1 i FAT2 jako záložní kopie, když je FAT1 nečitelná
- Popisuje přiřazení každého clusteru v oddílu
 - volný, vadný, konec souboru, číslo následujícího clusteru

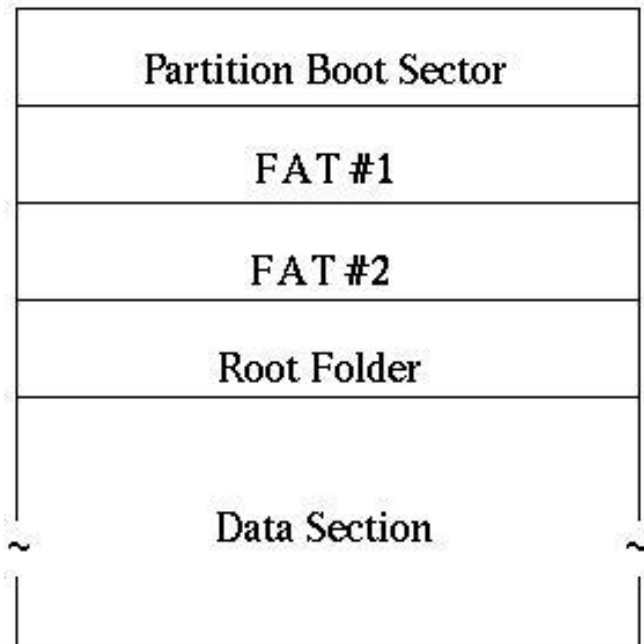
▪ Datové bloky

- Volné nebo obsahující části souborů (nebo vadné)
- První datový blok – začíná zde **hlavní adresář** (jméno souboru, velikost, kde začíná)
 - U FAT12, FAT16 byla jeho velikost nastavena na pevně při vytváření souborového systému
 - U FAT32 mohl být uložen kdekoliv a jeho velikost může narůstat

Disk, partition, FAT

- máme pevný disk (`/dev/sda`)
- např. druhá oblast disku (`/dev/sda2`) bude formátována na nějaký souborový systém, v našem případě z FAT, konkrétně může být FAT32
- tato oblast disku obsahuje to, co bylo uvedeno na předešlém slidu

Diskový oddíl s FAT



Diskový oddíl, který je naformátován na FAT (např. FAT32)

1. boot sektor (popis parametrů, případně kód pro bootování)
2. FAT tabulky, obvykle jsou dvě kopie
3. hlavní adresář daného oddílu
4. data

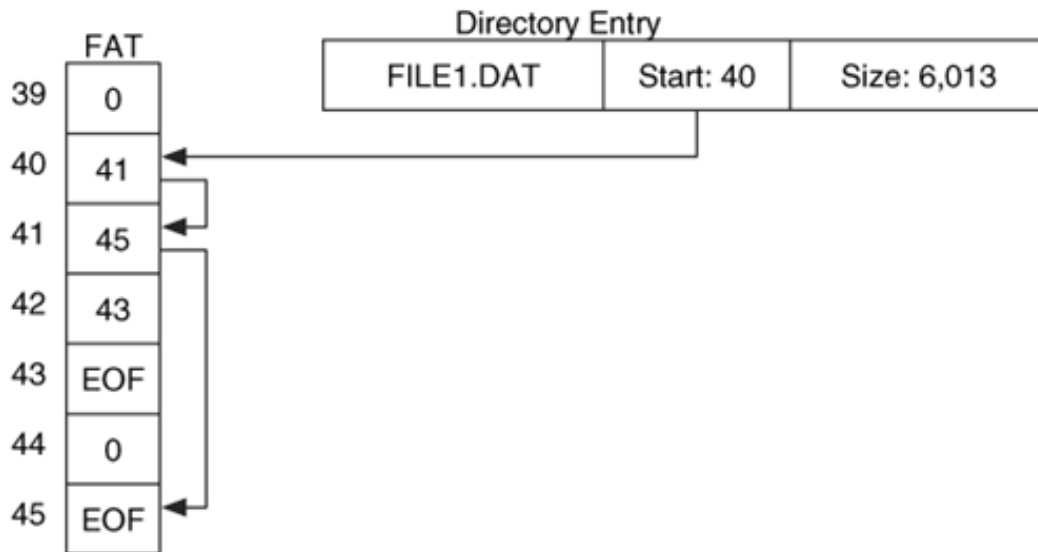
Zdroj obrázku:

<http://yliqepyh.keep.pl/fat-file-system-specifications.php>

FAT – jak poznáme, kde soubor začíná?

Z položky adresáře poznáme:

- Jméno souboru
- Kde začíná (40)
- Velikost (6013 bytů)
- Další atributy (čas a datum, hidden, archive,...)



V tomto obrázku by velikost bloku byla cca 2KB – 2048B
Značka EOF je číslo symbolizující konec souboru.

Víme tedy, kde začíná soubor (40) a umíme i určit, **jaká část posledního přiděleného bloku je využita daty souboru** (zbytek po dělení velikosti souboru 6013 velikostí bloku)

Vlastnosti FAT

Velikost tabulky FAT

- 80GB disk, bloky 4KB => 20 mil. položek
- Každá položka alespoň 3 byty => 60MB FAT

Použití

- Všeobecná podpora OS - DOS i Windows 10/11, Linux
- FAT12, 12 bitů, $2^{12} = 4096$ bloků, diskety
- FAT16, 16 bitů, $2^{16} = 65536$ bloků
- FAT32, 2^{28} bloků, blok 4-32KB, cca 8TB
(na číslo clusteru se používá jen 28bitů z 32 bitů)
- ExFAT – pro větší kapacity

Problémy s FAT

- fragmentace
 - „rozházené“ části souborů po celém disku
 - Snižuje výkon, možnosti precachování, úspěšnost zotavení
- ztracené clustery
 - Clustery označené jako používané, ale nepatří žádnému souboru
- překřížené FAT řetězy
 - Pokud jsou pro 2 a více souborů vyhrazeny clustery se stejným číslem
- Další poškození FAT
 - FAT řetěz neodpovídá velikosti souboru (moc krátký, dlouhý, ...)
 - Ukazatel na další cluster ukazuje za konec oddílu disku

Porovnání FAT32 x exFAT

exFAT vs. FAT32 Comparison

Feature	FAT32	exFAT
Maximum Volume Size	8 TB*	128 PB
Maximum File Size	4 GB	16 EB
Maximum Cluster Size	32 KB **	32 MB
Maximum Cluster Count	2^{28}	2^{32}
Maximum File Name Length	255	255
Date/Time resolution	2 s	10 ms
MBR Partition Type Identifier	0x0B, 0x0C	0x07

* Windows cannot format FAT32 volumes bigger than 32GB, though it supports larger volumes created by third party implementations; 16 TB is the maximum volume size if formatted with 64KB cluster

** According to Microsoft KB184006 clusters cannot be 64KB or larger, though some third party implementations support up to 64KB.

U FAT32
flash disku
snadno
narazíme na
4GB limit
velikosti
souboru

Zdroj:

<http://www.ntfs.com/exfat-comparison.htm>

Příklady filesystemů (!!!)

FAT

- paměťové karty, flash disky – často prodávané s formátem na FAT32
- Různé varianty FAT32, exFAT
- Nepoužívá ACL – u souborů není žádná info o přístupových právech
- Jen atributy archive, hidden, read-only apod.
- Snadná přenositelnost dat mezi různými OS

NTFS

- Používá se ve Windows XP/7.../10/11
- Používají ACL: k souboru je přiřazen seznam uživatelů, skupin a jaká mají oprávnění k souboru (!)
- Možnost komprimace, šifrování (BitLocker) – důležité pro firmy

Příklady filesystemů

Ext2

- Použití v Linuxu, nemá žurnálování, starší
- Práva – standardní unixová (vlastní, skupina, others), lze doplnit i ACL (komplexnější, ale samozřejmě přinesou zpomalení)

Ext3

- Použití v Linuxu, má žurnál (rychlejší obnova konzistence po výpadku)

Ext4

- Používá se i u současných Linux systémů
- stejně jako ext2, ext3 používá inody
- extenty – souvislé logické bloky
 - může být až 128MB oproti velkému počtu 4KB bloků
- nanosekundová časová razítka

Další používané filesystemy: XFS, JFS, btrfs, ZFS

NTFS

- nativní souborový systém Windows od NT výše
- **žurnálování**
zápisy na disk se zapisují do žurnálu, pokud uprostřed zápisu systém havaruje, je možné dle stavu žurnálu zápis dokončit nebo anulovat => konzistentní stav
- **access control list**
přidělování práv k souborům (na rozdíl od FAT)
- **komprese**
na úrovni fs lze soubor nastavit jako komprimovaný

NTFS pokračování

- šifrování

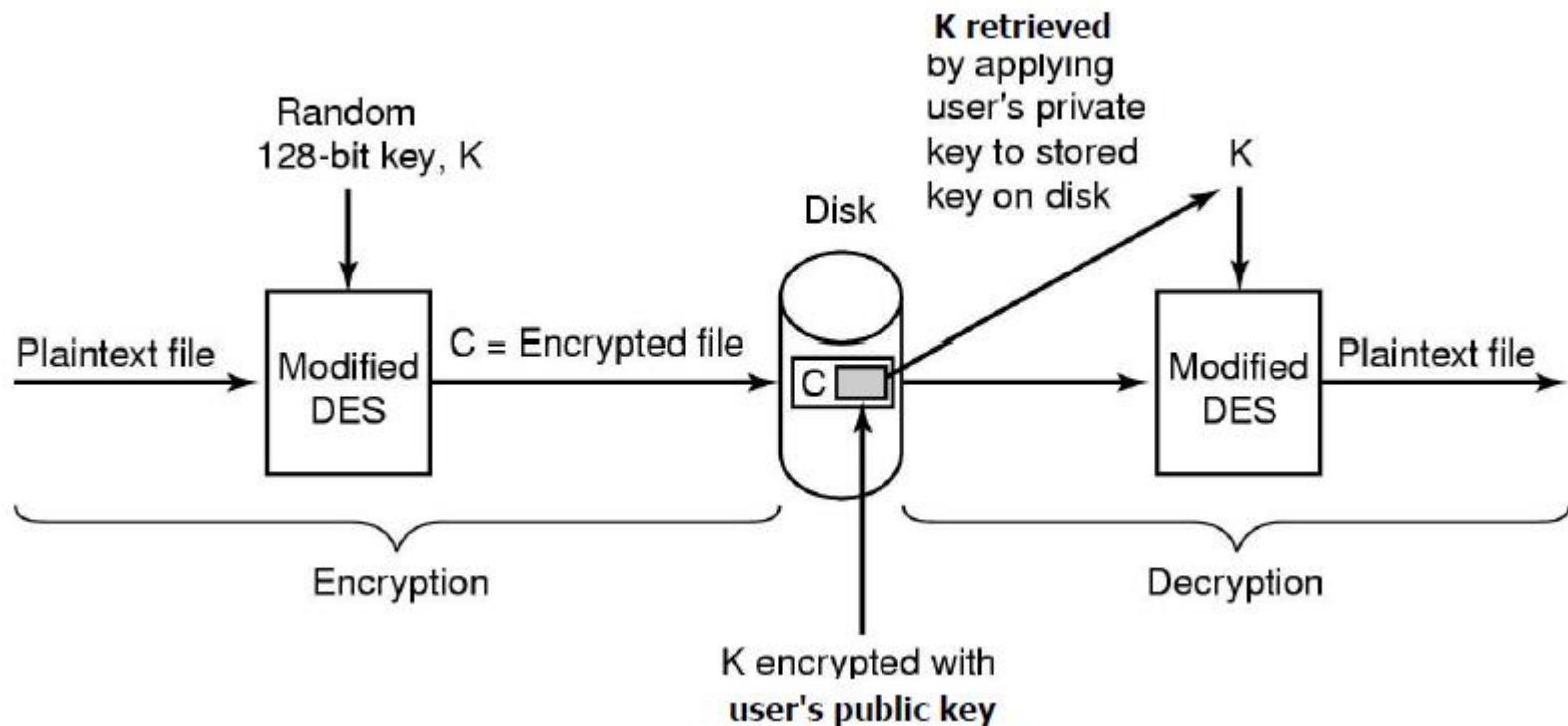
EFS (encrypting file system),
transparentní – otevřu ahoj.txt, nestarám se, zda je
šifrovaný

- diskové kvóty

max. velikost pro uživatele na daném oddíle
dle reálné velikosti (ne komprimované)

- pevné a symbolické linky

Šifrování



Při přístupu k souboru si odšifruji klíč K, a jeho pomocí získám zpět plaintext

NTFS struktura (!!)

- 64bitové adresy clusterů .. cca 16EB
- clustery **číslovány od začátku partition** – logická čísla clusterů
- systém jako obří databáze
záznam v ní odpovídá souboru
- základ 11 systémových souborů - metadata
vzniknou hned při formátování svazku
- **\$Logfile** – žurnálování
- **\$MFT** (Master File Table) – nejdůležitější (!!)
záznamy o všech souborech, adresářích, metadatach
hned za boot sektorem, za ním se udržuje zóna volného místa

NTFS struktura

- **\$MFTMirr** – uprostřed diskové oblasti, obsahuje část záznamů \$MFT, při poškození se použije tato kopie
- **\$Badclus** – seznam vadných clusterů
- **\$Bitmap** – sledování volného místa
pole bitů, význam: 0 – volný, 1 - použitý
- **\$Boot, \$Volume, \$AttrDef, \$Quota, \$Upcase, .**

podrobnosti:

<http://technet.microsoft.com/en-us/library/cc781134%28WS.10%29.aspx>

NTFS

Defaultní velikosti clusterů:

Volume size	NTFS cluster size
7MB – 512MB	512B
513MB – 1 024 MB	1KB
1 025MB – 2GB	2KB
2GB – 2TB	4KB

Při formátování si můžeme zvolit vlastní velikost clusterů až do 2MB

NTFS vybrané atributy souborů

\$FILE_NAME

- jméno souboru
- velikost
- odkaz na nadřazený adresář
- další

\$SECURITY_DESCRIPTOR

- přístupová práva k souboru

\$DATA

- vlastní obsah souboru

An orange scroll graphic with a rolled-up top and bottom edge, containing text.

atributy
definovány
v
\$AttrDef

NTFS

- adresáře
 - speciální soubory
 - B-stromy se jmény souborů a odkazy na záznamy v MFT
- alternativní datové proudy (ADS)
 - notepad poznamky.txt:tajny.txt
 - často útočiště virů, škodlivého kódu
- zkopírováním souboru z NTFS na FAT
 - ztratíme přístupová práva a alternativní datové proudy

NTFS – Alternate Data Stream

1. Pustit cmd.exe
2. notepad poznamky.txt
3. Text – veřejná poznámka, uložit, ukončit
4. Notepad poznámky.txt:tajny.txt
5. Text – toto je tajná poznámka, uložit, ukončit
6. dir

V adresáři vidíme poznámky.txt, ale nepoznáme, že soubor má alternate data stream tajny.txt

NTFS – ADS

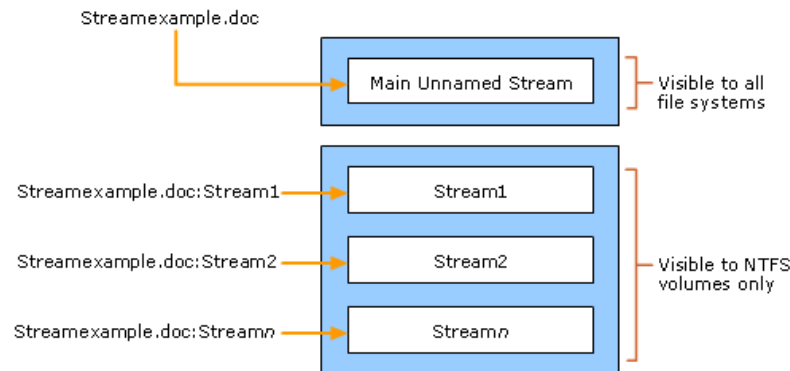
(Alternativní datové streamy)

Na NTFS filesystému:

notepad poznamky.txt

notepad poznamky.txt:tajny.txt

Unnamed and Named Streams



Obrázek: technet.microsoft.com

NTFS – způsob uložení dat (!!!)

