14. V/V podsystém

- původní funkce OS
- komunikace se světem a sekundární pamětí (virtuální paměť, soubory)

Typy V/V rozhraní:

- lidské – komunikace s uživateli (displej, klávesnice, myš, tiskárna, apod.)
- paměťové – ukládání dat (disk, páska, CD/DVD)
- komunikační – komunikace se vzdálenými zařízeními (síť, senzory, řídicí sběrnice)

Parametry V/V zařízení:

- rychlost přenosu, latence
- jednotka přenosu (byte, slovo, sektor, blok, paket)
- složitost ovládání (přímé ovládání řadičem, nepřímo přes řadič sběrnice)
- ošetření chyb
- návaznost na další části jádra (disk V/V, terminál, časovač)

Cíle:

Efektivita – zařízení jsou různě rychlá, vzhledem k rychlosti procesoru ale většinou hodně pomalá. V multiprogramovém systému nesmí provádění V/V brzdit provádění procesů.

Obecnost – historicky bylo každé zařízení ovládáno jiným způsobem (vrstva V/V) a jiným rozhraním (z hlediska programu, rozhraní jádra). Problém při přidávání nových zařízení, různě připojených (disk – interní, externí, USB, vzdálený, atd.). Je proto žádoucí na úrovni vyšších vrstev ovládání zařízení sjednotit a odlišnosti řešit až na nižších úrovních.

Základní funkce V/V podsystému:

A) abstraktní jednotné rozhraní periferních zařízení
B) efektivní využití (sdílení, spooling)
C) ochrana (autorizace přístupu)

Vrstvy V/V rozhraní

1. Rozhraní OS – zasílání požadavků procesy.
2. Logická úroveň – práce se zařízením jako logickým zdrojem, nezabývá se detaily řízení a komunikace.
3. Fyzická úroveň – logické operace převádí na sekvence příkazů pro řadič, programuje a spouští přenosy dat.

1. Rozhraní OS

Jednotné pojmenování V/V zařízení:
symbolické jméno → deskriptor (handle) otevřeného souboru

Unix – symbolická jména zařízení vytvořena nebo mapována v systému souborů */dev/name*, speciální soubory pro blokový, znakový přístup:

```
ls -l /dev
crw-rw----  1 uucp  dialer    28, 128 22 dub 13:43 cuaa0
crw-r-----  1 root  operator  13,   0 16 dub 15:22 da0a
crw-r-----  1 root  operator  13,   8 16 dub 15:22 da1a
crw-rw-rw-  1 root  wheel      2,  22 22 dub 13:33 null
crw--w----  1 owner tty        1,   0 22 dub 13:45 tty
crw-rw-rw-  1 root  wheel      2,  12 16 dub 15:22 zero
```

hlavní (major) číslo – identifikace ovladače
vedlejší (minor) číslo – identifikace jednotky, části
Historicky 8 + 8 bitů (dev_t), dnes většinou 32 bitů (BSD 8 + 24, Linux 12 + 20).
Jména původně vytvářena v normálním systému souborů příkazem *mknod*, dnes většinou automatickou registrací ve speciálním systému souborů (*devfs*, *pseudofs*).

MS Windows – speciální jmenný prostor *device**name* nebo GUID

Jednotné rozhraní V/V - funkce POSIX 1003.1 pro manipulaci se soubory: *open()*, *read()*, *write()* (deskriptor, pozice), *close()*
WIN32: *CreateFile()*, *ReadFile()*, *WriteFile(), CloseHandle()*.

Synchronní V/V - po dobu V/V je proces pozastaven
Asynchronní V/V – proces může mít více zahájených V/V operací, dokončení lze testovat nebo je oznámeno signálem
a) simulace pomocí neblokující operace (POSIX)
Příznak otevření O_NONBLOCK (O_NDELAY):

```
fd = open("/dev/tty", O_RDWR|O_NONBLOCK);
...
if (read(fd, buf, n) == -1 && errno == EAGAIN)
```

Data nejsou k dispozici, je třeba zkusit později.

Problém:
Jak čekat na připravenost několika deskriptorů – nelze stále testovat, aktivní čekání!

