

VYSOKÉ UCENÍ TECHNICKÉ V BRNĚ  
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

IOS - Operační systémy  
poznámky z přednášek

# Obsah

<b>1</b>		<b>4</b>
1.1	Uvod, prehľad operacných systému . . . . .	4
1.2	Základní pojmy . . . . .	5
1.3	Jadro operačního systému . . . . .	6
1.4	Typy jader OS . . . . .	7
1.5	Historie vývoje OS . . . . .	9
1.6	Prehľad technického vybavení . . . . .	9
1.7	Klasifikace počítačů . . . . .	10
1.8	Klasifikace OS . . . . .	11
1.9	Implementace OS . . . . .	11
1.10	Hlavní směry ve vývoji OS . . . . .	12
<b>2</b>		<b>13</b>
2.1	Příčiny úspěchu UNIXu . . . . .	13
2.2	Variety UNIXu . . . . .	14
2.3	Základní koncepty . . . . .	14
2.4	Struktura jádra UNIXu . . . . .	15
2.5	Komunikace s jádrem a hardwarová přerušování . . . . .	16
2.5.1	Hardwarové přerušování . . . . .	16
2.5.2	Zakazování přerušování . . . . .	17
2.5.3	Přístupy k zakazování přerušování . . . . .	18
2.5.4	Ovládací zařízení a přerušování . . . . .	18
2.5.5	Příklad komunikace s jádrem . . . . .	19
2.6	Nástroje programátora UNIXu . . . . .	20
<b>3</b>		<b>20</b>
3.1	Bash, shell, experimenty . . . . .	20
<b>4</b>		<b>20</b>
4.1	Bash, shell, experimenty . . . . .	20
<b>5</b>		<b>21</b>
5.1	Pevný disk . . . . .	21
5.2	Parametry pevných disků . . . . .	22
5.3	Solid State Drive - SSD . . . . .	23
5.3.1	Klady a zápory SSD . . . . .	23
5.3.2	Problematika zápisu u SSD . . . . .	23
5.4	Zabezpečení disku . . . . .	24
5.5	Disková pole (RAID) . . . . .	25
5.5.1	RAID 0 . . . . .	25
5.5.2	RAID 1 . . . . .	25
5.5.3	RAID 2 . . . . .	25
5.5.4	RAID 3 . . . . .	26
5.5.5	RAID 4 . . . . .	26
5.5.6	RAID 5 . . . . .	26
5.5.7	RAID 6 . . . . .	26
5.6	Opravy chyb u paritních disků . . . . .	27
5.7	Uložení dat na disk . . . . .	27
5.8	Fragmentace . . . . .	28

5.8.1	Externi fragmentace . . . . .	28
5.8.2	Interni fragmentace . . . . .	29
5.9	Pristup na disk . . . . .	30
5.10	Planovani pristupu na disk . . . . .	30
5.11	Logicky disk . . . . .	31
5.11.1	Zpusob ulozeni informaci o diskovych oblastech na disku . . . . .	31
5.11.2	LVM . . . . .	31
5.11.3	Ruzne typy souborovych systemu . . . . .	31
5.11.4	Chyby disku (souvislost s FS) . . . . .	32
5.11.5	Dalsi typy souborovych systemu . . . . .	32
<b>6</b>		<b>33</b>
6.1	Zurnalovani . . . . .	33
6.1.1	Implementace zurnalovani . . . . .	34
6.1.2	Copy-on-write . . . . .	35
6.1.3	Dalsi alternativy zurnalovani . . . . .	36
6.2	Klasicky UNIXovy system souboru (FS) . . . . .	37
6.2.1	i-uzel . . . . .	37
6.2.2	Kde a jak jsou ulozena data . . . . .	38
6.2.3	Pocty odkazu . . . . .	39
6.2.4	Limit maximalni velikosti souboru . . . . .	39
6.2.5	Vyhody a nevahy architektury FS . . . . .	40
6.3	Jine zpusoby organizace souboru . . . . .	41
6.3.1	Kontinualni ulozeni . . . . .	41
6.3.2	Zretezene seznamy alokacnich bloku . . . . .	41
6.3.3	FAT . . . . .	41
6.3.4	B+ stromy . . . . .	42
6.3.5	Extent . . . . .	44
6.4	EXT4 . . . . .	45
6.5	NTFS . . . . .	45
6.6	Organizace volneho prostoru na disku . . . . .	45
6.7	Deduplikace . . . . .	46
6.8	Typy souboru v UNIXu . . . . .	46
6.9	Adresar . . . . .	47
6.10	Montovani disku . . . . .	48
<b>7</b>		<b>50</b>
7.1	Symbolicke odkazy . . . . .	50
7.2	Blokove a znakové specialni soubory . . . . .	51
7.3	Pristupova prava . . . . .	52
7.4	Typy pristupovych prav . . . . .	53
7.5	Sticky bit . . . . .	53
7.6	SUID, SGID . . . . .	53
7.7	Typicka struktura adresaru v UNIXu . . . . .	54
7.8	Pouziti vyrovnacich pameti . . . . .	55
7.9	Operace se soubory . . . . .	56
7.9.1	Cteni . . . . .	56
7.9.2	Zapis . . . . .	56
7.9.3	Otevreni souboru pro cteni . . . . .	57
7.9.4	Cteni a zapis z/do souboru . . . . .	58

7.9.5	Prímy prístup k súboru . . . . .	59
7.9.6	Zavrení súboru . . . . .	60
7.9.7	Duplikácia deskriptoru súboru . . . . .	60
7.9.8	Rúsení súboru . . . . .	61
7.9.9	Dalsí operácie se súbory . . . . .	61
7.9.10	Adresárové súbory . . . . .	61
7.9.11	Blokové a znakové špeciálne súbory . . . . .	62
7.10	Terminály . . . . .	63
7.11	Roury . . . . .	64
7.12	Sockety . . . . .	64

# 1

**První přednáška:** Úvod do predmetu, prehľad operacnich systemu, zakladni pojmy, jadro operacniho systemu a jejich typy, historie vyvoje operacnich systemu, prehľad technickeho vybaveni, klasifikace pocitacu, operacnich systemu, hlavni smery ve vyvoji operacniho systemu.

## 1.1 Úvod, prehľad operacnich systemu

Operacni system je vyznamnou casti vypocetnich systemu, ty zahrnuji:

- hardware,
- operacni system,
- uzivatelske aplikacni programy,
- uzivatele.

Prehled nekterych OS:

- GNU/Linux
  - GNU/Debian - Ubuntu
  - Red Hat - RHEL, Fedora, Cent OS
  - SuSE
  - Gentoo, Arch Linux, Slackware (= nejstarsi live distribuce linuxu)
- BSD
  - FreeBSD, OpenBSD
- GNU
  - zn. GNU Is Not Unix
- MS Windows
- Mac OS X
  - jadro XNU = X is Not Unix
- Android, iOS
- Minix
  - pouziva intel ve svych cipech

## 1.2 Zakladni pojmy

Operacni system je program (resp. kolekce programu), která vytváří spojující mezivrstvu mezi hardware operacního systému a uživateli a jejich uživ. aplik. programy. OS dále spotřebovává zdroje, jako jsou paměť nebo čas CPU. (tldr: sw, spojující hardware, uživatele a programy)

### Cile OS:

- maximalní využití zdrojů počítače - drahé počítače, levnější pracovní síla (drive)
- jednoduchost použití počítačů - levné pc, drahá pracovní síla (dnes převládá)

### Zakladni role OS:

- správce prostředků
  - paměť, procesor, periférie
  - dovoluje sdílet prostředky efektivně a bezpečně
- tvůrce prostředí pro uživatele a jejich aplikací programy
  - vytváření abstrakcí, virtuálních objektů (resp. poskytuje standardní rozhraní, které zjednodušuje přenositelnost aplikací a zúčastnění uživatelů)
  - abstrakce jsou např.: proces, program, soubor
  - problémy abstrakcí jsou menší efektivita a nepřístupné některé nízkourovňové operace

### OS zahrnuje:

- jádro (kernel),
- systémové knihovny a utility (= systémové aplikací programy),
- textové (shell) či grafické uživatelské rozhraní (X Window).

Přesná definice, co vše OS zahrnuje neexistuje. Různé firmy a komunity to chápou různě. (GNU to chápe např. jako projekt svobodného OS, zahrnující jádro, utility, GUI, TUI, vývojové prostředky a knihovny, ...)

### definice:

*proces* je aktivita řízená programem

*program* je předpis, návod na nějakou činnost zakódovaný vhodným způsobem

*soubor* je kolekce záznamů (obvykle Byte) sloužící primárně jako základní jednotka pro ukládání dat na vnějších paměťových médiích

*adresář* je kolekce souborů

### 1.3 Jadro operacniho systemu

Jedna se o nejnizsi a nejzakladnejsi cast OS. Zavadi se jako prvni a bezi po celou dobu behu pocitacoveho systemu (tzv. reaktivni system, spis nez transformacni). Navazuje primo na hardware (pripadne virtualizovany HW) a pro uzivatele a uziv. aplik. zcela zapouzduje.

#### Bezi v privilegovanem rezimu:

- je mozne menit obsah registru hw, je mozne zadavat prikazy hw (neni mozne v uzivatelskem rezimu)
- musi byt podporovano v hardware

#### Jadro (obecne) zajistuje:

- zakladni spravu prostredku a tvorbu zakladniho prostredi jak pro uzivatele tak pro zbytek OS
- zahrnuje vsechny operace, kdy je potreba primo komunikovat s hardware (prepinani kontextu - jadro, plaovani procesu - nekdy v jadru, nekdy mimo, zavedeni stranky z disku, ..)
- sluzby pro zbytek OS a uzivatele, nektere zajistuje automaticky
- nektere sluby nejsou poskytovany automaticky, musi si o ne zadat, nazyvame to volani sluzeb, tzv. *system-call* (= systemova volani), ktere musi byt implemenovana uzitim specializovanych instrukci (intel: *sw preruseni*, *syscall*, *sysenter*)

#### Rozlisujeme dva typy rozhrani OS:

- *kernel interface* (nebo taky: ABI, Kernel ABI) - prime volani jadra pomoci specializovanych instrukci
- *library interface* - rozhrani vyssi urovne (napr. C knihovny), typicke sluzby jsou napr. *printf* z C - volaji se funkce ze systemovych knihoven, mohou ale nemusi vest na volani sluzeb jadra (bezne aplikace pracuji s timto rozhranim)

#### definice:

*transformacni system* je system, ktery dostane nejaky vstup, zpracuje ho a udela nejaky vystup (prekladac) - pokud se zacykli = chyba

*reaktivni system* se spusti a do (teoreticky) nekonecna reaguje na podnety uzivatele (spust procesu - spusti procesu) - pokud prestane pracovat = chyba

*prepinani kontextu* je situace, kdy na CPU bezi proces, ten chci pozastavit a nechat bezet jiny proces

*instrukce syscall a sysenter* - jakmile aplikace (bezi v uziv. rezimu) zavola takovou instrukci, dojde ke kontrolovani prepnuti do rezimu jadra, provede se sluzba, a pote se prepne zpet

ABI = Application Binary Interface

## 1.4 Typy jader OS

### Monolitická jadra

- vysokourovnove komplexni rozhrani s radou sluzeb, abstrakci, které mohou pouzivat vyssi vrstvy OS
- vsechny subsystemy jsou implementovany v privilegovanem rezimu, rezimu jadra, a zahrnuji napr. spravu pameti, planovani, meziprocessovou komunikaci, souborove systemy, ..
- vyhody: vysoka efektivita díky provazanosti
- nevychody: mala flexibilita pri praci s jadrem (ve filesystemu je chyba, chci zmenit jen implementaci filesystemu za novou verzi a vse ostatni nechat - nelze, je nutne cely system zastavit a znovu nastarovat, nelze menit nic za behu)

### Monolitická jadra s modulární strukturou

- vylepseni koncepce monolitických jader
- umoznuje zavadet/odstranovat subsystemu jadra v podobe tzv. modulu za behu
- vyhody: neni nutne cely system zastavovat a znovu bootovat pro vymenu jednoho modulu, vyssi bezpecnost - zavedou se jen moduly, které se budou pouzivat
- pouzivane v napr. FreeBSD, Linux

### Mikrojadra

- snaha minimalizovat rozsah jadra a rozsah jeho sluzeb
- nabizi jednoduche rozhrani, maly pocet abstrakci, sluzeb, typicky nabizi nejzakladnejsi spravu CPU, I/O zarizeni, pameti, ..
- vetsina sluzeb nabizenych monolitickými jadry (ovladace, vyznamne casti spravy pameti, planovani) je implemenovana mimo jadro v tzv. serverech (nebezi v privilegovanem rezimu).
- vyhody: flexibilita (vice soucasne bezicich implementaci ruznych sluzeb, dynamicke spousteni, zastavovani..), zabezpeceni (chyba v serveru / utok na ne neznamená ovladnuti celeho OS, ale jen daneho serveru)
- nevychody: vyrazne vyssi rezie

### Generace mikrojader

- 1. generace - napr. Mach
- 2. generace - napr. L4, mensi rezie nez 1. gen
- 3. generace - napr. seL4 nebo ProvenCore, duraz na zabezpeceni, navrh s ohledem na moznost formalni verifikace



## Hybridni jadra

- "neco mezi mikrojadry a monolitickymi jadry"
- jadra zalozena na mikrojadrech, rozsirena o kod, který by mohl byt implementovan ve forme serveru, je ale za ucelem mensi rezie tesneji provazan s mikrojadrem a bezi v jeho rezimu
- pouzivane v napr. Mac OS X (Mach + BSD), Windows NT (a vyssi), ...

## definice:

*servery (v oblasti mikrojader) jsou procesy*

*formalni verifikaci rozumime overeni urcitych vlastnosti systemu s platnosti matematickeho dukazu*

## linux prikazy:

*lsmod* - vypise aktualne zavedene moduly jadra

*rmmod* - maze moduly jadra

*modprobe* - zavadeni modulu do jadra

## 1.5 Historie vyvoje OS

### definice:

*preruseni* je elektricky signal, který jde od periferie po sběrnici k procesoru, na CPU vyvolá obsluhu preruseni - mechanismus umožňující rozbehnout operaci na periférii a o tu periférii se nestarat (periferie poté oznámí konec operace) (podrobně se tomu věnuje oddíl 2.5)

*multitasking* je současný běh více aplikací na jednom procesoru (může být s preemtivním nebo nepreemtivním plánováním)

*nepreemtivní plánování* zn. že úloha, kt. aktuálně běží na CPU může být od CPU "odstavena" pouze tehdy, když nějak komunikuje s jádrem (= požádá o službu jádra, např. periferní operace), dokonce lze použít specializované služby pro přepnutí kontextu (proces se dobrovolně vzdá CPU, tzv. yield služby) - výhoda: snadná implementace, nevýhoda: pokud se proces zacyklí (chyba), celý systém se zablokuje (poradí běží 1 úloha)

*preemtivní plánování* - proces může být odstaven od CPU bez nutnosti komunikace s jádrem, např. pomocí preruseni (jakéhokoli typu)

## 1.6 Přehled technického vybavení

### Procesor (CPU):

- radice, ALU, registry (IP, SP), instrukce, ..

