INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

VYSOKÁ ŠKOLA BÁŇSKÁ – TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA STROJNÍ

OPERAČNÍ SYSTÉMY

STRUKTURA ODKLÁDACÍHO ZAŘÍZENÍ

doc. Dr. Ing. Oldřich Kodym

Ostrava 2013

ISBN 978-80-248-3053-7

Tento studijní materiál vznikl za finanční podpory Evropského sociálního fondu (ESF) a rozpočtu České republiky v rámci řešení projektu: CZ.1.07/2.2.00/15.0463, MODERNIZACE VÝUKOVÝCH MATERIÁLŮ A DIDAKTICKÝCH METOD

OBSAH

9. STRUKTURA ODKLÁDACÍHO ZAŘÍZENÍ

OBSAH KAPITOLY:

Struktura a geometrie disku.

Plánování přístupů k disku, algoritmy obsluhy diskových front.

Management disku.

Management swapovacího prostoru (virtuální paměť).

Spolehlivost disku, disková pole.

MOTIVACE:

Chod operačního systému využívá mnoha standardních mechanizmů známých z jiných oblastí řízení i běžného života. Pevný disk (HDD) ve výpočetním systému má „výjimečné“ postavení. Je to zařízení s významným podílem mechanických částí, které způsobují, že výkonové parametry jsou i o více než 5 řádů horší než parametry ostatních komponent. Optimalizace tohoto subsystému tedy mají přímý vliv na výkon celého výpočetního systému. Systém souborů můžeme rozdělit do tří logických částí. V předchozích kapitolách jsme si ukázali uživatelské i programové rozhraní systému souborů a popsali vnitřní datové struktury a algoritmy, které OS užívá při implementaci těchto rozhraní. V této kapitole se budeme věnovat nejnižší části systému souborů – struktuře odkládacího zařízení.

CÍL:

Obsluha disku – geometrie a přístup, plánování FCFS, SSTF

Obsluha disku – plánování SCAN, C-SCAN, LOOK, C-LOOK

Management swapovacího prostoru

Spolehlivost disku, disková pole

1. ÚVOD

Systém souborů můžeme rozdělit do tří logických částí. V kapitole 7 jsme si ukázali uživatelské i programové rozhraní systému souborů. V kapitole 8 jsme popsali vnitřní datové struktury a algoritmy, které OS užívá při implementaci těchto rozhraní. V této kapitole se budeme věnovat nejnižší části systému souborů – struktuře odkládacího zařízení. Nejdříve krátce popíšeme algoritmy plánování disku, potom se zmíníme o jeho formátování, o vlastnostech a udržování bodovacího bloku, o poškozených blocích i o managementu prostoru pro odkládací paměť.

2. STRUKTURA DISKU

Disk (blokové zařízení s přímým přístupem) představuje v moderních počítačích nejběžnější odkládací zařízení. Před ním bývala k tomuto účelu užívána magnetická páska. Ta je však velmi pomalá v porovnání s vybavovací dobou operační paměti a zejména umožňuje pouze sekvenční přístup k souborům. V současné době jsou pásky užívány převážně jako zálohovací zařízení, případně pro ukládání informací, které nejsou často vyžadovány nebo jako transportní medium.

Na informace uložené na disku se odkazuje prostřednictvím adresy, která se skládá z několika částí: jednotka (*drive*), strana (*surface*), stopa (*track*) a sektor (*sector*). Všechny stopy, které jsou v daném okamžiku přístupné bez posunu hlavy, (tj. ekvivalentní stopy na různých stranách) tvoří *cylinder*.

Uvnitř stopy jsou informace ukládány v sektorech. IBM mainframy umožnují uživateli vybrat si velikost sektoru, ale většina jiných disků má velikost sektoru definovanou hardwarově. *Sektor* je nejmenší jednotka informace, která může být čtena nebo zapisována na disk. Velikost sektoru se může v závislosti na typu disku pohybovat od 32 B po 4096 B, většinou bývá 512 B. V jedné stopě bývá 4 až 32 sektorů a stop na jedné straně může být 20 až 1500.

Chceme-li přistoupit ke konkrétnímu sektoru, musíme specifikovat celou adresu, tj. jednotku, stranu, stopu a sektor. Aby bylo možno se snáze mezi sektory orientovat, jsou odděleny speciální značkou.

Před čtením sektoru se musí čtecí/zapisovací hlavy přesunout na správnou stopu (*čas vyhledávání – seek time*), elektronicky je vybrána správná strana. Potom čekáme (*čekací doba – latency time*) než se požadovaný sektor natočí pod hlavu. Seek time je podmíněn časem nutným pro posun hlav, je-li hlava na vzdálené stopě, je doba přesunu delší.