Test připravenosti na operaci (BSD):

```
int select(int nfds, fd_set *rfds,fd_set *wfds,
        fd_set *efds, struct timeval *timeout);
```

blokuje dokud není některý deskriptor připraven nebo nevyprší časový limit:

```
    fd_set rfds, wfds, efds;

    FD_ZERO(&rfds);       /* pro každý select! */
    FD_SET(fd, &rfds);    /* pro fd test čtení */
    ...
    tv.tv_sec = 10;     /* časový limit */
    tv.tv_usec = 0;
    if (select(N, &rfds, &wfds, &efds, &tv)>0) {
        /* rfds, wfds, efds mají nastavené
           bity pro připravené deskriptory */
        if (FD_ISSET(fd, &rfds)) {
            read(fd, buf, n);
        }
        ...
    } else /* vypršel časový limit nebo chyba */
```

Neřeší problém zcela - vlastní V/V se dělá klasicky:

- nějakou dobu trvá přenos dat,
- může blokovat, pokud se zařízením pracují jiné procesy.

b) asynchronní operace POSIX 1003.1b:

```
#include <aio.h>
struct aiocb {
   int aio_fildes;     /* deskriptor souboru */
   off_t aio_offset;         /* pozice */
   volatile void *aio_buf;  /* adresa dat */
   size_t aio_nbytes;        /* délka dat */
   struct sigevent aio_sigevent; /* signál */
   int aio_lio_opcode;       /* druh operace */
   int aio_reqprio;          /* priorita oper.*/
};
int aio_read(struct aiocb *acbp);
int aio_write(struct aiocb *acbp);
```

Dokončení oznámeno signálem *aio_sigevent*, převzetí stavu:

```
ssize_t aio_return(const struct aiocb *acbp);
```

c) MS Windows WIN32:

CreateFile() s příznakem FILE_FLAG_OVERLAPPED
standardní *ReadFile()* a *WriteFile()* s posledním parametrem:

```
struct OVERLAPPED {
    DWORD Offset, OffsetHigh;
    HANDLE hEvent;
};
fh=CreateFile(name, GENERIC_READ|GENERIC_WRITE, 0, NULL,
    OPEN_EXISTING, FILE_FLAG_OVERLAPPED, 0, NULL);
ok=WriteFile(fh, buf, size, &len, &overlapped);
...
ok=GetOverlappedResult(fh, &overlapped, &length, wait);
```

Dokončení je signalizováno událostí (*Event*), skutečnou délku a výsledek je třeba převzít pomoci *GetOverlappedResult()*.

Vyrovnávání V/V

Příklad: čtení z terminálu

```
fd = open("/dev/tty", O_RDWR);
length = read(fd, buf, 4096);
```

Během volání *read()* je proces pozastaven, ale data se musí průběžně ukládat na logickou adresu *buf* v adresovém prostoru procesu – část procesu musí být zamčená v paměti! Pro pomalé V/V zařízení to blokuje odkládání stránek (procesů).

- **znaková** (proudová) zařízení – zařízení přijímá (poskytuje) proud dat
- **bloková** (sektorová) zařízení – zařízení přijímá (poskytuje) pouze celistvé bloky dat, nelze číst/zapisovat jiné délky dat.

Typu vyrovnávání (buffering)

1. přímý přenos (bez bufferu)
2. jednoduchý buffer – při čtení se data čtou do systémového bufferu, po dokončení se zkopírujou do bufferu v procesu
3. dvojitý buffer – může probíhat současně čtení a kopírování dříve načtených dat do uživatelského bufferu
4. cyklický buffer – přijímaná data se ukládají postupně do cyklického bufferu, odsud se pak odebírají
5. cache – vyrovnávání diskových operací, v paměti jsou udržovány načtené a modifikované sektory, jsou libovolně adresovatelné (hashování) a spravované nahrazovacím algoritmem (LRU, LFU)

Vyrovnávání V/V blokových zařízení

Čtení/zápis není uvnitř jádra synchronní s požadavky programu, data jsou čtená dopředu, zápis je odložen → problém konzistence dat při výpadku. Operace *fsync()* dělá jednorázovou synchronizaci všeho (dat i metadat) – příliš náročné.