- **kódování délkou běhu**

- od pozice 100 máme např. uloženo:
A1, A2, A3, B1, B2, A4, A5, C1, ...

- soubor A bude popsán fragmenty

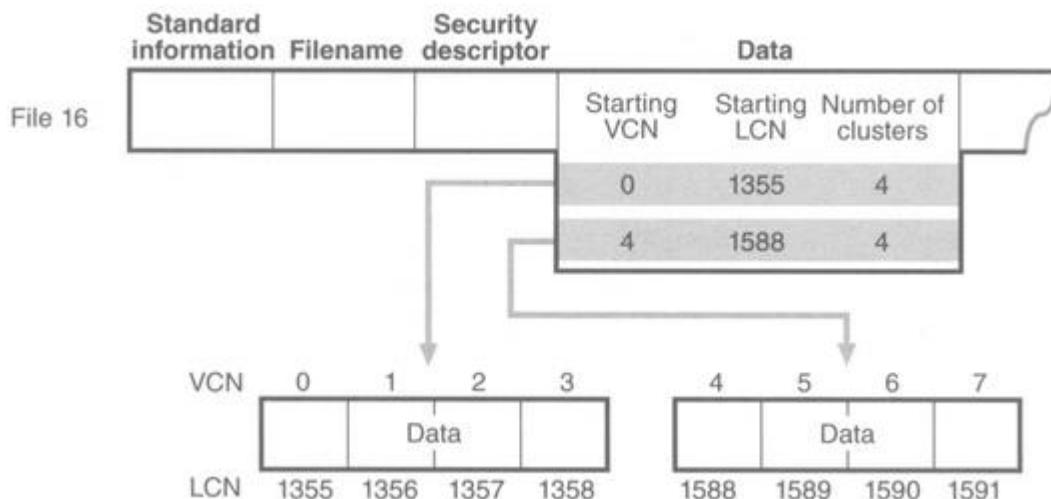
- **fragment**

- index
- počet bloků daného fragmentu

- v našem příkladě pro soubor A dva fragmenty:

- I. 100, 3 (od indexu 100 patří tři bloky souboru A)
- II. 105, 2 (od indexu 105 patří dva bloky souboru A)

Příklad – 2 fragmenty

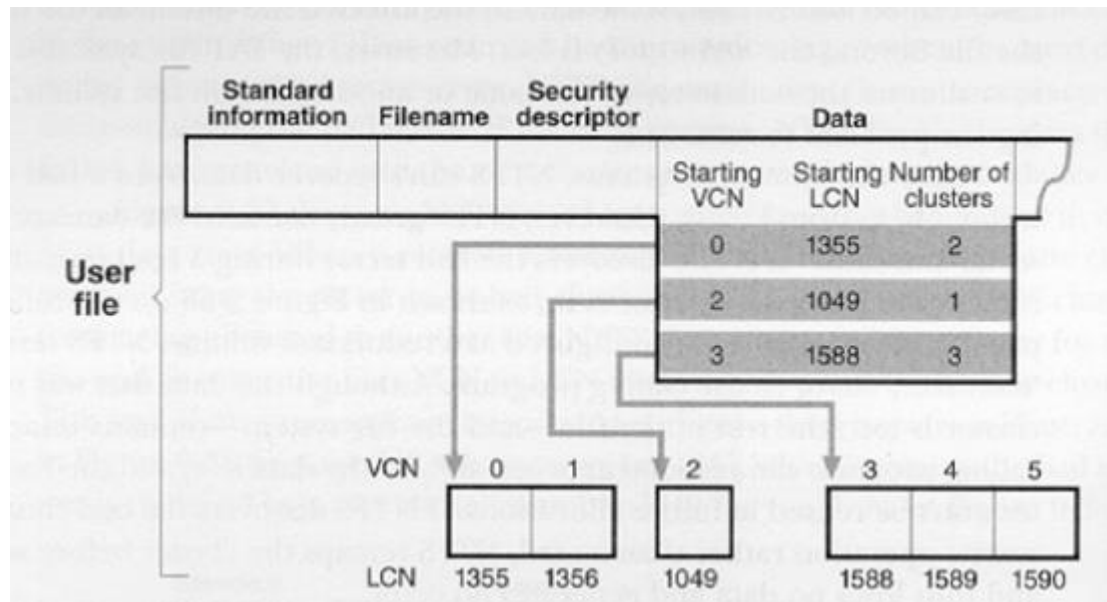


VCN – virtual cluster number (zde 0..7, i-tý cluster souboru)

LCN – logical cluster number (číslo clusteru na disku, číslované od začátku diskové oblasti)

Zdroj: https://www.datarecoup.com/blog/file-systems/ntfs-data-structure-and-recovery-internals/item_limitstart/page-4

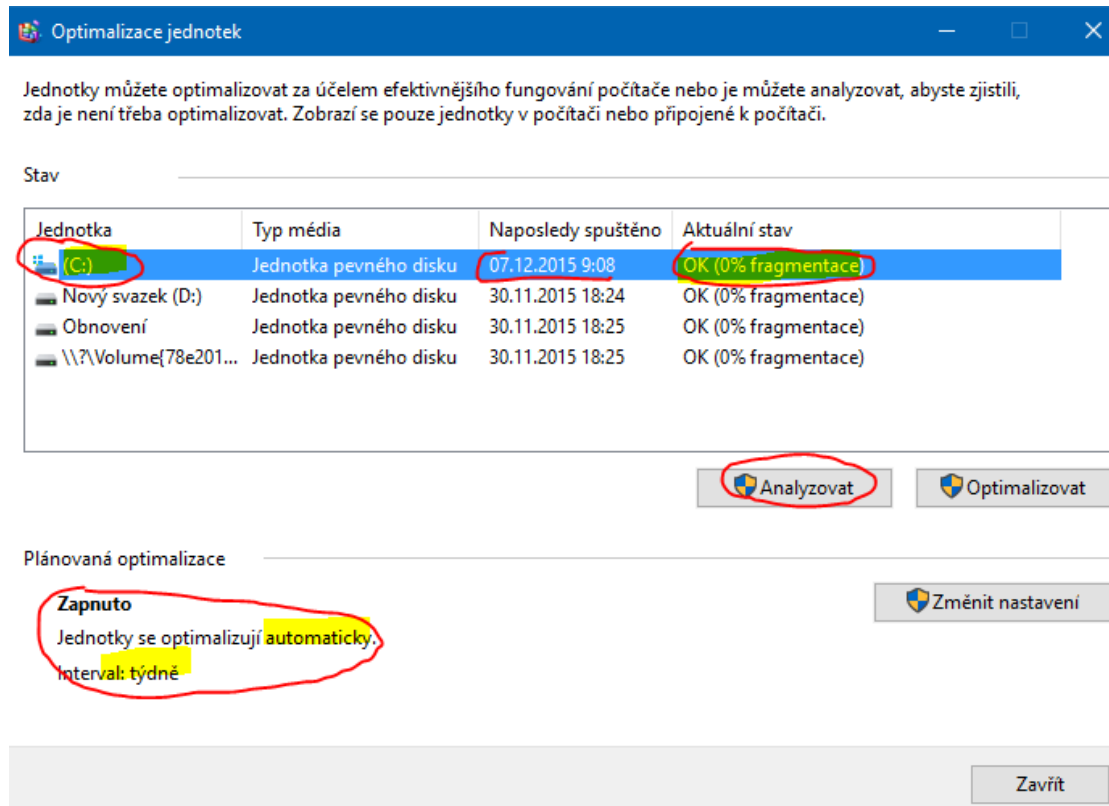
Příklad – 3 fragmenty



NTFS – způsob uložení dat

- V **ideálním** případě **1 soubor = 1 fragment**
(výhody kontinuální alokace)
- Defragmentovat můžeme jak celou partition, tak jen vybrané soubory (přes utility v sysinternals)
- Kontrola:
- Explorer -> disk C: -> pravá myš -> Vlastnosti -> Nástroje -> Optimalizovat a defragmentovat

NTFS - defragmentace



NTFS – Sparse Files

Soubor 17GB

- Užitečná data 7GB
- Nuly 10GB

Na disku zabere místo 17GB
jen 7GB u sparse file

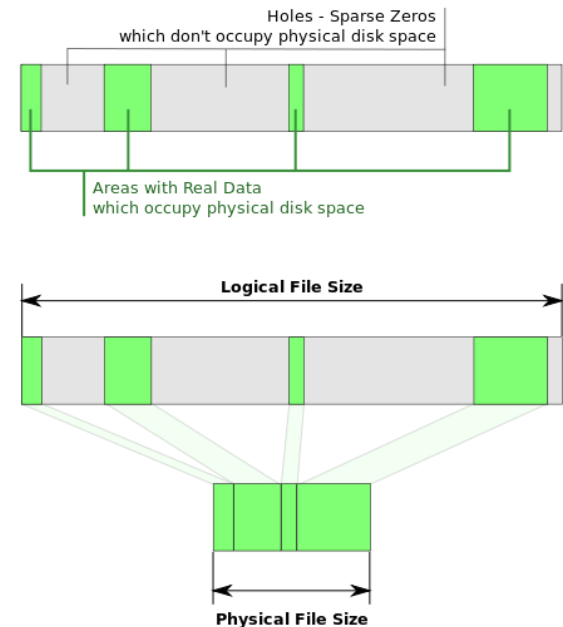
Příklad příkazů:

FSUtil File CreateNew temp 0x100000

FSUtil Sparse SetFlag temp

FSUtil Sparse SetRange temp 0x100000

Viz příklad na cvičení



Obr: wikipedia

Příklad 1 -zadání

Zadání příkladu:

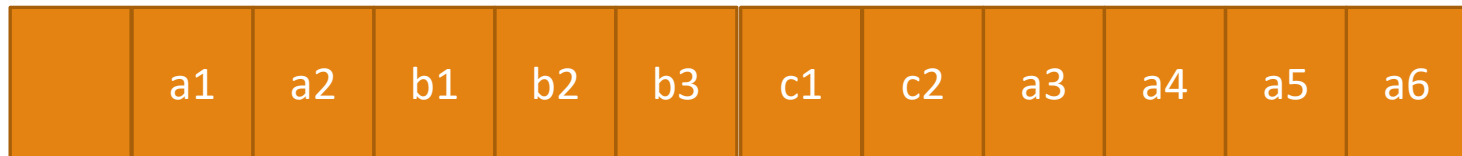
- Máme soubory a.txt, 5500B b.txt, 2200B c.txt, 1500B v hl. adresáři.
- Velikost alokační jednotky je 1024B (1KiB)
- Doplněte alokované datové bloky, nakreslete FAT tabulku, jak budou vypadat položky hlavního adresáře?