I/O přenos mezi diskem a pamětí je organizován v jednotkách jednoho či více sektorů zvaných *bloky*, aby se přesun maximálně zefektivnil. Adresa konkrétního bloku zahrnuje číslo stopy (nebo cylindru), číslo strany a číslo sektoru. Na každý disk lze tedy pohlížet jako na trojrozměrné pole bloků. Obvykle bývá toto pole operačním systémem zpracováváno jako jednorozměrné. Adresa bloku roste s bloky na jedné stopě, potom se všemi stopami v cylindru a na závěr od cylindru 0 k poslednímu na disku.

Užijeme-li s jako počet sektorů v rámci stopy, t jako počet stop v rámci cylindru, potom můžeme konvertovat diskovou adresu i-teho cylindru, j-te strany, k-teho sektoru na jednorozměrné číslo bloku b:

$$b = k + s * (j + i * t)$$

Poznamenejme, že při tomto mapování vyžaduje přístup k bloku $b + 1$, byl-li předtím čten blok b přesun hlavy pouze v případě, že b byl poslední blok jednoho cylindru a $b + 1$ je první blok dalšího. Pouze v tomto případě dojde k posunu hlavy, pouze vsak o jednu stopu.

Výše uvedené adresování bloku na disku metodou CHS (cylindr, hlava/strana, sektor) již vyčerpalo velikost definovaných datových struktur pro jednotlivé položka. Namísto jejich rozšiřování se přistoupilo k novému způsobu adresování diskových bloků – LBA (*linear block addressing*). LBA ponechalo na straně operačního systému (nejnižší vrstvy souborového systému) pouze číslo bloku (b ve výše uvedeném vztahu). Ostatní části algoritmu již realizuje disk sám o sobě. LBA navíc umožnilo významně zvýšit kapacitu disků tím, že na jednotlivé stopy může umístit různé počty sektorů – při stejné hustotě záznamu je na vnější stopě (která je fyzicky delší než stopa vnitřní) více sektorů. Tuto situaci nezle starší metodou adresování CHS postihnout.

3. PLÁNOVÁNÍ DISKU

Protože mnoho úloh požaduje od disku program a případně vstupní a výstupní soubory, je maximálně potřebné, aby disk byl tak rychlý, jak je to jen možné. OS může zvýšit průměrnou vybavovací dobu disku pomoci plánování pořadí požadavků o přístup k disku.

Rychlost disku sestává ze tří částí. Aby bylo možno přistoupit k požadovanému bloku, je třeba nejprve přesunout hlavu na patřičnou stopu nebo cylindr. Tato operace bývá označována jako *vyhledávání – seek* a čas pro ni potřebný jako *doba vyhledávání – seek time*. Je-li hlava na správné stopě, musíme čekat, než se pod ni natočí správný sektor (*doba čekání – latency time*). Poslední složkou je čas nutný pro přenos dat mezi diskem a pamětí. Ta bývá nazývána *čas přenosu – transfer time*.

Každé I/O zařízení má svou frontu nevyřízených dotazů. Jakmile proces požaduje čtení nebo zápis na disk, provede volání příslušné služby operačního systému. Toto volání musí specifikovat následující nutné informace:

- jedná-li se o čtení či zápis,
- jaká je disková adresa (číslo bloku, převedené modulem organizace souborů na jednotku, cylindr, stranu a sektor – CHS),
- jaká je paměťová adresa od které má dojit k ukládání nebo čtení,
- jak velká informace má být přenesena (počet bytů).

Jestliže je požadovaný disk a jeho ovladač volný, může být žádosti okamžitě vyhověno. Jestliže však buď disk, nebo ovladač obsluhují jinou žádost, většinou žádost jiného procesu, musí být nový požadavek zařazen do fronty čekajících.

V multiprogramovém systému s mnoha procesy nejsou příliš často fronty požadavků na disk prázdné. V tom případě, je-li aktuální požadavek vyplněn, je třeba vybrat z fronty další a ten obsloužit. Tato obsluha zahrnuje přesunutí hlavy na patřičnou stopu, čekání na otočení disku a na závěr vlastní přenos dat. Jak jsme uvedli výše, tyto algoritmy jsou dnes implementovány přímo ve firmwaru moderních disků (např. NCQ). Takové disky jsou schopny přijmout více požadavků k vyřízení.

