VYSOKE UCENI TECHNICKE V BRNE
FAKULTA INFORMACNICH TECHNOLOGII

IOS - Operacni systemy

poznamky z prednasek

2019/2020

Corse & US Layout

Obsah

1

Prvni prednaska: Uvod do predmetu, prehled operacnich systemu, zakladni pojmy, jadro operacniho systemu a jejich typy, historie vyvoje operacnich systemu, prehled technickeho vybaveni, klasifikace pocitacu, operacnich systemu, hlavni smery ve vyvoji operacniho systemu.

1.1 Uvod, prehled 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), ktera vytvari spojujici mezivrstvu mezi hardware operacniho systemu a uzivateli a jejich uziv. aplik. programy. OS dale spotrebovava zdroje, jako jsou pamet nebo cas CPU. (tldr: sw, spojujici hardware, uzivatele a programy)

Cile OS:

- maximalni vyuziti zdroju pocitace - drahe pocitace, levnejsi pracovni sila (drive)
- jednoduchost pouziti pocitacu - levne pc, draha pracovni sila (dnes prevazuje)

Zakladni role OS:

- spravce prostredku
  - pamet, procesor, periferie
  - dovoluje sdilet prostredky efektivne a bezpecne
- tvurce prostredi pro uzivatele a jejich aplikacni programy
  - vytvareni abstrakci, virtualnich objektu (resp. poskytuje standardni rozhrani, ktere zjednodusuje prenositelnost aplikaci a zauceni uzivatelu)
  - abstrakce jsou napr.: proces, program, soubor
  - problemy abstrakci jsou mensi efektivita a nepristupne nektere nizkourovnove operace

OS zahrnuje:

- jadro (kernel),
- systemove knihovny a utility (= systemove aplikacni programy),
- textove (shell) ci graficke uzivatelske rozhrani (X Window).

Presna definice, co vse OS zahrnuje neexistuje. Ruzne firmy a komunity to chapou ruzne. (GNU to chape napr. jako projekt svobodneho OS, zahrnujici jadro, utility, GUI, TUI, vyvojove prostredky a knihovny, ...)

definice:

proces je aktivita rizena programem
program je predpis, navod na nejakou cinnost zakodovany vhodnym zpusobem
soubor je kolekce zaznamu (obvykle Byte) slouzici primarne jako zakladni jednotka pro ukladani dat na vnejsich pametovych mediich
adresar je kolekce souboru

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 zapouzdruje.

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 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

Monoliticka jadra

- vysokourovnove komplexni rozhrani s radou sluzeb, abstrakci, ktere mohou pouzivat vyssi vrstvy OS
- vsechny subsystemy jsou implementovany v privilegovanem rezimu, rezimu jadra, a zahrnuji napr. spravu pameti, planovani, meziprocesovou komunikaci, souborove systemy, ..
- vyhody: vysoka efektivita diky provazanosti
- nevyhody: 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)

Monoliticka jadra s modularni strukturou

- vylepseni koncepce monolitickych 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, ktere 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 monolitickymi jadry (ovladace, vyznamne casti spravy pameti, planovani) je implemenovana mimo jadro v tzv. serverech (nebezi v privilegovanem rezimu).
- vyhody: flexibilita (vice soucaasne bezicich implementaci ruznych sluzeb, dynamicke spousteni, zastavovani..), zabezpeceni (chyba v serveru / utok na ne neznamena ovladnuti celeho OS, ale jen daneho serveru)
- nevyhody: 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, ktery 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, ktery jde od periferie po sbernici k procesoru, na CPU vyvola obsluhu preruseni - mechanismus umoznujici rozbehnout operaci na periferii a o tu periferii se nestarat (periferie pote oznami konec operace) (podrobne se tomu venuje oddil 2.5)

multitasking je soucasny beh vice aplikaci na jednom procesoru (muze byt s preemtivnim nebo nepreemtivnim planovanim)

nepreemtivni planovani zn. ze uloha, kt. aktualne bezi na CPU muze byt od CPU "odstavena"pouze tehdy, kdyz nejak zakomunikuje s jadrem (= pozada o sluzbu jadra, napr. periferni operace), dokonce lze pouzit specializovane sluzby pro prepnuti kontextu (proces se dobrovolne vzda CPU, tzv. yield sluzby) - vyhoda: snadna implementace, nevyhoda: pokud se proces zacykli (chyba), cely system se zablokuje (porad bezi 1 uloha)

preemtivni planovani - proces muze byt odstaven od CPU bez nutnosti komunikace s jadrem, napr. pomoci preruseni (jakehokoli typu)

1.6 Prehled technickeho vybaveni

Procesor (CPU):

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

Pamet:

- adresa
- hiearchie pameti (cache, RAM, disky, ... - bank pameti muze byt vice)
  - pameti se lisi spotrebou, kapacitou, rychlosti, cenou za jednotku
  - na vrcholu hiearchie jsou registry (nejrychlejsi, nejvyssi cena za jednotku, mala kapacita)
  - cache (vyrovnavaci pameti, ruznych urovni, L1 = level 1, L2, L3, ..)
  - primarni pamet RAM
  - sekundarni pameti - disky (SSD, HDD)
  - vyrovnavaci pameti disku
  - tercialni pameti (zalohy - nejnizsi cena za jednotku, nejpomalejsi, nejvetsi kapacita - pasky, CD/DVD, externi disky, cloudy, sitove disky, ..)

Periferie:

- disk (HDD, SDD,..), klavesnice, monitor (I/O porty, preruseni, DMA)

Sbernice:

- propojuji jednotlive komponenty
- na vrcholu hiearchie jsou sbernice propojujici CPU a pamet (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 surovy 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 - doporuceni aby se akce vykonavaly v realnem case
hard real-time - akce se musi vykonavat v urcitem case

1.9 Implementace OS

OS se obtizne programuji a ladi, protoze to jsou velke programove systemy, paralelni a asynchronni systemy, systemy zavisle na technickem vybaveni.