Synchronní režimy otevření souboru (POSIX 1003.1b):

O_DSYNC	Okamžitý zápis dat na fyzické médium včetně aktualizace stavových informací podstatných pro přístup k zapsaným datům. Zapsaná data zůstávají v systémové vyrovnávací paměti, při čtení nemusí být čtena z disku.
O_SYNC	Okamžitý zápis zapisovaných dat na fyzické médium včetně aktualizace všech stavových informací (Unix).
O_RSYNC	Čtení dat přímo z fyzického média. Pokud je současně nastaven příznak O_SYNC, jsou při každém čtení také aktualizovány všechny stavové informace.

Ovlivnění vyrovnávání – POSIX 1003.1-2001

```
int posix_fadvise(int fd, off_t off, off_t len,
int advise);
```

Indikace jak bude soubor zpracováván od pozice *off* v délce *len*:

POSIX_FADV_NORMAL	bez indikace – heuristika pro detekci sekvenčního přístupu
POSIX_FADV_SEQUENTIAL	sekvenčně – má smysl dopředné čtení
POSIX_FADV_RANDOM	náhodně – nemá smysl dopředné čtení
POSIX_FADV_NOREUSE	data nebudou znovu použita – bez vyrovnávání
POSIX_FADV_WILLNEED	data budou použita – lze načíst dopředu
POSIX_FADV_DONTNEED	data nebudou použita – data ve vyrovnávací paměti není třeba držet v paměti

Na jednu část souboru lze operace opakovat, např. SEQUENTIAL, zapsat data, DONTNEED (částečný fsync).

2. Logická úroveň

Zpracování V/V požadavků z vyšší vrstvy nebo z úrovně jádra. Řadí požadavky do front na patřičná logická zařízení, kontroluje práva a platnost operace. Vlastní průběh řídí ovladač zařízení.
Ovladače V/V:

- vlastní komunikace s řadiči V/V
- jednoduché nebo hierarchicky strukturované

Rozhraní ovladačů (SVR4, Solaris 9):

- konfigurace (detekce zařízení, autokonfigurace) – *attach()*, *detach()*, *getinfo(), open(), close()*
- V/V operace (čtení, zápis) – *read()*, *write()*, *strategy(), mmap()*
- asynchronní V/V – *chpoll()*, *aread(), awrite()*
- řídicí operace (formátování, operace s médiem) – *ioctl()*
- přerušení (asynchronní)

Unix – seznam zařízení (*device switch*):

- bloková zařízení: *bdevsw[]->d_open(), d_strategy(),...*
- znaková zařízení: *cdevsw[]->d_open(), d_read(), d_ioctl(),...*

Konfigurace seznamu zařízení: původně ručně, nyní nástroje (*idbuild*, *config*, apod.) nebo autodetekce. Hlavní číslo = index do seznamu zařízení.

SVR4 – **Device-Driver Interface/Device-Kernel Interface:**

- definuje vstupní body ovladače,
- kontext pro provádění volaných vstupních bodů (kontext uživatelského procesu nebo jádra, kontext obsluhy přerušení)
- rozhraní funkcí, které může ovladač používat z jádra (zamykání, přenos parametrů z uživatelského kontextu, alokace paměti, manipulace se seznamy, atd),
- rozhraní funkcí pro přístup k hardware (mapování paměti, registrace přerušení, alokace DMA, práce s registry V/V).

Cíl: binární přenositelnost ovladačů, instalace za chodu (dynamicky instalované moduly ovladačů – KLD, modules)

Vrstvené ovladače – SCSI, USB, FireWire
SVR4 – Portable (Storage) Device Interface (PDI)
FreeBSD - Common Access Method (CAM) pro SCSI, RAID
Princip - ovladače logických zařízení (disk, páska, CD) a vrstva zpracování příkazů dané sběrnice jsou standardní, specifický je pouze ovladač řadiče sběrnice.

Transformace adres v logické vrstvě

Při přenosu dat se střetáváme s různými adresovými prostory:

- Adresy uživatelských dat jsou z rozhraní jádra = logické adresy v adresovém prostoru uživatelského procesu.
- Adresy systémových dat jsou v logickém adresovém prostoru jádra.
- Adresy pro přístup k datům v rámci ovladače jsou v logickém adresovém prostoru jádra.
- Adresy pro řadič V/V kanálu jsou fyzické adresy v paměti (přistupuje přímo do paměti).

