

11., 12. Správa I/O Správa souborů

ZOS 2014, L. Pešička



Doplnění

□ Alokace paměti pro procesy

- explicitní správa paměti
- čítání referencí
- garbage collection

viz pdf s dodatkem – přečíst (!)

pamatuj !!



v C o paměť
žádáme **malloc()** a
máme ji uvolnit
free()

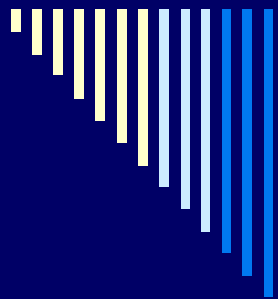
paměť nám přidělí
knihovna – **alokátor
paměti**, která
spravuje volnou
paměť

pokud paměť nemá
alokátor k dispozici,
**požádá operační
systém**
systémovým
voláním o přidělení
další části paměti
(další stránky)



malloc, brk, sbrk

- ❑ malloc není systémové volání, ale **knihovní funkce**
viz **man 3 malloc** x **man 2 fork**
- ❑ malloc alokuje paměť z haldy
- ❑ velikost haldy se nastaví dle potřeby systémovým voláním **sbrk**
- ❑ **brk** – syscall – nastaví adresu konce datového segmentu procesu
- ❑ **sbrk** – syscall – zvětší velikost datového segmentu o zadaný počet bytů (0 – jen zjistím současnou adresu)



používané vs. nepoužívané objekty

```
Object x = new Foo();
```

```
Object y = new Bar();
```

```
x = new Quux();
```

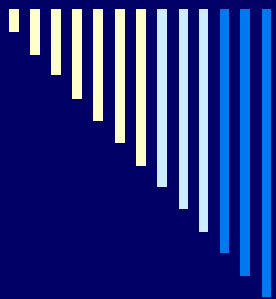
```
/* víme, že Foo object původně přiřazený x nebude nikdy dostupný,  
jde o syntactic garbage */
```

```
if ( x.check_something() )
```

```
    { x.do_something(y); }
```

```
System.exit(0);
```

```
/* y může být semantic garbage, ale nevíme, dokud  
x.check_something() nevrátí návratovou hodnotu */
```



Velikost stránky v OS

Standardní velikost je 4096 bytů (4KB)

huge page size: 4MB

large page size: 1GB



Zjištění velikosti stránky - Linux

```
#include <stdio.h>
#include <unistd.h>

int main(void) {

    printf("Velikost stranky je %ld bytu.\n",
        sysconf(_SC_PAGESIZE) );

    return 0;

}
```

příkazem na konzoli:
getconf PAGESIZE

man sysconf

eryx.zcu.cz: 4096
ares.fav.zcu.cz: 4096

Zjištění velikosti stránky - WIN

```
#include "stdafx.h"  
#include <stdio.h>  
#include <Windows.h>
```

```
int _tmain(int argc, _TCHAR* argv[]) {
```

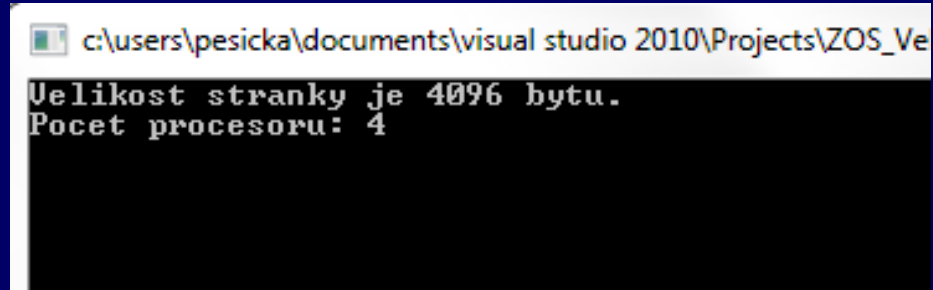
```
    SYSTEM_INFO si;
```

```
    GetSystemInfo(&si);
```

```
    printf("Velikost stranky je %u bytu.\n", si.dwPageSize);
```

```
    printf("Pocet procesoru: %u\n", si.dwNumberOfProcessors);
```

```
    getchar(); return 0; }
```

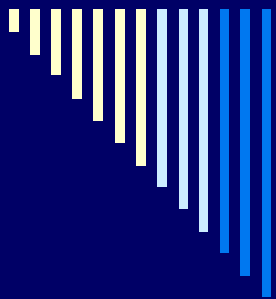


The screenshot shows a Windows command prompt window with the title bar "c:\users\pesicka\documents\visual studio 2010\Projects\ZOS_Ve". The command prompt displays the output of the program: "Velikost stranky je 4096 bytu." and "Pocet procesoru: 4".



OS

- Modul pro správu procesů
- Modul pro správu paměti
- **Správa i/o**
- Správa souborů
- síťování



Vývoj rozhraní mezi CPU a zařízeními

1. CPU řídí přímo periferii
2. CPU – řadič – periférie
aktivní čekání CPU na dokončení operace
3. řadič umí vyvolat přerušení
4. řadič umí DMA
5. I/O modul
6. I/O modul s vlastní pamětí



1. CPU řídí přímo periférii

- ❑ CPU přímo vydává potřebné signály
 - ❑ CPU dekoduje signály poskytovaném zařízením
 - ❑ Nejjednodušší HW
 - ❑ Nejméně efektivní využití CPU
-
- ❑ Jen v jednoduchých mikroprocesorem řízených zařízeních (dálkové ovládání televize)



2. CPU – řadič - periférie

Řadič (device controller)

- Převádí příkazy CPU na elektrické impulzy pro zařízení
 - Poskytuje CPU info o stavu zařízení
 - Komunikace s CPU pomocí **registrů** řadiče na známých I/O adresách
 - HW buffer pro alespoň 1 záznam (blok, znak, řádka)
 - Rozhraní řadič-periférie může být standardizováno (SCSI, IDE, ...)
-



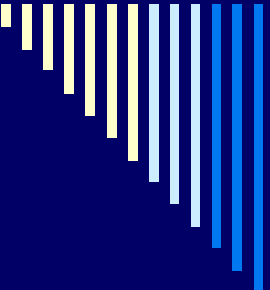
2. řadič – příklad operace zápisu

- CPU zapíše data do bufferu,
Informuje řadič o požadované operaci
- Po dokončení výstupu zařízení nastaví příznak, který
může CPU otestovat
- if přenos == OK, může vložit další data
- CPU musí dělat všechno (programové I/O)
- Významnou část času stráví CPU čekáním na
dokončení I/O operace



3. Řadič umí vyvolat přerušení

- CPU nemusí testovat příznak dokončení
- Při dokončení I/O vyvolá řadič **přerušení**
- CPU začne obsluhovat přerušení
 - Provádí instrukce na předdefinovaném místě
 - Obslužná procedura přerušení
 - Určí co dál
- Postačuje pro pomalá zařízení, např. sériové I/O



4. Řadič může přistupovat k paměti pomocí DMA

- ❑ DMA přenosy mezi pamětí a buffery
 - ❑ CPU vysílá příkazy,
při přerušení analyzuje status zařízení
 - ❑ CPU inicializuje přenos, ale sám ho nevykonává
 - ❑ Bus mastering – zařízení převezme kontrolu nad sběrnici a přenos provede samo (PCI sběrnice)
 - ❑ Vhodné pro rychlá zařízení – řadič disků, síťová karta, zvuková karta, grafická karta atd.
-



5. I/O modul umí interpretovat speciální I/O programy

- ❑ I/O procesor
- ❑ Interpretuje programy **v hlavní paměti**
- ❑ CPU spustí I/O procesor
I/O procesor provádí své instrukce samostatně



6. I/O modul s vlastní pamětí

- ❑ I/O modul provádí programy
- ❑ Má **vlastní paměť**(!)
 - Je vlastně **samostatným počítačem**
- ❑ Složité a časově náročné operace
grafika, šifrování, ...



Komunikace CPU s řadičem

- **Odlišné adresní prostory**
 - CPU zapisuje do registrů řadiče pomocí speciálních I/O instrukcí
 - Vstup: **IN** R, port
 - Výstup: **OUT** R, port
- **1 adresní prostor**
- **Hybridní schéma**



Ad – 1 adresní prostor

- Používá **vyhrazené adresy**
- Nazývá se *paměťově mapované I/O*
- HW musí pro dané adresy umět **vypnout cachování**
- Danou oblast můžeme namapovat do virtuálního adresního prostoru nějakého procesu (zpřístupnění I/O zařízení)



Ad – hybridní schéma

- Řídící registry
 - Přístup pomocí I/O instrukcí
 - HW buffer
 - Mapován do paměti
 - Např. PC
(buffery mapovány do oblasti 640K až 1MB)
-



RAID

- pevný disk
 - elektronická část + mechanická
 - náchylost k poruchám
 - cena dat >> cena hw
 - odstávka při výměně zařízení
 - náhrada hw, přenos dat ze zálohy - prostoje
 - SLA 24/7
 - větší disková kapacita než 1 disk
 - RAID
 - Redundant Array of Independent (Inexpensive) disks
-



Disk

Rotační disky

doba vystavení + rotační zpoždění

SSD disky

dražší, menší kapacita

Mix

SSD disk v kombinaci s rotačním diskem



RAID – používané úrovně

- RAID 0, 1, 5
 - RAID 10 .. kombinace 0 a 1
 - RAID 6 .. zdvojená parita

 - pojmy:
 - SW nebo HW RAID
 - hot plug
 - hot spare
 - Degradovaný režim – jeden (či více dle typu RAIDu) z disků v poli je porouchaný, ale RAID stále funguje
-



RAID 0

Dva režimy RAID 0:
zřetězení a prokládání

- ❑ není redundantní, neposkytuje ochranu dat
- ❑ ztráta 1 disku – ztráta celého pole nebo části (dle režimu)
- ❑ důvod použití – může být výkon při režimu prokládání (např. střih videa)
- ❑ Dva režimy – zřetězení nebo prokládání (stripping)

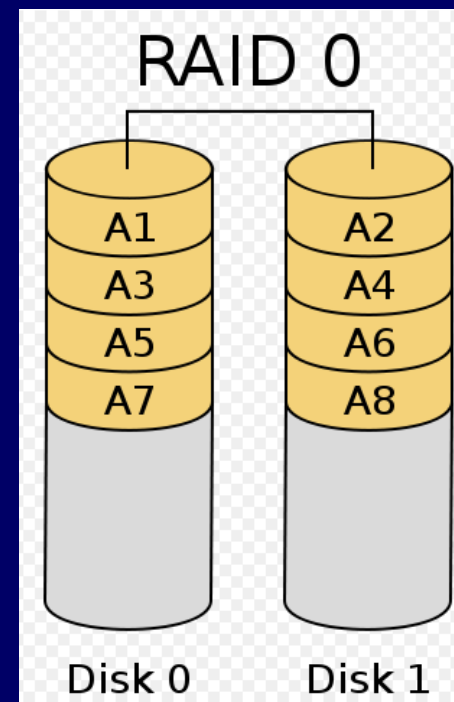
RAID 0

□ Zřetězení

- Data postupně ukládána na několik disků
- Zaplní se první disk, pak druhý, atd.
- Snadné zvětšení kapacity, při poruše disku ztratíme jen část dat

□ Prokládání

- Data ukládána na disky cyklicky po blocích (stripy)
- Při poruše jednoho z disků – přijdeme o data
- Větší rychlost čtení / zápisu
 - Jeden blok z jednoho disku, druhý blok z druhého disku



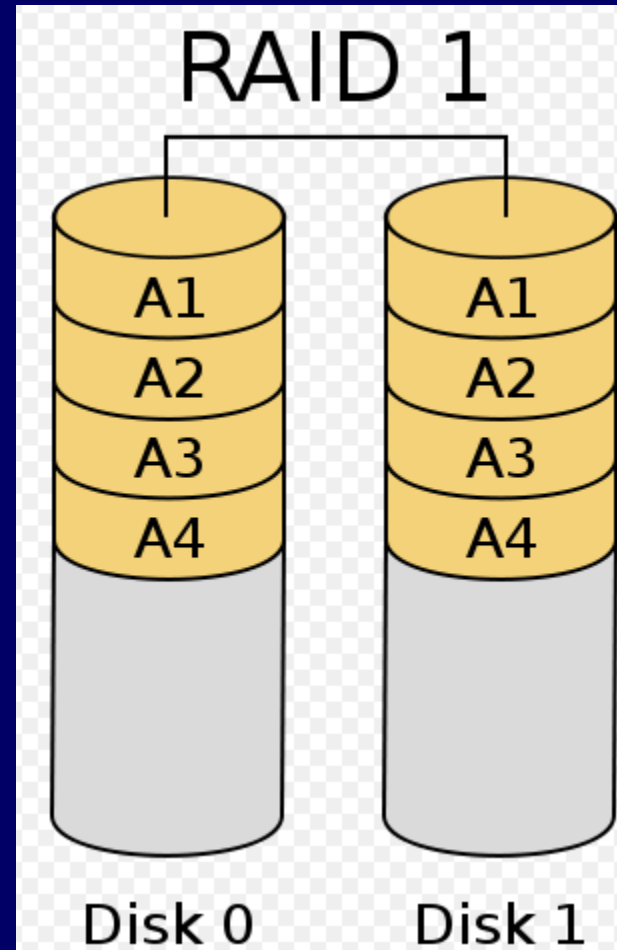
Na obrázku je režim prokládání, zdroj: wikipedia (i u dalších obrázků)



RAID 1

- mirroring .. zrcadlení
- na 2 disky stejných kapacit totožné informace
- výpadek 1 disku – nevadí
- jednoduchá implementace – často čistě sw
- nevýhoda – využijeme jen polovinu kapacity
- zápis – pomalejší (stejná data na 2 disky)
ovlivněn diskem, na němž bude trvat déle
- čtení – rychlejší
(řadič - lze střídat požadavky mezi disky)

RAID 1

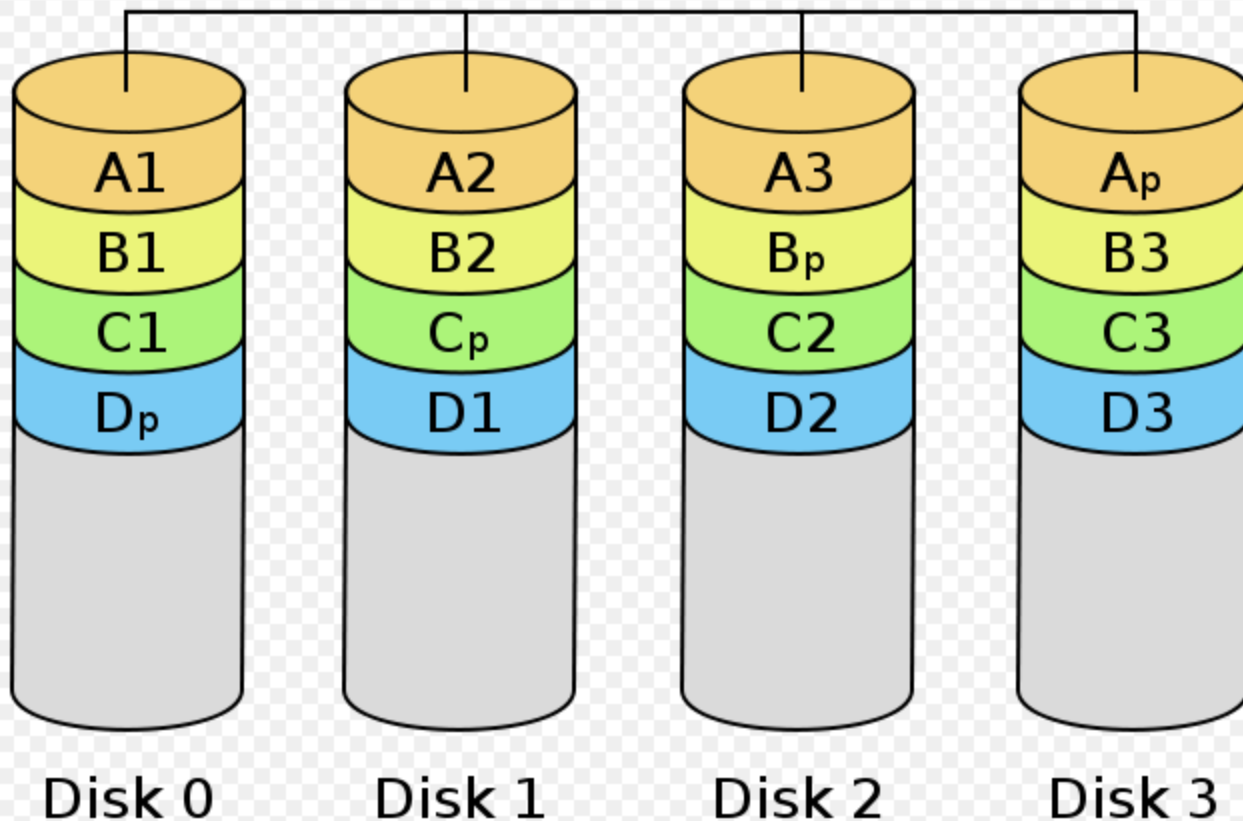




RAID 5

- ❑ redundantní pole s distribuovanou paritou
- ❑ minimálně 3 disky
- ❑ režie: 1 disk z pole n disků
 - 5 disků 100GB : 400GB pro data
- ❑ výpadek 1 disku nevadí
- ❑ čtení – výkon ok
- ❑ zápis – pomalejší
 - 1 zápis – čtení starých dat, čtení staré parity, výpočet nové parity, zápis nových dat, zápis nové parity

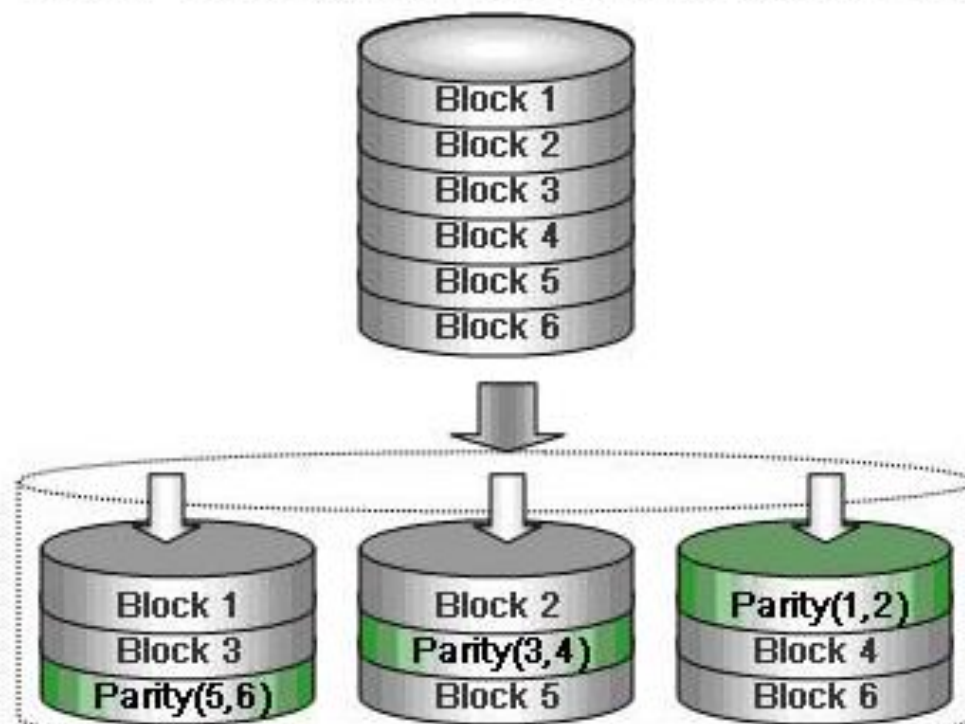
RAID 5



Např. RAID 5 z 4 disků 1TB, výsledná kapacita: 3 TB
Může vypadnout 1 z disků a o data nepřijde