Dusledky:

- setrvacnost pri implementaci (snaha nemenit kod, ktery pracuje spolehlive)
- pouzivani technik pro minimalizaci vyskytu chyb (inspekce zdrojoveho kodu, rozsahle testovani, podpora vyvoje technik formalni verifikace)

definice:

paralelni system zn. ze zde bezi vice aktivit soucasne
paralelni asynchronni systemy - procesy se prepinaji v okamzicich, ktere nelze dopredu presne predpovedet

1.10 Hlavni smery ve vyvoji OS

- neustale vylepsovani architektur (snizovani rezii jader,)
- bezpecnost, spolehlivost
- podpora stale vetsiho poctu procesoru, vice jader
- virtualizace
- distribuovane zpracovani (cloudy, kontejnery, Internet of Things)
- OS tabletu, mobilu, vestavenych systemu, ...
- vyvoj novych technik navrhu a implementace OS (podpora formalni verifikace)

definice:

bezpecnost zn., ze system je odolny vuci vnejsim utokum
spolehlivost zn., ze system ”nespadne sam 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 podobe, vstup zpracuji a na vystupu opet otevreny dokument (zadne binarni, zakodovane)

2.2 Varianty UNIXu

Hlavni vetve OS UNIXoveho typu:

- UNIX System V (puvodni system z AT&T),
- BSD UNIX (FreeBSD, NetBSD, ..),
- firemni varianty (AIX, Solaris, ..)
- Linux

Sousisejici 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 striktni podmnozina Single UNIX Specification, je to standard definujici zakladni textove prikazove rozhrani OS + API

2.3 Zakladni koncepty

Jsou dve zakladni koncepce (abstrakce) UNIXu: **procesy** a **soubory**.

Procesy mezi sebou komunikuji pomoci ruznych mechanismu meziprocesove komunikace - IPC (Inter-Process Communication) - roury, signaly, semafory, sdilena pamet, sockets, zpravy, streams, .. a pro komunikaci pouzivaji nejake I/O rozhrani (read, write, close, ..)

definice:

procesy jsou abstrakci probihajici nejake aktivity (viz 1.2)
soubory jsou abstrakci 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 jadrem komunikovat primo, nicmene nejcasteji komunikuji s jadrem pres knihovny. (viz. 1.3)
- Na druhem okraji (tesne nad HW) je vrstva abstrakce hadrware.
- 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 stejnych 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 jadrem 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 jadrem a hardwarova preruseni

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

Priklady nekterych sluzeb jadra (systemova volani v UNIXu):

- open, close, read - otevre/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 jadru asynchronne 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 obsluju 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 intrukci (breakpoint, overflow, ..)
- fault - po obsluze se znovu opakuje intrukce, ktera vyjimku vyvolala (vypadek stranky, deleni 0, ..)
- abort - dochazi k zavaznym problemum detekovanym 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 Interrput (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, ktery 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, ktere mohou byt v nekonzistentnim stavu (napr. zretezene seznamy procesu [ukazatele], ruzne si je projuje, nez je stihne propojit, prije 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,
  - nemusi se zakazovat preruseni,
  - muze bezet v specialnich procesech (interrup 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 prvnim pouziti se musi registrovat k obsluze urciteho 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 Priklad komunikace s jadrem

Synchronni komunikace je proces-jadro, asynchronni je hardware-jadro. Priklad (detailnejsi, ale na tema pristupy na disk viz 5.9):

- proces A zavola sluzbu read() a jadro ihned zacne volani obsluhovat (synchronni)
- nejprve se podiva do cache zda data, o ktera ma zajem proces A uz tam nejsou
- pokud ano, tak mu je rychle nakopiruje z cache na adresu, kterou pozaduje proces (bez komunikace s diskem)
- pokud data nejsou v cache, proces A bude pozastaven a jadro vyda prostrednictvim ovladacu disku prikaz k nacteni urciteho objemu dat, typicky vice nez zada uzivatel a nacita do vyrovnavaci pameti (ne na pozadovanou adresu)
- na procesoru dale bezi proces B, taky pozada o read(), zopakuje se to same co u A
- az disk dokonci operace jednoho z procesu (nemusi byt v poradi volani), disk posle preruseni na CPU
- jadro bude informovano, ze ma potrebna data pro proces A/B
- z cache nakopiruje pozadovana data na pozadovanou adresu
- pote se proces A/B probudi a bezi dal, to same se stane u dalsiho procesu

definice (pro 2.5.x):

asynchronni zn., bez prime-okamzite vazby na to co dela jadro nebo aplikace (tiskarna tiskne - operace nekdy skonci - ale nikdy nevim dopredu kdy presne)

synchronni zn., ze CPU neco provede a ihned se zavola preruseni (napr. deleni 0)

IRQ = interrupt request

radic preruseni = interrput controller, hardwarova jednotka, ktera predava preruseni do CPU - registruje prichozi IRQ, ty se dle priorit predavaji do CPU (preruseni je mozne take zamaskovat - nepredavat dal do CPU) v podobe cisla preruseni, CPU se automaticky prepne do chraneneho rezimu a spusti obsluznou rutinu definovanou jadrem (preruseni 1 - provede xxx, 2 - xxx, ..)

APIC = Advanced Programmable Interrupt Controller - distribuovany system, kazdy CPU ma lokalni APIC, externi zarizeni mohou byt pripojena primo / pres I/O APIC

NMI watchdog - jadro si nadefinuje, ze casovac mu kazdych n casovych jednotek posle toto preruseni - pokud dojde v jadre k uvaznuti pri obsluze jineho preruseni a vsechna preruseni budou zakazana, toto se vzdy dostane do CPU (jadro se muze zotavit)

vypadek stranky zn., (pamet je rozdelena na casti, ktere mohou byt rozdeleny na disk) kdyz proces bude sahat do pameti a sahne na stranku, ktera v ni neni - detekuje se ze stranka tam neni - poruseni ochrany pameti - jadro zkontruluje, zda proces nesaha kam by nemel, a pokud ne, tak mu stranku nahraje zpet do pameti a znovu se provede ta stejna instrukce

bezne synchronizacni prostredky jsou napr semafory nebo zamky a synchronizuji procesy

linux:

zakladni statistiky o obsluze preruseni 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: pevny disk, diskove sbernice, sektory, parametry pevnych disku, SSD, problematika zapisu SSD, zabezpeceni disku, diskova pole (RAID), ulozeni dat na disku, fragmentace, pristup na disk a jeho planovani, logicky disk.