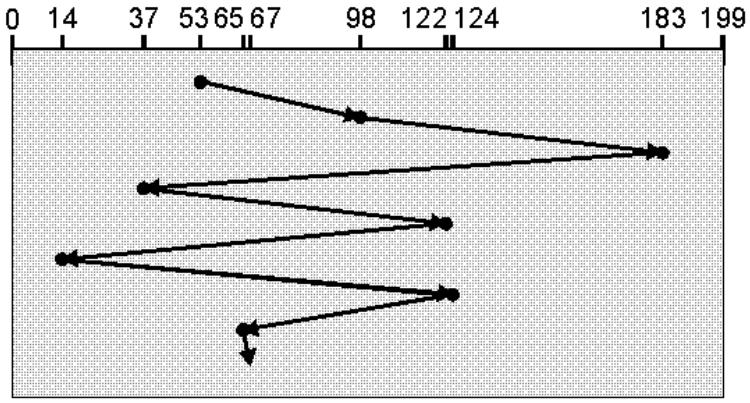
3.1 Plánování FCFS

Nejjednodušší metodou plánování disku je pochopitelně metoda *First-Come First-Served (první přisel, první dostal)*. Tento algoritmus je jednoduchý na naprogramování, nepodává vsak vždy optimální výkony. Uvažujme např. frontu požadavků na následující stopy:

98, 183, 37, 122, 14, 124, 65, 67

Jestliže je hlava na počátku na stopě 53, je třeba ji přesunout na stopu 98, potom na 183, 37, 122, 14, 124, 65 a na závěr na stopu 67, přičemž celkový přesun bude o 640 stop. Toto plánování je znázorněno na obr. 1.

Problém tohoto plánování jasně ilustruje například divoký výkyv od stopy 122 ke 14 a potom opět na 124. Pokud by požadavky na čtení ze sektoru 14 a 37 byly obslouženy po sobě, před nebo po požadavcích na stopy 122 a 124, celkový počet přecházených stop by podstatně poklesl, čímž by se zkrátila i vybavovací doba disku a zvýšil se tak jeho výkon.



Obrázek 1 - FCFS plánování disku FCFS

3.2 Plánování SSTF

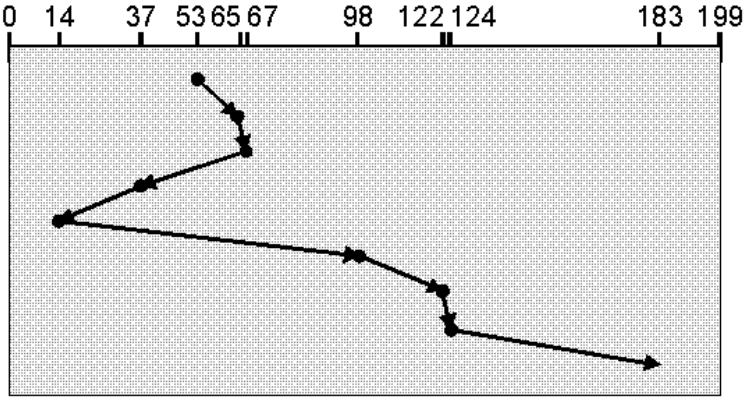
Zdá se tedy rozumné, obsloužit společně všechny požadavky, které se na disku odehrávají poblíž. Na tomto předpokladu je postaven plánovací algoritmus *Shortest-Seek-Time-First (první s nejkratší vyhledávací dobou)*. Algoritmus SSTF vybere z požadavků vždy ten, který vyvolá nejmenší přesun hlavy od aktuální pozice.

V našem příklade je první nejmenší přesun ze stopy 53 na stopu 65, další na stopu 67 atd. viz Obr. 91. Toto plánování vyžaduje v našem případě celkový přesun hlav o 236 stop, což je méně než třetina přesunu potřebného při FCFS. Implementací tohoto algoritmu dojde k podstatnému zvýšení rychlosti disku.

SSTF plánování disku je obdobou algoritmu shortest-job-first (SJF) při plánování CPU a stejně jako algoritmus SJF může vést k *umoření* některých požadavků. Připomeňme, že v reálném systému mohou I/O požadavky na disk přicházet v libovolném čase.

Mějme např. ve frontě dva požadavky na stopy 14 a 186. Jestliže v době, kdy je obsluhován požadavek 14 dorazí do fronty požadavek na stopu blízko u 14., bude příště obsloužen tento a ne požadavek na stopu 186. Pokud při jeho zpracovávání opět dorazí požadavek na blízké stopě, bude příště zase obsloužen tento. Teoreticky může tato situace vést k umoření požadavku na stopu 186. Je to sice nepravděpodobné, leč možné.