### Paměť:

- adresa
- hierarchie paměti (cache, RAM, disky, ... - bank paměti může být více)
  - paměti se liší spotřebou, kapacitou, rychlostí, cenou za jednotku
  - na vrcholu hierarchie jsou registry (nejrychlejší, nejvyšší cena za jednotku, malá kapacita)
  - cache (vyrovnávací paměti, různých úrovní, L1 = level 1, L2, L3, ..)
  - primární paměť RAM
  - sekundární paměti - disky (SSD, HDD)
  - vyrovnávací paměti disku
  - terciální paměti (zálohy - nejnižší cena za jednotku, nejpomalejší, největší kapacita - pásky, CD/DVD, externí disky, cloudy, síťové disky, ..)

### Periferie:

- disk (HDD, SSD,..), klávesnice, monitor (I/O porty, preruseni, DMA)

### Sběrnice:

- propojují jednotlivé komponenty
- na vrcholu hierarchie jsou sběrnice propojující CPU a paměť (FSB - Front Side Bus, HyperTransport QPI - Quick Path Interconnect)

- diskove sbernice (SATA/ATA, SCSI/SAS, USB)
- dalsi sbernice (NVLink - pripojovani nVidia GPU, PCI - rozsirujici karty ci disky, CAPI - IBM Tauer CPU, propojovani CPU a akceleratoru)

#### **definice:**

*I/O porty* = vstup-vystupni porty, predstavuji pametove oddeleny prostor od adresoveho prostoru bezne pameti, s temito adresami se komunikuje specialnimi instrukcemi (intel: inout)

*pametove mapovane I/O* je cast adresoveho prostoru bezne pameti neni pouzita pro praci s pameti, ale adresy jsou presmerovane do HW (neco co zapisu na danou adresu nebude v pameti ale v nejakem registru HW)

*DMA* zn. Direct Memory Access, souvisi s nezavislou cinnosti periferii - periferie mohou primo komunikovat s hardware (radic disku si sam z adresy pameti nacte data a pres sbernice je prenasi na disk, nebo naopak)

## **1.7 Klasifikace pocitacu**

#### **Dle ucelu:**

- univerzalni,
- specializovane
  - vestavene (palubni pc, spotrebni elektronika, ..)
  - aplikacne orientovane (rizeni db, sitove servery, ..)
  - vyvojove (zkouseni novych technologii)

#### **Podle vykonnosti:**

- vestavene pc, tablety, mobily, ..
- osobni pocitace (PC) a pracovni stanice (workstation) - dnes se nerozlisuje
- servery
- strediskove pocitace (mainframe) - vyrabi IBM, ladene na obrovsky I/O vykon a vysokou spolehlivost
- superpocitace - ladene na surový vypocetni vykon (vedecke vypocty, simulace)

## 1.8 Klasifikace OS

### Podle ucelu:

- univerzalni (UNIX, Linux, Windows, ..)
- specializovane (real-time - RT-Linux, databaze, web - z/VSE, mobilni - iOS, Android)

### Podle poctu uzivatelu:

- jednouzivatelske (CP/M, MS-DOS,..)
- viceuzivatelske (UNIX, Windows, ..)

### Podle poctu soucasne bezicich uloh:

- jednoulohove
- viceulohove (multitasking, ne/preemptivni)

### definice:

*soft real-time* - doporučení aby se akce vykonávaly v reálném case

*hard real-time* - akce se musí vykonávat v určitém case

## 1.9 Implementace OS

OS se obtížně programují a ladí, protože to jsou velké programové systémy, paralelní a asynchronní systémy, systémy závislé na technickém vybavení.

### Důsledky:

- setrvačnost při implementaci (snaha neměnit kód, který pracuje spolehlivě)
- používání technik pro minimalizaci výskytu chyb (inspekce zdrojového kódu, rozsáhlé testování, podpora vývoje technik formální verifikace)

### definice:

*paralelní systém* zn. že zde běží více aktivit současně

*paralelní asynchronní systémy* - procesy se prepínají v okamžicích, které nelze dopředu přesně předpovědět

### 1.10 Hlavní směry ve vývoji OS

- neustále vylepšování architektur (snížení rezí jader,)
- bezpečnost, spolehlivost
- podpora stále většího počtu procesorů, více jader
- virtualizace
- distribuované zpracování (cloudy, kontejnery, Internet of Things)
- OS tabletů, mobilů, vestavěných systémů, ...
- vývoj nových technik návrhu a implementace OS (podpora formální verifikace)

#### **definice:**

*bezpečnost* zn., že systém je odolný vůči vnějším útokům

*spolehlivost* zn., že systém "nespadne sám od sebe"

## 2

**Druha prednaska:** Unix - uvod: historie UNIXu (nezkousi se), priciny uspechu UNIXu, varianty UNIXu, zakladni koncepty, struktura jadra, komunikace s jadrem - hardwarova preruseni. Prehled programovani v UNIXu: nastroje programatora, ..

### 2.1 Priciny uspechu UNIXu

- viceprocesovy, viceuzivatelsky,
- napsan v C - prenositelny,
- zpocatku (a pozdeji) siren ve zdrojovem tvaru,
- "mechanism, not policy",
- "fun to hack",
- jednoduche uzivatelske rozhrani (terminal),
- skladani slozitejsich programu z jednodussich (tvoreni aplikaci typu filtr),
- hierarchicky system souboru,
- konzistentni rozhrani perifernich zarizeni

#### definice:

*"mechanism, not policy"* zn. snaha oddelit casti aplikaci (napr. GUI - oddelit zakladni rutiny pro vykreslovani grafiky od politik, tzn. koncove nastavby - barvy oken, umisteni tlacitek, .. - systematicke rozdeleni vede k lepsim optimalizacim a ladenim algoritmu a zaroven rychlym zmenam politik)

*"fun to hack"* zn., lide se na vyvoji podili, protoze je to bavi (nejen protoze jsou za to placeni)

*aplikace typu filtr* - jednoduche otevrene aplikace, na vstupu maji textovy dokument v otevrene podobě, vstup zpracuji a na vystupu opet otevreny dokument (zadne binarni, zakodovane)

## 2.2 Varianty UNIXu

### Hlavní větve OS UNIXového typu:

- UNIX System V (původní systém z AT&T),
- BSD UNIX (FreeBSD, NetBSD, ..),
- firemní varianty (AIX, Solaris, ..)
- Linux

### Související normy:

- XPG - X/OPEN, SVR4 - AT&T, SUN, OSF/1, Single UNIX Specification,
- POSIX - IEEE standard,
- Single UNIX Specification v3/v4 - shell, utility (CLI), API

### definice:

*POSIX* je striktní podmnožina Single UNIX Specification, je to standard definující základní textové příkazy rozhraní OS + API

## 2.3 Základní koncepty

Jsou dvě základní koncepce (abstrakce) UNIXu: **procesy** a **soubory**.

Procesy mezi sebou komunikují pomocí různých mechanismů meziprocesové komunikace - IPC (Inter-Process Communication) - roury, signály, semaforey, sdílená paměť, sockets, zprávy, streams, .. a pro komunikaci používají nějaké I/O rozhraní (read, write, close, ..)

### definice:

*procesy* jsou abstrakcí probíhajících nějakých aktivit (viz 1.2)

*soubory* jsou abstrakcí dat (viz 1.2)

## 2.4 Struktura jadra UNIXu

Zakladni podsystemy jsou sprava souboru a sprava procesu.

### Popis:

- Na hornim okraji jadra (smerem k uzivatelum, aplikacim) je vrstva implementujici rozhrani volani sluzeb, prostrednictvim ktere jadro prebira zadosti o sluzby od aplikaci. Rozhrani kontroluje zda ten, kdo o sluzbu zada ji muze volat, zda jsou parametry validni a rozhrani predava pozadavek dal do jadra.
- Aplikace mohou s jadem komunikovat primo, nicmene nejcasteji komunikuji s jadem pres knihovny. (viz. 1.3)
- Na druhem okraji (tesne nad HW) je vrstva abstrakce hardware.
- Mezi spravou souboru a hardware se nachazi ovladace, pote vrstva vyrovnavacich pameti, ktere souborove systemy pouzivaji ke zrychleni prace s relativne pomalymi disky (HDD, SSD - oproti RAM pomale)
  - OS se snazi vyhnout opakovanemu cteni stejných dat, proto si v jednom okamziku nacte vic dat nez uzivatel zada, ulozi si data do vyrovnavaci pameti (pri dostatku pameti) a data nacita odtud. (napr. C knihovny jsou pouzivane kazdym druhym programem - jsou v pameti temer porad).

### definice:

*ovladace* jsou programy slouzici k rizeni (zadavani prikazu, prebirani stavovych informaci, reseni mimoradnych stavu konkretnich periferii) - lze je (jako i prislusna zarizeni) rozdelit na znakova a blokova (kratsi definice viz 5.9)

*znakova zarizeni* jsou zarizeni komunikujici po jednotlivych znacich (klavesnice)

*blokova zarizeni* komunikuji po blocich (disk - sektory, resp. bloky)

*komunikaci s jadem* rozumime nastavovani parametru hardware, vydavani prikazu hw, obsluhu ruznych stavu do kterych se hw dostava (a o kterych je CPU a jadro informovano prostrednictvim preruseni)

*nastavovani parametru hw* se deje pomoci I/O portu nebo pametove mapovanych operaci (viz 1.6)



## 2.5 Komunikace s jadem a hardwarova preruseni

Sluzby jadra jsou operace, jejich realizace je pro procesy zajistovana jadem. Explicitne je mozne o provedeni urcite sluzby zadat prostrednictvim system call (viz 1.3).

### Priklady nekterych sluzeb jadra (systemova volani v UNIXu):

- open, close, read - otevře/zavře/čte soubor,
- write - zapisuje,
- kill - posle signal,
- fork - duplikuje proces,
- exec - prepise kod,
- exit - ukonci proces.

### 2.5.1 Hardwarove preruseni

- hardware interrupt je mechanismus, kterym HW zarizeni oznamuji jadu asynchrone vznik udalosti, ktere je zapotrebi obslouzit (dalsi mozna definice viz 1.5),
- zadosti o HW preruseni prichazi jako elektricke signaly (IRQ) do radice preruseni (APIC),
- procesor s radicem preruseni komunikuje pomoci I/O portu.

### Prijem nebo obsluhu HW preruseni lze zakazat:

- maskovanim preruseni,
- na CPU (instrukce CLI/STI na Intel/AMD - zakazou se vsechna krome NMI),
- ciste programve v jadre (preruseni se prijme, ale jadro si jen poznamena jeho prichod a neobsluhuje se)

### NMI:

- non-maskable interrupt je HW preruseni, ktere nelze zamaskovat na radici ani zakazat na CPU,
- pouziva se pri kritickych chybach pameti, sbernice, .. (alternativne se pouziva pro ladeni / reseni uvaznuti v jadre "NMI watchdog")

### Preruseni mohou vznikat i v CPU - jsou to synchronni preruseni, tzv. vyjimky (= exceptions):

- trap - po obsluze se pokracuje dalsi instrukci (breakpoint, overflow, ..)
- fault - po obsluze se znovu opakuje instrukce, ktera vyjimku vyvolala (vypadek stranky, deleni 0, ..)
- abort - dochazi k zavaznym problemum detekovanim CPU, neni jasne jak pokracovat - provedeni se ukonci (zanorene vyjimky typu fault, chyby HW detekovane CPU)

**Mohou existovat i dalsi typy preruseni:** (tato preruseni obsluhuje CPU zcela specifickym zpusobem (casto mimo vliv jadra, napr. na Intel/AMD))

- Interprocessor interrupt (IPI)

- meziprocesorove preruseni
- pouziva se pro preposilani preruseni z jednoho CPU na druhy nebo pro spravu cache (kazdy CPU ma svoji cache, do nich mohou mit CPU nacteny stejne adresy z pameti - pokud dojde ke zmenam v pameti, musi CPU informovat ostatni CPU o zmene)
- System management Interrupt (SMI)
  - preruseni typu sprava systemu
  - muze byt vyvolano HW i SW ve zvlastnich situacich
  - pokud se takove preruseni vyvola, tak se dostane ke slovu firmware, který provadi obsluhu ruznych chybovych stavu (prehrati, vybita baterie, ..)
  - v ramci SMI nebezi bezne aplikace ani jadro, nesmi obsluha SMI bezet prilis dlouho (system se muze dostat do nekonzistentniho stavu)

### **2.5.2 Zakazovani preruseni**

#### **Proc preruseni zakazovat?**

- v ramci obsluhy jednoho preruseni muze nastat dalsi preruseni,
- napr. na CPU bezi vypocet, neco nastane na disku, disk posle preruseni, to dojde k CPU a jadro zacne preruseni obsluhovat, v ten moment se neco stane na klavesnici a prijde dalsi preruseni,
- pote dale v ramci obsluhy muze jadro upravovat ruzne sve interni struktury, které mohou byt v nekonzistentnim stavu (napr. zretezene seznamy procesu [ukazatele], ruzne si je projuje, nez je stihne propojit, prijde dalsi proces a muze sahnout do pameti kam nema),
- proto obsluha preruseni musi byt synchronizovana a v pripade, ze se v ramci preruseni provadi nejaka kriticka operace je nutne vyloucit ostatni (vsechna) preruseni

### 2.5.3 Pristupy k zakazovani preruseni

Pokud vsak zakazu (nejaka/vsechna) preruseni, abych se mohl venovat obsluze jednoho a budu ho obsluhovat prilis dlouho, system se muze dostat do nekonzistentniho stavu (jako u SMI). Pouzivaji se proto dva pristupy:

- je snaha zakazovat jen preruseni s nizsimi prioritami,
- rozdelit obsluhu preruseni do vice casti (urovni).