5.1 Pevny disk

Popis:

- uvnitr maji radu kulatych ploten, zaznam se provadi na kazdem z tech dvou povrchu, je v soustrednich kruznicich (= tracks, stopy)
- vsechny plotny jsou na stejne ose, pridelane k sobe a rotuji soucasne
- k nacitani slouzi sada hlavicek, cteci a zapisove, jsou tam v tolika kusech, kolik je tam povrchu (napr. 3 plotny = 6 povrchu = 6 hlavicek), vsechny umistene na jednom rameni, vsechny hlavicky se pohybuji soucasne
- hlavicky jsou nastavene na sade nekolika stop (kruznic) o stejnem prumeru = cylindr,
- stopy se deli na sektory
- velikosti sektoru byly drive 512B, u CD/DVD 2048B, dnes 4096B

Adresace sektoru:

- ze zacatku se pouzival CHS - urci se se kterym cylindrem chci pracovat, dale s kterou hlavou a jakym sektorem v ramci stopy,
- v soucasne dobe se pouziva LBA, kde jsou sektory (bloky) cislovane (jako adresy v pameti) od 0 po n, diskova jednotka si musi tato cisla prevadet na CHS

Periferni ci diskova rozhrani:

- pouzivaji se pro pripojeni disku,
- nejbezneji se pouziva ATA, drive se pouzivala v paralelni verzi (PATA - jednotlive byty se posilaly paralelne, pri rostoucich rychlostech byl problem zajistit synchronizaci techto dat), nyni v seriove verzi (SATA)
- take se pouziva SCSI ci SAS (Serial Attached Scasi), USB, FireWire, FibreChannel, Thunderbold, PCI Express nebo NVMe (pripojovani nejrychlejsich SSD),
- nad temito rozhranimi muze byt dalsi HW rozhrani propojujici tyto sbernice, jako treba 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 tercialni pameti. V systemu vznika hierarchie pameti, viz. 1.6.

definice:

cylindr (v HDD) je mnozina stop o stejnem prumeru
sektor je nejmensi jednotka diskoveho prostoru, ktery mi umozni diskova elektronika nacist nebo zapsat
blok nebo *diskovy blok* je sektor v HDD
alokacni blok nebo *blok souboroveho systemu* je nejmensi jednotka, kterou umozni alokovat OS
CHS zn. Cylinder Head Sector
LBA zn. Linear Block Adress

5.2 Parametry pevnych disku

Pristupova doba sestava z **doby vystaveni hlav** a **rotacniho zpozdeni**.

Typicke parametry soucasnych disku jsou kapacita, prumerna doba pristupu (jednotky ms u HDD) , otacky a prenosova rychlost. U prenosovych rychlosti se rozlisuje *sustained tranfer rate* a *maximum transfer rate*.

Mazani dat probiha tak, ze se prepisou metadata, pouze se poznamena (OS), ze dany soubor byl smazan.

definice:

doba vystaveni hlavicek zn., ze pokud nejsou nastavene hlavicky na stope, se kterou chci pracovat (malokdy), tak je nutne pohnout hlavickami (vic zasunout dovnitr nebo vysunout)
rotacni zpozdeni je doba nez mi pod spravne nastavenou hlavicku najede sektor (narotuje se disk)
maximum transfer rate je spickova prenosova rychlost, jak maximalne rychle je schopen disk komunikovat po kratkou dobu (typicky rychlost predani dat z vyrovnavacich pameti disku)
sustained transfer rate opravdova rychlost cteni z ploten

linux:

hdparm [-t] umoznuje zmerit prenosovou rychlost a menit parametry disku, -T meri rychost prenosu z vyrovnavaci pameti OS (RAM)

5.3 Solid State Drive - SSD

Mohou byt zalozena na ruznych technologiich, nejcasteji na nevolatilnich pametech NAND flash nebo DRAM (se zalohovanym 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 - vyzaduje specialni pdporu).

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 jednotlive (prepisovat ne!),
- pokud chci prepisovat (jednu stranku), je nutne cely blok nacist do pameti, vymazat (zresetovat) a v pameti upraveny blok nacist zpet (= write emplification, 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 nektere 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 databaze, ktere si ukladaji data do velkeho predalokovaneho 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 chybnych sektoru, ..) - S.M.A.R.T (podporovana vsemi ”rozumnymi”disky)

Pak je mozne jeste provadet testovani na urovni OS, napr. e2fsck nebo badblocks nebo si nektere 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 sektoru.

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 souhlasi

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, ktery je rychlejsi a spolehlivejsi.

Muze byt implementovan:

- hardwarove (do rozsirujici karty pripojime nekolik disku a ta implementuje RAID),
- subsystemem v jadre,
- nektere souborove systemy maji implementaci RAID v sobe.

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

5.5.1 RAID 0

- data jsou rozlozena po dvou ci vice discich, ale kazdy datovy blok je ulozen jen na jednom disku (napr. dva disky, 0 a 1, prvni datovy blok [sektor, skupina sektoru] je na 0, druhy na 1, treti na 0, ...)
- vyssi efektivita cteni ci zapisu,
- je mozne paralelne cist ci zapisovat (do vice disku)
- prudce snizuje spolehlivost - pokud selze jeden disk, prijdu o data na nem

5.5.2 RAID 1

- disk mirroring, pro 2 a vice disku,
- vsechny bloky dat se zapisuji na vsechny disky,
- moznost cist a zapisovat paralelne,
- vyssi spolehlivost (data jsou na vsech discich)

5.5.3 RAID 2

- nejslozitejsi, proto se prilis nepouziva,
- pouziva zabezpecovaci Hemingovy kody,
- k urcitemu poctu datovych disku je urcity pocet zabezpecovacich disku,
- data se ukladaji na datovych discich na urovni bytu, k nim se dopocitavaji zabezpecovaci kody (napr. 4 datove - 3 zabezpecovaci),
- byty dat se rozlozi do vsech disku (ofc ty se musi prevest do bajtu a sektoru a zapisuje se to po sektorech)
- jediny RAID, ktery umi detekovat chyby, nektere i sam opravit, dokonce umi i zjistit, ktery 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,
- nevyhoda 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 redudance 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 paritnich disku