RAID 5

Raid 5 - Disk Striping with Single Distributed Parity



Zdroj:

<http://www.partitionwizard.com/resize-partition/resize-raid5.html>



RAID 5

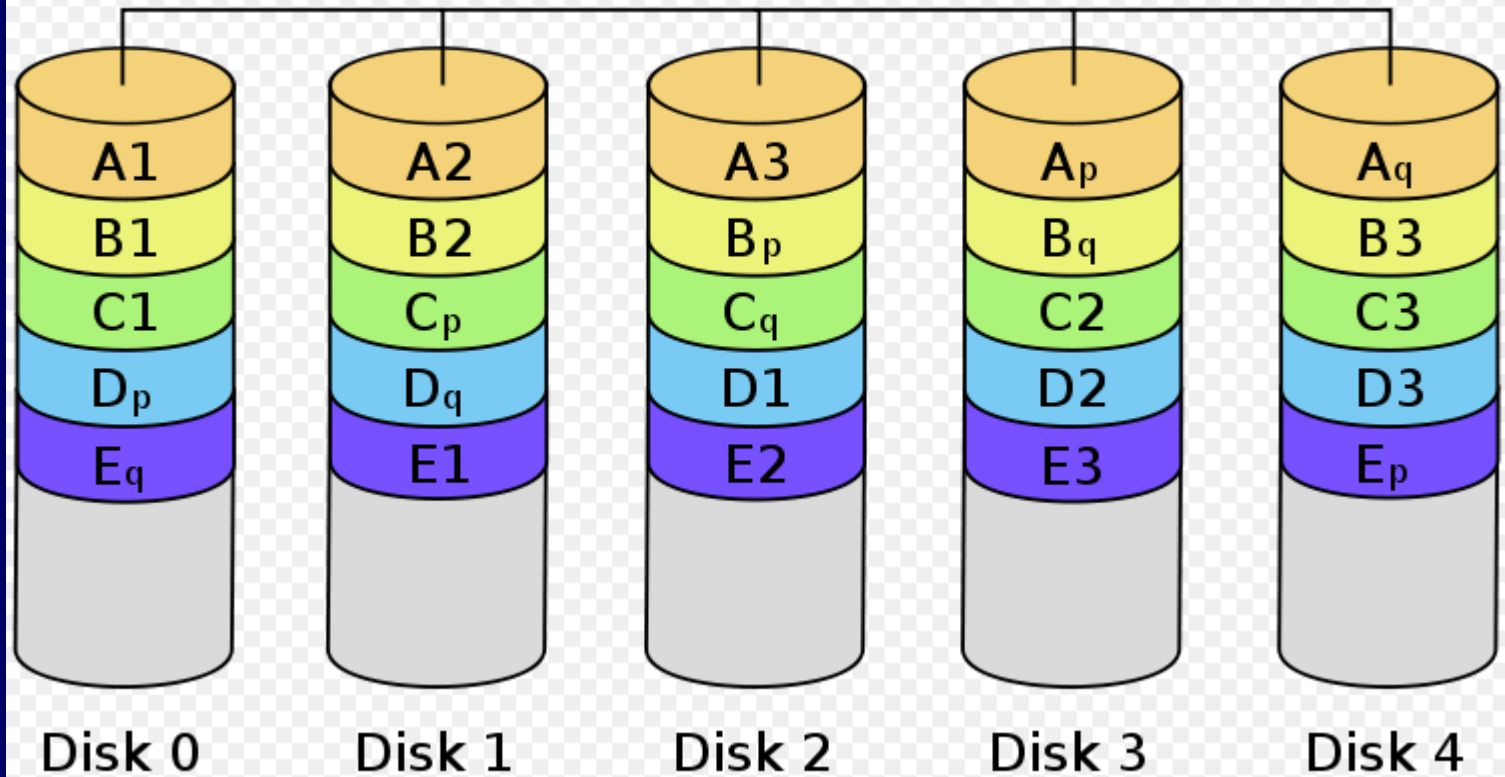
- nepoužívanější
- detekce poruchy v diskovém poli
- hot spare disk
- použití hot plug disků



RAID 6

- ❑ RAID 5 + navíc další paritní disk
- ❑ odolné proti výpadku dvou disků
- ❑ Rychlost čtení srovnatelná s RAID 5
- ❑ Zápis pomalejší

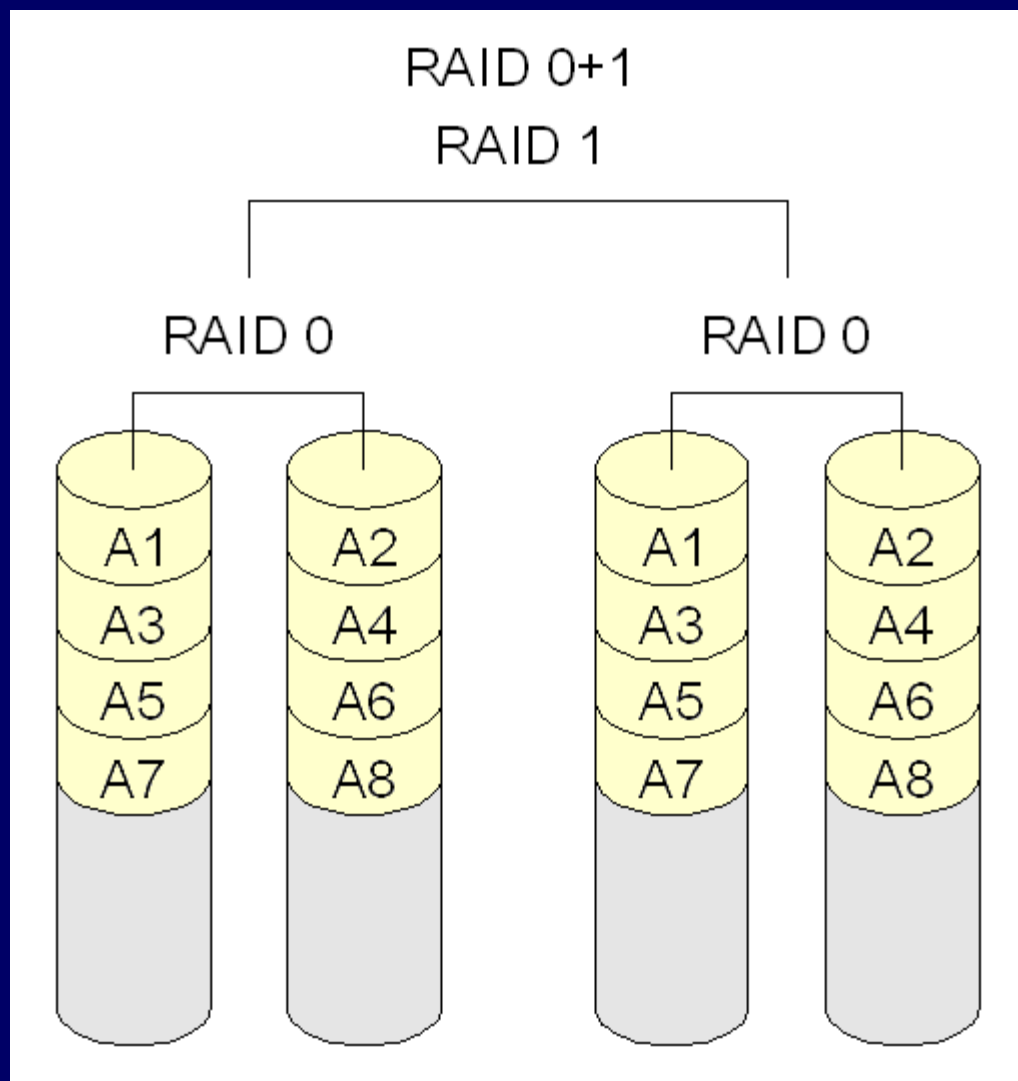
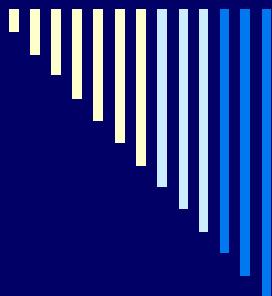
RAID 6

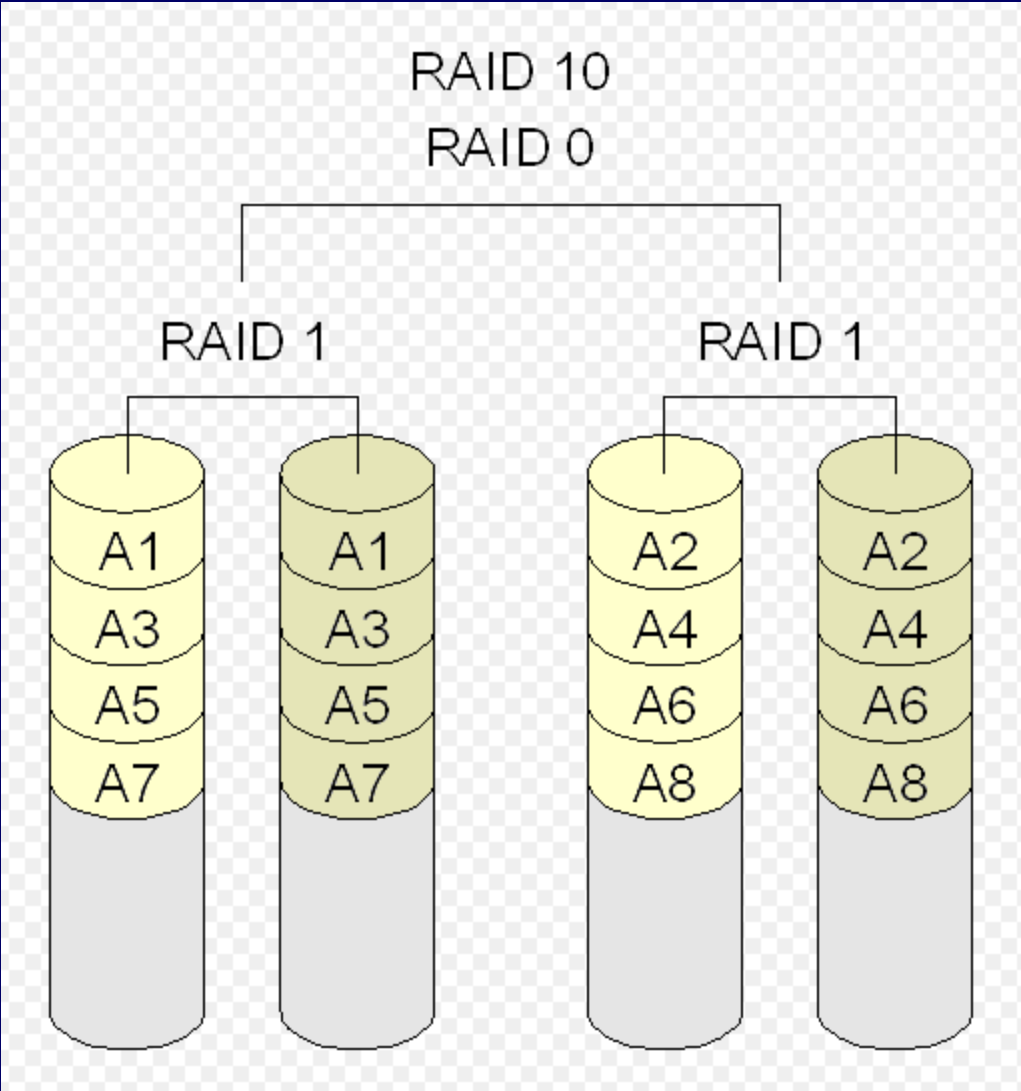
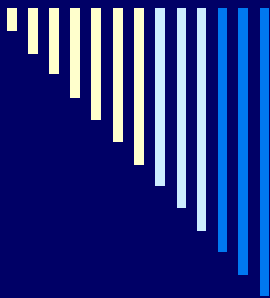




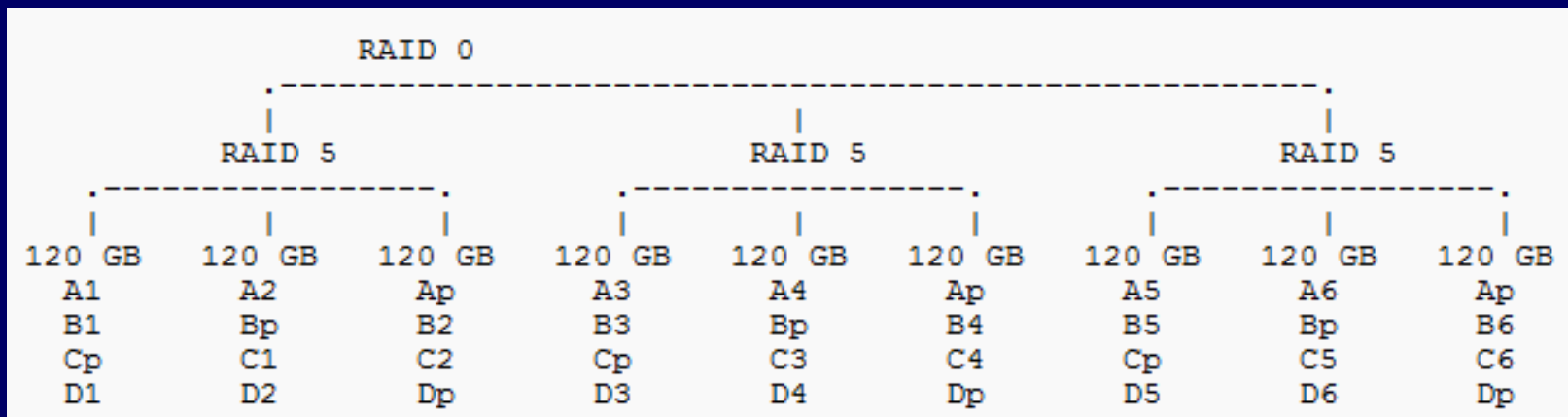
RAID 10

- ❑ kombinace RAID 0 (stripe) a RAID 1 (zrcadlo)
- ❑ min. počet disků 4
- ❑ režie 100% diskové kapacity navíc
- ❑ nejvyšší výkon v bezpečných typech polích
- ❑ podstatně rychlejší než RAID 5, při zápisu
- ❑ odolnost proti ztrátě až 50% disků x RAID 5





RAID 50



Zdroj obrázků a doporučená literatura:

<http://cs.wikipedia.org/wiki/Raid>

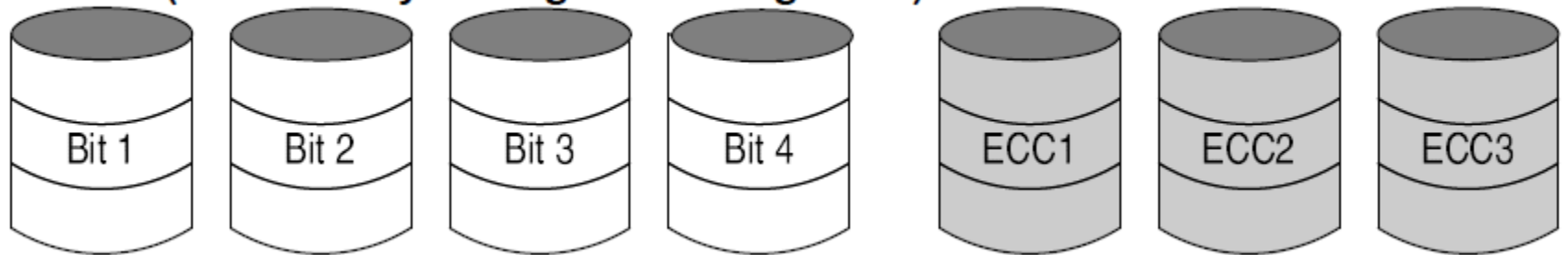


RAID 2

- Data **po bitech** stripována mezi jednotlivé disky
- Rotačně synchronizované disky
- Zabezpečení Hammingovým kódem
- Např. 7 disků
 - 4 bity datové
 - 3 bity Hammingův kód

RAID 2

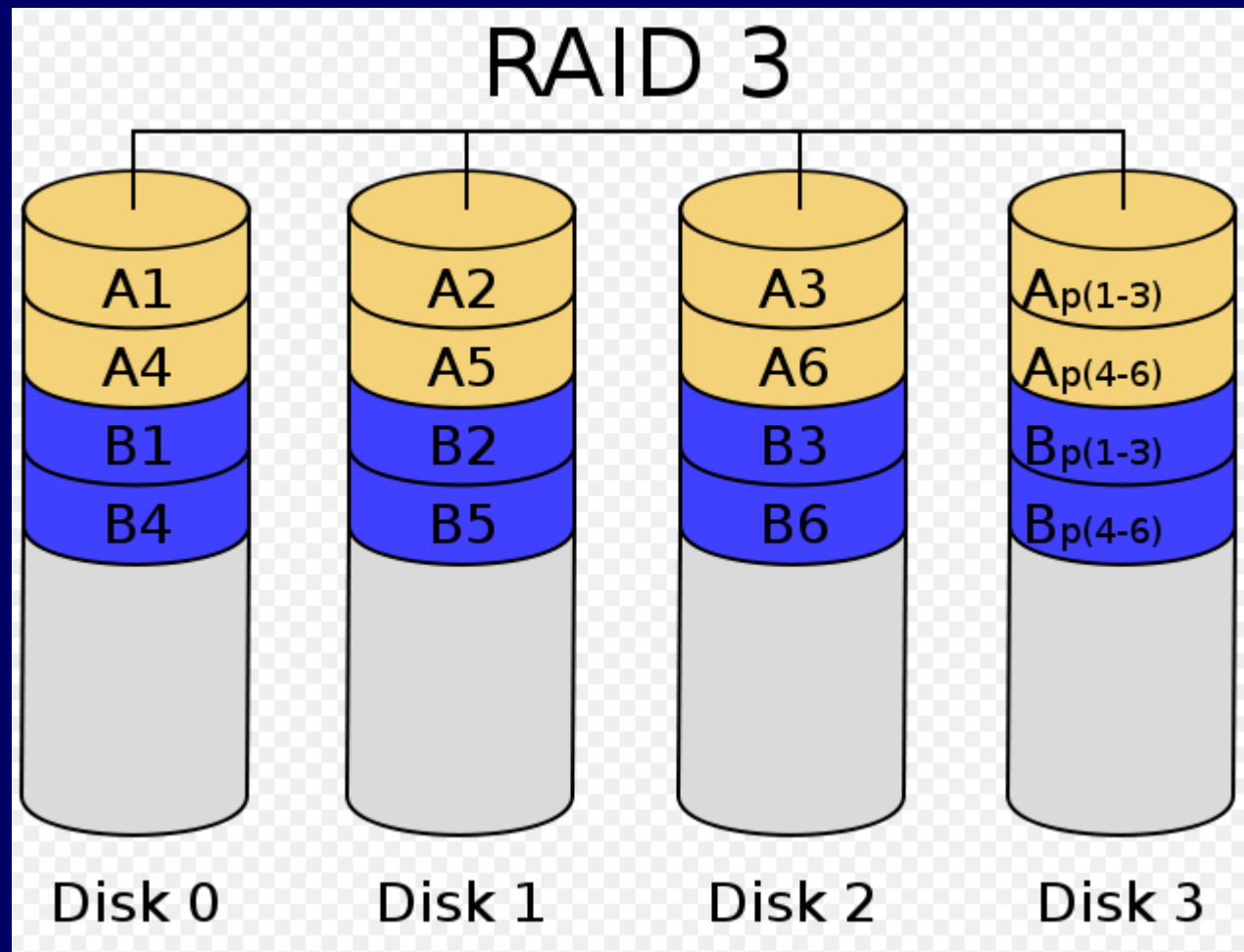
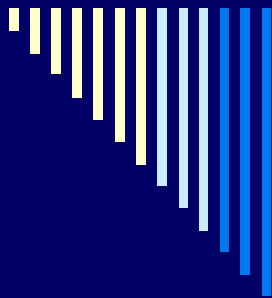
RAID 2 (redundancy through hamming code)