**Obsluha preruseni je casto delena na dve urovne:**

- 1. uroven:
  - ma byt co nejkratsi,
  - v ramci obsluhy preruseni se zakomunikuje nezbytnym zpusobem s HW (prevzani dat z/do HW, vydani prikazu HW, ..) a naplanuje se beh 2. urovne,
  - nelze pouzit bezne synchronizacni prostredky (protoze napr. CPU bezi nejaky vypocet, prijde preruseni z disku, jadro zacne resit 1. uroven obsluhy, nicmene obsluha != proces)
- 2. uroven:
  - dokoncuje obsluhu preruseni,
  - provadi se operace, kdy neni potreba komunikovat s hardware,
  - nemusí se zakazovat preruseni,
  - muze bezet v specialnich procesech (interrupt threads ve FreeBSD nebo tasklety/softIRQ v Linuxu),
  - mohou se pouzit bezne synchronizacni prostredky

### 2.5.4 Ovladace zarizeni a preruseni

- pri inicializaci ovladace (v Linuxu je to typicky modul) nebo pri jeho prvni pouziti se musi registrovat k obsluze urcitého IRQ,
- bud u nekterych zarizeni se pouzivaji (historicky) zafixovana cisla preruseni,
- nebo ovladac muze zjistit cislo preruseni tak, ze zakomunikuje s radicem sbernic, pokud to nefunguje,
- ovladac vyda prikaz zarizeni, ktere ma ovladat, aby zacalo vysilat nejaka preruseni (a "poslouchala" sbernici, "kdo se ozve"),
- pote se zaregistruje k obsluze prislusneho preruseni a hardware se pres tabulku preruseni ovladac "dostane ke slovu",
- vice zarizeni vsak muze pouzivat stejne cislo zadosti o preruseni
  - v takovem pripade jadro vytvori zretezeny seznam ovladacu, ktere maji zajem o dane preruseni
  - ovladace musi byt napsane tak, ze pokud jim dojde preruseni (o ktere maji zajem), tak musi zakomunikovat s tim zarizenim a zeptat se ho, zda opravdu to zarizeni poslalo dane preruseni
  - pokud ano - obslouzi se, pokud ne - preda se rizeni preruseni dalsimu ovladaci v seznamu

### 2.5.5 Příklad komunikace s jádrem

Synchronní komunikace je proces-jadro, asynchronní je hardware-jadro. Příklad (detailnější, ale na téma přístupu na disk viz 5.9):

- proces A zavola službu read() a jádro ihned začne volání obsluhovat (synchronní)
- nejprve se podívá do cache zda data, o která má zájem proces A už tam nejsou
- pokud ano, tak mu je rychle nakopíruje z cache na adresu, kterou požaduje proces (bez komunikace s diskem)
- pokud data nejsou v cache, proces A bude pozastaven a jádro vydá prostřednictvím ovladače disku příkaz k načtení určitého objemu dat, typicky více než žádá uživatel a načítá do vyrovnávací paměti (ne na požadovanou adresu)
- na procesoru daleko bezí proces B, taky požádá o read(), zopakuje se to samé co u A
- až disk dokončí operaci jednoho z procesů (nemusí být v pořadí volání), disk pošle přerušeni na CPU
- jádro bude informováno, že má potřebná data pro proces A/B
- z cache nakopíruje požadovaná data na požadovanou adresu
- poté se proces A/B probudí a bezí dál, to samé se stane u dalšího procesu

#### definice (pro 2.5.x):

*asynchronní* zn., bez prime-okamžité vazby na to co dělá jádro nebo aplikace (tiskárna tiskne - operace někdy skončí - ale nikdy nevím dopředu kdy přesně)

*synchronní* zn., že CPU něco provede a ihned se zavola přerušeni (např. dělení 0)

*IRQ* = interrupt request

*radic přerušeni* = interrupt controller, hardwarová jednotka, která předává přerušeni do CPU - registruje příchod IRQ, ty se dle priorit předávají do CPU (přerušeni je možné také zamaskovat - nepředávat dál do CPU) v podobě čísla přerušeni, CPU se automaticky přepne do chráněného režimu a spustí obslužnou rutinu definovanou jádrem (přerušeni 1 - provede xxx, 2 - xxx, ..)

*APIC* = Advanced Programmable Interrupt Controller - distribuovaný systém, každý CPU má lokální APIC, externí zařízení mohou být připojena přímo / přes I/O APIC

*NMI watchdog* - jádro si nadefinuje, že časovač mu každých n časových jednotek pošle toto přerušeni - pokud dojde v jádře k uvážnutí při obsluze jiného přerušeni a všechna přerušeni budou zakázána, toto se vždy dostane do CPU (jádro se může zotavit)

*vypadek stranky* zn., (paměť je rozdělena na části, které mohou být rozděleny na disk) když proces bude sahát do paměti a sahne na stránku, která v ní není - detekuje se že stránka tam není - porušení ochrany paměti - jádro zkontroluje, zda proces nesáhá kam by neměl, a pokud ne, tak mu stránku nahraje zpět do paměti a znovu se provede ta stejná instrukce

*bezne synchronizační prostředky* jsou např. semaforey nebo zamky a synchronizují procesy

#### linux:

základní statistiky o obsluze přerušeni jsou v */proc/interrupts*

## **2.6 Nastroje programatora UNIXu**

X-Window system, vzdaleny pristup pres X-Window uzitecne prikazy na linuxu, ovladani vimu, apod. - vice viz. 2. prednaska IOS, u zkousky to nebyva.

## **3**

### **3.1 Bash, shell, experimenty**

## **4**

### **4.1 Bash, shell, experimenty**

Treti a ctvrta prednaska je venovana hlavne shellu, prochazi se prakticky ruzne prikazy a provadi se experimenty, apod. - lepsi je shlednout + na zkousce nic takoveho nebyva.

## 5

**Pata prednaska:** Sprava souboru: pevný disk, diskové sbernice, sektory, parametry pevných disků, SSD, problematika zápisu SSD, zabezpečení disku, disková pole (RAID), uložení dat na disku, fragmentace, přístup na disk a jeho plánování, logický disk.

### 5.1 Pevný disk

#### Popis:

- uvnitř mají radu kulatých ploten, zaznam se provádí na každém z těchto dvou povrchů, je v soustředných kruznicích (= tracks, stopy)
- všechny plotny jsou na stejné ose, přidelané k sobě a rotují současně
- k načítání slouží sada hlaviček, čtecí a zapisové, jsou tam v tolika kusech, kolik je tam povrchů (např. 3 plotny = 6 povrchů = 6 hlaviček), všechny umístěné na jednom rameni, všechny hlavičky se pohybují současně
- hlavičky jsou nastaveny na sadě několika stop (kruznic) o stejném průměru = cylindr,
- stopy se dělí na sektory
- velikosti sektorů byly dříve 512B, u CD/DVD 2048B, dnes 4096B

#### Adresace sektorů:

- ze začátku se používal CHS - určí se se kterým cylindrem chce pracovat, dále s kterou hlavou a jakým sektorem v rámci stopy,
- v současné době se používá LBA, kde jsou sektory (bloky) číslovány (jako adresy v paměti) od 0 po n, disková jednotka si musí tato čísla převádět na CHS

#### Periferní či disková rozhraní:

- používají se pro připojení disku,
- nejbezpečnější se používá ATA, dříve se používala v paralelní verzi (PATA - jednotlivé byty se posílaly paralelně, při rostoucích rychlostech byl problém zajistit synchronizaci těchto dat), nyní v sériové verzi (SATA)
- také se používá SCSI či SAS (Serial Attached SCSI), USB, FireWire, FibreChannel, Thunderbolt, PCI Express nebo NVMe (připojování nejrychlejších SSD),
- nad těmito rozhraními může být další HW rozhraní propojující tyto sbernice, jako třeba AHCI, OHCI, UHCI, ..

### Diskove sbernice se lisi:

- rychlosti (SATA do 6 Gbit/s, SAS 22.5 Gbit/s),
- poctem pripojenych zarizeni (SATA desitky, 65535 SAS),
- maximalni delkou kabelu (1-2m SATA, 10m SAS),
- architekturou pripojeni (moznost pripojeni jednoho zarizeni vice cestami u SAS),
- seznamem prikazu, ktere to zarizeni umi (flexibilita pri chybach, selhani, zotaveni, ..)

Pres diskove sbernice je mozne mit pripojene i jine typy pameti, jako jsou flash disky, SSD, pasky, CD/DVD/BD ci terciální pameti. V systému vzniká hierarchie pameti, viz. 1.6.

### definice:

*cylindr* (v HDD) je množina stop o stejném průměru

*sektor* je nejmenší jednotka diskového prostoru, který mi umožní disková elektronika nacist nebo zapsat

*blok* nebo *diskový blok* je sektor v HDD

*alokací blok* nebo *blok souborového systému* je nejmenší jednotka, kterou umožní alokovat OS

*CHS* zn. Cylinder Head Sector

*LBA* zn. Linear Block Address

## 5.2 Parametry pevných disku

Přístupová doba sestává z **doby vystavení hlav** a **rotacího zpoždění**.

Typické parametry současných disků jsou kapacita, průměrná doba přístupu (jednotky ms u HDD) , otáčky a přenosová rychlost. U přenosových rychlostí se rozlišuje *sustained transfer rate* a *maximum transfer rate*.

Mazání dat probíhá tak, že se přepisují metadata, pouze se poznamená (OS), že daný soubor byl smazán.

### definice:

*doba vystavení hlaviček* zn., že pokud nejsou nastaveny hlavičky na stope, se kterou chci pracovat (malokdy), tak je nutné pohnout hlavičkami (víc zasunout dovnitř nebo vysunout)

*rotací zpoždění* je doba než mi pod správně nastavenou hlavičku najede sektor (narotuje se disk)

*maximum transfer rate* je spíková přenosová rychlost, jak maximálně rychle je schopen disk komunikovat po krátkou dobu (typicky rychlost předání dat z vyrovnávacích pamětí disku)

*sustained transfer rate* opravdová rychlost čtení z ploten

### linux:

*hdparm [-t]* umožňuje změřit přenosovou rychlost a měnit parametry disku, -T měří rychlost přenosu z vyrovnávací paměti OS (RAM)

## 5.3 Solid State Drive - SSD

Mohou byt zalozena na ruznych technologiich, nejcasteji na nevolatilnich pametech NAND flash nebo DRAM (se zalohovany napajenim) ci na kombinacich.

### 5.3.1 Klady a zapory SSD

#### Vyhody:

- rychly (okamzity) nabeh,
- nahodny pristup (mikrosekundy),
- vetsi prenosove rychlosti (stovky MB/s, ATA do 600MB/s, 3.5GB/s s M.2, 7GB/s s PCI Express 4),
- zapis muze byt mirne pomalejsi,
- tichy provoz, lepsi mechanicka a magneticka odolnost,
- obykle nizsi spotreba (neplati pro DRAM).

#### Nevyhody:

- vyssi cena za jednotku prostoru,
- omezeny pocet prepisu (nevyznamne pro bezny provoz),
- vetsi riziko katastrofickeho selhani,
- mensi vydrz mimo provoz (pri vyplem napajeni a skladovani),
- komplikace se zabezpecenim (bezpecne mazani nebo sifrovani prepisem dat - vyžaduje specialni podporu).

### 5.3.2 Problematika zapisu u SSD

NAND flash SSD jsou organizovany do stranek (typicky 4KiB) a ty jsou sdruzeny do bloku (typicky 128 stranek = 512 KiB).

#### Zapis nebo prepis dat:

- prazdne stranky lze zapisovat jednotlivé (prepisovat ne!),
- pokud chci prepisovat (jednu stranku), je nutne cely blok nacist do pameti, vymazat (zresetovat) a v pameti upraveny blok nacist zpet (= write amplification, zesileni zapisu - mnohonasobne zpomaleni),
- problem je mensi pri sekvencnim (pockam az budu mit dost dat tak aby pokryly blok) nez pri nahodnem zapisu do souboru.

#### Problem se sifrovanim a bezpecnym mazanim:

- diky tomu jak SSD prepisuji data se data nekolikrat presouvaji po disku,
- proto disk musi poskytovat hw podporu pro bezpecne mazani nebo sifrovani.

#### Reseni problemu prepisu u SSD:



- typicky ma SSD vice stranek-bloku nez je deklarovana kapacita (pri prepsani se zapise do volne stranky),
- po smazani dostatku stranek (tak ze tvori blok) se blok zresetuje - prikazem TRIM souborovy system sdeli SSD, ktere stranky jiz nejsou pouzivane (a ktere bloky muze SSD smazat),
- radic SSD muze stranky presouvat tak, aby si nektare bloky uvolnil (pokud je v bloku malo stranek, presunou se a blok se zresetuje),
- TRIM nelze pouzit vzdy (typicky pokud v souborovem systemu mame obraz jineho souboroveho systemu, nemusi byt mozne sdelit zakladnimu filesystemu informace o praznych blocich, apod. nebo data-baze, ktere si ukladaji data do velkeho predalokovaného prostoru, ci obrazy virtulanich stroji a virtualni disky)

Radic SSD presouva i dlouho nezmenene stranky, aby minimalizoval pocet prepisu stranek.

#### **definice:**

*nevolatilni* zn., ze pokud se vypne napajeni, tak obsah zustane zachovan (alespon po nejakou rozumnou dobu)  
*stranka* je nejmensi jednotka dat, kterou lze do SSD zapsat

## **5.4 Zabezpeceni disku**

Diskova elektronika typicky na ukladana data (sama o sobe) zabezpecuje kody, ktere umi pri naslednem cteni detekovat a pripadne opravit chyby - pouziva ECC. (detekce a oprava chyb je pouze v rezii disku, pokud disk detekuje chybu a neni prilis velka, chybu opravi a data ulozi na jiny sektor, poznaci si, ze ten sektor nema pouzivat)

Existuje technologie, ktere umoznuji zjistit, v jakem stavu disk je (statistiky, premapovani, pocet chybných sektorů, ..) - S.M.A.R.T (podporovana vsemi "rozumnymi" disky)

Pak je mozne jeste provadet testovani na urovni OS, napr. e2fsck nebo badblocks nebo si nektare filesystemy (RFS, ZFS) provadeji kontinualni kontroly toho, co se ve filesystemu deje. Tyto utility nebo filesystemy mohou chyby detekovat (a varovat) nebo opravit (pokud neni chyba prilis velka) ci vyradit pouziti nekterych sektorů.

#### **definice:**

*ECC* = Error Correction Code

*S.M.A.R.T* = Self Monitoring Analysis and Reporting Technology

*kontinualni kontroly (fs)* zn., ze si ukladaji sve dalsi kontrolni soucty, a pote si kontroluji pri praci se souborem, zda kontroly souhlasí

#### **linux:**

*smartctl* je prikaz umoznujici vyuziti technologie S.M.A.R.T (testy disku, statistiky, ..)

*smartd* je nadstavbou smartctl (pravidelne spousteni testu, ..)

## 5.5 Diskova pole (RAID)

RAID je technologie umoznujici z vetsiho poctu (levnejsich a ne prilis spolehlivych, vykonnych) disku vytvorit jeden disk, který je rychlejsi a spolehlivejsi.