501

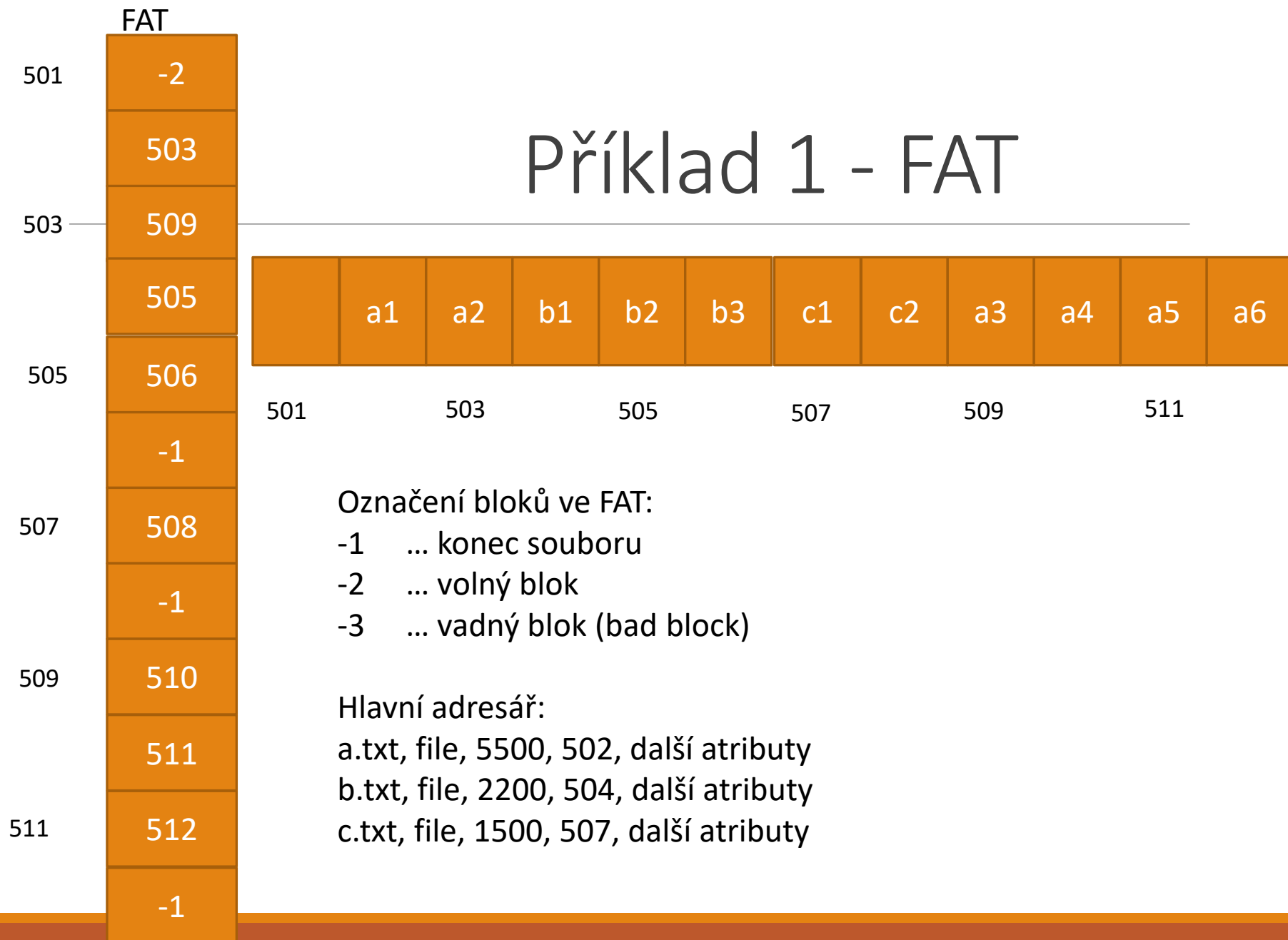
Příklad 1 - postup

- Ze zadání plyne, že soubory budou zabírat:
 - A bude zabírat 6 datových bloků
 - B bude zabírat 3 datové bloky
 - C bude zabírat 2 datové bloky
- Dokreslíme datové bloky např. takto:



501

Příklad 1 - FAT



Příklad 1 – prodloužení souboru C

Úkol: Nyní bychom chtěli soubor c.txt zvětšit na výsledných 2500 B.

1. Potřebujeme k souboru C přidat 1 další blok – c3, do stávajících se nevejde.
2. Musíme projít FATku a podívat se, kde je místo (tj. najít volný blok).
3. Procházením zjistíme, že jediný volný blok je 501.
4. Do datového bloku 501 nahrajeme potřebná data souboru c.txt.
5. Upravíme FAT:
501 bude obsahovat -1 jako konec souboru
508 bude obsahovat číslo 501, tedy adresu dalšího bloku souboru
6. Upravíme hlavní adresář, u souboru c.txt bude uvedene správná velikost

Příklad 1 - otázky

Jaké jednoduché kontroly konzistence můžeme provést?

1. Z adresáře známe přesné velikosti souborů. Můžeme zkontrolovat řetězy FAT příslušející těmto souborům, zda mají správnou velikost. Tj. má-li soubor 5500B a alokační jednotka 1024B, musí řetěz obsahovat celkem 6 bloků – ani víc, ani méně.
Možná chyba – řetěz by byl kratší, delší, případně by špatná hodnota směřovala do jiného cyklu (data jiného souboru).
2. Většinou máme více FAT tabulek – FAT1 a FAT2.
Můžeme zkontrolovat, že si tabulky odpovídají, pokud ne, ohlásit uživateli chybu.

Příklad 1 - otázky

Proč bývá více FAT tabulek?

FAT tabulka je kritická datová struktura, přepis v ní může znamenat ztrátu souboru či adresáře. Proto je typicky ve dvou kopiích – FAT1 a FAT2. Dojde-li k chybě při zápisu do jedné, druhá zůstane neporušena. (př. Zapisuji do FAT1 a vypnou proud, FAT2 zůstane nedotčena)

Co je to defragmentace?

Úplná – bloky budou uspořádány (a1..a6,b1..b3,c1..c3, volné místo)

Částečná – uspořádání (obsazené dat. bloky, volné datové bloky)

Příklad 1 - otázky

V čem je defragmentace výhodná?

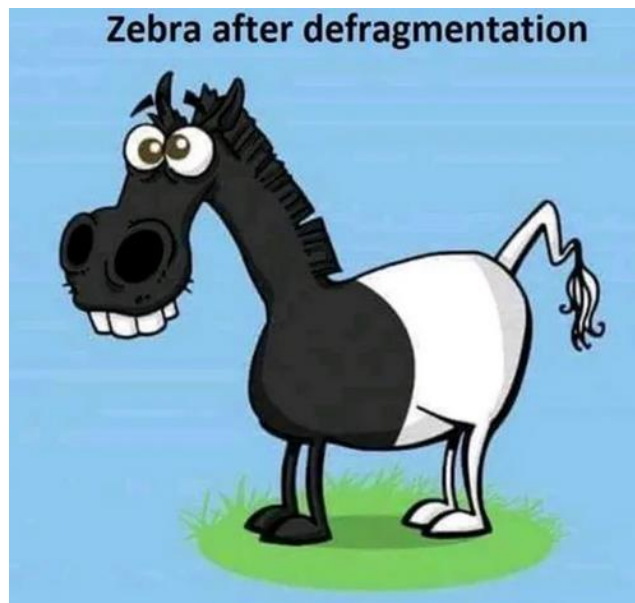
Výkonnost - bloky souboru jdou za sebou (možnost přednačítání dalších bloků dopředu).

Bezpečnost – pokud se FAT poškodí a bloky souboru jdou za sebou, je větší šance na rekonstrukci původního souboru.

Jak je to s FAT a přístupovými právy?

FAT obvykle přístupová práva neřeší, není zde uložena informace, jaký uživatel/skupina má jaká práva k danému souboru či adresáři.

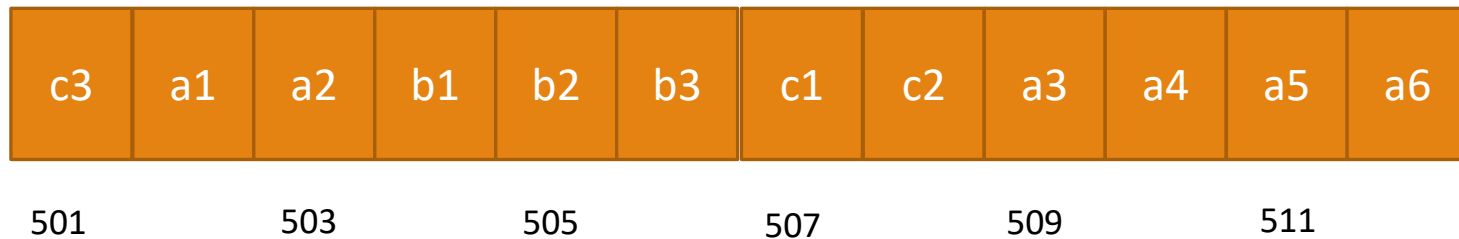
Poznámka



Zdroj:
<https://9gag.com/gag/apmrzzW>

Příklad 2

Jak by vypadal popis uložení souborů a.txt, b.txt, c.txt v NTFS?



Soubor A

- I. Fragment (502,2)
- II. Fragment (509,4)

Soubor B

- I. Fragment (504,3)

Soubor C

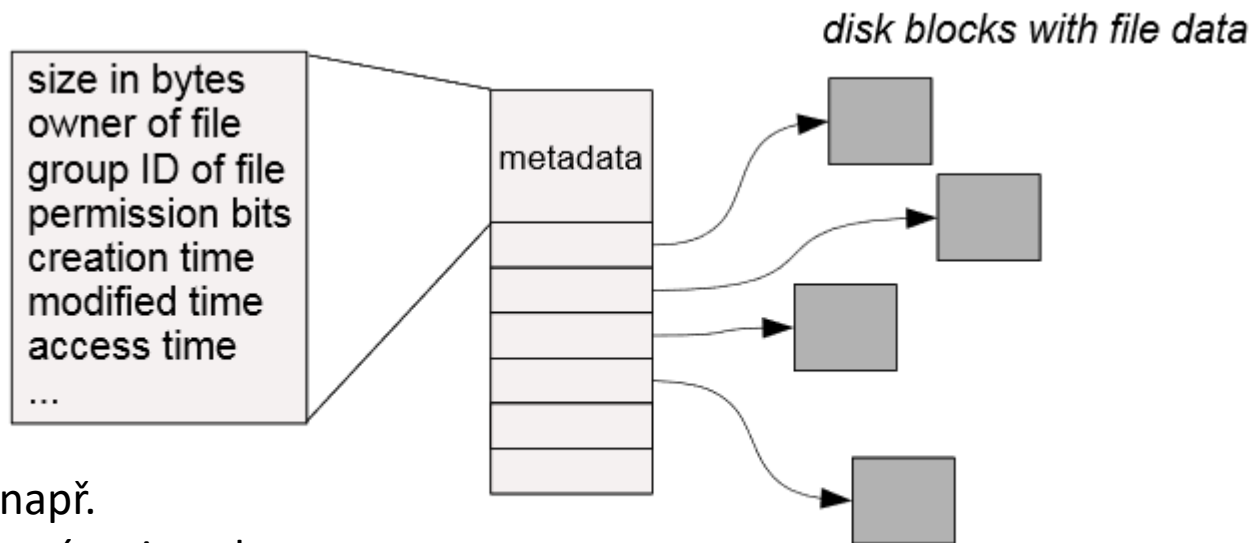
- I. Fragment (507,2)
- II. Fragment (501,1)

Systemy využívající i-uzlů (!)

- i-uzel – klíčová datová struktura, která obsahuje informace o souboru nebo adresáři
- Každý soubor (a tedy i adresář) je reprezentovaný i-uzlem (!!!!)
- i-uzel - datová struktura
 - Metadata popisující vlastníka souboru, přístupová práva, **velikost souboru**
pozor, není zde název souboru !!
 - Umístění bloků souboru na disku
 - Přímé odkazy (většinou 12) a nepřímé 1. 2. 3. úrovně
 - Abychom věděli, jaké bloky přistupovat

i-uzel

i-uzel neobsahuje jméno souboru
!!!

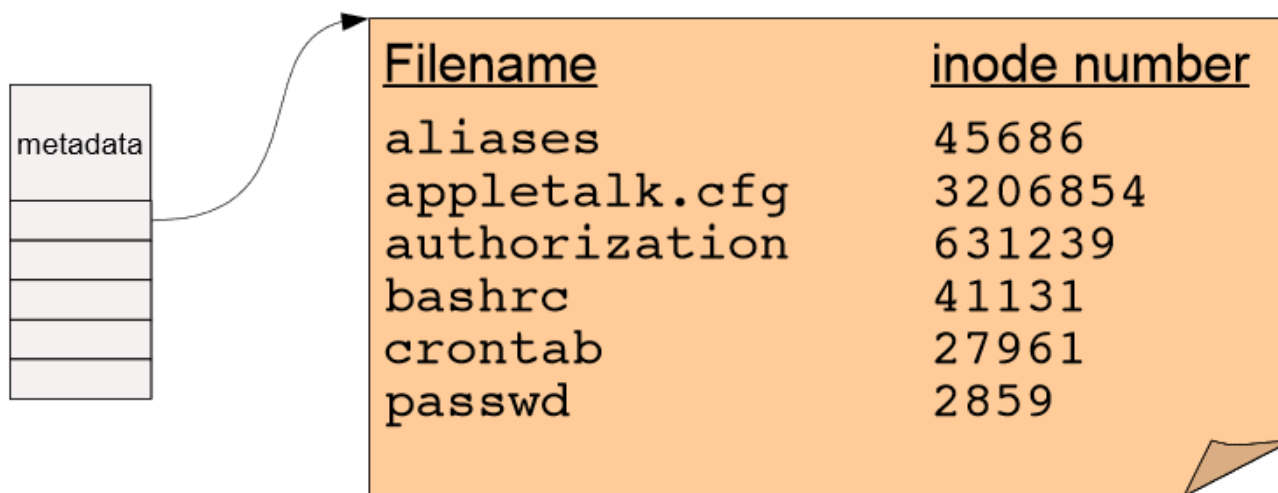


Dále je zde např.
počet referencí na i-uzel
(kvůli hardlinkům)

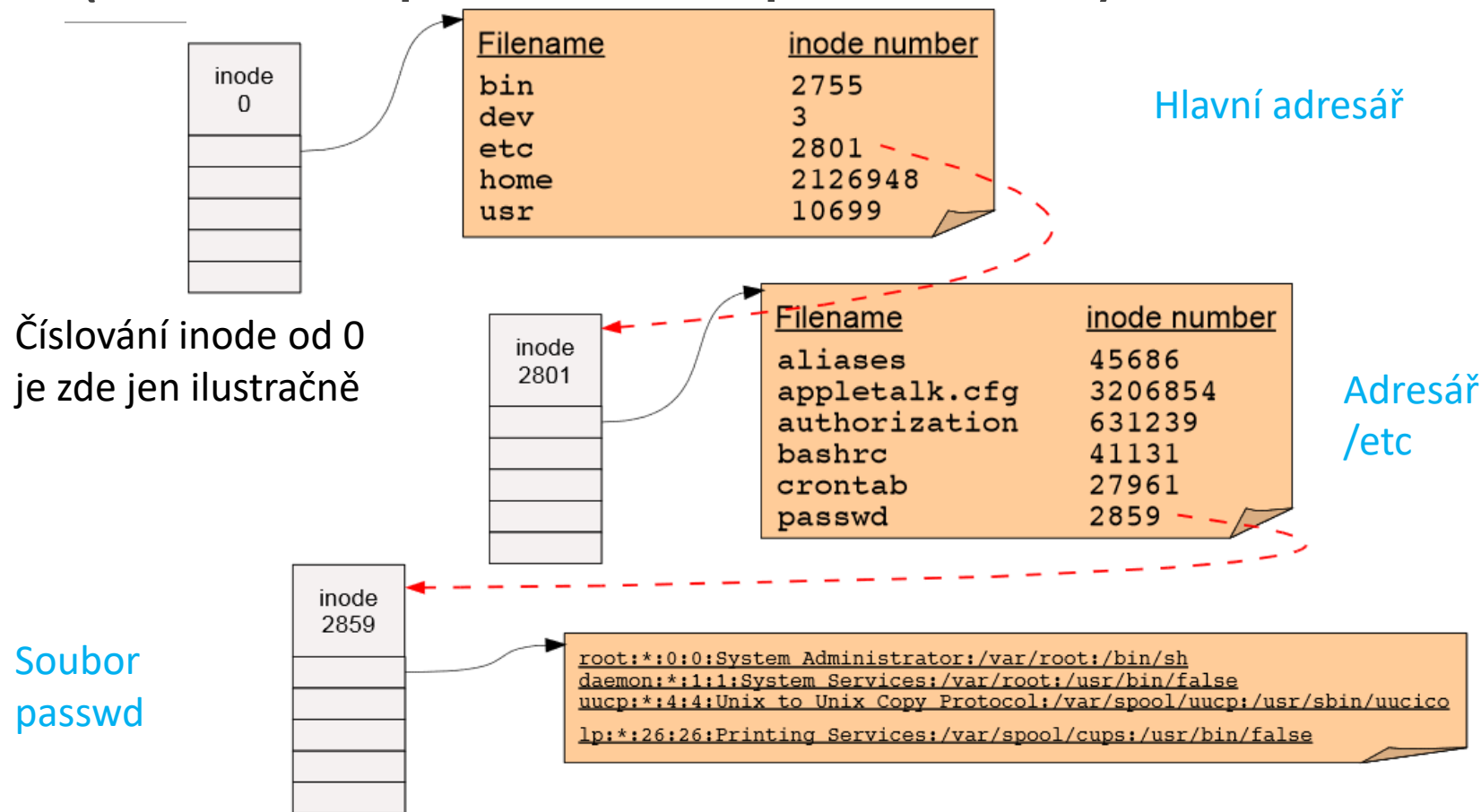
Obrázek znázorňuje jeden i-uzel (metadata , přímé adresy diskových bloků)
Pamatuj: 1 i-uzel = 1 soubor (obyčejný, adresář)

Adresář systému s i-uzly (!)