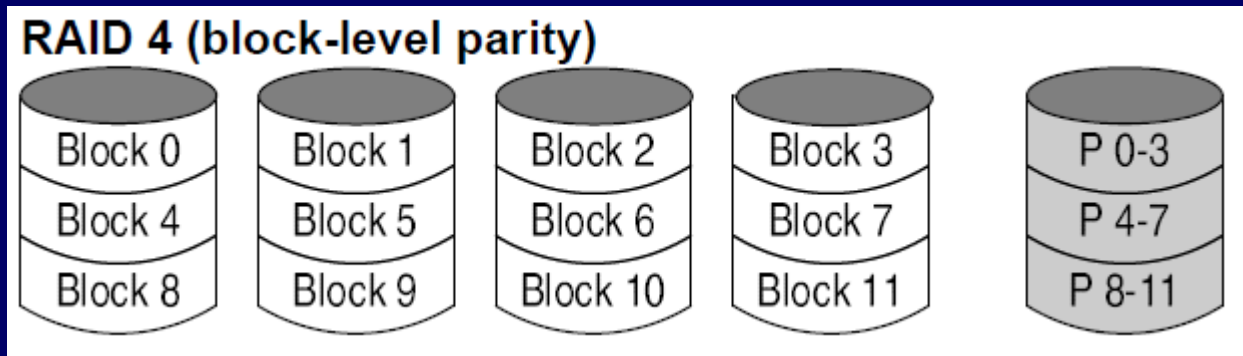
RAID 3

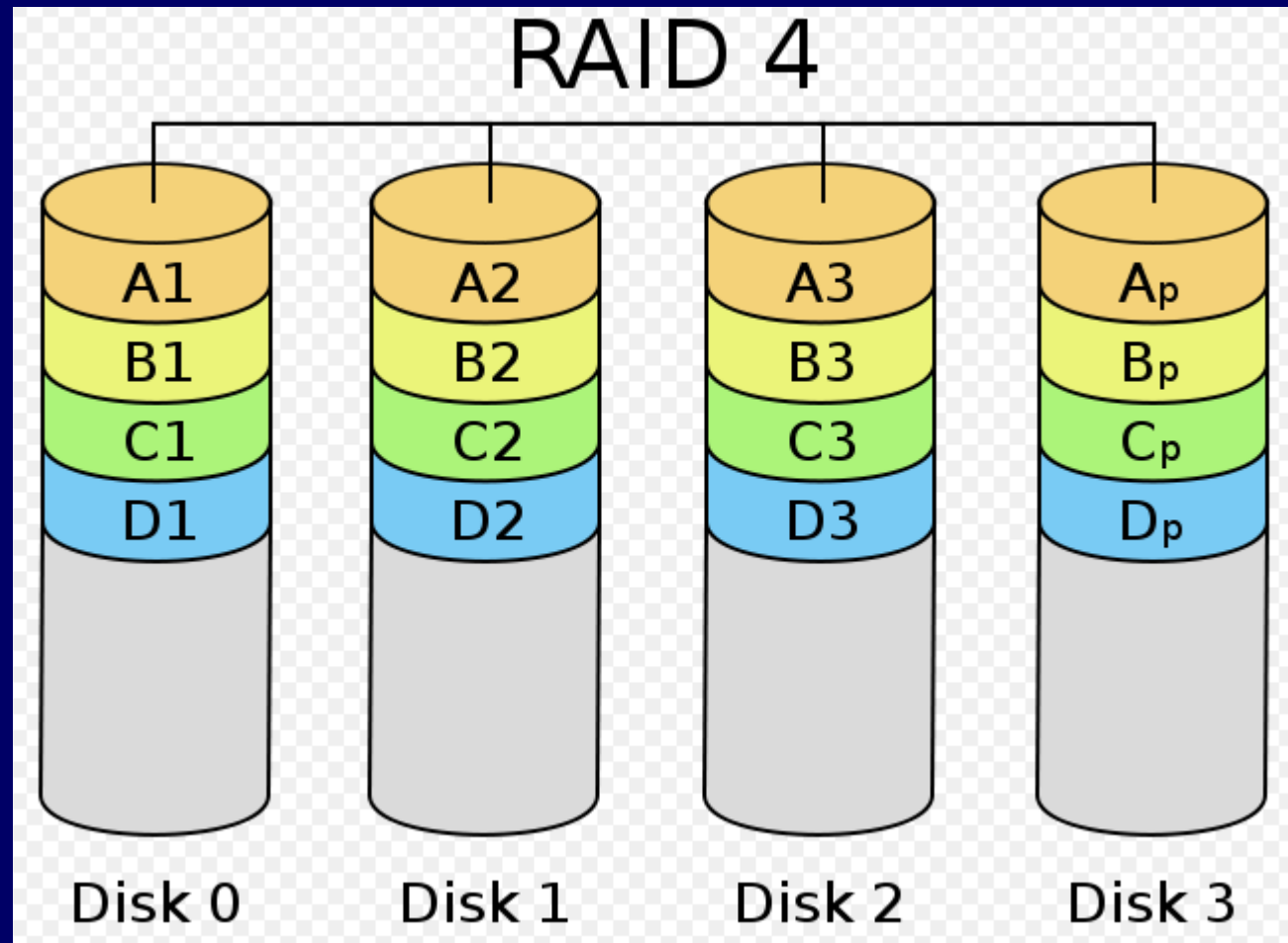
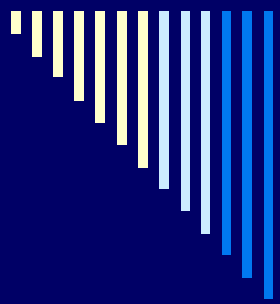
- N+1 disků, bitové prokládání
- Rotačně synchronizované disky
- Na N data, poslední disk XOR parita
- Jen 1 disk navíc
- Paritní disk vytížen při zápisu na libovolný disk – vyšší opotřebení



RAID 4

- Disky stripovány po blocích, ne po bitech
- Parita je opět po blocích







Problém rekonstrukce pole

- rekonstrukce pole při výpadku – trvá dlouho
 - po dobu rekonstrukce není pole chráněno proti výpadku dalšího disku
 - náročná činnost – může se objevit další chyba, řadič disk odpojí a ... přijdeme o data...

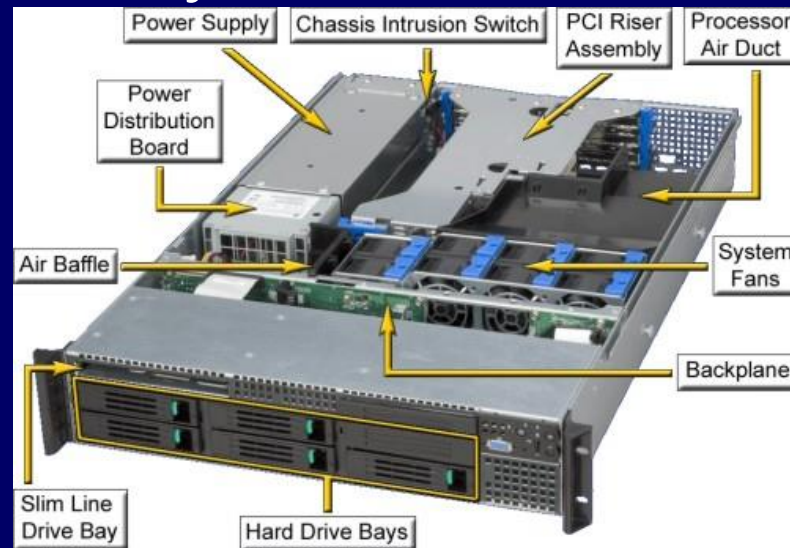


HOT SPARE DISK

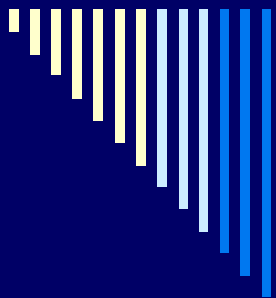
- záložní disk okamžitě připravený k nahrazení vadného disku
 - při výpadku disku v poli automaticky aktivován hot spare disk a dopočítána data
 - minimalizace rizika (časové okno)
 - Pole je degradované a je třeba vyměnit disk
 - Administrátor nemusí být poblíž
 - hot spare disk lze sdílet mezi více RAIDy (někdy)
-

HOT PLUG

- ❑ Snadná výměna disku za běhu systému
- ❑ Není třeba vypnout server pro výměnu disku
- ❑ „šuplík z přední strany serveru“







Ukládání dat

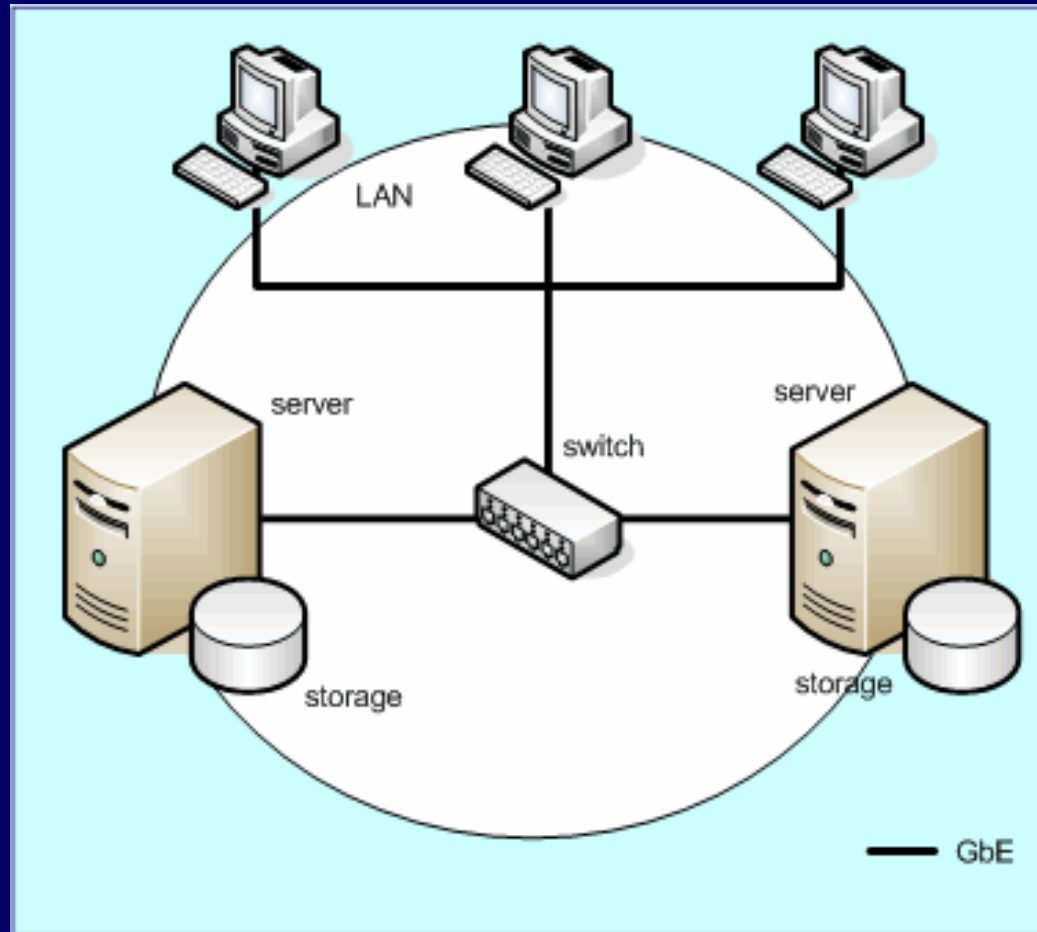
- DAS
- SAN
- iSCSI

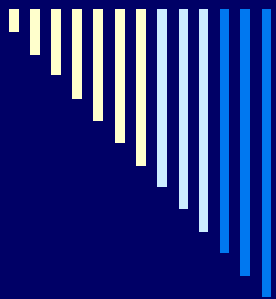


DAS

- Directly attached storage
 - ukládací zařízení přímo u serveru
 - nevýhody
 - porucha serveru – data nedostupná
 - některé servery prázdné, jiné – dat.prostor skoro plný
 - rozšiřitelnost diskové kapacity
 - disky přímo v serveru
 - externí diskové pole přes SCSI (vedle serveru)
-

DAS

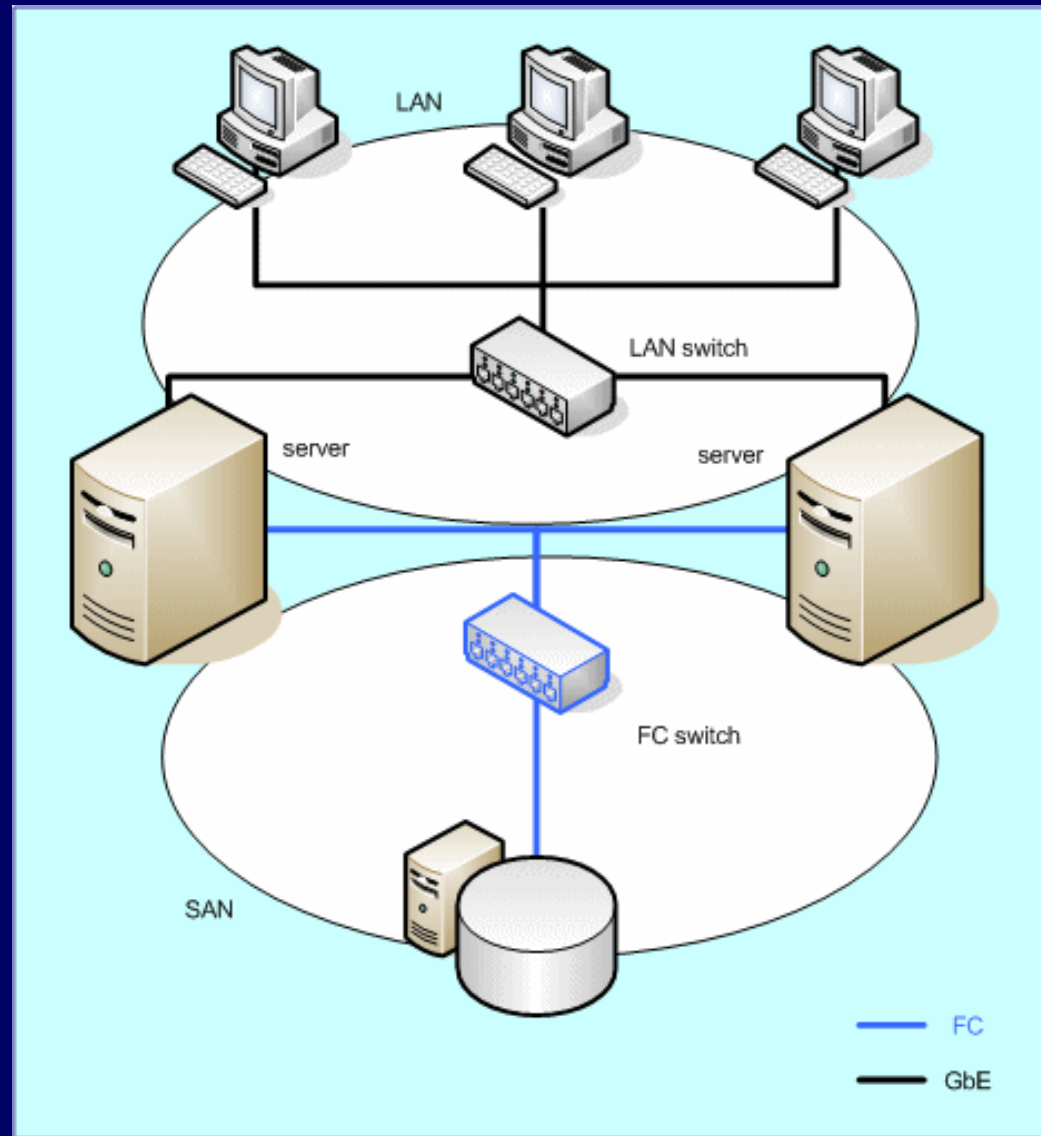




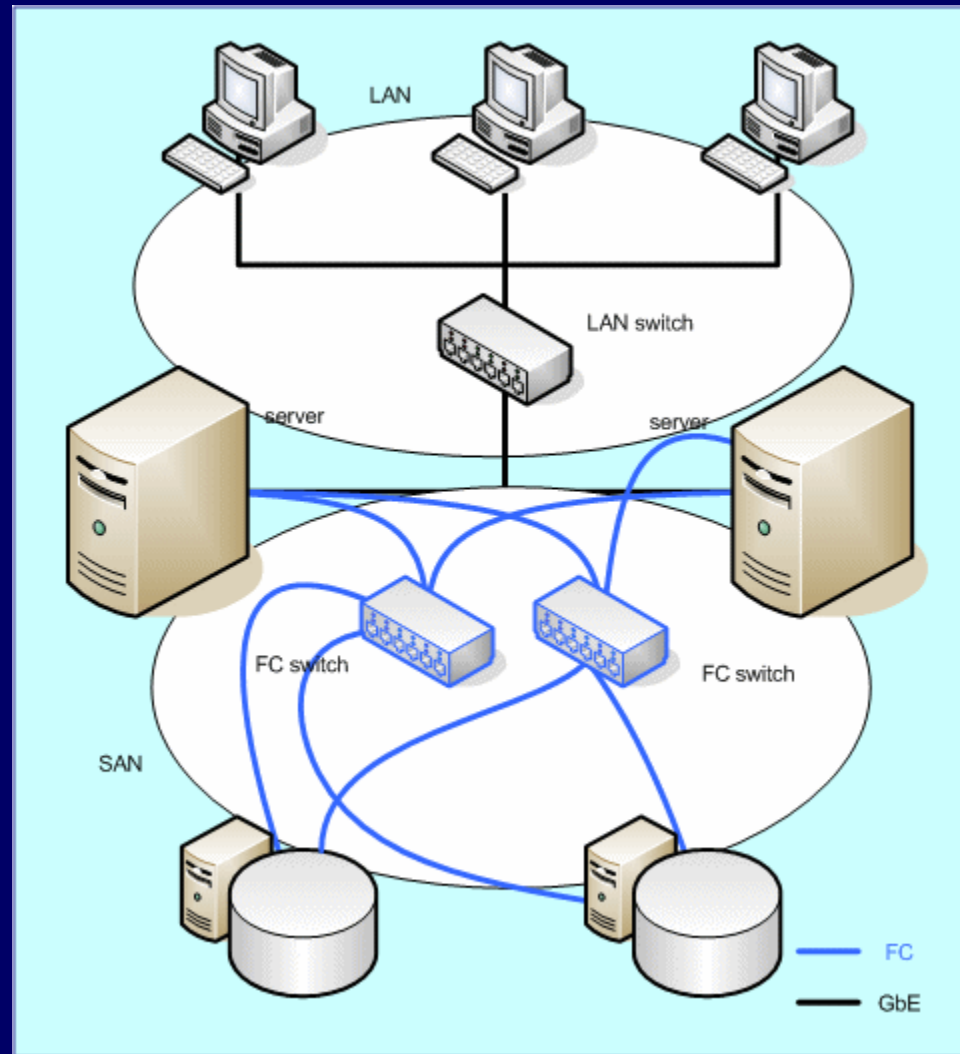
SAN

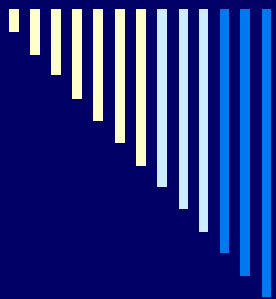
- ❑ Storage Area Network
 - ❑ oddělení storage a serverů
 - ❑ Fibre Channel – propojení, optický kabel
-
- ❑ např. clustery, společná datová oblast
 - ❑ high availability solution