- paritni disky pouzivane u RAID 3 - 6,
- jakmile clovek urci disk, ktery selhal, je mozne zreprodukovat jeho obsah,
- priklad: 4 disky, 1 paritni, treti datovy selze
  - prvni byty v datovych jsou 010 (potom v paritnim aby byl sudy pocet je 1), dalsi byty jsou 111 (licha parita, do paritniho disku se doplni 1 na sudou), dalsi jsou 011 (suda - v paritnim je 0)
  - selze treti disk, vymeni se za novy, prazdny
  - dopocitaji se data opet na sudou paritu: mam prvni byty 01? a v paritnim 1 - aby byla suda, v novem disku musi byt 0, dalsi byty 11? a v paritnim 1 - v novem musi byt 1, apod. ...

definice: (pro 5.5.x)

RAID = Redundant Array of Independent Disks
parita (bitu) je sudost/lichost bitu, pocet sudych/lichych 1 bitu
parity bit zn., ze na MSB se prida 1 pokud pocet 1 (bitu) je lichy

5.7 Ulozeni dat na disku

Diskova jednotka neumozni pracovat s nicim mensim nez sektor, ale typicky OS si sektory nejak seskupi (do vetsi jednotky) a neumozni pracovat s nicim mensim, nez je alokacni blok.

Logicka a fyzicka naslednost:

- 1 alokacni blok se namapuje fyzicky za sebou na diskovem prostoru,
- vice alokacnich bloku jiz nemusi byt fyzicky na disku za sebou (filesystem se vsak snazi o to, aby tomu tak bylo)

definice:

alokacni blok neboli cluster je skupina pevneho poctu sektoru, typicky mocnina 2 (nejmene 2^0 = 1 alokacni blok), pro sektory v ramci alokacniho bloku je zaruceno, ze jdou za sebou logicky i fyzicky (na disku) v souboru, dale je to nejmensi jednotkou diskovou diskoveho prostoru, se kterym bezne pracuje jadro (filesystem, uzivatel).

5.8 Fragmentace

5.8.1 Externi fragmentace

Rozumime jev, ktery vznika v pametech postupnym obsazovanim a uvolnovanim pameti, kdy v pameti vznika sekvence oblasti, ktere jsou volne a pouzite (a pouzite ruznymi soubory).

Priklad externi fragmentace:

- na disku vytvorim soubor 1, zabira urcite misto,
- pote dalsi soubor 2,
- pote soubor 1 chci zvetsit, tak se soubor 1 rozdeli na 2 casti - soubor 1.1 (puvodni misto kde byl - pred s2) a soubor 1.2, ktery bude za souborem 2,
- stejnym zpusobem zvetsim soubor 2 a vznikne sekvence s1.1, s2.1, s1.2, s2.2,
- nyni se rozhodnu smazat prvni soubor a budu mit sekvenci volne misto, s2.1, volne misto, s2.2 == externi fragmentace.

Externi fragmentace je i na plne obsazenem disku, kde staci, aby byl disk obsazen soubory nespojite (tzn. jeden soubor je rozdelen do vice casti, neni ulozeny na jednom miste, napr. s1.1, s2, s1.2 nebo viz priklad hore).

Negativni dopady externi fragmentace:

- na disku za urcitych okolnosti (v beznych FS nevznikaji) mohou vzniknout casti prostoru, ktere jsou jiz dale nevyuzitelne, protoze jsou prilis male (tldr vznik volnych useku, ktere nejdou vyuzit)
  - okolnosti (pri kterych vzniknou nevyuzitelne casti prostoru): pri alokovani diskoveho prostoru spojite (na miru souboru ci jeho castem, nepridelovani po jednotkach pevne velikosti) a navic budu mit dolni mez urcujici velikost diskoveho prostoru tak, aby byl pouzitelny (muze vzniknout v souvislosti s tim, ze do pouzitych diskovych oblasti si mohu ukladat pomocne informace, k cemu se pouzivaji - pokud bude informace vetsi nez ”volna dira”- nepouzitelna) - mam na disku bloky volneho mista o pozadovane velikosti (1GB, soubor, 0.5GB), ale protoze chci ukladat spojite (soubor o velikosti 1.5GB), nelze takove misto vyuzit
  - vznikne nespojite rozlozeny soubor (viz priklad) a je nutne si pamatovat v pomocnych datech = metadatech informace o tom, kde jednotlive casti souboru jsou (ukladaji se na mista, kde jednotlive casti jsou, ”odkazuji”se na dalsi metadata - dalsi casti smazaneho souboru),
- cim vice casti souboru - tim vice metadat - cim vice fragmentovane - tim vice je pristup na data pomalejsi (u HDD se ceka navic na natoceni hlavicek a rotace disku)

Souborove systemy se snazi negativni dopady fragmentace minimalizovat:

- rozlozeni souboru po disku (snaza ukladat soubory na disk tak, aby nebyly nutne za sebou, ale bylo mezi nimi volny prostor),
- pouzivani predalokace (uzivatel si pozada filesystem o vymezeni urciteho prostoru na disku, napr. databaze),
- odlozena alokace (=allocate-on-flush, filesystem nezapise ihned po zmene souboru, ale chvili pocka - pocita s tim, ze uzivatel bude chtit menit soubor ”za chvili”znovu - az nebude delsi dobu dochazet 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, ktere 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 fragmetace:

- 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, ktere 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 Pristup na disk

(viz 2.5.5) Proces kdyz chce nacitat/zpracovavat data, zavola sluzbu k tomu urcenou (read, write, .. - muze byt zabaleno i v nejakem knihovnim volani, napr. scanf zavola read), dojde k predani rizeni jadru, dostane se ke slovu jadro, podiva se do cache, pokud tam ta data ma, preda je, pokud ne, musi je nacist z disku - s diskem komunikuje pres I/O porty nebo pametove mapovane I/O porty, disku se predavaji prikazy pres jeho rozhrani (ATA disk - ATA prikazy), jdou z filesystemu pres ovladac prislusneho disku (pote to prochazi sbernicemi), ten komunikuje s radicem disku - disk dostane prikaz, jakmile dobehne operace, disk posle preruseni na procesor, tam se dostava ke slovu jadro, to zpracuje preruseni a zachova se podle nej (uspech - preda data, chyba - zpracuje ji).

definice:

ovladac je software, ktery umi komunikovat s urcitym typem zarizeni, jina definice viz. 2.4

5.10 Planovani pristupu na disk

Soucasti jadra je subsystem nazyvany planovac diskovych operaci, ktery shromazduje pozadavky od filesystemu (nacteni, zapsani dat z/do disku). Planovac si uklada pozadavky do svych planovacich front, pozadavky pripadne preusporadava a predava dal ovladaci ci radici disku k realizaci.

