Operační systémy

IOS 2019/2020

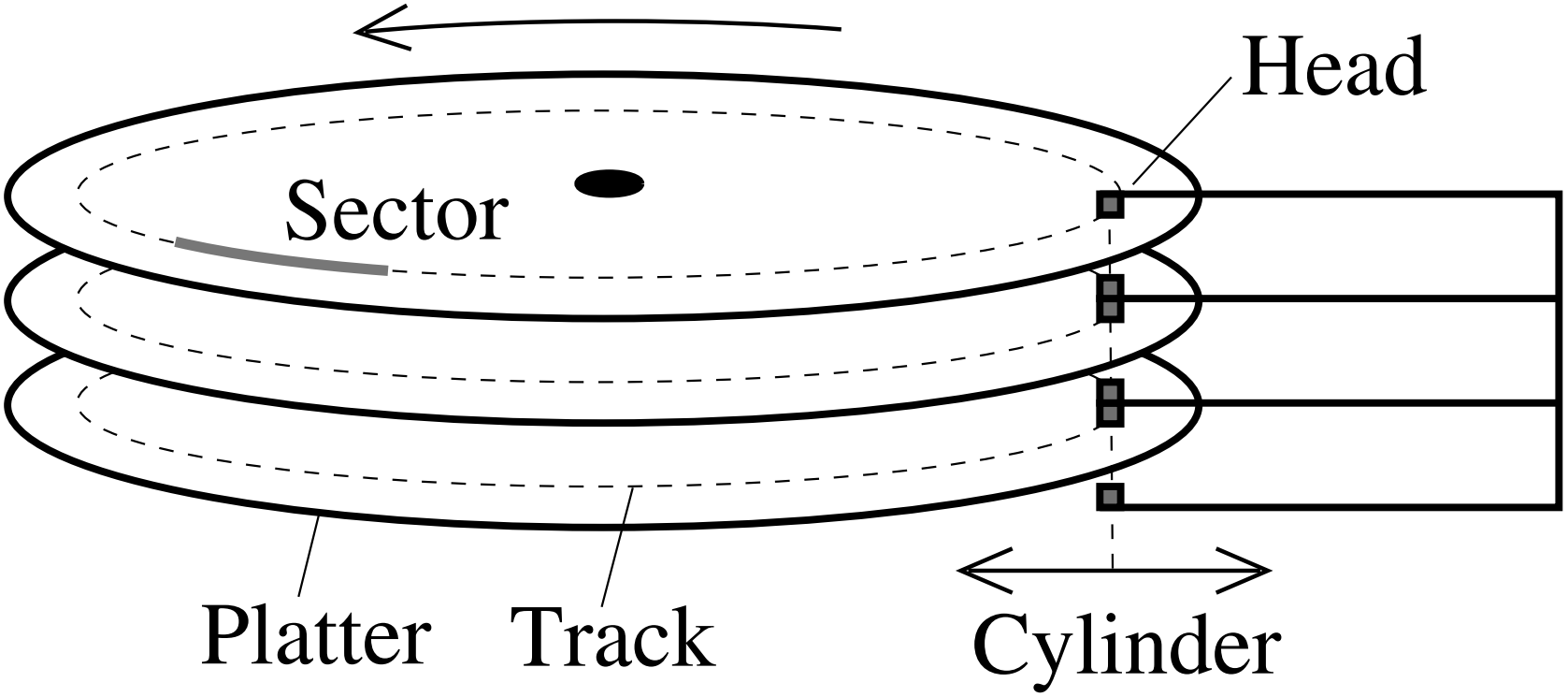
Tomáš Vojnar

vojnar@fit.vutbr.cz

Vysoké učení technické v Brně
Fakulta informačních technologií
Božetěchova 2, 612 66 Brno

Správa souborů

Pevný disk



- Diskový sektor: nejmenší jednotka, kterou disk umožňuje načíst/zapsat.
- Velikost sektoru: dříve 512B, 2048B CD/DVD/BD, nyní 4096B (příp. emulace 512B).
- Adresace sektorů:
  - CHS = Cylinder, Head (typicky 1-6 hlav), Sector
  - LBA = Linear Block Address (číslo 0..N)

❖ Pro připojení disků se používá řada různých diskových (či obecněji periferních) rozhraní: primárně ATA/PATA/SATA či SCSI/SAS, ale také USB, FireWire, FibreChannel, Thunderbolt, PCI Express, NVMe aj. Nad nimi může být další HW rozhraní jako AHCI, OHCI, UHCI, EHCI, xHCI, které je připojí k vyšším sběrnicím.

❖ Diskové sběrnice se liší mj. rychlostí (např. do 6 Gbit/s u SATA, 22.5 Gbit/s SAS), počtem připojitelných zařízení (desítky SATA/65535 SAS), max. délkou kabelů (1-2m SATA, 10m SAS), architekturou připojení (např. více cest k zařízení u SAS), podporovanými příkazy (flexibilita při chybách).

❖ Stejným způsobem jako disky mohou být zpřístupněny i jiné typy pamětí: flash disky, SSD, pásky, CD/DVD/BD, ...

❖ Vzniká hierarchie pamětí, ve které stoupá kapacita a klesá rychlost a cena/B:

- primární paměť: RAM (nad ní ještě registry, cache L1-L3)
- sekundární paměť: pevné disky, SSD (mají také své cache)
- terciární paměť: pásky, CD, DVD, BlueRay, ... (externí disk, síťový disk, cloud).

Parametry pevných disků

❖ Přístupová doba = doba vystavení hlav + rotační zpoždění.

❖ Typické parametry současných disků (orientačně – neustále se mění):

kapacita	do 16 TB
průměrná doba přístupu	od nízkých jednotek ms
otáčky	4200-15000/min
přenosová rychlost	desítky až nízké stovky MB/s

❖ U kapacity disku udávané výrobcem/prodejcem je třeba dávat pozor, jakým způsobem ji počítá: GB = 10^9B nebo $1000 * 2^{20}$B nebo ... Správně: GiB = $1024^3 = 2^{30}$B.

❖ U přenosových rychlostí pozor na sustained transfer rate (opravdové čtení z ploten) a maximum transfer rate (z bufferu disku).

❖ Možnost měření přenosových rychlostí: `hdparm -t`.

- `hdparm` umožňuje číst/měnit celou řadu dalších parametrů disků.
- Pozor! `hdparm -T` měří rychlost přenosu z vyrovnávací paměti OS (tedy z RAM).

Solid State Drive – SSD

❖ SSD je nejčastěji založeno na nevolatilních pamětech NAND flash, ale vyskytují se i řešení založená na DRAM (se zálohovaným napájením) či na kombinacích.

❖ Výhody SSD:

- rychlý (v zásadě okamžitý) náběh,
- náhodný přístup – přístupová doba od jednotek μs (DRAM) do desítek či nízkých stovek μs,
- větší přenosové rychlosti – stovky MB/s (do cca 600 MB/s, 7 GB/s s M.2), zápis může být (mírně) pomalejší (viz dále).
- tichý provoz, mechanická a magnetická odolnost, ...,
- obvykle nižší spotřeba (neplatí pro DRAM).

❖ Nevýhody SSD:

- vyšší cena za jednotku prostoru (dříve nižší kapacita, dnes teoreticky až 100 TB),
- omezený počet přepisů (nevýznamné pro běžný provoz),
- větší riziko katastrofického selhání, menší výdrž mimo provoz,
- možné komplikace se zabezpečením (např. bezpečné mazání/šifrování přepisem dat vyžaduje speciální podporu – disk může data sám přesouvat přes několik míst).

Problematika zápisu u SSD

❖ NAND flash SSD jsou organizovány do stránek (typicky 4KiB) a ty do bloků (typicky 128 stránek, tj. 512KiB).

❖ Prázdné stránky lze zapisovat jednotlivě. Pro přepis nutno načíst celý blok do vyrovnávací paměti, v ní změnit, na disku vymazat a pak zpětně zapsat.

- Problém je menší při sekvenčním než při náhodném zápisu do souboru.

❖ Řešení problémů s přepisem v SSD:

- Aby se problém minimalizoval, SSD může mít více stránek, než je oficiální kapacita.
- Příkaz TRIM umožňuje souborovému systému sdělit SSD, které stránky nejsou používány a následně vymazat bloky tvořené takovými stránkami.
- Řadič SSD může také sám interně přesouvat stránky, aby mohl některé bloky uvolnit.
- Přesto jistý rozdíl v rychlosti čtení/zápisu může zůstat.
- TRIM navíc nelze užít vždy (souborové systémy uložené jako obrazy, kde nemusí být možné uvolňovat bloky, které nejsou na samém konci obrazu; podobně u RAID či databází ukládajících si data do velkého předalokovaného souboru).

❖ Aby řadič SSD minimalizoval počet přepisů stránek, může přepisovanou stránku zapsat na jinou pozici; případně i přesouvá dlouho neměněné stránky.

Zabezpečení disků

❖ Disková elektronika používá ECC = Error Correction Code: k užitečným datům sektoru si ukládá redundantní data, která umožňují opravu (a z hlediska OS transparentní realokaci), nebo alespoň detekci chyb.

❖ S.M.A.R.T. – Self Monitoring Analysis and Reporting Technology: disky si automaticky shromažďují řadu statistik, které lze použít k předpovídání/diagnostice chyb. Viz `smartctl`, `smartd`, ...

❖ Rozpoznávání a označování vadných bloků (bad blocks) může probíhat také na úrovni OS (např. `e2fsck` a `badblocks`), pokud si již s chybami disk sám neporadí (což je ale možná také vhodná doba disk raději vyměnit).

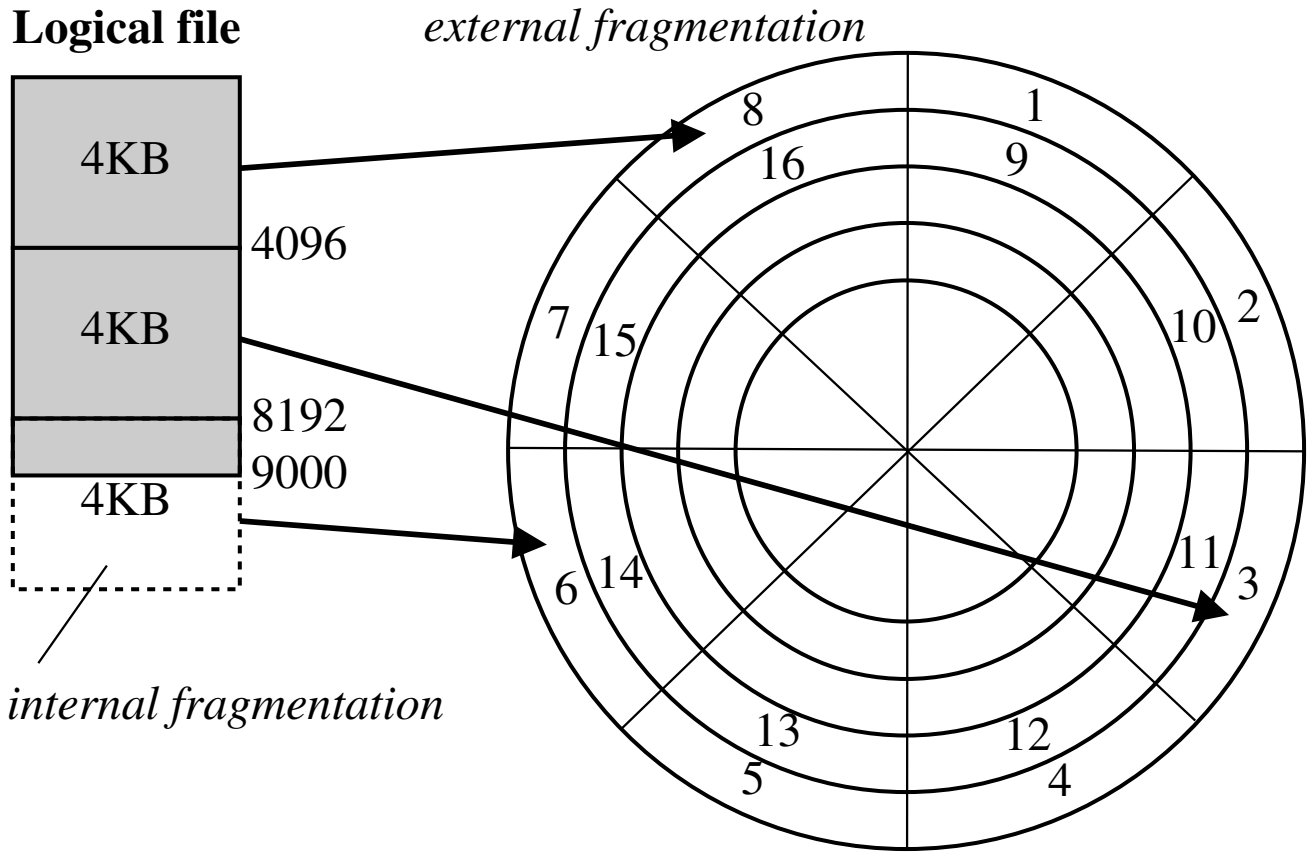
Disková pole

❖ RAID (Redundant Array of Independent Disks):

- RAID 0 – *disk striping*, následné bloky dat rozmístěny na různých discích, vyšší výkonnost, žádná redundance.
- RAID 1 – *disk mirroring*, všechna data ukládána na dva disky, velká redundance (existuje také RAID 0+1 a RAID 1+0/10).
- RAID 2 – data rozdělena mezi disky po bitech, použito zabezpečení Hammingovým kódem uloženým na zvláštních discích (např. 3 bity zabezpečení pro 4 datové: chybu na 1 disku lze automaticky opravit, na 2 discích detekovat).
- RAID 3 – data uložena po bajtech (nebo i bitech) na různých discích, navíc je užit disk s paritami.
- RAID 4 – bloky (sektory či jejich násobky) dat na různých discích a paritní bloky na zvláštním disku.
- RAID 5 – jako RAID 4, ale paritní a datové bloky jsou rozloženy na všech discích, redukce kolizí u paritního disku při zápisu.
- RAID 6 – jako RAID 5, ale parita uložena 2x, vyrovná se i se ztrátou 2 disků.