SAN



SAN



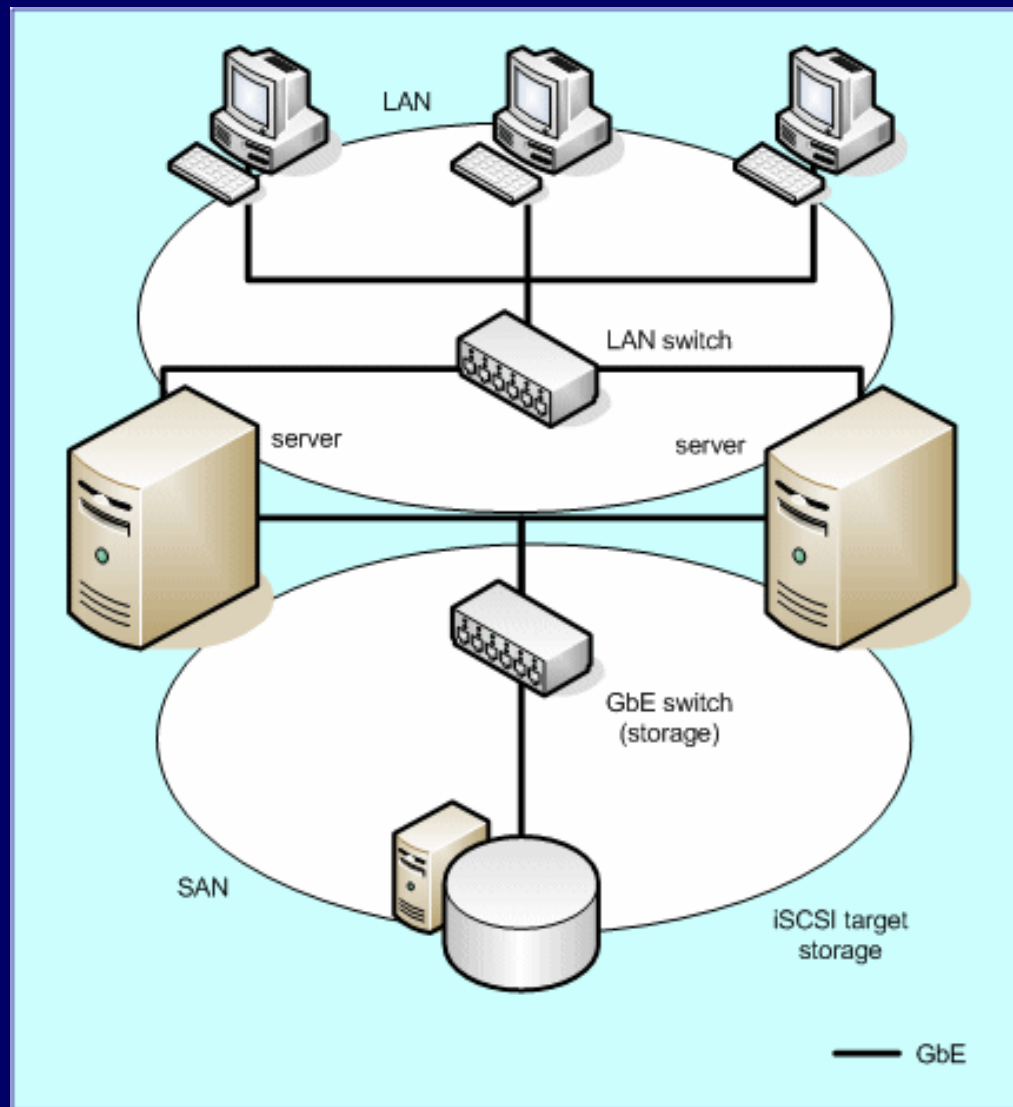


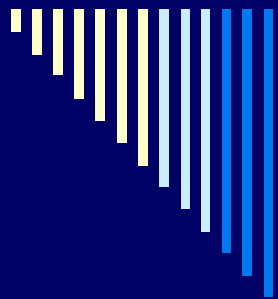
iSCSI

- SCSI + TCP/IP
- SCSI
 - protokol, bez fyzické vrstvy (kabely, konektory)
 - zapouzdření do protokolů TCP/IP
- gigabitový Ethernet vs. drahý Fibre Channel

- SCSI – adaptér, disk
- iSCSI – initiator (adapter), target (cílové zař. disk/pole)

iSCSI





Použité odkazy a další informace

- <http://www.vahal.cz/cz/podpora/technicke-okenko/ukladani-dat-iscsi.html>

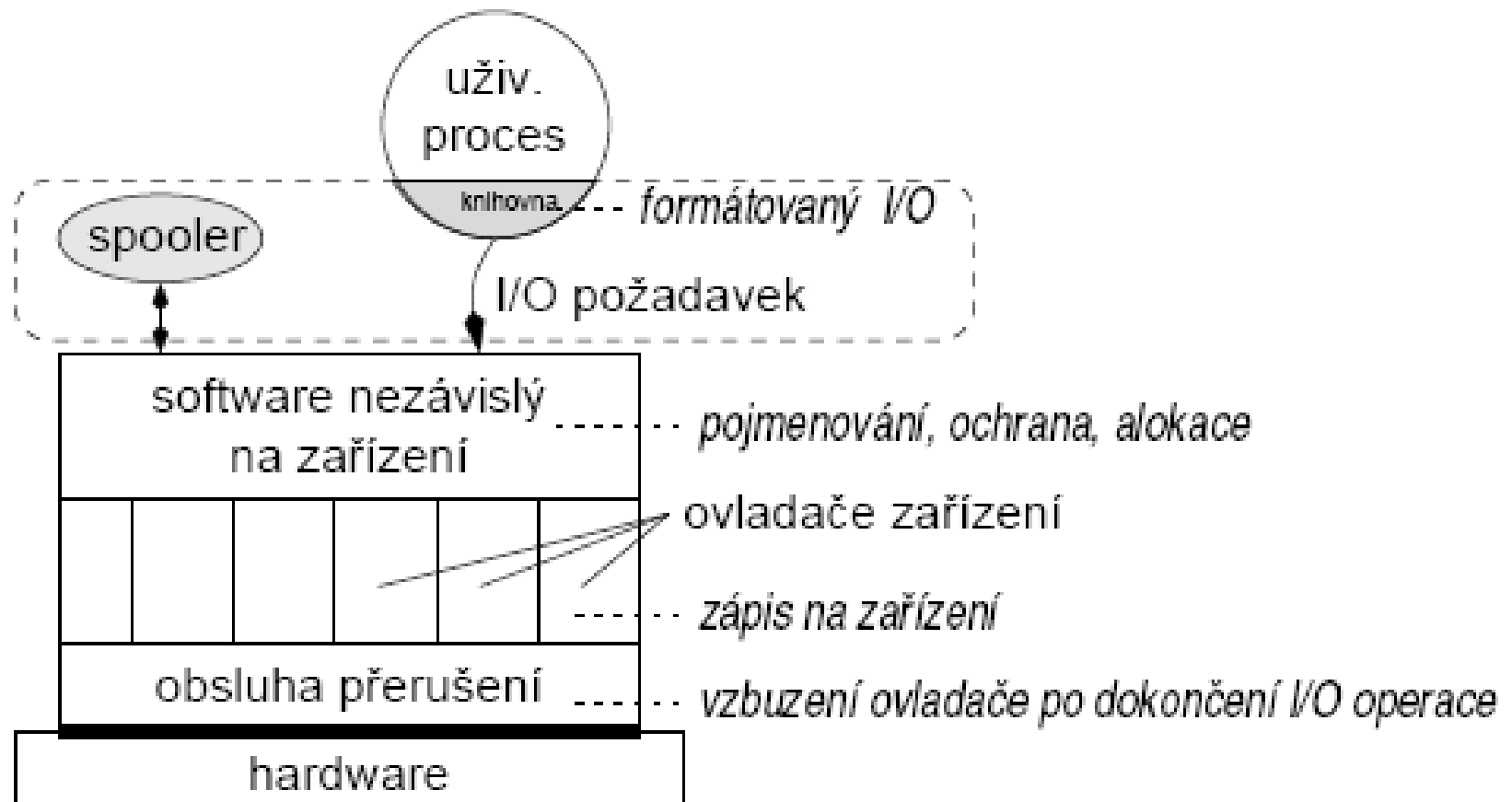


Principy I/O software (!!!!)

typicky strukturován do 4 úrovní:

1. obsluha přerušení (nejnižší úroveň v OS)
2. ovladač zařízení
3. SW vrstva OS nezávislá na zařízení
4. uživatelský I/O SW

Toto je potřeba znát !!





1. Obsluha přerušení

- ❑ řadič vyvolá přerušení ve chvíli **dokončení** I/O požadavku
 - ❑ snaha, aby se přerušením nemusely zabývat vyšší vrstvy
 - ❑ ovladač zadá I/O požadavek, **usne** - P(sem)
 - ❑ po příchodu přerušení ho obsluha přerušení **vzbudí** - V(sem)
 - ❑ časově kritická obsluha přerušení – co nejkratší
-



2. Ovladače zařízení

- obsahují veškerý kód závislý na konkrétním I/O zařízení (např. zvukovka od daného výrobce)
 - ovladač zná jediný hw podrobnosti
 - způsob komunikace s řadičem zařízení
 - zná detaily – např. ví o sektorech a stopách na disku, pohybech diskového raménka, start & stop motoru
 - může ovládat všechna zařízení daného druhu nebo třídu příbuzných zařízení
 - např. ovladač SCSI disků – všechny SCSI disky
-



Funkce ovladače zařízení

- ovladači předán příkaz od vyšší vrstvy
 - např. zapiš data do bloku n
 - nový požadavek **zařazen do fronty**
 - může ještě obsluhovat předchozí
 - ovladač zadá **příkazy řadiči** (požadavek přijde na řadu)
 - např. nastavení hlavy, přečtení sektoru
 - **zablokuje se do vykonání** požadavku
 - neblokuje při rychlých operacích – např. zápis do registru
 - **vzbuzení** obsluhou přerušení (dokončení operace) – zkontroluje, zda nenastala chyba
-



Funkce ovladače zařízení – pokrač.

- pokud OK, předá výsledek (**status + data**) vyšší vrstvě
 - status – datová struktura pro hlášení chyb
- Zpracuje další požadavky ve frontě
 - jeden vybere a spustí
- ovladače často vytvářejí výrobci HW
 - dobře definované rozhraní mezi OS a ovladači
- ovladače podobných zařízení – stejná rozhraní
 - např. síťové karty, zvukové karty, ...

Problémy s ovladači

- Chyba ovladače – pád systému
 - Běh v privilegovaném režimu (jádře)
 - Chyba v ovladači může způsobit pád systému

- Ovladač pro určitý HW i určitý OS
 - Můžete mít starší kameru s ovladačem pro Windows XP, ale třeba nebude použitelná ve Windows 8.1





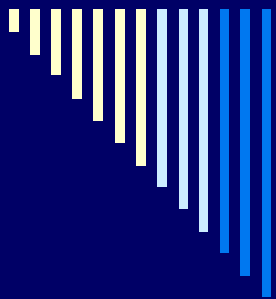
3. SW vrstva OS nezávislá na zařízení

- I/O funkce společné pro všechna zařízení daného druhu
 - např. společné fce pro všechna bloková zařízení
- definuje rozhraní s ovladači
- poskytuje jednotné rozhraní uživatelskému SW
- viz další slide...



Poskytované funkce (!)

- pojmenování zařízení
 - LPT1 x /dev/lp0
 - ochrana zařízení (přístupová práva)
 - alokace a uvolnění vyhrazených zařízení
 - v 1 chvíli použitelná pouze jedním procesem
 - např. tiskárna, plotter, magnetická páska
 - vyrovnávací paměti
 - bloková zařízení – bloky pevné délky
 - pomalá zařízení – čtení / zápis s využitím bufferu
-



Poskytované funkce – pokračování

- hlášení chyb
- jednotná velikost bloku pro bloková zařízení

v moderních OS se zařízení jeví jako
objekty v souborovém systému
(v mnoha OS je tato vrstva součástí logického
souborového systému)



4. I/O sw v uživatelském režimu

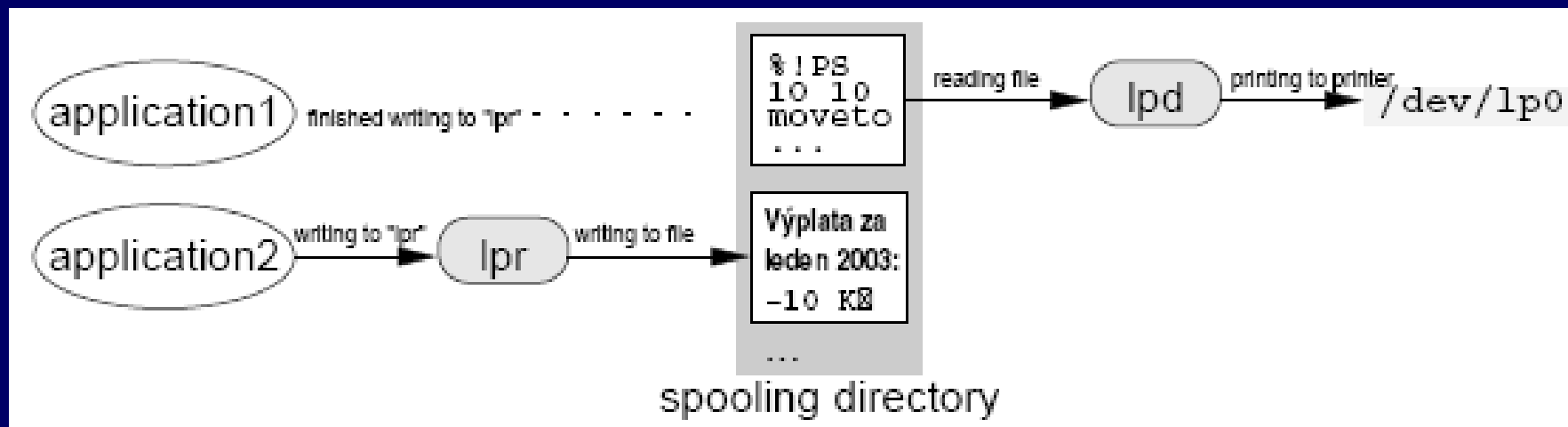
- programátor používá v programech **I/O funkce** nebo **příkazy** jazyka
 - např. printf v C, writeln v Pascalu
 - knihovny sestavené s programem
 - formátování - printf("%.2d:%.2d\n", hodin, minut)
 - často vlastní vyrovnávací paměť na jeden blok
- **spooling**
 - implementován pomocí procesů běžících v uživ. režimu
 - způsob obsluhy vyhrazených I/O zařízení (multiprogram.)
 - např. proces by alokoval zařízení a pak hodinu nic nedělal



Příklad spoolingu – tisk Unix

- k fyzické tiskárně přístup – pouze 1 speciální proces
 - daemon **lpd**
- proces vygeneruje celý soubor, lpd ho vytiskne
 - proces chce tisknout, spustí **lpr** a naváže s ním komunikaci
 - proces předává tisknutá data programu lpr
 - lpr zapíše data do souboru v určeném adresáři
 - spooling directory – přístup jen lpr a lpd
 - dokončení zápisu – lpr oznámí lpd, že soubor je připraven k vytisknutí, lpd soubor vytiskne a zruší

tisk Unix



lpd – démon (služba) čte ze spoolovacího adresáře a přistupuje k tiskárně

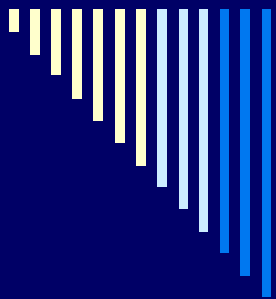
lpr – data, která chce aplikace vytisknout se zapisují do spoolovacího adresáře

Poznámka - spooling lze použít např. i pro přenos elektronické pošty



Souborové systémy

- potřeba aplikací **trvale** uchovávat data
- **hlavní požadavky**
 - možnost uložit velké množství dat
 - 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

- OS pro přístup k médiím poskytuje abstrakci od fyzických vlastností média – soubor
- soubor = pojmenovaná množina souvisejících informací
- souborový systém (file system, fs)
 - konvence pro ukládání a přístup k souborům
 - datové struktury a algoritmy
 - část OS, poskytuje mechanismus pro ukládání a přístup k datům, implementuje danou konvenci



File systém (fs)

- Současné OS – implementují více fs
 - kompatibilita (starší verze, ostatní OS)
 - Windows XP, Vista, 7, 8, 8.1, 10:
 - základní je NTFS
 - ostatní: FAT12, FAT16, FAT32, ISO 9660 (CD-ROM)
 - Linux
 - ext2, ext3, ext4, ReiserFS, JFS, XFS
 - 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. dva:

Linux: **/dev/sda, /dev/sdb**

Každý disk se může dělit na **několik oddílů**:

/dev/sda1, /dev/sda2, /dev/sda3 (1. disk)

/dev/sdb1, /dev/sdb2 (2. disk)

Každý oddíl (partition) – nějaký **filesystem**:

/dev/sda1 ext4

/dev/sda1 swap

/dev/sda1 ntfs

/dev/sdb1 fat32

Formátování disku:
/sbin/mkfs.ext4 /dev/sda1



Základní znalosti

`fdisk /dev/sda`

- ❑ Zobrazení rozdělení disku na oddíly (partitions)
- ❑ Možnost toto rozdělení změnit

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

- ❑ Zformátování oddílu na vybraný filesystem (zde ext4)



Počet oddílů

2 způsoby dělení

- **Master Partition Table (MPT)**

- ▣ Master Boot Record (MBR) – na počátku disku
- ▣ Umožňuje **4 oddíly primární**
- ▣ Chceme-li více, **3 primární a 1 extended**, který lze dělit na další oddíly

- **GUID Partition Table (GPT)**

- ▣ Nelimituje na 4 oddíly, např. Microsoft 124 oddílů
 - ▣ Používá např. Mac OS
-

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	Signatura MBR; 0xAA55 ^[1]	2
01FF	0777	511	AAh		
Celková délka MBR: 446 + 64 + 2 =					512