Planovac se snazi minimalizovat rezii disku.

Jednou ze strategii preusporadavani pozadavku (u HDD) je pouziti **vytahoveho alogritmu (elevator, SCAN alghorithm):**

- snaha, aby se hlavicka disku plynule pohybovala od stredu k okraji a zpet a vyrizovat pozadavky dle pohybu hlavicky,
- modifikace SCAN algoritmu je napriklad Circual SCAN, kdy se pozadavky vyrizuji pouze pri jednom smeru,
- dalsi modifikace jsou LOOK a C-LOOK, kde se hlavicka nepohybuje od stredu k okraji, ale pouze v tom rozsahu, kde je potreba provadet operace.

Planovac se muze snazit vice operaci sloucit do jedne operace (napr. operace v ramci jednoho bloku se sdruzi):

- takove kroky maji vyznam i u SSD,
- snaha vyvazovat pozadavky jdouci od jednotlivych uzivatelu (procesu),
- implementace priorit (prioritnejsi proces - pozadavky se vykonaji drive),
- snaha odkladat operace tak (v nadeji), ze je bude pote mozne sloucit,
- snaha implementovat casova omezeni na dobu cekani pozadavku,
- muze implementovat paralelizaci pozadavku predavanych do diskoveho subsystemu (modernejsi a velmi vykonne SSD umi resit operace paralelne).

linux:

pro zjisteni, jaky planovac pouzivame se staci podivat do */sys/block/<devname>/queue/scheduler*

5.11 Logicky disk

V pocitaci je mozne mit vicero fyzickych disku, ktere je dale mozne rozdelit na logicke disky a konkretni souborove systemy je mozne instalovat na logicke disky. Pro spravu a vytvareni logickych disku lze pouzit programy cfdisk, disk, gparted, ..

5.11.1 Zpusob ulozeni informaci o diskovych oblastech na disku

- MBR
  - v prvnim (nultem) sektoru byla tabulka obsahujici rozdeleni na 1-4 primarni partitions
  - pokud bylo nutne pouzit vice partitions, potom misto primarni se nahradila rozsirenou diskovou oblasti, ktera se dale mohla rozdelit na podoblasti zvane logicke diskove oblasti, kazda z nich popsana formou zretezeneho seznamu, EBR
  - pouzivane u starsich PC
- GPT
  - je tabulka (pole) o az 128 odkazech na jednotlive diskove oblasti,
  - stejny vyhrazeny prostor jako u MBR

5.11.2 LVM

- spravce logickych oblasti,
- umoznuje pokrocilejsi tvorbu logickych disku a
- do logickeho disku pridavat fyzicke disky (za behu),
- LVM muze byt bud primo ve filesystemu nebo v casti jadra (mezi filesystemem a planovacem).

5.11.3 Ruzne typy souborovych systemu

- fs (prvni 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, pote pouzivany ve Windows - velmi jednoduche a siroce podporovane),
- F2FS (fs pro efektivni prace se SSD), ISO9660, UDF, Lustre, GPFS (clustery, superpocitace),
- ZoneFS (ZFS).

Po koupe noveho disku a rozdeleni na logicke disky je nutne se rozhodnout, jaky souborovy system na prislusnem logickem disku bude pouzivan - je nutne disk **zformatovat** pro pouziti. Drive se pouzivalo i nizkourovnove formatovani (stare disky s nestabilnim magnetickym zaznamem).

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, ktera v jadre 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: pokracovani Spravy souboru. Zurnalovani, jeho implementace a alternativy, Copy-on-write, Klasicky UNIXovy system souboru FS, i-uzly, kde a jak jsou data ulozena, pocty odkazu, limit maximalni velikosti souboru, vyhody a nevyhody FS, jine zpusoby organizace souboru, EXT4, NTFS, Organizace volneho prostoru na disku, deduplikace, typy souboru v UNIXu, adresar, montovani disku

6.1 Zurnalovani

Je technika zalozena na vytvareni zurnalu.

- souborove systemy se zurnalem jsou treba ext3, ext4, ufs, XFS, JFS, NTFS, ..
- zurnalovani umoznuje spolehlivejsi (nikdy nemame obecne zajisteno, ze se nic spatneho stat nemuze) a
- rychlejsi navrat (nez nejake utility) do konzistentniho stavu po chybach
- data obvykle zurnalovana nejsou (velka rezie), ale mohou byt
- zavisi na tom, ze operace, ktere zurnalovani implementuji, se provedou ve spravnem poradi – nutnost spoluprace s planovacem, take disky si samy data preusporadavaji (nelze nijak ovlivnit)

Zurnal:

- zapis zurnalu je předřazený,
- vytvari se v nem cyklicky prepisovany buffer,
- předřazenost zapisu do zurnalu mi zaruci, ze operace pokryte zurnalovanim jsou atomicke - vytvari transakce

Kompromis mezi zurnalovanim a nezurnalovanim dat je **předřazení zapisu dat na disk pred zapisem metadat do zurnalu** (a nasledne zapis ostrych metadat na disk). Priklad:

- zapisuji do souboru - bud vytvarim zcela novy nebo ho zvetsuji (typicke zpusoby zapisu),
- pri zvetsovani se nejprve zapisou data na disk za existujici data (bez poznamenavani informace o tom, ze se soubor zvetsuje),
- pokud operace selze, soubor zustane v puvodnim stavu (diky neupravenym metadatum puvodniho souboru),
- teprve az data budou na disku, tak se do zurnalu zapise informace o zvetsovani souboru,
- pote se zmeni metadata souboru (a uzivatel se k datum dostane),
- pri selhani napajeni v moment, kdy jsou metadata v zurnalu, ale ne na disku, je mozne tyto metadata obnovit

Proces mazani souboru na disku:

- odstraneni zaznamu z adresare,
- uvolneni uzlu (metadat souboru),

- uvolneni oblasti pouzitych tim souborem

definice:

zurnal je specialni soubor ci specialni oblast na disku slouzici pro zaznamy modifikovanych metadat (dat o datech), pripadne i dat pred jejich zapisem na disk (v podobe beznych dat)
předřazený znamena, ze zapis do zurnalu se provede pred ostrym zapisem "uzitecnych"dat (ci metadat) na disk
atomicke operace zn., ze but operace uspeje cela (vsechny dilci kroky) nebo neuspeje vubec (zadny dilci krok)

6.1.1 Implementace zurnalovani

Existuji 2 zakladni pristupy k implementaci zurnalovani.
REDO:

- implementace na zaklade dokonceni transakci,
- pouziva napr. ext3, ext4,
- sekvence dilcich operaci (vytvarejici tu operaci, kterou chci provest) se zapise do zurnalu (zacatek, konec transakce, kontrolni soucet),
- pote se operace provadi na disku,
- po uspesnem dokonceni se transakce ze zurnalu uvolni,
- pri selhani a pote zotaveni se system podiva do zurnalu, podiva se po neuvolnenych transakcich, jestli jsou cele - pocatecni a koncova znacka, jestli sedi kontrolni soucet - pokud vse sedi, tak system provede vsechny operace znovu

UNDO:

- implementace na zaklade anulace transakci,
- v kombinaci s REDO se pouziva v NTFS,
- proklada zaznam dilcich operaci (ktere se maji provest) do zurnalu a nasledne jejich provedeni na ostrych datech (zaznamena dilci operaci - provede ji),
- probehne cela transakce - zaznam ze zurnalu se uvolni,
- pri chybe se eliminuji vsechny nedokoncene transakce (vsechny provedene dilci kroky se musi vratit - vrati se disk do puvodniho stavu)

Pri implementaci zurnalovani je klicove **dodrzeni poradi kroku, ve kterem se provadeji**. (U REDO napr. je nutne, aby se nejprve zapsaly sekvence operaci do zurnalu a teprve pote se provadely operace na disku) Pokud tato sekvence nebude dodrzena, zurnal nebude spravne fungovat.

6.1.2 Copy-on-write

Je alternativa k zurnalovani pouzivana napriklad 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 nejnovejsim 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, ktery popisuje veskery obsah disku, typicky se v nem vyhledava na zaklade unikatni identifikace souboru

adresare (copy-on-write) jsou specialni soubory ulozeny na disku, ktere 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:

- logovaci 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 Klasicky UNIXovy system souboru (FS)

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

Souborovy system byl rozclenen (na urovni logickych disku) na:

- boot blok - obsahoval informace (kod, cast kodu) potrebne pro zavedeni pri startu,
- super blok - informace o souborovem systemu (typ, verze, velikost, pocet i-uzlu, volne misto, korenovy adresar, volne i-uzly, ..),
- tabulka i-uzlu - tabulka (pole n i-uzlu) pouzita s popisy souboru,
- datove bloky - data souboru, metadata (pomocne adresovaci bloky).

Zakladni rozlozeni FS bylo zmodifikovano v navazujicich filesystemech:

- datove bloky byly rozdeleny do skupin,
- kazda skupina mela svoje i-uzly,
- duvodem byla lepsi lokalita, prostorova blizkost dat a metadat (typicky pri praci se soubory jsou nutne i jeho metadata),
- pote tedy ta struktura vypadala takto: boot blok, super blok, usek i-uzlu, usek dat, usek i-uzlu, usek dat, ..

definice:

i-uzel je zakladni datova struktura reprezentujici kazdy jeden soubor v typickych UNIXovych systemech (pozn. pri formatovani se urci dopredu maximalni pocet souboru, ktere na diskovem oddilu budou existovat)

6.2.1 i-uzel

Zakladni datova struktura popisujici soubor v UNIXu (nebo viz definice hore). Ke kazdemu souboru musi byt i-uzel. Ten obsahuje metadata o souboru:

- stav i-uzlu (alokovany, volny)
- typ souboru (obycejny, adresar, zarizeni, pojmenovana roura, ..),
- delka souboru v bajtech,
- casy mtime (posledni modifikace dat - zapis), atime (posledni pristup - cteni), ctime (posledni modifikace i-uzlu),
- UID, GID,
- pristupova prava (cislo, napr. 0644 = rw-r–r–),
- pocet pevnych odkazu (neboli jmen souboru),
- informace o tom, kde se nachazi data o souboru (tabulka odkazu na datove bloky a dalsi informace nebo odkazy na pomocne bloky s dalsimi metadaty, napr. ACL, extended attributes, dtime - udaj o smazani souboru, ..)

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

definice:

UID je cislo identifikace vlastnika

GIT je cislo identifikace skupiny

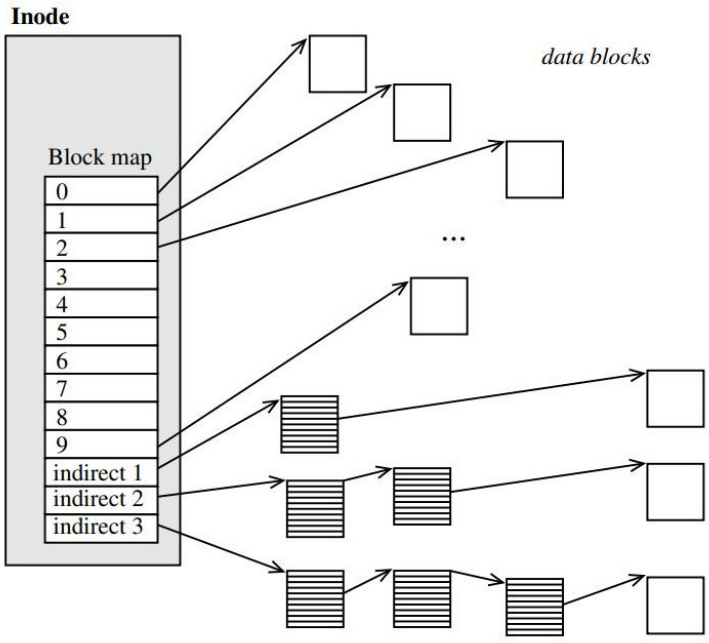
ACL zn. Access Control List, pristupove seznamy rozsirujici zakladni UNIXova prava tak, ze je mozne priradit konkretni prava ke konkretnim uzivatelum