Při stránkování je třeba zamapovat všechny stránky bufferu z uživatelského adresového prostoru do adresového prostoru jádra (pokud vyžaduje ovladač přístup procesoru k datům) a ovladači předat adresu v logickém adresovém prostoru jádra – vyžaduje modifikaci tabulky stránek jádra a TLB shootdown. Pokud ovladač nepotřebuje přístup k datům, je tato operace zbytečná, stačí zjistit fyzické stránky obsahující data, ty zamknout v paměti a předat jejich adresy kanálovému řadiči ve formě rozložené V/V operace (po jednotlivých fyzických stránkách).

3. Fyzická úroveň - úroveň řadičů periferních zařízení

Cíle:

- minimalizovat režii
- malé zpoždění odezvy (latence)
- maximální prostupnost

Příklad:
sériové rozhraní 9600Bd = 900 B/s → 1000 přerušení/s
10 uživatelů → 10000 přerušení/s (100 μs)

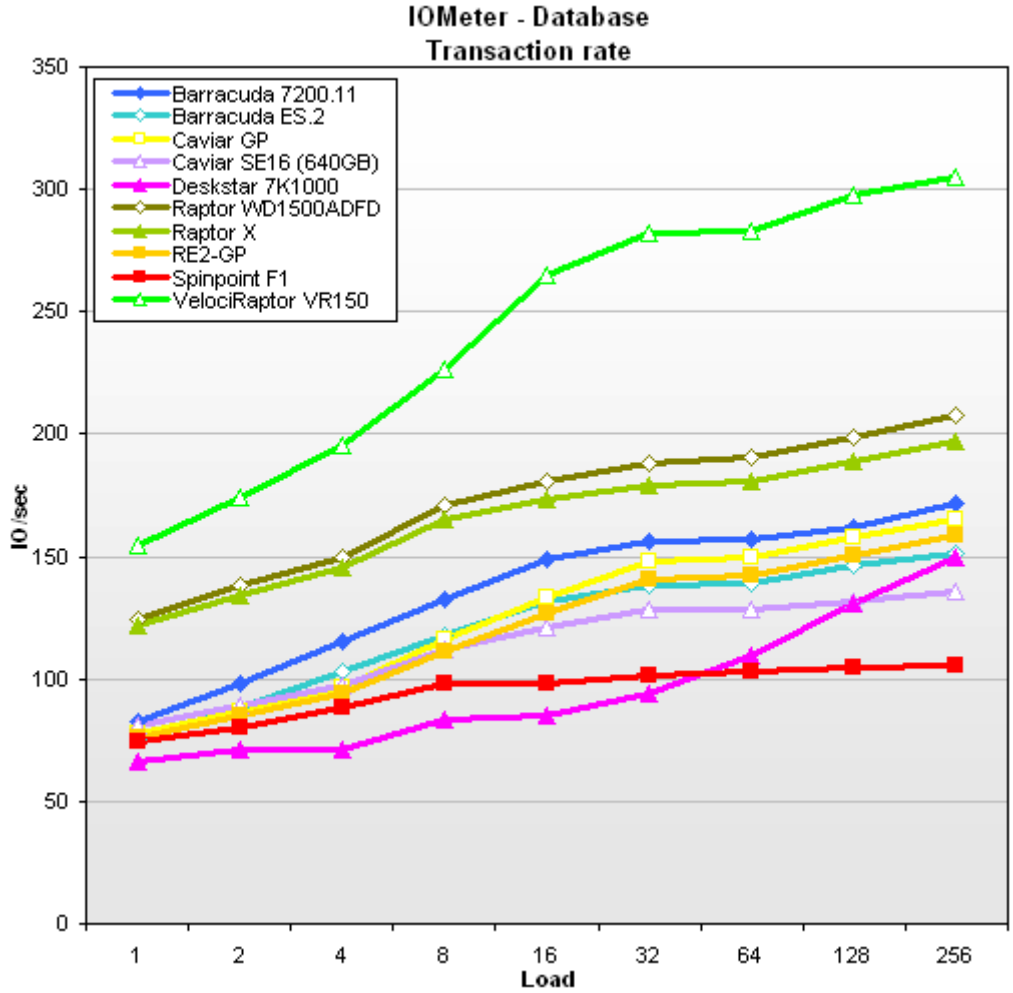
Realizace:
a) Programový přenos – každý byte/slovo jde do/z řadiče explicitně instrukcemi procesoru:
   - blokující, s testováním stavu – má smysl pouze pro krátké přenosy (nastavení řadiče), např. parametry diskové operace
   - signalizace dokončení přenosu pomocí přerušení – pouze pro pomalá zařízení a malé objemy dat (sériová linka, klávesnice, myš).