### Muze byt implementovan:

- hardwarove (do rozsirujici karty pripojime nekolik disku a ta implementuje RAID),
- subsystemem v jadře,
- nektère souborove systémy mají implementaci RAID v sobě.

Ruznych typu RAID je nekolik (tzv. raid levels).

### 5.5.1 RAID 0

- data jsou rozložena po dvou či více discích, ale každý datový blok je uložen jen na jednom disku (např. dva disky, 0 a 1, první datový blok [sektor, skupina sektorů] je na 0, druhý na 1, třetí na 0, ...)
- vyšší efektivita čtení či zápisu,
- je možné paralelně číst či zapisovat (do více disků)
- prudce snižuje spolehlivost - pokud selže jeden disk, přijdu o data na něm

### 5.5.2 RAID 1

- disk mirroring, pro 2 a více disků,
- všechny bloky dat se zapisují na všechny disky,
- možnost číst a zapisovat paralelně,
- vyšší spolehlivost (data jsou na všech discích)

### 5.5.3 RAID 2

- nejsložitější, proto se příliš nepoužívá,
- používá zabezpečovací Hemingovy kódy,
- k určitému počtu datových disků je určen počet zabezpečovacích disků,
- data se ukládají na datových discích na úrovni bytu, k nim se dopocítávají zabezpečovací kódy (např. 4 datové - 3 zabezpečovací),
- byty dat se rozloží do všech disků (ofč ty se musí převést do bajtu a sektorů a zapisuje se to po sektorech)
- jediný RAID, který umí detekovat chyby, nektère i sám opravit, dokonce umí i zjistit, který disk selhal

#### 5.5.4 RAID 3

- jednodussi zabezpeceni nez RAID 2, v podobe paritnich bytu,
- rozklada data po bajtech ci skupinach bajtu, ktere zabezpecuje partinim zabezpecenim (napr. 4 disky - 3 datove a 1 paritni).

#### 5.5.5 RAID 4

- je analogie (tak jako RAID 3, akorat ..),
- provadi se rozkladani na urovni bloku-sektoru,
- nevychoda u RAID 3 i 4 je pretizeni paritniho disku - pri zapisu/cteni se vzdy pracuje s paritnim diskem (a datovym) - na paritni disk se zapisuje tolikrat casteji, kolik mam datovych disku, tzn. vetsi pravdepodobnost selhani

#### 5.5.6 RAID 5

- prakticky se uz pouziva,
- funkce paritniho disku neni vyhrazena pro jeden disk, ale mezi disky tzv. rotuje,
- napr. v konfiguraci se 4 disky, prvni 3 datove bloky se ulozi na 3 disky, na poslednim bude parita, pro dalsi trojici se ulozi na 3. disk, pro dalsi na 2., dalsi na 1., a potom zase na posledni, apod. .. = rovnomerne zatizeni disku,
- diky parite jsme schopni opet detekovat a korigovat chybu v jednom disku (pocet bitu neni sudy - chybi tam parity bit),
- parita se pocita dle sektoru (prvni bit 1. sektoru, prvni 2. sektoru, ..),
- pokud selze vice disku, nelze dopocitat bity (data)

#### 5.5.7 RAID 6

- parita se uklada 2x,
- dokaze se vyrovnat se selhanim az 2 disku,
- vetsi redundance dat (obetuji se 2 disky jako parita)

RAID je mozne vytvorit i na jednom fyzickem disku (na kterem jsou logicke disky).

## 5.6 Opravy chyb u paritních disků

- paritní disky používají RAID 3 - 6,
- jakmile člověk určí disk, který selhal, je možné zreprodukovat jeho obsah,
- příklad: 4 disky, 1 paritní, třetí datový selze
  - první byty v datových jsou 010 (potom v paritním aby byl sudý počet je 1), další byty jsou 111 (lichá parita, do paritního disku se doplní 1 na sudou), další jsou 011 (suda - v paritním je 0)
  - selže třetí disk, vymění se za nový, prázdný
  - dopocítají se data opět na sudou paritu: mám první byty 01? a v paritním 1 - aby byla suda, v novém disku musí být 0, další byty 11? a v paritním 1 - v novém musí být 1, apod. ...

### definice: (pro 5.5.x)

*RAID* = Redundant Array of Independent Disks

*parita (bitu)* je sudost/lichost bitu, počet sudých/lichých 1 bitu

*parity bit* zn., že na MSB se přidá 1 pokud počet 1 (bitu) je lichý

## 5.7 Uložení dat na disku

Disková jednotka neumí pracovat s nicím menším než sektor, ale typický OS si sektory nějak seskupí (do větší jednotky) a neumí pracovat s nicím menším, než je alokační blok.

### Logická a fyzická následnost:

- 1 alokační blok se mapuje fyzicky za sebou na diskovém prostoru,
- více alokačních bloků již nemusí být fyzicky na disku za sebou (filesystem se však snaží o to, aby tomu tak bylo)

### definice:

*alokační blok* neboli cluster je skupina pevného počtu sektorů, typicky mocnina 2 (nejméně  $2^0 = 1$  alokační blok), pro sektory v rámci alokačního bloku je zaručeno, že jdou za sebou logicky i fyzicky (na disku) v souboru, dále je to nejmenší jednotkou diskovou diskového prostoru, se kterým bezne pracuje jádro (filesystem, uživatel).

## 5.8 Fragmentace

### 5.8.1 Externi fragmentace

Rozumíme jev, který vzniká v pamětech postupným obsazováním a uvolňováním paměti, kdy v paměti vzniká sekvence oblastí, které jsou volné a použité (a použité různými soubory).

#### Příklad externí fragmentace:

- na disku vytvořím soubor 1, zabírá určité místo,
- poté další soubor 2,
- poté soubor 1 chci zvětšit, tak se soubor 1 rozdělí na 2 části - soubor 1.1 (původní místo kde byl - před s2) a soubor 1.2, který bude za souborem 2,
- stejným způsobem zvětším soubor 2 a vznikne sekvence s1.1, s2.1, s1.2, s2.2,
- nyní se rozhodnu smazat první soubor a budu mít sekvenci volné místo, s2.1, volné místo, s2.2 == externí fragmentace.

Externí fragmentace je i na plně obsazeném disku, kde stačí, aby byl disk obsazen soubory nespojitě (tzn. jeden soubor je rozdělen do více částí, není uložen na jednom místě, např. s1.1, s2, s1.2 nebo viz příklad výše).

#### Negativní dopady externí fragmentace:

- na disku za určitých okolností (v běžných FS nevznikají) mohou vzniknout části prostoru, které jsou již dále nevyužitelné, protože jsou příliš malé (tldr vznik volných úseků, které nejdou využít)
  - okolnosti (při kterých vzniknou nevyužitelné části prostoru): při alokaci diskového prostoru spojitě (na míru souboru či jeho částem, nepřidělování po jednotkách pevné velikosti) a navíc budu mít dolní mez určující velikost diskového prostoru tak, aby byl použitelný (může vzniknout v souvislosti s tím, že do použitých diskových oblastí si mohou ukládat pomocné informace, k čemu se používají - pokud bude informace větší než "volná díra" - nepoužitelná) - mám na disku bloky volného místa o požadované velikosti (1GB, soubor, 0.5GB), ale protože chci ukládat spojitě (soubor o velikosti 1.5GB), nelze takové místo využít
  - vznikne nespojitě rozložený soubor (viz příklad) a je nutné si pamatovat v pomocných datech = metadatech informace o tom, kde jednotlivé části souboru jsou (ukládají se na místa, kde jednotlivé části jsou, "odkazují" se na další metadata - další části smazaného souboru),
- čím více částí souboru - tím více metadata - čím více fragmentované - tím více je přístup na data pomalejší (u HDD se čeká navíc na natocení hlaviček a rotaci disku)

#### Souborové systémy se snaží negativní dopady fragmentace minimalizovat:

- rozložení souboru po disku (snáze ukládat soubory na disk tak, aby nebyly nutné za sebou, ale bylo mezi nimi volný prostor),
- používání předalokace (uživatel si požádá filesystem o vymezení určitého prostoru na disku, např. databáze),
- odložená alokace (=allocate-on-flush, filesystem nezapíše ihned po změně souboru, ale chvíli počká - počítá s tím, že uživatel bude chtít menit soubor "za chvíli" znovu - až nebude delší dobu docházet ke

zmenam, pote hleda vhodny volny prostor)

Pri (intenzivnim) beznem pouzivani disku se vsak fragmentaci nelze vyhnout. Pokud by byla fragmentace prilis vyrazna, je mozne pouzit defragmentacni nastroje, které provadeji kopirovani, presouvani casti souboru a reorganizaci diskoveho prostoru tak, aby se fragmentace odstranila - casove narocna operace.

Prvniho negativniho dopadu externi fragmentace (nevyuzitelne a prilis male oblasti) je mozne se zbavit pri pouzivani alokaci po jednotkach pevne velikosti - alokacni bloky - vzdy je ale snaha alokovat spojite (v horsim pripade alokuji nespojite, pokud to nejde)

### **5.8.2 Interni fragmentace**

Nespojita alokace po jednotkach pevne velikosti (alokacni bloky) ma vyhodu, ze redukuje dopady externi fragmentace, ale potom vytvari interni fragmentace. Interni fragmentace se obvykle toleruje.

#### **Priklad interni fragmentace:**

- chci alokovat soubor o velikosti 9 000 B,
- mam 4 KiB velke alokacni bloky,
- potom je nutne alokovat 12 KiB pro tento soubor,
- ty zbyvajici 3 KiB v poslednim alokacnim bloku zustanou nevyuzite.

Existuje nekolik malo filesystemu, které se snazi resit interni fragmentaci (ReiserFS, ZFS) pomoci techniky zvane 'tail packing' ('zbavovani ocasku"souboru) - vice souboru muze pouzivat 1 fyzicky alokacni blok (zaplni se volne misto). Vetsine filesystemu toto vsak nepodporuje.

## 5.9 Prístup na disk

(viz 2.5.5) Proces když chce načítat/zpracovávat data, zavolá službu k tomu určenou (read, write, .. - může být zabaleno i v nějakém knihovním volání, např. scanf zavolá read), dojde k předání řízení jádru, dostane se ke slovu jádro, podívá se do cache, pokud tam ta data má, předá je, pokud ne, musí je načíst z disku - s diskem komunikuje přes I/O porty nebo paměťové mapované I/O porty, disku se předávají příkazy přes jeho rozhraní (ATA disk - ATA příkazy), jdou z filesystemu přes ovladač příslušného disku (pote to prochází sbernicemi), ten komunikuje s radicem disku - disk dostane příkaz, jakmile dobehne operace, disk posle prerušení na procesor, tam se dostává ke slovu jádro, to zpracuje prerušení a zachová se podle něj (úspěch - předá data, chyba - zpracuje ji).

### definice:

*ovladac* je software, který umí komunikovat s určeným typem zařízení, jina definice viz. 2.4

## 5.10 Planování přístupu na disk

Součástí jádra je subsystem nazývaný plánovač diskových operací, který shromažďuje požadavky od filesystemu (načtení, zápsání dat z/do disku). Plánovač si ukládá požadavky do svých plánovacích front, požadavky případně přeuspořádává a předává dal ovladači či radici disku k realizaci.

Plánovač se snaží minimalizovat rezii disku.

Jednou ze strategií přeuspořádávání požadavků (u HDD) je použití **vytahového algoritmu (elevator, SCAN algorithm)**:

- snaha, aby se hlavička disku plynule pohybovala od středu k okraji a zpět a vyřizovat požadavky dle pohybu hlavičky,
- modifikace SCAN algoritmu je například Circular SCAN, kdy se požadavky vyřizují pouze při jednom směru,
- další modifikace jsou LOOK a C-LOOK, kde se hlavička nepohybuje od středu k okraji, ale pouze v tom rozsahu, kde je potřeba provádět operace.

**Plánovač se může snažit více operací sloučit do jedné operace** (např. operace v rámci jednoho bloku se sdruží):

- takové kroky mají význam i u SSD,
- snaha vyřizovat požadavky jdoucí od jednotlivých uživatelů (procesů),
- implementace priorit (prioritnější proces - požadavky se vykonají dříve),
- snaha odkladat operace tak (v naději), že je bude pote možné sloučit,
- snaha implementovat časová omezení na dobu čekání požadavku,
- může implementovat paralelizaci požadavků předávaných do diskového subsystemu (modernější a velmi výkonné SSD umí resit operace paralelně).

### linux:

pro zjištění, jaký plánovač používáme se stací podívat do `/sys/block/<devname>/queue/scheduler`

## 5.11 Logický disk

V pocitaci je možné mít vícero fyzických disků, které je dále možné rozdělit na logické disky a konkrétní souborové systémy je možné instalovat na logické disky. Pro správu a vytváření logických disků lze použít programy cfdisk, disk, gparted, ..

### 5.11.1 Způsob uložení informací o diskových oblastech na disku

- MBR
  - v prvním (nultém) sektoru byla tabulka obsahující rozdělení na 1-4 primární partitions
  - pokud bylo nutné použít více partitions, potom místo primární se nahradila rozšířenou diskovou oblastí, která se dále mohla rozdělit na podoblasti zvané logické diskové oblasti, každá z nich popsána formou zretezeného seznamu, EBR
  - používane u starších PC
- GPT
  - je tabulka (pole) o až 128 odkazech na jednotlivé diskové oblasti,
  - stejný vyhrazený prostor jako u MBR

### 5.11.2 LVM

- správce logických oblastí,
- umožňuje pokročilejší tvorbu logických disků a
- do logického disku přidávat fyzické disky (za běhu),
- LVM může být buď přímo ve filesystému nebo v části jádra (mezi filesystémem a plánovačem).

### 5.11.3 Různé typy souborových systémů

- fs (první fs na unixu), ufs, ufs2,
- ext2, ext3, ext4,
- btrfs (inspirovan ZFS),
- ReiserFS, HSF+/APFS (Mac OS X), XFS, JFS, HPFS,
- FAT, VFAT, FAT32, exFAT (rodina FAT vznikla v MSDOS, poté používány ve Windows - velmi jednoduché a široce podporované),
- F2FS (fs pro efektivní práci se SSD), ISO9660, UDF, Lustre, GPFS (clustery, superpocítace),
- ZoneFS (ZFS).

Po koupi nového disku a rozdělení na logické disky je nutné se rozhodnout, jaký souborový systém na příslušném logickém disku bude používán - je nutné disk **zformátovat** pro použití. Dříve se používalo i nízkourovňové formátování (staré disky s nestabilním magnetickým záznamem).



#### 5.11.4 Chyby disku (souvislost s FS)

Na disku mohou vznikat chyby beznym opotrebenim, nevhodnym vypnutim napajeni, je zapotrebi opravit ridici struktury souboroveho systemu (program fsck - kontroluje konzistenci filesystemu nebo zurnalovani, copy on write, soft updates, ..).