Soubor obsahující dvojice
(název_souboru, číslo_odpovídajícího_i-uzlu)



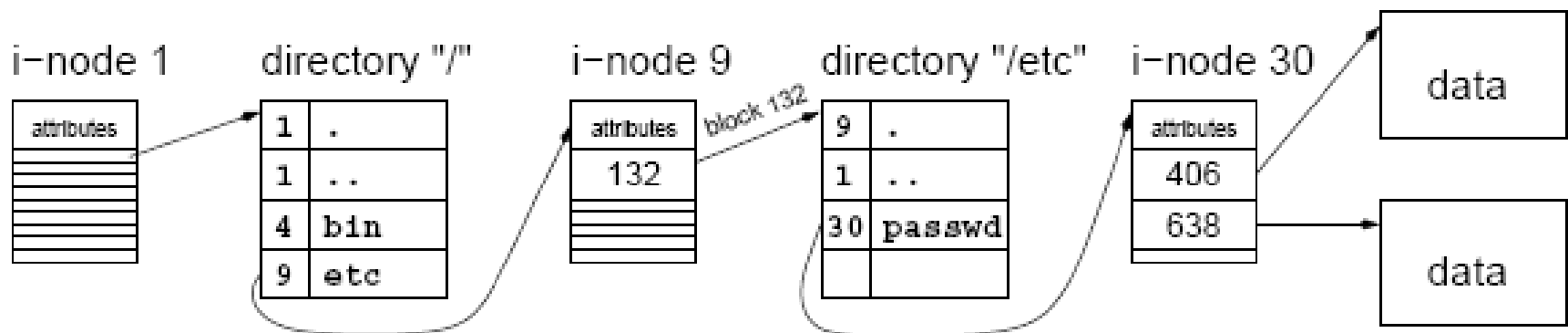
Prohledání cesty k souboru (zde např. /etc/passwd)



Projít: hlavní adresář, adresář etc a následně vlastní soubor passwd

Nalezení cesty k souboru `/etc/passwd` (ještě jeden příklad)

- V kořenovém adresáři najdeme položku „`etc`“
- i-uzel číslo **9** obsahuje adresy diskových bloků pro adresář `etc`
- V adresáři `etc` (disk blok 132) najdeme položku `passwd`
- i-uzel **30** obsahuje soubor `/etc/passwd`
- (uzel, obsah uzlu, uzel, obsah uzlu)



Umístění i-uzlů na disku

- 1 i-uzel = 1 soubor
- Pevný počet i-uzlů = max. počet souborů na daném oddílu disku (určeno při vytvoření fs)
- Pokud nám dojdou i-uzly, další soubor již nemůžeme vytvořit, ale pokud zbývají datové bloky, můžeme prodloužit stávající soubory

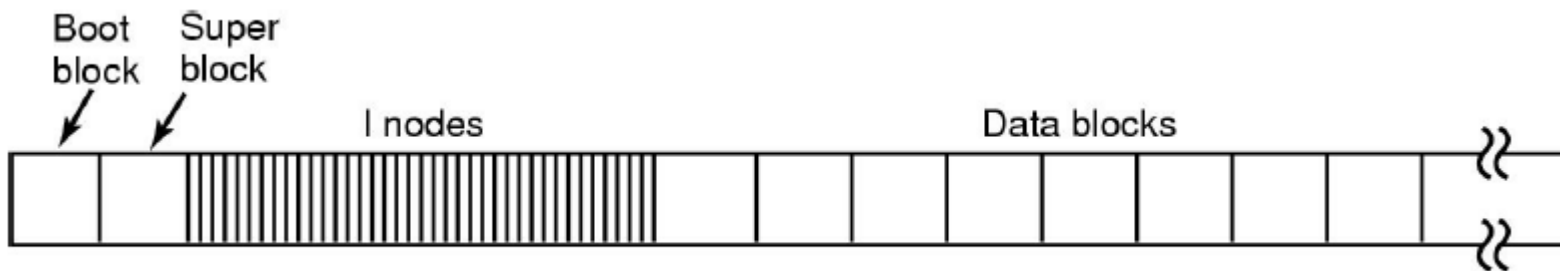
Unixové systémy s využitím i-uzlů (původní koncepce)

takto vypadá partiton disku (např. `/dev/sda1`)

původní rozdělení v Unixových systémech

novější rozdělení, např. v ext2 bude uvedeno na pozdějších slidech

superblock – příznak čistoty (byl korektně odpojený?), verze, počet i-nodů, velikost alokační jednotky, seznam volných bloků

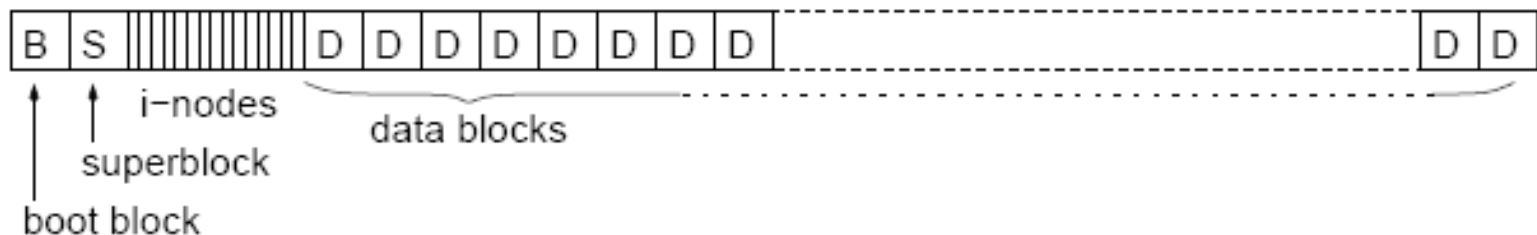


Původně volné bloky určovány seznamem volných bloků, superblock odkazoval na začátek tohoto seznamu; nyní se používají bitmapy

System s i-uzly původní koncepce

■ Struktura fs na disku

- **Boot blok** – může být kód pro zavedení OS
- **Superblok** – informace o fs
(počet i-uzlů, datových bloků, odkaz na seznam volných bloků..)
- **i-uzly** – tabulka pevné velikosti, číslovány od 1
- **Datové bloky** – data všech souborů a adresářů, volné bloky

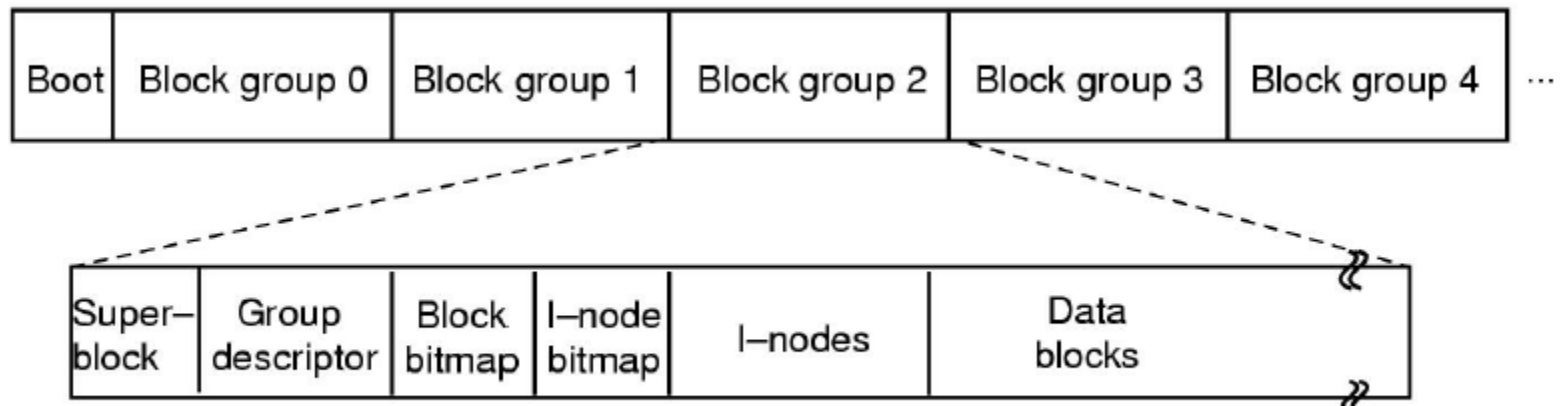


Implementace souborů – i-uzly

i-uzel obsahuje:

- Atributy
 - Jedním z nejdůležitějších je velikost, počet referencí
- Odkaz na prvních 12 datových bloků souboru
 - Může být různý počet, typicky 10-12
- Odkaz na blok obsahující odkazy na datové bloky (**nepřímý odkaz**)
- Odkaz na blok obsahující odkazy na bloky obsahující odkazy na datové bloky (**dvojitě nepřímý odkaz**)
- **Trojitě nepřímý odkaz**

Unixové systémy s využitím i-uzlů (novější, např. ext2 a výše)



skupiny i-nodů a datových bloků- v jednotlivých skupinách (block group)
duplikace nejdůležitějších údajů v každé skupině (superblock, group descriptor)

Jsou zde 2 bitmapy – který i-node je volný, který blok je volný (!!!)

Vlastnosti

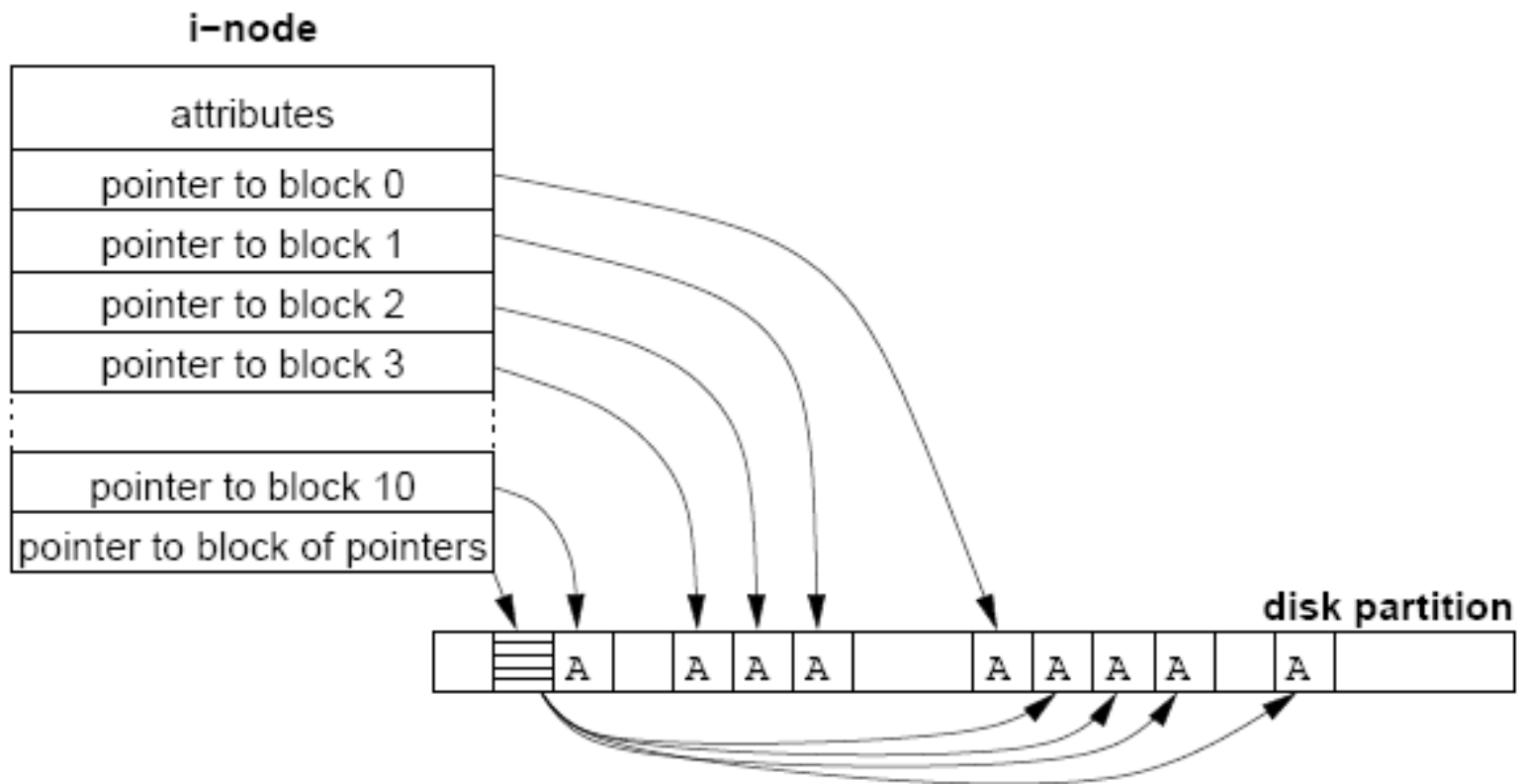
- klíčové informace jsou násobně duplikovány, např. superblock
- Bitmapa i-nodů říká, který i-node je volný
- Bitmapa datových bloků říká, který datový blok je volný
- I-nody a odpovídající datové bloky jsou blízko u sebe
- Chci vytvořit nový soubor
 - V bitmapě najdu volný i-node
 - Dále hledám v bitmapě datových bloků volné bloky pro data

I-uzly (!!)