SSTF algoritmus, i když představuje významné vylepšení oproti algoritmu FCFS, stále není optimální. Jestliže bychom např. provedli přesun hlavy ze stopy 53 na 37, ačkoliv tento přesun nepředstavuje nejmenší vzdálenost a potom na 14 a dále bychom obsloužili požadavky v pořadí 65, 67, 98, 122, 124 a 183, redukovali bychom celkový počet prošlých stop na 208.



Obrázek 2 - SSTF plánování disku

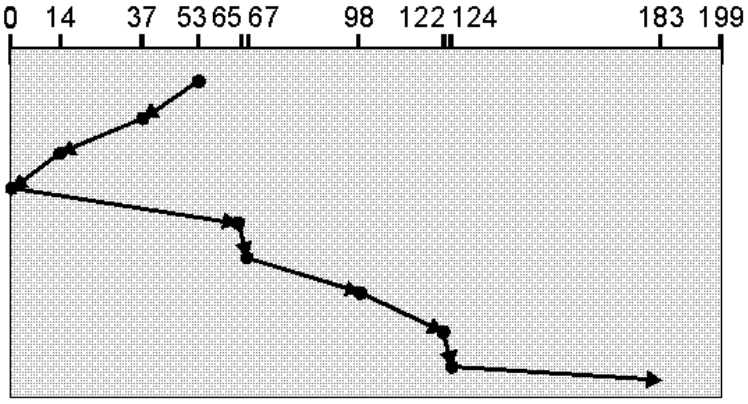
3.3 Plánování SCAN

Dynamická podstata plánování front vedla ke vzniku SCAN algoritmu. Čtecí hlava začíná na jednom konci disku a přesouvá se směrem ke druhému konci a obsluhuje jednotlivé požadavky v okamžiku, kdy dojede k patřičné stopě. Dobere-li se čtecí hlava opačného konce, obrátí směr svého pohybu a pokračuje dále. Čtecí hlava soustavně *projíždí (scans)* disk od jednoho konce ke druhému.

Opět použijeme stejného přikladu fronty požadavků:

98, 186, 37 122, 14, 124, 65, 67

Před tím, než můžeme aplikovat SCAN algoritmus, musíme znát směr pohybu hlavy. Hlava se opět nachází na stopě 53 a pohybuje-li se směrem k 0, budou nejprve obslouženy požadavky 37 a 14 a při pohybu hlavy nazpátek potom požadavky 65, 67, 98, 122, 124 a 183 tak, jak ukazuje obr. 3.



Obrázek 3 - SCAN plánování disku

Jestliže do fronty dorazí požadavek na stopu „před hlavou“, bude obsloužen okamžitě během aktuálního pohybu, a pokud přijde požadavek na čtení „za hlavou“, musí čekat, až se bude hlava pohybovat opět v opačném směru.

SCAN algoritmus bývá někdy označován jako *elevator algoritmus*, neboť je podobný chování výtahu, který obsluhuje požadavky na přepravu mezi patry budovy. Jinou analogií může být odhrnování sněhu z chodníku během sněhové vánice. Začneme na jednom konci chodníku, odhrnujeme sníh a v okamžiku, kdy jsme dorazili na druhý konec, obrátíme směr a odhazujeme nový sníh, který napadal za námi.

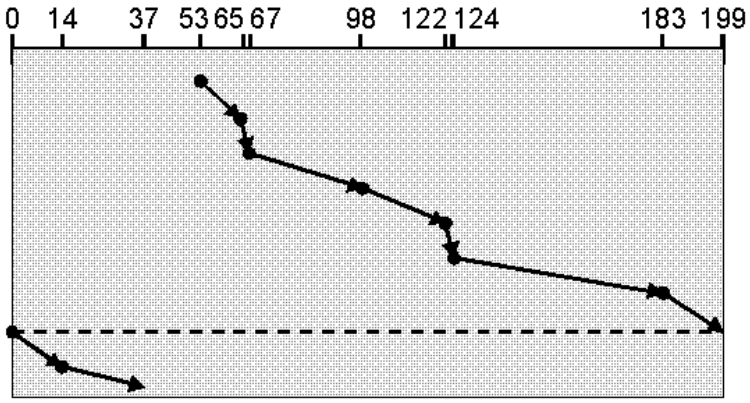
Předpokládáme-li rovnoměrné rozložení dotazů na jednotlivé stopy, uvažujme, jaká je hustota dotazů v situaci, kdy čtecí hlava dosáhla konce disku a mění směr. Nyní je obsluhováno často velmi málo požadavků, protože stopy, přes něž se hlava vrací, byly procházeny před malým okamžikem. Největší hustota požadavků je na stopy na druhém konci disku, neboť tyto požadavky čekají nejdéle.