#### 5.11.5 Dalsi typy souborovych systemu

**Virtualni souborovy system (VFS)** je vrstva, která v jadře zastresuje vsechny ostatni souborove systemy z toho duvodu, aby jine subsystemy jadra nemusely pracovat specialnim zpusobem s ruznymi souborovymi systemy.

Existuji take ruzne sitove souborove systemy, treba NFS.

#### Specialni souborove systemy

- neukladaji zadna data, obsah neni nikde na disku ani neexistuje zadna specialni cast pameti
- zpristupnuji napr. aktualni stav jadra - adresar /sys, sysfs filesystem,
- procfs filesystem v adresari /proc zpristupnuje informace o bezicich procesech (ale i o nejakych castech stavu jadra),
- tmpfs zase vytvari souborovy system v RAM.

#### definice: (pro 5.11.x)

*logicky disk* je taky diskova oblast, partition

*MBR* = master boot record

*EBR* = extended boot record

*GPT* = GUID Partition Table, GUID = Globally Unique Identifier

*LVM* = Logical Volume Manager

*formatovani* zn., ze se nainstaluji metadata (ridici data) souboroveho systemu do prislusne diskove oblasti, v ramci toho se mohou vymazat vsechna data na dane oblasti

## 6

**Šestá přednáška:** pokračování Spravy souborů. Zurnalování, jeho implementace a alternativy, Copy-on-write, Klasický UNIXový systém souborů FS, i-uzly, kde a jak jsou data uložena, počty odkazu, limit maximální velikosti souborů, výhody a nevýhody FS, jiné způsoby organizace souborů, EXT4, NTFS, Organizace volného prostoru na disku, deduplikace, typy souborů v UNIXu, adresář, montování disku

### 6.1 Zurnalování

Je technika založena na vytváření zurnálu.

- souborové systémy se zurnalem jsou třeba ext3, ext4, ufs, XFS, JFS, NTFS, ..
- zurnalování umožňuje spolehlivější (nikdy nemáme obecně zajištěno, že se nic špatného stát nemůže) a
- rychlejší návrat (než nějaké utility) do konzistentního stavu po chybách
- data obvykle zurnalována nejsou (velká režie), ale mohou být
- závisí na tom, že operace, které zurnalování implementují, se provedou ve správném pořadí – nutnost spolupráce s plánovačem, takže disky si samy data přeuspořádávají (nelze nijak ovlivnit)

#### Zurnál:

- zápis zurnálu je předřazený,
- vytváří se v něm cyklický prepisovaný buffer,
- předřazenost zápisu do zurnálu mi zaručí, že operace pokryté zurnalováním jsou atomické - vytváří transakce

Kompromis mezi zurnalováním a nezurnalováním dat je **předřazení zápisu dat na disk před zápisem metadat do zurnálu** (a následně zápis ostrých metadat na disk). Příklad:

- zapisují do souboru - buď vytvořím zcela nový nebo ho zvětšuji (typické způsoby zápisu),
- při zvětšování se nejprve zapisují data na disk za existující data (bez poznamenávání informace o tom, že se soubor zvětšuje),
- pokud operace selže, soubor zůstane v původním stavu (díky neupraveným metadatům původního souboru),
- teprve až data budou na disku, tak se do zurnálu zapíše informace o zvětšování souboru,
- poté se změní metadata souboru (a uživatel se k datům dostane),
- při selhání napájení v moment, kdy jsou metadata v zurnálu, ale ne na disku, je možné tyto metadata obnovit

#### Proces mazání souboru na disku:

- odstranění záznamu z adresáře,
- uvolnění uzlu (metadat souboru),

- uvolnění oblasti použitých tím souborem

#### **definice:**

*zurnal* je speciální soubor či speciální oblast na disku sloužící pro zaznamy modifikovaných metadat (dat o datech), případně i dat před jejich zápisem na disk (v podobě bezných dat)

*předřazený* znamená, že zápis do zurnálu se provede před ostrým zápisem "užitečných" dat (či metadat) na disk  
*atomicke operace* zn., že buď operace uspěje celá (všechny dílčí kroky) nebo neuspěje vůbec (žádný dílčí krok)

### **6.1.1 Implementace zurnalování**

Existují 2 základní přístupy k implementaci zurnalování.

#### **REDO:**

- implementace na základě dokončení transakcí,
- používá např. ext3, ext4,
- sekvence dílčích operací (vytvářející tu operaci, kterou chci provést) se zapisuje do zurnálu (začátek, konec transakce, kontrolní součet),
- poté se operace provádí na disku,
- po úspěšném dokončení se transakce ze zurnálu uvolní,
- při selhání a poté zotavení se systém podívá do zurnálu, podívá se po neuvolených transakcích, jestli jsou celé - počáteční a koncová značka, jestli sedí kontrolní součet - pokud vše sedí, tak systém provede všechny operace znovu

#### **UNDO:**

- implementace na základě anulace transakcí,
- v kombinaci s REDO se používá v NTFS,
- prokládá záznam dílčích operací (které se mají provést) do zurnálu a následně jejich provedení na ostrých datech (zaznamená dílčí operaci - provede ji),
- proběhne celá transakce - záznam ze zurnálu se uvolní,
- při chybě se eliminují všechny nedokončené transakce (všechny provedené dílčí kroky se musí vrátit - vrátí se disk do původního stavu)

Při implementaci zurnalování je klíčové **dodržení pořadí kroku, ve kterém se provádějí**. (U REDO např. je nutné, aby se nejprve zapsaly sekvence operací do zurnálu a teprve poté se prováděly operace na disku) Pokud tato sekvence nebude dodržena, zurnal nebude správně fungovat.

### 6.1.2 Copy-on-write

Je alternativa k zurnalovani pouzivana naprikad v ZFS (OpenZFS), BTRFS, ReFS (Resilient File System).

- kopie pri zapisu,
- zalozeno na tom, ze vsechna nova data / metadata se zapisi na disk, a pote se zpristupni,
- vyuziva se pritom toho, ze obsah disku je popsam hierarchickou stromovou strukturou,
- zmeny se provadeji v souladu s touto strukturou (od listu ke koreni),
- pokud vypadne napajeni v moment, kdy data (bloky) nejsou jeste zpristupnena, data nejsou dostupne z korene stromu a jakoby se nic nestalo,
- pokud se mi tyto data podari zapsat uspesne, postupne zacnu upravovat vsechny uzly vedouci az ke koreni a zpristupnim nova data,
- teprve po modifikaci korenu se stanou zmenena data (uzly) dostupne
  - koren je nutne zabezpecit, aby nedoslo k chybe pri zapisu do nej
  - stary korenovy uzel se neprepisuje, ale pouze se tam zapise nova verze korenoveho zaznamu (s casovym razitkem)
  - soucasne tam bude zabezpecovaci kod (kontrolni soucet),
  - pokud dojde ke krachu systemu, staci si nacist vsechny koreny, zkontrolovat kontrolni soucty, vybrat si vsechny, kde sedi kontrolni soucty, s nejnovjsim casovym razitkem a tyto pouziju (pokud dojde k chybe nez se stihne zapsat novy koren, pouzije se ten puvodni)

#### Vyhodami copy-on-write jsou:

- snimky souboroveho systemu (zapamatuje se pouze korenovy uzel - minimalni rezie),
- klony souboroveho systemu (vytvori se pozadovany pocet kopii korenoveho uzlu),
- vyhodou je, ze nezmenene uzly (data) a listy budou na disku pouze jednou, pouze je nutne si pamatovat zmenene uzly / listy a cestu ke koreni + koren (kopie stale ukazuji na stejny strom).

#### definice:

*stromova struktura* (copy-on-write) je vyhledavaci strom, který popisuje veskerý obsah disku, typicky se v něm vyhledava na zaklade unikatni identifikace souboru

*adresare* (copy-on-write) jsou specialni soubory ulozeny na disku, které jsou dostupne ve strome (stromove strukture)

*snimek souboroveho systemu* ulozi se obsah disku tak, ze je mozne se k nemu pozdeji vratit

*klon souboroveho systemu* je vytvoreni 2 kopii souboroveho systemu a od daneho okamziku je mozne s kazdou kopii pracovat samostatne (napr. pri vetsim poctu VM, kdy vsechny VM sdili stejny pocatecni obsah disku, ale od urciteho momentu kazda VM chce obsah menit samostatne)

### 6.1.3 Dalsi alternativy zurnalovani

#### Soft updates:

- pouziva se v UFS (FreeBSD systemy),
- filesystem se snazi sledovat zavislosti mezi tim jaka data a metadata se meni,
- uzpusobuje poradi zapisu metadat a dat na disk tak, aby v jakemkoli okamziku byl obsah na disku konzistentni (az na moznost vzniku "garbage")

#### Log-structured file systems:

- logovací souborove systemy (= strukturovane jako log),
- pouziva se v LFS, UDF, F2FS,
- cely souborovy system ma charakter jednoho velkeho logu,
- ktery se zapisuje v cyklicky prepisovane pameti napric celym diskem,
- posledni obsah disku je vzdy dostupny pres posledni zaznam (a odkazy, ktere z nej vedou),
- pri provadeni zmen se pridaji data napr. za aktualni konec vyuziteho diskoveho prostoru, prida se k tomu zaznam (o tom co se zmenilo), zpristupni se data z posledniho zaznamu.

#### definice:

"garbage" je cast prostoru na disku, ktera se tvari jako obsazena, ale neni

*logem* rozumime soubor, ktery obsahuje zaznamy o zmenach (nebo: log = zapis o zmenach)

## 6.2 Klasický UNIXový systém souboru (FS)

Je původní filesystem unixu (70. leta). Vyvinul se z něj UFS, z něj zase EXT2, 3 (pote vznikl i EXT4).

**Souborový systém byl rozčleněn (na úrovni logických disku) na:**

- boot blok - obsahoval informace (kod, část kodu) potřebné pro zavedení při startu,
- super blok - informace o souborovém systému (typ, verze, velikost, počet i-uzlu, volné místo, kořenový adresar, volné i-uzly, ..),
- tabulka i-uzlu - tabulka (pole n i-uzlu) použitá s popisy souboru,
- datové bloky - data souboru, metadata (pomocné adresovací bloky).

**Základní rozložení FS bylo zmodifikováno v navazujících filesystemech:**

- datové bloky byly rozděleny do skupin,
- každá skupina měla svoje i-uzly,
- důvodem byla lepší lokality, prostorová blízkost dat a metadata (typicky při práci se soubory jsou nutné i jeho metadata),
- poté tedy ta struktura vypadala takto: boot blok, super blok, usek i-uzlu, usek dat, usek i-uzlu, usek dat, ..

**definice:**

*i-uzel* je základní datová struktura reprezentující každý jeden soubor v typických UNIXových systémech (pozn. při formátování se určí dopředu maximální počet souboru, které na diskovém oddílu budou existovat)

### 6.2.1 i-uzel

Základní datová struktura popisující soubor v UNIXu (nebo viz definice hore). Ke každému souboru musí být i-uzel. Ten obsahuje metadata o souboru:

- stav i-uzlu (alokovaný, volný)
- typ souboru (obyčejný, adresar, zařízení, pojmenovaná roupa, ..),
- délka souboru v bajtech,
- čas mtime (poslední modifikace dat - zápis), atime (poslední přístup - čtení), ctime (poslední modifikace i-uzlu),
- UID, GID,
- přístupová práva (číslo, např. 0644 = rw-r--r--),
- počet pevných odkazů (neboli jmen souboru),
- informace o tom, kde se nachází data o souboru (tabulka odkazů na datové bloky a další informace nebo odkazy na pomocné bloky s dalšími metadaty, např. ACL, extended attributes, dtime - údaj o smazání souboru, ..)

Jmeno souboru neni v i-uzlu, ale je ulozeno v adresari.

#### definice:

*UID* je cislo identifikace vlastnika

*GID* je cislo identifikace skupiny

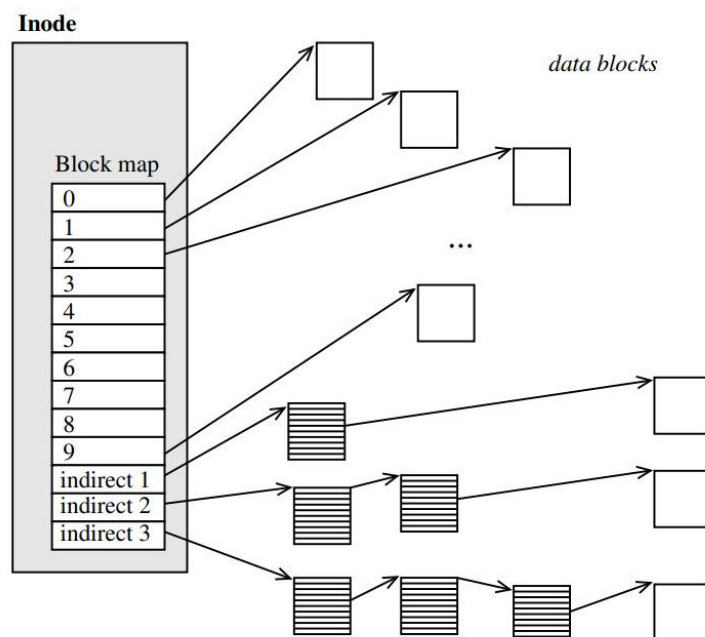
*ACL* zn. Access Control List, pristupove seznamy rozsirujici zakladni UNIXova prava tak, ze je mozne priradit konkretni prava ke konkretnim uzivatelum

*extended attributes* zn. rozsirene atributy, s jakymi specifickymi prave se muze soubor napr. spoustet

*ctime* je posledni zmena i-uzlu, vyuzitelne napr. pokud se zfaluje mtime, posledni zmena souboru se pozna prave podle ctime

### 6.2.2 Kde a jak jsou ulozena data

- v i-uzlu je rada primych (az 10, novejsi unixove fs maji 12) i neprimych odkazu na data,
- prime odkazy odkazuji na alokacni bloky na disku,
- pokud je potreba vice odkazu, pouzije se neprimy odkaz prvnι urovne, ktery odkazuje na specialni alokacni blok neobsahujici data, ale dalsi prime odkazy na data,
- pokud nestaci ani to, pouzije se neprimy odkaz druhe urovne, ktery odkazuje na pomocny adresovaci blok, ktery obsahuje dalsi neprime odkazy 1. urovne, ktere odkazuji na dalsi prime odkazy (a ty odkazuji na data) - vznikna strom
- pokud ani to stacit nebude, pouzije se adresovaci blok 3. urovne, vedouci na adresovaci bloky s neprimymi odkazy 2. urovne, kazdy z nich na bloky s odkazy 1. urovne, ty vedou na prime odkazy a ty na data.