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

ctime je posledni zmena i-uzlu, vyuzitelne napr. pokud se zfalsuje 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 prvni 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) - vznika 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 Pocty odkazu

- neprimy odkaz 1. urovne - pri 4 KiB clusteru je to 1024 odkazu (1 odkaz = 4B) = 1024 datovych bloku,
- neprimy odkaz 2. urovne - pri 4 KiB clusteru je to 1024^2 odkazu = stejny pocet datovych bloku,
- neprimy odkaz 3. urovne - pri 4 KiB clusteru je tam 1024^3 odkazu = stejny pocet datovych bloku

6.2.4 Limit maximalni velikosti souboru

Poctem primych odkazu neprimeho odkazu 3. urovne je dan maximalni pocet bloku, ktere je mozne v tomto souborovem systemu ulozit. Teoreticky limit velikosti souboru je tak:

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

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

Toto omezeni velikosti je pouze jednim z omezenim, ktere velikosti souboru omezuji. **Dalsi omezeni jsou dana:**

- dalsimi datovymi strukturami a typy, ktere pouziva FS (napr. datovy typ delky souboru v bajtech v i-uzlu),
- strukturami VFS (veskera prace s jakymkoli filesystemem musi projit pres VFS),
- rozhranim jadra,
- architekturou systemu (32b - velikost souboru bude 32b cislo + MSB je pouzit pro indikaci chyby [-1 bit pro data] - soubory maximalne do 2 GiB nebo dnes bezna architektura 64b - 64b velikosti)

Existuje Large File System Support, kde ve 32b systemu se nahradi vsechny udaje kde se pracuje s velikosti vetsim datovym typem - podpora souboru *vetsich jak 2 GiB*.

linux:

du [soubor] vypise zabrane misto v blocich vc. rezie (metadat)
ls -l [soubor] vypise velikost souboru v bajtech (pouze uzitecna data)
df vypise volne misto na discich
ls -i [soubor] zpristupni cislo i-uzlu souboru
is -e /dev/... n - vypis i-uzlu n na /dev/...
dumpe2fs - zakladni informace o souborovem systemu ext2,3,4
/dev/zero je soubor typu zarizeni generujici proud 0
dd if=[source] of=[dest] je nizkourovnove kopirovani

6.2.5 Vyhody a nevyhody 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 prvniho nebo urciteho 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 zpusoby organizace souboru

6.3.1 Kontinualni ulozeni

- neboli spojite ulozeni souboru na disku,
- na disku je jeden spojity usek dat reprezentujici soubor,
- vyhodami jsou rychle vyhledani adresy 1./urciteho bloku nebo vyhledavani nasledujicich bloku,
- nevyhody: soubory nebude mozne jednoduse zvetsovat pokud budou prilis blizko u sebe (bude nutne jej presunout na jine volne a vetsi misto, pokud to pujde ci provest defragmentaci a pote zvetsit soubor)
- nepouziva se prilis (kvuli sve nevyhode)

6.3.2 Zretezene seznamy alokacnich bloku

- kazdy alokacni blok obsahuje sva (uzitecna) data a na konci obsahuje odkaz na nasledujici alokacni blok,
- kvyhodami jsou rychly pristup na zacatek ci pruchod daty,
- knevyhodou je presun na nahodne misto v souboru - nutnost precist cely soubor az po dany blok (1 GiB soubor, chci posledni blok - musim precist cely),
- kdalsi nevyhodou je rozprostreni metadat po celem disku - pri drobne chybe na disku prijdu o data (tedy i metadata, kde jsou odkazy na nasledujici bloky) a dojde k velke ztrate dat (vsechna data ”za”ztracenymi daty jsou nepristupna),
- kneni prilis vhodna, nicmene se pouziva v souborovych systemech FAT

6.3.3 FAT

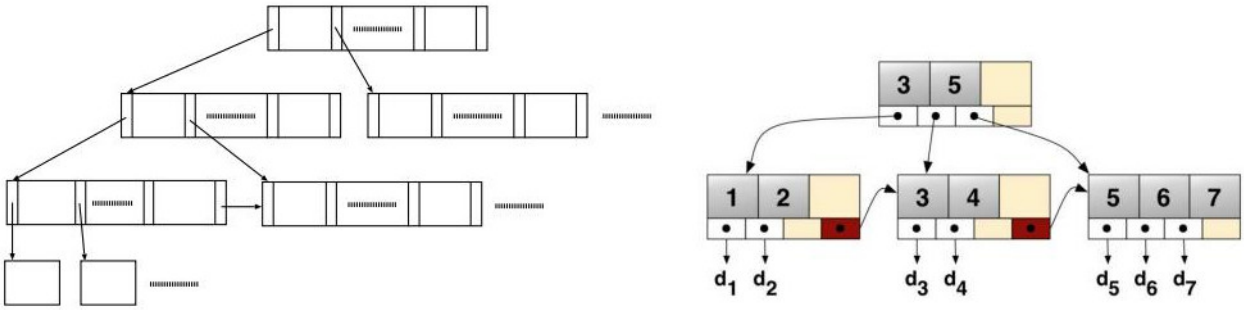
- File Allocation Table,
- od zretezenych seznamu se lisi tim, ze seznamy popisujici rozlozeni souboru na disku jsou ulozeny v separatni oblasti na disku (tzv. FAT),
- kde jsou tato data koncentrovana - rychleji se prohledavaji, lze vytvorit tak vice kopii FAT (prevence okamzite ztraty dat pri chybe),
- stale vznikaji problemy s rychlosti pri nahodnem pristupu (stale jde o zretezeny seznam)
- tabulka je pole, ktere obsahuje pro kazdy blok na disku 1 polozku, kazda polozka obsahuje odkaz na dalsi blok/polozku,
- pouziva se i dodnes (a je to velmi rozsirene), protoze je to jednoduche (napr. vestavene systemy)

