

VYSOKÉ UCENÍ TECHNICKÉ V BRNĚ  
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

IOS - Operační systémy  
poznámky z přednášek

# Obsah

<b>1</b>		<b>3</b>
1.1	Uvod, prehled operacnich systemu . . . . .	3
1.2	Zakladni pojmy . . . . .	4
1.3	Jadro operacniho systemu . . . . .	5
1.4	Typy jader OS . . . . .	6
1.5	Historie vyvoje OS . . . . .	8
1.6	Prehled technickeho vybaveni . . . . .	8
1.7	Klasifikace pocitacu . . . . .	9
1.8	Klasifikace OS . . . . .	10
1.9	Implementace OS . . . . .	10
1.10	Hlavni smery ve vyvoji OS . . . . .	11
<b>2</b>		<b>12</b>
2.1	Priciny uspechu UNIXu . . . . .	12
2.2	Varianty UNIXu . . . . .	13
2.3	Zakladni koncepty . . . . .	13
2.4	Struktura jadra UNIXu . . . . .	14
2.5	Komunikace s jadrem a hardwarova preruseni . . . . .	15
2.5.1	Hardwarove preruseni . . . . .	15
2.5.2	Zakazovani preruseni . . . . .	16
2.5.3	Pristupy k zakazovani preruseni . . . . .	17
2.5.4	Ovladace zarizeni a preruseni . . . . .	17
2.5.5	Priklad komunikace s jadrem . . . . .	18
2.6	Nastroje programatora UNIXu . . . . .	19
<b>3</b>		<b>19</b>
3.1	Bash, shell, experimenty . . . . .	19
<b>4</b>		<b>19</b>
4.1	Bash, shell, experimenty . . . . .	19
<b>5</b>		<b>20</b>
5.1	Pevny disk . . . . .	20
5.2	Parametry pevnych disku . . . . .	21
5.3	Solid State Drive - SSD . . . . .	22
5.3.1	Klady a zapory SSD . . . . .	22
5.3.2	Problematika zapisu u SSD . . . . .	22
5.4	Zabezpeceni disku . . . . .	23
5.5	Diskova pole (RAID) . . . . .	24
5.5.1	RAID 0 . . . . .	24
5.5.2	RAID 1 . . . . .	24
5.5.3	RAID 2 . . . . .	24
5.5.4	RAID 3 . . . . .	25
5.5.5	RAID 4 . . . . .	25
5.5.6	RAID 5 . . . . .	25
5.5.7	RAID 6 . . . . .	25
5.6	Opravy chyb u paritnich disku . . . . .	26
5.7	Ulozeni dat na disku . . . . .	26
5.8	Fragmentace . . . . .	27

5.8.1	Externi fragmentace . . . . .	27
5.8.2	Interni fragmentace . . . . .	28
5.9	Přístup na disk . . . . .	29
5.10	Planování přístupu na disk . . . . .	29
5.11	Logický disk . . . . .	30
5.11.1	Způsob uložení informací o diskových oblastech na disku . . . . .	30
5.11.2	LVM . . . . .	30
5.11.3	Různé typy souborových systémů . . . . .	30
5.11.4	Chyby disku (souvislost s FS) . . . . .	31
5.11.5	Další typy souborových systémů . . . . .	31
<b>6</b>		<b>32</b>
6.1	Žurnalování . . . . .	32

# 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 zdroje počítače - drahé počítače, levnější pracovní síla (drive)
- jednoduchost použití počítače - 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).