Obrázek 1: z prezentace IOS: Souborove systemy, slide 23 - odkazy v i-uzlech

### 6.2.3 Počty odkazu

- nepřímý odkaz 1. úrovně - při 4 KiB clusteru je to 1024 odkazů (1 odkaz = 4B) = 1024 datových bloků,
- nepřímý odkaz 2. úrovně - při 4 KiB clusteru je to  $1024^2$  odkazů = stejný počet datových bloků,
- nepřímý odkaz 3. úrovně - při 4 KiB clusteru je tam  $1024^3$  odkazů = stejný počet datových bloků

### 6.2.4 Limit maximální velikosti souboru

Počet přímých odkazů nepřímého odkazu 3. úrovně je dán maximální počtem bloků, které je možné v tomto souborovém systému uložit. Teoretický limit velikosti souboru je tak:

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

kde  $D$  je velikost bloku v bajtech (bezpečné 4096B),  $M$  je velikost odkazu na blok v bajtech (bezpečné 4B),  $N = D/M$ , je počet odkazů v bloku.

Toto omezení velikosti je pouze jedním z omezení, které velikosti souboru omezují. **Další omezení jsou dana:**

- dalšími datovými strukturami a typy, které používá FS (např. datový typ délky souboru v bajtech v i-uzlu),
- strukturami VFS (veskera práce s jakýmkoli filesystemem musí projít přes VFS),
- rozhraním jádra,
- architekturou systému (32b - velikost souboru bude 32b číslo + MSB je použit pro indikaci chyby [-1 bit pro data] - soubory maximálně do 2 GiB nebo dnes bezpečná architektura 64b - 64b velikosti)

**Existuje Large File System Support**, kde ve 32b systému se nahradí všechny údaje kde se pracuje s velikostí větším datovým typem - podpora souborů *větších jak 2 GiB*.

#### linux:

`du [soubor]` vypíše zabrané místo v blocích vc. rezie (metadat)

`ls -l [soubor]` vypíše velikost souboru v bajtech (pouze užitečná data)

`df` vypíše volné místo na discích

`ls -i [soubor]` zprístupní číslo i-uzlu souboru

`is -e /dev/... n` - vypis i-uzlu n na /dev/...

`dumpe2fs` - základní informace o souborovém systému ext2,3,4

`/dev/zero` je soubor typu zařízení generující proud 0

`dd if=[source] of=[dest]` je nízkourovňové kopírování



### 6.2.5 Vyhody a nevahody architektury FS

Neboli proc bylo navrzen FS prave tak, jak je. Architektura FS je totiz ovlivnena snahou o minimalizaci jejich rezie s relativne pomalymi disky (HDD, SDD), jedna se zejmena o bezne operace se soubory, jako je pruchod soubourem (otevru - prochazim od zacatku do konce) ci presun (seek), zvetsovani ci zmensovani (vc. mazani) souboru.

**Je nutne vzit do uvahy, z jakych (mikro)operaci se tyto operace sestavaji.** Jsou to operace:

- vyhledavani adresy prvnio nebo urcitetu bloku souboru,
- vyhledavani nasledujicich bloku,
- pridani ci odebrani bloku,
- alokace ci dealokace volneho souboru (informace o volnych oblastech, minimalizace externi fragmentace)

FS a jeho naslednici UFS, EXT2, EXT3 (EXT4 uz neni jeho naslednik!) predstavuji kompromis s ohledem prevazne na male soubory. (tyto fs funguji skvele pro male soubory - u vetsich souboru je nutne prochazet ci menit vetsi objem metadat)

Jistou optimalizaci pouzivanou i u klasickych filesystemu pro male soubory je ulozeni dat primo do i-uzlu. (pokud se tam data vlezou).

#### **definice:**

*symbolicky odkaz* je soubor odkazujici na jiny soubor (pouziva ulozeni dat primo do i-uzlu)

*rychle symlinky* maji data v i-uzlu

*pomale symlinky* maji data mimo i-uzel

## **6.3 Jine způsoby organizace souboru**

### **6.3.1 Kontinuální uložení**

- neboli spojitě uložení souboru na disku,
- na disku je jeden spojitý úsek dat reprezentující soubor,
- výhodami jsou rychle vyhledání adresy 1./určitého bloku nebo vyhledávání následujících bloků,
- nevýhody: soubory nebude možné jednoduše zvětšovat pokud budou příliš blízko u sebe (bude nutné je přesunout na jiné volné a větší místo, pokud to půjde či provést defragmentaci a poté zvětšit soubor)
- nepoužívá se příliš (kvůli své nevýhodě)

### **6.3.2 Zrežované seznamy alokačních bloků**

- každý alokační blok obsahuje své (užitečné) data a na konci obsahuje odkaz na následující alokační blok,
- výhodami jsou rychlý přístup na začátek či průchod daty,
- nevýhodou je přesun na náhodné místo v souboru - nutnost přejít celý soubor až po daný blok (1 GiB soubor, chci poslední blok - musím přejít celý),
- další nevýhodou je rozptýlení metadat po celém disku - při drobné chybě na disku přijdu o data (tedy i metadata, kde jsou odkazy na následující bloky) a dojde k velké ztrátě dat (všechna data "za" ztracenými daty jsou nepřístupná),
- není příliš vhodná, nicméně se používá v souborových systémech FAT

### **6.3.3 FAT**

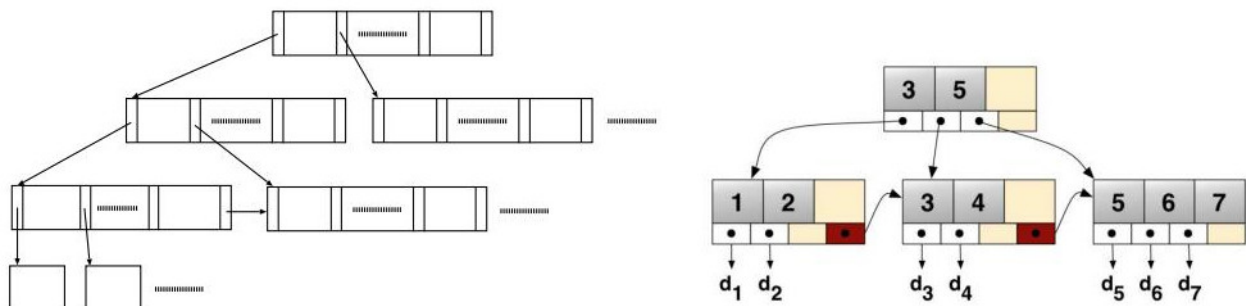
- File Allocation Table,
- od zrežovaných seznamů se liší tím, že seznamy popisující rozložení souboru na disku jsou uloženy v separátní oblasti na disku (tzv. FAT),
- kde jsou tato data koncentrována - rychleji se prohledávají, lze vytvořit tak více kopií FAT (prevence okamžité ztráty dat při chybě),
- stále vznikají problémy s rychlostí při náhodném přístupu (stále jde o zrežovaný seznam)
- tabulka je pole, které obsahuje pro každý blok na disku 1 položku, každá položka obsahuje odkaz na další blok/položku,
- používá se i dodnes (a je to velmi rozšířené), protože je to jednoduché (např. vestavěné systémy)

### 6.3.4 B+ stromy

- jsou datovou strukturou převzatou z databazových systému,
- mají dva typy uzlu - vnitřní a listové,
- vnitřní uzly jsou koreň, jeho následníci kromě listových, obsahují odkaz na následníka a vyhledávací klic,
- listové uzly také obsahují odkazy a vyhledávací klíče, odkazy vedou na data na disku, poslední odkaz na posledním listu odkazuje na list na stejné úrovni (jsou tak propojeny lineárním seznamem),
- používají se za účelem popisu rozložení dat na disku (obsah souboru, poté vyhledávací klic bude offset - číslo logického bloku v rámci souboru) nebo se používají pro adresare (klíče budou jména souborů) nebo pro popis celého obsahu disku (klic je dvojice i-uzel a posuv souboru)

#### Vyhledávání v B+ stromu:

- při hledání klíče (k) se podívám, zda je klic menší jak klic k<sub>0</sub>, pokud ano, půjdu níže, kde je k<sub>0</sub>, pokud ne, zjistím, jestli je klic mezi k<sub>0</sub> a k<sub>1</sub>, pokud ano, jdu druhým směrem, opakují po k<sub>n</sub>,
- pokud jsem níže, opakují to same co vys až nedojdu k listovému uzlu,
- zde hledaný klic najdu nebo zjistím že v této struktuře klic není
- poté mám odkaz na datový blok,
- v případě že chci číst dál, tak jdu lineárně po sobě po následujících listových uzlech



Obrázek 2: z prezentace IOS: Souborové systémy, slide 27 - B+ strom

### Prace s B+ stromy:

- jsou zde limity jak moc/malo mají být uzly zaplneny (strom se udržuje vyvážený) - pro uzly s  $m$  odkazy máme klíče 0, 1, .. až  $m - 2$  klíču (odkazu je o 1 méně než klíču + číslování od 0),
- pokud je strom tvořen solo korenem - nejméně může mít 1 odkaz, maximálně  $m - 1$  odkazu (poslední odkaz je použit jako ukončovač seznamu listů),
- pokud to není solo koreň, tak má nějaké následníky, minimálně jich má tak 2, maximálně  $m$ ,
- vnitřní uzel má tak  $\frac{m}{2}$  (zaokr. nahoru) až  $m$  odkazu, list  $\frac{m}{2} - 1$  (opět  $\frac{m}{2}$  zaokr. nahoru) až  $m - 1$  odkazu,
- vložení:
  - nejprve projdeme stromem od kořene k listům,
  - najdeme kam chceme vložit,
  - podíváme se, zda má list volný odkaz,
  - pokud ano - použijeme ho, pokud ne - list se rozštěpí na 2 poloviny a podívám se o úroveň výš, zda je možné namísto 1 listu linkovat 2 listy,
  - pokud ano - přidá se odkaz, pokud ne - nadřazený uzel se musí rozštěpit a postupovat o úroveň výš,
  - ... štepí se strom až případně se rozštěpí kořen a strom bude mít 2 kořeny
- rusení:
  - se opět od listové úrovně, tak, že se zruší odkaz v listu,
  - zkontroluje se, zda je uzel zaplněn v rámci daných limitů,
  - pokud ano - gut, pokud ne - podívám se na sousední uzly a pokud se provede přerozdělení tak, aby byly všechny uzly naplněny v rámci limitů,
  - pokud se to nepodaří, tak dojde ke sloučení listů,
  - posunu se o úroveň výš, zruším jeden odkaz, zkontroluji opět limity, zopakuji to samé,
  - ... až se může stát, že se zruší i kořen

B+ stromy a jeho varianty jsou používány pro popis diskového prostoru v filesovech jako XFS, JFS, ZFS, Btrfs, ReFS, .. v omezené podobě tzv. stromu extentů v EXT4, podobná struktura je i v NTFS

### definice:

*solo koreň* = jediný kořen

### 6.3.5 Extent

- pouziva se ke zrychleni prace s velkými soubory,
- umožnuji zmensit objem metadat (je mozne rict, ze nektère alokacni bloky jsou ulozeny pospolu = vytvari extent), potom budu popisovat rozlozeni souboru po extentech (ne po alokacnich blocich),
- prinese lepe vyvazene indexove struktury,
- rychlejsi mazani,
- jsou pouzity snad ve vsech systemech s B+ stromy,
- B+ strom se snadno kombinuje s extenty (neplati pro klacicky Unixovy strom - protoze ve stromu jsou explicitne ulozene vyhledavaci klice, ale Unixovy nema v zadnych strukturach [i-uzel] ulozenou velikost datovych bloku [protoze jsou vsechny stejne a konstantni])

Pokud pouzivame B+ stromy, tak rychlemu spojitému pruchodu pomaha listova uroven, pokud je prolinkovani listu pouzito. Pro male soubory muze predstavovat B+ strom zbytecnou rezii (data se bud ulozi primo v i-uzlu nebo nebo z nej mame prime odkazy na extenty z i-uzlu [do max. 4 extentu])

#### **definice:**

*extent* je jednotka vystavena na bloky, posloupnost promenneho poctu alokacnich bloku (jdoucich za sebou logicky v souboru i fyzicky na disku)

## 6.4 EXT4

Pouziva pro popis rozlozeni dat na disku strom extentu. Pro "male soubory"(mysleno soubory, na ktere je mozne se odkazovat az 4 extenty, pak tyto extenty budou odkazovane primo z i-uzlu; tldr soubory s malym poctem extentu)

### definice:

*strom extentu* je v principu B+ strom degradovany na maximalne 5 urovni, bez pouzivani vyvazovani (napr. prerozdelovani uzlu pri mazani) a zretezeni listu

## 6.5 NTFS

Zakladni datovou strukturou popisujici disk je MFT - Master File Table (ma pro kazdy soubor alespon 1 radek, na 0. radku popisuje samo sebe, 1. radek pripadne kopie MFT, pripadne metadata, pote obsahy souboru).

### Obsah souboru muze byt reprezentovan bud:

- pokud jde o kratky soubor, bude ulozen v MFT v jeho radku (vcetne metadat),
- soubor je rozdelen na extenty (ty jsou odkazovane primo z radku souboru v MFT, tak ze v radce souboru jsou informace o pocatecnim VCN a LCN a pocet clusteru, ktery dany extent obsahuje) - vyhledava se v tom stejne jako v B+ stromu
- pokud je extentu potreba vice nez se vleze na jeden radek, alokuji se pomocne radky (z hlavniho radku vedou odkazy na pomocne, z pomocnych vedou odkazy na disk) - prochazeni je opet ve stylu B+ stromu

### definice:

*VCN* - virtual cluster number, logicky blok souboru

*LCN* - logical cluster number, cislo fyzickeho bloku (souvisi s tim ze je to na logickem disku)

## 6.6 Organizace volneho prostoru na disku

V klasickem Unixovem FS a rade jeho nasledovniku (UFS, EXT2, 3), take v NTFS se pouzivaji bitove mapy, kde pro kazdy blok mam 1 bit. V bitove je mozne pote vyhledavat pomoci bitovych mask - zrychli vyhledavani.

### Dalsi mozne zpusoby organizace volneho prostoru:

- pouziji se alokacni seznamy (zretezeni volnych bloku na disku),
- zretezeni volnych polozek v tabulce (FAT),
- B+ strom (udrzovani informaci o tom, kde je volne misto, adresace velikosti a/nebo offsetem)
- volny prostor muze byt take organizovan po extentech.