- S každým souborem sdružena datová struktura **i-uzel** (i-node, zkratka z index-node)
- i-uzel obsahuje
 - Atributy souboru
(**velikost** souboru, **počet odkazů** na soubor, práva a pro koho jsou, časy vytvoření, modifikace,...)
 - Diskové adresy prvních N bloků souboru, typicky 10-12
 - Odkazy na adresy diskových bloků – 1. 2. 3. nepřímé úrovně

Poznámka

- Systémy s i-uzly jsou tradiční pro Unix
- Používají je dnešní filesystemy např. **ext2, ext3, ext4**



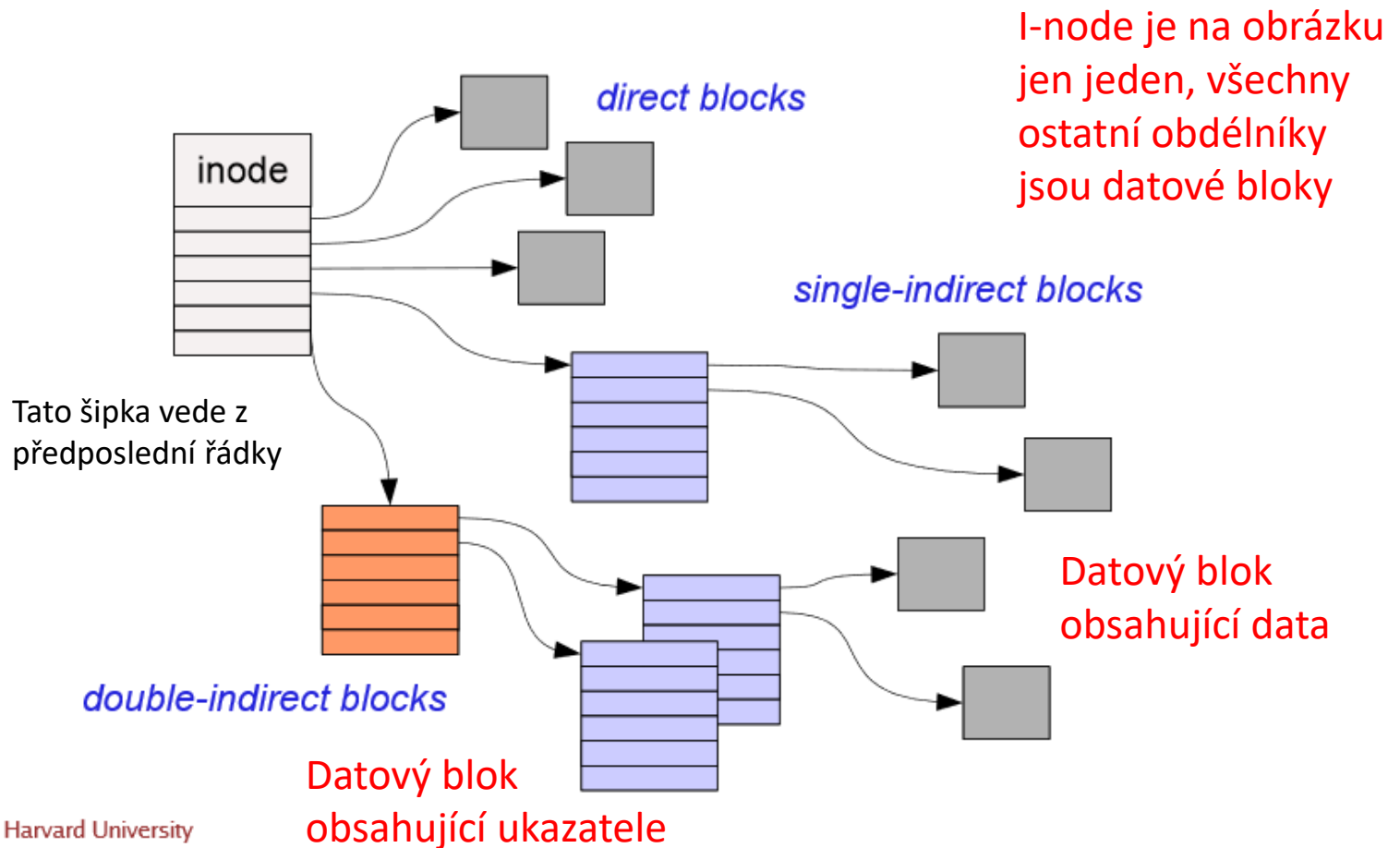
Pamatuj:

Cca 10-12 přímých odkazů na bloky obsahující data souboru

1. nepřímý – odkaz na datový blok obsahující seznam odkazů na data
2. nepřímý – viz další slide
3. nepřímý – viz další slide

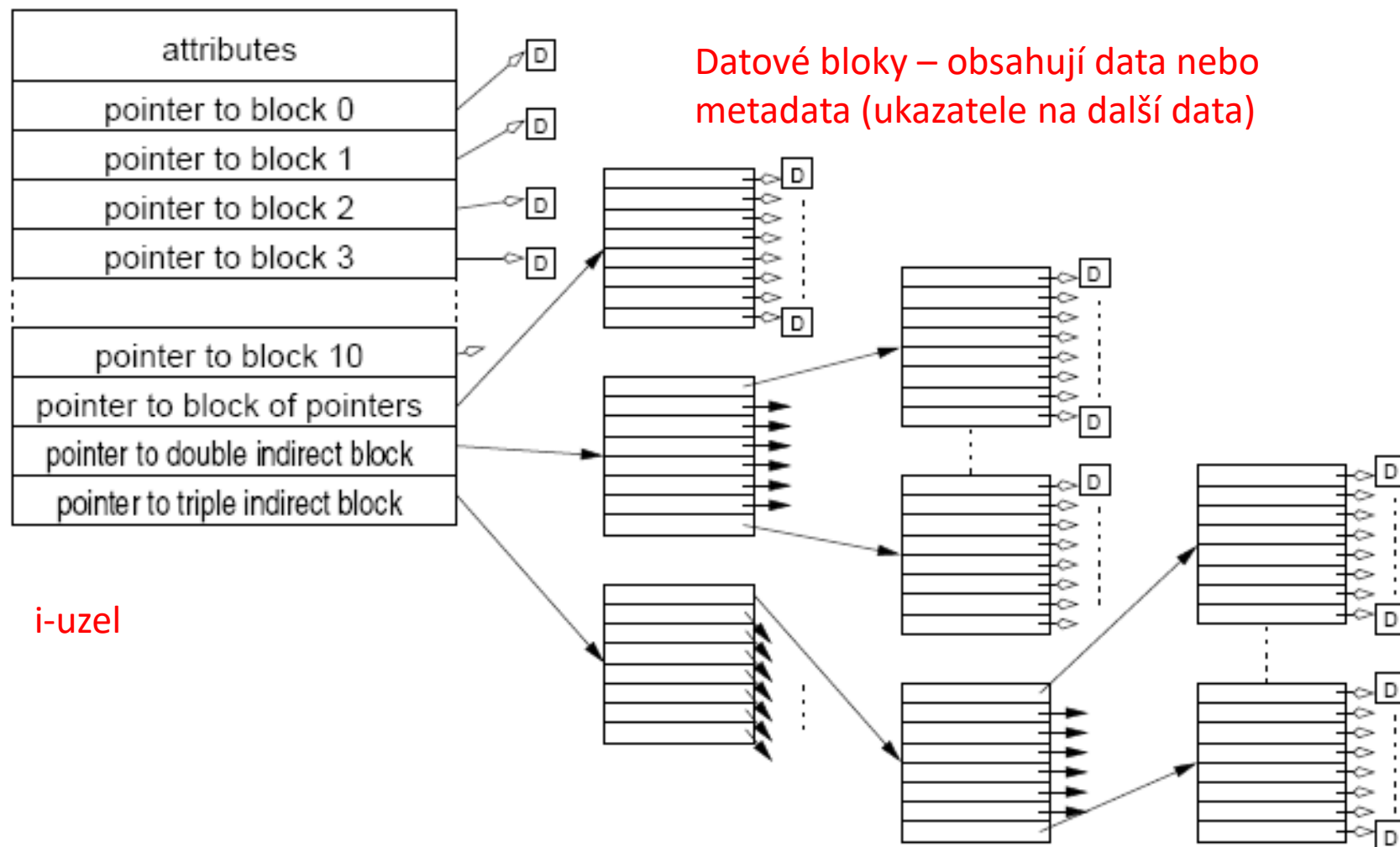
Důležité

- Nepřímé odkazy
 - Datový blok, místo aby obsahoval data souboru, tak obsahuje **odkazy na další datové bloky** využívané souborem
 - Datový blok tedy obsahuje **metadata** (zde ukazatele), místo dat souboru
- Mohou být 1., 2., 3. úrovně
 - Odkaz na blok z i-nodu -> data souboru (přímé odkazy)
 - Odkaz na blok z i-nodu -> **metadata** -> data souboru (1. úroveň)
 - Odkaz na blok z i-nodu -> **metadata** -> **metadata** -> data souboru (2. úroveň)
 - Odkaz na blok z i-nodu -> **md** -> **md** -> **md** -> data souboru (3. úroveň)



- Malé soubory – přímé odkazy na datové bloky => rychlý přístup k nim
- Velké soubory – využívají i nepřímé odkazy
- i-uzel má pevnou velikost – stejnou pro malý i velký soubor

UNIX v7 i-node



Datové bloky – obsahují data nebo metadata (ukazatele na další data)

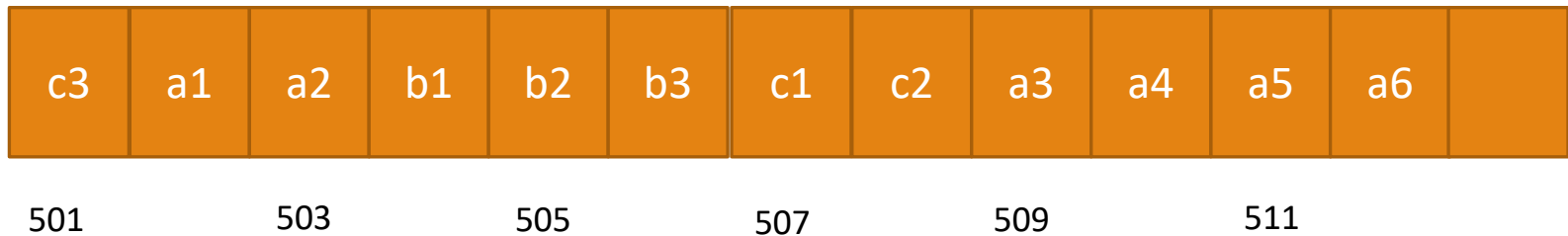
i-uzel

I-uzly - výhoda

- Po otevření souboru můžeme **zavést i-uzel** a případný blok obsahující další adresy **do paměti** => urychlení přístupu k souboru

Příklad 3

Nakreslete i-uzly odpovídající uložení souborů a.txt, b.txt, c.txt
Jak bude vypadat obsah hlavního adresare?



i-uzel 901

Velikost: 5500
Poč. odkazů: 1
vlastník: x
skupina: y,
práva (u,g,o)
Časy (vytv,
modif.)

502

503

509

510

511

512

Nepřímé
1. řád
2. Řád
3. Řád

i-uzel 902

Velikost: 2200
Poč. odkazů: 1
vlastník: x
skupina: y,
práva (u,g,o)
Časy (vytv,
modif.)

504

505

506

Nepřímé
1. řád
2. Řád
3. Řád

i-uzel 903

Velikost: 2500
Poč. odkazů: 1
vlastník: x
skupina: y,
práva (u,g,o)
Časy (vytv,
modif.)

507

508

501

Nepřímé
1. řád
2. Řád
3. Řád

Příklad 3 - adresář

Obsah adresáře bude:

a.txt 901
b.txt 902
c.txt 903

Jak by se adresář změnil po příkazu `ln b.txt d.txt`?

a.txt 901
b.txt 902
c.txt 903
d.txt 902

V i-uzlu 902 by se také počet odkazů zvětšil na 2 (dvě jména odkazují na stejný soubor).

Příklad 4

Předpokládejme **počet přímých odkazů v i-uzlu budou 3**.
Soubor a má velikost 6500B, alokační jednotka 1024B.