Uložení souboru na disku

❖ Alokační blok: skupina pevného počtu sektorů, typicky 2^n pro nějaké n, následujících logicky (tj. v souboru) i fyzicky (tj. na disku) za sebou, která je nejmenší jednotkou diskového prostoru, kterou OS čte či zapisuje při běžných operacích.



❖ Poznámka: Někdy se též užívá označení cluster.

Fragmentace

❖ Při přidělování a uvolňování prostoru pro soubory dochází k tzv. externí fragmentaci – na disku vzniká posloupnost volných oblastí a oblastí použitých různými soubory (může tedy existovat i na plně obsazeném disku), což má dva možné důsledky:

1. Vzniknou nevyužité oblasti diskového prostoru, které se nedají využít (a) v daném okamžiku nebo (b) vůbec.

   - Případ (a): pokud se použije *spojité přidělování* prostoru pro soubory, může na disku být v sumě dost místa pro uložení daného souboru, ale žádná volná oblast sama o sobě nepostačuje a soubor tedy nelze uložit.
   - Případ (b): pokud *není prostor přidělován v násobcích dostatečně velké jednotky prostoru* a současně je zapotřebí do každého alokovaného prostoru uložit kromě užitečného obsahu ještě metadata, nemusí být prostor dostatečný ani na uložení metadat (a pak je zcela nevyužitelný).

2. Při nespojitém přidělování prostoru po dostatečně velkých alokačních blocích výše uvedený problém nevzniká, ale data souboru jsou na disku uložena nespojitě – složitější a pomalejší přístup (menší vliv u SSD, ale i tam se může projevit).

Fragmentace

❖ Souborové systémy užívají různé techniky k minimalizaci externí fragmentace:

- rozložení souborů po disku: tedy neukládají soubory bezprostředně za sebe, je-li to možné;
- předalokace: alokuje se více místa, než je momentálně zapotřebí (např. `fallocate` na Linuxu);
- odložená alokace: odkládá zápis, než se nasbírá více požadavků a je lepší povědomí, kolik je třeba alokovat (např. "allocate-on-flush" v různých soborových systémech).

❖ Přesto bývají k dispozici nástroje pro defragmentaci.

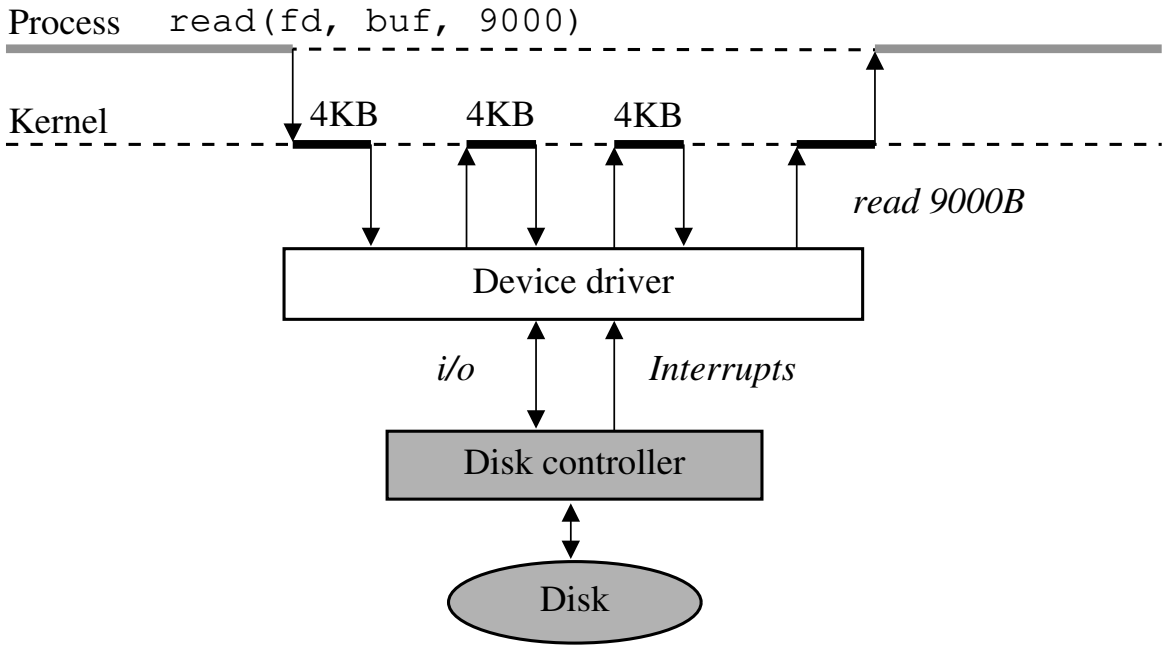
❖ Interní fragmentace – nevyužité místo v posledním přiděleném alokačním bloku – plýtvání místem.

- Některé souborové systémy (např. Btrfs) umožňují sdílení posledních alokačních bloků více soubory.

Přístup na disk

❖ Prostřednictvím I/O portů a/nebo paměťově mapovaných I/O operací (HW zajišťuje, že některé adresy RAM ve skutečnosti odkazují do interní paměti I/O zařízení) se řadiči disku předávají přes příslušné sběrnice a HW rozhraní příkazy definované diskovým rozhraním (ATA, SCSI, ...).

❖ Přenos z/na disk je typicky řízen řadičem disku s využitím technologie přímého přístupu do paměti (DMA). O ukončení operací či chybách informuje řadič procesor (a na něm běžící jádro OS) pomocí přerušení.



Plánování přístupu na disk

- Pořadí bloků čtených/zapisovaných na disk ovlivňuje plánovač diskových operací.
  - Přicházející požadavky na čtení/zápis jsou ukládány do vyrovnávací paměti a jejich pořadí je případně měněno tak, aby se minimalizovala režie diskových operací.
  - Např. tzv. výtahový algoritmus (elevator algorithm, SCAN algorithm) pohybuje hlavičkami od středu k okraji ploten a zpět a vyřizuje požadavky v pořadí odpovídajících pozici a směru pohybu hlaviček.
  - Další plánovací algoritmy: Circular SCAN (vyřizuje požadavky vždy při pohybu jedním směrem – rovnoměrnější doba obsluhy), LOOK (pohybuje se jen v mezích daných aktuálními požadavky – nižší průměrná doba přístupu), C-LOOK, ...
  - Plánovač může sdružovat operace, vyvažovat požadavky různých uživatelů, implementovat priority operací, odkládat operace v naději, že je bude možno později propojit, implementovat časová omezení možného čekání operací na provedení apod.
  - Možnost více vstupních front z různých CPU i více výstupních front pro paralelní zpracování zařízením (Linux: `blk-mq`).

- V Linuxu možno zjistit/změnit nastavení prostřednictvím `/sys/block/<devicename>/queue/scheduler`.

Logický disk

- Dělení fyzického disku na logické disky – diskové oblasti (partitions):
  - Starší systémy PC: MBR (Master Boot Record) – tabulka diskových oblastí s 1-4 primárními diskovými oblastmi, jedna z nich může být nahrazena rozšířenou diskovou oblastí, obsahující zřetězený seznam logických diskových oblastí, každou z nich popsanou EBR (Extended Boot Record).
  - Novější PC: GUID Partition Table (GPT) – vynechaný prostor pro MBR, hlavička GPT a následně pole 128 odkazů na logické diskové oblasti. Zabezpečeno kontrolními součty, s rezervní kopií GPT. GUID = Globally Unique Identifier fyzických/logických disků, založeno např. na kryptografii.
  - Pro správu diskových oblastí lze užít programy `cfdisk`, `fdisk`, `gparted`, ...
  - LVM = Logical Volume Manager: umožňuje tvorbu logických disků přesahujících hranice fyzického disku, snadnou změnu velikosti, přidávání a ubírání disků, tvorbu snímků, ... (někdy přímo v souborovém systému: ZFS).

- Formátování – program `mkfs`; existuje (existovalo) také nízkoúrovňové formátování.

- Kontrola konzistence souborového systému: program `fsck`.

❖ Různé typy souborových systémů: fs, ufs, ufs2, ext2, ext3, ext4, Btrfs, HFS+/APFS (Mac OS X), XFS (od Silicon Graphics, původně pro IRIX), JFS (od IBM, původně pro AIX), ZFS (od Sunu, původně pro OpenSolaris), HPFS, FAT, VFAT, FAT32, ExFAT, NTFS, ReFS, F2FS, ISO9660 (Rock Ridge, Joliet), UDF, Lustre (Linuxové clustry a superpočítače), GPFS (clustry a superpočítače), zonefs (disk rozdělen na sekvenčně zapisované či kompletně mazané zóny)...

❖ Virtuální souborový systém (VFS) – vrstva, která zastřešuje všechny použité souborové systémy a umožňuje pracovat s nimi jednotným, abstraktním způsobem.

❖ Síťové souborové systémy: NFS, ...

❖ Speciální souborové systémy: procfs, sysfs (souborové systémy informující o dění v systému a umožňující nastavení jeho parametrů), tmpfs (souborový systém alokující prostor v RAM a sloužící pro ukládání dočasných dat), ...

Žurnálování

- Žurnál slouží pro záznam modifikovaných metadat (příp. i dat) před jejich zápisem na disk.
  - Obvykle implementován jako cyklicky přepisovaný buffer ve speciální oblasti disku.
  - Operace pokryté žurnálováním jsou atomické – vytváří transakce: buď uspějí všechny jejich dílčí kroky nebo žádný.
    - Např. mazání souboru v UNIXU znamená odstranění záznamu z adresáře, pak uvolnění místa na disku.

- Systémy souborů se žurnálem: ext3, ext4, ufs, XFS, JFS, NTFS, ...

- Umožňuje spolehlivější a rychlejší návrat do konzistentního stavu po chybách.

- Data obvykle nejsou žurnálována (byť mohou být): velká režie.

- Kompromis mezi žurnálováním a nežurnálováním dat představuje předřazení zápisu dat na disk před zápis metadat do žurnálu: zajistí konzistenci při chybě během zápisu dat na konec souboru (částečně zapsaná nová data nebudou uvažována).

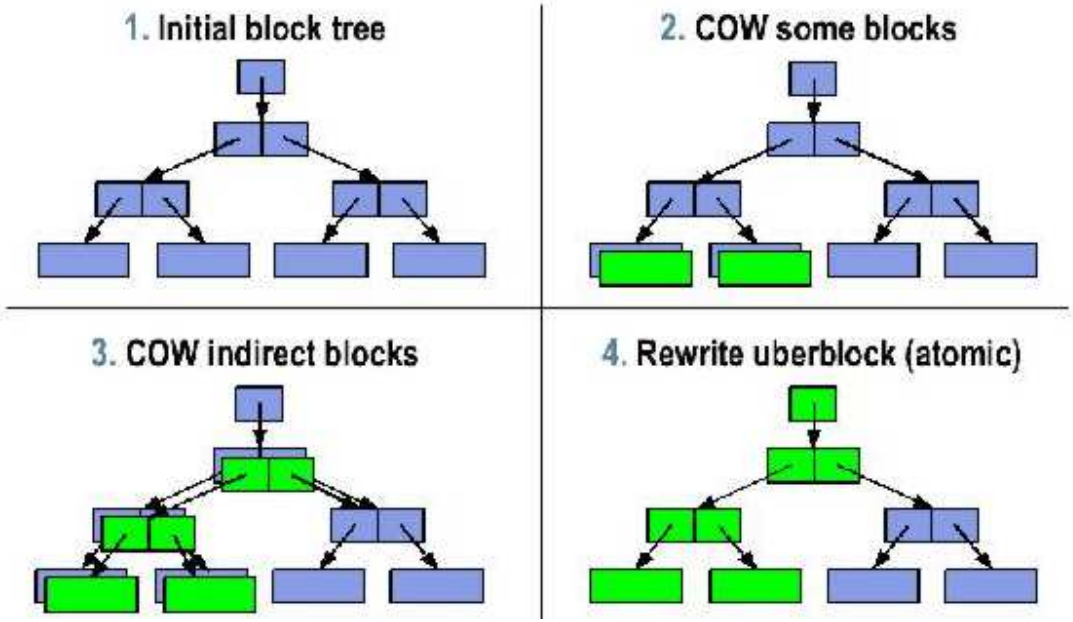
Implementace žurnálování

- Implementace na základě dokončení transakcí (REDO), např. ext3/4:
  - sekvence dílčích operací se uloží nejprve do žurnálu mezi značky označující začátek a konec transakce (příp. spolu s kontrolním součtem),
  - poté se dílčí operace provádí na disku,
  - uspějí-li všechny dílčí operace, transakce se ze žurnálu uvolní,
  - při selhání se dokončí všechny transakce, které jsou v žurnálu zapsány celé (a s korektním kontrolním součtem).
- Implementace na základě anulace transakcí (UNDO):
  - záznam dílčích operací do žurnálu a na disk se prokládá,
  - proběhne-li celá transakce, ze žurnálu se uvolní,
  - při chybě se eliminují nedokončené transakce.
- UNDO a REDO je možno kombinovat (NTFS).
- Implementace žurnálování musí zajišťovat správné pořadí zápisu operací, které ovlivňuje plánování diskových operací v OS a také případně jejich přeuspořádání v samotném disku.