## 6.7 Deduplikace

- podporována ZFS, NTFS, Btrfs, XFS, (ext4 ne) ..
- snaží se odhalit opakované ukládání stejných dat na disk a uloží je pouze jednou a odkazuje se na ně vícenásobně.
- systémy s deduplikací se snaží taková data detekovat (sekvence bitů, bloku, extentu, ..)
- založeno na kryptografickém hashování (hledají se data se stejným popisem, určí se shoda),
- může být realizováno při zápisu nebo dodatečně (žádost uživatele),
- může ušetřit diskový prostor (při virtualizaci, na mail serverech, repozitáři, ..), paměťový prostor i čas (zamezí se opakovanému čtení i zápisu),
- při menším objemu duplikace může naopak zvýšit spotřebu CPU času, spotřebu paměťového i diskového prostoru

### Rozdíl oproti copy-on-write:

- copy-on-write se může uchytit (klony, snímky) pouze tehdy, pokud duplikáty vzniknou činností samotného filesystemu (např. vytvoření virtuálu),
- zatímco deduplikace aktivně vyhledává duplikáty (např. uživatel, co stahuje stejné reklamní letáky)

## 6.8 Typy souboru v UNIXu

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

### definice:

*speciální soubor* je typ souboru reprezentující hardwarové zařízení (disk, hw, paměť) se kterým se komunikuje po blocích nebo znacích

*symbolický odkaz* je soubor obsahující jméno jiného souboru (odkazuje na něj)

## 6.9 Adresar

Je to kolekce jinych souboru na nejvyssi urovni abstrakce. Soubor obsahuje mnozinu dvojic jmeno a unikatni ciselne oznaceni.

### Jmeno souboru:

- drive limit 14 znaku, dnes az 255 (na konci musi byt '`\0`')
- ve jmene nesmi byt / nebo '`\0`'
- lati ze kazdy adresar v POSIX systemu vzdy obsahuje minimalne 2 jmena: . (odkaz na sebe) a .. (odkaz na rodicovsky adresar)

### Cislo souboru:

- u klasickeho souboru unixu je to cislo i-uzlu,
- v jinych pripadech to slouzi jako klic do dane vyhledavaci struktury (B+ strom)

### Implementace adresaru:

- pouzivaji se ruzne pristupy, lisi se jednoduchosti implementace ci rychlosti vyhledavani (vkladani),
- seznam (obsah souboru bude tvorena seznamem),
- B+ stromy (v NTFS, XFS, JFS, APFS nebo EXT3/4 - ty pouzivaji H-stromy: 1-2 urovne, bez vyvazovani a vyhledava se na zaklade zahashovaneho jmena)
- hashovaci tabulky v napr. ZFS

Soubor v UNIXu muze mit vice jmen. Dalsi jmena se vytvari pomoci prikazu `ln`

### linux:

`ln [existujici jmeno] [nove jmeno]` vytvori dalsi jmeno souboru



## 6.10 Montovani disku

### Princip motnovani disku:

- v UNIXu neni zadne oznaceni disku (A:, C:, ..), ale mame jeden adresarovy "strom",
- v systemu je jeden korenovy logicky disk,
- dalsi logicke disky se pripojuji programem *mount* do existujiciho adresaroveho stromu (korenovy adresar zarizeni se "slepi"s adresarem v mem stromu)

Pripojovací volby se mohou zadavat rucne (v terminalu) nebo se mohou predpripravit do */etc/fstab*. Soubor */etc/mstab* obsahuje tabulku aktualne pripojenych disku.

### Novejsi technologie umoznuji automaticke montovani nove pripojenych zarizeni:

- na linuxu bezne pracuje system *udev*, který
- rozpozna, ze se pripojilo nove zarizeni,
- vytvori odpovidajici soubor typu blokove zarizeni (*/dev/..*),
- informuje o tom zbytek systemu pomoci sbernice D-Bus,
- aplikace typu spravce souboru pak muze provest automaticke montovani (a dalsi akce),
- prednost ma vzdy */etc/fstab*,
- identifikace se nemusi provadet jen zarizenim (*/dev/..*), je mozne si vygenerovat unikatni identifikator a pouzivat ten (UUID)

### Technologie Automounter:

- subsystem jadra,
- pripojuje automaticky potrebne disky v situaci, kdy se pokusime pristoupit na pozici adresaroveho stromu, kam by takovyto disk mel byt pripojeny (napr. na */mnt* ma byt pripojena flashka, nemusi byt *mountla*, uzivatel da *cd /mnt*, *automounter* to zjisti a pripoji flashku sem),
- ma take nejaky cas, po kterem disk automaticky odpoji, pokud se nim nepracuje

### Union mount:

- technologie umoznujici sjednocujici montovani (v unixu dostupna pomoci filesystemu *UnionFS*),
- umoznuje do jednoho pripojneho bodu namontovat vice disku,
- obsah pripojneho bodu je sjednocenim obsahu disku,
- v pripade, ze na vice discich jsou soubory se stejnými jmeny vznikaji kolize, ty se resi napr. preddefinovanim priorit pripojovanych filesystemu a zprístupni se soubor daneho jmena z logickeho disku, který ma nejvetsi prioritu
- *UnionFS* ma copy-on-write semantiku, což umoznuje emulaci prepisovani neprepisovatelných medií (v 1 vetvi CD, neprepisovatelné, na tom je linuxova distribuce, soucasne se do stejneho bodu pripoji bezny disk s vyssi prioritou - na zacatku bude disk prazdny, budou videt vsechny soubory z CD, jakmile se

pokusim prepsat neco, UnionFS vytvori kopii na prepisovatelny disk)

**linux:**

*mount [co-pripojit] [kam-pripojit] pripoji logicky disk*

**Sedma prednaska:** Pokracovani Spravy souboru. Symbolicke odkazy,

### 7.1 Symbolicke odkazy

- je samostatny soubor odkazujici na existujici soubor,
- system pri otevreni automaticky otevre cilovy souboru - vicenasobne zpracovani cesty (ceska k symlinku a cesta uvnitr nej),
- soubor se smaze, pokud jeho pocet jmen klesne na 0,
- symlink muze odkazovat na neexistujici soubor (pri otevreni dojde k chybe),
- muze odkazovat i na jiny logicky disk,
- lze ze symlinku vytvorit cyklus (jeden odkazuje na druhy a druhy na prvni) - v systemu je preddefinovany maximalni pocet na sebe odkazujicich symlinku (pri prekročení dojde k chybe),
- symlinky lze využít napr. při upgradu systému.

#### Rozdíl rychlých/pomalých symlinku:

- obsahem symlinku je jméno cílového souboru,
- pokud jméno souboru není příliš dlouhé (vleze se do i-uzlu), potom se uloží do i-uzlu = rychlý symlink (staci otevrit jen i-uzel),
- pokud se jméno nezleze do (i-uzlu), alokuji se normalne alokacni bloky na disku = pomaly symlink

#### linux:

*ln -s [existujici soubor] [symbolicky odkaz]* vytvori symbolicky odkaz

## 7.2 Blokové a znakové speciální soubory

Soubory reprezentující rozhraní souborového systému k fyzickým (opravdový hw) či virtuálním zařízením (xterminals, ..). Souborový systém vytváří souborové rozhraní, tím umožňuje tyto soubory při určitých operacích identifikovat (jméno souboru, např. /dev/sdX), s celým zařízením lze také pracovat jako se souborem.

Typický zařízení sídlí v adresáři /dev

### Beze typy zařízení:

- /dev/hda - (drive)označení pro první fyzický disk na prvním ATA/PATA rozhraní
- /dev/hda1 - (drive) první logický disk na hda
- /dev/sda - první fyzický disk SCSI, navíc i disky SATA/PATA (jádro nad těmito disky emuluje SCSI)
- /dev/mem - obsah paměti (RAM)
- /dev/zero - nekonečný zdroj 0 bajtů
- /dev/null - soubor typu černá díra - cokoli se do něj zapisuje, do se zahodí (přesměrování výstupu programu tak, aby nás neotravovalo), při čtení se chová jako prázdný soubor
- /dev/random - generátor náhodných čísel
- /dev/tty - terminál
- /dev/lp0 - tiskárna
- /dev/mouse - myš
- /dev/dsp - zvuková karta
- /dev/loop - zařízení typu (smýčka) loop mi umožňuje připojit soubor jako disk (obraz souborového systému) k adresáři, jakoby se jednalo o nový fyzický disk

Tato označení závisí na použitém systému (Linux, distribuce, ..). Výhoda zavedení speciálních souborů je, že umožňují identifikovat zařízení, se kterými chceme pracovat.

### 7.3 Pristupova prava

V UNIXu jsou typicky rozlisena na prava pro vlastnika, skupinu vlastniku a ostatni. Existuje rozsireni ACL (access control list).

#### **Uzivatele:**

- jsou definovani administratorem systemu (root) v /etc/passwd,
- maji definovana sva UID - uzivatelska cisla (root UID = 0),
- kazdy soubor ma sveho vlastnika,
- chown - zmena vlastnika souboru (pouze root),

#### **Skupiny:**

- definuje administrator systemu v /etc/group,
- maji sva GID - cislo identifikujici skupinu uzivatele,
- v kazde skupine je uvedeno, kdo do te skupiny patri,
- kazdy uzivatel muze byt clenem vice skupin,
- jedna z nich je aktualni (pouziva se pri vytvoreni souboru)

#### **linux:**

*groups* - vypis skupin uzivatele

*chgrp* - zmena skupiny souboru

*newgrp* - novy shell s jinym aktualnim GID

## 7.4 Typy pristupovych prav

**Obycejne soubory:** r, w, x - pravo cist, zapisovat a spustit soubor jako program.

**Adresare:**

- r - pravo cist obsah adresare,
- w - pravo zapisovat (vytvaret a rusit soubory),
- x - pravo pristupovat k souborum v adresari (moznost cd adresar, ls -l adresar, ..)

**Typicky vystup pristupovych prav je:**

- ve formatu: [1:typ souboru] [3:prava vlastnika] [3:prava skupiny] [3:prava ostatnich]
- napr.: -rwx—r- (ciselne vyjadreni v 8 soustave 0704)
- je obycejny soubor, vlastnik ma vsechna prava, skupina zadna a ostatni maji prava na cteni

Zmena pristupovych prava se deje pomoci chmod. (Pro nespustitelne soubory je bezny chmod 0644).

**linux:** `chmod [1:pro koho][nova prava] [soubor/y]` zmena pristupovych prav

## 7.5 Sticky bit

Priznak, který pokud bude prirazen nejakemu adresari, tak se vytvori adresar, ve kterem i pres pravo cteni a zapisu souboru mohou uzivatele rusit pouze ty soubory, které sami vytvorili. (typicky adresar /tmp) (tldr: uzivatel muze mazat, ale pouze to co vlastni)

## 7.6 SUID, SGID

**S procesy jsou spojeny identifikatory jako jsou:**

- UID - realna identifikace uzivatele (cislo uzivatele, který dany proces spustil)
- EUID - efektivni identifikator pouzivany pro kontrolu pristupovych prav (vetsinou stejne jako UID)
- GID - realna identifikace skupiny (kdo spustil proces)
- EGID - efektivni GID, (stejne chovani jako u EUID)

Vlastnik programu muze propujcit sva prava komukoliv, kdo spusti program s nastavenym SUID. (tldr: propujci se prava mezi uzivateli, bezne se to pouziva v pripadech, kdy administrator propujcuje sva prava uzivatelum, napr. passwd)

Pri pouziti SUID bude UID = uzivatele, který proces spustil a EUID = identifikace vlastnika (= který prava pujcil). Pokud budou prava propujcena (pouzito SUID), misto x se vypise s, pokud tam x neni, vypise se S.

## 7.7 Typická struktura adresaru v UNIXu

### FHS - Filesystem Hierarchy Standard (Linux), (cast hierarchie):

- /bin - programy pro vsechny uzivatele (spustitelne, mohou byt zapotrebi pri bootovani - musi byt dostupne lokalne)
- /boot - soubory pro zavadeč systému (obrazy jadra, počáteční fs)
- /dev - specialní soubory - rozhraní zařízení
- /etc - konfigurační soubory pro systém i aplikace
- /home - domovské adresáře uživatelů
- /lib - sdílné knihovny a moduly jadra
- /media - připojny bod pro přenosná zařízení
- /mnt - připojny bod pro dočasné filesystemy
- /proc - informace o procesech a jádru
- /root - domovský adresář superuživatelů
- /run - dočasné informace o běžícím systému (demony)
- /sbin - programy pro superuživatelů (nutné pro bootování, ne vše je spustitelné superuživatелеm)
- /sys - informace o jádru, zařízeních, modulech, ..
- /tmp - dočasné pracovní soubory (obsah se mže při restartu)
- /usr - obsahuje dále adresáře a
  - soubory, které nejsou nutné při zavádění systému, struktura je zde podobná jako u / :
  - bin, sbin, lib,
  - include (hlavičkové soubory),
  - share (soubory je možné sdílet nezávisle na architektuře),
  - local (koreň další hierarchie určena pro lokální nestandardní instalace programu),
  - src (zdrojové texty jadra)
- /var - soubory menící se za běhu systému
  - obsahuje log (záznamy o činnosti systému),
  - spool (pomocné soubory pro tisk),
  - mail (poštovní příhrádky uživatelů)

**Nove tema: Datove struktury a algoritmy pro vstup/vystup**

## **7.8 Pouziti vyrovnacich pameti**

Cilem pouziti cache (vyrovnacich pameti) je minimalizace poctu pomalych operaci s periferiemi (disky). Hierarchie: kolekce, sbirka dilcich vyrovnacich pameti (s velikosti 1 alokacniho bloku, ci nasobku), nazyva se buffer-pool, muze mit pevnou velikost, spise je promenna.



## 7.9 Operace se soubory

### 7.9.1 Cteni

#### Prvni cteni alokacniho bloku:

- zjistí se, zda je blok v pameti,
- pokud ne, naalokuje se nový blok, může se využít již nějaký systémem predalokovaný a nevyužitý,
- načtou se data z disku, přesunou se do vyrovnávací pameti,
- vyrovnávací pamet je v prostoru jádra (bezne procesy zde nemají přístup),
- vykousne se z načtených dat tu část, o kterou má uživatel/proces zájem,
- nakopíruje se to do adresového prostoru uživatelského prostoru

#### Pri dalsim cteni:

- nejprve se opět vyhledá, zda je blok v pameti,
- pokud ano,
- nebude se číst z disku, pouze se z alokacního bloku vykousne část, o kterou má uživatel/proces zájem,
- tato část se uživateli předá

### 7.9.2 Zapis

#### Postup pri zapisu:

- nejprve se zjistí, zda je blok v pameti,
- pokud ne, přideli se vyrovnávací pamet,
- načtou se data z disku do vyrovnávací pameti,
- jádro převezme od procesu, který chce zapisovat data, která chce zapsat,
- prepíše jimi danou část alokacního bloku, dirty bit se změní (0 na 1),
- operace končí (neprovede se zápis na disk),
- časem se provede zpožděný zápis na disk a vynuluje se dirty bit

System sám od sebe s periodou prepisuje cache na disk, lze si to vynutit pomocí sync či fsync.