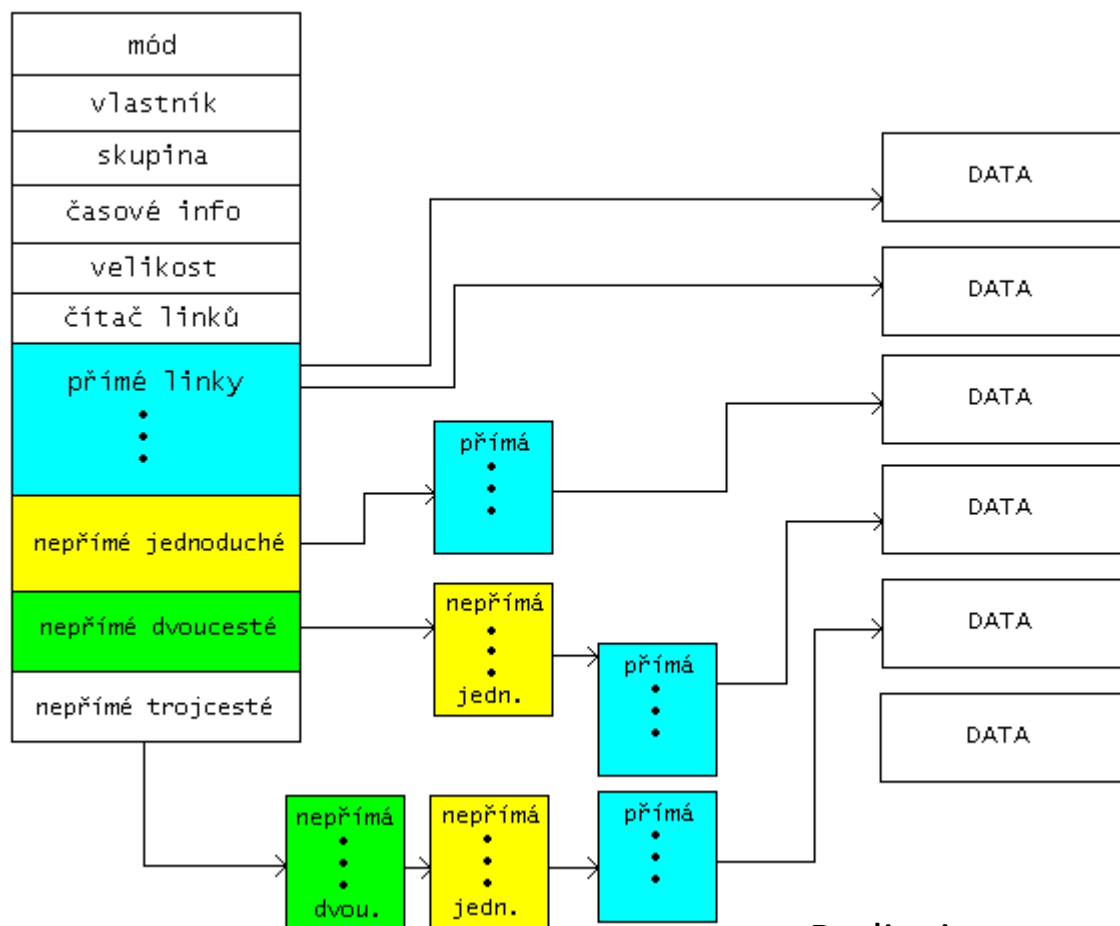
	a1	a2	a3	a4		a5	a6	504 506 507			
	501		503		505		507		509		511

i-uzel atributy (velikost, počet odkazů, práva, časy)
501
502
503
508

<- i-uzel 905

<- 1. nepřímý, odkazuje na seznam ukazatelů

i-uzly dle normy POSIX



Podívejte se:

<https://www.youtube.com/watch?v=tMVj22EWg6A>

zdroj: <http://cs.wikipedia.org/wiki/Inode>

i-uzly dle normy Posix

- **MODE** – typ souboru (obyčejný, adresář, znakový / blokový, FIFO)
- **přístupová práva** (u,g,o) – bity přístupových práv
pro vlastník, skupina, ostatní
- **REFERENCE COUNT** – počet odkazů na tento objekt
(vytvoření hardlinku zvyšuje počet)
- **OWNER** – ID vlastníka
- **GROUP** – ID skupiny
- **SIZE** – velikost objektu
- **TIME STAMPS**
 - atime – čas posledního přístupu (čtení souboru, výpis adresáře)
 - mtime – čas poslední změny
 - ctime – čas poslední změny i-uzlu (metadat)

i-uzly dle normy POSIX

- **DIRECT BLOCKS** – 12 přímých odkazů na datové bloky (data v souboru)
- **SINGLE INDIRECT** – 1 odkaz na datový blok, který **místo dat** obsahuje seznam přímých odkazů na datové bloky obsahující vlastní data souboru
- **DOUBLE INDIRECT** – 1 odkaz 2. nepřímé úrovně
- **TRIPLE INDIRECT** – 1 odkaz 3. nepřímé úrovně

v linuxových fs (např. ext4) ještě FLAGS, počet použitých datových bloků a rezervovaná část – doplňující info
(odkaz na rodičovský adresář, **ACL**, rozšířené atributy)

Implementace adresářů

- Před čtením je třeba soubor otevřít
- *open (jméno, režim)*
- Mapování jméno -> info o datech
poskytují adresáře !
- Adresáře jsou často speciálním typem souboru
- Typicky pole datových struktur, 1 položka na soubor

2 základní uspořádání adresáře (!!!)

1. Adresář obsahuje **jméno souboru, atributy, diskovou adresu souboru** (např. adresa 1.bloku)
(používá např. FAT)
2. Adresář obsahuje **pouze jméno + odkaz** na jinou datovou strukturu obsahující další informace, jako je i-uzel
(používá systém s i-uzly)

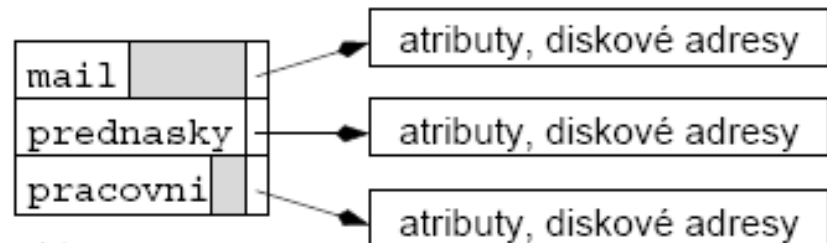
V praxi se používají oba dva způsoby (adresáře FAT vs. i-uzly)

2 základní uspořádání adresáře (!!)

mail		atributy, diskové adresy
prednasky		atributy, diskové adresy
pracovni		atributy, diskové adresy

a)

- a) Používá např. FAT
musí vědět:
- na jakém bloku soubor začíná
 - jak je soubor velký



b)

- b) Používá např. ext2, ext3
jméno a odkaz
obecně systémy s i-uzly
(i-uzel neobsahuje jméno souboru)

Adresáře v UNIXu

- adresář: obsahuje jméno souboru a číslo i-uzlu
- Číslo i-uzlu je indexem do tabulky i-uzlů na disku
- Každý soubor a adresář: právě 1 i-uzel
- V i-uzlu: všechny atributy a čísla diskových bloků
- Kořenový adresář: číslo i-uzlu 1

Příklad – adresář win 98

* položka adresáře obsahuje:

- jméno souboru (8 bytů) + příponu (3 byty)
- atributy (1)
- NT (1) - Windows 98 nepoužívají (rezervováno pro WinNT)
- datum a čas vytvoření (5)
- čas posledního přístupu (2)
- horních 16 bitů počátečního bloku souboru (2)
- čas posledního čtení/zápisu
- spodních 16 bitů počátečního bloku souboru (2)
- velikost souboru (4)

* dlouhá jména mají pokračovací položky

* veškeré "podivnosti" této struktury jsou z důvodu kompatibility s MS DOSem

cd Progra~1 vs. cd Program Files

Hard link (příkaz ln)

Soubor ve více podadresářích nebo pod více jmény

Hard links (pevné odkazy)

- Každý soubor má datovou strukturu, která ho popisuje (i-uzel), můžeme vytvořit v adresářích více odkazů na stejný soubor
- Všechny odkazy (jména) jsou rovnocenné
- V popisu souboru (i-uzlu) musí být počet odkazů
- Soubor zanikne při zrušení posledního odkazu
- Omezení
 - Není povoleno na adresáře, a také lze jen v rámci daného fs

Hard link v Linuxu: **ln** stare_jmeno nove_jmeno

Symbolický link (ln -s)

Symbolický link

- Nový typ souboru, obsahuje jméno odkazovaného souboru
- OS místo symbolického odkazu otevře odkazovaný soubor
- Obecnější – může obsahovat cokoliv
- Větší režie

Symbolický link v Linuxu: **ln -s** stare_jmeno nove_jmeno

Správa volného prostoru

Informace, které bloky jsou volné

Nejčastěji – bitová mapa nebo seznam

Bitová mapa

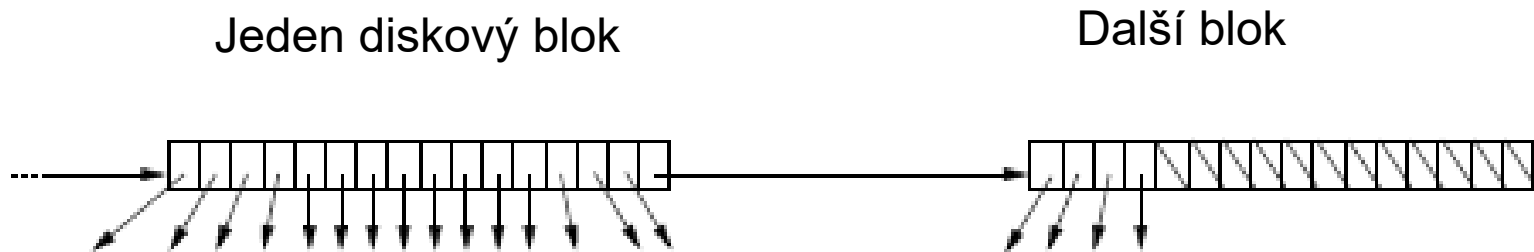
- Konstantní velikost
- Možnost vyhledávání volného bloku s určitými vlastnostmi (např. n volných bloků za sebou)
- Většina současných fs používá bitovou mapu

Správa volného prostoru

Seznam diskových bloků

- Blok obsahuje **odkazy na volné bloky** a **adresu dalšího bloku** ...
- Uvolnění bloků - Přidáme adresy do seznamu, pokud není místo blok zapíšeme
- Potřebujeme bloky pro soubor – používáme adresy ze seznamu, pokud nejsou přečteme další blok adres volných bloků
- Pokud není na disku volné místo, seznam volných bloků je prázdný a nezabírá místo

Seznam diskových bloků



Blok obsahuje:

- Odkazy na volné bloky
- Adresu dalšího bloku v seznamu

Kvóty

Účel – aby uživatel neobsadil celý disk a nechal místo i pro ostatní

Maximální počet bloků obsazených soubory uživatele

Ve víceuživatelských OS, na serverech

Hard kvóta

- Pevná mez, uživatel ji nepřekročí

Soft kvóta

- Po překročení uživatel dostane varování
- **Grace period** – po zadanou dobu může překročit soft kvótu, po uplynutí času už více neuloží

Spolehlivost souborového systému

Ztráta dat má často horší důsledky než zničení počítače

- diplomová / bakalářská práce
- Fotografie za posledních 10 let

Filesystém musí být jedna z nejspolehlivějších částí OS, snaha chránit data

- Správa vadných bloků (hlavně dříve)
- Rozprostřít a duplikovat důležité datové struktury, čitelnost i po částečném poškození povrchu

Konzistence fs

Blokové zařízení

OS přečte blok souboru, změní ho, zapíše

Nekonzistentní stav

může nastat při havárii (např. výpadek napájení) předtím, než jsou všechny modifikované bloky zapsány

Kontrola konzistence fs

Windows: **chkdsk**

UNIX: **fsck**, **fsck.ext3**, **e2fsck** .. viz man

Kontrolu spustí automaticky po startu, když detekuje nekorektní ukončení práce se systémem

Testy konzistence fs

Konzistence informace o diskových blocích souborů

- Blok patří jednomu souboru nebo je volný

Konzistence adresářové struktury

- Jsou všechny adresáře a soubory dostupné?

důležité pochopit rozdíl:

- kontrola konzistence souboru
- kontrola, zda je soubor dostupný z nějakého adresáře

Konzistence informace o diskových blocích souborů

- Tabulka obsahující
 - počet výskytů bloku v souboru
 - počet výskytů bloku v seznamu volných bloků
1. Položky v tabulce inicializovány na 0
 2. Procházíme informace o souborech (např. i-uzly), inkrementujeme položky odpovídající blokům souboru v první řádce tabulky (!!)
 3. Procházíme seznam nebo bitmapu volných bloků a inkrementujeme příslušné položky v druhé řádce tabulky

Konzistentní fs

Číslo bloku	0	1	2	3	4	5	6	7	8
Výskyt v souborech	1	0	1	0	1	0	2	0	1
Volné bloky	0	1	0	0	1	0	0	1	0

Blok je buď volný, nebo patří nějakému souboru, tj. konzistentní hodnoty v daném sloupci jsou buď (0,1) nebo (1,0)
Vše ostatní jsou chyby různé závažnosti

Možné chyby, závažnosti

(0,0) – blok se nevyskytuje v žádné řádce

- Missing blok
- Není závažné, pouze redukuje kapacitu fs
- Oprava: vložení do seznamu volných bloků

(0,2) – blok je dvakrát nebo vícekrát v seznamu volných

- Nemůže nastat u bitmapy !!!!!!!
- Jen pokud by bylo volné místo řešeno seznamem volných bloků a jedna položka v něm dvakrát
- Opravíme seznam volných bloků, aby se vyskytoval pouze jednou

Možné chyby, závažnosti

(1,1) – blok patří souboru a zároveň je na seznamu volných

- Problém, blok by mohl být alokován podruhé !
- Oprava: blok vyjmemme ze seznamu volných bloků
- Zde můžeme předejít závažné chybě, zde má prevence smysl

(2,0) – blok patří do dvou nebo více souborů

- Nejzávažnější problém, nejspíš už došlo ke ztrátě dat
- Snaha o opravu: alokujeme nový blok, problematický blok do něj zkopírujeme a upravíme i-uzel druhého souboru
- Uživatel by měl být informován o problému

Je zde nějaká chyba? A když tak jaká?

	číslo bloku:	0	1	2	3	4	5	6	7	8	9	11	12	13	14	15
výskyt	v souborech:	1	1	0	0	1	0	0	1	1	1	1	0	1	0	0
	volné bloky:	0	0	1	1	0	1	1	0	0	0	0	1	0	1	1

	číslo bloku:	0	1	2	3	4	5	6	7	8	9	11	12	13	14	15
výskyt	v souborech:	1	2	0	0	1	0	0	1	1	1	1	1	1	0	0
	volné bloky:	0	0	1	1	0	0	1	0	0	0	0	1	0	1	1

Kontrola konzistence adresářové struktury

- Tabulka čítačů, jedna položka pro každý soubor
- Program prochází rekurzivně celý adresářový strom
- Položku pro soubor program zvýší pro každý výskyt souboru v adresáři
- Zkontroluje, zda odpovídá počet odkazů v i-uzlu (*i*) s počtem výskytů v adresářích (*a*)
- $i == a$ 😊 pro každý soubor

Možné chyby

i > a

soubor by nebyl zrušen ani po zrušení všech odkazů v adresářích

není tolik závažné, ale soubor by zbytečně zabíral místo
řešíme nastavením počtu odkazů v i-uzlu na správnou hodnotu (a)

Možné chyby

i < a

soubor by byl zrušen po zrušení i odkazů, ale v adresářích budou ještě jména

velký problém – adresáře by ukazovaly na neexistující soubory

řešíme nastavením počtu odkazů na správnou hodnotu

Možné chyby

a=0 , i > 0

ztracený soubor, na který není v adresáři odkaz

ve většině systémů program soubor zviditelní na předem určeném místě

(např. adresář lost+found)

Další heuristické kontroly

Odpovídají jména souborů konvencím OS?