Alternativy k žurnálování

❖ Copy-on-write (např. ZFS, Btrfs, ReFS) – nejprve zapisuje nová data či metadata na disk, pak je zpřístupní:

- Změny provádí hierarchicky v souladu s hierarchickým popisem obsahu disku:
  – vyhledávací strom popisující uložení dat a metadat na disku, ne adresářový strom;
  – data vyhledává na základě unikátní identifikace souborů a posuvu v nich.
- Začne měněným uzlem, vytvoří jeho kopii a upraví ji.
  Potom vytvoří kopii uzlu nadřazeného změněnému uzlu, upraví ji tak, aby odkazovala příslušným odkazem na uzel vytvořený v předchozím kroku atd.
- Na nejvyšší úrovni se udržuje několik verzí kořenového záznamu se zabezpečovacím kódem a časovými razítky.
  Po chybě bere kořen s nejnovějším razítkem a správným kontrolním součtem.



Alternativy k žurnálování

❖ Poznámka: CoW nabízí rovněž bázi pro implementaci:

- snímků souborového systému: uložení stavu v určitém okamžiku s možností pozdějšího návratu – stačí zálohovat si starší verze kořene a z něj dostupné záznamy;
- klonů souborového systému: vytvoření kopií, které jsou v budoucnu samostatně manipulovány – vzniká několik kopií kořene, které se dále mění samostatně.

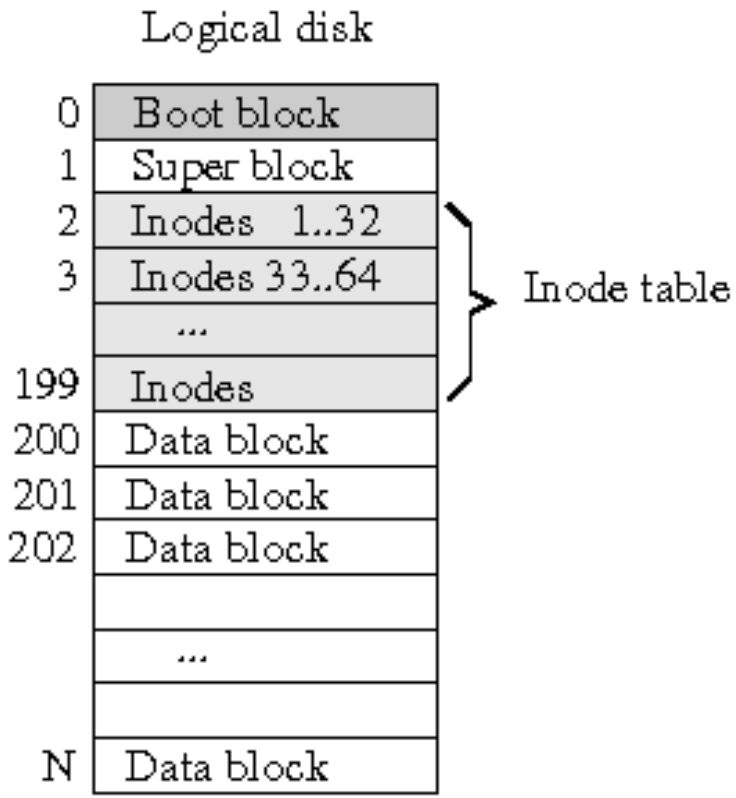
V obou případech vznikají částečně sdílené stromové struktury popisujících různé verze obsahu souborového systému.

❖ Další možnosti:

- Soft updates (např. UFS): sleduje závislosti mezi změněnými metadaty a daty a zaručuje zápis na disk v takovém pořadí, aby v kterékoli době byl obsah disku konzistentní (až na možnost vzniku volného místa považovaného za obsazené).
- Log-structured file systems (LFS, UDF, F2FS): celý souborový systém má charakter logu (zapsaného v cyklicky přepisované frontě) s obsahem disku vždy přístupným přes poslední záznam (a odkazy z něj).

Klasický UNIXový systém souborů (FS)

boot blok	pro zavedení systému při startu
super blok	informace o souborovém systému (typ, velikost, počet i-uzlů, volné místo, volné i-uzly, kořenový adresář, UUID, ...)
tabulka i-uzlů	tabulka s popisy souborů
datové bloky	data souborů a bloky pro nepřímé odkazy



❖ Modifikace základního rozložení FS v navazujících souborových systémech:

- Disk rozdělen do skupin bloků.
- Každá skupina má své i-uzly a datové bloky a také svůj popis volných bloků: lepší lokalita.
- Superblok se základními informacemi o souborovém systému je rovněž uložen vícenásobně.

i-uzel

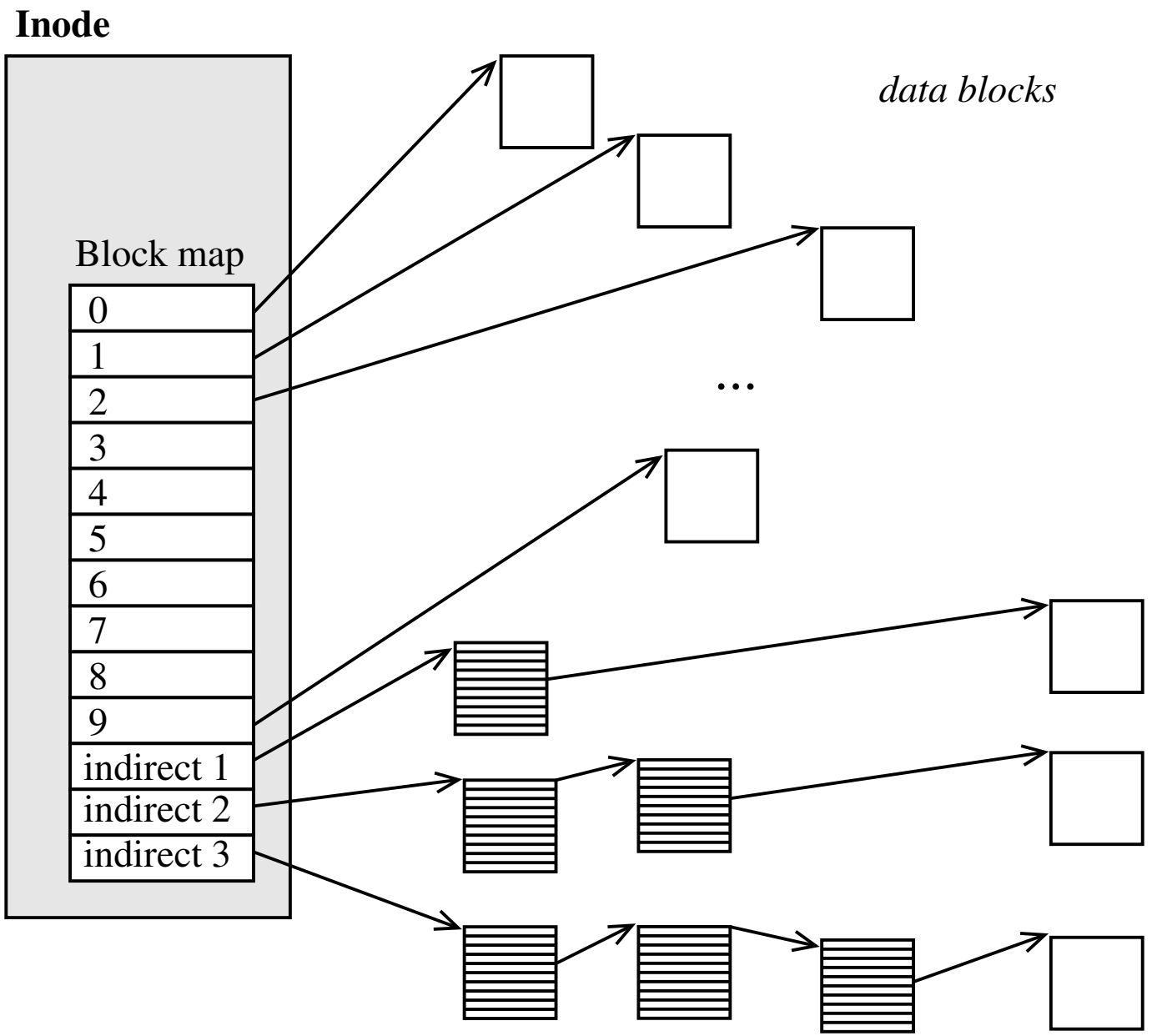
❖ Základní datová struktura popisující soubor v UNIXu.

❖ i-uzel obsahuje metadata:

- stav i-uzlu (alokovaný, volný)
- typ souboru (obyčejný, adresář, zařízení, ...)
- délka souboru v *bajtech*
- mtime = čas poslední modifikace dat
- atime = čas posledního přístupu
- ctime = čas poslední modifikace i-uzlu
- UID = identifikace vlastníka (číslo)
- GID = identifikace skupiny (číslo)
- přístupová práva (číslo, například 0644 znamená rw-r–r–)
- počet pevných odkazů (jmen)
- tabulka odkazů na datové bloky:

10	přímých odkazů (12 u ext2, ...)
1	nepřímý odkaz první úrovně
1	nepřímý odkaz druhé úrovně
1	nepřímý odkaz třetí úrovně

- (odkaz na) další informace (ACL, extended attributes, dtime, ...)


Inode
Block map
0
1
2
3
4
5
6
7
8
9
indirect 1
indirect 2
indirect 3
data blocks
...

❖ Teoretický limit velikosti souboru:

$$10 * D + N * D + N^2 * D + N^3 * D,$$

kde:

- $N = D/M$ je počet odkazů v bloku, je-li M velikost odkazu v bajtech (běžně 4B),
- D je velikost bloku v bajtech (běžně 4096B).

❖ Velikost souborů je omezena také dalšími strukturami FS, VFS, rozhraním jádra a architekturou systému (32b/64b) – viz Large File System support: podpora souborů $> 2GiB$.

❖ Co vypisují programy o velikosti souborů?

- `du soubor` – zabrané místo v blocích, včetně režie,
- `ls -l soubor` – velikost souboru v bajtech,
- `df` – volné místo na namontovaných discích.

❖ Zpřístupnění i-uzlu:

- `ls -i soubor` – číslo i-uzlu souboru soubor,
- `ils -e /dev/... n` – výpis i-uzlu n na `/dev/...`

❖ Základní informace o souborovém systému ext2/3/4: `dumpe2fs`.

❖ Architektura souborových systémů je ovlivňována snahou o minimalizaci jejich režie při průchodu, přesunu v souboru (seek), zvětšování/zmenšování souboru:

- snadnost *vyhledání adresy prvního/určitého bloku* souboru,
- snadnost *vyhledání následujících bloků*,
- snadnost *přidání/ubrání dalších bloků*,
- snadnost *alokace/dealokace volného prostoru*
  (informace o volných oblastech, minimalizace externí fragmentace).

❖ FS (a řada jeho následníků UFS, ext2, ext3) představuje kompromis s ohledem na převážně malé soubory.

- U větších souborů nutno procházet/modifikovat větší objem metadat.

❖ Další optimalizace pro malé soubory: data přímo v i-uzlu (např. u symbolických odkazů definovaných dostatečně krátkou cestou, tzv. fast symlinks).

Jiné způsoby organizace souborů

❖ Kontinuální uložení: jedna spojitá posloupnost na disku.

- Problémy se zvětšováním souborů díky externí fragmentaci nebo obsazení prostoru hned za koncem souboru.

❖ Zřetězené seznamy bloků: každý datový blok obsahuje kromě dat odkaz na další blok (nebo příznak konce souboru).

- Při přístupu k náhodným blokům či ke konci souboru (změna velikosti) nutno projít celý soubor.
- Chyba kdekoliv na disku může způsobit ztrátu velkého objemu dat (rozpojení seznamu).

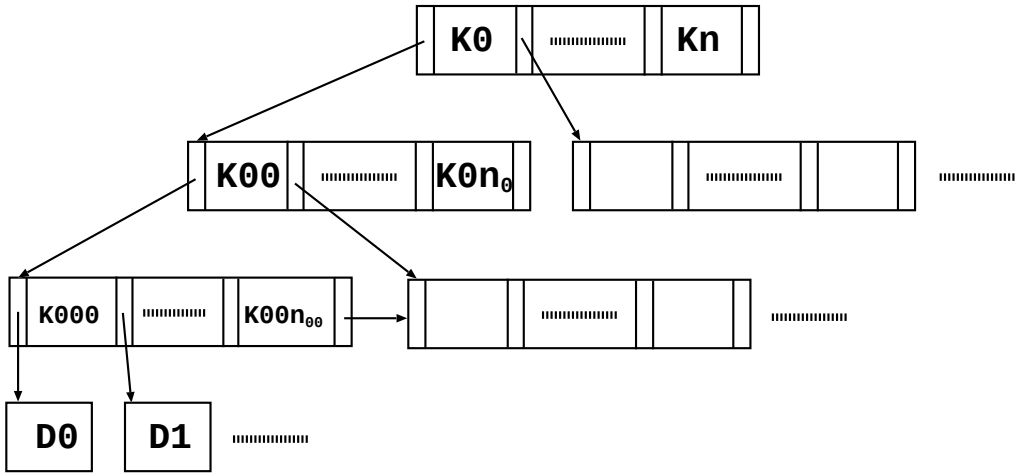
❖ FAT (File Allocation Table): seznamy uložené ve speciální oblasti disku. Na začátku disku je (pro vyšší spolehlivost zdvojená) tabulka FAT, která má položku pro každý blok. Do této tabulky vedou odkazy z adresářů. Položky tabulky mohou být zřetězeny do seznamů, příp. označeny jako volné či chybné.

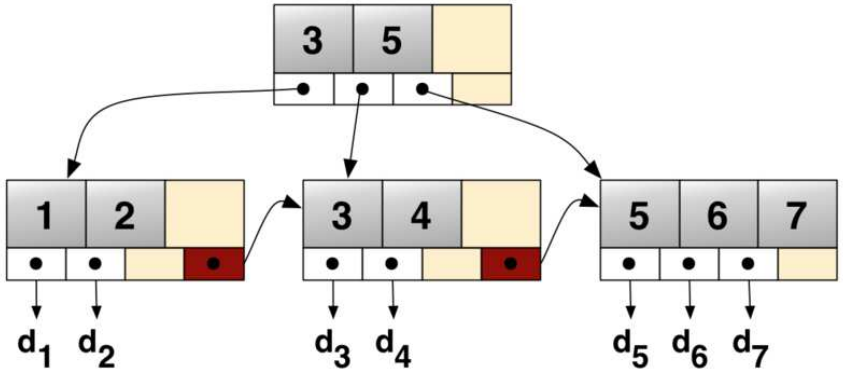
- Opět vznikají problémy s náhodným přístupem.