b) Přímý přístup do paměti (DMA):
   - přenos je naprogramován (pomocí a) do řadiče DMA – řadič periferního zařízení autonomně dodává data a DMA je ukládá/přebírá do/ze zadaného místa.

c) Kanálový přenos (PCI bus master):
   - přenos je popsán datovými strukturami v paměti, které tvoří seznam nebo jsou uloženy v poli, řadič perifierního zařízení si autonomně přepírá operace z těchto datových struktur a autonomně přenáší data z/do paměti.

Optimalizace diskových přenosů

Doba provedení = *vystavení* + *rotační zpoždění* + *přenos*
8-12 ms, 5-8 ms, 1 ms (4 KB, 4 MB/s)



Disky do serverů (SCSI/FC/SAS) umí zpracovávat paralelně více příkazů, dokážou si optimalizovat frontu požadavků – s rostoucím počtem rozpracovaných požadavků lze lépe optimalizovat a tím zkracovat dobu operace. Levnější disky (PATA/SATA-1) to obvykle neumí nebo nezvládají, musí pomoct systém. Je třeba hlavně minimalizovat vystavování:

- vhodná organizace systému souborů (viz dále)
- optimalizace fronty požadavků:

FIFO - vystavování podle příchodu požadavků (náhodné adresy)
fronta: 98, 183, 37, 122, 14, 124, 65, 67, hlava na pozici 53

```
|---|----|--------|-----------------|----|
 37  53       98              183
```

- vystavovaní o diference:
  = 45 + 75 + 146 + 85 + 108 + 110 + 59 + 2 = 640 cyl.
- OK pro malou zátěž, malý rozptyl doby obsluhy

SSTF (Shortest Seek Track First)

- vždy požadavek s nejmenším vystavením

pořadí zpracování fronty: 65, 67, 37, 14, 98, 122, 124, 183
vystavení o diference: 12+2+30+23+84+24+2+59 = 236 cyl.

- velký rozptyl doby obsluhy (stárnutí, starvation)

SCAN (výtah)

- nastaven aktuální směr vystavování
- vždy požadavek s nejmenším vystavením v daném směru
- když není žádný, obrátí se směr vystavování

Předpokládáme směr pohybu k 0
pořadí zpracování fronty: 37, 14, 65, 67, 98, 122, 124, 183
vystavení o diference: 16+23+51+2+31+24+2+59 = 208 cyl.

- lepší než SSTF, menší rozptyl, nenastává stárnutí

C-SCAN (Circular SCAN)

- jako SCAN, ale vždy se zpracovává v jednom směru
- po zpracování návrat na nejnižší pozici

pořadí zpracování fronty: 65, 67, 98, 122, 124, 183, 14, 37,
vystavení o diference: 12+2+31+24+2+59+169+23 = 322 cyl.

- menší průměrná doba zpracování než pro SCAN
- nízký rozptyl průměrné doby zpracování
- pro malou zátěž je SCAN lepší

N-step-Scan
Fronta je rozdělena na několik front o max. délce N, ty jsou zpracovávány postupně sekvenčně metodou SCAN. Během zpracování fronty jsou další požadavky přidávány do následující nezaplněné fronty. Pro N=1 degraduje na FIFO, pro velké N se blíží ke SCAN.

FSCAN
Varianta N-step-SCAN s pouze dvěmi frontami neomezené velikosti. Během zpracování jedné se přidávají nové požadavky do druhé fronty.

CFQ (Complete Fair Queueing, Linux)
Požadavky každého procesu se řadí do samostatné fronty, odsud se vybere vždy první požadavek a na ty se uplatní SCAN.