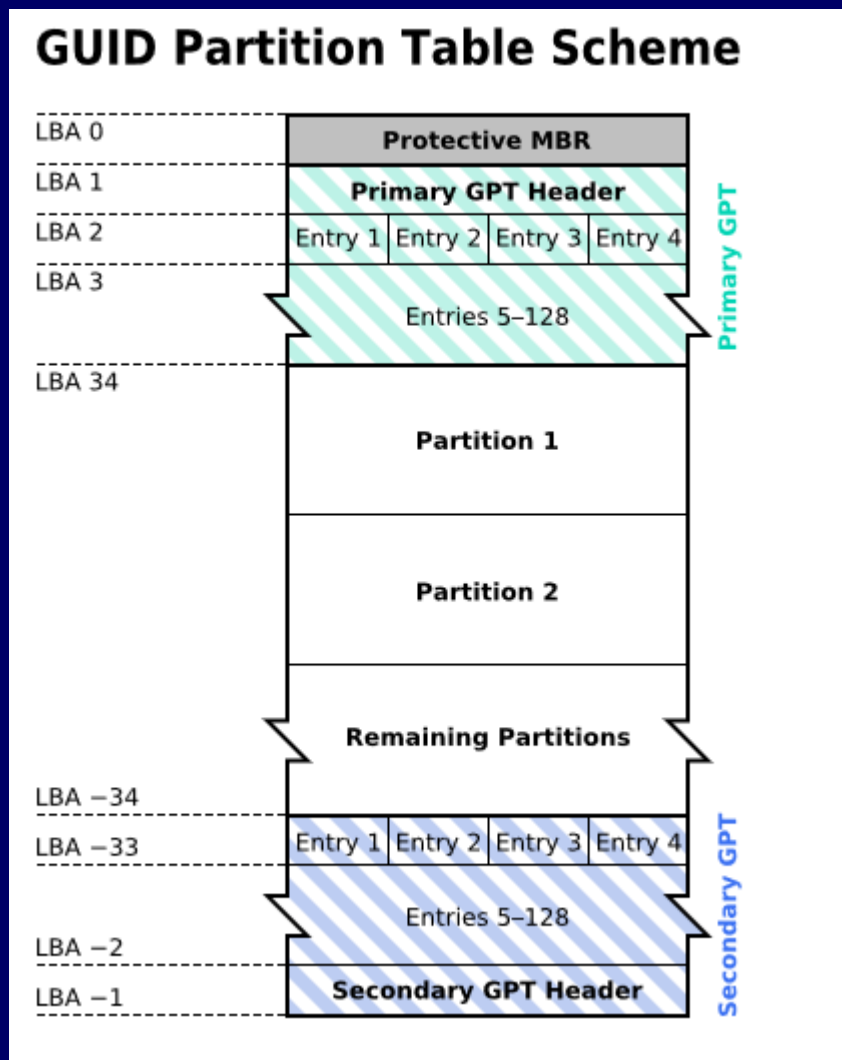
Zdroj obr.: wikipedia

Na začátku disku je MBR record, 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)



Nejsme omezeni na 4 primární oblasti

Součástí standardu **UEFI**, který by měl nahradit klasický BIOS (umožňuje např. secure boot)

Zdroj obr.: wikipedia



Historický vývoj

□ první systémy

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

□ později magnetické pásky

- vstup i výstup – pásky
 - soubor = množina záznamů na magnetické pásce
-



Historický vývoj - pokračování

- nyní – data na magnetických a optických discích
 - ISO 2382-4:1987
 - soubor – pojmenovaná množina záznamů, které lze zpracovávat jako celek
 - záznam – strukturovaný datový objekt tvořený konečným počtem pojmenovaných položek
-



Uživatelské rozhraní fs

- 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 – různé OS
- Windows NT, XP x Unix a Linux
- rozlišuje systém malá a velká písmena?
 - Win32API nerozlišuje: **ahoj**, **Ahoj**, **AHOJ** stejná
 - UNIX rozlišuje: **ahoj**, **Ahoj**, **AHOJ** rozdílná jména



Pojmenování souborů

- jaká může být délka názvu souboru?
 - WinNT 256 znaků NTFS
 - UNIX obvykle alespoň 256 znaků (dle typu fs)
- množina znaků?
 - všechny běžné – názvy písmena a číslice
 - WinNT – znaková sada UNICODE
 - βετα – legální jméno souboru
 - Linux – všechny 8bitové znaky kromě / a char(0)



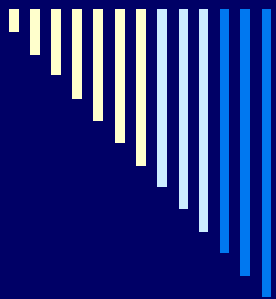
Pojmenování souborů

□ přípony?

- MS DOS – jméno souboru 8 znaků + 3 znaky přípona
- Winows NT a výše, Unix – i více přípon

□ další omezení?

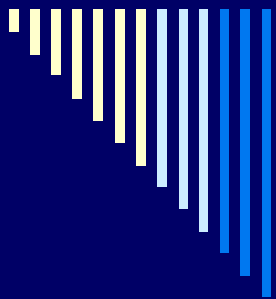
- WinNT – mezera nesmí být první a poslední znak



Typy souborů

OS podporují více typů souborů:

- obyčejné soubory – převážně dále rozebírány
 - data zapsaná aplikacemi
 - obvykle rozlišení textové x binární
 - textové - řádky textu ukončené CR (MAC), LF (UNIX), nebo CR+LF (MS DOS, Windows)
 - binární – všechny ostatní
 - OS rozumí struktuře spustitelných souborů



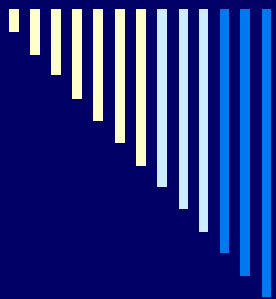
Typy souborů

□ adresáře

- systémové soubory, udržují strukturu fs

□ Linux , UNIX ještě:

- znakové speciální soubory (/dev/tty)
- blokové speciální soubory
 - rozhraní pro I/O zařízení, /dev/lp0 – tiskárna
- pojmenované roury
 - pro komunikaci mezi procesy
- symbolické odkazy



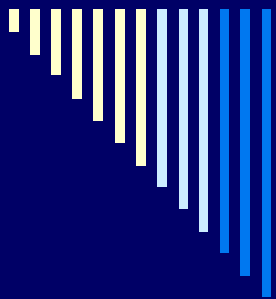
Vnitřní struktura (obyčejného) souboru

□ 3 časté způsoby

- nestrukturovaná posloupnost bytů
- posloupnost záznamů
- strom záznamů

□ nestrukturovaná posloupnost bytů (nejčastěji)

- OS obsah souboru nezajímá, interpretace je na aplikacích
- maximální flexibilita
 - programy mohou strukturovat, jak chtějí



Vnitřní struktura (obyčejného) souboru – pokrač.

- posloupnost záznamů pevné délky
 - každý záznam má vnitřní strukturu
 - operace čtení –vrátí záznam, zápis – změnění / přidá záznam
 - v historických systémech
 - záznamy 80 znaků obsahovaly obraz děrných štítků
 - v současných systémech se téměř nepoužívá



Vnitřní struktura (obyčejného) souboru – pokrač.

□ strom záznamů

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



Způsob přístupu k souboru

□ **sekvenční** přístup

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



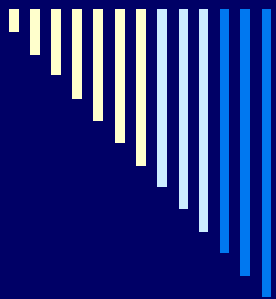
Způsob přístupu k souboru

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



Způsob přístupu k souboru

- v některých OS pro mainframy
 - při vytvoření souboru se určilo, zda je sekvenční nebo s přímým přístupem
 - OS mohl používat rozdílné strategie uložení souboru
- všechny současné OS – soubory s přímým přístupem



Atributy

- informace sdružená se souborem
- některé atributy interpretuje OS, jiné systémové programy a aplikace
- významně se liší mezi jednotlivými OS

- ochrana souboru
 - kdo je vlastníkem, množina přístupových práv, heslo, ...



Atributy - pokračování

□ příznaky

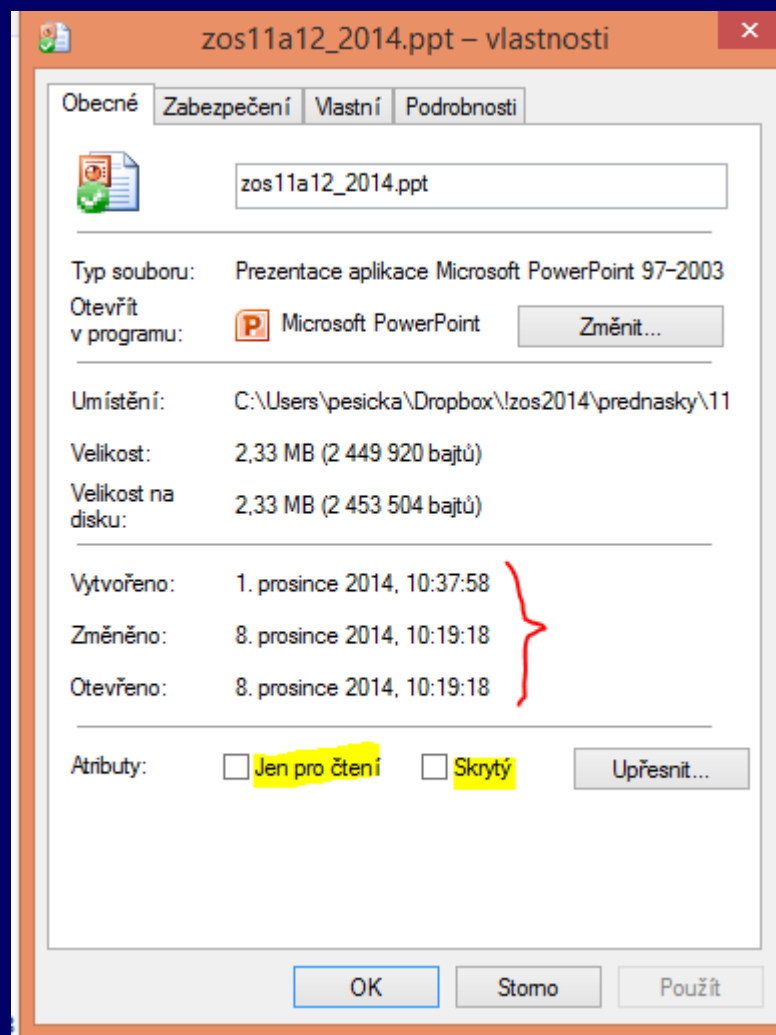
- určují vlastnosti souboru
- hidden – neobjeví se při výpisu
- archive – soubor nebyl zálohován
- temporary – soubor bude automaticky zrušen
- read-only, text/binary, random access

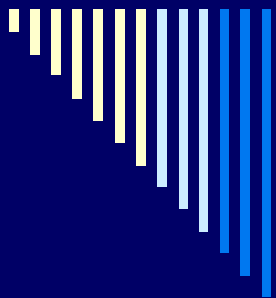
□ přístup k záznamu pomocí klíče

- délka záznamu, pozice a délka klíče

□ velikost, datum vytvoření, poslední modifikace, poslední přístup

Atributy





Služby OS pro práci se soubory

- většina současných – základní model dle UNIXu
- základní filozofie UNIXu – méně je někdy více



Několik jednoduchých pravidel

- **veškerý I/O prováděn pouze pomocí souborů**
 - obvyčejné soubory – data, spustitelné programy
 - zařízení – disky, tiskárny
 - se všemi typy zacházení pomocí **stejných** služeb systému
 - obvyčejný soubor – uspořádaná posloupnost bytů
 - význam znají pouze programy, které s ním pracují
 - interní struktura souboru OS nezajímá
 - jeden typ souboru – seznam souborů – **adresář**
 - adresář je také soubor
 - soubory a adresáře koncepčně umístěny v adresáři
-



Jednotný přístup nebyl vždy

- speciální soubory – pro přístup k zařízením
 - DOS – PRN:, COM1:
 - Poznámka – před příchodem UNIXu toto samozřejmé nebylo
 - většina systémů před UNIXem – samostatné služby pro čtení / zápis terminálu, na tiskárnu do souboru
 - mnoho systémů před i po UNIXu – mnoho různých druhů souborů s různou strukturou a metodami přístupu
-



Poznámky

- systémy poskytovaly „více služeb“ x model podle UNIXu – podstatně menší složitost
- téměř všechny moderní systémy základní rysy modelu převzaly



Základní služby pro práci se soubory

□ otevření souboru

- než s ním začneme pracovat
- úspěšné – vrátí služba pro otevření souboru – **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

- otevření 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 creat nikoliv create



Základní služby pro práci se soubory

□ operace **čtení** ze souboru:

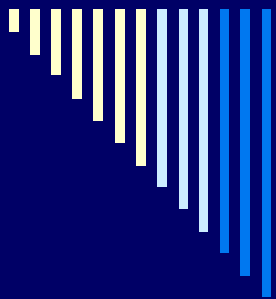
□ **read(fd, buffer, počet_bytů)**

- 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)**
 - význam parametrů jako u read
 - Uprostřed souboru – přepíše, 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
-



Příklad použití rozhraní – kopírování souboru

```
int src, dst, in;
src = open("puvodni", O_RDONLY);
dst = creat("novy", MODE);
while (1)
{
    in = read(src, buffer, sizeof(buffer));
    if (in == 0)
    {
        close(src);
        close(dst);
        return;
    }
    write(dst, buffer, in);
}
```

/ otevreni zdrojoveho */*
/ vytvoreni ciloveho */*

/ cteme */*
/ konec souboru? */*

/ zavreme soubory */*

/ ukončení */*

/ zapiseme prectena data */*



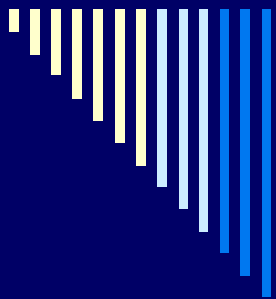
Další služby pro práci se soubory

- 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 512KB
- ❑ $512 * 1024 = 524\,288$.. od této adresy mapujeme
- ❑ 0 až 4KB souboru bude mapováno na 512KB – 516KB
- ❑ čtení z 524 288 čte byte 0 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



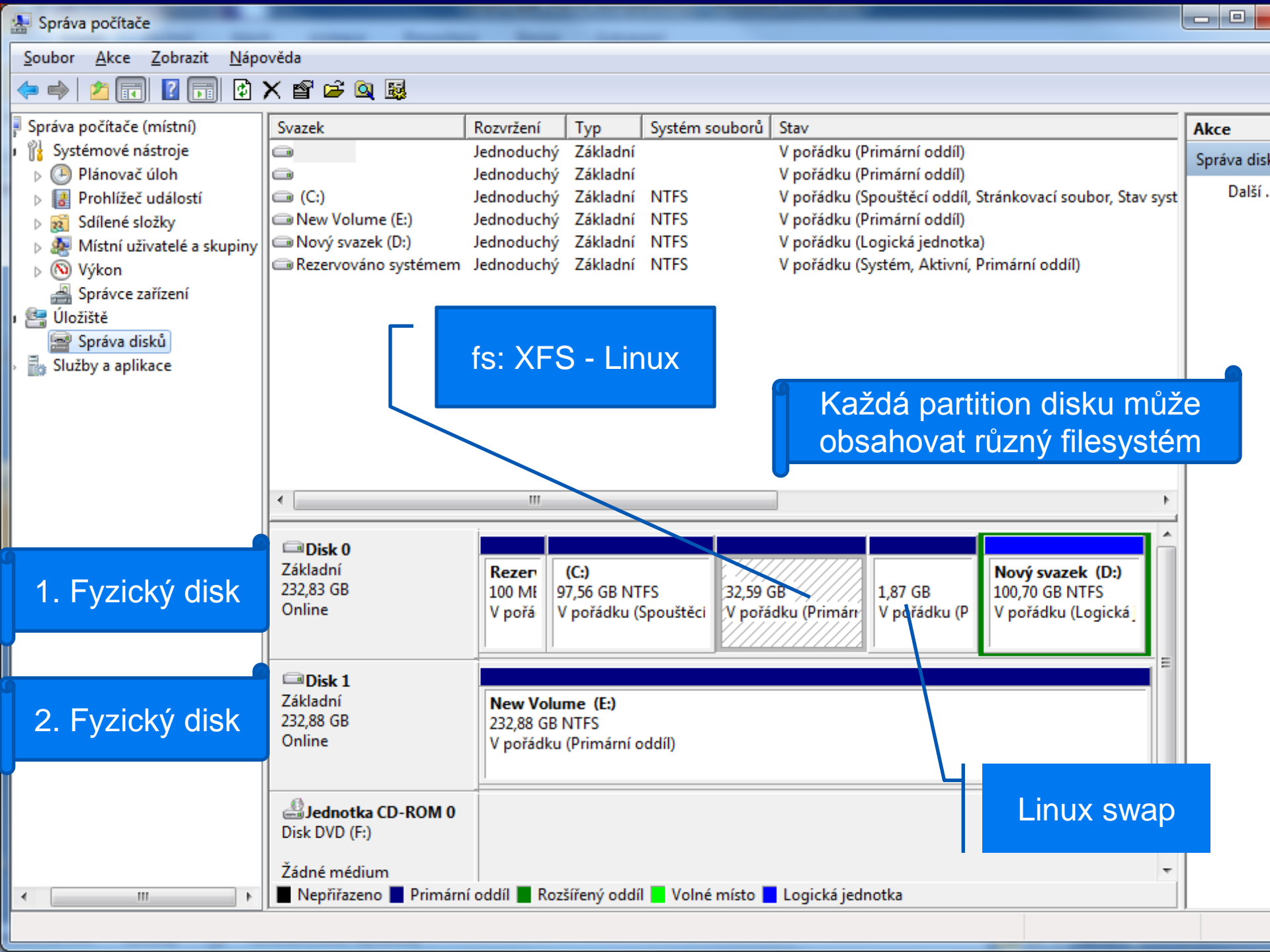
Problémy pam. map. souborů

- není známa přesná velikost souboru, 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 (partition) disku obsahuje jeden fs
- fs – 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)

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 (Primárn	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



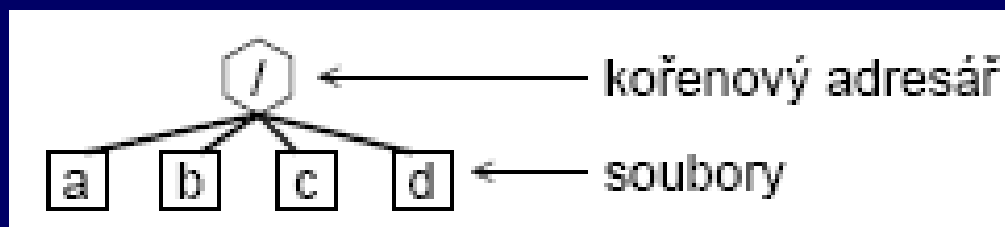
Základní požadavky na adresář

- procházení souborovým systémem (**cd**)
- výpis adresáře (**ls**)
- vytvoření a zrušení souboru (**rm, rmdir**)
- přejmenování souboru (**mv**)

- dále schémata logické struktury adresářů
 - odpovídá historickému vývoji OS

Schémata logické struktury adresářů

- **jednoúrovňový adresář**
- původní verze MS DOSu
- všechny soubory jsou v jediném adresáři
- všechny soubory musejí mít jedinečná jména
- problém zejména pokud více uživatelů