Jiné způsoby organizace souborů

❖ B+ stromy:

- Vnitřní uzly obsahují sekvenci $link_0, key_0, link_1, key_1, ..., link_n, key_n, link_{n+1}$, kde $key_i < key_{i+1}$ pro $0 \leq i < n$. Hledáme-li záznam s klíčem k, pokračujeme $link_0$, je-li $k < key_0$; jinak $link_i$, $1 \leq i \leq n$, je-li $key_{i-1} \leq k < key_i$; jinak užijeme key_{n+1}.
- Listy mají podobnou strukturu. Je-li $key_i = k$ pro nějaké $0 \leq i \leq n$, $link_i$ odkazuje na hledaný záznam. Jinak hledaný záznam neexistuje.
- Poslední odkaz $link_{n+1}$ v listech je užit k odkazu na následující listový uzel pro urychlení lineárního průchodu indexovanými daty.





Jiné způsoby organizace souborů

❖ B+ stromy:

- Strom zůstává výškově vyvážený.
- Limity zaplnění pro uzly s m odkazy (tedy klíči key_0 až key_{m-2}): sólo kořen 1 až $m-1$, kořen 2 až m, vnitřní uzel $\lceil m/2 \rceil$ až m, list $\lceil m/2 \rceil - 1$ až $m-1$.
- Vkládá se na listové úrovni. Dojde-li k přeplnění, list se rozštěpí a přidá se nový odkaz do nadřazeného vnitřního uzlu. Při přeplnění se pokračuje směrem ke kořeni. Nakonec může být přidán nový kořen.
- Ruší se od listové úrovně. Při nenaplnění minimální kapacity, pokus o přerozdělení mezi potomky daného předka. Nestačí-li, uzly se spojí a ruší se jeden odkaz na nadřazené úrovni. Nutno upravit klíče. Rušení může pokračovat směrem ke kořeni. Nakonec může jedna úroveň ubýt.

❖ B+ stromy a jejich různé varianty jsou použity pro popis diskového prostoru přiděleného souborům v různých souborových systémech:

- XFS, JFS, ZFS, Btrfs, APFS, ReFS, ...,
- v omezené podobě tzv. stromů extentů v ext4 (pouze pět úrovní, bez vyvažování, bez zřetězení listů), podobná struktura je i v NTFS.

Jiné způsoby organizace souborů

❖ Alokovaný prostor se často indexuje po tzv. extentech, tj. posloupnostech proměnného počtu bloků jdoucích za sebou logicky v souboru a uložených i fyzicky na disku za sebou:

- zrychluje se práce s velkými soubory: menší, lépe vyvážené indexové struktury; menší objem metadat, které je třeba procházet a udržovat; lepší lokalita dat i metadat.

❖ Extenty jsou použity ve všech výše zmíněných systémech s B+ stromy a jejich variantami.

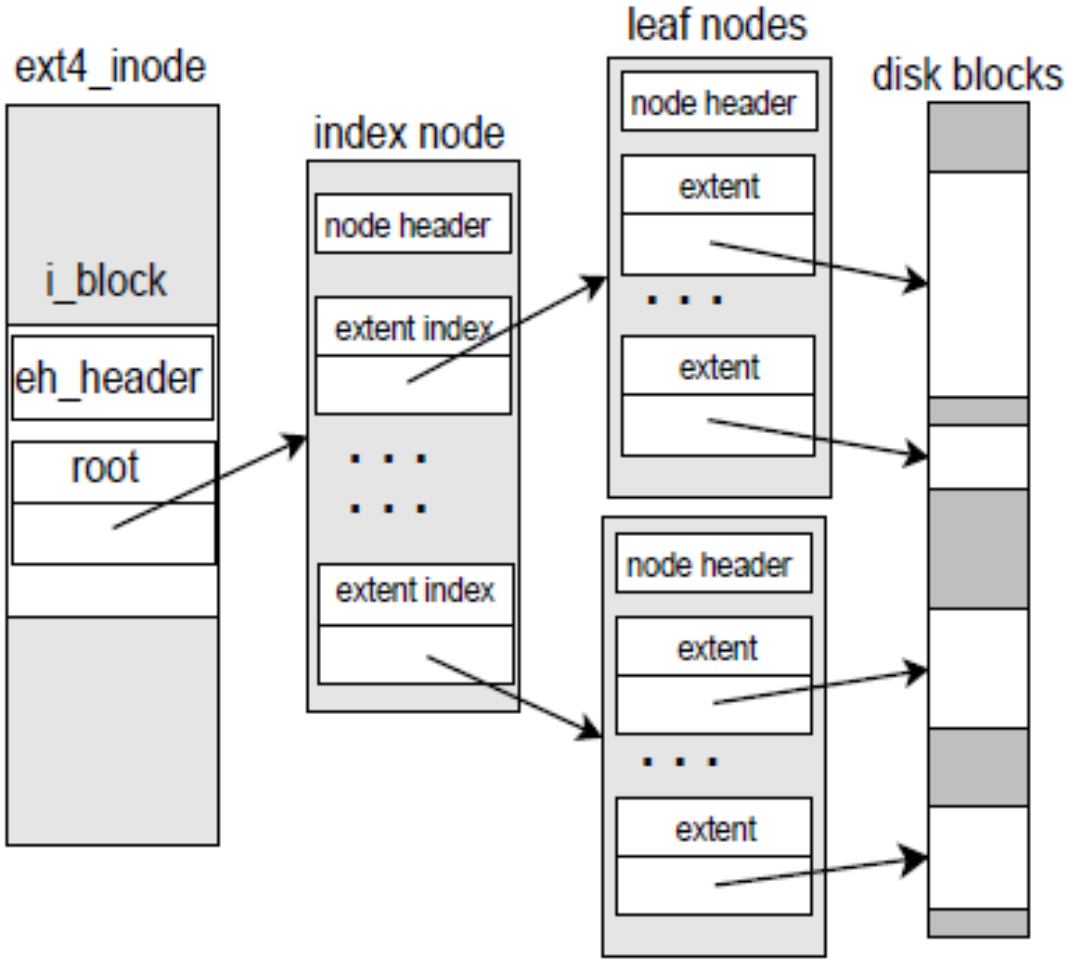
- B+ stromy se snadno kombinují s extenty. To neplatí pro klasický Unixový strom, který není kompatibilní s adresováním jednotek proměnné velikosti.
  - Pro bloky proměnné velikosti není kam ve vyhledávací struktuře uložit jejich velikost.

❖ Spojitému průchodu může pomoci prolinkování listů vyhledávacích stromů, je-li použito.

❖ Pro malé soubory může B+ strom představovat zbytečnou režii:
přímé uložení v i-uzlu nebo přímé odkazy na extenty z i-uzlu (do určitého počtu).

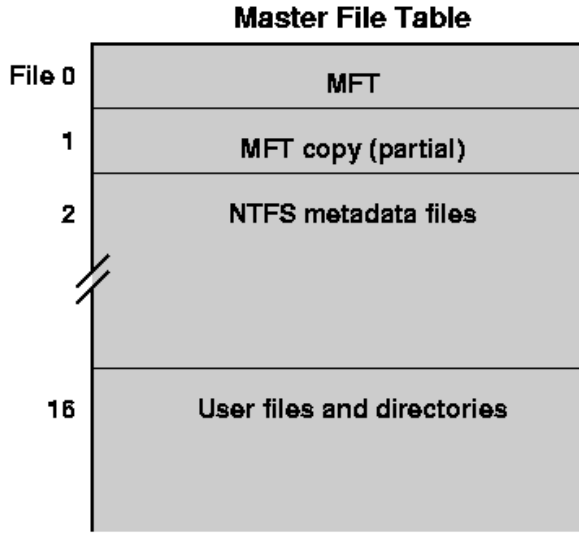
Ext4

❖ Strom extentů – v principu B+ strom degradovaný na max. 5 úrovní bez vyvažování a bez zřetězení listů:



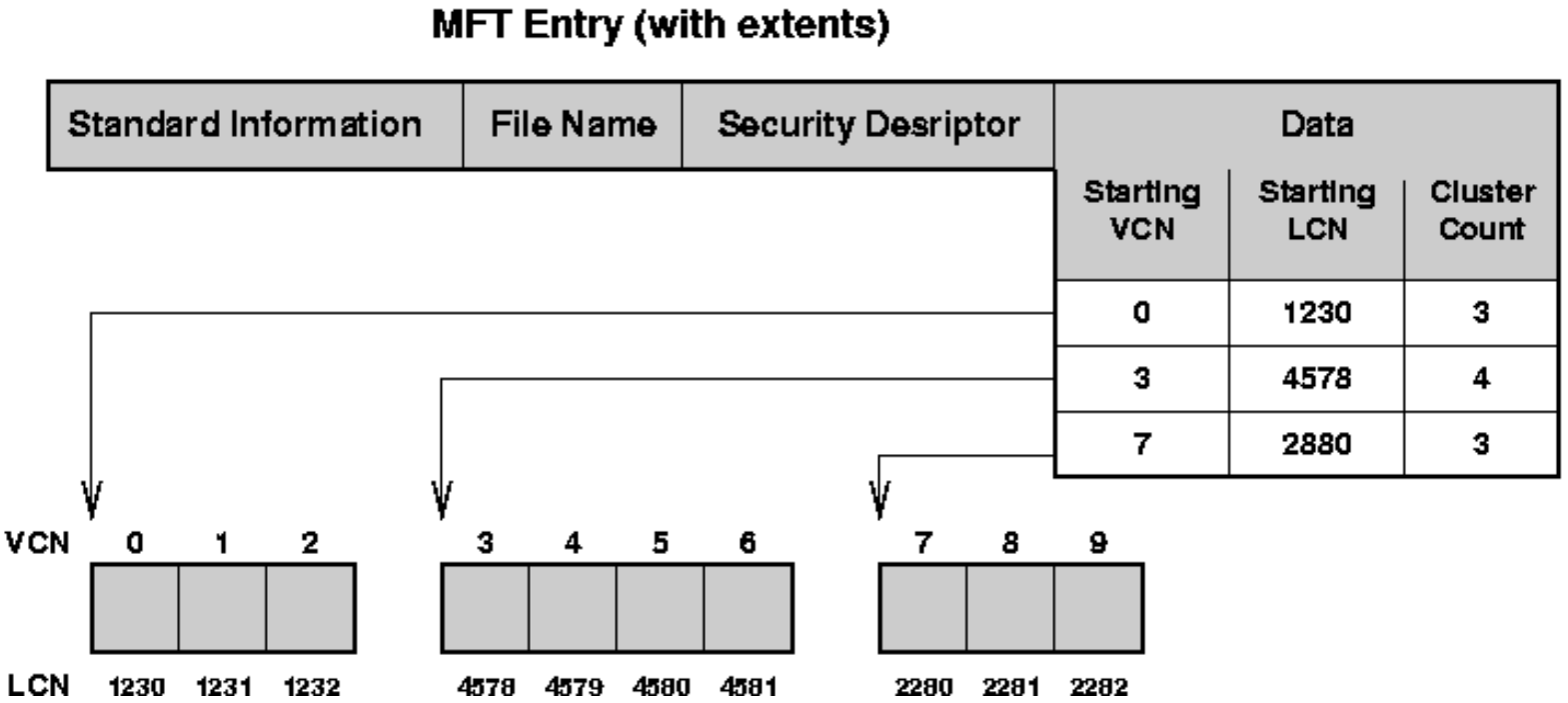
❖ Malé soubory: až 4 extenty odkazované přímo z kořenového uzlu extentového stromu umístěného v i-uzlu, příp. přímo v i-uzlu (symbolické odkazy).

NTFS



❖ MFT – Master File Table: alespoň jeden řádek pro každý soubor.

❖ Obsah souboru (a) přímo v záznamu MFT odpovídajícím příslušnému souboru, (b) nebo rozdělen na extenty odkazované z tohoto záznamu, (c) nebo z pomocných MFT záznamů odkazovaných z primárního MFT záznamu ve stylu B+ stromu.



Organizace volného prostoru

❖ Organizace volného prostoru v klasickém Unixovém FS a řadě jeho následovníků (UFS, ext2, ext3) a také v NTFS: bitová mapa s jedním bitem pro každý blok.

- Umožňuje zrychlit vyhledávání volné souvislé oblasti pomocí bitového maskování (test volnosti několika bloků současně).

❖ Další způsoby organizace volného prostoru zahrnují:

- seznam – zřetězení volných bloků,
- označení (zřetězení) volných položek v tabulce bloků (FAT),
- B+ strom – adresace velikostí a/nebo offsetem.

❖ Volný prostor může být také organizován po extentech.

Deduplikace

- Snaha odhalit opakované ukládání těchže dat, uložit je jednou a odkázat vícenásobně.
- Může být podporována na různých úrovních: sekvence bytů, bloky, extenty, soubory.
- Založeno na kryptografickém hashování, případně s následnou kontrolou plné shody.
- Může být implementováno při zápisu, nebo dodatečně (případně na přání).
- Může uspořit diskový prostor při virtualizaci, na mail serverech, repozitářích apod., paměťový prostor (sdílení stránek, vyrovnávacích pamětí) i čas (není nutno opakovaně číst/zapisovat).
- Při menším objemu duplikace může naopak zvýšit spotřebu procesorového času, paměťového i diskového prostoru.
- Podpora (někdy ve vývoji/externí): ZFS, NTFS, Btrfs, XFS, ...