Presná 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 (spusti proces - spusti proces) - 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 kontrolovanemu 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 používat vyšší vrstvy OS
- všechny subsystemy jsou implementovány v privilegovaném režimu, režimu jádra, a zahrnují např. správu paměti, plánování, meziprocsovou komunikaci, souborové systémy, ..
- výhody: vysoká efektivita díky provazanosti
- nevýhody: malá flexibilita při práci s jádrem (ve filesystému je chyba, chci změnit jen implementaci filesystému za novou verzi a vše ostatní nechat - nelze, je nutné celý systém zastavit a znovu nastarovat, nelze menit nic za běhu)

### Monolitická jadra s modulární strukturou

- vylepšení koncepce monolitických jader
- umožňuje zavádět/odstraňovat subsystemy jádra v podobě tzv. modulu za běhu
- výhody: není nutné celý systém zastavovat a znovu bootovat pro výmenu jednoho modulu, vyšší bezpečnost - zavedou se jen moduly, které se budou používat
- používane v např. FreeBSD, Linux

### Mikrojadra

- snaha minimalizovat rozsah jádra a rozsah jeho služeb
- nabízí jednoduše rozhraní, malý počet abstrakcí, služeb, typicky nabízí nejzákladnější správu CPU, I/O zařízení, paměti, ..
- většina služeb nabízených monolitickými jádry (ovládání, významné části správy paměti, plánování) je implementována mimo jádro v tzv. serverech (nebo v privilegovaném režimu).
- výhody: flexibilita (více současně bezcíh implementací různých služeb, dynamické spouštění, zastavování..), zabezpečení (chyba v serveru / útoku na něj neznamena ovládnutí celého OS, ale jen daného serveru)
- nevýhody: výrazně vyšší režie

### Generace mikrojader

- 1. generace - např. Mach
- 2. generace - např. L4, menší režie než 1. gen
- 3. generace - např. seL4 nebo ProvenCore, důraz na zabezpečení, návrh s ohledem na možnost formální 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 zakomunikuje 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 (poradě 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 pamětí (cache, RAM, disky, ... - bank pamětí 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 - nejnížší 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)
- diskové sběrnice (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, spotrební 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 uzivatelů:

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

### Podle poctu soucasne bezicich uloh:

- jednoulahove
- viceulahove (multitasking, ne/preemptivni)

### definice:

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

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

## 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 nemít 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 přepínají 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 tabletu, mobilu, vestavených 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 jadem - 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 a 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 probíhající nějaké aktivity (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 horním okraji jadra (směrem k uživateli, aplikacím) je vrstva implementující rozhraní volání služeb, prostřednictvím které jádro přebírá žádosti o služby od aplikací. Rozhraní kontroluje, zda ten, kdo o službu žádá, jí může volat, zda jsou parametry validní a rozhraní předává požadavek dál do jadra.
- Aplikace mohou s jádrem komunikovat přímo, nicméně nejčastěji komunikují s jádrem přes knihovny. (viz. 1.3)
- Na druhém okraji (těsně nad HW) je vrstva abstrakce hardware.
- Mezi správou souboru a hardware se nachází ovladač, poté vrstva vyrovnávací paměti, které souborové systémy používají ke zrychlení práce s relativně pomalými disky (HDD, SSD - oproti RAM pomale) - OS se snaží vyhnout opakovanému čtení stejných dat, proto si v jednom okamžiku načte více dat než uživatel žádá, uloží si data do vyrovnávací paměti (při dostatku paměti) a data načítá odtud. (např. C knihovny jsou používány každým druhým programem - jsou v paměti téměř pořád).

### definice:

*ovladace* jsou programy sloužící k řízení (zadávaní příkazů, přebírání stavových informací, řešení mimořádných stavů konkrétních periférií) - lze je (jako i příslušná zařízení) rozdělit na znaková a bloková (kratší definice viz 5.9)

*znaková zařízení* jsou zařízení komunikující po jednotlivých znacích (klávesnice)

*bloková zařízení* komunikují po blocích (disk - sektory, resp. bloky)

*komunikaci s jádrem* rozumíme nastavování parametrů hardware, vydávání příkazů HW, obsluhu různých stavů do kterých se HW dostává (a o kterých je CPU a jádro informováno prostřednictvím přerušení)

*nastavování parametrů HW* se děje pomocí I/O portu nebo pamětové mapovaných operací (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 - otevri/zavre/cte 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 asynchronne vznik udalosti, ktere je zapotrebi obslousit (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 poznamenava 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 běží 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 běží 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 uvažnutí 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 ctvrtá 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 disk, fragmentace, přístup na disk a jeho plánování, logický disk.