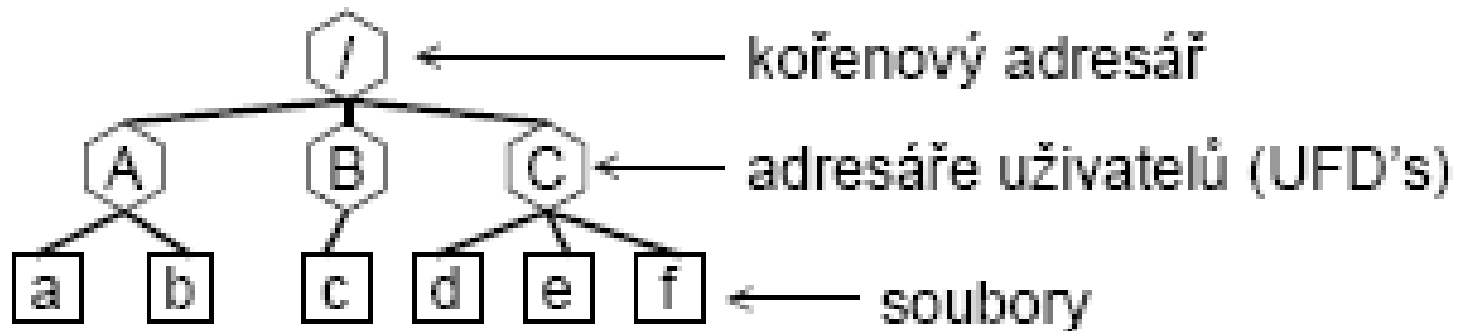
nelze mít dva soubory
příklad.c



Dvouúrovňový adresář

- adresář pro **každého uživatele** (User File Directory, UFD)
- OS prohledává pouze UFD , nebo pokud specifikováno adresář jiného uživatele [user] file
- systémové příkazy – spustitelné soubory – speciální adresář
 - 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
- dnes nejčastější, MS DOS, Windows NT
- adresář – množina souborů a adresářů
- souborový systém začíná kořenovým adresářem „/“
- MS DOS „\“, znak / se používal pro volby
- cesta k souboru – jméno v 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`

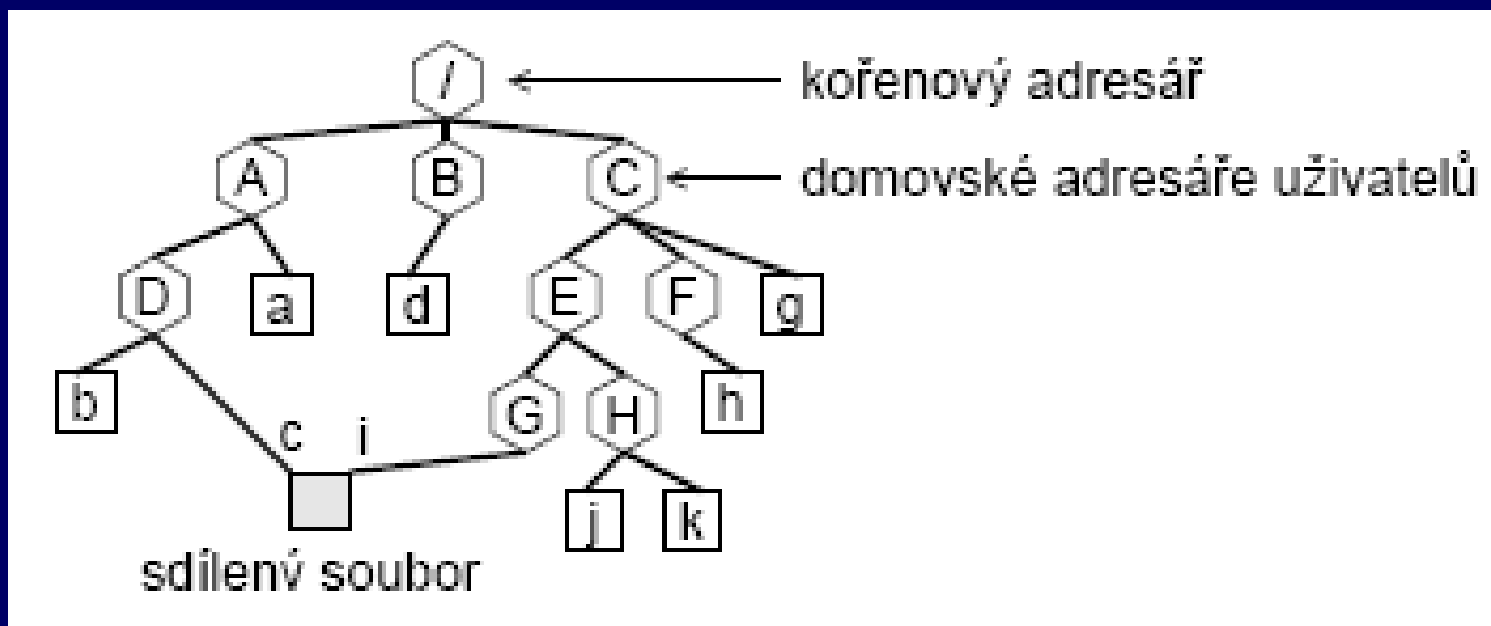


Acyklický graf adresářů

- např. 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 remove sníží o 1, 0 = není odkazován
- jak zajistit aby byl graf acyklický?
 - algoritmy pro zjištění, drahé pro fs



Acyklický graf adresářů



stejný soubor viděný v různých cestách



Obecný 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

- nejčastěji adresářový strom (MS DOS)
- UNIX od původních verzí acyklický graf
hard links – sdílení pouze souborů – nemohou vzniknout cykly

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)

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



Práce s adresářovou strukturou

□ čtení adresářů – UNIX / POSIX

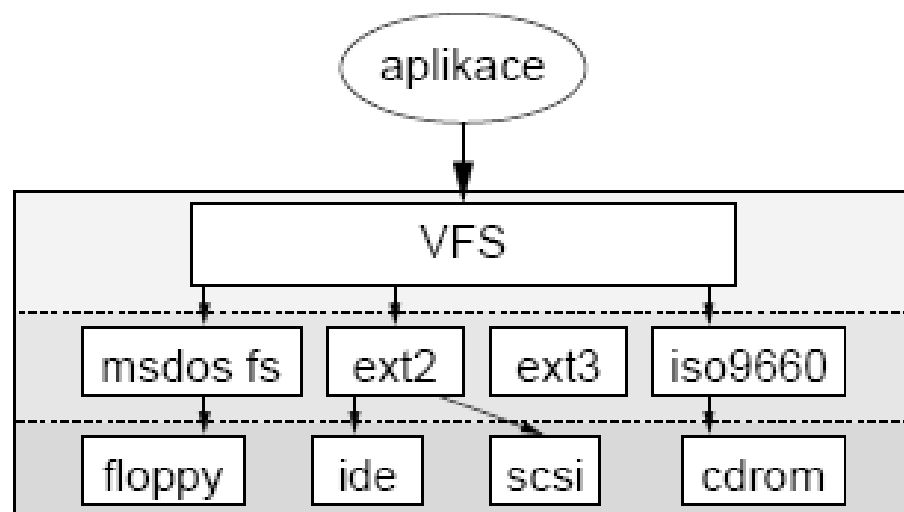
- DIRp = **opendir** (adresář)
 - otevře adresář
- položka = **readdir** (DIRp)
 - čte jednotlivé položky adresáře
- **closedir** (DIRp)
 - zavře adresář
- **stat** (jméno_souboru, statbuf)
 - info o souboru, viz man 2 stat

□ př. DOS: findfirst / findnext

ke všem uvedeným
voláním získáte v
Linuxu
podrobnosti
pomocí:

man 2 opendir
man 2 readdir
man 2 stat

Implementace souborových systémů (!!!)



logický souborový systém

moduly organizace souborů

ovladače zařízení



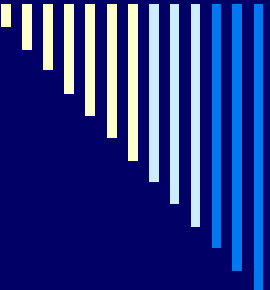
Implementace fs - vrstvy

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
 - Číslovány 0 až N-1
 - Převod čísla bloku na diskovou adresu
 - Volání ovladače pro čtení – zápis bloku
 - **Správa volného** prostoru + **alokace** volných bloků
 - Údržba datových struktur filesystemu
-



Ad 3 – ovladače zařízení

- Nejnižší úroveň
- Floppy, ide, scsi, cdrom
- Interpretují požadavky:
přečti logický blok 6456 ze zařízení 3



Jak VFS pracuje s konkrétním fs? (!)

- 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**

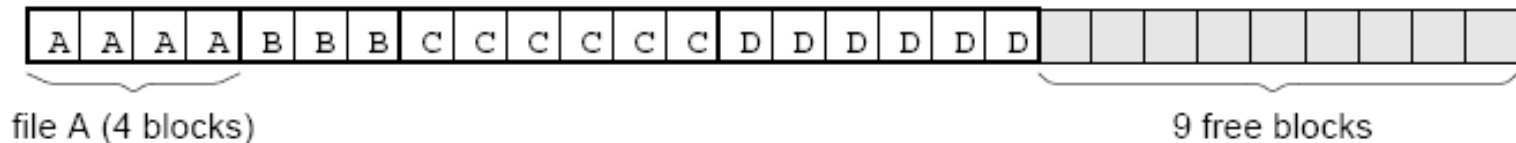


Implementace souborů

- Nejdůležitější:
které diskové bloky patří kterému souboru 😊
- Předpokládáme:
fs je umístěn v nějaké disk partition
bloky v oblasti jsou očíslovány od 0

Kontinuální alokace

- Soubor jako **kontinuální posloupnost** diskových bloků
- Příklad: bloky velikosti 1KB, soubor A (4KB) 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





Kontinuální alokace

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

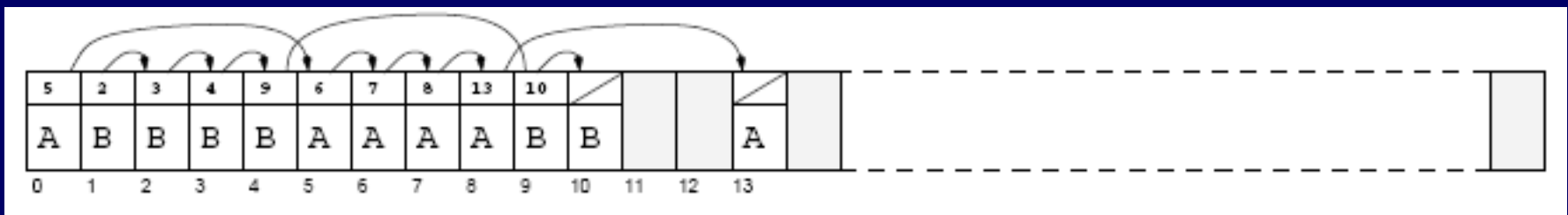


Lze dnes využít kontinuální alokaci?

- Dnes se používá pouze na read-only a write-once médiích
- Např. v ISO 9660 pro CD ROM

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 v adresáři)





Seznam diskových bloků

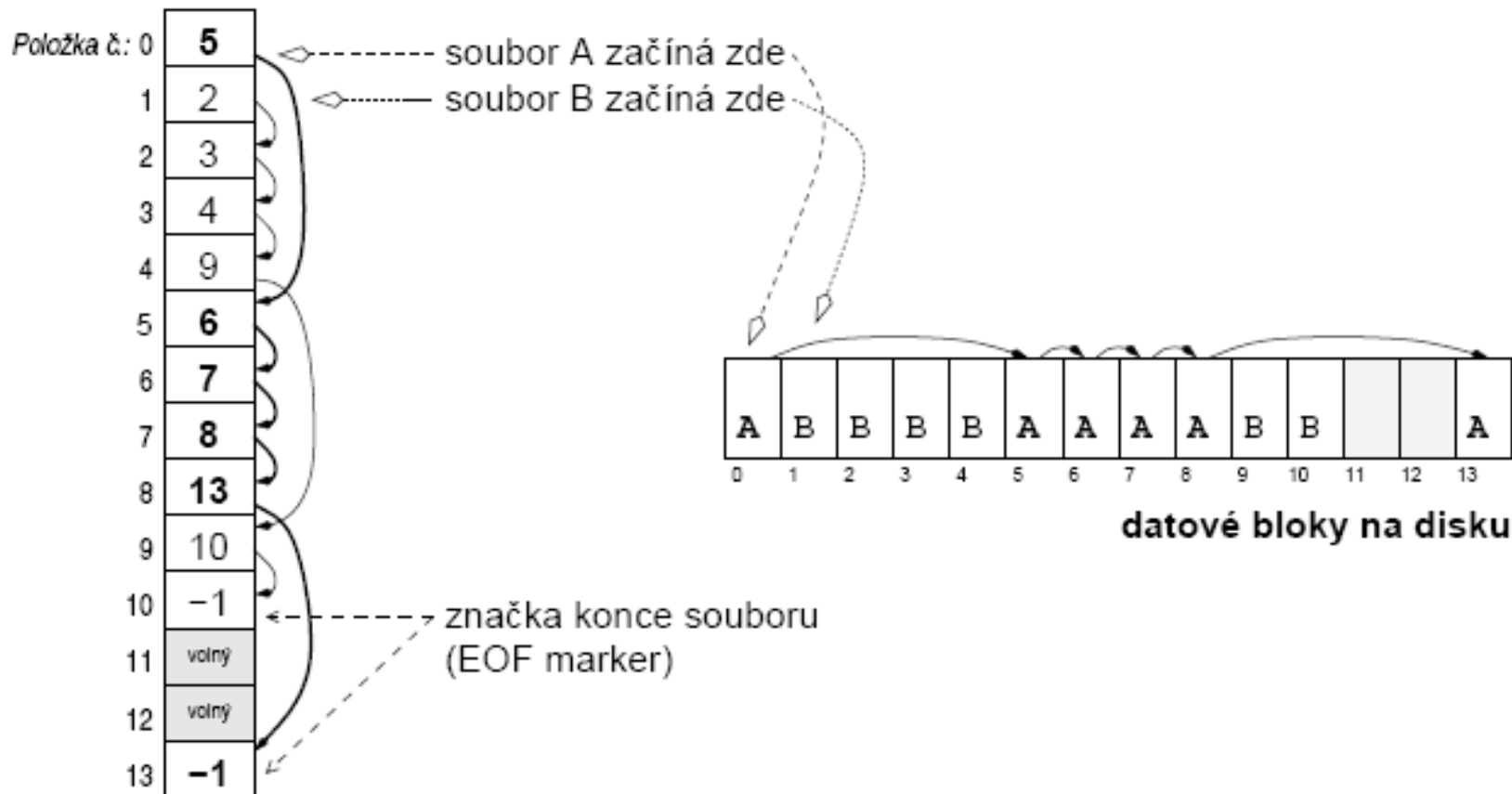
- Sekvenční čtení – bez potíží
- Přímý přístup – simulován sekvenčním, pomalé (musí dojít ke správnému bloku)
- Velikost dat v bloku není mocnina dvou
 - Část bloku je zabraná odkazem na další blok
 - Někdy může být nevýhodou



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

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

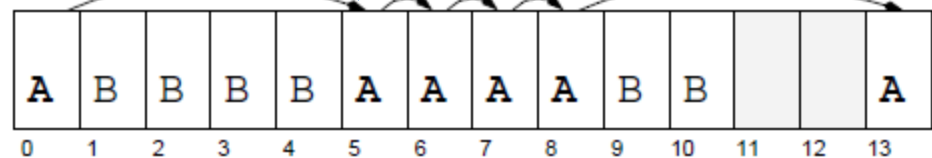
Tabulka FAT

Položka č.: 0	5	
1	2	
2	3	
3	4	
4	9	
5	6	
6	7	
7	8	
8	13	
9	10	
10	-1	značka konce souboru (EOF marker)
11	volný	
12	volný	
13	-1	

Fakt tu je!
Jdem dál na
6

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

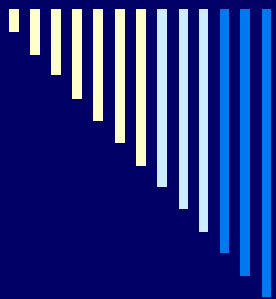
Na indexu 5 je i datový
blok souboru A (kus filmu)



datové bloky na disku

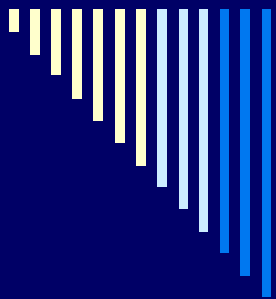
Co je na FAT důležité?

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

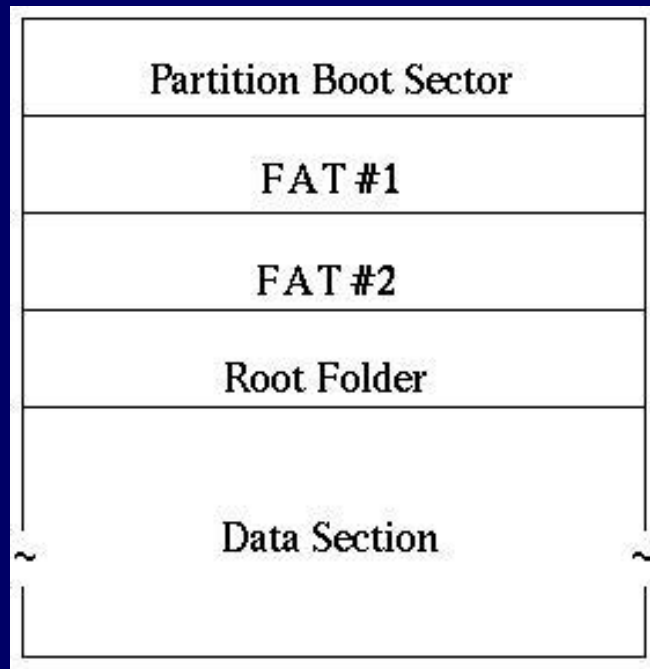
- FAT je ukázka implementace souborového systému, kde v druhé části máme **datové bloky** (obsahující např. části jednoho filmu) a v první části máme **indexy**, které nám říkají, pod jakým číslem se nalézá další odkaz
- Výhodou je, že s určitým souborem můžeme manipulovat, zrušit ho, prodloužit, atd., aniž bychom ovlivnili pozici ostatních souborů na disku



FAT

- máme pevný disk (`/dev/sda`)
- např. druhá oblast disku (`/dev/sda2`)
- tato oblast disku je rozdělena:
 - boot block
 - FAT1 tabulka
 - FAT2 tabulka (záložní kopie)
 - datové bloky obsahující části souborů
 - první je adresář: jména souborů, velikost, kde začíná

Diskový oddíl s FAT



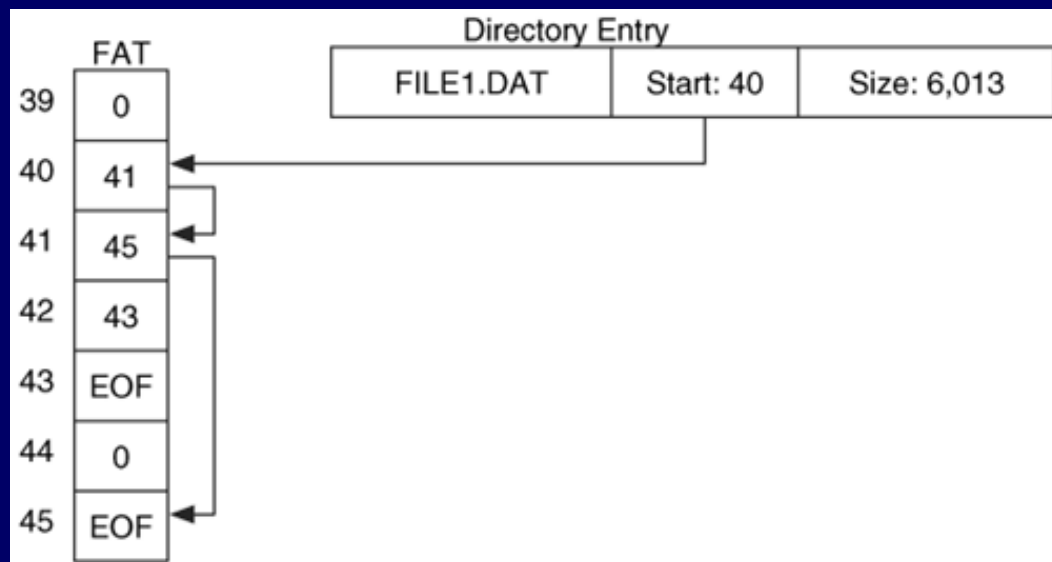
Diskový oddíl, který je naformátován na FAT (např. FAT32)

1. boot sektor (pokud se z daného oddílu bootuje)
2. 2 kopie FAT tabulky
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 adresáře poznáme:

- Jméno souboru
- Kde začíná (40)
- Velikost (6013 bytů)

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

V tomto obrázku by velikost bloku byla cca 2KB – 2048B



Vlastnosti FAT

- Nevýhodou je velikost tabulky FAT
 - 80GB disk, bloky 4KB => 20 mil. položek
 - Každá položka alespoň 3 byty => 60MB FAT
 - Výkonnostní důsledky (část FAT chceme v keši v RAM)
- Použití
 - DOS, Windows 98, ME, podporují Win NT/2000/XP
 - FAT12, 12 bitů, $2^{12} = 4096$ bloků, diskety
 - FAT16, 16 bitů, $2^{16} = 65536$ bloků
 - FAT32, 2^{28} bloků, blok 4-32KB, cca 8TB
 - ExFAT – používá B+ strom místo tabulky



Příklady filesystemů (!!!)