6.3.4 B+ stromy

- jsou datovou strukturou prevzatou z databazovych systemu,
- maji dva typy uzlu - vnitrni a listove,
- vnitrni uzly jsou koren, jeho naslednici krome listovych, obsahuji odkaz na naslednika a vyhledavaci klic,
- listove uzly take obsahuji odkazy a vyhledavaci klice, odkazy vedou na data na disku, posledni odkaz na poslednim listu odkazuje na list na stejne urovni (jsou tak propojeny linearnim seznamem),
- pouzivaji se za ucelem popisu rozlozeni dat na disku (obsah souboru, pote vyhledavaci klic bude offset - cislo logickeho bloku v ramci souboru) nebo se pouzivaji pro adresare (klice budou jmena souboru) nebo pro popis celeho obsahu disku (klic je dvojice i-uzel a posuv souboru)

Vyhledavani v B+ stromu:

- pri hledani klice (k) se podivam, zda je klic mensi jak klic k0, pokud ano, pujdu nize, kde je k00, pokud ne, zjistim, jestli je klic mezi k0 a k1, pokud ano, jdu druhym smerem, opakuji po kn,
- pokud jsem niz, opakuji to same co vys az nedojdu k listovym uzlum,
- zde hledany klic najdu nebo zjistim ze v teto strukture klic neni
- pote mam odkaz na datovy blok,
- v pripade ze chci cist dal, tak jdu linearne po sobe po nasledujicich listovych uzlech



Obrázek 2: z prezentace IOS: Souborove systemy, slide 27 - B+ strom

Prace s B+ stromy:

- jsou zde limity jak moc/malo maji byt uzly zaplneny (strom se udrzuje vyvazeny) - pro uzly s m odkazy mame klice $0, 1,$ .. az $m - 2$ klicu (odkazu je o 1 mene nez klicu + cislovani od 0),
- pokud je strom tvoren solo korenem - nejmene muze mit 1 odkaz, maximalne $m - 1$ odkazu (posledni odkaz je pouzit jako ukoncovac seznamu listu),
- pokud to neni solo koren, tak ma nejake nasledniky, minimalne jich ma tak 2, maximalne m,
- vnitrni uzel ma tak $\frac{m}{2}$ (zaokr. nahoru) az m odkazu, list $\frac{m}{2} - 1$ (opet $\frac{m}{2}$ zaokr. nahoru) az $m - 1$ odkazu,
- vlozeni:
  - nejprve projdeme stromem od korene k listum,
  - najdeme kam chceme vlozit,
  - podivame se, zda ma list volny odkaz,
  - pokud ano - pouzijeme ho, pokud ne - list se rozstepi na 2 poloviny a podivam se o uroven vys, zda je mozne namisto 1 listu linkovat 2 listy,
  - pokud ano - prida se odkaz, pokud ne - nadrazeny uzel se musi rozstepit a postupovat o uroven vys,
  - ... stepi se strom az pripadne se rozstepi koren a strom bude mit 2 koreny
- ruseni:
  - se opet od listove urovne, tak, ze se zrusi odkaz v listu,
  - zkontroluje se, zda je uzel zaplneny v ramci danych limitu,
  - pokud ano - gut, pokud ne - podivam se na sousedni uzly a pokud se provest prerozdeleni tak, aby byly vsechny uzly naplneny v ramci limitu,
  - pokud se to nepodari, tak dojde ke slouceni listu,
  - posunu se o uroven vys, zrusim jeden odkaz, zkontroluji opet limity, zopakuji to same,
  - ... az se muze stat, ze se zrusi i koren

B+ stromy a jeho varianty jsou pouzivany pro popis diskoveho prostoru v filesystemech jako XFS, JFS, ZFS, Btrfs, ReFS, .. v omezene podobe tzv. stromu extentu v EXT4, podobna struktura je i v NTFS

definice:

solo koren = jediny koren

6.3.5 Extent

- pouziva se ke zrychleni prace s velkymi soubory,
- umoznuji zmensit objem metadat (je mozne rict, ze nektere 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 spojitemu 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 clustero, 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

- podporovana ZFS, NTFS, Btrfs, XFS, (ext4 ne) ..
- snazi se odhalit opakovane ukladani stejnych dat na disk a ulozi je pouze jednou a odkazuje se na ne vicenasobne.
- systemy s deduplikaci se snazi takova data detekovan (sekvence bitu, bloku, extentu, ..)
- zalozeno na kryptografickem hashovani (hledaji se data se stejnym popisem, urci se shoda),
- muze byt realizovano pri zapisu nebo dodatecne (zadost uzivatele),
- muze usporit diskovy prostor (pri virtualizaci, na mail serverech, repozitare, ..), pametovy prostor i cas (zamezi se opakovanemu cteni i zapisu),
- pri mensim objemu duplikace muze naopak zvysit spotrebu CPU casu, spotrebu pametoveho i diskoveho prostoru

Rozdil oproti copy-on-write:

- copy-on-write se muze uchytit (klony, snimky) pouze tehdy, pokud duplikaty vzniknou cinnosti samotneho filesystemu (napr. vytvoreni virtualek),
- zatimco deduplikace aktivne vyhledava duplikaty (napr. uzivatel, co stahuje stejne reklamni letaky)

6.8 Typy souboru v UNIXu

- - je obycejny soubor,
- d adresar,
- b blokovy specialni soubor,
- c znakovy specialni soubor,
- l symbolicky odkaz (symlink),
- p pojmenovana roura,
- s socket

definice:

specialni soubor je typ souboru reprezentujici hardwarove zarizeni (disk, hw, pamet) se kterym se komunikuje po blocich nebo znacich
symbolicky odkaz je soubor obsahujici jmeno jineho souboru (odkazuje na nej)

6.9 Adresar

Je to kolecke 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)

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

Novejsi technologie umoznuji automaticke montovani nove pripojenych zarizeni:

- na linuxu bezne pracuje system udev, ktery
- 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 stejnymi jmeny vznikaji kolize, ty se resi napr. preddefinovanim priorit pripojovanych filesystemu a zpristupni se soubor daneho jmena z logickeho disku, ktery ma nejvetsi prioritu
- UnionFS ma copy-on-write semantiku, coz umoznuje emulaci prepisovani neprepisovatelnych medii (v 1 vetvi CD, neprepisovatelne, 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

7

Sedma prednaska: Pokracovani Spravy souboru. Symbolicke odkazy,

7.1 Symbolicke odkazy

TUDU