### 5.1 Pevný disk

#### Popis:

- uvnitř mají řadu kulatých ploten, zaznam se provádí na každém z těchto dvou povrchů, je v soustředných kruznicích (= tracks, stopy)
- všechny plotny jsou na stejné ose, přidelány k sobě a rotují současně
- k načítání slouží sada hlaviček, čtecí a zápisové, 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 sady několika stop (kruznic) o stejném průměru = cylindry,
- 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,
- nejběžnější 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 (více 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 zalohovanim 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 celý blok nacist do pameti, vymazat (zresetovat) a v pameti upraveny blok nacist zpet (= write amplification, zesileni zapisu - mnohonasobne zpomalení),
- 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 predalokovaneho prostoru, ci obrazy virtulanych 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 systemy maji implementaci RAID v sobe.

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

### 5.5.1 RAID 0

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

### 5.5.2 RAID 1

- disk mirroring, pro 2 a více disku,
- 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 (napr. 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,
- díky 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)
  - selze 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 ničím menším než sektor, ale typicky OS si sektory nějak seskupí (do větší jednotky) a neumí pracovat s ničím menším, než je alokační blok.

### Logická a fyzická následnost:

- 1 alokační blok se namapuje 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 rozdelí 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 (tj. vznik volných úseků, které nejsou využity)
  - 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)
  - 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 metadat - 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 (snaha 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 (filesystem nezapíše ihned po změně souboru, ale chvíli počká - počítá s tím, že uživatel bude chtít menit soubor "za chvíli" znovu - až nebude delší doba docházet ke změnám, poté hledá vhodný volný prostor)

Pri (intenzivním) bezném používání disku se však fragmentaci nelze vyhnout. Pokud by byla fragmentace příliš výrazná, je možné použít defragmentační nástroje, které provádějí kopírování, přesouvání částí souboru a reorganizaci diskového prostoru tak, aby se fragmentace odstranila - časově náročná operace.

Prvního negativního dopadu externí fragmentace (nevyužitelné a příliš malé oblasti) je možné se zbavit při používání alokací po jednotkách pevné velikosti - alokační bloky - vždy je ale snaha alokovat spojitě (v horských případech alokují nespojitě, pokud to nejde)

### **5.8.2 Interní fragmentace**

Nespojitá alokace po jednotkách pevné velikosti (alokační bloky) má výhodu, že redukuje dopady externí fragmentace, ale potom vytváří interní fragmentace. Interní fragmentace se obvykle toleruje.

#### **Příklad interní fragmentace:**

- chci alokovat soubor o velikosti 9 000 B,
- mám 4 KiB velké alokační bloky,
- potom je nutné alokovat 12 KiB pro tento soubor,
- ty zbývající 3 KiB v posledním alokačním bloku zůstanou nevyužité.

Existuje několik málo filesystemů, které se snaží řešit interní fragmentaci (ReiserFS, ZFS) pomocí techniky zvané 'tail packing' ('zbavování ocásku' souboru) - více souborů může používat 1 fyzický alokační blok (zaplní se volné místo). Většina filesystemů toto však 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 určitým typem zařízení, jina definice viz. 2.4

## 5.10 Planování přístupu na disk

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

Planovač se snaží minimalizovat režii 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.

**Planovač 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 planovač 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 (nulovém) 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 zřetězeného 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 filesystemu nebo v části jádra (mezi filesystemem a plánovačem).

### 5.11.3 Různé typy souborových systémů

- fs (první fs na Unixu), ufs, ufs2,
- ext2, ext3, ext4,
- btrfs (inspirovan ZFS),
- ReiserFS, HFS+/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, superpocity),
- 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

**Sesta prednaska:** pokračovani Spravy souboru. Zurnalovani,

### 6.1 Zurnalovani

TODO