3.4 Plánování C-SCAN

Variantou SCAN plánovacího algoritmu, která je navržena tak, aby čekací doba všech požadavků byla co nejjednotnější, je algoritmu C-SCAN (circular SCAN). Stejné jako algortimus SCAN i C-SCAN vyřizuje požadavky na disk tak, že posunuje čtecí hlavu od jednoho konce disku k druhému.

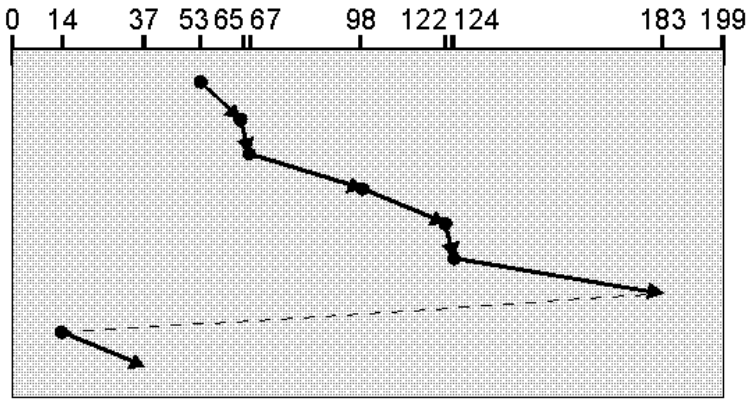
Jakmile čtecí hlava dojde do konce disku (na nejvyšší stopu) je okamžitě přesunuta opět na začátek (aniž by vyřizovala jakékoliv požadavky) a obsluhu žádostí provádí opět stejným směrem (viz obr. 4).

C-SCAN algoritmus podstatně upravuje pohled na disk. V tomto případě se zdá, jako by byl disk cyklický a poslední stopa navazovala na stopu první.



Obrázek 4 - C-SCAN algoritmus plánování disku

3.5 Plánování LOOK



Obrázek 5 - C-LOOK algoritmus plánování disku

Poznamenejme, že u obou dříve popisovaných algoritmů (SCAN i C-SCAN) dochází k pohybu čtecí hlavy od jednoho konce disku k druhému. Tímto způsobem však ve skutečnosti není žádný algoritmus implementován. Nejčastěji je hlava posunuta vždy k poslední žádosti v daném směru. Jakmile nejsou v aktuálním směru další požadavky, směr pohybu hlavy se

okamžitě obrátí. Tyto verze obou algoritmů jsou nazývány LOOK plánování (algoritmus se *podívá* po požadavku, než změní směr pohybu hlavy) resp. C-LOOK plánování.

3.6 Výběr algoritmu pro plánování disku

Máme-li k dispozici tolik algoritmů plánování disku, jak vybrat ten pro nás nejvhodnější? Algoritmus SSTF je velmi přirozený a častý. Algoritmů SCAN a C-SCAN jsou vhodnější pro systémy s velkým využitím disku. Je možné definovat optimální algoritmus, ale výpočty nutné pro optimální plánování mohou být náročnější než plánování pomocí SSTF či SCAN algoritmů.

Při užití kteréhokoliv plánovacího algoritmu je výkon disku značně podmíněn počtem a typem požadavků. Je-li ve frontě pouze zřídkakdy více než jeden požadavek na disk jsou všechny algoritmy stejně efektivní. Tehdy je vhodné využít ten nejjednodušší z nich – algoritmus FCFS.

Poznamenejme, že požadavky na disk jsou do značné míry ovlivňovány použitou alokační metodou. Program, který čte souvisle alokovaný soubor, vyvolá několik požadavků, které budou obslouženy na jednom místě disku. Naproti tomu spojovaný nebo indexovaný soubor obsahuje často bloky, které jsou široce roztříštěné po celém disku, což zvyšuje jeho využití za cenu časových ztrát při častých pohybech hlav.

Také umístění adresářů a indexových bloků má velký význam. Protože každý soubor musí být nejdříve otevřen a až potom užit a otevření souboru znamená prohledání adresářové struktury, je k adresářům přistupováno velmi často. Nutné přesuny čtecích hlav je možno částečně minimalizovat umístěním adresářů doprostřed disku, než k některému kraji. Je-li např. adresářová položka umístěna na první stopě a datový blok na poslední, potom se hlava musí nutně přesunout přes celý disk. Je-li adresářová položka umístěna poblíž středu disku, sníží se tato vzdálenost na polovinu.