□ **FAT**

- Starší verze Windows, paměťové karty
- Nepoužívá ACL – u souborů není žádná info o přístupových právech
- snadná přenositelnost dat mezi různými OS

□ **NTFS**

- Používá se ve Windows XP/Vista/7
- Používají ACL: k souboru je přiřazen seznam uživatelů, skupin a jaká mají oprávnění k souboru (!!!!)

□ **Ext2**

- Použití v Linuxu, nemá žurnálování
- Nepoužívá ACL – jen standardní nastavení (vlastní, skupina, others), což ale není ACL (to je komplexnější)

□ **Ext3**

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



Příklady filesystemů

□ ext4

- stejně jako ext2, ext3 používá inody
- extenty – souvislé logické bloky
 - může být až 128MB oproti velkému počtu 4KB bloků
- nanosekundová časová razítka

□ xfs

□ jfs



NTFS

- ❑ nativní fs Windows od NT výše
- ❑ **žurnálování**
všechny 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 (x FAT)
- ❑ **komprese**
na úrovni fs lze soubor nastavit jako komprimovaný



NTFS pokračování

- šifrování

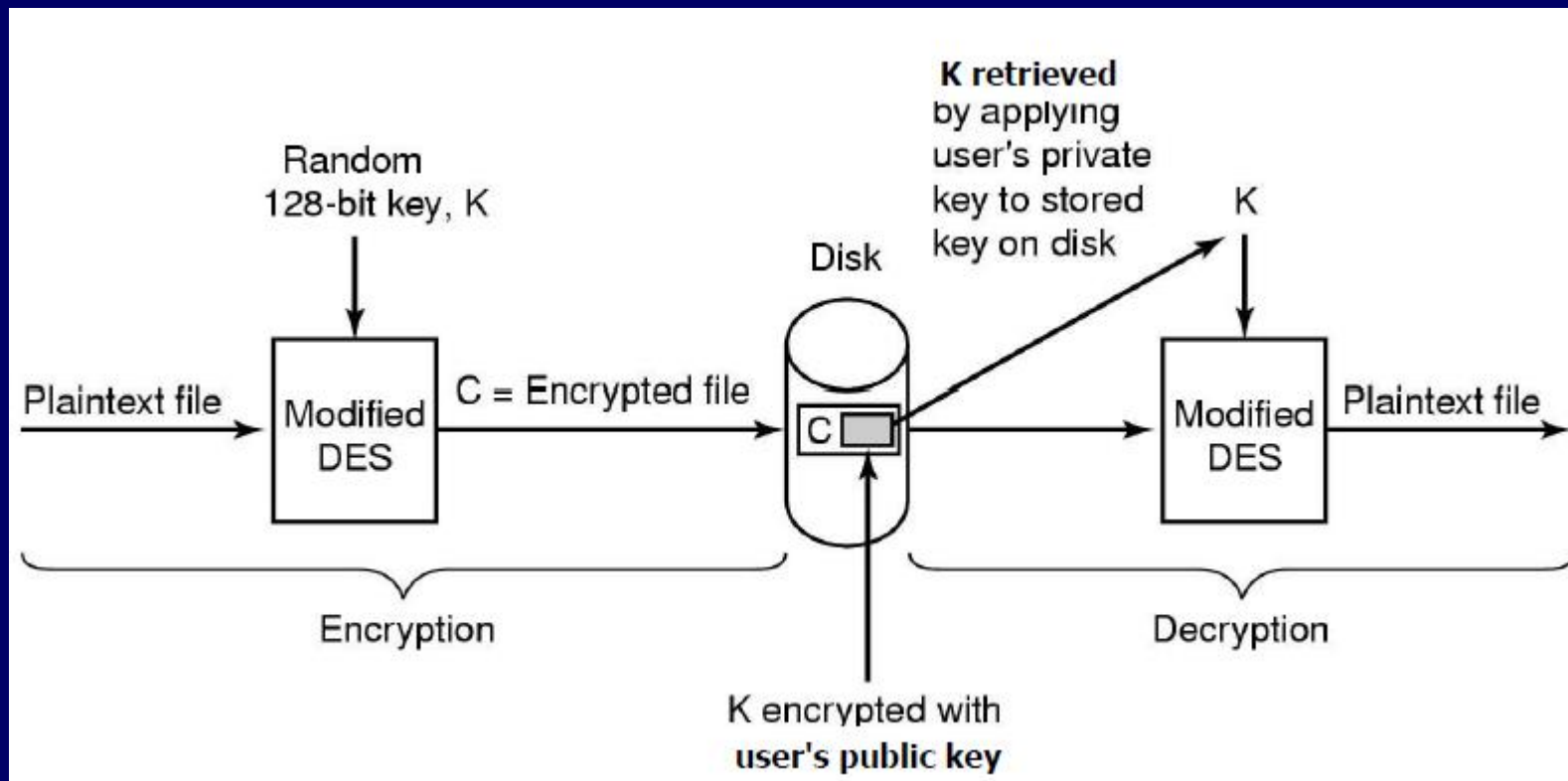
EFS (encrypting file system),
transparentní – otevřu ahoj.txt, nestarám se, zda je
šifrovaný

- 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 klusterů .. cca 16EB
- systém jako obří databáze
záznam odpovídá souboru
- základ 11 systémových souborů - metadat
hned po formátování svazku
- **\$Logfile** – žurnálování
- **\$MFT** (Master File Table)
záznamy o všech souborech, adresářích, metadatech
hned za boot sektorem, za ním se udržuje zóna
volného místa



NTFS struktura

- **\$MFTMirr** – uprostřed disku, obsahuje část záznamů \$MFT, při poškození se použije tato kopie
- **\$Badclus** – seznam vadných clusterů
- \$Bitmap – sledování volného místa
0 – volný
- \$Boot, \$Volume, \$AttrDef, \$Quota, \$Upcase, .

podrobnosti:

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



NTFS atributy souborů

- **\$FILE_NAME**
 - jméno souboru
 - velikost
 - odkaz na nadřazený adresář
 - další
- **\$SECURITY_DESCRIPTOR**
 - přístupová práva k souboru
- **\$DATA**
 - vlastní obsah souboru



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
-

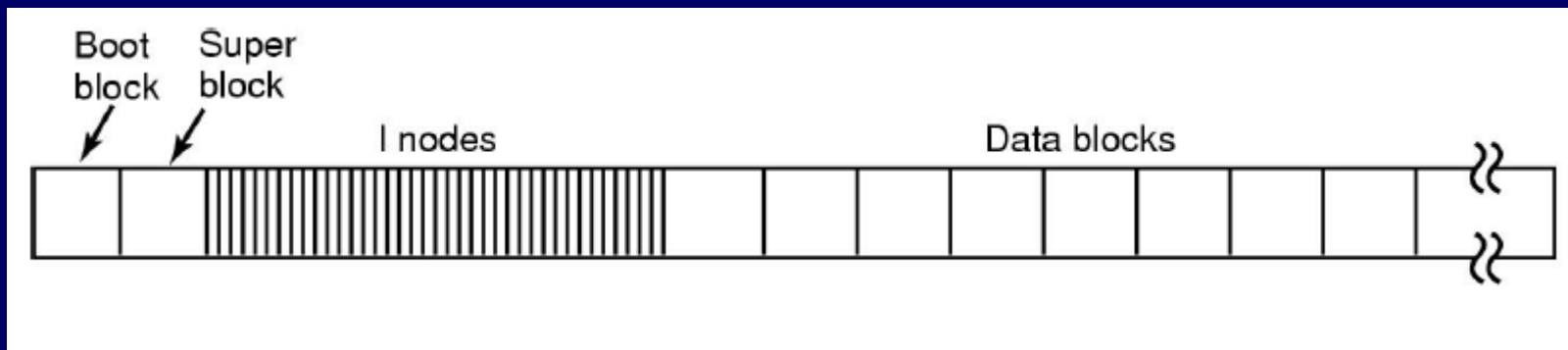


NTFS – způsob uložení dat (!!!)

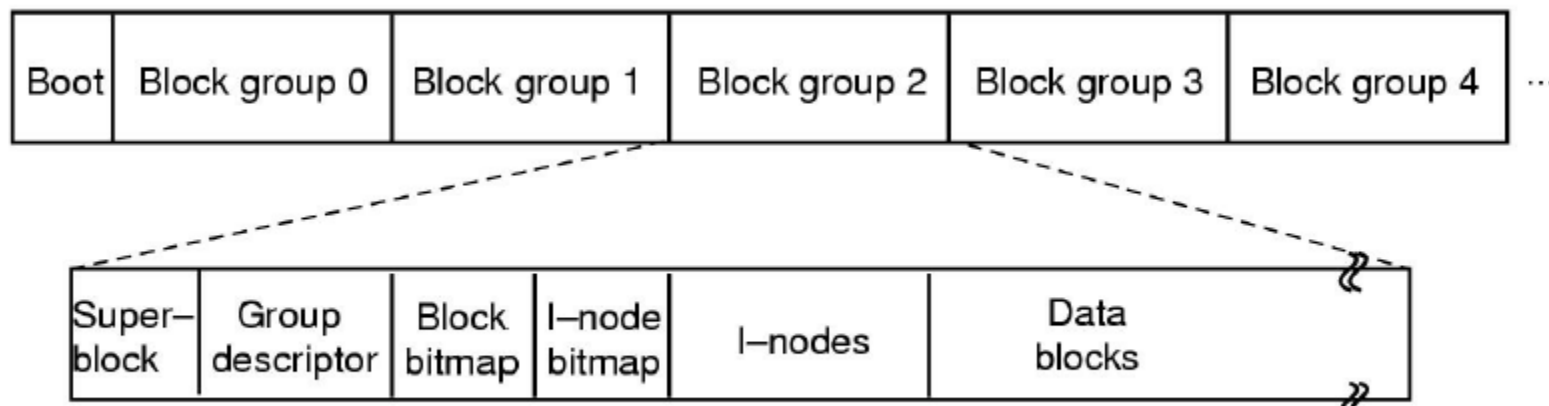
- kódování délkou běhu
- od pozice 0 máme např. uloženo:
A1, A2, A3, B1, B2, A4, A5, C1, ...
- soubor A bude popsán fragmenty
- **fragment**
 - index
 - počet bloků daného souboru
- v našem příkladě:
 - 0, 3 (od indexu 0 patří tři bloky souboru A)
 - 5, 2 (od indexu 5 patří dva bloky souboru A)

Unixové systémy s využitím i-uzlů (původní)

takto vypadá parititon disku (např. `/dev/sda1`)
původní rozdělení v Unixových systémech, v ext2 viz další slide
superblock – příznak čistoty, verze, počet i-nodů, velikost alok. jednotky



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



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

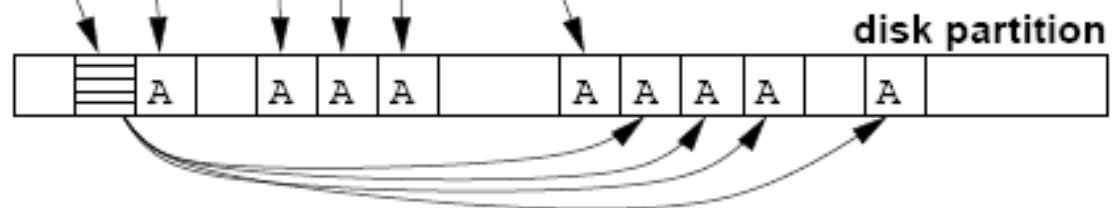


I-uzly (!!)

- S každým souborem sdružena datová struktura i-uzel (i-node, zkratka z index-node)
 - i-uzel obsahuje
 - Atributy souboru
 - Diskové adresy prvních N bloků
 - 1 či více odkazů na diskové bloky obsahující další diskové adresy (případně obsahující odkazy na bloky obsahující adresy)
 - Používá tradiční fs v Unixu UFS (Unix File System) a z něj vycházející v Linuxu, dnes např. ext2, ext3, ext4
-

i-node

attributes
pointer to block 0
pointer to block 1
pointer to block 2
pointer to block 3
...
pointer to block 10
pointer to block of pointers

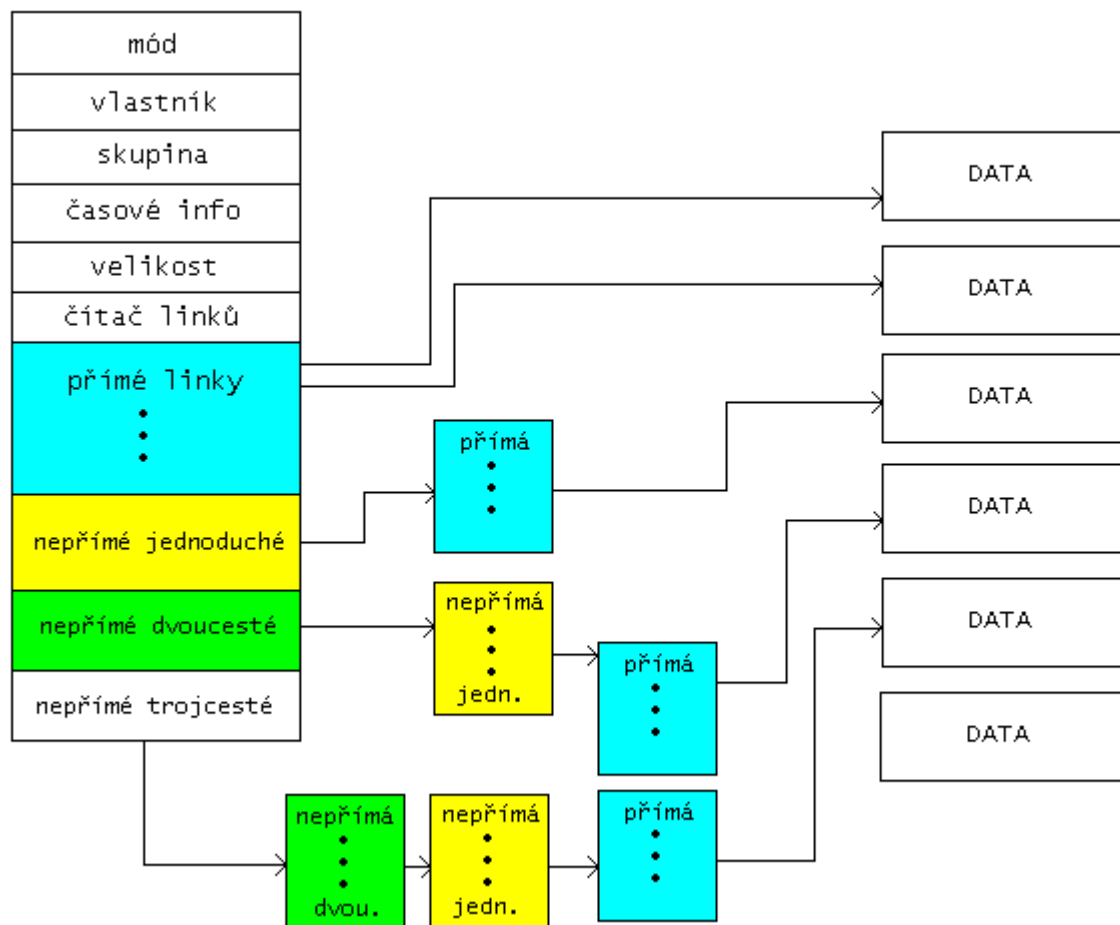


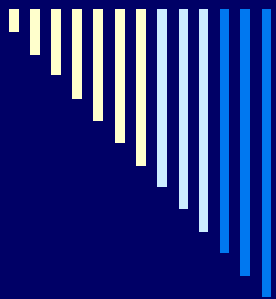


I-uzly - výhoda

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

i-uzly dle normy POSIX





i-uzly dle normy Posix

- ❑ **MODE** – typ souboru, **přístupová práva** (u,g,o)
- ❑ **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

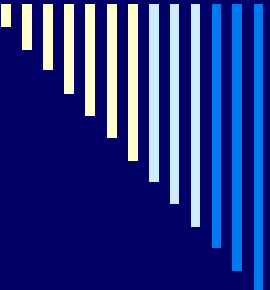
- ❑ **DIRECT BLOCKS** – 12 přímých odkazů na datové bloky (data v souboru)
- ❑ **SINGLE INDIRECT** – 1 odkaz na datový blok, který **místo dat** obsahuje seznam přímých odkazů na datové bloky obsahující vlastní data souboru
- ❑ **DOUBLE INDIRECT** – 1 odkaz 2. nepřímé úrovně
- ❑ **TRIPLE INDIRECT** –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

v linuxových fs (ext*) 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)
(implementuje DOS, Windows)
2. Adresář obsahuje **pouze jméno + odkaz** na jinou datovou strukturu obsahující další informace
(např. i-uzel) (implementuje UNIX, Linux)

Běžné jsou oba dva způsoby i kombinace

2 základní uspořádání adresáře (!!)