Zpracování chyb

- Vadné sektory – dnes řeší disk autonomně, dříve tabulka vadných sektorů v systému.
- Celý disk vadný:
  1. zálohování – off-line obnova dat, ne vždy do posledního stavu
  2. redundance – částečná nebo úplná duplicita dat na více fyzických discích, odolnost vůči výpadku *m* disků
     - na úrovni fyzické – duplicita diskových sektorů (RAID)
     - na úrovni logické – duplicita alokačních bloků systému souborů (LFS, ZFS)

RAID (Redundant Array of Independent Disks):

1. externí řadič připojený FC/SCSI/iSCSI
2. v interním řadiči (samostatný dedikovaný procesor)
3. na úrovni ovladače (levné SW řešení, RAID-0/1)
4. na úrovni mezivrstvy diskových zařízení (zcela v OS)

Úrovně RAID:

RAID-0 – konkatenace několika disků pro zrychlení, velikost bloku se volí 16 – 64 KB, žádná odolnost vůči výpadku.

RAID-1 – zrcadlení, data se zapisují současně na dva disky, číst se může z kteréhokoli, odolnost vůči výpadku disku.

RAID-10 – kombinace zrcadlení a konkatenace pro více než 2 disky.

RAID-2 – dedikované disky pro ECC originálů (nepoužívá se), lze dosáhnout různé úrovně odolnosti volbou parametrů Hammingova kódu ECC.

RAID-3/4 – dedikovaný disk pro paritu (XOR originálů), liší se počítáním parity, RAID-3 na úrovni bajtů (vyžaduje synchronizaci otáček disků), RAID-4 na úrovni celých bloků, min. 3 disky, odolnost vůči výpadku jednoho disku, paritní disk je neúměrně zatížen při zápisu (prakticky se nepoužívá).

RAID-5 – rotující rozprostřená parita po všech discích, min. 3 disky, odolnost vůči výpadku jednoho disku.

RAID-6 – duální rozprostřená parita, min. 4 disky, odolnost vůči výpadku dvou disků.

Kdy použít který RAID:
velká rychlost čtení – RAID-0, RAID-5/6
rychlý zápis – RAID-0, RAID-10
velká kapacita – RAID-5/6 (ale pomalý zápis)
pouze redundance – RAID-1 (nepřináší zrychlení)

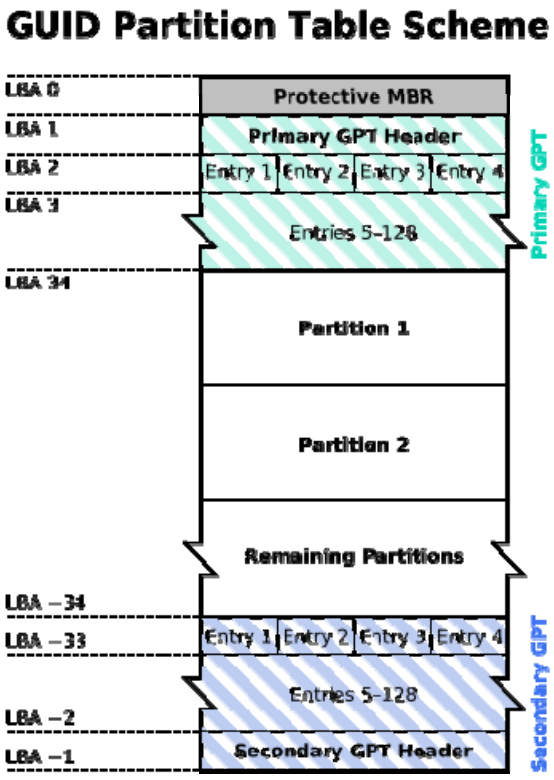
	disk0	***disk1***	***disk2***
RAID-0	*blok0,blok3,...*	*blok1,blok4,...*	*blok2,blok5,...*
RAID-1	*blok0,blok1,...*	*blok0,blok1,...*	
RAID-3	*blok0|byte0* *blok0|byte2,...*	*blok0|byte1* *blok0|byte3,...*	*blok0|byte0^byte1* *blok0|byte2^byte3*
RAID-4	*blok0, blok2,...*	*blok1, blok3,...*	*blok0^blok1,* *blok2^blok3,...*
RAID-5	*blok0, blok2,...*	*blok1,* *blok2^blok3,...*	*blok0^blok1,* *blok3, ...*
RAID-6	*blok0,* *blok2,*	*blok1,* *blok2◊blok3,*	*blok0◊blok1,* *blok2^blok3,*
RAID-6 disk3	*blok4◊blok5,...* *blok0^blok1,* *blok3,blok5,...*	*blok4^blok5, ...*	*blok4*