Mělo by být jasno, že výsledkem těchto úvah je algoritmus plánování disku, který může být, stejně jako ostatní algoritmy, samostatným modulem operačního systému a může z něj být odstraněn a nahrazen jiným, je-li to zapotřebí.

Jak FCFS tak SSTF algoritmus je vhodnou počáteční volbou. Poznamenejme, že hardwarový ovladač disku napomáhá tvůrcům OS při plánování disku. OS posílá hardwarovému ovladači požadavky ve FCFS pořadí a ten je pak provádí v poněkud optimálnějším sledu.

Bohužel, jestliže si OS nevšímá metody, jakou ovladač používá, řadí požadavky do sledu, který se mu jeví jako optimální, avšak hardwarový ovladač toto pořadí změní podle svého a celkovým výsledkem může být i snížení výkonu systému.

4. SPOLEHLIVOST DISKU

Disk bývá nejméně výkonná součást výpočetního systému. Má stále poměrně časté výpadky, díky kterým dochází ke ztrátám dat a výpočetního času (kdy musí být počítač odstaven z důvodu opravy). Opravy vyvolané chybou disku mohou trvat hodiny (kopírovaní ze zálohy) a navíc restaurována data nebývají zcela totožná se stavem při selhání disku. Zálohování bývá prováděno jednou za den či za týden a všechny změny provedené po poslední záloze jsou nenávratně ztraceny. Zvýšení rychlosti a spolehlivosti disku je tedy významný úkol technologického rozvoje v této oblasti.

Některé techniky pro zvýšení výkonu disku již byly vyvinuty. Tyto techniky spočívají ve využití spolupráce několika disků. Pro zvýšení rychlosti byl vyvinut *disk stripping (interleaving),* technika, kterou již nyní využívají některé systémy. Několik disků je

udržováno jako jedno logické zařízení, v němž je každý blok rozdělen do několika subbloků. Každý subblok je potom uložen na jiném disku.

Tím dojde k radikálnímu snížení doby přenosu bloku mezi pamětí a diskem, neboť jsou bloky přenášeny paralelně s omezením daným pouze přenosovým výkonem I/O. Snížení je obzvlášť velké, jsou-li disky *synchronizovány* (tzn., otáčejí se synchronně – první sektor stopy se objeví pod čtecí/záznamovou hlavou každého disku ve stejném čase), protože tehdy nedochází k prodlevám mezi přístupy k subbloku na jednotlivých discích. Další výhoda je patrná je-li mnoho malých levných disků takto užito namísto několika disků velkých a drahých. Zvýší se vybavovací rychlost, protože prohledávání disku je paralelní a může po něm být přeneseno více dat.

Tato myšlenka se stala základem pro vývoj redundantního pole levných/nezávislých disků (*RAID – Redundant Array of Inexpensive/Independent Disks*), které zvyšuje výkon systému a zejména poměr cena-výkon. Navíc se provádí i duplikování dat pro zvýšení spolehlivosti (proto redundant…). Takovéto diskové pole může být organizováno různými způsoby s různým výkonem, vlastnostmi a cenou.

Nejjednodušší RAID organizace je *mirroring* či *shadowing*, která spočívá v duplikování dat na každém disku. Toto řešení je samozřejmě velmi drahé, neboť je třeba dvakrát více fyzického diskového prostoru k uchování téhož množství dat. Ve vyšší organizaci RAID nazývané *block interleaved parity* jsou data ukládána na disk v blocích jako při běžné konfiguraci. Navíc je však uložen také *paritní blok*. Tento paritní blok je paritou všech ekvivalentních bloků na každém disku pole. Je-li např. v poli 5 disků potom sektory 0 disku 1 až 4 mají svou paritu v sektoru 0 disku 5.

Jestliže jeden disk z této skupiny 5 disků selže, znamená to výpadek jednoho datového bitu, který může být dopočítán ze zbylých bitů a parity. Výpadek jednoho disku tedy neznamená ztrátu dat. K té vede až současný výpadek více disků. Dalším zefektivněním technologie RAID může být rozložení zátěže rozloženým uložením parity na všechny disky.

Díky dobrému poměru výkonu, spolehlivosti a ceny jsou pole RAID poměrně často užívána a mnoho výrobců přichází se softwarem i hardwarem pro podporu RAID. Je možné ukázat, že pole 100 levných disků s paritou na 10 discích má předpokládanou dobu ztráty dat (MTDL – *mean time to data loss*) 90 let, pokud u jednoho disku počítáme tuto dobu 2 – 3 roky. Výrobci u disků často udávají parametr jiný: MTBF (*mean time between failure*).