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

a)

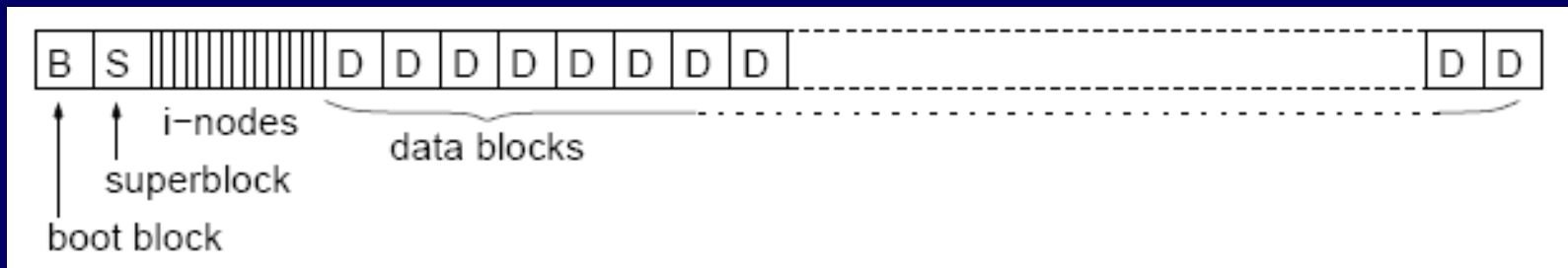
mail			atributy, diskové adresy
prednasky			atributy, diskové adresy
pracovni			atributy, diskové adresy

b)

Příklad fs (Unix v7)

□ Struktura fs na disku

- **Boot blok** – může být kód pro zavedení OS
- **Superblok** – informace o fs (počet i-uzlů, datových bloků,...)
- **i-uzly** – tabulka pevné velikosti, číslovány od 1
- **Datové bloky** – všechny soubory a adresáře



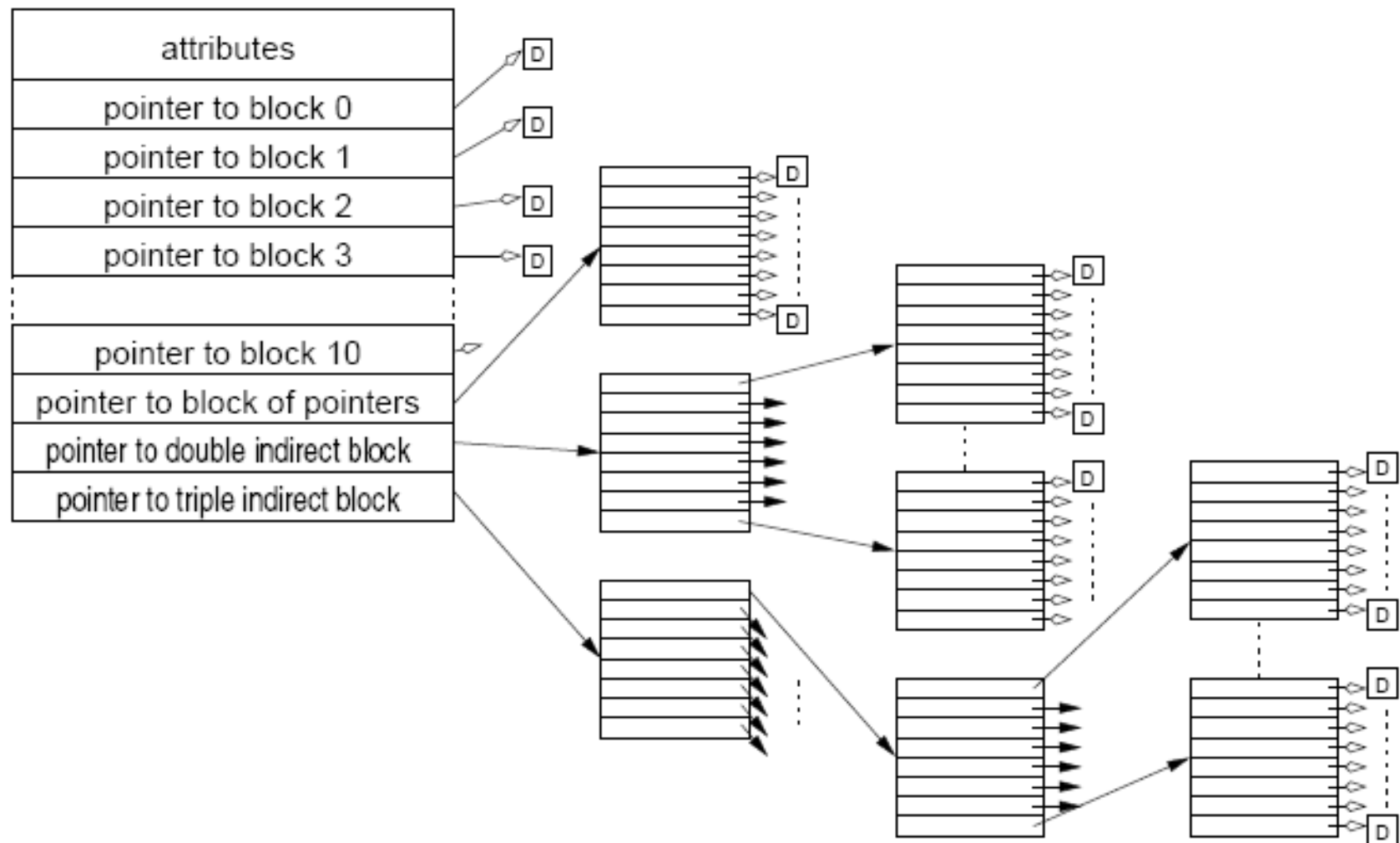


Implementace souborů – i-uzly

i-uzel obsahuje:

- Atributy
 - Odkaz na prvních 10(až 12) datových bloků souboru
 - 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**
-

UNIX v7 i-node





Pokračování příkladu

- Implementace adresářů:
tabulka obsahující jméno souboru a číslo jeho i-uzlu
- Info o volných blocích
seznam, jeho začátek je v superbloku
- Základní model, současné fs jsou na něm založeny

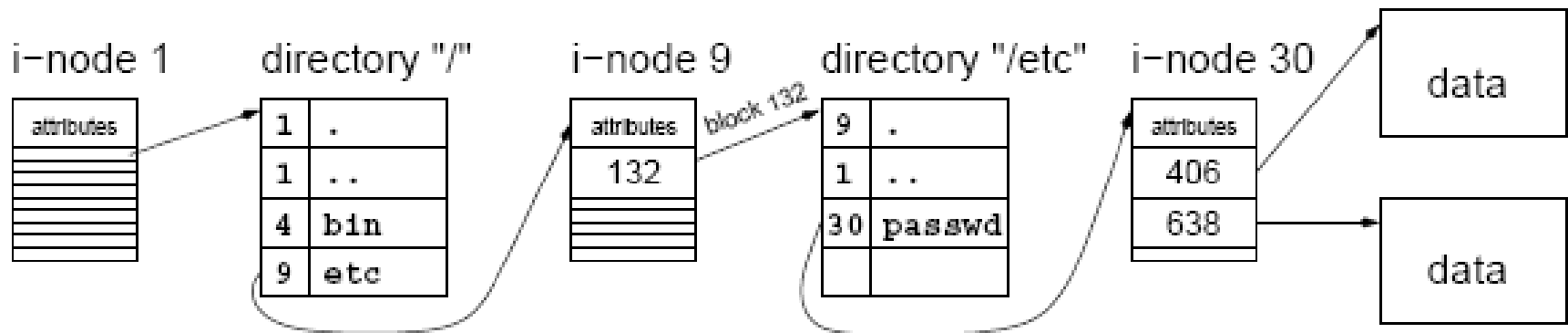


Adresáře v UNIX v7

- 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

□ Nalezení cesty k souboru `/etc/passwd`

- 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)





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



Sdílení souborů

- 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

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



Sdílení souborů

□ 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

- Info, které bloky jsou volné
- Nejčastěji – bitová mapa nebo seznam
- **Bitová mapa**
 - Konstantní velikost
 - Snažší vyhledávání volného bloku s určitými vlastnostmi
 - 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
- Problém najít volný blok s určitými vlastnostmi (např. ve stejném cylindru), prohledávat seznam, drahé,...

Seznam diskových bloků

Jeden blok

Další blok



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
 - Filesystem 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: scandisk, **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 (obvykle) 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 počtu výskytů bloku v souboru
- Tabulka počtu výskytů bloku v seznamu volných bloků
- Položky obou tabulek inicializovány na 0
- Procházíme informace o souborech (např. i-uzly), inkrementujeme položky odpovídající blokům souboru v první tabulce
- Procházíme seznam nebo bitmapu volných bloků a inkrementujeme příslušné položky ve druhé tabulce



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	2	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 tabulce
 - Missing blok
 - Není závažné, pouze redukuje kapacitu fs
 - Oprava: vložení do seznamu volných bloků
- (0,2) – blok je dvakrát nebo vícekrát v seznamu volných
 - Problém – blok by mohl být alokován vícekrát !
 - 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 vyjmeme ze seznamu volných bloků
 - (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 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



Jak funguje žurnál (!!)

1. Zapiši do žurnálu
 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
 - Může dojít ke ztrátě obsahu souboru, ale nerozpadne se struktura adresářů
-



Výkonnost fs

- Přístup k tradičnímu 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, drahé
 - 256GB cca 3-4 tisíce Kč
-



Výkonnost fs, 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, semaforey, ...)
 - **Subjekt** – entita schopná přistupovat k objektům (většinou **proces**)
 - **Objekt** – cokoliv, k čemu je potřeba omezovat přístup pomocí přístupových práv (např. **soubor**)
 - Systém **uchovává informace** o přístupových právech subjektů k objektům
-

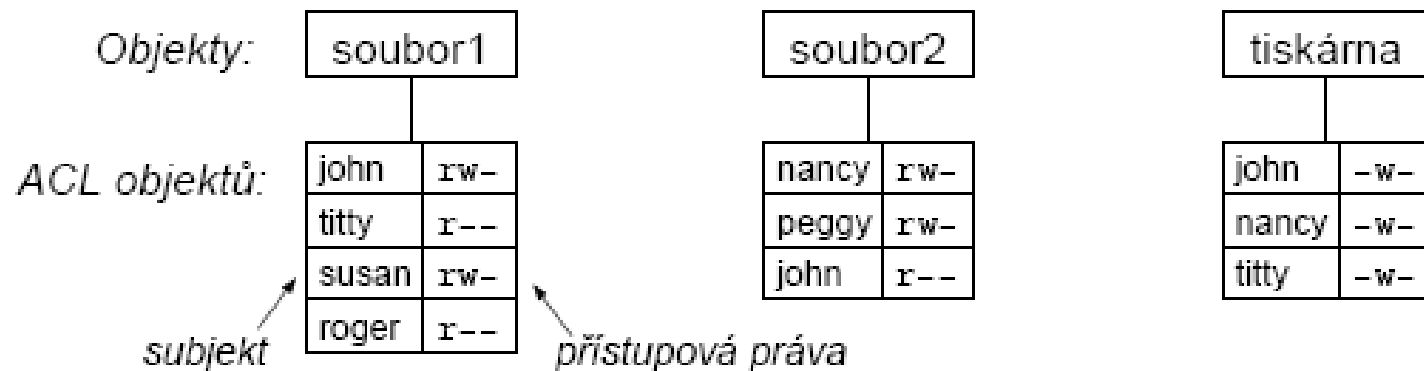


ACL x capability list

- 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



ACL příkazy

- **getfacl** soubor
- **setfacl** -m user:pepa:rw s1.txt

Přečtěte si článek:

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



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



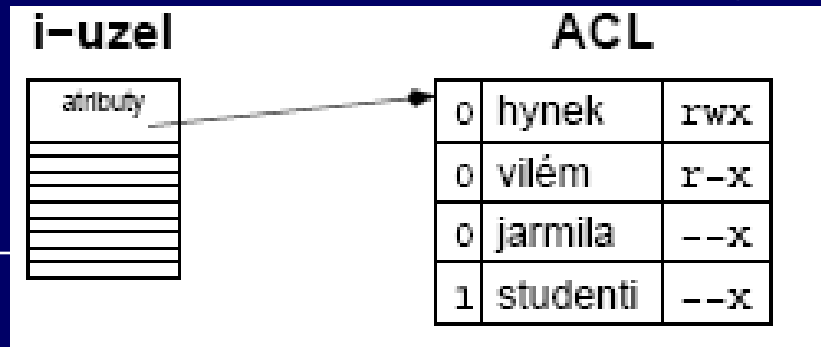
Klasická práva vs. ACL

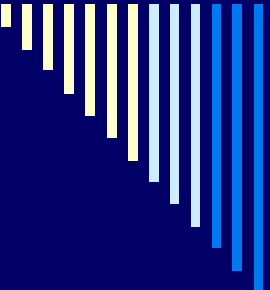
ACL má přednost před klasickými
unixovými právy (nastavenými chmod)

Tj. nastavením chmod 777 nezrušíme
ACLka 😊

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





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

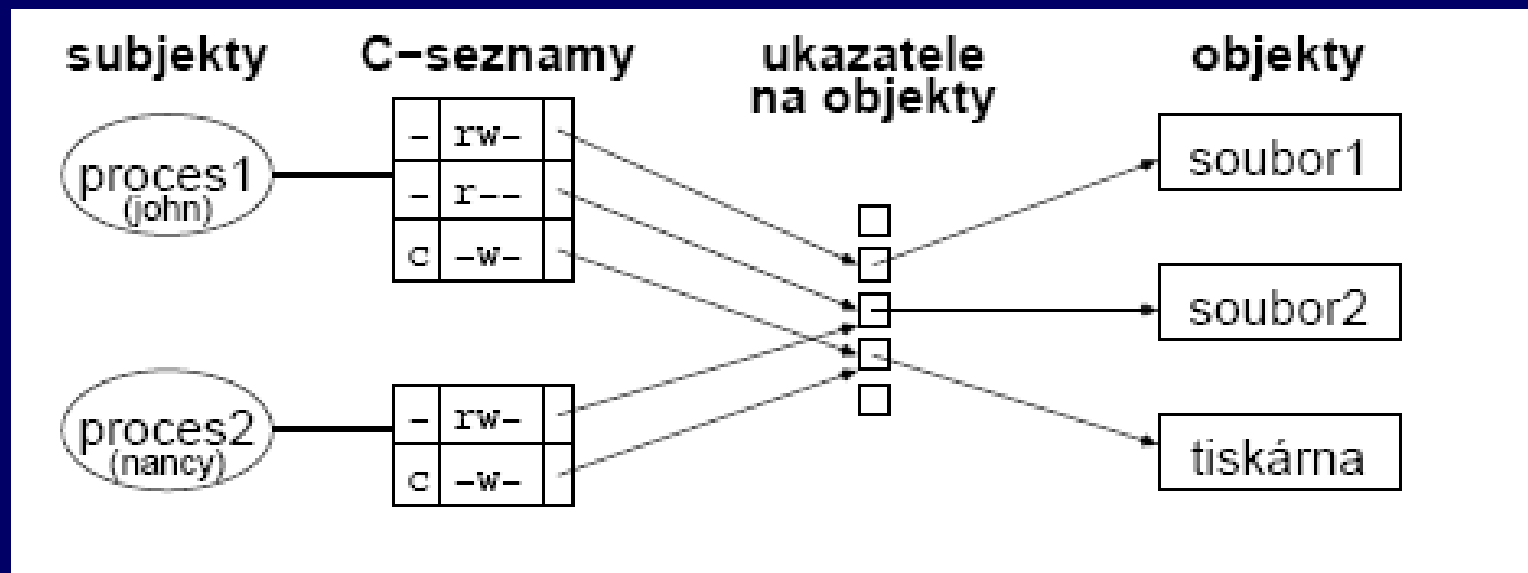
- Typ objektu
- **Práva** – obvykle bitová mapa popisující dovolené operace nad objektem
- **Odkaz na objekt**, např. číslo uzlu, segmentu, atd..



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
- ext2
 - klasická unixová práva (není to ACL)
 - vlastník, skupina, ostatní (r,w,x,s,...)
 - lze přidat ACL
- NTFS
 - ACL
 - lze měnit přes grafické UI, příkaz icacs, ...
 - explicitně udělit / odeprít práva
 - zdědit práva, zakázat dědění

Typy záloh



manipulací s
atributech
archive

□ normální

- zálohuje, označí soubory jako "zazálohované"

□ copy

- zálohuje, ale neoznačí jako "zazálohované"

□ incremental

- zálohuje pouze vybrané soubory, tj. pokud nebyly "zazálohované" nebo byli změněny a označí je jako "zazálohované"

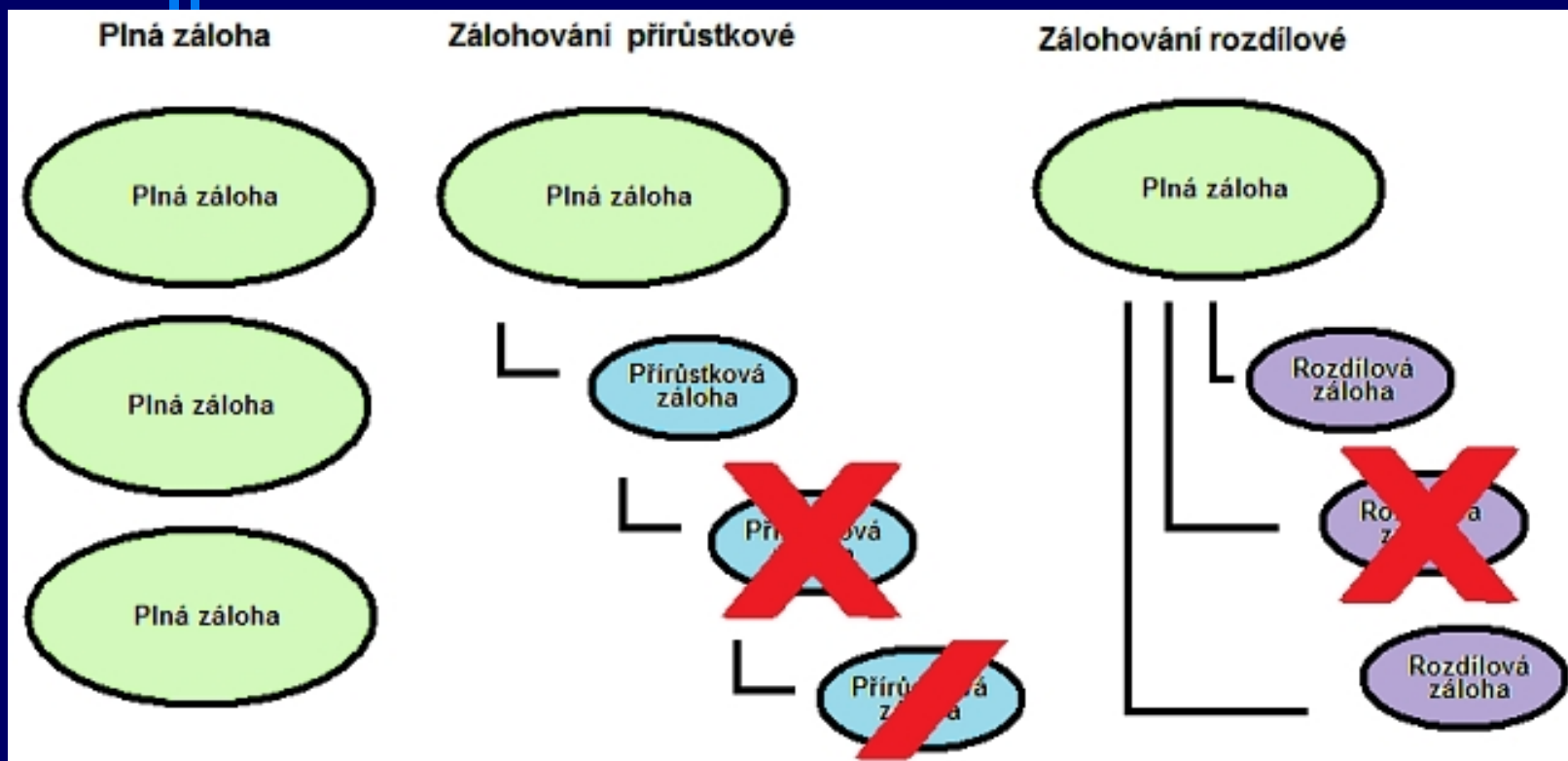
□ differential

- viz předchozí, ale neoznačuje jako "zazálohované"
- změny, které proběhly od plné zálohy
- diferenciální zálohy nejsou na sobě závislé

□ 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

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)

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** (GRUB, GRUB2, LILO, ...)

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

- můžeme prozkoumat příkazem **dmesg**

06. první proces **init**

/sbin/init , načte /etc/inittab , spouštění a vypínání služeb

/etc/rc.d/rcX.d



Co se děje při spuštění PC?

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

08. spustí se **login**
vyžádá si heslo, zkontroluje v `/etc/passwd`, `/etc/shadow` či jinde