- Když ne, soubor může být nepřístupný, změníme jméno

Nejsou přístupová práva nesmyslná?

- Např. vlastník nemá přístup k souboru,...

Zde byly uvedeny jen základní obecné kontroly fs

Journaling fs

Kontrola konzistence je časově náročná

Journaling fs

- Před každým zápisem na disk vytvoří na disku záznam popisující plánované operace, pak provede operace a záznam zruší
- Výpadek – na disku najdeme žurnál o všech operacích, které mohly být v době havárie rozpracované, zjednodušuje kontrolu konzistence fs

Příkladem fs s žurnálem je např. ext3, ext4

Příklad transakce

při vytváření souboru se vyrobí transakce (posloupnost operací, kterou je potřeba celou provést) do které náležejí následující operace:

- zapsání položky adresáře
- zapsání do i-uzlu popisujícího nový soubor
- zapsání bitu v bitmapě alokovaných i-uzlů
- zmenšení počítadla volných i-uzlů v superbloku.

Jak funguje žurnál (!!)

1. Zapiši do žurnálu plánované změny
2. Když je žurnál kompletní, zapišeme značku ZURNAL_KOMPLETNI
3. Začneme zapisovat datové bloky
4. Je-li hotovo, smažeme žurnál

Žurnál – ošetření výpadku (!!)

Dojde-li k výpadku elektřiny → nebyl korektně odmontovaný oddíl se souborovým systémem → pozná

Podívá se do žurnálu:

- a) **Je prázdný**
→ není třeba nic dělat
- b) **Je tam nějaký zápis, ale není značka ZURNAL_KOMPLETNI**
→ jen smažeme žurnál
- c) **V žurnálu je zápis včetně značky ZURNAL_KOMPLETNI**
→ přepíšeme obsah žurnálu do datových bloků

Co žurnálovat?

Všechny zápisy, tj. i do souborů

- Zapisují se metadata i data
- pomalejší

Zápisy metadat

- Rychlejší
- Např. že vytváříme, mažeme, přesunujeme soubor ale neukládá do žurnálu vlastní obsah souboru
- Může dojít ke ztrátě obsahu souboru, ale nerozpadne se struktura adresářů

Výkonnost úložiště

Přístup k tradičnímu rotačnímu disku řádově pomalejší než přístup do paměti

- Seek 5-10 ms
- Rotační zpoždění – až bude požadovaný blok pod hlavičkou disku
- Rychlost čtení (x rychlost přístupu do paměti)

Použití **SSD** disků

- Rychlé, lehké, malá spotřeba (výdrž notebooků)
- menší kapacita, dražší

Kombinace

- SSD disk na operační systém
- Rotační disk na data

Výkonnost - cachování

- Cachování diskových bloků v paměti
- Přednačítání (read-ahead)
do cache se předem načítají bloky, které se budou potřebovat při sekvenčním čtení souboru:

čtu blok A10 a rovnou nakešuji i blok A11

Redukce pohybu diskového raménka pro po sobě následující bloky souboru,...

Mechanismy ochrany

Chránit soubor před neoprávněným přístupem

Chránit i další objekty

- HW (segmenty paměti, I/O zařízení)
- SW (procesy, semafore, ...)

Subjekt – entita schopná přistupovat k objektům
(většinou **proces**)

Objekt – cokoliv, k čemu je potřeba omezovat přístup pomocí
přístupových práv (např. **soubor**)

System **uchovává informace** o přístupových právech subjektů k
objektům

ACL x capability list

Dvě různé podoby

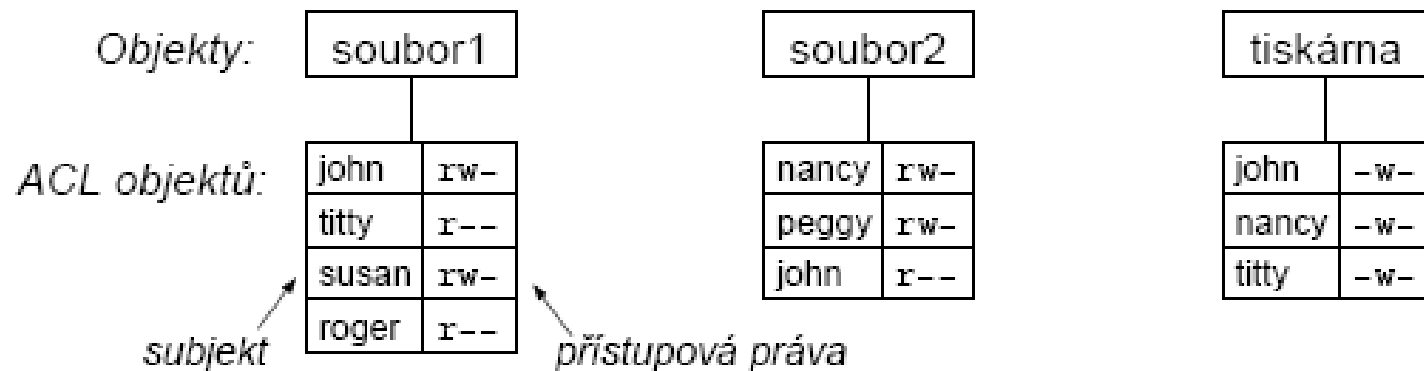
ACL – s objektem je sdružen seznam subjektů a jejich přístupových práv

Capability list [kejpa-] – se subjektem je sdružen seznam objektů a přístupových práv k nim

ACL (Access Control Lists)

S objektem je sdružen seznam subjektů, které mohou k objektu přistupovat

Pro každý uvedený subjekt je v ACL množina přístupových práv k objektu



ACL

Sdružování subjektů do tříd nebo do skupin

- Studenti
- Zamestnanci

Skupiny mohou být uvedeny na místě subjektu v ACL

Zjednodušuje administraci

- Nemusíme uvádět všechny studenty jmenovitě

ACL používá mnoho moderních filesystemů
(ntfs, xfs, ...)

ACL – příklad (!)

Např v **NTFS**:

Se souborem **data1.txt** je spojena následující **ACL tabulka**, která určuje, kdo smí co s daným souborem dělat. Počet řádek tabulky záleží na tom, pro kolik uživatelů skupin budeme práva nastavovat.

Klasická unixová práva (u,g,o) jsou příliš limitovaná – když chceme více skupin, více uživatelů atd. potřebujeme ACL.

uživatel / skupina	id uživatele	práva
0	505 (Pepa)	rw
1	101 (Studenti)	r
1	102 (Zamestnanci)	rw

Klasická unixová práva

■ `chmod 777 s1.txt`

- Práva pro vlastníka (u)
- Práva pro skupinu (g)
- Práva pro ostatní (o)
- Typ práv: r, w, x, (s, t)

■ Oproti ACL jsou omezující:

- Chceme pro více skupin různá nastavení
- Chceme pro více uživatelů různá nastavení

Klasická unixová práva u,g,o nejsou považována za ACL, Naopak systémy které je využívají se o ACL rozšiřují

Klasická práva vs. ACL

- Viz následující příklad
- Nastavovat tak, aby nebylo pochybností

ACL má přednost před klasickými unixovými právy
(nastavenými chmod)

-> nastavením chmod 777 nezrušíme ACLka 😊

Viz příklad dále

Ukázka I.

- Tři uživatelé: pesi, tom, tom2 na domácím Linuxu
- Pracujeme pod uživatelem pesi
- 1. `echo pokusny text > pokus.txt`
- 2. `ls -l pokus.txt` (výpis je OK)
- 3. `getfacl pokus.txt` (ACL k pokus.txt)
- 4. `setfacl -m tom:x pokus.txt` (změním ACL práva uživateli tom)
- 5. `chmod 777 pokus.txt` (klasická linuxová práva)
 - `//` napřed nastavím ACL
 - `//` následně klasická linuxová práva 777
 - `//` `chmod 777` nepřebije předchozí ACL nastavení

Ukázka II.

- 6. `sudo tom` (přepnutí na uživatele tom)
- 7. `cat pokus.txt` (permission denied)
- 8. `sudo tom2 ..` (přepnutí na uživatele tom2, su – tom2)
- 9. `cat pokus.txt` (výpis je OK)

Závěr:

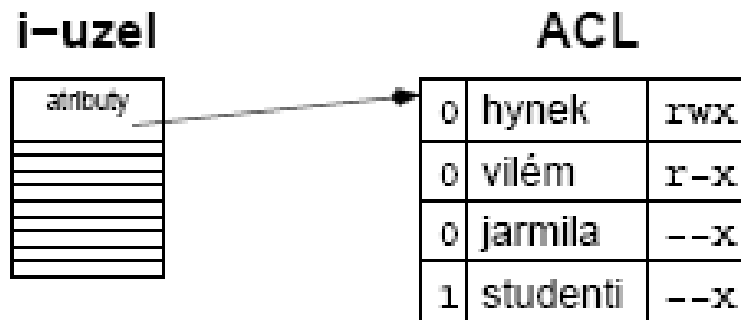
očekávalo by se, že `chmod 777` udělí práva všem, ale nastavení ACL zamezí přístup uživateli tom

Úloha: jak doimplementovat ACL do i-uzlu?

V i-uzlu by byla **část tabulky ACL**, pokud by se nevešla celá do i-uzlu, tak **odkaz na diskový blok** obsahující **zbytek ACL**

Každá položka ACL

- Subjekt: id uživatele či id skupiny + 1 bit rozlišení uživatel/skupina
- Přístupové právo Nbitovým slovem
1 – právo přiděleno, 0 – právo odejmuto



ACL příkazy pod Linuxem

`getfacl s1.txt`

`setfacl -m user:pepa:rw s1.txt`

Další informace:

<http://www.abclinuxu.cz/clanky/bezpecnost/acl-prakticky>

ACL a Linuxové fs

- `tune2fs -o acl /dev/sdXY`
- ACL je defaultní mount option u ext2,3,4 souborových systémů
- `setfacl -m "u:johny:rwx" abc`
 - Nastaví práva pro uživatele johny na soubor abc
- `getfacl abc`

ACL a Linuxové fs

getfacl abc

file: abc

owner: someone

group: someone

user::rw-

user:johny:rwx

group::r--

mask::rwx

other::r--

Jak zjistím, že se ACL používá?

Identifying files/directories that have ACL's

While the standard unix permissions are displayed with the `ls -l` command; the defined ACL's are a little more verbose and are not a part of the long listing. The command `ls` will tell you if a file or directory does have acl's, it's just not that obvious.

```
root@testvm:/var/tmp# ls -la | grep appdir
drwxrwxr-x+ 2 root appgroup 4096 May 27 10:45 appdir
```

As you can see there is now a `+` at the end of the directories permissions. This `+` is the indicator that this file or directory has acl's, from here you can use the `getfacl` command to see what they are.

Mechanismus capability lists (C-seznamy)

- S každým subjektem (procesem) sdružen seznam objektů, kterým může přistupovat a jakým způsobem (tj. přístupová práva)
- Seznam se nazývá capability list (C-list)
- Jednotlivé položky - capabilities

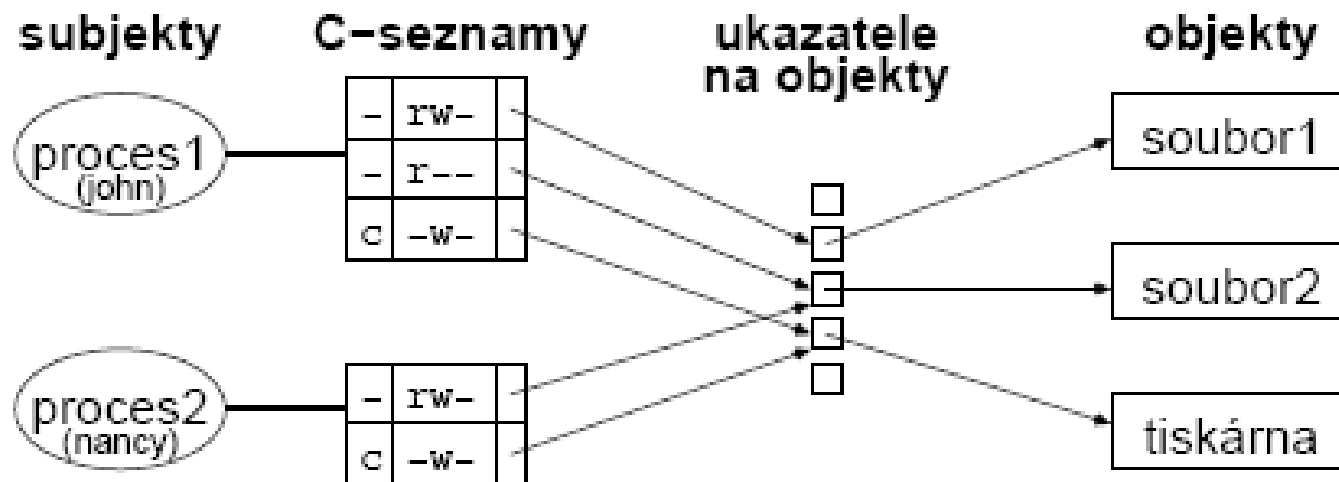
Struktura capability

- Struktura capability
 - Typ objektu
 - Práva – obvykle bitová mapa popisující dovolené operace nad objektem
 - Odkaz na objekt, např. číslo uzlu, segmentu, atd..

Capability

- Problém – zjištění, kdo všechno má k objektu přístup
- **Zrušení přístupu velmi obtížné** – najít pro objekt všechny capability + odejmout práva
- Řešení: odkaz neukazuje na objekt, ale na **nepřímý objekt** systém může zrušit nepřímý objekt, tím zneplatní odkazy na objekt ze všech C-seznamů

Capability list



Capability list

Pokud jsou jediný způsob odkazu na objekt (bezpečný ukazatel, capability-based addressing):

Ruší rozdíl mezi objekty na disku, v paměti (segmenty) nebo na jiném stroji (objekty by šlo přesouvat za běhu)

Mechanismus C-seznamů v některých distribuovaných systémech (Hydra, Mach,...)

Přístupová práva

FAT – žádná

- Jen atributy typu read-only, archive, ...

ext2, ext3, ext4

- klasická unixová práva (není to ACL)
- vlastník, skupina, ostatní (r,w,x,s,...)
- lze přidat ACL

NTFS

- ACL
- lze měnit přes průzkumníka souborů, příkaz icacls, ...
- explicitně udělit / odeprít práva
- zdědit práva, zakázat dědění

Proč je tolik filesystemů?

