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.

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 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,
- evyhoda 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 meboli 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 ¿ ”volna dira”- nepouzitelna)
  - 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 (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,

6.1 Zurnalovani

TODO