Typy souborů v UNIXu

- Příkaz `ls -l` vypisuje typ jako první znak na řádku:

`-`	obyčejný soubor
`d`	adresář
`b`	blokový speciální soubor
`c`	znakový speciální soubor
`l`	symbolický odkaz (symlink)
`p`	pojmenovaná roura
`s`	socket

Adresář

- Soubor obsahující množinu dvojic – "hard-links" – `(jméno souboru, číslo souboru)`:
  - `jméno souboru`:
    - mělo v tradičním UNIXu délku max 14 znaků, dnes typicky až 255 znaků,
    - může obsahovat jakékoli znaky kromě `'/'` a `'\0'`,
  - `číslem souboru` je u klasického Unixového FS (a souborových systémů z něj odvozených) číslo i-uzlu, které je indexem do tabulky i-uzlů logického disku (v jiných případech může sloužit jako klíč pro vyhledávání v B+ stromu apod.).

- Adresář vždy obsahuje jména:
  - `.` odkaz na sebe
  - `..` odkaz na rodičovský adresář

- Implementace – jednoduchost implementace vs rychlost vyhledávání/vkládání:
  - seznam,
  - B+ stromy a jejich varianty: NTFS, XFS, JFS, Btrfs, APFS, ext3/4 (H-stromy: 1–2 úrovně, bez vyvažování, vyhledává na základě zahashovaného jména),
  - (rozšiřitelné) hashovací tabulky (extendible hashing) – např. ZFS.

❖ Soubor v Unixu může mít více jmen:

- `ln jmeno-existujiciho-souboru nove-jmeno`

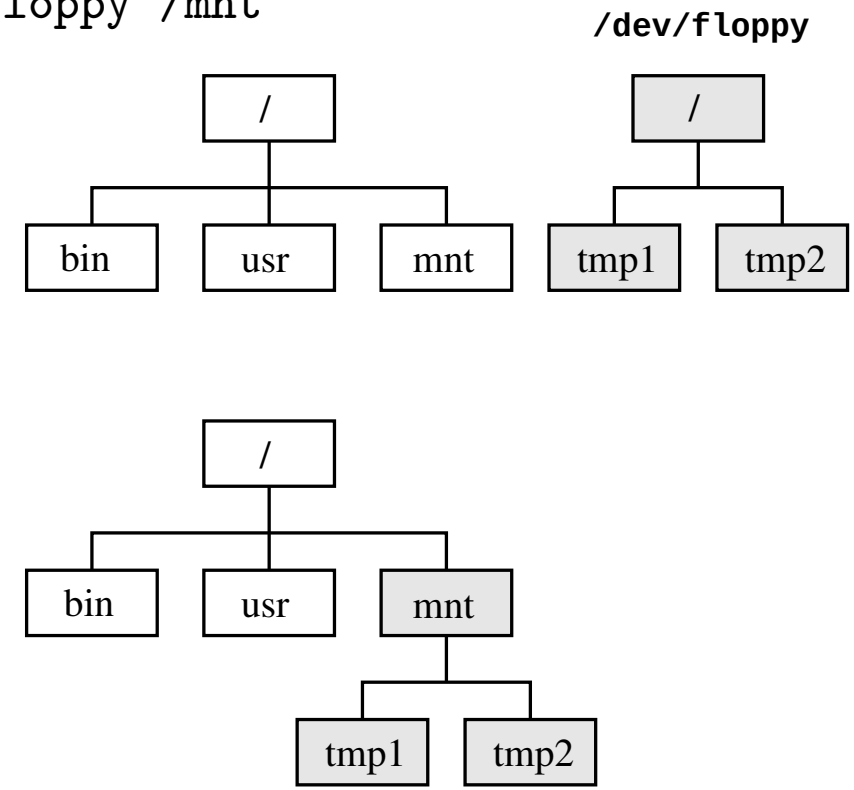
❖ Omezení: Obě jména musí být v rámci jednoho logického disku!

❖ Rušení souboru (`rm soubor`) ruší pevný odkaz (`jméno, číslo i-uzlu`) a snižuje počitadlo odkazů v i-uzlu. Dokud je počitadlo nenulové, soubor se nemaže.

Montování disků

- Princip montování disků:
  - Všechny soubory jsou v jednom “stromu” adresářů.
  - V systému je jeden kořenový logický disk, další logické disky lze připojit programem `mount` do již existujícího adresářového stromu.

- Příklad: `mount /dev/floppy /mnt`



❖ Poznámky:

- Parametry příkazu `mount` – viz `man mount`.
- Soubor `/etc/fstab` – popis disků typicky připojovaných na určité pozice adresářového stromu.
- Soubor `/etc/mtab` – tabulka aktuálně připojených disků.
- Některé technologie umožnují automatické montování nově připojených zařízení. Např. systém udev dynamicky vytváří rozhraní souborového systému na zařízení v adresáři `/dev` a informuje zbytek systému prostřednictvím sběrnice D-Bus, aplikace typu správce souborů pak může provést automatické montování a další akce (parametry může zjišťovat automaticky, čerpat z různých nastavení zúčastněných technologií, ale přednost má stále `/etc/fstab`).
- Automounter – automaticky připojuje potřebné disky při pokusu o přístup na pozici adresářového stromu, kam by měly být připojeny, a také po určité době neaktivity disky odpojuje (výhodné zejména u sítových souborových systémů).
- Union mount:
  - Montuje více disků (adresářů) do jednoho místa – obsah je pak sjednocením obsahu namontovaných adresářů s tím, že se vhodným způsobem řeší kolize (např. prioritou zdrojových disků/adresářů).
  - Plan9, Linux – UnionFS, ...
  - UnionFS: má copy-on write sémantiku: soubor původně v read-only větvi, při změně se uloží do read-write větve s vyšší prioritou.

Symbolické odkazy

```
ln -s existující-soubor symbolický-odkaz
```

- V datech souboru typu “symlink” je jméno cílového souboru.
- Systém při otevření souboru automaticky provede otevření cílového souboru.
  - Nutné vícenásobné zpracování cesty (cesta k symlinku, cesta uvnitř symlinku).
- Po zrušení cílového souboru zůstává symlink nezměněn.
  - Přístup k souboru přes něj vede k chybě.
- Symlink může odkazovat na i jiný logický disk.
- Řešení cyklů: omezený počet úrovní odkazů.
- Rychlé symlinky: uloženy v i-uzlu, pomalé symlinky: uloženy ve zvláštním souboru (užívá se tehdy, je-li cesta, která definuje symlink, příliš dlouhá pro uložení do i-uzlu).

Blokové a znakové speciální soubory

❖ Blokové a znakové speciální soubory implementují souborové rozhraní k fyzickým či virtuálním zařízením.

soubor	tvoří souborové rozhraní na zařízení
/dev/hda	dříve první fyzický disk (master) na prvním ATA/PATA rozhraní
/dev/hda1	dříve první logický disk (partition) na hda
/dev/sda	dříve první fyzický disk SCSI (nyní i emulované SATA/PATA)
/dev/mem	fyzická paměť
/dev/zero	nekonečný zdroj nulových bajtů
/dev/null	soubor typu "černá díra"– co se zapíše, to se zahodí; při čtení se chová jako prázdný soubor
/dev/random	generátor náhodných čísel
/dev/tty	terminál
/dev/lp0	první tiskárna
/dev/mouse	myš
/dev/dsp	zvuková karta
/dev/loop	souborové systémy nad soubory (`losetup`/`mount -o loop=...`)
.....	.....

Poznámka: Názvy závisí na použitém systému (Linux).

❖ Výhoda zavedení speciálních souborů: Programy mohou použít běžné souborové rozhraní pro práci se soubory i na čtení/zápis z různých zařízení.

❖ Příklady práce se speciálními soubory:

```
dd if=/dev/hda of=mbrbackup bs=512 count=1
cat /dev/hda1 | gzip >zaloha-disku.gz
cp /dev/zero /dev/hda1 # vynulování disku
```

Přístupová práva

- V UNIXu jsou typicky rozlišena práva pro vlastníka, skupinu a ostatní.
  (Rozšíření: ACL (access control lists), viz `man acl`, `man setfacl`...)
- Uživatelé:
  - Uživatele definuje administrátor systému (root): `/etc/passwd`,
  - UID: číslo identifikující uživatele (root UID = 0).
  - Příkaz `chown` – změna vlastníka souboru (pouze root).
- Skupiny:
  - Skupiny definuje administrátor systému (root): `/etc/group`,
  - GID: číslo identifikující skupinu uživatelů,
  - Uživatel může být členem více skupin, jedna z nich je aktuální (používá se při vytváření souborů).
  - Příkaz
    - `groups` – výpis skupin uživatele,
    - `chgrp` – změna skupiny souboru,
    - `newgrp` – nový shell s jiným aktuálním GID.

Typy přístupových práv

obyčejné soubory	
`r`	právo číst obsah souboru
`w`	právo zapisovat do souboru
`x`	právo spustit soubor jako program

adresáře	
`r`	právo číst obsah (`ls adresář`)
`w`	právo zapisovat = vytváření a rušení souborů
`x`	právo přistupovat k souborům v adresáři (`cd adresář`, `ls -l adresář/soubor`)

❖ Příklad: `-rwx---r--` (číselné vyjádření: `0704`):

- obyčejný soubor,

vlastník: čtení, zápis, provedení

- skupina: nemá žádná práva

ostatní: pouze čtení

❖ Změna přístupových práv – příkaz `chmod`:

```
chmod a+rw soubory   # všichni mohou číst i zapisovat
chmod 0644 soubor    # rw-r--r--
chmod -R u-w .       # zakáže zápis vlastníkovi
chmod g+s soubor     # nastaví SGID -- viz dále
```

❖ Výpis informací o souboru:

```
ls -l soubor

-rw-r--r-- 1 joe joe  331 Sep 24 13:10 .profile
typ
 práva
          počet pevných odkazů
            vlastník
                skupina
                     velikost
                          čas poslední modifikace
                                       jméno souboru
```

Sticky bit

❖ Sticky bit je příznak, který nedovoluje rušit či přejmenovávat cizí soubory v adresáři, i když mají všichni právo zápisu.

```
chmod +t adresar      # nastaví Sticky bit
chmod 1777 /tmp
```

❖ Příklad: `/tmp` má práva `rwxrwxrwt`

SUID, SGID

❖ Určení práv pro procesy:

UID	reálná identifikace uživatele = kdo spustil proces
EUID	efektivní UID se používá pro kontrolu přístupových práv (pro běžné programy je rovno UID)
GID	reálná identifikace skupiny = skupina toho, kdo spustil proces
EGID	efektivní GID se používá pro kontrolu přístupových práv (pro běžné programy je rovno GID)

❖ Vlastník programu může propůjčit svoje práva komukoli, kdo spustí program s nastaveným SUID.

❖ Příklad: Program `passwd` musí editovat soubor `/etc/shadow`, do kterého má právo zápisu pouze superuživatel `root`.

❖ Příklad propůjčených přístupových práv: `-rwsr-Sr-x fileUID fileGID`

- `s` = je nastaveno `x`, `S` = není nastaveno `x`,
- v našem příkladu `s`: SUID=set user identification, EUID:=fileUID
- v našem příkladu `S`: SGID=set group identification: EGID:=fileGID

Typická struktura adresářů v UNIXu

- FHS = Filesystem Hierarchy Standard (Linux), část:

/bin	programy pro všechny (nutné při bootování)
/boot	soubory pro zavaděč (obrazy jádra, počátečního souborového systému)
/dev	obsahuje speciální soubory – rozhraní na zařízení
/etc	konfigurační soubory pro systém i aplikace
/home	domovské adresáře uživatelů
/lib	sdílené knihovny, moduly jádra (nutné při bootování)
/media	přípojný bod pro přenosná zařízení
/mnt	přípojný bod pro dočasné souborové systémy
/proc	obsahuje informace o procesech a jádru
/root	domovský adresář superuživatele
/run	dočasné informace o běžícím systému (typicky od démonů)
/sbin	programy pro superuživatele (nutné při bootování)
/sys	informace o jádru, zařízeních, modulech, ovladačích
/tmp	dočasné pracovní soubory

Pokračování na další straně...