- Různé fyzické vlastnosti úložišť
 - ext3 – magnetické disky
 - iso9660 – DVD, CDROM
- Různé kapacity
 - FAT16 – disky do 2GB
 - FAT32 – vhodná pro disky do 32GB, ale problém se soubory > 4GB
 - Btrfs – multi TB disková pole
- Různé požadavky
 - FAT32 budete mít na PC spíš jen někde na SD kartě, flashce
- Otevřenost standardů
 - NTFS – uzavřená specifikace (ale zlepšuje se)

Zálohování - motivace

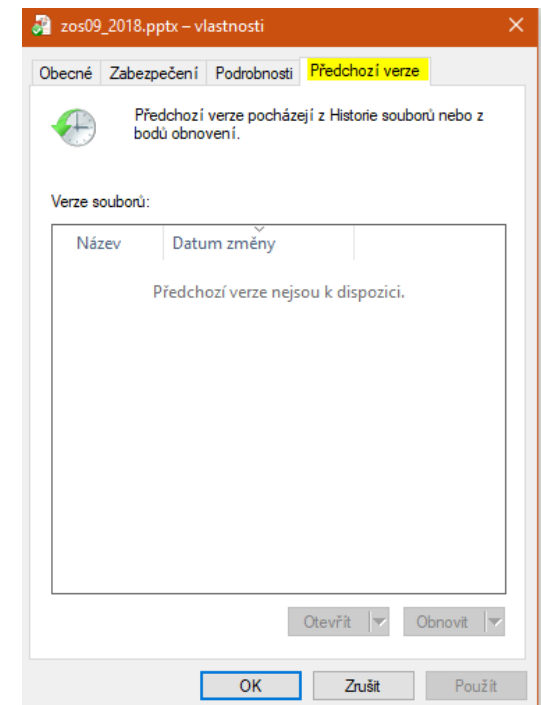
- Uvědomit si rozdíl mezi zálohováním a bezpečným úložištěm !
- Můžeme mít poměrně bezpečné úložiště RAID 6 nebo RAID 1, ale pokud si uživatel omylem smaže nějaký soubor, tak mu RAID nepomůže

Recycle bin, shadow copies

Některé systémy se snaží předcházet situaci, kdy si uživatel omylem smaže data.

Recycle bin – možnost obnovy z koše

Předchozí verze – vrátit se k původní verzi souboru



Zálohování

- Na externí médium (externí hardisk, DVD)
- Po síti – cloud, síťová složka

Pozor na **ransomware** viry

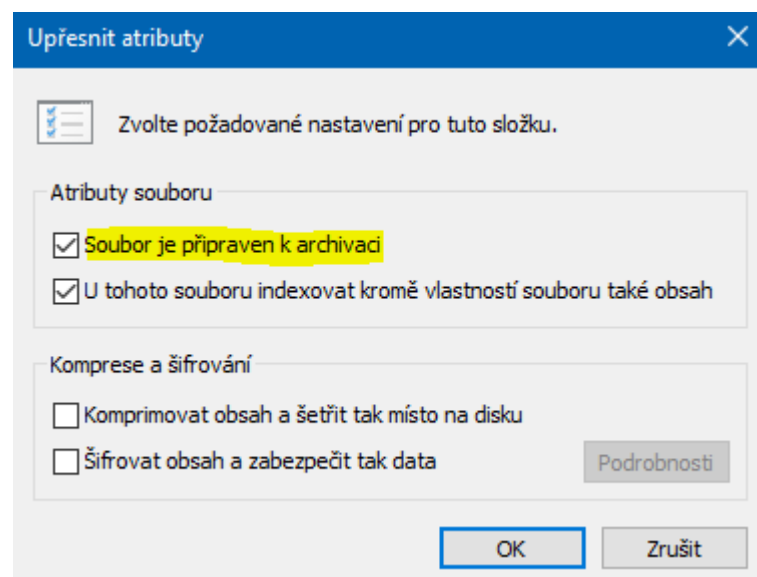
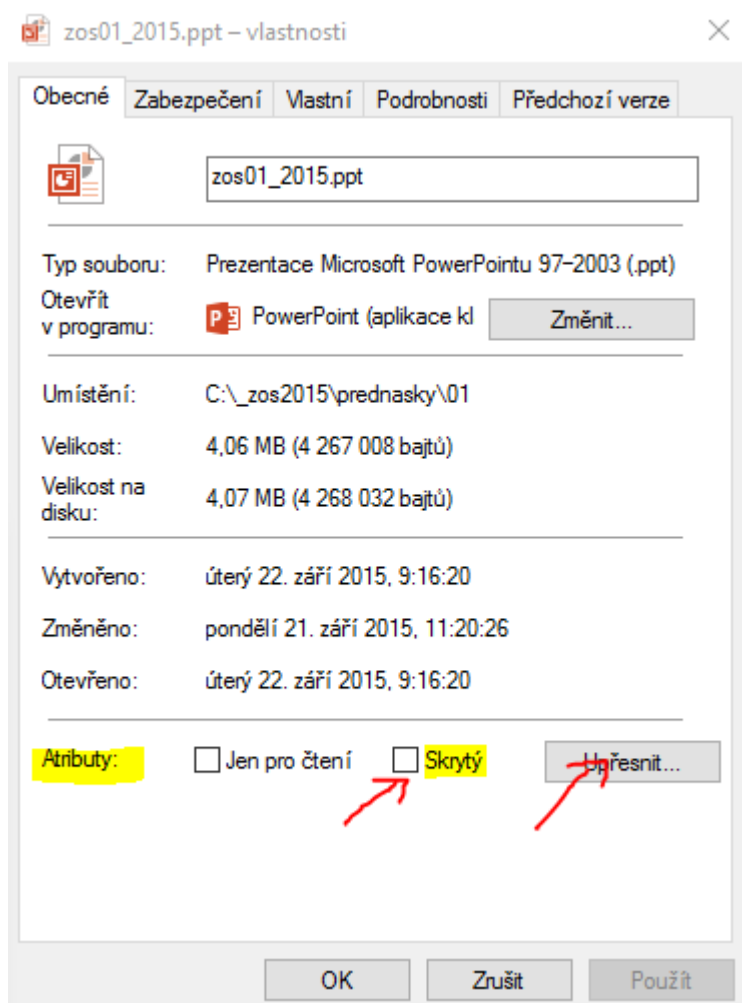
- Šifrují všechny dostupné síťové jednotky (lokální i síťové)
- Chtějí výpalné za odšifrování souborů

Zálohování

Důležitou otázkou je: **Co zálohovat?**

Atribut souboru **Archive**

- Nové soubory jej mají nastavený
- Při změně stávajícího souboru se také nastaví
- Při zálohování (např. inkrementálním) se atribut vynuluje
- Umožní dělat inkrementální zálohy



Typy záloh



manipulací s
atributem
archive

- **plná** (normální)
 - zálohuje, označí soubory jako "zazálohované,, (atribut archive schodí)
- **incremental** (přírůstková)
 - zálohuje pouze vybrané soubory, tj. pokud nebyly "zazálohované" nebo byli změněny a označí je jako "zazálohované,,
 - Potřebuji plnou + všechny následující incrementalní
- **differential** (rozdílová)
 - viz předchozí, ale neoznačuje jako "zazálohované,, (nemanipuluje s atrib. archive)
 - změny, které proběhly od plné zálohy
 - diferenciální zálohy nejsou na sobě závislé
 - Potřebuji plnou + poslední rozdílovou
- **copy**
 - zálohuje vše, ale neoznačí jako "zazálohované" (nemanipuluje s atributem archive)
 - nenaruší používané schéma zálohování
- **daily**
 - zálohuje soubory změněné dnes, ale neoznačuje je jako "zazálohované"

Typy záloh



zdroj a dobrý materiál k přečtení:

<http://www.acronis.cz/kb/diferencialni-zaloha/>

Co se děje při spuštění PC?

01. Pustíme proud do počítače 😊

02. **Power on self-test** (řízen BIOSem)

- test operační paměti, grafické karty, procesoru
- test pevných disků, dalších ATA/SATA/USB zařízení

03. spustí z ROM paměti BIOSu **bootstrap loader**

- prohledá boot sektor bootovacího zařízení (dle CMOS paměti)
- boot sektor - první sektor na disku, je zde umístěn zavaděč systému (boot loader)

http://cs.wikipedia.org/wiki/Power_On_Self_Test

Co se děje při spuštění PC?

- 04. pustí se zavaděč (např. **GRUB2**, **Windows zavaděč** ...)
 - může se skládat z více stupňů (stage),
 - v boot sektoru je stage1
 - možnost zvolit si jaký systém nabootuje (Linux, Windows)

- 05. zavaděč nahraje jádro do paměti a spustí ho
 - jádro píše na obrazovku info zprávy (viz příkaz **dmesg**)
 - připojí počáteční RAMdisk (**initrd**) – dočasný kořenový fs během zavádění
 - kořenový fs se později, jakmile je přístupný (nahrané ovladače), nahradí skutečným (**/etc/fstab**)

Co se děje při spuštění PC?

06. první proces **init**, nebo **systemd**
(systemd nahrazuje tradiční init proces)

/sbin/init , načte /etc/inittab , spouštění a vypínání služeb
/etc/rc.d/rcX.d

07. spustí program **getty** na virtuálních terminálech
zadáme uživatelské jméno

08. spustí se **login**

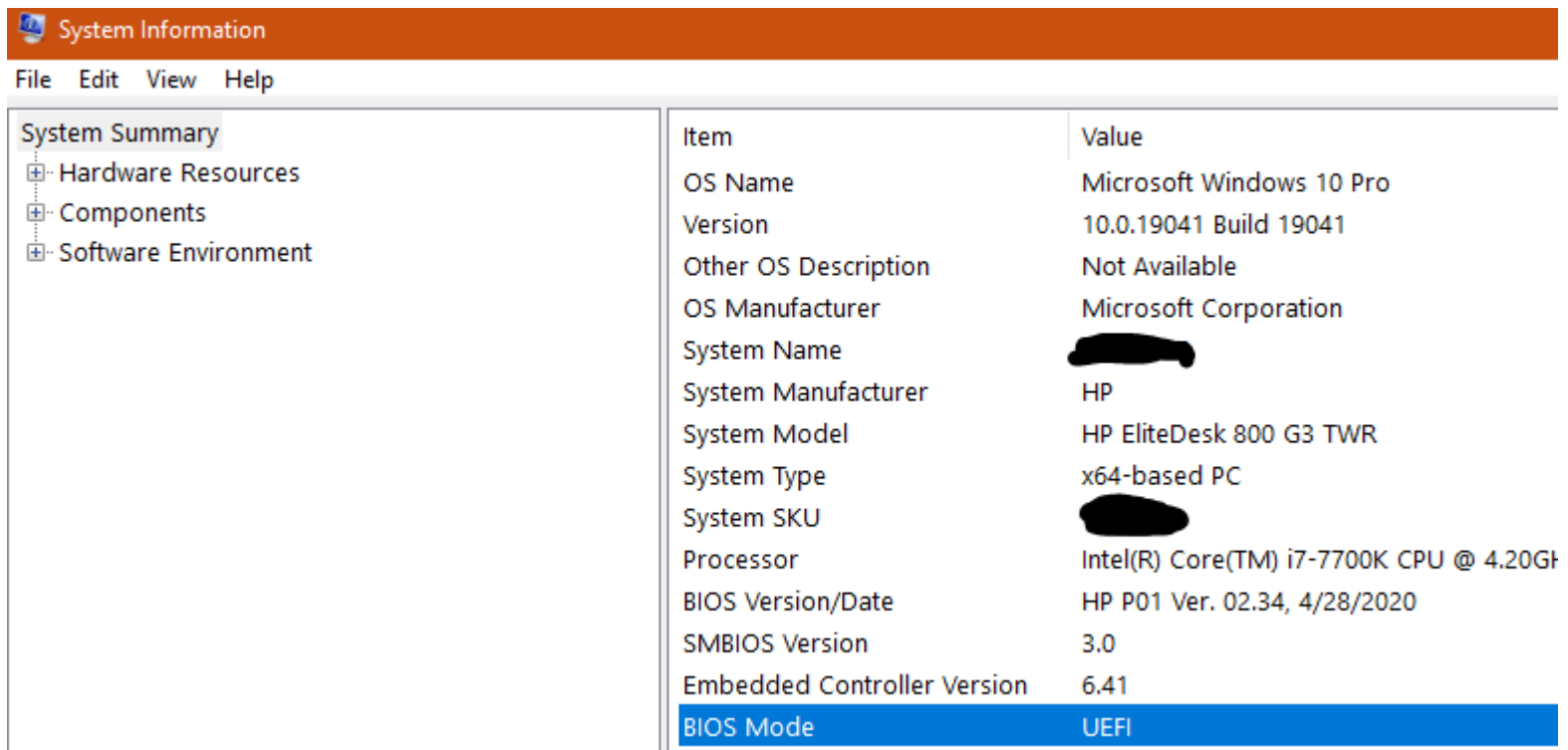
vyžádá si heslo, zkontroluje /etc/shadow či jinde, v /etc/passwd ví jaký shell bude uživatel používat, jaký má domovský adresář atp.

BIOS vs. UEFI

- BIOS - omezení
 - Je už léta, sice se vyvíjel (např. ACPI a podpora sleep), ale má řadu omezení
 - 16bit kód, 1MB paměti
 - Pomalá inicializace více HW zařízení souběžně
- UEFI
 - Nová zařízení podporují UEFI
 - Používá GPT rozdělení disku místo MBR
 - Běží v 32/64bit režimu
 - Podporuje Secure Boot – kontrola, zda škodlivý kód nenarušil proces bootování OS

UKÁZKA

Windows 10 – System Information (BIOS Mode: UEFI nebo Legacy)



The screenshot shows the Windows 10 System Information application. The left sidebar contains a tree view with 'System Summary' selected, and sub-items: 'Hardware Resources', 'Components', and 'Software Environment'. The main pane displays a table of system information.

Item	Value
OS Name	Microsoft Windows 10 Pro
Version	10.0.19041 Build 19041
Other OS Description	Not Available
OS Manufacturer	Microsoft Corporation
System Name	[REDACTED]
System Manufacturer	HP
System Model	HP EliteDesk 800 G3 TWR
System Type	x64-based PC
System SKU	[REDACTED]
Processor	Intel(R) Core(TM) i7-7700K CPU @ 4.20GHz
BIOS Version/Date	HP P01 Ver. 02.34, 4/28/2020
SMBIOS Version	3.0
Embedded Controller Version	6.41
BIOS Mode	UEFI