RAID-5,6 – levá nebo pravá rotující parita (^ je XOR, ◊ je RS)
Problém zápisu RAID-5 – při zápisu méně než (N-1)**blok* se musí dopočítat parita z obsahu existujících bloků. Operace zápisu musí tedy načíst nemodifikované bloky v řádku, dopočítat paritu a zapsat zapisované bloky + nový paritní blok. Pro rozumnou rychlost zápisu musí mít řadič cache na tyto operace:
Write-through – zápisová operace končí, až když se zapíše celý modifikovaný řádek na všechny disky, **problém krátkých zápisů** na úrovni systému,
Write-back – zápisová operace končí okamžitě, zapsané bloky zůstanou nějakou doby v cache, může spojovat sousedící zápisy a tím redukovat nutné čtení při zápisu, **problém při výpadku** (nekonzistentní řádky, nutná nevolatilní cache, obvykle zálohovaná baterkou).

Správa diskového prostoru (Volume Management)

Dříve – jeden disk = jeden systém souborů
Vývoj přinesl postupně větší disky, nutnost rozdělit diskový prostor na více oddílů – pro každý oddíl typ, začátek, velikost:

- UNIX Sytem V VTOC (Virtual Table of Content) - až 184 oddílů
- BSD label, až 8 oddílů, jeden obvykle přes celý disk
- partition table (PC) – 4 oddíly, limitováno velikostí 2TB
- EFI (GPT = GUID Partition Table) – max. 128 oddílů, začátek a velikost zadána 64bitovými čísly (LBA), podpora Linux, BSD, Mac OS X, Win 7/64bit



Potřeba vytváření větších systémů souborů přes více disků – spojování jednotlivých disků (RAID-0). Zavedení mezivrstvy mezi ovladači jednotlivých disků a systémem souborů – **správa svazků** (volume management). Dovoluje vytvářet logické disky rozprostřené na fyzické disky, případně včetně softwarového RAID-1 (HP-UX, IBM LVM, BSD geom, Linux md, apod.).

Problém 4KB/512B sektorů

Historicky (od roku 1956) je velikost sektoru na disku 512B.
S růstem hustoty dat na disku začíná být problém délka ECC kódu, začíná zabírat příliš velkou část sektoru (původně 6%, nyní až 13%). Pro zmenšení režie je nezbytné zvětšit délku sektoru.
Plán asociace IDEMA – od roku 2010 jsou dodávany disky s fyzickými sektory 4KB (www.bigsector.org). V první fázi budou emulovat na úrovni rozhraní disku sektory 512B, v druhé fázi bude emulace zrušena (2014 - objevují se první disky bez emulace).

Problém:
Systémy souborů sice používají alokační bloky 4KB a větší, ale nejsou zarovnány na hranici násobku 4KB od začátku disku!

0	*1*	*2*	*3*	*...*	*62*	*63*	*64*	*65*	*66*	*67*	*68*	*69*	*...*	
MBR						*AB1*	*AB1*	*AB1*	*AB1*	*AB1*	*AB1*	*AB1*	*AB1*	*AB2*
4K sektor 0						*4K sekt. 7*							*4K sektor 8*	

Zápisová operace logického 4K alokačního bloku zasáhne 2 fyzické 4K sektory – disk musí interně načíst obsah jednoho sektoru, změnit v něm posledním 512B, zapsat zpět, načíst další 4K sektor, změnit v něm počítečních 3,5KB a zapsat zpět

Proč?

První oddíl začíná z historických na cylindru 0, hlavě 1, sektoru 1 (číslováno od 1., 63 sektorů/hlavu) = LBA 63
Změna začátku oddílu je problém: fdisk, MBR kód, boot kód oddílu, operační systém => nutný přechod na GPT tabulku rozdělení disku.