Pokud je známo, že se prepíše celý alokacní blok (nebo se jedná o nový blok), buffer se vynuluje a nenacitají se data z disku do cache.

#### definice:

*dirty bit* je indikátor toho, jestli jsou data cache sladěna s obsahem na disku (0 - data v cache = disk, 1 - data cache != data disk - nuluje se zpožděným zápisem)

### 7.9.3 Otevreni souboru pro cteni

Pokud soubor jeste **nebyl otevren**:

- system musi vyhodnotit cestu a naleznou cislo i-uzlu (resp. cislo datove struktury poskytujici informace o danem souboru - pristupova prava, kde jsou ulozena data),
  - pri tom se postupne nacistaji i-uzly vseh adresaru vedouci na soubor,
  - pote se nacte i-uzel souboru,
  - system pouziva d-entry cache (specialni vyrovnavaci pamet pouzita pro preklad odpovidajicich jmen souboru na i-uzel)
  - dale alokuje polozku v tabulce V-uzlu,
  - z disku se nacte i-uzel,
  - vlozi se do nove alokovane polozky = vznikla rozsirena pametova kopie i-uzlu,
  - budou tam i informace navic (jako je pocet odkazu na danou polozku - danym i-uzlem muze pracovat vice procesu).
- v tabulce popisovacu vytvorime novou polozku,
  - tato tabulka je ulozena v zaznamu o procesu (tabulka procesu v jadre) nebo v uzivatelske oblasti,
  - pouzije se nejnizsi volna polozka zde,
  - naplni se odkazem na polozku v tabulce otevrenych souboru,
- pokud se otevreni vydari, vrati se cislo popisovace, pokud ne, tak vraci -1.

Tolik tabulek se pouziva pro zamezeni duplikaci udaju. Behem otevirani se provadi kontrola pristupovych prav. Soubor je mozne otevrit v rezimu pro cteni, zapis, cteni i zapis.

Dalsi otevreni souboru (**jiz jednou otevreny**):

- opet se vyhodnoti cesta k souboru a ziska se cislo i-uzlu,
- system se podiva do tabulku V-uzlu,
- zjistí, ze i-uzel uz tam je,
- nebude se znovu i-uzel nacistat z disku, pouze se zvysi citac pouziti i-uzlu,
- tabulka V-uzlu musi byt vyhledavaci (typicky vyhledaci struktury jako hash tabulka, strom, ..),
- naalokuje se nova polozka v tabulce otevreni (naplni se rezimem otevreni, pozici, odkazem na sdileny V-uzel),
- naalokuje se nove poloza ve file descriptoru ukazujici na nove otevreni (a ta se vrati)

### **Je mozne pridavat i dalsi identifikatory, napr.:**

- priznak, ze ma byt soubor vytvoren pokud neexistuje,
- pokud existuje, ma byt zkracen na 0,
- otevrit v rezimu pridavani (kdekoli je aktualne ukazovatko v souboru, tak v pripade zapisu se automaticky posune na konec a tam se prida),
- synchronni zapis (operace zapisu skonci az tehdy, kdyz se data zapisou opravdu na disk)

### **Pri chybe:**

- open vraci -1,
- nastavi se chybovy kod, který blize popisuje co se stalo (do knihovni promenne errno),
- existuji standardni chybove kody,
- lze pouzit standardni knihovni funkci perror

### **linux:**

*fd = open([jmeno souboru], [rezim]);* otevře soubor

*V-uzly* je tabulka i-uzlu filesystemu VFS

*tabulka popisovacu* je pole s radky cislovanymi od 0 (0 - stdin, 1 - stdout, 2 - stderr)

*tabulka procesu* v jadře je cast adresoveho prostoru, ve kterem ma jadro ulozene pomocne informace k procesu a ma sem pristup pouze jadro

## **7.9.4 Cteni a zapis z/do souboru**

### **Cteni:**

- zkontroluje se platnost popisovace (otevreni popisovace, soubor pro cteni),
- pokud se jedna o prvni pristup, naalokuje se cache, nactou se data do cache a z cache se prislusna data pouziji,
- pokud uz jsou data v cache, nactou se odtud,
- predani se deje pozadovanych z cache (RAM, jadro) do pole (RAM, cache adresoroveho prostoru procesu),
- funkce vraci pocet opravdu prectenych bajtu nebo -1 pri chybe (+ nastavi errno).

### **Zapis:**

- funguje podobne jako read,
- pred vlastnim zapisem kontroluje dostupnost diskoveho prostoru a tento prostor alokuje (rezervuje),
- vraci pocet opravdu zapsanych bajtu nebo -1

**linux:**

*read([popisovac], [adresa pameti, kam se ma zapsat], [kolik bajtu se ma nacist])* precte soubor  
*write([popisovac], [adresa pameti, ze ktere se nactou data], [kolik bajtu se zapise])* zapis do souboru

### 7.9.5 Primy pristup k souboru

Nahodne presouvani v souboru. **Postup:**

- zkontroluje zda je popisovac platny (je soubor otevren?)
- nastavi pozici offset bajtu od whence
  - SEEK\_SET - napr. 200 - posunu se od 200 bajtu od zacatku,
  - SEEK\_CUR - od aktualni pozice,
  - SEEK\_END - od konce souboru),
- nelze se posunout pred zacatek souboru,
- je ale mozne se posunout za konec souboru (a zapsat),
- vraci se vysledna pozice od zacatku souboru nebo -1

Posunem za konec souboru a naslednym zapisem vznikaji tzv. ridke soubory (sparse files):

- umoznuje na disku o nejake kapacite vytvorit soubor, který ma zdanlive vetsi velikost nez samotny disk,
- bloky do kterych se nezapisovalo nejsou alokovany a nezabiraji diskovy prostor (pri cteni se povazuji za 0),
- take muze vzniknout mazanim uprostred souboru (hole punching)



Obrázek 3: z prezentace IOS: Sprava souboru - ridke soubory

**linux:**

*lseek([popisovac souboru], [offset], [oproti cemu se chci posouvat])* primy pristup k souboru

### 7.9.6 Zavreni souboru

- zkontroluje se platnost file descriptoru (je vubec otevreny?),
- uvolni se dana polozka v tabulce popisovacu,
- system se podiva na odkazovanou polozku v tabulce otevrenych souboru,
- snizi se pocitadlo o 1,
- pokud bude pocitalo != 0, uzavirani skonci,
- pokud bude pocitadlo == 0, pokracuje se do
- prislusne polozky tabulky V-uzlu, snizi se zde pocitadlo o 1,
- pokud bude zde pocitadlo != 0, uzavreni skonci,
- pokud bude zde == 0, soubor se definitivne uzarve,
- uvolni se z pameti i-uzel z tabulky V-uzlu (se zmenenymi udaji - cas zapisu, pristupu, modifikace i-uzlu, ..),
- naplanuje se blok, ve kterem je i-uzel ulozen,
- casem se i-uzel zapise na disk,
- funkce vraci 0 nebo -1 pri chybe

Pokud se proces skonci, automaticky se zavrou vsechny jeho deskriptory. Uzavreni souboru nezpůsobí uložení obsahu jeho vyrovnávací paměti na disk.

#### linux:

*close([popisovac souboru])* zavře soubor

### 7.9.7 Duplikace deskriptoru souboru

- zkontroluje se platnost deskriptoru (je soubor otevren?),
- zkopiruje obsah puvodniho popisovace do noveho (odkaz ve fd tabulce se zkopiruje do dalsi polozky v teto tabulce - oboje ukazuji na stejnou polozku tabulky otevrenych souboru + inkrementuje se pocitadlo),
- automaticky se novy deskriptor uzavre (pokud je otevren),
- vraci index nove vytvorene polozky nebo -1,
- typicke pouziti je u presmerovani (stdin/stdout)

#### linux:

*dup([popisovac])* duplikace deskriptoru (duplikuje existujici popisovac do nejvyssiho volneho noveho)

*dup2([popisovac], [novy popisovac])* duplikace deskriptoru (do ktereho popisovace se duplikuje)

### 7.9.8 Ruseni souboru

- vyhodnoti se cesta, zkontroluje se platnost jmena souboru, pristupova prava (zapis),
- odstrani se pevny odkaz (=hard link) mezi jmenem souboru a i-uzlem,
- zmensi se pocet jmen v i-uzlu,
- pokud je pocet jmen == 0 a i-uzel nikdo nepouziva - i-uzel muze byt uvolnen a mohou byt uvolneny vsechny bloky souboru,
- dokud ma soubor alespon 1 jmeno nebo nema zadne jmeno ale je otevren alespon 1x, nelze soubor z disku opravdu smazat,
- funkce vraci 0 nebo -1 pri chybe

Je mozne provest unlink na otevreny soubor (smaze se az po jeho uzavreni) a pracovat s nim dale, vyuziti pri instalacich novych verzich programu, ktere aktualne bezi. (upgradovat upgradovaci program)

#### **linux:**

*unlink([jmeno soubor, prip. cesta])* rusi soubor  
*shred* - bezpecne mazani

### 7.9.9 Dalsi operace se soubory

#### **linux:**

*creat, open* - vytvoreni souboru  
*rename* - prejmenovani souboru  
*truncate, ftruncate* - zkraceni souboru  
*fcntl, lock* - zamykani zaznamu  
*chmod, chown* - zmena atributu  
*utime* - umoznuje zmenit casy prace se soubory (neumoznuje zmenit cas modifikace i-uzlu)  
*stat* - ziskani atributu (velikost, prava, ..)  
*sync, fsync* - vynuceni si zapisu vyrovnacich pameti

### 7.9.10 Adresarove soubory

Obsahuje dvojice cislo i-uzlu a jmena souboru. Adresare nelze zapisovat ci cist po bajtech.

#### **linux:**

*mkdir* - tvori se adresare (vytvori polozky . a ..)  
*opendir* - otevre adresar  
*readdir* - cte adresar  
*closedir* - zavre adresar  
*creat, link, unlink* - modifikace se provadi nepriamo vytvarenim/modifikacemi souboru

### 7.9.11 Blokové a znakové speciální soubory

Představují rozhraní k blokovým / znakovým zařízením (/dev/..., viz. 7.2)

- lze je vytvořit pomocí `mknod`,
- typicky tyto soubory vytváří jádro či démoni (`udev`, `devd` - při připojení zařízení se vytvoří automaticky příslušný soubor)

Při použití běžných souborových operací jádro mapuje operace na odpovídající podprogramy, které tyto operace implementují pro daný typ zařízení s využitím **tabulek**:

- znakových zařízení,
- blokových zařízení

Tyto tabulky obsahují ukazatele na funkce implementující příslušné operace v ovladačích daných zařízení.

Speciální soubory na disku zabírají **pouze i-uzel, kromě běžných údajů mají v i-uzlu typ souboru a 2 údaje**:

- hlavní číslo,
  - major number,
  - udává typ zařízení
  - odkazuje do tabulky zařízení (hlavní číslo = n-tý řádek tabulky),
- vedlejší číslo
  - minor number,
  - udává instanci zařízení
  - používá se jako parametr při volání určité operace - parametr funkce ovladače (číslo = které zařízení se má přesně použít)
- typ souboru určuje tabulku (blok, znak.)

#### **linux:**

*mknod* vytvoří speciální soubory

*ovladac* je sada podprogramů pro řízení určitého typu zařízení (nebo viz `xx` nebo viz `xx`)

## 7.10 Terminaly

Jsou fyzická či logická zařízení umožňující (primárně) textový vstup a výstup systému (po řádcích), editace vstupního řádku či speciální znaky (Ctrl+C SIGINT, Ctrl-D konec vstupu, ..)

### Rozhraní:

- /dev/tty - pro každý proces, který má řídicí terminal, odkazuje na jeho řídicí terminal
- /dev/ttyS1 - fyzické terminály na sériové lince,
- /dev/tty1 - virtuální terminály (konzole),
- pseudoterminály (/dev/ptmx - master, /dev/pts/1,..) tvořeny dvojicí master / slave, po každém otevření se vytvoří nový slave - emuluje komunikaci přes sériovou linku (umožňuje propojení určitých částí, např. SSH - propojení klienta se vzdáleným klientem)

### Různé režimy zpracování znaku (radkové disciplíny - line discipline):

- raw - neprovádí se zpracování znaku,
- cooked - zpracování všech řídicích znaků,
- cbreak - provádí zpracování maleho počtu znaku (ctrl+c, mazání, ..)

Nastavení režimu zpracování znaku je možné pomocí stty. Dale je možné nastavit **režim terminálu**:

- příkazy tset, tput, reset,..
- proměnnou TERM, ve které uložen aktuální typ terminálu,
- typy terminálu (příkazy terminfo, termcap)

Tyto příkazy komunikují s terminálem pomocí *escape sekvencí*. Knihovna curses je standardní knihovna pro řízení terminálu či tvorbu aplikací s terminálovým uživatelským rozhraním.

### definice:

*escape sekvence* jsou sekvence znaku escape, příkaz [parametry], escape, příkaz, ..



## 7.11 Roury

Jsou prostředkem meziprocsove komunikace. Rozlisujeme:

### Nepojmenovane roury

- nemaji adresarovou polozku, tedy neexistuji v souborovem systemu,
- lze s nimi pracovat pouze tak, ze se vytvori pomoci volani pipe (vrati cteci a zapisovy deskriptor), jakmile dojde k uzavreni - prace s rourou konci,
- mohou s ni pracovat bezne pouze pribuzne procesy,
- je dostupna pomoci popisovacu z tabulky popisovaci (pri klonu procesu se naklonuje tabulka popisovaci - proces bude ukazovat na stejne misto v tabulce otevrenych souboru),
- jedina vyjimka, jak je mozne odkaz na nepojmenovanou rouru predat je pres UNIXove sockety (krome klonovani procesu),
- vytvari se v kolonach (napr. paralelne bezici procesy  $p1 \rightarrow p2 \rightarrow p3$  - na presmerovani se pouzivaji nepojmenovane roury)

### Pojmenovane roury

- vytvari se pomoci `mknod` i s `mknfio`,
- existuji v souborovem systemu,
- mohou se zavrit, otevrit, apod.

Roury slouzi jako mechanismus meziprocsove komunikace. Implementovane jako kruhovy buffer s omezenou kapacitou. Procesy komunikujici pres rouru jsou synchronizovany.

#### definice:

*pribuzne procesy* - pokud jeden proces otevri rouru a zacne se klonovat, vsechny tyto procesy mohou s rourou pracovat

*konzumenti* - procesy, které ctou

*producenti* - procesy, které zapisuji

## 7.12 Sockety

TUDU