Typická struktura adresářů v UNIXu – pokračování:

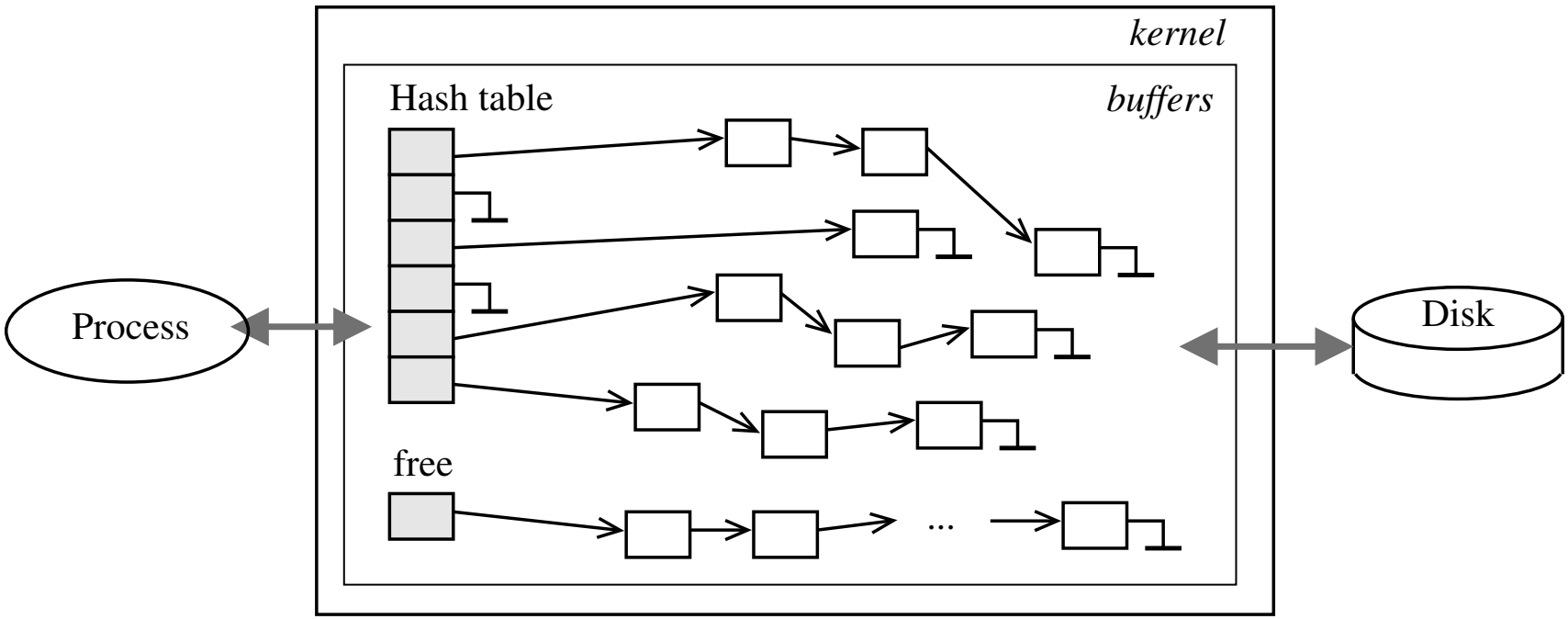
/usr	obsahuje soubory, které nejsou nutné při zavádění systému – může se přimontovat až po bootu (například ze sítě) a může být pouze pro čtení (například na CD)
/usr/bin,sbin	programy, které nejsou třeba pro bootování
/usr/lib	knihovny (statické i dynamické)
/usr/include	hlavičkové soubory pro jazyk C atd.
/usr/share	soubory, které lze sdílet (například přes síť) nezávisle na architektuře počítače
/usr/local	další hierarchie bin, sbin, lib,... určená pro lokální (nestandardní) instalace programů
/usr/src	zdrojové texty jádra systému a programů
/var	obsahuje soubory, které se mění při běhu systému
/var/log	záznamy o činnosti systému
/var/spool	pomocné soubory pro tisk atd.
/var/mail	poštovní přihrádky uživatelů

Datové struktury a algoritmy pro vstup/výstup

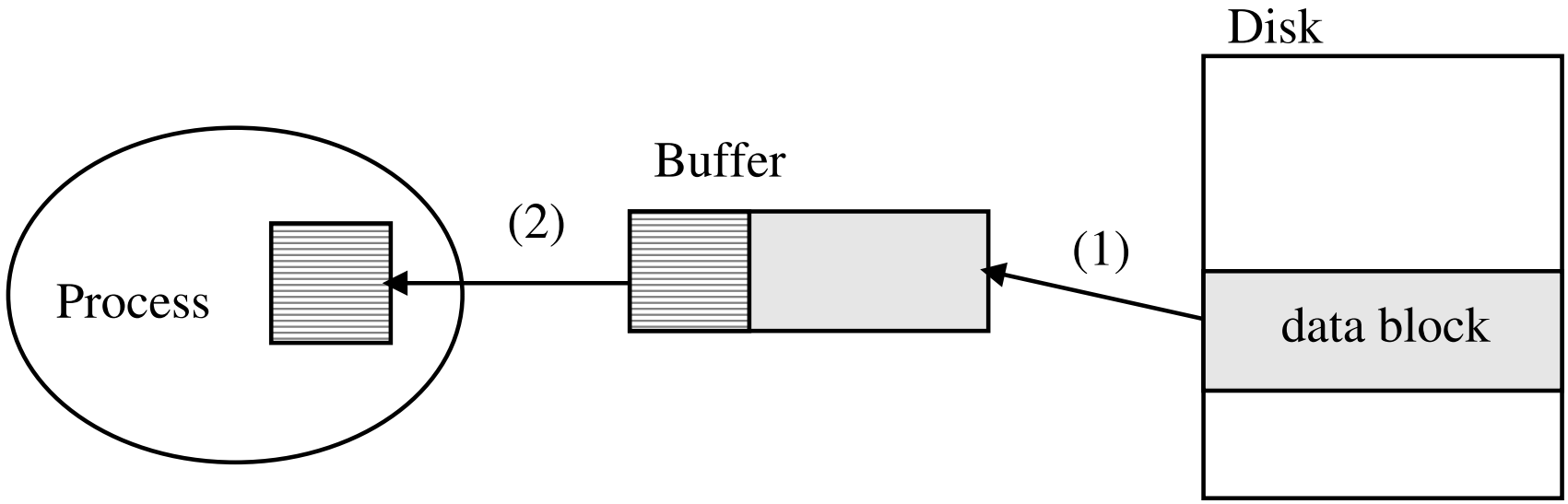
Použití vyrovnávacích pamětí

- I/O buffering:
  - Buffer = vyrovnávací paměť (VP).
  - Cílem je minimalizace počtu pomalých operací s periferiemi (typicky s disky).
  - Dílčí vyrovnávací paměti mívají velikost alokačního bloku (příp. jejich skupiny) a jsou sdruženy do kolekce (tzv. buffer pool) pevné či proměnné velikosti umožňující snadné vyhledávání.

- Možná implementace:



Čtení

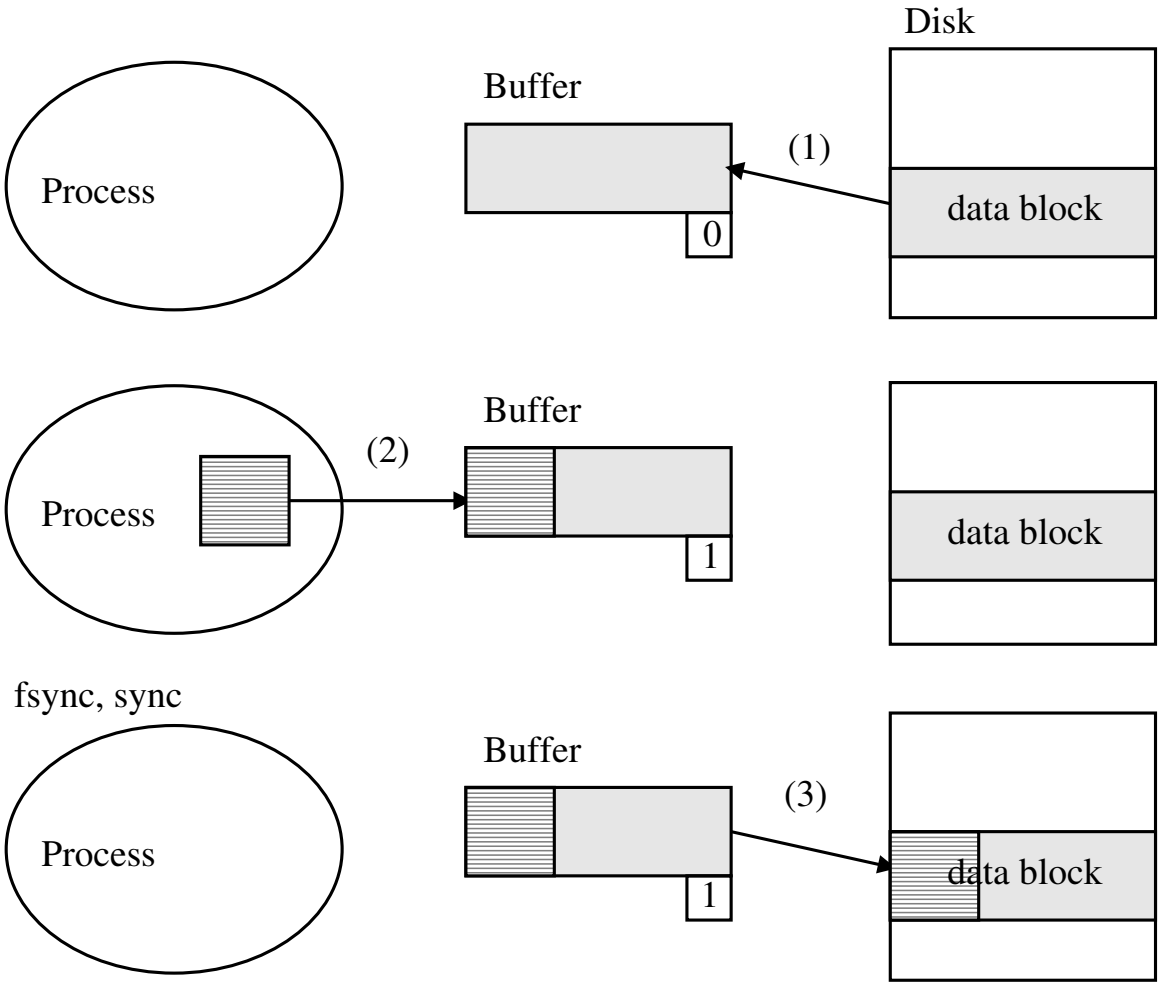


- Postup při prvním čtení (`read`):
  1. přidělení VP a načtení bloku,
  2. kopie požadovaných dat do adresového prostoru procesu (RAM→RAM).

- Při dalším čtení už pouze (2).

- Čtení, které překročí hranice bloku provede opět (1) a (2).

Zápis



- Postup při zápisu (`write`):
  1. přidělení VP a čtení bloku do VP (pokud se netvoří nový/zcela nepřepisuje),
  2. zápis dat do VP (RAM→RAM), nastaví se příznak modifikace (dirty bit),
  3. zpožděný zápis na disk, nuluje příznak.

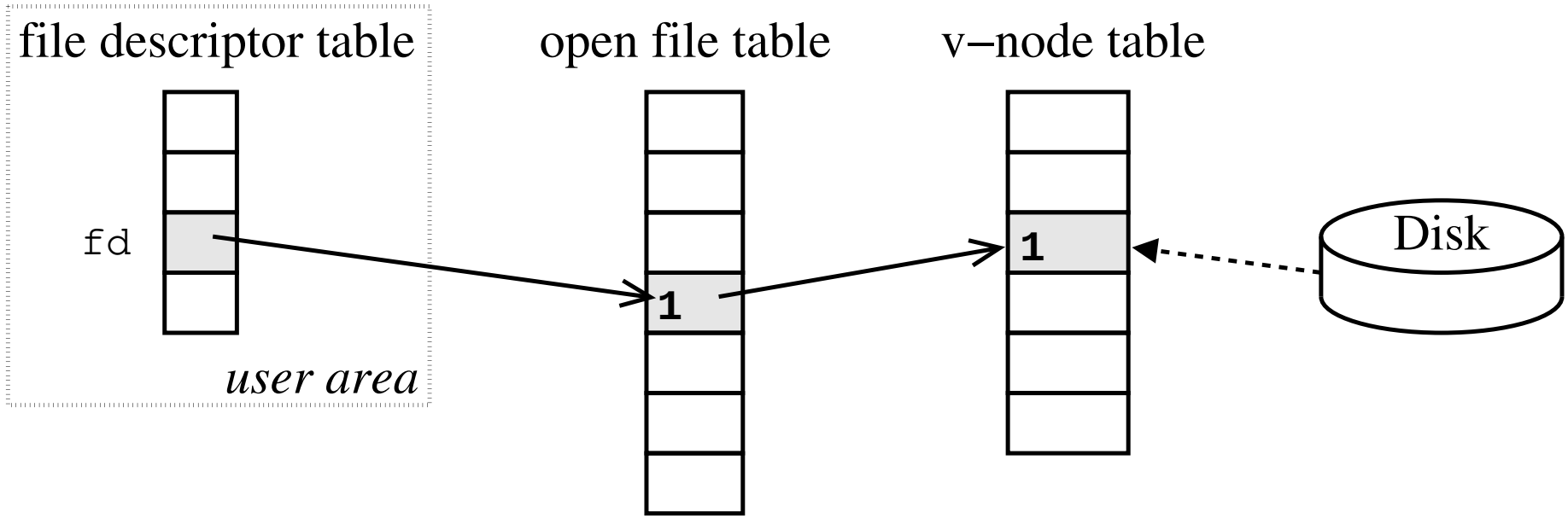
Otevření souboru pro čtení

```
fd = open("/dir/file", O_RDONLY);
```

❖ V případě, že soubor ještě nebyl otevřen:

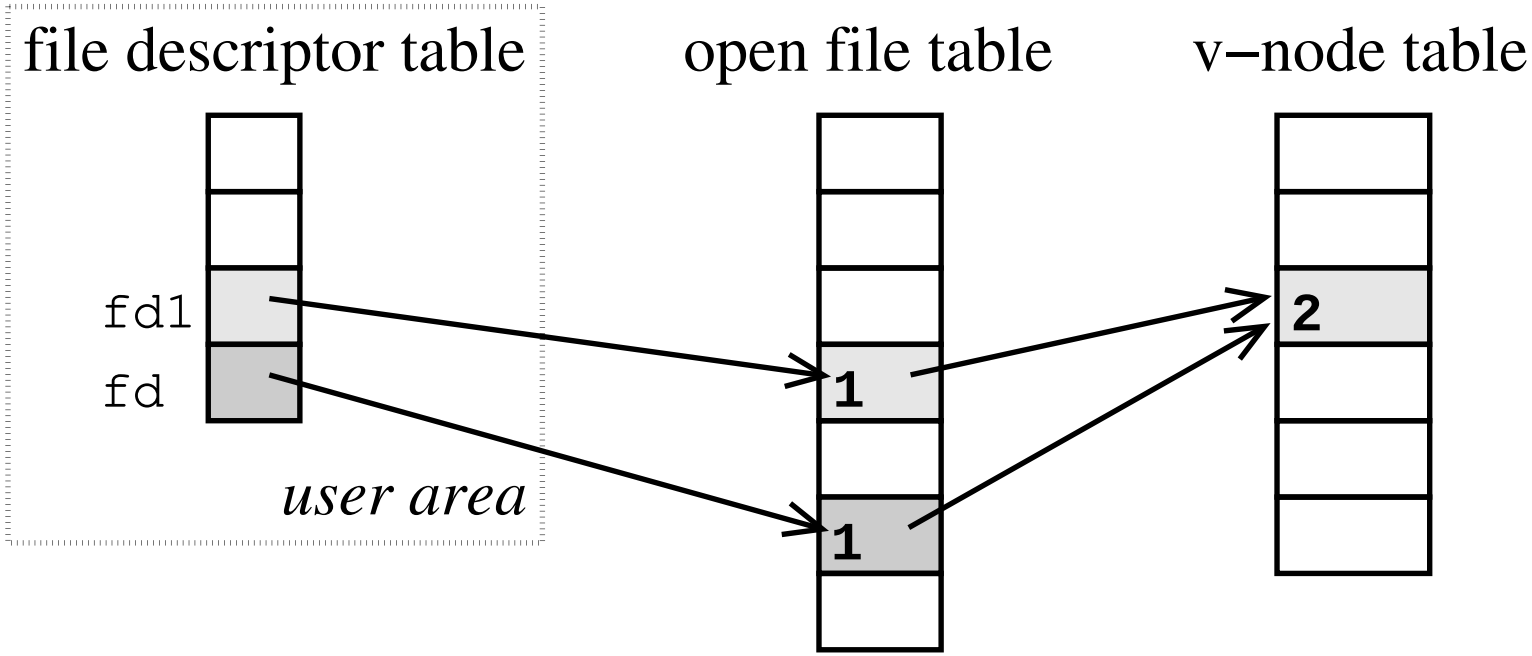
1. Vyhodnotí cestu a nalezne číslo i-uzlu: postupně načítá i-uzly adresářů a obsah těchto adresářů, aby se dostal k číslům i-uzlů pod-adresářů či hledaného souboru – čísla i-uzlů pro některá jména mohou samozřejmě být ve speciálních vyrovnávacích pamětech, tzv. d-entry cache.
2. V systémové tabulce aktivních i-uzlů vyhradí novou položku a načte do ní i-uzel. Vzniká rozšířená paměťová kopie i-uzlu: v-uzel.
3. V systémové tabulce otevřených souborů vyhradí novou položku a naplní ji:
   - odkazem na položku tabulky v-uzlů,
   - režimem otevření,
   - pozicí v souboru (0),
   - čítačem počtu referencí na tuto položku (1).
4. V poli deskriptorů souborů v záznamu o procesu v jádře nebo v tzv. uživatelské oblasti procesu vyhradí novou položku (první volná) a naplní ji odkazem na položku v tabulce otevřených souborů.
5. Vrátí index položky v poli deskriptorů (nebo -1 při chybě).

❖ Ilustrace prvního otevření souboru (čísla v obrázku udávají čítače počtu referencí na danou položku):



❖ Otevření již jednou otevřeného souboru:

1. Vyhodnotí cestu a získá číslo i-uzlu.
2. V systémové tabulce v-uzlů nalezne již načtený i-uzel: tabulka v-uzlů musí být implementována za tím účelem jako vyhledávací tabulka.
3. Zvýší počitadlo odkazů na v-uzel o 1.
4. A další beze změny.



- Při otevírání se provádí kontrola přístupových práv.
- Soubor je možno otevřít v režimu:
  - čtení,
  - zápis,
  - čtení i zápis

modifikovaných volbou dalších parametrů otevření:

  - vytvoření,
  - zkrácení na nulu,
  - přidávání,
  - synchronní zápis,
  - ...

- Při chybě vrací `-1` a nastaví chybový kód do knihovní proměnné `errno`.
  - Pro standardní chybové kódy viz `man errno`.
  - Lze užít knihovní funkcí `perror`.
  - Podobně je tomu i u ostatních systémových volání v UNIXu.

Čtení a zápis z/do souboru

❖ Čtení ze souboru – `n = read(fd, buf, 3000);`

1. Kontrola platnosti `fd`.
2. V případě, že jde o první přístup k příslušné části souboru, dojde k alokaci VP a načtení bloků souboru z disku do VP. Jinak dochází k alokaci VP a diskové operaci jen tehdy, je-li je to nutné (viz slajd k vyrovnávacím pamětem).
3. Kopie požadovaných dat z VP (RAM, jádro) do pole `buf` (RAM, adresový prostor procesu).
4. Funkce vrací počet opravdu přečtených bajtů nebo -1 při chybě (při současném nastevní `errno`).

❖ Zápis do souboru – `n = write(fd, buf, 3000);`

- Funguje podobně jako `read` (viz slajd k vyrovnávacím pamětem).
- Před vlastním zápisem kontroluje dostupnost diskového prostoru a prostor rezervuje.
- Funkce vrací počet opravdu zapsaných bajtů nebo -1 při chybě.

Přímý přístup k souboru

```
n = lseek(fd, offset, whence);
```

❖ Postup při žádosti o přímý přístup k souboru:

1. Kontrola platnosti `fd`.
2. Nastaví pozici `offset` bajtů od
   - začátku souboru pro `whence=SEEK_SET`,
   - aktuální pozice pro `whence=SEEK_CUR`,
   - konce souboru pro `whence=SEEK_END`.
3. Funkce vrací výslednou pozici od začátku souboru nebo -1 při chybě.

❖ Poznámka: Hodnota parametru `offset` může být záporná, nelze však nastavit pozici před začátek souboru.

❖ Sparse files – “řídké soubory”:

- Vznikají nastavením pozice za konec souboru a zápisem.
- Bloky, do kterých se nezapisovalo nejsou alokovány a nezabírají diskový prostor. Při čtení se považují za vynulované.
- Někdy také „hole punching“: mazání prostoru uvnitř souboru (např. `fallocate`).

Zavření souboru

```
x = close(fd);
```

- Postup při uzavírání souboru:
  1. Kontrola platnosti `fd`.
  2. Uvolní se odpovídající položka v tabulce deskriptorů, sníží se počitadlo odkazů v odpovídající položce tabulky otevřených souborů.
  3. Pokud je počitadlo odkazů nulové, uvolní se odpovídající položka v tabulce otevřených souborů a sníží se počitadlo odkazů ve v-uzlu.
  4. Pokud je počitadlo odkazů nulové, i-uzel se z v-uzlu okopíruje do VP a uvolní.
  5. Funkce vrací nulu nebo -1 při chybě.

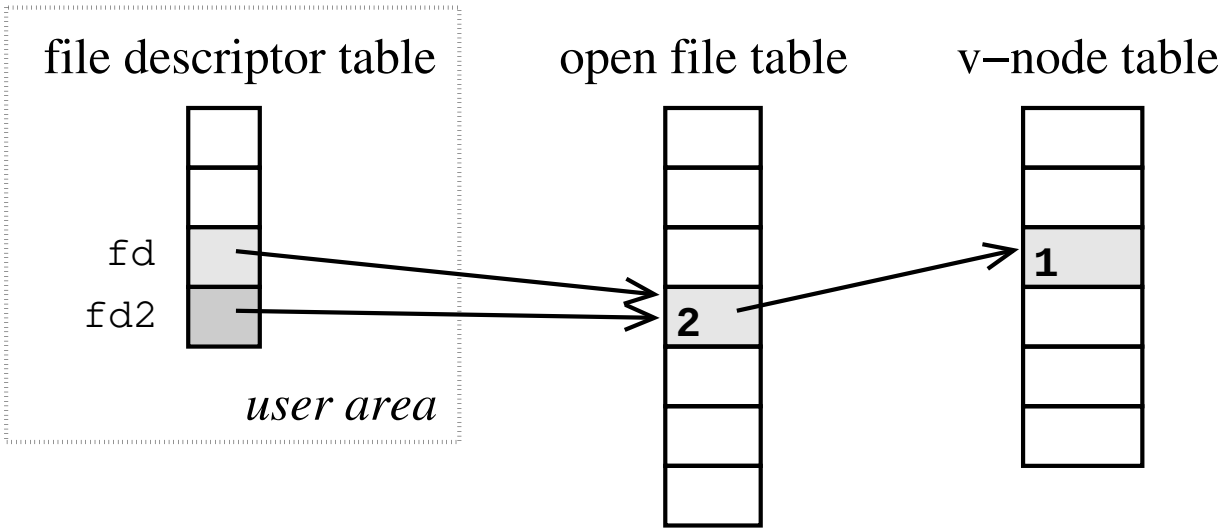
- Pokud se ukončuje proces, automaticky se uzavírají všechny jeho deskriptory.

- Uzavření souboru nezpůsobí uložení obsahu jeho VP na disk!

Duplikace deskriptoru souboru

```
fd2 = dup(fd);
fd2 = dup2(fd,newfd);
```

- Postup při duplikaci deskriptoru:
  1. Kontrola platnosti `fd`.
  2. Kopíruje danou položku v tabulce deskriptorů do první volné položky (`dup`) nebo do zadané položky (`dup2`). Je-li deskriptor `newfd` otevřen, `dup2` ho automaticky uzavře.
  3. Zvýší počitadlo odkazů v odpovídající položce tabulky otevřených souborů.
  4. Funkce vrací index nové položky nebo -1 při chybě.



- Poznámka: Použití pro přesměrování stdin/stdout.

Rušení souboru

```
x = unlink("/dir/file");
```

- Postup při rušení souboru:
  1. Vyhodnocení cesty, kontrola platnosti jména souboru a přístupových práv.
  2. Odstraní pevný odkaz (hard link) mezi jménem souboru a i-uzlem. Vyžaduje právo zápisu do adresáře.
  3. Zmenší počitadlo jmen v i-uzlu.
  4. Pokud počet jmen klesne na nulu a i-uzel nikdo nepoužívá, je i-uzel uvolněn včetně všech používaných bloků souboru. Je-li i-uzel používán, bude uvolnění odloženo až do okamžiku zavření souboru (počitadlo otevření souboru klesne na 0).
  5. Funkce vrací nulu nebo -1 při chybě.

- Poznámky:
  - Proces může provést `unlink` na otevřený soubor a dále s ním pracovat až do jeho uzavření.
  - Je možné zrušit spustitelný soubor, i když běží jím řízené procesy (výhoda při instalaci nových verzí programů).
  - Bezpečnější mazání: `shred`.

Další operace se soubory

- Vytvoření souboru: `creat`, `open`
- Přejmenování: `rename`
- Zkrácení: `truncate`, `ftruncate`
- Zamykání záznamů: `fcntl` nebo `lockf`
- Změna atributů: `chmod`, `chown`, `utime`
- Získání atributů: `stat`
- Zápis VP na disk: `sync`, `fsync`

Adresářové soubory

- Adresáře se liší od běžných souborů:
  - vytváří se voláním `mkdir` (vytvoří položky `.` a `..`),
  - mohou být otevřeny voláním `opendir`,
  - mohou být čteny voláním `readdir`,
  - mohou být uzavřeny voláním `closedir`,
  - modifikaci je možné provést pouze vytvářením a rušením souborů v adresáři (`creat`, `link`, `unlink`, ...).

- Poznámka: Adresáře nelze číst/zapisovat po bajtech!

- Příklad obsahu adresáře:

32577	.
2	..
2361782	Archiv
1058839	Mail
1661377	tmp

Blokové a znakové speciální soubory

❖ Představují rozhraní k blokovým/znakovým zařízením (disky, logické disky, terminály, myš, paměť, ...).

- Lze vytvořit pomocí `mknod`.
- Normálně řeší přímo jádro či různí démoni (např. `udev`, `devd`).

❖ Jádro mapuje běžné souborové operace (`open`, `read`, ...) nad blokovými a znakovými speciálními soubory na odpovídající podprogramy tyto operace implementující nad příslušným zařízením prostřednictvím dvou tabulek:

- tabulky znakových zařízení a
- tabulky blokových zařízení.

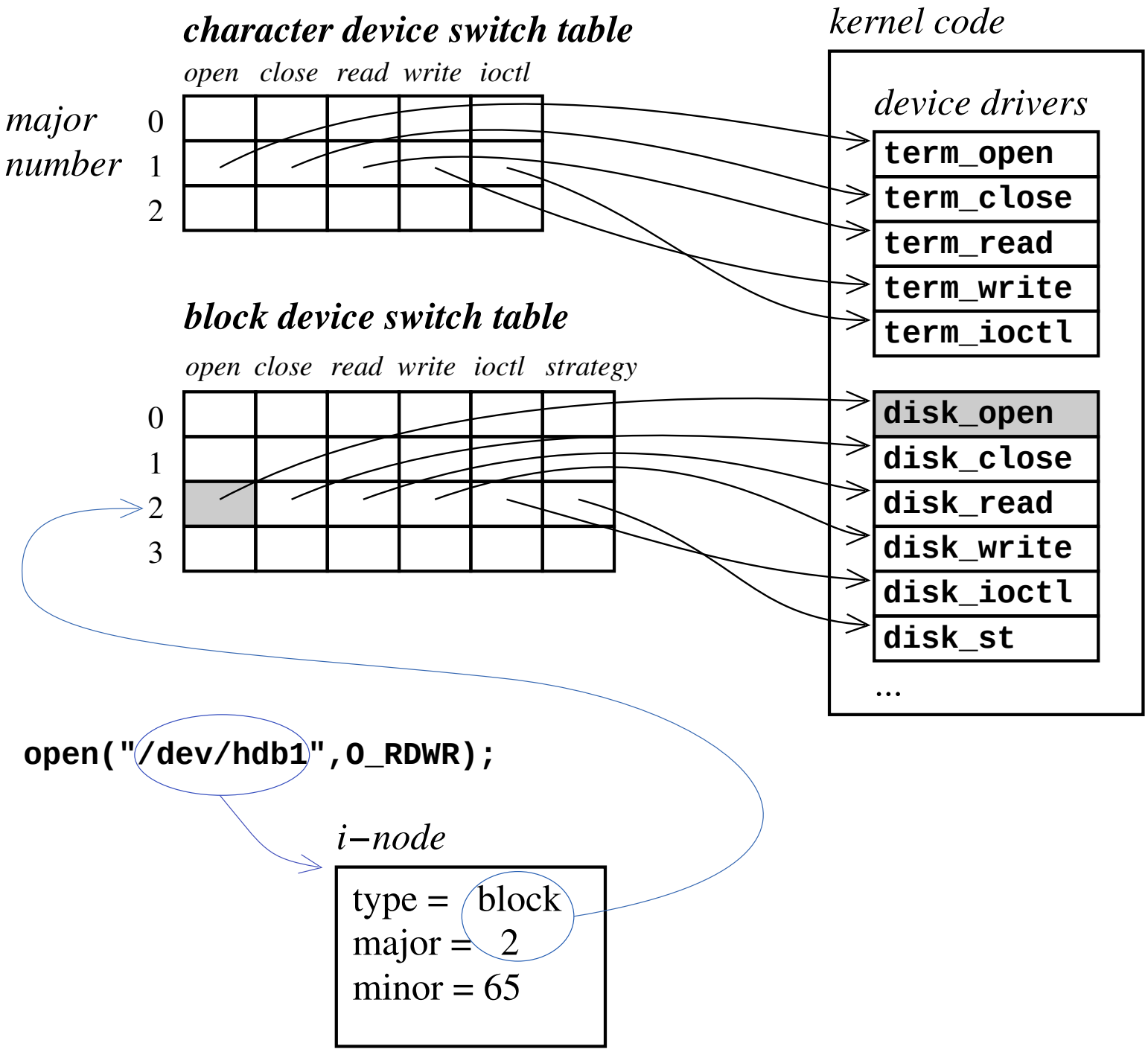
❖ Zmíněné tabulky obsahují ukazatele na funkce implementující příslušné operace v ovladačích příslušných zařízení.