Současný trend v oblasti pevných disků – disky SSD – přináší díky konstrukci bez mechanických dílů výrazný nárůst výkonový parametrů. Rovněž je ale nezbytné řešit nové problémy vyplývající z vlastností způsobu ukládání dat do buněk typu SLC nebo MLC.

4.1 Disková pole RAID

Pojem RAID vznikl v roce 1988, kdy byla Univerzitou California – Berkeley vydána publikace A Case For Redundant Arrays of Inexpensive Disks. Písmenko I bývá vysvětlováno jednak jako Inexpensive = levný (například Adaptec), jednak jako Independent = nezávislý (například Microsoft). Obojí je pravda, protože RAID pole je složeno z obyčejných sériově vyráběných pevných disků, které nejsou nijak upravovány.

Technologie RAID prošla postupným vývojem, kdy byly zjištěné nedostatky odstraňovány, nebo bylo odlišným přístupem dosaženo vyššího výkonu výsledného RAID pole.

Odlišné způsoby ukládání dat jsou realizovány buď softwarově, nebo hardwarově. V softwarovém řešení obsluhuje zápis do pole RAID operační systém (resp. speciální mezivrstva nebo přímo ovladač zařízení), a proto se jedná o nejlevnější řešení, které však trpí některými nedostatky. Především to je snížení výkonu a zvýšení režie operačního systému, protože veškeré operace musí řešit vlastní výpočetní systém a vzhledem k redundanci ukládání dat naroste i objem I/O operací.

Hardwarové řešení tyto nedostatky odstraňuje pomocí speciálního zařízení (řadič), který obstarává obsluhu RAID sám a hlavní procesor počítače tak není zatěžován. Problémem je, že většina lacinějších RAID řadičů na trhu je ve skutečnosti softwarově řízena, takže se vlastně o hardwarové řešení nejedná. Řadič diskového pole může být realizován jako rozšiřující karta, která je umístěna ve skříni počítače. Pak je obvykle limitujícím prvkem množství diskových mechanik, které lze do skříně umístit. Nejčastěji je řadič pole realizován jako externí zařízení, kdy skříň je optimalizována pro umístění a provoz maximálního počtu disků.

Pokud dojde při provozu RAID pole k výpadku některého disku (resp. členu pole), dostane se pole do tak zvaného degradovaného stavu, ve kterém je jeho výkon typicky nižší, avšak stále jsou všechna uložená data k dispozici. Správce počítače vymění havarovaný disk za nový a ten začlení zpět do pole (anglicky hot add), čímž začne tak zvaná rekonstrukce pole (rebuild), při které jsou dopočítány chybějící údaje a zapsány na nový disk. Data jsou typicky během rekonstrukce stále přístupná. Po dokončení rekonstrukce je RAID pole opět tak zvaně synchronizováno a poskytuje příslušnou ochranu dat. Někdy je v poli trvale k dispozici rezervní disk (anglicky spare), takže rekonstrukce pole může být zahájena zcela automaticky, bez čekání na zásah správce pole.

RAID pole vytváří logický (virtuální) úložný prostor, se kterým se dá typicky pracovat stejným způsobem, jako by to byl jediný pevný disk. Jednotlivé disky v poli nazýváme členy pole. Implementace RAID pole je typicky taková, že data na degradovaném nebo na právě rekonstruovaném poli jsou stále k dispozici, i když obvykle se sníženým výkonem (rychlostí čtení a zápisu). Logický disk vytvořený řadičem pole se výpočetnímu systému jeví jako běžný disk. Řadič umožňuje na svých připojených discích definovat i více disků logických a tyto připojit k různým výpočetním systémům.

Použitím diskového pole RAID lze v systému získat disk s vyšší kapacitou, vyšší bezpečností, rychlejším přístupem nebo kombinaci těchto vlastností.

5. ODKAZY NA POUŽITÉ A DALŠÍ INFORMAČNÍ ZDROJE

[1] MRÁZEK, Lubor. Operační systémy. *Operační systémy* [online]. [cit. 2013-03-02]. Dostupné z: <http://homen.vsb.cz/~kod31/vyuka/opsys/os.html>.

[2] BLATNÝ, Jan., et.al. *Číslicové počítače*. Praha: SNTL, 1992. ISBN 04-516-80 : 33.00.

[3] Operační systémy. KLIMEŠ, Šárka. *Slezská univerzita v Opavě* [online]. 2013 [cit. 2013-03-02]. Dostupné z: <http://axpsu.fpf.slu.cz/~vav10ui/obsahy/os/ospredn/ospredn.pdf>.