❖ Ovladač (*device driver*) je sada podprogramů pro řízení určitého *typu* zařízení.

❖ Speciální soubory typu zařízení (např. `/dev/sda`, `/dev/tty`, ...): mají v i-uzlu mj. typ souboru a dvě čísla:

číslo	*význam*
hlavní číslo (major number)	typ zařízení
vedlejší číslo (minor number)	instance zařízení

- Typ souboru (blokový nebo znakový) určuje tabulku.
- Hlavní číslo se použije jako index do tabulky zařízení.
- Vedlejší číslo se předá jako parametr funkce ovladače:
  – identifikace instance zařízení.


character device switch table
open close read write ioctl
major
number
0
1
2
kernel code
device drivers
term_open
term_close
term_read
term_write
term_ioctl
block device switch table
open close read write ioctl strategy
0
1
2
3
disk_open
disk_close
disk_read
disk_write
disk_ioctl
disk_st
...
open("/dev/hdb1",O_RDWR);
i−node
type = block
major = 2
minor = 65

Terminály

❖ Terminály – fyzická nebo logická zařízení umožňující (primárně) textový vstup/výstup systému: vstup/výstup po řádcích, editace na vstupním řádku, speciální znaky (Ctrl-C, Ctrl-D, ...), ...

❖ Rozhraní:

- `/dev/tty` – řídící terminál aktuálního procesu, odkaz na příslušné terminálové zařízení,
- `/dev/ttyS1`, ... – terminál na sériové lince,
- `/dev/tty1`, ... – virtuální terminály (konzole),
- pseudoterminály (např. `/dev/ptmx` a `/dev/pts/1`, ...) – tvořeny párem master/slave emulujícím komunikaci přes sériovou linku (např. použito u X-terminálu či ssh).

❖ Různé režimy zpracování znaků (line discipline):

režim	význam
raw	bez zpracování
cbreak	zpracovává jen některé znaky (zejména Ctrl-C, mazání)
cooked	zpracovává vše

❖ Nastavení režimu zpracování znaků (nastavení ovladače terminálu): program `stty`.

❖ Nastavení režimu terminálu (tedy fyzického zařízení nebo emulujícího programu):

- proměnná `TERM` – typ aktuálního terminálu,
- databáze popisu terminálů (možnosti nastavení terminálu): `terminfo` či `termcap`.
- nastavení příkazy: `tset`, `tput`, `reset`, ...

❖ Knihovna `curses` – standardní knihovna pro řízení terminálu a tvorbu aplikací s terminálovým uživatelským rozhraním (včetně menu, textových oken apod.).

Roury

- Roury (pipes) – jeden z typů speciálních souborů. Rozlišujeme:
  - roury nepojmenované
    - nemají adresářovou položku,
    - `pipe` vrací dva deskriptory (čtecí a zápisový), jsou přístupné pouze příbuzným procesům (tvůrce fronty je přímý či nepřímý předek komunikujících procesů nebo jeden z nich) – případně lze zaslat přes Unixové sockety,
    - vytváří se v kolonách (např. `p1 | p2 | p3`),
  - Roury pojmenované – vytvoření `mknod` či `mkfifo`.

- Roury representují jeden z mechanismů meziprocesové komunikace.

- Implementace: kruhový buffer s omezenou kapacitou.

- Procesy komunikující přes rouru (producent a konzument) jsou synchronizovány.

Sockets

❖ Umožňují síťovou i lokální komunikaci.

❖ Lokální komunikace může probíhat přes sockety pojmenované a zpřístupněné v souborovém systému.

❖ API pro práci se sockets:

- vytvoření (`socket`),
- čekání na připojení (`bind`, `listen`, `accept`),
- připojení ke vzdálenému počítači (`connect`),
- příjem a vysílání (`recv`, `send`, `read`, `write`),
- uzavření (`close`).

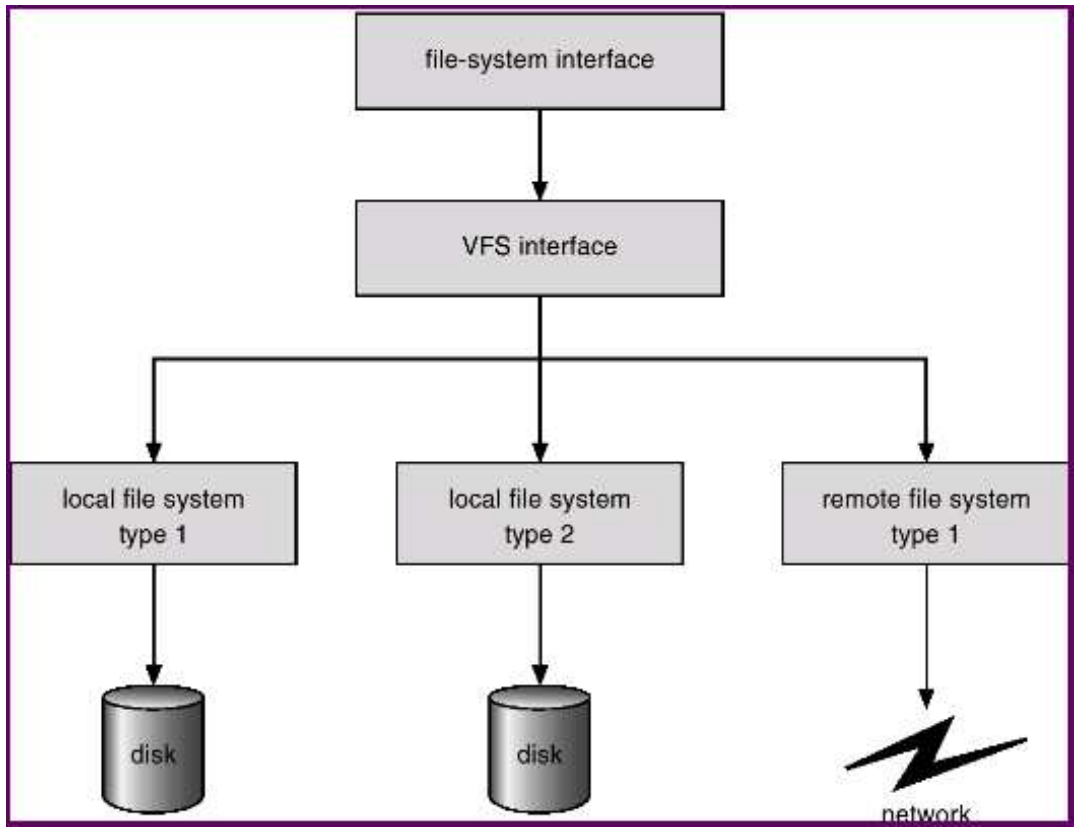
❖ Sockets podporují blokující/neblokující I/O.

- Pro současnou obsluhu více sockets jedním procesem (vláknem) lze užít `select`.
- Testuje dostupnost/čeká na dostupnost operace na množině popisovačů.
- `select` lze užít i u jiných typů souborů.

VFS

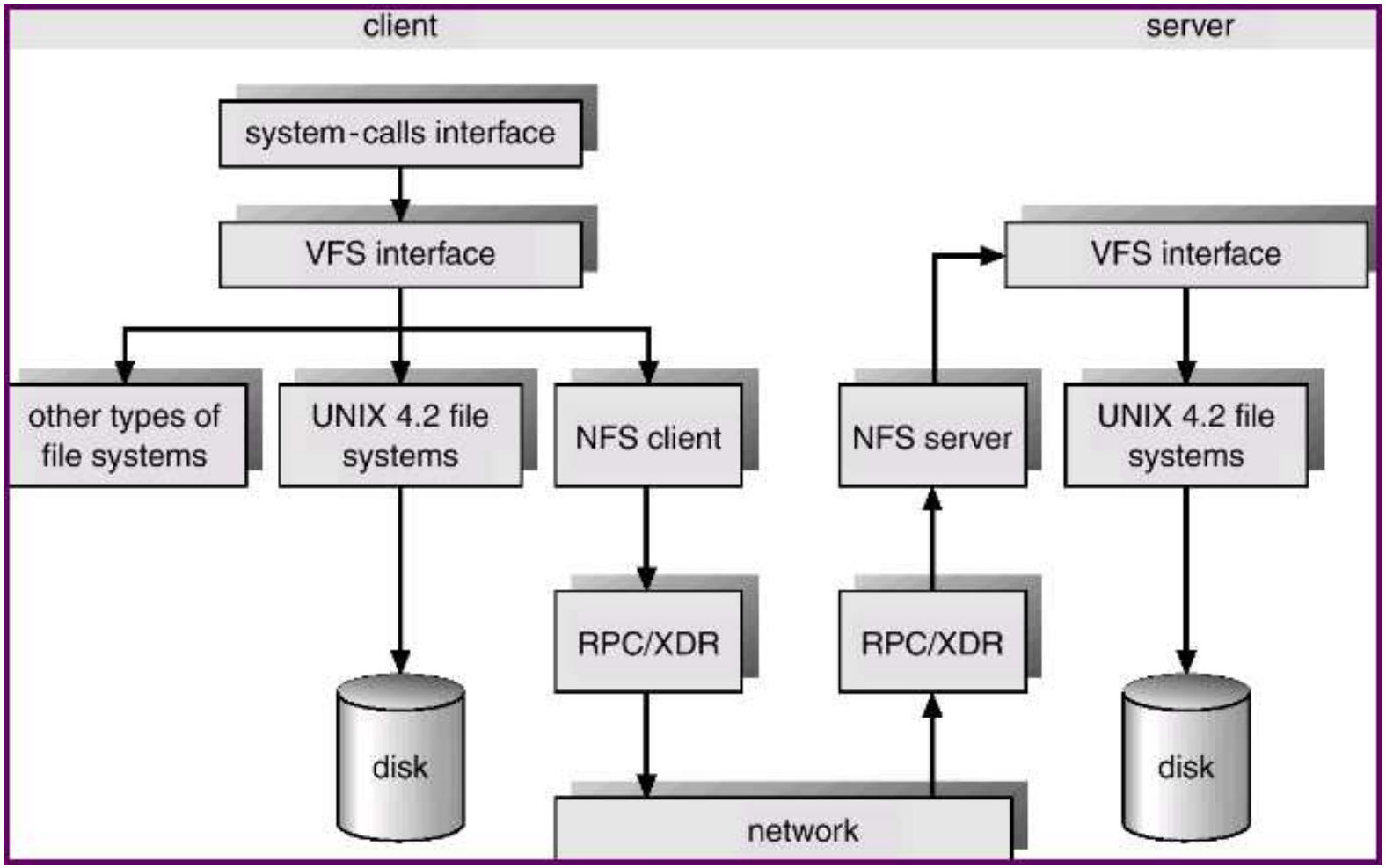
❖ VFS (Virtual File System) vytváří jednotné rozhraní pro práci s různými souborovými systémy, odděluje vyšší vrstvy OS od konkrétní implementace jednotlivých souborových operací na jednotlivých souborových systémech.

❖ Pro popis souborů používá rozšířené i-uzly (tzv. v-uzly), které mohou obsahovat např. počet odkazů na v-uzel z tabulky otevřených souborů, ukazatele na funkce implementující operace nad i-uzlem v patřičném souborovém systému apod.



NFS

❖ NFS (Network File System) – transparentně zpřístupňuje soubory uložené na vzdálených systémech.



❖ Umožňuje kaskádování: lokální připojení vzdáleného adresářového systému do jiného vzdáleného adresářového systému.

❖ Autentizace často prostřednictvím uid a gid – pozor na bezpečnost!

❖ NFS verze 3:

- bezestavové – nepoužívá operace otevírání a uzavírání souborů, každá operace musí nést veškeré potřebné argumenty,
- na straně klienta se neužívá cache,
- nemá podporu zamykání.

❖ NFS verze 4:

- stavové,
- cache na straně klienta,
- podpora zamykání.

Spooling

❖ Spooling = simultaneous peripheral operations on-line.

❖ spool = vyrovnávací paměť (typicky soubor) pro zařízení (nejčastěji tiskárny), které neumožňují prokládané zpracování dat různých procesů.

❖ Výstup je proveden do souboru, požadavek na jeho fyzické zpracování se zařadí do fronty, uživatelský proces může pokračovat a zpracování dat se provede, až ně přijde řada.

❖ V Unixu/Linuxu: `/var/spool`