[4] Operační systémy. VAVREČKOVÁ, Cyril. *Ostravská univerzita* [online]. 2013 [cit. 2013-03-02]. Dostupné z: <http://www1.osu.cz/~klimesc/public/files/OPSY1/Skripta/Operacni_systemy_1.pdf>.

[5] *Jiří Peterka - Přednášky* [online]. 2013 [cit. 2013-03-02]. Dostupné z: <http://www.earchiv.cz/i_prednasky.php3>.

[6] Úrovně diskových polí RAID. *RAID Labs* [online]. 2006 [cit. 2013-03-03]. Dostupné z: http://www.raid-labs.cz/zachrana-dat-a-obnova-raid-poli/novinky/urovne-diskovych-poli-raid

[7] ZAVORAL. Principy distribuovaných systémů. *Principy distribuovaných systémů* [online]. 2012 [cit. 2013-03-02]. Dostupné z: <http://ulita.ms.mff.cuni.cz/pub/predn/PDS/PDS.ppt>.

[8] KLIMEŠ, Cyril. Distribuované systémy. *Distribuované systémy* [online]. 2008 [cit. 2013-03-02]. Dostupné z: <http://www1.osu.cz/~prochazka/ds/SkriptaKlimes.pdf>.

[9] *Virtualizace* [online]. Wikipedie, otevřená encyklopedie, 2006 [cit. 2006-11-07]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Virtualizace>.

[10] *Intel zahajuje novou éru virtualizace* [online]. Intel, 2005 [cit. 2006-11-07]. Dostupný z WWW: <http://www.intel.com/cd/corporate/pressroom/emea/ces/247194.htm>.

[11] ŠINDELÁŘ, Jan. *Virtualizace – budoucnost IT infrastruktury?* [online]. Computer Press, a. s., 2005 [cit. 2006-11-07]. Dostupný z WWW: <http://www.zive.cz/h/Uzivatel/AR.asp?ARI=125685&CAI=2104>. ISSN 1214-1887.

[12] LORENC, Václav. *Virtuální servery v prostředí Linuxu* [online]. Brno, 2002 [cit. 2006-11-25]. Bakalářský projekt. Dostupný z WWW: <http://www.fi.muni.cz/~xlorenc1/mypage/doc/pdf/bc_project.pdf>.

[13] KREMSER, Luděk. *Virtualizace serverů, cesta k vyšší efektivitě IT* [online]. Fujitsu-Siemens Computers, 2005 [cit. 2006-11-07]. Dostupný z WWW: <http://www.tuesday.cz/upload/docs/d5a558fd-8b3f-4962-99f0-90c0bec00ef7.ppt>.

[14] HOLUB, Vladimír. *Virtualizace* [online]. DATA-INTER Opava, 2006 [cit. 2006-11-07]. Dostupný z WWW: <http://www.datainter.cz/web/dia.nsf/dx/Holub_virtualizace-datainter.ppt/$file/Holub_virtualizace-datainter.ppt>.

[15] HŮLKA, František. *Virtualizace - virtuální počítače ve vašem PC* [online]. Moderní správce, 2006 [cit. 2006-11-07]. Dostupný z WWW: <http://www.modernispravce.cz/clanky/tabid/53/ctl/ArticleView/mid/367/articleId/35/VirtualizacevirtulnpotaevevaemPC1dl.aspx>.

[16] VALÁŠEK, Michal. *Virtualizace: Virtual Server 2005 R2* [online]. 2000-2006 [cit. 2006-11-25].
Dostupný z WWW: <http://www.aspnet.cz/Articles/110-virtualizace-virtual-server-2005-r2.aspx>.

[17] *(Para)virtualizace pro každého – Xen* [online]. LinuxEXPRES, 2006 [cit. 2006-11-07]. Dostupný z WWW: <http://www.linuxexpres.cz/para-virtualizace-pro-kazdeho-xen>. ISSN 1801-3996.

[18] *VMware Infrastructure 3* [online]. VMware, Inc., 2006 [cit. 2006-11-25]. Dostupný z WWW: <http://www.vmware.com/products/vi/>.

[19] *VMware Products* [online]. VMware, Inc., 2006 [cit. 2006-11-25]. Dostupný z WWW: <http://www.vmware.com/products/home.html>.

[20] *PearPC - PowerPC Architecture Emulator* [online]. Sebastian Biallas, 2003-2006 [cit. 2006-11-25]. Dostupný z WWW: <http://pearpc.sourceforge.net/>.

[21] *XenSource Products - Xen* [online]. XenSource, Inc., 2005-2006 [cit. 2006-11-25]. Dostupný z WWW: <http://www.xensource.com/products/xen/>.